Derctuo

Kragen Javier Sitaker
Buenos Aires
December, 02020
Public domain work

Derctuo is a book of notes on various topics, mostly science and engineering with some math, from the first year of the COVID-19 pandemic, 02020 CE. Its primary published form is a gzipped tarball of 9MB of HTML files and sources, although there's also an inferior PDF version of about 1000 pages for reading on hand computers or printing. It uses a page size slightly smaller than standard for improved readability on hand computers.

My original plan was to write a reproducible computation system so that the book would be entirely reproducible from a minimal computational core, allowing all of its calculations to be not only verified but also easily extended, reused, and studied. I didn't get very far on that plan. Instead it's mostly just about a quarter million words of dead text, with some static inline images, plus a bundled library of source material, which is not included in the PDF version.

It contains some novel discoveries, but some of it is just my notes from exploring the enormous feast of knowledge now available on the internet to anyone who takes the time to taste of it, and some other parts are explorations that didn't pan out — left here only as a cautionary tale to the next explorer.

There are lots of notes in here that aren't "finished" in the usual sense; they end in the middle of a sentence, or say "XXX", or have a note in them that the foregoing is wrong in such-and-such a way. But I am publishing the final version of Derctuo today. I might make future versions of some of these notes, but not of Derctuo itself.

Derctuo is the sequel to Dercuano, a larger collection of my notes which I published in 02019.

Like Dercuano, Derctuo is lacking in scholarship: many of the notes reflect fields I understand poorly, and so often they fall victim to known pitfalls, and fail to describe how they relate to existing knowledge. It is best thought of as working notes published despite their incomplete state.

Often in Derctuo, as in Dercuano, I refer to something being "published". By this I generally mean "made public". There's a current fad in academia (over the last, say, half century) to use "published" to mean "brought up to the standards of scholarly publication and approved as such by means of scholarly peer review". This is usually not what I mean.

Public-domain dedication

the public domain. I wrote and published Derctuo in Argentina in 02020 (more conventionally called 2020, or 2020 AD).

The exception to the above public-domain dedication is the ET Book font family used, licensed under the X11 license (p. 13). This doesn't impede you from redistributing or modifying Derctuo but does prohibit you from removing the font's copyright notice and license (unless you also remove the font). The PDF embeds part of FreeFont and of the DejaVu fonts, whose copyright notices are also included (p. 14), but DejaVu and FreeFont are not used in the HTML tarball.

Gitlab

At this writing, there's a replica of this repo on Gitlab.

Notes

02020-04

• Difficulty estimation of programming tasks (p. 16) 02020-04-20 (2 minutes)
• Pure functional UI (p. 17) 02020-04-21 (4 minutes)
• Pure functional VM (p. 19) 02020-04-21 (1 minute)
• A reproducible vector-instruction VM? (p. 20) 02020-04-21 (updated 02020-06-17) (30 minutes)
• Ballpoint SPIF (p. 36) 02020-04-25 (7 minutes)
• Bitwise reproducibility (p. 39) 02020-04-25 (1 minute)

02020-05

• Reversible parsing (p. 40) 02020-05-11 (6 minutes)
• Bloomtags: a Bloom-filter tree for efficient and flexible database queries (p. 44) 02020-05-13 (21 minutes)
• Static hypertext on CCN (p. 51) 02020-05-16 (2 minutes)
• Feeds or streams on CCNs (p. 52) 02020-05-16 (15 minutes)
• Commit log transfer (p. 57) 02020-05-16 (1 minute)
• One pass sort (p. 58) 02020-05-16 (15 minutes)
• Optimized finger joints (p. 63) 02020-05-16 (4 minutes)
• Solar furnace CPC (p. 65) 02020-05-16 (12 minutes)
• Pandemic collapse (p. 69) 02020-05-17 (updated 02020-12-16) (22 minutes)
• Font rendering with all-pass filters (p. 76) 02020-05-18 (7 minutes)
• Single output build (p. 79) 02020-05-19 (4 minutes)
• Electronics kit (p. 81) 02020-05-23 (updated 02020-12-20) (14 minutes)

02020-06

• Sodium silicate (p. 86) 02020-06-04 (32 minutes)
• One big text file (p. 97) 02020-06-04 (updated 02020-06-06) (20 minutes)
• Monoid prefix sum (p. 105) 02020-06-05 (13 minutes)
• Writing a shopping list in TeX (p. 110) 02020-06-05 (4 minutes)

02020-07

02020-08

02020-09

02020-10

02020-11

02020-12

Topics

liabilities/LICENSE.ETBook

[This is the copyright notice from the ET Book font Dercuano uses.]

liabilities/dejavu-copyright

```
Format: http://www.debian.org/doc/packaging-manuals/copyright-format/1.0/
Upstream-Name: DejaVu fonts
Upstream-Author: Stepan Roh <src@users.sourceforge.net> (original author),
                 see /usr/share/doc/ttf-dejavu/AUTHORS for full list
Source: http://dejavu-fonts.org/

Files: *
Copyright: Copyright (c) 2003 by Bitstream, Inc. All Rights Reserved.
 Bitstream Vera is a trademark of Bitstream, Inc.
 DejaVu changes are in public domain.
License:
 Permission is hereby granted, free of charge, to any person obtaining a copy
 of the fonts accompanying this license ("Fonts") and associated
 documentation files (the "Font Software"), to reproduce and distribute the
 Font Software, including without limitation the rights to use, copy, merge,
 publish, distribute, and/or sell copies of the Font Software, and to permit
 persons to whom the Font Software is furnished to do so, subject to the
 following conditions:
 .
 The above copyright and trademark notices and this permission notice shall
 be included in all copies of one or more of the Font Software typefaces.
 .
 The Font Software may be modified, altered, or added to, and in particular
 the designs of glyphs or characters in the Fonts may be modified and
 additional glyphs or characters may be added to the Fonts, only if the fonts
 are renamed to names not containing either the words "Bitstream" or the word
 "Vera".
 .
 This License becomes null and void to the extent applicable to Fonts or Font
 Software that has been modified and is distributed under the "Bitstream
 Vera" names.
 .
 The Font Software may be sold as part of a larger software package but no
 copy of one or more of the Font Software typefaces may be sold by itself.
 .
 THE FONT SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
 OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OF MERCHANTABILITY,
 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF COPYRIGHT, PATENT,
 TRADEMARK, OR OTHER RIGHT. IN NO EVENT SHALL BITSTREAM OR THE GNOME
 FOUNDATION BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, INCLUDING
 ANY GENERAL, SPECIAL, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES,
 WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF
 THE USE OR INABILITY TO USE THE FONT SOFTWARE OR FROM OTHER DEALINGS IN THE
 FONT SOFTWARE.
 .
 Except as contained in this notice, the names of Gnome, the Gnome
 Foundation, and Bitstream Inc., shall not be used in advertising or
 otherwise to promote the sale, use or other dealings in this Font Software
 without prior written authorization from the Gnome Foundation or Bitstream
 Inc., respectively. For further information, contact: fonts at gnome dot
```

org.

Files: debian/*
Copyright: (C) 2005-2006 Peter Cernak <pce@users.sourceforge.net>
           (C) 2006-2011 Davide Viti <zinosat@tiscali.it>
           (C) 2011-2013 Christian Perrier <bubulle@debian.org>
           (C) 2013 Fabian Greffrath <fabian+debian@greffrath.com>
License: GPL-2+
 This program is free software; you can redistribute it
 and/or modify it under the terms of the GNU General Public
 License as published by the Free Software Foundation; either
 version 2 of the License, or (at your option) any later
 version.
 .
 This program is distributed in the hope that it will be
 useful, but WITHOUT ANY WARRANTY; without even the implied
 warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
 PURPOSE. See the GNU General Public License for more
 details.
 .
 You should have received a copy of the GNU General Public
 License along with this package; if not, write to the Free
 Software Foundation, Inc., 51 Franklin St, Fifth Floor,
 Boston, MA 02110-1301 USA
 .
 On Debian systems, the full text of the GNU General Public
 License version 2 can be found in the file
 /usr/share/common-licenses/GPL-2'.

Difficulty estimation of programming tasks

Kragen Javier Sitaker, 02020-04-20 (2 minutes)

As I was writing file `TODO.md` I was trying to figure out how fast or slow I actually am at programming and how predictable this is.

Ur-Scheme was 1553 lines of Scheme and mostly took me four weeks and three days, from February 3 to March 4 of 02008, exactly a person-month. David A. Wheeler's "SLOCCount" says it should have taken 3.8 person-months using the basic COCOMO formula `2.4 * (KSLOC**1.05)`, so let's suppose that instead of being the famous "10x programmer" I am a 3.8x programmer, not compared to normal modern programmers but to whatever losers the COCOMO model was calibrated on.

As another more recent data point, Dercuano's genpdf.py took me 5 days (0.25 person-months), and it's 550 lines of code, suggesting 1.28 person-months --- a productivity factor of about 5x for me. 5 is pretty close to 3.8.

I'd like to evaluate StoneKnifeForth's development speed in this way, but I don't have any reasonable way to do so, since I don't know how to evaluate either how many lines of code it is or how long it took me to write.

SLOCCount says that the part of BubbleOS I have written so far should have taken 17 person-months since it contains 6473 lines of code, although about 800 of those are the actuarial tables in the death clock Toki. In fact, I wrote most of it from October 12, 02018, to February 22, 02019, which is almost four months; this is also lower than SLOCCount's estimate by only a factor of about 3-5.

I wrote Dumpulse mostly October 15-17, 02017, about 0.14 person-months. Checking out the commit 52f10e5a5d22c9ef8f78992cdc95cbdb8ed4ee79 I get 646 lines of code, nominally 1.52 person-months. In this case the multiplier is closer to 10x. I think this is partly because I'd already been thinking and talking about how to do it and partly because I was able to stay pretty focused for three days --- although the git commits cluster into only four or five hours on each of those days.

I might be able to speed things up by taking advantage of new programming technology like generative testing, or by choosing especially conservative and well-understood designs.

Topics

- Programming (p. 807) (13 notes)
- Derctuo (p. 820) (9 notes)
- Psychology
- Dercuano
- BubbleOS

Pure functional UI

Kragen Javier Sitaker, 02020-04-21 (4 minutes)

How about a pure functional approach? An image is, perhaps, a function from (x, y) to (r, g, b), perhaps augmented with an aspect ratio (max x?); an animation is a function from time to images; a function is some code and some closed-over data; a graphical user interface state is an image or, perhaps, an animation, and a function from input events (such as mouse and keyboard events, but perhaps also idle time and timer expiry) to new states. Such definitions permit caching, checkpointing, undo, rendering frames in parallel, interrupting computations, and resampling, but no real composition --- no way to provide a GUI state as a parameter to another GUI state.

The function to render a character-cell display is pretty simple:

```
def pixat(x, y):
    row, xoff = divmod(x * cols, 1)
    col, yoff = divmod(y * rows, 1)
    glyph = font[text[row][col]]
    return glyph[round(xoff * font.height)][round(yoff * font.width)]
```

This is closed over variables `cols`, `rows`, `font`, and `text`.

If we've resigned ourselves to the cost of starting up and shutting down a new "process" for each keystroke, mouse movement, and frame to paint, a reasonable assembly-level interface for a machine-code computation to access its input data is to map all the input and state data "files" into a newly invoked process's memory space, one memory segment per file. Rather than identifying these segments by ordinal number, I think it's better to identify them by textual name, and expect the process to invoke a library function to look up the segment descriptor --- like Unix environment variables, but each name is associated with a whole memory segment rather than just a NUL-terminated string.

For composition of computations with arbitrary machine code, we need ways for a computation to produce more output than just an image and take more diverse input than just keyboard and mouse events. A capability to spawn child computations --- write output files, in effect --- would go some distance, but that only supports fanout, not the much more ubiquitous fanin. You need some kind of way to provide an existing computation as an argument to another computation, and the user interface affordances for this need to work in a more efficient way than simply iterating over all computations that exist, querying each one in turn.

A simple approach would be Golang-interface-like duck typing, where to request an object as input you specify a list of method names (or method type signatures) you want the object to support, and only objects supporting all of these methods are offered to the user as options. In some cases these may just be things like "asString" or "asImage". To support backward compatibility, you might be able to accept N different interfaces instead of just one.

A different way to do composition is using event channels: when an event is posted to an event bus, all the subscribers on that event bus are awoken with a copy of that event. Usually this approach implies some degree of nondeterminism; the Urbit approach is to wait on (possibly remote) futures instead of on pub-sub event channels. There is still potentially some nondeterminism in the Urbit approach, since in Urbit it is possible for a particular future to be satisfied by more than one different process, and generally whichever one arrives first is the one that wins.

A potentially more satisfying approach would be to make data files, rather than stateful computations, the fundamental objects of the world, but prescribe a FlatBuffers-like layout.

Topics

Pure functional VM

Kragen Javier Sitaker, 02020-04-21 (1 minute)

I'm trying to figure out how to specify reproducible computations for Derctuo in a way that won't cost me months of work before I can start using it.

Urbit approaches the problem of doing reproducible computations with a pure functional virtual machine rather than an imperative one. This rules some things out of scope: issues of efficiency, memory usage, and latency, for example. But it certainly simplifies the kind of computation whose purpose is to compute an unknown result, rather than to react to events in the world. And it might be possible to make it fast enough on modern machines, at least under most circumstances.

There's lots of information out there about how to do reasonably efficient evaluation of λ-calculus expressions, and I've done a few compilers along those lines myself. My Bicicleta work instead uses Abadí and Cardelli's ς-calculus as the basis, which is slightly more verbose than the λ-calculus but, I think, considerably more convenient for programming. Using name-value pairs rather than positional arguments to pass data around permits decentralized extensibility.

The Bicicleta interpreter that I wrote, however, is extremely slow, close to the speed of bash script. I'm sure I can do better than that, using approaches like those I used in Ur-Scheme.

Topics

A reproducible vector-instruction VM?

Kragen Javier Sitaker, 02020-04-21 (updated 02020-06-17)
(30 minutes)

A big part of the mission for Derctuo is to make computational experiments reproducible, both by removing nondeterministic choices from the implementation and by minimizing environmental dependencies. Can we reconcile this with efficiency by implementing a vector virtual machine?

This note describes an approach to Veskeno's design (p. 126) that I am not currently pursuing.

The background of the problem

Computational experiments are more compelling when they can use a larger fraction of the power of your computer, and typical interpreted languages waste on the order of 97% of your computer's computational power. Now that everybody's computer is massively parallel with 4-wide SIMD operations and 4-32 cores, even single-threaded nonvectorized C wastes on the order of 97% of your computer's computational power; typical interpreted languages like Python or PHP thus waste 99.9% of its power. (And that's assuming you don't have a GPU, which can easily push that to 99.99%.) In effect, using languages implemented in this way costs you three orders of magnitude of performance, pushing you 15 years into the past, to 02005 or so --- a performance price that implies a progressively longer timespan as we get further and further out of the shadow of Moore's Law.

Simple untyped virtual machines like Chifir, Dontmove, Wirth's RISC, or the Cult of the Bound Variable's Universal Machine suffer a similar performance hit: not only are they single-threaded, but also, like Forth, they typically spend about 5× as much work on instruction dispatch as they do on useful computation. This is less than all the suffering induced by all of Python's type-checking and bounds-checking, but it's still painful. This offers implementors an unappealing tradeoff: either they can accept painfully limited performance, or they can add a lot of complexity to their implementation in the form of clever optimizations to try to reduce the performance price, at the potential cost of breaking correctness.

One of the great historical advantages of languages like Octave, R, Numpy, Yann LeCun's Lush, and APL is that even a fairly straightforward interpreter is capable of achieving reasonable speeds, because the inner loops are not interpreted --- they happen within primitives of the language like a+b, +/a, or *\a. This is somewhat less true nowadays that our cache hierarchies are so deep and data locality is so important; while straightforward Python code usually runs around 3% of the speed of C, locality effects usually limit straightforward Numpy code to around 20% of the speed of C (comparable to interpreted Forth or something like Chifir), and

optimized Numpy code usually runs around 33% of the speed of C.

Nowadays, a potential additional interesting advantage is that programs in such languages expose data parallelism in a way that a relatively straightforward interpreter could potentially exploit, if the overhead for moving data between threads or processes in the host system is not too great. Maybe you could use 20% of your whole machine instead of 20% of one core.

You could easily imagine splitting a computation like this one across cores by either row or column, although perhaps not until it's much larger:

```
>>> np.arange(12).reshape((3, 4)) * (np.arange(3) + 4).reshape((3, 1))
array([[ 0,  4,  8, 12],
       [20, 25, 30, 35],
       [48, 54, 60, 66]])
```

Implementation limits

As I've said elsewhere, Lorie's UVC falls down on compatibility grounds when it refuses to apply limits to things like register bit sizes. Not putting a limit in the specification doesn't mean that implementations won't have limits; it just means that every independent implementation will have different limits, so the specification is insufficient for compatibility.

In particular, in this case, I think there should be maximum sizes on all arrays and indices, probably 2**32.

Determinism via non-mutation

Still, it seems likely that implementors still face an unappealing performance-correctness tradeoff, in a different way: they will want to perform loop fusion to avoid useless traffic to main memory, but for some virtual-machine designs, it would be easy for such loop fusion to produce different results in some circumstances, specifically when the output aliases one or more of the inputs. Numpy sometimes does produce unexpected results when the output of an operation aliases its input --- by itself, that doesn't necessarily violate the desideratum of reproducibility, but you would have to nail down precisely what results are required, and it would be easy for loop-fusion optimizations, among others, to accidentally break those results.

Still, these problems only arise if data is mutable. Numpy data is mutable, but, for example, APL data is purely immutable, at a logical level. You can say R[3] <- 4 in APL, and after that R[3] is indeed 4, but any aliases to R are not affected, though I think typically implementations avoid making a physical copy when possible. If this immutability were an inherent part of the virtual machine, the opportunities for such nondeterminism would be vastly rarer. There's still the possibility for an implementor to use reference counts to conditionally do in-place updates in order to reduce memory traffic (or memory usage) and botch it.

So, if the virtual machine definition treats arbitrary-sized arrays (up to the maximum) as if they were immutable atomic numbers, it

should mostly steer clear of this kind of nondeterminism. This also suggests treating the machine's memory as a storage not for bytes but for arrays, like a Python module is a storage for Python objects.

Toward an instruction set design?

Simple scalar virtual machines like those mentioned above commonly have 16 or so instruction opcodes: four or five arithmetic operations, one to four bitwise operations, some comparisons and conditional jumps, procedure call and return, and maybe load, store, load literal, and maybe some kind of I/O operations (both Chifir and the CBV UM have "read keyboard" instructions). By contrast, len(dir(numpy)) is 587, and that doesn't even include the 163 methods on Numpy arrays, though some are duplicates. Even old APLs normally have on the order of 60 built-in functions, without counting the results of operators like ×.+ or +/. Can this be reduced down to something reasonable? Maybe 32 opcodes or 64, not 700.

(Of course, many of these items in Numpy are non-fundamental operations like `average`, `bartlett`, and `fft`.)

Lush is unusual among array languages in that it exposes some inner machinery that is usually kept hidden; a Lush "matrix" or "tensor" consists of a "storage" and an "index". The storage is a one-dimensional array of some homogeneous atomic element type, and the storage is realized as a base pointer, a length, an element type, and flags indicating writability and memory-mappedness; the index contains a pointer to a storage, a start offset into that storage, a number of dimensions, and an upper bound and an address increment (possibly zero!) for each dimension. Exposing something like this in the instruction set might save the virtual machine a large number of index-manipulation operations: reshape, matrix transposition, matrix diagonal extraction, ravel, sliding windows (by having two dimensions with the same stride), shape extraction, take, drop, generating arrays filled with a constant, and so on.

One way to supply this facility would be to have the following:

- a shape(array) operation to extract a possibly-empty vector of dimension bounds;
- a reshape(array, shape, strides) operation which creates a new array of the given shape from the raveled elements of array, using the stride vector strides;
- and a drop(N, array) operation which drops the first N items of array.

If the array being reshaped or dropped is already irregular, we might have to copy it, and it isn't clear what drop() should do on non-one-dimensional arrays.

Could we get by with just one-dimensional vectors and slicing operations? The Python expression s[3:10:2] gives us a list of items 3, 5, 7, and 9; a similar instruction could take a vector, a start, a *count* rather than an end, and a stride, which could be zero. Even this could be decomposed into an index-generation instruction that produces the vector [3 5 7 9] (just as the Octave expression 3:2:10 does) and an indexing instruction. Is that kind of thing adequate to express my example Numpy expression from earlier looplessly in terms of

one-dimensional arrays?

```
>>> np.arange(12).reshape((3, 4)) * (np.arange(3) + 4).reshape((3, 1))
array([[ 0,  4,  8, 12],
       [20, 25, 30, 35],
       [48, 54, 60, 66]])
```

I don't think so. The column vector on the right [[4] [5] [6]] is in effect being transformed into [[4 4 4 4] [5 5 5 5] [6 6 6 6]], which you could get by indexing it with [[0 0 0 0] [1 1 1 1] [2 2 2 2]] in a gathering operation. (You can literally do this in Numpy: `(np.arange(3) + 4)[[[[0, 0, 0, 0], [1, 1, 1, 1], [2, 2, 2, 2]]]]`.)

Another tricky problem is how to compile something like `lambda x: np.arange(12).reshape((3, 4)) * x`. You could apply this to an x like the 3-column above, in which case you need to broadcast each of its elements across a row; but you could also apply it to an x such as np.arange(4), in which case you need to broadcast each of its elements across a column, or to a scalar, or to a 3×4 matrix, or for that matter to a 2×3×4 array like `np.arange(24).reshape((2, 3, 4))`. If you're going to insert a sequence of virtual machine instructions to distinguish among cases like these before every multiplication operation, you are going to incur enough interpretation overhead that actual vector programming languages will not run well on your vector virtual machine; if you want to have this broadcasting logic at all, it is probably better to push it down into the definition of the virtual-machine operation; and of course that would require the VM to see the values as N-dimensional arrays, not just vectors.

Operations on boolean arrays in APL are traditionally unified with, I think, gcd and lcm, but it seems to me more reasonable to unify them with pairwise max and min. In some sense, an N-bit integer in a computer is an N-item boolean vector, and this is an efficient way to represent boolean arrays; since we probably need pairwise max and min in any case, it might be best to specify two operations to translate back and forth between boolean arrays and arrays of N-bit integers, rather than specifying bitwise AND, OR, and NOT operations. An efficient implementation can do this without copying.

There's an indexing operation. Indexing a vector by an array index performs a gather, producing a result with the same shape as the index. It isn't clear what should happen when you index a multidimensional array by anything other than some scalars; see the section below, "Numpy indexing and broadcasting".

There's an index update operation. It produces a new array that is mostly the same as an old array, but has some indices replaced. For things like painting pixels in a framebuffer, it seems like it might be important to support things like `pix[xs, ys] = red`, although I guess you could reshape the framebuffer into a vector first and index it with `xs + ys * width`.

(The reshaping operation mentioned earlier could be seen as indexing an array with one or more indices with special characteristics, like "slice objects" or "range objects"; would it make sense to just provide an index generation operation and leave the reshaping to the indexing operation? A simple implementation could

omit optimizing the special case, and the extra orthogonality would allow it to be used with index update as well, maybe. But in some cases not optimizing that special case results in quadratic or worse memory blowup.)

What about reductions and scans? Like indexing of multidimensional arrays, these need some axis to run along, but they also need a binary operator. You could use the reshape operation to reorganize the axes so that the desired axis comes first, or maybe last.

Numpy indexing and broadcasting

There are several possible ways to index multidimensional arrays in Numpy:

```
>>> y                 # shape (2, 3)
array([['h', 'o', 'w'],
       ['d', 'l', 'y']],
      dtype='|S1')
>>> y[[0, 1, 0]]       # Indexing by default is on the first dimension
array([['h', 'o', 'w'],
       ['d', 'l', 'y'],
       ['h', 'o', 'w']],
      dtype='|S1')
>>> y[[0, 1, 0], ...]  # equivalent
array([['h', 'o', 'w'],
       ['d', 'l', 'y'],
       ['h', 'o', 'w']],
      dtype='|S1')
```

Indexing by a complicated thing replaces the indexed dimension with its shape:

```
>>> y[[[[[[0, 1, 0]]]]]]
array([[[[[['h', 'o', 'w'],
          ['d', 'l', 'y'],
          ['h', 'o', 'w']]]]]],
      dtype='|S1')
>>> y[[[[[[0, 1, 0]]]]]].shape
(1, 1, 1, 3, 3)
>>> y[..., [2, 2, 1, 2]]      # Here we index on the other dimension
array([['w', 'w', 'o', 'w'],
       ['y', 'y', 'l', 'y']],
      dtype='|S1')
```

If you're indexing along multiple dimensions at once, the indices must be conformable, as if you were adding or multiplying them together, which in a sense you are (see above about `xs + width * ys`):

```
>>> y[[0, 1, 0], [2, 2, 1, 2]]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>

IndexError: shape mismatch: indexing arrays could not be broadcast together with o
oshapes (3,) (4,)
>>> y[[0, 1, 0, 0], [2, 2, 1, 2]]
```

```
array(['w', 'y', 'o', 'w'],
      dtype='|S1')
```

This can lead to some ambiguity about where the dimensions taken from the index should be merged into the dimensions of the thing being indexed; Numpy seems to prefer the earliest candidate position:

```
>>> a
array([[[0, 1, 2, 3],
        [4, 5, 6, 0],
        [1, 2, 3, 4]],

       [[5, 6, 0, 1],
        [2, 3, 4, 5],
        [6, 0, 1, 2]]])
>>> a.shape
(2, 3, 4)
>>> a[[[1]], ..., [[1]]]
array([[[6, 3, 0]]])
>>> _.shape
(1, 1, 3)
```

This can be quite surprising in the presence of broadcasting:

```
>>> a[[[1], [1], [1], [1], [1]], ..., [[1, 1, 1, 1, 1, 1]]].shape
(5, 6, 3)
>>> a[[[1], [1], [1], [1], [1]], ..., [1, 1, 1, 1, 1, 1]].shape
(5, 6, 3)
>>> a[..., ..., [[1, 1, 1, 1, 1, 1]]].shape
(2, 3, 1, 6)
```

I think the Numpy behavior of an insufficient number of indices is disharmonious with Numpy broadcasting behavior in the following sense. If you write a function like `lambda x: x * [3, 1, 5]`, you are in some sense expecting that the *last* dimension of `x` will be 3 (or possibly 1). And if you say `x * [[2, 3, 1], [4, 1, 5]]`, you are expecting that its last dimensions will be (2, 3) (or broadcastable to (2, 3); for example, (1, 1), (1, 3), or (2, 1).) As a general principle, this means that you can write a function that works on, for example, an RGB triplet, and then apply it to some large collection of RGB triplets (perhaps an array of shape (320, 240, 3)), and hope that it will serendipitously generalize to application elementwise. And as long as broadcasting is the only thing being applied, this works:

```
>>> p = np.array([127, 63, 127])
>>> (p * [3, 1, 5]).clip(0, 255)
array([255,  63, 255])
>>> p = np.array([[127, 63, 127], [121, 23, 21]])
>>> (p * [3, 1, 5]).clip(0, 255)
array([[255,  63, 255],
       [255,  23, 105]])
```

But this fails once indexing comes into play. For example, we

could extract the green channel of `p` with `p[1]` or possibly `p[[1]]`. But this only works in the first case above; in the second case, instead of extracting the red channel of each pixel, it extracts all three channels of just the first pixel.

Many other Numpy operations have the same problem. If we want the sum of the three components of the pixel, for example, `p.sum()` gives them to us; but `.sum()` applies implicitly over all axes by default:

```
>>> p = np.array([[127, 63, 127], [121, 23, 21]])
>>> p.sum()
482
```

And even if we specify a particular axis, the axes are counted from the left, not the right:

```
>>> p.sum(axis=1)
array([317, 165])
```

To get behavior harmonious with the broadcasting behavior, we must specify a negative axis number:

```
>>> np.array([[[127, 63, 127], [121, 23, 21]]]).sum(axis=-1)
array([[317, 165]])
```

Other operations have even stranger behaviors, like implicitly flattening the array if no axis is specified:

```
>>> np.array([[[127, 63, 127], [121, 23, 21]]]).cumsum()
array([127, 190, 317, 438, 461, 482])
```

If we want to form a sum table of the color channel of each pixel, we can specify axis=-2:

```
>>> np.array([[[127, 63, 127], [121, 23, 21]]]).cumsum(axis=-2)
array([[[127,  63, 127],
        [248,  86, 148]]])
```

For better or worse, this fails on a single pixel:

```
>>> np.array([127, 63, 127]).cumsum(axis=-2)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: axis(=-2) out of bounds
```

This desideratum of supporting serendipitous vectorization by implicit rank polymorphism probably requires redesigning the "reshape" operator mentioned earlier so that it won't accidentally break this vectorization.

Octave indexing and broadcasting

Octave has totally different behavior, implicitly flattening for indexing with a single index:

```
octave:24> y = ['how'; 'dly'];
```

```
octave:9> y([1 2 1])
ans = hdh
```

You do, however, get Numpy-like behavior when you supply both indices:

```
octave:13> y(:, [1 2 1])
ans =

hoh
dld

octave:14> y([1 2 1], :)
ans =

how
dly
how

octave:32> y(:, [3 3 2 3])
ans =

wwow
yyly
```

Moreover, for Octave, "all objects have a minimum of two dimensions", so indexing once into a vector is really indexing into the second dimension of a matrix:

```
octave:15> z = 'waltz'
z = waltz
octave:17> z([1 2 1])
ans = waw
octave:18> z([1 2 1], :)
error: A(I,J): row index out of bounds; value 2 out of bound 1
octave:18> z(:, [1 2 1])
ans = waw
octave:19> z([1 1 1], [1 2 1])
ans =

waw
waw
waw
```

Note that this last result shows that Octave is not broadcasting the indexes together the way Numpy does.

You can extend the matrix to an arbitrary number of dimensions, treating this 1×5 matrix as a 1×5×1 array, or 1×5×1×1×...:

```
octave:34> size(z([1 1], [1 2 1], [1 1 1], [1 1], [1 1]))
ans =

   2   3   3   2   2
```

```
octave:22> z([1 1], [1 2 1], [1 1 1])
ans =

ans(:,:,1) =

waw
waw

ans(:,:,2) =

waw
waw

ans(:,:,3) =

waw
waw
```

Note that the output here shows that Octave's index order is closer to Fortran order than to C order: the rightmost indices vary most slowly, not most quickly. This is consistent if you read down the columns of each displayed matrix, but if you insist on reading each row from left to right before proceeding to the next one, as if you were reading English rather than Chinese, then the first two dimensions are an exception. This is even clearer looking at the behavior of `reshape`:

```
octave:54> w = reshape(1:24, [2 3 4])
w =

ans(:,:,1) =

   1   3   5
   2   4   6

ans(:,:,2) =

    7    9   11
    8   10   12

ans(:,:,3) =

   13   15   17
   14   16   18

ans(:,:,4) =

   19   21   23
   20   22   24
```

As with Numpy, you can get an output with a more complicated shape by indexing *once* with a more complicated shape:

```
octave:27> z([1 2 4; 4 1 2])
```

```
ans =

wat
twa
```

However, this doesn't work if you're indexing multiple dimensions, in which case instead of implicitly flattening the thing you're indexing into, as above, you implicitly flatten each index, in Fortran order, giving an INTERCAL-like flavor in this case:

```
octave:41> z([1 1], [3 1 4; 1 5 1])
ans =

lwwztw
lwwztw

octave:36> size(z([1 1], [1 2 1], [1 1 1; 1 1 1]))
ans =

   2   3   6
```

I thought that maybe Octave's (or rather MATLAB's) implicit flattening is where Numpy gets its implicit flattening, but in fact Octave *doesn't* implicitly flatten in `sum`, `prod`, `max`, and `cumsum`, which implicitly apply along the fastest-varying axis, which happens to be the leftmost:

```
octave:48> sum([3 1 4; 1 5 9])
ans =

    4    6   13

octave:49> max([3 1 4; 1 5 9])
ans =

   3   5   9

octave:50> prod([3 1 4; 1 5 9])
ans =

    3    5   36

octave:51> cumsum([3 1 4; 1 5 9])
ans =

    3    1    4
    4    6   13
```

However, these don't decrease the dimensionality of the result; they just shrink one of its dimensions to size 1:

```
octave:58> size(w)
ans =
```

```
   2   3   4

octave:59> size(sum(w))
ans =

   1   3   4
```

You can specify a different axis for the aggregation, as in Numpy:

```
octave:73> size(sum(w, 2))
ans =

   2   1   4
```

What about broadcasting? Unlike in Numpy, it's consistent with `sum` and indexing, in that it left-aligns the dimensions rather than right-aligning them, although this is somewhat confusing if you forget that it considers an ordinary row vector to be 1×N:

```
octave:60> w + [100 1000]
error: operator +: nonconformant arguments (op1 is 2x3x4, op2 is 1x2)
octave:60> w + [100 1000 10000]
warning: operator +: automatic broadcasting operation applied
ans =

ans(:,:,1) =

     101    1003   10005
     102    1004   10006
...
```

Still, though, there is no possibility of getting serendipitous multiplicity generalization in Octave on a function that uses indexing; indexing with too few indices will flatten the omitted trailing dimensions down into the last dimension. This is a generalization of what happens when you index with just a single dimension:

```
octave:69> w(:, 10)
ans =

   19
   20

octave:70> w(:, 10, :)
error: A(I,J,...): index to dimension 2 out of bounds; value 10 out of bound 3
octave:72> w(:, 1, 4)
ans =

   19
   20
```

R indexing and broadcasting

R almost completely lacks the kind of rank-polymorphism I'm

looking for.

R, like Octave, uses Fortran order (and 1-based indexing), and implicitly flattens when you index a matrix with just one index:

```
> y <- c('h', 'd', 'o', 'l', 'w', 'y')
> dim(y) <- c(2,3)
> y
     [,1] [,2] [,3]
[1,] "h"  "o"  "w"
[2,] "d"  "l"  "y"
> y[1]
[1] "h"
> y[1,]
[1] "h" "o" "w"
> y[,1]
[1] "h" "d"
> y[c(1,2,1),]
     [,1] [,2] [,3]
[1,] "h"  "o"  "w"
[2,] "d"  "l"  "y"
[3,] "h"  "o"  "w"
```

Unlike in Octave, this really is a special case for a single index; you can index a 2×2×2 array with one index or three, but not two:

```
> j <- c(1, 2, 2, 1, 1, 2, 2, 1)
> dim(j) <- c(2, 2, 2)
> j
, , 1

     [,1] [,2]
[1,]    1    2
[2,]    2    1


, , 2

     [,1] [,2]
[1,]    1    2
[2,]    2    1
> j[4]
[1] 1
> j[2, ]
Error in j[2, ] : incorrect number of dimensions
> j[2, 1]
Error in j[2, 1] : incorrect number of dimensions
> j[2, 1, 2]
[1] 2
```

This thing where the structure of a complicated index is replicated in the output doesn't seem to be present; indexing by `j` above just flattens `j` into a vector of indices:

```
> y[j]
[1] "h" "d" "d" "h" "h" "d" "d" "h"
```

```
> y[j,]
     [,1] [,2] [,3]
[1,] "h"  "o"  "w"
[2,] "d"  "l"  "y"
[3,] "d"  "l"  "y"
[4,] "h"  "o"  "w"
[5,] "h"  "o"  "w"
[6,] "d"  "l"  "y"
[7,] "d"  "l"  "y"
[8,] "h"  "o"  "w"
```

Multiple indices are not broadcast together, as in Numpy, but instead give a Cartesian product, as in Octave:

```
> y[j,j]
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,] "h"  "o"  "o"  "h"  "h"  "o"  "o"  "h"
[2,] "d"  "l"  "l"  "d"  "d"  "l"  "l"  "d"
[3,] "d"  "l"  "l"  "d"  "d"  "l"  "l"  "d"
[4,] "h"  "o"  "o"  "h"  "h"  "o"  "o"  "h"
[5,] "h"  "o"  "o"  "h"  "h"  "o"  "o"  "h"
[6,] "d"  "l"  "l"  "d"  "d"  "l"  "l"  "d"
[7,] "d"  "l"  "l"  "d"  "d"  "l"  "l"  "d"
[8,] "h"  "o"  "o"  "h"  "h"  "o"  "o"  "h"
```

`sum` and `cumsum` flatten by default, as in Numpy:

```
> sum(p)
[1] 482
> cumsum(p)
[1] 127 190 317 438 461 482
```

There is no optional "axis" argument, as in Numpy and Octave; instead there are some special-case functions:

```
> colSums(p)
[1] 317 165
> rowSums(p)
[1] 248  86 148
```

Broadcasting left-aligns dimensions, as in Octave, but seems to be limited to scalars and vectors, and has truly bizarre behavior:

```
> p <- c(127, 63, 127, 121, 23, 21)
> dim(p) <- c(3, 2)
> p
     [,1] [,2]
[1,]  127  121
[2,]   63   23
[3,]  127   21
> p * c(3, 1, 5)
     [,1] [,2]
[1,]  381  363
[2,]   63   23
```

```
[3,]  635  105
```

So far, so reasonable. But look at *this*:

```
> p + c(1, 2, 3, 4, 5, 6)
     [,1] [,2]
[1,]  128  125
[2,]   65   28
[3,]  130   27
```

The vector got implicitly reshaped! Weirder still, given a 2-vector, it gets broadcast down columns instead of across rows --- or does it?

```
> p + c(1, 2)
     [,1] [,2]
[1,]  128  123
[2,]   65   24
[3,]  128   23
```

If the matrix is square so that the vector could be broadcast either horizontally *or* vertically, it gets broadcast horizontally:

```
> p[,c(1, 2, 1)] + c(1000, 10000, 100000)
       [,1]   [,2]   [,3]
[1,]   1127   1121   1127
[2,]  10063  10023  10063
[3,] 100127 100021 100127
> p + c(100, 1000, 10000, 5)
      [,1] [,2]
[1,]   227  126
[2,]  1063  123
[3,] 10127 1021
Warning message:
In p + c(100, 1000, 10000, 5) :
  longer object length is not a multiple of shorter object length
```

The horrifying truth is that it's just replicating the vector down the columns to "broadcast" it --- it wasn't applying it to columns after all! `p[,1] + 1` is c(128, 64, 128), not c(128, 65, 128) as given above. But even when *it doesn't fit*, you only get a *warning*.

At the other extreme, suppose you want to add a 2×2 matrix to our 2×2×2 `j` above. Nothing doing!

```
> i <- c(10, 100, 1000, 10000)
> dim(i) <- c(2, 2)
> i + j
Error in i + j : non-conformable arrays
```

Given the above, you'd think we could do that if `i` is just a vector, but no, apparently that implicit flattened replication is just for matrices:

```
> dim(i) <- 4
> i + j
```

```
Error in i + j : non-conformable arrays
> dim(j)
[1] 2 2 2
> dim(i)
[1] 4
```

We can still add a 2-vector to `j`, and it broadcasts horizontally and depthwise as expected.

```
> j + c(100, 1000)
, , 1

     [,1] [,2]
[1,]  101  102
[2,] 1002 1001

, , 2

     [,1] [,2]
[1,]  101  102
[2,] 1002 1001
```

And a 4-vector broadcasts depthwise:

```
> j + c(10, 100, 1000, 10000)
, , 1

     [,1]  [,2]
[1,]   11  1002
[2,]  102 10001

, , 2

     [,1]  [,2]
[1,]   11  1002
[2,]  102 10001
```

But we cannot add a 2×2-element array, or a 2-element array, to `j`, because in R, vectors and arrays are different classes of things that just happen to look exactly the same most of the time:

```
> k <- c(10, 100)
> dim(k) <- 2
> j + k
Error in j + k : non-conformable arrays
> k
[1]  10 100
> c(10, 100)
[1]  10 100
> class(c(10, 100))
[1] "numeric"
> class(k)
[1] "array"
> dim(c(10, 100))
```

```
NULL
> dim(k)
[1] 2
```

As far as I can tell, for arrays to be conformable, they must have exactly the same shape, with no broadcasting.

Program serialization as strings of bytes: let's use text!

The usual way to represent programs for a virtual machine is as some kind of binary bytecode. This is relatively fast to load, but it requires at least some kind of assembler to construct it from a human-readable format (if not a compiler from a higher-level language) and probably some kind of disassembler as well to help with debugging. (If the virtual machine is producing the wrong results on some program, you need some way to puzzle out what the program is telling it to do, in order to figure out whether the bug is in the program or the VM.)

I think that for this purpose it might be a reasonable alternative for the virtual machine itself to parse a simple textual syntax that is sufficiently friendly to write directly by hand and read with a text file viewer, even if it lacks some of the amenities one might want in a programming language. For example, you might use a syntax similar to PostScript, or FORTH, or Lisp.

Topics

Ballpoint SPIF

Kragen Javier Sitaker, 02020-04-25 (7 minutes)

At the Ohio State University, as in many other places, there is a giant solid ball of granite floating in a pool of water. This is a surprising sight, since granite is not known for its buoyancy, but it's real; you can spin this three-meter-diameter sphere around with your hand and feel its massive weight slowly easing into motion in precisely the way a giant granite boulder sitting on the ground does not do.

This remarkable phenomenon is the manifestation of a fluid bearing, like the air bearings commonly used in semiconductor-handling equipment or an air-hockey table, but in this case the joint is a ball-and-socket type. Water is pumped up underneath the boulder, lifting it just enough to allow the water to escape around its edge, where without water it would rest on a circular stone "valve seat" whose diameter is almost as large as that of the boulder --- exactly like a ball-bearing-type check valve, with gravity instead of the spring. Only enough water pressure is needed to support the average vertical thickness of the boulder, $(2/3)\,\tau r^3 / \frac{1}{2}\tau r^2 = r/3$, about half a meter; at 2.4 g/cc and 9.8 m/s/s, that works out to about 12 kPa. In theory the water flow rate at this pressure can be arbitrarily low; and lower water flow rates give higher positioning precision, but also reduce the "side loading" force needed to crash the boulder into the valve seat, incurring static friction and potentially scratching it. In practice the boulder is thus suspended using several liters per second of water, which means that only on the order of 100 watts is required to sustain this numinous apparition.

A really delightful attribute of fluid bearings of any kind is that they have no static friction: as the velocity approaches zero, so do the viscous losses in the fluid and thus the friction; thus the boulder is always rotating. So one of their key applications is for low-velocity kinematic pairs.

This brings us to single-point incremental forming, in which you shape a metal sheet by pushing a metal finger into it and moving it around. SPIF, like 3-D printing, is capable of producing a wide variety of different shapes with no per-shape tooling, but it can produce fully-dense forged sheet-metal pieces with no material waste and no postprocessing required, just like the more usual kinds of sheet-metal presswork, sometimes approaching the capabilities of deep drawing. The toolpath planning process is somewhat more involved than for 3-D printing because you need to do a FEM analysis of candidate toolpaths to anticipate when they would cause the metal to overheat, wrinkle, tear, or get too thin.

In particular, friction with the forming tool is a major obstacle, and can be unpredictable. Typically the tool is a round shank of tungsten carbide with a hemispherical end that is polished smooth, and to reduce friction this tool is both lubricated with oil and rotated as it moves around the work.

It occurs to me that the floating-boulder trick offers a far more

expedient alternative, if you shrink it down and crank up the pressure. Instead of the round end of a carbide shank, you use a floating ball bearing --- ideally ceramic, but maybe just metal --- and support it in a liquid bearing that presses it against the workpiece. This allows you to roll it around the workpiece like the ball of a ballpoint pen rolls around on paper, entirely eliminating static friction and greatly reducing dynamic friction. (Ballpoint pen balls do have static friction because the ink isn't pressurized, so the analogy isn't perfect.)

The lubricant pressure needs to slightly exceed the average pressure across the contact area between the tool and the workpiece, which probably comes within about an order of magnitude of the yield stress of the workpiece metal, perhaps tens of MPa (ASTM A36 structural steel is supposed to have a yield stress of 250 MPa).

Lacking any experience with SPIF, my reasoning is as follows. If your tool diameter is small compared to the metal sheet's thickness, then it won't be able to form the whole sheet; it will just make an indentation into one side, which is not what we want. Also, if the tooltip is made of a similarly hard material, it will be deformed just as much as the workpiece, which is very much not what we want. If the tooltip diameter is a few times larger than the workpiece's thickness, then the pressure applied across the whole tooltip face sums up to a tensile force that is resisted by a ring of workpiece material around the outside of the tooltip, and perhaps only on some sides of it, and this allows you to do SPIF as desired. But if the tooltip diameter is many times larger, you will be needlessly giving up surface precision, and additionally you will have a greater tendency to just move the workpiece around and rip it rather than forming it as intended.

So I tentatively conclude that you probably want the tooltip diameter to be a few times larger than the workpiece thickness, but not two or more orders of magnitude larger. So, for example, if you are indenting an 0.2-mm-thick sheet with a 3-mm ball, you might need to, very roughly, overcome the yield strength in an 0.2-m annulus around the 3-mm ball using the pressure across the 3-mm circle; this requires about 1/14 of the yield stress to be present across the surface of the ball, about 18 MPa (2600 psi in archaic units) in the case of 250-MPa steel. This is a feasible but challenging and somewhat hazardous pressure for hydraulic fluids --- particularly when the lube is going to be squirting every which way --- so where feasible it would be nice to use a tooltip that is larger in comparison to the gauge of the stock.

(Of course, in reality the pressure distribution is not uniform across the face of the tooltip, nor is the tension distribution uniform around the outside.)

Topics

Bitwise reproducibility

Kragen Javier Sitaker, 02020-04-25 (1 minute)

The idea of reproducibility I want to base Derctuo on requires some explanation, since there isn't anything else out there that aims at this, as far as I can tell.

The objective of Derctuo's virtual-machine design is that running the same program with the same inputs always reproduces bitwise-identical outputs, unless it fails; that this should be the case even when executed on independent cleanroom reimplementations from the specification, whether this year or in 300 years, on the same hardware or different hardware; that implementing the virtual machine from the spec should require only a few hours of work; and that this virtual machine should be sufficient to reproduce all the computations I think are interesting.

Topics

Reversible parsing

Kragen Javier Sitaker, 02020-05-11 (6 minutes)

In Prolog you can write definite clause grammars, which make it very straightforward to write grammars, which can then be used both for text generation and for parsing:

```
: user@debian:~/devel/dev3; swipl
% library(swi_hooks) compiled into pce_swi_hooks 0.00 sec, 3,856 bytes
Welcome to SWI-Prolog (Multi-threaded, 64 bits, Version 5.10.4)
Copyright (c) 1990-2011 University of Amsterdam, VU Amsterdam
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).

?- [user].
det --> [the] | [a] | [that].
|: noun --> [buffalo] | [capacitor] | [philosophy].
|: vi --> [sucks] | [is, walking] | [glows].
|: vt --> [supersedes] | [clobbers] | [loves].
|: sentence --> det, noun, vi |
|:  det, noun, vt, det, noun.
|:
% user://1 compiled 0.00 sec, 4,816 bytes
true.

?- phrase(sentence, S), append(Y, Z, S), append(X, [buffalo], Y).
S = [the, buffalo, sucks],
Y = [the, buffalo],
Z = [sucks],
X = [the] ;
S = [the, buffalo, is, walking],
Y = [the, buffalo],
Z = [is, walking],
X = [the] ;
S = [the, buffalo, glows],
Y = [the, buffalo],
Z = [glows],
X = [the] ;
S = [a, buffalo, sucks],
Y = [a, buffalo],
Z = [sucks],
X = [a] ;
S = [a, buffalo, is, walking],
Y = [a, buffalo],
Z = [is, walking],
X = [a] ;
...
S = [the, buffalo, loves, that, capacitor],
Y = [the, buffalo],
Z = [loves, that, capacitor],
```

```
X = [the] ;
S = [the, buffalo, loves, that, philosophy],
Y = [the, buffalo],
Z = [loves, that, philosophy],
X = [the] ;
S = Y, Y = [the, capacitor, supersedes, the, buffalo],
Z = [],
X = [the, capacitor, supersedes, the] ;
S = Y, Y = [the, capacitor, supersedes, a, buffalo],
Z = [],
X = [the, capacitor, supersedes, a] .

?-
```

A disadvantage of DCGs is that, in standard Prolog, they don't terminate on left recursion and can take exponential time, although cuts can tame the exponential and I think tabled resolution can conquer both in some cases ("DCGs + Memoing = Packrat Parsing, But is it worth it?" by Ralph Becket and Zoltan Somogyi.)

Hmm, clearly I have a lot to learn about Prolog DCGs... Markus Triska's tutorial, Anne Ogborn's tutorial, the SWI-Prolog manual, and so on.

Anyway, what I was thinking was that for very straightforward kinds of "grammars", even a perfectly ordinary imperative language suffices:

```
void employee_card(card *s, employee *e)
{
  int_columns(s, 0, 6, &e->empno);
  columns(s, 6, 16, e->firstname, sizeof e->firstname);
  columns(s, 16, 26, e->lastname, sizeof e->lastname);
  blank_columns(s, 26, 80);
}
```

This plain C function could be invoked either for input or for output, if `card` contains a flag that indicates the direction and the `int_columns` and `columns` functions consult that flag. And similar bidirectional serialization/deserialization functions can be built for a wider class of grammars. Field widths need not be fixed, and field concatenation can be implicit:

```
void employee_csv(stream *s, employee *e)
{
  int_field(s, &e->empno);
  text(s, ",");
  delim_s_field(s, e->firstname, sizeof e->firstname, ',');
  delim_s_field(s, e->lastname, sizeof e->lastname, '\n');
}
```

Again, this function can be used either for input or for output, and multiple such functions can be composed together. If we add a little bit of backtracking, we can get polymorphic records:

```
void foo(stream *s, thing *t)
```

```
{
  begin(s);
  {
    equal_int(s, &t->type, TYPE_BAR);
    byte(s, 'B');
    nbytes(s, &t->bar.contents, sizeof t->bar.contents);
  }
  or(s);
  {
    equal_int(s, &t->type, TYPE_QUUX);
    byte(s, 'Q');
    s16_le(s, &t->quux.len);
    nbytes(s, &t->quux.contents, t->quux.len);
  }
  end(s);
}
```

On input, the calls to `equal_int` function as assignments to an integer field, while the calls to `byte` function as assertions about which byte comes next in the input; if one of these assertions fails, its effect on the input stream is backtracked, so that a subsequent call to `byte` can test the same input bytes. The backtracking state is set up by `begin`, restored by `or` in case of failure, and torn down by `end`.

On output, the situation is precisely the other way around: the calls to `equal_int` function as assertions about what should be found in `t->type` for that branch to proceed successfully, while the calls to `byte` emit literal bytes on the output — bytes which are buffered so they can be retracted if the case must be backtracked due to a subsequent failed assertion.

But this is still a very simple case; in particular it does not handle allocation, which in a C-like language probably must be part of the state restored in case of backtracking.

You could consider something like

```
child_node(s, &t->child, sizeof struct fulano);
struct fulano *f = (struct fulano *)t->child;
equal_int(s, &f->type, TYPE_FULANO);
byte(s, 'f');
decimal_int(s, &f->x);
byte(s, ' ');
decimal_int(s, &f->y);
```

where `child_node` creates a new allocation on input (deallocated in case of backtracking) and does nothing on output. But consider the infix expression

```
3 + 1000/2/2/2/2/2
```

which in prefix notation is

```
(+ 3 (/ (/ (/ (/ (/ 1000 2) 2) 2) 2) 2))
```

so unfortunately we have to read the rest of the input before we

know how deep down the tree 1000 goes. I'm not even sure Prolog DCGs handle this case in this form. I wrote a toy calculator program tonight to explore some of the above ideas, and I refactored the grammar to eliminate left recursion; here's a simplified form of how it parses terms like `1000/2/2`:

```
int term()
{
  int x = unary();

  begin();
  while (ok()) {
    /* save() ensures that our progress so far will not be backtracked */
    save();      /* zero or more multipliers, divisors, and modulos */
    begin();
    {
      op("*");
      int y = unary();
      if (ok()) x *= y;
    }
    or();
    {
      op("/");
      int y = unary();
      if (ok()) x /= y;
    }
    end();
  }
  end();

  return x;
}
```

`x /= y` in an AST would be something like `x = new_division_node(x, y)`. But it's deeply unclear to me how to make that work bidirectionally: the sequence of the input text is bottom-up, while normally the structure of an AST is top-down.

A somewhat related thing is operator precedence and associativity. If we take `+` to be associative, it might be reasonable to serialize both `(+ 1 (+ 2 3))` and `(+ (+ 1 2) 3)` in infix as `1 + 2 + 3`, but clearly `(- 1 (- 2 3))` is `1 - (2 - 3)` while `(- (- 1 2) 3)` is conventionally `1 - 2 - 3`. Similarly, precedence dictates that `(+ 3 (* 4 5))` can be `3 + 4 * 5`, but `(* (+ 3 4) 5)` requires extra parentheses: `(3 + 4) * 5`.

Topics

Bloomtags: a Bloom-filter tree for efficient and flexible database queries

Kragen Javier Sitaker, 02020-05-13 (21 minutes)

Suppose you have a large file of lines tagged with hashtags and you want to efficiently iterate over the lines satisfying a given hashtag intersection. What kind of index structure supports this?

You could use a tree of Bloom filters with a relatively high false-positive factor and add additional "synthetic tags" to improve precision for certain pathological queries. This seems like it will probably give reasonable efficiency, and it has some significant efficiency advantages over existing database indexing approaches for evaluating some kinds of queries.

The example problem: 8.5 billion lines of data with 8 hashtags each

For concreteness, let's suppose you have a tebibyte of 128-byte lines, each of which is tagged with 8 hashtags, which follow a perfect Zipf distribution, with the most common hashtag occurring in 25% of all lines, so the next most common ones are in 12.5%, 8.3̄%, 6.25%, 5%, 4%, etc., of all lines. So in total there are 8 gibilines. Let's suppose that the distribution of hashtags is otherwise uniform and uncorrelated, for example with 3⅛% of the lines being tagged with both of the two most common tags.

There are perhaps 4 gibihashtags, although the majority are one of the most common hashtags. So you can store a hashtag in 32 bits, but you might be able to get away with a lot less in the average case, so the 8 hashtags per line take up 32 bytes per line.

If the file is divided into 4-kibibyte blocks, there are 256 mebiblocks in the file. Reading any one of these blocks from a modern SSD costs about 50 µs (270 µs on the machine I'm using, with 120 megabytes per second throughput giving about a 32-kibibyte bandwidth-delay product, but it's second-rate, and most SSDs have both more bandwidth and many more iops, bringing theirs closer to 4K — which is the smallest request size they support anyway), and iterating through the 32 lines in it to determine whether they contain the hashtag; this is less than 100 instructions, so it'll be bottlenecked on main-memory bandwidth, which in turn is probably bottlenecked on SSD bandwidth.

I'm informed that NVMe devices get close to 1 GiB per second with 4k reads, and PCIe Optane devices can get 2.5 GiB per second (the PCI controller limit) with 4k reads, implying upwards of 600k iops. So, doing the query by sequential scan on an Optane drive would take 400ms per gibibyte and thus 410 seconds.

Well, a thing we can already do to improve the situation is to segregate a hashtags column or index elsewhere; it's only a fourth of

the total file size, so we can fit the hashtags of 128 lines into each 4-kibibyte block. This would get our query time down to 102 seconds.

Most hashtags are extremely specific, occurring in only a single line. If we have a query for such a hashtag, it would be nice to be able to follow a tree of Bloom filters down to the single hashtag-column block that contains the single line with that hashtag.

Bloom filter background

A Bloom filter is a bit vector. An m-bit Bloom filter for n keys e_0, … e_{n-1} with k independent hash functions h_0, … h_{k-1} such that $\forall i, j$: $h_i(e_j) \in [0, m)$ is a vector of m bits b_p which are 1 precisely when $\exists i, j$: $h_i(e_j) = p$, but 0 otherwise. That's all! You can see that if the h_i are random enough and m is large enough, then for some key d not in the set, you can usually find some bit in the filter that is 0, but would have been 1 if d were in the set; but some false-positive probability always exists, depending on k and the load factor f (the fraction of 1 bits), specifically f^k. Typical values of the bits-per-element parameter $c = m/n$ range from 2 to 16, and typical values of k are also about 2 to 16.

```
bh = lambda i, e: hash((i+1)*hash(e))  # circumvent Python's weak hash()
bloom = lambda m, k, e: ([1 if any(bh(i, ej) % m == p
                                   for i in range(k) for ej in e) else
                          0 for p in range(m)], k)
in_bloom = lambda (bits, k), e: all(bh(i, e)) % len(bits) == 1
                                    for i in range(k))
```

As Norm Hardy explains, there are a lot of nice tricks you can do with Bloom filters. Two of the relevant ones are unioning and folding.

```
def bloom_union((bits_a, k_a), (bits_b, k_b)):
    assert k_a == k_b
    return [ai | bi for ai, bi in zip(bits_a, bits_b)], k_a
```

You can OR several Bloom filters together, with or without a bit shift or bit rotation; the result is a Bloom filter with a higher load factor and consequently a higher false-positive rate that can be efficiently queried to determine if any of the child filters might be capable of containing the query key. With the shift or rotation, you can also determine which.

You can also *fold* a Bloom filter: take it and OR its two halves together to get a smaller Bloom filter with one less bit of address, and also a higher load factor and false-positive rate. For example, if you initially compute 64 Bloom filters with a load factor of 1.08%, you can OR them together to get a single Bloom filter of the same size with a 50% load factor, or you can fold one of them six times to get a 64×-smaller Bloom filter with the same set of keys as the initial filter but the same 50% load factor.

The Bloom-filter tree index structure

So suppose we take our 256 mebiblocks, each containing 32 lines with 8 hashtags each, and compute a gigantic Bloom filter for each one. We divide these into 64 groups of 4 mebiblocks, OR together the filters, and then rotate-and-OR together all these filters to produce a single master filter for the whole file with a 50% load factor, which when queried will tell us which of these 64 groups might contain the key. If we are satisfied with a 1/128 false-positive rate, we can use seven bits per key (i.e., seven hash functions). All together, this gives us 7 × 256Mi × 32 × 8 bits to set in this master filter to reach the 50% load factor, which works out to about 4.81×10^{11}, so we need about 690 gigabits, 87 gigabytes, in this master filter and in each of the 64 group filters. You can verify that `math.exp(math.log(1 - 1/690e9)*(7 * 256 * 2**20 * 32 * 8))` is about 0.5 in Python. It is probably most practical to compute these large filters in a blocked fashion, redundantly rehashing the whole file each time and discarding the hash values that fall outside of the current block.

Now by probing our 87-gigabyte master filter seven times with seven random reads, we can almost determine which of the 64 4-mebiblock groups contain a rare hashtag: we'll have on average 1.5 hits, one real one and 0.5 false positives on average. Each of these groups has a 1.4-gigabyte filter as well — but these aren't simply folded versions of the original 64 filters, but rather versions built from 64 smaller subfilters which are rotated before being added together.

So in this way, with 87 gigabytes per level of the tree, we have a five-level tree of Bloom filters which allow us to rapidly follow the trail down to an individual 4-kilobyte block of 32 lines; individual filters at each level cover respectively 256Mi, 4Mi, 64Ki, 1Ki, and 16 blocks, with sizes of respectively 87GB, 1.4GB, 21MB, 330KB, and 5KB per filter. If we must do on average 11 probes per intermediate level (7 in the correct block and 4 or so in the false positive, which half the time will contain no false positive) then our tree traversal requires respectively 7, 11, 11, 11, and about 2 block reads, for a total of 42 block reads, about a third of a second on classic spinning rust, 12 milliseconds on the SSD I have here, or 70 µs on a PCI Optane device. This is between 200 and a million times faster than the same query without the index.

XXX you don't have to keep probing once you've found a cleared bit; you'll only on average probe a node that was a false positive in its parent less than twice, in the case of 7 hashes 1 + ½ + ¼ + ⅛ + 1/16 + 1/32 + 1/64 = 1+63/64. Not 3. This means it's 39, not 42.

The total index tree is only 440 gigabytes, sizable but less than half as big as the original file.

For lower-selectivity hashtags, a filter probing sequence of the same length will yield not the sole matching line but the first of many matching lines.

An interesting thing to note is that queries for arbitrary monotonic† Boolean combinations of hashtags still require visiting only the same number of nodes to reach the first record of results, but probing more hash buckets in each node, in proportion to the number of hashtags that need to be inspected. This makes ordering by selectivity much less important than with traditional database index structures — although it still helps to check the most selective hashtags

first, the speedup for a query testing N hashtags is at most only a factor of N.

Like ordinary Bloom filters, this filter tree can be readily updated for insertions but not for deletions or updates.

† "Monotonic" here is equivalent to "can be expressed with only AND and OR", excluding connectives like negation, abjunction (set subtraction), and material implication.

Synthetic tags

A difficulty with the naïve approach is that intersections of common tags will be extremely common at the higher levels of the tree, but can still be rare in the leaves. The #250-most-popular tag, for example, will be present in 0.1% of lines, as is the #251 most popular, but (given our hypothesis that tags are uncorrelated) the combination is present in only one line in a million, some 268 lines in all. Yet the vast majority of tree nodes will have at least one descendant line containing each of these tags; even at the last level, almost half of them will. The solution is to generate another few million "synthetic tags" consisting of such combinations: all the pairs of the most popular few hundred tags, triplets in cases where the pairs are insufficiently rare, a few quadruplets, perhaps a quintuplet or two.

Considering different branching factors

What if we change the branching factor? We will start to run into efficiency problems once we are beyond a few machine-word-sizes of branching: 1024-way branching might be feasible, but 4096-way branching requires operations on vectors of 4096 bits and, thus, suffering. Let's consider branching factors of 256, 16, 8, and 512.

With a branching factor of 256, we need 4 levels of tree instead of 5, but 9 hashes instead of 7 (for a 1/512 false-positive rate), so each level takes 9/7 as much space for the same load factor, and we must probe each node in 9 places instead of 7. This works out to be very nearly equal to the 64-way branching case: a factor of 36/35 on size and slowness.

With a branching factor of 16, we need 7 levels of tree instead of 5, but only 5 hashes instead of 7 (for a 1/32 false-positive rate), so each level is only about 5/7 the size, and we only have to probe each node in 5 places instead of 7, so returning the first record from a query still requires about 44 block reads. This is exactly equal to the 64-way branching case.

With a branching factor of 8, we need 10 levels of tree instead of 5, but 4 hashes instead of 7, so each level is 4/7 the size and requires 4/7 the probing. This is slightly worse: 40/35 on both size and slowness.

With a branching factor of 512, we still need 4 levels of tree, except that the first level only has a branching factor of 2, which is silly; and we need 10 hashes instead of 7, so each level is 10/7 the size and requires 10/7 the probing, for a total factor of 40/35 on both size and slowness. This is a little worse than the factor-256 case, but only because it's 4 levels instead of 3. If the file were half as big, it would be 30/35, which is still almost equal.

This null result for varying branching factor by a factor of 64 is not what I expected! What if we consider far more extreme cases?

How about using a single Bloom filter with a branching factor of 268'435'456? Well, we probably need to crank up its precision a bit (from the 2^{-29} the above would suggest, using 29 hashes), or it will return us half the blocks in the file as false positives. (Above I was assuming that 50% false positives would be fine.) And each probe will be reading a vector of 256 mebibits (32 mebibytes) out of the filter, to be rotated and ANDed with the other probe results. So we need to do, say, 58 probes with 58 hash functions, doing 58 random reads of 32 mebibytes each, a total of 1.8 gibibytes, sucking up a few seconds of memory bandwidth. But then we have a giant bitvector that tells us exactly which couple of lines we need to look at to find the one we're interested in.

(XXX how much space does this use? Maybe it's less?)

This is worse than the more reasonable cases, but only because of the lower false-positive rate demanded and the larger bitvectors being transferred — the raw number of probes is still almost the same! It's within a factor of 2.

How about the other extreme — a branching factor of 2, false-positive probability of 1/4, thus probing each filter twice? Here each level of the filter needs to be about 200 gigabits or 25 gigabytes, about 2/7 of the size previously needed, but we need 28 levels instead of 7, so 56 probes. This is also slightly worse than the middle-of-the-road sizes mentioned above, but, again, by less than a factor of 2. This extreme, unlike the other one, is actually practical, just slightly suboptimal.

Blocked Bloom filters

"Cache-, Hash-, and Space-Efficient Bloom Filters" proposes "blocked Bloom filters", a slight variation on a normal Bloom filter that improves locality of reference. (This is also the paper that proposed Golomb-coded sets.) The idea is that, instead of scattering the bits for the *k* different hash functions for a single key all over a huge Bloom filter, you use the first few bits of hash output to pick a block of, say, 64 bytes, and then use the *k* different hash functions to index bits within that block. In theory, as long as *k* is small compared to the number of bits in the block, the performance difference is tiny between an ordinary Bloom filter and this variant.

This analysis mostly survives the adaptations to the Bloom-filter algorithm described above, and it has even greater advantages in the SSD or spinning-rust milieu. It has no trouble with folding large sparse filters into small dense filters. However, it does suffer somewhat from rotating and combining multiple filters. In all but the bottommost tree nodes, all the bits related to a high-frequency hashtag will be set, forming a (say) 64-bit word of all ones. If you have, say, 7 such words within a single 512-bit block, they will by themselves push that block's load factor above 60%, before any other keys are inserted. So it is not *k* that must be small compared to the block size, but 64_k_, or whatever the branching factor is.

The obvious thing to try is to use blocks of 4096 bytes, the disk's transfer size, rather than 64 bytes.

Using blocked Bloom filters means that probing for a single key in a single tree node requires only a single disk access, no matter how large *k* or the branching factor are, so, for example, our 5-level tree from before can be traversed in 5 random accesses rather than 39. This might ease the pressure towards smaller branching factors, perhaps favoring 512-way branching — wider branching factors don't save you any space but they do reduce access time!

Multiattribute queries and range queries

The above is all formulated in terms of "hashtags": each "line" has some set of hashtags. But what happens if we're considering records in a more traditional database? You might have a record like { "lat": -34.5384, "lon": -58.4636, "name": "Escuela Superior de Mecánica de la Armada", "neighborhood": "Nuñez", "city": "Buenos Aires", "country": "Argentina", "category": ["Internment camp", "Museum"] }, and you might want to query, for example, a list of museums in Argentina.

It's straightforward to transform each name-value pair into a "hashtag" such as "#country:Argentina" and "#category:Museum", generating multiple hashtags for multivalued attributes like "category" (which would be represented as a join table in an RDBMS). This combination of hashtags could then be used to walk the index tree to find the records; I think this is likely to be a little faster than doing the equivalent with ordinary database indices, because parts of the tree that have museums but nothing in Argentina, or things in Argentina but no museums, can be skipped over completely, while traditional database query plans can only skip over one or the other, (unless a multicolumn index happens to exist beginning with that pair of columns), and must heuristically guess which index will be more selective. But, for reasonably common hashtags, you'll still have to visit most of the nodes in the tree, unless the file happens to be sorted in a way that brings them close together.

The latitude and longitude fields, though, pose more of a problem, because it's unlikely that someone would query for "#lat:-34.5384" exactly. A much more likely scenario is retrieving latitudes in the range of -34.52 to -34.55 and some similar range of longitudes — the neighborhood including ESMA, Ciudad Universitaria, and the River Plate stadium.

One way to deal with this problem is to shatter the tag into "#lat:-0xx", "#lat:-3x", "#lat:-34.x", "#lat:-34.5x", "#lat:-34.53x", "#lat:-34.538x", "#lat:-34.5384x", and "#lat:-34.5384". This converts a single attribute value into 8 separate hashtags, a number which grows logarithmically with the number of distinct values in the file. Then, any contiguous range query on that field can be expanded into a query of one to eight of these tags with, at most, only about a 20% loss of precision.

Topics

Static hypertext on CCN

Kragen Javier Sitaker, 02020-05-16 (2 minutes)

Implementing a static hypertext system on top of a service like the retrieve-by-hash service described in Feeds or streams on CCNs (p. 52) is straightforward.

Static hypertext

Each hypertext page is a file stored in the system, consisting of a short metadata header followed by the page itself in a format such as HTML or PDF, and it is identified by its hash. Links to another page include the hash `H` of the file it's stored in. In this way, you can be certain that the linked page is precisely the version of the page that the author intended; no attacker can redirect you to a different page, not even the author herself at a later date, perhaps while being tortured by Mossad agents.

Of course, if the attacker can trick you into looking at a page of theirs instead, they can make a copy of an authentic page with all the links redirected to more pages they wrote. So all the security comes from the security of the initial link.

This secure linking mechanism is also applicable to things like stylesheets, image liabilities, software libraries, software configuration files, and text transclusions. In combination with a deterministic archival virtual machine with immutable semantics, this guarantees the interpretability of XXX

A single file can easily contain multiple different "pages", as TiddlyWiki does; the fragment-identifier mechanism of the XXX

A manifest mechanism XXX

Cache timing side channels XXX

Threat model

Topics

- Security (p. 811) (11 notes)
- Protocols (p. 813) (10 notes)
- Hypertext (p. 886) (3 notes)
- Content-centric networking/named-data networking (p. 974) (2 notes)

Feeds or streams on CCNs

Kragen Javier Sitaker, 02020-05-16 (15 minutes)

Suppose we take a Kafka approach with Merkle trees to publishing activity streams. If we presuppose an existing decentralized reliable retrieve-by-hash service that returns stored files when presented with a hash of their contents, we nearly have a workable decentralized publishing system. All that's needed is an unreliable publish-subscribe system to provide updates, and coupled with an aggregation system, it can work even at very low bandwidth.

The retrieve-by-hash service

The retrieve-by-hash service provides a single function, `get`; given some hash H, `get(H)` returns a blob (a file, a byte string) of some arbitrary size that hashes to `H` using some secure hash function. To be concrete, let's say it uses SHA-256 with the high bit set to 0, so the hash is fixed at 32 bytes.†

The interpretation of application blobs is outside the scope of this note, except to note that they can contain the hashes of other blobs and may also contain things of interest such as text, computer software, historical stock prices, or pleas for help; and they can be encrypted. In Static hypertext on CCN (p. 51) are some thoughts on building a hypermedia and software archival layer on top of this simple service. The implementation of the service is also outside the scope of this note.

However, since the hash is computed over the contents of the blob, it is (conjectured) infeasible to compute the hash for a blob whose contents are to be chosen in the future. So no blob can contain the hash of a blob that was created later; hash references can refer only to past information, not future information. So this service does not provide any way to find out whether something has happened, such as whether the Bitcoin price has exceeded US$10000 again yet, or to send or receive messages, or to update any information.

But a small notification message delivered over a publish-subscribe channel — the size of a single hash, or even a bit less — is sufficient to link to an arbitrarily large quantity of data stored in the service up to the time when the message was sent. The notification need not itself contain a signature, since it can link to a signature stored in the blob store, but it may be convenient to include a signature so that subscribers need not sort through spam or malicious notifications.

† The high bit is set to 0 to preserve the option of upgrading to other algorithms in a possible future where SHA-256 is broken; new kinds of hashes can be added in a backward-compatible fashion by setting their high bits to 1, and a successful attack on SHA-256 then cannot replicate those hashes. Security against Kardashev Type 3 adversaries probably requires a longer hash, but 255 bits should be enough for a Kardashev Type 2 adversary with quantum computers.

The Kafka architecture

Kafka pretends to be a publish-subscribe system, but it's really an

append-only fileserver. In a publish-subscribe system, subscribers ("clients" or more specifically "consumers" in Kafka lingo) subscribe to channels ("topics") and are notified immediately of new events published ("produced") on those channels. The way this works in Kafka is that a consumer makes a TCP connection to the server ("broker") that is the "leader" for a channel† and "fetches" new events on that channel, which is described as an "ordered 'commit log[]'". Each message added to this log is assigned a sequence number called an "offset"; the message's payload is an opaque byte array with a small amount of header metadata.

There are some 46 request types in the Kafka protocol, but we are only concerned with the requests Fetch and ListOffset. `Fetch(replica_id=-1, max_wait_time, min_bytes, topic_name, partition, fetch_offset, max_bytes)` returns all the messages on `(topic_name, partition)` starting from `fetch_offset`, waiting up to `max_wait_time` to finish responding if `min_bytes` bytes are not initially available, with a response size limit of `max_bytes`. `ListOffset(replica_id=-1, topic_name, partition, time=-1)` fetches the "log end offset" that will be assigned to the next message posted to the channel `(topic_name, partition)`, or optionally returns the offset of the oldest retained message `(time=-2)` or the oldest message before a given timestamp (given as the value of `time`.)

So, when you first connect to a Kafka broker, you can ask it what messages are retained on a channel with ListOffset, and then you can fetch some or all of those messages with Fetch, and you can send a Fetch request to get any future messages. It's up to you to remember what offset you have gotten messages up to; the broker doesn't know and doesn't care. If you lose a connection and reconnect, you can send another Fetch with the offset of the next message you haven't gotten yet, and it may return immediately, or it may block for up to `max_wait_time` — and its unblocking is the form of asynchronous notification provided by Kafka.

A benefit of not storing your session state on the server is that server failures can't lose it, and server security problems can't corrupt it; when the new server comes up, you can just continue reading messages where you left off. As long as the assignment of offsets to messages remains consistent, there is no risk that a message will be duplicated.

By waiting for consumers to fetch messages in this way, Kafka has less overload conditions — consumers that begin falling behind do not consume excessive network buffers or other memory on the server, nor do their TCP connections time out.

Kafka unfortunately does have problems in which messages can be duplicated because of flaky connections between brokers and *publishers* ("producers"). The problem is that the offset is assigned by the broker, not the publisher, because multiple publishers can publish to the same channel. Jay Kreps felt that this was a reasonable tradeoff given that the alternative would be potentially the assignment of conflicting offsets every time any of thousands of disks across LinkedIn's server farm failed. But recent versions of Kafka have added an additional set of producer IDs and sequence numbers to support message deduplication in this case.

A big issue in some environments would be that, since the Kafka

broker doesn't know what consumers might exist, it can't safely delete messages once they've been delivered. Kafka mostly tries to solve this problem by encouraging you to store your messages on cheap multi-terabyte spinning rust, and reducing the cost of that as much as possible, so you can delete your messages a week or two out instead of after a few hundred milliseconds.

Due largely to this design, Kafka has been historically the fastest message queue system out there. RabbitMQ can handle 100,000 messages per second on a normal PC, ØMQ can handle about 2.5 million (without persisting them to disk), and Kafka is just as fast while persisting to disk and replicating, and can scale up from there, for example to 7 million messages per second across a cluster at Criteo . Apache Pulsar is a new alternative designed to be faster.

And, fundamentally, all Kafka is doing is allowing producers to append batches of messages to a logfile, and allowing all the consumers to read that logfile and get notified when it gets extended. Is there a way we can provide that service with a decentralized system?

† The broker is actually the leader for, in Kafka terminology, a "partition" of a "topic", in order to support load balancing of a single topic both among brokers and among subscribers, but this is a useless epicycle; since it's transparent neither to publishers nor subscribers, it's equivalent to just using multiple topics. As for leaders, partitions have leaders because Kafka is a clustered system that automatically replicates data across a cluster of Kafka servers, but it is essential to avoid the assignment of the same offset in the same channel to two different messages.

Reading a logfile with a Merkle tree

A Merkle tree node is either an "internal" blob containing just the word "tree" followed by zero or more hashes, or a "leaf" blob containing just the word "leaf" followed by some data. The value of the second one is the data after the word "leaf"; the value of the first one is the concatenation of the values of the blobs to which the hashes refer. Given a long string, we can compute a Merkle tree for that string made of blobs of about 4 kibibytes by first dividing it into 4-kibibyte chunks, prepending "leaf" to each of them, then creating a first level of internal blobs representing concatenations of up to 128 of these leaf blobs (up to 512 kibibytes per first-tier internal blob), and if there's more than one of those, then creating a second tier of internal blobs of up to 128 of the first-tier internal blobs (up to 64 mebibytes per second-tier internal blob), and so on until you have a tier that consists of just one root node.

Suppose you have the hash of a Merkle tree node that you somehow know that is the root node of some version of Barbara's event log. So you get the corresponding blob from the retrieve-by-hash service and look at it. If it's a leaf node, you already have the whole event log, but if it's an internal node, you have to get the blobs it refer to from the retrieve-by-hash service if you want to read the log contents.

Suppose Barbara wants to publish another event. She appends the event to the end of her event log, perhaps just appended onto the last

leafblob or perhaps broken into many leafblobs, and then makes a new version of its parent internal node, and if any, *its* parent, and so on. Then somehow she publishes the hash of this new node.

Now, suppose you get another hash that you somehow know represents the root of this new version of Barbara's event log. You get it; both this one and the other one are internal blobs. You can look at the hashes to see which ones are new, and fetch just those. As long as Barbara has published less than 64 mebibytes so far, you only need to fetch two levels of internal blobs (an overhead of 8 KiB, plus 1/128 of the weight of Barbara's new data), plus Barbara's new data.

We can augment the Merkle tree internal blobs with size information for each subtree so that it becomes easy to navigate to a particular offset. We can augment the root blob with a cryptographic signature so that, if you somehow get hold of the root blob hash, you can verify that Barbara did indeed publish that version of her event log, with no further information. We can make a long string of such signed root blob hashes for different people, each labeled with that person's public key hash, and make a Merkle tree of *that*, and publish *its* root blob hash. But how do we get that root blob hash out to the masses? The retrieve-by-hash service can't do it.

The paging channel

Traditional phone networks work by setting up a "call", a reserved fixed-bandwidth channel between a pair of conversants, who can then exchange data over it. Different phone systems have different kinds of channels to allocate to a call: a copper pair, a frequency on an FDM coax cable, a SONET timeslot, an ISDN channel, or an AMPS FM channel pair, for example. Once the call is set up, it is free from interference; except in the case of equipment failure, it offers guaranteed bandwidth and reliability, and there is no need to resend data due to collisions with other senders as there is with Ethernet.

However, before the call is set up, some sort of communication channel needs to exist to bootstrap it. This is the so-called "control" or "signaling" or "paging" channel, and typically it provides a best-effort kind of service, with no guarantees of bandwidth or reliability, because the channel is shared among many uncoordinated users. (USB demonstrates that this is not the only possibility.)

Even a low-bandwidth paging channel can distribute the hashes of new root blob hashes pretty easily; you only need to transmit 256 bits, and if you only need to be secure against current attacks, you only need to transmit about 80 bits. There are several ways to implement such a channel: burning Bitcoin, shortwave radio, classified ads in the New York Times, gossip among locally connected nodes, IRC channels, comment threads on long-ignored news articles, timing channels in DNS TTL countdowns from shared caching DNS servers, shining lights on tall buildings at night, and so on.

If you are somehow in a position to broadcast such a hash, how do you choose which one to broadcast? Maybe you'd broadcast the hash of the index that was most up-to-date and had the largest number of publishers, perhaps creating your own by piecing together other indices you had access to. Or maybe you'd create your own by removing all the publishers you suspect of anti-Islamic views or

publishing misinformation about the covid pandemic. Maybe you'd include thousands of "publishers" selling penis pills, or maybe you'd copy someone else's index and remove the penis-pill sellers. It all depends on your desires.

But what about the people who retrieve that hash? What do *they* want to do? What kinds of paging channels will be responsive to their needs instead of the needs of whoever has the most money or the brightest arc lights?

Topics

Commit log transfer

Kragen Javier Sitaker, 02020-05-16 (1 minute)

In a Kafka-like system running on a kernel where memory is transferred rather than shared, the "commit log" for a channel could physically consist of the uncopied message buffers the producers had transferred to the broker. With copy-on-write functionality, these message buffers could be directly exposed to subscribers without ever copying them, although at the risk of exposing subscribers to information about the message bundle boundaries they are not supposed to depend on (this risk is already present in Kafka). With FlatBuffers and similar techniques, publish-and-subscribe within a single CPU could proceed at tens of gigabytes per second — billions of messages, hundreds of times faster than ØMQ or Kafka, which are about equally fast.

Although, in such a high-bandwidth system, how do you limit retention?

Topics

- Performance (p. 794) (24 notes)
- Systems architecture (p. 809) (12 notes)
- Protocols (p. 813) (10 notes)
- Publish/subscribe feeds (p. 889) (3 notes)
- Kafka (p. 951) (2 notes)
- Flatbuffers

One pass sort

Kragen Javier Sitaker, 02020-05-16 (15 minutes)

External sorting on modern general-purpose computers is almost invariably two-pass. What if we could make it one-pass? We kind of can, if we cheat. Especially with an SSD.

You might have up to 50 terabytes or so of disk attached to a modern computer; typical performance characteristics of each disk might be 10 milliseconds random access time and 200 megabytes per second of transfer bandwidth, and you might have a dozen or so such disks on your machine. Tapes have been relegated to niche applications. So you might reasonably want to sort up to about 20 terabytes. But you surely won't have less than 4 gibibytes of RAM, probably more like 64 gibibytes.

Generally the disk bandwidth is an unavoidable bottleneck, but the random access time of the disk (seek plus rotational latency) can dominate it if you're not careful. To keep the disk bandwidth lost to random access time below 10%, you need to transfer a sequential stream of 9 or more bandwidth-delay products every time you access the disk. With the figures given above, the bandwidth-delay product of each disk is about 20 megabytes, so you need to read a chunk of 180 megabytes after each random seek. If you read in chunks of only 40 megabytes, you'll be at only ⅔ of what the disk bandwidth could hypothetically handle; you don't start to see big performance losses until you're well below that. But if the chunks are only 2 megabytes, you're only able to use 9% of the disk's potential bandwidth, and your sorting will take 11 times longer than it should.

The standard mergesort approach is to fill RAM with input data, sort it, and write it back out to a temporary file. Ideally, you continue to read in data to add to the in-memory sorted data, replacing the data you've already written out, which in the worst case of the data being backwards gains you nothing, gains you a factor of 2 in the average case of the data being unsorted, and converts the procedure into a single pass in the best case of the data being presorted. Let's take the worst case, though: 20 terabytes of data in 16-gibibyte chunks gives us 1165 chunks.

So now we have 1165 individually-sorted temporary files of mostly 16 gibibytes each, and we want to merge them into a single output file. So if we divide our 16 gibibytes of RAM into 1166 buffers — one for output, the others for input — we have 14 mebibytes of buffer per file on average. If we wait to refill or flush each buffer until less than a mebibyte of slack is left in it, then we can read 26 mebibytes into the buffer, growing it from 1 mebibyte to 27 mebibytes. This gives us 57% I/O bandwidth usage, which is not great but possibly acceptable. If instead we have the expected 582 temporary files, we can read 55 mebibytes on each such occasion, which is 73% I/O bandwidth usage.

If we only have 4 gibibytes, though, our capability for efficient two-pass sorting is limited to just a terabyte or two. Two terabytes gets divided into 466 temporary files, and so we can only allocate 9

megabytes of buffer to each file, only getting 16 megabytes per transfer, slightly worse than the scenario above but still the right order of magnitude. Sorting files any larger will start to require three-pass sorting.

What's going on with this 20-terabyte output file, though? We can distribute our temporary files freely across half a dozen disks, so our aggregate input bandwidth from those disks may exceed a gigabyte per second, which we can then merge and write back out. But we can't write a gigabyte per second to any of our disks! We can only write at speeds like those if we're writing across all the disks at once. We have to use RAID or some kind of clever virtual filesystem that stores a virtual file in segments on many different filesystems. And then we may have lost, because if those segments are large and sequential, we may only be able to write *or read* them at 200 megabytes per second!

Cheat 1: merge on read

So suddenly we are faced with weird existential questions like, what is a file, anyway? It's not really a physical thing, but some set of operations and behaviors that work to store our data. What kinds of operations does it need to support, and what kind of behaviors does it have? Once the sorting process's output "file" has been created and closed, it probably doesn't need to support further writes; it just needs to support reading. What kind of reading? Is it enough to be able to open the "file" and get records from it one at a time in sorted order? Or do we also need to be able to tell where we are in it, and seek to previously told positions in it? Do we need to be able to find the size of the file and seek to arbitrary byte offsets?

If it is adequate to read sequentially, telling where we are, and seek to previously-told offsets, then we can skip the whole stage of merging the temporary files into an output file. Instead we can merely decree that this collection of hundreds or thousands of "temporary files" now constitutes the output file, which is now divided into these parallel "chunks". When you go to open the results for reading, we open *all of the chunks*, and when you read records, we do the merge right then, on demand, in RAM.

This has some advantages! After a single pass over the input, you can start processing the output, doing whatever it is you want to do with it. And you can save bookmarks in that output and seek to them again. But it has some disadvantages, too. A bookmark is the current read position in all the chunks at once! So representing it might take 8 kilobytes.

As an alternative to seeking to a *byte offset*, you could seek to a *key*, which would require adding extra crap to disambiguate any possible duplicate keys. Note that you can't detect duplicate keys during sorted dataset creation, only during reading, so you may need the key to include a chunk identifier that tells which of the thousands of chunks your desired record is in, along with a consistent ordering across the chunks.

Seeking to a key in this way would require doing a binary search in each chunk; if your records average 128 bytes, each expectedly-32-GiB chunk contains 128 mebirecords, so you need to

examine 27 keys in each of (expected) 582 chunks, about 16000 operations; of these, only the first ten in each chunk would involve random seeks, but we're still talking about potentially tens of seconds of waiting on spinning rust.

However, you can add a Lucene-like skip file to the dataset, containing 512 KiB of keys sampled from each chunk and their associated byte offsets in the chunks; if the keys are 16 bytes, you can fit 32768 keys per chunk into the skip file, so the skip file gets you within 1 MiB of the right place in the chunk. The whole skip file is only 32 MiB. This cuts the number of seeks needed by an order of magnitude, to only one per chunk.

Given a consistent ordering across chunks as mentioned above, it's possible to get all the way back to raw unidimensional byte offsets. Say your positions in the various chunks are {3532, 832, 483, ...}. So your byte offset in the entire dataset is the sum 3532 + 832 + 483 + ... But seeking to such a byte offset is nontrivial: you need to guess the right byte offset in each chunk, find the nearest record start, read the key, take the median key, find the nearest corresponding keys in all the other chunks (jumping some of them backwards and others forwards), and then iterate forward or backward as necessary — or possibly binary-search for the correct key, adding another factor of 4 or 5 to the seek-to-a-key procedures in the previous paragraphs.

This approach is pretty similar to LSM-trees.

A problem bigger than the seeking problem: opening the file requires 16 gibibytes of RAM for input buffer space! That doesn't leave a lot for your application.

Cheat 2: partition on read

So I was thinking there might be a better idea, but this turns out to not work very well.

Let's consider the case of sorting a 2-terabyte file in 16 gibibytes of RAM. First, we take a random sample of 32768 records from the input file to find out what the distribution of its keys is, and we pick 1023 key values that partition the inferred distribution more or less evenly, into 1024 partitions. We preallocate a temporary file for each of these partitions, a little bit bigger than we expect to need, say 3 gigabytes, we open them all at once, and we initialize a RAM write buffer for each temporary file.

Now, we start reading in the input file; as we read each input record, we determine which partition it goes into, and we append it to that partition's buffer. Whenever we run out of memory, we flush to disk whichever output buffer is fullest, which has an expected size of 32 megabytes.

When we are done with this partitioning process, we have 1024 "temporary files" of 2 gigabytes each. Each of them is unsorted internally, but has a known size, and each of them covers a disjoint part of the keyspace, and, importantly, is contiguous on disk — we aren't relying on the filesystem to magically defragment a bunch of badly fragmented writes.

So now, to open this "output file" and start reading it sequentially by key, our user program opens up the first partition file, reads 2

gigabytes into RAM, sorts them, closes the file, and begins iterating. When it gets to the end of the first partition, it opens the second partition and repeats the process.

This permits seeking to an arbitrary byte offset in the combined output file: you subtract the sizes of partitions until subtracting the next one would go negative, and that tells you which partition file you need to open. But it still takes a few seconds per seek.

So, when it works, this is an improvement over the previous technique: you only need 2 gibibytes of input buffer memory to "read" the output "file" instead of 16 gibibytes, and you can seek and tell with regular byte offsets. But seeking is still ridiculously expensive.

This approach also still requires some kind of RAID under the covers to stripe each file across disks.

Cheat 3: square-root hybrid

What if we combine both of these approaches? When "sorting" the data, partition it into 64 equal-weight keyspace partitions, as in cheat 2. When RAM is full, take the in-memory partition with the largest amount of data in it — in 16 gibibytes, the average in-memory partition will be 256 mebibytes, while the most bloated one should usually be around 512 mebibytes — and sort it, as in cheat 1, before writing it out to a "temporary file", let's call it a "chunk". If the total dataset is 2 tebibytes, similar to the cheat-2 example, then in the end there will be around 4096 such chunks, 64 per partition.

Now, to start reading the data "sequentially", you open the 64 files from the first partition and start merging them. Doing this efficiently requires enough buffer space for 64 files — say, 1.28 gigabytes on average to do 40-megabyte reads, but 2.56 gigabytes at startup.

But wait! Have we won anything? If we didn't partition the 2 tebibytes, we'd be writing out (in the expected case) 32-gibibyte sorted chunks rather than ½-gibibyte chunks. There would still only be 64 of them if we only had 2 tebibyte of data. So this partitioning doesn't buy the reader anything!

A bookmark to seek to might be represented as a partition number plus 64 32-bit file offsets. Or, as said previously, a key plus a chunk number.

So I think this doesn't really help. In fact, it hurts a little.

SSDs

Modern SSDs, as I understand it, can deliver 2.5 gigabytes per second of 4-kilobyte reads, but are limited to sequential writes due to the necessity of block erase. This suggests that a different organization of data in storage could work better — you can read 4-kilobyte blocks in whatever order you want, you just want to make sure that each such block is relatively coherent. Between blocks, they could form a linked list, no problem. But of course you still have the problem that it's going to be pretty difficult to form a block with all the record with keys in a given range of the keyspace before you've seen all the input — the last record in the input might be in that range.

On the machine I have here, it's more like 32 kilobytes per

transaction and only 120 megabytes per second.

So what happens if, instead of 32 megabytes per input stream, we only need 32 kilobytes? Generating output doesn't get any easier — and the trick in Cheat 2 of generating lots of output files in parallel gets a lot harder — but reading from 1000 files to merge them, as in Cheat 1, stops being a problem. Suddenly you only need 16 megabytes of RAM for your input buffers, 32 to start, rather than multiple gigabytes.

Topics

Optimized finger joints

Kragen Javier Sitaker, 02020-05-16 (4 minutes)

Laser-cut finger joints are a popular way of joining MDF into boxes. The fingers on each side of the joint are cut slightly wider than the spaces on the other side, because otherwise there would be slop around them twice the width of the kerf, which is on the order of 100 µm, and the joints would not join without glue.

This means that you cannot cut both sides of the shape from the same piece of fiberboard with a single zig-zag-zug sort of cut; you'd have that slop. So you apparently need to cut the two sides of the joint separately, doubling the cut time and the cost.

But do you? Suppose each finger and each space along the joint is 250 µm narrower than the preceding finger or space on that side, and you use a single zig-zag-zug. Then the pieces won't join together at the position they were cut out — but if you shift them by a single finger–space cycle, they will join firmly, with 25 µm of interference on each side. So if you shift the two pieces by a short distance relative to each other in the to-be-cut layout, and shift them back to assemble them, they will fit together snugly.

So, for example, you might have a 150-mm-long finger joint made of ten 15-mm-wide fingers, and you can overlap 90 mm of it in this way: 30 mm at the bottom (two fingers, one on each side) is just the left piece, 90 mm (six fingers) in the middle is overlapped, and 30 mm at the top is just the right piece. The finger widths vary from 31.25 mm at the bottom to 28.75 mm at the top. This results in 180 mm of cutting, plus 11x, where x is the thickness of the material — 33 mm if it's 3-mm MDF.

This offset or stagger might make efficient nesting more difficult, and thus increase material costs, or it might not. But, with MDF, cutting costs greatly exceed material costs.

An alternative to edge finger joints is mortise-and-tenon joints. These can be tight, like finger joints, but they can also be loose, with an extra 100 µm or so of slop deliberately left to ensure that pieces can slot together easily. A series of such mortise-and-tenon joints substitutes for a dado or groove joint, which cannot be themselves made by sheet-cutting technologies like laser-cutting. Using a sequence of such loose mortise-and-tenon joints rather than a finger joint both eases assembly and also reduces the chance that an out-of-tolerance cut will convert a tight fit into an impossible fit. A few SIGGRAPH papers have shown ways of designing arbitrary assemblies so that all the pieces slide into place with such joints, locking previous pieces into place; a final piece with an interference fit is adequate to hold the whole assembly together.

A different way to get finger joints to have interference fits, without staggering them, is to angle the edges of the fingers rather than using right angles, so the finger tips are narrower than the finger bases, and the spaces between the fingers are narrowest at their base. A difference of 120 µm over a 3 mm finger length works out to about a 92.3° angle rather than the 90° usually used. I haven't tried this, but

I suspect that the resulting joints will not be as strong as regular finger joints, because only about one fourth as much area is in contact, but they should be easier to assemble and fairly robust to process variation.

Topics

Solar furnace CPC

Kragen Javier Sitaker, 02020-05-16 (12 minutes)

I was thinking about how to reach high temperatures inexpensively and safely during this quarantine. Not, like, really high temperatures, but hotter than the oven.

Carbon foam made by carbonizing bread is probably the easiest and most accessible insulating refractory material for this kind of thing; it doesn't tolerate oxidizing conditions (it slowly burns above 700°), but in reducing conditions it gradually converts to graphite, which sublimes at 3642°.

Stefan–Boltzmann temperatures

"One sun", the solar constant, is standardly approximated as 1000 W/m², which is the Stefan–Boltzmann emissivity of a black body at 91.3°. So a perfectly insulated object in full sunlight will eventually heat up to 91.3°. Because at that temperature all the thermal radiation it emits is in the infrared, you can get it to heat up to higher temperatures by painting it with paint that is highly reflective in the infrared, or by putting infrared-reflecting glass in front of it, but for simplicity I'm going to be considering the blackbody case for now.

The 1368 W/m² on orbit corresponds to 121°. "Two suns", 2000 W/m², only corresponds to 160°, which is enough to cook, barely; you can reach this level of illuminance with a single flat mirror. Reaching 260° like this gas oven requires 4600 W/m², 4.6 suns, which is enough for soldering electronics. 600°, enough to fire some red clays and almost cast aluminum, emits 33 kW/m², 33 suns. 1000° is 150 suns, 1100° is 202 suns, 1600° (to melt quartz or pure iron) is 698 suns, and 2072° (to melt sapphire) is 1715 suns. Subliming graphite (3642°) would probably be impractical at 13300 suns. Quartic growth is a bitch. 5500° (63000 suns) is the absolute limit.

Cavity absorbers

A small hole leading into a large cavity, sometimes called a cavity absorber, behaves as a very good approximation of a blackbody, one you can't paint. At low temperatures, convection of air is a significant way to lose heat, but at the higher temperatures I'm interested in, almost all the heat loss is through radiation.

Probably the smallest hole it's practical to make in carbon foam and concentrate sunlight through is about 10 microns in diameter. Reaching 256 suns (1184°) then requires concentrating the sunlight from a 256-times greater area on this hole: a circle of 0.16 mm in diameter, for example, gathering about 20 microwatts.

The material inside the cavity mostly "sees" other material inside the cavity; nothing short of a cat's eye will send a significant fraction of the light coming in the hole the hole directly back out the hole. Almost all light that gets in needs to bounce around many times, losing energy each time, before it can get back out. So even the hole in the top of an opened empty beer can looks black, even though the

beer can is 95%-reflective aluminum on the inside.

Unless the cavity is meter-scale or larger, parts of the cavity that aren't the hole need to be well insulated to prevent the loss of more heat through conduction through the walls than from radiation through the hole.

Optics of concentration

So if you can concentrate 256 suns on a 10-micron hole into a sufficiently-well-insulated cavity, you should in theory be able to heat it up to 1184° with those 20 microwatts. This suggests that solar furnaces can perhaps be made fairly small, though see below about insulation thickness scaling.

It isn't sufficient to focus the sunlight from an 0.16-mm-diameter lens of any focal length whatsoever, though. If the focal length of the lens is too long, then the focused image of the sun will be too large and therefore diffuse. From the point of view of an ant passing through the projected image, the whole lens is as bright as the sun, but the lens is only a few times bigger than the sun from her point of view, so the power density is not that high. The f-stop of the lens needs to be wide enough to get to 256 suns — specifically the lens needs to look 16 times as wide as the sun, which is 0.53° (about 32'), so the lens needs to subtend 8.53°, which means any lens with 256 suns needs to have an aperture of f/6.72. So if its focal length is 10 mm, the lens needs to be at least 1.49 mm in diameter, at which point it (like any other lens with a 10-mm focal length) will project an image of the sun some 93 microns in diameter. You can only get 256 suns with an 0.16 mm diameter lens if its focal length is about 1.1 mm.

If you use a lens that's bigger and further away — for example, the 10-mm-focal-length, 1.5-mm lens suggested above — then most of the energy gathered by the lens will not enter the cavity. A 93-micron-diameter sun image with a 10-micron hole in the middle of it will gather about 100× as much energy as is actually put into the cavity. You might think that, in exchange, you don't have to constantly track the sun. No such luck! The Earth turns 360° per 24 hours, which is 0.25° per clock minute, so your sun image gets displaced by a sun diameter every 2.1 minutes, whether that's 10 microns or 90 microns. (It's slightly less when the sun is further from the equator, but what's important here is that it's 2 minutes, not 20 minutes or 2 hours.)

For lower concentrations, you can use a one-dimensional concentrator like a solar trough (or a glass rod), running parallel to the sun's path in the sky, but reaching hundreds of suns that way is not practical, though in theory it's possible.

Non-imaging optics such as a compound parabolic concentrator are said to improve the situation dramatically, permitting much wider input angles. You can use two developable compound parabolic concentrators made of aluminum foil (reflectivity 95%) on cardboard, at right angles to each other, to funnel light into the hole over a wider range of sun angles; the disadvantage over using a CPC that is a solid of revolution is that most of the light will be reflected from the aluminum twice instead of once before going in the hole, thus reducing efficiency.

The overall principle limiting the performance of NIO is conservation of étendue: the intensity of illuminance times the angle it's coming from. The thermodynamic limit is that you can't use the sunlight to heat things hotter than the sun's surface (5500°); you would reach that limit by arranging optics so that the poor ant sees solar surface in every direction, 4π steradians of nuclear flaming death, 63000 suns†. Conservation of étendue says that the reflection the ant sees is only as bright as the sun, and you can only do that if all those optics would direct any light the ant emits into some part of the sun's disc, which means that such optics necessarily have a very narrow angle of acceptance: 2.1 minutes later, the ant's remains will see only cool blue sky.

So it seems like you ought to be able to shape the optics such that you get 256 suns for 1/256 of the day before you have to reorient them; any light emitted from the hole would be redirected onto the sun's daytime path. Unfortunately, 1/256 of the day is only 5.625 minutes. So this doesn't help as much as you'd hope for these ceramic-firing applications; you need to use feedback control.

5 suns, 271°, enough for soldering or baking, can be achieved by optically coupling the hole to 4.8 hours of the sun's path. A one-dimensional trough CPC focused on a slit might be adequate; four flat mirrors spaced at angles around a hole might also work.

I'm not sure if I'm thinking this through correctly. Sunlight on the ground gives varying amounts of illuminance depending on the sun's angle; it's only a whole sun at noon (and only twice a year at that, and only if you're in the tropics). Sunlight reflected in a mirror surely does look just as bright as the regular sun when you're looking at the mirror (from an angle where you can see the sun in the mirror, anyway), but the mirror can be angled to spread it across a lot of ground.

† this 63000 ought to be 4π steradians divided by however many steradians the sun subtends, but I haven't calculated that.

Insulation thickness scaling

Above I said that you can get your cavity to 1000° with 20 microwatts of sunlight focused through a 10-micron hole if the cavity is well enough insulated and you have good feedback control orienting the reflector. But it turns out to be impractical to insulate the cavity well enough.

If your insulation material conducts heat at 0.3 W/m/K (typical for refractory bricks), your cavity's surface area is 6 cm², and you have a 1000 K temperature difference, then at 1 mm of insulation thickness you would lose 180 watts. Not microwatts or milliwatts, but entire watts. So you're seven orders of magnitude away from being able to reach 1000° with a millimeter of insulation. Exotic vacuum panels might be able to gain you some of those orders of magnitude back, but charred bread won't.

You can get maybe two or three of them back by making the cavity smaller and the insulation thicker, but I think that at some point it's sort of a lost cause because once the heat diffuses a few millimeters through the insulation it's diffusing through a much larger surface area again. So microwatt-scale kilns would need

building-sized insulation.

A more practical approach is to scale up to, say, 100 watts, which is about 320 mm × 320 mm of sunlight. If you concentrate that down to 20 mm × 20 mm (or a 23-mm-diameter hole), you have your 256 suns. The hole can be at the end of a bit of a bottleneck leading into a chamber of, say, 50 mm diameter, which is 65 mℓ and has a surface area of 7900 mm², 0.0079 m². This would require 24 mm of insulation thickness to lose the 100 W through conduction, so 100 mm or so should be adequate to get it most of the way up to that temperature. This ends up being 250 mm in total diameter, which is probably about as big as I can bake a loaf of bread.

So a solar furnace of subcentimeter total size probably isn't practical without vacuum multilayer insulation, but submeter is totally feasible.

Insulation stops being a difficult problem with large cavities. Consider scaling up by a factor of 200: a 10-meter-diameter cavity. Let's scale the hole up only by a factor of 50: it's a 1.15-meter-diameter circle, swallowing 256 kilowatts fed to it by hundreds of square meters of mirrors. It holds 524'000 liters, and its surface area is 314.15927 m². To keep its conduction losses down to 256 kilowatts, it only needs 400 mm of insulation! Now the chamber dwarfs the insulation; if you can dig it into the ground, you don't need any further insulation, although you might need to line it with a sturdy refractory in case it turns the ground into lava.

Topics

Pandemic collapse

Kragen Javier Sitaker, 02020-05-17 (updated 02020-12-16)
(22 minutes)

I was looking at a thread on the orange website and I was surprised by people's shortsightedness. They're talking about how the stock market remains high despite the pandemic's damage to the economy, but their spectrum of possible outcomes seems to stop at the Great Depression.

I see a lot of people in the US trying to understand the current disaster through the lens of the last big disasters that happened in the US: the US involvement in the Vietnam War, the US involvement in WWII, the Great Depression, the 1918 flu. Of course I don't know what is going to happen — and if I did, I wouldn't want to attract the resulting attention — but I think you should take a wider perspective. This might be an OCP, more like when Sherman marched to the sea or Cortés rode into Tenochtitlán. It might be more like Cambodia's experience of the Vietnam War than the US's. Things may change more than you expect. The US may not survive.

As Annalee Newitz writes in the New York Times of the Bronze Age Collapse:

> When their cities were swallowed by fire, the Bronze Age ruling classes lost everything, including the subjects they once controlled. Greece's population dropped by roughly 50 percent during this time, probably because of a combination of war, drought and migration, according to Sarah Murray, a classics professor at the University of Toronto and author of "The Collapse of the Mycenaean Economy." Mr. Cline believes that plagues may have driven people into the hinterlands, too.

It's hard to estimate the probability of such a country-destroying disaster, so it's tempting to just dismiss the possibility out of hand as outlandish — nothing like that could possibly happen, since the US has remained stable and indeed grown in power in our lifetimes, in our parents' lifetimes, in our grandparents' lifetimes, even in our great-grandparents' lifetimes. It's tempting to assume that an edifice that has thus stood the test of time will endure forever. Moreover, since the United Nations won World War II, the world has experienced a historically unusual period of relative peace, the Pax Americana, sometimes called the American Century.

Country collapse base rate estimation: on the order of 1% per year

But periods of peace and countries do not endure forever; they are wracked by invasions, revolutions, military coups, and simple collapse. What's the base rate of such events?

Let's consider specifically the kind of events that upend the established order in a country and consign the rich and powerful to poverty and death. Again, it's hard to measure precisely, but we can get within an order of magnitude. In recent centuries, most countries experience such a major upheaval about once every century or two. Once a decade is clearly too often, and once a millennium is too rare.

A cross-section across countries in the last few years

Let's consider recent events worldwide.

At the beginning of 2020, most countries still had the same way of life they had in 2010 and indeed in 2000, and investments made in 2000 were still secure in 2020. We can enumerate the exceptions: Egypt, Tunis, Iraq, Congo, Liberia, Syria, Afghanistan, Sudan (if you live in Darfur), the Rohingya regions of Myanmar and India, parts of the Niger delta, Yemen, the Crimea, Venezuela, Libya, and arguably Hungary, Mexico, Bolivia, Ukraine, Mali, Honduras, Kyrgyzstan, Ecuador, and Turkey. That's somewhere around 10 to 30 countries, although at the border, it's pretty fuzzy.

That's out of about 200 countries (again, pretty fuzzy), so we're looking at a rate of around 0.5% to 1% per year.

But is that really fair? Afghanistan has been a mess for generations, and the Crimea for centuries. Perhaps some polities, like Switzerland and the Roman Republic and Empire, are very stable, while others, like Afghanistan, are very unstable. (Note, though, that Rome suffered its share of disastrous revolutions and sackings before it finally fell.)

That's as may be, but for now we're just trying to establish the base rate. Later we can work out how large and frequent to expect local deviations to be, and in which direction.

A longitudinal survey of the recent histories of some random countries

Let's take a longitudinal survey looking at the histories of particular countries. If you look back in the history of any given country, you mostly only have to look back a few decades to the last such event, maybe a century or two. Let's look at every 17th country from Wikipedia's list of countries by water use.

- India? Partition, in 1947.
- Italy?† Following the Holocaust (killing some 15% of its Jewish population), in 1943 they were defeated in World War II, their prime minister hung upside down from a gas station, their king forced to abdicate, and much of their territory given to Yugoslavia and the United Nations; over the next 40 years they had successive economic crises and terrorist massacres.
- Syria? Currently on fire.
- Kyrgyzstan? The Soviet Union collapsed there in 1991, ushering in decades of poverty (22% of the population is still below the poverty line), the capital was looted during a popular uprising in 2005, the mafia keeps assassinating parliamentarians, the president fled in 2010, and the interim president requested an invasion from Russia to put down an incipient civil war.
- Cambodia? The Khmer Rouge killed all the intellectuals and a quarter of the population in 1975–8.
- The United Arab Emirates? The current state there was established when the British blew up Ras al-Khaimah in 1819, conquering the country.
- Oman? In the 18th century it was the preeminent power in the Indian ocean. After centuries of decline, Britain bombed the shit out

of it from 1957 to 1959 in order to bring the Imamate of Oman (incidentally, one of those thousand-plus-year-long states like Rome) under the power of its ally the Sultanate of Oman.
• Suriname?‡ It had a civil war in 1986–9 in the wake of the bloody 1980 coup (whose winner is the current president, though recently sentenced to 20 years in prison), and although the genocide of the Americans during the colonial era was far less complete there than elsewhere in America, only 4% of Suriname's people today speak an American language and less than 2% practice an American religion.
• Qatar? Doha and al-Wakra were sacked and looted by forces from Bahrain and Abu Dhabi in 1867, following which point Qatari sovereignty was established.
• Papua New Guinea? They were a major battleground of World War II in 1942–5.

So in sorted order the last time there was a country-destroying catastrophe in these countries were 1819, 1867, 1943, 1945, 1947, 1959, 1978, 1989, 2010, and 2020, with a mean date of 1947. We can see some clustering there: two of the countries were destroyed in World War II, and India's destruction (and subsequent glorious rebirth) as part of the collapse of the British Empire was surely related to World War II as well. Whatever the distribution and clustering of these catastrophes within any given country, we should expect that the distribution of intervals since the most recent catastrophe is the same as the distribution of intervals until the next one.

In particular, this controls for the problem of instability clumpiness: like unstable servers, unstable countries tend to remain unstable for decades, with one crisis or collapse rapidly following another, so if we just count the collapse events in the world over some period of time, we will get an unrealistically high number. Today, for example, Oman has been stable since 1959, but its previous century was riven with intrigues, secessions, civil wars, truces, invasions, and gradual subjugation by colonialist British boots.

This suggests an average time-since-violent-collapse (and thus also time-until-violent-collapse) of some 73 years, with a fairly smooth distribution containing a significant number of countries going out to 200 years of stability or so.

What about the USA? Does it not have 240 years of stability? Only from the point of view of the Northerners; the catastrophic depredations of the Civil War in 1861–5 reduced the Southern states, which previously included the richest part of the country, to a poverty from which they have not recovered 155 years later, although of course the poorest people in the South were thus immeasurably enriched. (Please note, I am not arguing that the Civil War should not have happened; I am merely saying that if you were a wealthy investor in the Confederacy, you would likely be ruined by the war.) It was no picnic for the North, who suffered some 800'000 casualties, some 10% of its fighting-age men.

Even so, 155 years of stability puts the US in the tail of our empirical distribution, bested only by the UAE in my sample above. Still, it should give us some pause that the US spent four of its 240 years at war with itself. Stability is not to be taken for granted.

† Oops, now I realize Italy is one country early. I guess I'll go with

it.

‡ Oops, now I realize Suriname is one country early.

What direction should we correct these order-of-magnitude estimates?

These figures give especial weight to the 20th and early 21st centuries. To some extent, this is justifiable: there are secular trends that change what is possible and what is probable, so events in the 16th century perhaps have less bearing on what could happen in the 21st century than events in the 20th do. On the other hand, we should be alert to the possibility that a short recent period, like 2000 to present, 1980 to present, or 1940 to present, is really representative of what is to come — it might happen to be anomalously stable or anomalously unstable, in a way that might not continue to hold true in the next decades.

Do we have strong reasons for believing this to be the case?

I don't think we do. On one hand the Pax Americana reduced the number of large wars, but the Cold War also destabilized countries — this was a factor in the Suriname coup mentioned above and the destruction of the Imamate of Oman, for example, and Cambodia was of course only able to get away with its abuses because so many saw them as needed measures that only affected those with privilege, anyway. Kyrgyzstan's collapse was a result of the Cold War's end, although it's done worse than many former Soviet republics — would it have been stabler and safer without the Pax Americana and consequent Cold War? Perhaps. Or perhaps it would have been more unstable and more dangerous.

A different objection, which I hadn't thought of until berndj raised it, is that the weightings above are biased toward small countries. Any particular country on the sampled list had an 0.5% chance of being Cambodia and an 0.5% chance of being the UAE, but their populations are only 15 million and 10 million, respectively, so a randomly chosen living person only has a chance of 0.2% or 0.1% chance, respectively, of happening to live there. But failure rate probably is not independent of country size! It could easily be that large countries tend to collapse much more often than small countries, or much less often. If you carried out my survey on a planet consisting of 199 tiny countries that each collapse every year on average, and one giant country containing 99% of the world's population that only collapses every ten thousand years, you'd incorrectly conclude that people's mean time to living through a collapse was a year. So we should take another look at large countries, even though we can't take a large sample of them.

The fact that India, the largest country on the list, with 18% of the world population, happens to have most recently collapsed (with genocidal massacres with and masses of refugees) precisely at the mean date of the survey, 1947, might be a coincidence.

The other largest countries are China, with 1.4 billion humans (also 18% of the world population); USA, with 331 million (4.2%); Indonesia, with 270 million (3.4%); Pakistan, with 221 million (2.8%); Brazil, with 212 million (2.7%); and Nigeria, which at 206 million (2.6%) pushes us over the 50% mark. I think these countries' most

recent country-destroying disasters, where times were harder than the US's Great Depression, the ruling classes lost everything, the rich and powerful were reduced to poverty, there was widespread violence, and mass emigration ensued, were, respectively: 1949, or 1958, if you count the Great Leap Forward; 1865; 1975–1999, if you count the invasion of East Timor, or 1965–6 otherwise; 1947 again, same disaster as India; arguably the military dictatorship and guerilla warfare following the coup against Goulart in 1964, though the resulting countrywide impoverishment was slow and prolonged, but otherwise the tumultuous 1889–1930 First Brazilian Republic, or the 1864–1870 War of the Triple Alliance, or the 18th-century invasion by Portugal; and one of the 1993–8 Abacha dictatorship, the 1976 coup, or the 1967–70 civil war.

So, based on tentative dates of 1865, 1930, 1947, 1947, 1949, 1965–6, and 1976 for the most recent country-destroying collapses in the seven largest countries that house the majority of Earth's human population, there doesn't seems to be strong evidence that large countries are either especially stable or especially unstable. Of course, after a collapse a country might break into smaller pieces, as the USSR did, so perhaps we care more about countries that used to be part of big countries — but we've already covered those by sampling small countries. The USA's recent stability just looks like an outlier.

How about USAmerican exceptionalism? Do we have strong reasons for thinking that the US is far more stable than other countries? I don't know that we have strong evidence either way. The US is still the world hegemon, and hegemons tend not to be invaded by the countries they dominate. But their economy frequently depends on their hegemonic status, which is fairly fragile, and its loss can precipitate major upheaval — even when internal power struggles don't.

So I don't think there's a strong justification for thinking that the US's risk of collapse in average years is significantly different from the 1% or so from the above.

But this is not an average year; we have covid — thus a 20% chance of collapse

This is an unusual time. An economist at the Federal Reserve has projected 34% unemployment in the second quarter of 2020, which is higher than the peak of the Great Depression — and that's six months into the covid pandemic, not three years in. Unemployment insurance claims are orders of magnitude above past records. Last month New York City started digging mass graves for the overload of coffins as its covid infection rate peaked. Also, the US elected a reality TV host as President, and he fired its pandemic preparedness team before the pandemic. Last month, the Yugoslavian he appointed chairwoman of the FDIC he appointed published a video begging the public to "please, keep your money in a…bank." The Michigan legislature just shut down this week to avoid getting shot by protestors armed with military rifles, encouraged by the President.

Some wag quipped that it's like having the 1918 flu, the 1929 stock market crash, and Warren G. Harding's presidential incompetence all at once.

This is not normal.

It's hard to predict what will happen. Right now, the chance of any kind of rare event is significantly increased because of the covid pandemic, even — perhaps especially — in the US. Moreover, events involving chaos and discord are especially favored.

So the chance of a US collapse is higher this year than its average 1%. Let's say it's 20%.

I don't venture to guess what a US collapse looks like. Typically things like famines and plagues don't directly topple governments or end cultures; they undermine their economic strength and political legitimacy, making it easier for other forces to assert themselves.

Forces? What other forces might assert themselves?

But what groups might be players? Recent new mass movements within the US include the Tea Party and Occupy, but the military (3.2 million employees of the US DoD) is in a better position to take over if the civil state fails. (The US police force is deliberately fragmented to reduce the chance of this; so is the military, but much less so.) Both the Tea Party and Occupy support positions with broad-based popular support.

The Mormons number some 6.6 million in the US, far more than the military, and have always planned to take over government in the US if given the opportunity, in order to build a utopian society they call "Zion". Economically and organizationally, they are well-prepared for hard times, and if there is a famine, the Mormons may be especially well prepared, because each family is required to store a three months' supply of food, water, and other essentials, a practice known as Family Home Storage. Mormon communities in Mexico have resisted the incursions of drug gangs with some success. I think it's unlikely that the Mormons' dormant plans to assume temporal power will be put into motion unless society is in frank collapse, because I don't think they have either the firepower or the moral force to effectively maintain control.

Drug gangs in the US already have functional apparatus for projection of force and have geographically widespread networks and functional countermeasures against the police, and over the border in Mexico have achieved substantial, though incomplete, independence from the Mexican state. However, drug gangs generally lack broad-based support in the population in the US, unlike in Mexico, and suffer from serious prejudice, much of which is racist in nature and thus not easily overcome by a change of circumstances.

Many large companies in the US have substantial material resources, well-exercised command hierarchies, committed workforces, and in many cases continuity-of-business plans for disasters. A few even have existing security forces. It's plausible to think that Walmart (2.2 million employees), Amazon (647k employees), CVS (295k employees), AT&T (254k employees), Ford (199k employees), or Alphabet (99k employees) might be able to take on the burden of protecting their assets and employees without a functioning government. Walmart might have a hard time due to its

low profit margins (US$3k/employee), but Amazon (US$15k), AT&T (US$76k), or Ford (US$18k) might be able to take the hit without collapsing; Alphabet (US$311k profit per employee) easily could, and indeed Alphabet has often come under fire for providing public services like food, laundry, and transportation to its employees. (CVS is currently losing money.)

Alphabet is in a unique position to defend itself from security threats, since no potential foe can operate without its services at present, leaving them exposed to intelligence gathering.

There are another several dozen companies in the US with over 100k employees: Accenture (515k), Kroger (453k), Home Depot (413k), Berkshire (though that's a conglomerate) (389k), IBM (381k), UPS (365k), FedEx (359k), the USPS (500+k), and so on. Large defense contractors include GE (283k), Boeing (153k), Honeywell (115k), Lockheed (102k), General Dynamics (101k), and Northrop (83k). Other telecoms include Comcast (184k), Verizon (145k), and Charter (99k); I mention these because availability of telecommunication is crucial to viability of any geographically distributed organization.

It's easy to imagine a consortium of these big companies entering into a security cooperation arrangement with one another in order to be able to continue operating, and big defense contractors can probably count on support from any such consortium.

Topics

Font rendering with all-pass filters

Kragen Javier Sitaker, 02020-05-18 (7 minutes)

You can use all-pass FIR filters to efficiently do subpixel letterform positioning of pixel fonts as well as obviate hinting. Pre-emphasis filtering can mitigate the readability loss from nonzero-size pixels and eye defocus. This can improve text rasterization. As far as I know, nobody is doing this, so I don't know it will work.

Fractional-delay all-pass FIR filters for spatial translation

There are a variety of fractional-delay filters commonly used in music for, e.g., Karplus–Strong delay lines. The optimal filter is a sampled sinc; with a delay of 0 or some integer number of samples, this has an impulse response of 1 in sample 0 or some other sample and 0 on all other samples, but when its delay is some noninteger number, all the samples are nonzero. Sinc itself dies off annoyingly slowly, but you can window the sinc to get a faster die-off (Lánczos resampling being one implementation of this), and uniform basis splines are another less explicit way to get an approximately windowed sinc with a limited basis. As de Boor's "B(asic)-Spline Basics" explains, these splines form a partition of unity, unlike the Lánczos kernel.

The same approach can be used to translate a sampled pixel image by some fractional number of pixels. If the source and target have the same resolution, this is just a convolution, with a kernel depending on the fractional part of the shift; if the original image is bilevel (black and white, so every pixel is either 1 or 0) doing this convolution in the spatial domain amounts to selectively adding up some of the weights in the convolution kernel to generate each output pixel, those that happen to land on white pixels. This therefore requires no multiplications.

If the source image has resolution higher than the target by some integer factor n, such as 2, 3, or 4, then I think this approach is still mostly valid, but now instead of a single convolution kernel you have n^2 of them, such as 4, 9, or 16 kernels, each a sampled sinc whose frequency is at the destination resolution. In particular, you can use an outline letterform rasterized to a high-resolution bilevel image to compute a grayscale image rasterized with perfect resampling (limited only by rounding), or very good resampling (limited by rounding and windowing). And the high-resolution bilevel image can be quite compact.

In particular, I think this gets rid of hinting. Hinting is a set of hacks which, among other things, deforms letterforms so that their stems and curves align more often with pixel centers and their borders run, as much as possible, halfway between pixel centers; this is important because, without that alignment, you lose spatial information about where they are to the sampling operation. This works very poorly with animation and with subpixel glyph positioning. But sinc filtering spreads that lost spatial information out to the surrounding pixels in the form of ringing, and as it happens,

your eyes can pick up on that. So you shouldn't need hinting.

Of course, on an LCD, you should sample at the LCD subpixels, usually R, G, and B from left to right, not to the square pixels containing them.

Efficient low-precision implementation with a multiplier

This operation of convolving a bilevel image with a convolution kernel has something of the flavor of binary long multiplication by an element of the kernel; each bit determines whether or not to add that weight at a particular spatial position in the output. And indeed you can carry it out with a multiplier under appropriate circumstances. Take the row of pixels 0011100111100001. Suppose 4 bits of grayscale in the output is enough; let's space out that number into a 64-bit word by inserting zero bits, so it becomes 0x0011100111100001. If we multiply this by a 4-bit weight such as 3, it becomes 0x0033300333300003. Suppose the next weight to the right is 4, and the next pixel to the right is 1, so we shift in that 1 on the right and get 0x0111001111000011, then multiply by 4 and get 0x0444004444000044, which we can add to the previous result to get 0x477304777300047, as well as the results from doing the same thing with the corresponding weights in the next row of the convolution kernel and the corresponding input pixels in the next (previous) row. Proceeding in this way I think we can get perhaps an 8× to 16× speedup over the straightforward convolution algorithm, at the expense of really miserable overflow behavior. The speedup is probably only 2× or 4× against a straightforward SIMD algorithm if you have SIMD instructions.

Because of the overflow behavior, you can't use 2's-complement for negative weights, which of course are everywhere in sampled sinc kernels. Two possibilities occur to me: represent the weights in sign-magnitude form, using the sign bit to determine whether to subtract or add the product from the running sum, or use an excess-N representation for the weights and the running sum, subtracting N from each pixel after each multiply-add.

Low-rank approximations

Low-rank approximations of the relevant sinc kernels may be useful in reducing the windowing error at a given computational load, and the SVD provides an easy way to find them; see notes/svd-convolution.html in Dercuano for details.

Nonzero-area pixels and pre-emphasis

Above I said that sinc resampling can produce a perfectly resampled image, but there are a couple of complications. First, conceptually the sampling comb is made of Dirac deltas, which concentrate a nonzero amount of energy into a point in space. But we live in a universe where doing that would require creating a black hole, which is both practically difficult and highly radioactive, so instead we approximate it by illuminating or darkening pixels of finite, nonzero size.

This amounts to convolving this ideal sampled signal with the

shape of a pixel, which acts as a zero-phase low-pass box filter with a sinc frequency response. The blurring of pixels by CRT beam dispersion or old-person eye defocus adds an additional low-pass characteristic, but one that's harder to measure. Since the pixel shape is smaller than the pixel spacing, its first null is well above the Nyquist frequency, so this low-pass characteristic can be corrected by "pre-emphasis": zero-phase linear time-invariant filtering of the original signal to attenuate the strongest frequencies and amplify the weaker ones, giving a perfectly flat frequency response. You may be able to fold this into the resampling filter described earlier, or you may want to do four high-pass IIR-filter passes in the four cardinal directions.

One-dimensional translation

An important special case of subpixel text spatial translation is horizontal translation. I think it's possible to use just a fractional delay filter in the X-axis in this case, dramatically reducing the computational cost.

Topics

Single output build

Kragen Javier Sitaker, 02020-05-19 (4 minutes)

Some build systems and dependency systems support build steps that produce multiple outputs. Make, on the other hand, identifies each build step with a single build artifact produced by that build step. This is a better approach.

An apparent benefit of multiple-output build steps is efficiency: perhaps the same compilation that produces an object file also produces, for example, a listing file, and producing them separately requires essentially running the compilation twice, with the same optimization settings (and all potential sources of nondeterminism removed.) The solution for this problem is to make an *output directory* be the resulting build artifact, containing both files.

The dependencies (inputs) of a build step can be determined by interposition, for example watching the system calls performed in order to find out what files are being opened. If the build step succeeds or fails at some point, then as long as it is deterministic, we can be sure that it will succeed or fail again with precisely the same results as long as none of the environment it observed while running has changed. In particular, this means that it is okay if it *would have* read some other potential input file if it had not encountered an earlier error — changes in that other potential input file will not change the error. And it is perfectly okay to read references from one input file, such as foo.c, to another, such as foo.h; as long as foo.c does not change, the resulting dependency set remains static.

Multiple outputs of a build step are, by contrast, messier. What happens if two separate possible build steps can create the same file? What happens if a build step creates a file on one occasion, but due to a change in its inputs, not on another? It's better to steer clear of such messy issues.

Although it may be most convenient to support a traditional filesystem API for producing build artifacts, it isn't necessary. Suppose we are constrained to produce one file per rule, as the standard output of a build script, but the build step runs inside an isolated filesystem bubble whose contents are discarded once it finishes. Then we can handle the above listing+object case as follows, using Make syntax but for convenience with inputs inferred as described above:

```
foo.tar:
    gcc -g -Wa,-adhlns=foo.lst -c foo.c
    tar cf - foo.lst foo.o

foo.lst:
    tar xf foo.tar foo.lst
    cat foo.lst

foo.o:
    tar xf foo.tar foo.o
    cat foo.o
```

You can do the same thing within a single process, but it generally takes more than two short lines of code to express it. And you could imagine a memory-centric version of this where the "foo.tar" output was in the format of a segment of (sharable, read-only) memory, and foo.lst and foo.o were "subsegments" of it. So this approach doesn't depend on the use of the filesystem.

Why might you want to split out a build artifact into multiple pieces this way? After all, any computation you can do on the basis of foo.o above can also be done on the basis of foo.tar. I think there are two reasons: decoupling and caching.

The linker should not be coupled to the fact that the compiler is generating a listing file. Rather, it should be insulated from that information. It should not have to fish the object code it's interested in out of a larger file containing mostly things it's not interested in. That's decoupling.

Moreover, if you make a change to the source code or the build script that doesn't change the object file, only the listing file, it would be nice to avoid rerunning the linker. If the linker doesn't even open the listing, we know it can't depend on its contents. So we can use the linker's cached output.

Topics

Electronics kit

Kragen Javier Sitaker, 02020-05-23 (updated 02020-12-20)
(14 minutes)

When I was a kid I had a Radio Shack "Science Fair 200-in-1 electronic project kit", similar to the 150-in-1 kit Fran Blanche recently talked about on her show. It was designed in 1981. I don't know if I built 10 circuits with it or 100, but probably somewhere in that range.

According to the manual, it contained:

• a bunch of wires, about 80 in all;
• 20 resistors (one 100Ω, two 330Ω, three 470Ω, four 1kΩ, one 2.2kΩ, two 4.7kΩ, and one each 10kΩ, 22kΩ, 33kΩ, 47kΩ, 100kΩ, 220kΩ, and 470kΩ);
• 3 diodes (one germanium 1N60, the others 1N4143);
• 4 transistors (two 2SC945 NPN, two 2SA733 PNP, all Si);
• 10 capacitors (one each 100pF, 0.001µF, 0.005µF, 0.01µF, 0.05µF, 0.1µF, 3.3µF, 10µF, and two of 100µF; mostly ceramic but with the four largest electrolytic);
• a 9V 500Ω relay;
• a 265pF variable capacitor for radio tuning;
• a 250µA 650Ω galvanometer;
• a "control and power switch" (a 50kΩ pot with a switch at one end);

• six standalone LEDs;
• a 3-volt incandescent lamp;
• a single-digit 7-segment LED display with cathode resistors already connected;
• a 350 µH ferrite loopstick antenna with two coils on it, one center-tapped;
• an SPDT switch;
• an 8Ω dynamic speaker;
• 2 small transformers, one suitable for driving the speaker from a signal in the neighborhood of 5V ("900CT: 8 ohm"), the other "input" ("4K CT: 2K"), both with a center-tap on the high-voltage side;
• a piezo earphone;
• a 7400 quad-NAND chip;
• a 7476 dual J-K flip-flop;
• a KC-4SA cadmium-sulfide light-dependent resistor;
• an enclosure for six AA batteries (and no plug-in power supply);
• a momentary-contact button "key";
• two screw post terminals; and, perhaps most importantly,
• the instruction manual, including instructions for 200 circuits you could build.

Not counting the wires, that's 62 components, most of which cost a cent or so nowadays, although I think at the time the kit was more like US$100. The components were mounted on brightly printed cardboard with some extension springs mounted around them; these served to grab the stranded copper wire when you fingered them

sideways. I don't know what the advantage of this method was over jumper wires in a standard breadboard, except that I guess each component terminal has a unique identifying number, so the wiring instructions in the manual could say things like "1-81-84, 2-41-49-55-176, 26-44-46,...", and you could be reasonably sure you'd hooked it up correctly.

The designs of the circuits are pretty interesting in that they are adapted to the very minimal resources and poor tolerances available in the kit; they include a few different single-transistor oscillators, for example. (I think they're Hartley oscillators, often using the center tap on the audio output transformer for their tapped coil, but I'm not sure I understand them.)

The circuits include various kinds of AM radio transmitters and receivers, various kinds of audio oscillators, games that control audio oscillators etc. with light, a "strobe light" with an LED, push-pull amplifiers, RTL and DTL logic gates, a "door alarm", random number generators, a divide-by-4 counter with decoded output, a VCO, a voltmeter, an ohmmeter, and so on. Many of the circuits use the speaker or piezo earphone as microphones.

It's been 39 years since it was designed, and a few of the components are obsolete (TTL logic, germanium diodes, and variable capacitors) while others are harder to find (CdS cells, piezo earphones, galvos, relays, incandescent bulbs). And nowadays, if you were designing something similar to build out of new parts, you might take advantage of some of the parts that are cheaper and more robust than they were then: power MOSFETs, op-amps (maybe LM324s, TLC272s, and as Viper-7 suggests (see file `notes/jellybeans.html` in Dercuano), TL084s for JFET input), Schottky diodes, Darlington arrays like the ULN2003, zeners, colored LEDs, some 555s, phototransistors, but especially and above all else, microcontrollers. If you're going to have discrete logic circuits, make them CMOS.

Toward a ghettobotics version

If we're limited to parts we can salvage from discarded equipment, what could we patch together?

The easiest way to get wire is from discarded wire, especially power cords, but sometimes also things like telephone line and coax.

Batteries are right out, but there are lots of perfectly capable AC power supplies out there. Surprisingly, the power supply often is not the first thing that breaks; sometimes it's the supply chain.

LEDs, silicon signal diodes, resistors, capacitors, buttons, and switches are abundant, and optointerruptors are found at times; most power supplies also contain transformers, inductors, silicon PN power diodes, and Schottky diodes. Speakers are reasonably common. Crystal resonators are also quite common (this VCR has nine of them), potentially permitting very high precision timing measurements. Potentiometers with knobs attached do occur occasionally, but trimpots are enormously more common.

Even this 12-watt LED lightbulb that burned out the other day in the bathroom has a little power-supply board in it containing two resistors, an MLCC capacitor, a diode, two electrolytic capacitors,

and a transformer (a center-tapped coil, really), plus a couple of chips (one of which may be a bridge rectifier), plus 14 bright LEDs in series, two of which are burned out. Perhaps the power supply works fine and it was just the LEDs that overheated, in which case I have a non-isolated power supply the size of my fingertip designed to supply some 56 volts, 300 mA, from 240VAC. Or perhaps it would be more useful in pieces.

Transistors are a little messier. The VCR, from 1996, has apparently several hundred of them, but apart from half a dozen power transistors in its power supply, they're mostly tiny surface-mount components. I more often find BJTs than MOSFETs, but in this case I haven't looked them up yet.

Inductors are a sufficiently expensive component that the 200-in-1 kit didn't have *any* except as part of its transformers and antenna. But they are straightforward to make by hand from wire, especially for low inductances, or to salvage from discarded equipment.

Connectors are another tricky question. The 200-in-1 kit had only 62 electronic components — including post lugs to attach wires to — but some 80 wires and 176 springs. The dude from Espacio de César demonstrated rigging up a solderless breadboard out of DIP sockets from old circuit boards — snip the two sides off and you have two rows of 2.54-mm-spaced socket holes you can plug pins into. Other connectors, such as DIMM slots or CPU sockets, may also work for this. Through-hole components are easy to slot into those, as long as the leads aren't too short, but surface-mount components need to have pins added to them.

Consumer electronics are by and large full of single-sided PCBs, which are full of jumper wires, which can be pressed into service as pins in a pinch, but a better alternative when possible is to rip apart male Molex-style connectos.

Connectors are also very valuable for a different reason: they permit modularity, and if you're generating, say, an audio or video signal, you can use them to connect it to something external.

7-segment LED displays can still be found in things like discarded clock radios or microwaves, but a better option may be to build them out of now-abundant LEDs and commonplace non-electronic materials like paper and aluminum foil.

CdS cells are virtually unheard of in the last decades, but phototransistors are ubiquitous, though most often infrared, often with shielding. LEDs can sometimes serve as photodiodes, too, although they are poorly characterized for this use.

A soldering iron and soldering flux may be difficult to improvise.

The circuit cookbook probably can't be as cut-and-dried as the Radio Shack cookbook was, because the available components will be more variable.

Bootstrapping sequence

You need to start from basic tools. First you need a power supply with voltage in a reasonable range. But you need to be able to detect that its voltage *is* in a reasonable range. How do you do that without

a multimeter?

See also the note on multimeter metrology (p. 502).

A voltage detector from four LEDs and two resistors

A white illumination LED from a lightbulb can probably dissipate a whole watt, no problem, which is 300 mA or so, and it will probably light up visibly with any current above 0.1 mA. You probably want a couple of separate measuring instruments here, made of two such LEDs in antiparallel in series with a resistor: one to ensure that the voltage is not outrageously high, one to verify that there is some useful voltage.

The not-outrageously-high detector uses a resistor in the 100kΩ–1MΩ range, which should illuminate the LED and heat up the resistor noticeably, but probably not burn up, if placed across a circuit carrying hundreds of volts. Still, you want to make sure you're using a through-hole kind of resistor for this to handle the heat, not a surface-mount. At 100V and 1MΩ you get 100µA, which should be visible on the LED, if barely. If both LEDs light up, you know it's AC.

The some-useful-voltage detector is used after you've established that the circuit doesn't have 100V or more on it, so it uses a resistor in the 330Ω–3.3kΩ range. So those same 100µA will appear, and the LED will start to light up, at 0.033–0.33 volts above the LED's forward voltage drop (typically 3V). At 100V the LED will have 30–300mA running through it and will illuminate brightly. XXX the resistor will explode

XXX Hmm, I need to rethink this a bit. Even at 3.3kΩ the resistor dissipates 3 W at 100V.

The resistors can be pulled from broken or surplus power supplies, which commonly have large resistors in them, and identified using the resistor color code, without a need for a multimeter. It will need to be verified that they do conduct electricity.

By attaching the some-useful-voltage detector to one side of the output of a known-good power supply, you also get a diode and continuity tester.

A variable-voltage linear power supply from a power transistor and a potentiometer

Once you know a given regulated DC power supply works, you need to be able to derive other DC voltages from it. Suppose it's 12V, the highest-voltage rail on an ATX power supply (and typically provided with a lot of current). You can rig a 10kΩ potentiometer across it to get a variable voltage reference, then feed that into the emitter (or gate) of a power transistor whose collector (or drain) is connected to the appropriate power-supply rail, thus giving you an emitter (or source) follower.

This allows you to get whatever regulated output voltage you want, up to a diode drop below the input voltage. But how do you know what voltage you're getting if you don't have a multimeter?

A string of LEDs with parallel resistors to measure power supply output voltage

Three or four LEDs in series to ground, ideally a 1.5-volt indicator type rather than a 3V illumination type, can provide some kind of indication of how high the input voltage is. At below 1.5 V, no LEDs will light. At 1.5 V, the bottom one will light, fed by a string of resistors to it from the voltage input. Successive resistors in parallel with the other LEDs will develop enough voltage to light those LEDs as the current rises; this requires them to have lower and lower resistances.

A Wheatstone bridge to measure unknown resistances and compare voltages

On one side of the bridge we use a potentiometer (presumed linear) with a knob glued to it; the other side pits the unknown resistance against a known resistance. Rather than Wheatstone's galvanometer across the middle, we use a pair of antiparallel LEDs in series with a small protective resistance. This may require that the input voltage be rather high, tens of volts, to get good precision.

With an AC source, I think this setup also works to measure ratios of capacitances or inductances.

Then, it should be possible to replace the crude LED pair with a delicate differential pair of NPN transistors.

These detectors of voltage differences can also be used to directly compare voltages, for example to calibrate positions on the potentiometer knob on the linear power supply against known regulated voltages, either from a multi-voltage power supply or from a 7805 or something.

A VCO to measure voltages and resistances more quickly and precisely

There are lots of circuits for this but I don't know which ones are simple, free of soakage, thermal coefficients, and whatnot. But if you build one you can hook it up to a speaker to listen to your signals; one of the 200-in-1 projects does this.

Topics

Sodium silicate

Kragen Javier Sitaker, 02020-06-04 (32 minutes)

Some notes on sodium silicate.

Nowadays sodium silicate, or waterglass, is principally employed in foundries as a glue for sand-casting of metals, as a concrete sealant against water, and as a grouting agent to solidify soft soils prior to construction projects. Such composites can, at best, be several times stronger than ordinary concrete made with portland cement, and they don't suffer from the grey discoloration of portland cement or, possibly, its carbon dioxide emissions. I'm interested in its possible uses for digital fabrication.

On Mercado Libre nowadays, companies like Geese Química are selling it for AR$140 per kg of "Silige" solution, which is US$1.13 at the current AR$124/US$1 price, and is probably about 400 g of sodium silicate, thus working out to about US$2.80/kg. This compares to AR$630 for 50 kg of portland cement, US$5.10, or 10.2¢/kg. Pure white portland goes for about 50% more, and hydraulic slaked lime is AR$220 for 20 kg, 3.5¢/kg. Portland cement is about 20% of the weight of the final concrete, and lime cement is about 25% of the weight of the final mortar, while for a similar strength sodium silicate can be 5% or less of the weight of the final solid; these numbers work out to 0.88¢/kg for lime concrete, 2.04¢/kg for portland concrete, and 14¢/kg for sodium-silicate-bonded concrete. The price of the aggregate closes the gap a little bit: construction sand costs about 5¢/kg and gravel costs about 3¢/kg, though both are usually sold by volume rather than weight. So the total materials cost might be 5¢/kg for lime concrete, 6¢/kg for portland concrete, or 20¢/kg for sodium-silicate concrete.

So, sodium-silicate-bonded concrete is about three or four times pricier than portland-cement-bonded concrete when they are the same strength. This probably explains why portland is widely used as a binder and waterglass is not. But I think waterglass may have some interesting advantages that can come into play with digital fabrication.

If simply allowed to dry, sodium silicate takes a substantial amount of time, and so it's common to cure it with curing agents — in foundry practice typically CO_2 gas, which can harden it within a few seconds, but in other cases by mixing it with a curing agent, such as calcium chloride or calcium hydroxide.

A lot of the existing literature on using waterglass as a binder focuses on how to *slow down* the curing to minutes or hours, in order to give it a long "pot life". But for digital fabrication, I think it might be more interesting to explore how to *speed up* the curing, ideally into the milliseconds to hundreds-of-milliseconds range. Then you could use it to "print" structures rapidly and with great freedom, without having to wait hours for each part of the structure to solidify before putting the next part in place. But is this feasible? How do we know the structures would be strong? Would it be resistant to weathering? What would it look like — would it suffer from the brutal, grim, gray

appearance of typical portland concrete? Can you stick it to regular glass?

It turns out that they probably would be strong and resistant to weathering, and they can have a wide variety of appearances, from glass to sandstone and a variety of matte or glossy colors. The waterglass itself is transparent, although commonly a bit greenish due to iron contamination. And the possibility of structuring it at the millimeter scale under digital control should make it possible to achieve both stiffness and resilience dramatically better than that of traditional concrete.

Other interesting attributes of waterglass

High-water-content waterglass is used as an intumescent firestop — when heated above about 450°, the glass softens and its water expands to steam, converting the solid, transparent, glassy waterglass into a solid glassy opaque white foam.

Waterglass is commonly used in pottery as a deflocculant, reducing the viscosity of clay slips.

The tensile strength of waterglass-cemented composites can significantly exceed that of ordinary portland concrete, and it has been used as a binder for demanding applications like grinding wheels.

Chemical gardens grow in a waterglass medium; this suggests the speed with which waterglass can be solidified if exposed to the right reagents.

KEIM and mineral paints

One crucial question here for construction purposes is whether waterglass can survive weathering — it's no

The Keim company in Germany, founded by Adolf Wilhelm Keim, has sold a line of silicate-based "mineral paints" for over a century, and the Bleeck company in the UK has recently begun selling a similar line in the UK. Keim has expanded to the UK and USA. These paints are principally based on potassium silicate as a binder, which is very similar to sodium silicate, the principal difference being that solid potassium silicate can be conveniently redissolved in water at room temperature, while sodium silicate requires strong heating. (Some Keim paints instead use sodium aluminum silicate.) These paints are notable for their durability — 15 years is a common lifespan, but Keim claims that they have lasted over 130 years on the Stein Am Rhein building, and that, although "they will normally give 20–30 years satisfactory performance before redecoration is required," it is also the case that "There are many examples of Keim Mineral Paints performing satisfactorily on lime render substrates for periods in excess of 100 years.". I'm not sure whether these examples are interior or exterior.

Their Soldalit brochure claims, "Color shades will not change for decades," and even recommends painting on top of acrylic or latex paint to protect it from weathering "for decades"; Soldalit, unlike their other paints, incorporates silica nanoparticles.

Wikipedia says, "The city hall in Schwyz and "Gasthaus Weißer Adler" in Stein am Rhein (both in Switzerland) received their coats of

mineral paint in 1891, and facades in Oslo from 1895 or in Traunstein, Germany from 1891."

Although sodium silicate itself is water-soluble and will thus redissolve in water, these paints "silicify" in contact with concrete or masonry, forming covalently-bonded water-insoluble hydrophobic products.

So all of this suggests that, in contact with calcite and quartz, these soluble silicates form insoluble materials that will weather at the rate of about the thickness of a coat of paint every 20 to 130 years. This compares favorably to portland cement.

Curing by displacement

Some sources talk about how calcium (hydr)oxide reacts slowly with waterglass because of its low solubility in water (1.7 g/ℓ), and magnesia (6.4 mg/ℓ), litharge (17 mg/ℓ), and minium (undetectably low) do not excel it in this, though Vail (see below) reports that they all cause "immediate precipitation". If we want to speed it further, since the cations are apparently the active element here, more highly soluble salts might be preferred — calcium chloride (750 g/ℓ) is evidently standard, but other possibilities include magnesium chloride (540 g/ℓ); Epsom salts, magnesium sulfate (270 g/ℓ); Norwegian saltpeter, calcium nitrate (1200 g/ℓ); magnesium nitrate (710 g/ℓ); aluminum hydroxide (100 mg/ℓ); aluminum acetate (soluble); alums such as potassium aluminum sulfate (140 g/ℓ) or sodium aluminum sulfate (210 g/ℓ); and neat aluminum sulfate (360 g/ℓ). I'd rather not deal with salts of lead, barium, strontium, cobalt, and so on, although iron might be okay.

I guess these polyvalent cations displace the sodium cations, increasing the degree of connectedness of the waterglass and thus rapidly precipitating it. It took me an embarrassingly long time to figure this out. (I'm preeetty sure aluminum will work for this too.)

What would be super awesome for this would be getting boron to form soluble divalent or trivalent cations, but bor*ate* is of course an anion; boron really likes to make covalent bonds, and most of the compounds you'd hope would be soluble salts are instead found in List of highly toxic gases.

The various mineral species that ought to be formed include the following. The Mohs hardness of the minerals can be taken as some kind of indication of the strength of bonding in the material, but since the materials being formed here are actually amorphous, it is technically incorrect to refer to them as *being* these minerals; the amorphous glass will have different characteristics, including hardness, density, thermal behavior, and perhaps even color.

• Calcium silicates: in the 2:1 Ca:Si ratio, this is the "belite" giving Portland cement its late strength, or "larnite" (Mohs hardness 6) in the wild. This is also called "lime olivine", although properly speaking olivine varies from forsterite (Mg_2SiO_4, Mohs 7, including peridot, a refractory melting around 1900°) to fayalite (Fe_2SiO_4, Mohs 6.5–7). Halfway-lime olivine is the rare monticellite ($CaMgSiO_4$, Mohs 5.5). [Tricalcium silicate], with a 3:1 Ca:Si ratio, is alite, which I think is weaker and tends to revert to belite and lime;

in the 1:1 Ca:Si ratio we have wollastonite ($CaSiO_3$, Mohs 4.5–5, melting at 1540°), noted for its whiteness and used as a filler in plastics, paint, and ceramics; it tends to form long acicular crystals when allowed to crystallize.
• Magnesium silicate: as mentioned above, in the 2:1 Mg:Si ratio, this is forsterite olivine.

I worry somewhat about olivines' vulnerability to weathering, since in an amorphous gel they will be even more exposed to reactions. But the way olivines weather is by incorporating water, as with iddingsite (Mohs 3). If hydroxyls are just incorporated into the olivine structure, you may get humite (Mohs 6–6.5), norbergite (Mohs 6–6.5), chondrodite (Mohs 6–6.5), and clinohumite (Mohs 6).
• Manganese silicate: this is the heavy mineral tephroite, Mohs hardness 6, which exists in a continuum with forsterite and fayalite.
• Aluminum silicate: this occurs naturally as topaz, Mohs hardness 8, although I'm not sure whether you can make topaz without fluorine, but also as several other minerals.

Topaz ($Al_2SiO_4(OH,F)_2$)has a 2:1 Al:Si ratio; other aluminum silicate minerals with the same ratio include andalusite, kyanite, and sillimanite, which are polymorphs of Al_2SiO_5. Kyanite, commonly used as a refractory, is the thermodynamically favored form at STP, and it's highly anisotropic, with Mohs hardness of 4.5–5 along one crystal axis and 6.5–7 perpendicular to it; it can be cooked into mullite and vitreous silica at 1100°. Sillimanite is Mohs 7 and andalusite, also commonly used as a refractory, is 6.5–7.5.

Kaolinite ($Al_2Si_2O_5(OH)_4$) has a 1:1 Al:Si ratio; it is a phyllosilicate clay, with almost negligible strength. Heating it above 550° converts it to metakaolin, a tranformation that is complete at 900°: $Al_2Si_2O_7$; this is used as an excellent pozzolan for pozzolanic cement, but it is still fragile. Further heating converts it into $Si_3Al_4O_{12}$ + SiO_2, quartz and a sort of spinel, above 950°; to platelet mullite 2(3 Al_2O_3 + 2 SiO_2) and cristobalite; at to acicular mullite (contaminated with the cristobalite) above 1400°, which remains solid up to 1840°.

Mullite itself — the key to the alchemists' famous Hessian crucibles — can also form at 3:2 or 2:1 ratios, but I suspect that isn't what you'll get by treating sodium silicate with aluminum salts.

Notes on existing research

Sodium silicate is a bit of a tricky beast to find good engineering data about, because it exists as a continuous spectrum between pure lye and pure fused silica, with a highly variable amount of water, and additionally can react with gases from the air as it hardens.

Gonzalez 2007

"Behavior of a sodium silicate grouted sand" by Gonzalez and Vipulanandan, 2007. Mixed "N-Sodium Silicate" ("Na_2SiO"(!!)·$3H_2O$) with "dimethyl ester" (which ester? "$C_{10}H_{10}O_4$" — clearly these are not organic chemists — "a byproduct of the nylon industry" — oh, apparently it's a random mixture of succinate, "gluterate" (glutarate?), and adipate?) and injected it into "medium dense sand" to grout it in a mold. Compressive strength of the sand was 300–1900 kPa, Young's modulus 200–500 MPa, but it

had creep. No explanation is given as to why they thought adding dimethyl esters would be interesting, but apparently they sped up the gelling, maybe as a source of CO_2, but weakened the final product. Strain at failure was 0.4%–2%. No samples without DME were included. No tensile or flexural strengths were recorded, I guess because they were interested in grouting sands for civil engineering purposes.

I have zero faith in Gonzalez and Vipulanandan; the formula they give for "sodium silicate" would actually be a metallic silicon-sodium alloy which would be at the very least violently reactive with water and possibly pyrophoric. The absence of a DME-free control is particularly glaring (for my purposes) and they don't talk at all about their CO_2-control measures.

Zhao 2011

"Nanoindentation and Brillouin light scattering studies of elastic moduli of sodium silicate glasses" by Zhao et al., 2011. Talks about a "large discrepancy" in Young's modulus measured by different methods (and offers an explanation). They prepared their sodium silicate with varying amounts of sodium (8, 20, 30, and 40 mol%) from Na_2CO_3 and SiO_2 (presumably crystalline) mixed in an agate pestle and then melted at 1500° or, for the 8mol%-Na glass, 1700°, and compared to fused silica. The idea is, I guess, that the sodium carbonate converts to Na_2O when you heat it up.

A thing I'm not clear about with these mole percentages is whether the metals are 8 mol% Na — thus, two sodium atoms per 23 silicon atoms — or whether the oxides are 8 mol% Na_2O — thus, two Na_2O units per 23 SiO_2 units, and therefore *four* sodium atoms per 23 silicon atoms. I'm pretty sure it isn't two sodium atoms per 23 silicon *or oxygen* atoms.

Astonishingly, they got plastic deformation out of the glasses by indenting it with a diamond-tipped "Hysitron TI 900 TriboIndenter", which they then measured with an AFM. The whole methods section of the paper is equipment porno.

They got a 72 GPa Young's modulus for fused quartz with all four measurement methods, down to 67 GPa at 8%, 61 GPa at 20%, 61 GPa at 30%, and about 59 GPa at 40%. There's a bunch of stuff in there about correcting the figures because at the higher sodium contents they give significantly different results, up to 64 GPa for nanoindentation for the 40%.

They also give a "hardness" value in GPa, ranging from 8 GPa for fused quartz down to 4–4.5 GPa for the 40% sodium glass. I'm guessing that this is the compressive yield stress, although I am surprised to learn that these glasses *have* a yield stress; I thought they would just deform elastically until they broke. But I guess in a small enough area you wouldn't have enough energy to propagate a crack, and so even if the glass there powdered, you'd squish it back into the glass surface ("indentation-induced densification", although it's not clear that there was any powdering going on). I don't know. The AFM images make it look pretty fucking rough, and in the glasses with larger amounts of sodium, there's a "pile-up" of plastically deformed material around the outside of the four-micron-wide

triangular craters. But in the lower-sodium glasses, the surface is totally flat outside the craters.

No tensile strength figures are given.

Redwine 1967

"The Effect of Microstructure on the Physical Properties of Glasses in the Sodium Silicate System", by Redwine and Field 1967. It's not a survey paper — it focuses on changes in physical properties that can be obtained by heat-treating glasses within a metastably-miscible concentration range — but it still gives a broader overview of the field. It gives values of Young's modulus E from 8.38–9.36 million psi (57.8–64.5 GPa in non-medieval units) depending on temperature, composition, and heat treatment, as well as measured values of shear modulus G (25–27 GPa), bulk modulus B (33–36 GPa), and Poisson's ratio μ (0.18–0.20). Linear TCE ranged from 4.64 ppm/° to 10.15 ppm/°. No strength of any kind is measured. Most of the paper is concerned with how these vary by temperature.

They don't seem to say how they made the glasses.

It suggests that at low temperatures Na_2O and SiO_2 are miscible at below about 77 mol% Si_2O and above about 97 mol% SiO_2, but between these limits there is a regime where the two materials spontaneously separate into different phases, presumably a sodium-rich phase and a silicon-rich phase. This immiscibility persists up to about 825°, above which they are miscible in all proportions. (The plot only goes down to 500°, though, perhaps because below that temperature the separation processes are too slow to observe.)

Mostly they focus on glasses of 7.2 mol% to 18.4 mol% Na_2O, which is to say, between 92.8 mol% SiO_2 and 81.6 mol% Si_2O, thus covering much of the range where this immiscibility occurs. Within the "unstable" region, they report that heat treatment resulted in phase separation into "two independently interconnected phases", while in the "metastable" region it resulted in "classical nucleation and growth of particles".

(Interestingly, the miscibility limit in this paper seems close to the "pile-up" limit displayed in Zhao 2011 above. This might be a coincidence.)

It might be interesting to see if laser heat treatment could induce this "heat treatment" effect in very small areas very quickly, as a way of writing data; for compositions right in the middle of the "unstable" region, say around 11 mol% Na_2O, the separation might be fastest. However, in the paper, they heat-treated for 1½ hours at 770° to get phase separation at 12.6 mol% Na_2O, so that might be very challenging. However, they noted that they were not able to obtain homogeneous glasses for some compositions, presumably because they could not cool them fast enough.

They measured the "dilatometric softening point" of the glasses from 500° for the highest-sodium variants (18.4 mol%) up to 735° for a heat-treated high-silica glass (7.2 mol% Na); this is the temperature at which heating the glasses does not dilate your dilatometer any further because the viscosity is low enough that it flows instead, which is of course dependent on how much force the dilatometer is clamping with.

The linear coefficients of thermal expansion ($^{\alpha}RT$-350)ranged from 4.64 ppm/° for heat-treated 7.2-mol% Na glass up to 10.15 ppm/° for 18.4-mol% Na without heat treatment, varying linearly. These numbers barely changed with heat treatment.

Ito 1982

"Dynamic Fatigue of Sodium-Silicate Glasses With High Water Content", by Ito and Tomozawa, 1982. These guys were also at RPI. They measured 40–70 GPa Young's modulus for dry sodium silicate and 3–50 GPa for glasses including a lot of water. They also measured its tensile strength but I can't understand their results.

They slowly (over several days) dried out some commercial sodium silicate solution (8.9 wt% Na_2O, 28.7 wt% SiO_2, $Na_2O{\cdot}3.3SiO_2$, which I guess is 23.2 mol% Na_2O) to various water contents around 25%, at which point it was solid; they sliced it into 1.7-mm-thick slips and and used four-point bending to measure its flexural strength, finding a strong dependence on speed of loading especially for higher-water-content glasses, which also had the highest Young's modulus, which was, insanely, *viscoelastic*.

Unfortunately the Y-axis labels on the fracture strength plots are very difficult to understand: it says "Log Fracture Strength (kg/mm²)", which is already ambiguous (is that a base-10 log or base-*e*?) but to worsen the situation, a legend helpfully explains: "$\log \sigma = (1/(n+1)) \log \dot{\sigma} + \log C$", only without the parentheses. Is that an empirical approximation formula or does it explain how the plotted numbers were derived? The numbers plotted, at any rate, range from about -0.1 to about 1.2, with the strongest glass typically being the one with 15.9% water, which is slightly stronger than the dry glass. If we suppose that this is a base-10 logarithm of the flexural strength, then we have a tensile strength of about 0.8–16 kg/mm², or 8–160 MPa in modern units. But I am not confident in that interpretation.

The Young's-modulus plot in Fig. 4 is, by contrast, decently labeled — it uses a logarithmic Y-axis but with ticks labeled in real units. It gives 4–7 thousand kg/mm² (40–70 GPa) for the dry glass, with numbers ranging from 0.3–5 (3–50 GPa) for the wet glasses.

Their figure 5 also plots Young's modulus, a theoretical Young's modulus limit at infinite stress rate, which is some three orders of magnitude lower, ranging from 1 kg/mm² to 5.5 kg/mm². I suspect they have mislabeled their plot.

They also plotted the Knoop hardness of the samples, in the range 50–400 kg/mm² (500–4000 MPa), decreasing with higher water content.

They cite "McMillan (1982)" as giving flexural strengths for soda-lime silica glass, which looks like a paper in "Non-Crystalline Solids" by McMillan and Chelebik, 1980, I think volume 38/39, p. 509. I think that's actually *Chlebik*, and the paper is perhaps "The effect of hydroxyl ion content on the mechanical and other properties of soda-lime-silica glass". But it seems like probably that paper doesn't cover soda-silica glass. (And they didn't say it did, after all.)

Medina 2009

This article has the deeply misleading title, "Water Glass as

Hydrophobic and Flame Retardant Additive for Natural Fibre Reinforced Composites," by Medina and Schledjewski, 2009. I say "deeply misleading" because waterglass is preeettty faaar from being hydrophobic! As noted above, drying the stuff out is really tough.

The article has a lot of problems like that. It describes a $Si(OH)_4$ moiety as "silane", talks about "natural fibers" as if they're all equivalent of (I was assuming cellulose because the descriptions they give don't fit chitin, keratin, asbestos, etc., but even if it's cellulose not all cellulose is the same — finally on page 3 we find out that the fiber they tested is 70% kenaf, 30% hemp, with no source given), never describes which acrylic resin it's using (I think, although sometimes it mentions "polyester", so maybe it's a polyester acrylic — although on page 8 they finally slip up and admit that it's one of the Acrodurs, whose composition is apparently secret), never describes how much sodium is in the waterglass it's using, uses a very crude flammability test, etc., etc.

But it's pretty interesting. Apparently they glued together some cellulose fiber mats with various mixtures of sodium-silicate waterglass and the unspecified acrylic resin, and got some decent boards out of it, and of course the waterglass made them flame retardant.

Because of the amount of crucial data omitted, apparently intentionally ("a new water glass type specially developed as hydrophobic additive for acrylic systems"), the paper falls far short of basic reproducibility criteria.

Fused quartz properties

The low-sodium endmember of the sodium silicate continuum is fused quartz, and that's the most highly polymerized part, so we would expect all sodium silicates to have tensile strength and hardness at most that of fused quartz.

http://www.quartz.com/gedata.html agrees with https://technicalglass.com/technical_properties/ on the curiously precise tensile-strength number of 48 MPa. Marijuana paraphernalia merchant https://highlyeducatedti.com/blogs/information/thermal-shock-vs-tensile-strength gives 67 MPa for flexural strength and 50 MPa for ultimate tensile strength, apparently quoting makeitfrom. It also gives 0.5 ppm/° linear TCE.

Stachowicz 2010

"Studies on the Possibility of More Effective Use of Water Glass Thanks to Application of Selected Methods of Hardening", by Stachowicz, Granat, and Nowak, 2010. They say that waterglass-bound foundry casting sand commonly has tensile strengths (R_m^U) in the 0.3–0.5 MPa range; with 5% waterglass in their sand they got tensile strengths as high as 3.6 MPa, with higher-sodium waterglasses generally giving stronger bonds.

They're concerned with binding foundry sand with small amounts (1.5–5.0%) of waterglass, and in particular with whether microwave heating can make it stronger and maybe allow you to use less than the usual minimum of 2.5%, which it apparently does. Also they were

able to microwave their samples for four minutes instead of oven-drying them for two hours.

It has a helpful table of waterglass grades used in foundries, with molar ratios of SiO_2 to Na_2O anging from 3.2:3.4 (grade 137) to 1.9:2.1 (grade 150).

I'm not sure whether their 1.5% and 5% etc. refer to the weight of the dried waterglass or to its wet weight. (Grade 137 is 35% solids, with the rest being water, while the very viscous grade 140 is 42.5% solids.) Anyway, the strength continues to increase quite linearly up to the 5% they tested, which makes me optimistic that strengths several times higher are feasible with higher binder content.

The linear extrapolation of the 1.5%–5% suggests a tensile strength of something like 50–70 MPa for solid 100% waterglass, which is consonant with my tentative 8–160 MPa interpretation of Ito 1982 and the 50–70 MPa numbers given above for fused quartz.

Carbon dioxide is not mentioned.

All nine entries in their bibliography are Polish.

MacKenzie 1991

"Silicate Bonding of Inorganic Materials, Part I", by MacKenzie et al., 1991.

XXX

Vail 1952

"Soluble Silicates: Their Properties and Uses", Vail, 1952. This is a thousand-page two-volume set full of valuable information.

It mentions that a major use of waterglass in the mid-1800s was "the hardening of stone to increase its weather resistance", further allaying my concerns about weathering, and it has a whole section on using it to bond grinding wheels. It mentions that Feuchtwanger claims to have introduced the use of waterglass in the US, using it to prevent rusting of naval weaponry.

It seems that when Vail wrote his book, sodium silicate was considerably more widely used than it is today: "There are few manufacturing plants which do not make some use of [soluble silicates]." Today I think it's kind of a niche product, despite the growing importance of avoiding phosphate runoff (silicates can substitute for phosphates as detergents). This consideration does not appear in the introductory section, although it does talk about how conservation may stimulate the use of silicates in the future.

With respect to the prospect of precipitating or "curing" waterglass, Chapter 2 ("Present Practices") begins wih the promising note: "Most of the impurities likely to be found in sand form insoluble silicates, and even small quantities, less than one per cent, can create serious difficulties." It has the appealing note that the old way of making it was "dissolving diatomaceous earth in caustic liquors", which does sound much easier than the standard approach of heating sulfate or carbonate of soda to some 700° to 800° in contact with sand. On the other hand, the standard approach is considerably more legal in Argentina.

It explains that the "so-called neutral glass", usually "pale bluish or

greenish", is 1:3.3 Na_2O:SiO_2, although IIRC the pH of the solution is still above 11, while the "alkaline" is 1:2.1. This probably explains why the pale greenish bottle I have doesn't burn my skin and was sold as "neutral".

Astoundingly, at this time it was still not known that solid waterglass, or indeed any solid, was amorphous! Vail says the question "might be of more academic than practical value", though he also said, "A sodium silicate is as nearly devoid of ordered structure as any known material."

It explains that finely divided dry waterglass sometimes *does* get dissolved in water at atmospheric pressure 100°, but to dissolve lumps of glass, 90–100 "pounds gage" steam pressure is used (psig I guess, so 700–800 kPa absolute).

It explains that the reason sodium silicate has eclipsed potassium silicate is just that sodium is cheaper than potassium.

I find this unjustifiably amusing: "Immediately after use, hydrometers should be washed thoroughly with warm water until alkali cannot be tasted on the glass..." — clearly a pre-OSHA book.

He points out that you can blow waterglasses just like you can blow other glasses, but that it can contain varying amounts of water "without substantially altering their appearance". This makes me wonder if they might be a particularly suitable material to attempt to 3-D print graded-index optics in.

It explains that alcohol precipitates waterglass just by removing water, which I had suspected but was not sure of. Also, he mentions doing the same with alkali metal salts or ammonia.

It includes the oldest citation I've seen: "A sodium silicate glaze is described in cuneiform records of the reign of Ashurbanipal, 668–626 B.C.: 10 mana of sand, 10 mana of alkali ash, and 1.67 mana of styrax gum were heated to white heat, cooled, crushed, and placed in a clean melting pot in a cold furnace."

A surprising thing mentioned a couple of times in the book is that potassium silicate does not effloresce, while sodium silicate does, a fact particularly relevant for production of fake stone; this afflicted Ransome's fake stone in 1861.

A technique frequently mentioned both in this book and in Keim's paint brochures is the inclusion of amorphous silica particles in the liquid — a sol of precipitated silica gel particles, for example, although diatomaceous earth should also work. This reduces the amount of the waterglass that must be gelled to form a solid gel, since the particles form part of the gel network. Other effects include thickening the liquid and making it colloidal and possibly thixotropic.

In Chapter 5, Vail refers to "immediate precipitation which occurs when calcium, magnesium, or lead oxides are mixed with concentrated silicate solutions", although it's not clear what timescale he's talking about.

Topics

One big text file

Kragen Javier Sitaker, 02020-06-04 (updated 02020-06-06)
(20 minutes)

Here's an interesting idea for how to do Derctuo: a giant WYSIWYG document whose source format is a plain text file including data, code, text, and formatting in a single document, potentially of 128 mebibytes or more; but with computational output rigidly segregated to a cache management system.

Precedents

Danny O'Brien's Life Hacks

The immediate inspiration for this is Danny O'Brien's "Life Hacks" ethnographic research finding the widespread use of One Huge Text File. He found that many of his interviewees maintained all their notes in a single humongous text file, which they navigated by text search. On a modern computer, Emacs incremental-search is capable of searching through hundreds of megabytes per second, so it's rare to even need any indexing.

Volks-Hypertext

Eric Raymond's "Volks-Hypertext" browser for the Jargon File demonstrated how to improvise a fairly instantaneous hypertext system atop a large text file: the text file was rendered in more or less the usual way, but keywords in curly braces like "{grok}" were treated as links to a line beginning with ":grok:", and the file was preprocessed to generate an index of all such lines after the fashion of ctags, with byte offsets stored. Searching the index file and jumping to a given byte offset was reliably fast, even in MS-DOS on a 386.

askSam

The cult semistructured database askSam has barely more structure: an askSam file is a collection of records, which are just free text strings up to a few kilobytes in size, with fields defined by searching for the field name followed by square brackets — everything on top of that is added by the askSam query language. A full-text index makes relatively powerful queries acceptably fast.

Alph and Halp

Darius Bacon's Alph (A literate programming hack) and Halp systems automatically re-evaluate all the specially-marked code in a document upon demand, placing the resuls of each snippet after the snippet itself.

Org-mode

Org-mode adds a little bit of lubrication to text-file viewing: the Emacs outline-mode ability to collapse and expand sections of the file, but with more pleasant keybindings. And it also has magic syntax for inserting hyperlinks: [[http://example.com/][example URL]] displays just as an underlined "example URL", but links to the given

URL, and it also supports links to places within the file. Org-mode's "src blocks" offer the possibility to display textual or graphical output inline in the editing buffer.

Cassowary and TeX

Cassowary is a constraint-based layout system that offers perhaps a bit less power than CSS, but has extremely efficient algorithms to execute it. (I haven't actually tried it.) TeX, too, has extremely efficient layout algorithms which also produce somewhat nicer results than CSS.

WordPerfect Reveal Codes

Before Microsoft Windows, WordPerfect was the most popular word processing software, and its users' favorite feature was a thing called "Reveal Codes", which split the screen into one half with the WYSIWYGish text you were editing at the top and a complete representation of the word processor's underlying representation at the bottom, with formatting markup displayed in between bits of text. This made it easy to see why your document was formatting incorrectly and fix it.

Lotus 1-2-3

Lotus 1-2-3 displays the tabular output of a program written by the user by defining formulas in cells. It analyzes the dependencies between the cells to discover as safe dependency order to recalculate them in when there is a change, and it only displays the source code of the cell you are editing at a given moment. It imitated VisiCalc, the "killer app", but its dependency-order recalculation was new.

Jupyter notebooks

Jupyter's "notebook interface" is an enhanced REPL which permits the inline display of graphics, text formatted with LaTeX or HTML, etc., as results of the REPL commands ("cells"). It is accessible via HTTP or HTTPS, allowing people to share code easily. Also, it stores the output in the same text file as the code in the cells, even when it is graphical or irreproducible.

Jupyter has become the standard interface to programming for an enormous number of people nowadays. But it has some serious drawbacks: the output displayed may not be up to date with the code in the file, re-evaluating the whole file may not be safe (it's common for people to put utility scripts in notebooks that do things like wipe a database), the output being interpolated into the source code makes the notebook files bulky and difficult to version-control with systems like Git, it's awkward to reuse code, and normally you have to start out the notebook with a bunch of preliminary noise like module imports.

Explorable explanations

"Explorable explanations" are, mostly, web pages containing interactive visualizations of algorithms; the best ones I've seen are Amit Patel's, for example his visualization of A* pathfinding or of generating terrain with Perlin noise. Mike Bostock, the author of d3.js, has written many excellent explorable explanations as well. The

objective is to explain how a given algorithm works by means of exhibiting its internal functioning on example data. Bret Victor has explored much of this territory as well, for example with his visualization of Nile, and articulated guiding principles for the field: that people engaging in creativity should be able to get instant feedback on the implications and results of their ideas.

ObservableHQ

ObservableHQ is Mike Bostock's exploration of how the notebook interface could be improved. It uses a slight extension of JS as its language, its cells each define a single value, and like Lotus 1-2-3, they are evaluated in dependency order.

R-Markdown and R Notebooks

Yihui Xie's R-Markdown is a system (included in the free-software R Studio, but also invocable from the command line) which extends Markdown with embedded chunks of code in the R statistical programming language and textual and graphical output produced by that code; the code is optionally not visible in the output (echo=FALSE). By default, this "knitting" of the source R-Markdown document into a PDF or HTML output with the graphics is a batch process, but for some time R Studio has also had the option to evaluate these embedded code blocks interactively with control-shift-enter, sending its output to the R Studio console pane. Because the chunks are normally run in order, it is up to the author to track the dependencies between them and topologically sort them in the file and to re-execute dependent chunks when changing a thing they depend on.

However, recent version of R Studio have added an "R Notebook" mode which displays the outputs of code blocks inline in an R-Markdown document (whether textual or graphical), instead of in a separate pane. Rerunning the code and thus updating these outputs after changing the code continues to require an explicit run-current-chunk command, so the author is still responsible for keeping track of the dependencies.

Unlike Jupyter, R Studio stores the output from the embedded code in a separate file: an "R notebook" named foo.Rmd will have an accompanying foo.nb.html which includes the text and graphics generated from it, while foo.Rmd itself contains only the human-authored source code. Xie's explicit ambition is to improve the reproducibility of computational research.

`make` and other build systems

Stu Feldman's `make` program, included with the UNIX operating system for the PDP-11, is directed at accelerating the feedback programmers need to improve their programs: by caching the results of compiling parts of the program, automatically determining which parts of the program have been edited since they were last compiled, `make` can greatly accelerate the process of rebuilding the program after a small change. It does this in an almost wholly compiler-agnostic fashion: like ObservableHQ, it only knows how to produce each of the intermediate results in the build process by invoking some opaque code, and what the inputs to that code are. `make` does this at the

granularity of files and batch program invocations, while ObservableHQ does it at the granularity of variables and snippets of code, but modern software like Lucet can reduce the overhead of starting and stopping a program to under 100µs, while modern software like FlatBuffers or HDF can reduce the overhead of a program consulting serialized input data structures to a minimum.

A limitation of `make` is that its knowledge of dependencies is not reliable --- it relies on the programmer to describe the dependencies in a "Makefile", but usually the Makefile fails to capture the full dependency graph. For example, it is common for `make` to be unaware that an object-code file depends on header files within a project describing the ABI of other object-code files, a case for which various "makedepend" systems have been devised; also, though, the object-code files depend on system header files external to the project and on the version of the compiler used, in the sense that different object code would be emitted if the compiler or system header files had been a different version. The fallback response to all of these problems is `make clean`, a conventional phony build target whose "build rule" deletes all the files created by the whole build process so that a subsequent execution of `make` will regenerate everything from the virgin source code.

Other build systems, such as Apollo DSEE, its imitation Vesta, their imitation ClearCase, Nix/Guix, Gitlab-CI, Urbit, and the popular Docker, instead run the build steps in an environment more or less isolated from anything that isn't explicitly provided to that build step as an input. Because of the limitations of determinism in conventional computing systems, these systems do still sometimes fail to deliver full bitwise reproducibility, but they do aspire to it, except possibly for Gitlab-CI.

SPARK

Apache SPARK XXX

ActivePapers

Konrad Hinsen's ActivePapers research effort XXX

The Java Virtual Machine

The JVM's WORA aspirations XXX

Geometer's Sketchpad, KSEG, and GeoGebra

Falstad's Circuit.js

Design

So suppose we have a thing that is "really" just a huge text file, but formatted in a WYSIWYG format like a book, and structured hierarchically into sections and subsections in an org-mode-like way. It uses a layout algorithm with good efficiency and adequate power. You can include snippets of code into the file, easily toggling whether the WYSIWYG view displays the code, its output, or both; output can even be easily interpolated into the middle of a paragraph, with a construct something like ${foo}. The code can easily run various kinds of ad-hoc queries on the file's own contents. Bits of code

defined in one section of the file can be invoked from other sections, although a hierarchical namespacing mechanism limits visibility and makes it easy to track dependencies. It's easy to define data tables and add computed columns to them, and use the data in those columns in other computations. The file can define user interfaces for things like drawing geometrical compass-and-straightedge constructions, RPN calculations, or schematic capture, and the data thus created becomes part of the text file — and then it can be used as input to other code.

The output of code is strictly segregated from the "source" text file, which contains only things the author explicitly chose to put into it, but the code is deterministic and the outputs are cached in a file off to the side so that they can be redisplayed without recalculating them.

You can toggle between a "source" view, which shows the full contents of the file, and the WYSIWYG view, or have both displayed at once.

The idea is that it should scale to 8 mebibytes or more of text written by a single author and perhaps 128 mebibytes of other data imported into the file from elsewhere: a personal memex, but taking advantage of the computer's power to augment human intellect through more than just copying and retrieval of information. A smooth path allows ideas to gradually be solidified and explored: from back-of-the-envelope calculations through sketches and simple simulations through to refactoring into reusable parameterized models.

A crucial question for navigation is how interactive searching of outputs works. If you stick to searching only the source-code form of the file, searching can be very fast, but in many cases you will be missing the most interesting data. On the other hand, that data can be immense and full of things that are essentially random noise.

Interactivity and persistence

Above I said that computational output is rigidly segregated to a cache management system — the code within the document cannot mutate the document. Only the user can do that. How can this be reconciled with the need to add sketches, photographs, geometrical constructions, circuits, DAGs, cellular automaton configurations, and the like? Surely the user cannot always be expected to type in text from which they can be computed!

Ephemeral explorable explanations like Amit Patel's A* examples mentioned earlier pose no problem for this model at all. A code chunk can evaluate to a function from (x, y) pairs to (r, g, b) colors, for example, to produce an infinitely zoomable, pannable image; that function (call it a "paint method") can run in an environment where it has no authority to access any state other than (x, y) coordinates of requested pixels or to mutate anything outside of its own local state. Mouse coordinates and time can be provided to a paint method in a similarly stateless fashion, as they are on Shadertoy.

Fragments

The movable blob position requires at least *some* state to persist from one call to the next; this can be handled by an object consisting

of a pair of pure functions: one that maps a (current state, user input event) pair to a new state (call this the "react method"), and another that maps the current state to an image or animation (the paint method from before).

This kind of state turns out to be sufficient to implement things like Falstad's Circuit.js! (However, such simulations additionally benefit from some kind of way to maintain their simulation state from frame to frame, even when there is no user interaction to react to; for the time being I will ignore this.)

Suppose we call this new persistent state, which can change in response to things like clicks and keystrokes, the "fragment"; it's analogous to the #fragment in an URL on the WWW. Accordingly, it provides the surrounding framework with the freedom to measure its persistent memory consumption, pause it, save a fragment, go back in time by reverting changes to the fragment (undo), explore alternatives from an earlier fragment (nonlinear undo), and copy and paste the fragment to somewhere else in the document. This is sufficient for things like sketching illustrations, self-contained circuit modeling, or doing geometrical constructions. Indeed, given camera access, it could even be sufficient for taking photos. (There's no reason the fragment needs to be limited in size like URL fragments traditionally are.)

The fragment itself is part of the source format document, just a part that can be edited by the widget's embedded code, subject to the restrictions above about undo and the like.

Methods other than "paint" and "react" could provide requested layout sizes or render the "widget" as a series of boxes rather than a single window onto a canvas.

However, so far all of this focuses on applet-like content: a calculator, compass-and-straightedge interaction, or circuit simulator, displayed in a window with text flowed around it, or perhaps overlapping part of it. It doesn't cover the kind of interaction you'd want for data visualization, much less a general computing platform: you want that calculated result to be accessible for further calculations elsewhere in the document! And you want to be able to feed the circuit you've modeled to other analysis functions that you write on the fly. You want the data to be open and accessible, not sealed inside an opaque Actor.

Darius Bacon points out that if the "fragment" state is some more structured thing, such as a state of, say, a relational database, it might be easier to deal with the opacity problem. Maybe it would be easy enough to say something like `drawing2.points[3].x` elsewhere in the document. (Formats other than relational data might be usable too, such as JSON structures, but they tend to vary more over time as navigational data is included.)

Blossoming forms

XXX rewrite

Interactive blocks, HTML forms, BASIC with line numbers, and HP 3000 terminals suggest a somewhat unrelated approach. I tried to write a FORTRAN program on the HP 3000 that the local computer museum got up and running. An interesting thing about the HP 3000

terminals is that they can do local editing, and apparently text files in their system have line numbers, like in old BASICs. So, the editor on the host is a pretty dopey line-mode thing similar to ed, but it includes the line numbers before the lines it prints out. So, locally to the terminal you can go up with the arrow keys into the scrollback buffer and edit one of those lines, interactively, inserting and deleting in a WYSIWYG way, and hit enter to send it back to the host. When you send it back to the host it has the line number still attached, and the editor interprets that as a command to replace the contents of that line number.

GW-BASIC did this too. Maybe Applesoft BASIC too?

HTML forms are kind of the same thing except that the line numbers are words and they're hidden. You could imagine an interactive block that's sort of similar to an HTML form but maybe without the submission delay to run code to see results, and you could imagine it having buttons in it that, when clicked, blossom out into new nested formlets there in place. As long as all the code can do is display its results, or blossom out into more little bomblets, the degree of danger is pretty limited.

However, can this approach really handle things like sketching with the mouse or a stylus, or schematic capture?

Naked objects

XXX

Thanks

To Darius Bacon for discussion of these ideas.

Topics

- ObservableHQ
- Notebooks
- Nix
- Lucet
- Jupyter
- Guix
- Explorables

Monoid prefix sum

Kragen Javier Sitaker, 02020-06-05 (13 minutes)

The parallel prefix-sum or scan algorithm makes it possible to calculate a prefix sum on N elements in O(log N) time on an unbounded number of processors. As Stepanov may have been the first to point out, this algorithm is applicable to general monoids, although its performance only remains O(log N) if the monoid operation can be computed in constant time.

Like unto many other parallel algorithms, parallel prefix sum can be easily converted into an incremental algorithm through a tricky time-space switcheroo: we can cache all the values computed during the algorithm, and upon a small change to the input, we can treat the values computed from unchanged parts of the input as if they were values computed on other processors, receiving them from the cache as if they were received over the network. This gives us a logarithmic-time way to incrementally update the reduction of an arbitrary (constant-time) monoid over an input sequence, since that is the final element of the scan — for example, the sum is the final element of the prefix sum. (Integer sum in particular admits more efficient implementations, because it is not just a monoid but an abelian group — in constant time, you can simply add the inverse of an element that is being removed. But, for example, semilattice operations are not so forgiving.)

In a sense any algorithm that produces a result from input data is a reduction followed by some kind of final postprocessing; the input data comes in some sequence, and in the degenerate case, the reduction is just in the free monad, concatenation — the reduction is just the concatenation, and then the final postprocessing is the algorithm itself. But of course that doesn't give us any parallelism or incrementality advantages.

Testing associativity in $O(N^3)$ time

Suppose we do have some kind of interesting iterative processing going on over the input data, though, formulated in a monoidal way: we have a lifting operation that maps an input element into a "lifted element", a composition operation that maps a sequence of two lifted elements into a single equivalent lifted element, and perhaps a postprocessing operation that maps a lifted element representing the whole sequence into the result we wanted. But to be able to use it correctly with the prefix-sum algorithm, we need to be sure the composition operation is really monoidal, which is to say, associative. How can we verify this?

It may not be possible to verify rigorously in all possible cases, but it is at least reasonably efficient to verify that it is associative over a given input string of N elements, requiring $O(N^3)$ time, using a dynamic-programming-like algorithm. The input string contains $N(N+1)/2$ nonempty substrings, each of which can be divided into two nonempty substrings in less than N ways. So we create an array of lifted elements for these $N(N+1)/2$ nonempty substrings, and we

calculate the reduction value for each of these substrings in all possible ways. For substrings of a single element, we simply use the lifting operation. For each substring of M > 1 elements, we test all of the possible M - 1 divisions into nonempty substrings by applying the composition operation M - 1 times; they should all produce the same value, which we then store into the array.

For "reasonable" composition operations, it should be possible to do this test for sequences up to lengths of a few hundred in under a second, perhaps a few thousand. This does not of course amount to a proof that the operation is monoidal, but it may be a fairly convincing test.

A trivial example: canonicalization of a binary carry-save sum

So, for example, the string ABCD, of length 4, has the 10 nonempty substrings A B C D AB BC CD ABC BCD ABCD. Suppose that, for some inexplicable reason, we want to reduce this string with the function λs c . s × 2 + ord(c), which takes the previous state, multiplies it by two, and adds the ASCII value of the input letter to it, starting with an initial state of 0. Our "lifted elements" are an ordered pair of integers (n, k), representing the function λs.s × n + k. The lifting function maps a letter c to the pair (2, ord(c)). The composition function maps two pairs (n1, k1), (n2, k2) to the equivalent function (n1 × n2, k1 × n2 + k2). So A becomes (2, 65), B becomes (2, 66), etc.; AB becomes (4, 196), BC becomes (4, 199), CD becomes (4, 202); ABC can be computed either as A + BC = (2 × 4, 65 × 4 + 199) or as AB + C = (4 × 2, 196 × 2 + 67), giving in either case (8, 459); and ABCD can be computed either as A + BCD, AB + CD, or ABC + D, giving the same result (16, 986) in all three cases.

(In this case, the postprocessing operation amounts to simply taking the second item of the tuple.)

Ropes

Thus if we annotate rope nodes with lifted elements, we can incrementally update the monoidal reduction of the whole rope even after insertion and deletion operations; it isn't necessary for the lifted elements to correspond to elements whose counts are powers of two. I think Raph Levien has done this for his Xi editor.

CRDTs

By applying this incremental monoidal reduction approach to logs of historical events with a well-defined total sorting order, we can derive a wide variety of efficient CRDTs. We use the standard union CRDT on a set of historical events, merging newly-received events into a rope of already-received events and recomputing the lifted elements on the updated nodes. This allows us to efficiently recompute the monoidal reduction over the updated dataset.

In particular, we can derive common CRDTs in this way, such as a dictionary updated by upserting and deleting key-value pairs; Okasaki's FP-persistent data structures are likely useful here. (I suspect this is actually how Datomic works.)

Further efficiency issues

Of course, if the lifted elements contain some arbitrarily large data structure, or if the composition operation or postprocessing is arbitrarily expensive, then you can lose the efficiencies. Running the above example composition function over the input string "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz" gives the lifted value (4503599627370496, 297237575809105796), each value growing by one bit per additional input character.

If memory is sufficiently expensive, or recomputation from scratch is sufficiently cheap, it may not be worthwhile to cache these lifted values for every element of the input sequence; it might be sufficient to cache one out of every 32–2048 values, thus saving 97–99.95% of the cache space while still limiting the work to recompute a value at a given location to the redundant reprocessing of 31–2047 input elements.

I think some of these ideas originated in discussions with Darius Bacon, but I can't remember.

Two text-editor-oriented examples

Columns

Text editors commonly want to know what column of the screen to display a character in, which depends on how many characters precede it on the same line. (And, possibly, how wide the screen currently is, and how wide those preceding characters are.) In the simple case we can compute this from the beginning of the editor buffer with a simple loop:

```
size_t col = 0;
for (int i = 0; i < point; i++) {
    col++;
    if (buf[i] == '\n' || col == screen_width) col = 0;
}
```

The state of `col` after an iteration of the loop depends on its state before that iteration and on `buf[i]`; it is either 1 + old `col` or 0. Arbitrary compositions of such iterations give us either (n + old `col`) % `screen_width` or n % `screen_width`, so those are our monoidal "lifted values". It's reasonable to store only one out of every 1024 or so such values, and we could probably use the sign bit to distinguish between them (at the expense of not being able to handle a single buffer occupying more than half of your address space), so we can probably use a single pointer-sized word for a lifted value.

In C we probably have to manually compile that into fiddly integer manipulation, so here is the composition function in OCaml, for clarity:

```
type lifted = Absolute of int | Relative of int
let compose left right = match (left, right) with
| _,          Absolute n -> Absolute n
| Absolute m, Relative n -> Absolute (m + n)
| Relative m, Relative n -> Relative (m + n)
```

```
# compose (Absolute 8) (Absolute 5) ;;
- : lifted = Absolute 5
# compose (Absolute 8) (Relative 5) ;;
- : lifted = Absolute 13
# compose (Relative 8) (Relative 5) ;;
- : lifted = Relative 13
# compose (Relative 8) (Absolute 5) ;;
- : lifted = Absolute 5
```

You can fire off a background thread upon opening a file to precalculate these values for the whole file, and then incrementally maintain them thereafter in the face of insertions and deletions. Moreover, since one of the cases of the composition function doesn't even depend on the left side, you can calculate it lazily *backwards* from a given position in the file — you need only search backwards until you find the beginning of a line.

Syntax highlighting

Text editors commonly do syntax highlighting based on data like which identifiers are in scope at a given point, and what their types are, and at least a tokenization of the source code, though sometimes not a full parse (since, after all, we don't want our syntax highlighting to entirely disappear if the input has a parsing error somewhere off the screen). Lexical scanning of the input is usually done with a DFA, or a slight extension of a DFA, for example to accommodate shell-script here-documents (the input for `<<wibble` ends at the first line that says `wibble`) or Lua fat parentheses (a string starting with `[=======[` continues to the next `]=======]`, where 7 has any value).

Considering purely the DFA case, we can consider the "lifted value" of a string to be the mapping from the state the DFA is in at the beginning of the string to the state that it's in at the end of the string. If the DFA has 16 states or less, we can represent such a mapping in a 64-bit register, since it contains 16 nybbles.

In this case the mapping tends to pretty quickly converge on a function that ignores its input, although there are exceptions. C doesn't have nested comments or multiline strings, so as soon as you see a newline you know you're not in a string, and as soon as you see a `*/` outside of a string you know you're not in a comment, so in C typically you can calculate the precise DFA state before going back too far. OCaml, by contrast, does have nested comments, so, in theory, to highlight any position in the file, you have to go back to the beginning of the file to ensure you're not inside of one. You can't tokenize it with a strict DFA.

The identifiers that are in scope at a given point can similarly be adduced, typically, by calculating a "lifted value" that is either a stack of identifiers that are in scope at various levels of scope stacked on top of whatever was in scope at the beginning of the substring, or a count of scopes to pop. So, for example, in a C-like language, this string

```
{
    int x = 3;
    {
```

```
        int y = 4;
        int z = 5;
```

might lift to the stack effect [{}, {x: var}, {y: var, z: var}], augmented with a bit of scanner information; meanwhile the string

```
            }
        }
    }
```

might lift to an instruction to pop three stack levels of declarations.

Languages like JS that have entities without forward declarations cannot be handled in this fashion; they need a preliminary pass over the whole file first to index the declarations.

Topics

Writing a shopping list in TeX

Kragen Javier Sitaker, 02020-06-05 (4 minutes)

I was watching Luke Smith on YouTube touting R-Markdown as a better alternative to LaTeX, and I was struck by his declaration,

> Now, the thing about LaTeX, and it's always the elephant in the room when you're talking about LaTeX, is that a lot of the *basics*, it, well, let's put it this way, LaTeX is great for making *research papers* and *term papers* and doin' advanced projects 'n' stuff like that, but LaTeX syntax is *very cumbersome*. So if I just wanna make a *shopping list* in LaTeX or something, I mean, I wouldn't make a shop, I make shopping lists on this [holding up a pad of paper], but, if I wanted to make a really simple document to give to my students or uh, you know, to give, you know, just a *memorandum* or something like that, uh, LaTeX is a pain because you can't just open it up and start writing, you have to `\documentclass{article} \begin{document} \end{document}`, all this kind of stuff, uh, to do thangs like bold, italics, you have to literally *go in and write* `/textbf{bla bla bla}`, and you know the *backslash*, it's like the most *annoying key* on the computer to actually, like, hit.

I thought I'd check to see if he was right, so I ran `emacs shoppinglist.tex` and typed

```
C-c C-e <return> <return> <return> C-c C-e <return>
c a r n e C-c C-j h u e v o s C-c C-j p o l l o C-c
C-b <return> <return> C-c C-b <return> C-c C-b <return>
<return>
```

which produced this shopping list, rendered and on the screen in an xdvi window:

```
\documentclass{article}

\begin{document}
\begin{itemize}
\item carne
\item huevos
\item pollo

\end{itemize}
\end{document}
```

It took about 30 seconds, but 10 of those were starting up a new Emacs so that I could time the process more easily.

The initial ^C^E prompted me for the environment name (default `document`) and documentclass (default `article`) and options (default none). The second ^C^E prompted me for another environment name; as it happened the default was `itemize` because the last thing I'd done in LaTeX was also to make a list, so I just hit `<return>` again. The ^C^J is the sequence to separate list items. Then the ^C^B sequences run latex and xdvi to see the rendered document.

Unsurprisingly he's also wrong about "things like bold, italics"; although you can `\textbf` if you want, it's probably easier to say

```
PUAs are {\bf losers}.
```

Which renders as, "PUAs are **losers**." It's two characters longer than the Markdown version, admittedly, and I do like Markdown a lot, but I think LaTeX is getting a bad rap here. You can totally write your shopping lists in LaTeX --- it's not quite as easy as writing them in Markdown but the difference is very small. Maybe 10 seconds of overhead.

Where LaTeX becomes difficult is when you're trying to do more complex things in it. Markdown saves you time there because you know you can't do complex things at all in Markdown, so you don't try.

The major advantage of R-Markdown from my point of view is that you can embed your R code in it.

Topics

A 6-bit “variac casero”

Kragen Javier Sitaker, 02020-06-06 (22 minutes)

Watching the YouTube channel of Espacio de César, I was amused to see him describe a “homemade 8-bit variac” (“variac casero de 8 bits”). He suggests winding 8 secondaries of different sizes on a single transformer whose primary is connected to 240 VAC: one that produces 1 VAC, one that produces 2 VAC, and so on up to 128 VAC. (He’s using a microwave-oven transformer, but recommends using a smaller one instead.) By connecting these to 8 pairs of banana-plug terminals in a metal box, you get a sort of variac; for example, if you want 42 volts, you can put in series the 2-VAC, the 8-VAC, and the 32-VAC winding with two jumper wires.

But there are other ways you can get 42 volts; for example, you can use the 32-VAC winding in series with the 16-VAC winding, then wire up the 4-VAC and 2-VAC windings *backwards* in series with that.

Balanced ternary gets you to 364 VAC in 1-volt increments in only 6 secondary windings

This suggests instead using balanced ternary. With a 1 VAC winding and a 3 VAC winding, you can get 1 VAC, 2 VAC (by wiring the two windings in series in opposition), 3 VAC, or 4 VAC (by wiring them in series). By adding 9 VAC, 27 VAC, 81 VAC, and 243 VAC windings, you can reach any voltage up to 364 VAC in 1-VAC steps, and this is the minimal number of windings you need to reach it.

Multitap secondary windings can deliver even more voltages with less terminals

That requires 12 banana-plug terminals, though. If you want to minimize the number of terminals rather than the number of windings, you might be able to do better with center-tapped windings.

For example, if you have one winding with three terminals whose two segments are 1 V and 2 V, you get 1, 2, and 3 VAC with three terminals; a second winding with three terminals whose two segments are 7 and 14 volts gives you all voltages from 1 to 24 volts AC; a third winding of 49 and 98 volts gives you all voltages from 1 to 171 VAC. That’s 9 terminals; a fourth center-tapped winding, with 343 and 686 volts in its segments, bringing us to 12 terminals as before, might then bring us from 1 to 1200 volts AC in one-volt steps. Or we could use a fourth 343-volt winding with no center-tap and get up to 514 volts with only 11 terminals rather than the 12 required by the balanced-ternary scheme to reach 364.

But what if we have *four* terminals on a winding? You could have, for example, a winding with a 1-VAC segment, a 3-VAC segment,

and a 2-VAC segment, in that order; this gives you 1, 2, 3, 4, 5, and 6 volts between its six different pairs of terminals. A second four-terminal winding with 13, 39, and 26 volts on its segments gets us 1–84 volts. A third winding with 169, 507, and 338 volts on its segments gets us 1–1098 volts, with the same 12 terminals that would give us 1–64 volts with César's binary scheme, 1–364 volts with the balanced-ternary scheme, or 1–1200 volts with the single-center-tapped scheme.

So it seems like the single-center-tapped scheme is optimal, at least to minimize the number of voltages you can get for a given number of terminals. The double-center-tapped scheme is very nearly as good, though, and it uses less jumper wires: you can reach any voltage up to 1098 volts with only two jumpers instead of the three you might need with the single center-tap.

One-volt precision is maybe more important when you're at 2 or 3 volts than when you're at 950 volts, so it would be nice if we could separate the voltage levels a bit more at higher voltages; unfortunately, the voltages on the various secondary windings do sum linearly, so you can't avoid this completely. But if you have one winding with segments of 1, 3, and 2 V and a second one with segments of 15 and 30 V, then you can do any one-volt voltage from 1–6 volts, 9–21 volts, 24–36 volts, and 39–51 volts, with just seven terminals and a single jumper.

```
subs = lambda items: set(sum(items[i:j])
                         for j in range(len(items)+1)
                         for i in range(j))
combos = lambda subses: {0} if not subses else set(a+b
    for c in subses[0] for a in [c, 0, -c] for b in combos(subses[1:]))
combos([subs([1]), subs([3]), subs([9]), subs([27]), subs([81]), subs([243])]
    ) == set(range(-364, 365))
combos([subs([1, 2]), subs([7, 14]), subs([49, 98]), subs([343, 686])]
    ) == set(range(-1200, 1201))
combos([subs([1, 3, 2]), subs([13, 39, 26]), subs([169, 507, 338])]
    ) == set(range(-1098, 1099))
```

I don't think we can do better by connecting triples of windings together in a Y configuration, like some BLDC motors, because the 1-3-2 setup already gives us six distinct voltages for the six distinct pairs of terminals, and they cover a contiguous range of integers.

A practical configuration

I think that, if you were going to do this in real life, the most practical configuration would use a single high-voltage winding with two terminals and two low-voltage windings with four terminals each, with a first winding of segments of ½, 1½, and 1 volt and a second winding of segments of 8, 24, and 16 volts. This gives you 0.5-volt resolution for 0–3 volts, 5–11 volts, and 13–19 volts, and 2-volt-or-better resolution up to 51 volts, all configured with a single jumper. This is not enough to kill you unless you are astonishingly fortunate.

The high-voltage winding might be 120 volts, which in combination with the low-voltage windings gives you voltages up to

171 volts, with an 18-volt gap between 51 and 69 volts; all of this for ten terminals and three secondary windings (plus the primary).

Ganging up two transformers

Now, if transistor *cores* are abundant and you just want to keep *windings* to a minimum, you could get a more favorable spread of high and low voltages by putting two separate transformers in the box, one fed from the power line with two to four terminals on its secondary brought out to the front panel, and a second transformer connected only to front-panel terminals, perhaps with two windings with three or four terminals each, either of which can be connected as a "primary" to the secondary of the first transformer. One reasonable winding configuration for the second transformer might be turns numbers of 1n-3n-2n on one winding and 10n-18n on the other. This affords 18 different stepups as low as 3:5 and as high as 1:28, including 2, 2½, 3, 5, 6, 7, 9, 10, 14, 18, and 28; and of course their reciprocals as stepdowns.

```
import fractions
' '.join(str(x) for x in sorted(f for n in subs([1, 3, 2])
                                for d in subs([10, 18])
                                for f in [fractions.Fraction(n, d),
                                          fractions.Fraction(d, n)]))
```

So if you had a center-tapped winding on the primary transformer with a 14-volt segment and a 134-volt segment, you could get 111 different voltages out of the combination of the two transformers, ranging from ½ VAC up to 4144 VAC. The full list is:

```
1/2 7/9 1 7/5 3/2 14/9 2 7/3 5/2 14/5 3 28/9 35/9 21/5 14/3 67/14
37/7 28/5 7 67/9 74/9 42/5 67/7 74/7 67/5 14 201/14 74/5 134/9 111/7
148/9 134/7 148/7 67/3 70/3 335/14 74/3 185/7 134/5 28 201/7 148/5
268/9 222/7 296/9 35 335/9 201/5 370/9 42 222/5 134/3 140/3 148/3
252/5 268/5 296/5 63 196/3 67 70 74 392/5 402/5 84 444/5 98 126 392/3
134 140 148 196 670/3 740/3 252 268 296 335 370 392 402 444 1340/3
2412/5 1480/3 2664/5 603 1876/3 666 670 2072/3 740 3752/5 804 4144/5
888 938 1036 1206 3752/3 1332 1340 4144/3 1480 1876 2072 2412 2664
3752 4144
```

```
' '.join(str(x) for x in sorted(set(f*v for n in subs([1, 3, 2])
                                        for d in subs([10, 18])
                                        for v in subs([14, 134])
                                        for f in [fractions.Fraction(n, d),
                                                  fractions.Fraction(d, n),
                                                  1])))
```

Or, as decimal approximations:

```
0.50 0.78 1.00 1.40 1.50 1.56 2.00 2.33 2.50 2.80 3.00 3.11 3.89
4.20 4.67 4.79 5.29 5.60 7.00 7.44 8.22 8.40 9.57 10.57 13.40
14.00 14.36 14.80 14.89 15.86 16.44 19.14 21.14 22.33 23.33 23.93
24.67 26.43 26.80 28.00 28.71 29.60 29.78 31.71 32.89 35.00 37.22
40.20 41.11 42.00 44.40 44.67 46.67 49.33 50.40 53.60 59.20 63.00
65.33 67.00 70.00 74.00 78.40 80.40 84.00 88.80 98.00 126.00
```

```
130.67 134.00 140.00 148.00 196.00 223.33 246.67 252.00 268.00
296.00 335.00 370.00 392.00 402.00 444.00 446.67 482.40 493.33
532.80 603.00 625.33 666.00 670.00 690.67 740.00 750.40 804.00
828.80 888.00 938.00 1036.00 1206.00 1250.67 1332.00 1340.00
1381.33 1480.00 1876.00 2072.00 2412.00 2664.00 3752.00 4144.00
```

```
' '.join('%.2f' % float(x)
        for x in sorted(set(f*v for n in subs([1, 3, 2])
                                for d in subs([10, 18])
                                for v in subs([14, 134])
                                for f in [fractions.Fraction(n, d),
                                          fractions.Fraction(d, n),
                                          1])))
```

Note that this still requires only 10 terminals: three on the main transformer's secondary winding, four on the auxiliary transformer's low-turns winding, and three on the auxiliary transformer's high-turns winding. Like the single-transformer "practical" configuration described above, it also requires four windings and at most two jumpers; it can produce fewer distinct voltages (only 111 instead of 153) but they are spaced out in a much more useful fashion: no more than 0.5 volts apart up to 3.1 volts, no more than 1 V apart up to 5.6 volts, no more than 2 V apart up to 10.6 volts, no more than 4 volts apart up to 46 volts, and so on.

It should be straightforward to come up with a better set of numbers for the windings, too, that give even more evenly spaced voltages, and perhaps at rounder numbers, although that aim seems to be in conflict with the aim of increasing the number of distinct voltages.

The above ignores the possibility of using the windings on the second transformer in autotransformer mode, so a larger number of configurations is actually possible; for example, you could hook up 14 volts to the 10n-turn winding segment and get 25.2 volts off the 18n-turn winding segment, a number which isn't in the above list. This relies on the primary transformer to provide galvanic isolation, which ought to be fine.

It's somewhat dubious whether you'd really want to use the higher voltages on such a gadget; they might need to be insulated to a degree that would make them impractical for the high currents encountered at low voltages.

Lightswitch reconfiguration

A lower-hassle way to get such flexibility, with only a *single* transformer, would be to mechanically switch the mains power between different primary windings. Two everyday single-pole double-throw lightswitches of the type commonly used to wire up hallway lights --- so that you can turn them on or off from either end of the hallway --- suffice to select among four of the six possibilities offered by a primary winding with two center taps, without any possibility of a short circuit. If the segments have a winding configuration 1n-1n-2n, then the four possibilities are 1n, 2n, 3n, and 4n; if instead they are 7n-1n-56n, then the four possibilities are 1n, 8n, 57n, and 64n.

This possibility of 1n, 8n, 57n, and 64n turns on the primary could be seen as a selectable multiplier of the secondary voltage: respectively 64, 8, 64/57 (about 1.12), and 1. Suppose that when the primary side is set to 8x, the medium voltage, the secondary side is like the low-voltage setup described above under "A practical configuration": a first winding of segments of ½, 1½, and 1 volts and a second winding of segments of 8, 24, and 16 volts. This gives you ½-volt resolution for 0–3 volts, 5–11 volts, and 13–19 volts, and 2-volt-or-better resolution up to 51 volts, all configured with a single jumper. Setting the primary side to 64/57 gets you roughly the same set of low voltages boosted by about 10%. But setting the primary side to 1x, the same secondary-side configurations give you 62.5-millivolt resolution from 0–375 mV, 625 mV–1.375 V, and 1.625–2.375 V, and ¼-volt-or-better resolution up to 6.375 volts.

Or, if you set the primary side to 64x — connecting only the middle segment of the primary winding — you get 4-volt resolution for 0–24 volts, 40–88 volts, and 104–152 volts, and 16-volt-or-better resolution up to 408 VAC. Ideally this 64x setting would be protected somehow so you didn't do it by accident. There's probably a reason they don't make power variacs with two sliders…

Since 63 millivolts to 408 volts is an unreasonably large range for a single apparatus — 100 watts at 408 volts is only 250 mA, while at 63 millivolts it would be **sixteen hundred amps** — maybe a better choice is to use a single four-terminal winding on the secondary side. It could be wired, say, 2–5–4, which can produce multipliers [2, 4, 5, 7, 9, 11], and windings on the primary side could be configured, say, 5n–2n–11n, providing divisors of 2n, 7n, 13n, and 18n, since 11n and 5n are inaccessible with the two-lightswitch configuration. This design is amusingly analogous to a trucker's 4×6 gearshift, except that truckers' gear ratios are a lot closer together.

If we set the lowest available voltage here to 1 VAC (2 on the secondary, 18n on the primary), then our 23 available voltages are 1.0, 1.38, 2.0, 2.5, 2.57, 2.77, 3.46, 3.5, 4.5, 4.85, 5.14, 5.5, 6.23, 6.43, 7.62, 9.0 (two ways), 11.57, 14.14, 18.0, 22.5, 31.5, 40.5, 49.5.

```
sorted([round(9*v/d, 2) for v in subs([2, 5, 4]) for d in [2, 7, 13, 18]])
```

This is an entirely reasonable set of voltages for a ghettobotics lab benchtop power supply, except that they're AC voltages. If you rectify these voltages and charge capacitors with them, they get higher by a factor of $2^{½}$: 1.41, 1.96, 2.83, 3.54, 3.64, 3.92, 4.9, 4.95, 6.36, 6.85, 7.27, 7.78, 8.81, 9.09, 10.77, 12.73, 12.73, 16.36, 20.0, 25.46, 31.82, 44.55, 57.28, 70.0.

This approach is also a lot more windings-efficient than the approach of varying only the secondary windings: it never uses less than 11% of the primary windings nor less than 18% of the secondary windings, so the transformer never needs to be more than about six times bigger than the minimal 50Hz transformer for whatever you're doing at the moment. By contrast, with windings of 1V, 3V, 9V, and 27V, the balanced ternary approach is using 2.5% of its secondary windings when it's outputting 1V. Normally the primary and secondary windings need to be about the same size because their cross-sectional areas per turn vary in nearly exact proportion to their

numbers of turns, so at 1 V it can only carry 1/40 of its maximum power.

What's the actual turns ratio n? If our input is 240VAC, it's about 26.67: say, 133 turns, 53 turns, and 293 turns in the three segments of the primary, if the secondary is actually wired with 2 turns, 5 turns, and 4 turns. If you're winding the transformers by hand, using an additional stepdown transformer (or two!) would be a great idea, just so you don't have to thread a wire through your transformer core over 900 times. This, though, suggests a return to the approach of the previous section, wherein each winding gives you an opportunity to reconfigure.

An 8-lightswitch reconfigurable design with two transformers but only two jacks

So, suppose we have a primary transformer with two center-taps on its primary hooked to the wall current through two SPDT switches, and the two center-taps on its secondary allow you to use two more SPDT switches to select one of four possible parts of the secondary, and those are connected to the primary of a second transformer via two *more* SPDT switches to select one of four possible parts of *its* primary, and on *its* output we have two *more* SPDT switches which hook up the output socket to it. No jumper wires and no possibility of shorting a winding with them. What does *that* look like? What kind of turns ratios can it give us?

I'm tired of designing, so I generated the random configuration ([25, 9, 32], [5, 2, 11], [25, 24, 28], [7, 2, 12]). That is, the first transformer has a primary winding with a 25-turn segment, an 9-turn segment, and a 32-turn segment, and a secondary winding with a 5-turn segment, a 2-turn segment, and an 11-turn segment; the second transformer has a 25-24-28 primary and a 7-2-12 secondary. (Maybe all the turns numbers are multiplied by some constant such as 1.5 or 2, since 2 turns might not be enough to couple well to the magnetic core.) What possibilities does this offer?

```
import random

def config(m):
    x = range(2, m)
    random.shuffle(x)
    x = sorted(x[:3])
    x[0], x[1] = x[1], x[0]
    return x

config(20), config(10), config(20), config(10)
```

I'm tired of calculating too, so I wrote code to calculate.

```
spdt = lambda (a, b, c): sorted([b, a+b, b+c, a+b+c])

ratios = lambda p, s: sorted(set(fractions.Fraction(n, d)
                                 for d in p for n in s))
' '.join(str(f) for f in ratios(spdt([25, 9, 32]), spdt([5, 2, 11])))
' '.join(str(f) for f in ratios(spdt([25, 24, 28]), spdt([7, 2, 12])))
```

This gives us the possible voltage ratios for the first transformer 1/33 2/41 1/17 7/66 7/41 13/66 7/34 2/9 3/11 13/41 13/34 18/41 9/17 7/9 13/9 2 and for the second transformer 2/77 1/26 2/49 1/12 9/77 9/52 2/11 9/49 7/26 3/11 2/7 3/8 21/52 3/7 7/12 7/8. These do indeed result in 256 different voltages, which range from about 0.2 volts up to 420 volts:

```
rs = sorted(set(240*t1*t2 for t1 in ratios(spdt([25, 9, 32]),
                                           spdt([5, 2, 11]))
                          for t2 in ratios(spdt([25, 24, 28]),
                                           spdt([7, 2, 12]))))
min(rs), max(rs), len(rs)
```

Specifically, the output voltages are 0.189 0.280 0.297 0.304 0.367 0.450 0.478 0.543 0.576 0.606 0.661 0.850 0.976 0.979 1.039 1.064 1.176 1.228 1.259 1.283 1.322 1.336 1.368 1.385 1.576 1.650 1.672 1.700 1.818 1.900 1.929 1.958 1.977 1.983 2.017 2.026 2.051 2.078 2.121 2.129 2.150 2.177 2.383 2.443 2.517 2.567 2.593 2.672 2.727 2.737 2.927 2.937 2.975 3.106 3.117 3.152 3.193 3.300 3.345 3.415 3.529 3.745 3.801 3.850 3.939 4.034 4.053 4.118 4.242 4.301 4.390 4.406 4.444 4.628 4.675 4.728 4.789 4.848 4.887 5.017 5.186 5.294 5.455 5.525 5.701 5.775 6.050 6.234 6.341 6.364 6.829 6.853 6.942 7.092 7.179 7.273 7.450 7.526 7.619 7.647 7.651 8.182 8.235 8.552 8.595 8.683 8.780 8.895 8.984 9.004 9.076 9.231 9.545 9.697 9.796 10.244 10.280 10.588 10.726 10.909 11.032 11.175 11.329 11.707 11.901 12.022 12.315 12.353 12.468 12.727 12.893 13.171 13.303 13.333 13.476 13.506 13.836 13.977 14.118 14.150 14.359 14.545 14.848 14.851 15.238 15.366 15.556 15.882 16.548 16.684 16.855 17.561 17.622 17.727 17.851 18.236 18.462 18.529 18.701 19.091 19.157 19.353 19.592 19.955 20.000 20.260 20.488 20.754 21.176 21.538 21.742 21.818 21.991 22.273 22.857 23.102 23.337 23.902 24.545 24.706 25.027 26.218 26.434 27.576 28.052 28.368 28.537 28.736 28.824 28.889 30.105 30.732 31.111 32.308 32.613 33.939 34.208 34.286 34.412 34.652 35.854 36.303 37.059 38.182 39.328 39.512 40.000 40.519 41.364 42.552 43.235 44.390 45.157 46.667 47.647 50.256 50.909 51.312 53.333 53.529 54.454 56.104 57.273 60.000 61.463 63.030 63.673 66.585 70.000 74.118 75.385 80.000 80.294 83.077 87.273 88.163 92.195 93.333 94.545 99.048 108.889 111.176 129.231 130.000 130.909 137.143 140.000 148.571 163.333 180.000 193.846 202.222 205.714 280.000 303.333 420.000.

```
' '.join('%.3f' % float(f) for f in rs)
```

This randomly generated configuration is maybe not a super great design but it's in some sense reasonable. Half the values are below 12 volts, there are 256 distinct values, the values are mostly only a couple percent apart in the middle of the range, and the range covers over three orders of magnitude. Over most of the range the design has considerably more precision in the turns ratio than the margin of error on the mains voltage.

This is kind of overkill, although the transformers are much more manageable. Maybe a single SPDT per winding with a single center tap on each winding and two center taps on the final output would be adequate: three lightswitches to "select a range" and then four output

terminals to give you six voltages simultaneously, 48 settings in all.

Topics

Tentative outline of a body of knowledge

Kragen Javier Sitaker, 02020-06-06 (updated 02020-10-28)
(10 minutes)

A possible ambition for Derctuo is to include all the background information needed to understand it, if I can find freely-licensed sources. So, for example, Pandemic Collapse (p. 69) talks about geography (the US, Tenochtitlán, Cambodia), historical events (the Vietnam War, the 1918 flu, the Bronze Age Collapse), economic concepts (unemployment, insurance, banks), and other institutions (the US DoD, the Mormon church, major corporations). Solar furnace CPC (p. 65) talks about physical properties of common materials, the Stefan–Boltzmann law, manufacturing processes of ceramics, thermodynamics, units of measurement, basic optics, and the structure of the solar system. CCN Streams (p. 52) talks about networked systems architecture, hashing, SHA-256, TCP/IP, disks, telephone networks, and all kinds of programming stuff.

What is the body of knowledge that would be needed to make sense of all this stuff? Consider the Stefan–Boltzmann law. To make any sense of the statement $j = \sigma T^4$ you need to know algebraic notation and what energy and temperature are, including the concept of absolute temperature. And you need to understand how solid objects have surface areas.

Geographic and historical knowledge in particular is sort of endless. Tenochtitlán is Mexico City today, with 8.8 million people, 0.11% of the world's population; Mexico City's Wikipedia page is 213kB, 33000 words; the destruction of Tenochtitlán (what is referenced in Pandemic Collapse) is mentioned briefly after 9% of the page. If you divided the world into, say, 2048 regions of equal population (4 million or so), and included 4096 words or so on each of these regions, you'd probably cover most of the geographic facts of importance comparable to the ruin of Tenochtitlán, in about 8.3 million words, about 30,000 pages; you could read it all, once, in three to six months.

Vital Articles

Wikipedia's "Vital Articles" constitutes an attempt to codify such a general-purpose body of knowledge. There are ten Level 1 Vital Articles, including "Human History" (21000 words, 137kB, mentions Mexico and the Aztecs, and has a couple of sentences on the European conquest of the Americas); 100 Level 2 Vital Articles, including 10 articles on history (the "early modern period" article has a couple of sentences on the European conquest out of 18000 words and 120kB and mentions the Aztecs, and so does "civilization") and 11 on geography (the "North America" article's 18000 words in 123kB does explain, "The Mayan culture was still present in southern Mexico and Guatemala when the Spanish conquistadors arrived, but political dominance in the area had shifted to the Aztec Empire, whose capital city Tenochtitlan was located further north in the

Valley of Mexico. The Aztecs were conquered in 1521 by Hernán Cortés."); and 999 Level 3 Vital Articles, including 80 on history and 99 on geography.

How about the killing fields of Cambodia under the Khmer Rouge, also mentioned in the same note? Among the Level 2 Vital Articles we find "Late Modern Period" (19000 words, 123kB) which mentions the Cambodian genocide, but no more; and "Asia" (15000 words, 104kB) which mentions "the Cambodian Killing Fields", but no more. We don't find enough detail to understand the allusions in Pandemic collapse (p. 69) until Level 3, which sketches the history of the Khmer Rouge in Cambodia in its articles "Vietnam", "Cold War" (36000 words, 233kB) including multiple paragraphs and a photo of a shelf full of skulls, "Mao Zedong", "Theravada", "Dictatorship", and especially "Genocide" (17000 words, 109kB).

So we can infer that probably, at least when it comes to understanding my historical references, having read all of Wikipedia's Level 3 Vital Articles are probably sufficient. This is not true for scientific knowledge; "Temperature", "Fire", "Electric light", and "Electromagnetic radiation" mention black body radiation briefly but do not mention the Stefan–Boltzmann law.

Unfortunately the Level 3 Vital Articles are some 20 million words and would blow out the 20-megabyte download budget for Derctuo, even without any pictures. The thought above of having about 4096 words for every 4 million people would be more than adequate for Cambodia, though, since in the 16384 words on Cambodia, we could surely find space to mention the Khmer Rouge.

Reading Level 3 might take a year at a reasonable level of reading speed, a bit over 1000 hours if you read it like a novel.

Possible plethoras of sources

Possible sources include MIT OpenCourseware, Wikipedia, Wikibooks, cnx.org (before it shuts down), OpenStreetMap, Project Gutenberg, the Internet Archive etexts collections, and for recent things, PLoS and arXiv.org. Boundless used to have some open-content textbooks but they seem to have mostly been lost, though fragments like their definition of limits survive in part. OERCommons has a search engine over thousands of freely licensed educational resources, of which nearly a thousand few are textbooks, such as Jim Hefferon's linear algebra book (CC-BY-SA, 7.5MB, 507pp.). (See also the section on "particular textbooks" below about Hefferon's work.) They also link to OpenStax (which I'd forgotten about), Delft OCW, CMU OLI, and another dozen or so similar initiatives.

GWU has a guide to open textbooks which links to most of the above.

Wikipedia has a list of notable CC works, including Connexions (which I guess is cnx.org), Khan Academy (cc-by-nc-sa), OpenLearn, OCW, something called "The Saylor Foundation", WikiEducator, 375000 CC0 artworks from the Metropolitan Museum of Art, deviantART, Flickr, Open Game Art, Openclipart, etc.

Many public-domain books, including some nonfiction, are in

Project Gutenberg and Wikisource, as well as the Internet Archive's books collection. Everything up to 1924 is PD in the US now, including *Rhapsody in Blue*. Parker Higgins collated many striking 1923 works in a zine last year, though I think a more striking work still is Kahlil Gibran's *The Prophet*. Also, perhaps, the Russells' *The Prospects of Industrial Civilization*. The Hathi Trust catalogues 53940 works published in 1923, of which 33105 are books.

A dismal assessment of OERCommons

The OERCommons textbooks mentioned earlier include 17 history textbooks, but most are too specific to include either of the events I was using as test points above. World Civilizations I (CC-BY) was the only one that seemed broad enough to mention Cambodia, but unfortunately has been lost. Western Civilization: A Concise History, Volume 3 (CC-BY-NC, 105k words, 10MB as .odt, 274 pp.) starts with Napoleon, too late to cover Cortés, but its volume 2 (CC-BY-NC, 87k words, 229 pp.) does devote a few paragraphs to the events.

Particular textbooks to check out

Jim Hefferon's *Linear Algebra*, *Theory of Computation*, and *Introduction to Proofs* are cc-by-sa 3.0 disjunction GFDL, with LaTeX source. He says the linear algebra text is "a popular text". I haven't reviewed the books yet, but some people seem to like them, though others tar them as unrigorous. And they come with exercise solutions and video lectures.

SICP is under cc-by-sa 4.0. I think *Structure and Interpretation of Classical Mechanics* is under cc-nc-by-sa 4.0. It's using MathJax.

Mathematics for Computer Science is a cc-by-sa 987-page PDF covering things like graphs, satisfiability, and linear recurrences.

I am greatly enjoying Reuleaux (p. 444)'s presentation of kinematics, which is in the public domain due to its age. However, the idea of reducing it to files of a manageable size seems daunting.

I really liked MacKay's *[Sustainable Energy Without the Hot Air]*. Disappointingly, his book on information theory is not available under a free license, and neither is *Without the Hot Air/SEWTHA* as it turns out.

PLOS ONE has a systematic reviews category, but most of the 1507 reviews therein are pretty narrow: "Healthcare-associated infection and its determinants in Ethiopia: A systematic review and meta-analysis" and the like, although "Fecal microbiota transplantation in inflammatory bowel disease patients: A systematic review and meta-analysis" sounds pretty interesting.

On the topic of formal logic, Sean Palmer recommends forall x, Tree Proof Generator (usable online at https://www.umsu.de/trees/), and the whole Metamath website, which is in the public domain, including things like the proof that √2 is irrational.

Gwern licensed his entire site under CC0. It mostly discusses IQ, epistemology, pharmacology, IQ, deep learning, other aspects of AI, statistics, genetics, IQ, politics, psychology, biology, programming, economics, and IQ, but occasionally strays from that focus. Uses

Mathjax. He explains his motivation:

> The goal of these pages is not to be a model of concision, maximizing entertainment value per word, or to preach to a choir by elegantly repeating a conclusion. Rather, I am attempting to explain things to my future self, who is intelligent and interested, but has forgotten. What I am doing is explaining why I decided what I did to myself and noting down everything I found interesting about it for future reference. I hope my other readers, whomever they may be, might find the topic as interesting as I found it, and the essay useful or at least entertaining–but the intended audience is my future self.

The source code of his pages (in a Pandoc-implemented language derived from Markdown) is accessible by appending .page to the URL. He says the whole thing is kept in Git, but I don't know where.

Topics

- Derctuo (p. 820) (9 notes)
- Archival (p. 853) (5 notes)
- Wikipedia
- Textbooks
- Public domain
- Creative commons

Ghettobotics soldering iron

Kragen Javier Sitaker, 02020-06-17 (4 minutes)

Espacio de César posted a video demonstrating how to make a usable 40-watt soldering gun out of a heavy mains transformer (rewound with a low-voltage secondary) and some heavy steel wire, saying he was looking for about ½ volt on the secondary. This seems like a very reasonable strategy to me but I can't salvage transformers — invariably they are already recycled before I encounter a discarded electronic item. Probably even something like an ATX power supply would be easier to find.

½V and 40W is about 80 amps, though, which is a bit more than ATX power supplies can usually provide.

That also implies about 0.006Ω. Is that about the right resistance? Iron's resistivity at room temperature is about 100 nΩm; guessing that César's heating element is about 100 mm long and 1 mm², we get 0.01Ω, so yeah, that's about right. 1010 carbon steel is about 143 nΩm, while stainless is several times higher at some 700 nΩm. Nichrome would be much better, at 1100 nΩm, and I'll probably find some sooner or later, since people constantly throw out broken hair dryers and space heaters.

We could get the same power out of a thinner wire at a higher voltage and lower current, but at more risk of burning the wire out. The wire fusing current estimates from Powerstream that I used in file `balcony-battery` in Dercuano suggest that 86 A is *already* enough to melt 11-gauge iron wire (2.3 mm), 43 A is enough to melt 15-gauge iron wire (1.5 mm), 21 A is enough to melt 19-gauge iron wire (0.9 mm), and 10.7 A is enough to melt 23-gauge iron wire (0.57 mm). So really César is already past the edge of safety and will melt his soldering tip if he holds the trigger down long enough.

Can you do it with simple electronics instead of a transformer?

You can't just PWM the AC line current through a 6-milliohm heating element; you'll trip the house's circuit breaker, and even if you don't, you're dropping the line voltage to zero temporarily, and other nearby appliances won't like that. But you ought to be able to PWM it into a hefty inductor with a hefty freewheel diode or ten, at least if they have enough ballast to prevent thermal runaway. Very crudely guessing, if you have a duty cycle of 0.01% or more and a PWM frequency of 10kHz, then your inductor just needs to prevent the current from rising to too much more than 80 amps in 10 ns, or 8 billion amps per second, at less than 340 V (the peak voltage). That only requires 43 nanohenries, which you might get without asking for it. But it also requires subnanosecond switching times for that current. Also, you need an input capacitor bank that can handle 80 amps of ripple current, which is doable but nontrivial.

You probably *could* PWM the AC line current into a high-frequency stepdown transformer, which could handle the 40 watts or whatever in a much smaller core. This is basically a flyback supply I think, just with a stepdown instead of a stepup? I don't know, I have to think about this stuff later.

Topics

An outline of the design process leading up to the Veskeno virtual machine

Kragen Javier Sitaker, 02020-06-17 (updated 02020-07-10)
(88 minutes)

¿Ves que no?

The primary goal of Derctuo is to present some calculations and computational simulations in a reproducible fashion, so that it is possible for other people to build on them. Unfortunately, and quite surprisingly, no suitable medium for such things currently exists — except in the limited sense that bytes and computers are potentially such a medium. But a raw sequence of bytes is meaningless without some kind of interpretation, a "file format", and as far as I can tell, no suitable file format currently exists.

"Veskeno" is the name I have adopted for such a file format, which unfortunately requires the development of a new virtual machine for reproducible computations. The reasons for this require some explanation.

The determinism of mathematics

Consider the polynomial $x^4 - x^3 - 5x^2 - x - 6$. We can reasonably make assertions about it; for example:

- "As you can see, this polynomial has two real and two imaginary roots."
- "At $x = 0$, this polynomial's derivative is -1, and at $x = 1$, the derivative is -5."
- "For real x, this polynomial is bounded below by a constant, but not bounded above."

Moreover, you can compute that, for example, one of the polynomial's zeroes is at $x = -3$, while another is at $x = 2$.

Or are they? Either way, you can calculate what the zeroes are, although it may not be easy — it's a matter of objective truth or falsity. If I've made an error, you can find it, and if I'm correct, you can verify that. And you can be sure that anyone else who does the calculations will get the same answer — regardless of their cultural background, ethical beliefs, or latitude, regardless of whether they're in 02020 CE, 02120 CE, or 12020 CE. Indeed, any rational being in any possible universe would get the same results. Unless they failed to understand or made a mistake. (Did I make a mistake above?)

As David Hume says:

> Algebra and arithmetic [are] the only sciences in which we can carry on a chain of reasoning to any degree of intricacy, and yet preserve a perfect exactness and certainty. We are possessed of a precise standard, by which we can judge of the equality and proportion of numbers; and according as they correspond or not to that standard, we determine their relations, without any possibility of error. When two numbers are so combined, as that the one has always a unit answering to every unit of the other, we pronounce them equal.

There are a variety of things for which we can make such objective assertions: which side won a chess game, given all the moves, for example, or that the word "fire" occurs 559 times in the King James Version of the Bible, at least if we can agree on which version that is, and what counts as an occurrence of the word — I omitted occurrences of "fiery", but counted words containing "fire", such as "firepans" and "firebrands".

The objective of Veskeno: make software, specifically Derctuo, run as reproducibly as other mathematics

> [C]omputer science is about formalizing imperative knowledge. The essence of programming is about imperative knowledge. It's about how to do things. … But it's no different than what we say in the SICP lectures: Mathematics is about how you think about what's true, following from various axioms. Computing is how you think about how to do things.
>
> — Hal Abelson, 2011

Since at least Church, Turing, and Gödel, we have a rigorous mathematical formalization of the notion of an *algorithm*, which is how we have managed to build digital computers that can be programmed to execute any algorithm in the first place. So we know that we can, in theory, come to the same kind of consensus about the behavior of an algorithm — in theory any algorithm whatever can be executed on any computer in the universe, on the same input data, and compute exactly the same results. And, again in theory, it does not matter whether the computation happens in 02020 CE or 02184 CE; the results will be exactly the same, precisely the same sequence of bits. In theory, programs are incapable of nondeterminism, unless the computer malfunctions.†

In practice, however, we have a very different situation, one prone to what Konrad Hinsen calls "software collapse", colloquially known as "bitrot": software that works perfectly on one machine or at one time, but fails to run correctly or even to compile on another machine or at another time; or it may run but be unusable in practice for one or another reason. Occasionally this happens because of changes in the universe of inputs and outputs — many IBM PC games ran too fast to be playable on the IBM PC AT, for example, and a user interface designed for mice may require too much precision to be usable in practice on a multitouch hand computer — but much more often software collapse happens because software or hardware dependencies changed their behavior, so the same program computes different results from the same inputs. Often a large body of tacit knowledge, concerning what changes have happened, must be drawn upon to repair software collapse; if the maintainers of the codebase are dead or uninterested, repairing it may be infeasible.

Veskeno's objective is to put the theory into practice, so that the algorithmic results in Derctuo are as reproducible as algebraic results. This way, it is hoped, the written record of algorithmic knowledge can engage the kind of ratchet of progress that has propelled mathematics and the natural sciences forward, so that each generation of researchers can build on the results of the previous generation, rather than — as normally happens with software — reinventing those

results from scratch. And it need not be dependent on the maintenance of a living tradition, since Veskeno can be reimplemented from its specification.

We want to have a high degree of assurance that, if a computation has occurred under Veskeno and the program and input data are available, we can reproduce the same computational results with a new implementation of Veskeno, although perhaps more slowly, or, with luck, more quickly; we want to minimize the chance that a bug in either the new or especially the old implementation breaks this reproducibility.

We adopt the following priorities in order to achieve this:

- The Veskeno specification should be sufficiently strict and detailed that, given any Veskeno program and its input data, any two correct implementations of Veskeno should produce bitwise identical results, unless one or both of them unpredictably fails. (See below about predictable and unpredictable failure.)
- The Veskeno specification should be sufficiently simple that a programmer should be able to implement it in an afternoon, given only the spec — without having access to running implementations to test against. Moreover, the implementation should be more likely correct, as defined above, than incorrect, once it passes all the tests in the spec — even if implemented on hardware that would be extremely unusual in 02020 CE, such as a decimal or ternary computer.
- A straightforward one-afternoon Veskeno implementation should be efficient and full-featured enough to run practical computations — for example, to run 1980s video-games at playable speeds, on mainstream 02020 CE personal computing hardware such as a Samsung Galaxy A10.
- It should be practical to generate working code for Veskeno without unreasonable space overheads, compilation-time costs, or headaches. However, since the objective is to spend hundreds or thousands of hours writing Derctuo so that someone can write a Veskeno virtual machine on which to run it in six hours or so, it's worth trading off 100 hours of effort programming *for* Veskeno to save even a single hour of effort writing a Veskeno implementation.
- The damned thing needs to get done in a month or two and have working software on it at that point.

A consequence of priorities #1 and #2 is that it should *usually* be impossible for a malicious attacker, upon examining two simple implementations of Veskeno that pass the tests in the spec, to construct a Veskeno program and input dataset that runs successfully on both implementations but produces different results.

This set of priorities leads to a very unusual set of design tradeoffs, one so alien to modern mainstream virtual machine design that the comment was heard, "I feel like this is designing a weapon or something."

There are varying levels of abstraction at which we could such reproducibility could be guaranteed; Veskeno takes the simplest approach, of prescribing reproducibility at the level of individual CPU instructions, which compose into reproducible macroscopic computations.

† Conventionally these results are stated for batch-mode algorithms which run for some finite period of time and then halt with a result, but it's straightforward to extend them to interactive processes — the batch-mode algorithm takes as input a previous state and an input event (which may be simply that there is no input of interest to report) and eventually produces a new state and perhaps some output. (Extending this statement to multithreaded programs and interrupt-driven I/O is less straightforward but in principle possible by treating these new sources of nondeterminism as more kinds of input events.)

A note about hardware performance

A Galaxy A10 (30 million sold in 02019) has 2 GiB of RAM and eight Cortex-A cores running at 1.35 to 1.6 GHz, capable in all of perhaps 20 billion 64-bit multiply-accumulate operations per second, plus a Mali-G71 MP2 GPU, which I think is about 50 gigaflops on two cores. A 1980s video-game might have 1 MiB of RAM and execute a million 16-bit multiply-accumulates per second.

So the *throughput* performance overhead budget here is about a factor of 2048 in space and a factor of some 131072 in time, though of course greater speed and less overhead would be desirable, since it would make much more elaborate computations reproducible. Typical straightforward low-level virtual machines can achieve time overhead factors of 3–20 and space overhead of 1.1–4, but we don't have to come close to that; moreover Veskeno is serial, imposing another factor of 16–64 of overhead on the throughput (p. 20) unless some kind of parallelism is possible. This leaves us some 128× of performance headroom.

A typical 1980s video-game might run on a 6502 like the 1.79MHz one in a Nintendo; a multiply routine for the 6502 takes 130 CPU clock cycles to multiply 8 bits by 8 bits and get a 16-bit result, while a version using a table of squares takes 79–83 cycles. At the 6502's max of 3 MHz this might give us 38000 8-bit multiplies per second, or only about 22000 on the Nintendo, working out to about 5000 16-bit multiplies per second; typically, though, 6502-based video-games paired the CPU with sprite hardware to do compositing of video-game characters onto a background, thus reducing the load on the CPU. By the end of the 1980s, though, some video-games ran on CPUs that were some 256 times faster, obviating the need for sprite hardware.

On the other hand, for faithfully reproducing the "feel" of human interactions with existing computer systems, simulating the *analog* behavior of computer hardware often demands significant computational work. XXX at masswerk.at has reproduced "Spacewar!", perhaps the first video-game done on a computer; he reports that the most difficult and time-consuming part was accurately reproducing the color change and exponential decay behavior of the PDP-1's display. Accurately simulating analog video artifacts like chroma subsampling, NTSC artifact colors, ghosting, blur, ringing, noise, and pincushioning, as the Apple2 XScreenSaver module does, can use an unboundedly large amount of digital computation.

Predictable and unpredictable failure

Above, I said, "any two correct implementations of Veskeno should produce bitwise identical results, unless one or both of them fails." Why is failure an option, and what kinds of failure handling can we have?

The simplest and most unavoidable kind of unpredictable failure is that an implementation, although correct, runs too slowly to be worth waiting until it finishes. Perhaps a given computation requires an hour of CPU time on an efficient implementation of Veskeno; an inefficient implementation might be 1000 times slower and run on a CPU that is 8 times slower, thus requiring 8000 hours, about 11 months. Such a computation is almost certain to be aborted before completion unless its results are of great interest and using the computer for other tasks is of little interest.

Dynamic memory allocation failure is another kind of unpredictable failure which, although it is not unavoidable, may be preferable to the cost of avoiding it. It is straightforward to write a program such that it can handle any amount of data that would fit into virtual memory — 4 gibibytes in the case of a 32-bit machine — while being able to handle smaller amounts of data, such as 100 kibibytes, in much smaller amounts of memory. We could avoid the possibility of runtime dynamic memory allocation failure by preallocating 4 gibibytes, so that the program will entirely fail to run on machines with only, for example, a gibibyte of RAM, even if given only 100 kibibytes of input. This would prevent it from running on, for example, the Samsung Galaxy A10 mentioned above, since it has only 2 gibibytes. (Kernel memory overcommit would have to be turned off to achieve this under Linux, since otherwise the Veskeno virtual machine process can be OOM-killed at any time.)

In many environments, it is very difficult to ensure that dynamic memory allocation failures cannot happen during execution, because many basic operations of the language can invoke dynamic allocation. In CPython, for example, even integer arithmetic can invoke dynamic memory allocation, and it is common even for languages like C to dynamically allocate function activation records on a stack, although perhaps this can be avoided during execution of Veskeno. Attempting to outlaw Veskeno implementations in such environments would be futile and probably counterproductive. Also, Veskeno itself is probably such an environment: a Veskeno virtual machine interpreter written in Veskeno can be useful for many things, but unavoidably will have less memory space available for the program it interprets than it has itself.

However, for some applications of Veskeno — not those in Derctuo — it would be desirable to ensure that no such unpredictable failures will arise, so that a Veskeno-implemented algorithm can be used to, for example, safely control an antilock braking system, a jet engine, a milling machine, or a self-balancing scooter. Typically in these cases a worst-case execution time is also demanded. For these cases, we would need to preallocate all the needed resources.

A third kind of possible failure arises from correctness checks on operations such as arithmetic, memory access, and I/O. Conventionally, for example, dividing by zero or dereferencing a null

pointer will raise an exception that can terminate a program or reset a computer. On some systems, arithmetic overflows may also raise such exceptions, the Ariane 5 maiden flight being one notorious example. For debugging, these exceptions are highly desirable, since they often point quite directly to the problem in the program, while incorrect results might easily be overlooked. They are different from the above kinds of failures, though, because they are not unpredictable: running the same program on the same input data will always produce the same failure — although in many systems the exception happens some time *after* the actual failure.

What would happen if a Veskeno program had the option to handle unpredictable failures? For example, if dynamic memory allocation sometimes reported failure to the program, or invoked an exception handler in the program. Then some executions of the program on the same input data would see a reported failure, while others would get success, so their executions would diverge — Veskeno could no longer guarantee that the results were equivalent. Even if the results were marked as "error output" rather than "algorithm results", since a failure had happened during the run, people would start relying on that error output.

So, because unpredictable failure is not deterministic (in terms of the supposed inputs) recovery from unpredictable failure must be impossible. This reasoning does not apply to predictable failures, and so it is reasonable to include predictable-failure cases in the Veskeno test suite.

However, even predictable failure cases pose some real difficulty in reproducibility, because they tend to be very poorly tested. Ordinary computations, outside the test suite, will not normally depend on the behavior of failure cases, so it is easy for a case to slip through where the virtual machine is supposed to detect a certain failure, but it fails to do so — a failure to fail, you might say. Then users will write programs that depend on the virtual machine's failure in that case, probably without knowing it, and their behavior will not be reproducible on other implementation of Veskeno.

A binary, rather than textual, file format

It's reasonable to consider, for example, core Lisp as the canonical representation for algorithms, by which is meant the usual definitions of S-expressions, CAR, CDR, CONS, QUOTE, NULL, ATOM, EQUAL, LAMBDA, some kind of conditional such as COND or IF, and some kind of recursive construct such as LABELS, LETREC, or global DEFUN; and, indeed, these constructs have a perfectly well defined deterministic semantics sufficient to express any computable function. Moreover, in any modern high-level language with garbage collection, you can write an interpreter for it in 30 to 120 lines of code, including the reader and the printer.

However, when we turn to thinking of testing and failures, many subtle considerations appear. LAMBDA and LABELS involve the introduction of symbols; what is the maximum acceptable length of these symbols? How many characters of them are significant — all of them? Are characters counted as bytes (in UTF-8?), as Unicode code points, or as UTF-16 code units? Which characters are allowed? Is

comparison case-insensitive, or, if case-sensitive, is it done in, for example, Normalization Form D? Is there a maximum nesting depth to lists, and what is it? How about a maximum length? Is the symbol NIL, or some other symbol, EQUAL to an empty list, or treated as falsehood in conditionals? Is the ASCII tab character treated as whitespace? How about vertical tab (^K)? How does EQUAL handle lists — does it treat them as always inequivalent, almost like EQ, or does it compare their contents, and if so does it have a recursion limit? Must the input file end with a linefeed character? What will the parser do if an extra right parenthesis is added to the end of the file? How about an extra left parenthesis? If an unused argument is specified as a nonterminating computation, will the computation succeed or not — that is, is evaluation lazy or eager? Does the answer depend on circumstances?

If mutable state is added — as it was immediately to Lisp, historically speaking — additional questions become relevant. What is the order of evaluation of arguments?

These problems are amplified by the fact that the answers may be dependent on the invocation context in a poorly specified way. As an example, CPython's default recursion limit is 1000 stack levels, which may give rise to a nesting limit of 333, 500, or 1000 for lists in a straightforwardly-written recursive-descent parser — but if that parser is invoked from a context already nested ten stack levels deep, these limits instead become 330, 495, or 990. CPython is unusual in that it handles stack overflow explicitly by raising an exception; most current and past language environments instead produce unpredictable incorrect results or crash at the operating-system level.

Since most of the Lisp primitives draw their arguments from potentially infinite domains, such as lists and symbols, which are at least exponentially large, running exhaustive tests for them is out of the question.

The depth and richness of these likely sources of implementation bugs would seem to make the following scenario almost inevitable: Alice implements Veskeno and builds and tests a program as a Veskeno virtual machine image in her implementation. Unbeknownst to Alice, her program depends on symbols with equal Normalization Form D being treated as EQUAL. Bob, perhaps three centuries later, implements Veskeno and attempts to run Alice's virtual machine image. It produces different results than it did for Alice. Bob concludes that Alice (RIP) was a superstitious fool whose reported results cannot be trusted, or perhaps a fraud.

This is precisely the scenario Veskeno is intended to prevent. Veskeno priority #2 says:

> ... the implementation should be more likely correct, as defined above, than incorrect, once it passes all the tests in the spec ...

Some of these problems are specific to Lisp and would not be present with, for example, a textual assembly-language format, but others are generic problems of most or all textual formats. And the advantages of textual formats seem to primarily redound to the benefit of the person writing a file in them, not to the implementor of a complete, correct interpreter of the file format. As the priorities say, "it's worth trading off 100 hours of effort programming *for* Veskeno

to save even a single hour of effort writing a Veskeno implementation." Consequently, a binary file format seems far more likely to be able to achieve Veskeno's aims.

An untyped 32-bit register machine with mod-2^{32} wraparound

The Veskeno virtual machine has 16 CPU registers and a RAM array; programs using a stack store the stack in the RAM. To ease compiling existing C code for Veskeno, the registers are 32 bits, despite the hassles that entails in languages like Java or on 16-bit hardware; it poses no difficulty for Veskeno implementations in languages like C.

The only arithmetic operations it offers are addition and subtraction, which behave mod 2^{32} as you would expect.

One great drawback of 32-bit arithmetic is that its input space is of size 2^{64}; as a result, exhaustive testing of an arithmetic operation would take half a million CPU years at Veskeno's 1-MIPS performance target. Most programmers today cannot spend half a million CPU years on an afternoon project because they do not have hundreds of millions of CPUs available, nor even hundreds of CPUs; it is plausible that this parlous situation of poverty will continue for some time.

The fibterp spike

As a simple experiment to get a handle on software complexity and interpretive slowdown, I hacked together the following minimal simulator for such a machine, together with a dumb Fibonacci program for it; this took 96 minutes and 119 lines of C, 21 of which are the dumb Fibonacci program in a sort of assembly language. This virtual machine has 11 instructions and word-addressed memory, but I think Veskeno itself will have more like 16 instructions and byte-addressed memory.

```
/* XIS: simple little RISCy bytecode interpreter as a sort of quick spike
 * to see how fast or slow it goes.  Answer: about 20× slower than
 * GCC on the same machine.
 */

#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>

typedef uint32_t u32;

typedef struct {
  u32 reg[16];
  u32 *mem;
} machine;


enum opcode { insn_push = 0x21, insn_pop, insn_lit16, insn_low16, insn_add, insn_
osub,
```

```
       insn_jl, insn_halt, insn_call, insn_ret, insn_mov };

static inline int
src_reg(u32 insn)
{
  return (insn >> 20) & 0xf;
}

static inline int
dst_reg(u32 insn)
{
  return (insn >> 16) & 0xf;
}

#define IF break; case
#define ELSE break; default
#define rSP 15
#define SP reg[rSP]
#define rPC 14
#define PC reg[rPC]

int interpret(machine *m, int a, int b, int c, int d)
{
  m->reg[0] = a;
  m->reg[1] = b;
  m->reg[2] = c;
  m->reg[3] = d;
  for (;;) {
    u32 dest, insn = m->mem[m->PC++];
    enum opcode op = (insn & 0xff000000u) >> 24;
    switch(op) {
    IF insn_push:
      m->SP--;
      m->mem[m->SP] = m->reg[src_reg(insn)];
    IF insn_pop:
      m->reg[dst_reg(insn)] = m->mem[m->SP];
      m->SP++;
    IF insn_lit16:

      m->reg[dst_reg(insn)] = (insn & 0xffff) | (insn & 0x8000 ? 0xffff0000u : 0)o
o;
    IF insn_low16:
      m->reg[dst_reg(insn)] <<= 16;
      m->reg[dst_reg(insn)] |= insn & 0xffff;
      abort();  // untested code
    IF insn_add:
      m->reg[dst_reg(insn)] += m->reg[src_reg(insn)];
    IF insn_sub:
      m->reg[dst_reg(insn)] -= m->reg[src_reg(insn)];
    IF insn_jl:
      if (m->reg[src_reg(insn)] & (1 << 31)) {
        m->PC += (insn & 0xffff) | (insn & 0x8000 ? 0xffff0000u : 0);
      }
    IF insn_halt:
      return m->reg[0];
```

```
    IF insn_call:
      dest = m->PC + ((insn & 0xffff) | (insn & 0x8000 ? 0xffff0000 : 0));
      m->SP--;
      m->mem[m->SP] = m->PC;
      m->PC = dest;
    IF insn_ret:
      m->PC = m->mem[m->SP];
      m->SP++;
    IF insn_mov:
      m->reg[dst_reg(insn)] = m->reg[src_reg(insn)];
    ELSE:
      abort();                  /* invalid instruction */
    }
  }
}

/* assemble register-register instruction */
#define a_rr(n, s, d) ((insn_##n << 24) | ((s) << 20) | ((d) << 16))
/* assemble register-dest instruction */
#define a_rd(n, d) a_rr(n, 0, (d))
/* assemble register-source instruction */
#define a_rs(n, s) a_rr(n, (s), 0)

#define a_k16(n, r, k) ((insn_##n << 24) | ((r) << 16) | ((k) & 0xFfFf))
#define a_jl(s, off)  ((insn_jl << 24) | ((s) << 20) | ((off) & 0xFfFf))
#define a_call(off)   ((insn_call << 24) | ((off) & 0xFfFf))
#define a_ret         (insn_ret << 24)
#define a_halt        (insn_halt << 24)

/* dumb fibonacci: if r0 < 2 then 1 else fib(r0 - 1) + fib(r0 - 2) */
u32 program[] = {
  a_call(1),                /* call fib */
  a_halt,
  a_rr(mov, 0, 1),          /* fib: r1 := r0 */
  a_k16(lit16, 2, 2),       /* r2 := 2 */
  a_rr(sub, 2, 1),          /* r1 -= r2 */
  a_jl(1, 13),              /* if r1 < 0, go forward 13 insns */
  a_rs(push, 0),            /* push r0 */
  a_k16(lit16, 3, 1),       /* r3 := 1 */
  a_rr(sub, 3, 0),          /* r0 -= r3 */
  a_call(-8),               /* call fib */
  a_rd(pop, 1),             /* pop input r0 into r1 */
  a_rs(push, 0),            /* save return value from recursive call */
  a_k16(lit16, 3, 2),       /* r3 := 2 */
  a_rr(mov, 1, 0),          /* r0 := r1 */
  a_rr(sub, 3, 0),          /* r0 -= r3 */
  a_call(-14),              /* call fib */
  a_rd(pop, 1),             /* pop saved return value into r1 */
  a_rr(add, 1, 0),          /* r0 += r1 */
  a_ret,
  a_k16(lit16, 0, 1),       /* r0 := 1 */
  a_ret,
};

int main(int argc, char **argv)
```

```
{
  int n = argc > 1 ? atoi(argv[1]) : 6;
  u32 mem[1024];
  for (int i = 0; i < 1024; i++) mem[i] = 0xdeafbeadu;
  memcpy(&mem[128], program, sizeof(program));
  machine m = { .mem = mem };
  for (int i = 0; i < 16; i++) m.reg[i] = 0xbadfadu;
  m.PC = 128;
  m.SP = 1024;
  int result = interpret(&m, n, 0xfeedbead, 0xfeedbead, 0xfeedbead);
  printf("%d\n", result);
  return 0;
}
```

Disassembly shows that GCC compiles the switch with a jump table. (I admit I spent another half hour after those 96 minutes looking to see why it was so slow...)

However, an important caveat here: because this virtual machine implementation does not bounds-check memory accesses, it fails to be deterministic.

This program is about 20× slower than native code on my amd64 OoO laptop and about 40× slower on my i386 Atom in-order netbook.

In theory, someone implementing Veskeno will not have to write and debug the Fibonacci program and other test programs as they are writing their virtual machine, much less revise the definitions of the instructions as they go; instead they can, hopefully, assume that the instruction set is adequate, the test cases are correct, and any bugs are in their interpreter. This should speed up their programming. In the past when I've implemented simple virtual machines such as Chifir and Brainfuck, it's taken me under an hour. (But, well, my Chifir implementation had a bug I didn't notice for months, and it might still have others.)

Fixed- or variable-length instructions

Variable-length instructions are more space-efficient — the usual reason for them, irrelevant here — and make it easy to include immediate constants of the full width of a register, thus avoiding the lit16/low16 hack in XIS, the RISCy spike above.

Fixed-length instructions have other advantages. They can make it impossible to represent invalid program-counter values, which would otherwise be a potential source of divergent behavior among implementations. They facilitate conditional-skip instructions, which permit the decoupling of conditional types (equality versus ordering) from jump types (direct or indirect). By making them extremely wide, as Chifir does, they too can contain register-sized immediate contents. And they facilitate having an opcode field of less than a full byte, which reduces the number of tests needed for invalid opcodes.

As with variable-length instructions, the most important advantage of fixed-length instructions in hardware is irrelevant for Veskeno: that they enormously simplify pipelined instruction decoding.

No floating point

The Veskeno virtual machine provides no floating-point operations, despite the importance of floating-point math for numerical algorithms and the importance of reproducibility for these algorithms. Instead, floating-point operations are provided by libraries written in Veskeno's instruction set, despite the heavy performance cost, so that they will have the same behavior on all Veskeno implementations. Three reasons for this are Gen gradual underflow, `-ffloat-store`, and FMA.

IEEE 754 standardizes the behavior of the basic floating-point operations — addition, subtraction, multiplication, division, and square root — to provide bit-exact results. Given this, it would be reasonable to expect that all modern machines would produce identical results when executing a floating-point algorithm consisting of only these operations. However, although it would be reasonable, it would be wrong.

One aspect of IEEE 754 is the handling of underflow — when numbers become too small to represent in normalized form, it is specified that they start losing bits of precision, which continues down to the smallest nonzero float. Currently, Intel's "Gen" GPUs do not implement this, because it is slow. Therefore it is not reasonable to assume that all future hardware will implement gradual underflow correctly.

GCC has a `-ffloat-store` flag for use with math coprocessor instructions for the 80387, 68881, and similar chips. The 80387 and its compatible descendents, included in every 386-compatible processor since the Pentium, always internally use 80-bit extended precision. This means that the results of a sequence of floating-point operations depends on whether intermediate results are rounded to 32 or 64 bits to be stored in memory or remain entirely inside the 80-bit register set, which in turn depends on how effective the optimizer is at register allocation. This can, for example, cause some successive approximation algorithms to loop infinitely. `-ffloat-store` requires them to *always* be stored in memory, despite the ensuing dramatic loss of performance, in order to guarantee deterministic behavior.

Even with the above caveats, some might wonder if the problem is limited only to hardware that is sort of sketchy, like Gen, or obsolete, like 32-bit Intel processors. But in fact a similar, though subtler, issue arose just in the last few years, with a new "fused multiply-add" or "FMA" instruction on 64-bit processors, which can often preserve an extra bit of precision. This means that the results of an operation can differ in the least significant bit depending on whether the compiler's optimizer was successful at employing FMA on a given program.

It must be anticipated that Veskeno virtual machine implementations will be compiled by such compilers. For Veskeno, the above-described level of nondeterminism is absolutely intolerable, even merely FMA, so taking advantage of hardware floating point is not an option.

Exhaustive testing is desirable but probably too slow in the target scenario

Single-operand arithmetic instructions are feasible to test exhaustively; dual-operand instructions less so. Consider this Python program:

```
#!/usr/bin/python3
import hashlib

def add16(a, b):
    return (a + b) & 0xFfFf

def test_add16():
    h = hashlib.sha256()
    for a in range(1<<16):
        for b in range(1<<16):
            s = add16(a, b)
            h.update(bytes([s & 0xff, s >> 8]))
        if not ((a+1) & 0xf):
            print(a)

    return h.hexdigest()

if __name__ == '__main__':
    print(test_add16())
```

This eventually produces the output:

```
ca284820199ced0d15c967098f8ffc59e583a8b4120375b09ef1da4366786ca0
```

This amounts to a compact summary of the overall behavior of the `add16` function; if a different function produced the same hash, we could be reasonably confident that its behavior on 16-bit unsigned numbers was the same as add16's. And by using Merkle trees we could detect deviations without finishing the whole test, and, more important, localize them in particular parts of the input. (A cross-cutting Hamming-code-like hashing strategy would permit pinpoint localization: with 33 hashes for different subsets of these 2^{32} test cases — one for odd-numbered test cases, one for test cases whose ordinal number is odd when divided by 2 rounding downward, and so on — we can easily determine which case is failing if only one is.)

But this test takes 13 hours and 49 minutes of CPU time to produce this output on this netbook, thus testing only some 86000 addition operations per second. CPython3 on this netbook is pretty close to Veskeno's target performance of a million multiply-accumulates per second, although they are 32-bit rather than 16-bit.

It's conceivable that this test could be optimized by up to about an order of magnitude, but not by two orders of magnitude; and it's more likely that a similar test in Veskeno would be *much slower*, because SHA-256 isn't a basic operation like addition. The corresponding exhaustive test for a two-operand 32-bit math operation would require ten orders of magnitude more computation; as mentioned before, 2^{64} microseconds is some 585 millennia.

So exhaustive testing of, say, 32-bit addition, is probably not

feasible at the target performance level within the target six-hour timeframe. Even exhaustive testing of 32-bit negation would take hours. Instead, randomized tests are probably a better fit.

This is not to say that exhaustive testing has no role, just that faster kinds of testing are needed.

No vector-valued registers

Numpy can typically easily achieve about 20% of C performance on mainstream hardware today, despite the slowness of the CPython interpreter, because the inner loops are in C. One design considered for Veskeno (p. 20) used vector-valued registers and RAM — each register or memory cell could hold a vector of very large size, and Veskeno would provide SIMD instructions like Numpy's operations. Thus the interpretive overhead of a simple bytecode interpreter loop would be amortized over larger numbers of fundamental operations, increasing the speed of Veskeno programs.

The plan is currently not to take this direction, for three reasons:

- This would create the possibility of thus allocating unpredictable amounts of memory in a way hidden from the Veskeno program itself, making it impossible to guarantee failure-free execution.
- The number of distinct SIMD instructions required seems like it is probably too large to implement — and, especially, debug — in an afternoon.
- Crude estimation suggests that a straightforward interpreter without any such tricks will be more than fast enough to satisfy the priorities as described above: 8–32× is a typical interpretive slowdown, and Veskeno is aimed at an interpretive slowdown of 131072× or less.

Not counted here is the serial-computation slowdown, which is estimated (p. 20) at 32×. Above it is estimated that a Samsung Galaxy A10, for example, can do about 70 billion multiply-accumulate operations per second, but single-threaded unvectorized code on it won't get more than about 1.6 billion, 44 times slower; out-of-order processors with more execution units close the gap a little. It would not be surprising for a virtual machine that exploits such data parallelism to exceed the speed of optimized single-threaded unvectorized C.

Possible coarse-grained parallelism

There is nothing in principle that prevents Veskeno from providing a "spawn" facility to run a "child" Veskeno computation, given a program and input data, and such a facility would be very useful for writing an automatic Veskeno test suite. If several such concurrent computations can be run, this might make it possible to gain back a parallelism factor of some 8 or so on most current hardware, and probably much larger factors in the future. Such a facility is potentially risky, though; it would need to be subject to a number of restrictions:

- Although it could report *predictable* failures in the child to the parent — out-of-bounds memory accesses, for example — it could not be allowed to report unpredictable failures such as running out of

memory. Unpredictable failures could be handled by automatically retrying or by propagating the failure to the parent, killing it as well.
• It probably needs to be impossible to interact with an incomplete child computation in order to ensure determinism. For example, the ability to inquire whether a child computation was still running, or had already completed, would probably violate determinism. Any attempt to access the results of the child computation before the child's completion must transparently block until the child is complete.
• The interface must be simple enough to implement — correctly! — as part of the same afternoon as the rest of Veskeno.

More elaborate kinds of interaction could in principle be specified; for example, the parent computation could be provided with facilities to single-step the child, examine its memory space while single-stepping, and so on, as long as this did not provide it with any information about unpredictable failures, machine load, and so on. But such a facility would probably be more complex both to specify and to implement than all of the rest of Veskeno, and at any rate it can be provided less efficiently within Veskeno, without any special effort from the virtual machine implementor, by a metacircular Veskeno interpreter.

Multiplication and division?

Perhaps Veskeno should have a multiplication instruction or instructions. Most modern processors have a single-cycle multiplier, and replacing that with a subroutine call is a heavy performance penalty for programs that do a lot of multiplication, on the order of 32× to 64×.

However, multiplication can and often does overflow (a whole word's worth of bits rather than just one), requiring separate instructions for the low and high word of the results, and signed and unsigned multiplication are different; so supporting multiplication is not as low-risk as supporting addition or subtraction.

Veskeno probably should not have a division instruction for several reasons: signed and unsigned integer division are not the same, it's ambiguous which way the correct result of negative signed division should round (quotient toward negative infinity or toward zero?), division by zero is potentially a predictable failure, and division is typically slow anyway, so the impact of not using the hardware integer division instruction is less severe — both because the gap in performance will be smaller than for multiplication and because programs are already written to avoid division in hot loops whenever possible.

I think probably the right choice is to omit multiplication from an initial Protoveskeno and see how far we get, then possibly add multiplication instructions if the lack is a sufficiently large performance loss.

Instruction-set translation

Rather than writing a C compiler backend to target Veskeno, it seems that binary translation from an existing instruction set which

already has good compiler support may be the best approach. Supporting 64–128 distinct instructions may be enough, perhaps even using very simple techniques that in effect simulate the registers and flags of the target processor.

I/O operations and determinism

PGP and GnuPG have historically used I/O operations to generate cryptographically random key bits: for example, by measuring the latency of electromechanical disk requests, which are influenced by turbulence inside the disk drive, they can produce a reliably different set of numbers on every execution; another approach is by measuring the timing of the user's keystrokes. The objective is that, if you ask PGP to generate keypairs on two occasions and type the same input at it to the best of your human ability, you will still generate two different *and unpredictable* private keys. (Modern operating systems provide this facility at a systemwide level using /dev/urandom, so that randomness gathered before GnuPG or OpenSSH starts can still provide them with unpredictable secret bits.)

So we can conclude that providing a program with the ability to read the current time, or to measure the time between inputs, can allow it to defeat any efforts at guaranteeing reproducible behavior. On the other hand, interactive applications like video-games generally must have time-dependent behavior: the Space Invaders and their bombs must continue moving at a consistent speed even if the player is not providing any new input; and when they do provide input, the results are in general dependent on *when* that input is provided. Moving Pac-Man to the left for one second does not have the same results as moving Pac-Man to the left for two seconds, and so on.

How can these requirements be reconciled?

As explained above about how programs are incapable of nondeterminism in theory:

> Conventionally these results are stated for batch-mode algorithms which run for some finite period of time and then halt with a result, but it's straightforward to extend them to interactive processes — the batch-mode algorithm takes as input a previous state and an input event (which may be simply that there is no input of interest to report) and eventually produces a new state and perhaps some output.

We could take this approach in Veskeno: run a *noninteractive* computation in Veskeno, starting from a snapshot of some previous state, and ending with a new state snapshot, part of which might be, for example, a screen image and some samples of audio to output, and another part of which might specify handlers to run for future input events, maybe including timeout events. To reproduce a deterministic sequence of such deterministic computations or explore alternative histories, it would be sufficient to record the initial state and the sequence of input events that were delivered, although it might be a useful accelerant to save snapshots of some intermediate checkpoint states.

User interaction isn't the only kind of I/O, though. It's common for programs to read from and write to a filesystem, for a variety of reasons. Doing this synchronously isn't in itself a source of nondeterminism — given a frozen filesystem snapshot that is part of

the initial state from which the Veskeno computation proceeds, presumably the program will always read the same data in response to the same seek() and read() calls, unless it alters it in between with a write(). But it is potentially a source of implementation complexity and bug-proneness.

Some filesystem access happens because programs are handling data that doesn't fit in their virtual memory. This might be reading a 100-kilobyte file on a 16-bit machine or writing a 5-gigabyte file on a 32-bit machine. For Derctuo, I can avoid this problem by making Veskeno not 16 bits, and not managing multi-gigabyte datasets. If Veskeno is at some point to be pressed into service wrangling multi-gigabyte datasets, it could be wedged into the model as if it were user input: instead of terminating the computation with event handlers for keystrokes and timeouts, a computation could terminate with an event handler for a block of data becoming available. (Or you could add I/O instructions for doing this to Veskeno; this would make it no longer compatible with the Veskeno specification, but arguably so would adding these new kinds of event handlers.)

Input and output data that *does* fit into virtual memory can simply be put in virtual memory; when a computation terminates, it can do so with an indication of where its results are to be found in memory. A straightforward Veskeno implementation can simply copy such data into a large byte array, while perhaps a trickier one can take advantage of mmap() and similar facilities.

Some filesystem access happens to decouple the environment in which a program runs from what the program does. For example, I have the file /usr/lib/python3.4/encodings/mac_greek.py on this netbook. If a program does not access this file, or enumerate the contents of the directory /usr/lib/python3.4/encodings, or look at how much space is left on the disk, its execution will not be affected by this file; but if I run CPython 3.4 and type `b'\xce'.decode('mac_greek')`, that file will be loaded and used to map that byte to U+0388. It's a resource available upon request, but otherwise unobtrusive.

Usually you can add new files to a Unix filesystem or new environment variables to a Unix environment without breaking any existing programs. This contrasts to, for example, adding new positional arguments to a function call. (Adding new fields to a C struct is a kind of middle ground: it breaks existing *compiled* programs, but not existing source code, because it's an incompatible change to the ABI but not the API.) This kind of decoupling via name-value pairs is a pervasive pattern for permitting the independent evolution of different software components.

To a great extent, such decoupling can be provided within a Veskeno image without any special support from the Veskeno virtual machine: a "filesystem", a tree of string-indexed blobs, can be built in memory and accessed via a filesystem-emulation library. This collapses if multiple gigabytes of data are needed, but my intent with Derctuo is to keep the total size of all the data in the image to double-digit megabytes.

It's common for physical computers to use "memory-mapped I/O": magical memory addresses which cause things to happen in the physical world when they are written or even read. This is costly to

provide in virtual machines in general, because nearly every time memory is read or written, a check must be made for these magical addresses. For Veskeno, it seems like a particularly bad idea, since it would be easy to omit the necessary replay functionality. If I/O operations are to be added to Veskeno computations, they should be added with explicit IN and OUT instructions.

Instruction counting and metacircular instrumenting compilers

Derctuo talks fairly often about the *efficiency* of possible algorithms. Nowadays this is a difficult thing to nail down: different algorithms may use different amounts of memory, different numbers of CPU instructions, differently-predictable memory access patterns, and afford different degrees of vectorization, out-of-order instruction-level parallelism, SIMT parallelism, and coarse-grained (multicore) parallelism, as well as having different patterns of communication between different cores. As hardware heterogeneity increases further into the dark-silicon era we are entering, this already gnarly efficiency landscape is likely to become more complex rather than simplifying.

But a simple first-order approximation to computational cost is to count the number of CPU instructions executed by a single-threaded version of the algorithm. Given a nailed-down instruction set like Veskeno's, this number should be as perfectly reproducible as everything else about a Veskeno computation, and it would probably only increase Veskeno's complexity by 2–5 lines of code, a simplicity loss of perhaps 1–5%. This may be a worthwhile cost to pay.

However, as with single-stepping, this is a facility that can be provided by a metacircular Veskeno interpreter: a Veskeno program that executes Veskeno programs. Veskeno's simple instruction set suggests that the binary-translation approach used by Valgrind would be an especially suitable approach.

Memory maps and relocatability

As long as Veskeno programs can access the raw bits of Veskeno memory addresses, reproducibility requires that those bits not change between executions and implementations. Environments like the JVM avoid this problem by not providing programs with access to those bits, relying on a relatively elaborate static type system that reliably distinguishes memory pointers from other data such as characters and integers. A less elaborate hybrid system is possible, in which pointers are loaded into special registers for pointers (or "segments" or "descriptors") and stored in special memory for pointers, like KeyKOS's "nodes"; but even such a scheme seems likely to be far more complex than Veskeno's complexity budget permits. (Still, see Segments and Blocks (p. 166).)

This means, in particular, that if there's a way to change the Veskeno memory map after startup, for example by mapping in the contents of a file (or part thereof) or the results of a child computation, it must happen at a deterministically chosen, well-specified address. It need not be insensitive to the previous execution of the computation — for example, it could happen at the

end of the current data segment — but it cannot happen at an address not specified in the Veskeno specification.

Self-modifying code

Veskeno does not need to permit self-modifying code; it could use a Harvard architecture, for example, like an AVR, and use a child-spawning facility like that described earlier if it wants to generate Veskeno code dynamically. But, if it does permit self-modifying code, it is essential for its effects to be deterministic, well-specified, and well-tested; it would not be acceptable for different Veskeno implementations to handle the same self-modifications differently. The simplest solution is to require that all modifications take effect immediately, even if to the immediately following instruction.

Related work

Preservation through emulation, e.g., SIMH

van der Hoeven and Lorie's UVM

Chifir

In The Cuneiform Tablets of 2015, Long Tien Nguyen and Alan Kay described their design for a simple archival virtual machine called Chifir, for which they report having successfully preserved Smalltalk-72.

They describe their requirements as follows:

• It can be described in a single Letter or A4-sized page using English and diagrams. A "one-pager" has a nice psychological quality of compactness and elegance to it; we were inspired by the half-page Lisp metacircular evaluator in the Lisp 1.5 manual 27.
• It can be implemented in a single afternoon by a reasonably competent programmer.

Implicitly, they also require that it be sufficiently powerful to run the system they want to preserve.

My implementation of Chifir took me an hour of programming and 111 lines of C, but because Nguyen and Kay have not published their Smalltalk-72 virtual machine image or any other test data for Chifir, my implementation may very well have bugs; it might take another hour or more to find and fix all of its bugs.

Chifir is a word-oriented 32-bit three-operand memory-to-memory machine with very fluffy instruction encoding. Its 15 instructions are roughly JMP, JZ, save-return-address, MOV, LD, ST, +, -, *, /, %, <, NAND, refresh-screen, and read-keyboard; the half-duplex nature of the read-keyboard instruction makes it impossible to emulate full-duplex systems like video-games on Chifir. But Nguyen and Kay did not intend for Chifir to be universal in the same way that Veskeno is; they say:

We think that trying to design a "universal" virtual machine to serve as the simple virtual machine is a bad idea, because trying to ensure compatibility with the entire design space of computer architectures will make the resulting "universal virtual machine" very complicated. In our opinion, this is the mistake of van der Hoeven et al.'s Universal Virtual Computer for software preservation [15]. They tried to

make the most general virtual machine they could think of, one that could easily emulate all known real computer architectures easily. The resulting design [25] has a segmented memory model, bit-addressable memory, and an unlimited number of registers of unlimited bit length. This Universal Virtual Computer requires several dozen pages to be completely specified and explained, and requires far more than an afternoon (probably several weeks) to be completely implemented.

Lisp

Lisp has a simple core — not quite as simple as SK-combinators or the λ-calculus, but still pretty simple. The basic forms are COND, LABELS (now normally called `letrec`), LAMBDA, and QUOTE, which are "special forms", and the regular functions CAR, CDR, CONS, ATOM, EQUAL, and NULL; these suffice to write a metacircular interpreter for Lisp or, for example, a normal-order λ-calculus reducer.

Because both CONS and function application implicitly allocate memory, as does LAMBDA in modern interpretations (where it produces a closure), it's difficult for Lisp programs to be failure-free — when run on a finite machine they can run out of memory and crash. But, at least initially, eliminating unpredictable failures is beyond the scope of Veskeno.

A binary format like various Lisps' FASL formats could both permit rapid startup and eliminate text-related parsing bugs.

However, the history of Lisp is littered with subtle bugs. McCarthy's 1959 paper published a Lisp metacircular interpreter that inadvertently defined Lisp with dynamic scope — a bug that remained ossified in Lisp for nearly a quarter century, with workarounds like FUNARGS — and contained a few other subtle bugs; an erratum is prepended to AIM-008 saying:

> The definition of eval given on page 15 has two errors, one of which is typographical and the other conceptual. The typographical error is in the definition of evcon where "l→" and "T→" should be interchanged.
>
> The second error is in evlen. The program as it stands will not work if a quoted expression contains a symbol which also acts as a variable bound by the lambda. This can be corrected by using instead of subst in evlen a function subsq defined by …

Note that at this point McCarthy had been working on Lisp for a few years and had a more or less working implementation due to Slug Russell, and yet his QUOTE did not work properly inside a LAMBDA.

Writing the following one-pager in Python took an hour for a programmer who has implemented Lisps more than once before, running into several minor bugs on the way; bugs may still remain.

```
def Eval(sexp, env):
    return (env[sexp] if type(sexp) is str else
            specials[sexp[0]](sexp[1:], env) if type(sexp[0]) is str
                                             and sexp[0] in specials else
            Eval(sexp[0], env)([Eval(arg, env) for arg in sexp[1:]]))

def evcon(branches, env):
    for q, a in branches:
        if Eval(q, env): return Eval(a, env)
```

```
def evletrec(args, env):
    assignments, body = args[0], args[1]
    env = env.copy()
    for name, a, b in assignments: env[name] = closure(a, b, env)
    return Eval(body, env)

def closure(args, body, env):
    return lambda vals: Eval(body, augment(env, list(zip(args, vals))))

def augment(env, nvpairs):
    env = env.copy()
    for n, v in nvpairs: env[n] = v
    return env

specials = {
    'cond': evcon,
    'letrec': evletrec,
    'lambda': lambda args, env: closure(args[0], args[1], env),
    'quote': lambda args, env: args[0],
}

base_env = {
    'car': lambda args: args[0][0],
    'cdr': lambda args: args[0][1:],
    'cons': lambda args: [args[0]] + args[1],
    'atom': lambda args: type(args[0]) is str,
    'null': lambda args: not args[0],
    'equal': lambda args: args[0] == args[1],
    't': True,
}

# produces ['b']
example_prog = ['letrec', [['assoc', ['k', 'kvs'],
                           ['cond', [['equal', 'k', ['car', ['car', 'kvs']]],
                                     ['cdr', ['car', 'kvs']]],
                                    [['null', ['cdr', 'kvs']],
                                     ['quote', []]],
                                    ['t', ['assoc', 'k', ['cdr', 'kvs']]]]]],
                         ['assoc', ['quote', 'y'], ['quote',
                                                    [['x', 'a'],
                                                     ['y', 'b'],
                                                     ['z', 'c']]]]]

# produces [['X', 'a'], ['X', 'small'], ['X', 'dog']], ['X', 'sat']]
example2 = ['letrec',
            [['subst', ['f', 'd'],
              ['cond', [['atom', 'd'], ['f', 'd']],
                       [['null', 'd'], ['quote', []]],
                       ['t', ['cons', ['subst', 'f', ['car', 'd']],
                                      ['subst', 'f', ['cdr', 'd']]]]]],
             ['x', [], ['quote', 'X']]],
           ['subst', ['lambda', ['de'], ['cons', ['x'], ['cons', 'de',
                                                         ['quote', []]]]],
            ['quote', ['a', ['small', 'dog'], 'sat']]]]
```

```
if __name__ == '__main__':
    import cgitb
    cgitb.enable(format='text')
    print(Eval(example2, base_env))
```

Running Lisp efficiently requires some kind of garbage collection; the above implementation inherits from Python not only GC but also its lists, recursive function calls, equality comparison, closures, I/O, error reporting, and truthiness, and it takes advantage of Python's dictionaries. Its behavior on argument-count mismatch is inherited from Python's `zip`. It constructs circular data structures, which old versions of Python would be unable to garbage-collect. Probably an implementation in a lower-level language like C would be considerably more efficient, but would also require implementing from scratch these Python bequests. In my experience this tends to take as long or longer than implementing the semantic core above expressed in Python.

Abadi and Cardelli's ς-calculus of objects

The JVM

ActivePapers

Nix and Guix

The Cult of the Bound Variable

32-bit unsigned

Darius suggests it's worth looking at the Sandmark contestants' bugs.

Corewar Redcode

Corewar is a game in which a multithreaded processor "MARS" runs two programs that try to kill each other, alternating instructions. Like the Burroughs 5000, MARS tags memory words as instructions or data; a program that attempts to execute a data word dies.

The textual Redcode assembly language is the standard format for specifying these programs; there is no binary program format. The determinism of MARS is intentionally limited: programs are loaded at random starting addresses. (Absent this measure, whichever program started running first could win by using its first instruction to store a data word in the other program's first-executed location.)

Wirth-the-RISC

In the 1990s, Wirth became interested in the potential of FPGAs for realizing processor designs, especially designs simplified so as to be easy to teach, without losing practicality. He produced a series of progressively more complex designs in Verilog, unfortunately called RISC0, RISC1, RISC2, RISC3, RISC4, and RISC5, and ported the Oberon system to run on them. Lacking a better name, I will just call them "Wirth-the-RISC".

Wirth-the-RISC is admirably simple, with four condition-code flags for conditional jumps; 16 conditions for jumps (including "always"), which can optionally be indirect and/or save a return

address; 16 register-to-register ALU instructions, some of which have two variants — signed versus unsigned MUL, for example, and ADD with or without carry; load and store instructions with offsets; and, for the RISC5 processor's interrupts, an instruction to enable or disable interrupts, and an instruction to return from them. Four of the ALU instructions are floating-point, though my impression is that the processor does not rise to the level of being practical for floating-point work — it has no double-precision and no square-root instruction.

The fact that Wirth-the-RISC successfully runs the Oberon GUI is a testament to the practicality of this design.

SOD32

Brainfuck

Brainfuck is a virtual machine of Urban Müller's design; it was not the first of the "esoteric programming languages" (that would be INTERCAL) — or even, I think, the second — but it was in a sense the one that established esoteric programming languages as a genre, inspiring the current profusion. The Brainfuck virtual machine is, like INTERCAL, deliberately difficult to program in, but unlike INTERCAL, implementing it is extremely easy. One day in 2014 I sat down to implement it from the spec, which I think took about an hour:

```
/*
 * Brainfuck interpreter.
 */

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char **argv) {
  char program[10000];
  int fd = open(argv[1], O_RDONLY);
  if (fd < 0) {
    perror(argv[1]);
    return 1;
  }

  int progsize = read(fd, program, sizeof(program));
  close(fd);

  unsigned char memory[30001];
  int pc = 0, mp = 0;
  while (pc < progsize) {
    /* printf("[%d]", pc); */
    /* fflush(stdout); */
    switch (program[pc]) {
    case '>': mp++; pc++; break;
    case '<': mp--; pc++; break;
```

```
    case '+': memory[mp]++; pc++; break;
    case '-': memory[mp]--; pc++; break;
    case ',': read(0, &memory[mp], 1); pc++; break;
    case '.': write(1, &memory[mp], 1); pc++; break;
    case '[':
      if (memory[mp]) {
        pc++;
        break;
      }
      int bc = 0;
      do {
        if (program[pc] == '[') bc++;
        if (program[pc] == ']') bc--;
        pc++;
        if (pc >= progsize) {
          fprintf(stderr, "unmatched [\n");
          return 1;
        }
      } while (bc);
      break;
    case ']':
      if (!memory[mp]) {
        pc++;
        break;
      }
      int bbc = 0;
      do {
        if (program[pc] == ']') bbc++;
        if (program[pc] == '[') bbc--;
        pc--;
        if (pc < 0) {
          fprintf(stderr, "unmatched ]\n");
          return 1;
        }
      } while (bbc);
      pc++;
      pc++;
      break;
    default:                    /* comment! */
      pc++;
      break;
    }
  }
  return 0;
}
```

After testing some simple examples, I downloaded Linus Åkesson's implementation of Conway's Game of Life (pbuh, QEPD):

```
          Linus Akesson presents:
                The Game Of Life implemented in Brainfuck

      +>>++++[<++++>-]<[<++++++>-]+[<[>>>>+<<<<-]>>>>[<<<<+>>>>>>+<<-]<+
  +++[>++++++++<-]>.[-]<+++[>+++<-]>+[>>.+<<-]>>[-]<<<++[<+++++>-]<.<<[>>>>+
<<<<-]>>>>[<<<<+>>>>>>+<<-]<<[>>>>.+<<<++++++++++[<[>>+<<-]>>[<<+>>>>>++++++++
```

As with INTERCAL, Brainfuck ignores anything it does not understand, so the textual comments do not interfere with the execution of the program.

This was a delightful experience, because by virtue of writing 68 lines of C, I had implemented a virtual machine capable of running any Brainfuck program, and had transformed the ASCII-art textphile above into a running implementation of the Game of Life! In principle, the C program above could compute any computable function, as long as it didn't require more than 30001 bytes of memory.

Brainfuck itself, though, is a finger pointing at the moon; it is not

the moon. It has no subroutine-call mechanism, it cannot run code generated at runtime, a straightforward implementation of it is absurdly inefficient, the encoding of its programs is also absurdly inefficient, and there have been several different incompatible semantics (for example, for the overflow of a memory location, and of course for the size of memory), so some Brainfuck programs are incompatible with some Brainfuck implementations.

Also, the issue of I/O is swept under the rug. The above Life implementation is interactive on a terminal; it draws the gameboard using ASCII art. You can correct your input errors with backspace only thanks to the line-editing capabilities provided by default by the kernel or the C library; by the same token, Brainfuck programs running in the above C implementation in the same way as Life cannot provide so much as tab-completion and overstrikes, much less mouse, key-release, and graphics handling. To emulate video-games, even with a sufficiently powerful implementation, Brainfuck would need a mapping between streams of input and output bytes and the input and output events of interest; this mapping, too, would need to be standardized for such an emulation to be portable among implementations.

Here's a sample dialogue with the Life program, using this implementation of Brainfuck:

```
 abcdefghij
a----------
b----------
c----------
d----------
e----------
f----------
g----------
h----------
i----------
j----------
>de
 abcdefghij
a----------
b----------
c----------
d----*-----
e----------
f----------
g----------
h----------
i----------
j----------
>df
 abcdefghij
a----------
b----------
c----------
d----**----
e----------
f----------
```

```
g----------
h----------
i----------
j----------
>fe
 abcdefghij
a----------
b----------
c----------
d----**----
e----------
f----*-----
g----------
h----------
i----------
j----------
>ee
 abcdefghij
a----------
b----------
c----------
d----**----
e----*-----
f----*-----
g----------
h----------
i----------
j----------
>ed
 abcdefghij
a----------
b----------
c----------
d----**----
e---**-----
f----*-----
g----------
h----------
i----------
j----------
>
 abcdefghij
a----------
b----------
c----------
d---***----
e---*------
f---**-----
g----------
h----------
i----------
j----------
>
 abcdefghij
a----------
```

```
b----------
c----*-----
d---**-----
e--*--*----
f---**-----
g----------
h----------
i----------
j----------
>
 abcdefghij
a----------
b----------
c---**-----
d---***----
e--*--*----
f---**-----
g----------
h----------
i----------
j----------
>
 abcdefghij
a----------
b----------
c---*-*----
d--*--*----
e--*--*----
f---**-----
g----------
h----------
i----------
j----------
>
 abcdefghij
a----------
b----------
c----*-----
d--**-**---
e--*--*----
f---**-----
g----------
h----------
i----------
j----------
>
 abcdefghij
a----------
b----------
c---***----
d--**-**---
e--*--**---
f---**-----
g----------
h----------
```

```
i----------
j----------
>
 abcdefghij
a----------
b----*-----
c--**-**---
d--*-------
e--*---*---
f---***----
g----------
h----------
i----------
j----------
>
 abcdefghij
a----------
b---***----
c--****----
d-**--**---
e--*-**----
f---***----
g----*-----
h----------
i----------
j----------
>q
```

These 8 generations of 10×10 Life required 98 CPU seconds on this netbook (with the Brainfuck implementation compiled with `cc -O5 -fomit-frame-pointer -Wall -std=gnu99` using GCC 4.8.4), illustrating the efficiency problems of Brainfuck. I took a couple of hours to write the following C version of Åkesson's awesome program, which, compiled the same way, was able to do 80000 generations in 1.424 CPU seconds, an efficiency difference of some 700k×, suggesting that the Brainfuck slowdown in this case is about 5 or 6 orders of magnitude.

```
#include <stdio.h>

enum { ww = 10, hh = 10 };

int board[3][hh][ww];

/* From the cells in `from`, compute a parallel array with the sum of
   cells above and to the left of that cell, including that cell
   itself.  For example:

    >>> x
    array([[1, 0, 1],
           [0, 2, 1],
           [1, 1, 1]])
    >>> x.cumsum(axis=0).cumsum(axis=1)
    array([[1, 1, 2],
           [1, 3, 5],
```

```
          [2, 5, 8]])
 */
void sum(int from[hh][ww], int to[hh][ww])
{
    for (int x = 0; x < ww; x++) {
        to[0][x] = from[0][x];
        for (int y = 1; y < hh; y++) to[y][x] = to[y-1][x] + from[y][x];
    }
    for (int y = 0; y < hh; y++) {
        int total = to[y][0];
        for (int x = 1; x < ww; x++) to[y][x] = total += to[y][x];
    }
}

/* Return total neighbors in the neighborhood that includes (xmin+1,
   ymin+1), (xmin+2, ymin+1), ... (xmax, ymin+1), (xmin+1, ymin+2),
   ... (xmax, ymax).  xmin and/or ymin will be negative if the
   neighborhood is intended to encompass the leftmost and/or topmost
   cells; xmax may be >=ww-1 and/or ymax may be >=hh-1 if it is intended
   to encompass the rightmost and/or bottommost cells.
 */
static inline int rect(int sums[hh][ww], int xmin, int xmax, int ymin, int ymax)
{
    if (xmax > ww-1) xmax = ww-1;
    if (ymax > ww-1) ymax = hh-1;
    int ul = xmin < 0 ? 0 : ymin < 0 ? 0 : sums[ymin][xmin];
    int ur = ymin < 0 ? 0 : sums[ymin][xmax];
    int ll = xmin < 0 ? 0 : sums[ymax][xmin];
    int lr = sums[ymax][xmax];
    return lr - ur - ll + ul;
}

/* Return total cells in the 3×3 neighborhood centered on (x, y). */
static inline int neighborhood(int sums[hh][ww], int x, int y)
{
    return rect(sums, x-2, x+1, y-2, y+1);
}

static inline int should_live(int cells[hh][ww], int sums[hh][ww], int x, int y)
{
    int n = neighborhood(sums, x, y);
    return cells[y][x] ? 3 <= n && n <= 4 : n == 3;
}

void generation(int from[hh][ww], int to[hh][ww], int scratch[hh][ww])
{
    sum(from, scratch);
    for (int y = 0; y < hh; y++) {
        for (int x = 0; x < ww; x++) {
            to[y][x] = should_live(from, scratch, x, y);
        }
    }
}

void print_board(int cells[hh][ww])
```

```
{
    putchar(' ');
    for (int x = 0; x < ww; x++) putchar('a' + x);
    putchar('\n');

    for (int y = 0; y < hh; y++) {
        putchar('a' + y);
        for (int x = 0; x < ww; x++) putchar(cells[y][x] ? '*' : '-');
        putchar('\n');
    }
}

/* Returns 1 if we should do another generation, 0 to quit */
int prompt(int cells[hh][ww])
{
    for (;;) {
        print_board(cells);

        putchar('>');
        fflush(stdout);

        int c1 = getchar();
        if (c1 == 'q' || c1 == EOF) return 0;
        if (c1 == '\n') return 1;

        int c2 = getchar();
        int newline = getchar();
        if (c2 == EOF || newline == EOF) return 0;

        int *cell = &cells[c1-'a'][c2-'a'];
        *cell = !*cell;
    }
}

int main()
{
    int which = 0;
    for (;;) {
        if (!prompt(board[which])) return 0;
        generation(board[which], board[!which], board[2]);
        which = !which;
    }
}
```

Urbit's Nock

Urbit is Mencius Moldbug's effort to establish an internet with a feudal, authoritarian structure, which he believes to be the ideal structure for a society. The basic foundation of Urbit is a deterministic, reproducible virtual machine called Nock, named after a political propagandist Moldbug admires despite Nock's private contempt for Jewish people. Nock implements a combinator-graph-reduction instruction set encoded as integers. The rest of the Urbit distributed computation system is built atop Nock.

Nock's basic instruction repertoire is too limited to be usably efficient for many of the tasks required for a distributed-computing system like Urbit; this is partly compensated using a mechanism called "jets". The Nock implementation recognizes certain pieces of Nock code at runtime and, rather than evaluating them instruction by instruction, instead invokes a "jet" — a subroutine written in C that is hoped to produce an equivalent result. Perhaps the most egregious example is an implementation of the Markdown document markup language, where a C implementation of Markdown is shamelessly substituted when a particular Nock implementation of Markdown is encountered.

Jets offer an apparent escape from the tradeoff between simplicity of specification and usable levels of efficiency. And, in theory, they provide an unambiguous behavior specification for the native code to adhere to. However, they aren't a viable option for Veskeno, both because they means that a practically usable implementation requires an enormous amount of code whose contents must be guessed at by the implementor, and because in practice that code will be buggy in all modern implementations, since we don't yet have sufficiently powerful formal methods for people to use them routinely, so if Veskeno used jets, no Veskeno results would be reproducible in practice.

Consequently Nock is less suitable than even Brainfuck as a basis for Veskeno.

Simplicity

Simplicity is Russell O'Connor's verifiable smart-contract language, designed for Ethereum. It is a very interesting project, but like Nock, it relies on jets to reach usable efficiency. It's capable of expressing only finitary computations — those that could in principle be expressed by a finite table of input-to-output mappings, although Simplicity is designed to be able to practically express finitary computations whose tables, though finite, would be too large to construct explicitly. Simplicity programs are guaranteed to terminate because, like Bitcoin Script, it lacks an iteration construct, relying on code repetition to achieve finite iteration.

For these reasons, Simplicity is even less suitable as a basis for Veskeno than Nock.

Wasm

Smalltalk-78

The LuaJIT "bytecode" format

Lua's register-based "bytecode" format — really a wordcode — is famous for its efficiency. Considering this program in C:

```
fib(n) { return n < 2 ? 1 : fib(n-1) + fib(n-2); }
main(int c, char **v) { printf("%d\n", fib(atoi(v[1]))); }
```

And its Lua equivalent:

```
function fib(n) if n < 2 then return 1 else return fib(n-1)+fib(n-2) end end
print(fib(tonumber(arg[1])))
```

Compiling with `gcc -O -fomit-frame-pointer fib.c -o fib` with GCC 4.8.4, on this Atom netbook, it takes 101–116 ms to compute 3524578 with `./fib 32` and 399–406 ms to compute 14930352 with `./fib 35`. Under PUC Lua 5.2.3, `fib.lua 32` takes 2.809–2.839 s and `fib.lua 35` takes 11.856–12.211 s, both with the same results. Under LuaJIT 2.0.2, `fib.lua 32` takes 196–212 ms and `fib.lua 35` takes 1.132–1.133 s.

So we can say that, on this crude microbenchmark, PUC Lua is 29–31 times slower than C, while LuaJIT is 1.6–2.9 times slower than C. Reputedly LuaJIT 2's "bytecode" interpreter, which Mike Pall wrote in assembly, is faster than many high-level languages' compiled code; unfortunately there does not seem to be an option to disable the JIT compiler for easy microbenchmarking.

It's somewhat to be expected that the extra type checks Lua must do will slow down the process, especially in software, especially on an in-order processor like this Atom. Perhaps that accounts for the speed difference between XIS, the RISCy spike above (1/20 native), and PUC Lua (1/30).

CPython is the usual contrast here. In CPython 2.7.6, this program takes 4.963–5.176 s to compute fib(32), 42–51 times slower than C:

```
#!/usr/bin/python
import sys
fib = lambda n: 1 if n < 2 else fib(n-1) + fib(n-2)
print(fib(int(sys.argv[1])))
```

LuaJIT uses its own slightly different "bytecode" format. As explained in the LuaJIT Wiki, the LuaJIT bytecode, like the PUC Lua bytecode, has a fixed 32-bit-wide format with 8-bit fields. The opcode is the least significant 8 bits; the 2-operand instructions have a 16-bit field as the second operand, which is usually an index into a constant table. There are 16 comparison ops (which conditionally skip the following instruction, which is always a JMP), 4 unary ops, 17 "binary" ops (one of which, string concatenation, is actually variadic), 6 constant ops, 7 "upvalue" and function ops, 11 ops for manipulating Lua tables (like the GSET, GGET, and TGETB operations above), 8 calling and iteration ops (like CALL and CALLM above), 4 return ops (like RET1 and RET0), 12 loop and branch ops, and 9 function-header pseudo-ops, for a total of 94 ops.

`luajit -bl fib.lua` dumps the bytecode:

```
-- BYTECODE -- fib.lua:2-2
0001    KSHORT   1   2
0002    ISGE     0   1
0003    JMP      1 => 0007
0004    KSHORT   1   1
0005    RET1     1   2
0006    JMP      1 => 0015
0007 => GGET     1   0      ; "fib"
0008    SUBVN    2   0   0  ; 1
0009    CALL     1   2   2
0010    GGET     2   0      ; "fib"
0011    SUBVN    3   0   1  ; 2
```

```
0012    CALL    2   2   2
0013    ADDVV   1   1   2
0014    RET1    1   2
0015 => RET0    0   1

-- BYTECODE -- fib.lua:0-4
0001    FNEW    0   0       ; fib.lua:2
0002    GSET    0   1       ; "fib"
0003    GGET    0   2       ; "print"
0004    GGET    1   1       ; "fib"
0005    GGET    2   3       ; "tonumber"
0006    GGET    3   4       ; "arg"
0007    TGETB   3   3   1
0008    CALL    2   0   2
0009    CALLM   1   0   0
0010    CALLM   0   1   0
0011    RET0    0   1
```

Many of these ops are specialized versions of basic operations; there are, for example, three SUB instructions, two of which are specialized to the case where one of the operands is a constant. Some of the operations are duplicated to provide the JIT compiler a place to record its success or failure at JIT-compiling the loop body.

There is no specialized version of the "$>=$" operation for comparing against a constant, so the "< 2" test in `fib` is compiled to KSHORT (load immediate) followed by ISGE; similarly, there is no specialized version of the `return` operation, so `return 1` is compiled to KSHORT followed by RET1.

As on the SPARC or in Smalltalk-80, each function evidently has its own set of registers; the main-program code at the bottom of the listing above begins by getting some variables fro the global namespace in registers 0, 1, 2, and 3, and then after calling `tonumber` (in register 2) and `fib` (in register 1) it expects to still find `print` in register 0, even though within `fib` the argument `n` is evidently in register 0. Thus no bytecode need be emitted to save and restore context upon function call or return.

The three-operand nature of LuaJIT's bytecode saves some operations, and thus some opcode dispatches, compared to the two-operand XIS code above, which has 19 instructions in the `fib` subroutine rather than 15. Where XIS has

```
a_rr(mov, 0, 1),            /* fib: r1 := r0 */
a_k16(lit16, 2, 2),         /* r2 := 2 */
a_rr(sub, 2, 1),            /* r1 -= r2 */
a_jl(1, 13),                /* if r1 < 0, go forward 13 insns */
```

LuaJIT has

```
0001    KSHORT  1   2
0002    ISGE    0   1
0003    JMP     1 => 0007
```

although perhaps this has as much to do with LuaJIT discarding the

subtraction result rather than storing it in a destination register. A recursive call `fib(n-2)` in LuaJIT is three instructions, and would be two if not for the possibility of something having rebound the name `fib`:

```
0010    GGET     2   0      ; "fib"
0011    SUBVN    3   0   1  ; 2
0012    CALL     2   2   2
```

while XIS requires six, due to explicit saving and restoring of argument registers:

```
a_rs(push, 0),            /* save return value from recursive call */
a_k16(lit16, 3, 2),       /* r3 := 2 */
a_rr(mov, 1, 0),          /* r0 := r1 */
a_rr(sub, 3, 0),          /* r0 -= r3 */
a_call(-14),              /* call fib */
a_rd(pop, 1),             /* pop saved return value into r1 */
```

I don't know if there's a way to get such implicit save/restore into a Veskeno-sized spec; maybe make some of the "registers" index off a stack pointer in memory that increments or decrements by some constant after a call, like a lobotomized SPARC? Where would you store the return address — would it eat a general-purpose register?

If I remember correctly, the SPARC has 64 general-purpose registers: 8 for global variables, and 48 in a "register window", of which 8 are shared with the caller, 8 are local, and 8 are shared with callees — so the window shifts by 16 on every call and return. The idea is that a simple, slow implementation can store all of these windows in RAM; a slightly less simple one can use 48 registers and save 16 to RAM on every call and restore them on every return; and a more sophisticated implementation can maintain a circular buffer that only "spills" to RAM when it gets full. Thus the "S" for "Scalable" in "SPARC".

Part of CPython's slowness is because CPython's bytecode is stack-based rather than register-based, commonly requiring about twice as many opcode dispatches as Lua. The above function is 18 CPython bytecode ops, rather than LuaJIT's 15; its leaf path is 7 ops rather than 5, and its non-leaf path is 16 ops rather than 11, so for this microbenchmark the dispatch penalty of stack-machine code is smaller than that typical factor of 2.

```
3           0 LOAD_FAST                0 (n)
            3 LOAD_CONST               1 (2)
            6 COMPARE_OP               0 (<)
            9 POP_JUMP_IF_FALSE       16
           12 LOAD_CONST               2 (1)
           15 RETURN_VALUE
      >>   16 LOAD_GLOBAL              0 (fib)
           19 LOAD_FAST                0 (n)
           22 LOAD_CONST               2 (1)
           25 BINARY_SUBTRACT
           26 CALL_FUNCTION            1 (1 positional, 0 keyword pair)
           29 LOAD_GLOBAL              0 (fib)
           32 LOAD_FAST                0 (n)
           35 LOAD_CONST               1 (2)
           38 BINARY_SUBTRACT
```

```
            39 CALL_FUNCTION           1 (1 positional, 0 keyword pair)
            42 BINARY_ADD
            43 RETURN_VALUE
```

As one specific example, this three-op sequence corresponds to a single LuaJIT op:

```
            19 LOAD_FAST               0 (n)
            22 LOAD_CONST              2 (1)
            25 BINARY_SUBTRACT
```

```
0008    SUBVN    2   0   0  ; 1
```

Both LuaJIT and CPython separate the comparison and the jump into two separate instructions; in LuaJIT the comparison is effectively a conditional-skip instruction as on HP calculators. Conditional skip is very easy to implement in software for a fixed instruction length, but very easy to implement *incorrectly* otherwise.

To complete the comparisons, the i386 code emitted by GCC in the tests above was as follows:

```
804844d:    56                  push   %esi
804844e:    53                  push   %ebx
804844f:    83 ec 14            sub    $0x14,%esp    ; useless waste
8048452:    8b 5c 24 20         mov    0x20(%esp),%ebx ; n
8048456:    b8 01 00 00 00      mov    $0x1,%eax       ; return 1
804845b:    83 fb 01            cmp    $0x1,%ebx       ; n <= 1?
804845e:    7e 1a               jle    804847a <fib+0x2d>
8048460:    8d 43 ff            lea    -0x1(%ebx),%eax ; n-1
8048463:    89 04 24            mov    %eax,(%esp)     ; pass arg
8048466:    e8 e2 ff ff ff      call   804844d <fib>
804846b:    89 c6               mov    %eax,%esi       ; save result
804846d:    83 eb 02            sub    $0x2,%ebx       ; n-2
8048470:    89 1c 24            mov    %ebx,(%esp)     ; pass arg
8048473:    e8 d5 ff ff ff      call   804844d <fib>
8048478:    01 f0               add    %esi,%eax       ; sum results
804847a:    83 c4 14            add    $0x14,%esp
804847d:    5b                  pop    %ebx
804847e:    5e                  pop    %esi
804847f:    c3                  ret
```

This is 11 operations in the leaf-call base case and 19 operations in the non-leaf recursive case. To avoid redundant saves and restores around the recursive calls, it keeps its local variables (`n` and the return value from the first recursive call) in callee-saved registers %esi and %ebx; this reduces the code size but has no real effect on performance. (If it had used caller-saved registers, as I did in the XIS code, the initial root call to `fib` would have avoided the cost to save and restore them, but that is not significant.)

It suffers from the shitty i386 C iBCS calling convention where everything goes on the stack. Revising it to

```
__attribute__((fastcall)) int fib(int n)
{
```

```
    return n < 2 ? 1 : fib(n-1) + fib(n-2);
}
```

yields about 17% shorter runtimes with `gcc -O -fomit-frame-pointer fib.c -o fib`, of 334–336 ms with `./fib 35`, and the following improved code, with only 17 instructions (12% less):

```
804844d:    56                  push   %esi
804844e:    53                  push   %ebx
804844f:    83 ec 04            sub    $0x4,%esp    ; still useless
8048452:    89 cb               mov    %ecx,%ebx       ; n
8048454:    b8 01 00 00 00      mov    $0x1,%eax       ; return 1
8048459:    83 f9 01            cmp    $0x1,%ecx       ; n < 1?
804845c:    7e 14               jle    8048472 <fib+0x25>
804845e:    8d 49 ff            lea    -0x1(%ecx),%ecx ; n-1, arg
8048461:    e8 e7 ff ff ff      call   804844d <fib>
8048466:    89 c6               mov    %eax,%esi       ; save result
8048468:    8d 4b fe            lea    -0x2(%ebx),%ecx ; n-2, arg
804846b:    e8 dd ff ff ff      call   804844d <fib>
8048470:    01 f0               add    %esi,%eax       ; sum results
8048472:    83 c4 04            add    $0x4,%esp
8048475:    5b                  pop    %ebx
8048476:    5e                  pop    %esi
8048477:    c3                  ret
```

(Adding `static inline` induces GCC to inline it into itself five levels deep, resulting in 242 instructions that include 32 recursive calls, and more than doubling the execution speed, to 157 ms runtime for `./fib 35`.)

This is getting pretty deep into optimization hacks; the justification is just that it illuminates some of the tradeoffs between different instruction-set choices.

SWEET-16

As I wrote in "bytecode interpreters for tiny computers" in 2008:

Steve Wozniak's SWEET16 16-bit virtual machine, included as part of Integer BASIC, supposedly doubled the code density of the 6502. The virtual machine itself was 300 bytes of 6502 assembly, implementing these instructions; here "#" means "[0-F]".

```
0x1# SET: load immediate                0x2# LD: copy register to accumulator
0x3# ST: copy accumulator to register   0x4# LD: load byte indirect w/ increment
0x5# ST: store byte indirect w/incr     0x6# LDD: load two bytes ind w/incr
0x7# STD: store two bytes ind w/incr    0x8# POP: load byte indirect w/predecr
0x9# STP: store byte ind w/predecr      0xA# ADD: add register to accum
0xB# SUB: subtract register from acc    0xC# POPD: load 2 bytes ind w/predecr
0xD# CPR: compare register w/acc        0xE# INR: increment register
0xF# DCR: decrement register            0x00 RTN to 6502 mode
0x01 BR unconditional branch            0x02 BNC branch if no carry
0x03 BC branch if carry                 0x04 BP branch if positive
0x05 BM branch if minus                 0x06 BZ branch if zero
0x07 BNZ branch if nonzero              0x08 BM1 branch if -1
0x09 BNM1 branch if not -1              0x0A BK break (software interrupt)
0x0B RS return from sub (R12 is SP)     0x0C BS branch to sub (R12 is SP)
```

0x01–0x09 and 0x0C have a second byte which is a signed 8-bit displacement. If you want a 16-bit jump, you can push it on the stack and RS.

That's it, 28 instructions, 300 bytes of machine code to implement them. And I thought the 6502 was already reasonable on code density, so this was apparently quite a win.

It's notable to me that his only ALU operations here are ADD, SUB, CPR, INR, and DCR; there are no bitwise operations, not even a shift-right. I'm guessing that SET was followed by a 16-bit immediate to load into R#, though that isn't mentioned in my notes.

This is about the right level of complexity for Veskeno, although I'd go 32-bit and trade some of the condition codes and branching options for some bitwise operations.

Darius Bacon suggested that one of the reasons XIS was so slow was that it didn't have a distinguished accumulator, so every binary operation had to index an array three times: once to read each input and once to write the output. (It also had to extract the relevant fields from the instruction word.) As with stack machines, a single-accumulator machine like the SWEET-16 reduces the number of operands that need to be decoded and indexed, at the expense of requiring a larger number of opcodes to be decoded for a given task.

Chip-8

Thanks

Discussions with Darius Bacon, John Cowan, and Sean B. Palmer greatly contributed to the Veskeno design, although undoubtedly any of them would be horrified at its deficiencies.

Topics

Convincingness

Kragen Javier Sitaker, 02020-06-20 (1 minute)

Dijkstra said that it was essential to get your knowledge out of your own head so that it wouldn't die with you: to transmit it to other people. But I'm selfish enough to find that less compelling than several other reasons.

One is that, when your knowledge is only in your own head, it's easy to fool yourself into thinking you know things that you don't really know. If you try to tell other people about it, sometimes they will be harder to convince. They may need to see more convincing evidence than the evidence you had gathered previously. By organizing the evidence and gathering more of it, you may discover that you were mistaken, in whole or in part; objectively observable evidence is generally higher-quality evidence. Sabine Hossenfelder wrote a very interesting article about her time listening to physics crackpots expound their theories, and in many cases they didn't have a clear idea of what kind of thing would count as evidence, or even as a physical theory.

Another is that they may offer contributions: they may suggest that the technique you have devised would be useful for something you hadn't thought of, or point out a weakness you hadn't seen, either because they know something you don't or just because their perspective is different from yours.

Topics

- Crackpots (p. 897) (3 notes)
- Epistemology (p. 961) (2 notes)
- Dijkstra

Lantern gears

Kragen Javier Sitaker, 02020-06-20 (updated 02020-06-28)
(1 minute)

"Lantern gears" is a term, apparently originating from Matthias Wandel, for a kind of gear that was ubiquitous in medieval clocks: two parallel discs joined by a circle of round bars around their perimeters. It would be a lantern if you put a candle in the middle. If it weren't for the discovery of the Hellenistic-era triangular-tooth-profile Antikythera Mechanism, I would have thought they predated meshing spur gears.

While it's tricky to get involute-profile gears to mesh without binding if they have fewer than seven teeth, lantern gears can work well with as few as three teeth, because the mating gear's teeth can sweep around the interior of the lantern gear. Medieval clocks, with their more primitive toothforms, did not come close to this level of optimization, but I think they did get substantially better gear ratios than they could have managed with triangular teeth.

Topics

Segments and blocks

Kragen Javier Sitaker, 02020-06-20 (updated 02020-12-16)
(51 minutes)

Consider the problem of efficiently implementing some kind of virtual machine, like Veskeno (p. 126) or the JVM. Often it's desirable for the virtual machine to be able to provide bounds checking and garbage collection, thus preventing indexing, type, and memory-management errors from provoking entirely unpredictable behavior.

Run-time bounds checking is expensive, though, so it would be nice to avoid it most of the time. The current standard approach to this is to hope your optimizing compiler will be able to hoist your bounds checks out of your inner loops. But I think there is a simpler and more orthogonal approach.

Exploring this, I think I found a way to write a featureful and adequately fast multitasking system with memory protection on microcontrollers, perhaps similar to Liedtke's pre-L4 designs, L3 and Eumel; it ought to straightforwardly support paradigms like transactional shared memory, ACID transactions, and access to filesystem snapshots, and even helps to support clustering.

Safe indexing without bounds checks

The 8086 and its descendants index the general-purpose register file in almost every instruction. The general-purpose register file consists of 8 registers (16-bit registers in the 8086, 32-bit in the i386, 64-bit in amd64), but no bounds-checking is required, because the index field in the instruction is only 3 bits, so indexing errors are impossible. (Amd64 adds additional instruction formats that can index a larger 16-register general-purpose register file, using 4-bit fields.)

Similarly, every memory reference on the i386 in protected mode indexes into some 4096-byte page with the last 12 bits of the effective address. This indexing, too, avoids any bounds checking — although the more significant 20 bits of the memory address are looked up in the processor's TLB, and if they are not found, a tree traversal is performed, with the possibility of a protection fault if no page is mapped.

So suppose our virtual machine provides access to pointer-free "string memory" in, say, 1024-byte blocks, and a virtual-machine instruction to index into the current block with an 10-bit index. A bytecode loop running in the virtual machine can freely generate such indices and read and write the current block without incurring any expense of bounds-checking. Of course, that the array or record being indexed by the virtual machine may be smaller than 1024 bytes, and wrapping around to the beginning may not be an acceptable handling of overflowing those bounds, so this may not provide bounds-checking from the point of view of the high-level language implemented — but it prevents the bytecode from corrupting the virtual machine's data structures.

Multiple block keys

Suppose we want to access more than 1024 bytes in our program? We can have multiple block pointers in "segment descriptor" or "block descriptor" or "block key" registers in the virtual machine. 4, 8, or 16 might be a reasonable number. How do we specify which segment to use? There are many possibilities. The read-string-memory and write-string-memory instructions could contain a segment field indicating which register to use; the virtual machine could provide an instruction that sets the current segment to one of the segment registers; different modes of accessing memory could use different current-segment registers (for example, instruction fetch, data read, and data write); you could use the 8086 "instruction prefix" mechanism where non-default segment registers are selected for a single instruction by a special instruction before it; or some combination.

Implicitly using a current-segment register avoids indexing into the array of registers in the virtual-machine implementation. Using a separate current-write-segment register for write access potentially permits enforcing read-only access to data. Even if no read-only *restrictions* are desired, in a virtual-memory system with no MMU, this would allow the block to be efficiently marked dirty so that it could be flushed back to stable storage; perhaps transactional memory, copy-on-write sharing, and checkpoint and rollback could be supported in this way as well. Similarly, explicitly selecting blocks for reading would efficiently give an LRU eviction system the data it needs to work, as well as permitting them to be faulted in from slower storage if needed.

Nodes

But suppose we want to access more than 8192 bytes of data in our program? We need some way to store the referents of block keys, but we cannot store them in blocks themselves — the program could overwrite them. Instead let us store block keys in a different kind of structure, which following KeyKOS terminology we will call a "node". A node contains, say, 64 block key slots. The virtual machine contains a "current node register" analogous to the "current block register", and a set of 4–16 node registers analogous to the 4–16 block or segment registers. A "read block key" instruction takes a 6-bit index and loads the corresponding block key into the current segment register; the analogous "write block key" stores the block key from the current segment register into the specified block key slot of the current node.

All the block-key slots of a node and all the block-key registers of the virtual machine are guaranteed to contain valid block keys at all times, so these virtual-machine instructions need do no validation.

64 block keys in a node give us access to 65536 bytes of data, and we can keep keys to another, say, 7168 bytes in the, say, 7 non-current block-key registers.

But suppose we want access to more than 72704 bytes of data? For that we use multiple nodes.

Node keys

In addition to the 64 block keys, a node *also* contains (say) 64 *node* key slots, and there are analogous "read node key" and "write node key" instructions which permit traversing and mutating the graph of nodes. Like the block-key instructions, these instructions require no validity checking; there is no way to copy a block key into a node-key slot or vice versa, and there is no way to copy either kind of key into a block or to copy data from a block into a key slot.

The read-key and write-key instructions have indirect versions that take the indices of a slot within a node from a virtual-machine register rather than the instruction itself. This permits programmatic indexing of the node graph without unwieldy 64-way conditionals or self-modifying or dynamically-generated code.

There is nothing to prevent the node graph from being arbitrarily cyclic or to prevent nodes from becoming unreferenced, but a garbage collector can safely traverse this node graph. Because the nodes are so large, 256–1024 bytes, this should be a relatively short process — a machine with 16 gibibytes of RAM and 64-bit pointers cannot accommodate even 16'777'216 nodes, and if the nodes are being used to index a tree of blocks in RAM, there can't be even 262'144 nodes. So a full garbage collection should normally be submillisecond. The corollary, of course, is that these nodes are not going to be a reasonable way to implement small data structures like a Lisp "cons" or "pair", costing at least some 64 times as much as a reasonable cons. Still, compared to CPython or Perl, that's still not that much.

The repertoire

So, the full inventory of operations is something like the following:

- read-string-memory(bk, u10) → u8 or u32 or something via the given block key, which may be implicit for efficiency;
- write-string-memory(bk, u10, u8 or u32 or something), analogously — these two operations might come in multiple widths;
- allocate-block() → bk, allocates a fresh block (with the destination location perhaps implicit);
- allocate-node(bk) → nk, allocates a fresh node all of whose block keys initially refer to the block given by bk;
- read-block-key(nk, u6) → bk, reads a block key from the node given by nk in the slot given by u6;
- write-block-key(nk, u6, bk), analogously;
- read-node-key(nk, u6) → nk, analogously to read a node key;
- write-node-key(nk, u6, nk), analogously.

Also, possibly one or more of the following:

- select-write-block-key(u4), start using the block key in the identified block key register for write operations;
- select-block-key(u4), analogously but either for data read operations or for all operations;
- far-call(u4, offset), transfer control to the code at the given offset in the block whose key is in the identified block key register; this is not applicable if the virtual machine's code is not itself stored in blocks;
- select-node-key(u4), start using the node key in the identified node key register;

• block-key-prefix(u4), use the block key in the identified block key register for the next operation only.

To index a larger memory area than a single block, you could use an operation sequence something like the following:

• r2 := r1; supposing r1 has the index
• r2 >>= constant 10
• select-block-key(r2); supposing the current node is an index of a 64KiB memory area
• read-string-memory(r1); supposing read-string-memory only pays attention to the low 10 bits.

In the case of accessing a multi-word chunk of data from a block, the first three operations can be amortized over many accesses to the same block.

Matrices

The once and future king of computer applications is numerical matrices, with applications such as matrix-vector multiply xGEMV, matrix-matrix multiply xGEMM, and eigenvalue computation xSYTRD/xGEBRD/xSTERF/xSTEDC accounting for a good deal of the usage of many computers — historically due to physics models, now due to artificial neural networks.

The obvious way to organize a matrix for access locality in a blocks-and-nodes system is to divide it into rectangular or square blocks; if it's 32-bit single-precision, an 8×8 block fits into a 256-byte storage block, and in 64-bit double-precision, two 4×4 blocks do. In SGEMV matrix-vector multiply, multiplying an 8×8 matrix block by an 8-element vector segment yields an 8-element partial-sum vector segment in 64 multiply-accumulates; in SGEMM matrix-matrix multiply, multiplying two 8×8 matrix blocks yields an 8×8 partial-sum matrix block in 512 multiply-accumulates.

These seem likely to be sufficiently large amounts of computation that the cost of faulting in a block will not be overwhelming, particularly if any I/O latency can be hidden with multitasking.

The *other* king of computer applications is slinging around pixels to put on the screen, and a similar 8×8 block of 32-bit BGRA pixels seems like a good fundamental unit to use there.

Related systems

As mentioned above, Jochen Liedtke wrote some systems somewhat similar to this design before writing L4, providing memory protection and process isolation on Z80-based systems with what I understand to be a trusted compiler.

The Burroughs B5000

The Burroughs B5000 is probably where this kind of structure derives from originally, but I still need to read THE DESCRIPTOR to learn about it.

The B5000 tagged every 48-bit memory word with a code/data bit, thus providing "W^X" functionality at a memory-word level rather than a page level; its descendants added two more tag bits,

providing dynamic typing at the hardware level, so that for example only a single ADD instruction was needed, dynamically dispatching to single- or double-precision addition; its "descriptors" indicated whether an array contained words or bytes (and, if bytes, bytes of which of the three supported sizes.)

The relation to KeyKOS

As I mentioned above, this is in some sense copied from KeyKOS, although there are some differences. KeyKOS didn't statically segregate block keys ("page keys") from other kinds of keys, and it didn't have "block key registers" or "node key registers" or any key registers other than the ones in the nodes.

KeyKOS had various other abilities.

It used the IBM 370 virtual-memory mechanism, and later the SPARC virtual-memory mechanism, to let the "virtual machine bytecode" be the regular CPU instructions, mapping many-page segments with the MMU so that the four-instruction sequence above was just a regular memory access.

Space and time were divided up hierarchically with "space banks" and "clocks" — you needed access to a non-exhausted space bank to allocate space and a non-exhausted clock in order to consume CPU time. The owner of a space bank could revoke all the storage allocated from it.

It had kinds of keys other than page keys and node keys — it had invocation keys and resumption keys supporting efficient remote procedure call between separate processes ("domains"), as well as keys granting access to other kinds of kernel objects such as space banks and clocks.

There was a "weaken" operation that could convert a normal node key or page key into a "sense key" which only permitted read operations — transitively, so that if a page key was fetched from a node via a sense key, that page key would also be returned as a read-only sense key.

There was a closely-held KEYBITS key to obtain the raw bits of a key, so that efficient lookups by key value were possible, though I think all processes were able to compare two keys for equality.

KeyKOS was transparently persistent: periodically it would stream out to disk all the dirty pages and nodes, then commit a checkpoint.

But I think even the minimal nodes-and-blocks structure described above is enough to be useful.

The relation to Forth

Forth systems traditionally used a very simple manual virtual-memory system instead of a filesystem. `2303 BLOCK` would ensure that 1024-byte block number 2303 from the disk was loaded into a block buffer, and return the address of that buffer; `UPDATE` would mark as dirty the last block thus referenced and ensure that it would be written to disk when necessary. Block eviction was guaranteed LRU, there were always at least two block buffers (GForth uses 20), and multithreading was cooperative, so you could be sure that the addresses of the two most recently referenced blocks would remain valid until you referenced another block or yielded control.

Forth does not make any attempt to separate pointers from other data or to check bounds on array indexing.

The relation to Smalltalk

A Smalltalk method normally runs with access to some local variables, including its arguments; a vector of instance variables in its receiver; and a pool of constants associated with, I think, the method. Different bytecodes are assigned to load and store from each of these "segments", except that the constant pool is not writable. There are no indirections there; the offsets are all hardcoded into the instructions. Arrays are instead treated as a separate class of object whose `#at:` and `#at:put:` methods are "primitives", handled by native code linked into the virtual machine.

Smalltalk does not have a notion of "pointer-free data"; its SmallIntegers, characters, booleans, and symbols ("selectors") are treated as full-fledged objects and nominally accessed by sending them messages, although some of them normally are implemented by storing all their (immutable) data in a tagged pointer rather than boxed in memory like CPython. Some selectors like `#ifTrue:ifFalse:` are special-cased by the virtual machine.

(Hmm, actually maybe Smalltalk does have such a notion: "bits" fields.)

So in a sense this is a simplification of the Smalltalk model, with just one uniform kind of node for instance variables, local variables, etc., but with storage for pointer-free bytes slapped onto the side.

Kaehler & Krasner's 1982 LOOM paper describes an approach that is very similar in many ways, although unfortunately they had not yet finished the system at the time they published their paper, saying, "Our LOOM virtual memory system is in its infancy. We are only beginning to make measurements on its performance." Other authors of the LOOM system included Althoff, Weyer, Deutsch, Ingalls, and Merry, with input from Bobrow and Tesler.

LOOM maintains an in-RAM cache of up to 2^{15} "resident" objects linked together with 16-bit short Oops, out of a possible total of 2^{31} objects on disk (occupying a maximum of 2^{33} bytes, since it was 1982), linked together with 32-bit long Oops. Nonresident objects' ambassadors in RAM are called "leaves". They mention that the average object in their system consumes 13 words in memory (26 bytes), plus perhaps a couple more words in the Resident Object Table. To save RAM, some short-Oop fields are just 0 ("lambda") instead of pointing at leaf objects, requiring LOOM to refetch the on-disk object to find the long Oop they're supposed to refer to.

LOOM de-lambda-izes the entire receiver, fleshing out lambdas into full leaves, before invoking a method. Thus it avoids null checks on every field access. This is reminiscent of the microcontroller-focused mechanism described above which brings blocks or nodes into memory when their keys are brought into a virtual-machine register.

Their short-Oop mechanism is table-based, unlike HotSpot's compressed-Oop mechanism, which represents a 64-bit object pointer as a 36-bit (?) offset from a global heap base address, shifted right by 4 (?) bits and thus stored in a 32-bit word. Being table-based permits

relocation of objects when their 4-word leaves are replaced by full-fledged resident objects after being brought in from disk. They do suggest using precisely HotSpot's compressed-Oop approach to support 2^{36} bytes of on-disk objects, though, and their RAM is 16-bit-word-oriented, so they can support 131072 bytes of objects in RAM, like the original Macintosh 128K, not merely 65536.

LOOM used reference counting for garbage collection, both on disk and in RAM.

Running on microcontrollers

This block-and-node system solves a lot of the problems that make bunches of microcontrollers a pain to program with even the kind of general-purpose software we had on 1970s home computers, despite nominally having tens or hundreds of times as much computational power.

Virtual memory with 256-byte blocks as pages

Using the loading of block key registers to drive a non-hardware-supported virtual-memory system should permit, for example, implementing a reasonably featureful and performant virtual-memory system on an AVR with an SPI Flash chip, perhaps with a somewhat smaller block size, like 256 bytes, and a somewhat smaller node size. At 5 megabits per second, a reasonable SPI speed, 256 bytes should take 409.6 microseconds to load or store, plus whatever overheads exist (I think about 25% on SPI itself? Plus erase time for Flash?)

Nodes should probably have 32 node keys and 32 block keys. Block keys of 32 bits in stable storage could address up to a terabyte, which is not too limiting; 128 bytes of such block keys would be 32 block keys, and it's probably reasonable to use a similar number of node keys. In RAM, such a node might shrink to 64 bytes; it probably isn't necessary to keep the 32-bit identifiers of nonresident nodes and blocks, because the extra latency to read 4 bytes from an arbitrary location in Flash is small, unlike spinning rust. (This of course suggests that the whole program of using virtual memory for such a system may be bad...)

No barrel shifters

Hardware without fast bit-shifting abilities, such as an AVR, might benefit in another way from 256-byte blocks: they could eliminate the need for a shift operation to compute the block-slot index from a flat address into an 8192-byte tree.

Multiprocessing and concurrency

A potentially interesting approach to the problem of personal computing on microcontrollers would be to share access to "disk" blocks using a MESI or similar cache-coherency protocol, with these "blocks" of 256–1024 bytes playing the role of cache lines. Then runnable processes can be migrated to whatever processor is idle, like on SMP. (You could presumably do the same thing on a Linux-like system with a SAN, running MESI at page granularity; has anybody tried this? Maybe Amoeba?)

Normally, in MESI, if a cache line is in Modified or Exclusive

state, a request from another cache to read it immediately transitions it to Shared state, guaranteeing forward progress. But there are possible alternatives; for example, you could "lock" a block or node for writing, so that attempts by other processes (on the same processor or not) to access that block or node will have to wait until you unlock it. Or, all blocks and nodes might be "copy-on-write" in the sense that each process writing to them has its own private copy, and all shared data might be immutable, with keys to new data transmitted explicitly via some kind of IPC mechanism, or some small safety valve for mutable data. Or, writes might use compare-and-swap semantics: multiple processes might be writing to the same page or node at the same time, but when the first of them commits its write, the others are aborted, either immediately or when they attempt to commit. (Presumably they can then be automatically retried.)

It's tempting to suggest that these mechanisms would make it easy to build highly concurrent shared mutable data structures, but history has not been kind to such optimistic statements.

Memory buses and hardware

The AVR itself supports SPI with I think an 8 MHz clock, but slower signals are less demanding on PCB layout. Also, some common SPI memories don't support such high speeds; according to file `jellybeans-2016`, the US$2.78 two-megabit STMicroelectronics M95M02-DRMN6TP EEPROM is only 5 MHz. Others do; the US$1.09 256-kilobit Microchip 23K256-I/SN SRAM claims 20MHz according to file `low-power-micros`, and the US$0.36 4-mebibit Winbond W25X40CLSNIG claims 104MHz. Memories cheaper than that tend to be only 400kHz I²C. I don't know how fast SD cards' SPI interfaces are, but they're also required.

If the SPI interface or whatever supports DMA, it might be feasible to run a second process for a couple thousand cycles while the first one was blocked on loading a block from external storage.

I'm not sure what the connectivity between multiple processors and the "disk" should look like; I²C tends to be only 400kbps, which would push block access times up to a spinning-rust-like millisecond level, and SPI is inherently single-master, so you couldn't connect multiple microcontrollers directly to a single memory chip. The CAN bus might work, but of course memory chips don't support it directly.

Probably you'd end up either connecting the processors into a ring, each with locally attached SPI memory, or dedicating one or two "kernel" processors to I/O arbitration, with a direct link to the memory and another direct link to each application processor.

Dynamically loading code blocks on a microcontroller

There are a few different ways a microcontroller like the AVR could handle dynamically loading code. First, it could just not do it at all, just using all this segments and nodes stuff to make it reasonably easy to run a little code with a lot of data. Second, it could dynamically load bytecode blocks into RAM and run them in an interpreter — the AVR is slow enough that this would be somewhat limiting, and it's certainly power-hungry, but this would allow relatively quick task switching. Third, it could dynamically load

machine-code blocks (whether somewhat dynamically created from bytecode or compiled ahead of time) and burn them into a "transient program area" in its Flash so it could run them, although this will limit its lifespan. Fourth, if it's a microcontroller that can run from RAM, which the AVR can't, it could just load blocks of machine code into RAM and run that.

STM32

Nowadays, as described in file `stm32`, it probably doesn't make sense to use an AVR; you should use at least a Cortex-M processor like the STM32; a 48MHz STM32F031x4 with 16 kibibytes of RAM costs US$1.30, and I think some STM32s are even cheaper than that. As bonuses, you get much lower power consumption and the ability to run code in RAM.

Copy-on-write

Copy-on-write is a little bit tricky, in that, if the same process or transaction refers to the same block via two different access paths — such as via block key register 3 and block slot 5 in some node — you probably want it to get the same version of the block. So it isn't sufficient to do the pure-functional-tree thing of "modifying" a pointer to the block by creating a new version of the node, and its parent node, and so on up to the root of the tree, because there is perhaps no tree. Instead, every time you go to load a block register, you must do a table lookup to see if the current process/transaction has a modified copy of that block, and, if not, conditionally create one. (And analogously for modifying nodes.)

The J1A

A potentially more interesting kind of microcontroller to use for this is the J1A Forth-like processor. It might be reasonable to extend it to do many of the block and node operations "in hardware", run several processors concurrently inside a single FPGA, and perhaps reconfigure other parts of the FPGA dynamically to assist with other computations.

Incremental, differentiable, and concurrent computation

(This is explored in more detail in the note on transaction-per-call systems (p. 722).)

Above I mentioned transactional memory for concurrency control as one possible application of this kind of virtual machine. The idea is that, to access the memory, you run some code inside a *transaction*, giving it some inputs when you start it, and buffer all its memory writes in a copy-on-write fashion; if the transaction runs to completion successfully, it tries to *commit*, at which point we check to see whether any block or node it read had been modified by some other transaction in the mean time. If so, we *abort* the transaction, discarding all of the buffered written data, and transparently restart it from the beginning; if not, it successfully commits, and its versions of that modified data become the active versions. It's a very simple idea, and it is commonly used to permit high levels of parallelism with very

straightforward, non-bug-prone, semantics.

As one example, you might have a piece of code that scans for an occurrence of the word "fuck" in a file, and sends an alert email if it appears, and another piece of code that modifies the contents of the file. If the scanning code happens to be reading through the file when the word "full" is overwritten with the word "sick", it might incorrectly conclude that the word "fuck" occurred, and send a spurious email, possibly getting someone fired. But if both pieces of code must run within transactions, which must commit for any externally-observable thing to happen, then any modification to the blocks read by the scanner will abort the scanner's transaction — unless it doesn't commit until after the scanner commits, in which case the scanner will see a consistent post-modification version of the file.

Thus this simple optimistic-synchronization rule makes the transactions perfectly serializable — the results are exactly the same as if all the transaction code had run in a single thread, in the order in which the transactions committed — and it guarantees forward progress. There are various kinds of optimizations that can be made to improve such a system's performance.

Long transactions

Consider, though, the situation of this scanner running on a large disk partition on which files are frequently being created and destroyed. Although the system never blocks, the scanner will never finish! By the time it comes to the end of the disk, certainly some other program will have modified some blocks it had already scanned, thus invalidating its results, and so it will be automatically restarted.

There are many ways to handle this "long transaction" problem; among them, pessimistic synchronization, nested transaction memoization, relaxed consistency, clever reordering, and spheres of influence.

Pessimistic synchronization

Pessimistic synchronization was historically the most common way to solve the problem. Instead of allowing all transactions to proceed, the scanner acquires "read locks" on every block or node it reads; if any other transaction attempts to write to such a block or node, it is paused until the scanner's transaction completes and then acquires a write lock; and if the scanner tries to acquire a read lock on a block that some other transaction already has a write lock on, the scanner blocks until the other transaction commits or aborts. The great benefits of pessimistic synchronization are that no work is ever wasted (so worst-case execution times can be computed) and no block ever need be copied. Its drawbacks include that it's easy to deadlock; it's difficult to get good scalability, since things block all the time; and, in real-time systems, it suffers from "priority inversion" where a low-priority task can hold a lock blocking a high-priority task, and a medium-priority task can then starve the low-priority and the high-priority task.

Nested transaction memoization

Nested transaction memoization is probably not something I just made up, but it works as follows. The scanner scans as follows, in a

made-up programming language with block arguments:

```
scan(word, file, start, end) = {
    return child_transaction {
        assert(word.len < blocksize)
        if (end - start < blocksize) {
            return contains(word, file, start, end)
        }

        mid = start + (end - start) // 2
        return (scan(word, file, start, mid + len(word) - 1) or
                scan(word, file, mid, end))
    }
}
```

`scan` starts by spawning a nested child transaction which can commit or abort before its parent does — by default, its abort will just retry it without affecting its parent, but once it commits, the blocks and nodes it read and wrote are added to the read and write sets of its parent, so any *later* changes to the blocks it read will then abort the parent; but there are some significant fillips we will see below.

If the area to scan is smaller than `blocksize`, then the scan is done directly, using a naïve string search or Boyer-Moore or whatever. We presume that this can be done quickly enough that, much of the time, we will finish before something else overwrites any of the data in that range, so our chance of being aborted is small.

Otherwise, `scan` proceeds by making two recursive calls to itself, which of course spawn their own nested transactions. If, during a commit, some read block is found to have been overwritten by a concurrent transaction, that transaction is then retried; but its earlier siblings remain committed.

So far, this seems to have ameliorated our problem only slightly: if something writes to the third quarter of the file while the fourth quarter is being scanned, then the transaction scanning the second half of the file will be aborted and retried. So our tiny chances of success, assuming a uniform distribution of write traffic, would seem to have improved only by a factor of 4, or less.

This is where *memoization* comes in and saves the day! Suppose that, instead of only remembering a flat list of blocks and nodes read and written by each *active* transaction in the stack, we also remember those read and written by *committed* transactions that are children or descendants of some active transaction, as well as the code and environment state needed to re-execute those transactions. Now, when we retry scanning the second half of the file, we can *revalidate* these read sets, and if they are still valid, we can "wink in" the write set without actually running any of the transaction code.

To be concrete, suppose the file consists of eight blocks (0, 1, 2, 3, 4, 5, 6, and 7), and we are retrying scanning the last four blocks because block 5 has changed. (I will disregard overlaps here.) The transaction to scan blocks 4, 5, 6, and 7 is invalid, so it begins re-executing, and the first thing it does is to spawn a child transaction to scan blocks 4 and 5. This child transaction is invalid, since block 5 has changed, so it spawns a child transaction to scan block 4. So far, memoization has

changed nothing.

But then a miracle occurs! Block 4 hasn't changed, so it doesn't need to be scanned; the `False` return value and (empty) write set of the block-4 transaction are instantly retrieved from the memo table. We proceed to spawn a child transaction to scan block 5, which has changed, so we rescan it byte by byte. It also returns `False`, and so the blocks-4-and-5 transaction returns `False`, and its parent transaction spawns a new transaction to scan blocks 6 and 7. But that transaction is also found in the memo table! So no code need execute; its (empty) write set is committed to its parent, and its `False` return value is returned.

So now our scan is complete, having scanned only the single block that actually changed and done additional O(log N) transaction revalidation work, through the beautiful gift of memoized nested transactions!

Like I said, I probably didn't just make this up. I just can't remember where I've seen it. Maybe Umut Acar's "self-adjusting computation".

Transactions that return immutable data — inevitably, newly created — poses no problem for this approach, and neither does mutating existing data. But allocating and returning new blocks and nodes does pose a difficulty for memoization, because memoization introduces aliasing! Without memoization, running the same transaction twice with the same inputs (including the state of the store) will allocate and return two separate sets of objects, but a naïvely implemented memo system would return two aliases to the same mutable objects. I think this can be solved by marking the blocks and nodes as copy-on-write, by having the memo system actually copy them before returning them, or by making them read-only.

Relaxed consistency

A common solution to the long-transaction problem is to use more relaxed isolation levels, at the risk of incorrect results. No more details will be given of this shameful practice.

Clever reordering, or MVCC

A different approach to the problem is to hope that the scanner's results can be retroactively inserted into the transaction history instead of being appended to it. This works surprisingly often; in the example code above, for instance, the scanner doesn't write any blocks — its only effect is to return a Boolean value — so it can trivially be run on any previous snapshot, and it is guaranteed that none of the transactions that committed in the interim would have had different results had the scanner transaction committed long ago.

This approach requires examining the write-set of the long transaction when it goes to commit to ensure that it's not overwriting any blocks or nodes that any transaction committed after its snapshot had read. If so, such a cyclic dependency violates serializability and thus cannot be tolerated; the long transaction must be retried anyway.

This poses the question of exactly where in history to (conceptually) insert the long transaction. But unless we are making

up a transaction log, there's no need to actually *compute* the serializable order to respect transaction isolation; it's sufficient that one exists. So it's sufficient to ensure that committing the transaction would not create a cycle in the bipartite graph of transactions and block/node versions.

Spheres of Influence

Retrying the long transaction, however, isn't the only possible solution! You could, instead, commit the long transaction and roll back and retry the already-committed *later* transactions, as long as no effects from them have escaped your rollback grasp. This is the idea of the "spheres of influence" idea from the ancient transaction processing literature, which I found in Gray & Reuter, and it's fairly similar to how the US banking system works: all numbers are provisional, subject to revision, until a few months have passed.

Incremental recomputation

Above, the use of memoized nested transactions was suggested to permit long transactions to complete successfully despite concurrent writes. But it should be apparent that this is a form of incremental computation: by memoizing results from previous partial computations, incremental changes can be accommodated efficiently, even when they're happening too fast for a batch-mode computation to run to completion successfully.

If the memo table is retained rather than being discarded as soon as the root transaction commits, it can be used to incrementalize future computations of similar transactions as well. In a database system, for example, this approach could largely transparently provide the performance functionality of standard indices, materialized views, and precomputed OLAP rollups, though perhaps not query optimization, since its very transparency complicates its use by a query optimizer.

What policy should be used to manage memo-table entries? Retaining too little will waste CPU cycles and perhaps miss real-time deadlines; retaining too much will waste RAM and perhaps also slow the system down. A unified memo-table-management system might be able to use robust heuristics to come to a reasonable global optimization solution, taking into account the observed computational cost of each transaction; but, lacking that, you probably need some way to manually specify the policy.

This memo table will suffer "false misses" under some circumstances that a smarter incremental computation mechanism might be able to take advantage of: computations that would be equivalent but end up reading the same data from different locations, for example, and in ABA cases where a location changes twice, ending with the same value it started with (a counter being incremented and then decremented, for instance).

Parallel computation with nested transactions

In the example code above, the child transaction results were used immediately; the parent transaction blocked until the child transaction was finished executing. But in many cases, including the above, it would be semantically acceptable to spawn multiple potentially concurrent child transactions, returning only a future for

the transaction's output from the initial spawn call, which is *later* blocked on — perhaps after spawning additional child transactions.

Differentiable computation with transactions

To compute a Jacobian of a computation with a small number of outputs and many inputs — the gradient, in the case that the number of outputs is one — reverse-mode automatic differentiation is much more efficient. But reverse-mode automatic differentiation requires propagating the gradient backward through the dataflow. For a short or highly regular computation, it's reasonable to materialize the whole dataflow graph in RAM at once, but not for long, iterative, and irregular computations, since the dataflow graph can contain trillions of nodes — in the limit, a node for every machine instruction executed on thousands of machines over a period of hours to months.

So the usual way to do this — if I understand correctly, which I may not — is to run the computation forward from the beginning to the end, saving its entire state on a "tape" of periodic checkpoints. If you have enough space, you can take the checkpoints close enough together that the full dataflow graph between any two adjacent checkpoints fits in RAM; then you can iterate backward through the checkpoints, building that dataflow graph in memory so as to propagate the Jacobian backward to the previous checkpoint. For the gradient case, this is theoretically about as fast as the original computation.

If that's too much space — perhaps a terabyte for 10 minutes of computation — you can thin out the tape to a logarithmically-small number of checkpoints, in exchange for a logarithmically-small (or log-squared?) slowdown. Perhaps instead of 1024 checkpoints, one per second, you might have 11 checkpoints: one from 1 second ago, one from 2 seconds ago, one from 4 seconds ago, and so on up to 1024 seconds ago. When the time comes to propagate the Jacobian from the checkpoint from 4 seconds from the end to the checkpoint from 8 seconds to the end, you first replay from the 8-seconds-from-the-end checkpoint to recreate the 6-seconds-from-the-end and 5-seconds-from-the-end checkpoints.

It should be apparent that, with manual control over the memo table, the memoized-nested-transaction mechanism described earlier can provide an efficient, space-sharing way to periodically checkpoint a computation — once we roll back everything that happened later, the *end* of each memoized transaction is a point to which we can quickly "fast-forward" from its beginning. Actually constructing the in-memory dataflow graphs and back-propagating the Jacobians, however, cannot be done by the mechanisms described earlier; they require more profound interfacing. XXX

Streams and reactive UI updates

Some of the transactions described above write their output to the block and node store. Others, though, are merely queries that return a value without mutating anything, at least not anything externally visible. By recording which blocks and nodes are read by a query transaction, as the transaction system does, we can automatically determine when query results have become out of date; the memoization mechanism described above provides a reasonably

efficient way to support polling, for example for screen updates, but if writing to a block or node can trigger an asynchronous invalidation notification (which can be responded to by repeating the query if desired), that may have lower latency, have higher throughput, or use less energy under some circumstances.

Nothing in the system design limits these approaches to read-only queries; they can apply equally well to "queries" that mutate the block and node store in a persistent way (as opposed to using nodes and blocks they allocate ephemerally as temporary storage, or allocate and then return). Indeed, if those queries write to no nodes and blocks that they also read and did not allocate, they can be mechanically guaranteed to be idempotent. (But see above about memoization introducing aliasing.)

Progress bars on such transactions probably cannot be provided through transactional mechanisms, since they have dataflow from uncommitted transactions. So, as with differentiable programming, metatransactional mechanisms are needed.

Modular blocking and composable memory transactions

The *Composable Memory Transactions* paper, which I need to reread, explains how to use an optimistic transactional memory with nested transactions, like the above, to support blocking patterns of communication by adding two more functions, `retry` and `orElse`.

`retry` conceptually simply aborts the current transaction, causing it to be automatically retried. But if the system responds by beginning to run the same transaction code again with the same inputs and the same state of the store, it would simply deterministically reach `retry` a second time, and so on, busy-waiting. So a more reasonable system, like the one they actually implemented, waits to retry the transaction until the store has changed — specifically, until some transactional variable that it had read before invoking `retry` has changed.

The `orElse` operator provides a way to recover from such failures, by composing two alternative child transactions into a larger child transaction. If the child transaction that is its left argument fails because of invoking `retry` (though not because of a conflicting write by a concurrent transaction), then control flows to the alternative transaction that is its right argument. If that transaction also fails, then the transaction resulting from `orElse` fails.

Thus `retry` provides a way to convert a polling interface, such as reading a transaction variable to see if something is ready, into a blocking interface, while `orElse` provides a way to either combine two sources of blocking into an alternative source that only blocks while both sources are blocking, like Unix `select(2)`, or to convert a blocking interface into a polling interface (by providing a second alternative that does not block).

Precisely the same interface would work on top of nodes and blocks.

The requirements of transactional systems limit the applicability of familiar interprocess communication.

For example, you could try to implement a byte pipe between two

transactions by a memory block whose first two words contain beginning and end pointers into a ring buffer that is the rest of the block. A transaction could attempt to add bytes to the ring buffer and return the number successfully written, blocking with `retry` if there is not room, or to remove bytes from it and return them, blocking with `retry` if there are no bytes to remove. If a pipe-reader and a pipe-writer try to mutate the block at the same time, one will succeed and the other will fail at first, then retry and succeed. So, at first, this sounds like a standard Unix pipe.

But, if the pipe-reader's parent transaction is aborted, the pipe-reader's modifications to the pipe block will be rolled back. As long as the two share a parent transaction, then all the pipe-writer's modifications will, too; and, until they do share a parent transaction (that is, until any levels of transactions separating them from their lowest common ancestor transaction have committed), their communications won't be visible to one another — the writer can't unblock the reader or provide it bytes, and the reader can't unblock the writer.

Snapshot debugging

Debuggers don't.

What debuggers do is provide programmers visibility into a program's internal state in order to can formulate and test hypotheses until they diagnose a bug. Traditionally debuggers do this by providing three fundamental services: memory and register inspection --- letting programmers go inside the program "spatially"; breakpoints and single-stepping --- letting them "go inside" the program temporally; and mutation --- letting them change the program's state while it's stopped. Essentially by scripting these, more sophisticated facilities are commonly built, like "stepping over" a subroutine call, disassembly, source-code display, displaying the call stack and local variables, injecting code into the program, and so on.

(Single-stepping can be implemented by scripting breakpoints, simply by repeatedly placing and removing breakpoints, and this is a common way to support single-stepping on platforms with no native single-stepping support. Implementing breakpoints by scripting single-stepping is also possible but generally impractically inefficient.)

However, with imperative programming languages, these three basic facilities are frustratingly inadequate, because very commonly by the time the programmer sees that some memory location has a wrong value in it, the code that placed that value there is long gone. With sufficient patience and meticulosity, repeated re-executions of a program can eventually find how the location was changed by using inspection and breakpoints, but this is an extreme measure, and often it must be repeated more than once. So there are a couple more basic facilities commonly provided by modern debuggers: watchpoints and reverse execution.

Breakpoints stop the execution of the program when it executes a particular instruction; watchpoints, by contrast, stop its execution when it modifies a particular memory location. This can be provided inefficiently by scripting single-stepping and inspection --- after each single step of the program, the debugger's script inspects the memory

location to see if its value has changed --- but on many platforms there are more efficient ways to solve that problem, either using virtual-memory hardware or using special CPU registers devoted to implementing watchpoints.

Watchpoints allow a single re-execution to find how a given memory location was changed, but reverse execution improves on this. OCaml's debugger was the first debugger I know of to provide reverse debugging, which it did by exploiting Unix's fork(2) to make copy-on-write snapshots of the program state; XXX

Topics

Slide rule addition

Kragen Javier Sitaker, 02020-06-22 (3 minutes)

Slide rules can't add and subtract. Could they?

If $a \neq 0$, then $a + b = (1 + b/a)a$. Suppose our slide rule reads with a precision of ±0.2%; then if $b/a < 0.002$, we can just round this to a, and if $b/a > 500$, we can just round it to b. But in between, when they're of roughly similar magnitudes, we might want to use this to calculate a decent approximation of the sum.

In Logarithm Land, we have:

$$\log(a + b) = \log(1 + 10^{\log b - \log a}) + \log(a),\ a \neq 0$$

We can add the ability to evaluate this to a slide rule as follows. Define $j(c) = \log(1 + 10^c)$. Add three j scales to the body of the slide rule: one with marks at $\log(1 + 10^c)$ for $1 \leq c \leq 10$, one with marks for $10 \leq c \leq 100$, and one with marks for $100 \leq c \leq 1000$.

To add two numbers of the same sign, then, take the larger one as b. Use the C and D scales to compute $\log b - \log a$ as a position on the D scale; move the cursor to it. Look on the appropriate j scale to read the numerical value of $(1 + b/a)$. Now use the C and D scales to multiply that numerical value by a, which perhaps you still have encoded in the position of the slide.

Working backwards from there, we want the slide to be in a position that encodes multiplying by a, which means that the D-scale index should be aligned with a on the C scale. This means that b needed to be on C originally.

So the procedure is: choose b to be the larger of the two summands, exchanging them if necessary. Align the slide to a on the C scale. Move the cursor to b on the C scale; its position on the body is now $\log b - \log a$, so the cursor on the D scale now indicates b/a. Look up the numerical value of $1 - b/a$ on the appropriate j scale with the cursor. Move the cursor to that numerical value on the D scale. Now read $a + b$ on the C scale with the cursor.

I'm not sure if there's a similarly convenient approach using the CI and DI scales, or if there's a way to use that same j scale for subtraction, or if it works better to use $b < a$ (could that give you fewer j scales?).

Peter Alfeld reports that Jeff Weiner reports that the Pickett Microline 115 and the Pickett 901 rules *can* add and subtract, but it turns out that those just have linearly-ruled X and Y scales; they can't add and subtract numbers found on the standard scales like C, D, A, and B, nor do their sums and differences appear there.

Now, of course this is not very useful if your precision is ±0.2%: you only have three sig figs, and adding two three-digit numbers in your head isn't that hard. You could imagine a higher-precision slide rule using verniers, finer details, and/or larger dimensions, perhaps folded helically. This approach might then be more useful.

Topics

Hacker calendar

Kragen Javier Sitaker, 02020-06-28 (updated 02020-12-03) (15 minutes)

The humans like to memorialize dates by holding annual celebrations. What kinds of dates would a hacker culture memorialize?

I'm putting together a date table here in a sort of cuckoo-hash fashion: most dates commemorate individual hackers, and for most of the hackers the relevant known dates are their birthdate and their death date. When there is a collision, I can usually move the person in question to a different relevant date: their death date if they're listed at their birthdate, or vice versa.

So, for example, to insert Laplace, Hao Wang moved from May 20 to May 13, allowing Jean Sammet to move from March 23 to May 20, freeing up March 23 for Laplace. Laplace's death date, March 5, was occupied by William Oughtred, who died June 30, which is already quite full with two difficult-to-move events: the feast of Ramon Llull and the release of OpenGL.

• January 1, 1992: Grace Hopper died (born December 9, 1906)
• January 4, 1643 (O.S. December 25, 1642): Newton born; died March 31, 1727 (O.S. March 20, 1726), published Principia July 5, 1687?
• January 8, 1642: Galileo Galilei died after 8.5 years of house arrest (born February 15, 1564, sentenced by the Inquisition June 22, 1633(?))

• January 11, 2013: Aaron Hillel Swartz committed suicide to escape government persecution (born November 8, 1986)
• January 14, 1901: Tarski born (died October 26, 1983)
• January 15, 2001: the founding of Wikipedia
• January 19, 1912: Leonid Vitaliyevich Kantorovich (Леони́д Вита́льевич Канторо́вич) born (died April 7, 1986). Published "Математические методы организации и планирования производства" in 1939.
• February 7, 1990: Alan Perlis died (born April 1, 1922)
• Feburary 8, 1920: Bob Bemer born (died June 22, 2004)
• February 11, 1897: Emil Post born (died April 21, 1954)
• February 13, 1258: the destruction of the House of Wisdom (بيت الحكمة) by Mongol soldiers in the Siege of Baghdad (though that was only the first day of a week of destruction)
• February 17, 1600: Giordano Bruno, who first proposed that the stars were distant suns, burned at the stake for, among other things, teaching reincarnation and possessing the writings of Erasmus.
• February 23, 1855: Gauss died (born April 30, 1777)
• February 24, 1709: Jacques de Vaucanson born (died November 2, 1782)
• March 1, 1990: the Secret Service raided Steve Jackson Games for publishing GURPS Cyberpunk
• March 1, 86 BCE: Sulla sacked Athens, having burned Plato's Academy. (However, this needs to be corrected for calendar

alignment.)
• March 4, 1959: John McCarthy (born September 4, 1927; died October 24, 2011) published Artificial Intelligence Project Memo 8, "Recursive functions of symbolic expressions and their computation by machine", describing the LISP language he and Steve "Slug" Russell (born 1937, still alive) had developed on the 704.
• March 5, 1574: William Oughtred born (died June 30, 1660)
• March 7, 1917: Betty Holberton born (died December 8, 2001)
• March 11, 1890: Vannevar Bush born (died June 28, 1974)
• March 18, 1905: Einstein sends his paper on the photoelectric effect, for which he received the Nobel Prize, to *Annalen der Physik*, which published it on June 9. Einstein was born on March 14, 1879, and died on April 18, 1955. In his *annus mirabilis* 1905, he published relativity and some other stuff, including his thesis (April 30). *Annalen der Physik* received the relativity paper on June 30 and published it September 26.
• March 20, 1956: Kurt Gödel (born April 28, 1906, died January 14, 1978) wrote to John von Neumann, posing essentially the P vs. NP problem
• March 21, 1768: Fourier born (or, alternatively, December 21, 1807, he presented his paper "on the propagation of heat in solid bodies")
• March 23, 1749: Laplace born (died March 5, 1827); published Bayesian probability theory in 1812
• March 25, 1914: Norman Borlaug born, who saved a billion lives with dwarf wheat (died September 12, 2009)
• March 31, 1596: Descartes' birth (died February 11, 1650)
• April 7, 1761: Thomas Bayes died (birthdate unknown)
• April 9, 1806: Isambard Kingdom Brunel born (died September 15, 1859)
• April 14, 1935: Emmy Noether died (born March 23, 1882)
• April 15, 1707: Euler born (died September 18, 1783)
• April 25, 1903: Kolmogorov born (died October 20, 1987)
• April 26, 1920: the death of Srinivasa Ramanujan (born December 22, 1887)
• April 29, 1911: founding of Tsinghua University (then 清華學堂, now 清华大学)
• April 30, 1995: the NSFNet backbone shut down, ending the NSFNet Acceptable Use Policy which prohibited most for-profit activity on most of the internet.
• April 30, 1916: Claude Shannon born (died February 24, 2001) (Shannon Day was celebrated April 30, 2016)
• May 2, 1519: Leonardo da Vinci died (born April 14 or 15, 1452)
• May 7, 1711 (O.S. April 26): David Hume born. (Died August 25, 1776)
• May 10, 1933: nationwide book burning by Nazis, following the German "Law for the Restoration of the Professional Civil Service", which on April 7, 1933 eliminated all Jewish and Communist public employees, including professors, with some exceptions;
• May 11, 1918: Richard Feynman born (died February 15, 1988)
• May 13, 1995: Hao Wang (王浩) died (born May 20, 1921)
• May 17, 1902: the discovery that the Antikythera Mechanism had gears (probably brought up July 1901)
• May 20, 2017: Jean E. Sammet died (born March 23, 1928)
• May 31, 1832: Évariste Galois killed in a duel (born October 25,

1811)
• June 7, 1954: Alan Turing committed suicide (born June 23, 1912)
• June 9, 597: St. Columba died (born December 7, 521). Legend has it he fought the battle of Cúl Dreimhne in 561 to defend his right to copy St. Finnian's psalter.
• June 16, 1915: John Tukey born (died July 26, 2000)
• June 22, 1910: Konrad Zuse born (died December 18, 1995)
• June 27, 1831: Sophie Germain (Monsieur Antoine-Auguste Leblanc) died
• June 28: Tau Day
• June 30, 1992: OpenGL released
• June 30, feast day of Ramon Llull, whose works were prohibited by the Spanish Inquisition (traditional death date June 29)
• July 1, 1646: Leibniz's birth in Leipzig (O.S. June 21) (died November 14, 1716)
• July 10, 1856: Nikola Tesla (Никола Тесла) born (died January 7, 1943)
• July 16, 1945: the Trinity event
• July 17, 1912: Poincaré died (born April 29, 1854)
• July 20, 1969: Apollo 11 lands humans on the moon for the first time at 20:17 UTC
• July 25, 1926: Ray Solomonoff born (died December 7, 2009)
• August 6, 2002: Dijkstra died (born May 11, 1930)
• August 8, 1900: David Hilbert (born January 23, 1862; died February 14, 1943) presents ten of his 23 famous problems at the International Congress of Mathematicians in Paris.
• August 9, 1927: Marvin Minsky born (died January 24, 2016)
• August 12, 2013: Warren Teitelman died (born 1941)
• August 17, 2004: Xiaoyun Wang (王小云), Dengguo Feng, Xuejia Lai, and Hongbo Yu of Shandong University published their break of MD5.
• September 5, 1977: launch of Voyager 1, omitting "Here Comes the Sun" for copyright reasons
• September 9, 1941: Dennis Ritchie born (died October 12, 2011)
• September 17, 1826: Riemann born
• September 26, 1983: Stanislav Yefgravovich Petrov (born September 7, 1939; died May 19, 2017) refused to nuke the US when a radar system malfunctioned, thus saving human civilization
• September 27, 1983: The inauguration of the GNU Project
• September 30, 1993: WSMR-SIMTEL20, one of the greatest libraries of software in the world, was shut down at 1600 hours Mountain Daylight Time and its 165,000 files destroyed, following a copyright lawsuit from the Louis E. Wheeler Co., as reported in Network World, January 16, 1995 ("Army gets caught in software piracy firestorm").
• October 4, 1957: launch of Sputnik 1
• October 18, 1931: Thomas Edison died (born February 11, 1847)
• October 29, 1998: the sale of the Archimedes Palimpsest
• the fourth month of the inundation season: when Ahmose wrote the Rhind Papyrus
• October 30, 1961: Tsar Bomba test
• October 31: octal Newtonmas — oct 31 = dec 25
• November 2, 1988: the helminthiasis of the internet with the Morris worm

• November 6, 1717: J. S. Bach imprisoned (or March 31, 1685: J. S. Bach born, or July 28, 1750, Bach died (Episcopal feast day))
• November 8, 1848: Gottlob Frege born (died July 26, 1925; published the Begriffsschrift in 1879) although he was an anti-Semite
• November 11, 1918: 11:00: Armistice Day
• November 19, 2020: USA's National Science Foundation decides to demolish the Arecibo Observatory, the largest single-reflector telescope until 2016, and crucial to predicting asteroid impacts; it was irreparable and collapsing after being damaged in hurricanes over previous years, following decades of decay. It was built 1960–63.
• November 26, 1894: Norbert Wiener born (died March 18, 1964)
• November 30, 1858: Jagadish Chandra Bose, who invented semiconductor diodes and submillimeter light, born (died November 23, 1937)
• December 1, 1975: the publication of the first version of Scheme as the paper "Scheme: an interpreter for Extended Lambda Calculus", 1975, AIM-349
• December 7, 1873: Cantor sends Dedekind his proof of the uncountability of the reals
• December 8, 1864: George Boole died (born November 2, 1815)
• December 9, 1968: Doug Engelbart's Mother of All Demos (born January 30, 1925; died July 2, 2013)
• December 10, 1815: Augusta Ada Byron (later Lovelace) born (died November 27, 1852). In September 1843 her translation of Luigi Menabrea's 1842 French notes on Babbage (born 1791)'s Analytical Engine were published in *Scientific Memoirs*, including the first computer program in Note G of its "notes by the translator"; translated into C by Sinclair Target in 2018
• December 17, 1706: Gabrielle Émilie Le Tonnelier de Breteuil, Marquise du Châtelet, who arguably discovered energy, was born (died September 10, 1749)
• December 23, 1790: Jean-François Champollion born (died March 4, 1832), deciphered the Egyptian demotic script in 1806 and, while awaiting trial for treason, the hieroglyphs in 1822.
• December 28, 1903: John von Neumann born (died February 8, 1957)

Things I don't know dates of:

• Lu Ban (魯班) (dates unknown)
• Shandong University 山东大学 founded (dates unknown)
• Zhang Heng (張衡) (dates unknown)
• Su Song (蘇頌) (dates unknown)
• Guo Shoujing (郭守敬) (dates unknown)
• Sunshu Ao (孫叔敖) (dates unknown)
• Shen Kuo (沈括) (dates unknown)
• Yī Xíng (一行) (dates unknown)
• Liu Hui (劉徽) (dates unknown)
• Mozi (墨子) (dates unknown)
• Zu Chongzhi (祖沖之) (dates unknown)
• Heron of Alexandria (dates unknown)
• Eudoxus of Cnidus (dates unknown) (when was his eclipse?)
• Dakṣiputra Pāṇini (dates unknown)
• Akṣapāda Gautama (dates unknown)
• Ahmad, Muhammad and Hasan bin Musa ibn Shakir, the Banu

Musa who wrote the كتاب الحيل Kitab al-Hiyal (dates unknown) in the House of Wisdom
• the first solar power plant entered production in Egypt (Frank Shuman's "Solar Engine One" in Maadi, 1912-1913)
• Chomsky hierarchy? https://doi.org/10.1016%2FS0019-9958%2859%2990362-6 1959 "On certain formal properties of grammars" (though Chomsky himself is still alive)
• Aristotle (dates unknown)
• Brahmagupta (dates unknown)
• Something about Knuth? The publication date of TAOCP volume 1?
• Stephen A. Cook's SAT paper establishing NP-completeness? http://www.cs.toronto.edu/~sacook/homepage/1971.pdf https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.406.395
• something to memorialize the Mohists and the other scholars who fell to Qin Shi Huang?
• something to memorialize the Library of Alexandria?
• something to celebrate Euclid
• al-Khwarizmi's book https://en.wikipedia.org/wiki/The_Compendious_Book_on_Calculo ation_by_Completion_and_Balancing?
• something to celebrate Aryabhata and the Aryabhatiya?
• publication of the break of Merkle's knapsack algorithm (date unknown)
• something about the Nine Chapters on the Mathematical Art
• the release of the first version of Haskell (dates unknown)
• July 1562: the burning of the Maya codices by Bishop Diego de Landa (date unknown)
• the publication of Alice in Wonderland
• the founding of Sun
• the going on sale of the Altair 8800
• founding of the University of Leipzig
• burning of the last copy of the Yongle Encyclopedia
• Cornelis Drebben
• Jaquet Droz?
• Inauguration of the EDVAC?
• Alexander Humboldt?
• the defeat of Kasparov
• something about Prometheus
• the rescue of the library of Timbuktu
• Mozart?
• Haskell released
• THERAC-25
• Ariane 5
• Chernobyl
• Lavoisier's execution
• Linux announced
• 4.4BSD-Lite released
• Jean Bartik? May have made ENIAC a stored-program computer.
• Kalashnikov?
• April 1962: Spacewar!
• James Watt?
• the Cultural Revolution

• McCarthyism
• Alonzo Church?
• Ctesibius?
• Genesis Block mined?
• Morse's first telegraph message sent?
• Santos Dumont's first flight?
• Fred Fish?

Of secondary importance:

• February 13, 1805: Dirichlet born
• October 30, 1632: (O.S. October 20) Christopher Wren born (died March 8, 1723 (O.S. February 25))
• Rudolf Carnap
• August 6, 1667 (O.S. July 27): Johann Bernoulli born (died January 1, 1748)

Rejected:

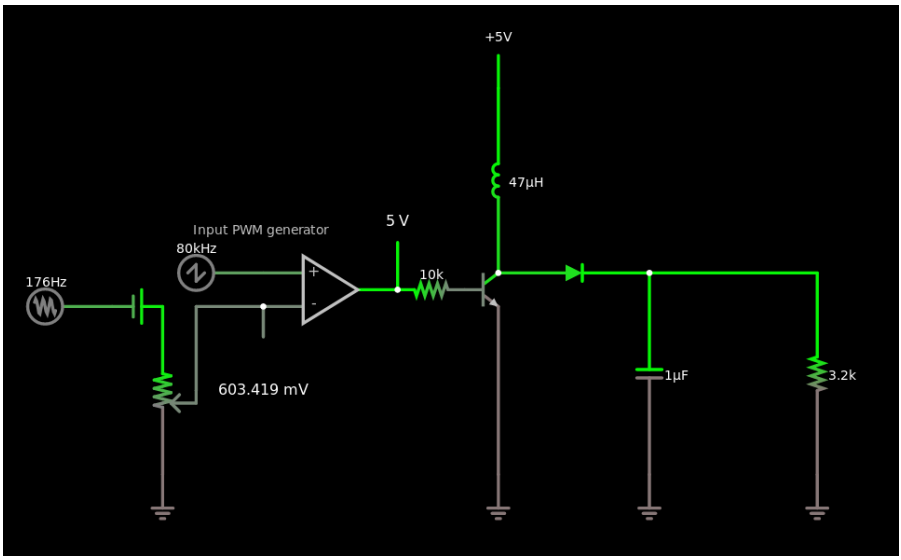
• Kathleen Booth? no, she's still alive
• Michael Rabin (no, he's still alive)
• September 10, 1839: Charles Sanders Peirce born (died April 19, 1914) (but he supported racism-based slavery, and wasn't as important as other significant hackers in the history of logic)
• Ed Fredkin (no, he's still alive)
• Ivan Sutherland (no, he's still alive)

Topics

Trying to drive a speaker with a buck converter

Kragen Javier Sitaker, 02020-06-29 (4 minutes)

This circuit doesn't work:




```
$ 1 5e-9 5.692113234615338 60 5 43
R 416 432 352 432 0 4 80000 2.5 2.5 0 0.5
170 272 464 208 464 1 20 50000 2.5 0.01
v 272 464 320 464 0 0 40 2.5 0 0 0.5
a 416 448 544 448 9 5 0 1000000 0.6034186806095027 1.5999999949944597 100000
0 544 448 544 384 1
x 349 395 478 398 4 12 Input\sPWM\sgenerator
0 416 464 416 544 1
174 320 464 352 624 0 1000 0.8762000000000001 Input attenuator
g 320 624 320 656 0
w 352 544 352 464 0
w 352 464 416 464 0
t 608 448 640 448 0 1 -3.5804133275459917 0.692890403465214 100
w 640 464 640 624 0
g 640 624 640 656 0
l 640 432 640 256 0 0.000047 -0.04307177750648704
r 608 448 544 448 0 10000
R 640 256 640 208 0 0 40 5 0 0 0.5
c 784 432 784 624 0 0.000001 6.171399381253317
g 784 624 784 656 0
w 784 432 944 432 0
r 944 432 944 624 0 3200
g 944 624 944 656 0
d 640 432 784 432 2 1N5711
o 6 1024 0 4354 8.183476519740355 9.765625000000001e-155 0 1 input\sv
o 20 1024 0 4099 10 0.8 0 2 20 3
```

The first problem is that this isn't a buck converter as intended; it's a boost converter!

Here the 3200Ω resistor represents an 8Ω speaker seen through a 20:1 audio output transformer. You put an AC volt across it, you get 310 µA, which on the output is supposed to be 50 millivolts and 6.2 mA, which is 8Ω. The quadratic way transformers transform impedances always confuses me!

The input comparator generates a PWM signal that shorts the coil-diode junction to ground periodically through an NPN transistor. (Probably using a MOSFET would be a better idea.) The idea is that when the coil is shorted to ground, it builds up energy in the form of a magnetic field with a progressively growing current, and then when the transistor turns off, that energy is delivered to the load, with the current progressively dropping until the next cycle. At the extreme where the transistor is always off, the 5V power supply is simply connected through to the output through a diode and an inductor. At the extreme where the transistor is always on, the current through the coil and the transistor will progressively increase until the transistor burns out.

Near that extreme, there is a short time when the transistor turns off, and the enormous current through the coil must flow instead to the load, charging up the capacitor very rapidly. This can produce, in theory, an arbitrarily high voltage.

Managing voltages on the order of 5 volts, cycle times on the order of 10 µs, and currents on the order of 1 mA, we probably want an inductance that produces about 5 volts with a slew rate of about 1 mA per 10 µs. This would be about 50 mH, three orders of magnitude larger than the 47µH inductor I have here. At these speeds and impedances, this inductor looks like a wire; inductive reactance is just ωL, so a 50 mH inductor would be 25 kΩ, while this 47µH inductor is 24Ω, which is a wire in comparison to the 3200Ω load.

The output capacitor serves to keep the voltage from rippling too much. Again, in the 5V 10µs 1mA regime, we want a capacitor that is large compared to one that would discharge completely in that time, which would be 2 nF. The 1µF cap I have in there will take 500 cycles to discharge, or charge. So probably I want something on the order of 47 nF.

Changing the components to these values does indeed make the circuit sort of work, although it's still a boost converter.

Topics

Using Numpy for non-numerical computation: what would a good example be?

Kragen Javier Sitaker, 02020-06-29 (updated 02020-06-30)
(3 minutes)

I saw someone saying they'd never needed to use Numpy, and so never learned it, because it was for a specific use case that wasn't theirs. This seemed to me like maybe they didn't appreciate its versatility, so I thought I'd try out some non-numerical or semi-numerical computation with Numpy, and maybe Pandas.

Counting words in a string

The usual approach for counting words in Python, of course, is `len(s.split())`. But we can do things with Numpy too. First, let's get some text into a Numpy array:

```
>>> import numpy as np

>>> text = "This isn't anything more than a text string, with some words.  Let's o
ocount them all"
>>> ta = np.array(list(' ' + text))
```

Now let's find the spaces and the non-space things following them:

```
>>> sp = (ta == ' ')
>>> ta[1:][sp[:-1] & ~sp[1:]]
array(['T', 'i', 'a', 'm', 't', 'a', 't', 's', 'w', 's', 'w', 'L', 'c',
       't', 'a'],
      dtype='|S1')
>>> ''.join(ta[1:][sp[:-1] & ~sp[1:]])
'TiamtatswswLcta'
```

That seems to have worked; we can count the words just by summing the boolean vector:

```
>>> (sp[:-1] & ~sp[1:]).sum()
15
```

Let's use this approach to count the words in the King James Bible:

```
>>> b = np.memmap('bible-pg10.txt', '|S1', 'r')
>>> sp = (b == ' ')
>>> (sp[:-1] & ~sp[1:]).sum()
749219
>>> len(open('bible-pg10.txt').read().split())
824146
```

Hmm, what happened?

```
$ wc bible-pg10.txt
 100222  824146 4452069 bible-pg10.txt
```

So wc agrees with .split().

```
$ grep -P '^\S' bible-pg10.txt | wc
  74927  823934 4400033
```

So it seems like there are 74927 lines beginning with a non-whitespace character, which precisely accounts for the difference. We want to treat newlines as whitespace as well, and also, if the file starts with a word (which it does), we want to count that word too.

Current Numpy contains an `isin` function to test set membership, but my old version doesn't. No matter! We can use .any() as a substitute:

```
>>> sp = (b == [[' '], ['\n'], ['\r']]).any(axis=0)
>>> b[sp[:100]]
memmap([' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '\r', '\n', '\r', '\n', ' ',
        ' ', ' ', ' ', ' ', ' ', ' ', ' '],
      dtype='|S1')
>>> ''.join(b[:100])

'\xef\xbb\xbfThe Project Gutenberg EBook of The King James Bible\r\n\r\nThis eBoo
ok is for the use of anyone anywhe'
>>> ws = (sp[:-1] & ~sp[1:])
>>> ws[0] = True
>>> b[ws[:100]]
memmap(['\xef', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '\n', ' ', ' ', ' ',
        ' ', ' ', ' ', ' ', ' '],
      dtype='|S1')
>>> ''.join(b[1:][ws[:100]])
'\xbbPGEoTKJBTeiftuoaa'
```

That seems pretty reasonable. So how many words are there?

```
>>> ws.sum()
824146
```

Good! That takes 307 ms on this laptop, says IPython:

```
%%time
b = np.memmap('bible-pg10.txt', '|S1', 'r')
sp = (b == [[' '], ['\n'], ['\r']]).any(axis=0)
ws = (sp[:-1] & ~sp[1:])
ws[0] = True
print(ws.sum())
```

How fast is the standard approach?

```
%%time
len(open('bible-pg10.txt').read().split())
```

147 ms, twice as fast! So this is not a very compelling example!

Topics

Modelica notes

Kragen Javier Sitaker, 02020-07-06 (updated 02020-07-07)
(9 minutes)

I haven't found a good explanation of what Modelica is, so here's my effort. It's a *multi-domain* textual language for *numerical simulation* of models of *continuous-time* systems *hierarchically* composed of *lumped* elements whose behavior is specified through *acausal equations*. It is particularly suited for designed physical systems such as machines or chemical plants. (Sometimes the term "cyber-physical systems" is used to emphasize the importance of control systems.) Also, it has aspects to facilitate graphical display of the models as block-and-line diagrams, and comes with a large library of standard components. Its fundamental basis is ordinary differential algebraic equations of finite dimensionality, but it also supports hybrid simulation with discrete events.

Here's my effort to explain my understanding of what this means, bearing in mind that I've never used Modelica, so some of this may be laughably wrong.

Multi-domain

The aspects of interest of a machine such as a bicycle commonly span domains such as the mechanical, electrical, thermal, hydraulic, and even digital, with interactions between them. A bicycle may have mechanical aspects such as the transmission of power from the pedals to the wheels through the sprockets, electrical aspects such as the generation of power for lights from wheel hub generators and its storage in batteries, thermal aspects such as the generation of heat in a braking disc, hydraulic aspects if the braking system is hydraulic, and digital aspects if there are sensors like a speedometer or actuators like a brushless hub motor.

Modelica can describe models that cross these different domains; however, typically each component only exists in one or two domains. For example, a copper pipe might have mass, three-dimensional orientation, electrical resistance, thermal mass, and hydraulic roughness and diameter, thus crossing several domains; but typically is only modeled in one or two of these domains. I think this is because Modelica simulators typically refuse to simulate if there are some variables whose values they cannot determine, so using such a multi-domain component is a nuisance, since it obligates you to describe all the domains at once.

Consequently there is, for example, a Resistor component in the standard library, and also a cross-domain HeatingResistor component, which has a temperature and a thermal port. I think that if you use the HeatingResistor component you end up having to connect the thermal port.

This also brings up a potentially larger issue, which is the closed-world assumption of Modelica models: more or less inevitably they assume that you have included all the important aspects in your model.

Numerical simulation

Given a model written in Modelica, implementations such as OpenModelica can run one or many simulations of the model over some time period from specified initial conditions. These simulations can be quite precise; for example, the standard Berkeley SPICE3 set of components, is included in the standard library, and its accuracy has been validated to some extent against SPICE3 itself.

However, there are a number of other things you might want to do with a model other than simulate it. You might want to do "model identification" to estimate the model's parameters from measurements of a real system; you might want to validate some behavior of the model for all possible scenarios rather than just one (for example, showing that the model is unconditionally stable); you might want to optimize the model to find out what parameter settings are in some sense "best"; you might want to rigorously prove that two models are equivalent, or show how they differ; and so on. As far as I can tell, Modelica implementations do not typically support these other possible operations, or give them much lower priority. Even operations on the differential-equation system other than initial-value problems are generally unsupported.

The nature of the simulation is fundamentally numerical; although discrete-time systems are supported, it's not really the focus of Modelica, and I'm not clear that you're going to be able to write a compiler or something in it. I don't think there's any way to create new objects during the course of the simulation.

Continuous-time

Fundamentally Modelica reduces your model, or at least the continuous-time part of it, to a set of differential algebraic equations which it can then numerically integrate with methods like Runge–Kutta. So most of your model variables theoretically take on an infinite number of values during the simulation. This separates your model from the solver, allowing you to apply different solvers to the same model.

Hierarchical

A Modelica model can be used as an element in another, larger model; many models consist only of interconnected smaller models, containing no explicit equations of their own. So, for example, a hydroponic system might contain an irrigation system as an element, which contains pumps, pipes, and a feedback control subsystem; the control subsystem might contain a power supply, sensors, a microcontroller, and actuators; the power supply might contain diodes, inductors, optoisolators, transformers, resistors, and a buck controller; the buck controller might contain transistors, diodes, and resistors. Modelica can in theory model at all of these levels, reducing them all to a single system of differential algebraic equations for simulation.

I haven't quite seen any Modelica models with that level of detail, but I've seen people describe a number that come close to it.

Lumped

Although Modelica models are continuous in time, they are not continuous in space; the elements of the system are idealized to points. So Modelica cannot model a continuous heat distribution throughout a tank of water, a waveform moving through an electrical transmission line, or the stress distribution in a strut, although if you discretize these things yourself you can get it to simulate the discretized approximation.

In particular, I think there isn't even a way in Modelica to model a delay of a continuous-time signal, such as you might get from an improperly terminated cable.

(However, I've seen people simulating, for example, the water-hammer effect in a pipe with the proprietary Modelica simulator SimulationX; I assume they're using a discretized approximation of the pressure waves.)

Acausal equations

Modelica models are composed of (possibly differential) equations rather than causal relationships in which effects result from causes; the standard example of this is the equation $V = IR$ for a resistor, from which you can calculate the current if you know the voltage, the voltage if you know the current; if you know neither *a priori*, you may still be able to incorporate it into a system of equations that eventually allow you to determine both, the simplest example being two resistors in series with a battery.

This is pretty essential not only in circuit analysis but in a variety of different domains: mechanical force and displacement are similarly interdependent in a steady-state spring, as is flow rate and pressure drop in a hydraulic system, for example.

Etc.

Modelica supports compile-time units checking, but I'm not yet clear that its units support extends to full vector-space dimensional analysis.

Could you get faster simulation results with interval arithmetic, affine arithmetic, or especially reduced affine arithmetic?

I wish there was a way to describe an object like a hollow copper cylinder of such-and-such dimensions and have all of its properties potentially available — but only if you ask for them. For example: electrical resistance, flow resistance, cost, mass, stiffness, tensile strength, effective RF resistance with skin effect, volume, temperature, thermal insulation, and so on. I don't think there's a way to do this kind of thing in Modelica itself, but you could do it in a higher-level language that compiles to Modelica.

The other thing is that Modelica suffers a bit from the assembly-language disease where you have to invent a name for every intermediate value, worsened by the COBOL problem of DATA DIVISION. PROCEDURE DIVISION. A model or other class is divided into a section of variable declarations (which can instantiate other classes used as components — a circuit model, for example, might instantiate resistors and op-amps) and a section of equations, which can include connections between components. (There are

some other miscellaneous sections that are sometimes present as well.) So, for example, in a circuit model you must give a name to every circuit component, even if it's something like "R37". The standard rebuttal to this complaint is that you should be using the graphical model editor anyway, which I do not find convincing.

Topics

Ultra machining

Kragen Javier Sitaker, 02020-07-06 (updated 02020-07-18)
(5 minutes)

I've been watching a lot of videos of people explaining and demonstrating how they machine metal parts with modern CNC lathes and mills, as well as more exotic tooling like wire EDM and SLS machines. It occurred to me that they're still mostly not taking much advantage of the possibilities of what CNC machines could do.

First, a lot of vertical milling is done with cylindrical endmills. Cylindrical endmills have to be unreasonably long and slender in order to be able to reach a reasonable depth. For a given end diameter, tapered endmills offer a much better tradeoff of reach versus rigidity than cylindrical endmills do. But they have the disadvantage that, with three-axis milling, they don't permit milling vertical walls. But that's obviously fixable with five-axis milling.

However, it's even possible with four-axis milling, if your axes are Y, Z, A, and B. The X-axis can be fixed at the center of rotation of the B-axis; it need not move. This also eliminates the heavy and finicky serial-kinematics prismatic joint upon prismatic joint of a standard gantry, which I think may be a leftover from manual machining.

We pay way too much for rigidity. Machine tools are conventionally built out of iron and steel, which are pretty rigid, but also pretty expensive — even to buy, but especially to shape. Other materials are nearly as rigid and a hell of a lot cheaper, so you can use far more of them. Above I mentioned granite, but other candidates include concrete, brick, and even plaster. Building machine tools out of concrete is a fascinating and underexplored area.

Aside from that, I think rigidity is perhaps overrated. Hermle is building their best machines, not out of granite, but out of a granite-epoxy composite, as I understand it because it damps vibration better. In manual machining, rigidity (and taking up the backlash) was the only way to get an accurate reading on where your cutting tool was relative to the workpiece, because you didn't have any feedback --- the machinist might not be running open-loop but the machine tool was. Nowadays we could use closed-loop feedback on relative tool-workpiece positioning, which would also stop a lot of crashes, but we don't.

When you're filing a part by hand, the file is held in your hand, which is about a hundred times more compliant than the floppiest machine tool frame. But, if you hold the file firmly, it cuts cleanly and doesn't chatter; and you can file your parts down to single-micron tolerances if your micrometer is that good. That's because you're damping chatter instead of just resisting it, and because you're using closed-loop feedback on when you're cutting, when you're not, and how much you've cut.

The standard cure for chatter in a machine tool is to add rigidity: to your setup, to your tool, to the tool frame, whatever. But increased rigidity doesn't eliminate vibrational modes; it increases

their frequency, decreases their displacement, and increases their force. What *eliminates* vibrational modes is nonlinearity, like the viscoelastic behavior of your hand meat on a file, or — ironically — metal parts banging together and moving energy from a lower-frequency vibration to a higher-frequency vibration. The more rigid and linear a system is, the higher its Q factor!

So I'd like to see more about other approaches to chatter that don't depend on rigidity. Damp vibrations with sand and gravel. Actively cancel chatter with piezoelectric actuators instead of passively resisting it. Cut with files with randomly-spaced teeth, perhaps made with carbide inserts. I don't know what will work.

As for closed-loop feedback, it's possible for interferometric systems like ERIM's HoloMapper from 1997 to get submicron measurements at millions of pixel locations across the surface of a part at once, without making contact. (At the time the latency was four minutes, but there's no reason it needs to take that long now.) Using this in real time as you're machining would mean sacrificing flood coolant, but modern carbide tools can cut steel pretty well dry.

I've previously written about geometric-optics sparkle feedback, where a sparkle pattern from sparkle glued to a rigid body indicates simultaneously its position and attitude to a camera at a known location with a point-source light at a known location. Combined with a reference mask that obscures some of the sparkles, this should be capable of giving relatively precise feedback,

sparkle feedback

kinematic mounts plus clutches

Topics

Importing the WHO's COVID-19 data into SQLite

Kragen Javier Sitaker, 02020-07-10 (2 minutes)

I downloaded the WHO CSV of covid data and imported it into SQLite to query it as follows:

```
$ sqlite3 covid-data.sqlite3
SQLite version 3.11.0 2016-02-15 17:29:24
Enter ".help" for usage hints.
sqlite> .mode csv who
sqlite> .import WHO-COVID-19-global-data.csv who
sqlite> .schema
CREATE TABLE who(
  "Date_reported" TEXT,
  " Country_code" TEXT,
  " Country" TEXT,
  " WHO_region" TEXT,
  " New_cases" TEXT,
  " Cumulative_cases" TEXT,
  " New_deaths" TEXT,
  " Cumulative_deaths" TEXT
);
sqlite> select sum(deaths) from (
   ...> select " Country", max(cast(" Cumulative_deaths" as decimal)) as deaths
   ...> from who
   ...> group by " Country"
   ...> );
508456
sqlite> select " Country", max(cast(" Cumulative_deaths" as decimal)) as deaths
   ...> from who
   ...> group by " Country"
   ...> order by deaths desc
   ...> limit 8;
"United States of America",126573
Brazil,58314
"The United Kingdom",43730
Italy,34767
France,29760
Spain,28752
Mexico,27121
India,17400
sqlite> select max(Date_reported) from who;
2020-07-01
```

The `cast` is necessary because otherwise the sorting is performed ASCIIbetically, producing the wrong answer. Here's Argentina:

```
sqlite> select Date_reported, " Cumulative_deaths", " Cumulative_cases" from who
   ...> where Date_reported in ('2020-05-01', '2020-05-15', '2020-06-01', '2020-06-15', '2020-07-01')
```

```
   ...> and " Country" = 'Argentina';
2020-05-01,215,4304
2020-05-15,345,6973
2020-06-01,530,16214
2020-06-15,819,30295
2020-07-01,1283,62268
sqlite> select Date_reported, " Cumulative_deaths", " Cumulative_cases" from who
   ...> where (Date_reported like '%-01' or Date_reported like '%-15')
   ...> and " Country" = 'Argentina';
2020-03-15,2,45
2020-04-01,24,966
2020-04-15,101,2336
2020-05-01,215,4304
2020-05-15,345,6973
2020-06-01,530,16214
2020-06-15,819,30295
2020-07-01,1283,62268
```

For Derctuo I want to be able to do queries like this interactively and easily (more easily than SQL) and plot the results. The CSV in question is 1.07 megabytes, but gzips to 191kB, and I suspect would be under 100kB with a simple column-oriented database doing delta-compression.

Topics

Migrating app snapshots

Kragen Javier Sitaker, 02020-07-10 (updated 02020-07-11)
(14 minutes)

Consider the problem of migrating a running program on demand to whatever computer you have handy. Perhaps a "master" copy of the program's running image lives on a "home server", and when you want to use a device, you take out a "lease" on the application's image and start downloading it to the device and using it.

As a sort of reference case, I have a newly installed Ubuntu virtual machine (p. 209) which consumes 11 gigabytes on disk and is configured with 2 gibibytes of RAM, and I'm using a 20Mbps Argentine internet connection at the moment. Downloading these 13 gigabytes of data to a local machine would take about an hour and a half.

However, you might be able to reduce this time in a number of ways:

• You might be able to **demand-page** it to some extent, prioritizing the transfer of blocks of the memory or disk image that the application is blocking on to download first. This way you might be able to use the virtual machine considerably earlier than an hour and a half. (Of course, some kind of prefetch strategy could make demand-paging work a lot better.)
• You might be able to **cache** it. If the image is organized as a (materialized) Merkle tree, then you can download only the blocks that aren't already present locally; moreover, the rsync algorithm (ideally with a zsync-like precomputed index) may offer further benefits, allowing typical filesystem changes to be transferred very rapidly. Merkle-tree storage implies indexing the blocks of the image by a secure hash, which will automatically deduplicate them. After the base Ubuntu install, for example, I installed a bunch of development tools and some software projects in a derived image, which used only 2 gigabytes in the derived disk image, which would take only 13 minutes to transmit. (Probably some things changed in RAM, too, but I don't have a good way to measure them.)
• You can **compress** the transferred data, using an algorithm like gzip (LZ77) or LZSS. For example, the 11-gigabyte Ubuntu install mentioned above gzips to only 4.2 gigabytes, reducing the initial setup time to about 40 minutes (including RAM); the 2-gigabyte derived image — the deltas to set up a development environment — compresses to 1.03 gigabytes, about seven minutes.
• You can **run the app on the server** while the transfer is happening, transmitting screen images and input events over the network in parallel with the streaming of the memory image. This of course means that the image is being partly invalidated while it's being transferred, but this measure may be enough to reduce the pause by orders of magnitude. If the app's rate of invalidating pages is lower than the available bandwidth, for a long enough period of time for the previously invalidated pages to be transferred, the pause will be reduced to zero.

• You can **get a faster internet connection**. For example, if you have a 400Mbps connection instead of 20Mbps, the same transfer would take 5 minutes instead of an hour and a half.
• You can **use less storage**. For example, the Emacs process I'm typing this note in has a virtual memory size of 308 megabytes, of which 16 megabytes is resident; the 308 megabytes includes all of its shared libraries and Lisp code, though 250 megabytes of it is two mappings of the 125MB /usr/share/icons/hicolor/icon-theme.cache, which hasn't changed in eight months and gzips to only 17 megabytes. So a full app snapshot of this Emacs process would take two minutes rather than an hour and a half, or 30 seconds with gzip, and if only the 16 megabytes were needed, it would take only six seconds.
• You can **flush caches**. Most in-memory application state is not vital and can be regenerated from other, more compact state — a decompressed image in BGRA can be regenerated from its JPEG, for example. If the application can be notified to flush caches in preparation for checkpointing, then everything gets easier. It probably isn't necessary to have a special case for the Linux disk cache, though, since indexing by hash takes care of that already.

Leases, stealing, and committing

How would you get state back onto the home server? Unless you want to require every app to be written in terms of CRDTs or event sourcing, you need some kind of concurrency control, specifically mutual exclusion.

The most reasonable solution is to acquire a **lease**, a time-limited lock, on the application state you're "checking out". So when you start snarfing the dirty pages into your tablet, the tablet might acquire a three-hour lease it renews every hour. As long as it holds that lease, any attempt to check out the application state on another machine will fail, telling you to close it on your tablet first. When you close the application on the tablet, it releases its lease, so the lease terminates earlier than the three-hour deadline, which simply serves as a timeout to permit automatic recovery in case of device failure.

Periodically the tablet checkpoints the local state of the application locally, then (if still connected to the internet) begins streaming the dirty pages of that checkpoint back up to the server as a possible future commit. Once that checkpoint finishes streaming, it optionally commits it on the server, then makes a new checkpoint and starts streaming that one to the server. Since the checkpoint isn't modified while it's streaming, the streaming process is guaranteed to finish in finite time, however slow the connection, although it might take a long time on a slow connection. The state that is committed is always a consistent checkpoint from a single point in time, but it may be somewhat out of date.

So if your local device fails, you only lose the last few minutes of work; the rest, up to the last committed checkpoint, is saved on the server.

This approach permits internet-disconnected operation for a limited period of time as well, for which purpose you might want a longer lease, maybe a day or two up to a month or two. This poses the problem of what happens if the device owning the checkout is

lost, stolen, or broken; in such a case you will want to **steal** the lease, so any state on the lost device becomes **orphaned** and cannot be committed to the original application image, though it can perhaps be committed as a new image that branched from the original.

"Read-only checkouts" are also useful: checkouts of the application image that succeed even if a lease is outstanding, acquire no lease themselves, and cannot commit, used for consulting data in the app without making (persistent) modifications to it.

Committing from an expired or orphaned lease or a read-only checkout can be allowed if no other commits have happened since the checkout and there is no lease outstanding.

Reasons for migrating

The main reasons for wanting to migrate a running app to the computer in your hand are (a) interaction latency, (b) disconnected operation, and (c) experimentation you might not want to deploy. The main reasons for wanting to migrate it to a server are (a) greater compute resources, (b) higher bandwidth and lower latency to the rest of the internet, (c) making it available to interact with other people, and (d) potential recovery from device failure.

So you could, for example, check out a website onto your netbook, modify some things about its setup while disconnected, test it locally to ensure it's working as desired, then commit it to the server once you reconnect to the internet. Or you could stream checkpoints of your digital audio workstation to your home server so that if it breaks or gets stolen you suffer minimal interruption to your work. Or you could interactively edit a 3-D scene on your laptop in Blender, then migrate your Blender session to your rendering cluster to run faster overnight. Or your could periodically checkpoint a long-running compute job on a cluster, on individual machines or cluster-wide, saving the snapshots to a different machine in order to recover from partial failures.

An interesting special case is where the device you're running on doesn't have enough space for a whole snapshot, so it needs to occasionally demand-page in bits of the image while it's running. This could make it feasible to run memory-hungry applications like Slack on machines with relatively little RAM, although swapping over the network like that can be slow.

Another sort of special case is where the "home server" is just a local disk, and effectively you're just implementing checkpointing and software suspend. This should give you quicker boots (if you can circumvent slow BIOS/UEFI/Linux anyway) and, if you run out of battery, you'll recover to the latest consistent checkpoint when you come back up.

More generally, you can have topologies other than a simple client-server topology; you could write out checkpoints to a local disk, which is streaming them to another server elsewhere, or you could have a distributed net of servers for block storage, leases, and commits, with some kind of quorum system for things that require consensus. Multiple clients on the same LAN can promiscuously share new blocks that might be useful for migration. And so on.

Security issues

Cloning a machine containing secrets, including entropy pool data, can lead to the inadvertent disclosure of secrets with many cryptosystems; for example, it can lead to nonce reuse, or the computation of multiple RSA keys containing common factors. It would be advisable to consider any such random numbers to be nonrandom after a checkpoint. Moreover, host-based security measures like retry limits and sleeps between wrong-password retries are entirely circumvented if the attacker can snapshot and replicate the host, but of course in that case the attacker owns the hardware and the game is over anyway.

The migrated state is subject to corruption from whatever host it's been migrated to, so in effect the running application is trusting every host that has ever committed to it in the past; any of them can have inserted arbitrary malicious code into the imge. In theory a defender might be able to detect this, but in practice probably would not.

In the form described above, the application state is also entirely vulnerable to the server; a malicious server can steal information and make arbitrary modifications to it. If you were willing to give up the possibility of executing applications on the server, you could reduce this vulnerability to some extent by signing and encrypting the application state on the clients, perhaps even limiting the server's powers to mere denials of service; you'd have to be careful about replay attacks, and it might not be possible to stop them entirely, and of course the amount and pattern of encrypted data blocks read and written might provide a malicious server with access to information we would prefer to conceal from it.

Concrete implementation approaches

QEMU's CLI has "stop", "cont", "savevm" and "loadvm" commands that might be a sufficient hook to implement such a system, reducing the problem to a problem of synchronizing qcow2 images (or, possibly, snapshots thereof). QEMU also has a live migration feature (I don't know how this works) and the ability to create a "copy-on-read" image with a remote "backing file", which is awfully similar to the features described above; however, VM snapshotted states from the backing file are not available in the derived image.

QEMU now has a machine type called "microvm" intended for booting single-application virtual machines.

I wrote about a user-level virtual-memory system that would facilitate this kind of copy-on-write thing (p. 166).

WebAssembly is an obvious implementation technology to try, both in that the client apps could be web browsers and in that WebAssembly runtimes are likely to support the kinds of isolation and snapshotting that would be useful for this kind of thing, as well as often being more manageable than entire Linux installations.

Docker of course is commonly used for running single (server) applications in an isolated environment, and it extensively uses copy-on-write to keep its disk space usage somewhat manageable. A typical Docker image using Alpine Linux might be 700 MB, five

minutes. (I thought it was a lot smaller, but the ones I have here are that big.) It would be interesting to try replicating Docker instances around.

Topics

Virtual machine setup

Kragen Javier Sitaker, 02020-07-10 (updated 02020-07-14)
(17 minutes)

I set up a virtual machine this week using the virtual-machine emulator QEMU with KVM under Ubuntu 20.04.

Objectives

I want to have a cloud development server. A problem with this in the past has been upgrades: if I don't upgrade the machine's software, it gets out of date and progressively more painful to do things on. But when I do upgrade it, I'm at risk of the machine not booting any more, perhaps requiring a crash cart to visit it, or even plugging the disks into another machine (that still boots) to recover their data.

Amazon AWS allows you to snapshot an EC2 volume before trying an upgrade, so you can roll it back if things go badly. Other virtualization and paravirtualization systems have similar capabilities. The simplest solution is just to use QEMU running under a popular system with good support; Ubuntu 20.04 is supported until 2025, for example. Then the "hypervisor" operating system installed on the physical hardware can remain relatively untouched by whatever development activities I'm doing, while the guests can evolve at will.

It would also be nice to be able to use a sandbox with some chance of containing potential attacks to a single more or less disposable virtual machine.

Also, there are some experiments I've been wanting to try for a while involving incremental snapshots of virtual machines (p. 204), and this might be a nice stepping stone.

Initial setup procedure

In order to get KVM working, first we had to enable "Virtualization Technology" in the Dell PowerEdge R610 machine's BIOS; it was disabled by default, as indicated by the `kvm-ok` command, although enabled by default in Ubuntu 20.04's kernel and present in the CPU, which `/proc/cpuinfo` says is an "Intel(R) Xeon(R) CPU E5649 @ 2.53GHz".

I was having a hard time setting up Debian inside QEMU, so I snarfed the Ubuntu install ISO (SHA256 e5b72e9cfe20988991c9cd87bde43c0b691e3b67b01f76d23f8150615883c0e11) instead. This is a reconstruction of what would have had the right effect (I mistakenly used QED instead; see "Escaping QED" below):

```
qemu-img create -f qcow2 ubuntu-base.qcow2 32G
kvm -hda ubuntu-base.qcow2 -cdrom Downloads/ubuntu-20.04-desktop-amd64.iso -m 2G
```

`kvm` is the command installed by the `qemu-kvm` package which is just equivalent to `qemu-system-x86_64 -enable-kvm`. (Older versions of `qemu-kvm` were actually a separate branch of QEMU I think, but it's still more convenient to invoke it this way.)

At first I made the mistake of making the disk too small; Ubuntu 20.04 claims to need at least 8.6 GB to install, and in fact used 8.8 GB. (The QCOW2 format is allocate-on-write, so even though the virtual disk is 32 GB, the `ubuntu-base.qcow2` file it's stored in is only 8.8 GB, since it's mostly unused.) Also, QEMU's default memory size turns out to be 128MiB, which is too small, and Ubuntu's installer "reported" this fact by displaying a blank text-mode screen with a blinking cursor and never doing anything else; `-m 2G` or something is needed.

At first I was having trouble with keyboard focus in QEMU, which I think may be a matter of using the obsolete and buggy window manager `wm2`; I worked around this by running QEMU with `-vnc :2`. QEMU by default has no authentication on its VNC interface; rather than fixing this (see below about the options to fix that) I just packet-filtered VNC on the machine hosting QEMU and, for good measure, X-Windows too:

```
iptables -A INPUT -s 127.0.0.0/24 -p tcp --dport 5900:6100 -j ACCEPT
iptables -A INPUT -s 192.168.0.0/24 -p tcp --dport 5900:6100 -j ACCEPT
iptables -A INPUT -p tcp --dport 5900:6100 -j REJECT
```

(A little additional work was needed to get this to take effect at every boot.)

This is a little dodgy given that network traffic from the virtual machine itself appears to come from localhost, since it's using the `user` networking type (Slirp), so different virtual machines have free rein to connect to VNC and X servers.

To connect remotely to the server from outside its local network, I'm tunneling over `ssh`, which works pretty well:

```
ssh -C -L 5902:localhost:5902 server
```

That way I can run `xvncviewer :2` on the machine I'm sshing from, and `ssh` encrypts and compresses the data over the network, as well as (implicitly) authenticating me by making the connection to the VNC server come from localhost.

Once I had Ubuntu installed, I could run the virtual machine without the CD-ROM:

```
kvm -hda ubuntu-base.qcow2 -m 2G
```

But rather than running directly from there, I used it as a base for cloning further copy-on-write disk images, which is a feature of the QCOW, QCOW2, and QED virtual disk formats:

```
qemu-img create -b ubuntu-base.qcow2 -f qcow2 ubuntu-dev0.qcow2
qemu-img create -b ubuntu-base.qcow2 -f qcow2 ubuntu-dev1.qcow2
chmod 444 ubuntu-base.qcow2
```

Now ubuntu-base.qcow2 is what Proxmox calls a "template": you can't start it but you can create and start clones of it.

And I wrote a script to launch virtual machines with these cloned

disk images:

```
$ cat dev0
#!/bin/sh
kvm -hda ubuntu-dev0.qcow2 -smp 12 -m 2G "$@"
```

This approach allows me to clone new virgin virtual disks at a cost of some 200 kB (plus whatever is used thereafter, typically tens of megabytes to gigabytes) and 250 milliseconds. That way I won't have to install Ubuntu again.

Escaping QED

Initially I used the deprecated disk image format QED (`-f qed`) because I misunderstood the QEMU documentation to be saying that it had some extra features; to fix it, I did this:

```
qemu-img convert ubuntu-base.qed -O qcow2 ubuntu-base.qcow2
```

This took 4–6 minutes and shrank the file to 8.8 GB. Then I needed to recreate the dev child image and reinstall the things that I had installed in it previously.

Making a backed QCOW2 image is actually significantly slower than doing it with QED, but not enough to matter for my purposes; doing this with QED took 10–11 milliseconds:

```
$ time qemu-img create -b ubuntu-base.qcow2 -f qcow2 ubuntu-dev0.qcow2

Formatting 'ubuntu-dev0.qcow2', fmt=qcow2 size=34359738368 backing_file=ubuntu-base.qcow2 cluster_size=65536 lazy_refcounts=off refcount_bits=16

real    0m0.244s
```

The resulting derived file is only 197kB; after spending ten minutes installing stuff in it, it's 1 GB.

Interestingly, both QCOW2 and QED can use a file in a different format or even accessed over HTTP as the backing file, so I could put that base image (or the QED one) up on a web site and remotely lazily clone it!

Recovering disk space used by deleted VM snapshots

After I used `savevm` a couple of times, `qemu-img` reported, at one point:

```
$ qemu-img info ubuntu-dev0.qcow2
image: ubuntu-dev0.qcow2
file format: qcow2
virtual size: 32 GiB (34359738368 bytes)
disk size: 5.67 GiB
cluster_size: 65536
backing file: ubuntu-base.qcow2
Snapshot list:
ID        TAG                 VM SIZE                DATE       VM CLOCK
```

```
1         tetris1             1.5 GiB 2020-07-10 16:40:17   00:01:43.207
2         ready               1.5 GiB 2020-07-10 16:59:52   00:11:43.959
Format specific information:
    compat: 1.1
    lazy refcounts: false
    refcount bits: 16
    corrupt: false
```

So it seems like the VM-state snapshots show up as disk-state snapshots. I have deleted them:

```
qemu-img snapshot ubuntu-dev0.qcow2 -d tetris1
qemu-img snapshot ubuntu-dev0.qcow2 -d ready
```

But this does not reduce the size of the QCOW2 file all the way back down; `du -h` and `qemu-img info` show that it's still occupying 3.9 GB of real space, and its file size in `ls -lh` is still 5.7 GB (so it's somewhat sparse).

I thought maybe `qemu-img convert` might solve the problem, but it seems that `qemu-img convert` produces an image without a backing file — so it's ten gigs. It turns out that the way to avoid this is using `qemu-img rebase`, as explained in the qemu-img man page:

```
qemu-img create -b ubuntu-dev0.qcow2 -f qcow2 ubuntu-dev0-copy.qcow2 # 92 ms
qemu-img rebase -b ubuntu-base.qcow2 ubuntu-dev0-copy.qcow2  # 76773 ms
```

This produces a 2.4-gigabyte copy which `qemu-img compare` reports is identical to `ubuntu-dev0.qcow2`. (I'm not sure but I think I have about 2.4 GB of devtools stuff installed in this image, above and beyond what's in the base image.)

Results

So far everything seems reasonably okay except that screen redraws are painfully slow.

In single-CPU user-level compute performance, QEMU with KVM seems to only cost on the order of 5%, if anything: `./fib 40` inside QEMU takes 632–663 ms, while on the host machine it takes 619–641 ms. However, the host machine has 12 CPUs with hyperthreading, thus 24 "CPUs", while the QEMU-emulated machine initially had only a single virtual CPU.

It turns out QEMU has an `-smp` flag that's just off by default. Running `./dev0 -smp 12` (or later adding `-smp 12` in the `dev0` script) and building Yeso with `make` takes 9.3–10.2 seconds. `make -j 12`, to run up to 12 compilation processes in parallel when possible, takes 1.8–2.2 seconds; that's more than a 5× speedup. On the host machine, the corresponding numbers are 7.4–8.4 seconds and 1.41–1.45 seconds, suggesting that QEMU's overhead for system things like file I/O and process management is more like 30%. And on the host machine `make -j 30` is even faster, at 1.35–1.40 seconds, but unsurprisingly provides no additional speedup on the 12-CPU virtual machine.

Over my high-latency internet connection to the server, graphical user interfaces are a bit slow, perhaps in part because of bandwidth

limits; repainting a full 1024×768 virtual screen takes 5–15 seconds. However, browsers typically load pages a lot faster; they're just slower to scroll. It might be worthwhile trying XPra or Spice to see if I can get faster screen updates, or just using ssh and/or Mosh when possible.

Running with `-vnc :1` I can get a console in my terminal window with `-monitor stdio`. This is apparently how to use the `set_password` command to require a password on the VNC server (required with `-vnc :1,password` supposedly). (SASL is also an authentication option.) Also apparently `-vnc localhost:1` would also only allow connections from localhost, though without any real authentication.

By using `savevm tetris1` at the monitor prompt `(qemu)` I can save a virtual machine image that I can later revive with `kvm ... -loadvm tetris1`, thus returning to a particular point in the Tetris game I was playing. Doing this bloats the .qcow2 file from 1 GB to 2.6 GB, presumably with a RAM image, and takes about 15 seconds, during which time the VM is paused, which is pretty disruptive. Reloading from this image is, I think, faster than saving (or booting), but it still takes 15 seconds to repaint my screen over this slow internet connection.

A lazy clone of a disk image (QCOW2 at least) doesn't share the snapshots of its backing file. Presumably I could clone an already-booted virtual machine (with the booted state in a VM snapshot) by `cp foo.qcow2 bar.qcow2`.

XPra

I decided to try XPra to see if I could get a more usable remote display for graphical things than VNC, which was too slow. On my outdated Linux Mint laptop, I installed XPra 0.15.8 (from 2015):

```
sudo apt install xpra python-rencode python-gtkglext1
```

I installed the last two packages listed because, without them, though XPra worked, it complained as follows about missing Python libraries:

```
2020-07-14 21:28:33,437 rencode import error: No module named rencode
2020-07-14 21:28:33,987 Warning: 'rencode' packet encoder not found
2020-07-14 21:28:33,988  the other packet encoders are much slower
2020-07-14 21:28:33,988 xpra gtk2 client version 0.15.8 (r11211)
2020-07-14 21:28:34,044 OpenGL support could not be enabled:
2020-07-14 21:28:34,044  cannot import name gdkgl
```

On the Ubuntu 20.04 server, I installed XPra 3.0.6:

```
sudo apt install xpra
```

Then I was able to launch a remote xterm displaying on my local display via

```
xpra start ssh:serverhost --start=xterm --remote-xpra=xpra
```

and later reattach to the session containing the xterm with

```
xpra attach ssh:serverhost --remote-xpra=xpra
```

Within the xterm I could then run

```
./dev0
```

in order to launch the QEMU KVM virtual machine as described previously.

Without the `--remote-xpra=xpra` option, I was getting failures with this error:

```
bash: /home/user/.xpra/run-xpra: No such file or directory
2020-07-14 21:31:30,499 failed to receive anything, not an xpra server?
2020-07-14 21:31:30,500   could also be the wrong username, password or port

2020-07-14 21:31:30,500   or maybe this server does not support 'unknown' compres
sion or 'bencode' packet encoding?
2020-07-14 21:31:30,500 Connection lost
```

There's still highly noticeable lag, but it seems dramatically more usable than VNC. And VNC had more trouble with my keymapping. XPra is reportedly using peaks of up to about 16 megabits per second. My initial impression of XPra: *this is fucking awesome*.

It might be more reasonable to run XPra within the guest instead of on the host (that way copy and paste would work, for example, and I wouldn't be limited to the screen space of the virtual machine's emulated graphics card), but this was an easier way to get started, and it allows me to handle the guest bootup process as well.

With this combination of XPra versions, I do get this error message, but everything graphical except setting cursors seems to work:

```
2020-07-14 21:27:06,962 error creating cursor: object of type 'int' has no len()
(using default)
Traceback (most recent call last):

  File "/usr/lib/python2.7/dist-packages/xpra/client/gtk_base/gtk_client_base.py"
, line 329, in set_windows_cursor
    cursor = self.make_cursor(cursor_data)

  File "/usr/lib/python2.7/dist-packages/xpra/client/gtk_base/gtk_client_base.py"
, line 359, in make_cursor
    if len(pixels)<w*h*4:
TypeError: object of type 'int' has no len()
```

Unknowns to probe/things to try

What's the most reasonable way to enable ssh into these virtual machines? I'd need to disable password authentication and do some

kind of port forwarding. By default QEMU does its networking with Slirp, but it can alternatively use TUN/TAP or L2TPv3. There used to be a `-redir tcp:2222::22` option that looks like it will work, which I think is now spelled `-net user,hostfwd=tcp::2222-:22`.

How about Mosh?

Is there some way to save VM state snapshots in a copy-on-write way so that I can journal aggregated machine state changes out over a network for point-in-time recovery? Even cooler would be if I could unfreeze from such a snapshot when an ssh connection came in.

Can I get Ubuntu or Debian to boot in QEMU with KVM with `-nographic`?

What's the easiest way to do copy-paste in and out of QEMU, when not using ssh? Am I better off using spice (see also) or curses? Apparently Spice makes it easier.

Is my window manager really what's at fault in the keyboard focus problem?

How insecure is KVM?

How about accessing files on the guest's filesystem? There are `-fsdev` and `-virtfs` flags to QEMU, but I'm not sure what they do.

Is there an advantage to kvm -M pc-q35-focal? The default is pc-i440fx-focal.

What do Bonnie++ and lmbench think? Does using the virtio block controller instead of emulated IDE help? The Proxmox dox say:

> It is highly recommended to use the virtio devices whenever you can, as they provide a big performance improvement. Using the virtio generic disk controller versus an emulated IDE controller will double the sequential write throughput, as measured with bonnie++(8). Using the virtio network interface can deliver up to three times the throughput of an emulated Intel E1000 network card, as measured with iperf(1). [1]

Can I do KVM Inception, running QEMU with KVM inside of QEMU with KVM? I think the answer is yes, Android Studio says the answer is yes, for testing Android apps inside the virtual machine it would be extremely convenient for the answer to be yes, but kvm-ok in the virtual machine says no.

Topics

Long distance radio

Kragen Javier Sitaker, 02020-07-17 (19 minutes)

I've previously written about ultraslow radio for decentralized global digital communication, but since then I've read a bit more about the topic, including a little bit of the ample literature on amateur radio DX, QRP, and contesting.

Due to skywave propagation, hams using MF and HF radio routinely communicate 1000 km or more with transmit powers on the order of one watt (there's a "thousand-miles-per-watt" award); under exceptional conditions, transmissions of 1000 km on 1 mW of transmitted power have been reported. Typical transmission modes include (very slow "QRSS") CW and the WSJT modes, many of which are around one bit per second.

So now I see how to build infrastructure that permits global data communication at hundreds of kilobits per second when the ionosphere is favorable, without emitting a noticeable amount of radio interference, and without requiring more power than is easily available by energy harvesting. A global network of low-power kilometer-scale phased arrays can speak ultrawideband MF and HF to each other, but ultrawideband at higher frequencies internally and to nearby mobile radios.

Power levels

A Wi-Fi card might emit 200 milliwatts, although the little FM radio transmitters you might plug into your MP3 player, legal since 2006 in the EU and longer in the US and Canada, are only about a microwatt, 10 nW in the US, 50 nW in the UK, 25 microwatts in Japan. The US allows 100 mW unlicensed narrowband AM radio transmitters, so I think 10 milliwatts per transmitter site ought to be reasonable.

In a memory-holed YouTube video, Naomi Wu recently reviewed the Ulefone Armor 3WT FRS cellphone, which includes a 2W FRS walkie-talkie. She reports that in Shenzhen she can get several blocks of range, which is to say, several hundred meters. FRS and GMRS radios commonly transmit at such powers; GMRS is permitted up to 50 watts, though WP says 1-5 watts is more common in practice, and FRS in the US was limited to 500 mW until 2017; FRS commonly gets a kilometer or so of range, though (again, WP says) tens of kilometers are possible "under exceptional conditions...such as hilltop to hilltop". 3G mobile phones also transmit 2 W. So if there's no regulatory or interference problem, it's reasonable for even a handheld device to transmit at 1-2 watts. (Most cellphones are, I think, up to 1 watt.)

Handheld ferrite loopstick antennas are capable of transmitting and receiving MF signals like those used for AM radio, but their antenna efficiency is fairly low. A better approach for mobile stations is probably to use higher frequencies to connect handheld devices to large, fixed infrastructure like a long-distance phased array, which then handles the long-range communication. Still, these short-range

links might be able to reach many kilometers. (LoRa at 915 MHz can reach 10 km in rural areas, though fewer km in cities; one-watt GSM cellphones can talk to a base station 35 km away, and a "timing advance limit" has been hacked into some GSM equipment to extend that range further.)

A handheld device is inevitably a point source of interference, with the unavoidable inverse-square interference pattern that implies. A kilometer-scale phased array is, by contrast, a diffuse source, so it can emit at a much higher power before it starts to become a nuisance to neighbors.

GPS

GPS receivers cost a few dollars and receive signals at -125 dBm or less; some can lock in a signal at -142 dBm, which is quite impressive considering that the thermal noise on a 2-MHz-wide GPS channel is about -111 dBm. They are made cheaper by the fact that they run at over 1 GHz, so they don't need large antennas. Acquiring these signals is feasible because they are perfectly uncorrelated over long periods of time, like an LFSR. Ultrawideband techniques have the same virtue.

Ultrawideband and frequency bands

Modern impulse radio ("ultrawideband") should be able to essentially eliminate interference with the nearly orthogonal narrowband signals conventionally used. A commercial AM radio station, for example, might transmit at 10 to 100 kW over a bandwidth of 20 kHz, on the order of 1 W/Hz. A 10mW impulse radio whose pulses are evenly spread across the whole medium-wave AM broadcast band from 526.5 kHz to 1606.5 kHz would average 9 nW/Hz, eight orders of magnitude quieter, easily below the noise floor, although it might become (faintly) audible if it were 30 dB higher in a particular compass direction because of (see below) phased-array directional transmission.

This 1080 kHz bandwidth gives a temporal precision of about a microsecond, suggesting a few hundred kilobits per second of possible transmission speed.

Transmitting over the shortwave band from 2.3 to 26.1 MHz would permit multi-megabit transmissions, though of course subject to ionospheric conditions; there used to be 500-kW Voice of America broadcasting on this band, though I'm not sure there still is, but Wikipedia tells me there are 1200-kilowatt shortwave broadcasters, and I think their bandwidth may be 10 kHz.

(Commercial FM radio typically also transmits at a few tens of kW, but it's in the 87-105 MHz range, where there's no significant ionosphere propagation.)

Chirping and wider bands

Chirping the transmitted pulses, like LoRa or chirped radar, would avoid the need for high peak-to-average power ratios that might otherwise pose a difficulty, and would also reduce the time-domain artifacts that would otherwise appear to unintentional wideband receivers. Straightforward chirping wouldn't help to avoid

narrowband receivers, though; if you were to chirp from 526.5 kHz up to 1606.5 kHz in 1.08 milliseconds, you're only chirping 1 kHz per microsecond, so you only spend 20 microseconds in each 20-kHz-wide AM station. This would only attenuate the part of the impulsive noise added to AM above 50 kHz, which the humans can't hear anyway.

You could imagine doing several simultaneous chirps, though, which might help more; one that sweeps from 526.5 kHz up to 548.1 kHz over that millisecond, while another sweeps from 548.1 kHz up to 569.7 kHz, and so on. Effectively each chirp would be a single AM station wide, and spread over the whole millisecond, thus strongly attenuating the parts of the impulse above about 1 kHz, making it considerably less audible. Presumably this waveform still retains the time-domain precision deriving from its >1MHz bandwidth.

A more effective way to reduce interference might be simply spreading the signal over a wider bandwidth by using shorter pulses. If the pulses were 30 ns instead of 1000 ns, for example, going up to 33 MHz instead of 1.5 MHz, you'd have 15 dB less power in any given station's 20 kHz band, 0.3 nW/Hz, about 95 dB quieter than AM broadcasters --- 63 dB because of transmitting at 63 dB lower power, plus 32 dB because it's spread across 17000 times as much bandwidth.

Phased-array transceivers

Directional transmission at MF (300 kHz to 3 MHz) and HF (3 to 30 MHz) would seem to require impractically large antennas: even 30 MHz is 10 meters, and 300 kHz is 1 km. However, phased-array transmission and reception from an antenna array distributed over a significant geographical area should be possible, and with practical numbers of transceivers (10 to 1000 transceivers) significant degrees of directionality should be possible; without understanding the math, I'm guessing it would be 10 to 30 dBi, with the additional advantage (for skywave propagation) that most of the energy would propagate horizontally. (My intuitive reasoning is that in the direction of the wave, all 1000 transmitters are in phase, so the amplitude is 1000 times higher than the wave from a single transmitter, while in other directions, it's only 32 times higher, so it's 32 times higher in the direction of transmission, which means 1000 times higher power.)

How would you coordinate a phased array of radio transceivers to transmit data? It's a bit like the firing-squad problem in cellular automata; they can use lower-power, higher-bandwidth, higher-frequency local radio among themselves to compute precise relative geolocations, synchronize their clocks, and buffer up bits to be sent in a phased-array fashion, or after being received in a phased-array fashion. They could use, for example, the 1800 MHz GSM spectrum, or the 2.4 GHz unlicensed spectrum. Time-domain signaling across a GHz of bandwidth should permit baseline measurements with a precision of a few centimeters.

Of course the same phased-array correlation approach can be used for reception. Probably MIMO techniques to augment bandwidth are not directly applicable over such long distances due to diffraction.

However, such a phased array could easily transmit to several destinations at once, or receive from several senders at once. If there

are multiple relay stations available, it may be possible to augment the point-to-point bandwidth between two phased arrays by relaying the information in parallel over geographically diverse routes, like Ethernet channel bonding.

Diffraction

For the diffraction limit to be better than 30 dBi, so the phased array is limited by the number of transmitters rather than the aperture, the diffraction beam divergence needs to be less than 4 pi/1000 steradians, very crudely, which I think means less than about 110 milliradians, 6 degrees. Suppose we're using $1.220\lambda/D$, the Airy limit for a circular aperture, as an approximation, and we use 1 MHz for λ: 300 m. So we want 1.220 300 m/D = 0.11, so D = 1.220 300 m / 0.11 = 3.3 km, like, a transmitter every 100 m. Or 10 km if we want to get all the way down to 300 kHz. Normally we'd worry about sidelobes from spreading the transmitters too far apart, but I think that problem disappears with ultrawideband signals, since the sidebands for all the different frequencies are in different places.

However, if the transceivers are all on the ground, which is nearly planar, we're still going to have massive diffraction in the vertical direction, as our energy is spread across 30 degrees or more, even after half of it is reflected from the ground.

If your energy is spread evenly over 6 degrees, then after traveling a quarter of the way around Earth, what is left of it will be spread over some 700 km of width; this is perhaps 200 times the distance it was spread over originally, if the original phased array was 3.3 km, and of course it is also spread out vertically in a nonuniform way between the surface and the ionosphere. 200 times is a surprisingly modest -23 dB, although of course that's not the attenuation from the transmitter; it's the attenuation from the open spaces in the tens of meters between the transmitters to the place a quarter of the way around the world.

It might be necessary to confine the beam to a narrower horizontal angle than 6 degrees to compensate for the unavoidable vertical spread.

Energy harvesting

Running transceivers on harvested RF energy may permit embedding them in concrete or underground, or hanging them from trees. But it probably would not permit average transmitted power of 10 milliwatts or more; 100 microwatts might be more reasonable.

Passive reflection instead of transmission

Passive reflection by disconnecting an energy-harvesting antenna might be the most efficient way to produce pulses, and might also be more regulatorily acceptable. In urban areas, energy-harvesting researchers have found 1 to 100 microwatts per square centimeter in each of several different bands, including AM radio, digital TV, and especially the GSM and 3G bands. A simple calculation suggests that an MF AM radio loop antenna enclosing 10 m^2 at 2 km from a 50 kW broadcasting station intercepts about 10 m^2 50 kW / 4 pi (2 km)^2 = 10 mW, although probably in practice the number is somewhat larger. Such an antenna might be illuminated by several

such stations. By selectively making the antenna open-circuit at certain moments, those 10 mW will be reflected instead of absorbed at those moments, across all the frequencies that efficiently couple to the antenna.

Such passive reflection avoids the necessity to convert RF energy to stored voltage and then back again, with its attendant losses of probably some 20 dB, and since it does not transmit any energy, it might avoid regulatory entanglements; moreover it will not produce any energy on any frequencies that are not already in use. However, it makes it impossible to harvest energy on one band (such as GSM) and transmit it on another, and it makes chirping impossible. For communication on higher frequencies, antenna directivity might also be relevant; your antenna system might reasonably be organized to reflect the incoming illumination toward the destination.

Harvested solar energy

Worth noting is that 10 milliwatts of full sunlight is 0.1 cm^2, or about 0.7 cm^2 of a commonplace solar cell. So even a few square centimeters of PV cells would provide much more power *on average* than all this RF energy-harvesting stuff, even in areas brightly illuminated by cellphone towers. They might be able to produce alternating magnetic fields that transfer power wirelessly to a larger, less visible transceiver, perhaps embedded in a wall.

Low-duty-cycle communication

Lower-duty-cycle communication might reduce the degree of interference with other systems, and would surely reduce the energy transmitted per bit. As I understand it, there's no floor on energy transmitted per bit with a given noise floor, if you transmit slowly enough. If you're doing pulse-position modulation with 100-nanosecond timeslots, then you can transmit one bit in 2 timeslots, two bits in 4 timeslots, three bits in 8 timeslots, etc.; at some point your timing synchronization between the transmitter and receiver will start to suffer, but a regular quartz crystal has drift of about 10 ppm, while a temperature-compensated crystal oscillator (TCXO) is typically around 1 ppm. So you could imagine, for example, transmitting one pulse every 65536 timeslots (6.55 ms) to represent a 16-bit symbol. To get the same error probability per symbol, you'd need to send it at a higher amplitude than if you were sending one pulse every other timeslot, but I think only something like 6 times higher, assuming AWGN. (XXX make this rigorous, or at least do some experiments)

If that's correct, you get about 5x the energy efficiency per bit by using such a low-duty-cycle system, but you transmit 4096 times slower. However, it might increase interference with existing licensed uses of the spectrum, for example introducing more audible impulsive noise into AM radio.

Low-duty-cycle communication has an interesting relationship with chirping, since the effect of chirping is precisely to extend the duty cycle. On one hand, if the underlying signal you're trying to transmit isn't low-duty-cycle, chirping it won't do any good --- your chirps will overlap, and so you won't get the PAPR improvement you normally get from chirping. On the other hand, that PAPR is

precisely what allows you to leave your radio turned off most of the time and save power, so if you "improve" it too far, you will exceed your power budget.

Encoding

Of course you want to use error-correction coding so that no one pulse is strong enough to be received clearly at the destination; you want the pulses to be tens of dB below the noise floor so that substantial coding gain is needed to detect them, even near the source. The best way to ensure non-interference is non-detectability.

Estimating potential results at 1-100 megabaud

It's already commonplace for QRP hams to reach 1 bit per second transmitting 1000 km on 1 watt. Conservatively, phased-array transmission should buy you 20 dB, while phased-array reception should buy you another 20 dB. Supposing that those hams are not in the bandwidth-limited regime of the Shannon limit, using ultrawideband may not buy you any extra bandwidth, just keep you from slamming into a narrowband bandwidth ceiling. 1000 transmitters at 10 mW each works out to 10 watts rather than 1 watt, giving you another 10 dB, for a total of 50 dB, or 100 kilobaud, per phased-array-to-phased-array link. If you can talk to ten phased arrays at once, that should give you a megabaud. But if the phased arrays miraculously work out to buy you 30 dB instead of 20, you'd have 100 megabaud.

Alternative communication media

Earth-moon-earth or "moonbounce" communication is already commonplace among hams and sometimes is high enough bandwidth to hold voice conversations over. Doing the equivalent using passive MEO satellites would require more precise and dynamic tracking, to the point that it's probably only practical at microwave frequencies, but would suffer the d^4 loss of the moonbounce path over a much shorter distance, and still would cover most of a terrestrial hemisphere. LEO satellites have an even shorter path loss and larger cross-section, but only cover a thousand km or so. Meteor-trail communication is an existing well-known technique for high-bandwidth opportunistic communication at a similar range. And the ocean's SOFAR channel, though it has only a few kHz of bandwidth, has better attenuation characteristics, more consistency, and lower noise than the ionosphere route.

Topics

A generic universal entity-component simulatorium

Kragen Javier Sitaker, 02020-07-18 (1 minute)

What's the minimal core of something like a MOO, but using an entity-component system? You need to handle incoming telnet connections, do some kind of parsing on those connections, have a player object, and have a room object. You need some way to define new verbs, to identify objects in commands, to produce descriptions of rooms and their contents, and to create new rooms, doors, and other entities. And you need some kind of scheduling system for future scheduled events. You need to be able to checkpoint the world to disk and to load such a checkpoint at startup.

Components decompose attributes of entities along hypothetically orthogonal dimensions, like location, description, door connectivity, and so on.

It's not immediately apparent that there's a best way to resolve verbs with multiple possible definitions. Maybe the best way is to associate methods with components, and when a verb is invoked on an object with multiple components, activate all the methods. For example, rooms might belong to the description component, but also the room component, and when you describe a room, it should also list its contents' names. But I guess that needs to come in a well-defined sequence.

But maybe I'm overcomplicating things at first, and I could just make objects be dicts or something. Or an edge-labeled graph.

Topics

Line-numbered ISAM buffers

Kragen Javier Sitaker, 02020-07-18 (updated 02020-07-23)
(14 minutes)

Darius and I have talked occasionally over the years about the problem of text editor buffers. Editor buffers, like the ones in Emacs, need to support a few operations efficiently:

- Traversing the text sequentially, for example to repaint the screen or search for a string or regular expression.
- Adding markers to the text.
- Determining what markers are present at a given location.
- Jumping to a marker.
- Inserting and deleting text anywhere in the buffer.

From the point of view of the beginning of the buffer, text moves when you insert and delete things before it. The tricky part is that the markers need to move with the text; it isn't good enough to just store a byte offset for each marker.

Ideally we'd like all of these operations to be sublinear in the size of the buffer, and we'd like the buffer to be able to be at least nearly as big as RAM, if not the disk, and we might have many markers per line, for example to store syntax-highlighting properties of the text, so the number of markers also grows linearly as the text grows. If any of these operations take linear time instead of, say, logarithmic or at least square-root time, then they will become unbearably slow when we open a gigabyte-sized file, much less a terabyte-sized one.

I think Raph Levien has come up with a design for this in Xi based on ropes, but I don't know what it is.

I was lying in bed thinking about G-code and BASIC interpreters and the HP 3000, and I realized that you can more or less solve this with an ISAM approach, and this is probably what Darius and I had come up with before I forgot it until tonight. You represent the buffer as an (in-RAM) ISAM file with synthetic, meaningless keys. ISAM supports the following operations efficiently:

- Go to the first record whose key is equal to or following a given key.
- Go to the next record by key (or report failure).
- Go to the previous record by key (or report failure).
- Read the key and value of the current record.
- Delete the current record.
- Insert a new key-value pair into the file.

All of these take at most logarithmic time; 2 and 3 are typically constant time. (It's common for ISAM systems to support an update operation as well, but in the absence of concurrency, this can be synthesized from read, delete, and insert.) There are a variety of ways to implement this, though B*-trees and LSM-trees are the most popular.

How does this give us buffers? Well, when we read a file into a buffer, we break it into blocks of, say, 256 bytes, and assign each one a

sequential string ID to serve as its key; perhaps AAA, AAB, AAC, and so on, or if the file is a terabyte, AAAAAAA, AAAAAAB, AAAAAAC, and so on. When you add a marker in a block, you update the block to include a pointer to the marker, and you store the block key and the byte offset in the marker.

When you change text within a block, you must keep the block from growing too large; you may need to split the block, perhaps splitting block AAB into AAB and AABA. This requires updating the key stored in each marker that has moved to the new block. If you don't split the block, you must update the byte offset stored in each marker that would have moved to the new block.

To ensure that traversal remains fast, you might also have to keep blocks from becoming pathologically small, perhaps merging what little remains of block AAD into the end of block AAC and removing AAD if both of them have shrunk a lot.

Because traversing the blocks sequentially is fast, traversing the buffer sequentially is fast. Adding a marker is very fast, requiring only an update interaction. Finding what markers are present at a given location is fast because it only involves inspecting the current block, which is never very large. Jumping to a marker is fast because the marker contains the key to the block, which permits navigating to it via ISAM. Inserting and deleting may involve ISAM operations.

But why ISAM? Undo and incremental monoids

Why ISAM rather than just a doubly-linked-list piece table? You could include memory pointers to the pieces in the marker objects instead of ISAM keys. Inserting and deleting into a doubly-linked list is easy; you have to update all the markers concerned, but that is true with ISAM as well. And ISAM adds a logarithmic slowdown to the jump-to-a-marker operation, which would instead be constant-time with pointers to pieces. So is there any advantage of ISAM here?

Well, ISAM can provide FP-persistence. Ropes are "persistent" in the FP sense: a reference to a rope refers to a given state of that rope, so an undo history can be implemented simply as a list of pointers to ropes that share structure. You can implement ISAM in an FP-persistent way, and if the references from the buffer blocks to the markers are indirected through an FP-persistent dictionary data structure (whether some variant of ISAM or just a hash table) then the whole buffer structure can be FP-persistent.

Ropes don't have an obvious way to handle markers, though. Rope nodes are immutable. If you store markers in an immutable rope node, you can copy them to a new node if you make modified versions of it, easily supporting operation #3 --- but how do you support operation #4, jumping to a marker? Storing a pointer to a rope node in a marker doesn't help --- even if that rope node *is* in the version of the buffer of interest, you can't traverse the graph to its parent, because it may have many parents, some of which are in the version of interest and some of which are not.

The ISAM approach provides FP-persistence, like ropes, without losing the ability to track down a marker; its compensating drawback

is that copying text from one buffer to another, or from one place to another in the same buffer, requires copying all the text's characters. (*Cut* and paste can avoid this.)

Monoidal computations

(See also Monoid prefix sum (p. 105).)

Aside from simple undo, there's another set of operations commonly required in text editors which can be supported efficiently by ISAM or ropes, but not in any way I can see with a simple linked-list piece table: things like syntax highlighting, line numbers, and display column, which are generically a monoidal computation on the sequence of characters from the beginning of the file to a given point.

Basically the problem is that whether, say, a given line in a buffer is line 123 or line 124 depends on all the bytes before that line; inserting a single newline early in the buffer increments the line numbers of everything after it, but if this takes time proportional to the number of lines in the buffer, then it will be unusable on sufficiently large buffers. On the other hand, if you don't store any line-number information, then going to a given line number will be unusably slow on sufficiently large buffers. (It's okay for that to require a full buffer scan the first time, since there's no way to avoid that, but not every time.)

Parallel prefix sums

The parallel prefix-sum algorithm offers a solution to this problem for general monoids. If your buffer is made up of some kind of tree with text in its leaves, and traversing the tree left to right gives you the order of the text in the buffer, you can cache the monoid value for just the text within the subtree rooted at each node. Then, to calculate the monoid value for some prefix of the buffer, you use the monoid operation to combine the values in the tree nodes within that prefix, which is linear in the tree depth and thus logarithmic in the buffer size. Updating a leaf similarly merely requires invalidating and potentially recalculating the cached monoid values in its logarithmic number of ancestors. In the case of monotonic values like line numbers, you can also efficiently do a search for a given value using binary chop.

A gibibyte-sized concrete example

As a concrete example, suppose we have a 1-gibibyte buffer stored in a 16-way B-tree whose leaves all happen to be 1024 bytes at the moment, and we want to calculate what the line number is at a typical position like byte 474,340,006. Each lowest-level internal node embraces 16384 bytes; each node at the next level is 256 kibibytes; each node at the next level is 4 mebibytes; at the next level, 64 mebibytes; and the single top-level node is the whole gibibyte.

- The first 7 64-mebibyte children of the root node are entirely before that position, and we can use a cached number of newlines stored in the root node for each of those children to add up the number of lines in the first 469,762,048 bytes of the file, leaving 4,577,958 bytes.
- Those bytes contain a single full 4-mebibyte block at the next level;

we can add in its number of newlines, cached in its parent block, leaving 383,654 bytes over.
• Those bytes contain a single full 256-kibibyte block at the next level; we can add in its number of newlines, cached in its parent block, leaving 121,510 bytes over.
• Those bytes contain 7 full 16-kibibyte blocks at the next level; we can add in their numbers of newlines, cached in their common parent block, leaving 6822 bytes over.
• Those bytes contain 6 full 1024-byte leafnodes; we can add in their numbers of newlines, cached in their common parent block, leaving 678 bytes over.
• Finally, we can iterate over those 678 bytes to count the newlines in them, and we have our answer.

So, in total, we had to add up 22 numbers, found in five blocks, and examine 678 bytes of text, totaling about 1 us; and the worst case is only about three times more operations, and the same number of random memory accesses, so about the same time. This is about four or five orders of magnitude faster than just iterating over all the text.

If you insert or delete a newline in this buffer, you need to revise five of those numbers. You can alter the tradeoff slightly — for example, within each node you can cache the prefix sums of its children rather than their raw values, resulting in faster queries and slower updates (worst case with tree height 5 and 16-way blocks, 5 reads and 80 updates), or you can use a binary-tree structure within each block (worst case 20 reads and 20 updates). But the number of random memory accesses stays the same.

Monoidal incremental tokenization

It may not be obvious that syntax highlighting can be incrementally handled in the same efficient way. Syntax highlighting is typically mostly a function of tokenization, which is typically regular except in exceptional cases, such as here-documents in shell or Perl. Regular expressions can be handled by an NFA; the elements of the monoid in question are mappings from sets of NFA states to sets of NFA states, and the monoidal operation is composition of such mappings. Typically any block of text of more than a few hundred bytes has only a few NFA states possible at its end, sometimes only one.

Monoidal incremental layout

Typically the column at which you display a character depends on the font you're using, your wrap width, the kind of wrapping you're using (character, word, or hyphenated, say), and the characters before it on the (logical) line, which may be arbitrarily long. As described in Monoid prefix sum (p. 105), you can efficiently compute this incrementally in the same way. (Occasionally it also depends on the rest of the characters in the physical on-screen line, if you are justifying, or the layout choices of the rest of the paragraph, if you are doing some kind of TeX-like layout optimization.)

Topics

Retro teletext

Kragen Javier Sitaker, 02020-07-18 (updated 02020-07-23)
(18 minutes)

Reading Sowing the Wasteland I thought the TICCET idea of using color TVs and, in the absence of a keyboard, touch-tone telephones as time-shared minicomputer terminals was pretty interesting. But driving a TV isn't trivial; black-and-white is 3 MHz of bandwidth, and a DG Nova isn't really up to synthesizing that in software like an AVR ATMega328 is, much less color. (And this was before VHS and Betamax; even Ampex videotape machines were huge, expensive things that couldn't freeze-frame.) Similarly, recognizing DTMF tones isn't that trivial to do in software either.

And it seems like the system didn't really work out that well:

> But in the event, the Reston system failed to live up to expectations. For all the rhetoric of bringing on-demand education and social services to the masses, the Reston TICCIT system offered nothing more than the ability to call up pre-set screens of information on the television (e.g., a bus schedule, or local sports scores) by dialing into the MITRE Data General computers. It was a glorified time-and-temperature line. By 1973, the Reston system went out of operation, and the Washington D.C. cable system was never to be. One major obstacle to expansion was the cost of the local memory needed to continually refresh the screen image with the data dispatched from the central computer.

But what could it have been? Is there a plausible way that a 1972 computer system could have provided computation on demand to 128 TV-screen terminals?

Sharing character generators among TVs

First, let's reduce the problem a little bit by allowing party-line collaboration. You'd have 128 terminals, but only 32 separate screen images, so when the system was fully used, most people would be sharing a screen with a group of others.

Now let's suppose the screens are each displaying 12 lines of 40-column ASCII text; 40 columns is about the limit of what you can fit into NTSC, and 12 lines (like the VT-50) is about the minimum window that's useful for reading and writing text, although some machines like the original BlackBerry, the TRS-80 Model 100, and Motorola two-way pagers have gotten away with less. If those 12 lines of text are 8 scan lines each, each screen needs 96 scan lines of text. (The other scan lines could be colored with color bars or something.)

Now, NTSC has 483 visible scan lines (out of 525 total), so you have almost precisely one fifth of the vertical span of the screen with text drawn on it. This means that you can reasonably timeshare a single font ROM between five screens, so you only need seven font ROMs to draw 32 screen images. When the font ROM is being read by the character generator for one screen, the other four screens in its group are painting color bars. (They can have staggered VBIs if it's desirable to display the text in the same vertical position on every screen.)

This reduces our screen-painting memory requirements to:

• 7 single-ported font ROMs of 96 5x8-pixel character glyphs, for a total of 26880 bits of font ROM;
• 32 40x12 buffers for the 7-bit characters on each screen, for a total of 107520 bits of RAM;
• some registers for which TV was viewing which of the 32 "channels" and where the cursors are and so forth. A hardware base-address register for the screen buffer might be useful for quick scrolling and quick page-flipping, at least if the page you want to flip to is already in video RAM.

The ROMs and RAMs need to be read very quickly while painting the screen. An NTSC frame is 33.37 milliseconds, so each scan line is 64 microseconds, so each of the 200 pixels across the screen is 318 ns. However, we can transfer five pixels at a time from the ROM, so we have 1.59 microseconds to do it, and we can pipeline that with the following read from the RAM.

40x12 is close to the 40x24 the failed 1979 Prestel system delivered in England, nearly a decade later, but with color, using a set-top box.

This works out to 210 bits of ROM and 840 bits of RAM for each of the 128 concurrent users, or 840 bits of ROM and 3360 bits of RAM for each of the 32 channels. You also need shift registers for the bits of the codepoints on the current text line for each five-screen group, and logic for demuxing the pixels from the character generator to the right NTSC channel, and things like that, but basically the summary is that this is a design that would have been dramatically more economical than VT-52s and things like that.

Generating the rest of an NTSC signal --- the front porches, back porches, and timing --- is of similar complexity to a black-and-white TV set. It's something you can do with a couple dozen transistors, maybe less.

You do need a separate 3-MHz-bandwidth channel for each of the 32 channels, but cable companies were already in the business of multiplexing 32 or 64 or 96 channels onto shitty coax, then filtering and demodulating them at individual TVs. In fact, TVs at the time didn't have the option to take "composite" baseband video input; in the 1980s, my TI 99-4/A came with a cheap RF modulator to modulate its baseband video output onto either VHF channel 3 or 4, and we had to use it. Modulating each of these channels onto a separate frequency wouldn't have added much to the cost.

The 1969 Nova cost US$3995, but US$7995 once you added 8 kilowords of RAM (131072 bits). It had a 1200-nanosecond cycle time, though ROM took only 300 ns; the 1970 SuperNova had not only a 16-bit parallel ALU (four 74181s) but also a 800-ns cycle time, and once it had semiconductor RAM (later in 1970) the RAM cycle time was also 300 ns. This is plenty fast enough to meet the deadlines described above.

This gives us a reasonable guess as to what the required 107 kilobits of character code memory would have cost: about US$3500, about US$110 per channel or US$27 per user. This would have been about two orders of magnitude cheaper than a 1975 not-yet-available VT52, which sold for US$1350 even in 1980 (according to terminals-wiki, anyway).

But would it have been responsive and usable? Touch-tone has a lot of latency, and the Nova wasn't a super powerful machine anyway. If we figure on five memory cycles for an average instruction (typical of microprocessors a few years later) 800 ns per cycle gives us 4 microseconds per instruction, 250,000 instructions per second, a little slower than an 8080. (Wikipedia says the Nova 1200, the original Nova, executed loads and stores in 2.55 us, accumulator instructions like ADD in 1.55 us, DIV in 3.75 us (if present), so this is probably not too far wrong.) If we figure that handling a keyboard interrupt might take 100 instructions, it should still be able to do 2500 interrupts a second, although that seems a bit high for a machine of that vintage. So it might be rough to do, say, interactive word processing on it, but simple calculations and programming ought to be within grasp.

With 32 channels, each channel gets only about 8,000 instructions per second on average, which is not nearly enough; even operations like scrolling the screen would take a noticeable part of a second if the machine were fully loaded. But if most users are idle most of the time, it might be feasible.

8 kilowords of memory divided among 32 channels only gives you 256 words of memory per channel, or maybe 128 words once system software takes up a bunch of space. This is not very much; for example, it's less than the text on the screen. In practice you probably need a full 32 kilowords of memory, a kiloword per user, if you're going to have them pair-programming BASIC or making (not-yet-invented) spreadsheets or something. And that's probably a US$20k machine, plus the US$3500 terminal driver system: US$23500 to drive 32 channels to serve 128 users, US$700 per channel or US$180 per user. With a little thought, the machine could easily have included a bulletin-board system and electronic mail, though entering text on a touch-tone phone is no picnic, especially since this was 27 years before Tegic T9.

I think this would have been a total steal, though I guess it's possible inflation is tricking me.

Suppose you wanted to make it actually cool? Square-wave music like the IBM PC wouldn't have been hard to add, but recording and playing back PCM was probably not in the cards. Broadcasting your phone voice to whoever else was in your group, though, would have been doable in the analog domain. Per-character color would, I think, have been a poor tradeoff, but maybe per-line color would have been adequate.

A light pen would have needed only about microsecond resolution to identify a given letter on the display, but it isn't entirely clear how it would go about transmitting this information over a regular phone line. If there was a way to feed it the front and back porches from the NTSC signal, there might be some hope, but otherwise it seems like whatever internal timing reference it had would drift hopelessly. (This was before the quartz watch revolution.) Encoding its position in a pair of audible tones would not be unreasonable. Nowadays, of course, the whole prospect of a light pen is hopeless with LCD panels. A tone-generating mouse, however, would be entirely usable, both then and now.

Modern AVRs

You could probably build something like this today with an ATMega328 (about 20 times the speed of a Nova but with only 8 KiB of RAM) and the Arduino TVout library for a group of five displays. You could use an analog demultiplexer chip and some 10MHz op-amps (these exist now, though maybe not in 1972) as buffers to put each line onto the right output video signal, and probably bitbang the PS/2 protocol on five keyboards, although it might be hard to meet the PS/2 deadlines when you're stuck in a timer interrupt handler for most of the 64 microseconds.

Slower scanning

Suppose you could use a long-persistence phosphor like the ones conventionally used on analog oscilloscopes, and commonly used on computer terminals at the time (which is why the screens were green.) (This would also make light pens impossible.) Then you wouldn't have to repaint the screen thirty times a second; you could repaint it, say, every two seconds, even without exotic and finicky direct-view bistable storage tubes (DVBSTs) like the Tek 4014 used.

If you try to apply this in a simple way, by sharing the character generator circuitry and ROM between more screens, it doesn't really help, because the main cost of the system described above is really the RAM. But we can use it instead to reduce the amount of RAM needed and increase the system's flexibility, because we don't need special video RAM to feed the character generator at reliably high speeds; we can generate scan lines, vector paths, or at least text lines on the fly from application data. If the computer system runs at 200,000 instructions per second and can devote half of this to generating video signals, and we need to repaint every two seconds, then we only have about 6,300 instructions available per screen repaint (if we are generating 32 channels).

At such a low speed, perhaps the best we could do would be to use a character generator that reads from a specified position in main memory. If we shoot for 12 lines of 80 5x8 characters, like a VT50, per two-second screen, but continue with the 64 microsecond line scan time, then our single character generator can drive 325 slow screens, which greatly exceeds the memory capacity of the computer to do anything useful with.

Suppose instead we shoot for 1-second updates of 32 24-line screens. That's 6144 total scan lines, one every 0.163 ms; once every 8 scan lines (1.3 ms, 260 instructions) we need to update the character generator's line-start pointer. That's still probably too much load on such a slow computer as the original Nova, but it's within the bounds of plausibility. It would be straightforwardly achievable on the 300-ns-cycle 1970 SuperNova if using SRAM instead of core.

Memory access contention might be another issue: if the character generator doesn't have its own internal buffer for one line of bytes, it has to read a character from main memory every 5 pixels, generating 8x as much memory traffic. If it only reads 80 bytes every 1.3 ms, at 300 ns per 16-bit word (which I said you probably need anyway) it would need to use memory for 12 microseconds out of the 1300 to read them, and even with 1200 ns core it would only need 48

microseconds. Without the internal buffer these numbers go up to 96 microseconds and 384 microseconds respectively, the second one amounting to about a third of the total memory bandwidth and thus having a significant, highly undesirable impact on the CPU's speed. Moreover, it would also need strong guarantees of timeliness — it wouldn't be able to tolerate any extra memory latency, so it would need to have priority over the CPU. The 80 bytes of memory would cost about US$39 if they cost the same as the core memory add-on for the Nova described earlier, but probably in practice you'd have to use semiconductor memory, which would cost a few times more. This would clearly be a good tradeoff for 7% of the whole computer's performance.

It's probably worthwhile actually to stick the whole array of line-start pointers in main memory instead of trying to update a character generator register from an interrupt handler thousands of times a second. There are 768 of them, which would amount to 1536 bytes if they were in an array, some 1% of all of RAM, which is reasonable. (If all of the monitors had unique text on all of their lines, we could dispense with the pointers, but that would be 61 kilobytes, 47% of RAM. So it's probably necessary to have some degree of sharing in order to free up space for application data; the array of pointers is the easiest way to do this. This could be as simple as some blank lines.)

The modern inversion

So much for 1972. Now it's 2020, 48 years later, and TS-80P soldering irons routinely have STM32F microcontrollers in them: 48–72 MHz, a 32-bit parallel ALU, RISC with nearly one instruction per clock, 32-128 KiB of Flash, maybe 20 KiB of RAM, hardware multiply, hardware floating point in some cases, 1500 pJ per instruction; maybe US$2 in quantity 1. That's about the same amount of Flash as the Nova's typical RAM, plus a somewhat smaller additional amount of RAM. What can we do with that?

Well, there's no need to use character generators, that's for sure. You can bitbang NTSC no problem: a 64-microsecond scan line is 2000–5000 32-bit instructions instead of, like, 13 16-bit instructions. You can bitbang *color* NTSC, which is beyond the capacity of an AVR. You can bitbang multiple NTSC composite signals in parallel.

If we crudely estimate that US$180 per user in 1972 is equivalent to about US$3600 today — reasonable based on gold and petroleum prices, though the CPI would suggest more like US$1800 — then we can afford some 1000–2000 microcontrollers per user, tens of megabytes of SRAM, tens of billions of operations per second. You could reasonably dedicate a microcontroller per scan line on an NTSC or even megapixel screen, if that would be a helpful thing to do, which it probably isn't.

Probably a more useful approach is, instead of only interfacing to humans through the physical objects that are easiest to interface through, such as televisions, to attempt to interface though objects that are more difficult, compensating for the difficulty to some extent through software. This involves using control systems with the available actuators to structure the objects so they are usable as further

transducers. Digital fabrication, including both shaping processes (subtractive, additive, deformation) and assembly processes (welding, soldering, screwing), enables the creation of objects with enormously more transducers than the simple vacuum tube that is a 1972 television.

Computation and control have become cheap; we need to leverage that into cheap actuation and cheap sensing.

Topics

The orbital drive and stepped planetary drive

Kragen Javier Sitaker, 02020-07-28 (updated 02020-08-02) (10 minutes)

I recently saw an amusing YouTube video of something called an "orbital drive", by "Skyentific"; it's a sort of differential planetary pulley without a ring gear, where a motor spins a planet cage around two sun gears of different sizes, which are connected to the planet idlers with belts. The sun gears are planar, coaxial, and in parallel planes, while the planet gears span both planes. One sun pulley is held fixed, while the other is free to rotate, one tooth per cage revolution if the two suns differ by one tooth (and the planets don't change tooth count between sun planes). It's claimed to be backlash-free (because it uses pulleys, I suppose) and of course because it is differential it has a high reduction ratio, in the neighborhood of 100:1.

The Wikipedia article on epicyclic gearing points out that, if you use gears instead of pulleys, you can use two rings instead of two suns, both simplifying hooking up the assembly and reducing its size, though perhaps at the cost of requiring the planets to change size between the sun planes. ("During World War II, a special variation of epicyclic gearing was developed for portable radar gear...")

It occurred to me that if you use only a single planet, it can perhaps be quite large compared to the ring gears, and you can cut a third ring gear into the center of it which you drive with a small pinion, thus gaining a further reduction without increasing the size of the assembly. Because this pinion and ring are subject to much smaller forces than the other gear teeth, they can be much thinner.

To be concrete, consider the case where the ring gears have 103 and 106 teeth, the two steps on the planet have 69 and 71 teeth, the inner ring on the planet has 50 teeth, and the pinion that drives it has 7 teeth. (Using involute teeth the depthing cannot be correct for both the 69:103 mesh and the 71:106 mesh, but the difference is about 0.014%, so it's tolerable. Hmm, can you even use involute teeth on a ring gear?) Let's consider one revolution of the 106-tooth ring in the rotating frame of reference of the planet "cage". The 106-tooth ring and the 71-tooth planet each rotate 106 teeth. The 69-tooth planet rotates 106*69/71 = 103.014 teeth.

Wow, I didn't expect THAT. Is that real? Hmm, consider one rotation of the planet: 71 teeth on one step, 69 on the other, resulting in 71/106 rotation on the 106-tooth gear and 69/103 rotation on the 103-tooth gear, about 0.009% of a rotation difference between them. Ratio this up via brute force: 106 rotations of the 71-tooth gear, for a total of 7526 teeth of rotation in that plane, rotates the 106-tooth ring 71 times; the same 106 rotations of the 69-tooth gear are 7314 teeth, which work out to 71.0097 rotations of the 103-tooth gear. Let's consider 103 times that: 10918 rotations of the 71-tooth gear are 775178 teeth, 7313 rotations of the 106-tooth gear and 10918 rotations of the 71-tooth gear. Those same 10918 rotations of the 69-tooth gear give us 753342 teeth of rotation in its plane, driving the 103-tooth

gear through 7314 rotations. Seems legit: a 10918:1 reduction in the differential rotation! So let's continue.

But this seems impossible; you would think that the planet would have to return to its initial position after 106 rotations. Like, if you mark the most-meshing tooth on it at the beginning, and also mark the corresponding space between teeth on the 106-tooth ring, then after 106 rotations you would think it would have to come back to rest in exactly the same marked place on the ring, which means that the 69-tooth planet is also in exactly the same place relative to the 106-tooth ring, since it's rigidly fixed to the 71-tooth planet. So how could the 103-tooth ring be displaced by a fractional tooth?

Now, each revolution of this planet is 50 teeth on its inner ring, which is 50/7 rotations of the pinion, coaxial to the outer rings, that drives it. This provides a further reduction of 7:50, for a total of 7:545900, or about 1:77985.7.

But that's in the frame of reference of the planet cage. Let's switch to the frame of reference of the 106-tooth ring gear and let the planet cage spin. Every time the planet rotates through 106 teeth (and 106/71 rotations) in its own frame of reference, the planet carrier rotates by one revolution in this frame of reference. Its inner ring rotates by 50*106/71 teeth in its own frame of reference; from this we must subtract the single rotation of the frame of reference itself, so 50*(-1 + 106/71) teeth, which works out to about 24.64789 teeth. This is about 3.52113 rotations of the pinion.

Remember that, if the dubious and probably wrong calculations above are correct, the reduction is only 7314:1 from the perspective of the 106-tooth ring — that is, every time the 106-tooth ring rotates 7314 times in the frame of reference of the carrier, the 103-tooth gear rotates 7313 times. So this is the appropriate multiplier for the pinion relative to the 106-tooth ring: every time the pinion rotates 3.52113 times, the planet cage rotates once, and the 103-tooth gear rotates 1/7314 of a revolution, for a total reduction factor of only about 1:25753.5.

Do the geometries work out? Suppose we use a tooth module of 2 mm for the outer rings and the planet teeth that engage them and 1.5 mm for the inner ring and pinion. Then our circumferences are respectively 212 mm and 206 mm for the outer rings and 142 mm and 138 mm for the inner rings, so the diameters of the pitch circles are 67.4817 mm, 65.5718 mm, 45.2000 mm, and 43.9268 mm, and their radii are 33.7408 mm, 32.7859 mm, 22.6000 mm, and 21.9634 mm. So the center of the 71-tooth planet would ideally be at 11.1408 mm from the center of the 106-tooth ring, and the center of the 69-tooth planet at 10.8225 mm from the center of the 103-tooth ring, a difference of 318.3 microns. This is probably sufficient for reliable meshing, but will definitely introduce an undesirable amount of backlash, and only one tooth will be in contact at a time on the 69:103 plane. If this is unacceptable, it might be feasible to have the 103-tooth ring revolve in a circle with that 318-micron radius, although that would be a lot more reasonable if the difference were in the opposite direction.

Then the inner ring's pitch circle is 75 mm in circumference, and the inner pinion's 10.5 mm, giving pitch circle radii of 11.9366 mm

and 1.67113 mm respectively. This means that the pinion center will be 10.2655 mm from the planet and inner ring center, which is almost a full millimeter off the desired outer ring center, which is 11.14 or 10.82 mm from the planet center, as calculated above. This can be fixed easily enough by using a slightly larger module (this gear with a ring gear cut into its inner surface will not be a standard part anyway) or slightly more teeth on the inner ring. With lantern gears (p. 165), where the teeth of the inner ring are round dowels rather than normal teeth (feasible since they can be anchored laterally to the bottom of the 69-tooth planet layer) pinions with as few as three teeth are feasible.

To keep the edges of spur-gear teeth in different layers from rubbing on one another, it may be desirable to separate the two planes, either with a mere spacer or with a solid circle that extends out past the teeth of either planet. If the circle is large enough, it could extend out past the teeth of the ring gear as well, preventing any edge-on-edge contact.

The same differential principle can be applied to get larger reductions from the original "orbital drive" without sacrificing the use of toothed pulleys: by increasing and decreasing the planet sizes nearly in proportion with their respective suns, we can achieve very large reductions indeed, far less than the one or two teeth per revolution delivered by harmonic drive (strain-wave drive) or cycloidal reducers.

The large unbalanced mass of the single planet may be a problem, as it is in cycloidal drives, where it is conventionally balanced by a second cycloidal drive in half-phase with the first. However, the initial 1:3.52 reduction from the pinion reduces this problem; the angular velocity of the planet is 3.52 times lower than it would be for a cycloidal drive driven at the same speed, and its acceleration is thus some 12 times lower than it would be if you drove the planet cage directly. However, the planet's center of mass is considerably further from the center of the ring than the thing that moves around in a cycloidal-drive system, compensating somewhat for this advantage.

Topics

• Mechanical things (p. 795) (19 notes)
• Gearing (p. 888) (3 notes)

Fossil geothermal

Kragen Javier Sitaker, 02020-08-02 (updated 02020-11-13)
(12 minutes)

In addition to the fossil fuels that powered Song China and the Industrial Revolution, Earth has stored an even larger amount of energy as "fossil heat": heat that has been produced in the crust by crustal radioactivity over its 4.6-billion-year lifespan (so far) that has not yet had time to escape to the surface. Additionally, an even larger amount of heat energy is stored in the mantle and core from Earth's initial formation, produced by the gravitational potential energy of the matter that formed it.

This fossil geothermal energy, if extracted at an unsustainable rate, could provide orders of magnitude more power than combustible fossil fuels ever have; moreover, exploiting it would release no carbon dioxide, only heat.

Economics and current outlook

Currently photovoltaic power is so inexpensive (€0.17 per peak watt, working out to about €0.85 per average watt at a typical 20% capacity factor) that it is uneconomic to build heat engines to produce power, whether to harness heat from fossil fuels, from nuclear energy, from solar concentrators, or from such geothermal sources. If new manufacturing technology or new heat-engine designs can reduce the cost of heat engines to below the cost of PV cells, it could enable the exploitation of this fossil energy. Otherwise, it is unlikely to happen before most of Earth's insolation is being converted to electricity.

There is probably no profit or economic incentive to go underground other than as a temporary measure. However, it would provide a measure of security against potential global disasters such as comet strikes, Carrington-class events, global totalitarian dictatorships, global thermonuclear war, pandemics worse than covid, and the like; whatever could survive independently underground would be relatively safe from such events.

Ultimately available geothermal energy

Historically, geothermal energy has only been available in hotspots with existing water reservoirs. So-called "hot dry rock" or "enhanced geothermal systems" geothermal involves hydrofracking of deep crustal rock and pumping water through it; this can be done anywhere on Earth.

To give round numbers, the whole mantle is at 1000° or more, has a specific heat of about 0.7 J/g/K, and weighs about 4×10^{24} kg; this amounts to a thermal energy of some 2.8×10^{30} J relative to the temperature at the surface, and so perhaps 1.1×10^{30} J of energy practically extractable at 40% Carnot efficiency. (In fact, the innermost part of the mantle is closer to 3700°, so this is a conservative estimate.) If extracted over 1000 years, this would amount to 35 exawatts. By contrast, total terrestrial insolation at the usual standard "solar constant" of 1000 W/m² is only 0.13 exawatts,

about 250 times smaller.

(The specific heat of the mantle is fairly uncertain. The work I've been able to find suggests that the specific heat of $CaTiO_3$ perovskite is in the neighborhood of 0.5 to 1.0 J/g/K depending on temperature, while $CaSiO_3$ [calcium metasilicate] and $MgSiO_3$ perovskites, which compose much of the mantle, have a heat capacity in the range of 75–125 J/mol/K. I figure calcium is 40, magnesium is 24, silicon is 28, and oxygen is 16, so those are 100–116 g/mol, which is in the range of 0.7 to 1.25 J/g/K. Regardless, most things have a specific heat of around 1 J/g/K, water being a bit of an outlier at almost 4.2, and heavy monatomic gases like xenon being a bit of an outlier in the opposite direction at about 0.1.)

A heat engine requires a hot reservoir and a cold reservoir, but the cold reservoir need not be the surface of Earth; a larger volume of rock at a shallower depth would also suffice.

Enabling human survival underground

So a subterranean civilization, if it existed, could reach Kardashev Level 1 without going above the surface. But could it exist?

The humans' survival has a number of prerequisites other than energy. They need cool, oxygen, nutritional compounds, gravity, water, quiet, sleep, love, beauty, a sense of purpose, a relatively chemically inert environment (lacking, for example, hydrogen sulfide or chlorine), waste disposal, space, low pressure, and probably light.

XXX restructure this part

They can only directly harness energy provided chemically, the most practical form of which is to grow plants, which need most of the same things, also provide nutritional compounds, and definitely do need light.

Cool can be provided in an underground chamber by insulating and refrigerating it, pumping the heat into a cold reservoir elsewhere. Oxygen can be extracted electrolytically from oxygen-containing rocks, which is most of them. Gravity is unavoidable on or in Earth. Water is abundant in the crust down to at least several kilometers; the Kola Superdeep Borehole found that in that location hydrogen was abundant even deeper than that, although perhaps that suggests that oxygen wasn't. Quiet is the default state underground, though soundproofing might be needed in the vicinity of heavy machinery. Sleep, love, beauty, and a sense of purpose can be constructed by the humans themselves. A chemically inert environment might use nitrogen, which is relatively scarce underground, or helium, which is abundant.

Space can be provided by producing oxygen from oxygen-containing rocks, as described earlier, and pumping it closer to the surface. The oxygen will either oxidize other rocks, if there are any nearby that aren't already fully oxidized, or bubble to the surface harmlessly. The reduced rocks will occupy less space than the original rocks. Alternatively, if there is access to the surface, spoil can be pushed to the surface, and especially at shallow depths it may be possible to uplift an area of land to create space beneath it — the reverse of the subsidence often associated with, for example,

brine-based salt mining.

(Neal Stephenson explored this theme fictionally in his novel *Seveneves*, in which he posited that space underground could not be expanded, so his hypothetical underground civilization had to make do with the space that had already been excavated before the disaster the novel is built around.)

Waste can be disposed of by recycling, which is mostly a matter of separating wastes of unknown composition into their ingredients, or by isolation, which is mostly a matter of keeping wastes of dangerous composition away from the humans and their equipment, consuming space. Aboveground there is no shortage of space; belowground, generally whatever material is used must first be mined. If the waste can be melted into fully dense solids, it need occupy no more space than the original rocks from which it was mined, but that might turn out to be more difficult than just making more space to store looser waste in.

Low pressure is scarce underground, and the details depend on the circumstances, but it can generally be provided by supporting the rock above a cavern with materials of greater compressive strength than the other rocks. If they have 10% more compressive strength, they enable you to fill 9% of the space with air; if twice the compressive strength, half; if ten times the compressive strength, 90%; and so on. Salt poses special problems, as it tends to flow horizontally back into open spaces, but this takes decades or centuries; other rocks will behave similarly at sufficiently high temperatures.

Light, air purification, food cultivation, air conditioning, cooling, oxygen production, and rock electrolysis will all consume energy and require specialized equipment.

Current tech limitations: ≈2× the size of the current economy for a century

Much of the above calculation of geothermal energy abundance isn't concerned with current technological limitations, but with the ultimate limitations. What's accessible within current limitations?

The amount of thermal energy in the crust is considerably smaller than the amount in Earth as a whole; the temperature at the Moho crust–mantle boundary ranges from 200° to 400°, and the crust is only some 1% of Earth's mass, so we're talking about maybe 10^{28} J in the crust. So far, despite 63 years of effort, the humans have not been able to drill into the mantle; the Kola Superdeep Borehole ("Кольская сверхглубокая скважин") only reached 12.3 km of drilling depth before being doomed by the 180° temperature found there and the collapse of the USSR. (The crust is typically 30–50 km thick on continents, 5–10 km thick in the ocean.) The KTB superdeep borehole persevered until reaching 260° at only 9.1 km of depth. These temperatures are suboptimal for driving heat engines, since water's critical point is 374° and 22 MPa, but nevertheless clearly quite feasible.

Suppose we can routinely access the top 11 km of continental crust, and that it's routinely 210°, in between the Kola numbers (14 km, 180°) and the KTB numbers (9 km, 260°), and that temperature

increases linearly from here to there, which is conservative. Ocean covers 71% of Earth, so the continents are about 148 million km², 1.48×10^{14} m². Rock is about 2.4 g/cc so these top 11 km are about 3.9×10^{18} tonnes of rock. If it were all 210° and 0.7 J/g/K, the thermal energy to drop it to 20° would be 5.2×10^{26} J, so a linear increase gives you half that, 2.6×10^{26} J. Rather than actually calculating the Carnot efficiency, I'll just assume it's about 25%, giving 6.5×10^{25} J electric. If that were to be extracted over the next century, it would yield almost 21 petawatts, electric, or 620 000 "quadrillion BTU per year" (electric) or 180 million terawatt hours (electric) per year, in the medieval units used by the IEA.

XXX https://en.wikipedia.org/wiki/World_energy_consumption say 18 terawatts. That means this is not "about twice world marketed energy consumption" but rather about 1200 times. Also usually geothermal people only consider the top 6 km reasonably usable with modern technology. FEEX

This is about twice current world marketed energy consumption, but that doesn't include sunlight on fields, which a purely subterranean civilization would need to include.

This should be sufficient to develop technology for deeper drilling and/or Dyson-sphere construction.

It's plausible that the amount of available energy with current technology is a few times larger than this, because the above does not take hotspots and tectonically active zones into account, nor the ocean floor.

Earthquakes

Enhanced geothermal systems projects in Pohang and Basel have been canceled after causing earthquakes locally; in Pohang no humans died but more than a hundred were injured, though in both places the earthquakes were fairly minor. We can expect that widespread use of EGS would produce widespread minor earthquakes, even as it depletes the source of energy that drives volcanism and seismic activity.

Even if it does not pose a risk to surface civilization, for example because of being located far from surface cities, this induced seismicity would be clearly detectable from the surface, while the tunnels and increased oxygen emissions probably would not. In places with little natural seismic activity, it would be more conspicuous than in places with a great deal.

Topics

- Energy (p. 812) (11 notes)
- Mole people (p. 941) (2 notes)
- Geothermal

Pyrolysis 3-D printing

Kragen Javier Sitaker, 02020-08-02 (updated 02020-11-24)
(20 minutes)

I heated up the tip of my zirconia knife to orange heat yesterday. To my surprise, much of the blade turned black; I guess the knife had some oil on it, though I thought it was clean. In the hotter parts of the blade, this carbon deposit burned away, but in the cooler parts it remained. Steel wool and brass wool proved ineffective at removing the deposits.

This led me, as most things do, to thinking about 3-D printing. Suppose instead of depositing a hot liquid onto a cool workpiece which then freezes the liquid in place, as FDM printers do, we deposit a *cool* liquid onto a *hot* workpiece, which pyrolyzes it into a solid? Sort of like chemical vapor deposition, but from a liquid so you can deposit it selectively?

Charring organics into carbon

So, for example, you could deposit warm bitumen and pyrolyze it to carbon at around 350°.

Almost any organic substance would work; so, for example, vegetable oil, sugar, starch, and dissolved gelatine would all work, but possibly other things would work better. Small molecules tend to volatilize before carbonizing (though any cook can tell you that even light vegetable oils will carbonize before volatilizing completely, though perhaps not before migrating to cooler parts of a surface where you don't want them), molecules with a lot of oxygen tend to produce more bubbles, and highly saturated molecules (like the ones in bitumen) and aromatic molecules are more resistant to pyrolysis, so perhaps the ideal substance would be a high-molecular weight, highly unsaturated aliphatic hydrocarbon. (However, aromatic groups tend to promote cross-linking, which helps to prevent volatilization and melting before pyrolysis.)

Moreover, it would at least remain viscous at pyrolysis temperatures, like polycaprolactone (not to be confused with polycaprolact_am_, which is common nylon 6 and not viscous at all), though polycaprolactone itself is saturated and contains oxygen. Something that solidifies before reaching pyrolysis would be even better (see below about polymer-derived silicon-based ceramics.) Somewhat polymerized linseed oil is a good possibility. Nylon 6,6 and nylon 6 are not very viscous or unsaturated but are a good possibility; their amide bonds could play the role of the unstable unsaturated bonds. The urethane groups of polyurethanes contain both double bonds and amide bonds, making them especially promising, though the popular polyurethanes are highly aromatic. Polyisoprene, especially if vulcanized into a thermoset with sulfur, would work perfectly.

Decomposition can be accelerated with additives; PET notoriously takes up water from humid air and then hydrolyzes rapidly at melt temperatures if not dried, so ordinary water may be a viable option,

despite its tendency to produce large bubbles. Acids and bases may also help to accelerate such decomposition — ideally for this process the additive would itself volatilize or decompose, leaving only carbon. Ammonia, sulfuric acid, nitric acid, acetic acid, formic acid, and hydrogen cyanide are possible degradation-enhancing additives. Cellulose acetate has a well-known autocatalytic degradation reaction with acetic acid, but this produces a goo which may be too liquid at pyrolysis temperatures.

Additives like alkali metals and halogens might accelerate decomposition as well, but would probably remain in the final product.

Using thermosets such as the aforementioned vulcanizing polyisoprene has the advantage that you don't have to worry about the feedstock melting and running off the workpiece before charring, so you don't need to prefer unsaturated aliphatic compounds. Normally thermoset polymerization is tightly controlled to reduce the risk of heat degradation of the material produced, but in this context that ceases to be a problem. So any of the common thermosets — phenolics, epoxies, polyisocyanurates, urea resins, thermosetting polyesters like Lucite, melamine resin — should be fine. Thermosetting is the approach universally taken for preceramic polymers used for producing silicon-based ceramics.

Charring polymers into silicon carbide, silicon nitride, and silicon oxynitride

People have already done this since the 1970s, though, until recently, mostly to produce fibers such as Nicalon and Tyranno, and coatings. The technique is called "preceramic polymers", "precursor ceramics", or "polymer-derived ceramics", though typically in that technique the polymer shape is fully formed before pyrolysis begins — an approach that can be taken for all of the materials discussed in this note, including carbon and the metal oxides discussed below.

Large-molecule silicones are usually thermosets. Polydimethylsiloxane ($(SiO(CH_2)_2)_n$) has a 2-to-1 carbon-to-silicon ratio, which is twice the ideal for producing silicon carbide, so polymers that have been used instead include carbosilazane resin, poly(methylsilazane), poly(methylchlorosilane), and poly(carbosilane), which pyrolyze in nitrogen to silicon carbide, yielding a ceramic whose mass is some 60–75% of that of the original polymer (the "ceramic residue yield").

An excess of silicon is preferable to an excess of carbon for producing high-temperature ceramics, since silicon melts at "only" 1414°, (XXX Cold Plasma (p. 560) says 1460°) while carborundum is stable to 2830° and graphite is stable to 3642°. Poly(carbosilane), $(H_2SiCH_2)_n$, pyrolyzes in nitrogen to essentially pure carborundum, but in other precursors some carbon is lost as methane or carbon monoxide during pyrolysis.

To get silicon nitride instead, an ammonia atmosphere is required both to supply more nitrogen than can be jammed into the polymer and to cleave off unwanted methyl groups. It is helpful but not

necessary for the original polymer to contain nitrogen; in fact, ammonia pyrolysis can convert Nicalon to silicon nitride.

An attractive aspect of these processes is widely reported to be that low temperatures, in the 1100°–1300° range, are sufficient to produce these ceramics by pyrolysis, while the standard sintering processes require 1800° or more, and additionally contaminates the ceramic produced with "sintering aids", in order to avoid even higher temperatures. So not only can polymer-derived ceramics withstand higher temperatures than are required for their production, they can even withstand higher temperatures than the same ceramics when they're processed conventionally!

Some of these processes require a "curing" step in between plastic forming, such as spinning, and the pyrolysis step, in order to keep the plastic from melting before pyrolysis is complete. This curing may happen by way of cross-linking, as in rubber vulcanization, or by evaporation of solvents and other plasticizers. This approach is also applicable to pyrolytic carbon production described above.

A problem that commonly afflicts these processes is structural damage due to pyrolytic gas production during pyrolysis, which is a major reason for requiring high ceramic residue yields. As with traditional fired-clay ceramics, this is a bigger problem with thicker material sections (nonexistent below a few hundred microns), and it is to be expected that an incremental additive process in which the material is pyrolyzed before more material is laid on top of it should enable the fabrication of thicker cross-sections.

Another problem that commonly afflicts these processes is dimensional inaccuracy due to shrinkage during pyrolysis, and deposition during pyrolysis will also reduce this problem, since the shrinkage will affect individual beads as they are laid down, not the fabricated article as a whole, whose dimensional precision will be determined grossly by the positioning precision of the end-effector and only finely by shrinkage and wiggle.

Of course, there are certain practical difficulties attending the construction of a "hotend" and manipulator that can function in an environment that keeps the workpiece at 1100°–1300°. A combination of liquid-cooling and refractory insulation for a manipulator arm would probably be necessary. The thermal gradient near the deposition point poses additional difficulties: the cooler material being deposited onto the hot workpiece will locally cool and contract the workpiece, inducing stresses that could produce cracks.

Boron nitride, aluminum nitride, boron carbonitride, silicon oxycarbide, silicon carbonitride, SiCNO, SiBCN, SiBCO, SiAlCN, and SiAlCO have also been synthesized by this route. Some of these ceramics cannot been synthesized in any other known way.

Exposure to reactive substances has been used instead of or in addition to heating to remove the unwanted moieties. Examples include ammonia, nitrogen dioxide, reactive plasma, and highly alkaline solutions. These approaches could likely also be used with the other materials discussed in this note; incremental deposition of the feedstock, as by fused deposition modeling, would give the reactive environment access even to material that is ultimately buried inside the part.

Of special note here is HRL Laboratories' high-density stereolithography resin, which produces almost-fully-dense silicon oxycarbide when pyrolyzed at 1000° in argon ("Additive manufacturing of polymer-derived ceramics", Science, January 2016, many authors and Tobias Schaedler). Their recipe is mercaptopropyl methylsiloxane and vinylmethysiloxane (in a 1:1 molar ratio of thiol to vinyl groups), plus the usual cocktail of stereolithography additives; pyrolysis resulted in "42% mass loss and 30% linear shrinkage" to amorphous $SiO_{1.34}C_{1.25}S_{0.15}$ but apparently no porosity or surface cracks. To reduce porosity and cracking, they limited feature size to 3 mm and heating to 20°/minute (or, according to their supplemental materials, 1°/minute), but it is not clear to me what the crucial factors were.

Metal and semimetal oxides

(For the purpose of the following, consider "metals" to include boron, silicon, and aluminum as well as the usual metals.)

A number of metal oxides form minerals with desirable properties, and it might be desirable to form them into particular shapes; many of these metal oxides are themselves refractory and chemically resistant, so casting or dissolving them is difficult. In particular, the oxides of aluminum, zirconium, silicon, titanium, chromium, thorium, and uranium are all hard, refractory ceramics, most occurring naturally as minerals.

But perhaps salts of the same metals can be formed into the right shape, whether as an solution (for example in water), a gel, a paste, or as solid particles; then calcined to yield the oxides? As the preface to the IUPAC–NIST Solubility Data Series volume on formates said in 2001:

> Bivalent metal formates could be used as precursors for the production of catalysts because they show excellent miscibility in the solid state, i.e., they form mixed crystals that dissociate at relatively low temperatures (about 300 °C) to form the respective oxides and mixed oxides. Catalysts for the decomposition of alcohols have been prepared by the thermal decomposition of Ni and Mg formate mixed crystals, from Cu and Mg formate mixed crystals, and from the double salts $CuSr_2(CHO_2)\cdot 8H_2O$ and $CuBa_2(CHO_2)_6\cdot 4H_2O$. ...

For this we need metal salts that decompose on heating, but ideally are soluble in water (IUPAC-NIST database); moreover, they probably need to be soluble *together* so they don't precipitate in the nozzle. Basically these are either metal cations with anions that decompose on heating — nitrate, sulfate, or organic anions — or ammonium or hydronium with metal-complex anions. Tetramethylammonium is a possible alternative cation but for now I'm going to ignore it. Here's a list of candidates.

cation	anion	g/100g water (20° if possible)	decomposition temperature
aluminum	nitrate	74	150°
	sulfate	36	900°
	formate	6	
chromium(III)	nitrate	81	100°

	sulfate	"readily"	700˚ (to acid)
ammonium	dichromate	high	
	paratungstate	high?	600˚
(hydronium)	boric acid	low	
	chromic acid	169	
	alumic acid		
	tungstic acid	low	
	titanic acid		
	zirconic acid		
calcium	nitrate		500˚
	acetate		160˚
	formate		200˚?
	sulfate	≈0 ☹	
magnesium	acetate	53	
	oxalate	low	620˚
	chromate	137	
	formate	14	
	nitrate	69.5	
	sulfate	35	
zirconium	sulfate	52.5	
	nitrate	yes	100˚
	acetate	?	
	formate	?	
	tungstate	low	
titanium	sulfate	?	
	nitrate	?	
	formate	?	
	acetate	?	
cobalt	nitrate	84	
	sulfate	less	
copper(II)	nitrate	83.5	
	sulfate	very	
	formate	7	
ferrous ammonium	sulfate	27	
iron(II)	nitrate	134	
	sulfate	29	680˚
	oxalate	poor	
iron(III)	nitrate	138	
	sulfate	slight	
	oxalate	slight	
	chromate	decomposes	
lead(II)	acetate	44	
	nitrate	54	
	sulfate	≈0 ☹	
lead(IV)	acetate	"reversible hydrolysis"	
nickel	acetate	high?	
	nitrate	94	
	sulfate	44	
	formate	low?	
tin(II)	sulfate	19	
	nitrate	?	
yttrium(III)	acetate	9	
	formate	26 (at 50˚)	
	nitrate	123	

	sulfate	7	
zinc	formate	5.2	
	nitrate	98	
	sulfate	54	
	acetate	30	
thorium(IV)	nitrate	191	
	sulfate	≈0 ☹	
uranyl	nitrate	122	
	sulfate	21	
	acetate	8	

I can't find any concrete information about ammonium aluminate; I suspect it doesn't exist, although a number of chemical suppliers have it in their catalogs. Ammonium silicate apparently does exist but is too finicky to be of any practical use. Ammonium borate also seems to exist, but information about it is rare.

Tetraethyl orthosilicate is commonly used in a way similar to this to produce silica gel, but it is itself liquid rather than being water-soluble, and its decomposition is driven by exposure to water, not to heat.

Halogen complexes might be another thing to check out: titanium and zirconium both complex with halogens, and it may be possible to drive off the halogens with enough heat.

Glasses (frits) of metal oxides melt at lower temperatures; may be suitable fillers

Titanium, zirconium, aluminum, magnesium, chromium

Aluminum: nitrate (74%, decomposes at 150°) and sulfate (36%, decomposes below 900° to SO_3 and cubic γ-alumina) are highly soluble. Also occurs in soluble aluminates, but there is no aluminate of ammonia, so you can't get alumina by calcining it; strontium aluminate is a glow-in-the-dark pigment and a refractory cement good to 2000°.

Chromium: ammonium dichromate is fairly soluble. Chromium(III) nitrate and especially sulfate are highly soluble; hexavalent chromium oxide too, but we don't want that.

Boric acid is fairly soluble in water at 100°, nearly half as much at 50° (13% or so)

Calcium: nitrate is highly soluble, decomposes at 500°; soluble acetate releases acetone at 160° leaving carbonate; soluble formate decomposes at 300°, maybe to CaOH and CO, or like NaCOOH to an oxalate (CaC_2O_4, insoluble) and hydrogen (at 360° for Na?), then to a carbonate releasing carbon monoxide (at 290° for Na, 200° for calcium oxalate)? Calcium will precipitate lots of things including sulfate.

Magnesium acetate 53%; chromate 137%; formate 14%; nitrate 69.5%; sulfate 35%.

Zirconium: sulfate 52.5%. Nitrate has been successfully calcined to produce zirconia: https://pubs.acs.org/doi/10.1021/cm060883e

Titanium:

cobalt? vanadium? manganese? nickel? copper? zinc? tin?

bismuth? strontium? barium? lithium?

Cobalt nitrate is 84% soluble in water; sulfate a bit less so.

Copper(II) nitrate is 83.5% soluble in water, substantially more than sulfate.

Ferrous ammonium sulfate is 27% soluble. Iron(II) nitrate 134%; sulfate 28%; iron(III) nitrate 138%.

Lead acetate 44%; lead(II) nitrate 54%. Lead(II) will precipitate sulfate.

Lithium acetate 40.8%; nitrate 70%; sulfate 34.8%; tartrate 27%.

Nickel acetate "easily soluble", nitrate 94%; sulfate 44%; everything else pyrolyzable almost insoluble. (Its highly soluble chloride is not relevant.) Hexaamminenickel chloride is soluble in anhydrous ammonia and decomposes with heat, presumably to yield nickel.

Ammonium paratungstate pyrolyzes to tungsten trioxide at 600°, which is the soft mineral tungstite; the paratungstate ion has a tendency to precipitate from aqueous solution over time. There's also a "metatungstate" oxyanion with 12 tungstens in it which is more soluble and stable in highly acidic solution.

Tin sulfate 19%; nitrate?

Yttrium: Yttrium(III) acetate 9%, nitrate 123%, sulfate 7%.

Zinc: formate 5.2%, nitrate 98%, sulfate 54%, acetate 30%.

Uranium, thorium

Thorium: thorium(IV) nitrate 191%, sulfate almost insoluble.

Uranium: Uranyl nitrate 122%, sulfate 21%, acetate 8%.

Filled systems

A common use for preceramic polymers, apart from the fibers and coatings mentioned earlier, is as polymeric binders for powdered ceramic — perhaps the same ceramic that the polymer will pyrolyze to. Filled systems like this have a variety of advantages:

- The resulting part, if the filler consists of fully-dense particles of the same ceramic, is denser and therefore stronger than if made entirely from the polymer.
- The powder may be easier to produce than the polymer, reducing cost, or even naturally abundant, as with quartz.
- Controlled composites of different materials, or materials of different morphologies, can be thus produced; this may improve mechanical properties or simply be cheaper.
- Other filler powders can be used to provide other properties; for example, early-90s research at MIT (Seyferth et al., 1992))produced silicon-carbide/metal-carbide composites from poly(methylsilane) and organometallic polymers, but found it necessary to mix in metal powder to eliminate free carbon from the pyrolysis result.
- The grain structure of the resulting material can be controlled more precisely and customizably than that of objects made by liquid casting or sintering.
- The filler may be adequate to maintain the shape of the material as it heats up to the pyrolysis temperature, even if the liquid does not

cross-link to form a thermoset.
• The filler may provide egress paths so the gases evolved during pyrolysis don't crack the nascent ceramic structure, even if the filler itself burns out before pyrolysis of the ceramic precursors, thus forming a porous green structure. This is the "Ceramicore" process by Weifeng Fei; it's also a traditional way of making insulating refractory bricks from fired clay and organic fillers like sawdust, but Fei's process infuses a liquid preceramic polymer into a continuous fibrous matrix.

The point about controlled composites bears further exploration. For example, pure amorphous carbon is quite weak, but if used in small quantities to cement iron filings, the composite would achieve significant strengths. Like cancellous bone, a porous composite made by pyrolyzing a binder between fully-dense whiskers of a ceramic will tend to be far more fracture-resistant than the same material if nonporous. A mixture of different kinds of particles can provide desirable combinations of properties unachievable in a homogeneous material, such as high surface hardness along with high crack resistance — again, like bone. Anisotropic filler orientation can provide anisotropic mechanical properties — again, like bone, or wood.

Ferromagnetic fillers like powdered iron can make a ferromagnetic bulk material with low electrical conductivity, but ceramic binders can maintain dimensions at different temperatures more precisely than the usual organic binders used for powdered-iron cores; also, iron's Curie temperature is 770°, which many ceramics can withstand easily, but organic binders can't even approach. (And cobalt's is 1115°!)

The cheapest possible combinations would be sugar or flour with quartz sand or glass fiber, but at least in my low-temperature, poorly-controlled experiments (up to perhaps 400°–600°) the carbon resulting from sugar pyrolysis adheres very poorly to glass, represented by the glazing of stoneware, and to quartz sand. I could scrub it off easily with steel wool, and even scratch off some with a fingernail. Surface preparation, for example with plasma (perhaps in a fluidized bed), could conceivably improve the situation. Carbon should stick well to carbon fiber, though, and many things stick well to glass. And, as I said above, to my sorrow carbon sticks beautifully to zirconia.

Magnesium oxychloride

Boron nitride

in ammonia?

Olivines

Forsterite, including peridot, is Mg_2SiO_4, while fayalite is Fe_2SiO_4; these are the endmembers of the olivine spectrum. Calcium cation substitutions also occur, modifying the structure and making it softer, going all the way to larnite, the belite of portland cement.

Mullite

Ordinary clay pottery

Self-propagating high-temperature synthesis

Other

Titanium carbide? Zirconium carbide (3530°)? Tantalum carbide (3850+°)? Zirconium diboride? Gallium nitrate (soluble, decomposes to GaN in flowing ammonia at 500°–1050°)?

Topics

Machine teeth

Kragen Javier Sitaker, 02020-08-02 (updated 02020-12-31)
(8 minutes)

Yield strengths and ultimate tensile strengths cover a similar-sized, but lower and narrower, range than Young's moduli. They generally correlate, although there is substantial deviation — the plastics have immense deformations at break, the metals smaller, the ceramics smaller still.

Material hardness and the tooth principle

Here's a table of very approximate quantitative figures:

	Young's modulus (GPa)	yield stress (MPa)	tensile rupture (MPa)
diamond	1000	brittle	60000
tungsten carbide	600	brittle	300
sapphire	400	brittle	400
carborundum	400	brittle	120-3000 (?)
cubic zirconia	200	?	700
A36 steel	200	250	500
zircon	200	brittle	300
tooth enamel	80	brittle	20
soda-lime glass	70	brittle	100
6061 aluminum	70	76-370	130-410
quartz	80	brittle	40
concrete	25-50	brittle	4
lead	14	creeps	15
wood (along grain)	9-14	brittle	5
poly(methyl methacrylate)	3	brittle	70
poly(ethylene terephthalate)	3	?	70
gypsum plaster	1.4	brittle	3
high-density polyethylene	0.8	?	20
styrofoam	0.005	brittle	0.4

Critters (the technical term) use teeth, claws, and beaks to cut things, maneuvering them into position with softer tissues. A smallish tooth can have an even smaller point that digs into the material to cut, supported by a wider base rooted in a "handle" of material softer than the tooth itself, which is held in still softer material.

Machines can use the same technique, and sometimes do: lathes, fly cutters, boring bars, shapers, and D-bits all cut with a single point, often made of a cermet; it's common to dress grinding wheels with a single-pointed diamond mounted at the end of a steel dressing tool; a woodworking plane commonly uses a steel blade held in a wooden frame; and a hand file commonly consists of case-hardened steel teeth on the surface of a piece of softer steel, held in a softer wooden handle, held in a still softer hand.

For example, you could imagine a tungsten-carbide tooth (sometimes these are called "teeth", other times "tools", "bits", or

"inserts"; analogous artifacts in archaeological contexts are called "microliths") shaving a 100-micron-thick, one-millimeter-wide shaving ("chip") as it scrapes along a steel surface. WC (the most unfortunate chemical formula ever) is several times stronger in compression, some 1500 MPa, but suppose we limit ourselves to its 300 MPa tensile strength; then the tungsten carbide will keep cutting as long as the force is less than 30 N. This is enough to get the steel to yield so that the carbide can propagate a crack under the chip. (Carbide's higher compressive strength clearly helps a lot here.)

The carbide can be held in a softer material such as steel or even aluminum; to keep those 30 N under the 76 MPa worst-case yield stress of the aluminum, we need at least 0.4 mm^2 of contact area between the carbide and the aluminum. So the carbide tooth itself could be tiny, with a 100-micron-long, 1-millimeter-wide point, supported on an 800-micron-tall pyramid with an 800-micron-diameter circular base. In fact, at such a low stress, even PMMA and PET would be strong enough not to rupture, although they would certainly creep; a more conservative approach would be to use a truncated aluminum cone with, say, an 800-micron circular tip, 3 mm height, and a 3-mm-diameter circular base, supported on wood, HDPE, PMMA, PET, or many other possible materials.

It probably isn't practical to cut most steels with something much smaller than that tooth, because the steel is too ductile; you'll end up just forming the steel instead of cutting it. The surprising thing is that 0.2 mm^3 of tungsten carbide, about π milligrams, is sufficient to enable cutting steel. 200 surface feet per minute (in the medieval units commonly used in machining in the USA) is probably achievable, which works out to 1.02 m/s in SI units, so this is a material removal rate of some 102 mm^3/s of steel, about 0.8 g/s, removing the mass of the carbide tooth itself roughly every 4 milliseconds.

Assuming a 15 minute tool life, this means that this tooth can remove about a quarter of a million times its own mass in steel during its lifetime.

A single gram of tungsten carbide contains enough material to make some 300 such teeth.

It is possible to substitute an intermediate-hardness material for the base of the tooth. Steel, a harder aluminum, or yttrium-stabilized zirconia would work. (You could try zircon, but I suspect it would be too fragile.)

As economic context, here in the kitchen I have a cheap zirconia kitchen knife that's about 100 mm × 25 mm × 1.5 mm, which is about 2800 mm^3 of zirconia, enough for some 14000 such teeth; such a knife costs some US$7, about 0.05¢ (US$0.0005) per tooth. I also have a high-speed steel hacksaw blade, which is about 310 mm × 12.7 mm × 600 µm (300 mm between the mounting-hole centers), about 2400 mm^3, which was even cheaper (about US$1.50 per blade), and is also suitable for cutting unhardened steel.

Alternative metal-cutting tooth materials

Traditionally, steel was cut merely with case-hardened steel, but this has its limitations. 19th-century advances in steel improved tool

life considerably, but today steel is usually cut with ceramics, especially the tungsten carbide mentioned above.

Three other hard ceramics mentioned above — sapphire, carborundum, and zirconia — may be more easily produced from terrestrial materials than tungsten carbide. Tungsten atoms are outnumbered by silicon in Earth's crust about a million to one. Sapphire is made from aluminum and oxygen; aluminum is very nearly as common as silicon, while oxygen is even more common. Carborundum is made from silicon and carbon; carbon is outnumbered by silicon only about 300 to 1, and of course diamond is entirely carbon. Zirconium is outnumbered by silicon about 3000 to 1, making it about 300 times as abundant as tungsten. Zirconia is quite brittle if not stabilized with, for example, 2–3% of yttria, but fortunately yttrium is only about one order of magnitude rarer than zirconium itself.

Zirconium has the additional merit of being relatively easy to concentrate, since it forms separate grains of zircon (jacinth, $ZrSiO_4$) in many igneous rocks, including most granites, which are easily separated from sand by their high density (4.6 g/cc); they are also separable by their refusal to melt at any reasonable temperature (below 2500°, though they sinter at much lower temperatures). Zircon itself, perhaps even naturally-occurring crystals, may be usable as a material for cutting metal; but zirconium is readily, if expensively, derived from it by calcining with carbon and chlorine, then reducing the resulting zirconium tetrachloride with magnesium, the same Kroll process used to reduce titanium; and zirconia is superior to zircon in almost every way.

Historically carborundum was discovered by Acheson running an electric arc through a mixture of clay and coke in an iron crucible, insulated by the granular materials themselves, in 1890; but Despretz probably made it without knowing in 1849 by joule-heating of a carbon rod embedded in silica sand, which is essentially the "Acheson process" used today; sawdust and salt are former additives now little used. (This is also the first process for making synthetic graphite, by heating the carborundum until the silicon sublimes at 4150°. XXX wouldn't graphite sublime too? Shouldn't that be 2830°?)

Sapphire is normally refined as an intermediate step in the production of aluminum, for example by the Bayer process: low-silica bauxite is digested in 170°–180° lye (or anhydrous 1200° sodium carbonate and coke, in the Deville process) to obtain sodium aluminate, from which crystalline aluminum hydroxide is precipitated (by cooling, by neutralization with CO_2, or simply by evaporation with seed crystals).

Zirconium carbide can be made simply by carbothermic reduction of zirconia with graphite; it is even harder than zirconia itself (?), though it has "poor oxidation resistance over 800°".

Topics

3-D printing iron by electrodeposition?

Kragen Javier Sitaker, 02020-08-15 (11 minutes)

Speculation

You can form an arbitrary iron shape by simultaneous electroforming from a large number of parallel needles? In a near-boiling, oxygen-starved solution of hydrochloric acid, you can run current through some needles, but not others, to promote the deposition of iron on a cathode surface a short distance away (millimeters or less) around just those needles. By withdrawing the array of needles as the cathode grows, the inter-electrode distance remains constant. If the needles are themselves iron, they will dissolve anodically and be deposited (in electrolytically purified form) on the cathode, and will need to be fed in through some kind of wire feed mechanism, but if they are graphite or a noble metal, then iron must instead be supplied in ferrous form by pumping in more electrolyte to replace the spent electrolyte; pumping it through the centers of the needles themselves is one possibility.

If the needles are moved laterally as well as being withdrawn, they can produce features of finer detail than the spacing between needles.

And, of course, if the current is reversed, the same method produces local anodic dissolution and becomes selective electrochemical machining, rather than selective electrodeposition.

By dispersing fine graphite or amorphous carbon particles in the electrolyte, so that some of it gets included in the iron deposits, it is possible to deposit iron that can later be converted to steel by heat treatment, causing the carbon to diffuse; this can be localized to just certain layers of the workpiece. Alternatively, the iron can be case-hardened if the softness and ductility of pure iron is undesirable.

Other metals commonly electroplated can be 3-D printed in the same way; an Argentine savant has already demonstrated this process with copper, but zinc, tin, chromium, nickel, gold, silver, cadmium, cobalt, lead, and even some alloys such as bronze and brass can be shaped in this way. Additionally, layers of different metals can be alternated, and fillers such as graphite, aluminum oxide, and clay particles can be included, especially if the cathode voltage is kept moderate enough to prevent bubbling.

Other liquid electrolytes, such as ionic liquids and perhaps even deep eutectic systems, might permit the use of a wider range of metals, more rapid electrodeposition of iron, or lower risks than a near-boiling strong hydrochloric acid solution.

The needles, if iron, need not be pure iron; they can contain metallic impurities as long as their standard electrode potential is significantly more negative than iron's -0.44V. In particular, zinc, magnesium, aluminum, and the rare earths should not be a problem. Most other metals, however, would plate out in preference to the iron, including virtually everything you can electroplate in water

(except zinc and maaybe chromium).

Historical background

In https://www.finishing.com/94/56.shtml there is some discussion of different ways to electroform or electroplate with iron; Colin Braathen writes:

I agree that ferric chloride is not a good basis for plating; most documents on the subject stress the importance of keeping iron 3+ levels in the bath low. Air for agitation is likely to oxidise the ferrous ion to ferric, so I plan to hermetically cover the bath with clingwrap and agitate using argon… Bath heating will be by quartz tube with a Nichrome coil inside, salvaged from a cheap radiant room heater, glanded into the bath walls with silicone. Bath lining will be PVC, heat gun welded at the seams and bonded to a support shell with air-moisture-curing polyurethane glue (bath temperature will be uncomfortably close to the glass transition temperature of PVC).

I, too, have plated iron from a sulphate solution as a tryout. I got about 1 mm before my power supply burned out (15A on about 0.5 sq.ft, i.e. 30 A/ft). The iron was brittle to the point of being crumbly I could almost crush it in my fingers, and I had incipient dendrites. … Chloride baths are run at higher temperature and, apparently, can produce a ductile stress-free deposit if done right.

… The bath will be Fisher-Langbein, FeCL2.4H2O 300-450 g/l, CaCl2 150-190 g/l, 85 °C, pH 1.5, current density 2 9 A/dm2 (20 85 A/ft2). The low pH should help to minimize formation of the ferric salt.

See also:

https://ukdiss.com/examples/electrodeposition-iron-co-deposits-des.php is an academic fraud company ("dissertation writing service") publishing a purported dissertation on iron electrodeposition in ionic electrolytes whose author's name has been removed.

https://encyclopedia2.thefreedictionary.com/Iron+Plating:

an electrolyte whose main constituent is ferrous sulfate or chloride. … electrodeposition proceeds at room temperature with an insignificant concentration of acid in the electrolyte at a rate of the order of 1 micron per hr. For repair work, the temperature and acid concentration are increased. The iron layer is deposited more quickly, the ferrous chloride solution is more concentrated, and the temperature is about 100°C.

https://www.scientificamerican.com/article/electro-plating-with-iron/ https://archive.org/details/scientific-american-1869-11-27

Electro-Plating with Iron

Scientific American volume 21, number 22, p. 346

November 27, 1869

The Hon. Cassius M. Clay†, late U. S. Minister to Russia, has recently returned from St. Petersburg, bringing with him some fine specimens of iron electrotypes, done after the process of Prof. Jacobi and Klein. We have before alluded to this important discovery. By its use, nearly all forms of electro-plating, such as engravings, stereotypes, medallions and ornaments, may be done in iron, with a fineness of texture which is really surprising.

Its importance and value will be appreciated when we reflect that the iron electro-plates are about five times more durable than the ordinary copper electro-plates.

Mr. Clay has presented us with an iron electro-plate copy 'Of a copperplate engraving of the Prince Imperial of Russia. This plate is six inches square,. and beautifully done. It is one thirty-second of an inch in thickness, and has a color closely resembling that of zinc. These iron electrotypes are now used by the Russian Government with complete success for the printing of bank notes.

The process was patented in this country through the Scientific American Patent

Agency, Sept. 29, 1868, and further information can be had by addressing C. M. Clay & Co., 45 Liberty St., New York.

The following description of the process we copy from the patent specification :

"Our invention consists in the application of a practical galvano-plastic process as to the deposits of iron on molds, or any other form, for reproducing engravings, stereotypes, and for other useful or ornamental purposes.

"The galvano-plastic bath we use is composed of sulphate of iron, combined with the sulphates of either ammonia, potash, or soda, which form, with sulphate of iron, analagous [sic] double salts.

"The sulphate of iron may also be used, in combination with the chlorides of the said alkalies, but we still prefer the use of sulphates.

"The bath should be kept as neutral as possible, though a small quantity of a weak organic acid may be added, in order to prevent the precipitation of salts of peroxide of iron.

"A small quantity of gelatin will improve the texture of the iron deposit.

"As in all galvano-plastic processes, the elevation of the temperature of the bath contributes to the uniformity of the deposit of iron, and accelerates its formation.

"For keeping up the concentration of the bath, we use, as anodes, large iron plates, or bundles of wire of the same metal.

"Having observed that the spontaneous dissolution of the iron anode is, in some cases, insufficient to restore to the bath all the iron deposited on the cathode, we found it useful to combine the iron anode with a plate of gas-coal, copper, platinum, or any other metal being electro-negative toward iron, and which we place in the bath itself.

"As a matter of course, this negative plate may also be placed in a separate porous cell, filled with an exciting fluid, as diluted nitric or sulphuric acid, or the nitrates or sulphates of potash and soda.

"For producing the current, we usually take no more than one or two cells of Daniels' or Smee's battery, the size of which is proportioned to the surface of the cathode.

"It is indispensable that the current should be regulated, and kept always uniform, with the assistance of a galvanometer, having but few coils, and therefore offering only a small resistance.

"The intensity of the current ought to be such as to admit only of a feeble evolution of gas-bubbles at the cathode, but it would become prejudicial to the beauty of the deposit if gas-bubbles were allowed to adhere to its surface.

"The same molds, as employed for depositing copper, may also be used for depositing iron, only it is advisable, in employing molds made of lead or gutta-percha, to cover them previously with quite a thin film of galvanic copper, formed, in a few minutes, in the usual way, and then oring [sic] them, after having washed the molds with water, immediately in the iron-bath.

"The film of copper may be removed from the deposit either by mechanical means, or by immersion into strong nitric acid.

"The deposited iron is very hard, and rather brittle, so that some precaution must be taken in separating it from the mold. By annealing, it acquires the malleability and softness of tempered steel.

† This is a different Cassius Clay than Muhammad Ali.

https://www.pfonline.com/articles/iron-plating(2)

The iron plating bath is particularly useful for when large build-ups (50–100 thousand[th]s of an inch) are required. There are a number of different baths available: ferrous chloride, ferrous fluorborate, ferrous sulfamate and ferrous sulfate are common examples. Of these baths, the most common is the ferrous chloride bath…

Then Kushner gives the recipe: 40–60 oz./gal. of ferrous chloride dihydrate, 20–35 oz./gal. of calcium chloride, 185–210°F, 20–80 amps per square foot without agitation or up to 200 with agitation (which must not be air to prevent oxidizing the ferrous ions), high-quality iron anodes, pH 0.2–1.8 using HCl. There is some confusion in the recipe.

https://patents.google.com/patent/US2745800A/en

By John Poor, 1953.

Topics

Peroxide and bleach

Kragen Javier Sitaker, 02020-08-15 (2 minutes)

To my surprise, last night I learned from a chemist friend that mixing hydrogen peroxide with (sodium hypochlorite) bleach liberates oxygen, presumably from the decomposition of both substances, leaving water and sodium chloride as well as the oxygen gas. How much oxygen should it liberate?

In addition to weight percentages (3% being the usual article of commerce in pharmacies) H_2O_2 is commonly sold by "volumes": "20 volumes" yields 20 mℓ of O_2 gas from the decomposition of the H_2O_2 in 1 mℓ of the solution. O_2's molecules weigh 32 daltons, so a mole of it weighs 32 g; two moles of H_2O_2 are needed to produce one mole of O_2, and they will weigh 34 g each, 68 g in total. The combined gas law is that $PV = nRT$, where [$R \approx 8.3144598$ kPa ℓ / mol / K] is the universal gas constant, so at 20° = 293.15 K a mole of an ideal gas at 101.325 kPa would occupy $8.3144598 \times 293.15 / 101.325 \approx 24.055$ ℓ. The density of pure H_2O_2 is 1.450 g/cc, thus 21.32 mmol of O_2 per cc, which gives 512.85 cc of O_2 gas per cc of H_2O_2, so pure hydrogen peroxide would be "512.85 volumes". So "20 volumes" H_2O_2 is only about 3.900% by weight. (But Dr. Google says it's actually 6%, so my calculations must be off.)

The common bleaches sold here are 57 g Cl/ℓ and 25 g Cl/ℓ. Sodium hypochlorite is NaClO, with one atom of oxygen per atom of chlorine. ...

Topics

Cyclic fabrication systems

Kragen Javier Sitaker, 02020-08-17 (updated 02020-09-10)
(56 minutes)

"Cyclic fabrication systems" is a term Matthew Moses, Hiroshi Yamaguchi, and Gregory Chirikjian invented to describe a collection of materials, tools, and processes capable of reproducing itself. A CFS is the keystone of economic autarky, of resilience against faraway catastrophes, and of escaping resource scarcity traps, and its exponential growth rate has historically been the major limit on worldwide economic growth; consequently the study of CFSs is, or should be, central to economics, although it is a relatively neglected topic, involving as it does cross-disciplinary concerns from engineering, materials science, chemistry, metrology, and economics.

A programmable, fully automated autotrophic CFS with a growth rate exceeding that of the economy would eliminate the scarcity of capital goods that has been the foundation of human economics for two million years.

With that in mind, I thought it would be worthwhile to survey historically and potentially existing CFSs. There are various aspects of cyclicity: there's the simple geometric question of how a machine can reproduce the shapes of its own parts in the same materials; there's the metrology question of how to measure half a millimeter on a ruler measured in millimeters; there's the control question of how a negative-feedback system can produce another working active negative-feedback system, there's the chemistry question of how to produce a large amount of materials with the necessary degree of purity, starting from impure and unknown natural materials and a small amount of known and pure materials; and there's the energy question of how such an assemblage of machines can continue harvesting energy from its environment, for example building an engine that can harness the available solar or chemical energy.

Geometry

The first category of CFSs to survey are those concerned with imposing some existing geometry on some existing material. For now we're not so concerned with how the geometry or the material is created.

A feature common to many geometry CFSs is a sort of rock-paper-scissors dynamic; in a given manufacturing process, typically one material is stabler than another, so perhaps it can be used to shape that other material.

For example, you can melt wax into a pewter mold, you can melt pewter into a steel mold, and you can melt steel into a greensand mold; but you can't melt greensand into any kind of mold, both because it requires unreasonably high temperatures and because it ceases to be greensand when you melt it. Instead we rely on the fact that greensand at room temperature is soft enough to be rammed around patterns, made of materials such as wax, with tools made of materials such as steel.

Pewter beats wax, steel beats pewter, greensand beats steel, and wax beats greensand. Thus we form the cycle that makes our fabrication system cyclic.

Flintknapping

Since the Paleolithic, the humans have shaped tools by banging rocks together, a process called "knapping". Arguably this is not a CFS, because typically the hammerstones and other flintknapping tools such as antlers and copper pressure-flaking tools are not themselves shaped by flintknapping or flintknapped tools.

Knapping is somewhat limited in the geometries it can achieve, and it can only shape materials that break in the right way, such as glass, obsidian, and flint.

Grinding

A major innovation in manufacturing technology some 35 millennia ago, perhaps in Japan, was shaping stones by grinding them with abrasives, rather than chipping. It had spread to the Levant and Europe by some 10–18 millennia ago, where it is considered a distinguishing mark of the Neolithic. Grinding permits greater liberty of materials, surface finishes, and geometry; abrasives can shape any solid material and can achieve arbitrary geometry down to submicron scales. In particular, in the late Japanese Paleolithic and in the Mesolithic and Neolithic, polished stone axes were much more durable than traditional knapped axes; hole-drilling permitted much more adaptable and secure forms of assembly; and Kebaran mortars and pestles began to automate the more mechanical aspects of food digestion.

From a CFS perspective, there are several great features of grinding. One is that the geometry of the workpiece can be more precise than the geometry of the tool, because it is determined primarily by the *movement* of the tool, as constrained by the workpiece and other external systems, rather than the tool's geometry. Another is that, by grinding three surfaces against each other with abrasive between, a precisely flat surface can be achieved without any flatness reference. A third is that some hard materials are relatively easy to break, so we can get a rock-paper-scissors cycle with only two materials: a hard, brittle abrasive such as sapphire or quartz and a softer, tougher hammering material such as copper or iron. A fourth is that, because of the aforementioned movement feature, it's actually possible to get a CFS with just a single material such as sandstone or concrete; you can dress a grinding wheel with an abrasive dressing stick just by moving the stick back and forth across the wheel's face while spinning it. A fifth is that grinding generally does not require heat or a minimum tool pressure, so it can be done while the workpiece is not deformed, providing more precise geometry than other ways of shaping materials.

Consequently various kinds of grinding, including lapping, play central roles in all kinds of precision manufacturing even today. Especially on glass materials, deep submicron precision is feasible. Grinding is also commonly the only cutting process used for applications like cutting rebar or concrete on construction sites and smoothing over weld beads, where its material flexibility and low

equipment cost outweigh its low material removal rate. In modern machine shops, grinding is used for cutting nearly-finished parts to final dimensions and for shaping metal-cutting tools out of materials that are too hard to drill or cut on the mill or lathe.

Pottery

Since the late Paleolithic or early Neolithic, the humans have made pottery by sintering ("firing") composites of clays, sintering aids, fillers, and sand into a sort of ceramic. Sintering in general has the advantage that the sintered material can remain solid at temperatures exceeding those necessary to sinter it in the first place (for pottery, in the 700°–1500° range depending on composition), so if the temperature remains relatively predictable, a kiln for sintering such pottery can itself be made of the same pottery. Moreover, it is possible (and, in historical practice, usual) to sinter the kiln itself in place, rather than sintering firebricks in a separate kiln and then assembling them into a kiln.

Clays go through a series of states of plasticity according to their hydration. Above about 25% water, they are colloidal liquids called "slips"; around 22%, purely plastic solids or thixotropic liquids, which shrink substantially as they dry further; around 20%, they remain plastic over a large range of deformations but become capable of brittle fracture ("leather-hard") and almost do not shrink upon further drying; and below about 18%, they are fragile, brittle solids, which do not shrink on further drying. (The precise transition points vary by clay composition, soaking time, and aqueous solute content.) Firing results in further shrinkage. The other components of the so-called "clay body" can enhance plasticity and reduce sintering temperatures, and they tend to decrease shrinkage. The sintered clay body is much stronger and can be nonporous, especially if sintered a second time coated with a so-called "glaze" which melts fully rather than just sintering; typically this results in further shrinkage.

Because fired or even dried clay can plastically deform leather-hard or fully plastic clay, it is straightforward to use so-called clay "seals" to reproduce geometry (in negative), and such "bullae" have been a key security measure for commercial transactions for some 12 millennia, since the beginning of agriculture if not before; recently metals are more commonly used instead of clay, but this innovation was unknown until just a couple of millennia ago. This process of molding can be carried through multiple generations of clay seals, though not without significant loss of fidelity, including, in particular, shrinkage.

Such a clay-on-clay "sealing" process is probably responsible for the oldest instance of movable-type printing, the Phaistos disk; and of course movable type in Asia was made from pottery long before Gutenberg.

In the leather-hard state, the clay body is still plastic enough to take the impressions of seals, but also brittle enough to be cut with blades. Typically these are made of metal in current practice, but blades made of fired clay also work. Because most of the shrinkage has already happened, forming clay in the leather-hard state results in much more precise dimensions.

Clay is commonly "slipcast" in porous molds made of plaster of Paris: a slip is poured into a mold, and the absorption of water into the mold solidifies a layer of the slip in contact with the mold. The remaining slip is poured out, and the cast piece contracts as it dries and can then be demolded. Presumably it is possible to make the molds of porous fired clay instead of plaster, though I have never heard of it being done and have not attempted it myself.

Dried clay is also friable enough to be easily abraded or carved by tools, including tools made of fired clay.

The precision of all of these shaping processes is limited by the shrinkage and deformation of the ceramic during sintering, if sintering is done; by the grain size of non-clay tempers, such as sand, in the clay body; and by the shrinkage and deformation of the clay body to its dry state, for shaping processes that rely on plasticity.

Fired-clay ceramics play a key role in many more elaborate CFSs as well, because (depending on composition) they can withstand relatively high temperatures without losing their shape, they can easily be shaped to complex geometries while plastic (especially if dimensional precision is not critical), they are themselves relatively hard and can embed even harder abrasives, and they are very inexpensive.

Like most ceramics, clay has extremely small elastic deformation in all states, including the fired state, on the order of 0.01% strain at failure. This permits relatively high geometric precision, especially when shaping dried clay, but it complicates the use of fired clay for springs and other flexures.

Worth mentioning is the standard procedure for foaming pottery, which is to mix a granular organic material such as sawdust or used yerba mate into the clay body. In an oxidizing kiln the organic material burns out completely, leaving voids in the clay which reduce its weight, impede crack propagation, and improve its thermal insulating capabilities, and at higher void fractions permit easy fluid flow through the fired piece. Because of the crack-propagation improvement, the foamed pottery can be cut to shape almost as if it were wood. I've tried void fractions from 50% yerba up to 85% yerba, which last was quite fragile; above 66% yerba, the material permitted easy airflow. This porosity can be beneficial to the firing process in allowing the fabrication of thicker shapes without steam explosions.

So, the cycles here are: fired clay shapes plastic clay (by sealing, slipcasting, or cutting), plastic clay dries into dried clay, and dried clay is fired into fired clay; and fired clay carves dried clay, and dried clay is fired into fired clay.

Hot forging

Cold iron is harder than hot iron or even hot steel, so if you press them together the cold iron will reshape the hot metal without itself being reshaped. This is common to a number of metal-shaping processes including hot rolling, hot forging, and wire drawing; commonly hammering is used to achieve sufficiently high pressures to deform the hot metal. (It is also the reason the World Trade Center collapsed, despite the temperatures not being hot enough to melt its steel beams; the heat softened them.)

However, the metal thus formed deforms during cooling, so these processes generally cannot achieve tight dimensional tolerances.

So the cycle here is that cold iron forges hot iron, and hot iron cools into cold iron.

Hardened ferrous tools, cutting and hammering ferrous metal

Files and other hardened iron and steel tools can be used to cut unhardened iron and steel, which can then be hardened. Similarly, hardened iron and steel hammers can be used to cold-forge unhardened iron and steel.

There are several ways to harden ferrous metals, but nearly all of them involve a great deal of heat, and so impose a certain amount of uncontrolled deformation. "Case hardening" by diffusing carbon (and possibly nitrogen) into the surface of iron is a form of solid-solution hardening known for two or three millennia; "quenching", applicable to carbon steels, is another. In quenching, the metal is heated until it converts from ferrite to austenite, then cooled too rapidly for it to reform ferrite at the surface, leaving it in the metastable state of martensite, which is much harder than ferrite. (Sometimes the term "case hardening" is also used for quenching only the surface of a piece of metal.)

Another is "work hardening" by hammering the surface ("cold working" or "cold forging"), but typically the resulting hardening is relatively small. Cutting highly work-hardenable metals like copper is difficult, because work hardening hardens the surface as soon as you have cut it, and perhaps even before; consequently, the carburization and quenching processes described above are the ones used for cyclic fabrication systems. Work hardening plays many critical roles in metallurgy, but historically not in CFSs.

(Other forms of hardening, such as precipitation hardening, are not applicable to iron and ordinary steels, though they are applicable to some other alloys, like 17-4 stainless and beryllium copper.)

So the cycle here is that hardened steel cuts unhardened steel, and then quenching or carburizing unhardened steel makes hardened steel.

Sandcasting and lost-wax casting

As mentioned above, sandcasting is a common CFS, in the sense that once you have a pattern for a castable shape, you can ram soft greensand around the pattern in a flask to make a mold, disassemble the flask, remove the pattern, reassemble the flask, and pour molten metal into it to reproduce the shape of the pattern — first perhaps arsenical bronze, but also copper, and more recently tin bronze, cast iron, tin, pewter, pot metal, Zamak, pig iron, high-silicon aluminum alloys, brass, and so on.

Greensand is sand (typically quartz; minerals that outgas at high temperatures such as calcite or gypsum are forbidden) containing a small amount of wet clay as a binder. There is little enough clay so that the mold is very porous, thus permitting easy passage of gas through the sand, among other things to allow the mold to dry before casting. Typically the clay used is bentonite, since it can function as a

binder at the lowest levels, and those levels are low enough that its expansivity is not a problem.

Aside from the raw materials, sandcasting requires minimally a pattern, a riddle, a ram, a flask, a crucible, and a kiln (called a "furnace"). The crucible and the furnace are necessarily formed of refractory materials, and in usual practice must be exposed to air, so pottery is the traditional material for them, making this CFS dependent on the pottery CFS when used for ferrous metals; the non-ferrous metals mentioned above can instead use ferrous crucibles, and in theory could use ferrous furnaces as well, but pottery is much more practical. The pattern and flask are traditionally made of wood, but metal would also work fine. The ram can be made of wood or metal. The riddle probably cannot be practically cast of metal, because it needs to have many small holes, and casting is bad at those. The standard nowadays is to use a riddle of woven wires on a wooden frame, but I think you could make one of fired clay (though my first attempt to do so was not very successful) or woven flax sized with some kind of abrasion-resistant coating.

In sandcasting, although the mold is destroyed when the casting is shaken out of it, the pattern can be reused many times. Nowadays the patterns are often 3-D printed, but another bootstrapping option might be to carve the pattern from foamed pottery.

I think older than sandcasting is lost-wax casting, which is similar, but with a pattern of wax (traditionally a mix of beeswax and pine resin, perhaps with some dry powdered clay as a filler) and a mold of pottery. Rather than removing the pattern from the mold before firing the mold, the wax is simply left in the pattern, where it melts and then burns out, just like the organic fillers used for foaming pottery. This does not permit the reuse of the pattern, but can reproduce finer details than sandcasting, though with worse dimensional precision.

A modern innovation in these processes is "lost foam casting", where the pattern is burned out as with lost-wax casting, but the mold is sand as with sand casting. By making the pattern from an organic foam such as styrofoam, it produces little enough gas that it need not be removed from the mold ahead of time, but is instead burned out by the hot metal; this eliminates the need for a binder and the need to disassemble the flask, and styrofoam is easier to cut than the traditional wood or, especially, metal.

Die-sink/ram and small-hole EDM ("electric discharge machining")

EDM is capable of holding very tight tolerances (on the order of a micron, or somewhat higher at larger material removal rates) and cutting very hard materials, even diamond. You bring an electrically conductive workpiece together with a tool ("die") in an insulating liquid, such as diesel fuel or deionized water, holding a voltage across them, until a spark occurs, vaporizing some of both the workpiece and the die. The vapor immediately condenses in the liquid, and some time later you move them toward each other until a new spark occurs somewhere else. This repeats, thus eroding both the workpiece and the die. They never quite make contact, and they never heat up

except in a very thin surface layer, so the deformations that limit the precision of mechanical cutting processes are absent.

The die can be of the same material as the workpiece. However, to minimize the necessity to manufacture new dies, it's desirable to use die materials that are eroded less than the workpiece because of having higher boiling points, higher enthalpy of fusion and/or vaporization, or higher thermal conductivity; graphite, copper, brass, and tungsten-copper are ideal, but even things like aluminum last longer than steel (I think?), which in turn lasts longer than stainless steel, which in turn lasts longer than tungsten carbide and similar materials.

In particular, copper and graphite electrodes can cut steel with almost no wear, removing more than 100× as much workpiece as electrode.

"Small hole EDM" is a variant of this approach, usually categorized separately, in which the tool electrode is a thin tube used to "drill" a hole into or through the workpiece. Dielectric fluid is fed through the tool at high speed. The tool is rotated as it is fed into the hole, eliminating circumferential variation, and also permitting the hole to be significantly wider than the tool itself if the center of rotation is eccentric. EDM drilling can thus produce very accurately cylindrical holes, even in very hard materials. (And this is a crucial supporting process for wire EDM, as explained later.)

Small-hole EDM drills are often insulated up to the tip so that the hole only widens near the tip, enabling it to remain the same diameter over a long distance.

Hypothetically, if the die were of a shape that can be produced by helical extrusion, then it could be fed into the cut in the helical path along which it was extruded. The end of the die might be consumed in the process, but, especially for through holes, this is unimportant.

An EDM die can also be moved to cut out a shape as if it were an end mill, known as "orbiting"; if this is done with a die with a thread profile ("EDM tapping"), it cuts a thread into the hole it's in with the same thread pitch as the die's thread, but a larger diameter. Small pits in the threads of the die are unimportant, as they will be covered by the orbital motion.

Hypothetically, you could also move the die in two directions of motion at once, as if for lapping; although I do not know of this process being used in practice, the same process could produce three accurately flat surfaces by translating and rotating them relative to each other while using them to erode one another via EDM.

EDM is also done at times with dies that are wheels analogous to grinding wheels; as with grinding wheels, small irregularities in the wheel surface are inconsequential, so they can make surfaces that are more accurately cylindrical than they are themselves. Presumably these EDM wheels must periodically be dressed like grinding wheels, but using EDM rather than grinding. Vollmer calls this process "disc erosion", and their process uses copper-tungsten wheels and an oil dielectric to sharpen cutting tools made of polycrystalline diamond or tungsten carbide, while Setco calls it "spark erosion grinding" and uses cold-rolled steel wheels and I think a water dielectric to cut delicate

metal honeycomb parts for jet engines.

A CFS based on die-sink EDM might be able to use a single conductive material as both a workpiece and a tool, reorienting the tool during the cutting process to produce on it the kinds of radii and edges that will be needed for later features to be cut into the workpiece.

However, a much more efficient die-sink-EDM-based CFS would use at least two materials, one harder and easier to cut with EDM, such as steel or tungsten carbide, and the other softer and more resistant to spark erosion, such as graphite, brass, or copper. A single steel cutting tool can cut hundreds of brass electrodes to precise shapes, and a single brass electrode can erode hundreds of steel workpieces to precise shapes.

However, an EDM machine cannot be made entirely from metal parts, because the only electrical path between the cutting tool and the workpiece must be the cutting arc. Thus some insulating material, such as a plastic or ceramic, is needed to complete the cycle.

(These accounts of EDM-based CFSs take the electronic servo control systems necessary to control the EDM motion as given, a lacuna I will remedy later on.)

In many cases, however, wire EDM supplemented with small-hole EDM is orders of magnitude faster than die-sink EDM. Normally die-sinking EDM is only used to finish parts to final dimensions after cutting them to approximate shape using faster processes.

Wire EDM

Wire EDM removes material from a conductive workpiece through the same spark-erosion process as other kinds of EDM, but the tool electrode is a thin brass wire, tens to hundreds of microns in thickness. This wire passes through a thin kerf in the workpiece, cutting it to an arbitrary two-dimensional shape with an arbitrary taper, while running through the workpiece at high speed. Given a starting hole made by some other process, such as drilling or small-hole EDM, wire EDM can enlarge the hole to an arbitrary shape. By cutting a stack of sheets, typically welded together at the edge, wire EDM can cut the same shape into many sheets at once.

The eroded brass wire must be melted and redrawn before being used again, since it has spark-erosion pits at unknown places along its length which could cause it to break if used a second time.

Die-sink EDM can cut arbitrary three-dimensional shapes, while wire EDM is more restricted in the geometry it can produce. But die-sink EDM must vaporize and wash away all the negative space of the desired part, while wire EDM need only vaporize a kerf of tens or hundreds of microns around its surface, thus potentially permitting a speedup of a thousand or so.

Wire EDM and small-hole EDM can cut through a meter or more of material in a single operation, so it's straightforward to imagine a sheet-cutting operation producing 20,000 identical parts from 50-µm-thick sheet stock in a single operation. (Other sheet-cutting processes, like waterjet, plasma, oxy-gas, bandsaw, and laser, are not so flexible; they tend to blow the layers apart.) But it's not clear that this would be an especially fast or cheap way to do it.

A CFS based on wire EDM could surely cut most of the parts of the EDM machine itself from steel or brass stock, then assemble them using an assembly system made similarly. The wire itself, if not treated as a "vitamin" provided from outside the system, could be drawn from brass stock using EDM-cut drawing dies. (These might even be workable without small-hole EDM.) Again, I will postpone the question of the necessary control electronics, and again an insulator is required.

Grinding, grinder dressing, and machining

As mentioned earlier, grinding plays an important role in modern machining, as does hardening of steels for cutting tools, but its overall cycle is more complex. Metal is mostly cut with a lathe, drill press, milling machine, or hand file, using ceramic/cermet ("hardmetal") or hardened steel cutting tools ("machining"). These tools are typically shaped and resharpened with a grinding wheel, which can be silicon carbide, cubic boron nitride ("borazon" or "qingsongite"), or diamond, or (for hardened steel tools only) aluminum oxide, garnet, or zirconia†. Aluminum oxide, zirconia, and silicon carbide are the usual materials. This grinding wheel wears and loads, and must be brought back to shape ("dressed") periodically; Adam Martin of Helical Solutions explains that a diamond grinding wheel requires dressing every 500 to 600 tungsten-carbide tools.‡ Dressing a diamond or silicon-carbide wheel can be done with another silicon-carbide wheel, as Helical does; aluminum-oxide wheels can be dressed with a diamond tool or with a star wheel.

A star wheel is a steel wheel with many points that is free to rotate; bringing it in contact with a spinning grinding wheel sets it to rotating, and its points whack into the surface of the grinding wheel, chipping it and knocking off grains. Moving the star wheel back and forth as it spins evens out the local variations in its shape, making the surface of the grinding wheel accurately cylindrical or conical, depending on whether the movement is parallel to the grinding-wheel axis.

Star wheels can be made by grinding their parts from steel stock and assembling them, or more rapidly by cutting the steel stock with cermet or hardened steel tools.

Grinding wheels are made by casting a mix of abrasive and binder in a mold, and often then firing the piece to harden the binder. The mold can be cut from metal; common binders include clay, magnesium oxychloride, and organic polymer resins including rubbers, and have historically included sodium silicate and shellac; but a wide variety of cements work at low speeds. Historically, grinding wheels were often simply cut from sandstone, whose quartz grains are hard enough to cut steel but not tungsten carbide; they are typically bonded together by calcite deposited hydrothermally. Wheels using the "superabrasives", diamond and cubic boron nitride, commonly use metals as binders.

As mentioned above, bonded-abrasive sticks can also be used to dress bonded-abrasive grinding wheels. They can be made in the same way as the wheels, but are normally more porous. Traditionally this porosity is achieved in a similar way to foamed fired-clay pottery, with a filler that burns out during firing, but, for low-firing binders

like rubbers and shellac, a lower-boiling-point filler such as naphthalene is needed.

Grinding done fast can easily produce temperatures high enough to dissolve diamond into transition metals like iron, so diamond abrasives are usually not used on metals. Cubic boron nitride is nearly as hard and does not suffer from this problem. Also, diamond burns in air at 650°, while boron nitride does not burn at all — it forms an impermeable boria layer (though this melts at 490° and starts vaporizing at 1100°, well below its boiling point of 1860°), then begins to react with transition metals around 1400°.

Silicon carbide abrasives don't last as long as alumina, accounting for their lower popularity despite their higher hardness.

Sometimes the ratio between the workpiece wear and the grinding-wheel wear is called the "G ratio"; the G ratio depends on the abrasive material, the bond, machining speed, feed rate, and workpiece material. Typical G ratios are 2–200, but can even be smaller than unity. This is orders of magnitude smaller than the ratio between the wear on a machining tool and the chips removed from the workpiece, which is in the neighborhood of tens of thousands up to millions (p. 251), so in the machine shop grinding is only used as a finishing operation, similar to die-sink EDM. This permits a machining CFS to achieve much higher offspring numbers than a simpler grinding CFS.

So, among the geometry-production CFSs in modern machine shops, we find: hardened steel cuts steel, which is then hardened, then ground with an aluminum-oxide grinding wheel, which in turn is dressed with a steel star wheel, which was also cut with hardened steel; tungsten carbide is ground with a diamond wheel, which is dressed with a silicon-carbide dressing stick, and both the wheel and the stick were cast in steel molds cut with tungsten carbide; silicon-carbide grinding wheels are dressed by grinding them with other silicon-carbide wheels; and many variations.

Tungsten carbide cutters are themselves mostly shaped by other processes and may not be ground at all; in particular, they are mostly made by hot isostatic pressing ("HIP") of tungsten-carbide powder, cemented with cobalt. This is done mostly with steel equipment made by the processes described above.

† I'm not sure whether zirconia can be used to cut tungsten carbide, why nobody makes grinding wheels out of tungsten carbide, or why zirconia is usually used together with aluminum oxide instead of alone.

‡ However, in the same video, Martin also claims that tungsten carbide is made by sintering tungsten with cobalt, so he may not be an entirely reliable narrator.

Electrochemistry, including ECM ("electrochemical machining") and electrodeposition

This involves several different applications of the same process, one which is somewhat less familiar from daily life than grinding, cutting, and spark erosion. It involves a current between two electrodes in an electrolyte; typically the electrolyte is aqueous, although ionic liquids are possible, including deep eutectic systems.

A paradigmatic case is nickel plating of steel, in which a nickel anode and steel cathode are immersed in a solution of, for example, sodium chloride. The power supply sucks electrons out of the nickel anode, ionizing nickel atoms at the surface of the electrode, which float freely in the electrolyte as positive Ni^{2+} ions, attracted to the negatively charged cathode, where they are reunited with electrons and form metallic nickel again. Thus the anode is gradually dissolved while metal is deposited on the cathode.

In this case the sodium does not deposit on the cathode because it is much easier to ionize — its ionization energy is 495.8 kJ/mol, its reduction potential is -2.71 volts, and its electronegativity is 0.93 Pauling units — not only than the nickel, but even than the water itself. Nickel's ionization energy is 737.1 kJ/mol (and its second ionization energy is 1753.0 kJ/mol), its reduction potential to the hydroxide is -0.72 volts, its reduction potential to the nickel(II) ion is -0.25 volts, and its electronegativity is 1.91 Pauling units. Water's reduction potential to electrolyze hydrogen is -0.8277 volts. So nickel precipitates at a lower voltage than is required to produce hydrogen, and hydrogen is produced at a lower voltage than is required to produce sodium, although mercury electrodes can change the situation by amalgamating the produced sodium.

(I'm not sure about this, for a couple of reasons. Nickel cations go into the solution, turning it light green, but the bulk solution does not become positively charged like a positive electret, so either it must be losing other cations like the sodium, or it must be gaining additional anions to compensate, which would have to be hydroxyl anions formed by producing hydrogen gas. But nickel chloride is highly acidic, not basic.)

Because the nickel's crystal structure is relatively compatible with the steel's, it can form a strongly adherent film on the surface.

This process, and analogous processes using other metals, is used in seven main ways, three of which are more or less geometry production:

• **Electroplating** of a thin film of metal on the surface of some substrate, which might even be a film of graphite paint. This can be used for appearance's sake (as in the case of gold-plating base metals for costume jewelry) or to modify some other aspect of the object's properties. For example, steel thus plated with nickel or chrome is harder and less prone to corrosion. (I think it might also be less prone to fatigue.)
• **Galvanoplasty** of bulk metal shapes, also known as electroforming or electrotyping, where the electroplating action is continued until it is much thicker than just a thin film. Historically geometry was imposed on the resulting shape by depositing it on the inside of a mold, like slipcasting of pottery, or on the outside of a mandrel in the shape of the desired object, but nowadays it should feasible to use electronic control of anode position and current to deposit metal selectively. Electroforming can hit nanometer tolerances, thus being suitable for reproduction even of holograms. Sometimes the term "electroforming" is limited to the case where the mandrel or mold is conductive and "electrotyping" to the case where it is not.
• **Electrochemical machining** simply reverses the roles of the

electrodes from galvanoplasty: instead of using the cathode as the workpiece and the anode as the tool, it uses the anode as the workpiece and removes parts of it using the cathode, much like EDM. But EDM passes a current between electrodes separated by a *dielectric* by producing an avalanche breakdown of that dielectric which produces plasma hot enough to *vaporize* part of the workpiece electrode and, usually, the tool electrode. ECM, by contrast, passes a current between electrodes separated by an *electrolyte*, carried by ions. As with EDM, by positioning the tool electrode, erosion can be carried out selectively in some places and not others.

The other uses of electrochemistry are corrosion removal, sacrificial anode corrosion protection, electrochemical batteries, and electrowinning of metals, which are not geometry-reproduction processes and so do not concern us here.

There is a gray area between electroforming and electroplating, "dimensional recovery", where a film is plated onto a metal part to enlarge it by microns to hundreds of microns. Since the non-mandrel side of the electroformed object has relatively uncontrolled geometry, this is usually preliminary to a later subtractive process such as grinding which produces the final geometry.

The electrochemical processes, both deposition and erosion, take place faster at some places and times and slower at others. They can be limited by ionic availability, especially for deposition, and by voltage. Generally the deposits are smoother when the limit is from ionic availability, while voltage limits tend to deposit dendrites (I do not understand why) so it is common to add organic thickeners to the water as "leveler brighteners" — originally gelatin and nowadays mostly secret chemicals, although some people have reported success with things like dishwashing detergent, vanillin, and corn syrup. (There are other kinds of "brighteners" also used in electroplating which work by other means.)

There have been some experiments using electrochemical machining to shape nonconductive materials such as soda-lime glass; the idea is that the electric field through the workpiece is balanced by an accumulation of ions on its opposite surfaces, one of which (in close proximity to a "cutting" electrode) is attacked by them. Since this section is dedicated to CFSs that are demonstrated to work, I will not further consider here these experiments, nor other possibilities like using anodic dissolution as a source of divalent cations to precipitate silicates, phosphates, organic anions, and so on.

In cases where dissolution of an anode is unacceptable, for example because no suitable anode is available, anodes of graphite, amorphous carbon, platinum, or palladium can be used; these will not dissolve anodically. I assume this is because they're held together by covalent bonds rather than metallic bonds, but I don't really know.

Deposition of metal onto the cathode is unavoidable — even coal and graphite can be electroplated, and have been since the very inception of the process — but if the cathode is not itself vulnerable to such erosion, the deposits can be removed thereafter simply by reversing the current.

A simple geometric CFS using electrotyping might make a mold using wax, paint graphite onto it, electrotype copper onto the

graphite, remove the copper from the mold, then cast a new wax mold on the copper. A more advanced version that avoids the dimensional-imprecision problem of wax shrinkage would use a parting layer, perhaps of graphite dust, to electrotype copper directly onto copper. My understanding is that this was common practice from a year after the invention of the process in 1848 until the 1930s.

A more complex geometric CFS using electrotyping and electrochemical machining would first use moving electrodes to selectively electrodeposit a metal, such as copper, into a rough pattern, then use electrochemical machining with moving electrodes to trim it to the precise shape. Each of these processes is individually well-explored.

Resin casting

A soft resin such as latex or silicone can form a mold for casting a hard resin such as a polyester or epoxy, and vice versa. Moreover, either type of resin can be used to manipulate the other kind in its semi-polymerized state. Resin polymerization differs from the liquid–solid phase change of conventional forms of casting in that it does not necessarily, or indeed normally, involve any change in dimensions. (Dimensional changes can be achieved by impregnating a soft resin with a solvent before or after casting, respectively shrinking or growing the product.) Resin casting is used by the Grating Lab to mass-produce research-grade diffraction gratings from a single master grating ruled on glass by a ruling engine.

Resin casting can of course also make molds for many other kinds of casting, use forms or patterns made by them, or modify the resin systems with fillers.

Others

There are a lot of other possibilities; I will mention a few of them here.

Selective etching is widely used in semiconductor and MEMS manufacturing; for example, hydrofluoric acid removes silicon dioxide, but not silicon or organic photoresists, while piranha removes organic photoresists but none of the layers in chips, including silicon dioxide. But it's also used in more prosaic ways: hot water with alum in it, for example, will eat steel but not aluminum, copper, tin, or zinc, a fact commonly used to remove broken drillbits; so you could imagine a CFS using alum in place of grinding to shape steel cutting tools for brass. Nonpolar solvents like carbon dioxide, alcohol, acetone, xylene, or toluene will usually dissolve many nonpolar organic resins but usually not sugar or ionic solids, while polar solvents like water, ammonia, glacial acetic acid, and ionic liquids (including deep eutectic systems) can dissolve many ionic solids like salt, sugar, or potassium silicate, but usually not nonpolar solids.

Bread dough is easy to shape. Calcining bread in a reducing atmosphere produces carbon foam, which is refractory to 6000°, more than hot enough to bake more bread in and even calcine it, or for that matter for casting metals, carbothermic reduction of iron, or even carbothermic reduction of aluminum. On Earth such a device may oxidize on the outside during operation, where it's exposed to air, but this can be tolerated in various ways: making it large enough to

survive one or more operations, coating the outside with a layer of something more resistant to oxidation (but not necessarily heat), or operating it in deep space or in a nitrogen atmosphere, for example. At human scales, amorphous carbon foam is a disappointingly weak material, but this is less of a problem at the micron scale where all the real action is.

Above I mentioned that clay bodies for pottery form a single-material CFS because they can be sintered into a kiln suitable for firing more of the same kind of clay; this is because of a curious property of sintering, that the material being sintered holds its form throughout, though not its dimensions. This is a general property of the sintering process, not limited to clay; granular polymers, glasses, metals, and other ceramics can all be sintered at temperatures below their melting points and while holding their shapes, and this is routinely done in many industrial processes. So in fact nearly any solid can be granulated and used in place of clay with appropriate binders, sintering aids, and atmosphere, and adequate temperature control; thus you can form a furnace capable of doing more of the same kind of sintering.

I have previously written about the possibility of using solid-solution "hardening" on sintered objects. The general outline of the process is that, before sintering, the "green" object contains at least a low-melting sintering aid and a high-melting filler; during sintering, the sintering aid solidifies and densifies the object (perhaps without fully melting, and perhaps partly dissolving the filler). Then you soak the object at a near-sintering temperature for quite a while so that the sintering aid diffuses into the still-solid filler. Given sufficient solubility of the sintering aid in the filler, the interstitial areas with pure sintering aid will disappear, leaving only solid solutions of the two (or more) materials, with the expanded solid grains in intimate contact with one another. For suitable mixtures, the resulting solid solution will remain stable even up to considerably higher temperatures.

If you squint hard enough, you could describe the hardening process of plaster of paris in this way; calcium sulfate hemihydrate is the "filler", water is the "sintering aid", and room temperature is the "sintering temperature" at which the water dissolves into the plaster, forming calcium sulfate dihydrate as the solid solution, which then remains stable up to some 150°. I suspect a similar dynamic is at play in the well-known use of boron donors as fluxes for soda-lime quartz glass, which I believe produces a borosilicate glass with a higher softening point than even the original soda-lime glass. (Boria melts at only 450°, but laboratory borosilicate glasses like type-7740 Pyrex can be used up to 500°, soften around 820°, and finally melt at 1648°, a temperature at which neat boria vaporizes rapidly.)

If the sintering aid forms a eutectic with the filler, it need not even be lower-melting; for example, a tiny amount of table salt can be used in this way to stick ice cubes together at temperatures between the melting points of the eutectic (-21.2°) and pure water ice (0°), even though salt's melting point is higher. The eutectic water-salt solution is initially liquid at the interface between ice and salt crystals, but after several minutes the salt diffuses into the ice until no salt or eutectic is left. So you can do this process at -20° and get a solid that

is stable, though weak and creep-plagued, up to 0°.

A very large number of binary, ternary, and quaternary solid-solution systems can be coaxed to perform in this super-sintering way at the right temperature in the right proportions. (The need to control the ice–salt reaction described above to within ±10.6°, i.e., ±3.9%, may be atypically demanding.) Moreover, their properties can be improved further by using the sinterable material itself as a binder for a different filler that is inert at the process temperatures; for example, you could thus use salted ice as a binder for sawdust (pykrete), or brass as a binder for steel (whether in the form of powder, chopped fiber, hollow spheres, solid spheres, some other shape, or some combination).

This is classified as a "geometry" possibility, since the circularity involved is that of using, say, brass tools to give the desired geometry to a "brass clay", which are then fired in a brass furnace made from the same material, to produce finished brass parts.

Stick, flame, wire, or friction welding might be a plausible way to additively build up the parts of a stick, flame, wire, or friction welder; the only part of the welding apparatus that experiences welding temperatures is the filler metal and the workpieces. As with EDM and ECM machines, parts of the welding machine need to be electrical insulators as well.

Earlier I mentioned that work hardening is not typically used as a way to harden metal so that it can cut or hammer out copies of itself, because the hardening process takes place during cutting or hammering. This may be less of a concern for cyclic fabrication systems than for traditional production systems, because the objective is to maximize reproductive rate rather than tool life, but there's also another possibility: bending and forming. You could imagine, for example, folding fingers out of aluminum foil which were sufficiently stiff to grasp other aluminum foil and fold it into more fingers, or bending a wire into a tight coil that is then used as a tube to guide other similar wire to be similarly bent. And of course you can clearly use wadded-up aluminum foil to make forms between which you press virgin foil. In cases like these, where it is possible to bring many folds or coils of tool to bear on a single fold or coil of workpiece, work-hardening might be more practical — desirable, in fact, since work hardening is the negative-feedback mechanism that distributes the bending evenly along a curve instead of concentrating it at a kink, as happens when you try to bend a drinking straw.

(Aluminum foil in particular is an appealing raw material for experimentation because it's cheaply and easily available, ships pre-annealed, work-hardens readily, has good mechanical properties in the tempered state, and is typically of some 10 μm in thickness, with submicron roughness on one side.)

I mentioned foamed clay earlier, as well as carbon foam from bread. Foamed materials have a variety of potentially appealing properties for cyclic fabrication systems: they tend to be much better thermal insulators than the fully-dense material, which helps make chambers capable of heating and cooling; achieving a given stiffness with the foamed material requires much less mass, though more volume, than doing it with the fully-dense material; and they have very appealing

cutting properties. Sintering clay or other materials causes them to shrink, and not perfectly uniformly, so it is common to grind sintered parts to precise dimensions after sintering. Foaming greatly facilitates such cutting because the bubbles tend to arrest crack propagation and reduce the material's hardness. The 25%-dense foamed pottery I made was soft enough to be carved with a fingernail, and it is common for people to cut foamed refractory silica-alumina firebricks with woodcutting tools. Foams also tend to have a Poisson ratio close to 0, which is potentially helpful for dimensional precision. Finally, foams tend to be much more flexible, even elastically, than the fully-dense materials they are made from, which can facilitate the use of flexures.

Polymer-derived ceramics are a very interesting possibility discussed in some detail in Pyrolysis 3-D printing (p. 242), which also mentions another "supersintering" alternative to the diffusion-based system mentioned earlier: if the "sintering aid" responds to heat by pyrolyzing into a solid substance that is stable to higher temperatures, like the bread dough mentioned earlier, it will produce a solid object with geometry stable up to those higher temperatures. Depending on the geometry and strength of the filler particles, the pyrolysis products need not even be particularly strong to produce a strong object.

I've previously written about using various kinds of chemical cements for 3-D printing, whether powder-bed or extrusion, focusing on those that can be activated relatively quickly, including things like double-metathesis reactions, pH-activated gelling agents, heat-induced solvent evaporation, and the precipitation induced by divalent cations in a number of aqueous ionic systems. Almost any of these approaches, if workable, would provide a CFS.

For example, in "Likely-feasible non-flux-deposition powder-bed 3-D printing processes" in Dercuano, I suggested selectively jetting water onto a powder bed consisting of 1.6 kg/l quartz sand, 170 g/l unfired-bentonite clumping cat litter, 270 g/l calcium chloride, and 190 g/l diammonium phosphate, with the calcium chloride wet-mixed with the cat-litter bentonite, then the mixture dried and powdered, then dry-mixed with the other ingredients and thereafter protected from air; perhaps adding a minority of wood flour, I suggested, would help with tensile strength. This mixture was hypothesized to set up rock-solid immediately upon being moistened by precipitating calcium phosphate in the interstices of the bentonite. If such a mixture works and produces a strong solid, you could probably build the entire 3-D printing machine out of it. (Except for the electronics, of course. And perhaps a little sealant for the water pipes.) None of the ingredients or their processing would cause any difficulty to the cemented product, unlike the case with trying to grind steel with the same steel or cast brass into a mold made of the same brass.

Metrology

The fundamental difficulty of metrology is to make finer measuring instruments. With feedback and incremental refinement, it is easy to make geometry finer than our manufacturing processes: we alternate between measuring the geometry and refining it, thus approaching perfection to within the limits of our materials and

measurement capability, however crude our tools may be. Thus, for example, a patient worker with pine pitch, fine abrasive, two glass blanks, a razor blade, and a candle can grind a parabolic mirror to within a fraction of a wavelength of light.

However, making measurement instruments that can measure more finely than our existing measurement instruments can measure, that is a real difficulty. There are many aspects to this difficulty! To name a few axes, there is the kind of quantity being measured: mass, length, volume, luminous intensity, time, angle, information, temperature, frequency, force, pressure, speed, density, voltage, electric current, energy, power, magnetic flux, magnetic flux density, electrical resistance, inductance, capacitance, area, stress, elasticity, tensile strength, compressive strength, roughness, viscosity, refractive index, specific heat, spectral radiance, pH, chemical concentration, magnetic permeability, electrical permittivity, etc.; the measurement to a desired precision of an absolute standard, such as a given number of oscillations of a given spectral line, or the distance traveled by light in a given time; the accurate subdivision of such an absolute standard or a standard derived from it; the shielding, balancing, or cancelation of unwanted effects that disturb a given measurement; the estimation of how successful such efforts have been; and so on.

Topics

Foil-marking glass

Kragen Javier Sitaker, 02020-08-18 (4 minutes)

(Untested.)

Suppose you have a layer of aluminum foil or gold leaf on top of a piece of glass and you vaporize a pinhole in the metal with an arc. You can position the electrode that creates the arc to submicron precision, and by controlling the energy of the arc, you can vaporize a precisely controlled amount of the metal, creating a hole comparable to or slightly smaller than the thickness of the foil — household aluminum foil is about 10 μm, aluminized-mylar Doritos bags are about 1 μm, and gold leaf is about 0.1 μm. Smaller controlled thicknesses may be feasible if deposited onto a substrate like boPET or polyimide, for example by vacuum coating.

Some of the vaporized metal will impact the glass, blowing air out of the way in the process, and condense there, thus locally depositing metal on the glass.

This provides a simple way to locally deposit a fairly precisely controlled amount of metal onto a substrate, insulating or conductive, with submicron positional control. Aside from the potential utility of marking the surface of the glass or other substrate, by repeating the process layer by layer, you can manufacture arbitrary three-dimensional shapes from the metal with submicron precision, and much higher material deposition rates than are possible with electroforming.

Doing this with noble metals such as gold, platinum, or iridium should be possible even in air, but more reactive metals such as silver, copper, aluminum, and iron probably require an inert-gas atmosphere, or at least nitrogen. Conductive oxides like those of lead or silver might permit the use of this process for simultaneous arc deposition and reduction in a reducing atmosphere such as acetylene, but hydrogen contamination of the resulting metal might reduce the utility of this approach.

Electric arcs can easily reach temperatures high enough to sublimate even carbon, so this process can deposit even very refractory metals like tungsten, molybdenum, or tantalum. Moreover, by putting a little distance between the substrate and the feedstock, very thin films can be formed.

In most cases, though, minimizing the distance between the substrate and the feedstock would be desirable. One way to achieve this is to cut channels at known locations into the surface of the substrate, say a one-millimeter grid of 100-micron-wide channels, and pull a light vacuum on those channels. In that way the gas necessary to sustain the arc in the process as described above can be employed to eliminate the unwanted gap rather than sustaining it.

Under vacuum, instead of an arc, a high-power electron beam or focused ion beam could instead be used to locally vaporize the feedstock, as in e-beam etching and FIB milling. This should permit nanometer resolution rather than the mere submicron resolution routinely attainable with mechanically positioned tooltips.

Of course, the localized heating of the glass substrate may also have effects, desirable or otherwise. If they are desirable, the optimum "feedstock" may be a film of conducting graphene or graphite rather than a metal. And, as Russ of Sarbar Multimedia points out, you can use a laser rather than an arc to heat the feedstock, and if the substrate is transparent, you can place the feedstock on the opposite side of the substrate so that the laser side of the feedstock is the one touching the substrate.

Topics

Inductively-coupled plasma torches

Kragen Javier Sitaker, 02020-09-10 (5 minutes)

An inductively coupled plasma torch could operate at atmospheric pressure without consumables. An initial seed plasma is provided by a glow discharge, a short-lived conventional arc, or (if we permit consumables) a conventional oxidation flame; then it is advected into the center of an induction coil, where inductively-coupled power transfer brings it to a high temperature and propagates it opposite from the advection direction to sustain it in a constant position. In addition to conventional water cooling, the coil can be separated from the plasma by low-permittivity, non-ferrimagnetic, insulating refractory ceramics; for example, magnesia, lime, silicon nitride, alumina, urania, thoria, or boron nitride; the ceramic itself might be actively cooled as well. (Refractories unusable due to high conductivity include graphite, amorphous carbon, silicon carbide, tantalum carbide, zirconia, and the diborides and nitrides of hafnium, titanium, and zirconium; and silica is probably too low-melting, although fused quartz does have an attractively low TCE.)

The electrodeless plasma thruster article suggests further possible ways to initiate plasma formation, including electron guns and laser ionization, and I suppose in theory a sufficiently powerful ultrasound wave converging on a point ought to heat it enough to produce plasma too, as in sonoluminescence, but doing that without a liquid seems like it would be hard.

The problem remains of how to limit the damage to the ceramic walls from the plasma, since plasma-ceramic contact would surely ablate the surface fairly rapidly; under uniform conditions the outer plasma will tend to shield the inner plasma from receiving inductively-coupled energy, so the natural tendency is for the plasma zone to grow. Even under adverse applied magnetic field conditions, by establishing a gas-flow profile within the induction ring in which the flow near the walls is much faster than the flow in the center, it should be possible to adjust the induction power so that the plasma is self-sustaining only in the center, while being blown away faster than it can form around the outside. (Of course, if the plasma were to reach the ceramic wall it would also be self-sustaining there, since in contact with the wall it would be stationary, but the plan is to avoid this.)

It might be possible to manipulate the magnetic field conditions instead of the gas flow conditions to keep the plasma away from the walls, for example by making the induction coil smaller than, and axially displaced from, the ceramic aperture. I think this would imply that the field would get stronger axially into the torch body, creating a strong tendency for the plasma to propagate into unprotected areas of the torch, but this could be countered by a stronger negative advection divergence: all the gas closer to the induction coil would be moving too quickly for the plasma to spread into it. I'm not sure if this is feasible.

In this scenario there is still radiative transfer of heat from the plasma to the ceramic walls, but this can easily be kept low enough to avoid wear to the ceramic. If an arc between conventional graphite electrodes is used to initially ignite the plasma, the electrodes will erode somewhat, but if we're talking about one spark every 20 minutes of use or something like that, it should be easy to make the electrodes big enough to last the life of the rest of the torch.

Such a plasma is of course easier to sustain in gases like argon or at lower pressures, but air plasma has the great advantage of not requiring any consumables, just an air compressor.

Probably the frequency required to efficiently couple into the plasma would be on the order of a megahertz for human-hand-tool-sized torches, hundreds of kilohertz for larger torches, and several megahertz for smaller ones.

The torch might require active electronic control at submillisecond timescales to stabilize the plasma and keep it from either blowing out or flashing back. Both the complex impedance of the induction coil and the blackbody radiative flux from the hot plasma could provide crucial feedback information.

Operating such a torch in a pulsed mode might be feasible and simplify the process further: the induced current in the plasma tends to Z-pinch it into a toroidal plasmoid while repelling it from the induction coils, and hence from the torch. Sakharov reportedly took this to the logical extreme by vaporizing a small aluminum ring with eddy currents into a self-contained plasmoid traveling at 100 km/s, powered by an EPFCG.

Topics

Oxygen generator rocket

Kragen Javier Sitaker, 02020-09-10 (1 minute)

Could a rocket use chemical absorption of oxygen and hydrogen into some other chemical to provide storable non-cryogenic fuel for an engine with the attractive 450-to-528-second specific impulse of LH_2/LOx engines?

Obviously if you allow water the answer is "yes", but then an unattractively large amount of energy is taken up in "releasing" your "fuel" from its "storage". Like, several times more than your engine yields. This might be okay under some circumstances but probably not the ones where people want to use chemical rockets, like escaping a gravity well.

The problem with other forms of fuel storage is that they leave storage mass behind that is comparable to the mass of the fuel itself, which is a nonstarter when you need ratios like 100:1 just to hit orbit.

Maybe, though, you could grind up this "ash" and either dump it out or, especially early in the process when you want more thrust and less velocity, mix it into the reaction mass. I don't think that saves the idea.

Topics

Penalized bits

Kragen Javier Sitaker, 02020-09-10 (3 minutes)

In topology optimization, you typically design a structure for maximum rigidity by beginning with a block of fog and then using gradient descent to minimize a penalty function which has a few different terms: one for the structural property of interest (such as rigidity under a given load), one for regularization of the problem to rule out physically-unrealizable checkerboard solutions full of discontinuities, and a third fog-penalty term which forces the elements toward 100% density and 0% density and away from 50% density, again preferring physically-realizable models.

What if you apply the same approach to bits? Suppose, for example, that you want to find the representation of an integer n in binary. You could start with 32 real numbers initially set to 0.5, then use gradient descent or something to optimize $|2^0v_0 + 2^1v_1 + 2^2v_2 + \ldots + 2^{31}v_{31} - n|^2 + \alpha\Sigma_i v_i^2(1-v_i)^2$, for example. This function's only zero (for real v_i and positive α) should be the correct binary representation of the number. At all other points it takes on strictly positive values, and it's differentiable everywhere. Moreover, although I haven't looked, I think it's convex, so its only local minimum is the global minimum. So it should be tractable for gradient descent. Certainly gradient descent with random restarts should always solve it, though if the random restarts are actually required then maybe it would take an exponentially large time for such problems. Genetic algorithms should have no trouble solving it in a reasonable amount of time.

Now, although it's I hope at least highly plausible that the above approach will work for such a simple problem, think about more interesting Boolean functions. For example, given a binary multiplication algorithm, the above approach can probably do division, or, more interestingly, a square root. Can it do LDPC decoding? How about inverting other less tractable functions on bitvectors, like a round of SHA-256? If you write down some inputs and outputs of a branch-free block of instructions for some CPU, and express the execution of a few arbitrary instructions as a similar Boolean function of those instructions' bits, can it do superoptimization?

Purely agnostic approaches like this — perhaps we should say "knowledge-free" or "ignorant", or "assumption-free" if we want to use a euphemism — will surely be inefficient for many problems, even if they can solve them at all. Suppose we train a neural network on the distribution of plausible solutions, as explored by Lunz, Öktem, and Schoenlieb for inverse imaging problems: as we apply ad-hoc penalties in topological optimization to solutions containing fog and singularities, we can train various kinds of neural networks to recognize the structure of plausible solutions, using them to penalize unlikely solutions, such as superoptimized code containing invalid instructions. This way our search can probably converge much more quickly than a purely ignorant search that doesn't know a multiplication from a superoptimization.

Topics

Phosphate precipitation

Kragen Javier Sitaker, 02020-09-10 (12 minutes)

I finally started trying out the recipe I'd come up with in Dercuano for a kind of instant 3-D printing cement based on the precipitation of water-soluble phosphate by pretty much any polyvalent cation. (See "Likely-feasible non-flux-deposition powder-bed 3-D printing processes".)

Initial experiences

So I bought 2 kg of calcium chloride desiccant (AR$778, US$5.72, US$2.86/kg) and 2 kg of diammonium phosphate fertilizer (AR$850, but AR$470 of that was delivery; the marginal cost of the fertilizer is AR$190/kg, US$1.40/kg).

The first observation is that this fertilizer is not pure diammonium phosphate. The individual prills have substantial variation in color, and they do not dissolve fully in water, even at boiling. A slight ammonia smell evolves on boiling the water, and is absent from the bags of fertilizer. Filtering the liquid through a coffee filter produces a transparent brown syrupy liquid, leaving most or all of the solids behind (I'm doing this in cut-up aluminum cans, which are not as good as glassware for seeing small amounts of cloudiness).

This phosphate liquid fails to dry even upon being sealed in a room-temperature drying chamber sharing air with the calcium chloride for several days. (The chamber is Saran Wrap over the top of a cut-off can, so it may be leaky, but I don't see any deliquescence on the calcium chloride, so it's at least not very leaky.)

The phosphate liquid instantly produces a thick white suspension of a fine precipitate upon being poured into a solution of the calcium chloride. Presumably this is some kind of calcium phosphate, along with whatever fluoride may have been present as a contaminant.

After filtering through another coffee filter, it has the mouthfeel of pure clay, making my teeth slide against one another with quite a bit of difficulty, but no grittiness, demonstrating that there are no crystals above the micron scale. The taste is also slightly bitter and salty, so I probably didn't wash the filtrate enough. To the touch of the hand, the suspension resembles a thin kaolin slip. It dries on the skin to a powder resembling rock-climbing chalk.

The calcium chloride seems relatively pure: it is plain white, dissolves completely in water, and its only smell is a faint whiff of quicklime. Left in open air for a few days with a drop or two of water, it gradually begins to deliquesce, producing a liquid that feels "oily" because it's not evaporating. Nevertheless, it is not a food-grade chemical either, and it's labeled "for industrial use only", so I shouldn't have tasted it.

I also added some of the phosphate solution to an aqueous solution of some magnesium chloride I had lying around, which also produced an immediate precipitate of, presumably, trimagnesium phosphate, dimagnesium phosphate, magnesium ammonium phosphate, or a mixture. This precipitate was slightly brown in color and settled out

fairly quickly, while the calcium precipitate did not visibly settle out at all over the minutes before I filtered it. Presumably both of these differences owe to the crystals being larger or rounder than the calcium precipitate. The magnesium precipitate tastes the same as the calcium precipitate.

Upon drying, the calcium precipitate has a consistency somewhat like dried mud; I can pick up pieces of it with my fingers and break them apart with my finger relatively easily. However, rubbing it between two fingers breaks it into a white powder too fine to have any gritty feel, rather like cornstarch. So it seems that the ammonium chloride (or whatever) that is binding together the crystals of apatite (or whatever) isn't able to hold them in clumps of more than a micron or so in size; it might in fact just be van der Waals forces between the apatite crystals.

After a couple of days of room-temperature air-drying, in the calcium precipitate one crystal large enough to glint in sunlight can be seen from the proper angle, but the rest of the powder still appears as a matte-white, purely Lambertian surface. Some 10% contraction on drying is evident.

I have not managed to acquire clay yet, but it occurs to me that this calcium phosphate powder (if that is what it is) is probably an adequate alternative and may be a superior one. The grain size is about the same as that of clay, the expansion upon absorbing water is probably smaller than clay's and perhaps insignificant, the crystal habit can be made to be needlelike or (like clay) platy, the price is only a little higher, and the aspect ratio of the grains should be only a little worse. Where it might be superior is that the apatite cement I propose to selectively deposit can clearly bond well to these grains, while its ability to bond to grains of clay remains a significant unknown. Also, the lower expansivity might enable it to produce a higher-density final composite material.

Gargouri et al.'s purification

A 2011 paper from Gargouri et al., "Synthesis and Physicochemical Characterization of Pure Diammonium Phosphate from Industrial Fertilizer", explains that in Tunisia the "diammonium phosphate" industrial fertilizer is only 75% diammonium phosphate, the remainder including "Co, Cu, Fe, Mn, Mo, Ni, Zn, F, As, Al, Hg, Pb and Cd". They report getting their cheap industrial DAP almost as pure as the laboratory DAP they bought from Fisher, simply by recrystallizing it with 70% water and 30% alcohol, decoloring with charcoal. They report these results:

ppm	Fe	Al	Mg	Ag	As	Co
Pb	Hg	Si	Sn	Ti	Cr	Zn	Cd	Cu	Ni	Mn	V
plant DAP	6769	4273	4907	6	26	5419
22	3	150	382	93	525	1203	34	59	25	65	1341
plant DAP recrystallized (water-alcohol)	24	37	14	-	3	3
7	-	70	-	2	27	41	3	4	17	1	47

| commercial DAP (Fisher) | 15 | 22 | 9 | - | 3 | 2 | 0 | 7 | - | 38 | - | - | 25 | 11 | 3 | 2 | 17 | - | 9 |

This amounts to a reduction from 2.5% of these impurities down to 0.3%. 2.5% is a lot less than 25%, and I'm not sure what happened to the other 22.5%; it might be impurities they also removed but didn't measure, such as O (for example in OH or SiO_2), Ca, and F. Their analysis of the P and N content before and after their purification (46% and 17.7% before, 49% and 18% after) does not support the possibility that 25% of the original material was made of non-ammonium, non-phosphate components. However, some of the "25% of impurities by weight" they cite might have been compounds like ammonium fluoride and magnesium phosphate. Or maybe it was just a typographical error where they were missing a decimal point.

I should see if filtering with charcoal reduces the brown color. Also, especially if I can get vacuum filtration set up, recrystallization as per the standard procedure would eliminate impurities that are still soluble in the solution after cooling. Greg Sittler suggested a water-driven venturi as a vacuum pump.

Another approach is to make the solution basic, which will precipitate hydroxides of (among other things) iron, nickel, copper, and cadmium, but not ammonium or phosphate:

g. All hydroxides are insoluble except those of the alkali metals. ... Ammonium hydroxide does not exist.

The usual way to do this is with lye, but I don't have access to lye; however, household ammonia solution should also work. Also, sodium carbonate or sodium bicarbonate, which I do have, would precipitate the transition metals ("e. All carbonates, sulfites, and phosphates are insoluble except those of sodium, potassium, and ammonium"), but by the same token you would think those would be precipitated already in a phosphate-rich environment. (Iron(III) phosphate, ferric orthophosphate, is slightly soluble in water, but probably not enough to give the brown color.) So maybe I should try it and see what happens but not expect success.

Other notes on next steps

Previously I'd written that you'd want to get the ammonium chloride out of the finished piece by leaching it out with water. But ammonium chloride evidently dissociates and "sublimes" at 337.6°; initially I thought the mix of corrosive gases it produced would be something I wouldn't want around, but apparently the gas on cooling re-neutralizes to ammonium chloride rather than going around corroding solid objects it encounters, so that might actually be a reasonable way to remove the side product.

I guess the immediate next step is to dissolve some calcium chloride in water and soak a little sand, a little of the supposed calcium phosphate powder, and a little of a mixture of both with it, then let it dry. Actually ideally I would do this with both calcium chloride and what I suppose to be DAP in order to see what the resulting substances are like, since I suspect that calcium chloride in between the grains of filler will work better than the other way around because it will favor needlelike nanocrystals. But that might turn out to be

wrong.

A further thing to try might be to use different pH levels. I have household ammonia to alkalinize the mix pretty thoroughly, but like the ancient alchemists, no strong acids.

Witch-burnings, thoughtcrime, and Inquisitions: how to avoid torch-wielding peasants

As always with scholarship, there is danger from the thoughtless prejudice of the ignorant, which so often has turned into violence, as in the cases of Giordano Bruno, Aaron Swartz, Alan Turing, the Maya codices, and Qin Shi Huang's burying of the scholars.

Ammonium chloride is on the national list of "controlled chemical substances" which unauthorized people are not allowed to have or make; but, then again, so are everyday products like aqueous ammonia solution, lye, acetic acid, ethanol, isopropanol, methyl ethyl ketone (the solvent in dry-erase markers), quicklime, slaked lime (whitewash), acetone, ethyl acetate (nail polish remover), red phosphorus (as found on matchboxes), nitromethane, sodium carbonate, sodium bicarbonate, and kerosene. Phosphorus, hydrochloric and sulfuric acids, toluene, dichloromethane, and acetone are even in "list 1" along with actual drugs I won't mention here. Ammonium chloride is in "list 3" along with ethanol, isopropanol, sodium sulfate, and kerosene.

The definition of "product" is something of 30% purity or better of (the total of) substances from lists 1 or 2 P/V (which I suspect means "per volume"), or 20% purity or better of hydrochloric acid or aqueous ammonia; except that if it's impossible to separate the substances by physical means, higher concentrations may be approved on a case-by-case basis. Perhaps this is the reason I can buy vinegar at the grocery store, lye at the hardware store, and nail polish remover at the pharmacy, even though they are all in list 2: they are dilute.

Notably absent are sulfates (of anything but sodium), sulfur trioxide, sodium percarbonate, phosphoric acid, nitric acid, and nitrates (though sodium *nitrite* is included in list 3).

So, I think as long as I stay away from acetone and hydrochloric and sulfuric acids, I shouldn't run into any pitchfork-wielding peasants.

Topics

Notable quotes from Steinmetz's 1892 hysteresis paper

Kragen Javier Sitaker, 02020-09-10 (2 minutes)

I was reading Steinmetz's 1892 paper about hysteresis, and I was struck by the readability and elegance of the prose used by learned societies at the time, next to which most modern academic writing feels leaden:

> The subject that we have to-night before us, and which you will find so ably dealt with by Mr. Steinmetz, relates to that phenomenon of molecular friction, which Mr. Ewing has denominated "hysteresis." Mr. Ewing, as we all know, has made the subject so peculiarly his own, that one might at first suppose there was nothing new to be known about it; but I am confident that after this paper is read, those of us who read it with Mr. Steinmetz will find that there is something new under the sun. We will now hear Mr. Steinmetz's paper.

This was at the beginning of 1892; in 1891 he had published "Elementary Geometric Theory of the Alternate-Current Transformer" in *Electrical Engineer*, volume (?) 11, 1891, "627ff. on p. 627; 12 (1891): 12ff." He *also* published things in *German* starting in *1890*: "Das Transformatorenproblem in elementargeometrischer Behandlungsweise", *ETZ*, 11 (1890): 185-186, 205-206, 225-227, 233-234, 345-348, and in 1891, "Anwendung des Polardiagrams der Wechselstrom für inductive Widerstände", *ETZ*, 12 (1891): 394-396, 405-407.

Here, in the Q&A, he tries to set that jackass Pupin straight about some mistakes Pupin was making:

> On the other hand, to make the current considerable only for a moment, while immediately before and after it is small, either the induced E. M. F. must suddenly decrease enormously, and the next moment increase just as suddenly --- which is impossible, because it is the differential quotient of magnetism --- or the primary E. M. F. had to rise and decrease again very suddenly, and such a sudden rise, and immediately afterwards decrease of primary impressed E. M. F., not only is an electro dynamic alternator unable to produce, but no electric circuit would permit a current of such enormously large value and short duration to pass.

Topics

The programmable world

Kragen Javier Sitaker, 02020-09-10 (0 minutes)

Moore's Law has come to its end. The astounding exponential progress in micro- and nanofabrication of electronics that has characterized the Transistor Age, since Engelbart first observed it in 01959, has leveled off; CMOS clock speeds have remained frozen at 0.5–4 GHz since 02005, and

We are seeing a

pictures under glass

3-d printing

dynamicland

microcontrollers

optimization

experimentation

security and privacy

Topics

Smart plumbing

Kragen Javier Sitaker, 02020-09-10 (updated 02020-09-12) (11 minutes)

Home device networking, sometimes called home automation, is elusive, despite predictions of "home computers" going back to at least Gordon Moore's "Cramming" article in 1965, and Ray Bradbury's speculation in his 1950 story "There Will Come Soft Rains". X-10 powerline signaling was developed in 1975 and on sale in Radio Shack and Sears in 1978; it was affordable by the late 1980s. Despite this, home device networking mostly remains a novelty, very unlike, for example, networking and automation in industrial manufacturing. Automation has crept into the dwellings of the humans mostly in non-networked forms: toilet tank float valves; washing machines; dishwashers; thermostats on the air conditioner, furnace, and hot water heater; timers on lights; shuffle on the CD player. Now in 2020 home illumination is beginning to be automated in earnest, with multiple competing brands of color-changing LED lightbulbs being used by criminals to launch distributed denial of service attacks.

Still, though, the majority of the house remains stubbornly unmechanized. In my view, this is largely the fruit of twelve millennia of building houses to remain habitable despite being entirely passive.

Water

This leads to some drawbacks. Consider how smart plumbing might work. A small pressurized tank under your sink provides immediate availability of high-volume water at whatever pressure you choose; the tank refills at leisure by requesting water from upstream, which can come by way of a trickle sized for average flow rather than maximum flow. A shower, for example, might use 15 liters per minute (250 ml/s) for half an hour per day, for an average of 5 ml/s, 50 times less.

This trickle could travel through a tiny pipe the size of a coffee stirrer, but alternatives include miniature aqueducts, greenhouse groundwater filtered through sand, a tiny fountain trickling down the bathroom wall, and a mobile broom-shaped robot with a bucket. None of these are prone to the catastrophic failure modes that characterize traditional high-pressure in-wall water pipes, such as flooding your basement and destroying everything stored there, Fantasia aside.

Thermal and humidity control

Michael Reynolds's Earthships are designed to permit passive climate control by way of a thermosiphon-driven seasonal thermal store consisting of an earth berm somewhat larger than the house itself with cooling pipes running through it. The greenhouse section of the house, with near 100% glazing and facing the equator, is heated by the sun. To heat the house's living spaces, the cooling tubes are

closed and the doors to the greenhouse are opened, permitting natural convection to carry the heat into them; to cool the living spaces, the cooling tubes and the greenhouse's skylight are opened, so that natural convection carries hot air out of the greenhouse, which is replaced by air that passes through the cooling tubes into the living spaces, from which it passes into the greenhouse with difficulty past the closed doors.

Reynolds and typical hippieish Earthship buyers see the manual opening and closing of the skylight and cooling tubes as an extra benefit: it keeps the residents in touch with the natural climate, and because of the large thermal mass of the walls, floor, and roof of the living spaces, it's rarely needed even for comfort, and never for simple safety. But more sophisticated redundant thermal homeostasis systems, trading off different candidate thermal stores based on remaining reserves and predictions of future weather, could probably do the same job without all the expensive earthmoving.

For example, when plenty of energy is stored in the house's batteries due to recent sun, or especially when they are already full and the sun is still shining, it might be essentially free to run a vapor-compression or ammonia-absorption heat pump to top up thermal-mass or phase-change reserves of either heat, cold, or both, or to directly heat or cool the living spaces. When heating the living space, the cold from the heat pump's evaporator (or corresponding part) might be more efficiently stored in a cold thermal reservoir instead of vented to the outside air.

Cold reservoirs above the house and hot reservoirs below it permit the natural convection of air through butterfly valves like those in a car's throttle, which consume energy only when their setting is being changed; this permits not only manual system operation in the case of a power failure but also fail-safe measures where, if nobody is home, a default thermal coupling to the reservoirs keeps expected temperature swings within safe limits. Such reservoirs can in many cases do double duty as drinking-water cisterns.

When the reservoirs run low, or an inability to replenish them for a long time is predicted, such active heat pumping can also reduce the drain on their thermal stores for future use. Also, house ventilation to the outside through countercurrent or regenerative heat exchangers, which costs some heat or cool as well as humidity control, can be diminished in exchange for increased use of forced-air HEPA filters — a measure also warranted when outdoor air quality is poor, for example during the acoughalyptic wildfires ravaging California and Washington as I write this.

And, of course, when trying to keep the indoor temperature from rising, lowering equatorial-facing awnings to shade windows, and raising polar-facing awnings to unshade them, is another possible alternative, which must be traded off against reduced garden growth and glossy LCD visibility, on the bright side, and reduced illumination and increased human depression, on the dark side. This is also sensitive to time of day: east is "equatorial" in the morning, "polar" in the afternoon, and west vice versa, while both are super "polar" at night.

Another tradeoff that can be made is the evaporation of collected

rainwater in rooftop tanks to reject stored heat, especially at night, with a fan, or when it's windy; or the heating of water in passive solar collectors to acquire stored heat. Such passive solar collectors can do double duty as awnings; such evaporation tanks can do double duty as swimming pools. A cooling tower, indoors or outdoors, is also a fun place for kids to play in the spray on a hot day.

Different kinds of thermal stores may require different quantities of resources and have different "self-discharge rates" as well as different impulse responses. Water at 0° represents a cool resource of some 100kJ/kg when the objective temperature is 23°, and can be stored in a pit lined with geomembrane, possibly surrounded and topped with some kind of insulation. Ice at 0° represents an additional cool resource of another 333kJ/kg when the objective temperature is anything above 0°, so you can quadruple the density of your storage if you can reach a temperature below 0°. A bunch of thin coolant pipes running through a trench in a yard, a few centimeters apart, can heat and cool the soil in a cylinder a few centimeters around them as a daily thermal store; but if they are instead a couple of meters apart and deep, they can also heat and cool the soil a few meters around them as a seasonal thermal store. These stores can be orders of magnitude larger than water tanks of the same cost, but their available heat flux is lower.

Other phase-change materials that could be useful in this context include Glauber's salt — sodium sulfate decahydrate, which melts at 32°, yielding 252 kJ/kg — and a eutectic of NaCl and Glauber's salt, which melts at 18°, yielding 286 kJ/kg. Glauber's salt can provide a very compact high-temperature reservoir, while the eutectic with sodium chloride can provide a low-temperature reservoir which, though it provides less heat capacity than water, can be operated at a much more convenient temperature. Glauber's salt costs some US$0.05/kg, some fifty times as much as water, but a 10-megajoule heat or cool reservoir — roughly a person-day with perfect insulation — adds some US$2 to the materials cost.

For cooking or hot-water heating, it may be worthwhile to use a higher-temperature thermal store than Glauber's salt can provide — for example, a pebble bed of stones or ceramic beads over which air is passed, or a phase-change reservoir of sulfur that melts at 115° yielding 54 kJ/kg, or of the famous "solar salt", a eutectic that melts at 220°.

Even simple conversion of battery energy into heat may have a role; the wattage of the active heat pump will inevitably be limited, perhaps to a kilowatt or so, but there is no need for such a limit on resistive heating. I bought a 600-watt nichrome resistor for boiling water a few weeks ago for US$1, complete with shitty power cable and shitty plastic protective cage and ceramic form.

So the climate control system, responding to human commands, has many available tradeoffs. Among others, it can spend water to get cool; spend battery to get cool, heat, or both; spend battery to generate light; spend light to get cool; spend water and air dryness to get cool, or spend heat and water to raise air humidity; spend battery to get air purification; spend heat or cool, depending on the outdoor temperature and pollution level, and possibly air humidity or dryness,

to get air purification; and spend cool to generate light and garden growth. (This is not counting the small amount of battery needed to pump air and water around and operate valves.)

By using active control of these tradeoffs with cybernetics, optimal control theory, Bayesian modeling of future climate, and Black–Scholes option theory, it should be possible to achieve Earthship-like comfort and security without the orders-of-magnitude overprovisioning that makes the Earthship design so expensive.

Topics

Inorganic burnout

Kragen Javier Sitaker, 02020-09-11 (updated 02020-09-12)
(18 minutes)

Foamed pottery, as described in Cyclic Fabrication System (p. 260) , is a broadly useful product, easily improving the properties of fired-clay ceramics for a number of uses. However, in its usual form, not only does it make a terrible smell; it also requires organic materials as an input, actually in larger volume than the clay; it requires oxygen to react with the materials; and it requires a fairly high temperature, at a bare minimum 250° but typically more like 800°. These requirements are not always desirable or feasible.

There are other uses for solids that can be thus "burned out" without disturbing surrounding materials; they can be used as support material for initially unsupported things, for example during assembly or 3-D printing, and investment casting and other lost-wax casting relies on burning out the wax from the mold. Similarly, in lost-foam casting, the foam pattern is burned out of the mold by the hot metal being cast.

So what kinds of "support materials" would make good candidates for such processes?

Evaporants for burnout

If you need to do your burnout at low temperatures, carbon dioxide sublimes at -79° and is very cheap and pretty inert. A number of other common compounds are solid at accessible temperatures below room temperature and are then easily evaporated, such as ammonia (-78°, boiling at -33°), sulfur dioxide (-72°, boiling at -10°), the highly toxic cyanogen (-28°, boiling at -21°), and water (0°). However, I can't think of any such compounds that don't melt first at atmospheric pressure or are totally nonpolar like CO_2; nonpolar compounds mostly tend to have a pretty wide liquid range, which is annoying here. At even lower temperatures the toxic, inflammable CS_2, may be an option, melting at -112°, but it doesn't boil until 46°! It's famous for dissolving insoluble things like cellulose, phosphorus, rubber, sulfur, and asphalt.

But at higher temperatures there are a number of inorganic and mostly-inorganic compounds that are easy to vaporize or thermally decompose into gases, which can perhaps then escape (through pores in the mass you want to remove them from, if they're embedded within it, as in the case of investment casting). Sometimes these gases are reactive, toxic, or both, especially at high temperatures; for example, table salt (NaCl) melts at 801°, boils at 1465°, and starts evaporating rapidly at much lower temperatures of 1100°–1200°, a property used in preparing fired-clay pottery — but the resulting sodium gas is highly reactive! Catalyzed by steam, it reacts with the surface of both the pottery and the kiln (or saggar) to produce a sodium silicate glaze, the desired result.

Even before the burnout, it's possible for solid support materials to react, especially if the material they're supporting is fully or partly

liquid, as is the case with pottery clay bodies, for example, which are colloids plasticized with water. For example, dinitrogen pentoxide is a crystalline solid which melts at 41° into a liquid which boils at 47°, so it might seem like a reasonable candidate; but it reacts violently with water to form nitric acid, which can oxidize a wide range of materials to nitrates, so it will not work in systems where water may be present.

So with this burnout process, questions of support-material compatibility arise, especially at higher temperatures, as well as human and environmental safety if the process is being done near humans or within Spaceship Earth or another spaceship.

Candidate high-temperature evaporants

Formamide is organic but only one-fourth carbon; it melts at **2°** and at 180° decomposes to mostly carbon monoxide and water. If overheated or catalyzed by acids it produces HCN. Formamide is miscible with water and so nontoxic that it's used as a cryoprotectant for vitrification, but it is also teratogenic.

Hyponitrous acid is an inorganic solid, if a dangerously unstable one, which spontaneously decomposes to nitrous acid and water over weeks **at room temperature**. I'm not clear on whether this decomposition takes place in solid form or not, or what its boiling point is.

Ammonium nitrite also slowly decomposes to water and nitrogen even at **room temperature**; perhaps it is stable at some lower temperature, but usually it is stabilized instead by an alkaline aqueous solution. However, it, too, is dangerously unstable under many circumstances.

Hydroxylamine itself, used as a photographic developer, might be a reasonable candidate: it's an inorganic solid, melting at **33°** and decomposing at 58°, but unstable in a poorly-understood way. Still, its flashpoint isn't until 129°, and its oral LD_{50} is around 400 mg/kg, which sounds considerably more innocent than most of the amines mentioned below.

Pyrosulfuric acid melts at **36°** but may not be a good idea, being strong enough to *protonate sulfuric acid*. Moreover, I'm pretty sure that when it decomposes from heating, it decomposes into SO_3 and sulfuric acid.

Dinitrogen pentoxide melts at **41°** and boils at 47°. It's a strong oxidizer and a strong acid, of course, but much less horrifying than the more common nitrogen oxides, which it will produce in ultraviolet light. Eventually it decomposes to nitrogen dioxide and oxygen even at room temperature, though.

Ammonium bicarbonate decomposes into ammonia, water, and carbon dioxide at **42°** and is widely thus as a leavening agent for cookies and crackers, as well as an acidity regulator, fertilizer, and fire extinguishing agent. Decomposition is already rapid at 36°. It's an irritant that can cause lung damage.

Caro's acid melts at **45°** but is probably contraindicated, being dangerously unstable itself and "one of the strongest oxidizers known".

NH_3OHNO_3 is another inorganic solid; it melts at **48°** and decomposes somewhere in the 200°–300° range, but it is also very toxic and, except in aqueous solution, dangerously unstable.

Cyanogen bromide, a common organic synthesis reagent, is an inorganic solid which cleaves peptide bonds and reacts with water to produce HCN and HOBr. It also has a tendency to produce cyanide while in storage. It melts around **50°** and then boils around 61°. Unlike almost every other material on this list, it contains no oxygen, but it does contain carbon.

Ammonium carbonate decomposes into ammonia and carbon dioxide at **58°** and is used like ammonia bicarbonate for leavening, often in a mixture, as well as an emetic and photographic lens cleaner.

Ammonium carbamate occurs with the carbonate and bicarbonate, with which it is used as leavening (and with which it spontaneously interconverts), and it decomposes at **60°**, also to ammonia and carbon dioxide.

Marshall's acid is similar to Caro's acid but instead decomposes without melting at **65°** and, I think, isn't itself dangerously unstable, as long as you keep it far away from any organics. I imagine that it decomposes into mostly SO_3, though.

Ammonium sulfite, used as a food additive, a photographic fixer, and a safer alternative to lye for straightening hair, also decomposes at **65°**, into "sulfur dioxide and oxides of nitrogen". It's also used to make blast-furnace refractory-lining bricks; US patent 2,724,887 from 1955 explains that in aqueous solution it works as a source of a sulfite ion which oxidizes to sulfate and somehow prevents iron-oxide contamination in the bricks from causing them to disintegrate under blast-furnace conditions. (Mysteriously, though, he forgot to patent this, patenting only the use of lithium chloride for the same purpose.)

Ammonium oxalate is organic but only about 20% carbon; it melts or possibly decomposes at **70°** and presumably decomposes at a higher temperature, below about 130°, I think. It's so nontoxic that it's found in kidney stones and is used as an anticoagulant for blood transfusions. However, the decomposition products include, at first, oxamide, and later hydrogen cyanide.

In Project Pluto a similar purpose in assembling a hot reactor was answered with naphthalene mothballs, which melts at **80°** and boils at 216°, requiring no oxygen. They also sublime pretty rapidly at room temperature, typically millimeters per month (or, in SI units, hundreds of picometers per second.) But mothballs are still organic.

NH_4ClO_3 decomposes at **102°** to nitrogen, chlorine, and oxygen, but is dangerously unstable; Wikipedia says, "Even solutions are known to be unstable ... it should only be kept in solution when needed, and never be allowed to crystallize." Ammonium chlorite and hypochlorite are even worse.

Ammonium acetate is organic but only about one-third carbon; it melts at **113°**, boils at 117°, and decomposes to liquid acetamide and water at 165°. Acetamide is acutely nontoxic but possibly carcinogenic, and doesn't decompose until 221°. Ammonium acetate is deliquescent and sufficiently nontoxic to be used as a diuretic, a

biodegradable de-icer, and a food additive for buffering pH. Crystallizing it from a water solution is difficult.

Just plain crystalline sulfur melts at **115°**. In air it will burn enthusiastically shortly thereafter (its flashpoint is 160°, its autoignition temperature 232°), but absent oxygen, it boils at 448°. It evaporates with surprising speed even at and below its melting point, though.

A unique property among the materials mentioned here is sulfur's metastable "solid" amorphous form, easily produced by quenching molten sulfur from above 170°, where it is blood-red; this amorphous polymeric form is red or brown and very plastic, like chewing gum ("a more or less sticky mass"), and can be remelted at 120°. At room temperature this material initially seems to show some surface tension, fingerprints in its surface disappearing over the course of several minutes, so it is really just a viscous liquid, but in a few hours to days it hardens and becomes glassy, though without changing color, a phenomenon attributed to semi-crystallization of this amorphous polymeric phase into "ω-sulfur" and the simultaneous partial crystallization of the non-polymeric impurity.

Some sulfur dioxide in the sulfur is apparently necessary for the formation of this amorphous phase, and is normally formed when solid sulfur is exposed to air.

This "quick-quenched" form of sulfur reportedly has S_8 rings dissolved in it, lowering its glass transition temperature to -30°; these can be removed by washing with CS_2. (Some twenty allotropes of solid and liquid sulfur are known, complicating this enormously; some of them are even metallic.)

You could recrystallize this amorphous form either by waiting long enough (apparently many years) or by annealing it, reconstituting the familiar brittle yellow α-sulfur. Reportedly above 90° the recrystallization becomes "rapid", which seems to mean "hours" or "minutes" rather than "seconds", and is associated with a volume loss of some 8%. This may be useful for, for example, using the sulfur as modeling clay, then recrystallizing it to its usual form. However, four hours at 100° does not seem to be enough time to have any noticeable effect, even when the dark "amorphous" sulfur is in contact with what seems to be yellow monoclinic α-sulfur. Even remelting to the low-viscosity liquid form doesn't seem to revert it fully from dark brown to bright yellow.

Ammonium formate is organic but only 20% carbon; it melts at **116°**, and decomposes into water and the nontoxic formamide at 180°, at which point the formamide decomposes into carbon monoxide and ammonia. Ammonium formate is deliquescent and relatively nontoxic. The above-mentioned considerations for formamide apply.

Hydroxylammonium sulfate is a stabler and less toxic salt of the hydroxylamine mentioned above; it's used in color film emulsions, decomposing at **120°** to SO_3, N_2O, NH_3, and water, in a reaction exothermic if heated past 138°. It irritates skin but won't even damage your eyes if you splash it in them; however, it's acidic and a strong reducing agent.

The inorganic solid ammonium persulfate, used in hair bleach and as a food additive, also decomposes at **120°**. However, though its toxicity is relatively low, it's a strong enough oxidizing agent to oxidize copper and nickel, and it's very acidic. Worse, I imagine that when you do manage to decompose it, it decomposes into ammonia and Caro's acid.

The fertilizer, flame retardant, and herbicide ammonium sulfamate melts at **131°** and decomposes at 160°, presumably to ammonia and sulfamic acid, an "intrinsically safe" household cleaning product which melts at 205° and then decomposes to nitrogen, water, and sulfur oxides.

The inorganic solid Hydroxylammonium chloride decomposes around **156°**, but I don't know what it decomposes into. I'm guessing that nitrogen oxides, probably hydroxylamine, nitric oxide, and hydrogen chloride would be in the mix; maybe nitroxyl, hyponitrous acid, and/or hydrogen, too.

NH_4NO_3 is another solid that can be entirely decomposed with moderate heating, sometimes in a dangerous chain reaction. It melts at **170°** and decomposes exothermically at 210°. This decomposition can produce relatively innocent nitrous oxide and water; even more innocent oxygen, nitrogen, and water; or deadly and caustic acids, ammonia, and acidic nitrogen oxides; depending on the conditions of decomposition. It, too, is entirely inorganic. An additional disadvantage, or advantage, is that it deliquesces above 59% humidity at 30°, or even lower humidities at higher temperatures.

Ammonium thiosulfate is a photographic fixer, fertilizer, defoliant, and nontoxic cyanide alternative for heap-leach mining of gold and silver. It presumably decomposes if you heat it up enough, but I don't know at what temperature; different sources suggest **180°** or 150°. It decomposes to sulfur oxides and ammonia, normally, though sometimes it can produce hydrogen sulfide, and even without heating, it can corrode even copper.

Sodium persulfate is an almost non-hygroscopic compound used as a hair bleach, a soil conditioner, an oxidizer for zinc, a pickling agent for copper, and a polymerization initiator which decomposes at **180°**. Presumably this yields sulfur oxides, probably SO_3, and possibly oxygen, and leaves behind a sodium oxide residue, which doesn't boil until 1950°. In some situations, for example in fired-clay pottery, this residue may be tolerable; most of the other anions mentioned here can be used with sodium similarly, but I will mostly focus on burnout without residue.

NH_4ClO_4 is an inorganic solid that decomposes around **200°** into HCl, nitrogen, oxygen, and water, but it's exothermic enough that this can be dangerous; its autoignition temperature is only 240°. It's also a fairly widely used oxidizer. Its acute toxicity is low, but its chronic toxicity is high, and of course the decomposition products are caustic.

Ammonium sulfate is another inorganic solid that can be entirely decomposed by heat at **235°–280°**, producing ammonia, nitrogen, sulfur dioxide, and water. It's pleasantly stable and nontoxic, being widely used as a fertilizer, a food additive, 30% of worldwide fine particulate pollution, and (refreshingly for this list) a flame retardant. It's almost alone among ammonium salts in emitting no significant ammonia at room temperature. Ammonium bisulfate is an intermediate product in the decomposition, melting at 147°, for better or worse.

Ammonium chloride, when heated to **338°**, decomposes into ammonia and hydrogen chloride. These are presumably quite caustic in the gas phase, but are known to recrystallize innocently on delicate fossils to reform ammonium chloride. And ammonium chloride is entirely inorganic, and could even be used if oxygen were scarce.

Phosphorus pentoxide melts at **340°** and then boils at either 360° or 423°? I don't understand this but I don't want it going on anywhere

near me. Phosphorus oxides are very complicated and hard to predict, and they tend to be hygroscopic.

Oxamide, used as a fertilizer and flame retardant, loses water at **350°** to form the highly toxic cyanogen. It's organic but only about one-third carbon. Different sources claim that it melts at 163°, at 300°, or not at all.

Nitramide? Probably too dangerous. Ammonium dinitramide? Probably too dangerous, and also how to make it is a secret.

acetamide? Urea? Melts at 134°, only 20% carbon.

Sulfamic acid

Iodine

Thiocyanates? Thiocyanides? Bifluoride? Cyanate? Fluoride? Hydrosulfide? Iodate? Iodide?

Thiourea?

Potential fusible solids

If melting like naphthalene rather than vaporizing is acceptable, then under some circumstances there are a variety of solids that can be easily melted.

Highly water-soluble and deliquescent solids

If water won't damage the thing you're trying to remove the support material from — not the case for clay bodies for pottery, of course, but plausibly the case under many other conditions —

Ammonium nitrate

Calcium chloride

Magnesium chloride, zinc chloride, ferric chloride, carnallite, potassium carbonate, potassium phosphate, ferric ammonium citrate, potassium hydroxide, sodium hydroxide...

phosphorus pentoxide?

Solids that can be removed by a reagent

Pretty much any solid can be removed by the appropriate reagent, but not always quickly, and the trick is to pick something that won't damage the thing you're trying to remove it from.

Topics

Micro material sorting

Kragen Javier Sitaker, 02020-09-12 (2 minutes)

Placer mining sorts grains of sand roughly by density, so it can separate out gold from the other minerals, which are mostly less dense by a factor of five or more. With somewhat more difficulty it can separate out other dense minerals like monazite, zircon, and even magnetite. A miner with a gold pan and some water might be able to pan the gold out of 200 mℓ of sand in some 30 seconds; if the mean sand grain is some 30 microns in diameter, this amounts to picking the dozen or two grains of gold out of several billion grains of sand, a rate of several hundred millions of grains per second. Impressive, for a human.

Various kinds of froth flotation, flocculation, and deflocculation of crushed rock work in a similar way to separate out massive numbers of grains by various physical properties.

Suppose that we instead use machine vision, X-ray diffraction, frequency-dependent complex electric permeability, and so forth to sort through individual grains. For example, nearly all granite has grains of zircon in it, precious for its use in the manufacture of the resilient refractory ceramic zirconia. If we break up the granite, perhaps we can use automated feedback systems to sort through at least a few thousand crystals per second per machine, needing only a few hundred thousand machines to compete with the 49er with his gold pan.

Of course, in 02020, the idea of constructing a few hundred thousand machines sounds absurd, but in fact I am typing this on a computer containing several tens of billions of transistors and capacitors, one which is already obsolete. So in fact such things are not impossible; MEMS simply has not focused on them thus far.

Topics

Sparse sinc

Kragen Javier Sitaker, 02020-09-17 (12 minutes)

It occurred to me that if you wanted to upsample a signal by a lot, it would be very convenient to have a sparse approximation of a sinc at the original sampling rate, sampled at the new sampling rate --- "sparse" in the sense that you don't need very many multiplies.

To be concrete, suppose you have a 1024-sample fragment that was captured at 50 Msps, and you would like to upsample it to 300 Msps, about 6144 output samples, depending on how you handle the ones at the ends. 1024 of these output samples are simply the original 1024 samples, but the others have relatively significant contributions from a lot of original samples: for a given output sample, about 15 input samples have contributions attenuated by less than 20dB, about 150 have contributions attenuated by less than 40dB, and almost all have contributions attenuated by less than 60dB. So if you want an output waveform whose error is 60dB or better below the signal --- and this may well be what you want if some parts of the signal are 60dB quieter than others --- then you will not reach your objective by doing a time-domain convolution with a windowed sinc, unless you're willing to do the whole 5242880-multiply-accumulate job.

Standard, well-known solutions

There are several standard answers to this.

One is to use Lanczos2 or Lanczos3 interpolation and call it good: the signal you produce has very significant errors if compared to a truly bandlimited signal, but it also has very good locality, which is sometimes preferable. The lanczos3 kernel has six periods of support, so in one dimension each output sample (of those that must be interpolated) is a weighted sum of six input samples, so you only need 30720 multiply-accumulates.

(One reason you might care a lot about locality is if you're doing this operation "online", i.e., before you have the whole signal. This inherently precludes getting anything close to the right answer, since the sinc impulse response extends very far into the past before the impulse.)

Another is to do the Fourier transform, pad with high-frequency zeroes to the larger size, and do the inverse Fourier transform. (In the more general case where the filter kernel isn't a sinc, you can do the filtering with one complex multiply per frequency in the frequency domain.) I think a 1024-point FFT using the radix-2 Cooley-Tukey algorithm requires (N/2) lg N complex multiply-accumulates, and a complex multiply-accumulate is four real multiply-accumulates, so this is 20 real multiply-accumulates per sample; a 6144-point FFT using the mixed-radix version of the algorithm I think requires 22 real multiply-accumulates per sample. So this ends up being 155 648 (real) multiply-accumulates, which is five times slower than the Lanczos3 approach, but unlike Lanczos3, gives the correct answer except for rounding errors.

(Rarely noted is that it's reasonable to do the Lanczos time-domain

interpolation with fixed-point or low-precision floating point, while the Fourier transform really needs floating point and usually double precision. Nowadays, with the availability of low-precision SIMD operations in commodity hardware, this is starting to become a tradeoff we can make in practice even without taping out our own chips.)

The third, and I think the most used in this sort of context, is to pad with high-frequency zeroes *in the time domain*, inserting in this case five zero samples in between each pair of the original samples, then low-pass filter *in the time domain* to interpolate. Sometimes this is more efficient if done in two or more stages, and there are a lot of options for how to do the time-domain filtering, which can give you arbitrarily small errors if you're willing to pay arbitrarily large computational costs and especially if you can use acausal filters.

The `filtfilt` function in Octave or SciPy applies a given IIR filter twice, once going forward in time and once going backward, to cancel out its phase shifts and incidentally double its stopband suppression (and passband ripple). So if we have a low-pass IIR filter in the usual form that gives us, say, 30 dB of stopband suppression, then using it with `filtfilt` would give us 60 dB of stopband suppression and no phase distortion. I don't know very much about signal processing but I think the following interaction with Octave means that a 16th-order elliptic filter would mostly do the job, though with some error in the top 1% of the Nyquist frequencies (the top 250 kHz of our 25 MHz Nyquist frequency):

```
>> ellipord(.99/6, 1.01/6, .000001, 30)
ans =  16
```

I think `[B, A] = ellip(16, .000001, 30, 1/6)` actually computes the filter, but I haven't tested it and don't have enough experience to be very confident.

So, if that's correct, then this gives us something very close to the right answer with 32 (real) multiply-accumulates per upsampled sample on each of the two passes, for a total of 64×6144 = 393 216 multiply-accumulates. This is comparable to the Fourier approach, but if we were slightly less enthusiastic about precision, we can get it to be even more comparable. For example, suppose our 50 Msps is actually sampling a signal that's bandlimited to 20 MHz, so maybe we don't really care what happens between 20 MHz and the Nyquist 25 MHz; then we could use only a 9th-order elliptic filter:

```
>> ellipord(.8/6, 1/6, 1e-6, 30)
ans =  9
```

And if we additionally can tolerate passband ripple of 0.001 dB, or 0.0005 dB per pass, we can get down to a 7th-order filter:

```
>> ellipord(.8/6, 1/6, .0005, 30)
ans =  7
```

So now we only need 28×6144 = 172032 multiply-accumulates.

Kooky sparse solution

So the sinc impulse response is, you know, one big double-wide hump in the middle, and then a bunch of wiggles, all of the same width and almost the same shape. What if you could factor those oscillations, or most of them, into linear combinations of a small number of basis functions? Like, maybe something that looks like a single hump between two zeroes separated by the sampling interval, maybe with some tails on it extending out into the neighboring sampling intervals. Like a Gabor wavelet, maybe. Or maybe just the near-sinusoidal hump. And then maybe one or two things that represent the biggest deviations in those wiggles, since some of them near the double-wide hump are kind of skewed to the left or the right, although the ones way out in the boondocks are fairly precisely sinusoidal.

You can make a sparse impulse train which, convolved with your single hump or Gabor or whatever, and added to the doublewide hump, gives you a pretty good approximation of the sinc. Convolving with an exponentially decaying alternating impulse train is easy enough (it's a feedback comb filter: $y(n) = x(n) - \alpha y(n-k)$) but in this case we want to convolve with an impulse train that decays subexponentially. Here are some possible solutions:

- Try to approximate 1/n as a sum of decaying exponentials. Unfortunately this has a sort of discontinuity or step function where each new exponential begins. If your initial exponential decay goes from amplitude 2/3 at 3/2 down to amplitude (-)2/5 at 5/2, then at 7/2, instead of 2/7, its amplitude will be 6/25, which is too low by a ratio of 21/25; we can add in a new exponentially-decaying pulse train there with an amplitude of 8/175, and then at 9/2, when we want (-)2/9 as our amplitude, the original exponential will have decayed down to (-)18/125, leaving a gap of (-)88/1125. This is actually *larger* than the previous gap of 8/175, so our second exponential decay can't fill the gap — it would need to be growing rather than shrinking! So we would need to pick some decay rate for it and start a third exponential, and so on. This doesn't seem like a promising avenue because it's going to be a long time before the errors are small enough that we can stop adding new exponentials on every cycle.
- Try to approximate 1/n as a sum of functions that eventually tail off into exponential decay but have a substantial subexponential plateau before that. For example, you can have a pipeline where your input signal goes into one stage of exponential ringdown, which is used to excite an output stage that does another exponential ringdown, perhaps a much slower one. The impulse response of this system should be a gradually growing alternating impulse train which asymptotes up to some maximum amplitude as by exponential decay, then dies away with essentially the exponential-decay behavior of the output stage. This two-stage pipeline avoids having a step function, but it still has a step-function discontinuity in the *derivative* of its envelope.
- A three-stage pipeline can have an envelope that looks like the integral of the envelope of the two-stage pipeline, with a smoothly feathered sigmoid startup, followed by the usual exponential decay

(which stabilizes the otherwise inherently unstable integrator). This pushes the discontinuity in the envelope out to the second derivative.

• Use one or two stages of feedforward combs to sort of flatten out and shape the plateau of a two- or three-stage pipeline.
• Way out in the far tails, use different Gabor basis functions that have many oscillations in their envelopes, not just a few; that way you can use impulse trains of much lower frequencies. I'm not sure this actually helps because the cost of convolving with an alternating impulse train using a feedback comb filter isn't dependent on the frequency of the impulse train.
• The Gabor basis function in particular has an existing efficient sparse approximation that I've written about previously in Dercuano.
• Presumably you need to use some sort of bidirectional filtering to get the left tail of the sinc impulse response as well as the right — though probably by computing the left filter and the right filter on the original signal, then adding them together, rather than composing them as in `filtfilt`. But maybe a more complicated topology of this kind of thing can help out in adding a smooth decay to the left side of these globs of alternating impulses. Like, if you do the two-stage pipeline thing both to the left and to the right, you could add them together with an offset in order to get rid of the steep slope of the initial "attack".

I don't know enough yet to know whether this will yield a more efficient and/or precise solution than the standard time-domain IIR bidirectional filtering approach, but it does seem plausible. In particular a great deal of the signal processing in this approach can be done at the lower frequency rather than the higher frequency.

Topics

An index of the 1880 edition of Cooley's Cyclopædia

Kragen Javier Sitaker, 02020-09-17 (updated 02020-10-23)
(9 minutes)

The 1880 sixth edition of Cooley's Cyclopædia is available in the Archive (vol. 1, vol. 2). However, the scans are 400 dpi and render rather slowly on my netbook, tens of seconds per page, or eight seconds in mupdf (which nevertheless doesn't prefetch or give any kind of "working" notice!) so I thought I would add a partial headword index to accelerate the process of finding things.

Without this index, a binary search for a headword would require looking at 11 or 12 pages distributed over both volumes, about a minute and a half of rendering time. With it, the worst case should be 6 pages, but the average more like 4; about 5% of pages are included directly in the index.

- Feeding bottle, p. 717 (vol. 1 733/916)
- Feeding bottle–Feet, p. 718 (vol. 1 734/916)
- Felting–Fermentation, p. 719 (vol. 1 735/916)
- Fermentation (cont.), p. 720 (vol. 1 736/916)
- Fern–Fever, p. 722 (vol. 1 738/916)
- Finings–Fire, p. 734 (vol. 1 750/916)
- Gas, p. 784 (vol. 1 800/916)
- Gems, p. 790 (vol. 1 806/916)
- Gin, p. 796 (vol. 1 812/916)
- Gluten–Glycerin, p. 809 (vol. 1 825/916)
- Green pigments (cont.), p. 821 (vol. 1 837/916)
- Gum (cont.), p. 829 (vol. 1 845/916)
- Gunpowder (cont.), p. 831 (vol. 1 847/916)
- Gunjah–Gut, p. 834 (vol. 1 850/916)
- Hederin–Hemp, p. 842 (vol. 1 858/916)
- Hesperidin–Hollands p. 844 (vol. 1 860/916)
- Hydrocyanic acid (cont.), p. 854 (vol. 1 870/916)
- Hydrogen, p. 856 (vol. 1 872/916)
- Hyocholic acid–Hysterics, p. 859 (vol. 1 875/916)
- Ink, p. 896 (vol. 1 912/916, last)
- Lac dye–Lacquer, p. 938 (vol. 2 50/916)
- Lamp (cont.), p. 944 (vol. 2 56/916)
- Lamp black–Lanthopine, p. 946 (vol. 2 58/916)
- Lapis–Larch bark, p. 947 (vol. 2 59/916)
- Lead (cont.), p. 950 (vol. 2 62/916)
- Light, electric (cont.), p. 963 (vol. 2 75/916)
- Liqueur de la motte–Liquor, p. 988 (vol. 2 100/928)
- Lithofracteur–Lithography, p. 994 (vol. 2 106/928)
- Liver and bacon–Lodging-houses, p. 996 (vol. 2 108/928)
- Lotion (cont.), p. 1001 (vol. 2 112/928)
- Lubricating compounds–Luncheons, p. 1013 (vol. 2 125/928)
- Meat, p. 1038 (vol. 2 150/928)
- Meconin–Medicines for passenger ships, p. 1050 (vol. 2 162/928)
- Meerschaum–Mercurial disease, p. 1052 (vol. 2 164/928)
- Mercury, p. 1053 (vol. 2 165/928)
- Mercury (cont.), p. 1056 (vol. 2 168/928)
- Meslin–Metals, p. 1063 (vol. 2 175/928)
- Mortification–Moulds, p. 1088 (vol. 2 200/928)
- Neuralgia (cont.), p. 1100 (vol. 2 212/928)
- Nitric acid (cont.), p. 1106 (vol. 2 218/928)
- Nitro-glycerin, p. 1109 (vol. 2 221/928)
- Norfolk fluid–Notices, p. 1112 (vol. 2 224/928)
- Novargent–Nuisance, p. 1113 (vol. 2 225/928)
- Oils (mineral) (cont.), p. 1138 (vol. 2 250/928)
- Olein–Oleometer, p. 1188 (vol. 2 300/928)
- Paranaphthalin–Parasites, p. 1238 (vol. 2 350/928)
- Pepper (cont.), p. 1258 (vol. 2 370/928)
- Percentage–Percolation, p. 1260 (vol. 2 372/928)
- Percolation (cont.), p. 1261 (vol. 2 373/928)
- Percussion–Periodic acid, p. 1262 (vol. 2 374/928)
- Peristaltic persuaders–Petroleum, p. 1263 (vol. 2 375/928)
- Phylloxera vastatrix, p. 1288 (vol. 2 400/928)
- Picoline–Pies, p. 1293 (vol. 2 405/928)
- Pills (cont.), pp. 1298–1314 (vol. 2 410–26/928)

Topics

Spark gap logic

Kragen Javier Sitaker, 02020-09-20 (updated 02020-12-16) (25 minutes)

Thinking about Marx generators last night, I realized that their traditional elements — two-electrode air spark gaps, capacitors, resistors, and a high-voltage, low-current DC power supply — are probably sufficient to implement universal sequential digital logic, It may be limited to a few kilohertz, but air-gap flashbulbs can produce microsecond-level discharges when cooled by a quartz heatsink, so faster speeds might be achievable.

A Marx generator is a simple sort of RC relaxation oscillator used as a source of high-voltage pulses when efficiency is not important. A series of capacitors in series, separated by similar-sized spark gaps, are charged through a resistive network connecting their anodes and another connecting their cathodes, which, as long as little current flows, respectively maintains the anodes at similar voltages and maintains the cathodes at similar voltages, so the voltage across each spark gap is nearly the negative of that across each condenser; once this voltage rises to a high enough level, the air in the gap experiences avalanche breakdown with a large current and effectively connects two capacitors in series, immediately overwhelming the breakdown of the adjacent spark gaps. This very rapidly produces a chain reaction and a very high voltage, which, as I understand it, then discharges on a timescale primarily limited by the self-induction of the elements of the system, commonly nanoseconds to microseconds.

(Ordinary electrostatic discharges, like from walking across the carpet and touching a doorknob, have rise times in the range of a nanosecond or so, so the rise time will not be the limiting factor in performance; the recovery time will.)

Without the voltage gain, you can build such a relaxation oscillator with a single RC timing circuit with a spark gap in parallel with the capacitor. Paschen's curve has a minimum at one atmosphere at 327 V and 7.5 μm, tailing off to linear growth at 3.4 MV/m, so, for manual work, it might be expedient to work with some 4 kV and gaps of 1 mm. You can easily adjust the period of this oscillator by changing the RC time constant, although the arc ignites at somewhat imprecise times due in large part to the irregular availability of free so-called "seed electrons" at the cathode, provided by photoelectrons or background ionizing light and other particles.

(Common neon-sign transformers provide 2–15 kV RMS at 18–30 mA, according to Wikipedia, a much less frequently lethal current than the typical 500 mA of a microwave oven transformer.)

In particular, it's straightforward to make a triad consisting of a "primary oscillator" running at, say, 500 Hz; a "reference oscillator" at half that frequency, say 250 Hz; and a "bit oscillator" also running at 250 Hz. (At 1 mm the arc should ignite around 3.4 kV; with a 4 kV power supply, that should take about 1.9RC, and the remaining voltage after the arc extinguishes should be small; so, to get 250 Hz, a 4-ms period, we could use an 10MΩ resistor and a 210 pF capacitor, a

1MΩ resistor and a 2100 pF capacitor, and so on, although the resistors will start to get rather hot at lower resistances.)

In isolation these three oscillators will tend to drift relative to one another, but I think this can be remedied. If we use another capacitor to couple the voltage spike from the triggering of the primary oscillator into the reference oscillator and the bit oscillator, we can advance the timing of the reference oscillator and the bit oscillator so that they run at exactly half the frequency of the primary oscillator, if they were running slower. The voltage spike early in the charging process won't be enough by itself to fire the spark gap, but when the capacitor voltage is already nearly high enough to strike a spark, it will easily overwhelm the dielectric strength of the air in the gap.

Now the phase relationship between the reference oscillator and the bit oscillator is quantized at either 0° or 180°, so the phase of the bit oscillator stores a single bit of data.

For reliable storage, it is essential that the free-running frequency of the reference and bit oscillators be slower than or equal to that of half the primary oscillator; otherwise, they may spontaneously fire early, resulting in phase drift and eventually a bit error. Various expedients are available: the use of a slightly larger resistance or capacitance, of course, but also a slightly larger spark gap that will *never* fire without the excess stimulation provided by the primary oscillator; or a resistive network that charges the capacitor up to only a fraction of the power supply.

It is worth mentioning at this point that a cascade of two or three low-pass RC filters before the spark gap, rather than a single one, can provide a more desirable voltage waveform at the spark gap — one that remains lower for a longer fraction of the cycle, thus widening the voltage safety margin against early firing.

Now that we have a reliable way of storing a bit, we have the problem of constructing digital processes that evolve in time rather than merely remaining stable, by coupling two or more bit-storage devices. And in particular we want to be sure we can achieve chaos or instability, known in the world of digital logic as "fanout" or "amplification". The example of the Marx generator shows that this is definitely achievable.

One easy way to achieve amplification, though stepping outside the framework of the Marx-generator parts mentioned above, is step-up transformers.

Air-gap flashes pass a lot of charge between the main electrodes at, typically, 20 kV, to produce the bright flash, triggering this with a quartz-insulated "ignition tube" electrode at a much higher voltage like 70 kV, but a much smaller amount of charge due to a lower capacitance. The higher voltage provides initial ionization in the gap, which triggers the discharge of the lower-voltage but higher-energy arc.

This provides energy gain, but not voltage gain — a high voltage is used to switch a lower voltage. A direct way to attack this problem is by using a pulse transformer to step up voltages, so that a spark at a relatively low voltage can produce a lower-current pulse at a much higher voltage, used to trigger other gaps.

But, as we have seen above, more indirect routes are also available.

For example, if a 3400V spark gap has been charged up to 3000V, then a 700V impulse will trigger it to conduct, discharging the 3000V down to perhaps 50V or 150V. This impulse can be coupled in via a small capacitor, requiring a correspondingly small amount of charge and energy. This can be facilitated by a small resistance between the spark gap and the 3000V-charged capacitor — in this way a 700V impulse coupled in thru a coupling capacitor need not charge the 3000V capacitor, only the capacitance of the spark gap itself, which will typically be in the picofarads. If we suppose the spark gap is 10pF and the pulse has a rise time of 1μs, a few hundred kilohms would suffice, a number not normally considered a "small resistance", but it's one or two orders of magnitude smaller than the other resistances discussed above.

Another more indirect route is used by the Marx generator itself: rather than using a *small* coupling capacitor to trigger the discharge of a larger storage capacitor, we can couple the triggering pulse through the large storage capacitor itself, suddenly increasing the voltage on its other end by a factor of up to 2.

A third possibility involves resistive networks. By charging a small capacitor (to ground) through a resistive network from two or more other capacitors (to ground), its voltage can be brought rapidly to a weighted average of their voltages, and if a spark gap is in parallel with it, it will either discharge repeatedly through the spark gap, or not, according to whether its terminal voltage is greater or less than the spark gap's breakdown voltage. The pulses thus generated can be used to trigger other, higher-energy spark gaps (and their time of occurrence can be synchronized with the primary oscillator by coupling in a little of the primary oscillator to them), or the resulting larger current draw on the "input" capacitors can be sensed.

A particularly interesting possibility here is the use of such a threshold device as a phase detector. Given reasonable waveforms on two bit oscillators, their instantaneous sum will rise above some threshold for a while each cycle if they are in phase, but for a high enough threshold, not at all if they are out of phase. This provides us with the operation XNOR; combined with negation and access to the reference oscillator, we can construct in some sense all boolean functions.

This same sort of threshold device may be of particular interest as a display pixel, glowing or not according to whether its voltage reaches the threshold and thus provokes repeated discharges. Low-pressure plasmas, xenon, and mercury vapor coupled with phosphors may be useful in boosting visible light output for a given power level.

A fourth possibility is to *interrupt* a continuing arc, ballasted for example by a resistor or the self-inductance of the wires, with a voltage pulse that temporarily robs the spark gap of the tens of volts necessary to continue conduction. This, however, seems much more precarious and sensitive.

Encoding bits directly as voltage levels rather than oscillator phases seems like it would be more challenging, if feasible at all, unless you are spending the enormous amount of energy required to keep a spark gap in a state of continuous arcing, or find a way to employ glow or

corona discharge like an old Dekatron, which I suspect would cost significant speed. So I suspect the phase-encoding approach, although less simple, is probably more practical.

The above falls short of being a fully worked out design for digital sequential logic using spark gaps, resistors, and capacitors, but I think it amounts to a convincing argument that it's practically doable, though only over a fairly narrow voltage range in the normal atmosphere (400–4000V); it might be easier to debug something closer to the middle of that range, like 1200V, than designs near the limits, like the 4000V I was considering above. Reducing the pressure or substituting a friendlier gas like argon (127V at 10μm) might help.

Sources of deviation from designed behavior include:

• Resistors changing resistance as they heat up and as voltages change: this is particularly a problem for old carbon-composition resistors, although modern high-precision metal-film resistors are not completely immune.
• Spark gap size variation between devices: this may be particularly a problem at lower voltages and lower gap sizes.
• Spark gap wear: the spark gaps will increase in size and decrease in smoothness over time as the frequent electrical discharges vaporize parts of the electrodes; moreover, some materials may form insulating oxide layers, increasing the breakdown voltage significantly. This can be minimized by reducing the operating frequency; by using higher radices rather than binary, with the reference oscillator running at ⅓, ¼, or less of the primary oscillator frequency; by using electrodes of graphite, copper, or tungsten-copper such as Elkonite; or possibly by using a dielectric coolant liquid rather than gas, or by running cooling water through the electrodes.
• Spark initiation delay: avalanche discharge is triggered by a seed electron, which breaks free at a random time sometime after the avalanche threshold is passed. In neon-tube logic of the 1960s and 1970s, this problem was often remedied by adding radioisotopes to the electrodes or by keeping them brightly illuminated with visible or ultraviolet light, while in vacuum tubes it was remedied by heating the cathode with a resistive filament and also coating it with a low-work-function material such as an alkali-metal oxide. Other possibilities include using small gaps so that field emission is more important; using sharp points, perhaps even carbon fibers as used in modern ozone generators, to provide corona discharge in advance of the avalanche discharge; using larger electrodes, perhaps with anodes full of holes to permit light to illuminate the cathode; and suffusing the whole machine with slightly ionized gas or ultraviolet light. The use of graphite might worsen this problem because its higher (≈4.6 eV) work function reduces field emission and the emission of photoelectrons.

Another crucial question about such devices is to what degree they can be miniaturized and sped up. Near-kilohertz discharge rates should be straightforwardly achievable, but neon-tube logic topped out around a kilohertz due to the relaxation time required for the gas to deionize.

Intuitively I would expect higher-ionization-energy gases to recover faster — this is why air-gap flashes use air (primarily nitrogen)

instead of the more efficient xenon, because it gives submicrosecond recovery times, and N_2 (1503 kJ/mol = 15.58 electron volts) is close to optimal here, though, e.g., hydrogen (1488 kJ/mol = 15.43 electron volts) is close. SF_6 (≈15.8 eV) may also be worth considering. Perhaps also higher pressures accelerate the recovery time, accounting for the difference between the millisecond recovery time of an ordinary low-pressure xenon camera flash and the 10μs cited for xenon in Wikipedia's air-gap flash article. Electric-discharge machining routinely uses hundreds of thousands of sparks per second in, typically, deionized water, which is pumped through the spark gap at a high flow rate; typically this uses several hundred volts to initiate the spark and an average of tens of volts and several amps during cutting.

Both higher pressures and higher ionization energies would tend to promote miniaturization.

So, because both air-gap flashes and EDM routinely have submicrosecond recovery times, I think submicrosecond recovery times are probably feasible, putting this kind of logic in the performance range of 1960s CD4000-type CMOS.

Nanosecond-scale recovery times would probably be more challenging and would probably require mechanical removal of the ionized dielectric; for example, if the spark gap is 10 μm wide, it the electrodes can very reasonably be 10-μm-diameter rods with holes in their center for coolant flushing. For the coolant to travel the 5 μm required to clear the still-ionized material from the interelectrode gap in, say, 100 ns, it needs to be traveling at 50 m/s, Mach 0.15, and slightly faster passing through the hole, which is probably achievable; waterjet cutting machines achieve many times that speed through holes in that size range, and the use of a gas would decrease the viscosity far below what a waterjet must withstand. This amounts to a volume flow rate of under 10 microliters per second, again, plainly within the bounds of feasibility. We can conclude that active dielectric flushing is a practical way to increase spark-gap logic operational frequencies well into the megahertz, though probably not to 100 MHz.

What about energy usage? If each spark gap has a capacitance of 10 pF and discharges at only 500 V, it contains 1.25 μJ in its electrical field at discharge time, which will be almost entirely dissipated by the spark; at 1 MHz discharge rates this amounts to 1.25 W. We might thus suspect that active coolant flushing of some sort is necessary to prevent the electrodes from entirely vaporizing, quite aside from its potential utility for accelerating recovery times.

However, the minimal capacitance of a spark gap of 10 μm diameter and 10 μm spacing can be approximated with the infinite-plate formula $C = \varepsilon A/d$, which gives some 70 attofarads in this case, five orders of magnitude smaller, so in fact the spark-gap capacitance will not be the limiting factor in such cases, even using a high-permittivity dielectric like water.

This in turn suggests an electrical energy cost on the order of 10 picojoules per bit operation, comparable to modern CMOS, although of course that doesn't account for the energy cost of pumping all that dielectric through the gap. Also, such low energy costs per operation

probably require much higher operational frequencies — for RC = τ = 2.1 ms as above, you'd need a 30-teraohm resistor, which would normally be called an "insulator". So 1 nJ is probably achievable but 10 pJ may not be.

Conductive-mesh spark-gap electrodes may be a more effective way to deliver light, dielectric, and coolant to the spark gap, permitting as they do the use of spark gaps with electrodes much larger than their interelectrode spacing without diminishing the fluid flow.

Over timescales not too long compared to the relaxation (deionization) time of a dielectric, it may be feasible to use a flowing fluid as a delay-line memory. An input spark gap in the center of a tube of, for example, rapidly flowing atmospheric-pressure xenon, produces a series of plasma blobs which are rapidly carried downwind, still glowing; some 10 μs later, an output spark gap also in the center of this tube detects their presence by virtue of the discharges they ignite at well below its usual breakdown voltage. A partial vacuum behind a hole in one of the electrodes of the output spark gap sucks these blobs into the interelectrode gap. To minimize the time-domain degradation of the memory waveform, this entire stream of memory plasma is kept well away from the walls of the tube by the xenon flowing around it, which ideally would be in inviscid flow so that even the curl of the flow field remains close to zero. Operated at 10 MHz such a tube could store at least 50 Manchester-encoded bits; gases that relax more slowly than atmospheric-pressure xenon could afford larger capacities and less-demanding operational frequencies.

Somewhat surprisingly for a digital-logic device that can plainly be constructed by hand from Victorian-era materials, the above dimensional figures strongly suggest that microscopic realizations of this family of devices might be not only feasible but even practical, particularly with higher pressures and modern insulators like teflon (as opposed to sealing-wax and gutta percha). With adequate plumbing, they might be capable of speed rivaling modern solid-state electronics, or at least 1980s solid-state electronics. Spark gaps of under a micron should be effective at a megapascal or so of gas, or perhaps at atmospheric pressure with liquid dielectrics.

Another amusing application of such relaxation oscillators might be as microphones: below the Paschen minimum, even a very small change in the electrode spacing should produce a very large change in the breakdown voltage of the spark gap and consequently both the frequency and the breakdown voltage of a free-running RC oscillator. (I'm not sure if it also increases the jitter.) Above the Paschen minimum, it should produce a smaller, nearly linear, but still fairly reliable change in these variables.

The oscillation period also depends, of course, on the resistance and capacitance, and in many applications it may be more practical to modulate the capacitance rather than the spark-gap size. 3 mm of ordinary glass ought to provide about 30 kV of dielectric strength, or fifty times that if fused silica instead; the 2 cm^2 of a finger touch, coupled with the relative permittivity of around 5 for glasses, gives a capacitance of about 3 pF, which may be a detectable touch. Lower

voltages and thinner materials may be a more practical way to detect human touch, or simply mechanical deformation of air-dielectric capacitors through levers.

DTIC document 633669 from 1991, "Hydrogen spark gap for high repetition rates", reports 10-µs recovery times to 17% for a 1.4 MPa 2.5-mm "unblown" hydrogen spark gap and 100-µs to 42%, about an order of magnitude faster than air; this is attributed to hydrogen's high thermal diffusivity. That is, by "undervolting" the gap to 17% of its normal breakdown voltage (some 120kV), they can trigger discharges at 100-kHz rates, or 10 kHz at 42% of its normal breakdown voltage. Recovery to 90% takes about 1 ms for hydrogen and 10 ms for air, "dominated by the cooling time of the hot channel" rather than its deionization. It also points out that narrower gaps permit closer gas contact to metal surfaces, thus cooling the gas more rapidly, as well as lower inductance and resistance, and that the gas requires some time to recover its density after being rarefied by thermal expansion. They were working with three-electrode trigatron-type devices and report that "the recovery time varied little from millijoules to kilojoules of transferred energy", though it would be unsurprising if the picojoules I contemplate above did result in significant variation. (Hopefully the smaller energies would also result in longer electrode life than the hundreds of shots at which they reported substantial electrode wear.)

Above I haven't considered inductance, but of course at high enough speeds at a given length scale, inductive impedance will dominate resistance. Decreasing the length scale also helps with this.

The related DTIC document A636361, "A laser-triggered mini-Marx for low-jitter, high-voltage applications", describes an interesting way of triggering spark gaps with ±700-ps 2σ jitter — by using ultraviolet light to ionize SF_6 in the spark gap (in this case by a frequency-quadrupled Q-switched Nd:YAG laser) it is possible to ignite a plasma in a precharged spark gap, which then activates a Marx generator with rise times in the range of 2 ns. There are a variety of ways that such spark gaps can detect light, ranging from the reduced jitter from photoelectric seed electrons to this sort of ionization-induced ignition-voltage reduction, and of course a traditional Geiger counter is nothing more than a spark gap arranged to detect ionizing light and other particles.

A low-voltage way to try out some of these ideas is to replace the spark gaps with transistors, or perhaps diodes, in reverse avalanche mode, as in Look Mum No Computer's Super Simple Oscillator, which uses two unspecified terminals of a 2N3904 in parallel with a 10µF capacitor. Another, better-explained variant of the design uses a 2N4401 with the emitter toward Vcc and the collector toward ground, in series with an LED and in parallel with a 3300µF (!) capacitor. (Thank you very much to Hideki and splud on ##electronics for linking me!)

Folklore says red LEDs suffer reverse avalanche discharge around 5V and often survive it, so they might be an alternative to the transistor or spark gap. Their lifetime might be limited in this application, or it might not. Probably something like a 1N4001, or any ordinary rectifier or small-signal diode, would have an

inconveniently high breakdown voltage, which increases the chance of damage to the diode, as well as power consumption and electric shock risk.

In either case you're depending on properties of the components that are not specified by the manufacturers because they're irrelevant to their usual uses, so consistent results from component to component may be hard to obtain.

Topics

- Contrivances (p. 790) (44 notes)
- Electronics (p. 792) (42 notes)
- Physics (p. 796) (18 notes)
- Physical computation (p. 826) (7 notes)
- Spark gaps
- Oscillators
- Marx generators

Copper salts

Kragen Javier Sitaker, 02020-09-21 (updated 02020-09-23)
(8 minutes)

For Hot fabrication (p. 320) I was thinking it might be useful to be able to smelt copper into a mold with a thermite reaction. But for that you need cuprite, cuprous oxide, which I do not know where to buy.

I think you'd probably have to synthesize it.

Copper has many salts, but more cupric salts than cuprous salts, and its cupric salts are more often water-soluble.

Red **cuprous oxide** (cuprite) is insoluble in water, though not in acids, and "degrades to [cupric oxide] in moist air", while black **cupric oxide** (tenorite) is also insoluble, though, again, not in acids — or alkalis, with which it yields cuprate salts; and there is additionally a hypothetical trivalent copper oxide which would be a strong oxidizer. White **cuprous sulfate** "decomposes rapidly in presence of moisture" to copper and the highly soluble skin-staining **blue vitriol**. "Little evidence exists for" yellow **cuprous hydroxide**, which is "extremely easily oxidized" to insoluble **cupric hydroxide**. **Cuprous nitrate** is almost unknown; blue-green **cupric nitrate** is a common water-soluble bulk chemical, easily synthesized from copper and silver nitrate. **Cuprous iodide** is precipitated from cupric ions and potassium iodide, releasing iodine; there is no cupric iodide, or at any rate only a very unstable one. Rare blue **cupric fluoride** is "highly soluble in water"; there is no cuprous. Dark green water-soluble **cupric acetate**, a component of verdigris and used to make Paris green, can reportedly be heated with copper to produce white **cuprous acetate**. Basic **cupric carbonate hydroxide**, another component of verdigris, is the insoluble and hard malachite and azurite, depending on the amount of hydroxylation; the neutral gray cupric carbonate readily hydroxylates given half a chance, or decays to cupric oxide at low CO_2 concentrations; there is no cuprous. Blue-green **cupric phosphate** and phosphate hydroxide are the insoluble rare mineral libethenite, or more commonly pseudomalachite, depending on hydration and hydroxylation — or, with aluminum, turquoise; again there is no cuprous.

So, excluding exotics, there seem to be no stable water-soluble cuprous salts except (to an almost undetectable degree) the chloride, iodide, and bromide.

(Incidentally, the above suggests that copper ions offer a way to separate iodide and, if cupric, elemental iodine from a mixture of chlorides and iodides, though I'm not sure it would work as well for sodium iodide.)

WP suggests synthesizing cuprous oxide by the following routes: directly oxidizing copper by heating it in air (but then how do you separate the two oxides?); reducing cupric solutions to cuprous with sulfur dioxide (maybe sodium thiosulfate might also work; the ammonia in ammonium thiosulfate would probably form a tetraammine-copper(II) complex, which which would be pretty but

might interfere); reacting cuprous chloride with bases to precipitate the cuprous hydroxide; or reducing alkaline cupric solutions with reducing sugars in Fehling's or Benedict's tests, where the copper is complexed with potassium sodium tartrate or sodium citrate, respectively, to prevent precipitation of cupric carbonate. (Fehling's test is mentioned in Cooley's 1880 Cyclopædia, for example under "Urine", along with Trommer's test, under "Sugar", which mixes blue vitriol with sugar and "an excess of hydrate of potassium [KOH]", then heats it to precipitate what sounds like cuprous oxide. The advantage of Fehling's test is reputedly that it was more selective toward sugars and easier to prepare. Benedict's test wasn't introduced until 1907.)

Benedict's test sounds relatively easy, but it consumes reducing sugars, such as glucose, fructose, or lactose — but not sucrose! Though you can hydrolyze sucrose with HCl — or, apparently, by boiling it with citric acid.

I think you can also, crossing the streams, use a cupric solution to oxidize copper metal, reducing the cupric ions to cuprous ions, and simultaneously producing more cuprous ions from the metallic copper; this is reported to work with cupric chloride, for example. The etchant is maintained at an acidic pH with HCl so that the cupric chloride remains soluble, and the temperature is ideally maintained at 50° to accelerate the etching process, though Adam Seychell explains that this is not necessary; he prefers 30° to reduce the HCl fumes.

Electrolytically pure copper metal is readily available, if expensive, and easily oxidized electrolytically to the chlorides, probably with a little contamination from the iodide which is insignificant in this context. Somewhat less pure blue vitriol is available for US$5–$7/kg as a fungicide and alguicide for swimming pools, though it, too, is of course cupric rather than cuprous. At about 250 g/mol (pentahydrate) it's about 25% copper (63.546 g/mol) by weight.

The thermite reaction with cupric oxide is hazardously rapid, so cupric contamination is to be avoided here; washing the nearly-insoluble white cuprous chloride thoroughly with water should suffice to purify it, but after purification the resulting cupric chloride must be kept dry to prevent disproportionation from recontaminating the mix. Similarly, cuprous oxide oxidizes to cupric oxide in moist air, which is dangerous in this context.

Insoluble tribasic copper chloride (atacamite, another component of verdigris) is manufactured in bulk as a nutritional supplement (though not for humans — a substitute for blue vitriol for livestock) and a fungicide.

The electrochemistry chapter of Simon Quellen Field's scitoys suggests making cuprous and cupric oxide by heating copper in air to red heat for half an hour, then flaking the black cupric oxide off (by allowing it to cool) to reveal the cupric oxide underneath. I suspect this is more practical as a way of making cupric-oxide semiconductor thin films than as a way of obtaining bulk cupric oxide. (The author reported 50 μA at 0.25 V from their 0.01 m² cupric-oxide solar cell, using salt water as the other electrode.)

There's also "red plague": red cuprous oxide forming on the surface of copper over months in humid environments due to galvanic

corrosion with silver plating.

Of all the options for producing cuprous oxide, I think the most practical is probably to generate soluble cupric chloride by electrolysis of aqueous sodium chloride by copper electrodes, followed by reduction to cuprous chloride with copper and HCl, then precipitation as cuprous oxide with ammonia, yielding also ammonium chloride — although perhaps a different base like sodium carbonate or bicarbonate, or calcium hydroxide, is required to prevent the copper from complexing with ammonia. It would be super cool if acetic acid were strong enough to allow the reduction and etching of copper to proceed; Cooley's 1880 Cyclopædia reports that dibasic acetate of copper is "prepared on a large scale in France by exposing copper to the air in contact with fermenting wine-lees", so I suspect that vinegar may be sufficiently strong to etch the copper. Where I am most uncertain is as to whether it is sufficiently strong to maintain cuprous chloride in solution. The Cyclopædia claims that heating cupric chloride reduces it to cuprous chloride "at a high temperature", which Wikipedia suggests is 993°, well above its 498° melting point, and a bit too high for this kitchen; WP claims cuprous chloride is stable to 1490°, though melting at 423°.

Distillation of ammonia by absorption into pure water may be feasible over days at room temperature; some random YouTuber reported successful deposition of a great deal of verdigris by putting their vinegar-and-salt-painted copper in a sealed box with a dish of ammonia nearby. Another video experiment report by the impressive mopatin claims success at *reducing* copper oxide, or perhaps the sulfides, or perhaps dissolving them, with a mix of vinegar and salt. Others report success at making copper acetate from copper metal, vinegar, and low-concentration H_2O_2, which last they claim is not necessary.

Topics

Hot fabrication

Kragen Javier Sitaker, 02020-09-21 (updated 02020-09-23)
(16 minutes)

Thinking a bit about thermite, it occurred to me that, for sand casting or investment casting of metal objects on the scale of centimeters, it might be best to generate the metal object right on top of the mold, thus avoiding the necessity to open a hot furnace, carry a red-hot or white-hot crucible, and pour the crucible. Copper in particularly is notoriously tricky to cast in this way.

Microwave oven kilns

The first version of this process I saw was back in the 1990s with microwave-oven casting: some guy whose name I forget stuccoed his clay lost-wax mold with magnetite and graphite as susceptors, taped over his microwave-oven fan, and microwaved the assembly until it was hot enough to melt metal. A more recent incarnation of a similar idea used a microwave-oven-sized tiny porous-insulating-refractory kiln with charcoal inside of it to calcine magnesia to make a magnesium-oxychloride knife from mostly seawater; the refractory is transparent to microwaves, and avoids the necessity to cover up the ventilation fan, but the charcoal picks up the microwaves. The stucco guy explained that magnetite works better than graphite, but only up to its Curie temperature, at which point graphite starts to work better because of its higher resistance; presumably less pure forms of carbon such as charcoal have higher resistivities and therefore work down to lower temperatures as well.

I've also had success igniting arcs between pieces of charcoal and steel wool inside a microwave oven, and such arcs would also work to heat up the interior of an insulated ceramic space like that. (I also left a glass of water elsewhere in the microwave to limit the risk of magnetron overheating in case my susceptors proved too reflective.) Peepholes in an optically-opaque insulating refractory kiln, whether for use in a microwave oven or not, might permit the pyrometric inspection of the blackbody interior when the microwave and thus the arc is turned off. (I was doing the arcs out in the open on top of a bed of granulated salt, because I had no sand; the molten salt globs were easier to remove from the granulated salt than the molten glass globs were from the glass floor of my microwave oven.)

(Silicon carbide is another susceptor with a wider temperature range than either magnetite or graphite.)

Thermite

If you can ignite thermite (whether by arcs, Joule heating, or by any other means), in a sand funnel scooped out of the top of a metal-casting mold, then you can presumably fill the mold with the liquid metal produced, whether that is iron from an aluminum/hematite thermite or copper from an aluminum/cuprite thermite. Moreover, if the reaction chamber is sufficiently refractory for the reaction not to melt through its bottom, it can be used to heat

an insulated reaction chamber rapidly to 2500° to 2800°, a temperature that can be calibrated by the thermite's stoichiometry rather than regulated by feedback, and which may be useful for other reactions that are difficult to perform at more convenient temperatures, such as the graphitization of amorphous carbon foam, carbothermic reduction of metals with refractory oxides, and so on.

(This process is routinely used with graphite crucibles or sand molds for welding copper conductors and railway tracks.)

Here in Argentina, on Mercado Libre, hematite ("pigmento oxido de hierro rojo") costs US$4 to US$6 per kg, while magnetite is in around the same range. I can't find cuprite; see the note on copper salts (p. 317) for more. Cupric oxide (tenorite) is much easier to prepare in bulk, but the resulting thermite acts with deadly rapidity; you might be able to reduce this menace by diluting the thermite mix with less-active hematite, excess aluminum, copper filings, iron filings, silica sand, or even — at the risk of producing hot caustic gas — blue vitriol. A small amount of excess aluminum and perhaps hematite or iron should produce an aluminum bronze rather than copper; $CuAl_{10}Fe_3$ is 8.5%–11% aluminum and 2%–4% iron, the remainder being copper. Aluminum bronzes are lighter, stronger, more corrosion-resistant, and lower-melting than copper.

Any excess aluminum would necessarily require that the reaction vessel be carefully purged to eliminate gaseous oxygen to reduce the risk of an aluminum fire. At ordinary temperatures carbon dioxide is not sufficiently inert to escape this danger, although paradoxically above about 2300° at atmospheric pressure the equilibrium goes the other way, and carbothermic reduction of aluminum becomes possible.

In any case, mixing enough of the desired end product into the thermite ought to tame it adequately, although thermite is plagued with hazards stemming from surprising interactions accelerating its action.

Sulfur thermites

Sulfur is another possible oxidant for powdered metals, particularly in a sealed pressure vessel. This poses the risk of the sulfur boiling if the reaction is not fast enough and pressure is not contained, but it has a couple of very interesting benefits. First, it is possible to weld aluminum with this method, producing only aluminum sulfide (melting point 1100°, well above aluminum's 660°; very hard but decomposes in water to aluminum hydroxide and hydrogen sulfide, from which the sulfur can easily be recovered). Second, iron pyrite is a beautiful and interesting material in its own right, quite aside from its historical usage in starting fires and in semiconductor diodes. Finally, while metal oxides like those of iron, manganese, and copper inevitably leave behind a residue of the reduced metal as well as the molten oxide, sulfur can produce a pure molten metal sulfide if the stoichiometry is correct.

Welding and sintering

Such welds can in principle be executed by pressing a powder of the thermite, or even just the oxidant, in between blocks or grains of the solid material to be welded, then igniting the mass, thus forming

the weld. Although the sulfides formed will weaken the weld, being as they are weaker than the bulk metal (except perhaps in the cases of aluminum and zinc), if the weld remains hot long enough for them to spheroidize, the loss in strength may be minimal. As I wrote in Dercuano, it may be possible to use such a process as a way to rapidly sinter a green body primarily composed of metal grains.

(I suspect it's possible to weld magnesium by this method as well, a task which is challenging in general and impossible with ordinary thermite.)

Applications of thermites to 3-D printing

By selectively depositing a small amount of oxidant in a bed of metal powder which is then suffused with an inert gas before ignition, as I wrote in Dercuano, it should be possible to 3-D print near-net mostly-metal objects with a rough surface and a small amount of oxide or sulfide trapped in spheres deep inside of them. Moreover, this should also be possible with selective deposition of a paste consisting mostly of metal powder with a small amount of oxidant and binder; the oxidant might be liquid sulfur or amorphous sulfur, in which case no extra binder would be needed; or the oxidant might be crystalline sulfur or oxides, hydroxides, or nitrates of metals, in which case the binder would be some other material such as an aqueous bentonite colloid or molten lead-tin solder. A third 3-D printing process would involve selectively jetting a binding agent, perhaps just water, into a powder bed deposited layer by layer, followed by the depowdering of the green body in the way that is currently usual for such powder-bed processes; then the green body would be ignited, perhaps after drying.

In all of these 3-D printing processes, you could use inert, dense, high-boiling, and endothermic fillers to reduce the tendency of the ignited body to evolve gas and blow itself apart; incorporating adequate porosity into the design would also help. Silica, silicon, lithium, phosphoric acid, olivine, lead, tungsten, and many other substances could be useful here. Sodium chloride and alumina are in use for this purpose today.

Other ways to heat a pocket furnace

Other ways to heat a charge of metal in an insulating chamber immediately atop a mold, so as to drop it into the mold once it finishes melting and immediately make a cast, include the use of a small in-place carbon-electrode electric arc furnace (as demonstrated by, for example, The King of Random, RIP); a small induction heating coil, which can be placed outside the refractory chamber itself (I've written about sealed insulated induction furnaces previously in Dercuano); and optical heating through a peephole, either by a focused laser or by focused sunlight, perhaps previously "collimated" by passing it through a "pinhole" before focusing, in order to permit the usage of smaller solar ovens.

Peepholes of transparent silica aerogel or aerogels of other high-temperature ceramics, such as yttria or yttralox, may permit pyrometry and optical heating without loss of heat to convection, although at these temperatures radiative heat loss is probably more

significant. They can also prevent contamination of the reaction chamber by reactive gases such as oxygen, though perhaps purging the vicinity or directly the chamber with other gases such as argon may be more effective, or the loss of scarce reaction products such as vapors of gold or mercury.

A more everyday way to melt refractory metals and reduce ceramics less refractory than tungsten is to heat them with a carbon arc in argon on a water-cooled copper hearth, which can provide the necessary grounding. However, this approach is not very efficient due to the high thermal losses into the copper, and might result in copper or carbon contamination of the melt.

Zirconium-based ceramics

Another potentially interesting powder-bed 3-D printable end product, which I didn't appreciate at the time, is the possibility of printing in yttria-stabilized cubic zirconia. This could be done either by sintering or fusing a bed or green body of powder of the ceramic with any of the thermite compositions described above, or, more outlandishly, by selectively oxidizing a bed or green body of *metallic zirconium* powder (just US$12–33/kg from China in 2015-19, according to the USGS!) with the appropriate percentage of yttrium present (about 10% on an yttria basis, ideally already as the trivalent oxide, US$3–8/kg, but conceivably just as the metal, US$34–48/kg, or as some other salt such as the acetate, formate, nitrate, or sulfate, all of which are water-soluble; or as the hydroxide). The oxidation process would inevitably leave bits of the reduced oxygen-donating metal behind, probably trapped inside the zirconia mass, probably weakening it somewhat and potentially cracking it. (The candidate donor metals — iron, copper, chromium, and so on — are strong, hard, and tough, but cubic zirconia is much harder, so it cannot transfer mechanical loads to them unless it cracks.)

No such concern about donor metal remnants applies for oxidation to zirconium *carbide*, which can perhaps be done with just zirconium and carbon; however, the temperature is probably high enough to melt any remaining zirconium, so depositing a bit of zirconium into a bed of graphite or diamond dust might be better than vice versa. Zirconium carbide is even more refractory than zirconia (to the point that I doubt the above-mentioned thermites can sinter it), but its standard enthalpy of formation is smaller, only -207 kJ/mol to zirconia's -1080 kJ/mol and alumina's -1675.7 kJ/mol; so it's conceivable that it might need a boost to fully fuse upon ignition. Analogous considerations apply to zirconium boride, for which this process is already in use, under the name "self-propagating high-temperature synthesis".

One clever trick from zirconium boride SHS, probably applicable to this entire class of processes, is to include metallic magnesium in the mix to capture unwanted oxygen from the feedstocks, preventing it from outgassing; the magnesia can then be removed with mild acid leaching after cooling. Glucina might work better for such oxygen immobilization in the sense of occupying less volume, and it is harder than magnesia, though slightly less refractory; but removing it later requires more aggressive chemicals, and of course it is considered very toxic.

Cubic zirconia is the stable structure for zirconia above 2370°, so in the case of producing zirconia it is probably necessary for the temperature to exceed this level. Candidate alternative stabilizing dopants include calcium, titanium, and magnesium; calcium and magnesium oxides in particular might be particularly easy to handle, and they provide superior mechanical properties to yttria! However, historically sintering them has been too difficult, a problem this thermite-printing process might solve; they also have worse high-temperature stability than traditional yttria-stabilized zirconia.

Including aluminum in the zirconium mix might offer some additional advantages. I think it won't give you hotter results — while I *think* aluminum has a higher energy density of 83.8 MJ/ℓ than zirconium; alumina's molar mass is 101.960 g/mol, while zirconia's is 123.218 g/mol, giving respectively standard enthalpies of formation of -16.4 MJ/kg and -8.8 MJ/kg, advantage aluminum; however, while alumina's specific heat is the low 0.88 J/kg/K, zirconia's specific heat is a super-low 0.27 J/g/K, so I think zirconium as a thermite fuel will actually get hotter than aluminum, though I think it's entirely likely I'm misunderstanding how to apply the thermodynamics. Also, though, the resulting alumina–zirconia composites offer better mechanical properties than either ceramic alone, being harder than zirconia but tougher (higher tenacity) and consequently stronger than alumina.

Several of these powder-bed processes might benefit from the powder bed being pressed in a hydraulic press, as for hot isostatic pressing but without the heat, at the time of ignition. This would tend to accelerate the reaction dangerously, but it might also diminish the tendency for the reaction to blow the workpiece apart or produce porosity, or for the heat produced to deform the workpiece.

Of course other similar metals, such as titanium, tantalum, hafnium, niobium, vanadium, molybdenum and thorium, can be used instead of zirconium to 3-D print similar ultra-high-temperature ceramics in similar ways; titanium carbide and molybdenum boride, among many other examples, have been made by SHS. As another example, there's a 1997 paper by Sundaram et al., that got titanium diboride by SHS of magnesium, amorphous boria, and rutile.

Topics

- Carborundum (p. 976) (2 notes)
- SHS (self-propagating high-temperature synthesis)
- Powder bed
- Graphite

Aluminum-air batteries

Kragen Javier Sitaker, 02020-09-23 (4 minutes)

There is a very interesting device called the aluminum-air fuel cell: a consumable aluminum anode (-1.662 V to the oxide, but the relevant oxidation here is the oxidation to the hydroxide at -2.31 V), separated from a porous carbon cathode by a thin porous insulating material soaked with an electrolyte such as sodium chloride. With a caustic potash electrolyte this produces 1.2 V per cell, but table salt provides a wholly acceptable 0.7 V.

Aluminum's atomic weight of 26.981 538 4(3), its generous three electrons per atom, the electron's charge of $1.602\ 176\ 620\ 8 \times 10^{-19}$ C, and Avogadro's number of $6.022\ 140\ 9 \times 10^{23}$ atoms / mole together give us 10.727 928 available coulombs per kilogram of aluminum, or about 7 or 8 MJ/kg at 0.7 V. This is a respectable fraction of aluminum's energy density as a fuel, 31 MJ/kg! (When burned in oxygen.) It's probably as good as you could expect from fueling a steam turbine from aluminum and air, say. This is astonishing because batteries normally don't come anywhere close to heat-engine energy-density territory.

Of course you can scale it down in a way that you can't scale down a steam turbine: a gram of aluminum should provide you with 7 or 8 kJ, and only 13 mg of aluminum is necessary to provide 100 J.

Amateur aluminum-air batteries commonly use a copper current-collector grid on the carbon cathode rather than nickel, but I suspect that will suffer anodic corrosion to copper chloride over time. Replacing the copper wires when you replace the aluminum anode should not be too hard, but neither is plating them in nickel, if you have some; maybe lead and/or tin would also work.

Removing the gelatinous hydrated aluminum hydroxide may be more difficult; maybe some sodium fluoride or monosodium phosphate would work for that if they don't corrode the aluminum fuel itself, but then they become additional consumables. (There's also the possibility that tridentate citrate complexes might help.) Mixing some ethanol or isopropanol into the electrolyte might encourage the hydroxide to de-gel without creating *too* much toxicity or fire hazard.

I've been trying to figure out what kind of small generator would work to provide, say, a laptop with long autonomy; I've been looking at model-airplane two-stroke diesel glow engines and things like that, since those seem to be the smallest heat engines around, but it's hard to find information about their efficiency, and they're also messy and noisy and don't scale down to low power. In Dercuano I concluded that under ten milliwatts on average, during use, was adequate for a responsive interactive computing experience, mostly to update the screen. 12 hours a day of usage for a month at 10 mW works out to 13 kJ, which is only about 1.6 kg of aluminum, so maybe the aluminum fuel cell can solve my problem. It certainly seems to scale down better than heat engines do.

It would probably be difficult and somewhat dangerous to get high

current output from such a battery, for all that aluminum is easily available in 10-µm-thick foil. I think you'd have to finely powder it, with all the risk of class-D fires and possible waste via air oxidation that that would entail. But you could likely get it to work.

Magnesium is another possible anode fuel for such a battery; GE produced such a device in the 1960s, using the same NaCl electrolyte. I think it does not have the problem of fouling the anode with a sticky gelatinous hydroxide, but I think its specific energy is lower, because a magnesium atom has but two electrons to give for its battery; but the voltage is higher, which might compensate.

Topics

A digital Dagarti might save your life

Kragen Javier Sitaker, 02020-09-23 (3 minutes)

Suppose your exocortex's radar detects a sniper's bullet destined for your head. How soon does it need to detect this to yank you out of harm's way?

In a worst-case scenario, the bullet is aimed at the middle of your chest, and it must jerk you either to the left or to the right by 200 mm to save you. To avoid killing you in the process, it must observe safe limits on the accelerations your body can handle; let's suppose 20 gees is a safe limit for horizontal accelerations. 20 gees over 200 mm is a kinetic energy of some 39 J/kg or 8.8 m/s. Your time-average speed over those 200 mm is, however, only half of that, or perhaps less if the acceleration is ramped up gradually: say, 4 m/s. Thus some 50 milliseconds are needed; at typical sniper-rifle muzzle velocities of 1000 m/s, this means the bullet must be detected at most 50 meters away.

This is probably hopeless with on-body radar for flechette rounds, but for better or worse, those have not been widely adopted. Spitzer bullets *have* been widely adopted, but perhaps from a 7.8-mm-diameter bullet we can expect a radar cross section of 10 mm² or so anyway. At 60 meters, this is about 3 nanosteradians, which I think is a unidirectional path loss of 87 dB, so the reflection will be 174 dB quieter than the emitted radar signal. The exocortex would have to detect this doppler-shifted -174-dB signal within a few milliseconds. This is probably not feasible with conventional narrowband radars but may be feasible with UWB pulse radar.

Possible alternatives include deploying radars tens of meters out, and taking advantage of passive radar from 3G, Wi-Fi, and the like, so that the path loss is only 87 dB. From any point of view that isn't directly in front of the bullet, the radar cross section will be much larger, and the bullet will have to come considerably closer to at least some of the radars on its way to your head.

I've previously written on kragen-tol about "bulletproof hail" which takes the alternative more Dagarti-ish approach of reacting to a detected incoming bullet by swinging one of several candidate "hailstones" of metal into position to directly block it. This is an easier approach in the sense that the hailstones can be constructed to withstand much larger accelerations than 20 gees, and accelerating them requires less force and energy than doing the same to your entire body, so a much later detection time would be acceptable; the hailstones could be initially positioned closer together than at 400-mm intervals in the sphere around your body; and it might pose less risk of injury to you. However, it also has the potential disadvantages that such a system would be highly visible and could also be used offensively, which might impose some practical obstacles to its use.

Topics

Solar netting

Kragen Javier Sitaker, 02020-09-23 (9 minutes)

One of the surprising results of the precipitous decline of photovoltaic panel pricing (which has lowered the cost of energy in sunny places), is that the tempered glass used to prevent the paper-thin PV silicon wafers from being broken by hailstones, or rocks thrown by rowdy teenagers, is now nearly as expensive as the PV cells themselves.

Greg Sittler tells me that one solution to this is to protect the PV panels with chicken wire, which absorbs some 10% of the insolation, suspended some distance above them. Chicken wire is tossed-rock-proof and hailstone-proof, and typically withstands a few decades of weathering. Galvanized 20-gauge chicken wire is about US$2.90/kg (from wireclothman), which is about US$1.30 per square meter for the "fine" 1-inch (25-mm) mesh size, or about US$3 per square meter from Ace Hardware. This seems like a very good solution to me. (You might need two staggered layers in order to reliably stop small rocks.)

Chicken wire is made of galvanized mild steel, which mostly deforms plastically upon impact, thus being broken by a sufficient number of repeated impacts. However, its hexagonal structure is better suited to absorbing impacts than a square mesh, because it's inherently stretchy, though not quite so stretchy as a knit-fabric structure would be; a knit-wire structure would be able to absorb much more impact energy for a given amount of wire by virtue of experiencing more macroscopic deformation, just as a hexagonal mesh can absorb more than a square mesh. Alternative materials that occur to me include music wire, nylon, PTFE, UHMWPE, rubber, polycarbonate, glass fiber, basalt fiber, polyimide, polyamide-imide, glass-fiber-reinforced polyamide-imide, and polyurethane.

The ideal material for this purpose would combine a high mechanical energy capacity like rubber (2–9 kJ/ℓ), nylon (0.3–2 kJ/ℓ), or ASTM A228 music wire (11–14 kJ/ℓ); excellent ultraviolet resistance like steels, PTFE, glass fiber, and basalt fiber; and low price, like nylon. Many of the polymers listed above can be heavily filled and mixed with UV-absorbing and free-radical-scavenging components in order to improve their UV resistance, but generally not to the decades of endurance in thin fibers needed for maintenance-free PV operation.

An alternative, then, suggested by Greg, is to combine virtues by using fairly stiff but UV-proof fibers like glass fiber, mild steel, or PTFE to form "trampoline panels" that are suspended around the edges by springs made from a material with a higher mechanical energy capacity. These springs might be coils, knit fabric, cantilevered leaf springs, zigzag fibers, foam blocks, or in some other shape, but at any rate they can be protected from the sun, so they can be made of cheap materials like nylon, polyurethane, or perhaps mild steel.

Both the springs and panels might need to resist creep, which might

require using a high-melting material rather than an organic polymer, though teflon and polyimide might be good enough. Also, though, it might be the case that creep is not a real concern here, because the normal relaxed loading scenario is a tiny fraction of what the protective layer must be designed to tolerate during an impact. So, for example, according to Du Pont's Teflon PTFE Properties Handbook, at room temperature teflon creeps by 100% in a few hours under a 10-MPa load (close to its ultimate strength), but takes hundreds of hours to creep by 1% under a 3.5-MPa load.

It might be helpful to use multiple different fiber sizes, like the ripstop nylon used in parachutes. A small rock of 10 mm diameter might typically weigh 3 g and be hurled at some 15 m/s, thus carrying some 300 mJ. Stopping it within 100 mm thus requires at least 3 N of force along its direction of motion. Suppose it strikes a single strand of the net, which deforms to catch it; then these 3 N might be 15 N in each direction along the strand, so the wire must withstand some 15 N.

The fiber diameter needed to resist this rock depends on the material chosen (for the trampoline panels, if those are used). Teflon's ultimate strength is about 10 MPa; that of rubber, about 16 MPa, depending on fillers; that of polyimide (Kapton), about 200 MPa; that of mild A36 steel, about 500 MPa, though its yield stress is lower, around 200; that of nylon, about 900 MPa; that of music wire, gel-spun UHMWPE, or E-glass fiber, about 3 GPa; that of S-glass or basalt fiber, about 5 GPa.

So, depending on the fiber chosen, you might need a fiber of 1.4 mm (of teflon, suggesting that teflon may be too weak for this) or of 60 μm (of basalt fiber) to stop the small rock.

But consider a larger rock, 100 mm in diameter, weighing 3 kg, thus carrying some 300 J of energy. Stopping it in the same 100 mm requires 3 kN of force, or perhaps 15 kN if it is being stopped by a single strand, requiring a 44-mm-diameter teflon bar or a 2-mm-diameter basalt-fiber rope. If the strands are 10 mm apart then perhaps we can ensure that it strikes at least 27 of them (9 in each of three basket-weave directions), so perhaps the load is only some 600 N each, requiring teflon fibers of "only" 8 mm diameter, converting the rock shield into a very effective and expensive PTFE sunshade, or 400-μm basalt rope.

So, the ripstop approach would be to make, say, 90% of the fibers thin, weak, and cheap, to catch the small rocks, and the other 10% stronger, either by making them thicker or using a stronger material.

The stronger "ripstop" or "reinforcement" cables can be thick enough to carry a thick UV-protection layer, made, for example, of carbon-black-filled teflon. Even UV-protection fillers in a polymer might slow the degradation of such a thick cable to a tolerable degree during the panels' design lifetime.

For example, you could use 0.2-mm (AWG 32, much thinner than usual 20-gauge 0.8-mm chicken wire) galvanized mild steel wire spaced 10 mm apart, then 3-mm (AWG 8) music wire every 100 mm, which you would also have to galvanize. If the weave goes in three directions, this works out to 300 m of thin wire (30 g) and 30 m of the thick wire (2 kg) per square meter.

Unfortunately at onlinemetals.com, 0.125-inch music wire costs US$15.79 per pound, or US$34.80/kg, 11 times the price (by weight) of chicken wire; so 2 kg/m² is US$70 per square meter; while PV modules, including the glass, currently only cost about US$40 per square meter, so this is still too expensive. (MSC offers a similar price; it's not just onlinemetals.)

Given that, though, I'm pretty sure it's possible to relax the problem to get the cost down to a reasonable level.

Probably what I need to quickly vet materials for such uses is a cost per newton meter: the cost for a meter of cable thick enough to resist a load of one newton. Music wire at US$15.79/pound times 7.9 g/cc divided by 3 GPa is about 90 microdollars per newton meter, while mild steel at US$2.90/kg times 7.9 g/cc divided by 500 MPa is about 46 microdollars per newton meter. (Maybe the bulk metal is cheaper than chicken wire, though.) Nylon rope costs US$81 for 250 feet of ⅜" 3050-pound-test braided rope, or in SI units, 76 m of 10-mm 1380-kg-test braided rope; that's about 80 microdollars per newton meter, but not UV-resistant.

Amazon suggests "Campbell galvanized steel wire rope, 7×19 strand core, 3/16" bare OD, 250' length, 840 lbs breaking strength" for US$61.92 ("+$209.82 shipping & import fees" to Argentina), which is 4.8 mm bare OD, 76 m, 3.7 kN, and is claimed to weigh 16.5 pounds. From the weight presumably the cross section is only about 12 mm², so as a solid round steel rod its diameter would be 4.0 mm, the rest I suppose being air between strands. The strength would thus work out to 300 MPa, so either it's a bit underrated or the spool weighs a lot; one buyer claims that the spools he's tested broke at over 1000 lbs (4.4 kN), which works out to 360 MPa. Taking them at their word, though, this works out to 220 microdollars per newton meter, not including shipping; considerably pricier than the other alternatives.

Topics

Mild bases

Kragen Javier Sitaker, 02020-09-23 (updated 02020-10-01)
(3 minutes)

Reading about cuprous oxide (p. 317), I saw a couple of plausible synthesis routes: one via Trommer's test, which requires cupric ions from, say, blue vitriol, and reduces them at high pH with a reducing sugar, and one via electrolytically produced cupric chloride, which etches copper to make cuprous chloride at low pH, at least if chlorine is available, and which can then be precipitated as cuprous oxide with a base.

Unfortunately, both of these routes require a base, and the main bases I have handy are household ammonia, baking soda, and sodium percarbonate, which decays to washing soda and oxygen when wet. The trouble with the sodium compounds is that they both contain carbonate, which will bind any cupric ions quite firmly indeed, and maybe cuprous ones too; the trouble with the ammonia is that it will complex with the cupric ions and may make it harder to reduce them to cuprous.

I also have household bleach handy, which includes some soda lye in order to stabilize the sodium hypochlorite. One possibility might be to neutralize the bleach with hydrogen peroxide, producing oxygen, water, and sodium chloride, which I think will leave the lye intact. But I'd rather avoid lyes if I can. (I could probably produce it more easily by electrolysis from sodium chloride, and maybe leaching hydrate of potassa from wood ash would also be an option. I'd thought I could also calcine the sodas, but apparently that doesn't produce lye until past 2000°.)

Another possibility that occurs to me is to produce a metal hydroxide through electrolysis. But the water-insoluble hydroxides like that of aluminum are of limited usefulness for reactions like those above — though they might be able to raise the pH to 7, which might be enough to convert solvated cuprous chloride to insoluble cuprous oxide. And the water-soluble hydroxides of the alkali and alkaline-earth metals wouldn't need electrolysis to produce them if I had the metals, but the metals are a huge pain.

Slaked lime is, of course, another possibility, but it probably needs to be freshly calcined, which is also not easy to do in this kitchen.

A somewhat less mild base, with its own toxicity problems, is sulfide of soda. It is made by carbothermic reduction of sal mirabilis and can also be made from sulfide of lime with enough washing soda, though markgollum says the yield is poor. I am not sure if sulfide of lime also acts as a base, but it can be produced by carbothermic reduction of alabaster or, along with washing soda, from chalk and sulfide of soda, as practiced by that great physician; or, historically, by heating oyster shells with sulfur to red heat, though less than 1084°, again according to markgollum. Slaked lime with sulfur is also reported to work at only around 100°. Sulphuretted hydrogen over muriate of lime has also been suggested, though carbonic acid gas liberates sulphuretted hydrogen from the sulfide of lime.

Topics

Magnesium fuel

Kragen Javier Sitaker, 02020-09-23 (updated 02020-10-09)
(13 minutes)

Magnesium has an energy density of 43.0 MJ/ℓ and a specific energy of 24.7 MJ/kg. This is among the highest energy densities of any easily burnable fuel — iron, polystyrene, polyethylene, and lithium borohydride are similar, while the more difficult aluminum, carbon, and silicon are up in the 70–84 MJ/ℓ range. (See, however, the note on aluminum-air batteries (p. 326) and the note on lithum as a fuel (p. 371).) It excels iron at specific energy, and polystyrene, polyethylene, and lithium borohydride excel it. But burning polystyrene, polyethylene, and lithium borohydride produces a lot of gas, spreading out the heat a great deal. So, for compact, easily ignited fuel to produce a high temperature, magnesium is pretty much tops. As a bonus, it's pretty abundant and easily obtained from seawater; see notes below on smelting.

Energetics of magnesia

Magnesia has a molar mass of 40.3 g/mol and a heat capacity around room temperature of 37.2 J/mol/K; dividing these two gives an unremarkable specific heat of 0.923 J/g/K. Magnesium itself has a molar mass of 24.3 g/mol, so magnesia (MgO) is 60.3% magnesium; burning a kg of magnesium yields 1.66 kg of magnesia, and, as mentioned above, 24.7 MJ. From this we can derive that, if its specific heat remained constant, the resulting magnesia would be at 26500°, which means that in practice the upper limit to the temperature will be imposed by heat loss mechanisms and the finite speed of combustion, since this is several times hotter than the surface of the sun.

Thus we have magnesium flashbulbs.

Consider a kilojoule. We can store it in 23 microliters of magnesium weighing 40 mg. Liberating it requires another 26 mg of oxygen, for example from the air, which contains it at about 210 mg/ℓ, so about 130 mℓ of atmospheric-pressure air are needed; the reaction can be arranged to proceed at rates of anywhere from tens of watts or so up to a megawatt by controlling the introduction of the air, as long as the hot magnesium doesn't start reducing its reaction chamber, or of course melting it. If it is necessary to carry the oxidizer as well, water works well once the reaction is going, since water is 89% oxygen; 26 mg of oxygen as water thus occupies 29 µl. (See below, though.)

This makes magnesium appealing as a compact way to store energy capable of safe, controlled high-power release. One of the few examples of this being done in practice is the MAGIC engine developed by Mitsubishi and Takashi Yabe and others at the Tokyo Institute of Technology, which also used a water oxidizer; Yabe has also worked on magnesium-air fuel cells.

The oxygen-magnesium reaction produces no gaseous products unless the temperature is allowed to go very high (magnesia boils at

3600°, though magnesium melts at 650° and boils at 1091°), but the water-magnesium reaction produces hydrogen. The MAGIC engine secondarily burns the hydrogen produced in air to recover the enthalpy of formation of the water, which was drawn from the initial water–magnesium reaction. Water's standard enthalpy of formation is −285.83 ±0.04 kJ/mol and its molar mass is 18.01528(33) g/mol. Magnesia's are −601.6 ±0.3 kJ/mol and 40.304 g/mol (compared to, say, −1675.7 kJ/mol and 101.960 g/mol for alumina, nearly the same energy density); although I'm not very sure of my understanding of the thermodynamics, I think this means that splitting the water sucks up about half of the heat you'd otherwise get out of the reaction, since both MgO and H_2O have a single oxygen, so a mole of H_2O produces a mole of MgO; so you would need about twice the amount of magnesium to produce a given amount of energy.

The hydrogen also soaks up 28.836 J/mol/K of heat, lowering the potential maximum temperature further, but I think by another factor of less than 2. So we're still talking about maximum temperatures that exceed magnesia's boiling point.

(Under appropriate conditions you can generate hydrogen at room temperature from magnesium and water.)

Using oxygen rather than water you should get the full 601.6 kJ/mol, which divided by magnesium's 24.3 g/mol works out to 24.8 MJ/kg, close to the 24.7 cited above. This makes me think I *am* understanding the thermodynamics properly.

Engine design

For controlling the reaction rate, the most appealing option would seem to be preheating the magnesium to somewhat below its melting point, then introducing the oxidizer at a controlled rate. The temperature will immediately rise enough to melt the magnesium, which over a long enough timescale will reduce the available area for the reaction to take place; but in many circumstances the reaction can be run to completion on a shorter timescale than that, and the increasing temperature may be an effective countervailing force.

Maintaining the magnesium fuel in large solid pieces until near time to use would be a useful safety measure. These would be much harder to ignite accidentally. Perhaps the simplest approach would be a round magnesium rod that twists in a device exactly like a manual pencil sharpener to shave off shavings of a calibrated thickness.

Under some circumstances, it might be best to first preheat some magnesia by burning magnesium, with little or no gas release, and then use a second, later burst of gas to move the generated heat to where it's needed. This decouples the time during which the combustion happens — which may be limited by, for example, considerations such as the one mentioned above of burning the magnesium to solid magnesia fast enough that it doesn't melt into a round mass with little surface area, or inversely by the inability to burn the magnesium as fast as would be desired because of limited surface area — from the time during which the heat is transferred to where it will be used, which might be shorter or longer than the combustion time.

Recharging, or smelting

Recharging spent magnesium fuel should be considerably easier than the analogous process for aluminum, which is especially interesting for use as a motor vehicle fuel. Something like three fourths of magnesium today is produced in China by the Pidgeon silicothermic process, boiling magnesium vapor at sub-atmospheric pressures out of mixed MgO and ferrosilicon powders at 1200°–1400°, and further stabilizing the silica byproduct with CaO. However, the historically dominant process was electrolysis of molten $MgCl_2$ produced from HCl and MgO; the electrolysis releases the Cl_2, which can be exothermically recombined with H_2 with ultraviolet light, even in aqueous solution, which tames the process a bit. Pure $MgCl_2$ melts at 714°, but, e.g., Davy fluxed it with corrosive sublimate to discover magnesium at a tamer temperature; so a recharging apparatus of a reasonably small size and temperature might be feasible.

A new, more direct process uses a solid zirconia electrolyte to directly electrolyze MgO at 1150°–1300°, in order to drop the cost of magnesium for *structural* applications in vehicles. The cathode is a bath of molten MgO through which argon is bubbled, coming out containing Mg vapor. The O_2 can travel through the zirconia to the cathode, made, for example, of molten copper, tin, or silver, or of a zirconia–nickel cermet coating on the zirconia. Magnesia-stabilized zirconia is more stable in the molten salt bath, but lower conductivity; they found some kind of sooper seekrit ingredient to keep the molten MgO from corroding regular yttria-stabilized zirconia. Like the Pidgeon process, the magnesium produced is in vapor form, and so a distillation step inherently purifies the reaction product. (With appropriate "fluxes" or molten-salt solvents, this same SOM process has been used to smelt iron, silicon, tantalum, and titanium.)

Magnesium sulfide

Sulfur is an alternative oxidant; MgS has a molar mass of 56.38 g/mol and an enthalpy of formation of -347 kJ/mol, which works out to 14.3 MJ per kg of magnesium or 6.2 MJ per kg accounting for the oxidant as well. This reaction is used commercially to remove unwanted sulfur from steel. It doesn't melt until 2226°, and its boiling point is not known, though probably a bit lower than magnesia's. Interestingly, it can react with oxygen to give epsom salt rather than the little-known sulfite.

The sulfide's heat capacity at room temperature is 45.6 J/mol/K, which works out to a specific heat of 0.809 J/g/K. This extrapolates out to 7600 K. This is a lot cooler than the extrapolated temperature for magnesia but still pretty toasty.

This might be useful in cases where limiting the reaction rate is not desired, but it probably isn't safe in more than milligram quantities because of the rapidity of the reaction.

Magnesium-silica reactions

Although, as explained above, you can smelt magnesia with ferrosilicon at 1200°–1400° with lime, because the gaseous magnesium leaves the reaction and the lime stabilizes the silica

product as larnite, under normal conditions the reaction tends in the opposite direction: magnesium will reduce silica to silicon. As the International Magnesium Association cautions:

> The refractories used in the furnace should be high in alumina or magnesia because molten magnesium can react violently with even small amounts of silica (often [sic] present in ceramic materials).

SiO_2 has an enthalpy of formation of -911 kJ/mol, while MgO's enthalpy of formation is -601.6 kJ/mol. So at room temperature I think the reaction would be $SiO_2 + 2Mg \rightarrow 2MgO + Si + 293$ kJ per mol of silica, noticeably exothermic, though of course a very low reaction rate. Magnesia's 40.304 g/mol and silica's 60.08 g/mol add up to 140.69 g for the left side of this reaction, which works out to about 2.08 MJ/kg with both reagents in the denominator. Counting just the magnesium, 3.63 MJ/kg, so (I think) that's the intrinsic energetic cost of the Pidgeon process's endothermic aspect. The intrinsic energetic cost of the silicon in the ferrosilicon feedstock is three times as much per kg of magnesium produced.

The easily-accessible temperature-dependent equilibrium reversal of this reaction interestingly makes magnesium somewhat interconvertible with silicon, even in the impure form of ferrosilicon, assuming you have ample supplies of their ubiquitous oxides. Metallurgical-grade silicon is mostly produced by carbothermal reduction, though aluminothermal reduction is also performed; this suggests a mostly solar thermal route to smelt magnesium.

One of the most interesting aspects of the solid oxide electrolyte process for magnesium production mentioned above is that the resulting white-hot magnesium vapor is capable of reducing not oxides only of silicon but indeed of nearly any metal, including exotics like titanium and tantalum (though not, apparently, zirconium and yttrium, at least not fast enough to prevent the electrolysis from proceeding). Magnesium's first ionization energy is 737.7 kJ/mol, kind of a middle-of-the-road value for metals, so I don't think the issue is that it's extremely easy to oxidize magnesium. And from an entropic point of view you would think temperatures well above magnesium's boiling point would tend to favor the reduction of magnesium, as it does in the Pidgeon process — presumably at a high enough temperature titanium would instead reduce magnesia to magnesium vapor, just as silicon does. I guess I need to go read about Gibbs free energy.

At 1200° titanium dioxide is solid (melting point 1843°), magnesia is solid (melting point 2852°), and titanium is solid (melting point 1668°), but magnesium is a gas (boiling point 1091°, as mentioned above).

A practical aspect of the silica-magnesium reaction is that you maybe shouldn't throw silica sand on an unwanted magnesium fire. You will be disappoint.

Topics

Ancient batteries

Kragen Javier Sitaker, 02020-09-23 (updated 02020-12-31)
(4 minutes)

The Baghdad Battery was not used for electroplating, but Mythbusters has convincingly shown that it could have been: ten similar copper-iron cells with lemon-juice electrolyte produced a total of four volts and successfully electroplated a token.

Did the ancients use electrochemistry without knowing it?

I wonder if particular applications of electrochemistry were actually used but not understood. For example, in, I think, Huckleberry Finn, which is full of folk beliefs about such things, a character is at one point faced with the problem of what to do about a counterfeit quarter, which had worn and was showing the copper around the edge. So they put the coin inside a baked potato overnight, which restores the silver appearance to the edge. Huckleberry Finn was published in 1884 but purports to describe the beliefs and folkways of people in Mark Twain's childhood around 1840.

Silver's standard electrode potential is +0.7996 V, while copper's is +0.520 V. Alkali metals and alkaline earth metals, which are the most enthusiastic about giving up their electrons, have among the most negative standard electrode potential, so this means that, in a copper-silver circuit, copper has a slight potential to give up electrons to the electrolyte and oxidize, dissolving copper cations into the potato, while silver would tend to acquire electrons from solution and be reduced. This calculation is confirmed by the existence of the "red plague": partly-silver-plated copper conductors exposed to moisture form red cuprous oxide through galvanic corrosion, rather than the silver anodically protecting the copper. This is unfortunately the reverse of the mass flow direction that would be required for the counterfeit-quarter trick to work.

The other metals known to the ancients were lead (-0.126 V), gold (+1.52 V, but I think not actually usable), iron (-0.44 V), tin (-0.13 V), mercury (+0.85 V), and, in India, zinc (-0.7618 V, or -1.199 V as zincate). Among other significant battery electrode materials, carbon (whose electrode potentials are, I think, irrelevant) and air (oxygen? +0.401 V?) were also known. Although the insulators litharge and minium were known, lead dioxide was not.

(In modern lead-acid batteries, the lead dioxide is formed electrolytically by the sulfate electrolyte, but of course this requires an external voltage source.)

Of these the champions on the positive side are mercury and silver, while the champions on the negative side are iron and zinc. This suggests the possibility of constructing a battery with some 1.8 volts per cell, almost the same as modern batteries, out of ancient metals, or something like a lead-acid battery, but using a different electrolyte, out of ancient materials. It would probably be more practical,

though, to use plentiful copper, rather than precious mercury and silver, despite the rather miserable voltages available.

Gold probably *does* work

Above I said I didn't think gold would work, thinking it would be too inert to interact with electrolytes. But gold here wouldn't be the electrode that would need to dissolve, so Bernd Jendrissek tried some gold-plated copper contacts along the edge of an old SIMM with a paper towel soaked with salty vinegar and a copper wire on the other side of the paper towel, getting 0.3 V and a few microamps. So, if Jendrissek's report is correct, it seems that gold *does* work.

Topics

Modern material processing

Kragen Javier Sitaker, 02020-09-24 (updated 02020-09-26)
(8 minutes)

A lot of laboratory apparatus for processing exotic materials hasn't changed much since the time of Bunsen — though electric heating mantles are new, and now stir bars are covered in teflon instead of glass, many other things are still the same. What might it look like if we designed it from scratch today?

Perhaps we would focus more on continuous-flow processing rather than batch processing, handling very small quantities of liquid or other material at a time. Maybe we'd use teflon gaskets more. Our lego blocks might be smaller than traditional flasks; we might have micro-channel blocks that swing open at a parting line to expose the interiors of the channels for cleaning, pressed together by a clip. Captive gaskets between them would separate the cavities along the parting line.

Some blocks could incorporate their own electrical heating elements, catalyst foams, valves, and so on. For thermal insulation, they could incorporate vacuum "panels" containing multilayer insulation, for example of gold leaf or aluminum foil.

Microfluidic devices are capable of executing many experiments at once, and are widely used for this in biology today.

At ordinary temperatures gaskets would be made of teflon, but higher-temperature blocks would use gaskets that are brittle at room temperature, softening as their Tg is exceeded. Alternatively, modern fabrication technology is capable of shaping mating faces to match with submicron precision, which by itself may reduce leakage sufficiently without gaskets. If not, greasing the mating faces with liquid (whether water, table salt, sulfur, H_2SO_4, lithium with its delightfully low vapor pressure, other liquid metals such as lead-tin solder, or something else) may be an option, although they probably couldn't be separated cold.

There is in most cases no need for the surface of vessels to have a significant affinity for liquids; superhydrophobic or omniphobic surface treatments would significantly reduce corrosion, cross-contamination, and cleaning effort. Fluorinated surfaces are another, cheaper alternative. Moreover, in many cases, such a surface would prevent the entry of water or other liquids into small channels without pressure being applied.

The use of such blocks could make high-pressure processing routine, since they can easily be made to contain high-pressure materials.

Glass is fragile; in many cases it could be replaced by fluorinated plastics or fluorinated ceramics. Modern fabrication techniques are capable of mechanically carving channels through many materials, and often of molding them as well.

One possible candidate replacement for glass is fluorite, which cost US$300 per tonne in 02020, can be optically clear even into infrared and ultraviolet, and has good resistance to some corrosive agents. It

doesn't melt until 1418°. Although it is noticeably more fragile than the traditional borosilicate, its major flaw, which is fatal for many uses, is that it cannot deal with strong acids.

Crystalline alumina is better than borosilicate in every way except that it is very difficult to shape. There is no known alumina glass, and polycrystalline sintered alumina ceramics are in wide use, but they are never transparent. General Electric's 01961 "Lucalox" sodium-vapor lightbulbs ("transLUCent ALuminum OXide") were *translucent* sintered alumina; the 1959 Nature report on them said they were "presumably sintered *in vacuo*", and that objects viewed from a distance were "blurred as though through frosted glass". A 01996 article by J.E. Burke, one of its inventors explained that it was doped with a fraction of a percent of MgO before sintering at 1800°, preventing pore retention by preventing discontinuous grain growth that traps pores within grains, and that pore-free alumina is translucent because alumina is birefringent. He also notes that to bring out grain boundaries by etching, he resorted to etching the alumina with molten exotic $K_2S_2O_7$, and that sintered pore-free alumina is "the only successful material found to date [01996] for containing the plasma of the high-pressure sodium-vapor lamp".

Yttria-stabilized zirconia is already seeing some use in labware, but usually in a non-transparent form — though transparent YSZ is widely used for jewelry. It is outstandingly resistant to corrosion, fairly hard (though less so than alumina), and above all, tough. When hot (900°–1300°, maybe lower) it conducts electricity in the form of oxygen ions, and thus can be used to electrolytically add or remove oxygen from a hot reagent.

Other transparent ceramics include yttria doped with 10% thoria to prevent discontinuous grain growth, analogously to Lucalox (the former Yttralox, which contains no alumina, despite its name) and lanthana-doped yttria, melting at 2430°. Unfortunately yttria is vulnerable to concentrated hydrochloric acid with ammonium chloride, thoria is radioactive, and lanthana is soluble in dilute acids. Also, Rosenflanz's alumina-containing rare-earth glass-ceramics from 2004; yttria–alumina garnet (YAG); aluminum oxynitride spinel; magnesium aluminate spinel, aka just "spinel"; topaz; silicon carbide; chrysoberyl; beryl, such as emerald; berlinite; chalcogenide and phosphate glasses; transparent glass-ceramics like Corning VISION; zircon (zirconium silicate); rutile; boron nitride; and so on.

Fluorinated polymers may be a reasonable inert alternative to ceramic materials for many low-temperature purposes, as well. Tubes of materials like PVDF (Kynar) can carry most materials with impunity, though only up to 150° in that particular case, and also not very aggressive materials. Teflon can withstand both higher temperatures and more corrosive materials, and many plastics and even metals can have their surfaces fluorinated in order to make them more inert to their contents, as well as in most cases decreasing wettability, as mentioned earlier.

Blocks can be instrumented with not only stir bars and heaters but pumps, thermometers, pH meters, dielectric spectrometers, regular spectrometers, imaging spectrometers, sonar (to measure specific acoustic impedance, speed of sound (from which density can be

inferred), and distance to a liquid surface), immersion densitometers, ion exchange beds, flow meters, chromatography columns, distillation columns, NMR equipment, reflectometers, X-ray fluorescence and diffractometry equipment, and strain gauges for both pressure and weight.

Pumps need not use a shaft through a sliding seal; like stir bars, their impeller can be driven by a magnetic field from without, a trick even easier to do with a flow meter. Also, piezoelectric devices can set up oscillations in the fluid that drive a "fluidic diode" type of pump; in some cases pure fluidic pumping driven by a stream of a friendlier fluid can be used; and in some cases peristaltic pumping is applicable. In the special case of conductive liquids, magnetohydrodynamic pumping can be used.

By moving a supporting rigid support along an unchanging beam and measuring the changing torque at the support, the mass distribution of the beam can be easily calculated, most precisely toward the center of the beam. In general there is a tradeoff between precise measurement of weight and sealed couplings that permit easy fluid passage through the apparatus, but (at least without a pressure difference between inside and outside) it can be eased somewhat with flexible couplings, in particular plane-like couplings, like flexible circuit boards with fluid channels bored through them.

Topics

Materials shopping list

Kragen Javier Sitaker, 02020-09-25 (updated 02020-12-20)
(1 minute)

Pharmacy

- sulfur
- lunar caustic
- agua blanca del codex
- borax?

Pool place

- pool pH lowerer
- blue vitriol
- activated charcoal
- infusorial earth (diatomea)
- indicator paper
- concentrated detergent
- waterglass

Hardware store

- lime (not at shitty one) (not at good one)
- caustic soda (check, though mislabeled)
- converter (check)
- boric acid or borax (not at shitty one) (not at good one) (check, borax)
- infusorial earth (not at good one)
- plaster (check)
- magnesium (not at good one)
- silicone (check)

Garden store (not Catrilo 4713, that's a pet store)

- Glauber's secret salt
- green vitriol
- infusorial earth
- boric acid or borax (fertilizante)
- urea (fertilizante) (or at gas station)

Health food store

- Alum
- Salt without sodium (done)
- Cream of tartar (not at local one)
- Citric acid

Grocery store

- baking soda (cheaper via Viento Norte? AR$103/kg)
- food coloring
- depilation wax

Parks/neighborhood

- pine resin
- copper wire

Recycler

- copper
- lead

Bath salts

- epsom salt (also available at pharmacy)
- kaolin
- baking soda
- borax

Art supply store

- clay
- kaolin
- resins?
- red copper oxide for ceramic
- modpodge?

Paint store

- converter
- hematite
- glow-in-the-dark
- resins!
- waterglass?
- lime?
- carbon
- masilla reparadora inoxidable?

Topics

- Materials (p. 788) (51 notes)

Toolpath optimization

Kragen Javier Sitaker, 02020-09-27 (updated 02020-09-30) (19 minutes)

A general recipe for planning how to make things to fill some requirement Q: given a simulation of the construction process S and a sort of distance measurement M, minimize M(S(P), Q) for some given Q; P is some sort of plan, such as a toolpath.

In many cases it's convenient to separate the simulation S(P) into a simulation C of the construction process and a simulation E of the usage or testing of the artifact produced by it, E(C(P)). For example, C might convert a toolpath into 3-dimensional geometry and an expected cycle time, and E might perform a finite element analysis and preserve the cycle time. In this case our loss function expands to M(E(C(P)), Q).

This simple recipe ramifies in all sorts of interesting ways.

Probably nothing in here is original; it's all pretty obvious to someone who knows this stuff. But I haven't seen it presented this way anywhere else, it wasn't obvious to me, and I don't see it used in practice much.

Construction simulation C

Most CAM systems have very simple simulation capabilities, sometimes nothing more than incrementally setting voxels to zero or even, for a CNC lathe, pixels. Consequently the most thoroughly automated manufacturing processes have been those like single-point metal cutting in which the interaction between the tool and the workpiece is very simple. Modern waterjet cutting systems are learning to compensate for waterjet divergence. But the modern simulation techniques used in systems like ANSYS are capable of simulating extremely complex systems, and simulation of, for example, work hardening makes it possible to simulate the dynamics of sheet metal stamping or single-point incremental forming with some confidence.

Simulating FDM 3-D printing requires simulating the change in plastic viscosity as it cools after coming out of the hotend, as well as the changing tensions in the viscoelastic material as it stretches, squishes, cools, and droops under the influence of gravity. Layer-to-layer adhesion is a product of, among other things, the heating of already-solid material by newly-deposited material. If you want to accurately predict the effect toolpath variations will have on the mechanical properties of such parts, you need to simulate these effects.

A particularly interesting class of techniques achieving prominence in computer graphics in recent years are the "material point methods", which are hybrid particle–field simulation methods capable of achieving surprisingly visually convincing simulations of snow, cloth, sand, water, hair, cracking mud, and other difficult materials, as well as complex material interactions. It's possible that the MPM might make it possible to adequately simulate more

complex material dynamics during the construction process, such as the thixotropic and frictional behavior of clay smeared by a spatula. If not, perhaps other numerical techniques would be adequate; for example, FEM with or without adaptive remeshing or finite volume methods.

However, in order to make it practical to run an optimization algorithm over your simulation algorithm, there are special considerations. One is that it needs to be fast enough to be run many times. Another is that, for many of the currently most successful optimization algorithms, it needs to be *differentiable*, typically using reverse-mode automatic differentiation.

Evaluation simulation E

Once you've simulated the thing being made, you need to evaluate the simulated thing's simulated performance. Perhaps you are interested in its appearance from a certain angle, or the load it can bear over a certain area, or its acoustic frequency response; literally anything that can be quantified in simulation can be used for evaluation.

This simulation will often use different algorithms than the construction simulation. But it, too, is subject to the constraints of rapidity and possibly differentiability.

Measurement M of fitness for purpose Q

Once you've computed the total performance evaluation, you want to measure how *fit* that performance is for the purpose you had in mind, reducing its badness (or equivalently its goodness) down to a single scalar score, so that optimization becomes a meaningful concept. This may include a variety of factors; for example, bridge designs that fail to span the gap, fall down, or block the river beneath might all get very heavy penalties, much larger than bridge designs that merely cost too much.

Optimization algorithms

Many high-dimensional optimization algorithms can be used. The most fashionable at the moment, due to their extensive development for optimizing ANN parameters, are variants of gradient descent such as Adam; but you can also use things like Nelder–Mead, genetic algorithms, and the goofy hybrid of Nelder–Mead and the method of secants that I wrote about in Dercuano, and probably also lots of things I don't know about yet.

Some of these algorithms require the gradient of the loss function with respect to the design variable vector P.

Many of these algorithms in their usual form require a space of constant, finite dimensionality over which to optimize; depending on the structure of the problem, it may be straightforward to add more design variables over time.

It's worth noting that finding the *true* optimum is often unnecessary and nearly always infeasible in practice. The above algorithms generally give a *close approximation* of a *local optimum*, but are not guaranteed to be able to do even that.

There is an applicable generic optimization approach used in a number of very interesting recent research papers from Disney Research Zürich on the computational design of linkages, compliant mechanisms, metamaterials, acoustic responses, and so on. The process starts by generating a database of random samples from a parameter space of 2–10 dimensions, characterized according to their properties of interest (our E above). Then, for each new design objective Q, the database is searched; the nearest random sample is selected, and continuous optimization algorithms such as gradient descent are applied to generate the nearest point in the property space (E) reachable by any design in the parameter space (our P, or perhaps C(P), since the Disney group generally doesn't try to simulate the fabrication process itself). Sometimes multiple components from the database are combined into a single design. Sometimes additional optimization algorithms are used, often in a new, higher-dimensional parameter space.

Tolerances

Performance *predictability* or *stability* may be a crucial factor for fitness: if a particular toolpath achieves very high fitness, but toolpaths displaced by tiny errors achieve very low fitness, then perhaps it is not a very good design. The traditional way to handle this in analog circuit design is by Monte Carlo simulation: running many simulations with component values slightly perturbed in the ways that they are known to be perturbed in real life, for example by temperature or manufacturing variation, and with some noise added to the input. In this way we can distinguish predictably good designs from designs that might be good once in a blue moon.

Affine arithmetic, interval arithmetic, reduced affine arithmetic, and SAT solvers are alternatives to Monte Carlo simulation which may be more efficient for a certain problem.

Incrementalization

A straightforward application of the above recipe requires you to repeatedly revise P, re-evaluate C from scratch on it, re-evaluate E from scratch on the result, re-evaluate M from scratch on that result, then possibly do all of that again backwards and in high heels to calculate the gradient of M with respect to P, and finally run a optimization step. But in most cases P is almost the same as a previous P, so most of the answers will also be almost the same.

Caching-based incrementalization approaches, like Umut Acar's "self-adjusting computation", can often provide speedups of five or so orders of magnitude if P is *almost the same* in a very particular sense: if most of its components have *no* change, but some of them have *arbitrary* change. Incrementalizing the procedure in this way is not helpful for gradient descent as such, since very few of the components of the gradient are ever precisely 0, but if we modify the optimization procedure to search along only one or a few dimensions of P at a time ("coordinate descent") — perhaps the ones whose component in the gradient is largest — then we may be able to get a big speedup out of this kind of incrementalization. SKETCHPAD's constraint-satisfaction algorithm used a relaxation approach

somewhat similar to this.

(Self-adjusting computation and similar approaches also suffer from some of the same time–space tradeoff difficulties associated with reverse-mode automatic differentiation; indeed, the memoization store produced by self-adjusting computation can be used directly for reverse-mode automatic differentiation. I suspect the usual periodic-checkpoint approach to reverse-mode automatic differentiation may be more difficult to apply to self-adjusting computation.)

(When doing coordinate descent with some kind of memoization, it may be possible to speed the affair up by not memoizing the intermediate evaluations during each line search or hyperplane search. Also, updating the gradient incrementally would probably blow all the advantages of incrementalization, so maybe you want to do that search with successive parabolic interpolation or something.)

I think the self-validating arithmetic approaches mentioned above can also offer, in some cases, an alternative incrementalization approach. For example, we can calculate bounds on the possible values of M — and all intermediate variables cached by a memoizing incrementalizer — for values of P within a certain many-dimensional bounding box. If a gradient-descent step moves our estimate of P, but only a few of the new components are outside the bounding box previously used, we can do the computation incrementally as before, updating just those components.

There are some loose ends there with respect to how big a bounding box you pick, and when you decide to shrink it, but I think it's tractable.

This hybridization of self-validating arithmetic with self-adjusting computation is more general than just checking the inputs; intermediate memoized values can also be checked to see if they are within previously computed bounds, or can be made so by narrowing the bounds on the new input value, and in this case the previously memoized values can be used from there on.

Replanning during construction (automated improvisation)

As the actual process of making a thing happens, new information comes to light. Perhaps some dimension was achieved to better-than-expected precision, or a block of wood has less knotholes than feared, or a piece of clay is drier than expected. In these cases a fully general response is to rerun the whole planning process, given the current (or near-future) state as the starting point, in order to take advantage of the new information.

At times it's necessary to plan out expectations which will permit the original plan to continue to be followed. For example, perhaps the profile of light on a rotating clay object should be within certain limits; if not, actuation should immediately cease, reverting to some sort of "safe" or "home" position likely to do minimal further damage, until a new plan can be formulated. Less dangerous departures from expected results may permit the original plan to continue while new plans are hatching.

XXX anytime

Indirection in construction and design

A lot of the process of making things is indirect, to the point of shaving the proverbial yaks. Sometimes this indirection is necessary to get the job done at all; at other times it merely improves efficiency or quality. Sharpening your knife or your wood-planing blade every few minutes of cutting, or whenever they get dull or nicked, will allow you to cut faster despite the lost time. A form tool on the lathe can often produce a particular contour much faster than a single-point cutter can; grinding the form tool may save you time. Adding an assembly step at the end of a process can allow you to stamp a product out of sheet steel instead of milling it out of a billet, making it orders of magnitude cheaper. A PLA FDM 3-D printer can't make things out of sheet steel, but it can definitely print press-forming dies for a sheet-metal brake, or beading dies for a bead-rolling machine.

So it's worthwhile to keep in mind the possibility of *indirect* construction, by constructing tools or parts that are then used — perhaps many times — for the desired final product.

Using a single stamp or thread-cutting die or D-bit or mold or whatever many times during the making of a thing implies that many parts of that thing will be the same, which in some sense means that your vehicle of indirection will be a compromise between the needs of those different parts. This compromise has a computational benefit, though: it reduces the dimensionality of the space to be optimized. Moreover, the design of that reusable part is potentially valuable for other, unrelated designs, perhaps stored in a database like the Disney Research Zurich linkages mentioned above.

The invention of reusable approaches that can be applied to many parts of a design is not limited to physical tooling, though; things like "gusset", "tube", and "truss" are commonly useful to reduce the mental effort of mechanical engineering, and things like "differential pair", "negative feedback", and "cascode" are commonly useful in the same way in analog electronic design.

There's a hypothesis that the reason structures like bipinnate compound leaves occur in totally unrelated families of plants (ferns, mimosas, and fishtail palms, for example) is that they are computationally simple to describe in some absolute sense. But another possibility is that they're simple to describe *in terms of highly conserved plant genetic capabilities*. With this in mind, you could imagine optimizing not a specific toolpath itself but a sort of "genome" or "program" to generate a toolpath — an indirection in the design process analogous to the indirection of a reusable drillbit in the construction process. Doing this successfully will give a design containing reusable parts, not just in the sense of actual immutable parts such as a hinge but also in the sense of design tricks like cascodes and gussets.

Experiment design and system identification

Above I described C as a function of one variable: P is the design

toolpath, C(P) is the object or range of objects resulting from executing that toolpath, and E(C(P)) is the performance of that object. But really C is also a function of the manufacturing process and the materials' properties; we could say C(P, T), where T is this description of the process and materials.

In some cases not enough is known about either the manufacturing process or the materials' behavior in use — T, that is — to simulate them with any confidence. In such a case we have a different design objective, one in some sense diametrically opposite to the "tolerances" section above: we want to know the cheapest and quickest toolpath that will reduce our uncertainty about the unknown variables. So, for example, to shape something out of plastic clay so that it will work, we would like to use a toolpath whose results vary as little as possible over a wide range of plasticities, since the clay's plasticity varies rapidly over time and in different parts of the clay. But, to find out what that plasticity *is*, we would like to use a toolpath whose results vary *as much as possible*. This is perhaps in a sense the difference between science and engineering, or exploration and exploitation in reinforcement learning, but we still want the results to vary minimally with *other* unknown properties such as ambient illumination, so that we can confidently interpret our experiment's results.

To some extent it may be possible to mix such experimentation into the construction process to support replanning; perhaps prodding the clay a bit in a spot we will smooth over later anyway, for example, can yield useful observations without affecting the final result. In general constantly adding a little bit of noise well within tolerances can provide a "subliminal" experimental result of the effect of that noise; to the extent that the phenomena involved are linear, we can confidently extrapolate from these very small effects to much larger ones. The noise can even be much smaller than existing noise in the system, only detectable by correlating over a large interval (for example, imaging a large area of clay, or feeling a long plasma cut in metal). Impulse responses for a convolution reverb are sometimes acquired from real spaces in this way by using white-noise excitation.

The simplest way to interpret the results of such experiments is to use the same simulation-optimization process as used for design, minimizing M(E(C(P, T)), Q); but now the toolpath P is fixed (it's the experiment we performed), Q is our observations from the experiment, T (the description of the process and materials) is the "design variables" for the optimizer, and E and M are the observations we have available and the probability of various kinds of errors and corruptions in them, instead of real-world performance of a design and how well that performance fulfills engineering requirements.

"Making things"

Although above I've focused on manufacturing, this approach is quite generally applicable to control and design problems; the "things" being made need not be physical objects. In the cybernetics literature approaches like the above are commonly called "optimal control theory".

Topics

Reducing sucrose

Kragen Javier Sitaker, 02020-09-30 (7 minutes)

Sucrose's enthalpy of formation is -2221.2 kJ/mol according to NIST, and it contains 11 oxygens, 12 carbons, and 22 hydrogens. What would happen if you decomposed it in a very oxygen-hungry environment? When would the reduction of the sucrose be exothermic?

Candidate reduction products

If you were to strip off just the oxygens you would be left with $C_{12}H_{22}$, which is four hydrogens short of being the saturated hydrocarbon dodecane, whose standard enthalpy of formation is about -350 kJ/mol (and of combustion about -7900, of which about 400 are water condensing). 22 hydrogens are enough to fully saturate 10 carbons, producing decane, $C_{10}H_{22}$, with a standard enthalpy of formation of -300 kJ/mol.

Bicyclohexyl is fully saturated as well and is $C_{12}H_{22}$, -273 kJ/mol enthalpy of formation (and, of combustion, -7600). 1-dodecylene, with a single unsaturated bond, is $C_{12}H_{24}$, thus needing a couple of extra hydrogens, with -165 kJ/mol enthalpy of formation, thus being considerably less stable. 1-dodecyne is also a $C_{12}H_{22}$, but is more exotic; though Sigma-Aldrich will sell it to you, giving data like its boiling point (215°) and density; they don't include thermodynamic data, but I'd guess it's even less stable. 1,9-decadiene exists (anyway Sigma will sell it to you and there are papers about using it) and both (e, Z)-2,4-dodecadiene ($C_{12}H_{22}$) and 2,4-dodecadiene ($C_{10}H_{18}$) are found in NIH's data; the former "has primarily been detected in saliva" (!!) but no thermodynamic data is available. No alkadienes higher than 1,7-octadiene have Wikipedia pages.

Suppose each of the monosaccharides decomposed separately, though? We might end up with $2C_6H_{11}$, which doesn't seem to exist, or $C_6H_{10} + C_6H_{12}$. Cyclohexene (-40 kJ/mol Δf liquid) or 1,5-hexadiene (+50 to 100 kJ/mol Δf) are C_6H_{10}, while hexene (most common isomer, 1-hexene, -74 kJ/mol Δf) and cyclohexane (-156 kJ/mol Δf) are C_6H_{12}. None of these seem super appealing especially compared to decane.

Including CO_2

So suppose you were to generate $2CO_2$ (-393.5 kJ/mol each), sucking up four of your 11 oxygens, leaving 7 oxygens and $C_{10}H_{22}$, decane, at -300 kJ/mol, for a total of -489.0 kJ per mole of sucrose. I guess you'd need 1734.2 kJ/mol to make that enthalpically favorable? That's 247.7 kJ per mole of oxygen *atoms*. Water's enthalpy of formation is -285.83 ±0.04 kJ/mol, which would seem to suggest that you should be able to burn hydrogen with sucrose as an oxygen source, barely; the trouble with this is that it would imply that you could get energy by taking the oxygen out of sucrose, leaving its carbons and hydrogen behind, and pairing it with new hydrogen from the environment. This seems fishy to me.

Trying to understand bond energies

The https://en.wikipedia.org/wiki/Bond-dissociation_energy of O-H bonds is typically on the order of 440 kJ/mol (e.g., in methanol), but 497 kJ/mol in water and only 360–380 kJ/mol in phenol; C-H bonds are typically around 420 kJ/mol (e.g., in ethane, or 470 in benzene); and H-H bonds are 436 kJ/mol. So in the micro-reaction H-C-O-H + H_2 → H-C-H + H_2O, we are ripping apart an H-H bond and a C-O bond, and forming a C-H bond and an O-H bond in the resulting water. WC claims the C-H bond is 411 kJ/mol and the C-O bond is 358 kJ/mol, though those numbers are suspiciously exact for numbers outside of context — WP gives a 360–380 kJ/mol range for some examples of the latter, for example, and 372–556 for the former; another source, though, gives the C-O bond energy in glucose as precisely the 358 kJ/mol given by WC, and the C-H energy as 414. So roughly we should expect to get (414 + 497 - 436 - 358 = 117) kJ/mol of hydrogen. This is almost three times higher than the discrepancy in the previous paragraph (each mole of hydrogen produces a mole of oxygen) but in the same direction.

(I think the discrepancy is easily explained: it comes from the extra four hydrogens glomming onto the carbon backbone and releasing their own 400 or so kJ/mol, releasing more oxygen to react with the extra hydrogen.)

However, if this is correct, I think it *doesn't* imply that the dehydration of sucrose, for example by heating, should be exothermic — there we are breaking the H-C bond and the C-O bond and forming an H-O bond instead, for a total of (497 - 414 - 358 = -275) kJ/mol. But I think that actually when this is done with vitriol instead of heating it *is* exothermic, and the vitriol is a mere catalyst; I'm not sure if this is correct. Supposedly the spontaneous dehydration should be exothermic.

Aha, Darius Bacon points out that I'm not accounting for the carbon finding a lower-energy state afterwards, where those carbon valences are connected to something else — which is amusing, since that's what the first few paragraphs of this very note are about. So if that carbon goes and bonds to one other carbon from some other molecule on each side, we gain like 350–380 kJ/mol per carbon (times two bonds, but divided by two carbons per bond) which puts us back in exothermic territory by like 100 kJ/mol, close to the hydrogen. (And indeed the page linked above says it's due to the formation of graphene that the reaction is exothermic.)

(There's also a double-bonded oxygen in sugars which I'm ignoring; in monosaccharides its bond energy is like 800 kJ/mol, and in sucrose it glues the two monosaccharides together with an ether bond. Hydrogen might not liberate it, though above some temperature that would be entropically favorable.)

Hydrogen vs. other reducing agents

Hydrogen wouldn't need to generate CO_2 to comfortably liberate oxygen from sugars the way I discussed earlier, because it can saturate the carbons just fine on its own. Other reducing agents seeking to oxidize themselves from sucrose might need to, thus gaining only 7 oxygens from the sucrose instead of the 10 or 11 gained by hydrogen;

so their oxidation products would need to have a more negative enthalpy of formation than water's -285 kJ/mol of oxygen atoms.

So for non-hydrogen reducing agents, we get 7 moles of O per mole of sucrose (342.30 g/mol), which works out to 112 g of O per 342 g of sucrose, 33% oxygen by weight. We might get even more if the reducing agent can reduce CO_2, but at a higher enthalpy cost. If the reducing agent can't reduce at least H_2O, it probably won't be able to reduce sucrose either.

Sucrose caramelizes at 186°, so if you want to reduce sucrose rather than water, you'd better do it before that temperature. Other polysaccharides such as cellulose or chitin may survive to higher temperatures, and things that can reduce sucrose can probably reduce them too.

Topics

Wang tile chemicals

Kragen Javier Sitaker, 02020-09-30 (updated 02020-12-31)
(2 minutes)

Wang tiles are one of the simplest Turing-complete systems. You have some set of square tiles of a single size and the task of tiling, say, the infinite two-dimensional plane with them; the restriction that makes this difficult is that the tile edges are colored with some finite set of colors, and the colors of the adjacent edges of contacting tiles must match. In Hao Wang's original proposal the tiles were forbidden to rotate. It's straighforward to see how you can translate, say, binary addition or a Turing machine into this formalism; moreover it can be deterministic or nondeterministic.

This is a handy way to generate things like random game boards, but I was thinking of a different application.

What if each tile type is a type of molecule? Molecules can be highly selective about what kind of reaction sites they bind to, and modern organic chemistry is able to perform quite sophisticated syntheses. You can of course have molecules that bind together in three dimensions rather than two, or rotate, but that additional power is not necessary in this case, as long as you can keep them from glomming together in undesired ways too much. (It might improve efficiency, though.)

This could allow you to self-assemble a massively parallel computation on a solid substrate. It might be desirable to use only one molecule type at a time to keep them from binding together in the solution rather than on the substrate. This sequence of reagents itself can constitute an input stream being processed massively in parallel, but for the basic Wang-tile abstraction, you just need to cycle through all the reagents repeatedly enough times.

(This turns out to have been Erik Winfree's 1998 PhD thesis. Thanks to Darius Bacon for the heads up!)

Topics

Scraping Sciencemadness

Kragen Javier Sitaker, 02020-10-01 (updated 02020-10-05)
(4 minutes)

I want to snarf some of sciencemadness before it goes down, such as:

http://www.sciencemadness.org/talk/viewthread.php?tid=1245&page=2

Initial look at URL patterns

That thread is in forum 2 http://www.sciencemadness.org/talk/forumdisplay.php?fid=2, which has 216 pages such as http://www.sciencemadness.org/talk/forumdisplay.php?fid=2&page=3. They link to pages like http://www.sciencemadness.org/talk/viewthread.php?tid=156102 which may themselves have page numbers. Sometimes they link to attachments like http://www.sciencemadness.org/talk/files.php?pid=643614&aid=820955 and include images like http://www.sciencemadness.org/talk/images/xpblue/default_icon.gif.

There's the risk that a thread in that forum might link to a thread in another forum, and then another, etc., but I think that mostly won't happen.

First stab at crawling

So some regexps would be something like

```
http://www\.sciencemadness\.org/talk/forumdisplay\.php\?fid=2(?:&page=\d+)?
http://www\.sciencemadness\.org/talk/viewthread\.php\?tid=\d+(?:&page=\d+)?
http://www\.sciencemadness\.org/talk/files\.php\?pid=\d+&aid=\d+
http://www\.sciencemadness\.org/talk/images/.*
```

So I think the command is something like this:

```
time wget -r -l inf -np --regex-type pcre -w 17 --retry-connrefused \

  --accept-regex 'http://www\.sciencemadness\.org/talk/(?:images/.*|files\.php\?pid=\d+&aid=\d+|viewthread\.php\?tid=\d+|forumdisplay\.php\?fid=2(&page=\d+))' \
  http://www.sciencemadness.org/talk/forumdisplay.php?fid=2
```

I can't use `-N` because the messageboard doesn't provide Last-Modified. `-nc` doesn't do the right thing because wget doesn't know to reparse the on-disk forum indexes. I have to use `-l inf` because the default is 5.

Happily they do have a robots.txt that implicitly allows this kind of mirroring.

The above does end up with a bunch of duplicates:

```
www.sciencemadness.org/talk/viewthread.php?tid=27851&goto=search&pid=310821
www.sciencemadness.org/talk/viewthread.php?tid=27851&goto=search&pid=310836
```

etc. So far this is only a minor irritant, a tarpit that has sucked up 21 out of the 190 files I've snarfed so far on a single thread; wget isn't smart enough to notice that these are all just redirects to things like

```
http://www.sciencemadness.org/talk/viewthread.php?tid=27851#pid310836
```

This suggests that maybe the regexp is not being required to match the whole URL, just the beginning (or maybe anywhere). Also I hadn't allowed the &page= on the viewthread regexp at the time, so I guess it's pretty certain.

A second attempted crawl

All right, trying again; seems to be working better now:

```
time wget -r -l inf -np --regex-type pcre -w 17 --retry-connrefused \

  --accept-regex 'http://www\.sciencemadness\.org/talk/(?:images/.*|files\.php\?p
oid=\d+&aid=\d+|viewthread\.php\?tid=\d+(?:&page=\d+)?|forumdisplay\.php\?fid=2(?:o
o&page=\d+)?)$' \
  http://www.sciencemadness.org/talk/forumdisplay.php?fid=2
```

(Apologies for the poorly formatted regexp. Probably `(?x:...)` formatting across lines would have been a good idea...)

Initially I tried it with `-w 1.7` until I was sure I'd fixed *that* problem. Now, half a gig later, it seems to be doing okay, though some images have been uploaded twice. Maybe `--page-requisites` would be a good idea but I don't know how it interacts with `--accept-regex`. Maybe also `-k --adjust-extension` would also be useful.

After 20-some hours this seems to be doing okay with something like 1200 thread pages in 700 threads and 3000 attachments successfully downloaded, totaling 1.1 GB:

```
while :; do
    echo "$(ls talk/|grep -Po 'tid=\d+'|sort -u | wc -l)" \
         "$(ls talk/|grep -Po 'tid=\d+(?:&page=\d+)?'|sort -u | wc -l)" \
         "$(ls talk/|grep -Po 'aid=\d+'|sort -u | wc -l)"
    sleep 10m
done
```

There are a few cases where the same file is downloaded under two different attachment IDs, resulting in some bloat, but it seems to be a minority of the total.

The pagination of the forum goes up to page 216, and I think it's 30 threads per page, suggesting that the total number of threads is a bit under 6500, and so I'm something like 11% done. (If so, I'm going to run out of space on this disk.)

Aha, in fact it says on the front page of the forum: 98022 posts in 6460 threads ("topics"). Total stats: 36333 topics, 497573 posts, 288119 members. So the forum I'm snarfing is about 20% of the total

number of posts, and I'm about 10% or 15% done with it.

I was missing this file:

```
wget -x http://www.sciencemadness.org/talk/js/header.js
```

And this directory:

```
wget -r -w 21 -np http://www.sciencemadness.org/scipics/
```

...which turns out to have a lot of interesting stuff in it. And actually the default `-l 5` wasn't enough, snarfing only 499 MB in 2249 files.

After another day I'm up to 1412 threads, 2236 pages, and 6201 attachments, 2.1 gigabytes; one quarter done with this forum.

After another day my netbook crashed, and wget can't recover, so I need to find a better way to spider the site.

Topics

Secure Scuttlebutt is a cool idea whose realization has fatal flaws

Kragen Javier Sitaker, 02020-10-02 (updated 02020-11-06) (17 minutes)

Some notes from reading the Secure Scuttlebutt protocol specification. I'm coming into this with some prejudices since what I've heard is that SSB is a pretty good design, similar to some things I've been toying with for some time, but with some fatal flaws that make interoperability very difficult (which turns out to be sort of true, due to problems with JSON canonicalization). So I'm looking at it with more of an eye to implementing a similar but incompatible protocol than with an eye to implementing SSB itself.

My overall sense is that the protocol seems pretty sloppy, although probably workable for its intended purpose.

The formatting of the document seems okay but I'm not wild about essential parts of it being represented as PNGs.

Ed25519 keys as identities is reasonable; might be better to use hashes of them. See below.

Broadcasting discovery packets once a second ("advertising") seems excessive, though maybe that's because I've been thinking a lot about low-bandwidth networks like FidoNet, LoRa, and shortwave radio. Maybe it would be better to broadcast on the order of once every 256 seconds, after detecting a network change, or after seeing an advertisement for a previously unknown peer. The advertisement packet format seems to be about 50 bytes, so the advertisements alone suck up 400 bits per second per peer; ten such peers would render a 4-kilobit-per-second channel useless. It would take fewer peers to saturate the same connection if framing and lower-level headers also consumed bandwidth.

I don't think the terminological distinction between "identity" (sometimes called "feed ID") and "feed" pulls its weight.

I like the terminology of "feed" for "all the messages posted by that identity", and I like the one-dimensional URL-like serialization of @keys, %messages, and &blobs. I'm not sure it's worth it to make them *not* URLs — particularly since keys, the protocol element it's most important to keep lightweight (for example because of their use in invite codes), include a verbose namespace identifier ".ed25519" at the end! (However, I'm not a fan of the alternative JSON representation of the discovery packet used in the pub message format, where, instead of being concatenated into a single string, the host, port, and feed ID/identity are stored in the three properties "host", "port", and — can you guess? "feed"? "id"? "identity"? — no, "key", of a JSON "object".)

To elaborate, an Ed25519 key is 32 bytes. Encoded in base64 it's 44 bytes, for 12 bytes of encoding overhead, and the ".ed25519" at the end adds another 8 bytes. Ed25519 is not quantum-cryptography-resistant and birthday attacks are not useful here, so if we are willing to accept (classical) brute-force resistance of

only 2^{119}, we need only use 120 bits of public key hash as an identifier, or 15 bytes; this base64-encodes to 20 bytes. If, at some future time, non-Ed25519 keys are desired, they can be signified by beginning their representation with one of the 31 printable ASCII characters that are not valid Base64 — namely, space or any of the punctuation except for '=', '+', or '/'. Thus, instead of the representation 'FCX/tsDLpubCPKKfIrw4gc+SQkHcaD17s7GI6i/ziWY=' specified in SSB, we can b64decode that, sha256 it twice, take the first 15 bytes, and b64encode the result:

```
>>> k0 = base64.b64decode('FCX/tsDLpubCPKKfIrw4gc+SQkHcaD17s7GI6i/ziWY=')
>>> sha256 = lambda s: hashlib.sha256(s).digest()
>>> base64.b64encode(sha256(sha256(k0))[:15])
'c71K2cieLixgzReT8TkT'
```

I argue that '@c71K2cieLixgzReT8TkT' is a more reasonable feed ID than '@FCX/tsDLpubCPKKfIrw4gc+SQkHcaD17s7GI6i/ziWY=.ed25519', and a classical (non-quantum) brute-force attack on it still requires an expected 2^{119} hashing operations, about 21 quintillion machine-years at one hashing operation per machine-nanosecond. The drawback is that in order to actually validate any signatures you need to somehow obtain the whole 255-bit public key, so the protocol needs to have some way for you to get it. I think this is probably a good tradeoff.

You could imagine the full URL for a peer owning that key might be something like `p9://one.butt.nz:8008/@c71K2cieLixgzReT8TkT` or `p9://138.68.8.185:8008/@c71K2cieLixgzReT8TkT`, which is noticeably shorter than *just the key* in SSB's format, let alone the whole discovery packet, which I am mostly retyping here because in the original document it's in PNG:

```
net:192.168.1.123:8008~shs:FCX/tsDLpubCPKKfIrw4gc+SQkHcaD17s7GI6i/ziWY=
```

So for example the URL can be encoded as a version 3 QR code (29×29), while the SSB discovery packet requires version 4 (33×33). The "invite code" explained later on is a total fail; the example is `one.butt.nz:8008:@VJM7w1W19ZsKmG2KnfaoKIM66BRoreEkzaVm/J//wl8=.ed25519~r4hIBk7KC07a9Gknj7Qiuuo4+Et/TS2rjgl6gYgw3OIM=`, which I also retyped because it's also in a PNG. This requires a version 6 QR code (41×41). See later for invite codes.

If there's some kind of advertisement service where you can publish your current IP:port, then it might be adequate to say `p9:@c71K2cieLixgzReT8TkT`, which is a version 2 QR code (25×25), 43% smaller than the SSB discovery packet above.

What's with this `p9` idea for an URL scheme? Well, shorter is better, as long as it doesn't pose too much risk of a collision. There are only nine two-character URL schemes in Wikipedia's list, one of which is the `ni:` scheme for naming data by hash; the W3C also lists `bk:` and `kn:`, one of which was partly my fault. IANA lists provisional registrations for `qb:` — and `ssb:`! Also, only ten of IANA's list of 335 schemes (provisional and otherwise) use digits. So the risk of an URL scheme clash is quite low. "p" is for "prate (p. 367)", and 9 is an

auspicious number signifying permanence in Chinese. (You might want to use the "URL-safe" variant of base64 in which - and _ are used instead of / and +; in that way you could still interpret / as a path separator in the usual way.) On the downside, "p9" is Chromium's and Microsoft WSL's server implementation of the 9P protocol, so it might not be a good name for a protocol identifier.

("p8", as in "prate", might be better.)

A thing I'm not entirely sure about is whether you'd want to use the same *identity* to talk about two different *topics*. Evidently, for example, you use a "long term public key" not only to identify a feed (and presumably sign messages or blocks of messages on the feed, though I haven't gotten that far yet) but also to authenticate incoming connections as a server and to authenticate yourself on outgoing connections as a client. I think it might be worthwhile to separate these functions. (Also, does the key-exchange protocol support "name-based virtual hosting"?)

I'm not entirely sanguine that Dominic Tarr has evidently invented his own key exchange protocol, but it does purport to provide security properties that aren't present in the other key exchange protocols I know about. However, I am not hip to the state of the art in key exchange protocols, and never have been, actually. I wonder how well it protects against DoS attacks. The fact that it's built on NaCl is promising but of course not a guarantee against misuse.

The box stream protocol headers are rather bulky at 34 bytes. At 300 baud N81 that's over a second for just the header. For an 8-byte payload that works out to 425% overhead. However, it may not be feasible to offer the cryptographically strong authentication purportedly offered by this protocol at a significantly lower cost than that.

By contrast to the box stream protocol, the rather bulky feed-ID/public-key/identity serialization, and the JSON RPC protocol *body*, the RPC protocol *header* is bummed to within an inch of its life, with five bit fields in one byte — which dealigns the following fixed-size binary numerical fields.

The SSB protocol doc isn't always clear about the usual must/should distinction; at one point, for example, it says, "JSON messages don't have indentation or whitespace when sent over the wire." Does that mean we should reject JSON messages containing whitespace? What if it's inside a string? In another case, it says, "Because this is the first RPC request, the request number is 1." What should you do if the first RPC request you get on a stream is numbered 0, as any sane person would do, instead of 1?

I'm not sure "createHistoryStream" is a good name for "subscribe". I mean it sounds more like "publish" than "subscribe".

The first example RPC response seems like it might have a typo:

```
"key": "%XphMUkWQtomKjXQvFGfsGYpt69sgEY7Y4Vou9cEuJho=.sha256",
```

I thought previously we said "%" was for messages, not keys? Also, this seems to be a hash, not a key. (And as such it could use the `ni:` URL scheme mentioned earlier, or for that matter `magnet:`. Would those be better?)

And we start to see the difficulty with the message signature scheme: we have a JSON structure under the name "value", then what is purported to be an Ed25519 signature. But Ed25519 cannot sign JSON; it signs blobs. Perhaps later we will see how this is resolved.

The pairing of request 1 with response -1, request 2 with response -2, and so on, is a bit goofy. And the name "RPC" doesn't seem entirely apt. But these are minor details.

A perhaps more serious issue — for some applications, anyway — is that, when you request the messages from a feed, you apparently have no idea if you are going to receive 150 bytes or 150 megabytes of response, and no way to stem the flood if it overwhelms you, other than the usual TCP mechanisms, assuming you're speaking over TCP. (A mechanism is given "to abort a stream before it is finished" but it's of the XON/XOFF flavor, not the ENQ/ACK or TCP/ZMODEM sliding-window type, so a FIFO in the system will totally defeat it. Later on we do see that `createHistoryStream` has a `limit` option, but it's a limit on the number of messages, not a number of bytes.)

I think the partitioning between messages and blobs is probably a good idea for many purposes, though the example messages given for the first few requests don't refer to any blobs; they just have text bodies. So the introduction of *blobs.has* (or, as it's spelled elsewhere, `["blobs","has"]`) is a bit startling.

I worry a little bit about the potential information leaks associated with *blobs.has*. Should I consider the set of blobs that my Scuttlebutt client *has* to be public information, or at any rate visible to everyone I connect to? Might having a blob indirectly reveal something I consider private?

As I've mentioned previously in personal communications, I don't think there's any benefit in a feed like the Scuttlebutt feed to including the hash of the previous message in each message. Including such hashes in general cannot defend against message *blocking*: if nobody is willing to give you Alice's message #4, then the fact that you know its hash from reading Alice's message #5 does not in itself help you to find out message #4's content. Rather, including such hashes is designed to prevent attackers from *silently* blocking messages (without blocking all following messages) and from *altering* messages. But Alice's signature on message #4, if calculated over something that includes the sequence number 4, already defends against those two attacks.

Specifying within the message content the hash algorithm to be used to sign the message also seems like a bad idea to me, although a relatively harmless one — as with the link to the previous message, it only wastes a little bandwidth.

The `["Message format", "signature"]` section finally explains the JSON canonical serialization used for signatures, which is kind of terrible; it refers to the ECMA-262 6th-edition spec for `JSON.stringify`! And it's a different JSON serialization from the one used on the wire, because it contains mandatory whitespace. Moreover, it entirely fails to mention the order of dictionary keys, and the example message in the SSB document to which it refers does not have the keys sorted in any

discernible order. ECMA-262 does not appear to specify the dictionary key order either: stringify calls SerializeJSONProperty which calls SerializeJSONObject which calls EnumerableOwnNames which uses the ordering of the [[Enumerate]] internal method which, in the 6th edition, explicitly says "order of enumerating the properties is not specified", although, as I understand it, the committee is going to standardize that iteration happens in insertion order, or has already done so.

Later, contradicting what the "Signature" section says, the "Message ID" section says, "Like with signatures, dictionary keys must appear in the same order that you received them." This is at least an implementable specification, although it precludes the use of most JSON libraries.

Basically this is the same design error as XML canonicalization and ASN.1 DER, only botched. If you google "How Not To Sign JSON" this is literally what you will find.

On the `createHistoryStream` semantics, I think six options is too many. The basic request ought to be "please send me messages on feed X, starting from sequence number Y, up to a limit of Z bytes," and the response ought to indicate when all the known messages have been sent and the peer is now waiting for new messages to forward on to you. There's no need to have the option not to know when you're up to date (it costs one bit on each message at worst), no need to have other ways to fetch messages, no need for an option to omit messages that arrived at the peer before you sent your subscribe, and no need for the peer to tell you when it received messages, much less an option to turn that behavior off.

I do think the decision to handle feeds and blobs separately is good. In fact, I think that in many cases they should be even more separate than they are in SSB. The lack of this separation is one of the weaknesses in Van Jacobson's CCN.

Blob IDs do not need to be dozens of bytes long (again, presuming no quantum cryptanalysis, 120 bits should be fine), and there's no need to have separate get and getSlice procedures ("methods"!?), nor multiple argument formats. A single request to fetch up to X bytes from blob Y starting from byte offset Z, whose response includes the actual blob size, is sufficient; if you want the whole blob then you can just set X to be very large, or, after the first request, use the actual remaining size of the blob.

A better hashing scheme such as BLAKE3 would enable the verification of partial blobs (down to 1 KiB) as well as entire blobs, with a little bit of extra metadata. (BLAKE3 is also about 30 times as fast as SHA-256 on modern multicore SIMD hardware.) I'm not sure exactly how truncating BLAKE3 hashes affects its security.

I need to come back and read the wants/haves stuff more later, except that I really don't like the name "createWants". It seems like a reimplementation of CCN, inheriting some of its weaknesses.

In a lot of cases I would like to store my blobs in a totally different system from the gossip system used for pub/sub broadcasting. For example, I might want to use a DHT to assign blobs to blob servers, rather than replicating them to every peer.

I think "feeds can follow other feeds" is... not a useful idea. But it's entirely separable from the rest of the protocol.

"Invite codes" serve a couple of purposes, but I think they can be served more easily. I'll have to read the invite-code stuff later.

Posting private messages on your regular feed, but encrypted, is probably a reasonable thing to do, but in other cases it would be more useful to create a feed for a particular person-to-person conduit. However, encrypting messages and then posting the ciphertext as base64 strings in JSON is a stupid thing to do. It's like pre-yEnc alt.binaries Usenet.

Topics

Prate thoughts

Kragen Javier Sitaker, 02020-10-02 (updated 02020-12-30)
(12 minutes)

So I just looked at Secure Scuttlebutt (p. 361) and I'm trying to figure out what would be better. Layering would be better.

The secure gossip protocol

The secure gossip protocol in Prate provides a peer-to-peer reliable authenticated publish-and-subscribe service to higher-level protocols.

A *journal* consists of a *public key* and an append-only sequence of sequentially numbered *pages*, each of which is *sealed* with a signature from the *private key* corresponding to the public key. Each page consists of a *header*, a delimited sequence of bytes interpreted as the page number and an arbitrary set of name-value pairs; a *body*, which is an arbitrary sequence of any number of bytes; and the seal, which is a cryptographic signature of the concatenation of the header and the body.

When two *peers* are talking about a journal --- later we shall discuss how this may come to pass --- which is identified by a *hash* of the journal's public key, they can ask one question:

• Please send me pages N and up from journal X, up to a maximum of M bytes.

The polite response to this question takes of one of the following forms:

• I can't or would prefer not to.
• I know pages numbered up to N' from journal X, whose public key is Y. Page N consists of 301 bytes: <....>. Page N+1 consists of 1820 bytes: <...>. Page N+2 consists of 238332 bytes.

The positive response may include the contents of zero or more pages. It includes the full public key, since the public keys use Ed25519 and are therefore only 32 bytes, which is compact enough to always send. The pages may be sent immediately or not until later; indeed, they may not exist at the time they are requested. Consequently N' may be less than the maximum page number sent.

The final information sent for a positive response is, at times, the size of a page that was not sent because it was too large to fit within the byte limit M.

Holders of private keys must ensure that they never seal two distinct pages in the same journal with the same page number, and that they never seal a page with a page number less than an already-sealed page on the same journal; in this way journals must remain append-only. They should start numbering the pages in each journal at 0 and number them sequentially without skipping any numbers. Peers must never send pages that are not sealed, whose seal is invalid, or that have not been requested, either by preceding N or by causing the total responses to a request to exceed M. They must never send a public key that does not have the correct hash, either.

As in BitTorrent, it does not matter at all who the two peers are, because any information published unencrypted in a journal is assumed to be public, so sending it to any peer is okay; and, because the pages in the journal are sealed, they authenticate themselves, so it is equally valid regardless of who you got it from.

Peers can freely choose whatever retention policy they want for journal pages, as well as choosing when and whether to make or fulfill requests. Of course if they do not have a copy of page N, they cannot send it when requested, but in this case they may send page N+1, N+2, etc., as long as they fit within the byte limit. However, if they do send any pages, they must send the lowest-numbered pages they have, or (if they do not fit in M) their byte count. They may not choose, for example, to send page N+1 when they could have sent page N, or page N+2 when they could have sent page N+1.

Again, as in BitTorrent, you might choose which peers and journals to devote your resources to based on your past interactions with them and/or their identities. For example, if a peer asks you for pages from journal X, you might ask them for pages from the same journal, especially if you couldn't satisfy their request. And if they do satisfy your requests, you might prioritize satisfying their requests in the future, perhaps even subscribing to journals you aren't really interested in, but that they have expressed interest in. As another example, you might send requests for pages optimistically to peers who you have no real reason to think can satisfy them.

One uncertainty: although sealed page numbers ensure that malicious third parties cannot alter the history of an existing journal except by unanimous replay attacks (a form of censorship), they do not ensure that the legitimate author only publishes pages in order. Including the hash of the previous page in each new page, like Secure Scuttlebutt, Bitcoin, and Git, would prevent the author from publishing pages out of order. Does this matter?

Journals, topics, and identities

An *identity* is an agent in a distributed system, such as a human or a running program. A *topic* is a set of messages an identity might want to subscribe to. Prate's gossip protocol, described above, does not directly provide the ability to subscribe to or "follow" topics, which is surprising because it is claimed to provide pub-sub. It only provides the ability to subscribe to journals, which is implemented by asking other peers to send you pages from them.

Generally there is not a one-to-one relationship between journals and topics, topics and identities, or identities and journals. We can implement both identities and topics to a significant extent as groups of journals. To subscribe to a topic, you somehow obtain a list of journals that belong to it, then subscribe to all those journals. To publish to a topic, you create a new journal for your publications on that topic, then somehow try to advertise your journal so that others will subscribe to it.

Journals cannot usually be shared between identities, because, as explained above, holders of private keys must ensure that they never seal two distinct pages in the same journal with the same page number. This means that, for example, if a person wants to publish

both from their laptop and from their cellphone, they will need to create one journal on each, unless they are willing to take on the obligation of assigning page numbers manually. Otherwise, the possibility exists that, while their cellphone is in airplane mode, they will seal a page on their laptop, save it on their pendrive, accidentally run over the laptop with their pickup, then seal another page on their cellphone with the same page number and promulgate it, then later promulgate the doppelganger page on their pendrive from a different laptop. This possibility is clearly intolerable, so they should either use two separate journals or keep the private key on a centralized server that both the laptop and cellphone use.

Signaling, advertising, discovery, spam, and denial of service

How does the new kid on the block make her first acquaintance? Once this is done, the acquaintance can introduce her to others as per the Granovetter diagram, who can introduce her to still others, progressively widening her circle of acquaintances. But how can the progress get started?

For example, suppose there's a well-known journal (call it Factsheet 9) that periodically publishes new lists of journals that publish on particular topics. If you want to subscribe to news about wildfires, you can subscribe to Factsheet 9, peruse its past pages for announcements of wildfire journals, then subscribe to all those wildfire journals. Similarly for Google outages, time zone changes, or new erotic fiction. But if you want to *publish* news about a wildfire, somehow you must persuade Factsheet 9 to list your new wildfire journal. How can you establish contact as a complete unknown?

Whatever solution is adopted to this problem, allowing complete unknowns to establish initial contacts, is vulnerable to Sybil attacks and spam, and so it cannot be considered reliable. But that does not mean that no solution exists.

Alternative terminology

I've considered a number of alternative terms for "journal" and "page". Perhaps "journal" should be "feed", "stream", "channel", "ledger", "scroll", "book", "codex", "file", "hair", "thread", "tune", "battery", "log", "dynasty", or "chain", while perhaps "page" (the unit of committing to a journal) should be "transaction", "line", "drop", "entry", "scrap", "chapter", "cell", "verse", "slice", "parcel", "morsel", "packet", "commit", "king", "block", or "link". So we might say we append commits to a log, or lines to a file, drops to a stream, or entries to a ledger, or kings to a dynasty, rather than pages to a journal.

"Page" has the misleading connotations of mutability and a fixed size. "Log", "journal", and "ledger", and to a lesser extent "feed", have the right append-only connotation.

"Appending lines to a file" or "to a log" sounds reassuringly low-tech and helpfully connotes variable-sized-ness, but misleadingly connotes a size closer to 64 bytes than, say, 2048, which I think is more likely in the sweet spot. It also misleadingly connotes plain text,

and it might lead to confusion when we're trying to talk about the implementation: "What do you mean, the file is stored in several files?"

Secure Scuttlebutt (p. 361) uses "feed" and "message", and following that convention might help comprehensibility for people who know SSB. Kafka (p. 52) uses "topic partition" (described as an "ordered 'commit log[]'", leading me to favor "log" and "commit") and "event".

Detached and batched signatures

As Perry Lorier pointed out when I brought it up, it might be useful to separate the signature from the page, so that a single signature-verification operation is sufficient to validate all pages up to a given point in the journal. Moreover, a Merkle chain over the pages permits verifying the whole journal up to that point with some degree of independence from actually reading the pages.

That is, given a signing operation S_k(hashval) and a hashing operation H(data), when you author a page P_i, you can compute a_i = H(P_i), b_i = H("cons" || b_{i-1} || a_i) and s_i = S_k(b_i), and distribute all three of them. (We can take b_0 to be some convenient public value such as "".) Someone who wants to verify the journal signature up to i needs only s_i and all the a_j and b_j for $j \leq i$, which is (for many signing algorithms) considerably more compact than a signature per page and also faster to verify. This then allows them to verify particular page data if and when they actually get the pages.

Thanks

To Perry Lorier for a very helpful discussion.

Topics

- Systems architecture (p. 809) (12 notes)
- Security (p. 811) (11 notes)
- Protocols (p. 813) (10 notes)
- Merkle graphs (p. 885) (3 notes)
- Publish/subscribe feeds (p. 889) (3 notes)
- Chat (p. 973) (2 notes)

Lithium fuel

Kragen Javier Sitaker, 02020-10-04 (7 minutes)

Aside from lithium's well-known use in batteries (rechargeable and otherwise) it seems like it might be useful as a fuel, similar to the tamer magnesium (p. 335), or alloyed with it.

Property	Li	Li_2O	LiH	Li_2S	O	S	H_2O	SO_2
Molar mass, g/mol	6.94	29.88	7.95	45.95	15.999	32.06	18.015	64.066
Density, g/cc	0.534	2.013	0.78	1.67	.001429	2.07	1.00	.00263
Enthalpy of formation, kJ/mol	0	-595.8	-90.65	-447	0	0	-285.83	-296.81
", MJ/kg	0	-20.01	-11.4	-9.401?	0	0	-15.87	-4.63
Heat capacity (room temp) J/mol/K	24.860	54.1	27.9	??	29.378	22.75	75.39	39.87
Specific heat, J/g/K	3.58	1.81	3.51	??	0.92	0.71	4.184	0.622
Extrapolated ΔT, °C	0	11000	3200	??	0	0	3800	7400
Melting point, °C	181	1438	689	938	-219	115	0	-72
Boiling point, °C	1330	2600	900	1372	-183	445	100	-10

The extrapolated temperature changes here are what the temperature would be in forming the compound from the elements, if its specific heat stayed the same as that at room temperature and it underwent no phase transitions. But of course specific heats typically rise with temperature (at 1200 K SO_2's heat capacity is 55.81 J/mol/K) and several of these materials do in fact undergo phase transitions.

The lithium combustion equation should be $4Li + O_2 \rightarrow 2Li_2O$, producing 1192 kJ per mole of O_2 and per four moles of Li, thus 298 kJ per mole of Li; that's 42.9 MJ/kg, quite impressive, almost twice magnesium's 24.7 MJ/kg. The energy density then would be 22.9 MJ/ℓ, lower than magnesium's 43 MJ/ℓ.

So it turns out that lithium as a fuel burned with oxygen has a specific energy even higher than magnesium, though much lower density. Because of that and because magnesia's specific heat is higher than lithia's, burning lithium probably will reach lower temperatures.

At 2%, magnesium is much more abundant in Earth's crust than lithium — despite lithium's name, it's only found at 20 ppm. This is a disadvantage for lithium as an energy carrier. Aside from lithium's higher specific energy, though, it also melts at a reasonable temperature which might make it possible to use as a liquid fuel in some kinds of engines, though for some of them the refractory nature of lithia would be a drawback.

Lithium, like magnesium, is produced by molten-salt electrolysis of the chloride, in this case fluxed with KCl, and at a very friendly temperature of 450°. I don't know of a lithium equivalent to the Pidgeon silicothermic reduction process for magnesium.

Reacting lithium instead to the hydride or the poorly characterized sulfide would produce about half as much heat, although that's still a pretty acceptable energy density, and those reactions also produce no gases, just solid or liquid salts. It may be feasible to produce the

sulfide with a reaction between molten lithium and molten sulfur, producing molten lithium sulfide, although sulfur's tendency to polymerize around lithium's melting point may be an obstacle.

Also, though, those salts can themselves be used as fuels!

Hydride and sulfide fuels

Either of these can be burned as a fuel with oxygen to produce the other half of the heat, plus additional energy from oxidizing the anion. They may be more or less convenient than the lithium metal due to their higher stability in air (?) and melting points. The sulfide has a higher boiling point than lithium as well.

As for the hydride's stability in air, WP says that lumps of it form a protective tarnish in humid air, inhibiting further reaction, doesn't ignite "in a metal dish" (?) until heated past 200°, and is "less reactive with water" than Li, but still "highly reactive" and "reacts violently with water". Furthermore you cannot extinguish its fires with ordinary (presumably quartz) sand.

The sulfide is reported to be deliquescent in air and, like other metal sulfides, hydrolyzes to produce sulfuretted hydrogen. Unlike lithium or the hydride, it does not seem to pose an explosion risk with water, just a poison gas risk.

The hydride combustion equation would be something like $2LiH + O_2 \rightarrow Li_2O + H_2O$, yielding steam and 595.8 + 285.83 - 2×90.65 kJ per two moles of the hydride, 700 kJ, which is 350 kJ per mole of the hydride or 44 MJ/kg. This is almost the same as the specific energy of lithium itself! It's an energy density of 34 MJ/ℓ, the same as gasoline or diesel fuel, much better than lithium, and nearly as high as magnesium.

The sulfide combustion equation would be $2Li_2S + 3O_2 \rightarrow 2Li_2O + 2SO_2$, yielding SO_2 gas and (2×595.8 + 2×296.81 - 2×447) kJ per two moles of the sulfide; that's 892 kJ per two moles or 446 kJ per mole, which is 9.7 MJ/kg and 5.8 MJ/ℓ. That's still usable as a fuel, but it's at the low end of what's usable, and the corrosive gas is probably a killer drawback.

The hydride may also be interesting as a potential working fluid for high-temperature heat engines due to its large expansion when heated; it decomposes to liquid lithium and gaseous hydrogen at 900°–1000°, and at 1330° the lithium boils, I think to individual atoms rather than diatomic molecules. So each mole of hydride will, I think, produce three moles of gas. At room temperature the hydride's 0.78 g/cc is 0.098 mol/cc, 98 moles per liter, while at 1330° the 294 moles of gas produced from that liter would ideally each occupy some 132 liters, a total of nearly 39 cubic meters.

Whether the sulfide would act similarly is, I think, anybody's guess.

The borohydride of lithium has been discussed as a possible fuel, though no boron-containing fuel is in use today for excellent reasons.

There is also a metastable lithium aluminum hydride $LiAlH_4$, containing a "tetrahydroaluminumate" or "alanate" or "tetrahydridoaluminate(III)" or "alumanuide" ion; it decomposes to

lithium hydride and a different lithium aluminum hydride Li_3AlH_6. It contains nearly as much hydrogen by weight as lithium hydride itself, and is even denser at 0.917 g/cc, so it contains more hydrogen by volume.

Topics

Globoflexia

Kragen Javier Sitaker, 02020-10-05 (updated 02020-10-10)
(37 minutes)

Globoflexia, or balloon twisting, is a popular form of entertainment, especially for children; a skilled balloon twister can make an evocative, if cartoonish, sculpture of an animal or person within a few seconds to a minute. Extremely elaborate sculptures are feasible over a few hours; because the material is so light, getting all of its compressive strength from air, sculptors can easily build and manipulate sculptures far larger than themselves. Because the balloons leak, the sculptures are ephemeral, lasting at most a few days.

Much to my surprise, there's a world globoflexia conference every two years at which teams from dozens of countries compete.

What if you could use globoflexia as a medium of expression for more permanent ideas? Obviously you can photograph the sculptures, thus making images or videos of them, but the underlying three-dimensional form of the object is lost.

So I've been thinking about three different ways to do this: photogrammetry, spray foam, and papier-mâché.

Photogrammetry

If you can 3-D scan balloon sculptures into a computer, you can use them as a means for telling the computer what to do; this could be, as in Dynamicland, a real-time interactive process of shaping computations with your hands, with real-time projected feedback, or it could be more a kind of batch data-entry thing, for example for designing three-dimensional shapes for later tweaking and automated fabrication, whether at the same scale, a larger scale, or a smaller scale.

Existing photogrammetry methods do not work for balloons. But the balloons in question are not inherently algorithmically difficult: each is a well-controlled solid color, displaying gradients of intensity corresponding to local degree of stretch and illumination, with well-controlled specular highlights. The images generally only contain edges at the edges of the balloon silhouette, at wrinkles around twists, around specular highlights, and outside the balloons. These highlights give a fairly precise read on the surface angle and curvature at a particular point, as do the silhouette edges.

Moreover the balloon sculptures' shapes are themselves well-behaved: the surface at most points has a relatively smooth curvature determined mostly by the gauge pressure and the tensions in two directions. Rubber's complex pseudoelastic thermodynamic behavior is not so complex as to make this a very difficult problem.

Further information can be obtained by looking at the balloon sculpture from different angles, as is normally done in photogrammetry, thus scanning the specular highlights and silhouette contours over the surface.

Given this information, it remains to optimize a model of the balloon sculpture to account for the observed photos as parsimoniously as possible, using standard methods like finite element analysis, Markov-chain Monte Carlo, gradient descent, and genetic algorithms.

Spray foam

What if you fill the balloons with a hardening foam instead of air?

Conventional polyurethane expanding spray foam insulation has been available for decades. You spray it as a thixotropic liquid foam, which accommodates itself to the container it's in before slowly polymerizing into a light, hard, thermally and electrically insulating foam with substantial mechanical strength. Some formulations form waterproof closed-cell foams, while others form lighter-weight open-cell foams. There are formulations that ship as pairs of liquids to be mixed in a gun, for high-volume applications, and other formulations that you just squirt out of a can.

The materials that form polyurethane foams are fairly reactive until they've finished forming the foam, and that may be a fatal flaw for squirting them into rubber balloons: they may corrode the balloons and pop them before the foam has hardened.

Polyurethane is not the only possible hardening foam. Latex foam is widely used for pillows, mattresses, and theater special-effects makeup ("prosthetics"), in which last use it is typically cast in molds before curing. Protein foam is a popular dessert, both as meringue (with air whipped into it) and as gelatin foam in so-called "molecular gastronomy" or "modernist cuisine", where the gelatin gel is mixed with nitrous oxide under high pressure and low temperature, like canned whipped cream. Gelatin foam is also widely used for makeup, sometimes whipped like meringue, but sometimes foamed with non-double-acting baking powder (for example baking soda with cream of tartar) or even yeast.

There's also been a lot of work in recent years on foamed concrete, sometimes called "aircrete". This consists of portland cement, water, a surfactant (Suave shampoo is reputed to work well, though there are also specific surfactant mixes from companies like Drexel), possibly some foam stabilizers (I suspect gelatin might work well for this), and a great deal of gas. Sometimes sand is used, but rocks are never used. The original 1920s process for foaming concrete ("autoclaved aerated concrete") used aluminum powder mixed into the concrete mix. After molding the concrete was heated in an autoclave to react the aluminum with some of the lime in the concrete, thus foaming the concrete with hydrogen gas, as well as accelerating the formation of the calcium silicate hydrates that bond the concrete. The more common method nowadays is more like meringue: air is mechanically mixed into some of the water before mixing in the wetted cement.

You probably can't foam any traditional concrete with anything similar to baking powder, because the strongly basic nature of portland cement, lime, Sorel cement, and refractory calcium aluminate will destroy the baking-powder acid without producing any gas. Non-traditional concrete binders like low-alkalinity

waterglass or molten sulfur might be less corrosive, but probably also are not a realistic way to fill balloons. And I suspect that at room temperature aluminum powder will not produce hydrogen fast enough.

The strongly basic nature of these cements might also cause them to attack the balloons.

Most resins can be foamed in a way similar to polyurethane spray foam. Radio-controlled airplane hobbyists commonly mix a secret "foaming agent" from R&G into two-component epoxy to get an epoxy foam, for example. One publication on the use of polysilazane for this purpose suggests that powdered aluminum mixed with soda lye is the usual foaming agent!

Whatever foam is chosen, whether one of the above or something else, the idea is simply to fill the balloons with the foam or incipient foam rather than just air. Then you twist the balloons into the right shape, carrying the foam along with them, and leave them there until the foam has hardened. You may want to spray some kind of adhesive onto the joints, since otherwise the foam in the different segments of the balloons will only be connected together through the balloon rubber, which may not be very stable.

Papier-Mâché

An alternative, and possibly complementary, approach is to put something on the *outside* of the balloons that hardens there, forming a hollow, tubular, continuous version of the shape you have made with the balloons. The traditional material for this is strips of paper dipped in wheat paste, but there are many possible variations on the papier-mâché theme.

In addition to wrapping the balloons tightly, you can use the balloons themselves merely to form a frame over which sheets of adhesive-soaked fiber reinforcement are draped.

For the adhesive, rather than wheat paste, you could use:

- PVA glue;
- hide glue;
- silicone;
- epoxy resin and similar resin systems, if they don't attack the balloons;
- cyanoacrylate adhesive;
- plaster of Paris (thank you, Javier Candeira!);
- sodium silicate waterglass;
- slaked lime, if it doesn't attack the balloons or fibers;
- portland cement, if it doesn't attack the balloons or fibers and the color isn't a problem;
- calcium aluminate cement, if it doesn't attack the balloons or fibers, and refractoriness is desired;
- Sorel cement, if it doesn't attack the balloons or fibers, and maximal strength is desired;
- a so-called "geopolymer cement";
- latex paint;
- shellac;
- polyurethanes;

• urea-formaldehyde resin;
• phenolic resin;
• various kinds of solvent-based plastic cements such as PVC dissolved in acetone, if the solvent doesn't attack the balloons or fibers;
• constant-tack adhesives like those used in scotch tape;
• wet clay, whether simply allowed to dry or later fired;
• tar, though probably using a solvent rather than heat;
• paraffin or other waxes, if the balloons can handle their melting temperatures;
• spray foam;
• linseed oil;
• castable refractory mix;
• other adhesives;
• some mix of the above.

For the fiber reinforcement, instead of paper, you could use:

• nothing;
• heavily perforated paper;
• paper towels;
• cotton cloth, whether light like cotton tulle or heavy like canvas, and whether with a narrow weave like twill to maximize the strength of the fabric or a loose weave to ensure good adhesion between the adhesive on both sides of the fiber;
• burlap (aka Hessian), especially sisal or jute, to minimize cost and ensure good adhesion between the adhesive on both sides of the fiber;
• gauze, as in traditional plaster casts for broken bones (thank you, Javier Candeira!);
• mosquito netting;
• fiberglass cloth, as in traditional glass-reinforced polymer layups or in printed circuit boards, although if the binder is strongly basic you might need to use alkali-resistant fiberglass;
• carbon fiber;
• large flakes of mica;
• ceramic fiber like those used in refractory blankets and flocking, if resistance to high temperatures is desired (typically these fibers are mixes of mullite, alumina, zirconia, and silica);
• basalt fiber;
• webbing like that used in car seatbelts, made from nylon or other fibers;
• steel window screens;
• aluminum or fiberglass window screens, if the binder is not strongly basic;
• stainless steel cloth;
• thicker and stronger metal reinforcement such as traditional rebar tie-ups, hardware cloth, chicken wire, or expanded sheet metal;
• copper wires;
• gel-spun ultra-high-molecular-weight polyethylene fibers;
• nonwoven bargain-basement felted polyester fabric ("friselina");
• other fibers;
• some mix of the above.

If the adhesive is less flexible than the fiber reinforcement (e.g., has a higher Young's modulus), then the fiber reinforcement may just

weaken the binder instead of strengthening it, although it can produce some "strain hardening" behavior where the adhesive cracks but the fiber keeps the adhesive cracks from opening wider and thus continuing to propagate. Still, even weak fibers can hold the adhesive in position until it sets, and for some purposes the adhesive alone will be strong enough without any help from the "reinforcement".

The fiber reinforcement may have other purposes as well, other than shaping or strengthening; for example, if the adhesive is transparent, decorative or informative images can be printed on the fiber reinforcement; conductive fiber reinforcement can provide Faraday-cage protection against EMI more cheaply and flexibly than sheet metal; gold leaf or aluminum foil can provide high reflectivity; and so on.

It may be worthwhile to also include other additives in the adhesive, whether inert fillers or reactive; for example:

- pigments;
- quartz sand for extra strength;
- olivine or zircon sand for extra strength at higher temperatures;
- clay, such as bentonite, functionalized if necessary to bond well with the adhesive, in order to increase strength or decrease oxygen permeability;
- other soils, such as silt, as the lowest-cost fillers available;
- encapsulated air bubbles to reduce density, such as hollow microspheres of steel, glass, or plastic;
- vermiculite, perlite, pumice, or similar foamed minerals to reduce density;
- polystyrene foam beads or similar foamed plastic beads to reduce density;
- lead, bismuth, or steel particles to increase density;
- rubber particles to increase shock damping and reduce rigidity;
- graphite, amorphous carbon, or silicon carbide to increase electrical conductivity and/or heat resistance;
- donors of alkali metals and boron to reduce melting point and increase the thermal coefficient of expansion, for example to facilitate fire-glazing the surface afterwards — carbonates of sodium and potassium, boric acid, and borax are traditional here;
- foaming agents like baking powder;
- plasticizers like phthalate esters;
- chopped fibers, for example of basalt fiber or any of the other types mentioned earlier, or other fibers such as hair clippings, horsehair, paper fibers as in ordinary papier-mâché, sawdust or other wood fibers, grass clippings, used yerba mate, bamboo fibers, or straw;
- broken glass, for example for decorative purposes;
- abrasives, such as aluminum oxide or silicon carbide;
- pesticides such as copper chloride, blue vitriol, salt, or clove oil to prevent biodegradation, for example by insects eating wheat paste and cotton fibers;
- UV blockers such as titanium dioxide to prevent photodegradation;
- additives to increase effective heat capacity, such as microencapsulated phase-change materials;
- milled mica, stainless steel, aluminum powder, or other glitters for a sparkly metallic appearance;

• catalysts;
• sodium polyacrylate or similar hygroscopic polymers to make the surface hygroscopic or cause changes in shape with environmental humidity;
• other fillers and additives;
• some mix of the above.

In some cases you will want to start with a lightweight, fast-hardening system such as gauze and plaster of Paris, then overlay it with a heavier system that the balloons alone wouldn't be able to support. There are many other reasons you might want to use multiple layers, including making sandwich panels with a light, weak inner core and stronger faces, and allowing earlier layers time to dry.

In cases where the first layer has no fiber reinforcement, it might be useful for that first layer to be sprayed onto the balloons rather than placed there by hand. This would allow it to be applied more rapidly and easily, and it could perhaps be strong enough once hardened to support significantly more weight than the balloons themselves. Spray foam seems particularly appealing for this application.

If you combine this process with the foam-filling process you can get shapes built from tubes with strong, rigid, hard surfaces braced by a weaker foam within.

Also, of course, most of these processes can be used on top of a form produced by some other method than globoflexia; for example:

• 3-D printing;
• bending a wire armature;
• using an existing object such as a vase;
• blowing glass;
• vacuum-forming plastic;
• blow-molding plastic;
• electrotyping;
• modeling with clay or other modeling compounds;
• origami, whether with paper, sheets of PET or other plastic, aluminum foil, or other materials;
• commanding a motorized reusable "armature" to assume a certain position until the papier-mâché draped over it hardens;
• CNC machining;
• cutting, folding, and assembling shapes out of cardboard or MDF, though this requires special attention to the adhesive's water content;
• laser cutting;
• piling up sand or other soil, whether with an additional binder or not;
• assembling Legos, Meccano, Ramagon, Heckballs, modular T-slot aluminum framing, or other reusable "construction set" parts;
• building latticework structures out of other kinds of members, for example, metal trusses or Tensegrities like Kenneth Snelson's;
• manual carving of carveable materials such as metals, wood, foamed concrete, alabaster, graphite, lightweight refractory bricks, tuff, sandstone (natural or artificial), or papercrete;
• hot-wire cutting of fusible foams such as styrofoam or polyisocyanurate;
• inflatable shapes made in ways other than balloon twisting; for example, connecting sheets of polyethylene into a large balloon

sculpture using a hot-wire heat-sealing machine;
• assembly of a variety of objects, for example with hot glue;
• forming sheet metal, for example by hammering, stamping, single-point incremental forming, or bead rolling;
• assembling panels or other shapes cut or otherwise shaped from closed-cell polymer foam or other materials;
• carpentry;
• basket weaving, whether from traditional materials such as bamboo and rattan or non-traditional materials such as Ethernet cable and sheet-metal strips;
• other techniques for producing three-dimensional shapes;
• some combination of the above.

In some cases it will be most convenient to apply the adhesive to the fiber reinforcement after it is already in position, but in other cases, especially with porous adhesives, it will be most convenient to combine them ahead of time.

During the process of adding layers of fiber reinforcement and (possibly filled) adhesive, it may be convenient to embed other elements in the object being constructed, in non-random positions. For example, you can embed sensors, heating elements, LEDs or other lights, antennas, pancake coils, and wires to feed all of these. For some purposes it is best to cover these with a layer of adhesive and/or fiber, for example to prevent abrasion or electrical short circuits, while for other purposes exposed electrodes or other actuators may be useful.

Specific combinations

The above outlines a large design space of processes, a few of which are already in use:

• The ordinary kinds of papier-mâché, which commonly use untwisted balloon forms, but sometimes wire armatures.
• "Cloth mache" [sic] in which fine cotton cloth is used instead of paper, either as the last layer or as several layers, sometimes with PVA glue rather than wheat paste.
• Duct tape, masking tape, strapping tape, scotch tape, and electrical tape are all composites of adhesive with reinforcing fibers; strapping tape fibers are often parallel glass fibers with a PET or polypropylene backing, masking tape is paper, scotch tape is cellophane, electrical tape is generally plasticized PVC, and duct tape is often polyester scrim with an LDPE backing. (Velma Stoudt's original "duck tape" used cotton duck, as the name implies.) Duct tape also typically includes a powdered aluminum filler in the LDPE for reflectivity. All of them can use a wide variety of so-called pressure-sensitive adhesives. So, pre-combining the adhesive with the reinforcing fiber makes for a very convenient and versatile way to make and repair things.
• Découpage, in which the "fiber reinforcement" is primarily decorative and the adhesive is transparent, and the form is usually made by carpentry rather than globoflexia.
• Standard fiberglass composite construction uses one or more layers of fiberglass cloth as fiber reinforcement, sometimes laid up on top of forms made by hot-wire cutting of styrofoam, then smoothed down

with epoxy. Often additional layers of epoxy without cloth are added after the first in order to provide a smooth surface without any fiberglass sticking out of it.

• "Textile-reinforced concrete" is ordinary portland cement reinforced with high-modulus cloth rather than rebar; typical fibers used for the textile include carbon fiber, AR glass, and basalt fiber. The forms are typically made of wood or styrofoam rather than by globoflexia. Making TRC forms by globoflexia would likely require plastering the balloons first with a lightweight support material such as ordinary papier-mâché.

• Concrete canvas inflatable tents use a form made of a large inflatable polyethylene bag to support a pre-sewn canvas fiber reinforcement pre-impregnated with an adhesive system made of portland cement and quartz construction sand, which is wetted with water before inflation.

• The same kind of concrete canvas is commonly used as a geotextile, in which case the form is just the earth's surface.

• "Ferrocement" uses a form made by bending rebar and "fiber reinforcement" made of lighter-weight metal cloth, such as hardware cloth or chicken wire; once the first layer of fiber reinforcement is in place on the form, an adhesive system of typically portland cement, quartz construction sand, and water is troweled on, often followed by more layers of fiber reinforcement. Usually a layer of adhesive is also added to the *inside* of the structure to cover up the steel. The first layer of adhesive often includes a chopped-fiber additive more to control the rheology of the mix so it doesn't fall through the holes in the fiber reinforcement than to strengthen the final product. This has been used for decades for lightweight buildings and cheap boats.

• Very similar to ferrocement, fine-art sculptors commonly build forms as bent-wire armatures, cover them with chicken-wire fiber reinforcement, and trowel on a plaster of paris adhesive, with or without strengthening fillers of sand and, for example, horsehair or sisal.

• Traditional lath-and-plaster construction makes forms of lath rather than balloons, mixes horsehair and sand into the plaster-of-paris adhesive additives to improve strength, and uses no further fiber reinforcement.

• Traditional gauze-and-plaster medical casts for healing broken bones were an inspiration, as mentioned above. In this case the form is grown in a womb rather than twisted from balloons. This technique turns out to go back to the Middle Kingdom of ancient Egypt, around 4000 years ago; archaeologists call it "cartonnage".

• Shotcrete has been on many occasions sprayed onto inflated domes to make Barbapapa-style houses. In this case the balloons are very large and not twisted. Typically the concrete is portland cement, sand, water, and chopped fiber, which last helps reduce slumping.

• Papercrete is a very-low-compressive-strength, low-weight portland cement concrete where consisting of portland cement, water, and cellulose fibers from paper. Sometimes it is applied to forms but more commonly it is used to build walls.

• Paperclay is a composite of cellulose fibers from paper, clay, and typically sand or grog. It is usually shaped by hand rather than being applied to forms. It has superior green strength to more common clay materials, reputedly improving freedom of modeling. When fired,

the paper burns out, leaving a lightweight porous material.
• I have seen one person explain their system for constructing lightweight RV furniture from blocks of styrofoam cut with a hot knife, assembled, and then bonded together by coating the inner and outer surfaces of the furniture with window screens glued to the foam with latex paint. This forms sandwich panels whose compressive and shear stiffness come from the foam and whose tensile and flexural strength comes from the window screens.

However, the systematization suggests many new promising combinations. For example:

Lime-concrete furniture

The initial form is produced by globoflexia and wrapped in three layers of gauze strips dipped in fresh plaster of Paris. Fifteen minutes later, steel window screens are draped over the plaster frame, and a thick mix of slaked lime, water, quartz construction sand, and chopped fibers is troweled onto the screens. Two more such layers of screen and lime plaster are immediately applied. A few hours later, the outer surface is painted with sodium silicate to increase its resistance to abrasion; sufficient air can still enter through the porous plaster and lime cement to cure the piece over the next 24 hours. Filling the final piece with foamed portland concrete, made in the usual way, is optional.

Cement water pipes

A balloon for twisting is inflated but not twisted. It is wrapped in five layers of paper towels dipped in wet portland cement, sand, and chopped basalt fiber, leaving the balloon's ends exposed. The entire resulting concrete tube is wrapped in stretch plastic wrap to keep it from drying out. After 48 hours, the balloon is popped, exposing the inner surface of the pipe, which is then painted with a solution of potassium silicate to waterproof it.

Bargain-basement roofing

A light roof metal truss is built by bending and arc-welding together rebar. Jute burlap cloth is dipped into hot tar and laid on top of the truss, one square meter at a time, overlapping squares in a shingled pattern. Three layers should be sufficient for a rainproof roof sufficiently flexible not to suffer damage from hail. However, it is a fire menace, it will get very hot in the sun and may drip, and you cannot walk on it. Coating the top and bottom surfaces with aluminum foil will ameliorate these defects slightly.

Cheap, lightweight inert pipes

A balloon for twisting is inflated but not twisted. A4-sized paper is dipped in low-melting paraffin and wrapped around the balloon in three layers. Once the paraffin is cool, the balloon is popped or untied and removed. The non-knot end of the balloon can be left open, forming a closed tube, or closed. To improve strength, a fourth and fifth layer of paper dipped in two-component resin rather than paraffin can be added on the outside.

Flexible heat-resistant oven mitts

The clay form of a hand is slipcast from a clay slip containing a

mildly acidic flocculating additive such as vinegar, using a porous mold made of plaster of Paris. The hollow slipcast clay form is demolded and tightly wrapped in four layers of loosely woven cotton or linen cloth, richly smeared with high-temperature acetic-acid-catalyzed silicone ("red RTV"). Once the silicone has cured, the still-wet clay interior is washed out with water and a mild base such as baking soda in order to deflocculate the clay. The resulting piece, once the acetic acid has escaped, can withstand temperatures up to some 240°; substituting a cloth with higher-temperature capabilities should allow it to handle 280° continuously or brief exposures to 320°.

(Other reversible flocculants might be epsom salts, which can be deactivated by barium carbonate, and muriate of lime, which can be deactivated by soda ash or barium carbonate. I've also seen muriate of lime recommended as a *de*flocculant, presumably to throw down vitriol in the form of alabaster.)

It might be necessary to protect the cotton from hydrolysis by the acetic acid before curing is complete; this can be done either by replacing acetic-acid silicone with more expensive tin-catalyzed or platinum-catalyzed varieties, or (with lower certainty) by impregnating the cotton with a buffer of, for example, baking soda.

This is a case where a balloon form shaped by globoflexia is probably better than clay, actually, because it's both easier to shape to the appropriate smooth blobby shape and easier to remove. But it's important to remove it completely, because the balloon latex will probably fail at a much lower temperature.

Translucent all-natural low-VOC objects

By wrapping your twisted balloons in gauze soaked in shellac, you can get a waterproof, light, flexible material that allows significant light through, due to the gauze's light weave. Alcohol is emitted as the shellac dries, but this is a fast process; once dry the material emits almost no VOCs.

A coarse filter unplugged by heat

If you stamp one or two layers of a loose steel mesh such as a window screen, impregnated with warm paraffin wax, with a die, then you have a waterproof and chemically inert plug which, at a predetermined temperature (one calibratable within 5°) will melt and allow liquid to flow through freely, while filtering out particles larger than the mesh. This could be useful for some kinds of over-temperature safety valves, for example for resin casting, where, if the resin starts to overheat, the ideal thing to do might be to dump it quickly out of the mold into something that dilutes and cools it. It is possible for this mesh to have a much larger area than the aperture it covers, which may be desirable for keeping it from getting clogged by particulates.

Under some circumstances it might be better to use injection molding to inject the paraffin around the reinforcing mesh. This would provide more consistent paraffin thickness but, I think, less consistent mesh protection thickness.

Ultralight tools for corrosive environments

By cutting the shape of a stirrer out of, for example, styrofoam, you can get a very lightweight tool. But styrofoam is soluble in all kinds of solvents, and it's kind of weak. By wrapping it with fiberglass cloth, as is done to construct some boats and aircraft, you can greatly strengthen it. A coating of, for example, paraffin, low-density polyethylene, teflon, epoxy, or polyester casting resin, could both firmly adhere the fiberglass reinforcement to the foam and add substantial chemical resistance.

Carved aircrete furniture

You can pour portland cement foamed in the usual way, by mechanical aeration of a surfactant-water solution before mixing in the cement, into forms that are merely blocks. The next day, once the cement has partly set, you can sculpt these blocks into desirable shapes using hand tools like hacksaws, wood rasps, wire saws, hammers, and so on. The resulting surface will be porous and friable, and therefore not directly suitable for furniture use, and also an ugly gray unless you used super fancy portland cement. Several coats of lime mortar (slaked lime and quartz construction sand) can give it a hard shell, perhaps reinforced in key places with copper wires. The next day, a coat of polyurethane finish for heavy-duty floors can seal the lime and provide a softer, warmer surface to sit on or rest your feet on.

Fiber-reinforced pottery

The usual kind of pottery is fragile. The ceramic fibers used in foundry blankets are much less fragile, and some of them can be used up to 1600°. You could perhaps take segments of refractory-fiber cloth like these foundry blankets, dip them in a clay slip, and drape them over forms (for example, blow-molded from thermoplastic) to make a shape of two or three millimeters of thickness. Once the clay slip was plastic, but before it became leather-hard, you could add another millimeter of clay to the inside and outside. After drying and biscuit-firing these pottery pieces in the usual way, the clay should be sintered into a solid body; you can get a good biscuit fire out of at least some ball clays in 6 hours at 1020°, at which temperature some ceramic fibers are still quite inert. So they should remain embedded as fibrous reinforcement in the finished ceramic, making it dramatically less fragile.

However, care must be taken to ensure that the chemistry of the clay is compatible with that of the blanket. Pure zirconia fiber (or yttria-stabilized zirconia fiber) would probably be perfectly safe, but I think everybody includes at least alumina and usually silica in their ceramic foundry blanket fiber. (Vendors of pure zirconia fiber say it can be used up to 2200°.) I suspect that any low-firing clay would be able to flux and dissolve silica out of part-silica fiber, and maybe alumina too. The end product might still be stronger than ordinary ceramics, though.

Silicon carbide fibers are more widely available than zirconia fibers; four companies already sell them commercially, at least two since the 1980s (under the names Nicalon, Tyranno, Sylramic, and Ultra SCS). I think they are not attacked by clays at common pottery-firing temperatures, and they are already in use for reinforcing ceramics — but I think primarily ceramics otherwise made of sintered silicon carbide, not fired clay.

If desired, a second glaze firing can glaze the pieces to give them waterproof surfaces and provide protection against abrasion and crack initiation. However, this poses the risk that the more aggressive fluxing of the glaze might attack the fiber reinforcement; this is the reason for the extra protective layer of clay without fiber in it. If this is a problem, a possible alternative to traditional glazing is waterglass allowed to dry on the ceramic and then crosslinked by, for example, exposure to calcium chloride.

"Ceracement": refractory "ferrocement"

The usual ferrocement recipe uses iron (and consequently a little iron oxide), portland cement, and quartz, none of which is very friendly to temperatures above 1000° or 1500°. Calcium aluminate cement can replace the portland cement, and olivine, sapphire, or carborundum can replace the quartz, but what can replace the iron?

Refractory metals like tantalum and niobium are well known, but very expensive. Ceramic fibers like those mentioned above (zirconia, alumina, carborundum) might be adequate; the "ceracement" structure won't need flexurally-stiff reinforcement to hold it up, since it can hold itself up once the cement is set.

At even higher temperatures calcium aluminate fails and needs to be replaced with higher-temperature castable refractory binders such as aluminum phosphate.

Shatter-resistant grinding stones

Modern synthetic grinding stones have a variety of compositions: sapphire, silicon carbide, cubic boron nitride, etc., bonded with rubber, thermoset resin, waterglass (mostly historically), Sorel cement, and so on. But they tend to fail in a brittle fashion rather than a ductile fashion, which frequently kills people when they are spinning fast around people.

Cutoff discs are like thin grinding wheels, but they are usually reinforced with a fiber, typically fiberglass, I think.

Perhaps grinding wheels could be made with much heavier fiber reinforcement to encourage them to fail in a more ductile fashion. High-energy-capacity fibers like rubber, nylon, or music wire might work better for this than high-modulus fibers like fiberglass and basalt fiber.

Water-activated concrete tape

Coat a roll of cotton scrim fabric with a low-temperature nonpolar thermoplastic adhesive like EVA. Heat the cloth and run it through a pile of premixed quick-setting dry lime cement and construction sand, which sticks to the EVA and coats the cloth. Allow the cloth to cool before spooling it onto the takeup roll. Seal the finished roll hermetically in a reclosable container.

The resulting tape can be torn by hand like duck tape, although gloves are advised. Once a form is wrapped with it to a few millimeters thick, and flexed into the desired shape, you can moisten the tape around the form to start the cement setting. Water can soak through it easily, and it will amalgamate into a cotton-reinforced mortar mass.

Perhaps such tape can be laid between bricks or stones to hold them

together, rather than troweling in mortar.

Other cements can be substituted, such as portland cement or calcium aluminate, which would give stronger results. There may be faster-setting high-strength cement formulations that are not in traditional construction use and that would activate the tape even faster. Using plaster of paris instead of the cements suggested would provide much faster results (and perhaps this is already in use) but much lower strength.

One particularly interesting possibility is using dissolved sodium-silicate waterglass as cement, which is somewhat tacky immediately and will set up hard when dried; but a variety of things will cause it to set up immediately and become water-insoluble, such as carbon dioxide gas or, I think, calcium chloride or magnesium sulfate. So you could perhaps spray solutions of those on the tape, once it is applied, from a spray bottle.

Steel wire mesh would be a stronger alternative to cotton scrim, and might still be possible to tear by hand.

Topics

DNS Cache Rendezvous: a permissionless signaling channel for bootstrapping end-to-end connections

Kragen Javier Sitaker, 02020-10-07 (13 minutes)

In today's internet, IP addresses are often assigned dynamically and unpredictably, but not changeably at will. So if you want to send a person IP packets, you need some way to find out what their current IP address is. If they're behind a NAT, you may also need to find out what their current port is on that IP address so that you can do NAT hole-punching. (Some kinds of NAT don't even support that, but most do.)

The IPv4 + UDP port data is 48 bits. If you could get that data, or most of it, to your contact, the two of you could establish a UDP connection. So you need some kind of rendezvous point.

Here's a permissionless, harmless, efficient solution vaguely similar to private information retrieval protocols.

Background on the DNS

Let's consider the side channel associated with a caching DNS server and a domain name with a relatively long TTL value, such as 18000 seconds. For example:

```
100.172.217.172.in-addr.arpa. 17429 IN  PTR eze06s02-in-f4.1e100.net.
eze06s02-in-f4.1e100.net. 34228 IN  A   172.217.172.100
```

From an authoritative server the TTL on eze06s02-in-f4 is actually 86400 seconds, so what we're seeing here is that someone sharing the DNS server with us did a lookup of this domain name 86400-34228 = 52172 seconds ago, plus or minus a second or so. They have effectively written about 16.4 bits into this DNS server's cache, which now anyone who the DNS server is willing to respond to can read.

There's a "norecurse" bit you can set on a DNS request. This doesn't prevent the DNS server from returning you a value from its cache, but it does prevent it from going and fetching the value. This is useful because it permits a nondestructive read of this timing data. Here we see two identical queries on the public DNS server 8.8.8.8, one of which fails with SERVFAIL, and the other of which succeeds, because 8.8.8.8 is not only anycasted, but also its local instance seems to be load-balanced on a per-request basis:

```
$ dig +norecurse @8.8.8.8 eze06s02-in-f4.1e100.net.

; <<>> DiG 9.10.3-P4-Ubuntu <<>> +norecurse @8.8.8.8 eze06s02-in-f4.1e100.net.
; (1 server found)
;; global options: +cmd
;; Got answer:
```

```
;; ->>HEADER<<- opcode: QUERY, status: SERVFAIL, id: 2151
;; flags: qr ra; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;eze06s02-in-f4.1e100.net.  IN  A

;; Query time: 44 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Wed Oct 07 13:36:44 -03 2020
;; MSG SIZE  rcvd: 53

$ dig +norecurse @8.8.8.8 eze06s02-in-f4.1e100.net.

; <<>> DiG 9.10.3-P4-Ubuntu <<>> +norecurse @8.8.8.8 eze06s02-in-f4.1e100.net.
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 12898
;; flags: qr ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;eze06s02-in-f4.1e100.net.  IN  A

;; ANSWER SECTION:
eze06s02-in-f4.1e100.net. 21586 IN  A   172.217.172.100

;; Query time: 42 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Wed Oct 07 13:36:47 -03 2020
;; MSG SIZE  rcvd: 69
```

The DNS cache rendezvous protocol

Suppose you have a prearranged list of 128 DNS servers somewhere on the internet that do not use load balancing, are not anycast, and probably are willing to provide recursive, caching DNS resolution to each of you. And you also have a prearranged list of 128 long-TTL DNS records.

What we are going to do is store a 50-bit value in this 128x128 matrix of caching DNS servers by splitting it into ten 5-bit chunks, and storing each 5-bit chunk in the cached TTLs of these 128 shared recursive caching DNS servers. First, we take the key we want to associate information with and we hash it to produce a long random bitstring. Each 14-bit chunk of this hash identifies one bucket in the 128x128 matrix: a particular DNS record on a particular caching server. We are going to store each 5-bit chunk of information redundantly in 5 such buckets as the low-order 6 bits of the clock when we make the request, except for the least significant bit, which is not reliable. In order to do this, we send a single 50-byte packet to the indicated server in the middle of the indicated time interval,

requesting a recursive retrieval of the indicated DNS record.

If our request is successful, and the record was not previously in the cache, the server will cache the record for its TTL. So we have successfully published 5 bits of data in such a way that the DNS server will send them to anyone who subsequently requests the same record before the TTL expires. Then they need only add the remaining TTL to their current clock and subtract the origin TTL to find out when the original request was sent to a precision of one second.

But the request may not have been successful, or the record may have been previously in the cache. So we store the same data in 4 more buckets, which in general will be on other DNS servers. Now, someone who wants to read this 5-bit chunk of data can make the same 5 requests --- though ideally with the no-recurse bit turned on, so that they won't prevent the data from being published in the future if it's currently not published --- and simply take the most common value from among the results. Really they probably only need to read two or three of the buckets on most occasions, since two or three equal values is already very strong evidence.

Repeating this process 9 more times stores or retrieves 50 bits of data. Of these, 48 are the IPv4 address and port at which to contact the publisher of the advertisement.

So advertising in this way requires sending out about 50 packets of about 50 bytes each, about once every 5 hours, and receiving the same number of response packets of about 70 bytes each; this works out to about 1–2 bits per second both inbound and outbound. Retrieving such an advertisement requires only about 25 requests and responses.

The choice of using the low 6 bits of timestamp imposes a minimum latency of 64 seconds on publishing new data (which could be reduced, except in cases of strong interference, by XORing the data stored in each bucket with more bits derived from the key); if a longer latency were acceptable, you could store more bits per bucket, thus requiring fewer buckets, fewer packets, less bandwidth, but more latency. For example, with half an hour of latency, you could get 10 bits per bucket, not counting the ignored LSB, rather than 5. Inversely, you could get latency down to 4 seconds by storing only one bit per bucket.

In a sense, although in absolute terms the cost is very low, in relative terms it's fairly high: to publish 48 bits of data --- 6 bytes --- you send out 2500 bytes and receive 3500 bytes. That's three orders of magnitude of bloat. It's only efficient in absolute terms because the service required is so minimal.

A large number of publishers can use the same 128x128 matrix as long as they aren't trying to stomp on each other's keys, because each one only uses 50 out of the 16384 buckets. However, it's easy for anyone who has the whole matrix to deny service to everyone.

A possible partial defense against that is to distribute different versions of the matrix from a central authority to different participants in the system, having for example two possible alternatives for each row and two possible alternatives for each column, half of which are concealed from each participant. Geographically distant participants will share, on average, half the headings and one quarter of the buckets, and so mildly more queries

will be needed.

Another partial defense would be to use a much larger matrix, like a million by a million, which preliminary tests suggest is feasible (see below). Then anyone who wants to flood the whole matrix needs to send out *one trillion packets*, like, fifty terabytes. Every five hours: ten terabytes an hour.

Any attacker who knows the key of a publisher is likely to be able to jam their broadcasts. This suggests that perhaps keys should be per-relationship, not per-identity: if Alice uses one key to announce her location to Bob and another to announce it to Carol, then Bob can't jam the information Carol is reading, unless Carol tells him the key or he can jam essentially the whole matrix.

A publisher might need to use two or three keys for rapid failover when its IP address changes unanticipatedly, since it can't overwrite its previously published data.

Of course there are more efficient error-correction codes than repeating each symbol N times, and using these might be worthwhile.

There are 300 million servers currently providing this service

One key question I didn't know about when I started this is how many publicly-accessible recursive DNS servers are still out there. I started out fairly confident that the answer is "more than 128". I think the answer is actually "hundreds of millions" because the very first random IP address I generated, 39.188.24.230, happened to be willing to answer my DNS query:

```
$ dig @39.188.24.230 www.google.com.

; <<>> DiG 9.10.3-P4-Ubuntu <<>> @39.188.24.230 www.google.com.
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 37948
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;www.google.com.            IN  A

;; ANSWER SECTION:
www.google.com.     141 IN  A   31.13.95.38

;; Query time: 423 msec
;; SERVER: 39.188.24.230#53(39.188.24.230)
;; WHEN: Wed Oct 07 14:16:44 -03 2020
;; MSG SIZE  rcvd: 48
```

The next hundred or so I generated got me 5 answers (all different!), so probably about 5% of all the possible IPv4 addresses have working (?) recursive DNS servers that are willing to answer queries from my Argentine residential address. That's about 200

million currently existing and accessible servers. Presumably a tiny fraction of them are anycast or dynamically load-balanced like 8.8.8.8, while the rest would work fine.

Further random sampling refines that estimate to about 7%, which is about 300 million servers. I generated 1000 random IPv4 addresses, sent a DNS query for www.google.com to each one (once), and got back 67 responses and 934 timeouts. Not sure where the 1001th request went.

Detectability

You could easily tell someone using this technique from a normal DNS user: if publishing, they're sending DNS queries with recursion turned on to several different DNS servers. Stub resolvers send queries with recursion turned on, but normally only to your ISP's nameserver, or to 8.8.8.8 or opendns or alternic or something, not to 50 different servers. Someone only reading would look like someone running their own caching DNS server, in that they're sending out queries with recursion turned off, except that many of their queries are getting non-authoritative results. Queries with the NR bit set generally will get either authoritative results or no results; it's very unusual for someone to send a no-recursion query and just happen to get a successful response from some random cache.

The publisher might look like a sysadmin trying to debug a DNS problem. I can't think of any activity that would look like the person trying to read.

That said, I don't know what kind of offbeat DNS things happen in the wild nowadays. Maybe there's some common DNS implementation bug that looks just like this.

Alternatives

Other permissionless signaling channels for such low-bandwidth rendezvous tricks include blog comment sections, wiki edits, Freenode, OFTC, Tor hidden services, altering the latency of publicly-accessible servers on low-bandwidth connections by packet-flooding them, shortwave radio, moonbounce, Usenet, web forums, and IM service statuses. Most of these impose significant social costs on others, could disappear at any time, or have other drawbacks that this public DNS side channel does not have.

Topics

- Security (p. 811) (11 notes)
- Protocols (p. 813) (10 notes)
- Independence (p. 817) (9 notes)
- DNS

Nodebook: autotagging quantities for ad-hoc calculation and example-based end-user programming

Kragen Javier Sitaker, 02020-10-07 (7 minutes)

05:16 <xentrac> I think I have a UI direction for bicicletaish/halpish stuff that I think might be really
appealing

05:17 <xentrac> the basic interaction is that you're typing text into a text editor, but the editor is
watching your text for "quantities"

05:18 <xentrac> it might have different patterns for a "quantity". Like clearly 3.14 would be a quantity, and

for my purposes I want to also recognize things like 3.14 mm and 3.14 m/s

05:18 <xentrac> but you could imagine an arbitrarily wide range of quantities. anyway it's looking for them

in your text and initially it just tags them, say with a dotted underline

05:19 <xentrac> now that gives you the option of scrubbing it with your mouse to change it, bret-victor-style,
but so far that's not very useful

05:20 <xentrac> you can correct its autotagging if it fails to notice a quantity or if its parse boundaries

05:21 <xentrac> so this nodebook node is, so far, just some text with markup. and, I don't know, maybe if you

type a #fe77cc color you get a color swatch, and clicking on it pops up a color picker, or
something

05:23 <xentrac> so the next thing you can do is that you can initiate a calculation, which starts by popping

up a menu of recognized quantities in the neighborhood, and you can select one of them or you
can start typing more numbers and operators and stuff

05:24 <xentrac> and so you can perform a concrete calculation on these concrete q

uantities, and by default the
formula and the result are displayed there in your text too

05:25 <xentrac> so now when you scrub on things your document responds, recalculating. and you can mouse over

parts of the formula or navigate it with keys in order to see intermediate quantities

05:26 <xentrac> it's still all basically text, though maybe latex or something is rendering your formulas

05:28 <xentrac> so maybe you write "An air conditioner of 2 ton capacity is 7033 W; if it consumes 12 A at 240
V, that's 2880 W, so its coefficient of performance is 2.44."

05:28 <xentrac> and here 7033 W, 2880 W, and 2.44 are all calculation results, which might have formulas
displayed before them

05:28 <xentrac> And you probably have some kind of command to control the units and precision of such displays

05:30 <xentrac> the next thing you can do is to name the quantities, whether directly entered quantities or

the results of calculations (which are implemented in the same way for execution, but perhaps
not in the user interface)

05:32 <xentrac> now, once you have named the quantities, you can do what-if questions, like "By contrast, if

it must consume {i2=} 25 A, its CoP is {this{i=this.i2}.cop:%.2f}."

05:33 <xentrac> where the stuff in {} is not displayed in the document but is a goofy way I just came up with
to try to describe what's underneath.

05:34 <xentrac> This also allows you to do optimization ("goal-seek") calculations where you specify a model,
a set of design variables, and an objective function to minimize

05:34 <xentrac> and plotting, where you specify a range for one or more independent variables and one or more

dependent variables to plot, along with specifying what kind of plotting you want

05:36 <xentrac> and tabulation, where you specify a set of columns, the number of rows, and override values
for some cells

05:37 <xentrac> note that so far this is all without any functional abstraction!

o no nested scopes yet, just

"the document (or universe or node)" and conditional versions of o
othe document with some
variables overridden and maybe not fully displayed

05:38 <xentrac> there's a hierarchical structure to the execution but not to the o
odefinition (except insofar as

maybe a formula might be written in a context-free language rathe
or than, say, Forth)

05:40 <xentrac> now if you can write multiple independent documents like this, yo
ou maybe have everything you

need, but I do think it would be handy to highlight a block and s
oay "refactor tthis block of
text into a child node"

05:40 <xentrac> which would leave the display of the document pretty much unchang
oed, except for some formulas,
since by default you'd be transcluding the child node there
05:41 <xentrac> but would maybe make it easier to reuse

05:42 <xentrac> I think this is a more appealing user interface than observablehq
o, but it can support the same
kinds of interactions

05:44 <xentrac> It supports fully concrete example-based computation, with condit
oionals and lazy evaluation

it's Turing-complete, and it doesn't inherently require vectors i
on the data model, for better
or wose
05:44 <xentrac> worse

05:47 <xentrac> and of course each of these "documents" or "nodes" could exist in
o a "nodebook", with or
without a [human-readable] name, and be invocable by one another

05:48 <xentrac> for testing purposes, I think it would be useful to have an "asse
ortEqual" formula operator

which produces an error object with explanatory text and a hyperl
oink to its invocation site

05:49 <xentrac> and that interpolation of such an error object into a textual tem
oplate (since the underlying

representation of the node is a set of instance variable definiti
oons, the value of one of

which is a formula using a textual template and a textual substit
oution operator)

```
 05:50 <xentrac> would produce text similar to non-erroneous interpolation, but flo
agged as an error as well

 05:51 <xentrac> so you could just look for (top-level) nodes whose display value o
was an error, and put a bunch
                 of asserts into one as tests
 05:51 <xentrac> maybe they would be red in a display

 05:51 <xentrac> I'm curious what you think, and sorry for subjecting you to this o
steam-of-consciousness
                 explanation
```

This seems like it might really work as a Wiki thingy.

So much for the user interface. What should the storage format look like?

The traditionalist approach would be to define a plain ASCII (or Unicode) text grammar and write a parser for it. This would have the advantage of making source control easier. But it also involves writing a parser and a deparser, as well as the user interface, and then fixing bugs where they mismatch.

Probably a better approach is to serialize data in some kind of very general format that is still likely to be source-controllable. The Lisp object-graph memory model maps reasonably well to something like YAML.

It's possible to use multiple files like R Markdown notebooks, where the source is stored in one file and the rendered result in another file next to it (perhaps not checked in to Git). Rendering to HTML is also super important.

Topics

Single-bridge Tor deanonymization?

Kragen Javier Sitaker, 02020-10-07 (4 minutes)

I just saw this:

```
06:14 -!- rabbitear_g [~rabbitear@gateway/tor-sasl/rabbitearg/x-03735317] has quit [Remote host closed the
          connection]

06:14 -!- bb-8 [~bb-8@gateway/tor-sasl/bb-8] has quit [Read error: Connection reset by peer]

06:14 -!- DiffieHellman [~Ident@gateway/tor-sasl/diffiehellman] has quit [Write error: Connection reset by
          peer]

06:14 -!- andreas303 [~andreas@gateway/tor-sasl/andreas303] has quit [Write error: Connection reset by peer]

06:14 -!- stipa [~root@gateway/tor-sasl/stipa] has quit [Write error: Connection reset by peer]

06:14 -!- Ryuuguu [~Ryuuguu@gateway/tor-sasl/ryuuguu] has quit [Remote host closed the connection]

06:14 -!- martian67 [~martian67@about/linux/regular/martian67] has quit [Read error: Connection reset by peer]

06:14 -!- ZombieChicken [~weechat@gateway/tor-sasl/forgottenwizard] has quit [Read error: Connection reset by
          peer]

06:14 -!- CombatVet [~c4@gateway/tor-sasl/combatvet] has quit [Read error: Connection reset by peer]

06:14 -!- milkt [~debian@gateway/tor-sasl/milkt] has quit [Read error: Connection reset by peer]

06:14 -!- kreyren [~kreyren@fsf/member/kreyren] has quit [Read error: Connection reset by peer]

06:14 -!- bamdad [~bamdad@gateway/tor-sasl/bamdad] has quit [Read error: Connection reset by peer]
06:14 -!- bamdad [~bamdad@gateway/tor-sasl/bamdad] has joined ##electronics
```

13 users knocked off Freenode ##electronics at once, out of 630 people. One of them, kreyren, was using a project hostname cloak. Presumably a Tor node somewhere went down --- I think an exit node, due to the ECONNRESET error message. This event can be

observed with subsecond precision.

Suppose you wanted to deanonymize a Freenode Tor user. You could set up a bridge or a Tor entry node. Periodically you could drop connections from users who use it, a normal event that can be provoked by backbone routing problems or Wi-Fi signal fades, after which the user will retry connecting to Tor. If you log the time of this event while simultaneously observing Freenode, you can see if it correlates with your target Freenode users going offline with a "Remote host closed the connection" message. If so, you log the IP address and port.

These are relatively rare events; I observed one 8 minutes ago and another 11 minutes ago in this same channel, giving a rate on the order of 200 kiloseconds, so even a single "hit" is a $p < .001\%$ event --- good enough, as they say, for government work. Two hits on different days would be a stronger confirmation and would also allow you to characterize the Tor user's IP address distribution.

An uncertainty that I need to test out is whether closing the circuit from its origination point within the Tor network will immediately close all the outgoing TCP connections from that circuit from the Tor exit node, and if so, whether it's a "connection closed" kind of normal situation or more an RST RST RST kind of thing.

Another uncertainty is how many Tor entry nodes a given user will end up using. If they choose randomly with a nonzero probability for each entry node, they will eventually use all of them, so every entry node will have the opportunity to launch this attack. Bridges, however, as I understand the situation, are treated differently, and may be a defense against the attack: each Tor bridge is only revealed to some users, and the use of a Tor bridge would hide the real IP of the user from the entry node. So if you try to launch this attack with a single bridge against a single user, you will fail with high probability; if you try to launch it against many users, you will succeed with only a few of them.

I'm not clear that this is something anybody needs to respond to or defend against in any way, even if I'm correct, since Tor is not designed or claimed to defend against a global passive adversary --- that is a very difficult problem to solve. And of course there are some well-known problems with malicious exit nodes, and at least one person has been prosecuted for sending a bomb threat to his university over Tor, because he was the only person connecting to the Tor network from the campus at the time the threat was sent. But I'm surprised that such a simple *active* attack seems so likely to work.

Topics

LOGSL: Lisp object-graph serialization language

Kragen Javier Sitaker, 02020-10-07 (updated 02020-10-09)
(8 minutes)

LOGSL: Lisp object-graph serialization language. Not a markup language. I insist that it is not merely aping YAML.

Problem to solve

The problem to solve is mostly the problem of Python pickle: to serialize a possibly-cyclic in-memory object graph, then deserialize it. However, I have a couple of desires that pickle fails to fulfill:

- I would like it to be mostly human-readable and line-oriented so that I can check the result in to Git and successfully resolve update conflicts.
- I would like it to not be a security hole by default, even at the possible cost of being less convenient to use.

(I seem to be undecided about whether this is a single-programming-language thing or a cross-programming-language thing. Maybe it should be a single-programming-language thing. And maybe that language shouldn't be Python.)

Inspiring examples

Here are some examples of syntax I think might be worth supporting:

```
- one
- two
- three
```

That's a list or array containing three byte strings.

```
x 37
y 38
```

That's the dictionary represented in JSON as {"x": 37, "y": 38}. The ordering of the keys is mandatorily ASCIIbetical.

```
[Point]
x 37
y 38
label A
```

That's an object of class Point whose instance variables are {"x": 37, "y": 38, "label": "A"}. The ordering of the keys is mandatorily ASCIIbetical.

```
- "31"
```

```
- "32"
```

That's a list of two strings. Without the quotes they would be integers. Strings that contain only ASCII alphanumeric characters and the punctuation _, -, ., ?, and @, and do not start with "-", ".", or a digit, must be represented as barewords as in the previous examples. All other strings, such as those that start with "3" or contain spaces, must be represented with doublequotes, backslashing backslashes and embedded doublequotes.

```
[Rect]
start
    [Point]
    x 1.5
    y 2.4
end
    [Point]
    x 3.1
    y 2.6
```

That's an object of class Rect whose instance variables start and end are Point objects. The indentation must be four spaces.

```
- "ø"u
```

That's a list containing a Unicode string consisting of a single codepoint. In the concrete syntax this codepoint is represented by a quote, two UTF-8 bytes, another quote, and a lowercase "u". This bullshit is Python's fault, and in decent languages that just store Unicode in byte strings as Pike and Ritchie intended, producing such an abortion will require the use of a custom mapping to a LOGSL-specific Unicode class.

```
# John Doe
[Person]
firstname John
lastname Doe
wife (Mary Roe)
```

That's a definition of an object labeled "John Doe" so that it can be referred to elsewhere, specifically by the reference "(John Doe)". Its instance variable "wife" is indirected through just such a reference, to an object named "Mary Roe". Such definitions must occur in ASCIIbetical order following the main object graph. Their identifiers are arbitrary but must be unique. All those objects that are referred to more than once must be defined in this way. Other objects may be defined in this way as well, for example to keep indentation manageable.

No objects not transitively referenced from the main object graph may be thus defined.

The label line "# John Doe" must be preceded by a blank line, unless it is at the beginning of the file. Other blank lines are forbidden in LOGSL.

If such a label line is at the beginning of the file, it is a label for the root of the main object graph, enabling things within the object graph to refer to that root.

The main object graph, and indeed all such top-level objects (the others being labeled objects), is constrained to be an aggregate object such as a dictionary, a list, or a class instance, not a primitive object such as a string, a number, or null, which is represented as "???".

Hmm, that restriction could be avoided, especially with colons:

```
# John Doe
[Person]
firstname: John
lastname: Doe
wife: (Mary Roe)
```

Dictionary keys that are compound objects could be referred to by title:

```
(John Doe) 5
(Mary Roe) 18
```

Python calling interface

To enable serialization and deserialization of class instances without implicitly granting LOGSL sources the permission to instantiate arbitrary classes, a set of classes or other factories must be provided to the deserializer. Each must be possessed of a unique name.

In Python, the default behavior for deserialization should be something like the following. Get the name from `__name__`. In the case of classes, magically set up an object as follows:

```
obj = klass.__new__(klass)
obj.__dict__ = instance_variables
```

Other behaviors can be provided by a factory object that has `__name__` and can be called; an AliasedClass factory is provided to enable the resolution of name conflicts:

```
class AliasedClass:
    def __init__(self, name, klass):
        self.__name__ = name
        self.klass = klass

    def __call__(self, instance_variables):
        obj = self.klass.__new__(self.klass)
        obj.__dict__ = instance_variables
        return obj
```

Other factory functions or objects can be used to support schema upgrade.

For serialization, we need to supply more or less the same whitelist, and also the possibility of snipping unwanted object references at output time --- the link from the banana to the gorilla, or at least

from the gorilla to the rest of the jungle.

Python pickle does this by defining methods on the banana object; at this point the interface ("the copy protocol") is extremely complex, involving methods known as `__getstate__`, `__getnewargs__`, `__getnewargs_ex__` (I'm not kidding), `__reduce__`, and, just to add insult to injury, `__reduce_ex__`. In a simple case, `Banana.__getstate__` can simply return a copy of its instance variables dictionary with `gorilla` set to null (`None`).

I think that probably a better approach for such cases is to include something other than a class in the whitelist of "classes", which undertakes the work of computing different serialization data. The simplest case is AliasedClass, where we might want to map the class name of the object back to the alias we're expecting to find at deserialization time. This requires making an entry that maps the runtime dynamic class to the AliasedClass instance. But in another case we might want to, say, produce a "reduced" banana:

```
def Banana(banana):
    d = banana.__dict__.copy()
    del d['gorilla']
    return 'Banana', d
```

Somehow, this function must be associated with the class it is intended to reduce, perhaps with a function attribute like `Banana.klass = fruits.Banana`.

It may be worthwhile to define a similar sort of thing for producing candidate labels like the "John Doe" example above. Python's default `repr` for class instances is terrible in that it includes hexadecimal memory addresses, which create unnecessary merge conflicts and false diffs in IPython notebooks. Even "Point 1", "Point 2", "Point 3" would be better, but "Point x=37" would be better still. "Rect start=<**main**.Point object at 0xb64858ac>" would not be an improvement, though.

Golang calling interface

The Golang standard library includes serialization in "Gob", JSON, generic arbitrary XML, and a generic "binary" format. All of these use reflection a lot. So I think it's probably okay for LOGSL to use reflection too.

I don't know how to use reflection in Golang but I bet the source code for those four standard library modules is a good example to work from.

JS calling interface

JS lacks byte strings. Otherwise I think it'll be pretty similar to Python.

Topics

Ancient machinists

Kragen Javier Sitaker, 02020-10-08 (26 minutes)

YouTube keeps recommending me fringe-science videos with catastrophist theories of history, positing the existence of prehistoric high-technology civilizations terminated by a Younger Dryas impact event. I decided to watch one entitled, "Evidence for Ancient High Technology - Part 1: Machining".

Sawing stone blocks

After the author, who goes by "UnchartedX" or "Ben", spends 21 minutes complaining about how Wikipedia and the archaeological establishment are suppressing the theories he favors so they won't lose tenure†, in between deprecating "savages", he finally gets to explaining some actual arguments. He points out that some basalt blocks at what he says is the Old Kingdom Egypt site of Abusir are sawn: the cut surface has striations or grooves on it, with a characteristic spacing of a few millimeters and a height typically under a millimeter, and the block was broken off after being sawn most of the way through, showing that the kerf was a few millimeters thick. He says that it has "clearly been cut by a blade with a very distinctive circular arc to it", but to me the striations look straight. He claims that the striations imply a "rapid rate of cutting", which I don't think follows at all; he doesn't explain why he thinks this.

He says that the radius of the curvature of the striations suggests a circular saw of 8 or 9 meters.

(In passing it's worth noting that, while Abusir does have Fifth-Dynasty Old-Kingdom pyramids, from around 2400 BCE, it also has the remains of a Ramesside temple from only about 1250 BCE, during the New Kingdom. Ben doesn't mention why he thinks these blocks are from the Old Kingdom.)

To me it seems more likely that the cuts were made with an abrasive "wire saw", as is commonly done in quarrying today, rather than a circular saw; but probably using plant-fiber cord, thin wood boards, or copper wire or sheet, to move the abrasive sand through the cut, since steel cable and synthetic diamond were unavailable. Quarrying stone with plant-fiber ropes, water, and sand is a well-documented technology in recent centuries, and could easily date back to the Neolithic. The block seems to be about a meter across, and a meter-long piece copper wire seems more likely to me than a circular saw of 8 or 9 meters in diameter. It would produce the observed kind of striations, but of course without a very consistent radius.

Wikipedia explains that abrasive sawing is the mainstream theory for how the Egyptians cut granite blocks, citing among other things Denys Stocks's 2003 book on the subject. I wonder why Ben doesn't mention this, or indeed abrasive cable sawing at all. He does say, "Until barely 100 years ago, the technique used in the field to cut through granite was far different, and far more primitive," talking about wedge-wetting.

He points out that most of the blocks at the site do not have such striations, suggesting that they were polished smooth, which is clearly a thing that the Egyptians did with stone (see below about flatness).

The standard sawing technology for millennia has been a bow saw or bucksaw, where the blade is held in tension by a frame with some levers, usually tensioned with a twisted pair of ropes; although, with the recent advent of cheaper, stiffer, and harder steels, it's become common to use blades that are just cut from sheet metal, often with the stiffness of the blade itself enabling the saw to be pushed through the cut. Even today you use a bow saw or hacksaw with an abrasive wire blade to cut materials that are too hard for metals to cut.

He points out that at Giza (Fourth Dynasty, Old Kingdom) there are stone blocks with similar saw marks that run only partway through the stone. To me some of these cuts are distinctly curved rather than planar, which is easy to achieve with a wire saw but very difficult with a circular saw — you'd have to make the saw blade accurately spherical (or cylindrical, like a hole saw or core drill) rather than a flat disc. Indeed, somewhat later he points out a sawn granite block at Abu Rawash whose surface is visibly concave, which to my mind clearly demonstrates that it was cut with something like a cable saw or bucksaw, not a circular saw.

You would think that if the Egyptians had 9-meter-diameter circular saws for cutting huge blocks with, they would also make much smaller saws for making the smaller cuts, which would leave circular striations with a much smaller radius as they moved through the cut.

Stocks, who doesn't seem to have one of the academic positions so scorned by Ben, actually went to Egypt to try out abrasive cutting with copper tools; he has some 16 other academic publications from 1986 to 2013 on such subjects in addition to his 2003 book. Most of them are in the academic journals that won't accept Ben's work. However, Stocks doesn't seem to be in favor of the cable-saw theory, instead advocating the use of 6-mm-thick copper sheets to drive the abrasive, using no lubricant — not only because water increased wear on his reed tube drills, but also because it slowed cutting in his tests with both saws and tube drills, rather than speeding it, and was more inconvenient to remove; however, it is known that the later Minoans used water or oil lubricants for abrasive cutting of stone with reeds. Ben attacks the copper-blade approach as impractical, I think rightly so — a cable saw would be more efficient, cheaper, and produce similar markings.

† This suggests that Ben doesn't know what "tenure" actually is, seriously impairing his ability to understand the motivations of the opponents he demonizes.

Core-drilling stone

Ben makes much of spiral grooves found in cores drilled out of stone using core drills, saying these are "machining marks not explainable by the tools and techniques in the archaeological record". Wikipedia claims that core drills date to 3000 BCE in Egypt, so he seems to be in accordance with the "establishment" he so bitterly attacks. He even shows a museum tag saying, "UC.44985: Basalt

tube drill core from enlarged hole. *Tools & Weapons*, LII, 61; p.45. ?Dyn. IV," so I guess he knows this is already the mainstream theory. Nowadays of course we use metal or cermets for our core drills, but hollowed wood or bamboo would probably also work, because what cuts is the abrasive.

Stocks did some experiments using Egyptian hollow reeds, very similar to bamboo, as bow-driven tube drills, with some success, but he thinks these were displaced by copper tube drills not long after 3600 BCE, resulting in a "rapid increase in the manufacture of hard and soft stone vessels" at that time, and mentions that in Minoan Crete (around 2000 years later) emery has been found adhering to drilled-out cores. He was not able to drill granite with the reeds. Emery is impure corundum, or sapphire, which is the major abrasive used today in industry and is dramatically more effective than quartz; Stocks believes, however, that emery was not available in Egypt.

I suspect that if you needed to cut basalt or granite with a reed tube drill, you might have more success after fire-hardening it, a technology 400,000 years old, predating stone weapons, though not stone tools. Using an oil abrasive would have avoided softening the reed with water, and probably would improve the efficiency of the process.

Beaten copper tubes from the Fifth Dynasty have been found, and would be much harder than either reeds or cast copper tubes. They can also be much thinner, also improving the efficiency of the process by reducing the kerf width. Despite this, Stocks believes that cast copper predominated, in part because in his tests soft metal worked better for abrasive cutting.

Ben claims that the evidence of the spiral groove on Petrie's UC 16036 tapered red granite core is suppressed by mainstream archaeology textbooks that tilt photos of it to make it appear non-spiral. However, Stocks's book — my reference for mainstream archaeology — discusses this groove as an established fact, mentioning Petrie's resulting hypothesis that it demonstrated the use of tube drills with jeweled teeth firmly fixed to the tube, rather than abrasive cutting; Stocks rejects this as impractical. He also points out that Petrie found verdigris in tube-drill holes as well as saw cuts, suggesting sawing with copper or bronze, and in one case even bronze particles in a New Kingdom tube-drilled hole.

Another non-mainstream theory is that the Egyptians used sound vibrations to drive their tube drills, rather than bow drills; modern loudspeaker-driven experiments have demonstrated that this is a practical approach. Stocks points out that in his experiments, the tapering observed in many ancient Egyptian tube-drilled holes and cores was only observed with bow drills, which tend to rock the drill back and forth in the hole. Also, tomb paintings show woodworkers with bow drills.

Flatness

Ben makes much of the "flatness" of the various surfaces he observes. However, it's easy to see that many of the masonry walls he admires are made of stones tightly fitted together upon installation, rather than by using the flat surfaces that a circular saw would easily

produce. Corners are curved, and hollows are cut into the corners of stones (presumably by grinding away the points of contact), to enable the blocks to fit together despite the extremely visible non-flatness of their mating surfaces.

Some of the surfaces are indeed quite flat and well-polished, with in many cases parallel scratches from the grinding and polishing process.

I have a vague memory that the technique of grinding three trial surfaces against one another pairwise to achieve flatness, refined by Henry Maudslay with hand scraping of metal, dates to ancient Sumeria, however, I can't find my source for this. Clearly it wouldn't require any technology or materials unavailable at the time, though. Sandstone or fired clay pottery with sand can easily achieve 100-micron flatness in this way, and fired clay tempered with silt instead of sand can reach 10 microns.

In modern machining practice, once you have a flat surface, a standard way to transfer this flatness to a new surface is by "lapping": putting a piece of sandpaper on your flat granite surface plate, then moving the part to be lapped around on the sandpaper in order to grind it flat. This technique has been routinely used in optics for centuries to achieve surfaces perfect to within a fraction of the wavelength of light, though without interferometric inspection, roughness of several microns is more commonly achieved.

A very similar process allows you to produce surfaces that are perpendicular to very high precision, once you have a surface plate; three trial squares are tested against one another on the surface plate, being ground at their high spots.

In the sequel video, Ben claims that Christopher Dunn‡ has used a modern straightedge to measure some surfaces in some kind of stone box in the Serapeum as being flat to within one thou, 25 microns, strongly suggesting the use of the three-surfaces grinding method; the perpendicular surfaces examined were also perpendicular to within measurement precision (claimed to be much better, but he shows a photo of the measurement being taken with a machinist's square not capable of such precision). Ben does not mention why Dunn didn't use a dial indicator to examine flatness to micron precision, and indeed describe's Dunn's method as "relative rudimentary testing".

Although this video claims to be about "precision", Ben doesn't provide a single metric of precision in the whole video; he never says, "The interior of this vase is spherical to within 100 microns," or "The two sides of this statue's face vary from one another by no more than one millimeter." He only says things like "basically perfect" and "identical".

Ben makes much of the fact that incised hieroglyphs are not polished, even when cut into flat polished stone surfaces, claiming that this demonstrates that a much later and more primitive culture cut the hieroglyphs than that which made the flat surfaces themselves. To me it seems more likely that hieroglyphs are unpolished to improve the visual contrast with the surrounding stone.

‡ Christopher Dunn is author of *The Giza Power Plant*, a book claiming that the Great Pyramid was actually a power plant

harnessing "harmonic resonance" to convert seismic energy to hydrogen and microwave energy. Ben promotes this book in the video.

Lathes

Bow lathes are well-attested from ancient Egyptian drawings. Ben is puzzled about how ancient Egyptian stone vessels were made ("I'd like to see *anyone* try and make these by hand with copper chisels and the known techniques of ancient Egypt," he says in his second video), but a glance shows that they were made on lathes — not continuous-rotation lathes like modern lathes, but reciprocating lathes like bow lathes and pole lathes, which allow you to, for example, leave handles on the side of your jug, though those handles don't enjoy the precise circularity of the lathe-cut surfaces. Bow lathes and pole lathes were the common form of the lathe until the 18th century.

However, it must be admitted that the oldest surviving Egyptian depictions of lathes date only from 1300 BCE, only 3300 years ago. It hardly seems surprising that the ancient Egyptians had lathes another 1000 years before that. Ben does eventually mention lathes, explaining that Petrie believed these pieces to be lathework. As far as I know, this is also the current mainstream academic archaeological opinion as well; Ben claims that it is not, because the wheel was not known at this date. But a lathe does not require wheels any more than a tube drill does.

Immediately after quoting Petrie talking about alabaster vases, Ben starts talking about how amazing this is, particularly in "these very hard materials". But alabaster is the second softest stone of all; it's gypsum, also known as plaster of Paris or sheetrock; you can carve it with your fingernails. (Some archaeological "alabaster" is actually calcite, which is harder than gypsum, but only slightly. Chalk is calcite.)

Perhaps Ben doesn't know what alabaster is and didn't think it was worth looking it up, just like he seems to not know what "tenure" is. But at some point the evidence starts to suggest that Ben is not just misinformed or deluded but deliberately deceiving people.

Of course, lathework on such brittle materials would need to be carried out by abrasion in the last stages, not cutting with gouges or chisels.

Stocks in his book points out that the interiors of many of these jars and vases were bored out using stone and wood boring tools, which are clearly depicted in hieroglyphs.

The schist disc

Ben is also mightily impressed (in his second video) by a beautiful schist disc in the Cairo museum with three graceful thin hyperboloids symmetrically carved around a wheel; it's usually known as the Egyptian Tri-Lobed Disc. He perhaps is not aware that you can make such hyperboloids by cutting a series of straight lines between evenly marked points along two curves.

Again Ben lies about the nature of a stone in order to persuade the

ignorant: he calls schist "this very hard stone", but the defining characteristic of schist is that it is very friable due to high phyllosilicate content, which is in fact where its name comes from: σχίζειν, to split.

Sometimes the artifact is described as "metasiltstone", described as a weakly metamorphosed form of siltstone or silty shale favored by Egyptian sculptors for its suitability for thin carvings like this.

Weight

Ben makes much of the fact of moving stones that weigh tens of tonnes. But if one person can lift 50kg, then, without levers, you only need 200 people to lift ten tonnes, which is a small number compared to the population of Egypt at the time. And if they can get 5:1 leverage with some logs, then you only need 40 people.

Moreover, most of the process of moving a large stone like that doesn't involve actually lifting it; it's much less effort to slide it horizontally or on a seked-2 ramp, and you can do that with just ropes rather than having to stick stuff underneath it. And a fellow in Canada has demonstrated his proposed stone-manipulation technique with a several-tonne chunk of concrete: you balance it horizontally on a small number of stone pivots near its center of mass, push down on one end to lift it off all but one pivot, rotate it around that one pivot, possibly position pivots anew, then release its weight so that it settles on the pivots again. Then you can start again from the other end, allowing you to move the pivot you were using previously. He was able to move this slab entirely by himself, using the slab itself as the lever.

Geometry of statues

Ben, quoting Dunn, is very impressed with the geometric precision of the heads of the Rameses statues at Luxor and the Ramesseum, claiming that the only way to make such shapes nowadays is with CNC machining; in 1970, he claims, it would have been impossible. But he show Dunn's diagrams demonstrating that the heads' shapes are mostly composed of simple circular arcs and convex solids of revolution, carefully planned to be tangent to one another. (Dunn claims you'd need NURBS, but his diagrams demonstrate the opposite.)

I do think it's clear that the statues, like Tibetan sand paintings and like pyramids since the time of Djoser, are laid out geometrically, using precise procedures and measurements. But I don't think this requires 18th-century technology, much less 21st-century technology. The Romain du Roi typeface was thus laid out in two dimensions on a regular grid with circles and arcs in 1692.

The simplest brute-force approach would be to simply measure out a large number of points in space. Since we're presumably talking about enormous numbers of workmen sawing and grinding granite for decades, the hard part is not the stone cutting; it's knowing which stone to cut and which to leave. That is, the problem is measurement, or sensing, not actuation.

A point in open space can be precisely located by its distance from three reference points, which can be measured out precisely by metal

chains. (Aside from the reflection ambiguity, of course, which would not have presented the problem for sculptors that it does for GPS.) Copper's linear coefficient of expansion is 16–17 ppm/K; a five-meter-long copper chain will thus lengthen and shorten by some 800 microns with a 10° temperature change, less than a millimeter. The lengths of pieces of wood, clay, or plaster are even more precisely constant, and even plant-fiber rope would probably be good enough. Distances measured with tiny copper chains on a small plaster scale model of a sculpture can thus be scaled up to a full-scale megalithic sculpture with submillimeter errors.

Ben, as usual, never quotes a tolerance, but he does include some measurements from one of Dunn's books, which are given to only four significant figures, which would not be enough to detect millimeter-scale errors.

Moreover, the Egyptians seem to have understood the Pythagorean Theorem by the time of the Berlin Papyrus 6619 (12th or 13th Dynasty, around 1800 BCE), several centuries before Ramesses, and it was in widespread use in Mesopotamia at that time as well. So you wouldn't have needed to make a scale model; you could have calculated.

Drawing tangent arcs and tangent lines on sand (or paper, or papyrus) is easily done with compass and straightedge, both of which are ropes stretched thin over sand until the Hellenistic period. This may not be appreciated by those who have never done it.

To make an arc tangent to another arc at point P, draw a line through P and the center of the arc. Any arc through P centered at any point Q on this line will be tangent to the other arc there. They form a beautiful smooth curve.

Given a point P on a line L, select a point Q on L. L and the circle through Q centered at P have two intersections; one is Q; call the other R. The circle through Q centered at R and the circle through R centered on Q have two intersections; the line through them is perpendicular to L at P.

To make an arc tangent to a line L, and choose a point P on L. Draw a line M through P perpendicular to L. For any point Q on M, an arc centered at Q starting at P will be tangent to the line at P.

To make a line tangent to an arc at a point P on the arc, draw a line M from the center of the arc through P. The line perpendicular to M through P is tangent to the arc at P.

Solids of revolution are most easily produced by a series of circles around their axis, each circle produced by swinging two chains around the axis from two reference points on it. Given a smooth curve in one plane containing the axis, you can use it to repeatedly set the distances on the two chains to a point on the curve, then generate the rest of the circle. However, this will not work for convex surfaces to be cut out of a solid material, because the chains would have to be inside the material.

However, there are many other contrivances that can be put to the same use, one of them being the lathe, which was already in use, as we have seen. But a wooden board that rotates around two pivot points, like a door, would work just as well; the curve to generate the solid

of revolution can be cut into the board, allowing it to be accurately cut into a plaster model. To directly apply the method to the full-size statues without a scale model, you probably need a different way to swing your pattern points around your axis.

(To be continued?)

Backward progression

One of the few things Ben says that I agree with, in his second video, is that there seems to be a "backward progression of technological capability" in ancient Egyptian technology, with Old-Kingdom artifacts being often more finely made than Middle-Kingdom artifacts, which are more finely made than New-Kingdom artifacts, which are more finely made than Hellenistic-period artifacts. The Egyptians themselves at the time are known to have made similar comments, and of course the Bronze Age Collapse loomed large in Classical Greek mythology and popular culture.

I think the reason for this is relatively easy to understand: technological progress is created by innovators like Imhotep, Djoser, Champollion, and Feynman, not by conservative traditionalists. Innovators must constantly struggle against conservatism to make any progress. But it's easy to see, looking at the development of hieroglyphic writing, ancient Egyptian art, and ancient Egyptian metallurgy, that for millennia Egypt was a very traditionalist, conservative society, even for its time; 1200 years after Imhotep, they deified him and thus halted progress. Even the adoption of iron smelting in Egypt took from 2000 BCE until the neo-Assyrian conquest in 671 BCE, several centuries after neighboring kingdoms.

Topics

Level shifter

Kragen Javier Sitaker, 02020-10-08 (updated 02020-10-10)
(9 minutes)

We were talking about level shifters; I asked if a voltage-divider level shifter from 3V3 to 5V was going to sink current into the 3V3 pin, and cloudevil surprised me!

Passive level shifters

```
14:06 < cloudevil> xentrac: No, it doesn't have to, depending on setup.
                           |

14:07 < cloudevil> xentrac: Something like 3.3K from 3.3V out to ground, 1K from
3.3V out to 5V in, and 2K  |

                   from 5V in to 5V.
                           |

14:22 < cloudevil> It is not much good (without great care paid) if you need to d
o a MHz signal, and level  |

                   shifters may be the right way there.
                           |
```

In Falstad's circuit simulator, that's:

```
$ 1 0.000005 10.20027730826997 50 5 43
R -48 160 -80 160 0 2 40 1.65 1.65 0 0.5
r -48 256 -48 160 0 3300
r 32 160 -48 160 0 1000
r 32 80 32 160 0 2000
R 32 80 32 48 0 0 40 5 0 0 0.5
g -48 256 -48 272 0
368 32 160 80 160 0 0
```

This does indeed work as advertised. When the 3.3V wave is at 3.3V, it holds the 3k3 at 1 mA. Then the 3k voltage divider between there and the 5V supply divides the 1V7 difference into 570 mV across the 1k and 1.13 V across the 2k, with the same 570 µA through both, so the 3V3 source only has to source 430 µA. This pulls the input on the 5-V chip to a very acceptable 3.87 V. Then, when the 3V3 I/O pin is pulled down to ground, you instead have 5V split across the 3k voltage divider, so the 5V I/O pin is at 1.67 V, which is probably still okay, though well above TTL's 0.8V threshold, and precisely at 5V CMOS's ⅓Vdd threshold.

(It works a little better if you use 470R instead of 1k in the middle. You don't get a lot of noise immunity but you do get some, at least with CMOS thresholds.)

Now, 3V3 is almost precisely 5V CMOS's ⅔Vdd threshold, so you

may be able to just use a wire. And going in the other direction, 5V to 3V3, you can just use a 2:1 voltage divider, when a simple current-limiting resistor isn't enough. (Generally 3V3 input pins, when they aren't actually 5V-tolerant, have some specified limit on how much current they can sink from a higher-voltage place, like half a milliamp or something.)

I just hadn't realized that a non-bogus level shifter could be so simple and passive.

Active and bidirectional level shifters

Here's a more elaborate circuit shifter on Falstad's circuit simulator:

```
$ 1 0.000005 10.20027730826997 50 5 43
R -48 160 -80 160 0 2 40 1.65 1.65 0 0.5
g 160 240 160 288 0
t 128 192 80 192 0 1 0.642686555624928 0.6693572978862428 100
t 192 192 240 192 0 1 -4.2135673793996755 0.02667074272282007 100
r 128 192 240 48 0 4700
r 192 192 80 48 0 4700
w 80 48 80 176 0
w 80 208 80 240 0
w 240 208 240 240 0
w 240 240 160 240 0
w 160 240 80 240 0
w 240 48 240 176 0
r 80 48 80 -16 0 1000
R 80 -16 80 -48 0 0 40 5 0 0 0.5
r 240 48 240 -16 0 1000
R 240 -16 240 -48 0 0 40 5 0 0 0.5
r -48 160 32 160 0 470
w 128 192 128 160 0
w 128 160 32 160 0
368 240 48 336 48 0 0
368 -48 160 -48 96 0 0
```

This circuit uses a simple RS latch made out of four resistors and two bipolar transistors to do level-shifting in a way that has, I think, some bidirectional potential. The output on the right generates 0.06 V or 4.24 V according to the state of the latch, and the 3V3 input (connected through a lower-value resistor) is strong enough to overpower the latch's state.

Now, the reason I say this has some bidirectional potential is that, if the 3V3 pin isn't driving anything (you disconnect the square-wave source), then you can drive the "output" directly with 0V or 5V, which is also enough to overpower the latch. The leftover part is that your tri-stated 3V3 pin isn't being driven to 3V3; it's being driven to just a Vbe.

This can be remedied by driving the feedback override in the level shifter through a voltage divider:

```
$ 1 0.000005 10.20027730826997 50 5 43
r -144 -16 -144 128 0 15000
```

```
r -144 128 -144 240 0 22000
t -144 240 -80 240 0 1 -4.763777148265759 0.10791218667768503 100
g -80 256 -80 304 0
r -80 224 32 224 0 33000
t 32 224 80 224 0 1 0.529525420053551 0.6374376053938159 100
g 80 240 80 304 0
r -80 224 -80 -64 0 1000
R -80 -64 -80 -96 0 0 40 5 0 0 0.5
R 80 -64 80 -96 0 0 40 5 0 0 0.5
r 80 -64 80 208 0 1000
w 80 208 128 208 0
w 128 208 128 -16 0
w 128 -16 -144 -16 0
w 128 208 208 208 0
s 208 208 208 -64 0 1 true
R 208 -64 208 -96 0 0 40 5 0 0 0.5
s 208 208 208 288 0 1 true
g 208 288 208 304 0
368 208 208 272 208 0 0
w -144 128 -208 128 0
s -208 128 -208 288 0 1 true
g -208 288 -208 304 0
368 -208 128 -272 128 0 0
s -208 128 -208 16 0 1 true
R -208 16 -208 -32 0 0 40 3.3 0 0 0.5
```

This is basically the same latch circuit as before, but now the 3V3 input is connected to the middle of the feedback resistor, so it sees a lower voltage.

Of course cloudevil points out that seven discrete components are probably not cheaper than a level shifter chip! And I might want to consider under what conditions it might oscillate. But mostly I just thought it was an interesting way to tackle the problem. And, amusingly, it's only one more component than two purely passive unidirectional level shifters made from three resistors each — but it contains two transistors, which are both more expensive and often slower than resistors.

(It's kind of goofy to describe a resistor by itself as "slow" or "fast"; it's really the whole circuit. But I did it anyway.)

DocScrutinizer05 proposed the following alternative bidirectional level-shifter circuit, based on a design he saw from NXP, which uses 8 components instead of 7, but only a single transistor:

```
$ 1 0.000005 10.20027730826997 50 5 50
f 544 240 544 304 32 3 0.02
v 240 160 320 160 0 0 40 2.3 0 0 0.5
w 320 160 352 160 0
w 544 160 544 240 0
g 240 160 240 192 0
R 656 192 768 192 0 0 40 15 0 0 0.5
r 352 160 352 304 0 10000
w 352 304 528 304 0
s 304 304 336 304 0 1 true
s 592 304 624 304 0 1 true
```

```
w 656 192 592 192 0
w 336 304 352 304 0
w 304 304 256 304 0
g 256 304 256 384 0
w 560 304 592 304 0
r 592 304 592 192 0 10000
g 624 304 624 400 0
d 400 160 352 160 2 default
r 544 160 592 160 0 47000
w 592 160 592 192 0
d 544 160 496 160 2 default
d 432 160 400 160 2 default
d 496 160 432 160 2 default
o 14 128 0 4354 19.999206274746793 0.0001 0 2 14 3
o 7 128 0 4354 4.600396862626605 0.0001 1 2 7 3
```

I need to think more about how this works; it's explained in NXP appnote 10441.

Topics

Merkle ropes

Kragen Javier Sitaker, 02020-10-09 (15 minutes)

(From a discussion with Nick Johnson, though any errors are my responsibility, not his. And probably a lot of this is not novel, being implicit in Okasaki at least.)

Bitcoin and similar systems include a hash of the previous block in each new block, forming a chain of blocks. But verifying that any particular previous block in the block chain is truly an ancestor of what you believe the tip block to be requires verifying all the blocks in between.

(Often a more interesting question is whether a current candidate tip block really does have the height it claims to have. The approach outlined below is not primarily concerned with that question, but it does permit fast noninteractive probabilistic proofs of it.)

Merkle blockchain ropes and logarithmic-time appending

What we would ideally like is a Merkle *tree* of the previous blocks instead of a linear Merkle block *chain*. Both can be thought of as Cedar's "ropes", which represent sequences as the leaf nodes of a binary tree; in OCaml:

```
type rope = Leaf of string | Fork of rope * rope
```

Or, more abstractly, a rope of some arbitrary type `'a` or α:

```
type 'a rope = Leaf of 'a | Fork of 'a rope * 'a rope
```

Sometimes there's a third alternative:

```
type 'a rope = Leaf of 'a | Fork of 'a rope * 'a rope | Empty
```

In these Merkle graphs, the pointers are realized as message digests rather than memory addresses. A chain is just the degenerate case where the binary tree is pessimally balanced. What we want is to incrementally rebalance the tree as we append to the end of it.

If you have 8 leaf nodes to form into a perfectly balanced binary tree, you need to construct 7 internal nodes ("forks"). The first fork can be added when you add the second leaf, but the second fork cannot be added until you add the fourth leaf, at which point you can also add the third fork. Then the fifth leaf does not enable adding any new forks, so it would have been okay to wait to add the third fork until then. The sixth leaf enables adding the fourth fork, the seventh leaf does not enable adding any new forks, and the eighth leaf makes it possible to add the fifth fork (over leaves 7 and 8), the sixth fork (over forks 4 and 5), and the seventh fork (over forks 3 and 6).

You could think of these fork nodes as being an "index", like that in a relational database, of the block chain; but, instead of allowing

you to answer queries quickly, they allow you to construct proofs quickly.

On average, you only need to add one fork for each leaf after the first, but sometimes you cannot add it immediately. So if you add at most one fork with each new block (which adds a leaf), you will start to fall behind. But you fall behind only by a logarithmic amount; the 256th leaf enables the construction of 8 new forks, so at that point the perfect binary tree will be 7 blocks delayed from the state of the blockchain.

In essence each new fork consolidates the two previous smallest remaining binary trees into a larger binary tree; the trees can be held in a stack of fully compacted trees and a queue of possibly not fully compacted trees. Before adding a new leaf, we compact two trees if possible, then add the leaf to the queue. Here's what the `stack; queue` state looks like as we add the first 40 leafnodes, one at a time:

1: ; 1	11: 8 2; 1	21: 16 2 2; 1	31: 16 8 4 2; 1
2: 2;	12: 8 2 2;	22: 16 4; 1 1	32: 16 8 4 2 2;
3: 2; 1	13: 8 4; 1	23: 16 4 2; 1	33: 16 8 4 4; 1
4: 2 2;	14: 8 4 2;	24: 16 4 2 2;	34: 16 8 8; 1 1
5: 4; 1	15: 8 4 2; 1	25: 16 4 4; 1	35: 16 16; 1 1 1
6: 4 2;	16: 8 4 2 2;	26: 16 8; 1 1	36: 32; 1 1 1 1
7: 4 2; 1	17: 8 4 4; 1	27: 16 8 2; 1	37: 32 2; 1 1 1
8: 4 2 2;	18: 8 8; 1 1	28: 16 8 2 2;	38: 32 2 2; 1 1
9: 4 4; 1	19: 16; 1 1 1	29: 16 8 4; 1	39: 32 4; 1 1 1
10: 8; 1 1	20: 16 2; 1 1	30: 16 8 4 2;	40: 32 4 2; 1 1

The new leaf can embed the fork within it, thus "signing" the fork at insignificant extra cost. The pointers in the fork can be the hashes of the two blocks containing its child nodes, annotated with bits to indicate whether the pointers are leaf pointers or fork pointers. You might think that when the fork is included, this eliminates the need to include the hash of the previous block in the new block, because the previous block can be reached by following the right-child pointers down the tree of forks; but in fact the new fork might not include the previous block. So you still need the previous block pointer.

Moreover, to permit efficient traversal of the trees, each fork also needs to include a pointer to the previous fork on the stack, if any. In the above example, the fork containing the first 16 leaves is created with the 19th leafnode, but is not merged into a 32-leaf fork until leafnode 36. When you're looking at leafnode 35, how are you going to find the older 16-node fork? You need a pointer back to leaf 19.

From a rope perspective, your sequence of blocks is a concatenation node of the stack and a queue; the stack is either empty or a concatenation of a stack of all the balanced trees before the last, and the last balanced tree; each balanced tree is either a leafnode or a concatenation of two balanced trees; and a queue is either empty or a concatenation of a queue and a leafnode

From the rope perspective, this structure is just a rope; the distinctions between the state, stacks, queues, and balanced trees can be entirely implicit in the structure. A state is a fork of a stack and a queue; a stack is empty or a fork of a stack and a balanced; a balanced is either a leafnode or a fork of two balanceds; a queue is empty or a

fork of a queue and a leafnode. So the "type" of each fork (state, stack, queue, or balanced) is encoded by its parent's type and which side it's on.

Each new block encodes, in a sense, a new balanced and a new stack with it as the right child and a new queue with a new right child (which is the block itself). The only funny business is that, as I've described the queue above, consuming items from the front of the queue requires reconstructing all the forks inside the queue, and these forks are not recorded at all in the blockchain.

The length of the queue is bounded by the height of the tree, so it's only logarithmic. With an ephemeral data structure (rather than an FP-persistent structure like an orthodox rope) you could do the queue operations in constant time, making the whole node-append operation constant-time. This isn't important for block-chain applications, but it could be useful in other rope applications.

However, although *appending* to the structure can be thus made constant-time, it still takes logarithmic time to *query* it, which is usually more common, so I think logarithmic time for appending will almost always be good enough.

Code for the table

The data for the above table was produced by the following code; I then formatted it into columns:

```
s, q = [], []

for _ in range(40):
    if len(s) > 1 and s[-1] == s[-2]:
        s[-2:] = [s[-1] + s[-2]]
        q.append(1)
    else:
        q.append(1)
        if len(q) > 1:
            n = q.pop(0)
            n += q.pop(0)
            s.append(n)
    print('{}:'.format(sum(s) + sum(q)),
          ' '.join(str(i) for i in s) + ';', ' '.join(str(i) for i in q))
```

Probabilistic chainheight validation

As for the probabilistic proof mentioned earlier, if you annotate each fork with the total amount of hashing (PoW) work in its leafnodes, then you can choose a random leaf node to which to validate the path down from the current state, for example in proportion to the amount of hashing it is claimed to represent. If you've been fed a fake blockchain that pretends to represent 10% more work than it really does, then you'll have a 10% chance of finding a discrepancy, say where a fork's work total isn't the sum of its children's work totals, or where the previous node in the tree isn't actually the predecessor it specifies. This assumes the thief can't predict which random node you'll try to validate.

By validating several randomly chosen leafnodes this way, each in

logarithmic time, you can achieve an arbitrarily high confidence level that the whole blockchain is valid, as it claims to be. For example, after validating 44 random leafnodes in this way, you would have a 99% chance of finding one that was in the faked 10%, I think. If only 1% was faked, you would need to check 459 random leafnodes to have 99% chance of detecting the fraud.

But this is a fairly small cost. Suppose there are 8 million leafnodes, so you may need to go through as many as 44 forks on your way down to a leaf; if all you have is a sequential file of blocks and an index of it by hash, this could take 87 random accesses, about a second on spinning rust with 8-ms seek times. So you could finish the 459-leafnode probabilistic validation in under ten minutes.

Less pessimistically, you could arrange the 8 million forks into a B-tree; each consists of, say, a 32-byte hash for each of its two children, 64 bytes in total. On spinning rust, you might use a 524288-byte treenode size, which can holds 8192 forks, really 8191: 13 levels of the tree. The up to 23 stack items and the up to 23 queue items can be held in RAM. So validating each leafnode involves reading two B-tree blocks and the leafnode, 3 random accesses rather than 87, plus checking the hashes. So you can finish all 459 validations in 10 seconds rather than 10 minutes.

On SSD you can use 16384-byte B-tree nodes — 255 forks each, 8 levels of tree — and access them in 100 µs each. So traversing 23 levels of the binary tree requires traversing only 3 levels of the B-tree plus a leafnode, 400 µs, so your access time for the 459 leafnode validations is 180 milliseconds.

(Hmm, actually I realize I didn't include the chainheight annotation in the sizes of the forks, but the difference is not very large.)

Application to LSM-trees

Much of the logic above isn't concerned with precisely what operation we do to merge together two trees into a larger tree. In the above, it was simply a matter of constructing a new fork with the two trees as children, a constant-time and constant-space operation. But it's all highly suggestive of Lucene's merge-based approach, also used in LevelDB, nowadays called a "log-structured merge tree". In an LSM tree, when you merge two 16-item segments into a 32-item segment, you have to read through each of them sequentially in order to sort them into order by key, perhaps constructing some kind of skip file or index to enable rapid random access by key.

The amount of work becomes smaller if we use N-way merges: instead of first merging 1+1 = 2, then 2+2 = 4, then 4+4 = 8, then 8+8 = 16, then 16+16 = 32, which involves appending an item to a new segment 63 times, we directly merge 1 + 2 + 4 + 8 + 16 = 32, thus only doing it 32 times. We can also exchange some more variability in query time for a lower index construction time, by delaying merges for a longer time, say until the new segment will be 4× or 8× larger than the largest old segment being merged, not just 2× as in the examples above.

The larger point, though, is that LSM-trees fundamentally involve more work per item than just concatenating them into ropes. But it's

only logarithmically more work, and we can spread it evenly over the time when items are added in an analogous way. The temporal sequence won't be exactly the same: generating the 32-item merged node, done atomically at block 36 in the above example, will take 16 times as much work as generating the 2-item merged nodes, which in the above example are created at blocks, 2, 6, 37, etc.

So we will generate these larger merged nodes gradually, over the course of adding several leafnodes. And I think the amount of work we do per leafnode will gradually increase, but only logarithmically. I haven't worked out the reality yet. Each new leafnode might include some logarithmically large set of pointers to segments it's merging and cursor positions in them, and a chunk of merged data.

What does this have to do with blockchains? Well, you could plausibly collectively generate a queryable database index in this way as part of a blockchain, but efficiency is in some sense not the point of a blockchain — inefficiency is. A blockchain is designed to make it infeasibly inefficient to violate the established consensus rules on who owns what.

Outside of blockchains, I suspect that LevelDB does in fact work this way in order to avoid unbounded pauses when inserting and deleting.

Topics

A seamless CMG-driven walker

Kragen Javier Sitaker, 02020-10-11 (updated 02020-10-12)
(6 minutes)

A control moment gyroscope, CMG, or гиродин is a device used, typically in pairs, to control the attitudes of large spacecraft, as an alternative to reaction wheels. Someone has built a cute little cube-shaped robot called Cubli that can get up and walk by rolling from one corner to another using reaction wheels, so the robot has no externally protruding components or pivoting joints, other than the bearings for the reaction wheels themselves. It would be interesting to do something similar using CMGs.

The idea is that you'd have something like an opaque, matte tetrahedron with rounded corners, or perhaps some more irregular shape, of a few hundred mm in diameter, mostly made of some very light material; however, inside of it would be hidden two or more gimbaled gyroscopes, whose rims would contain most of the mass of the whole device, as well as several motors, a battery, and control electronics. If the walker decides to start walking, it spins up its CMGs and starts torquing them in order to stand up on, for example, one corner, and move around.

As a concrete example, perhaps the height of the tetrahedron would be 720 mm, but the last 100 mm of the point are rounded off. The inscribed sphere has diameter 360 mm, so perhaps the largest gyroscope has diameter 350 mm and can rotate to any angle within; the rim of its perfectly toroidal rotor is, say, 100 mm in minor diameter, with a circular cross section centered at 170 mm from the center, thus 340 mm major diameter. The torus has a volume of about 8.4 liters, so if it is mostly or entirely made of lead, 11.34 g/cc, then it will weigh about 95 kg, too heavy for most humans to lift. I think it should be possible to build the rest of the machine — frame, bearings, smaller gyros, gimbals, motors, cables, etc. — under 5 kg. So 95% of the machine's mass will be in its primary gyro, which can safely be spun at some 30 m/s.

At this speed its kinetic energy would be some 43 kJ, enough to drain a 2400-milliamp-hour USB power pack just to spin it up, on the order of 50 g of Li-ion battery. (The 10050-mAh USB power pack next to me weighs 205 g.) So probably 1 kg or more would need to be battery.

So clearly this beast could have an angular momentum to be reckoned with, and with the appropriate gearing, motors, and secondary CMGs, would have no trouble at all slowly lifting itself off the floor to stand on one point, or walking across the floor on two of its points. It could perhaps walk up and down stairs, light up, vibrate, make sounds, and, by balancing on one point, serve as a cocktail-party coffee table, though keeping it from being a very noisy and vibration-heavy table would take substantial engineering of the bearings.

In addition to walking, it could tilt a bit to one side and rotate on its rounded point, which would cause it to roll across the floor rather

than merely walking.

Equipped with a sense of touch to feel things placed on top of it, it could balance a ball on its center, constantly tilting slightly to nudge the ball back toward its center. It might even be able to simultaneously engage in such a motion while balancing an object on its top.

If you wanted it to carry things around, though, a more useful polyhedral shape would have an edge between two vertices, usable for walking, opposite a flat face, so that it could walk while objects remained on its upper surface mostly by friction, minimally tilting back and forth to shift its weight between these two feet. An equilateral triangular prism, for example, would work; so, too, would a square pyramid, though there is only one angle for such a pyramid at which one of its triangular faces will be horizontal when its center of gravity is over the opposite edge.

More irregular shapes would offer more versatility.

A prototype of 1% the mass could probably be constructed. Instead of weighing 100 kg, it would weigh about 1 kg. You'd scale it down by a linear factor of, say, 0.22. So the tetrahedron would be 158 mm tall, the incribed sphere 79 mm diameter, the rotor rim 22 mm thick, centered 37 mm from the center (74 mm major diameter). This rim has a volume of 89 ml, 1.01 kg. If we also scale down the rotor linear speed and leave its angular speed alone, it's only going 6.6 m/s, which is still 1700 rpm. (I guess I should work out what the scaling laws for CMGs are; I think that small CMGs are worse than reaction wheels.) The kinetic energy has dropped even more: now it's only 22 J. I feel like this would still probably work but you might need to spin up the motors.

The total mass left over, if it scaled the same way, would be about 50 g.

A collection of such contrivances possessed of concave surfaces, or even surfaces that could be horizontal near floor level, could climb atop one another; with adequate friction, they could then function as if parts of a single body, with rolling contact between them rather than flexible joints or bearings.

Topics

Rigid glider

Kragen Javier Sitaker, 02020-10-12 (1 minute)

In CMG Walker (p. 420) I wrote about how to make a mobile machine with a totally rigid and seamless surface: no joints, no wheels, no flexion.

A glider or submarine of this sort could perhaps work exclusively by moving its center of mass internally, like the OrbSwarm robots, rather than using reaction wheels or CMGs; this is a significant part of how hang-gliders work. In the case of a submarine, a similar tactic can alter its buoyancy, which can be used to propel it forward, though this requires either a flexible surface or some transport of mass, probably water, across the surface, for example through a hole.

Topics

- Contrivances (p. 790) (44 notes)
- Mechanical things (p. 795) (19 notes)
- Flying machines (p. 957) (2 notes)
- Submarines

Skip list variants

Kragen Javier Sitaker, 02020-10-12 (4 minutes)

Skip lists are generalized sorted linked lists that incorporate logarithmic-time insertion, deletion, and search. In their normal form, the link pointers point in only one direction, let's say forward. The generalization is that each node has a "height", a positive integer chosen randomly from an exponential distribution, which tells *how many pointers* it will contain. The height-1 pointers are the usual linked-list pointers, but the other pointers skip nodes; a height-P pointer, instead of pointing to the next node, points to the next node *of height P or greater*.

Skip lists are transposed, randomized B-trees. You take a B-tree, replace all of its adjacency relationships within B-tree nodes with a previous or next pointer — turning the node into a doubly-linked list — and then replace all child pointers with new adjacency relationships. (You can treat the child pointers as going to the first piece of the split-up child node, or the last — it doesn't matter, as long as the other pieces are reachable from it.) Now you have a skip list, but an unorthodox one, with fairly regular spacing between nodes of the same height.

The B-tree invariant that each node has between N and M keys translates into a new invariant in these lists: if you can reach a previous node by following a single pointer at height P, then it will take between N and M hops by following a single pointer at height P - 1.

You can do the usual operations of insertion and deletion, including appending, on such a list without violating its invariants. Whenever you create a node containing a pointer at height P - 1, you must check to see how many nodes surround it that do not contain a pointer at height P. If there aren't enough, you need to extend the node to height P, then repeat the process.

(It is by no means clear that this is superior to the standard approach of picking a random height for each new node, but it gives a one-to-one mapping to B-trees maintaining their usual invariants. Clearly you can use the skip-list random-level approach with the B-tree layout as well.)

A somewhat inconvenient aspect of the standard skip-list implementation is that the nodes are many different sizes, which is more difficult for dynamic memory allocation (and some type systems) to handle. It's reasonable to ask if there's a way to avoid this.

If the distribution of heights is 50% 1, 25% 2, 12½% 3, and so on, then the mean height will be 2. You could maybe provide 2 link fields per node, which sounds like... a binary search tree!

So here's an idea. What if you assign heights as before, but only have two pointers per node: one to the immediately next node, the other to the next node *of a larger height than the immediately next node*? (I also thought of "of a larger height than the current node," and "preceding a node of a larger height than the immediately next node" and such things.) This ought to permit rapid traversal in essentially the

same manner as an ordinary skip list.

XXX try it

Clearly you can also just sort of bulldoze the pointers of an ordinary skip list across the following nodes, so a node of height 4 will have a heigh-4 pointer, then be followed with a node with a height-3 pointer, then one with a height-2 pointer, and then if either of those nodes had a height of more than 1 themselves, then the corresponding pointer will be pushed onto the next node. The difficulty with this variant is that insertion and deletion is no longer cheap.

Topics

VGA oscilloscope?

Kragen Javier Sitaker, 02020-10-13 (5 minutes)

I mentioned the CCD oscilloscope idea from Dercuano on ##electronics, and Stipa pointed out that the whole laser mirror thing was far more complex than needed. You can just use a VGA monitor!

In more detail, the problem to be solved is that of building an oscilloscope out of discarded junk, for ghettobotical purposes. With an oscilloscope, designing, debugging, and characterizing electronics becomes enormously easier; some old vacuum-tube oscilloscopes had a rolloff (-3dB point) of 10MHz, but the standard basic analog oscilloscope for decades was 20MHz. So expedients like wiring your signal source to your sound card through a resistor and some limiter diodes are grossly insufficient: your sound card is 20kHz, three orders of magnitude crappier.

Once you can bring the problem into the digital domain, everything else becomes easy, because you can do it as slowly as you like. But how can you digitize a 20MHz signal if you don't have access to the market? 50Msps ADCs are quite scarce in the municipal waste stream.

Analog oscilloscopes achieved these bandwidths by using a cathode-ray tube; the beam's Y deflection was what you observed on the screen. So you might think that you could salvage an old TV and use its CRT, but TV CRT beams are magnetically deflected, while oscilloscope CRT beams are electrostatically deflected, allowing deflection frequencies two or three orders of magnitude higher at reasonable voltages. TV CRTs are not going to be useful for that; they're designed for horizontal scanning with a 15.734-kHz sawtooth wave (for NTSC; PAL and SECAM vary slightly.)

However, the beam *intensity* is modulated at higher frequencies, up to about 5 MHz in the case of NTSC, PAL, or SECAM TV; more promisingly still, tens of MHz in the case of computer monitors, whose VGA connectors still accept analog waveforms for red, green, and blue. A monitor running at 1600×1200 at 85 Hz, which was high-end in the 1990s but quite likely junk today if it's a CRT, is drawing 163 million pixels a second, so it can "sample" signals up to 80 MHz or so. A lower-end 60Hz 1024×768 monitor is only drawing 47 million pixels per second, but that's still enough for more than the 20 MHz for a basic oscilloscope. Pixel brightness is not linear in signal voltage, being transformed by the "gamma curve", but it's not outrageously nonlinear. And an RGB VGA monitor, like nearly all of them, will "sample" three channels at once, which is very respectable indeed.

You also need electronics to generate sync pulses, of course.

So how do you get the signal off the screen? With a camera, of course. Many-megapixel cameras are now common on cellphones, and they are fast enough, high enough resolution, and clean enough (with good signal-to-noise ratios) that you can use them capture individual pixels from an individual video frame off a monitor.

(My previous proposal used a laser diode and two spinning mirrors to scan the laser beam across a reflective screen — certainly doable with discarded materials, but substantially more difficult.)

Once you have that data, it's a Simple Matter of Programming to find the individual scan lines, compensate for lighting and viewing-angle variability, undo the gamma-curve transformation, look for triggers in the signal stream, and draw a waveform with the resulting data.

There are some limitations of this method. All data is lost during the horizontal blanking interval and vertical blanking interval, unless you are driving three separate monitors out of phase to avoid this. (Two monitors is not enough to eliminate these dropouts because the VBI of each monitor will inevitably contain many HBIs of the other.) There are cases where this matters: where you're watching for a single-shot event and you really need a lot of data on both sides of it, for example. Calibration may drift, since slight brightness changes don't affect the normal use of monitors; occasionally displaying a calibration frame or two instead of the input signal may help with this.

Aside from its use as electronics test equipment, this approach should work well for software-defined radio, in which case it is probably possible to use analog-domain downconversion and frequency filtering to smear the signals of interest around in the time domain enough that the HBI and VBI are less problematic.

Topics

Wire machines

Kragen Javier Sitaker, 02020-10-13 (updated 02020-12-31)
(12 minutes)

The bent-wire locking toggle used to close the tops of some bottles is a beautiful little piece of mechanical design, and CNC wire benders can produce such devices at high speeds and high precision. And bending wire manually is often expedient and relatively easy. Galvanized mild steel wire is widely available at very low cost, and rusty mild steel wire is constantly discarded by the side of the road. And in both US English and Argentine Spanish, there's a common figure of speech for an expedient, low-quality solution that refers to it: "duct tape and baling wire" and "atar con alambre", respectively.

Straightening with two plates

You can straighten a piece of wire by rolling it back and forth between two flat surfaces after approximate straightening; it's helpful to feed it in incrementally so that you're only straightening a little bit at once. I've used a slab of granite and the back of a scrap ceramic floor tile in this way, getting bent-up wire to sub-diameter deviations from straightness over short lengths. It's easier if your wire is round rather than needing to be twisted to approximate roundness.

Work-hardening and annealing

Mild steel can't be hardened martensitically, but it can experience considerable work-hardening, and this plays a significant role in shaping things from mild-steel wire by hand, for example with needlenose pliers.

It's usually a difficulty: once you bend some wire, you ain't never gonna unbend it, because the place you bent is harder than the rest of the wire. But if you buy some wire, it doesn't come straight. It's coiled, if you're lucky. If you grab discarded rusty wire off the street, it's gonna be bent all to hell, and that's going to introduce some unpredictability.

CNC wire-bending machines deal with this problem by bending the wire back and forth in all directions with rollers, in order to straighten it and harden it uniformly before they shape it. This way all the wire is hardened to a greater extent than the kinks were beforehand. Even if you can do this by hand, you may regret it, because the uniformly hardened wire is uniformly harder to bend, too.

Maybe a better approach is to anneal the wire. Annealing steel properly is a pain in the ass. You need to austenitize it, which takes temperatures ranging from 738° to 900° for mild steels, with the highest temperature being pure iron; in itself that's not too hard, but then the recommended cooling rate is about 11 mK/s, or 11°/kilosecond. Not only does this take all day, it also demands a degree of temperature control well beyond the capability of a stove burner or butane torch, which can easily heat the wire up to 1000°, high enough to anneal steel, but probably can't cool it down any

slower than about 1°/s, 90 times faster.

However, mild steel is less demanding than quench-hardenable steels in its annealing requirements, and I think you can get substantial softening at these much higher cooling rates; the literature seems to claim that anything from 5°/s to 200°/s should be just fine. I took a straightened wire and heated two spots on it on the stove burner, one to orange heat and the other a bit below, for 20 minutes, then let it cool over the course of about a minute, so on the order of 20° to 50° per second. Subsequent bending happened preferentially at the annealed spots, showing that they were softer than the rest of the wire.

Work hardening is not all bad, though. When you bend wire, it bends into a smooth curve instead of kinking like a drinking straw because the parts that have bent least are the softest. (In a drinking straw, or in a collapsible tube in general, the parts that have bent most are the softest, which is why they bend even more, forming the kink.) And, by twisting two wires around one another, we can make them dramatically harder, in the sense of having a much higher yield stress, though no stiffer elastically.

Creepage and nicking

Design for bending wire, like design for bending sheet metal, requires a creepage allowance; tight bends are not ideal points, but curves. Tighter and more precisely placed bends can be achieved, at the expense of strength and stiffness, by nicking the wire at the desired bend location. This permits bends whose radius is less than a single wire diameter. However, imprecision in nicking (I'm using needlenose pliers) can result in either a failed bend or a cut wire. This takes about a joule with these needlenose pliers and this baling wire I have here, which is about 1.7 mm in diameter; but it depends on the shape of the nicking dies (these have about a 90° dihedral included angle) and on the hardness of the wire. I measured this energy by dropping a known weight of 827 g from a known height of 100 mm onto the handles of the needlenose pliers.

Compliance

Wire is very springy. Although baling wire is much less springy than music wire or other hardened steels — both in the sense of elongating less before beginning plastic deformation, and in the sense of having lower moduli of elasticity — it is still capable of storing enormously more energy elastically per volume than other everyday materials like wood, fired clay, glass, paper, cardboard, or PET. (Rubber beats it, though.)

As a result, to an enormous extent, you can change the compliance of a wire structure by changing its geometry. If you can use an arbitrarily large amount of wire, you can get an enormous but not arbitrarily large amount of compliance into a given space. (If you can make the wire arbitrarily thin, you can get to an arbitrarily large amount of compliance.)

In the opposite direction, achieving high rigidity is more difficult. To some extent you can make progress with triangulated structures made of short struts, and wires as straight as you can get them; but

connecting wires together with high rigidity without welding is very difficult. Twisting two wires together increases the thickness of a strut, but at the same time it turns each wire into a helix, which is less rigid.

When two or more wires are twisted together, they spring back when you stop twisting, loosening their grip on one another. Thereafter they are connected with a screw joint with substantial backlash. I think it's possible to take the backlash out; the usual way is to preload two coaxial screw joints with a spring under axial compression, though tension works just as well. I haven't tried this with twisted wire yet, but I suspect it will be difficult.

Wireframe design as graph traversal

Suppose you want to make a regular rhombic dodecahedron of a given size with a minimal amount of wire and no unnecessary cuts, outlining its edges in wire. By Euler's theorem about the bridges of Königsberg (pbuh), you cannot do it with no cuts and no repeated edge traversals; for all that the rhombic dodecahedron has 6 vertices of degree 4, it also has 8 vertices of degree 3, and each of those vertices requires either the beginning or ending of a wire, or a double traversal of an edge, for example by twisting two wires together.

So, one way to do it is to build the shape from 4 wires, each starting and ending at one of the degree-3 vertices.

But suppose instead we duplicate one of the edges out of these degree-3 vertices, representing our double traversal of an edge. This converts it into a degree-4 vertex, but the duplicated edge connects at the other end to one of the old degree-4 vertices, which now becomes a degree-5 vertex — so far we are no better off than before! But that vertex is exclusively connected to (old) degree-3 vertices, so we can simply duplicate one of its edges to one of them, converting what was originally a degree-4 vertex into a degree-6 vertex, and eliminating the other degree-3 vertex.

Now we have only 6 odd-degree vertices. By repeating this process twice more we can reduce it to 2, where the wire is free to begin and end, curled around the kink between two more edges; and our wireframe is complete.

A similar process can be done with wireframes in general, but it clearly demonstrates the difficulties introduced by odd-degree vertices in hand-twisted baling wire. If you are designing a wireframe *de novo* rather than using one Archimedes thought up, you may prefer to avoid odd vertices entirely. For example, if you create a so-called "geodesic dome" by subdividing the triangles of an octahedron rather than the traditional icosahedron, you will have greater inequality in strut lengths, but no odd vertices. The curvature will be provided ultimately by degree-4 vertices rather than degree-3 or degree-5 vertices.

You might think that an alternative basis for subdivision may be the cuboctahedron, the polyhedral dual of the rhombic dodecahedron, since all its vertices are degree-4; but, by the time you eliminate its square faces as an offense against omnitriangulation, you are back to having a subdivided octahedron. It's a distinction without a difference.

Dimensional stability

Even baling wire is fairly dimensionally stable, due to steel's temperature coefficient of expansion of some 12 ppm/°, compared to most other common materials. Brick and glass (even soda-lime) exceed its dimensional stability, as does the lengthwise dimension of wood.

Kinematic pairs and flexures

It's easy enough to twist some baling wire into a tight helix that another piece of wire can fit through, forming a cylindrical joint. But its translational motion is highly unsatisfactory, due to the usual problems of translational motion in kinematic pairs between rigid bodies, but exacerbated by the compliance of the wire. As a revolute joint it works better, but it's somewhat prone to unwanted translational motion. I think it's usually possible to overcome this, but it takes some doing.

In many cases, though, I think a flexure design may be more suitable for the characteristics of the wire. Clothespin-like spring clips bent from a single piece of wire, with or without a helical flexural pivot, are well-known, and of course the paper clip is a flexural bent-wire machine ubiquitous throughout the modern world.

Ends and burrs

Snipped wire ends have sharp burrs on them. Deburring is an option. With needlenose pliers, though, it is generally easier to curl them into a tiny circle so that these burrs are up against the edge of the wire a few millimeters away. This is enough for many purposes.

Paperclips use smoothly curved radii to grab paper without tearing it, a potential problem despite their neatly deburred ends.

Topics

Thermistors, resistance temperature detectors, and other thermal sensors

Kragen Javier Sitaker, 02020-10-14 (updated 02020-11-06)
(12 minutes)

How can you measure a wide range of temperatures without exotic materials? High-temperature thermocouples and thermistors typically use things like platinum and iridium because they don't oxidize in air.

A cartridge heater, as I understand it, consists of a resistive heating element coiled up, packed into a solid insulator made of something like MgO or BeO, combining very high resistivity and high dielectric strength with high thermal conductivity; all packed into a metal can to exclude oxygen and avoid abrasion of the relatively weak insulator (very weak in the case of MgO).

You can get fairly fine (≈100μm) copper wire by, among other things, untwisting Ethernet cables or some power cables. Annealed copper's temperature coefficient of resistivity α is about 0.0039, comparable to platinum's. You could maybe stick such a coil of fine copper wire inside some kind of insulator, such as quartz sand, and do a four-wire resistance measurement on it, using more copper wires twisted onto it.

Exploring a candidate design: millikelvins to 1000° should be attainable

To be concrete, 200 mm of 100-μm-diameter annealed copper wire should be about 439 mΩ at room temperature, about 610 mΩ at 120°, and about 2.15 Ω at 1020°, close to copper's melting point of 1085° (though probably α diverges a bit by then). If you were to dump a 1-amp pulse through it through two of the wires for 100 μs, then it should develop a voltage on the order of 400–2000 mV across the other two wires; 100 μs is plenty of time to measure the voltage and current waveforms even with a 44.1 ksps ADC, much less a 1 Msps ADC.

What error should we expect? Let's suppose we can calibrate out the thickness of the thermistor, copper's nonlinear temperature-resistance curve, and the impurities in the particular copper wire we're using.

An input impedance of 1 MΩ on whatever voltage measurement thing you have connected to the other two wires would give you a parasitic offset current on the order of 1 μA, causing a relative error of 10^{-6} or so on the temperature measurement. You can get op-amps with much lower offset current than that; they would provide a more precise measurement. For example, TI's datasheet for the LM741 — not a spectacularly low-offset-current op-amp — says its offset current at 25° is typically 20 nA, worst-case 200 nA, and the whole bias current is typically 80 nA, worst-case 500 nA.

The current you measure on the current wires outside the device would differ from the current through the "thermistor" only due to parasitic capacitances around the thermistor; if these were, say, 10 pF, then the current error would be around 10 pA, a relative error of 10^{-11}. For this error to rise to 10^{-6} you would need 1 μF, a totally unreasonable amount of parasitic capacitance.

Parasitic inductance is larger, but it's less of a problem. Suppose you have 100 nH of parasitic inductance in each pair of leads, which you can reduce by keeping them as closely antiparallel as possible, and another 100 nH in the "thermistor" itself (which you could reduce by alternating winding directions, FWIW). This would be very large; axial-lead TVS diodes are around 10 nH while surface-mount devices are closer to 4 nH. And suppose the current ramps up to 1 A in 10 μs. This produces 10 mV of voltage drop on the current leads and 10 mV of inductive voltage in the thermistor superimposed on the actual desired voltage. The first doesn't matter; the second, without any further measures, would produce a temperature error of some +6° during that time. Since the current through the voltage-sensing wires is sub-microamp, the error from their inductance is a million times smaller.

The reason this is less of a problem is, first, once the current finishes ringing, for the rest of the 100-μs pulse, the inductance introduces no error at all; second, you're probably applying more voltage than that, so the ramp time is even faster; third, if you take two or more samples of the voltage and current during the pulse, it's easy to decompose the results into an inductive component and a resistive component.

Heat is the biggest error. 1 amp through 439 mΩ is 439 mW, which is a lot. In 100 μs, it's only 43.9 μJ, which is not a lot. But this is a small wire; at copper's density of 8.96 g/cc, it's only 14 mg of wire. Still, at copper's molar heat capacity of 24.440 J/mol/K, which divided by its atomic weight of 63.546(3) g/mol gives 0.38460 J/g/K, that's a temperature rise of 8.2 millikelvins, 14 μV. Crudely, this would introduce an error of about 3×10^{-5} in the measured voltage, about 30× larger than the other sources of error discussed above.

However, this heating error is zero at the beginning of the measurement process, and on short time scales it is proportional to the integral of squared current so far. (Over slightly longer time scales the heat will start to leak away.) So I think it's probably relatively straightforward to cancel this source of error, at least down to the same 10^{-6} level as the offset current. If your ADC is fast you might be able to just use a 3-μs current pulse.

The whole process of generating heat and conducting it away from the wire into the rest of the device can be fairly closely approximated as a linear time-invariant process, so we can estimate its impulse response function, then cancel it, probably down to a 10^{-8} level, particularly if you stimulate it with random current pulses rather than regularly spaced ones.

It might be hard to get such a low error in the measurement of voltages and currents, due to things like drift, noise, and the temperature coefficients of the measurement device. Still, it's achievable, and even an 0.1% error in voltage would be an error of

1 K, which is enough for most of my purposes.

All of this is really quite astonishing, and it makes me wonder if I am overlooking some kind of enormous source of error.

It's important that thermal expansion and contraction not change the resistance of the current path, for example by tightening and loosening connections. For the voltage-measurement path, this is less important. So probably the right way to make the device is by tying and/or soldering some voltage-measurement wires onto the relevant part of the current-measurement wire.

Alternatives

Carbon, being a semiconductor, has a temperature coefficient about an order of magnitude higher, and it can also handle higher temperatures than the 1085° of copper. You can get carbon-composition and carbon-film resistors already in sealed ceramic "cases", though generally those aren't built to handle more than about 300°. (The paint on traditional axial-lead packages is organic and burns. Maybe surface-mount packages are better.)

Steel wire and stainless steel wire have temperature coefficients similar to copper's, but can withstand higher temperatures without melting, typically up to 1400° or more, though they must be protected from oxygen at these temperatures. They are better than copper for use in hot-wire cutters (at temperatures low enough that they don't burn) because they are stronger. They're typically thicker, though; stranded picture-hanging wire from an art store has the thinnest stainless-steel wire commonly encountered.

Tungsten wire is also similar, and comes much thinner in conveniently packaged US$1 quartz-halogen lightbulbs to protect it from oxygen, though perhaps not all of those envelopes will themselves withstand temperatures over 1000°. (And these lightbulbs can do double duty as heating elements.) However, they don't normally have four wires. I have such a 28W 220V lightbulb here; at room temperature (20°?), this untrustworthy multimeter, which measures a short circuit as 10Ω, measures it as 147.3–148.3Ω, so maybe it's actually 138Ω or so. In theory tungsten's α should be around .0045, according to the Wikipedia table above, so at 100° it ought to be about 188Ω and measure about 198Ω. I boiled it for 20 minutes on the stove, miraculously without breaking it, and it measured 185–186Ω. But the multimeter had gone to measuring a short circuit as 3Ω, so maybe that's really about 183Ω. Not the most spectacular agreement with theory but it's within measurement error. Next day short-circuit reading is 0.3Ω and bulb reading is 141Ω. (To work at 28W at 220V it would need to increase to 1700Ω; if it's at 1100° then its average α over that interval would need to triple, which is totally plausible. Some 70W-rated halogens I bought measure 51Ω so it's even more plausible.)

Silicon carbide, commonly used for heating elements and abrasives, is also a semiconductor, and can withstand even higher temperatures (decomposing at 2830°), protecting itself against air up to about 1600° by oxidizing the surface to silica. I don't know what its temperature coefficient of resistance is, but I imagine that it's negative and larger than copper's.

When using the same element to sense temperature and raise it, you can either measure the average temperature to which the element is heated (though not the peak, which is what you usually really want) by measuring the voltage and current while it's on, or the temperature of its surroundings by turning it off and letting it equilibrate.

RTDs

This turns out to be what's standardly called a "resistance temperature detector" or "resistance thermometer", which is commonly used for high-precision temperature measurements, usually with a precision more like 60 millikelvin than 1 millikelvin. Copper is indeed a material commonly used for them, though usually not above 150° because of oxidation; other common materials are nickel, despite its poorer coefficient, and platinum; the most common configuration is Pt100: platinum with 100Ω at 0°. This higher resistance reduces the influence of lead-wire resistance.

I don't think copper is going to be able to reduce silica or magnesia, but if that turns out to be wrong, as a last resort you could pack it in copper oxide.

Wikipedia explains that above 660° your sensing filament tends to get contaminated with metal from the can it's in, so people usually use thermistors at higher temperatures despite their poorer precision and repeatability, typically having errors on the order of 2°. Because RTDs typically have microscopically thin wires flapping around, like lightbulbs, they tend to be less durable than thermocouples. The alternative to flapping around is to use a "wire-wound RTD" where the wire is wrapped around an insulating winding core, but the winding core generally has a different thermal coefficient of expansion from the metal, which causes strain in the metal, which introduces at least nonlinearity and maybe nonrepeatability.

The problem with picking up metal from the can seems like it could be reduced by using a can made of fused quartz or sapphire or something, or using carborundum and periodically burning off any metal that has deposited.

Many thanks to Greg Sittler for the very informative discussion!

Topics

Atkinson differential blower

Kragen Javier Sitaker, 02020-10-14 (updated 02020-12-31)
(10 minutes)

The four-stroke Atkinson differential engine uses a clever arrangement of linkages to move its two pistons in a single cylinder, out of phase with one another. It can use reed valves, like a two-stroke engine; both the intake valve and the exhaust valve are at the same end of the cylinder.

The orthodox Atkinson cycle

The sequence is as follows. Suppose the intake and exhaust valve are at the right end of the cylinder, and the pistons are close together, with the intake and exhaust valve between them. First the pumping piston, on the left moves to the left, opening the intake valve and sucking air and fuel into the cylinder. Then the working piston, on the right, moves to the left, covering the valves and compressing the fuel-air mix against the now-stationary pumping piston. Then the spark fires, and the working piston moves to the right, providing the power stroke. Finally the working piston passes the exhaust valve, the hot gas escapes through it, and the pumping piston follows it to the right, preparing to pump in the new fuel-air mixture.

Quiet piston blowers

On ##electronics cloudevil was talking about piston-powered blowers, displacing multiple liters of air per stroke, suggesting that they could be much quieter than traditional types of blowers, though of course they'll still produce turbulent airflow. But such a blower will fail to be quiet if it's using poppet valves or reed valves, since those produce an impulse every time they open and close.

You could imagine using a round, triangular, or teardrop-shaped port or hole in the side of a cylinder as a valve; when the piston passes over it, it opens or closes, but not impulsively. But this has two problems.

First, in the Atkinson engine design, the intake and exhaust valves are at the same end of the cylinder, so if there's no reed valve or anything, they'll be open at the same time. This is no way to make an air pump.

Second, if there's a pressure difference across the valve as it opens, the airflow through it will still start suddenly and with a lot of turbulence, so it will be noisy, though maybe less so than a reed valve.

Both of these problems can be solved by moving the valve openings to opposite ends of the cylinder and redesigning the cycle for pumping without such events.

First, the left cylinder is at the left end of the cylinder, just to the left of the intake port, and the right cylinder is to the right of the intake port. Second, the right cylinder moves to the right, almost to the exhaust port, at the right end of the cylinder; in this way air is

sucked into the cylinder from the intake port. Third, both cylinders move in unison to the right, first closing the intake port and then opening the exhaust port. Fourth, the right cylinder stops, while the left cylinder continues moving to the right, expelling the air through the exhaust port. Fifth, both cylinders move in unison, close together, back to the left.

These movements can easily be scripted by cams to minimize the bandwidth of the pistons' movements, thus eliminating the direct production of sound above, say, four times their movement frequency, which might be 2 Hz. Then only turbulence and surface roughness are left as noise sources.

Why four times? If the pistons moved back and forth in unison a single time, or with a simple difference in phase, they could move in a perfect sinusoid, thus generating no sound from their sheer movement at frequencies higher than their movement frequency. But the movement I described above is not simply sinusoidal, so it would involve some harmonics. 8 Hz is inaudible, but 20 Hz or more might be audible and highly annoying. Of course air turbulence and surface roughness will generate higher-frequency noise no matter what the cylinders' movement.

You might be able to find a lower-displacement purely-sinusoidal movement pattern with the right characteristics — most crucially that the cylinders be closer together when moving leftwards than when moving rightwards, and the same distance apart when the intake port closes and when the exhaust port opens. If a single sinusoid can't do the job, you might be able to find something that only uses two or three harmonics rather than four, and thus enable higher operating frequencies while remaining purely infrasonic, maybe up to 5 Hz or 10 Hz. But I'm confident that with four harmonics you can do it.

Engines

You can take the same approach and apply it to the problem of making an engine, too, in the sense of a device that converts heat energy into mechanical energy.

The most direct approach is to feed steam or pressurized air into the intake; then the suction stroke becomes the power stroke, but you still have a sharp noise when the pressurized air gets over to the outlet port, and that noise of course represents wasted energy. Similarly, when the small space between pistons moves over the intake port, the pressure in it is the exhaust pressure, which is low. If you revise the cycle somewhat so that the space between pistons is zero when they open the input port, then continues expanding after the input port is closed, you can get all the adiabatic energy in the input working fluid. Alternatively, rather than making the space zero, it could simply be smaller than its size when the exhaust port closed by an amount sufficient to bring its pressure up to the intake pressure.

However, if you want it to be a standard four-stroke internal-combustion engine, you need the following cycle:

- Intake: pistons separate, pulling in some fuel-air mixture from the intake port.
- Close intake: pistons move in unison to the right, closing the intake

port.
• Compression: pistons approach one another, compressing fuel-air mixture.
• Expansion: spark fires, pistons separate, allowing expansion.
• Open exhaust: pistons move in unison to the right, opening the exhaust port.
• Exhaust: left piston continues moving to the right, reducing volume of chamber and expelling exhaust gases.
• Close exhaust: pistons move in unison to the left, closing exhaust port.
• Return: pistons move in unison to the intake port.

This requires some fancy camwork; the followers pressing on the cams during the expansion stroke is what drives the engine forward. Alternatively it may be possible to design a linkage that does all this, which may be beneficial in terms of being easier to adjust.

Adjustment of the cycle might be useful for a variety of reasons:

• Exhaust gas recirculation: by including some exhaust gas from the previous stroke in the mix, you can reduce pollution by lowering combustion temperatures and improve engine efficiency. In this engine you can simply not bring the pistons all the way together, so some exhaust remains trapped between them when they return to the intake.
• Atkinson cycle: by using a greater expansion ratio than compression ratio — that is, by compressing the gas less in step #3 than the combustion products expand in step #4 — you can improve energy efficiency. Any pressure difference remaining between combustion products and the outside world when the exhaust port opens in step #5 represents a waste of energy and a source of noise.
• Anti-Atkinson cycle: by using a greater compression ratio than expansion ratio, you can get more power at the expense of lower efficiency, more noise, and less complete combustion.
• Throttling: by increasing or reducing the amount of fuel-air mix brought into the cylinder on each stroke, we can increase or reduce the engine's power output further. This eliminates the need for a butterfly or other throttle valve and the associated vacuum losses.

A second cylinder configured as described above can be used to harvest further energy from the exhaust, in the manner of double-expansion or triple-expansion steam-engines; this will also reduce noise further, as the second cylinder serves as a sort of muffler for the first. (You could also use such an engine as a muffler for a conventional internal-combustion piston engine, surely not a novel idea, since triple-expansion steam-engines go back generations.)

An engine with two or three cylinders cascaded in this way can be fitted with valves to redirect the flows of gases to answer demands that change from moment to moment: now a triple-expansion engine with a single combustion chamber, one cylinder feeding into the next, for greater efficiency, now a three-cylinder engine with all three cylinders burning fresh fuel for greater power.

By virtue of moving the expanding gas chamber down the cylinder as it expands, the loss of heat to the cylinder walls is reduced — while this benefits an internal-combustion engine less than a steam-engine,

it may pose difficulties in keeping the hotter parts of those walls lubricated.

As the space between the pistons can be reduced indeed to zero, requiring no accommodation for the opening of valves, the spark-plug is redundant in these classes of engines; it can be designed to ignite purely by adiabatic heating as in Diesel's engine.

Topics

Inspiration

Kragen Javier Sitaker, 02020-10-15 (3 minutes)

I've seen the kingdoms blow like ashes in the winds of change, yeah, but the power of truth is the fuel for the flame. So the darker the ages get, there's a stronger beacon yet. … If the world is night — shine my life like a light.

— Indigo Girls, "Let It Be Me"

When your vision stays clear in the face of your fear; when you see turning out the lights, which is their only power, when we stand like spotlights in a mighty tower — all for one and one for all! Then we sing the common call.

— (same)

Everything that we call Invention, discovery in the higher sense, is the ultimate outcome of the original perception of some truth, which, long perfected in quiet, leads at length suddenly to and unexpectedly to productive recognition.

— Goethe, quoted by Reuleaux in the introduction to Kinematics of Machinery, as translated by Alexander Blackie William Kennedy

Help, I'm a rock! Somebody! Please! Help, I'm a rock!

— The Mothers of Invention

Thank you so much for your warm hospitality this evening. We are so grateful to play for you. Thank you so much, friends; we are so privileged to be able to gather in moments like this when so much of the world is plunged in darkness and chaos.

— Leonard Cohen, at a concert in London 2008, before playing *Anthem*

Don't dwell on what has passed away. … Ring the bells that still can ring; forget your perfect offering. There is a crack in everything. That's how the light gets in.

— Leonard Cohen, *Anthem*

Staring at the blank page before you, open up the dirty window, let the sun illuminate the words that you could not find; reaching, so close you can almost taste it — release your inhibitions! Feel the rain on your skin; no one else can feel it for you, only you can let it in. No one else, no one else can speak the words on your lips. Drench yourself in words unspoken, live your life with arms wide open; today is where your book begins. The rest is still unwritten!

— Natasha Bedingfield, *Unwritten*

Wandering Star, for whom it is reserved the blackness, the darkness, forever. … And the masks that the monsters wear to feed upon their prey. … Always doubled up inside, taunted, cruel.

— Portishead, *Wandering Star*

Topics

Oscillating flexion

Kragen Javier Sitaker, 02020-10-15 (updated 02020-10-16)
(11 minutes)

A single perfectly rigid reciprocating rod transmits power only intermittently; near the moment between its movement in one direction and the movement in another, the power it transmits falls to zero, unless it experience an infinite acceleration in that movement, and thus too an infinite force, if its mass not be infinitesimal. This must happen twice per cycle.

Two such rods can transmit power continuously by virtue of being out of phase. If the resistance against which they push and pull be variable, they can even transmit constant power; this is the case for example if they are each in simple harmonic motion in quadrature and connected to equal idealized dashpots; for the power transmitted by one varies as $k \sin^2 xt$, the other as $k \cos^2 xt = k(1 - \sin^2 xt)$.

A more interesting case is where two pushrods push and pull a crankshaft rotating at a constant speed: a pushrod whose displacement at time t is $y \sin xt$, whose velocity is thus $xy \cos xt$, has a lever arm for its crank of $k \cos xt$. If its force should vary in proportion, $z \cos xt$, then the power it transmits will be $xyz \cos^2 xt$, just as in the dashpot case, and summing with a pushrod in quadrature provides constant power transmission.

Now we can see also that the torque on the crankshaft from the first pushrod is $kz \cos^2 xt$, and so the pushrod in quadrature brings us also to constant torque.

If we want to transmit power some distance, pushrods are impractical because they will buckle unless supported. If we use four cranks instead of two, we can substitute four longitudinally oscillating cables rather than two pushrods, transmitting power only through two of the cables at any given moment, and clearly getting the same constant power transmission.

This is temptingly analogous to two-phase AC power transmission, strongly suggesting the possibility of improving efficiency by using three cables oscillating at phase angles of 0°, 120°, and 240°, rather than four quadrature angles. The situation is not totally analogous; if the tension on each cable varies in proportion to its velocity as before, the power transmitted by a cable at phase angle φ is ReLU(sin $xt + \varphi$)^2. I think these still sum up to a constant, but if not, varying the tension according to some different curve can clearly provide constant power transmission over three cables. I'm not entirely sure that this would also provide constant torque, but actually I don't care.

That's because the reason I think this is interesting is for continuously transmitting power between parts of a flexure; flexagons aside, most flexures can't manage continuous rotation, so the traditional forms of continuous mechanical power transmission such as belts, shafts, and gears are unhelpful. Cable transmission also has the generally-noted property of fitting conveniently into smaller spaces.

Such oscillating cable transmission might use radically

non-sinusoidal force and displacement profiles — for example, think of four cables, of which at any given time three are under high tension transmitting power, while the fourth is under low tension but higher velocity, being returned to its starting position. This sort of thing can, I think, increase the material efficiency of such power transmission systems.

A cable tensioned at 2 GPa traveling at 1 m/s is transmitting 2 gigawatts per square meter, which is 2 kilowatts per square millimeter. If the cable can only be safely tensioned to 200 MPa, it can only transmit 200 W/mm² at this speed. In theory you could increase the cable speed arbitrarily — for example, your wimpy 200 MPa cable would transmit 4 MW/mm² at 20 km/s — but, aside from concerns about sonic booms and friction heating, the process of reversing the cable's direction of movement involves accelerating it, which also requires force, and that force adds to the cable's tension.

However, for cable lengths short compared to the free breaking length of the cable material, this acceleration can reach many gees before the load on a cable during the return stroke equals the load during the power delivery stroke. Indeed, the number of gees is precisely the ratio of the cable length to the free breaking length. So, for example, gel-spun UHMWPE at 3 GPa and 0.96 g/cc has a free breaking length of 319 km, so a 100-mm-long cable can be accelerated by pulling on one end at about 31 million gees before it breaks.

Return strokes faster than some 10–100 times the acoustic length of the cable will result in waves noticeably propagating back and forth in the cable, unless it's properly acoustically terminated to prevent such reflections. This is precisely analogous to the phenomenon in RF electronics with unterminated transmission lines, but I don't know of any electrical power transmission scheme for which it is essential to keep the mean drift velocity of the charge carriers in the cable to zero, so the analogy only goes so far.

To reduce the total displacement of the cable and thus permit higher instantaneous velocities and power densities, it might be desirable to transmit power at much higher acoustic frequencies than this, such that indeed many wavelengths of the tension wave fit within the cable. Taking this step renders the power transmission capability of the cable independent of its length.

As one point among many in this possible design space, consider four parallel cables oscillating in simple sinusoidal motion in quadrature, carved from ASTM A36 steel, each 100 μm square. According to Machine Teeth (p. 251) A36 yields at 250 MPa, and its Young's modulus is 200 GPa. It weighs 7.9 g/cc, so if they're 200 mm long, each weighs about 16 mg. Suppose they're oscillating longitudinally at 3 kHz by a distance of 100 μm. Their peak speed is only 1.9 m/s, but their peak acceleration is 36 km/s/s, 3600 gees, which requires about 570 mN peak acceleration force, working out to 57 MPa, comfortably below A36's yield stress. Such a stress will elongate the wire by 0.03%. If the one or two wires actively transmitting power at any given time are loaded sinusoidally up to 125 MPa, the peak power transmitted on a wire is 2.4 watts, but about 1.1 W of that is "returned" during the return stroke for the wire. I think this means that the total net power is consistently about 1.3 watts for

the whole assemblage.

This is a promising but not outstanding amount of power to transmit through something the thickness of a beard hair. How can we increase it?

If we increase the distance of displacement, holding constant the frequency, then we linearly increase the velocity and thus the power, at the expense of linearly increasing the acceleration. If we increase the frequency instead, while holding the displacement constant, then we linearly increase the velocity but quadratically increase the acceleration.

If we use a (nonexistent) material of the same physical characteristics except that it had a lower density, or if we were transmitting over a shorter distance, it would reduce the force required to accelerate the wires; we could trade that for a reciprocally higher acceleration. If instead we used a material with the same physical characteristics except a higher yield stress, such as a harder steel, then we could proportionally increase both the acceleration *and* the load force.

The totally free way to improve the system seems to be decrease the frequency linearly while increasing the distance quadratically, thus holding the acceleration constant while increasing the velocity and thus the power reciprocally with the frequency. So maybe we could increase the total displacement to 10 mm while decreasing the frequency to 300 Hz, increasing the power to about 13 watts, a much more respectable power level. Further development in this direction would seem to be dependent on very precise motion control to avoid having to space the wires further apart.

If we drop the density by a factor of 8 while multiplying the yield stress by 10, then the first would allow us to increase the frequency and power further by a factor of 2.8 (increasing the acceleration by 8), and the second would allow us to increase the frequency by a factor of 3.2, but also the tension by a factor of 3.2, increasing the power by a factor of 10. I think this means you could transmit 360 watts of mechanical power through your new hair-thin gel-spun UHMWPE cable, at least for 20 mm.

(In practice I doubt this could be sustained continuously; the imperfectly elastic nature of any real material results in some heat dissipation from stretching and relaxing it, and UHMWPE is, I suspect, worse in this aspect than steels. Moreover it melts at a very low temperature; tens of milliwatts of such dissipation would likely be fatal.)

With the acoustic traveling wave mode of energy transmission mentioned earlier, the maximum power per unit area is the energy's elastic energy density (half its yield stress multiplied by its yield strain) multiplied by the speed of sound in the substance.

Consider how the example above compared to electrical power transmission through an electrical cable of comparable size. If you enclosed three 40 AWG solid copper wires, each 80 microns in diameter, in 20 microns of Kynar insulation, you would have a similar-sized cable. You could run three-phase AC power over it at whatever frequency was convenient; your phase-to-phase voltage would be limited by the breakdown voltage of 40 μm of Kynar, while

your RMS current would be limited by the resistance per meter and heat dissipation per meter of the cable at Kynar's maximum service temperature of 149°. You can decrease the radius of the conductor and increase the thickness of the Kynar by an equal amount, thus increasing the voltage linearly but also increasing resistance as the square of the remaining wire.

The dielectric strength is supposedly "1700 V/mil" for a short period of time; if we figure that's 1000 V/milli-inch in practice, that's about 40 volts per micron, so 1600 volts peak, 1100 VAC RMS phase-to-phase. 40-gauge wire is rated for 90 milliamps over short runs. I think this ends up at a few hundred watts, too, so it's the same ballpark.

Topics

Reuleaux

Kragen Javier Sitaker, 02020-10-15 (updated 02020-10-18)
(19 minutes)

I'm reading Reuleaux's 1874 book on machine kinematics, or rather its English translation, and I thought I'd make some notes. Reuleaux formulated kinematics as it is studied today, in terms of lower and higher kinematic pairs with varying degrees of freedom, such as revolute, prismatic, and cylindrical joints.

This is one of the rare cases where the Google scan is of less bad quality than the Archive's own scan; kinematicsofmachooreulrich.pdf has the left side of many pages cut off, while so far Google's kinematicsmachi01reulgoog.pdf seems okay. As always it is lower in resolution and color depth, but so far this seems to be an improvement, accelerating as it does the rendering. It seems to lack OCR text, though, while the Archive's own scan has somewhat passable OCR. The OCR is still not good enough to copy and paste, but sometimes it may be faster to edit the results than to retype from scratch.

The Archive's scan also does a less appallingly bad job of capturing the figures, though it still leaves something to be desired. (Compare p. 31, for example; 49 of 646 in the Archive's scan, 52 of 651 in Google's.)

Mupdf in particular handles the total absence of OCR text rather disappointingly when you initiate a search, freezing for a few minutes before it comes to the end of the document.

Many aspects of the typography and usage of this work deviate from modern standards, but none more glaringly than its use of increased letter spacing for emphasis, where modern practice, or even common 19th-century English practice, would use italic. This was of course necessary in German blackletter typesetting, which entirely lacks italic, but I suspect that even the original German version of this work was set in modern humanist letters rather than blackletter. (Perhaps I should go back and check.) I often fail to notice this on the first reading of a passage.

Where I've quoted passages using this form of emphasis below, I've replaced it with italics.

The typography *does* use italics to distinguish mathematical variables and references to points in figures.

Introduction

It's interesting that Reuleaux traces the development of kinematics though a whole series of 19th-century researchers, including (to my surprise) Ampère. I had sort of thought that the systematic study of kinematics had sort of been stuck at the level of Archimedes' "simple machines" still taught in the vulgar elementary schools: the inclined plane, the screw, the lever, the wheel and axle, the pulley, and the wedge. But indeed Reuleaux outlines the development of the ideas by Galileo, by Hâchette, by Borgnis, by Lanz, by Monge, by Willis, by Ampère, by Newcomen, by Watt, and by a dozen others, prior to

himself. Justifiably, though he characterizes all their theories as "wrecked".

It's intriguing to see the etymology and sense development of "automatic" and "automation":

Moreover, as a further fruit of this uncertainty there has been an attempt to construct yet another special study which demands mention. This is the so-called *Automatics*, the study of the realization in mechanism of motions either supposed or given by mathematical expressions. For this further attempt at separation we have to thank the engineer E. Stamm, who wishes again to divide his subject into pure and applied parts, as fully described in his *Essai sur l'automatique pure*, 1863.

Of course the term "automaton" is much older.

When Reuleaux speaks of "the real end before us — the progressive development of the machine," it is difficult to avoid being reminded of Forster's *The Machine Stops* from a few years later.

The modern development of flexure design methods is a disquieting echo of the disjointed and "wrecked" development of mechanisms of which Reuleaux complains; the stamp-collecting botany of the *Handbook of Compliant Mechanisms* closely resembles what he deplores in Monge's and Lanz's work. This is startling since, of course, we have a fairly complete theory of elastic deformation as well as well-developed practical numerical methods of computation. But in Reuleaux's time, the analogous theory of rigid motion was similarly well-developed; you can design a planar four-bar linkage to tween between three predetermined positions with compass and straightedge, and the whole lovely theory of quaternions commonly used for rigid motion in modern games engines predated this book by decades. As Reuleaux laments:

Here again is a point from which the weakness of the method hitherto employed can be surveyed at a glance. Its difference from the ideal method is not that it employs the inductive instead of the deductive method ; that would indeed be no advantage, but it might still be defensible. No, it has been entirely unmethodical. It has chosen no fixed method of investigation, or rather, it has not found any in spite of zealous search...

Rhetorically, this Introduction is a masterpiece; Reuleaux promises the world to his diligent student, while warning them of the difficulty of their quest and deprecating their existing knowledge:

We often do not know ourselves how closely wedged-in our ideas are by the boundaries which education and study have drawn around us. ... all these pile up mighty hindrances. I cannot therefore shorten the way, although the truths to which it leads are of great simplicity. ... to conclude in the words of Göthe, " What is not understood is not possessed."

It's interesting to note that in Reuleaux's time it was not yet known how recent the introduction of the wheel was, due to the still primitive state of the archaeological science; he claims "carriages" are known from the oldest times, while in fact they seem to postdate even such recent artifacts as the Great Pyramid.

His account here of Watt's parallel motion is sticking with me. I keep thinking about it.

General Outlines

Nature of the Machine-Problem

I just can't get over the evocative nature of the prose, and the sharp

contrast with the uniform oatmeal mediocrity of modern academic writing:

So soon as the force Q begins to act it calls forth in the interior of the wheel, the shaft and the supports, internal molecular forces, opposite in direction and exactly equal to it. ... there are opposed to all external forces others concealed in the interior of the bodies forming the system,...

after which he proceeds to quote a verse from Schiller (!).

Reuleaux actually comments a bit on the question of flexures here: "In actual machines...we use however only those materials which...alter their form under the action of external forces very little, so little that the corresponding variations from the original form may be neglected."

I enjoy his description of the strength or rigidity of bodies as being "latent forces", in contrast to actually existing "sensible forces".

His definition of a machine is thought-provoking, as much for the aspects it omits (material handling, information, control, energy, force, thermodynamics, friction, efficiency, metrology, electricity, programmability, strength, tolerances, troubleshooting, wear, reliability, cost, clamping) as for what it includes:

A machine is a combination of resistant bodies so arranged that by their means the mechanical forces of nature can be compelled to do work accompanied by certain determinate motions.

It's a damned sight better than "a combination of simple machines", though, and it doesn't exclude the use of kinetic energy or hydraulics as many inferior definitions have.

It's interesting that this more or less explicitly excludes the boilers and condensers of the steam-engines Reuleaux has given such prominent placement earlier.

The Science of Machines

Here we see again the primacy Reuleaux grants to position and motion, but also what a high priority he places on forces. So far there's no mention of compliance and backlash except as an enemy; nothing about springs or preloading.

General Solution of the Machine-Problem

Even with the limited definition Reuleaux has given, this section heading seems amazingly ambitious.

The moving bodies are prevented, by bodies *in contact with them* [emphasis in original]... this contact ... must take place continually, ...

This seems to exclude backlash and clearances entirely!

I have no idea what a "plummer block" is.

This is the section where he introduces the **kinematic pair**.

It's fascinating that in is Figure 4, the first kinematic pair he introduces (before he even introduces the term) is not any of the basic ones, but rather a block sliding in an irregularly curved plane channel as the channel constrains it to rotate unevenly. Then on the *next* page his figures 5 and 6 are classic screw and prismatic joint.

Aha, and after Figure 9 he explains the meaning of "higher kinematic pair":

Accordingly the reciprocal combinations of two elements gives us again *a pair of*

elements, which may differ from either of the single pairs of which it is composed.

(And, as with his Figure 4 example of the weird sliding block, his first explicit example of a higher kinematic pair is an arbitrary weird thing, rather than something well-known.)

I really appreciate that Reuleaux is giving one or two examples first and only then giving definitions.

Also, he defines *kinematic chain* here, far more clearly than I'd heard it defined before. So I'm finding this reading very rewarding so far.

I'd never thought of this before, but there's an interesting analogy between how changing the position of one element in a closed kinematic chain (a term I'm not sure I understand completely yet) alters the position of all the others, and how changing the current or voltage of one element such as a capacitor or inductor in an electrical circuit alters the current and voltage of all the others. In both cases you can't analyze a single component in isolation; to see the system non-wholistically you must find other elements into which to decompose it, such as eigenstates or linearly superimposed circuits. But I don't know what those components could be for either a closed kinematic chain or a nonlinear electrical circuit. (And I don't think anything similar to the linear decomposition of circuits is known for kinematic chains, or things like the linkage-synthesis papers I've been reading would be using it.)

The aside "a cylindrical pin fitting a corresponding eye, the axes of all being parallel" immediately calls to mind Hoberman spheres and the like; if the axes instead all intersect at a point, you get the same sort of linkage, but thus constrained to a sphere rather than a plane. And of course there are other relationships that can provide more interesting movements. (Hoberman's insight was similar but had to do with lines drawn transversely through the centers of multiple such joints.)

Aha, and here we have another key definition of Reuleauxian kinematics:

A closed kinematic chain, of which one link is thus made stationary, is called a mechanism.

At this point it's worth comparing 19th-century European automata with 19th-century Japanese automata; while both treat the outward form and appearance of the mechanism as being as important as its position, work, and movement, the Japanese automata extensively used cable drive, including on uneven cams, while the European automata like Jaquet-Droz's Writer used exclusively rigid bodies. Reuleaux is seeking to embrace not only pulleys and belts in his analysis but even hydraulic machines, but he has barely mentioned anything flexible or hydraulic yet, despite the world-shaking importance of the mechanization of textile manufacture in the late 18th and early 19th century, in part by Vaucanson himself.

So we can see in some sense why Babbage's work was so unsuccessful and why the Writer would not be equaled for some 150 years, roughly until Bush's Differential Analyzer. To keep the world of machinery intellectually manageable, its paradigm deliberately excluded consideration of the aspects crucial to such achievements. In Reuleaux so far I see not even the smallest hint of the approach that animated Zuse, Shannon, and Turing.

Here we see boldface in the text for a definition:

The effort thus applied performs mechanical work which is accompanied by determinate motions; the whole, that is to say, is a **Machine**.

We also see the first mention of clocks, though nothing about the questions of metrology, calibration, and cancelation of errors that had enabled the conquest of Latitude with machinery a century before, as well as the first mention of balances (in the sense of a weighing-scale); though he does promise to "consider these questions systematically" later.

One of the words whose usage has apparently changed significantly since this book was translated is "empirically", which seems here to mean "by trial and error".

He does finally mention "the spinning-machine" in this section, and the sewing-machine. The thread and fabric of the sewing-machine seems to fit very poorly into Reuleaux's theory — the whole machine exists to put them into certain regular motions with respect to one another, but their motion is neither rigid nor determinate, and substantial parts of the machine exist purely in order to modulate the friction and tension on them.

Phoronomic Propositions

"Phoronomic" is explained in the first section here; it means something like "concerning the study of the geometric motions of rigid bodies".

Preliminary remarks

Phoronomy, etc.

Relative motion in a plane

Newtonian relativity.

Relative motion of two points is considered only in terms of their distances, thus omitting rotation — rotational orientation is not considered proper to a point. Motion of a point relative to a plane it's moving in requires only its distances from two points, not three, permitting a reflection ambiguity — perhaps chiral orientation is not considered proper to a plane.

Interestingly, it seems that when he considers motion of a line relative to another line, he *does* consider sliding motion along the line to be motion.

Temporary Centre; the Central Polygon

It's surprising that he considers "the Phoronomics of point-systems" to be "exhausted" by propositions in two dimensions only!

This idea of a "temporary center" of rotation, for any arbitrary rotation, is very interesting. It reminds me of the compass-and-straightedge four-bar-linkage construction technique.

"Open polygons" occur frequently in computer graphics but they are usually called something else.

I don't understand this "reciprocal polygon" yet. Why exactly must it exist, and be reciprocal?

§ 7. Centroids; Cylindric Rolling

Oh, this makes the reciprocal polygon thing a bit clearer, though it's still not clear why it must exist. Boy, this sure is a different meaning of "centroid" than the one I'm familiar with, though not *completely* unconnected — the centroid of a uniform shape is its center of gravity, which is the center on which it turns when it has only angular momentum.

All this terminology of "common, curtate, and prolate trochoids" is new to me, and I think I'll have to look it up.

When he says, "*All relative motions of con-plane figures may be considered to be rolling motions*", he seems to be omitting the possibility of pure translational motion, which is the limit of rolling about an infinitely distant center.

It's nice that he is getting into three-dimensional objects moving now, but I can't help but wonder if there are possibilities of three-dimensional motion other than the possibilities arising from extending two-dimensional motions prismatically, although Reuleaux claims there aren't. For example, motions where the instantaneous axis of rotation twists from one moment to the next. And what is the instantaneous axis of rotation of a common screw?

It is now apparent that the reciprocal polygons of which Reuleaux was speaking may have quite different shapes.

§ 8. The Determination of Centroids

The construction he gives for finding the second point M_1 in Figure 19 (p. 66) escapes me at the moment. I will have to come back to this.

Oh. Because M_1 is notionally part of the same rigid body as *P and* Q, its distances to them are not changed by rolling; at the *given point in the movement has been brough into coincidence with* ***O*$^*_1$, so those distances are the distances from the moved positions*** P_1 and Q^*_1.

Figure 20 in the Google scan (p. 67, 88/651) is partly illegible; it is identical to Figure 10. In the Archive's scan it is perfectly legible. Incidentally, I think this page shows that the scans were taken from two separate physical copies, as the Archive's scan is stamped, "Reese Library of the University of California".

Aha, and here he confronts the question of pure translational motion, using the device of "infinitely distant points", which seems pretty kosher, although not all this geometry so far is projectively invariant. (Or is it?)

Figure 22 is blowing my tiny fucking mind.

§ 9. Reduction of Centroids

I am totally not understanding this. I thought the motion uniquely determined the two centroid curves? But now we can invent more centroids for the same motion?

Here he confronts the question of parallel motion in all of its complexity.

Hmm, there's a clue about these "secondary centroids" in the part where he talks about gear teeth (though he never says "gear", always "spur-wheel").

§ 10. Rotation about a Point

Now Reuleaux seems to be taking up the question that had worried me earlier, that of motions in space that are not merely the extrusion of motions in a plane.

§ 11. Conic Rolling

Whoa, trippy, dude.

§ 12. Most general Form of the Relative Motion of Rigid Bodies

Okay, great, now we're getting into screwing motions?

Oh wow.

§ 13. Twisting and Rolling of Ruled Surfaces

Holy shit.

Hmm, he has a hypoid drive here, although he doesn't call it that.

Pairs of Elements

§ 14. Different Forms of Pairs of Elements

Topics

Intervals and gradients

Kragen Javier Sitaker, 02020-10-16 (4 minutes)

Darius Bacon linked me to https://twitter.com/paniq/status/1317063053339406336:

@paniq@mastodon.social / L. . Ritter / @paniq:

apparently these good people solved the explosion in steps when the ray gets close to the surface during SDF sphere marching (thanks to @Atrix256) https://diglib.eg.org/handle/10.1111/cgf13951

i must say i'm a little angry with myself because i spent hundreds of hours on getting around this exact problem and couldn't find a better way.

there's only two shadertoys yet https://shadertoy.com/view/WdKczW

the classical sphere marching method: lipschitz continuity guarantees that the blue line never intersects the diagonal of the red stepping function, which always moves as many units forward as it has measured upwards.

plotting the first derivative of our curve (green), the light green lines designate the global lipschitz limit that the function is guaranteed never to overstep.

the purple lines bracket the local upper and lower bounds of the first derivative within a local region.

plotting the limits as as a wedge of tangents, all positive, we see that the region can't possibly contain a root, and thus deem it safe to skip the entire interval.

whereas in this less beneficial interval, our tangent bundle can only guarantee us safe passage up until the cyan colored point.

this region here however allows us to make a step twice as a large as a simple SDF lookup would permit us.

it is easy to see how this method becomes particularly useful when we are grazing surfaces with low curvature, as the tangent width narrows, and we can practically perform an interation of the newton-raphson method.

the authors don't do a good job of showing these connections. the provided formula is effectively a version of the newton-raphson method which uses a gradient interval instead of a local gradient.

to compute the gradient interval, you use a combination of automatic differentiation and interval arithmetic on the first derivative. here are primitives for both:

- Interval Arithmetic https://shadertoy.com/view/lssSWH
- Derivative Arithmetic https://shadertoy.com/view/4dVGzw

the new arithmetic primitives will likely look a lot like joint ranges from revised affine arithmetic (https://shadertoy.com/view/4sV3zm), except that we get a right facing interval cone instead.

a much more important consequence of all this is that the lipschitz continuity requirement $|f'(x)| <= 1$ no longer applies as we always see a local gradient cone. you can use this to trace any old implicit function, not just only distance functions.

bonus: with 1 more sample at the right end of the interval, we can reuse the same gradient interval, flip it, and use it to truncate our extrapolated destination. in this case, we discover that we can skip the entire interval rather than having to jump to the cyan point.

the right hand samples amortize over time, as we can reuse them in the next iteration.

Matt Keeter @impraxical:

Dumb question: if you can do interval arithmetic on your function, why not directly check the interval result in the target segment (to see if it contains 0), rather than the interval of the gradient in the segment?

@paniq:

what do you do when the interval contains a possible root?

Ah, that explains why this is useful :D

I agree that affine arithmetic could tell you how far you could safely walk; I'm curious to see how "point + gradient range" (this paper) compares to "affine region" (RAA), in terms of ease of computation / tightness of bounds / etc.

@paniq:

RAA is costly, and has overshoots i.e. false positives, which means you're frequently backtracking. building intervals on the first derivative should in many cases be cheaper.

Topics

Plaster foam

Kragen Javier Sitaker, 02020-10-16 (updated 02020-11-08)
(8 minutes)

Plaster of Paris is easily formed at room temperature, nontoxic, and somewhat refractory, withstanding temperatures up to 1200°; it decomposes by releasing vitriol at 1460°. But it doesn't insulate against heat extremely well, and it weakens somewhat when heated enough to start dehydrating again, becoming easily crumbled with the fingers. It reputedly strengthens again above 800°, but if that's true, it must be a process that takes more than the few minutes I tried. It has a bad reputation for use in forging iron, because while it will survive up to 1200°, it slowly degrades at iron-forging temperatures. NIST wrote about the various stages of alabaster calcination in the 1940s.

Once re-calcined, the plaster remains solid (except that portion heated to white heat, which evolves a vitriolic air), but is enormously more fragile than before; rubbing it between fingers produces a fine floury powder. It also contracts slightly, producing cracks when heating is uneven. These are more serious problems for larger objects, which more easily collapse under their own weight, than for smaller ones.

Fired clay and portland-cement concrete can be foamed — aerated with bubbles — to improve their insulation capabilities, make them more resistant to crack propagation, and reduce their weight, at the expense of strength. I haven't heard of anyone doing the same thing with plaster of Paris or lime cement.

There's a special-effects material described as "aerated plaster" called Gypsnow: it expands rapidly when wet; remains soft; absorbs impacts. "Place 3.7 liters or 125 oz of water into a 5 gallon plastic bucket. Add 10 lbs of Gypsnow and mix with an electric drill, pour the mixture into a lined plastic form and after about an hour you can remove the item." It claims that this is "aerated" but to me it sounds like maybe it has styrofoam or some hygroscopic polymer in it.

Mixing expanded perlite or, better, expanded vermiculite into the plaster would surely work. But I think they are less refractory than the plaster itself.

WP says vermiculite bonded with vaguely specified adhesives including sodium silicate is good to 1150°. It says perlite is only good to 850°. Hydrated sodium silicate itself will foam up like vermiculite or perlite when you heat it, but the remaining solid material is still sodium silicate, and retains the very low softening point of alkali silicates. Perhaps this is less of a problem in the vermiculite composite, where perhaps the sodium can diffuse away into the vermiculite rather than leaving points of contact very vulnerable to melting.

Abandoned US patent application US20160339606A1 describes trying to reinforce plaster of Paris with graphite and "cenospheres" (making the plaster a syntactic foam; for example, of vermiculite) so it will retain more strength for use in molding high-temperature (310+°) thermoplastics. (The patent application pointed out that once

you've used it in molding you can wash it out with water, since the plaster becomes water-"soluble" again at the casting temperatures.)

The standard way to foam fired clay is to mix it with sawdust, coffee grounds, used yerba mate, or a similar granular material that will burn off in the kiln. This might work with alabaster plaster too. I have made a lump of plaster mixed with yerba; it is quite hard, less dense than plain plaster, and pale green, and it seems to survive fire a bit better than plain plaster; but I haven't yet had the chance to fire such an article long enough to burn out all the organics.

Naphthalene is reportedly used in this way to make porous grinding wheels, boiling out at quite reasonable temperatures instead of needing to be oxidized away like yerba mate. A wide variety of substances would work as alternatives; they need to survive the plaster's setting process in solid form — so water ice, for example, will not work; be capable of breaking apart into granules and remaining as separate granules, ruling out, for example, chewing gum and candle wax; not be soluble in or reactive with water; not react with the plaster itself, which I think might rule out, say, iodine; be easily removable after the plaster is set, for example by heating (as naphthalene) or dissolution with another solvent (as brimstone or rubber in disulfuret of carbon); and, ideally, be very cheap.

Other compounds with these properties include para-dichlorobenzene, camphor, brimstone, rubber, and chlorargyrite, which last requires ammoniated water to dissolve. A more extensive note on some such possibilities, though mostly water-soluble, is in Inorganic Burnout (p. 294). Many common plastics including LDPE are also suitable; though more expensive than the plaster itself, they are reusable. Brimstone has the unique advantage of being very cheap as a petroleum waste product, and its common form melts into a thin liquid at 115°. However, it requires precautions against fire.

Fired-porous-clay kitty litter is another possible porous aggregate, similar to vermiculite but denser.

Plaster muffins

Ttk Ciar suggests mixing a low-boiling-point material into the mix, such as isopropanol, so that it will form bubbles when heating. Maybe a baking powder would work well, as suggested by "Ken" in 2013, who also suggested trying dishwashing detergent. I baked such a loaf of plaster with baking powder; the top of the foam seemed a bit spongy during baking, but the expansion seems to be less than a factor of 2. I turned off the oven after about 20 or 30 minutes of baking; a couple of hours later it seems to be setting reasonably well, resembling a lava rock, so I demolded it and put it in a plastic bag to finish setting overnight.

The cylinder is about 30 mm high and about 80 mm in diameter.

Upon being flamed with a butane torch, the foamed plaster turns first black before turning a brighter white than originally, signaling the presence of some off-white organic compound, maybe bitartrate of soda. It seems to be an open-cell foam; I can blow through it, which is probably why it didn't expand further. Unfortunately it also has an aroma something like formaldehyde, so I question whether putting my lips on it was a good idea.

It was somewhat harder the next day, and another day later a powdery white efflorescence had formed on the surface, containing tiny soft white needlelike crystals. I suspect that the evaporation at the surface of the block carried with it either the unreacted baking powder or its reaction products.

I'd used an empty tuna can as the mold for this plaster muffin in the oven, and I think I'd scratched its internal plastic protective layer, perhaps while mixing the plaster with a chopstick, because there are a couple of thin lines of rust-colored discoloration on the bottom of the muffin. There was also a spot of discoloration on top, which I don't know the source of.

As an interesting note on crack propagation, I was able to snap the muffin in half with my hands after scoring the top and bottom surfaces with fingernails, and then later to drive a metal wire through the whole height of the cylinder. No visible cracks propagated out from the wire.

Another possible way to foam it: hydrate it with water with a lot of dissolved gas, perhaps Priestley's carbonated water, under pressure if necessary. The gas will bubble out and form a foam, but not before the pressure is released.

Topics

Fluidic household pumping

Kragen Javier Sitaker, 02020-10-18 (updated 02020-10-19)
(7 minutes)

Earthships are cooled by the sun. The greenhouse contains a skylight, which when open produces suction from the buoyancy of the sun-heated air. This sucks air through the dwelling spaces from the cooling tubes that run through the giant thermal mass behind the house. When this is not desired, the inhabitants close the skylight and the cooling tubes and open the doors to the greenhouse to let the heat in.

Thus can a tiny air pressure difference produce an enormous heat flow: the viscosity of air is low, so if an opening is wide, even an imperceptibly tiny difference in pressure can produce a mass flow as fast as the wind, which can carry with it an immense power of heat.

To be concrete, let us suppose that we have an opening of two meters square with a subtle breeze of 2 m/s flowing through it. Evidently 8 m^3/s of air flows thus; at 1.225 kg/m^3 this is about 9.8 kg/s, a staggering rate of 35 tonnes per hour. At typical air's isobaric mass heat capacity of 1.012 J/g/K this works out to 9900 watts per kelvin. That is, if the air traveling through this aperture have a temperature a single degree hotter or colder than the space it enters, it brings or displaces 9900 watts. If the temperature difference be ten degrees, the power is 99 kilowatts, and if it be 40°, the power is nearly 400 kilowatts.

Air has some other remarkable advantages as a heat transfer medium, aside from its low viscosity. It's less toxic than any other fluid. It's relatively non-corrosive up to 300° or so, and up to over 2000° for oxides, phosphates, and fluorides. It spans a wide range of temperatures in gaseous form, from -182° (oxygen's boiling point) to some 1600° while preserving its nontoxic and noncorrosive character, and up to an unlimited temperature if this is not important. For the time being, it's easily available, and the cost is low.

At temperatures between 0° and 100° we can direct the flow of air with nearly any everyday solid material, including polyethylene bags, waxed or lacquered paper, styrofoam, plaster, wood, oiled muslin, PET bottles, mica, EVA glue, cardboard, clay, glass, and aluminum foil; outside this temperature range it can become more demanding. Within this temperature range the most diaphanous of materials can contain airflow without injury, needing only enough support from wires or the like to prevent its collapse. Inflatable tubes, with stagnant air in a tube alongside the moving-air tube, may provide the lightest-weight support of all.

The idea inevitably suggests itself to raise a flexible chimney with a hot-air balloon to produce suction from sun-heated air. A solid bootstrapping heating chamber provides initial warm air to initially inflate the ultralightweight balloon, made perhaps of mylar; once it achieves inflation, it can serve as its own air-heating chamber, particularly if the exterior is visibly transparent and painted or mixed with an infrared-blocking paint and/or a low-infrared-emissivity

coating, while an inner mylar partition is black, re-emitting the visible light absorbed as thermal infrared into the interior of the balloon. Alternatively, the balloon can be fed from the chimney itself, which if fastened to the top of the balloon can inject any available hotter air into the balloon, displacing less-hot air out of an open balloon bottom.

The chimney itself can have a construction similar to that of the balloon, and perhaps either or both would benefit from a light springy spiral or two to prevent their collapsing or rippling, resulting in airflow obstruction.

Of course, a more conventional rigid chimney can also be used, with a greenhouse at its base and perhaps transparent panels within it to permit further heating of the air during its ascent. If it can be laid upon the slope of a hill or mountain, it would require minimal material.

Unlike chimneys intended for flue gases, such chimneys need only handle air in the range of 50°–200°, depending on the design, and can thus use non-heat-resistant materials, perhaps even PET.

Even on cloudy days, greenhouses and greenhouse-effect chimneys can achieve significant temperature rises and thus significant pressures.

Such a comparatively high-pressure, low-volume flow can be called upon to fluidically pump larger volumes of air at lower pressures, as in the Dyson bladeless fan. I'm not quite sure how to use this negative gauge pressure to pump air *into* spaces at a zero or positive gauge pressure, particularly without sending the hot air into those areas; some ideas follow.

By accelerating air down a wide tube toward a suction aperture, the air can be given momentum, and the air that misses the suction aperture can then be diverted through other tubes wherein it loses some, but not all, of its speed; this air, now at a positive gauge pressure (?), can then be used to direct other air at still lower speeds.

Another approach, which may sort of be the same thing, is to use the low pressure to suck a flow down a long tube, which near the end has an aperture in its wall to another similar long tube, transferring some of its momentum to the fluid in the other tube, and mixing somewhat.

Bistable fluidic valves of the well-known design using the Coandă effect can be used to change the direction of flows for an arbitrarily long period of time with just a puff of air.

If you're just using the vacuum to drive a closed-circuit thermodynamic system, the problem vanishes; you can maintain the whole system at a negative gauge pressure and bring in nozzles of zero-gauge-pressure air as "compressed air" to blow the flow wherever it's desired.

One difficulty arises if your thermodynamic system is evolving some gases you don't want in your chimney, perhaps because they are toxic or corrosive. In some cases you can bubble them through limewater or something, but in other cases no such convenient outlet is available. In such a case some kind of pumping system like the ones I mentioned above seems essential, whether purely fluidic or merely

pneumatic.

Pumping via pneumatic, rather than fluidic, methods at low pressure simplifies the problem enormously; a flexible balloon of any kind (polyethylene, cloth, origami) within a larger space serves as a gas-tight piston, while a flap of flexible material over an aperture serves as a check-valve. A balloon that inflates to obstruct an air passage provides a simple form of pneumatic switching that easily permits the construction of oscillators and the like.

Topics

Muriate thermal mass

Kragen Javier Sitaker, 02020-10-18 (updated 02020-10-28) (11 minutes)

A common phase-change material used for thermal mass in household climate-control applications is the miraculous salt of Glauber, issuing from his production of muriatic acid. Its melting at 32.38° enables it to store comparatively immense amounts of latent heat to be released upon its resolidification.

Consider, though, muriate of lime, notorious for its deliquescence throwing off heat. This, too, is a process that can be reversed by the application of gentle heat, driving out water from the material — nearly the reverse of the process with the salt of Glauber. Burns are said to have resulted from injudicious ingestion of crystals of this muriate. Yet it is very cheap and much more widely available than the salt of Glauber.

Unlike the situation with the salt of Glauber, the heat-absorbing reaction requires the reversible separation of two components, the muriate and the water. This is a virtue in that it can delay both the absorption of the heat until a higher temperature is reached, by virtue of holding the water vapor in by pressure, and, more importantly, the evolution of the heat at lower temperatures, by preventing moisture from entering the mixture again until the heat is desired. So sealing only, not thermal insulation, is required.

Perhaps this exothermic disssolution can supply an alternative form of "phase-change thermal mass" for household use. Not only can it be used to store heat for heating the house, but also it can provide stored heat to drive a desiccant-refrigeration cycle of the well-known type, if the desiccant can be regenerated at a suitable temperature. One such suitable desiccant is muriate of lime itself, in higher states of hydration than those used to store the heat.

In this way it is possible to store up the heat of the sun during the day to power an air-conditioning system during the night.

It supports monohydrate, dihydrate, tetrahydrate, and hexahydrate solid states, as well as a liquid solution and an anhydrous solid state. The anhydrous state weighs 2.15 g/cc, melts at 772°, and boils at 1935°; the monohydrate weighs 2.24 g/cc and dehydrates at 260°; the dihydrate weighs 1.85 g/cc and dehydrates at 175°; the tetrahydrate weighs 1.83 g/cc and dehydrates at 45.5°; and the hexahydrate weighs 1.71 g/c and dehydrates at 30°. Thus gentle heating can get us to the dihydrate, but fiercer heating would be required to obtain the ferociously hygroscopic anhydrous salt.

The anhydrous salt's standard enthalpy of formation is -795.42 kJ/mol, compared to -1403.98 kJ/mol for the dihydrate and -2608.01 kJ/mol for the hexahydrate. However, presumably much of that is already in the water, whose standard enthalpy of formation is -285.83 kJ/mol; six times that is -1715 kJ/mol, leaving only -97 kJ/mol from adding water to the anhydrous form `(- 2608.01 795.42 (* 6 285.83)) = 97`; from the hexahydrate to the dihydrate we have `(- 2608.01 1403.98 (* 4 285.83)) = 60` kJ/mol. The molar mass of the

anhydrous muriate is 110.98 g/mol, and of the water is 18.015 g/mol, so this is 147.01 g/mol for the dihydrate or 219.07 g/mol for the hexahydrate; in theory, then, we can produce 408 kJ per kg of the dihydrate by hydrating it to the hexahydrate, whose heat capacity is 300.7 J/mol/K; this works out to a temperature rise, theoretically, of 199.5°. (The heat of dissolution of the hexahydrate I do not know.) This 408 kJ/kg compares very favorably to 251.2 kJ/kg for the salt of Glauber.

(The water-solubility of the muriate does increase with temperature, so it is also possible to absorb heat in this fashion, as with the salt of Glauber. I am not entirely clear on how the humidity of air and temperature interact with the degree of hydration of the salt.)

Short-circuiting the process a bit, you could cool the house at night with the sun's daytime heat by dehydrating the muriate with sun, then let it cool, eliminating its sensible heat. Then, when cooling is desired, we can first dry some air by passing it over the muriate, then cool the air back to the outdoor temperature by passing it through a recuperator or regenerator, then cool the air further by evaporating water into it.

To cycle the desiccant between liquid and solid states, it would be useful for the solid state to be granular; otherwise it will tend to agglomerate into a solid mass impermeable to air. Spray drying is a potentially useful way to do this; to avoid getting grains that are too small, perhaps an updraft can be used to select only drops in a certain size range, before passing the remainder into a continuous updraft that keeps them at a stable height until they are dry.

Alternatively, the liquid could be held in very carefully leveled, very shallow pans, drying into a thin crust in the bottoms of these pans when 50° air is passed over them. This way the solid salt will have a large surface area without being granulated. The pans can be made of thin plastic, for example PET, polystyrene, or PMMA, and surrounded with high walls on all sides, with the airflow coming in from and departing from above, so that an error in tilt will not spill the solution.

Giglio's thesis

I was pleased to discover Evaluation of heat available from calcium chloride desiccant hydration reaction for domestic heating in San Francisco, CA, Michael Giglio's 2017 mechanical engineering master's thesis at Santa Clara University. He calls out as advantages of this "thermochemical energy storage" approach its ability to function without insulation and the higher storage capacity available for dehumidification and cooling, and reports a "best-case scenario" of 19 kWh/m³ [68 MJ/m³] of storage capacity for heating, using "a dilution reaction between 100% concentrated [muriate of lime] and water to a 20% solution". While there are surely conveniences to using a purely liquid system, I think higher densities can be achieved using solids. Giglio's final design used a closed system rather than exchanging moisture with ambient air.

It's unfortunately very poorly written, with a great deal of repetition, occasional contradictions, and much ambiguity (What would a "100% concentrated" solution be? The anhydrous muriate?),

but I managed to slog through it.

He reports (p. 7, 17/71):

A solar driven liquid desiccant cooling system has been developed in Singapore by L-DCS and shown to provide a storage capacity of 183 kWh/m³, while a similar system has been developed by ZAE Bayern that utilizes a lithium chloride solution and district heating to provide 12 kW of cooling and a storage density of 150 kWh/m³ [11].

These densities work out to 660 kJ/ℓ and 540 kJ/ℓ respectively. His ref. 11 is "Hublitz, A., 2008, 'Efficient Energy Storage in Liquid Desiccant Cooling Systems,' Dissertation, Technical University of Munich."

He also reports that muriate of lime costs US$0.10–$1.00/kg, but I've found it at retail in Argentina to cost more like US$1.60–$4/kg. He also reports (p. 44, 54/71) that he bought it for US$100/kg from Sigma-Aldrich.

He reports that the enthalpy of dissolution of the compound should be "-740kJ/kg of desiccant", but I don't know whether that's dissolution of the anhydrous salt or what.

On p. 27 (37/71) he reports that he got a 63° temperature rise by adding water to muriate of lime; a bit later he explains that this was solid, but it's unclear whether it was the anhydrous form or one of the solid hydrates. On p. 37 (47/71) he claims to have gotten 288 kJ per kg of muriate of lime.

Giglio concludes that his prototype system would cost too much to be economical if scaled up, estimating its cost at US$13000 with 8 square meters of solar collector costing US$8000, 285 kg of muriate of lime costing US$2850, and two copper-tubing heat exchangers costing US$775 each, and consequently providing a levelized cost of energy of US$0.86/kWh with a 30-year lifespan, competing with US$0.04/kWh for natural gas, 21½ times too expensive; by his calculations, US$600 would be the price point at which such a system would be cost-competitive.

I think US$500 is a more reasonable cost estimate for this quantity of desiccant, and using copper is a terrible idea — not only is it expensive, but it's vulnerable to attack from the chlorine, and it cannot be made thin. Polyethylene, polypropylene, or polytetrafluoroethylene would be much better materials. Most of the necessary heat exchange can probably be provided by bubbling air through a liquid desiccant or blowing it over a solid one. And thermal solar collectors using air can be as simple as thin transparent glass or plastic, which costs about US$20/m², not US$1000/m², over a styrofoam box painted black on the inside. (See Solar Netting (p. 330) for how to prevent cheap glass from being broken by hailstones, using chicken wire.)

As noted in Desiccant Climate Control (p. 489), there seem to be wholesale vendors that will sell you a tonne of muriate of lime for apparently US$272. This would bring the cost of the desiccant in Giglio's system down from the US$28'500 he was paying Sigma, or the US$2850 he estimates, to US$78. This 36× reduction has a significant impact on the economic plausibility of such systems! Also, it's likely that the wholesale vendor is selling the anhydrous salt, since that's the most useful form for use as a desiccant, while what Giglio

bought from Sigma is likely the hexahydrate, since that's the form most stable when exposed to air at room temperature.

(The heat transfer rate of a bubbler is limited by the size of the bubbles, but a saturated solution of muriate of lime gets quite frothy, from which I conclude that either the muriate itself reduces the surface tension substantially, or my sample is contaminated with a surfactant. Either way, if a heat and/or vapor exchange is effected through direct contact, as in a droplet-column or bubbler exchanger, cyclonic separators or something might be necessary to filter the desiccant out of the air.)

Moreover, as mentioned above, this thermal collector and desiccant can serve not only to provide house heating, but also house cooling and, when necessary, dehumidification. This means it's competing against not just natural gas but also the electricity to run an air conditioner and, perhaps, a refrigerator. Desiccant Climate Control (p. 489) goes into more detail here. It also notes some other desiccants that are more than an order of magnitude cheaper than muriate of lime.

Topics

Calcium strengthening

Kragen Javier Sitaker, 02020-10-21 (updated 02020-10-24)
(23 minutes)

Plaster of Paris and lime mortar are widespread, cheap, rigid materials that can be easily shaped, especially before they harden. But they don't harden that much. What if we could harden them further, once they were shaped, or otherwise strengthen them?

In all of what follows, it's worth keeping in mind that the material produced can be formed either as a solid or, sometimes more easily, as a foam. Solid foams can permit faster permeation of liquid reagents, permitting the reaction throughout the whole material of a reagent that would otherwise only be able to reach the surface, and can also have superior mechanical properties in some applications.

Coating

The simplest way to alter such objects' material properties is to coat them with a liquid or gel which then adheres to them. For example, portland-cement mortar, calcium-aluminate mortar, magnesium-phosphate mortar, shellac, wax, sodium silicate, paint, or glued paper. Often a shell of a hard, dense, or tensilely-reinforced material around a lightweight, rigid core provides better tradeoffs of strength and weight, and sometimes even better absolute resilience to impacts, than either material could alone.

The material in my note on globoflexia (p. 374) goes into considerably more detail about this family of construction, as well as the note in Dercuano about sandwich panels.

Infiltration

Plaster is fairly porous, lime a bit less so unless you find some way to foam it. You can make plaster even more porous by the expedient of making it with more water. These porous substances can have their pores infiltrated with other substances, such as polymerizing resins; vacuum or high pressure may be a useful way to make this infiltration more complete.

Other useful infiltration materials might include thermoplastics, paraffin wax, molten sulfur, and reagents for replacement reactions.

Replacement reactions

As I mentioned in Dercuano, there are many anions that precipitate as water-insoluble solids (chart) when confronted with polyvalent cations such as calcium, magnesium, iron, copper, zinc, titanium, manganese, boron, or aluminum. Anions with such behavior include phosphate, carbonate, alginate, silicate, sulfide, and also usually hydroxide and occasionally oxalate or sulfate (with barium or calcium).

In the note there on "Likely-feasible non-flux-deposition powder-bed 3-D printing processes", I suggested using binder jetting to exploit these reactions to instantly cement powders. But a possibly

even more interesting possibility is using such reactions to change the properties of an already-existing object made from, for example, plaster of Paris or lime. For example, tadelakt is made by precipitating calcium stearate (etc.) in a lime surface from soluble stearates (etc.) such as that of sodium or potassium, then burnishing the surface; thus lime is waterproofed.

It might be reasonable to use other semi-soluble or semi-solid sources of polyvalent cations as well, such as gelatinous aluminum hydroxide, or iron metal in the case of phosphate conversion coating.

Phosphates

There are lots of phosphates, many water-insoluble, hard, and refractory, even without getting into pyrophosphates, metaphosphates, and polyphosphates. A wide variety of phosphate minerals exist in nature.

I've gone into some details on possible combinations in the note mentioned above. Perhaps you could, for example, apply a solution of diammonium phosphate, monoammonium phosphate, or trisodium phosphate to an object made of plaster of Paris, calcium hydroxide, or calcium carbonate, and thus get a harder object made partly or wholly of calcium phosphate. The ammonium-carbonate combinations seem particularly appealing, since it can be thermally decomposed to fairly harmless materials. Phosphoric acid probably would not work on plaster of Paris, and although it reacts with calcite (same source), I'm not sure it *strengthens* the material in the process.

One thing that does *not* work is ordinary hardware-store aqueous phosphoric acid at room temperature and pressure; after a couple of days no change was observable, and the plaster crumbled just as easily as before. This is in retrospect completely obvious: commercial phosphoric acid is prepared by precisely this reaction, in reverse. I've now gradually added enough baking soda to mostly convert the phosphoric acid into some kind of gel; we'll see if the phosphate of soda presumably now dominating the scene is any more effective at phosphating the calcium.

Gelatinous aluminum hydroxide is another appealing target for this kind of phosphate replacement reaction, since it is so easily molded; perhaps it would yield extremely insoluble and refractory aluminum phosphate (whether berlinite or in some other form, most likely an amorphous one), along with either caustic soda or easily-boiled-off aqueous ammonia. Maybe some such reaction would be useful for preventing neutral-electrolyte aluminum-air batteries (p. 326) from getting clogged up with slime.

Magnesium phosphates

Although as a mineral its Mohs hardness is 6, magnesia or periclase is a cheap "crushable ceramic" commonly used for electrical insulation of heating elements due to its high thermal conductivity, and experiments have been done reacting it in this way with monoammonium phosphate; they explain:

> Phosphate cements possess mechanical and chemical properties that are superior to those of ordinary hydraulic (Portland) cements, … The reaction between a reactive form of magnesia and acid ammonium phosphate is very rapid and exothermic, and the materials cannot be practically used as such. Thus, the use of calcined or

deadburned magnesia is suggested.

This is music to my ears, since of course instantly setting cements are precisely what I most want for 3-D printing. Also, they mention that struvite, the very soft ammonium magnesium phosphate that I feared would be formed from such reactions, does in fact form, but *decomposes to monomagnesium hydrogen phosphate at 55°* by losing its water and ammonia!

To retard the setting and permit molding, and in particular to avoid increases in temperature that would be fatal to their waste-immobilization purpose, they include boric acid as a setting retardant. They also included sodium tripolyphosphate, to increase strength and reduce porosity, sand, and grinding dust (probably mostly aluminum oxide and steel, with significant amounts of fiberglass); the monoammonium phosphate:water:magnesia relationship seems to have been 3:2:4, probably by weight.

They report final compressive strengths in the 20–40 range (MPa, I assume), and tensile strengths in the 1–2.5 MPa range.

Another 2011 paper explains:

Magnesium phosphate cements (MPCs) have been extensively used as fast setting repair cements in civil engineering. They have properties that are also relevant to biomedical applications, such as fast setting, early strength acquisition and adhesive properties. However, there are some aspects that should be improved before they can be used in the human body, namely their highly exothermic setting reaction and the release of potentially harmful ammonia or ammonium ions...

They also used borate (as sodium borate) as a retardant, and also used larger grains of phosphate salt. They reported that monosodium phosphate rather than phosphates of ammonium gave an amorphous result instead.

Very interestingly, this also mentions "apatitic calcium phosphate cements", which have been investigated by the same authors and others as possible bone cements.

Yet another paper, this one in 2015, reports on 3-D printing:

Strontium ions (Sr^{2+}) are known to prevent osteoporosis and also encourage bone formation. Such twin requirements have motivated researchers to develop Sr-substituted biomaterials for orthopaedic applications. …developing Sr-substituted $Mg_3(PO_4)_2$-based biodegradable scaffolds. … powder printing, followed by high temperature sintering and/or chemical conversion…. strength properties of 36.7 MPa (compression), 24.2 MPa (bending) and 10.7 MPa (tension) were measured.

They were using powdered trimagnesium diphosphate with strontium replacing varying amounts of magnesium (up to ⅓), sintering it, crushing it, 3-D printing it with a bit of hydroxypropylmethylcellulose, depowdering it, sintering it again, and then soaking it in diammonium phosphate to post-harden it.

See also below about zinc phosphate dental cement.

Phosphates of zinc, manganese, and iron

Phosphate conversion coating coats steel with water-insoluble phosphates of these three metals by taking advantage of their solubility in acid, such as (of course) phosphoric acid.

Ferric phosphate is what protects the Iron Pillar of Delhi, and also some of my girlfriend's kitchen pans, from rusting, despite its porosity. It can be achieved using nothing more than phosphoric acid.

Wikipedia leads me to believe that it should be orange to brown, but mixing hardware-store phosphoric acid "converter" with powdered orange rust gives a black insoluble compound instead, and so too is the coating on the pans produced by boiling Coca-Cola in them.

Zinc phosphate is sometimes deposited on steel in the same way; the steel reduces the hydrogen ions at its surface, precipitating zinc phosphate out of solution. It's also used together with magnesium phosphate as a dental cement; zinc oxide and magnesia are mixed with phosphoric acid on a glass plate to allow them to cool, giving a pot life of a few minutes. Wikipedia explains:

> Zinc phosphate dental cement is one of the oldest and widely used dental cements. It is commonly used for luting permanent metal and zirconium dioxide restorations and as a base for dental restorations. Zinc phosphate cement is used for cementation of inlays, crowns, bridges, and orthodontic appliances and occasionally as a temporary restoration.
>
> It is prepared by mixing zinc oxide and magnesium oxide powders with a liquid consisting principally of phosphoric acid, water, and buffers. It is the standard cement to measure against. It has the longest track record of use in dentistry. It is still commonly used; however, resin-modified glass ionomer cements are more convenient and stronger when used in a dental setting.

Manganous phosphate is used similarly for metal protective coatings. Natural paragenetic combinations with iron phosphate include triplite (Mohs 5–5.5), triploidite (Mohs 4.5–5), and purpurite (Mohs 4–5, without iron). Another score of other minerals include manganese and phosphate.

All three of these relatively hard phosphates, or families of phosphates, can reasonably be formed by reacting phosphoric acid with the respective oxides, which are easy to prepare and acquire, and relatively inert (except, of course, for the heptoxide of manganese.) I suspect that other soluble phosphate salts would also work as phosphate donors. Most of the oxides are soft materials that are easy to shape and even cast, though solid pyrolusite (dioxide of manganese) is 6–6.5. In the form of fine powders with a little binder, the materials might be more easily shaped before being bonded with a phosphate donor.

Other possible cation-donating solids include the hydroxides (more or less equivalent to the oxides, if we're talking about aqueous reactions) and the chloride of zinc.

Phosphate of boron

Boron phosphate is a somewhat refractory material, subliming above 1400°, and water-insoluble in its crystalline form. However, both the reaction and the crystallization seem to be fairly slow at room temperature.

Phosphates of zirconium

There is a very interesting monozirconium diphosphate but I suspect that zirconia will not yield it easily. You could surely deposit zirconium nitrate on an inert surface, wash it with aqueous lye to produce mostly insoluble zirconium hydroxide, and react that with phosphoric acid; there might be easier routes.

Etc.

Copper? Titanium?

Carbonates

Perhaps you can convert plaster of Paris to the harder lime, once shaped, by one of the following approaches:

- Soak it for a long time in sodium bicarbonate.
- Or sodium carbonate.
- Hell, or wash it heavily in aqueous ammonia. Or lye.
- Or soak in ammonium carbonate, then heat it up above 58° to drive off the ammonia.
- Or wash it heavily with aqueous ammonium carbonate.
- Or blast it with hot CO_2, which seems like maybe the most likely approach.

I'm not confident that any of these will work. Hot CO_2 works to convert the sulfide of calcium into calcium carbonate, releasing sulphuretted hydrogen, but you cannot convert plaster of Paris into the sulfide as far as I know. Heating the plaster past 1400° in air will outgas vitriol, leaving behind lime, which is so much smaller that it tends to fall apart. Perhaps heating it to a somewhat lower temperature in a CO_2 atmosphere, particularly under high pressure, would work better; and perhaps it would help if something else were removing vitriol from the gas chamber.

A process that would more likely work: carbothermically reduce the sulfate to the sulfide, perhaps with carbon plasma, carbon monoxide, or ethylene, rather than solid carbon, and then blasting the sulfide with hot carbonic acid gas to liberate sulphuretted hydrogen and produce the carbonate.

This is not the kind of tranquil process of painting on some kind of conversion liquid that I was hoping for.

Lots of *other* polyvalent cation donor materials can productively form insoluble carbonates, though. Barium, copper, iron, lead, manganese, nickel, and zinc, for example.

Alginates

I haven't seen a whole lot about alginates except for the usual dental-mold and spherification stuff, using soluble sodium alginate and insoluble calcium alginate. I imagine that most candidate polyvalent cations would work to coagulate the stuff. In particular, though, washing lime or plaster of Paris with a solution of sodium alginate ought to give you a waterproof surface, similar to tadelakt.

Silicates

Presumably washing the surface of lime or plaster of Paris with soluble silicates such as those of sodium or potassium would strengthen and waterproof the surface, and perhaps also improve its refractory properties. By applying these solutions to an open-cell foam, perhaps the change could be usefully obtained throughout the material.

As with phosphates, the possibilities of aluminum anions here are tantalizing: can you mold something out of gelatinous aluminum hydroxide, then harden it with sodium silicate? But the silicates of aluminum are enormously varied, ranging from kaolin and zeolites to mullite.

One form of natural magnesium silicate, with a 3:4 Mg:Si ratio, is talc, itself very easily carved, even with thumbnails, before being fired to hardness. Synthetic magnesium silicate, for example as a food additive or a plastic filler, is routinely precipitated in amorphous form by mixing sodium silicate with the nitrate, chloride, or sulfate of magnesium.

Another form of natural magnesium silicate, with a 2:1 Mg:Si ratio, is forsterite olivine, including the gemstone peridot. Olivine is a spectrum between forsterite and the silicates of iron (fayalite) and manganese (tephroite). Forsterite is just as hard as quartz and considerably more refractory; forsterite melts at 1890°, fayalite at merely 1205°, and tephroite at only 1345°.

So you could imagine that a sufficiently small amount of silicate added to a concentrated source of somewhat soluble magnesium, such as magnesia, would produce forsterite, or an amorphous polymorph thereof. The carbonate of magnesium (magnesite, Mohs 3.5–4.5) is some four times as soluble as the oxide, and the fluoride (sellaite, Mohs 5–6) a little less soluble than the oxide.

Borates

Boric acid can form insoluble, hard, and sometimes refractory borates of many polyvalent cations, as well as the water-soluble borax; worth mentioning are chambersite: Mohs 7 (manganese); boracite: Mohs 7–7.5 (magnesium); suanite: Mohs 5.5 (also magnesium); and hilgardite: Mohs 5 (calcium and chloride). Other borate minerals are known.

Really though I suspect that the most promising thing to do with borates is to burn them into boria or to somehow convert them into boron nitride. Ammonium borate seems like the ticket:

- 10.9% soluble by weight at room temperature
- stable to about 230°F (110°C), at which point it loses all but two moles of water. If heated sufficiently, it releases the balance of its hydration water and decomposes to boric oxide and ammonia.

(Though boric acid decomposes to boria at 300°.)

Sulfides

Generally the sulfides have the problem that they slowly decompose to produce sulphuretted hydrogen and sulfuric acid, given access to moist air. Carbothermic reduction of plaster of Paris produces calcium sulfide.

Fluorosilicates

Toxic ammonium fluorosilicate is reasonably water-soluble, as are the fluorosilicates of copper, ferrous iron, lead, lithium, manganese, and magnesium, but the fluorosilicates of barium and calcium are much less so.

Oxalates

Oxalates of soda, potassa, and ammonium are fairly water-soluble, while oxalates of magnesium, silver, scandium, iron, and barium are practically insoluble, and the oxalates of lime, copper, and zinc are almost totally insoluble. WP says the oxalate of lime starts to

decompose at 200°, though, so it's not very heat-stable — but what it decomposes *to* is, I think, the carbonate. It looks like that's right, but the temperature is around 500°, not 200°. The magnesium oxalate, similarly, decomposes to the carbonate between 420° and 620°.

To take a particular example, the oxalate of potassa (LD_{50} 660 mg/kg orl-rat) dissolves 36.4 g/100 mℓ water at 20°, the sulfate of potassa 11.1 g, the oxalate of lime 670 µg, and the sulfate of lime 255 mg. This suggests that a solution of 10% potassium oxalate will eventually convert plaster of Paris into >99% insoluble oxalate of lime, which can then be gently heated to get limestone.

Fluorides

Bernd Jendrissek very graciously pointed out that a fluoride replacement reaction is commonly used to harden teeth and make them more acid-resistant by converting hydroxyapatite to fluoroapatite. The fluoride of calcium, fluorspar, is both harder and less water-soluble than either its sulfate or its carbonate, and so a double metathesis with a soluble fluoride salt such as sodium fluoride might plausibly work to harden plaster bodies. These salts are somewhat poisonous; NaF's LD_{50} is about 100 mg/kg, so it's used as rat poison, but also in toothpaste and to treat osteoporosis.

Sodium monofluorophosphate, as used in some toothpastes, might be another alternative, doing the phosphate conversion and fluorination in a single step; its LD_{50} is [about 500 mg/kg][40].

[40: https://en.wikipedia.org/wiki/Sodium_monofluorophosphate

Inert fillers and high-temperature activation

There's a fourth totally different approach to strengthening these quasi-refractory calcium compounds, one that doesn't involve room-temperature gas-phase or aqueous reactions.

Both plaster of Paris and ordinary lime cement remain solid up to high temperatures — plaster of Paris decomposes to quicklime above 1400°, while fully carbonatated lime decomposes to quicklime starting much colder, above 825°, but then quicklime itself remains solid to 2613°. However, it may be in smaller pieces than the original shape, if the changes in volume were enough to crack the shape.

One possible approach to the problem is to incorporate inert needlelike material into the original plaster to bridge the gaps; mullite can be bought in crystals for this purpose for making pottery, or as polycrystalline fibers for foundry linings. At lower temperatures, wire of steel, copper, or stainless steel can work. Plant fibers, such as sisal, sawdust, or used yerba mate, char between 200° and 300°; but the charcoal can survive and continue to add significant strength up to much higher temperatures than the rest of the components, unless oxygen burns it out first.

Even ordinary quartz sand may help. Suppose your plaster mix is 90% quartz sand and 10% plaster of Paris binder, and when you heat the plaster enough to dehydrate it, the plaster shrinks by 0.2% linearly (0.6% in volume). But, since each linear dimension is only about 1.2%

plaster binder across most of the perpendicular cross-sectional area, the linear shrinkage is only 0.0024% instead of 0.2%. This can make the difference between heat cracking the material and not.

(Quartz is not the ideal material because it dunts at 573°, expanding 0.45%. Many other tempers are used in pottery to improve this situation, crushed-brick grog being one of the most common.)

However, we can go further and actually use fillers that will actually *react with* the calcium compounds at high temperature. I saw this on a sciencemadness thread, but I don't know who to credit: for example, you can incorporate an "inert" filler such as rutile, which at 1300° and above will stop being inert and combine with the quicklime to form calcium titanate (melting point 1975°). Even silica, particularly in an amorphous form such as infusorial earth, or as soluble silicates, might work for this; larnite melts at 2130°. And clays could provide both alumina and silica.

(Titanate also forms mineral salts with manganese, magnesium, barium, lead, zinc, iron, and half a dozen other metals. These are basically all insoluble, even the metatitanate of soda, but supposedly there's a water-soluble triethanolamine titanate.)

Topics

Minimal cost computer

Kragen Javier Sitaker, 02020-10-23 (updated 02020-12-01)
(12 minutes)

Dave Jones, author of EEVBlog, has been playing with these Padauk one-time-programmable PMS150 and PMS150C microcontrollers which cost 2.7¢ (1 yuan I guess?), and his fans on the EEVBlog Forums have put together the Free PDK and easy-pdk-programmer-hardware projects to make the Padauk chips more usable (so you can have, you know, `for` loops and function arguments).

So consider a personal computer consisting of a 5¢ USB micro-B jack, three 8¢ CPUs, three 5¢ bypass capacitors, a 60¢ 8-kibibyte 20MHz SPI SRAM module, a 15¢ 2-megabyte NOR Flash chip, a 30¢ piezoelectric buzzer (here in Argentina, US$2 as part of a musical Christmas card, but a 10-pack from Anri TV costs US$4), and a resistive touch input surface made of paper and powdered graphite (rather than 3M Velostat) using Chris Harrison's "Electrick" electrical field tomography technique. The total BOM cost is US$1.49, plausibly reducible to under US$1, and it is probably capable of executing about 24 million 8-bit Padauk instructions per second, or 2.4 million interpreted instructions per second, superior to an original IBM PC. It might also be feasible to bitbang SPI to a MicroSD card, to bitbang PS/2 keyboard input, and to bitbang video output signals for NTSC, PAL, or VGA.

Overview of the Padauk microcontrollers

In EEVBlog #1306 Jones starts by buying a STM32F072C8T6 on Digi-Key for, I guess, the Easy PDK Programmer Hardware, then demonstrates how to instead upload the BOM CSV file from easy-pdk-programmer-hardware to LCSC's website. Then he uploads the Gerbers to JLCPCB and PCBWay for fabrication as a demo. I haven't watched the rest of the series yet, because there's only so much Dave Jones I can deal with in a day.

Digi-Key doesn't carry the PMS150 or other Padauk micros; you have to buy them via LCSC or some other Chinese distributor. EEVBlog forum user spth reports that they're available for less than 1¢ on Taobao.

Jay Carlson wrote a review of the PMS150 family of microcontrollers. He explains that the family runs from 512–4096 words of program memory and 64–256 bytes of RAM, all running at up to 16 MHz (normally 8 MIPS). The PMS150C is 3.18¢ and has 64 bytes of RAM, 1024 words of ROM, 6 I/O lines (including an 8-bit PWM line), and runs on 2–5 (?) volts at 450 µA/MHz. The cheapest one with 256 bytes of RAM is the 8.64¢ PMS133, which also includes a 14-bit ADC, 18 I/O lines, 3072 words of ROM, 2 8-bit PWM lines, 3 11-bit PWM lines, running on 2.2–5 volts at 750 µA/MHz.

He says they can run on a high-speed 16MHz internal oscillator ("IHRC"), as well as a low-speed internal oscillator of tens of kHz ("ILRC") for lower-power operation.

Carlson is mightily impressed that some of these processors have hardware multithreading, context switching on every instruction, though with only two threads. He tried this out with a WS2812B adaptor, for which he had to bump the clock speed up to 18 MHz.

Power efficiency: 1 µW sleep, 1800 pJ/insn

He also measured that the PMS150C only used 350 nA on 3.3 V in sleep mode, concluding, "A CR2032 battery could power this thing in sleep mode for 10–15 years — the limiting factor would be the self-discharge of the battery itself."

By comparison, my notes in Dercuano say the datasheet says a STM32L in "stop" mode uses 540 nA with the RTC running and 290 nA with the RTC stopped, which I calculated at 134 years of a CR2032, which of course won't physically last that long, as Carlson said. This is comparable, but its low-power modes sound like more of a pain to wake up from. I estimated that the STM32L0 requires 210–400 pJ per 32-bit instruction; the PMS150C's 450 µA/MHz at two clocks per 8-bit instruction at 2.0 volts would be 1800 pJ per 8-bit instruction, about an order of magnitude less efficient. However, the STM32L0 costs US$1.50 rather than US$0.03.

Memory

If you want the Padauk chip to be a general-purpose computer rather than translating some voltage levels or performing a biquad filter on PCM data as Carlson did for his review, you're going to need some external memory, at least 4 KiB or so. Adesto (the new brand for Atmel's SPI Flash chips and so on) has published a bizarrely titled whitepaper about "AI memory", in which they claim their new EcoXiP line of nonvolatile memory offers a better tradeoff for SPI execute-in-place uses, because large SRAM and especially PSRAM chips use a lot of power, while large SPI nonvolatile memory chips have historically been intended for booting and are consequently kind of slow.

The upshot seems to be that, if the firmware on the Padauk chips themselves is running some kind of interpreter on instructions it loads over SPI, you might be able to run them faster if you use either EcoXiP or SRAM (or PSRAM), than if you try to get by with *only* nonvolatile storage. This is the reason for including the 60¢ Microchip 23K640T SRAM chip in the design sketch at the top of this document. Microchip's datasheet claims it requires 4 µA of standby current, which is 10× more than the microcontroller, and 3 mA of read current at 1 MHz, which presumably scales to 60 mA at 20 MHz, as much as all three microcontrollers put together. (Also that particular chip won't run at 5 V, but presumably there are comparable or better chips that will.)

Prospects for self-hosting a development environment

EEVBlog's fans ported SDCC to the hardware, under the architecture names "pdk14" and "pdk15", so new these devices have a free-software toolchain. SDCC is a pretty decent C compiler, mostly ANSI C99, used among other things to write CP/Mish. Could it be

made to run on the Padauk platform?

SDCC is in all about 10 megs of source code, compressed, and only "officially" supports the 386 and amd64 platforms these days, on MacOS, Linux, and Microsoft Windows, including Cygwin. It's built using autoconf and maybe automake, though, and the ChangeLog is full of mentions of platforms like FreeBSD, OpenBSD, NetBSD, sparc64, SPARC, ARM, PowerPC, ppc64, and even the Alpha (though that was in 2006). Support for compiling it with Borland C was removed in 2009.

The stripped executable is about 2.8 megs on i386, roughly the internal RAM of 10949 PMS133s, which would cost US$946 (BOM cost). So clearly it wouldn't be a straightforward port; you'd need to use some kind of external memory, which probably means using some kind of a virtual machine.

On this Atom netbook SDCC 3.5.0 takes 40–52 ms of user time to compile hello.c for Z80 (this old SDCC doesn't support Padauk) as follows:

```
$ cat hello.c
#include <stdio.h>

int main() { printf("hello, world\n"); return 0; }
$ time sdcc -mz80 -c hello.c

real    0m0.066s
user    0m0.040s
sys     0m0.016s
```

Valgrind claims that this takes 36'131'256 instructions, so you probably need at least to run a few million 32-bit-equivalent instructions per second to make SDCC usably fast.

Fred Brooks made a famous declaration (in *The Mythical Man-Month*?) that the System 360 linker around 1968 was probably the most advanced overlaying linker that would ever be written, since virtual memory made overlays obsolete. But SDCC's targets mostly don't have virtual memory, though many do support banking, so SDCC supports overlays today.

The most immediate problem with compiling SDCC with itself is that it does `#include <memory.h>` but does not provide such a header file for its targets. This is not in itself a terrible problem (SDCC does provide `<string.h>`) but it does suggest that nothing like this has ever been tried.

Another problem is that, even if you can get the Padauk chips to emulate one of its supported platforms, it doesn't support generating code for any of the targets mentioned above that the ChangeLog suggests it's been ported to, or indeed for any target that GCC supports. (It used to have AVR support, but that has been removed.)

So to get a self-hosting compiler on the Padauk chips, SDCC doesn't seem like a particularly promising starting point.

Building an in-circuit emulator for the chips (important because they're mostly one-time programmable and don't have a lot of extra pins to devote to debugging) seems like it would be pretty difficult to

do with the chips themselves. If you were willing to accept an order-of-magnitude slowdown it might be reasonable.

Building a programmer board using more of these microcontrollers (rather than, say, an STM32) might be feasible.

Portable power

What if you didn't have to plug the fucking thing in all the time?

In my note on aluminum-air batteries (p. 326) I noted that ghetto aluminum-air batteries reportedly have an energy density of 7 or 8 MJ/kg, which is to say, 7 or 8 joules per milligram (!). At 1800 pJ per instruction, each milligram of aluminum buys you 3–4 billion instructions, a few hours of computing at Commodore-64 speeds, or maybe an hour at IBM PC speeds. (On the other hand, if you're relying on a piezo tweeter for output, you may burn through your energy noticeably faster.) Using my estimate from the Dercuano note on keyboard-powered computers, that basic word processing functionality needs about 30 μJ per keystroke (mostly to update an e-paper screen not considered in this design), each milligram of aluminum buys you around 33000 keystrokes, about a chapter's worth of writing.

If we're running all three microcontrollers at their full 16-MHz speed, we're using about 22 mA or, say, 60 mW. This is 0.6 cm² of full sunlight, or 6 cm² if you're using those 10%-efficient amorphous solar cells from solar calculators, or 48 m² if you're using cuprous oxide solar cells made in your kitchen from expensive household materials.

If you had a capacitor or battery you were efficiently charging and discharging, then one second of charging at those 60 mW (60 mJ) would pay for 30 million instructions or 2000 keystrokes of word processing.

One gram of aluminum would be enough to run the computer for 3–4 trillion instructions: almost 40 hours at full speed, or almost 4000 hours, 5 months, at Commodore-64 speeds.

(None of these figures include power consumption for electric field tomography, VGA signal generation, etc., or more problematically, the RAM mentioned above.)

In LED Computation (p. 480) I link to an article by James Bryant of Analog Devices saying that a 5-mm red LED can generate 20 μA in photovoltaic mode in full sunlight. If we assume that this is at 1.6 V, roughly the forward voltage of the LED, and we can step it up to a voltage the CPU can run on — or use a few smaller LEDs in series — then it's 32 μW. At 1800 pJ per instruction, that's 18000 instructions per second, not very powerful but still faster than a typical calculator.

Topics

Abbe-limited DRO

Kragen Javier Sitaker, 02020-10-24 (updated 02020-12-31)
(11 minutes)

How precisely can you measure position? How precise can a servo be? What if it's bodged together from mass-market junk?

Fused-quartz glass scales

It's practical to illuminate a fused-quartz optical scale with 365-nm-wavelength LEDs. A modern CMOS imaging sensor can fairly easily reach the Abbe limit; if we take our numerical aperture as 1.4, the Abbe limit is $\lambda/2.8$; at 365 nm that's 130 nm. We can reasonably expect to estimate the centers of blobs in the image to within about a quarter of the resolvable feature size, so we should be able to estimate our absolute position along the scale to within about 35 nm by this method.

This is only about an order of magnitude worse than the recently best available optical position sensors, such as Renishaw's 5-nm-resolution "RELM" scales that limited the precision of Awtar and Parmar's XY HiPER NAP from 2008, and it should be achievable with cheap off-the-shelf parts. The EPSRC Center for Innovative Manufacturing in Ultra Precision at Cranfield University routinely achieves better precision than this, 10 nm in many cases and down to 1 nm in some cases, but I don't know how.

Temperature compensation

Fused quartz's linear thermal coefficient of expansion is about 0.5 ppm/K, so a meter-long fused-quartz scale will lengthen or shorten by 35 nm every time the temperature changes by about 70 millikelvins. I don't think infrared thermometers can measure temperature with this precision, so *compensating for* temperature variations would probably require the use of some kind of contact thermometer; I've argued in my note on Thermistors (p. 431) that this level of temperature measurement precision is fairly easily achievable without super-precision circuitry. Note that precise temperature *calibration* is not necessary, but the absence of *drift* is.

But contact thermometers are slow, at least the macroscopic ones I was considering in that note, so it seems likely that a more viable approach is to *prevent* temperature variations of more than 70 mK or so. To do this, you would use PID thermostatic control of a stream of coolant, such as water, air, or hydrogen, continuously pumped past the fused-quartz scale to prevent thermal variation; and you can calibrate the impulse response of the scale's thermal expansion to the error signal from the PID control system, so you can to some extent compensate for smaller variations.

500-millikelvin-precision temperature control has been a sine qua non for precision manufacturing since Michelson's first grating engines.

A different and possibly more difficult problem is that, if you're measuring motion along such a scale, you may actually be interested

in the relative positions of things attached to the scale and the sensor. And those things will almost certainly have a much larger TCE than fused quartz, as well as being subject to some side loading that causes them to deform elastically. It may be feasible to use the above-described techniques to take measurements from different points of them to quantify these deformations.

A cellphone camera as sensor

Consider the Samsung Galaxy Note 20 rear camera, which is 108 megapixels (presumably 9000×12000) with, if I understand correctly, 0.8 µm pixel pitch; it is capable of delivering 720p video at 960 frames per second. (0.8 µm would give us a sensor physical size of 7.2 mm × 9.6 mm, which is quite reasonable.) For the pixels of such a sensor to correspond to 130-nm regions of the surface being focused on, the optics need to magnify it by only about 6.1×, which is to say that the focal-plane sensor would need to be 6.1× further from the lens plane than the surface is.

Improving tracking with noise

You might think that this technique would require a specially printed pattern on the glass scale, and indeed that is the standard approach, but that is not necessary; you can simply store high-resolution photographs of the whole scale and match the image against them. Relatively efficient algorithms for this kind of thing are used for precise orientation by star tracking in spacecraft, but that really isn't important; a particle filter would probably be perfectly adequate, since most of the time your new position and velocity can be nearly extrapolated from the old ones. A meter is only about 6 million pixels long at 130-nm resolution, so probably a few tens of megabytes would suffice for a long enough map.

As I wrote in Dercuano in a note on sparkle servos, resolution can be improved with moiré effects. In this case, though, the "moiré grating", which obscures part of the field of view except for some subpixel-sized holes, needs to be in the optical near field of the glass scale to work, say positioned within 200 nm or so of it. But if another transparent object is scraping against the fused-quartz scale, not only can it deform it elastically, but also it will either scratch the glass, be scratched itself, or both. So I think you need either a lubricating film, an air bearing with high-quality filtered gas, or some other kind of arrangement to hold the two within a few dozen enanometers without making contact.

Other positioning feedback approaches

Hard disks routinely servo back to the same 80-nm-wide track reliably within a few milliseconds; the heads float on an air bearing tens of nanometers away from the constantly-moving disk surface. I'm not sure there's any practical way to reuse this amazing feat of engineering for nanopositioning.

To achieve higher positioning resolution than an optical system can provide, a scanning tunneling microscope ("STM") may be an option. These can provide subatomic precision (on the order of 0.01 nm) but are not normally used as positioning servos. Also, they are sensitive to shock and vibration. However, they can be homebrewed with a few

hundred hours of work and can operate in air.

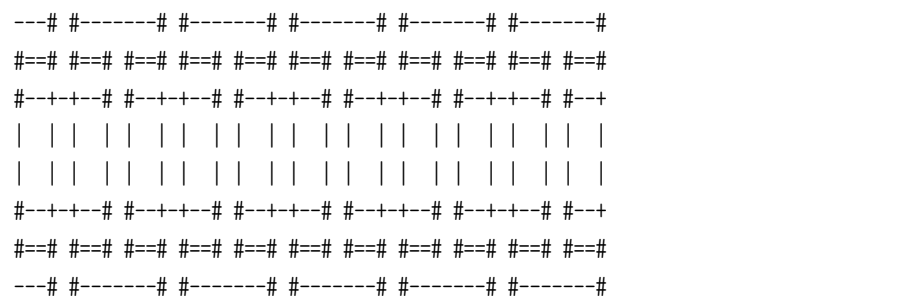
Capacitive sensors routinely achieve nanometer precision, but only over submillimeter distances. I'm not sure if there's a way to improve this.

Maybe a holographic approach with light has some hope of working?

Temperature compensation metamaterials

Metamaterials similar to gridiron pendulums could potentially take the place of fused quartz and Zerodur for applications like this.

You could imagine, for example, three concentric metal tubes, the inner and outer tube of steel and the intermediate tube of zinc, with the inner tube being interrupted at 0 cm, 2 cm, 4 cm, and every other even centimeter; the outer tube being interrupted at 1 cm, 3 cm, 5 cm, and every other odd centimeter; and the zinc tube being interrupted at every centimeter, and soldered to the interrupted steel tube at that point. In this way thermal expansion of the zinc shortens the tube, while expansion of the steel lengthens it, but by a slightly smaller factor (about 12 ppm/K against about 30–35 ppm/K for zinc.) By adjusting the kerf width at the interruptions, the expansion and contraction can be perfectly balanced.



Consider the section from 0 cm to 2 cm with an *effective* kerf width k of 1 mm. We have a total of 38 mm of steel pipe in the "kinematic chain": 9.5 mm on the outside, 19 mm on the inside, and another 9.5 mm on the outside, 40 mm − 2k. And we have 18 mm of zinc pipe, in two 9-mm lengths, 20 mm − 2k. Suppose its TCE is 32 ppm/K. If the temperature increases by 100° the zinc pipe will lengthen by 0.32%, to 18.0576 mm, thus shortening this section by 57.6 µm, and the steel pipe will lengthen by 0.12%, to 18.0456 mm, thus lengthening it by 45.6 µm. This already gives us a much reduced thermal expansion coefficient, but by widening the "kerfs" to a bit more than 3 mm to diminish the zinc with respect to the steel, we can reduce it to zero.

However, that zero depends on the precision of our effective kerf width (which extends roughly to the middle of the solder joint), on the precision of our estimation or measurement of the thermal coefficients involved (which vary with temperature), and on the precision of equality of temperature of the different components. Still, it should be straightforward to excel invar's 1.5 ppm/K performance.

The resulting object, if properly soldered, has strength similar to a single layer of the zinc tubing, if it were a continuous pipe, and a

lower stiffness — the compliance is roughly the sum of the compliances of the three pipes.

There are a number of common materials with smaller TCE than steel, but nearly all of them are brittle: soda-lime glass (9 ppm/K), limestone (8), granite (7.9–8.4), sapphire (5.3 or 8.1, depending on which part of that table you believe), tungsten carbide (4.9), brick masonry (4.7–9), graphite (4–8), industrial porcelain (4), borosilicate glass (4), wood parallel to the grain (3–5), silicon (3–5), mica (3, presumably parallel to the grain), carborundum (2.77), and diamond (1.1–1.3). Common materials in the table with substantially larger TCEs include basically all organic chemicals (with some polymers up into the hundreds), plaster (of Paris?) (17 ppm/K), 304 stainless (17.4), aluminum (21–24), fluorite (19.5), magnesium and its alloys (25–27), lead (29), wood across the grain (30), and rock salt (40.4). Kapton is notable in this table among organic polymers for having a TCE of only 20, though a different source gives 55 for "polyimide". Unfilled epoxies are claimed to be in the 45–65 range.

You can do this kind of trick with materials whose TCE is arbitrarily close together, but you need more layers of pipe; that's why traditional gridiron pendulums had seven vertical bars rather than five, because in the Victorian age they just had brass and steel, no aluminum or magnesium, and even zinc was comparatively exotic. To get by with just three layers you need materials whose TCE differs by more than a factor of 2.

By using materials further apart in TCE, such as brick masonry and magnesium alloys or rock salt, you may be able to get by with much smaller "compensatory pieces". Brick masonry has the additional advantage that it is cheap enough and hard enough that you can achieve very high stiffness with it at a reasonable cost.

A different metamaterial approach would be to use leverage to amplify small differences in thermal expansion into larger compensating contractions.

Topics

LED computation?

Kragen Javier Sitaker, 02020-10-25 (5 minutes)

Could you construct universal sequential digital logic with just LEDs?

It's straightforward to use LEDs for diode logic, which can give you sum-of-products logic, up to monotonicity — you can get any monotonic logical function that way, but diode logic alone doesn't give you inversion.

LEDs can function as photodiodes, although not very good photodiodes. So you could imagine using the light from one LED to switch another LED. But it would seem that you can't get any current gain that way: each charge-carrier pair that gets annihilated in the transmitting LED produces at most one photon, and then produces at most one charge-carrier pair in the receiving LED. And there are losses at every stage of this process, thanks to non-ideal quantum efficiencies and the like, so you can't even get to unity gain.

I think there are at least four ways to solve this problem, which sort of blur into each other.

• You can get *voltage* gain, because the voltage in the emitting LED will be close to its usual forward voltage, say 1.6V, while I think the voltage in the receiving LED can be much higher if it's back-biased, say 5V. But it's easy to trade that off for current gain by putting LEDs in series. For example, you can put three 1.6V LEDs in series and thus generate three photons per charge carrier.
• You may be able to increase the number of photons with fluorescence, though at the cost of speed.
• You can use a "regenerative" design using positive feedback, in which the back-biased receiving LED is in series with one or more forward-biased LEDs which also illuminate it. This way, most of the electrons produce one or more photons on their way to wherever they're going, thus allowing another charge carrier pair to spawn in the receiving LED.

(One problem that occurs to me with the above techniques is that it's going to be hard to get more irradiance at the photodetector junction than, like, in the rest of these diodes that are glued together.)

• You can initiate an avalanche discharge in the receiving LED and directly get current amplification after all, similar to how SPADs work. Like, if you're close to the diode's reverse avalanche voltage, maybe you can reduce that voltage threshold by varying irradiation, and thus get both voltage gain and current gain.
• You can get amplification through a bridge configuration. As long as you don't exceed the reverse breakdown voltage, an LED works as a (not very sensitive) differential voltage detector. However, this still suffers from a lack of current gain.
• You can use LEDs as if they were PIN diodes to switch RF signals by changing their capacitance with a DC bias, providing enormous current gain (like a JFET, leakage current down in the femtoamp range controlling an RF current up in the milliamp range) despite

below-unity voltage gain. But then how do you rectify the RF signal? A faster LED, I suppose.

Apparently a 5-mm red LED can generate over 20 μA as a photovoltaic diode in full sunlight, while 1N4148 diodes only generate about 10 nA. Assuming a 19.6 mm² area and 1000 W/m², the total solar power incident on the diode is 19.6 mW, which would be 12.3 mA at 1.6 V. So that's an efficiency of about 0.16%, compared with 16% for common low-cost photovoltaic cells.

Typically LEDs work better to detect slightly shorter-wavelength light, which is a major reason the red LED has such poor efficiency in sunlight. So that 0.16% might really be a quantum efficiency on the order of 2% or so in the right wavelength band.

These data make me think that getting even a current "gain" of 0.1 is going to be quite difficult with the first three approaches, much less getting a current gain above 1.0. The situation can be improved somewhat by heating up the LEDs you want to have lower bandgaps and cooling down the ones you want to have larger bandgaps, but maybe not to the point where those approaches are feasible.

Topics

Sequestered CO_2 would fill many oil fields

Kragen Javier Sitaker, 02020-10-25 (2 minutes)

Suppose we build big machines to remove CO_2 from the atmosphere to get back down from 400 ppm back to pre-industrial levels. If we froze all that excess CO_2 into a big block of dry ice, how big would it be?

The atmosphere has a mass of about 5.15×10^{18} kg, and 0.414% of that is CO_2, by volume. The average molecular weight of dry air is about 28.95 g/mol, while CO_2 is 44.01 g/mol, making it 1.52 times the density of dry air. (The CO_2 page on WP says 1.53 times.) This means that CO_2 is about 0.629 / (0.629 + 100 - 0.414) = 0.628% of the atmosphere by mass, or 3.25×10^{16} kg. Pre-industrial levels were about 250 ppm, so we only need to remove about 164 ppm, or 164/414 of the total; that's about 1.29×10^{16} kg.

The density of dry ice varies from 1.4–1.6 g/cc, so this would be about $8–9 \times 10^{12}$ m³ of dry ice, roughly a cube 20 km on a side. This is the volume of about 50–60 trillion barrels of oil, almost a thousand times the size of the Ghawar supergiant oil field, the largest among the 40 000 or so oil fields in the world.

If you pump the CO_2 back into the ground, the high pressure will convert it to highly compressed supercritical CO_2, almost the same density as dry ice, so this is probably a reasonable estimate of how much underground storage space is needed.

Topics

- Global warming
- Environment
- CO_2
- Atmospheric carbon capture

Residue number systems

Kragen Javier Sitaker, 02020-10-26 (2 minutes)

I was wondering about residue number systems and I was thinking that in particular bases that are close to a power of 2 are relatively easy to reduce modulo. Nathan Laredo came up with the list 63 62 61 59 55 53. (* 63 62 61 59 55 53) = 40978178010. 68 719 476 736 is the number of possible 36-bit numbers; 40 978 178 010 is a reasonably large fraction of that. All the fractional bits of loss between all those don't add up to even a whole bit of loss!

So consider, like, 1411 and 8675309. 1411 % [63 62 61 59 55 53] is [25 47 8 54 36 33]; 8675309 reduces to [20 21 11 8 49 4]. If we multiply these elementwise we get [500 987 88 432 1764 132], which reduces to [59 57 27 19 4 26]. And *waves hands* with the Chinese Remainder Theorem we can get the product of 1411 * 8675309 = 12240860999, modulo 40978178010 anyway.

This works for addition and subtraction; you can only get division if the bases are actually prime instead of relatively prime I think.

The reason this is potentially interesting is that six circuits to multiply two six-bit numbers modulo a six-bit base are a lot cheaper than one circuit to multiply 34-bit numbers, and have a lower path length to boot, so you can clock them faster. So RNSs like this get used a lot for high-sample-rate DSP.

Topics

COVID-19 risk and vitamin D

Kragen Javier Sitaker, 02020-10-27 (updated 02020-10-28)
(12 minutes)

Today the study "Vitamin D Status in Hospitalized Patients With SARS-CoV-2 Infection" was published. The key point from the abstract is

> Vitamin D deficiency was found in 82.2% of COVID-19 cases and 47.2% of population-based controls (p<0.0001)… No causal relationship was found between vitamin D deficiency and COVID-19 severity as a combined endpoint or as its separate components.

This strongly suggests a causal role for vitamin-D deficiency in covid hospitalization, and probably in risk of infection. This might explain some of the curious patterns, where the previously healthiest countries are the ones where COVID-19 has spread fastest.

There are several other such studies and a website that summarizes them.

Bayesian calculation from this study suggests a risk ratio of 4.2

Let's take a look at the basis for this statistic:

> including 216 patients aged ≥ 18 years, with confirmed COVID-19 admitted to the University Hospital Marqués de Valdecilla in Santander, Northern Spain from March 10 to March 31, 2020, and 197 sex-matched population-based controls recruited from the Camargo Cohort (14,15) during their last follow-up visit on January–March of the past year.

I'm a bit fuzzy on all this statistics stuff, so I'm going to walk through this in baby steps to make sure I get it right.

During that period Spain went from 3'258 confirmed covid cases to 111'541, out of a population of 47'400'000. This range of 34× during the study period makes it a bit difficult to do the calculations I want to do, but let's use, say, 50'000, since presumably the vast majority of the patients admitted to hospitals during that period were admitted toward the end. (And as we'll see at the end, this doesn't matter anyway.)

Presumably only a fraction of the people with COVID-19 were hospitalized; at present 157'881 people have been hospitalized out of 1'046'132 PCR-confirmed cases, or 15%. So maybe 7500 people were hospitalized with covid during that time, and maybe we can assume that the 82.2% number is typical of them: 6200 hospitalized covid patients with vitamin-D deficiency, 1300 hospitalized covid patients without it. Out of the total population, if we assume the 47.2% number from the previous year is typical, we have 22 million people who weren't hospitalized with covid and were vitamin-D deficient, and 25 million people who weren't hospitalized with covid and weren't vitamin-D deficient either.

So, 6200 out of 25 million vitamin-D-deficient people were hospitalized with covid at the time (250 out of every million people), and 1300 out of 22 million non-vitamin-D-deficient people were (59 per million). So the relative risk is 250 ÷ 59.

That's a relative risk of 4.2. If you were vitamin-D deficient in Spain at that point, you were 4.2 times as likely to get hospitalized with covid than if you weren't deficient, probably because the deficiency raises your risk of catching covid. A lot. This is a *huge* relative risk. (Or is it? Apparently risk ratios are the thing to use instead of odds ratios.)

Note that 6200 and 1300 are products of my estimate of the number of people hospitalized with 82.2% and (100% - 82.2%) respectively. So you get the same relative risk regardless of whether the actual number of hospitalized people was 750, 7500 or 75'000. (At 750'000 or more it might start to matter if people who later got covid were excluded from the "population-based control group" or not, but even now, in October, Spain hasn't hospitalized nearly that many covid patients.)

Possible confounding factors exist

Are there other explanations, other than vitamin D deficiency causing an increased risk of serious covid, probably through causing an increased risk of covid?

Well, the most obvious is that covid could *cause* vitamin D deficiency, for example by interfering with digestion or by directly depleting vitamin D stores. I don't know enough about vitamin D metabolism to be very confident in this, but I don't think it's very likely; as a fat-soluble vitamin, it can be stored for long periods of time, so I think the body usually contains a fairly huge amount compared to what it can use in the first week or two of a covid infection.

A second possible connection is for a common cause to produce both vitamin D deficiency and covid susceptibility. As Aaron Ferrucci points out, this could be something as simple as spending time indoors and not getting exercise outside.

The above is not exhaustive, but it hopefully clarifies that the posited protective effect of vitamin D against covid might not really exist, despite the astounding risk ratio computed above.

There are other recent papers like "Vitamin D and COVID-19", Bilezikian et al., that strongly suggest a causal mechanism, though that one cautions that it's "a putative clinical link that at this time must still be considered hypothetical."

Doses and sources: 2000 IU daily from mostly supplements

Normally 40 IU is 1 µg. I'm not sure if the weird IU-density variability thing that comes into play with some other vitamins is at play here, but for now I'll assume it's not.

The US RDA is 600 IU or 15 µg, with the tolerable upper intake level being 4000 IU or 100 µg; Australia and NZ instead recommend 10–80 µg/day, and the EU 15–100 µg/day, same as the US.

Given this, I'd think supplementing with a dose of some 2000 IU/day would be strongly advisable, as well as getting lots of UV-B exposure.

Gwern suggests it's important to take it in the morning, not at night, and reports that he's taking 5000 IU per day. He says overdose starts around 70'000 IU, so it might be a good idea to start the vitamin-D regimen with a single dose on the order of 20'000 IU. He also suggests that "an hour on the beach" is likely to give you 10'000 IU, and so this should be a safe daily dose. The Endocrine Society Clinical Practice Guideline on the subject counts 4'000 IU daily as "maintenance tolerable upper limits", and suggests that adequate blood levels "may require at least 1500–2000 IU/d[ay]". It confirms Gwern's thing about sunlight: a minimal erythemal dose (mild first-degree sunburn) is 20'000 IU!

Gwern also recommends vitamin D supplementation for life extension, quite aside from covid and nootropic reasons: it extends your life by an expected four months or so.

ChristianKI wrote a vitamin D primer on Lesswrong, recommending among other things to take vitamin K2 as well; this is a common practice for OTC supplements.

The CPG also mentions that the circulating half-life of the 25(OH)D form is 2–3 weeks, which reinforces my earlier-mentioned skepticism that a covid infection could drop vitamin D levels rapidly enough to provoke a deficiency in the study linked. And it mentions that body fat sequesters the vitamin, increasing the risk of deficiency, which might explain several puzzling things about covid, including how smokers are at lower risk for covid in countries with high obesity — except that smoking *lowers* vitamin D in Copenhagen and in Guangzhou, so the smoking link is nonexistent.

Damn, this CPG is a fucking goldmine.

Getting such large quantities of vitamin D from food is very difficult.

Vitamin D_3, cholecalciferol, is used as rat poison and possum poison with a LD_{50} of about 10 mg/kg, so if I were a possum the acutely lethal dose would be about 1.2 grams, about 48 million IU. In humans there are concerns with continued doses over 4000 IU per day, as mentioned above.

Fish: I'd need to eat 450 g of mackerel per day costing US$1.28

This can of *jurel* ("jack mackerel", the marketing name for horse mackerel, which is not a mackerel) says it contains 300 grams of fish and 12.5 grams of fat. This supposedly contains 4.6 IU of vitamin D per gram, so the can I just ate should have given me 1380 IU, two days' worth of the minimal allowance but only about 8 hours of the upper limit. I don't remember how much it cost, but maybe AR$150, 88¢, 640 microdollars per IU, or US$1.28/day for 2000 IU/day.

Eggs: useless

An egg only has about 44 IU, 1% of the upper intake level, 2% of my goal, and 14% of the US RDA. Eating four dozen eggs a day to get to 2000 IU is probably not a good idea. I don't think eggs contribute enough to be worth consideration here.

Cod liver oil: the best option at 15¢/day

Cod liver oil (*aceite de higado de bacalao*) as a supplement is 100 IU/g or 450 IU per spoonful, and eating several spoonfuls of it per day seems plausible (and is the recommended dose). 150 mℓ of cod liver oil goes for AR$825 in a bottle, which is US$4.85,or 3.2¢/mℓ, which is basically a gram I guess. This works out to 72 microdollars per IU, or 14.4¢ per day for 2000 IU/day.

Fortified milk: not useless but even worse than fish

This box of La Serenísima instant dry whole milk says it contains 400 g to make 3 ℓ, and a 200-mℓ serving contains 2.1 µg (84 IU, 42% of the daily value, which I guess we can deduce is 5 µg, ⅓ of the US/EU value and ½ of the .au/.nz value.) This serving supposedly has 26⅔ g of dry milk in it, so it's about π IU of vitamin D per gram of dry milk, a bit less than the fish. I think the price per gram is also similar or maybe a little higher. I'd need to eat 600 g, a box and a half of dried milk, per day, to reach 2000 IU per day. Eating nearly a kilo of dried milk per day, consisting mostly of lactose, seems even less appealing than eating hundreds of grams of fish.

On the plus side, it's a lot more feasible to eat 600 g of dried milk than it would be to drink 4.8 liters of milk.

Perhaps not entirely coincidentally, this supplementation level is precisely the maximum that would be allowed in the US.

The Armonía brand cut-rate instant dry whole milk is basically the same.

Pills: 3.7¢ per day, but perhaps less trustworthy

There are several brands of vitamin D supplements available; Puritan's Pride sells 100 softgels of supposedly 250 µg each (10k IU) for AR$3120 (US$18.35). This is 18.35 microdollars per IU or 3.7¢ per day. Now Foods sells 120 softgels of the same dose for AR$3850 (US$22.60), or 22.6 microdollars per IU or 4.5¢ per day. Their recommended dosage is one pill every three days, which seems pretty reasonable.

Lower dosages tend to cost about the same per pill rather than per IU.

Topics

Some of the cheapest memory ICs

Kragen Javier Sitaker, 02020-10-27 (updated 02020-10-30)
(1 minute)

I thought I'd look at some cheap chips. These are not the absolute best price/performance in any Pareto sense but maybe they're sort of close.

Memory

• US$0.10 for the STM M24C02-FMN6TP I²C 400kHz 256-byte EEPROM
• US$0.10 for the Atmel AT34C04-SS5M-B I²C 1MHz 512-byte EEPROM
• US$0.13 for the Fremont FT93C46A-UTR-T SPI 2MHz 1KiB EEPROM
• US$0.18 for the ON Semiconductor CAT24C64WI-GT3 I²C 1MHz 8-KiB EEPROM, reducing to less than US$0.15 in quantities over 100
• US$0.19 for the Catalyst 2156-CAT25C64VA-1.8-ND SPI 10MHz 8-KiB EEPROM
• US$0.27 for the GigaDevice GD25D05CTIGR 100MHz 64-KiB NOR FLASH with dual SPI
• US$0.29 for the Adesto AT25SF041B-SSHB-B 85 MHz 512-KiB NOR FLASH with quad-I/O SPI
• **US$0.15** for the Cypress S25FL116K0XBHI030 108MHz 2MiB NOR Flash with SPI and quad-IO "SPI" in a 24-BGA
• US$0.48 for the Winbond W25Q16JVSSIQ 133MHz 2MiB NOR Flash with SPI and quad-IO "SPI", down to about 38¢ in quantity 100
• US$0.76 for the Cypress S25FL064LABNFI043 108MHz 8MiB NOR Flash with SPI and quad-IO "SPI" in an 8-USON

Topics

Desiccant climate control

Kragen Javier Sitaker, 02020-10-27 (updated 02020-11-24)
(31 minutes)

In Muriate Thermal Mass (p. 459) I did some basic evaluation of the reversible hydration of the dihydrate of muriate of lime to the hexahydrate as a way to store up heat.

I concluded that it was a very attractive alternative to the miraculous salt of Glauber for domestic space heating applications, because although it needs to be heated to 45.5° to recharge it, by my calculations it holds 408 kJ/kg of heat energy and can produce temperature rises of well over 100°; and it's considerably more flexible in when it releases that heat, instead of trying to release it all the time and needing to be somewhat restrained by thick thermal insulation.

However, I now conclude that I had only begun to scratch the surface of the amazing possibilities of such systems; the Cromer cycle is just the beginning.

It may be that all of this is irrelevant now that photovoltaic panels are becoming so cheap, with module prices down to 15¢ per peak watt, that collecting the large amounts of solar heat that make these things attractive is actually more expensive than just storing energy in batteries. I suspect not, though; more exploration of this theme below.

Heating

In Muriate Thermal Mass (p. 459) I described the use of muriate of lime a desiccant for heating in the following process, which turns out to be well-known:

- Heat the desiccant to regenerate it, driving off water vapor, for example with sunlight.
- Store the dry desiccant at ambient temperature until it is desired to heat, for example, your floor, your blanket, the air in your house, or some hot water.
- Add water to the dry desiccant to produce the desired heat, which can be produced either at the point where it's desired, or transferred using some kind of liquid coolant. Using an efficient heat exchanger such as a countercurrent recuperator or pebble-bed regenerative heat exchanger can make the heat transfer almost complete.
- Store the spent desiccant until heat is available to regenerate it, for example from sunlight.

By itself this process can produce only a limited range of temperatures — with muriate of lime my simplified calculation in Muriate Thermal Mass (p. 459) estimates a temperature rise of just under 200°, but other desiccants may not perform so spectacularly, and I suspect that even muriate of lime would require high pressure in step #3 to achieve this, since otherwise the water will volatilize.

However, a cascaded heating cycle can achieve higher temperatures, as follows:

• As above.
• As above.
• As above, but instead of transferring the heat directly to the desired location, transfer it to more ambient-temperature dry desiccant and, separately, ambient-temperature water.
• Repeat step 3 one or more additional times, but using the now-hot desiccant and water to reach even higher temperatures. On the last repetition, transfer the heat to the desired location.
• Store the spent desiccant from all stages until heat is available.

If there were no heat losses and the absorption reactions were equally exothermic at all temperatures, this would give you unlimited temperatures, but neither of those is true.

But heating is just the beginning.

Cooling

Refrigeration is monumentally important, historically. The introduction of refrigerated cadaver ships is what made Argentina the richest country in the world for a shining few decades, since suddenly we could export our virtually unlimited supply of cow corpses to Europe. Lee Kuan Yew credits refrigeration, specifically air conditioning, for making it possible for Singapore to develop economically:

Question: Anything else besides multicultural tolerance that enabled Singapore's success?

Answer: Air conditioning. Air conditioning was a most important invention for us, perhaps one of the signal inventions of history. It changed the nature of civilization by making development possible in the tropics.

Without air conditioning you can work only in the cool early-morning hours or at dusk. The first thing I did upon becoming prime minister was to install air conditioners in buildings where the civil service worked. This was key to public efficiency.

A simple air conditioning cycle is as follows:

• Heat the desiccant to regenerate it, as before.
• Allow the dry desiccant to cool to outdoor temperature, then store it that way if desired. If necessary, rapid cooling can be achieved through either a heat exchanger with outdoor air or through direct contact with closed-cycle dry air which is itself cooled by a heat exchanger with outdoor air.
• Pass indoor air over the ambient-temperature desiccant to eliminate its humidity, though this warms it up.
• Evaporate water into the warm, dry air to restore its humidity and cool it to a lower temperature than before; vent this cold air into the indoor space.
• Store the spent desiccant until regeneration energy is available, if necessary.

Optionally, after step 3, you can cool the dried, warm air by running it through a heat exchanger with outdoor air. This improves the efficiency of the system. As with the heating cycle, you can improve the system's temperature range by cascading the reaction, cooling the air (and dry desiccant) through two or more stages.

Several different kinds of mass-exchange contact between the

desiccant and the air are feasible: liquid desiccant can be sprayed into an air column or a fountain, or pumped over pads of excelsior, as in a traditional swamp cooler, or air can be bubbled through it; a solid desiccant can be held in granular or foam form in a rotating wheel, or in a packed bed, or stuck to the surface of many flat plates in a sealed box or boxes, which may or may not double as solar thermal collectors.

This is a perfectly orthodox desiccant refrigeration system. It can be simplified to an outdoor evaporator and a pair of exposed fountains in an indoor space, one of water and one of desiccant, though this may be more advisable for oil of lime than for, say, oil of vitriol. In general all the options for contact with liquid desiccant are also options for contact with liquid water.

It can be used as well for refrigeration and even freezing of food or water, and water ice may be a denser and cheaper way to store cold until it is needed than as dry desiccant; water ice is 333.55 kJ/kg. Depending on the efficiencies of the cycle, though, it's entirely plausible that storage of energy in the form of some desiccants might be an even denser way to prepare for the need for cooling than water ice, and it certainly has a better shelf life and more flexibility.

Interestingly, brine of muriate of lime is commonly used in industrial refrigeration as a coolant rather than a desiccant — by virtue of remaining liquid down to -50° it permits the transport of a whole lot of cold in a very small pipe.

Dehumidifying

In addition to the above-mentioned heating and cooling applications, stored dry desiccant can of course be used to dehumidify indoor air simply by passing the air over it. And of course if you can heat up water, as explained above under "Heating", you can humidify as well by passing air over the heated water.

Dehydration sounds like an extremely niche use (raw-food vegans and a few other people have 500-watt home dehydrators), but I think it isn't; it just hasn't been available at a low enough price previously. Consider the Earthship's list of six basic human needs satisfied by architecture: energy, garbage, sewage, shelter, water, and food. Dehydration is directly applicable to three of those basic human needs: garbage, sewage, and food.

Garbage and sewage are mostly problems because they rot and stink and carry pathogens.

(Garbage is a more complex problem with many aspects, though: used motor oil, radium paint, nickel-cadmium batteries, demolition debris, linseed-oil-soaked rags, and so on; but food waste in particular, like these chicken bones I have here, is mostly a problem for those three reasons. And the physical volume of non-construction non-food garbage can be kept pretty minimal, like, cubic meters per person per year, or less. So I'm focusing here on food waste as the central core of the garbage problem.)

At Burning Man we deal with food waste first of all by dehydrating it, after which we can burn it or just store it in its inert dry state until it's time to carry it away. There, it's easy to dehydrate things: you

put them in one of those plastic netting bags they sell oranges in and hang them out in the sun and wind.

Abundant desiccant regeneration capabiity makes it possible to dehydrate food waste thus even in humid climates.

And similarly for sewage. At Burning Man, we just copped out and used chemical toilets, and at a local ecovillage here they used to use potash. Piss they would dilute with water and use as fertilizer, but shit they would pickle with wood ash from the cooking fires. Eventually, they switched to composting toilets, and getting those to work with aerobic mesophilic rotting instead of the usual noxious kind is a matter of partial dehydration: you cover your shit in the bucket with dry leaves, which also cut down on the relative nitrogen content, which gives you better rotting. Civilized people don't have dry leaves, so instead they use sawdust or coir or something in their composting toilets.

But, if you have ample desiccant capacity, you can use that to arrest decay completely, and then you can either burn the remains or you can bake them to kill all the pathogens and then use them as safe manure.

In the food category, there are a lot of foods that can be preserved for longer periods of time by dehydration than by refrigeration, although the change in flavor may be agreeable or disagreeable. Outside of the former Tawantinsuyu, food dehydration is usually done hot, which also changes the flavor of the food; in particular, dehydrated eggs are made by a desugaring and flash-spray-drying process that requires significant amounts of equipment and chemistry to replicate. Freeze-dried food in, for example, the US, is merely a novelty: Astronaut Ice Cream, etc. The tradition of chuño from Tawantinsuyu is unknown. And freeze-drying also changes the flavor of food, mostly through changes to mouthfeel, though chuño also owes its flavor to a fungus that grows during the process.

I think that food preservation by desiccant drying should be feasible at low temperatures, and might offer possibilities for food preservation with much less impact on flavor.

Thus a stored desiccant is a sort of all-purpose indoor climate control resource, capable of blowing hot or cold, like Aesop's traveller in the satyr's cave.

But wait! Don't touch that dial! There's more!

Water harvesting

Water supplies are a major concern for the humans' survival in many places, to say nothing of their ability to farm. But if you can dehumidify by sucking water out of the air into a desiccant — and the equilibrium relative humidity for some of these desiccants is very low indeed — then you can recondense that water when you regenerate the desiccant, particularly if you can chill a sort of cold trap to help the water recondense.

This allows you, in theory, to harvest an amount of water limited only by the available low-grade heat energy (solar or otherwise) and the amount of humidity in the air you can lure into the moisture vaporators of your moisture farm.

Alternative desiccants

Although in the above I chiefly referred to the properties of the muriate of lime, many other possible desiccants exist and could be thus applied, including alabaster, amorphous mesoporous magnesite, zeolites, silica hydrogel, ferric chloride, polyacrylate of sodium, pearl-ash, salt, sugar, silica aerogel, chloride of zinc, oil of vitriol, activated charcoal, soda-ash, lye, the bromide or muriate or nitrate of lithia, quicklime, oxide of phosphorus, nylon 6, carnallite, chloride or sulfate or perchlorate of magnesia, waterglass, infusorial earth, cellulose-rich waste plant material such as sawdust and straw, porous dehydrogenated hydroxide of alumium, fired clay ceramic, *unfired* clays, other desiccants, and mixtures of the above.

Different desiccants have different tradeoffs, and some may not be well suited to some uses; for example, perchlorate of magnesia must be regenerated under vacuum, quicklime must be regenerated at the inconveniently high temperature of 512°, and the hydration of alabaster only produces a temperature rise of some 60° or less. It's unlikely that muriate of lime is the optimal choice, but I haven't investigated the tradeoffs thoroughly.

Muriate of lime can work as an aqueous solution, avoiding the massive inconvenience of your lovely pebble bed deliquescing into a sticky, solid, impermeable lump, but I suspect that a porous solid mass of alabaster may have even better heat transfer properties than the aqueous solution of muriate of lime. And alabaster doesn't deliquesce; at worst it may crack a bit. As explained in Plaster Foam (p. 453), I got a nice porous alabaster biscuit by mixing calcined powdered alabaster with baking powder and baking it in a tin in an ordinary oven.

One of the chief figures of merit here is the price — either per joule or per kg of water absorbed. Other relevant quantitative information includes the minimal relative humidity the desiccant can reach, the temperature needed to regenerate it, the maximum temperature rise it can produce, its water capacity, and its energy capacity. Relevant qualitative information includes whether it is solid or liquid, its viscosity if liquid, its tendency to clump if solid (fixable in some cases with larnite or similar substances), and its hazards if spilled or inhaled.

But let's look at prices first.

Muriate of lime, or oil of lime, has been one of the chief desiccants used for centuries, and it is relatively cheap — US$1.60/kg here in Argentina, for example, as I noted in Dercuano. A few other candidates approach or excel this price. Slaked lime sells for US$0.12/kg at retail here, and alabaster as the hemihydrate for about US$0.40/kg, and the USGS gives its wholesale price as about US$8/tonne, thus US$0.008/kg, while giving the various potassa products including pearl-ash as closer to US$800/tonne (US$0.80/kg) as fertilizer, and soda-ash as US$150/tonne (US$0.15/kg). Lime is calcined from limestone, which the USGS lumps with crushed stone in general at US$12/tonne (US$0.012/kg), but the calcining and slaking process is a significant cost by comparison. Raw natural zeolites have their wholesale price given as US$50–300/tonne (US$0.05-US$0.30/kg), various clays as US$10–140/tonne (US$0.01–0.14/kg), and infusorial earth US$310/tonne (US$0.34/kg).

The USGS report on magnesia gives no explicit price but it seems to be about US$0.70/kg. For rock salt they say US$60/tonne (US$0.06/kg).

Waste plant material is often free or of negative cost, but often must be treated to stop fire and rot.

These bulk minerals, except for pearl-ash and magnesia, have wholesale prices in the two orders of magnitude of US$0.005–US$0.5/kg. Probably most industrially-produced materials are unable to approach that range, though perhaps a few, such as lye, oil of vitriol, and muriate of lime might make it.

L29Ah was kind enough to point out that the random Russian website opt6.ru offers a tonne of 99.2%-pure muriate of lime for 21000 rubles; a ruble is presently US$0.01294 reportedly, so that's US$272/tonne or US$0.272/kg. If this price is correct, it's toward the high end of the price per kg of the cheap desiccants, and 34× the price of calcined alabaster, but it's still kind of within the range.

Using the 408 kJ/kg number from [Muriate of Lime], this price works out to 1.5 MJ/$, 666 nanodollars per joule.

Alabaster is especially tempting due to its 34 times lower price per mass, and also because it doesn't glom together into a sticky mass when you regenerate it, though it can when you hydrate it.

Alabaster's molar mass is 136.14 g/mol anhydrous, 145.15 g/mol as hemihydrate, and 172.172 g/mol as dihydrate. Converting the dihydrate back to the hemihydrate is more difficult than with muriate of lime, requiring 100°–150°, and conversion back to the anhydrous form requires 180°. Upon hydration it can reportedly reach 60°. The key datum I lack here for comparison is the enthalpy of formation of the different hydrated forms.

Quicklime is notorious for producing enough heat to boil water when rehydrated, and it's very nearly as cheap as alabaster. However, regenerating it requires inconveniently high temperatures, and it's lethally caustic.

Farulla et al. characterize "thermochemical thermal energy storage" systems like these as storing 120–250 kWh/t, or 430–900 kJ/kg in SI units, much higher than sensible-heat thermal energy storage systems at 10–50 kWh/t (36–180 kJ/kg); but it claims TCTESs also cost €8–100/kWh (2500–32000 nanodollars per joule), far more than the €0.1-10/kWh (32–3200 n$/J) of sensible TES, identifying these high capital costs as a key reason for TCTES's non-adoption.

It seems plausible that one of the materials described above could deliver low capital costs, in the range of sensible TES costs or even lower. Farulla et al. are not unaware of these materials, and they even survey a number of published results from prototypes using them, as well as results designed for both heating and cooling. However, it seems that a great deal of research in the field has been focused on somewhat more exotic and therefore costly materials such as bromide of strontia, synthetic zeolites, muriate of lithia or baryta, and so on. I need to finish reading their paper.

Efficiency and comparison with electrical alternatives

Wholesale photovoltaic modules at 15¢ per peak watt at 20% capacity factor cost US$0.75 per average watt, which is 86.4 kJ per day. At a 6% annual discount rate an average watt amounts to 30.6 net present MJ in the first year, 59.3 in the first two years, 135 in the first five years, 235 MJ in the first ten years, 362 MJ in the first 20, 487 MJ in the first 50, asymptoting to 509.7 MJ at infinite time. (That is, although it produces 30.6 MJ per year, the 1580 MJ it produces in its first 50 years are only worth 487 present MJ to us at a 6% discount rate.) So photovoltaic modules work out to 680 MJ per US$ (at 6% APY).

If your thermochemical energy storage system can *store* 10 MJ per US$, which Farulla *et al.* say that current TCES systems don't come close to, how does that compare? How about the 2500–32000 nanodollars per joule (0.03–0.4 MJ/$, US$2.50–32/MJ) reported by Farulla *et al.*? How do you measure energy storage against energy generation?

Well, they aren't really commensurable. No amount of photovoltaic modules on your roof will allow you to run the air conditioner at night, and no amount of calcium chloride will heat or cool your house if it's all fully hydrated; the TCES as such trades off against batteries, not solar panels. And it doesn't trade off against all uses of batteries. And it also trades off somewhat against other climate control systems like vapor-compression refrigerators.

But when I was looking at balcony batteries a couple of years ago in Dercuano, lead-acid batteries cost US$23–73/MJ, which I don't think has changed much (though possibly lithium-ion will surpass them in a year or two). In crude terms this is about 1 to 700 times more expensive than a TCES, depending on whether you rely on Farulla *et al.*'s reports on existing prototype systems or my optimistic projections from possibly impractical but very cheap desiccant materials. But that doesn't include the cost of the vapor-compression air conditioning system itself.

Energy storage is strongly complementary, in the economic sense, to solar energy, and this is responsible for much of the interest in thermal energy storage systems in recent years. The cheaper TES is, the more valuable solar modules become; the cheaper solar modules become, the more valuable TES is. TES can't fulfill all of the energy storage needs for intermittent solar and wind energy, because it has very poor round-trip efficiency for mechanical energy, light, and so on. So batteries will still be needed.

(Still, for small low-power things like clocks and cellphones, TES might be a useful backup power source, perhaps using a thermoelectric generator or a Stirling engine.)

However, it's very likely that you can get more solar energy for your TES by gathering solar thermal energy than by gathering electrical energy with solar cells with an efficiency of 16% or 21%. And you can do it with collectors that are cheaper than photovoltaic modules, which still cost US$0.15 per peak watt. For example, you can use 1 m × 1 m × 19 mm boxes made of thin styrofoam, open at the top (one of the large faces), painted black on the inside, with plastic wrap wrapped around them to let light in, and smeared with a "chemical sunscreen" to slow UV damage. The airspace within

permits air to be blown through there, using a couple of holes in the back of the box, to harvest the heat. I think these will be about 30% efficient. The material would cost about US$20 for a 4'×8' sheet (3.0 m² in non-medieval units), so that's about 1000W peak for US$20, US$0.02 per peak watt.

(Rather than plastic wrap, you might be able to use UV-blocking polyester film intended for outdoor use.)

So solar collectors for a TES can probably be about a factor of 5 or 10 cheaper than photovoltaic modules.

Scaling down

One of the great advantages of thermochemical energy storage is that you don't need to insulate it. This, in turn, means that you can scale it down from building-sized systems to very small systems, and the stored energy has a shelf life of potentially years; "self-heating cans" have used muriate of lime for many years, for example.

You could thus scale these systems down to a wearable size, providing personal climate control.

Innovation considerations

If this is such an advantageous technology, why hasn't it been adopted previously? The humans have used fire to warm themselves for at least a million years. The calcining of alabaster goes back at least to Old Kingdom Egypt, the calcining of lime even further, to the Neolithic, before Çatal Höyük. Tubes of dried clay for guiding air date back, I think, at least to the beginning of iron smelting in the Hittite empire 4000 years ago. Evaporative cooling via the qanat goes back 3000 years in Iran. Texts purporting to be from 1200 years ago, by Jabir ibn Hayyan ("Geber"), described the "spirits of salt", and undoubtedly observed their action on chalk, producing bubbling and oil of lime. Émilie du Châtelet's discovery of energy was published in 1756, after Leibniz's pioneering efforts in the 1670s and 1680s. Thermodynamics was well-developed in the 19th century. Solvay began mass production of soda ash, with a byproduct of muriate of lime, in 1864. Gibbs described his "available energy" in 1873. Refrigeration and air conditioning was developed in the late 1800s, and the hazards of leaks of toxic and caustic refrigerants were such a major issue that Einstein and Szílárd patented their ammonia-absorption refrigerator without moving parts in 1930, a variant of the 1922 Munters–von Platen design, and Electrolux immediately put it into production; the same year, Midgley famously snuffed a candle with a breath of dichlorodifluoromethane, which he'd developed for the same humanitarian reasons, and which became the most popular refrigerant for decades. Harold Ellingham published his "Ellingham diagram" in 1944.

So the materials needed for thermochemical energy storage systems have been not only available but abundant for centuries, if not millennia; the theory necessary to design them for a century and a half; and they fulfill needs that have been universal human experiences for a hundred times longer than civilizations have existed. So, if these systems are so advantageous, why have they not been applied widely?

In the particular case of sewage, given the depth of mind-crippling tabus on the subject, I don't think we need much reason for slow diffusion of shit-handling innovations; the US still hasn't adopted bidets, for example. Squat toilets like the traditional Turkish and Japanese designs help greatly with constipation. The US has a huge constipation problem. Nevertheless, diffusion of squat toilets is actually negative, because aping the less-functional English design is more prestigious than using a design that works better anatomically. Garbage suffers from similar mind-crippling tabus, but they are less severe, and indeed garbage-handling practices have changed dramatically and rapidly in past decades.

So, in the case of garbage, but especially in the case of food preparation, air conditioning, and heating, I think we need a better explanation. There are some commercial installations using desiccant air conditioners, dating back to the 1980s in some cases, but it is not a widely adopted technology. There are even a few cases of using thermochemical energy storage for both heating and cooling in this way, though I haven't seen previous suggestions of using a single thermochemical energy store for so many different purposes: space heating, heat for cooking, air conditioning, air dehumidification, food refrigeration, food dehydration, garbage dehydration, and sewage dehydration.

Moreover, many of the deployed and research systems use expensive desiccants such as lithium bromide; I can't find any trustworthy sources on its cost, but I doubt it approaches the US$0.27/kg price of muriate of lime or US$0.008/kg of alabaster. Lithium carbonate costs US$13/kg and is 18.8% lithium, making the lithium cost US$69/kg. Bromine costs US$2.19/kg. Lithium bromide is 8% lithium and 92% bromine. This suggests a cost of US$5.50 for the lithium and US$2 for the bromine, per kg of lithium bromide, thus US$7.50.

I tentatively suggest that perhaps what I am proposing in this note has not been tried, though I cannot imagine why not.

A complement to compressed air? Maybe

Compressed air has been a widely used temporary storage form for energy for over half a century; air-powered tools are common in auto shops all over the world, and the non-electric Amish in particular have developed quite an economy of compressing air with windmills, shipping compressed air around in tanker cars, storing it in enormous underground tanks, and so on, with the objective of easing their work without becoming dependent on the "English" for electricity.

One of the disadvantages of compressed air energy storage is that, when air is adiabatically compressed, much of the compression energy is lost as heat rather than being stored in the compressed gas. Another is that when room-temperature compressed air expands, it cools, and this cooling can condense water out of it, which tends to cause various kinds of problems in compressed-air-powered and compressed-air-handling machinery.

The solution, in theory, is isothermal compressed-air energy storage, where the air is cooled to maintain a constant temperature as it is being compressed, and heated back up as it is being decompressed.

Doing this on a small scale is difficult, because doing it the normal way requires access to some kind of "heat absorbing and releasing structure" connected to a huge heat reservoir, such as a lake or the ocean, to keep its temperature change minimal. But phase-change and thermochemical energy-storage systems have the possibility of absorbing and later releasing massive amounts of heat without changing their temperatures; thermochemical systems additionally have the possibility of releasing the heat at a lower temperature than it was initially provided. This reheating not only improves the efficiency of the energy storage device; it also avoids condensation.

Topics

Bluepill aspirations

Kragen Javier Sitaker, 02020-10-30 (updated 02020-11-01)
(9 minutes)

So I got a Blue Pill for US$4. This one has a CS32 rather than an STM32 in it; it says CKS32F 104C8T6 NMCM4 2013 A or something on it. Later today the ST-Link should arrive. All I know so far is that the board weighs 6 g, measures 22 mm × 52 mm without the pin headers soldered, and lights its power LED when plugged in over USB. So I am currently in that delicious limbo where everything is possible and nothing is yet difficult.

(Aha, now my ST-Link has arrived, and the board powers up with it correctly. The programming connector is labeled with reasonable clarity 3V3 SWIO SWCLK GND, matching 8 of the 10 pins on the ST-Link; missing are 5.5V, RST, and SWIO.)

CS32

People have reported success flashing these with openocd after tweaking its config a bit; they also mention that the GD32 mirrors Flash into RAM, presumably at startup, which presumably causes slower startup but apparently faster run speed and maybe lower power consumption; the CKS part may or may not be the same. The CKS chip supposedly has 64K of RAM instead of 20K (or 40K?), which might be a plus. Also, 128K of Flash instead of 64K. But then others in that thread reported that it really only has 20K. And so does the datasheet.

It was hard to find the Chinese datasheet but I did finally find it in an stm32duino forum thread. Another page of the forum thread has another datasheet. It seems to have two 12-bit 1 Msps ADCs, some kind of DAC including an LFSR, 2 I²C channels, 2 18Mbps SPI channels, CAN (!), 3 USARTs, JTAG, USB, an RTC, etc. (I'm not sure the STM32 part has all of these!) Officially the clock only goes up to 72 MHz.

Apparently you can run it at 80 MHz, though not the 120MHz or 128MHz of the GD32 parts. User Macbeth says they couldn't get the stm32duino bootloaders to work on their *possibly* CKS chips (which claim to be brand-name STM32) but could get mecrisp to work:

> I figured these dodgy STM32s just have a crippled USB port but then I flashed 'mecrisp' which is an implementation of Forth running over its own USB serial and it works perfectly! Very odd.

Maybe it would be worthwhile to look at the boot ROM to compare it to ST's with `$ st-flash read system.bin 0x1fff000 2048` using the st-flash program from Arduino's STM32Tools. User ag123 says the bootloader matches ST's.

Aha, I measured R10; it's 1.5kΩ as it ought to be, not 10kΩ as in the original Blue Pill design. Hopefully the silkscreen hasn't been switched around, so that R10 is the correct R10.

Possible things to try

Clearly the first thing to do is to try to get it to blink an LED.

Then it would be nice to get a USB bootloader programmed to see if that works, so that I can program it (with the Arduino IDE?) without the ST-Link dongle.

The next thing to try is probably something with audio: generate some bytebeat and try to feed it into my microphone port (or this speaker's audio in) with a hacked cable. Hmm, better find those audio plugs I scavenged... although alligator clips or copper wire twisted around a phono plug would work too. It has real DACs, too, not just PWM.

Next thing is probably hooking up these potentiometers I have here as analog inputs.

Next thing after that is to get a voltmeter running with analog inputs, limiting diodes, and a voltage divider or three. (Though see Multimeter metrology (p. 502).) If USB serial is working then I can transmit the result over USB serial. Otherwise I'll be limited to blinking an LED or doing speech synthesis or something. Or a modem.

Being able to run it off two AAA (or AA or C) cells, or a CR2032, would be pretty handy. I should try that. In theory it should be good from 1.8V to 3.6V.

From voltmeters, the next steps branch out: oscilloscope-style data acquisition at 1 Msps, ohmmeter, milliammeter, and thermometer, using some random component as the temperature sensor. In particular I want to calibrate it to use these quartz-halogen lightbulbs as temperature sensors, but I suspect that at room temperature. Averaging ADC readings over 16 seconds should in theory permit adding an extra 12 bits of precision to the ADC, giving us 24 bits of precision. But what is the temperature sensor being measured *against*?

As Weston said in his Weston-cell patent, a voltage standard that varies with temperature (like the Daniell cell or the Clark cell) is dependent on the thermometer for its precision.

I think it has some kind of ability to write to Flash under program control. This should enable me to tell whether it was turned on while disconnected from USB.

I'd like to then see about using scavenged LEDs as light sensors and photocells. Three or four LEDs in series ought to be enough to provide it with 5V which can then get regulated down, but this probably involves disconnecting the power LED. (Too bad it's in the wrong direction to use as a photocell...)

It would be good to verify that the part really does have 128K of working Flash; some forum users report only 64K.

Some kind of component ID thing? Like an LCR meter. This is pretty similar to complex impedance sensing, which would be useful for identifying materials. Touch sensing and electrical impedance tomography seem like promising next directions to move in.

A logging wattmeter should be pretty simple, but will need multiple boards. And megohm resistors.

All of the above (except generating audio and blinking an LED) is just measurement exercises. Although those are useful, some actuation would be super helpful too. The simplest thing is probably

turning a motor (or an ATX power supply?) on and off with a transistor. Maybe I can use a triac or something to put a timer on the water pump.

Of course the noblest of all actuators is the spark, capable of marking metal, perforating plastic, and deodorizing apartments. The spark can also sense; in particular it can detect flame. The board ought to be able to time such events to within ±6 ns.

Given some kind of temperature sensor and heating element, it ought to be able to do PID control of temperature. These 240V 70W quartz-halogen lightbulbs are under 150Ω at room temperature, over 180Ω at 100°, and $(240V)^2/70W$ gives 822Ω at operating temperature; cf. Thermistors (p. 431). Although XXX I may have miscalculated something in there... anyway, if the resistance stays the same, at 83 mA I should be able to get a watt out of them, 120 mA for 2W, 190 mA for 5W, 260 mA for 10W, 373 mA for 20W. At 3.3 V the most I can get is 76 mW, but 12 V should be able to give 1 W.

Dimming some colored LEDs also seems promising.

If I can rig up some kind of weighing scale, maybe using capacitive sensing, I ought to be able to measure materials more precisely.

A couple of electronic gadgets I've been putting off doing anything with are this PAL acoustic delay line and these linear servos from inkjets.

Speaking of PAL, bitbanging PAL or NTSC would be a pretty awesome thing to do, although in practice probably bitbanging VGA would be easier and more useful.

Another extremely awesome thing to do with it would be to get it to be a USB HID so that it can type on my computer.

Topics

Multimeter metrology

Kragen Javier Sitaker, 02020-11-01 (updated 02020-11-27)
(23 minutes)

A standard TL431 voltage reference is only accurate to ±0.5% at best, more often ±2%. TI's REF5050 is accurate to ±0.1% or ±0.05% in the "high-grade" version, with 3 ppm/K temperature drift and 50 ppm/1000 hours, but those cost US$5–10. The internal voltage reference in the STMF103C8 used in many Blue Pill boards is specified as 1.16–1.26V over the -40° to 105° range, an error of about ±4.2%, with a temperature coefficient of 100 ppm/K. The datasheet for the CKS32 in the Blue Pill I got says the same thing in §5.3.4 内置的参照电压: its V_{REFINT} (内置参照电压) goes from 1.16V (最小值) to 1.26V (最大值).

This is pretty shitty fucking accuracy. 4.2% is 42000 ppm. For measuring temperature with a tungsten wire near room temperature, a ±4.2% error is a ±10° error, and it gets larger at higher temperatures. Using a standard TL431 ripped out of some scrapped power supply we can cut that error to 19000 ppm or so, but that's still nothing to write home about. How the fuck are you supposed to build a fucking voltmeter? Or anything that depends on voltage for accuracy?

By contrast, the quartz crystal on the Blue Pill is 8 MHz with, probably, an error of about 10 ppm, common for watch crystals. The damn chip can measure voltages to 1.5 digits of accuracy and time to 5 digits of accuracy, 4200 times better. (And it can count electrical pulses to a lot more accuracy than that, to the point that the concept of tolerance stops being meaningful.) We can get down to probably 1 ppm timing error if we can measure the temperature to compensate, 0.1 ppm if we can control it. (And nowadays we can reference to GPS.)

Like, something similar to the open-source Transistortester AVR by Karl-Heinz Kübbeler and Markus Frejek, now commonly known as the "M328 Transistor Tester" (Instructables) (English manual) or "LCR TC1 ESR meter", which sells for US$38 here in Argentina or US$13 in the US. But with better accuracy, precision, and repeatability, and ideally without any bought components.

The 0.1%-precision LM4040CIM3X-20-NOPB voltage reference costs US$1.67 from Mouser.

How good are multimeters usually?

Some random YouChube video from TheSignalPath (#175 (ii)) found about 25 ppm of voltage difference between his not-recently-calibrated Fluke 744 Documenting Process Calibrator and his not-recently-calibrated 8-digit Keithley DMM 7510, so tens of ppm is achievable in ordinary electronics labs, but expensive; he apologized for the lack of calibration, so apparently he was expecting better. His resistance error was worse, 4600 ppm at 100Ω, 400 ppm at 1kΩ, 20 ppm at 10kΩ; this pattern makes me suspect that this error was due to not using a 4-wire measurement. His current error was about 250 ppm. Testing the 744 against its own meters he instead got 400 ppm emf error and 500 ppm resistance error; he neglected to thus test its current measurement capability.

I tested a shitty digital multimeter from 2016, US$6 from the

hardware store, against another similar meter to get an idea of how bad they are, and also how aggressive they are. It seems like the old multimeter's diode-testing range uses up to 1.4 mA at up to 2.72 volts. Their diode test readings differ by about 2%, resistance by about .03%, voltage by about .3%. pretty impressive metrology for US$6. These are lower bounds on their actual errors, but they're unlikely to be conservative by more than an order of magnitude or so.

Canceling out voltage errors in measurements

Ratiometric voltage measurements with the dual ADCs ought to be a lot more precise, and probably limited only by the 12-bit bit depth and whatever the noise is. So, for example, if we run an exactly known *current* through a tungsten lightbulb and measure the voltage, we're subject to that shitty ±4.2% precision; but if we put some *unknown* voltage across a series combination of the lightbulb and an exactly known *resistance*, we can get a much higher-precision measurement.

Resistance standards

How do we get an even approximately known resistance, though? Typical resistors used to be ±20%, now they're ±1%, but nothing close to 1 ppm. On MercadoLibre we can buy a temperature standard resistance made of Weston's manganin, intended for converting precise voltage measurements to current measurements or vice versa, for US$20. At 25° manganin's temperature coefficient of resistance crosses zero, having fallen from 6 ppm/K at 12° on its way to -42 ppm/K at 100° and -52 ppm/K at 250°, before crossing zero again at 475°. This suggests that a ±10° error of temperature measurement around 25° would give you about a ±5 ppm error of resistance. However, these resistors are not so precisely made, having a ±0.5% error, 5000 ppm; one wonders if they are correctly annealed or even truly made of manganin. Better precision calibration resistors are available.

A better option might be a known capacitance; you can easily measure the RC time constant as long as your voltage reference isn't too noisy over the short time involved. $C = \varepsilon A/d$, so it's straightforward to construct a capacitor with a known capacitance; however, if we want its capacitance to be known to within, say, 10 ppm, we need less than 10 ppm error on all of ε, A, and d. A is relatively easy: a 1 m x 1 m square of foil is a square meter to within 10 ppm if its width and height are each accurate to within 7 ppm (i.e., an average of 7 μm) and it's square to within $\cos^{-1}(1-10\text{ppm})$, which is about 15 minutes of arc. Getting d accurate to within 10 ppm is more difficult; if the dielectric is intended to be 100 microns, its average thickness must be correct to within 1 nm. But getting ε accurate to within 10 ppm is feasible in only one way: vacuum. Even the permittivity of air is 590 ppm higher than vacuum, and it varies according to pressure and temperature. Common solid dielectrics are hopeless; WP gives polypropylene's relative permittivity as "2.2–2.36", i.e. ±35000 ppm. A vacuum capacitor of these dimensions would have a capacitance of 88.5419 nF.

Oversampling

Every time we quadruple the measurement time, we add another bit of precision, because the variances of Gaussian noise combine additively. So the sum of four measurements has four times the variance, thus twice the standard deviation, as a single measurement, and their average thus has half the standard deviation. The Blue Pill's ADCs run at 1Msps, so if we take 1048576 samples, 1.05 seconds' worth, we can get a 22-bit-precision reading, which has a quantization error of ±0.125 ppm. Probably averaging 65536 or 32768 samples is a better tradeoff, giving you more like 1 ppm error in exchange for much faster data acquisition.

Hamer's monograph on Clark, Daniell, and mostly Weston cells

In despair, I turned to a shitty scan of NBS Monograph 84, from 1965-12-15, titled, "Standard Cells: Their Construction, Maintenance, and Characteristics", by one Walter J. Hamer. He explains to some extent the construction of the Daniell cell, the Clark cell, and the Weston cell, which served as laboratory references from 1836 until 1990. A Daniell cell, using copper, zinc, and their sulfates, is relatively straightforward to construct, but drifts rapidly over time owing to the mixing of the sulfates, depends strongly on the purity of the copper and the zinc, and has a horrific temperature coefficient. The Weston cell was the first to have a roughly zero temperature coefficient, which it achieves by the use of cadmium salts, which unfortunately are rather difficult for me to procure.

Hamer also goes into the history of other metrological standards for voltage (a word he disdains to use, preferring "emf") and other electromagnetic units. He points out that in theory you can use an absolute electrometer as a standard, using the permittivity of free space and precise measures of current, but that this gives an error on the order of 100 ppm. (He doesn't go into detail, but it turns out this is a matter of charging a known capacitance, constructed as above, to a known voltage or charge and measuring the resulting force; one version is an electrostatic balance where the electrostatic repulsion between two plates is countered by a sufficiently precise weight to return them to their original position.)

I suspect that you could construct a microscopic absolute electrometer that would give you much more precise readings; all the absolute electrometers I can find in the literature were on the order of 100 mm in diameter with on the order of 10 mm plate separation, and so involve weights on the order of 10 N but electrical forces on the order of 10 mN when operated at a few kV, and are additionally tricky to operate under vacuum. If you were to scale it down by a factor of 10,000, you would have capacitors with 10-µm-diameter plates separated by 1 µm, weighing on the order of 1 ng; with those precise dimensions it would be 0.88541878 femtofarads. But what would the force be?

In a parallel-plate capacitor, $C = \varepsilon A/d = q/V = 2E/V^2$, so the capacitance varies directly with the plate area, inversely with the plate spacing, inversely with the voltage at a fixed charge, directly with the energy at a fixed voltage, and inversely with the squared voltage at a fixed energy. If you were to start with the plates in contact (but miraculously not discharging; maybe they're triboelectric insulators like packing tape and its adhesive) the capacitance would be infinite, so the voltage would be zero. If you then pull the plates apart to some distance, you have to add some energy as the capacitance starts to fall and the voltage correspondingly to rise, proportional to the distance.

Now, if we measure dE/dd at some distance then we get the force between the plates there. $E = \frac{1}{2}CV^2$, and as we pull the plates apart at a fixed charge the capacitance drops (proportional to distance) and the voltage rises (also proportional to distance): $q/V = \varepsilon A/d$, so $V = qd/\varepsilon A$, so $V^2 = q^2d^2/\varepsilon^2A^2$, and $E = \frac{1}{2}CV^2 = \frac{1}{2}\varepsilon Aq^2d^2/\varepsilon^2A^2d = \frac{1}{2}dq^2/\varepsilon A$. So that means the force between the plates is *constant*: dE/d$d = \frac{1}{2}q^2/\varepsilon A$ regardless of the distance (as long as the parallel-plate approximation is valid). So for example if we charge this cap to 1 kV it would have 0.88541878 picocoulombs on it, producing a force of 0.88541878 millinewtons.

Actually you can't charge it up that far because after 20–40 MV/m you get field emission across the vacuum which limits you to 20–40 V/μm. Say you charge it to 10 V instead. Now it's only 88.541878 nanonewtons.

That doesn't sound like a whole lot, but it's 8000 times larger than its weight instead of a thousand times less, so you could easily make it overwhelmingly the largest force on the capacitor. It's a lot less than an atmosphere, though.

As a standard of resistance, he describes the construction of an air-core inductor of known dimensions and thus computable inductance, and the measurement of its E–I relationship at different frequencies to obtain a precise standard for the ohm; but he describes the Wenner method, which seems to be some kind of differential measurement that I don't fully understand, getting a ±5 ppm precision. He also mentions using the rotation of a magnet in a coil, or a coil in the earth's magnetic field. (The quantized Hall effect is the modern absolute standard since 1990.)

He says that using computable capacitors (like my microscopic thought experiment above) to check the ohm measure is "less involved" and "may be used [text lost] an annual basis" to check resistance standards against absolute units in preference to the inductive approach. (He also says they normally used 1-pF computable capacitors rather than the 88000-pF jobby I used above for calculations.) "Thompson-Lampard theorem" seems to be the key term here.

Given the possibility of measuring a computable inductor or computable capacitor with a microcontroller, I'd think that it would be easier to do the measurements in the time domain rather than the frequency domain.

Note that a precise standard of resistance, or a precise standard of both time and either inductance or capacitance, allows you immediately to convert between precise current and precise voltage. Time is, as I said, easy enough now.

Hamer then explains that an absolute standard of current is available in the form of a current balance, in which you measure the electromagnetic repulsion or attraction between two conductors against a standard weight, and of course this is the standard definition of the ampere; he says this is about ±6 ppm. The method he describes requires some kind of measurement over time, but I think current balances are more commonly used in a quasistatic fashion, in which after adding a mass, the current is increased until repulsion returns the mass to its original position, thus requiring only measurement of length and mass to calibrate. And I think that in fact they are commonly used nowadays for measuring masses in terms of known currents rather than vice versa.

Force balances have the stunning metrological advantage that the difference in force produces an acceleration, the acceleration integrated over time produces a velocity, and the velocity integrated

over time produces a displacement. So even a very small force imbalance can produce an easily visible displacement, particularly if you use it to deflect a mirror reflecting a laser pointer across the room.

It's feasible, though not an everyday occurrence, to get weights that are calibrated to within 1 ppm, though care must be taken to compensate for local gravitational fields and atmospheric pressure and humidity. Atmospheric pressure can diminish by as much as 14% in hurricanes, and of course varies by much more when you go up in elevation or underwater; a 100 g steel weight that occupies 12.6582 mℓ under some conditions is displacing about 15 mg of air, diminishing its apparent weight by 150 ppm. This can decrease by 2.5% from humidity or 14% from weather-related pressure variation, causing a deviation of some 20 ppm. Thermal expansion and contraction of the weights is not a concern; the 36 ppm/K volumetric expansion coefficient of steel means that the above-described weight will occupy 12.6628 mℓ at a temperature 10° higher, displacing about 5 µg more of air, lowering its apparent weight by 5 µg or 0.05 ppm.

Joe blocks, similarly, can measure lengths on the order of 100 mm down to a precision of 0.1 µm or so, though they expand and contract by 12 ppm/K, thus requiring 80 mK control to reach 1 ppm.

The Clark cell, Hamer says, was adopted in 1893 as 1.434 volts at 15°, though the modern value is 1.4328, and its temperature coefficient is -1.15mV/K. It uses a zinc or zinc amalgam anode, a mercury cathode, and a saturated aqueous electrolyte of zinc sulfate, plus mercurous sulfate paste as a depolarizer, to prevent hydrogen buildup on the plate (converting it instead into oil of vitriol; removing impurities in the mercurous sulfate lowered the voltage by another 300 µV). This would probably also be challenging for me to fabricate. We can estimate that a ±10° temperature error would result in a ±11.5mV error, about ±8000 ppm, which is why the Weston cell was adopted as the new standard in 1908.

The Weston cell has a temperature coefficient of about 41 ppm/K, substantially better than the 800 ppm/K given above for the Clark cell.

We can see why the NBS controlled the temperature of its Weston-cell room to within 1°, the temperature of its oil baths for the Weston cells to within 10 mK, and the oil baths' temperature during measurements to within 1 mK. Such measures applied even to the Clark cell would have reduced its temperature-induced voltage error to some 0.8 ppm, at which point other errors would surely dominate.

Calibrating voltage by measuring power in a bomb calorimeter

It occurs to me that power might be a useful way to calibrate voltage given a known current or especially a known resistance, because if you can measure the power of a known resistance to within 1000 ppm, you know the voltage to within 1 ppm; moreover, $\Delta T = \int P\, dt / C_{th}$, so you get the same kind of integration effect as with force

balances, but only to the first order, not the second order. Somewhat surprisingly, although specific heats vary with temperature, some are in fact known to the required precision; for example, according to definitions.units, the energy to raise a gram of water from 14.5° to 15.5° is 4.18580 J, the energy to raise it from 19.5° to 20.5° is 4.18190, and the energy to raise it from 0° to 100° averages 4.19002 per degree. So you could imagine, for example, heating 1000 kg (±1 kg) of very-well-insulated water by applying an unknown voltage to a known resistance (±1 ppm) from 0° to 100° (each ±10 mK) and measuring the time required (±1000 ppm), and thus getting a measurement of power to a precision of 1000 ppm, and thus measuring the voltage to 1 ppm. The imprecision of the specific-heat number only adds about 1.3 ppm to the *power* imprecision, and thus 1.7 parts per trillion to the voltage imprecision.

XXX hmm, maybe I screwed up that resistance calculation too?

XXX all of this stuff about calculating from power is based on a wrong logic step. 1000 ppm power error gives you 500 ppm voltage error (499.9 to be exact), not 1 ppm.

The main sources of imprecision in such an experiment would seem to be the insulation of the water, which would have to leak less than 0.4 joules during the experiment, and the original measurement of the resistance.

Computable inductors are easier to isolate from the environment

Which brings us back to the resistance-measurement problem. It occurs to me that it a computable *inductor* might be a more precise way to measure a resistance, particularly if it can be made very small in physical dimensions so that its magnetic dipole does not impinge on materials with a substantial permeability; wood's, for example, is 0.43 ppm higher than the vacuum, a situation three orders of magnitude more promising than the corresponding situation with capacitors, and teflon's is even closer (how much closer is not known).

Differential measurement of computable capacitors or inductors

A different tack might be to use some kind of differential measurement to precisely calibrate the dielectric constant or permeability of some material whose presence cannot be avoided but whose quantity can be varied. Operating the same air-gap capacitor at various air pressures, for example, might enable you to extrapolate its capacitance at hard vacuum, without having to actually produce 100-mPa vacuums.

Sensors

All these "sources of error" can equally well be seen as "observable variables". The only difficulty is untangling them. If your capacitor's resonant frequency (with a given coil) varies linearly with the air pressure, then by measuring that resonant frequency with 1 ppm accuracy, you can measure the air pressure with 1-ppm accuracy. If it also varies dramatically with the air's moisture content, well,

congratulations, you have a moisture sensor (p. 566) too, as long as you have some other way to sense air pressure that varies differently with humidity. (Inversely, ideally, or failing that, not at all.) Everything varies with temperature, but if some things vary more than others and you can keep it at the same temperature, you can measure the temperature. If you find something that varies a *lot* with temperature, maybe like the leakage current in a Schottky, you have a very precise temperature sensor, which you can use to cancel out temperature effects on other things, as long as you can characterize them. Force of gravity makes your watch crystal run faster when it's sideways? Great, kid, you gotcherself a MEMS accelerometer that costs 25¢.

A photoelectric-effect voltage standard?

Can we use light of a precisely known color — a low-pressure sodium line, for example — to produce photoelectrons with a precisely known energy, and thus a photocell with a precisely known cutoff potential? Unfortunately the work function of the cathode material is subtracted from that energy, and this depends on the purity (and identity, and crystal grain orientation) of the surface material, and the temperature also affects the cutoff potential slightly. Typical variations between differently prepared samples of the same element are ±10%.

Still, I suspect this might be a feasible way to get some sort of voltage standard, even if not a very good one.

It might be possible to precisely measure aging-stable photoemission in air if you use a metal with a sufficiently conductive and nonhygroscopic oxide, such as zinc.

You also get photoelectric emission inside solid-state semiconductor junctions; that's how photodiodes work. However, I suspect the cutoff voltage for this solid-state photocurrent may be fuzzier, just because of the messy nature of condensed matter.

Topics

Guide to finding datasheets and avoiding malicious datasheet SEO sites

Kragen Javier Sitaker, 02020-11-02 (updated 02020-12-22)
(7 minutes)

Getting datasheets used to be easier, but it's gotten harder on the modern web due to SEO scum. Google is almost useless. https://yandex.ru/ gives much better results for datasheet searches. (Compare the results for searching [bp2842 datasheet] on the two engines: Yandex gives you datasheets for the TI TL2842BP, which may be a slightly wrong chip, but Google gives you random bullshit. The term "Даташит" may be helpful,) However, you run into CAPTCHAs occasionally on Yandex. For non-obsolete parts, Digi-Key is often a better source. Still, though, you need to blacklist some providers.

Yandex puts a link saying "pdf Посмотреть" after search results that actually point to PDFs. Do not follow this link; it goes to docviewer.yandex.ru. But it does allow you to distinguish PDF links from fake links. (Except that the SEO scum sometimes generate fake PDFs.)

Most of the sites mentioned have their own search engines.

Octopart has its own search engine, and it's useful if you want to buy things, and it does provide datasheet links, but for example searching for bp2812 on Octopart gives you sealed lead-acid batteries. With a datasheet, mind you. Searching for "c3205" (the marking on the common 1990s 2SC3205 transistor) similarly produces no useful results.

Known-good sources

• datasheetspdf.com: iframe with PDF datasheet accessible from the "Download Foobar3103 datasheet" link as well as the "Foobar3103 datasheet" links in the left column. C3205, for example.
• ndatasheet.com: alternate domain for datasheetspdf.com, saying, "The site has been moved : DatasheetsPDF.com" at the bottom. BP2812, for example.
• chipdip.ru
• datasheet-pdf.com: iframe with PDF datasheet in third iframe on page. C2878, for example. Decent filenames too. Apparently scraped from datasheet4u.
• njr.com: direct links to PDF show up in Google, but only for their products. NJM4565, for example.
• Mouser: datasheet link after "Datasheet:" saying "Foobar3103 Datasheet (PDF)", but only for current products. NJM4565, for example.
• digchip.com: iframe "ifr data" on page reached from "Download Foobar3103 datasheet" link; BA5936S, for example.
• datasheet4u.com: iframe with PDF datasheet on page reached from

"PDF Download: *[IMG] Foobar3103 datasheet PDF*" link, same as datasheetspdf.com. C3199, for example.
• kazus.ru: iframe "datasheet pdf" contains PDF datasheet with unreasonably long filename, on page reached by posting "[Foobar3103.PDF (338 Kb)]" form. TL2842BP, for example.
• rlocman.ru: link "Скачать" to vendor's site via a redirector. TL2842BP, for example.
• power-on.tech: links labeled "Datasheet Foobar3103" and "Скачать Datasheet Foobar3103" on main page. UC2842B, for example.
• chinesechip.com: PDFs directly linked from Google, albeit with goofy GUID firenames. BP2812, for example.
• ibselectronics.com: PDFs linked directly from Yandex, with good filenames. BP2832, for example.
• onsemi.com: PDFs linked directly from Yandex and Google, with good filenames, but only for ON Semiconductor products. Note that they've recently fucked us over by breaking the Fairchild links from Digi-Key.
• st.com: PDFs linked directly from Google, with good filenames, but only for ST products. With shitty filenames.
• alltransistors.com: iframe and "Abrir como PDF" links of the form https://alltransistors.com/pdfdatasheet_panasonic/2sd1512_e.pdf on "Foobar PDF datasheet" page with URL of form https://alltransistors.com/es/pdfview.php?doc=2sd1512_e.pdf&dire=oo_panasonic on link of form "2sd1512_e.pdf" on "Foobar . Datasheet. Equivalente. ..." page linked from Google, for example, 2SD1512.
• datasheetcafe.com: link under header "Foobar Datasheet" labeled [FOOBAR.PDF] linking to link of the form http://j5d2v7d7.stackpathcdn.com/wp-content/uploads/2015/09/Too-T2140.pdf, for example, TT2140.
• yoreparo.com: people post service manuals in the forums.
• tvsat.com.pl: datasheets directly linked from Google, e.g., 2SB985
• datasheet.octopart.com: datasheets directly linked from Google, e.g., 2SA1015
• pdf.voron.ua: datasheets directly linked from Google, e.g., 2SA984
• vishay.com: datasheets directly linked from Google, but only for their products (including old Siliconix parts)
• nxp.com: similarly, but including old Freescale and Philips parts; e.g. the mc9s08sg32
• infineon.com: similarly, but including old International Rectum Fryer parts
• irf.com: similarly, though they redirect to infineon.com (with a working link to the PDF)
• datasheet.lcsc.com: similarly, but they carry a huge array of current parts, including many even Digi-Key doesn't; for example, the HT7333

Broken at the moment

• ru.datasheetbank.com

Malicious but sometimes usable if nothing else works

• datasheetarchive.com: iframe with PDF accessible via "PDF" link in "PDF" column — but for the wrong part! A1286, for example. However, I *did* get the right datasheet for CXA1498S.
• kynix.com: links to alldatasheet.com, for example for STP7N60FI.

Blacklist; *never visit* (at least if you want the datasheet)

• radiolibrary.ru: provides lots of information in HTML, in Russian, but no datasheet
• datasheetq.com: iframe "contentpdf" on "DOWNLOAD" link redirects to home page. A1286, for example.
• web-bcs.com: refreshes to page with no PDF; A1286, for example.
• datasheet.es: not only no PDF link, but malicious SEO alt text ("Foobar3103 arduino") on links to PNGs that have *had all the text removed*. "PDF descargar" link with "download.php?id=foobar" points to HTML page containing only malicious cloaking JavaScript redirecting you to the HTML page. CXA1498S, for example. Does have text ripped from the PDF in HTML, though.
• datasheet26.com: same as datasheet.es, but in Russian. BP2812, for example.
• datasheetcatalog.com: no PDF link; link labeled "Download *Foobar3103 pdf datasheet* from FOOCORP" is JS, linking to an URL ending in ".pdf", but that page is HTML and just links you back to the original page and similar ones. CXA1498S, for example.
• y-ic.com: generates PDFs with no actual information about the chip, containing only advertisements. BP2812, for example.
• transistordata.com: PDF pages are 404; also has hits for things it doesn't have datasheets for
• assets.nexperia.com: 403 Forbidden for wget
• worldwayelec.com: generates fake datasheet PDFs containing only ads, has no actual information on parts; AVC479, for example.
• alldatasheet.es/alldatasheet.com: previously quite difficult: PDF with application/octet-stream content-type and .html URL ending, accessible via form POST of "[Download]" button, on page accessible via "Download" link to URL of form `https://pdf1.alldatasheet.es/datasheet-pdf/download/35940/ROHM/BA3126N.html`, in a locked filing cabinet with a sign saying "Beware of the jaguar"; BA3126N, for example, or TL2842BP. Was a last resort. Now the button says "[If You Want to View Datasheet, Click To Here !!]" instead, but doesn't work.
• ic-components.com: generates fake datasheet PDFs containing only ads; AS12W-K, for example.

Transistor part numbers

kludge explains:

There are three religions: Japanese numbers, Pro-Electron numbers, and American JEDEC numbers. Japanese numbers all start with 2S so they don't bother printing the 2S part. American ones start with 2N but they print it. Pro-Electron ones have two letters for type and then more numbers.

So if you have a Japanese transistor that says C3205 on it, maybe look for 2SC3205.

Surface-mount parts are a bitch. There's a book I've seen somewhere...

Topics

Audio vector image

Kragen Javier Sitaker, 02020-11-04 (2 minutes)

I was just thinking about transmitting a live vector image over analog ultrasound in as simple a way as possible, for example to provide an oscilloscope display. The minimum is an X coordinate and a Y coordinate, with brightness/blanking being optional; if we want the image to stay "live", rather than "slow-scan", we need at least 24 fps. If it's analog there isn't a defined X resolution or Y resolution, more a signal-to-noise ratio kind of thing. So then we have the question of how complex a picture we want to be able to encode.

Probably the minimum interesting picture is a single letter, which is about two or three cycles per frame, and so at least 48 Hz of bandwidth per channel, probably more like 100 Hz per channel. You could encode the signal in any number of ways: AM, suppressed-carrier single sideband, FM, and so on. AM uses up twice the bandwidth but is otherwise pretty identical in characteristics.

The next step up might be a word, maybe 500 Hz per channel for five letters, maybe 15 or 20 cycles per frame. This is sufficient for simple oscilloscope waveforms, too, if you modulate the X scan to slow down before reaching sharp peaks and valleys. Simple analog circuitry can't do that, though.

The next step up in complexity might be 3000 Hz per channel, and at this point I'm going to guess that this is about 60 curved lines, or a reasonably elaborate drawing. At this point, if we're trying to fit in underneath the 20kHz cutoff of many audio systems, people with good hearing will be able to notice the signal, because it'll be 14 kHz to 20 kHz.

Topics

Dead bugging

Kragen Javier Sitaker, 02020-11-04 (3 minutes)

As mentioned in Ghettobotics Nonshopping List (p. 516) I measured the individual copper conductors in a stranded power cable as 100 μm, a distance usually described as "the width of a human hair" — my beard is human, perhaps, but apparently not my head. I thought I'd work out a little bit about the properties of 100-μm wiring.

This would be 38-gauge wire on the AWG scale, and it should be about 2.2Ω per meter. (This may be correct, but I'm not able to measure it with much accuracy. My meter measures a short circuit as 0.8Ω and a 150-mm length of this wire as 2.8Ω. At least it doesn't say 10Ω.) So, for circuits with diameters in the millimeters with impedances in the kilohms, this wire is perfectly adequate for carrying voltages around.

If it's carrying 100 mA it should drop 0.22 V/m, or 0.22 mV/mm, and dissipate 22 mW/m, or 22 μW/mm. So you might want to use multiple strands for your ECL chips' V_{CC} and GND pins, or for your precision analog supplies.

So perhaps you can prototype a circuit by gluing a bunch of SMD components to a piece of paper or wood, a millimeter or two apart, then soldering this hair wire to their terminals, all under a microscope. You should only need a few micrograms of solder per joint, and melting that solder shouldn't require much heat. The hair wire doesn't come insulated, but at centimeter scales it's sufficiently rigid and elastic that it won't easily produce accidental shorts on the workbench from vibration or other random forces; nevertheless you might want to do some conformal coating with nail polish or lacquer or something before sending your little circuit out into the big, scary world. In some cases you might even pot it.

A spot welder might be more appropriate equipment than a traditional iron that relies on thermal conduction. Thermal conduction is necessarily fairly slow, so if you're soldering onto a large thermally-conductive mass, you need to heat the whole thing up, not just the solder for the connection. This increases the energy needed by orders of magnitude. Still, a regular soldering iron does work.

Non-surface-mount components, as well as BGAs and the like, can perhaps be handled by dead-bugging them, gluing them with their legs in the air.

Here's a photo of three such wires soldered to three of the pins of a through-hole SZIP, taken with my hand computer's built-in camera, which I used for the soldering as well:

Topics

Ghettobotics nonshopping list

Kragen Javier Sitaker, 02020-11-04 (updated 02020-12-21)
(22 minutes)

So, I have some basic gifted, bought, or lent electronics equipment: a multimeter with a broken lead and I think broken current measurement, a low-temperature hot glue gun, an audio cable, an amplified battery-powered speaker, a non-temperature-controlled soldering iron that leaks some mains current into the workpiece, a Blue Pill, an ST-Link, a couple laptops, a netbook, some USB chargers and cables, a USB power pack, some tin snips, some yogurt cups, some acetic transparent silicone caulk, a couple of cellphones running Android, some quartz-halogen lightbulbs, some butane lighters, a butane blowtorch, some box cutters, a vise-grip, nail polish, ethanol, some nail-polish diluent, some Q-tips, needlenose pliers, a scrap terry-cloth towel, some electrical tape, a carpentry tape measure, a hammer, some chopsticks, an electric screwdriver, some hex tips for it, and some popsicle sticks. My current objective is to bootstrap from this to a reasonable electronics lab, buying a minimal additional amount and replacing as many as possible of the bought items with custom items, made out of garbage.

Of these, I think it's reasonable to list the following as reusable electronics-workshop equipment:

- a multimeter
- glue gun
- soldering iron
- Blue Pill
- ST-Link
- box cutters
- needlenose pliers

Recent repairs and improvisations

Although listing my numerous follies would probably help people to avoid repeating them, these are just the successes.

Soldering sponge

The soldering iron came with a stamped-steel stand to hold the tip off your workbench. But to clean the tip of a soldering iron (and also to reduce the temperature of a non-temperature-controlled iron like this one) you need a tip cleaner. Typically this is a wet cellulose sponge. But all the sponges here are polyurethane, which would not work; it would melt onto the tip instead of cleaning it.

So I cut a square of some 70 mm from the terry-cloth towel with the tin snips, wet it, and put it under the soldering iron stand. The towel turns out to be cotton, so this works well.

Multimeter lead repair

I repaired the broken multimeter test lead by using some modeling wax (paraffin + rosin) as flux. To solder the wire, I carved back the plastic around the broken wire with the box cutter, stripped the wire

with the flush cutters, fluxed the wires, tinned them with some solder from generous joints in a discarded VCR, and touched them together, held in my hand.

To reinsulate it, I should have used hot glue, but instead I used transparent silicone caulk; this sticks inadequately to the plastic, isn't stiff enough to provide adequate strain relief, took a day to cure, and may turn the wire into copper acetate.

To stiffen it, I squirted hot glue all over it, wrapping it all the way around the silicone in places, since the EVA sticks inadequately to the silicone. The hot glue didn't stick adequately to the plastic, making it a strain concentrator instead of a strain relief; this turned out to be because a thin layer of silicone was in between.

I melted the hot glue with a butane stove lighter, scraped the molten hot glue out of the way with a popsicle stick coated in silicone, and then removed the intrusive silicone. Then I used nail polish diluent and a Q-tip to scrub the plastic, then applied more hot glue to complete the repair.

Solder braid

I cut a meter from the power cord of a pressure washer I had found disassembled and discarded on the sidewalk a few months ago; after slitting the end of the outer insulation with a box cutter, I was able to pull one of the inner insulated conductors out to tear the outer insulation lengthwise, just using my hand. It contained two stranded copper wires made of dozens of 100-µm-wide strands. (I measured using the cellphone as microscope; see below.). I separated out three hanks, trimmed them with the flush cutters, tied them into an overhand knot, and began braiding. After I had 220 mm of braid, I cut it at both ends with the flush cutters. I then heated it with the soldering iron to coat it with the modeling wax as flux.

This solder braid makes it enormously easier to scavenge soldered parts with large or numerous pins.

Cellphone as microscope and flashlight

My cellphone can focus down to about 70 mm. This body is so old that it can no longer focus its eyes closer than about 210 mm. The cellphone is a little less clear than these eyes; they can read 3.8-mm-tall text from 1600 mm, while the camera can only read it from about 900 mm. (This works out to about 3.8/6/1600 ≈ 400 µrad ≈ 1'20" of resolution for the eyes, or 700 µrad or 2'30" for the camera; double these numbers for resolution in terms of distance between lines.) So even without any additional optical magnification the cellphone can provide about 1.7× magnification of small objects, and of course it's easy to expand the resulting blurry pictures on the screen to be much larger, which I measure as 20× the linear size of the original object; this doesn't make any new details visible but does make the same details more obvious.

It is of course common to use cellphones as flashlights or to use an LED on them to illuminate an object being photographed.

Attempts to use cheap pocket Fresnel magnifiers to improve the resolving power of the cellphone have so far met with failure. I'm considering buying one of those "60×" clip-on "cellphone

microscopes”.

Closeup image quality is improved substantially by supporting the cellphone on top of a table on a a 75-to-76-mm-tall triangular prism, with triangle sides of 55, 55, and 60 mm, folded from a 75-mm-wide strip cut from a cardboard box discarded by the supermarket using box cutters, measured with the tape measure, marked with a ballpoint pen, and taped into shape using electrical tape. The triangle firmly supports the cellphone's center of gravity and the place where you have to tap to take a photo, without obstructing its camera, “flash”, light sensors, or power button. This reduces non-parallelism, eliminates motion blur, and keeps the imaging range near optimal (as close as possible without losing focus.) So this probably gets to better than 1.7× resolving power, maybe 3×.

Both of the cellphones I happen to have here are more or less equivalent for this.

password2 points out that there's a program called scrcpy which might be useful for this, allowing me to disable the screensaver and display the cellphone's screen on a laptop or netbook connected over USB or TCP/IP. This would afford greater magnification, because those screens are bigger, though of course not greater detail, and probably with more latency.

My equipment wishlist

In addition to the stuff above that I already have, I want:

- helping hands
- a bench power supply
- magic tweezers (i.e., with an integrated LCR meter)
- a second multimeter (with working current measurement and a buzzer)
- alligator-clip wires
- breadboards
- an oscilloscope
- a temperature-controlled hot-air setup
- a temperature-controlled soldering iron
- a 3-D printer
- a custom PCB fabrication machine such as a mill
- an autonomous solar-powered personal computer
- a stock of consumable parts (see below)
- some way to organize parts
- a workbench
- a portable tool chest for carrying all this around with a place for every item
- a precision multimeter
- a printer
- a dremel
- a Kelvin-sensing power supply
- thermometers
- a Kelvin-sensing ohmmeter
- a thermal imager
- a function generator
- a caliper
- a protocol analyzer

- a microscope, and
- micro-scale waldos.

These are not in priority order.

Desired consumable parts

I want:

- solder
- solder braid
- flux
- resistors
- capacitors
- LEDs
- inductors
- microcontrollers
- connectors
- breadboard jumper wires
- transistors
- triacs
- batteries
- supercaps
- crystals
- op-amps
- linear regulators
- switchers
- buttons and other switches
- photodiodes
- phototransistors
- Darlingtons
- potentiometers
- other linear and rotational sensors
- MOSFETs
- motors
- motor drivers
- MOSFET drivers
- speakers
- microphones
- temperature sensors
- heating elements
- diodes
- zeners
- voltage references
- level converters
- humidity sensors
- pumps
- valves
- displays
- cameras
- accelerometers
- gyros, and
- touch sensors.

These are not in priority order either.

Breadboards and jumper wires

A standard solderless breadboard is extremely useful for prototyping and testing circuits. But the original breadboards were actual boards, made of wood, used for the same purpose; to connect two or more components, you would put their leads under a washer, dished outwards, and screw the washer to the board to pinch their leads under its edge, making a tight connection. This is feasible but far less convenient than the now-standard approach.

In my childhood, I would take government welfare cheese boxes and poke electronic components through the non-corrugated, non-plastic-coated cardboard. Panel-mount components I would poke through all the way, then screw them down to the cardboard as normal; for through-hole components, I would poke just the leads through. Then I could connect the leads inside the box using alligator-clip jumpers. All of this was made easier by the use of parts that hadn't been used yet, so the leads were still long, and the near complete nonexistence of surface-mount parts. This approach is convenient; construction paper is of the appropriate weight, and so are the inner and outer walls of single-wall corrugated cardboard, which can be soaked apart in water.

A disadvantage to this approach is that it doesn't scale to more than a few dozen connections.

At the time, I also had a Radio Shack "300 in 1" electronics kit, which had about 50 components mounted on a piece of cardboard next to some extension springs. By thumbing a spring to the side, you opened up space between its coils to insert one or more wires into.

In some YouChube video, *Espacio de César* demonstrated the construction of a now-conventional solderless breadboard from a pile of 2.54-mm-pitch DIP sockets; he snipped out the middle of the sockets, stacked five socket sides together, and wired them up in the conventional way. I suspect that even a very modest effort in this direction would yield useful results: three 8-pin molex female connectors or socket sides stacked up would be sufficient for a fairly wide variety of circuits with discrete through-hole components.

Other candidate sources of such female connectors include connectors on cables and jumpers for configuring boards by connecting two such adjacent pins.

The usual way to use the now-conventional breadboard is with jumper wires made of hookup wire with nothing on the ends, but you can also solder traditional male pins onto the ends of wires in order to make it easier to plug and unplug them.

The absolute minimum hardware for that kind of convenient plugging and unplugging *of pins* is female-female jumper wires: basically a single-pin female molex connector soldered onto each end of a jumper wire. You engulf a lead of a component with each pin, thus setting up a two-terminal net. Nets with three or more terminals can be constructed with Y-shaped female–female–male jumpers, which additionally have a third wire attached with a male pin on it; these can be daisy-chained to make nets of any degree of complexity.

These female–female jumpers seem to be more or less the standard that the Arduino world is converging on: each component is on its

own little PCB, with some 2.54-mm-pitch pins sticking off the edge. (I've even seen such a PCB for an ULN2803, which is a 2.54-mm-pitch DIP with eight Darlingtons on it.) You can either plug it into your breadboard or you can hook up each of the pins directly to some other pin on some other board with a female–female wire. Often, 1970s-style rainbow cable is involved between the two ends of the cable.

If I rip apart a largish motor, dead or otherwise, I can get many meters of thick magnet wire out of it, which is directly suitable for use as breadboard-style hookup wire once you sand its ends. For very-low-frequency or very-low-current circuits, twisty ties from bread bags can also be used; and jumper wires from single-sided PCBs can sometimes work for short distances.

So, connectors are the most immediate, urgent necessity.

Tool chest

How can the lab fit into a portable toolchest? Suppose it's 15 kg, if it has wheels, and maybe 200 mm × 400 mm × 500 mm, an acceptable size to carry on the bus, stuff under your seat, or carry through a doorway. (This ATX tower PC in front of me is 200 mm × 450 mm × 450 mm, a very similar size, though a marginally more inconvenient one.)

The toolchest needs to be able to contain the 7 pieces of equipment I have, plus the 27 other pieces of equipment I want, and the 44 kinds of consumable parts. In a very crude averaging sense this allocates about 190 g to each of these 79 "items", including the toolchest itself, and just over 500 mℓ per "item". Because any item larger than these averages must be compensated for by items additively smaller, the vast majority will need to be smaller than that; I think it makes sense to "budget" 40 g and 100 mℓ per "normal item" in order to have space left over for the few that can't squeeze in that way.

As some motivating examples, this multimeter — a lightweight handheld one — weighs 152 g without its leads, 202 g with its leads, and occupies 70 mm × 25 mm × 125 mm without its leads, 219 mℓ. The flush cutters weigh 64 g; the soldering iron weighs 174 g with its stand and "sponge". The iron is some 230 mm long and, with the cord, 60 mm in diameter, thus occupying some 650 mℓ. The flush cutters occupy 150 mm × 60 mm × 10 mm when closed, thus some 90 mℓ, the only item to come in under the 100 mℓ "normal item" threshold, though they exceed the weight threshold by more than half.

So the vast majority of "items" need to be much smaller than this, and probably these three in particular need to be replaced by smaller and lighter items.

Suppose the items have a Zipf distribution of weight and volume, which is maybe pessimistic but probably not much. Then the first ten weights would be 3 kg (perhaps the 3-D printer), 1.5 kg, 1 kg, 750 g, 600 g, then 500 g, 430 g, 380 g, 340 g, 300 g; the next ten between 150 and 275 g; the next ten between 100 and 150 g; the next ten between 75 and 100 g; the next ten between 60 and 75 g; the next ten between 50 and 60 g; the next ten between 43 and 50 g; and the last nine between 38 and 43 g. In particular this gives the quartiles as

about 50, 100, and 150 g. And the top few volumes out of 40 ℓ are 8 ℓ, 4 ℓ, 2.7 ℓ, 2 ℓ, 1.6 ℓ, etc.; and the quartiles are roughly 130 mℓ, 200 mℓ, and 400 mℓ.

This suggests that my "normal item" quotas above were maybe a bit too stingy. A "normal item" can be 100 g and 200 mℓ; every item over the threshold must be compensated for by an item below it. Still, of the three test items, only the flush cutters are under these thresholds.

Suppose the resistors are close to the median and I have, say, 1000 resistors. (An SMD sampler book I was looking at online has 25 resistors of each of 170 values, 4250 resistors in all. It costs about US$8 overseas but about US$50 in Argentina.) Then the resistors average 100 mg each and 200 μℓ each. Right now I have about 50 resistors (five of which are trimpots) and they weigh about 3 grams, so I think this is realistic.

But how can I keep them organized? This is actually far more pressing than the problem described above of how to reduce the multimeter's weight by half; to build circuits with the hundreds of components I've already scavenged from discarded machinery, I must be able to find the components I need quickly, I must be able to connect them together, and then I must be able to probe the resulting assemblage in some way to find out why it's broken.

Ideally I'd press the "heat" button to heat the smart tweezers' tips to desoldering temperature, pick up a component with them, and the smart tweezer would chime, indicating that the component had been successfully characterized; I could then press a "print" button with my foot, storing the test results and printing a tiny envelope describing the component and giving the crucial test values. Then I would put the component into the envelope, and file the envelope in the larger envelope for its type.

A common 100-mW 0805 resistor is 2 mm × 1.25 mm × 0.5 mm, totaling 1.25 mm³ or 1.25 μℓ; you can fit 800 of them in a milliliter. So the whole collection of 4250 resistors mentioned earlier would, without the paper, amount to some 6 mℓ.

Resistor envelopes

I'm trying out categorizing physicaly-small resistors by E12-series resistance and putting them into paper envelopes, one for each E12 value (10 12 15, 18 22 29, 33 39 47, 55 68 82). Most of the resistors I've salvaged so far are in the 100Ω–100kΩ range, which requires 37 envelopes, much like Merlin Mann. There are a few higher and quite a number lower, but so far I don't have an accurate way to measure resistances below 100Ω. Hopefully I can rig something up soon.

So far I only have on average two or three resistors of each denomination, many with very short leads. I probably need at least four times the number of resistors I have already, a couple hundred, to make even smallish projects straightforward.

I found my helping-hands to be very helpful for resistor measurement: I can connect one lead of the multimeter to one alligator clip, place the resistor in the other clip where I can see it through the glass, and touch its other lead with the other probe. I got

up to about two resistors a minute that way. I suspect that a better ohmmeter and using alligator clips will speed me up further.

I also have a partitioned plastic box, like those for fishing tackle, which is partitioned into E3 resistor values.

Bench power supplies

I have a 12-volt power brick that includes current limiting, as I found when I used it to electrolytically dissolve some copper; for some 45 minutes the output voltage was well below 12 volts. But I need something where I can twist a knob to scan across a range of voltages. The "Tech Ideas" YouTube channel from India points out that there's typically a TL431 on the low-voltage side of modern isolated switching power supplies, directly hooked up to the feedback optocoupler, and the TL431 is programmed with a voltage divider, so you can replace its fixed voltage divider with a pot, and maybe upgrade the output-side filter caps, and you get typically a 3V–25V adjustable switcher, with whatever current limiting the original supply had.

One difficulty is that you typically want to know what voltage you're getting, so you probably want to add some voltmeters on the output. Also it's common to want voltage down to zero; for low-power loads you can do this very easily with a linear output stage consisting of an emitter follower following a potentiometer.

Topics

Foaming infiltration

Kragen Javier Sitaker, 02020-11-06 (1 minute)

Infiltration of a porous object is a common and very useful technique for strengthening, waterproofing, or otherwise altering shapes fabricated initially by a process that inherently produces a porous green body or other shape.

One difficulty is that you can't infiltrate with solids; the infiltrant needs to be liquid. In some cases it would be possible to infiltrate with a colloid or suspension, but once the solvent is removed, the shape becomes porous again.

Various ways of causing solid grains of material to expand into a foam are known: shooting rice out of steam cannons, heating waterglass (or vermiculite or perlite) to drive out the water, baking muffins to react their baking powder, and so on. If such processes are applicable to the infiltrant grains and produce a closed-cell foam, they can seal up the porosity.

Construction spray foam illustrates another approach to foaming via heating: the polymerization reaction of the foam produces heat which produces the foaming gas.

Topics

Hard sticky balls

Kragen Javier Sitaker, 02020-11-06 (1 minute)

Bearing balls are commonly machined to unbelievably tight tolerances, like, deep submicron roundness, I think. This is exploited in many kinds of precision machinery. They're also quite hard and cheap.

So if you take two bearing balls that are coated in some sort of sticky liquid or plastic material such as clay, and you press them together until their surfaces touch, their centers are a very precise distance apart. If the sticky substance then hardens without expanding, it will preserve this very precise distance.

If you have three bearing balls thus all mutually attached, they will have not only precise distances between them, but also precise angles. If you add a fourth in contact with the first three, it will be in a precisely located position in space relative to them, as long as it doesn't shove them apart.. So, too, will any further sticky balls stuck onto the mass.

This permits the construction of arbitrarily large shapes with micron-level precision and some degree of geometric freedom, which is larger if you use multiple different sizes of balls. If the balls are millimeter-scale or smaller, you can get very substantial structures with reasonable strength. If surfaces are finished with non-sticky balls, those surfaces too will be precisely located.

"Voxel-based 3-D printing" is a name sometimes used for this sort of process.

Topics

- Manufacturing (p. 799) (17 notes)
- Digital fabrication (p. 802) (17 notes)
- Bearing balls

OCR with linear optimization

Kragen Javier Sitaker, 02020-11-06 (1 minute)

Can you use linear optimization to efficiently solve the OCR problem?

One reasonable linear objective function might be a weighted sum of the entropy of the text and the number of wrong ("noise") pixels, or perhaps the total absolute pixel error. You can use the standard one-of-N mixed integer linear programming approach to select which glyph is at a given position, and a reasonable entropy function might use digraph frequencies in your language of choice. You might be able to use some design variables that indicate the X-Y position of each glyph, and the font would perhaps be a shitload of parameters.

Alternatively, simple hill-climbing with a nonlinear problem state ment might be simpler to formulate and work adequately; it might "learn" the font simultaneously with the position and angle of text on the page. Gradient descent could more efficiently provide fine glyph positioning and adjust the glyph contents. (Or just simple mean/median?)

Topics

- Mathematical optimization (p. 816) (9 notes)
- Gradient descent (p. 887) (3 notes)

Pit firing

Kragen Javier Sitaker, 02020-11-06 (3 minutes)

One of the problems with refractory materials is that they tend to be brittle at room temperature. The ductile-brittle transition of a material, if it exists, tends to be not too far from its melting point, so materials that are ductile at room temperature tend to not survive 1000° or 1500°, with a few exceptions.

As a result, when they heat up and cool down, they tend to crack.

The Neolithic solution to this was to pit-fire pottery: you bury it in a pit and throw burning coals on it, then, perhaps, cover up the pit, partly or fully, with dirt. It's fine if the dirt is brittle and cracks, because it's already powder; you aren't demanding any strength of it. If it crumbles, it crumbles onto the other dirt and pot underneath it.

A modern equivalent of this is the salt bath or sand bath used by opticians and labs to heat up materials to a desired temperature, either to partly melt or soften them, or to provoke some reaction.

It occurred to me that you can do this with heating elements in the salt or sand, thus achieving a high-temperature capability without constructing a castable refractory with any kind of strength. Quartz sand is cheap, and olivine sand isn't that expensive. You can jam temperature sensors into the sand too, and they might be able to be at a significant distance from the thing that's heating up, so they can be at a lower temperature.

Vermiculite, charcoal, ash, cat litter, and plaster of paris may also be useful; they are somewhat refractory substances that thermally insulate better than sand. Even perlite should be useful up to a point, and that point is about 900°.

Higher temperatures may require the use of arc heating rather than solid heating elements, although carborundum heating elements can extend this considerably.

Carborundum itself was discovered in just such a way and is still produced by this process: a sort of arc furnace is set up under a layer of silica sand with carbon electrodes, originally in an iron crucible, but nowadays at a much larger scale.

To some extent you should be able to measure the temperature distribution within the pile of fluff with temperature sensors that are not exposed to its hottest part. You can estimate the conductivity, thermal mass, thermal resistance to ambient, and heat input using a few sensors at known or estimated locations, a continuous measurement of the thermal input, and some PDE solvers. This could potentially permit very precise control of the temperature distribution within the pile.

Topics

Machine-readable PNG circuit diagram watermarks

Kragen Javier Sitaker, 02020-11-06 (1 minute)

Fractint stored the fractal parameters in its GIF89a output files, so that if you or someone else loaded them into Fractint later, you could recalculate at a higher resolution, or explore other parts of the fractal, or vary parameters.

Screenshots or PNG files are a common way to share circuit diagrams from circuit simulation programs. What if machine-readable circuit topology information was included in these files? You could take the Fractint approach and just include a special parameter block, but that will be lost if someone, say, crops the image, or uses a screenshot program to get the image, or transcodes it to JPEG. What if you sort of barcode or watermark it in the image itself?

The first circuit given in Level Shifter (p. 411) has 6 components; in the encoding used by Falstad's circuit simulator, it occupies 224 bytes, or 133 bytes gzipped, about 22 bytes per component. I think it's probably possible to beat this by about a factor of 3, about 7 bytes per component, 56 bits per component. Given that each component occupies about 500 pixels it seems like it should be pretty feasible to do this.

Topics

- Electronics (p. 792) (42 notes)
- File formats (p. 827) (7 notes)
- Falstad's circuit simulator (p. 828) (7 notes)
- Communication (p. 830) (7 notes)
- Nostalgia (p. 834) (6 notes)
- Coding (p. 869) (4 notes)
- Steganography

Arduino support for STM32

Kragen Javier Sitaker, 02020-11-06 (10 minutes)

I'm trying to get my Blue Pill board to run Blink, the hello-world of embedded development.

https://www.instructables.com/Getting-Started-With-Stm32-Using-Arduino-IDE/ says:

- Launch Arduino.cc IDE. Click on "File" menu and then "Preferences".
  The "Preferences" dialog will open, then add the following link to the "Additional Boards Managers URLs" field:
  "http://dan.drown.org/stm32duino/package_STM32duino_index.json"

But I thought maybe this might be outdated and maybe it's better to use ST's official package. Boy, was I ever fucking wrong. It's a fucking trojan horse.

https://github.com/stm32duino/wiki/wiki/Getting-Started says:

• Launch Arduino.cc IDE. Click on "File" menu and then "Preferences".
The "Preferences" dialog will open, then add the following link to the "Additional Boards Managers URLs" field:

https://github.com/stm32duino/BoardManagerFiles/raw/master/STM32/package_stm_index.json

This installs, by default, version 1.9.0, which occupies tens of megs, takes fucking forever, and then gives an error about STM32CubeProgrammer. What the fuck is STM32CubeProgrammer?

https://github.com/stm32duino/wiki/wiki/Upload-methods says

STLink

Deprecated since core version > 1.5.0 replaced by STM32CubeProgrammer (SWD)

Requires a ST-Link/V2 device connected to the PC over USB and to the board via the SWD interface.

(Actually, it's not deprecated, it's fucking missing.)

...

STM32CubeProgrammer

Since core version > 1.5.0

...

Requirement

To use those upload methods, STM32CubeProgrammer have to be installed manually as it is not provided through the tools packages.

... In any case, if the STM32CubeProgrammer binary is not found, user will be warned like this:

```
STM32_Programmer.sh/STM32_Programmer_CLI.exe not found.

Please install it or add '<STM32CubeProgrammer path>/bin' to your PATH environment:`

https://www.st.com/en/development-tools/stm32cubeprog.html`

Aborting!
```

Okay, fuck, I guess I have another fucking hoop to jump through. What's this STM32CubeProgrammer thing? Some kind of open-source firmware uploader?

No, that couldn't be more wrong. https://www.st.com/en/development-tools/stm32cubeprog.html is pants-shittingly menacing:

[Please choose a sub-application] An end application is required.

Nature of Business:

[____ ___] A nature of business is required.

Military Related:

[__] A military relation status is required.

Software Country/Region of Use:

[_____ _] A country of use is required.
Please keep me informed about future updates for this product.
[ ]

Comment:

[ ] I accept all Terms & Conditions of the Export Control regulations Accept Terms & Conditions
Confirm Request Cancel

...

Request for software successfully submitted. The approval process may take up to 48 hours. After you have
been approved, you should receive a link to the requested software via email.

...

If you don't want to login now, you can download the software by simply providing your name and e-mail address in the form below and validating it.

...

For security / validation purposes, all software download requests must originate from a valid email address.

BY INSTALLING COPYING, DOWNLOADING, ACCESSING OR OTHERWISE USING THIS SOFTWARE PACKAGE OR ANY PART THEREOF (AND THE RELATED DOCUMENTATION) FROM STMICROELECTRONICS INTERNATIONAL N.V, SWISS BRANCH AND/OR ITS AFFILIATED COMPANIES (STMICROELECTRONICS), THE RECIPIENT, ON BEHALF OF HIMSELF OR HERSELF, OR ON BEHALF OF ANY ENTITY BY WHICH SUCH RECIPIENT IS EMPLOYED AND/OR ENGAGED AGREES TO BE BOUND BY THIS SOFTWARE PACKAGE LICENSE AGREEMENT.

\4. This software package or any part thereof, including modifications and/or

derivative works of this software package, must be used and execute solely and exclusively on or in combination with a microcontroller or a microprocessor devices manufactured by or for STMicroelectronics.

\9. The software package is and will remain the exclusive property of STMicroelectronics and its licensors. The recipient will not take any action that jeopardizes STMicroelectronics and its licensors' proprietary rights or acquire any rights in the software package, except the limited rights specified hereunder.

\10. The recipient shall comply with all applicable laws and regulations affecting the use of the software package or any part thereof including any applicable export control law or regulation.

\11. Redistribution and use of this software package partially or any part thereof other than as permitted under this license is void and will automatically terminate your rights under this license.

Regardless of whether you'd have any ethical reason to obey this, or whether it could actually be enforced in court or not, I have a CKS32, not an STM32, and the above is a crystal-clear threat from ST: buy our hardware or else. Ew ew ew ew ew.

I'm installing version 1.5.0 now, which is still literally 82 fucking megabytes. And still takes fucking forever. Like on the order of half a fucking hour.

This done, "STLink" appears as an option for "Upload method" under the Arduino 1.8.14 "Tools" menu. However, the "Port" submenu is grayed out, and evidently it can't find the "STLink". On replugging it, I see in `dmesg`:

```
[802528.168367] usb 2-1: USB disconnect, device number 6
[802531.948113] usb 2-1: new full-speed USB device number 7 using uhci_hcd
[802532.117227] usb 2-1: New USB device found, idVendor=0483, idProduct=3748
[802532.117248] usb 2-1: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[802532.117261] usb 2-1: Product: STM32 STLink
[802532.117273] usb 2-1: Manufacturer: STMicroelectronics
[802532.117285] usb 2-1: SerialNumber: L/28S5KN
```

And I have a new file in /dev/char:

```
lrwxrwxrwx  1 root root   18 Nov  6 19:54 189:134 -> ../bus/usb/002/007
```

I can read some data from it, so it's probably not a permissions problem:

```
$ < /dev/char/189:134 xxd
00000000: 1201 0002 0000 0040 8304 4837 0001 0102  .......@..H7....
00000010: 0301 0902 2700 0101 0080 3209 0400 0003  ....'.....2.....
00000020: ffff ff04 0705 8102 4000 0007 0502 0240  ........@......@
00000030: 0000 0705 8302 4000 00                   ......@..
```

What that means is anybody's guess. The device disappears if I unplug the "STLink".

In theory, according to https://github.com/rogerclarkmelbourne/Arduino_STM32/wiki/Programming-an-STM32F103XXX-with-a-generic-%22ST-Link-V20

%22-programmer-from-Linux, OpenOCD should work:

```
$ openocd -f /usr/share/openocd/scripts/interface/stlink-v2.cfg \
          -f /usr/share/openocd/scripts/target/stm32f1x.cfg
Open On-Chip Debugger 0.9.0 (2018-01-24-01:07)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html

Info : auto-selecting first available session transport "hla_swd". To override use 'transport select <transport>'.

Info : The selected transport took over low-level target control. The results might differ compared to plain JTAG/SWD
adapter speed: 1000 kHz
adapter_nsrst_delay: 100
none separate
Info : Unable to match requested speed 1000 kHz, using 950 kHz
Info : Unable to match requested speed 1000 kHz, using 950 kHz
Info : clock speed 950 kHz
Error: libusb_open() failed with LIBUSB_ERROR_ACCESS
Error: open failed
in procedure 'init'
in procedure 'ocd_bouncer'



default@default-Aspire-one:~/Downloads/arduino-nightly$ sudo openocd -f /usr/share/openocd/scripts/interface/stlink-v2.cfg -f /usr/share/openocd/scripts/target/stm32f1x.cfg
[sudo] password for default:
Open On-Chip Debugger 0.9.0 (2018-01-24-01:07)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html

Info : auto-selecting first available session transport "hla_swd". To override use 'transport select <transport>'.

Info : The selected transport took over low-level target control. The results might differ compared to plain JTAG/SWD
adapter speed: 1000 kHz
adapter_nsrst_delay: 100
none separate
Info : Unable to match requested speed 1000 kHz, using 950 kHz
Info : Unable to match requested speed 1000 kHz, using 950 kHz
Info : clock speed 950 kHz
Info : STLINK v2 JTAG v29 API v2 SWIM v7 VID 0x0483 PID 0x3748
Info : using stlink api v2
Info : Target voltage: 3.139057
Warn : UNEXPECTED idcode: 0x2ba01477
Error: expected 1 of 1: 0x1ba01477
in procedure 'init'
in procedure 'ocd_bouncer'
```

That's wonderful! That's precisely the error I've seen other people report with the CKS32 devices. So I made the relevant config change:

```
$ diff -u /usr/share/openocd/scripts/target/stm32f1x.cfg ~/devel/dev3/cks32f1x.cfg

--- /usr/share/openocd/scripts/target/stm32f1x.cfg  2018-01-23 22:08:20.000000000 -0300
+++ /home/default/devel/dev3/cks32f1x.cfg   2020-11-06 20:46:58.606514993 -0300
@@ -31,7 +31,7 @@
       set _CPUTAPID 0x3ba00477
    } {
       # this is the SW-DP tap id not the jtag tap id
-      set _CPUTAPID 0x1ba01477
+      set _CPUTAPID 0x2ba01477
    }
 }
```

And then OpenOCD apparently works; anyway it doesn't crash immediately like before and the LED on the "STLink" turns from orange (?) to blue:

```
$ sudo openocd -f /usr/share/openocd/scripts/interface/stlink-v2.cfg
               -f ~/devel/dev3/cks32f1x.cfg
Open On-Chip Debugger 0.9.0 (2018-01-24-01:07)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html

Info : auto-selecting first available session transport "hla_swd". To override use 'transport select <transport>'.

Info : The selected transport took over low-level target control. The results might differ compared to plain JTAG/SWD
adapter speed: 1000 kHz
adapter_nsrst_delay: 100
none separate
Info : Unable to match requested speed 1000 kHz, using 950 kHz
Info : Unable to match requested speed 1000 kHz, using 950 kHz
Info : clock speed 950 kHz
Info : STLINK v2 JTAG v29 API v2 SWIM v7 VID 0x0483 PID 0x3748
Info : using stlink api v2
Info : Target voltage: 3.135959
Info : stm32f1x.cpu: hardware has 6 breakpoints, 4 watchpoints
^C
```

So OpenOCD is successfully connecting! But I'm still some distance away from burning the "blink" sketch onto the hardware.

The access-denied problem that led me to try running it with sudo straces as follows:

```
open("/dev/bus/usb/002/008", O_RDWR)    = -1 EACCES (Permission denied)

write(2, "Error: libusb_open() failed with"..., 53Error: libusb_open() failed wit
oh LIBUSB_ERROR_ACCESS
```

And indeed I don't have permission to write to that device, only read:

```
$ ls -l /dev/bus/usb/002/008
crw-rw-r-- 1 root plugdev 189, 135 Nov  6 20:49 /dev/bus/usb/002/008
```

However, I *am* in /etc/group as belonging to plugdev, I guess I just haven't logged out and back in since then. So I launch the arduino IDE in the plugdev group:

```
arduno-nightly$ sg plugdev ./arduino
```

However, that doesn't help; the "Port" submenu of "Tools" is still grayed out. And `sudo ./arduino` of course doesn't have the STM32 board package installed. I copied everything in my ~/.arduino15 to root's, and then the board package is installed, but it has the same problem. I tried `chmod 666 /dev/bus/usb/002/008`, and then OpenOCD works without `sudo`, but the Arduino IDE still has a greyed-out "Port" menu.

Next steps, I guess, are to try Arduino with the Dan Drown stm32duino stuff (or maybe rogerclarkemelbourne's zip file (actual file)?); or to try to program it with OpenOCD; or to use OpenOCD to install some kind of USB bootloader on it.

But I think that's about all for tonight; I've been trying things on and off for 9 hours. I'll see what I can manage tomorrow. The fact that now OpenOCD works means that success is in sight.

Topics

Swashplate screwdriver

Kragen Javier Sitaker, 02020-11-06 (1 minute)

If you have a knob with two ball bearings in its center with a shaft running through them, and that shaft is connected via a hinge to another shaft which is held in some position but not free to rotate, then you can rotate the second shaft by changing the plane in which the knob is without changing the angle at which the hinge is bent. This is easy to do with your hand; the movement is the same as that used to spin a Powerball gyroscope. The knob functions as a swashplate through which you can turn the second shaft with an adjustable mechanical advantage, which can be arbitrarily small when the angle between the shafts is arbitrarily small, and can go up to the diameter ratio between the second shaft and the knob.

This amounts to a new kind of screwdriver, which allows you to unscrew screws arbitrarily fast without a motor. It does need some way of being held in the screw.

I thought this might work especially well using a ball-end hex wrench, with the second shaft being the screw itself, but I don't think that works.

Topics

Thermal expansion speaker

Kragen Javier Sitaker, 02020-11-06 (1 minute)

Suppose that instead of stringing string between two cups to make a telephone, you string the thin copper wire I talked about in Copper Segelín (p. 538) and produce the sound by heating the wire with joule heating by passing varying amounts of current through it, thus making the wire expand. How much current do you need, and what does the frequency response look like?

Well, you probably need to stick to a few hundred mA to avoid melting the thin wire, as calculated there.

Topics

Copper segelín

Kragen Javier Sitaker, 02020-11-06 (updated 02020-11-08)
(19 minutes)

There's discarded styrofoam on the streets every day, so I want a hot-wire foam cutter, for which the best wires are stainless steel. But the fine wires I have handy here are not stainless steel: they are real steel and copper. Also I have nichrome wires, but they are not fine. Which is okay, neither am I, but it's not ideal.

These copper wires are about 100µm in diameter (38 AWG) and are strands of wire from a stranded power cable. I've calculated that this should be about 2.2Ω/m, and simple measurements seem to confirm that this is in the ballpark. Copper oxidizes in air, but only quite slowly below about 300°, and isotactic polystyrene melts at only 240°. And molten foam charring and then burning on the surface of the wire ought to reduce the copper oxide thus formed back to copper, rather than oxidizing it.

I can probably use a high-frequency buck converter to efficiently produce a high enough current through these wires to heat them into the appropriate temperature range, and I could probably even servo it with the resistance of the wire; see Thermistors (p. 431).

So how many amps do I need to reach, say, 270°? That depends on the black-body emission of the wire, which must balance the joule heating from its resistance at that temperature. 2.2Ω/m is also 2.2 W/m/A², which means that, say, at 100 amps, it would be dissipating 22000 W/m, but at 10 amps, only 220 W/m.

Copper itself doesn't have super high emissivity, but the charred foam or black copper oxide that will undoubtedly cover it in use has virtually 100% emissivity, so at 270° it will emit about 4900 W/m². π 100 µm is of course π 100 µm²/µm, or 0.314 mm²/mm. 4900 W/m² is about 1.5 mW/0.314 mm², so that works out to about 1.5 mW/mm of length, or 1.5 W/m.

So if you have, say, 100 mm of this wire, you need about 150 mW to maintain it at 270° in the face of radiation (a fact independent of the wire material except for its emissivity, as well as the source of the heat), and maybe a few times that when foam is cooling it down. As mentioned above, this is 1.5 W/m (a fact additionally independent of its length), and so for copper we need about 830 mA, which is again independent of the length. The only difficulty is that, for 100 mm, our resistance is only 220 mΩ, so our 830 mA must be delivered at about 180 mV.

If we want to deliver this with a buck converter from a 5-volt power supply such as a USB charger or battery pack — and 150 mW or even 450 mW is a totally reasonable power level for that — we need about a 3.6% duty cycle on the pass transistor. That is, 96.4% of the time, the freewheel diode or equivalent will be supplying the (average) 830 mA through the inductor, and the transistor will only be supplying it 3.6% of the time.

But wait! The resistivity I was using to compute 2.2Ω/m is the *room-temperature* resistivity of copper. As noted in Thermistors (p.

431), copper's room-temperature coefficient of resistivity α is about .0039Ω/°Ω, and its resistance is quite linear with temperature up to over 300°, so actually at 270° we should expect the resistance to be 4.4Ω/m. This drops our current requirement to 580 mA but actually increases the voltage requirement to almost 260 mV. This permits the use of a less extreme duty cycle, about 5.2%.

How could you use the wire's resistance to control the buck converter? When its resistance is low, like 220 mΩ instead of 440 mΩ, you want to feed it more current so it heats up; and if its resistance gets higher, you want to feed it less. A lot less, in fact, if it gets more than a little bit higher, since at 300° the copper won't last long.

There are other desiderata, though. You never want to use more than 2.5 W, both to stay within the original USB spec and because that's an absurd amount of power to dump into 7 mg of copper; it should get you up to temperature in less than a second and allow you several milliseconds to respond to stop overshoot. And you never want to dump *much* more than the steady-state 150 mW in there once the wire is decently hot, because it might just be locally cooled by water or something, and you might overheat the rest of it. And you never want to try to push the voltage *or current* too high, because the output might have gotten open-circuited (because the wire broke) or short-circuited. This suggests that you might want to shift into discontinuous conduction mode for shutdown cases, so that both the current limit *and* voltage limit can be low, and have some parallel bleeder resistor that will drop less than 100 mW at whatever max voltage is acceptable, thus keeping the buck's output from soaring. For example, a 10Ω 0805 resistor would pass 70 mA at 700 mV, thus 49 mW.

270° is 1.55 W/m; 280° is 1.67 W/m; 290° is 1.79 W/m; 300° is 1.92 W/m. So as long as we don't have too much insulation around the wire (of something other than foam!) we have about a 24% power cushion, where if we're sending in too much current we just get a too-high temperature instead of actually melting the wire. This amounts to about an extra 11% of current. In theory it might be possible to detect "abnormal heat dissipation" resulting from, say, cold spots, and that might work better than hoping the cold spot isn't more than 24% of the wire.

This level of complexity makes me think using a microcontroller might be easier than trying to design an analog circuit that does everything automatically with separate components.

2.5 W with 220 mΩ is 3.37 A or 740 mV, so the maximum current and voltage definitely shouldn't be higher than that.

Bang-bang

The simplest possible control scheme for this, other than just relying on the wire's power dissipation, would probably be bang-bang control: heat the wire at epsilon over the desired voltage level, say 280 mV instead of 260 mV, until the resistance increases to the expected level of 440 mΩ. Then cut off the supply for a while.

It would be ideal if you could just leave it turned off until the resistance has dropped by some amount of hysteresis, but you can't do

that because you don't know what the resistance is when the power supply is turned off, so maybe it would be reasonable to use a period of time that's not so long that the wire has cooled down too much. At 150 mW and 7 mg and copper's room-temperature heat capacity of 24.440 J/mol/K and 63.456 g/mol, thus 385 mJ/g/K, thus 2.7 mJ/K, thus 18 ms/K, it seems like 10 ms would be a reasonable time to turn off for.

Incidentally, this works out to 56 K/s, which means that even without using multiple watts at startup, you only have to wait about 5 seconds before it's up to temperature — less, really, since using a constant voltage means the lower 2.2Ω cold resistance will give you 600 mW initially.

Series sense resistors are bad

To measure the resistance, one way would be to put a sense resistor in series with the cutting wire, and you probably want it to have a resistance that's small compared to the cutting wire itself so it doesn't get too hot — say, a few milliohms instead of a few hundred milliohms. You probably can't really do this by adding a voltage probe to the cutting wire itself, because your probe section will heat up too, and the voltage you read won't depend on the wire's temperature, just where the probe is and what the total voltage is. If ±10° is an acceptable error, then the cutoff resistance can range from 426 mΩ to 443 mΩ, about a 4% error (±2%), some of which can come from the sense resistor, say about 2% (±1%). Maybe 5 mΩ ±1% would be suitable.

Probably the way to deal with this precision requirement is to make a current-measurement resistor out of the same wire as the cutting wire, but using about 50 strands of it, cut to, say, 113.6 mm ±1%, to get 5 mΩ. Or you could build or buy a milliohmmeter before you try to build the foam cutter.

The other problem with the series-sense-resistor approach, though, is that you also have to measure its voltage to within ±2%, or better ±1% to avoid spending the entire error budget on voltage measurement and having none left over for, I don't know, variation of ambient temperature. But its voltage is 580 mA times 5 mΩ, or 2.9 mV, and 1% of that is 29 µV. That's not a lot of noise immunity for a switching power supply running on 5 volts.

Instead, measure the current with the bleeder

Above I said that you need a bleeder to keep the output from soaring when the wire breaks. If we make the dubious assumption that the input voltage and buck duty cycle are precisely constant, then in CCM the average output voltage will also be precisely constant, but the current will vary according to the load. So, if we suddenly open-circuit the output with a second transistor, the output inductor will suddenly be feeding only the bleeder, but its current can't change instantaneously, so the voltage there will suddenly jump enormously, potentially even higher than the 5-volt input. This new peak voltage will immediately start to fall, but the peak should fairly precisely be the output current multiplied by the bleeder.

And if the peak is too low, then the cutting wire has too much resistance, so we should leave it turned off for 10 ms to maybe cool

down.

For this to give us a precision measurement, the bleeder needs to stay at a constant temperature, and so can't dissipate much heat during normal operation. Let's spitball 1 mW. At 280 mV, this would suggest using a 78.4Ω resistor. If that 280 mV was previously feeding the 220-mΩ cutting wire, it will have been producing 1.27 A (!), and if this resistor suddenly finds itself carrying 1.27 A, it's going to spike up to a somewhat dismaying 99.6 V. If the wire is at temperature, and thus 434 mΩ, then it's only 645 mA, and we see only 50.6 V. So when the inductive spike falls below 50.6 V then we need to turn off the wire for a while, maybe by just leaving the spike-generation transistor turned off. Well, at least we don't have to worry about microvolt noise anymore!

At these voltages, a peak detector consisting of a diode and a capacitor will be pretty precise; the variation in the diode's forward drop is going to introduce maybe 0.3% error at 50.6 V, and the capacitor's capacitance doesn't matter for the voltage peak as long as it's not so large that the inductor's current droops significantly (more than 1% I guess?) before the capacitor is charged.

Then we can just divide down the capacitor's sample of the peak voltage to some kind of reasonable level to see if it's so low that we should leave the output turned off for a while. Or, from a different perspective, to see if it's high enough to turn the cutting wire back on. And the divider network, though it also needs to be precise to sub-percent levels, can also double as a bleeder resistor to drain the peak-detector capacitor so that the next peak actually gets detected.

This suggests also using the peak-detector draining to time the time until the next sample — that is, when the capacitor has drained far enough, we turn off the output spike-generation transistor, just as we would if the capacitor hadn't charged far enough in the first place (because the current was too low, because the output resistance was too high, because the wire was too hot). We probably need some kind of Schmitt-trigger action to make sure the transistor turns off quickly instead of gradually; I gotta think about how to do the comparison of the peak-detector output to the reference voltage by using something simpler than a differential pair.

We can, of course, vary the bleeder/measurement resistor to any similar convenient value; we just have to vary the peak-detector divider factor proportionally.

The 78.4Ω example value results in 3.6 mA of bleeder current at the normal 280 mV, which seems like it ought to be enough to keep the output from soaring. I guess I don't really know how to calculate that, but it's a pretty non-negligible amount of current.

Or maybe just use the wire voltage for feedback

On ##electronics Famine suggested instead feeding the wire with a constant-current supply and servoing off the wire's voltage; they gave an example linear circuit that drives an output Darlington off an op-amp which compares to a reference voltage from a voltage divider. This still means you need measurement precision of a couple of

millivolts, but in a linear circuit like the one they designed, that's totally reasonable, especially if it's running off a battery or something instead of a USB power bank; and it avoids having hundreds of volts anywhere in the circuit. The only real drawback I see is that it can't step up the current from what the power supply can provide, the way a buck converter can, and at any reasonable input voltage it burns most of the power in the output Darlington.

Here's Famine's circuit in the format of Falstad's circuit simulator:

```
$ 1 0.000005 0.529449005047003 50 5 43
R -960 -160 -960 -192 0 0 40 10 0 0 0.5
r -960 -48 -960 -96 0 1000
t -960 -48 -912 -48 0 -1 1.271599532576758 -0.586456396571938 100
r -960 0 -960 48 0 84
r -912 -32 -912 48 0 1000
t -912 -32 -864 -32 0 1 -1.858055929148696 0.6422963295577914 100
w -912 -96 -912 -64 0
w -864 -48 -864 -96 0
w -864 -96 -912 -96 0
w -864 -16 -864 48 0
w -864 48 -912 48 0
w -912 48 -960 48 0
g -912 48 -912 80 0
r -960 -96 -960 -160 0 1000
34 default-led1 0 9.32e-11 0.042 4.6 0
162 -960 -48 -960 0 2 default-led1 0 1 0 0.01
w -864 -96 -832 -96 0
w -960 -96 -912 -96 0
r -832 -16 -832 -96 0 20000
r -832 48 -832 -16 0 2200
w -864 48 -832 48 0
207 -832 -16 -800 -16 4 WREF
207 -832 -96 -800 -96 4 REF
207 -832 -128 -864 -128 4 VIN\p
207 -320 0 -272 0 4 VIN-
t -400 -128 -400 -160 1 1 -8.961181213242781 0.7739571557760819 100
R -416 -160 -416 -192 0 0 40 10 0 0 0.5
w -384 -160 -320 -160 0
r -320 0 -320 48 0 0.265
g -320 48 -320 64 0
207 -832 -128 -800 -128 4 WREF
207 -416 -48 -416 -16 4 OUT
w -544 -128 -544 -160 0
w -576 -160 -544 -160 0
w -720 -80 -720 -112 0
w -592 -80 -592 -48 0
w -720 -80 -592 -80 0
r -720 -160 -720 -112 0 10000
w -720 -48 -720 -80 0
w -576 -112 -576 0 0
g -544 -96 -544 -64 0
r -576 -160 -720 -160 0 3000
t -576 -112 -544 -112 0 1 -1.0940530012757717 0.6209041892124895 100
g -592 48 -592 64 0
g -720 48 -720 64 0
```

```
r -576 -112 -720 -112 0 2200
w -720 0 -576 0 0
w -592 16 -592 -16 0
w -720 0 -720 -16 0
w -720 16 -720 0 0
r -592 16 -592 48 0 10000
r -720 16 -720 48 0 1000
207 -624 -32 -640 -32 4 VIN-
t -624 -32 -592 -32 0 -1 -0.550781216679921 -0.574071418383921 100
t -688 -32 -720 -32 0 -1 -0.3620094258350334 -0.5800382859876023 100
207 -544 -160 -496 -160 4 OUT
207 -688 -32 -672 -32 4 VIN\p
R -720 -160 -720 -192 0 0 40 10 0 0 0.5
t -416 -96 -416 -128 1 1 -8.306594149635059 0.6545870636077218 100
w -432 -128 -432 -160 0
w -432 -160 -416 -160 0
r -416 -96 -416 -48 0 220
370 -320 -160 -320 0 1 0 0
```

Alternative energy sources

Using only 150 mW means you could use even a CR2032 coin cell; Energizer suggests theirs has under 10Ω of internal resistance for much of its lifetime, and under 20Ω until it's almost dead. 3 volts at 20Ω is 150 mA, which is half a watt. The battery won't last long at that kind of drain. Its typical capacity is given as "235 mAh", or in SI units 846 coulombs, or about 2.5 kJ, but you'll be lucky to get a tenth of that at these high drains. So the battery might only last a few minutes to half an hour or so.

Various kinds of capacitors can hold a few hundred joules as well XXX

Topics

Alien screws

Kragen Javier Sitaker, 02020-11-06 (updated 02020-11-11)
(4 minutes)

How would an advanced alien civilization design screws or similar fasteners?

Four improvements in particular occur to me: bayonet connections, two-quarters threads, hollow threads, and positive screwdriver coupling.

Bayonet connections and two-quarters threads

Bayonet connections like those used for camera lenses, BNC cables, fluorescent light starters, US childproof pill bottles, and some kinds of lightbulbs (the English standard) are far superior to screws in most cases, though they do require a spring.

I understand that some barrels use threads that run around only one quarter of the barrel, leave a slot for one quarter of the barrel, run around another quarter of the barrel, then leave a slot for the final quarter. So, by rotating the barrel to the position where the threads do not engage, you can just insert it translationally, then screw it a quarter-turn to lock it into place. This scheme also allows fractional numbers of starts and variable mechanical advantage, similar to bayonet connections, and of course you can use it with larger numbers of starts as well.

By adding a stop at the end of the quarter-turn rotation, as in bayonet connections, you can prevent the screw from being turned too far, which will weaken the connection.

If there is a bit of waviness to the thread, an energy barrier can be provided which will make the screw resistant to coming out during impact or vibration; it must rotate a significant amount before the energy gradient tends toward screwing out, as it always does in a conventional screw, rather than back in. This undulation must come out of the clearance between the inner and outer threads, so it may be necessary to increase the thread spacing to permit it.

By using different thread pitches at different positions along a tapped hole that joins two or more pieces, you can cause the screw to squeeze them apart or, more promisingly, together, by an amount determined by the *difference* between the thread pitches. Conventional bolted joints depend on the screw head and the nut bearing against the tapped material for this, but if the hole is long, the screw threads can engage more material, thus producing a joint stronger against pullout. This also eliminates the need for a screw head in many applications, allowing the use of a headless screw similar to a grub screw.

Hollow threads around a smooth shank

A conventional screw is only stretched in three areas: any shanks where threads are absent; near its head; and near each place where it

passes through a gap between two parts. This purely local stretching makes for maximal rigidity, and consequently maximal fragility to impacts. If this is not desirable, one approach is to leave a long unthreaded shank between the threads needed to provide enough engagement to prevent pullout, in either the screw or the hole or, ideally, both. A better tradeoff can be provided in many applications by making the threads at each end a hollow cylinder maintained in *compression*, free to slide along an internal unthreaded shank or tension rod which is welded to the thread-covered pipes at each end of the fastener. (So they're only sliding to the extent that either the threads or the rod are deforming.) This may reduce the screw's tensile strength somewhat, because the shank is narrower than it would otherwise be — although the thread roots are no longer available as crack-initiation stress risers. But it increases the impact resistance of the joint, because the screw can extend further before breaking, so more energy is needed to break it.

Positive screwdriver coupling

Positive screwdriver coupling might involve a ball bearing that pops out of the side of a square-drive screwdriver to engage a cavity in the screw head, with a shim that slides into place inside the screwdriver to lock the ball bearing in place. A simpler design uses an L-shaped screwdriver blade half the width of the screwdriver and a shim that slides in behind it to push the bottom of the L into a slot in the screw head.

Topics

The Spungot sentential database for end-user logic programming

Kragen Javier Sitaker, 02020-11-06 (updated 02020-12-31) (27 minutes)

I've written a little bit previously about a sort of pattern-matching Prolog, where instead of dealing with explicitly given relations:

```
father(fred, mary).
parent(X, Y) :- father(X, Y).
parent(X, Y) :- mother(X, Y).
ancestor(X, Y) :- parent(X, Y).
ancestor(X, Y) :- parent(X, Z), ancestor(Z, Y).
```

you deal with strings of text, such as lines in a CSV file or a chat transcript, or sentences:

```
Fred is Mary's father
Mary is John's mother

'Foo' is 'Bar''s father
----
Foo is Bar's parent

'Foo' is 'Bar''s mother
----
Foo is Bar's parent

'Alice' is 'Bill''s parent
----
Alice is Bill's ancestor

'Alice' is 'Carol''s parent
Carol is 'Bill''s ancestor
----
Alice is Bill's ancestor
```

From this we can deduce:

```
Fred is Mary's parent
Fred is Mary's ancestor
Mary is John's parent
Mary is John's ancestor
Fred is John's ancestor
```

In Prolog-style top-down search, this is kind of tricky, since you kind of have to guess which inference rules can match which patterns, but in Datalog bottom-up inference, there's no difficulty; each newly inferred sentence need only be matched against the premises of all the rules to see if it enables additional sentences to be inferred. This won't be nearly as efficient as Prolog, but that's fine. It's probably

efficient enough for many uses just by brute force, and a little indexing on infrequent words should go the rest of the way.

I was thinking about this last night and got really excited. I think it might offer an easily usable system with enough expressive power to be useful.

Basic UI

Interactively or in batch mode, a UI for such a database can add a table of results underneath each set of premises, which are distinguished from ground facts by containing variables. (Above I've marked these with apostrophes, but other syntax might be better, especially for natural languages containing contractions.) In batch mode, it could simply process a text file and produce a file annotated with deductions and query results.

Negation

But you can also add negation, using the standard Datalog stratification approach (done dynamically rather than statically):

```
'Aaron' is indicted
\+ Aaron is guilty
----
Aaron is falsely accused
```

I'm not totally clear on how this works without doing the kind of textual rule analysis that I said above was difficult.

I think this approach might work:

• Initially, put all the purely monotonic rules in stratum 0, and all the nonmonotonic ones in stratum 1.
• Do inferences stratum by stratum.
• Maintain a list of inferred assertions for each rule, and of course the derivation for each inferred fact.
• When a newly inferred fact I requires the retraction of some previously inferred fact II, that means that II was inferred too early. So we retract all the assertions inferred from the rule by which II was inferred — call it R(II); add an inequality constraint putting it in a *strictly greater* stratum than R(I), calling I's rule's stratum S(R(I)): "S(R(II)) > S(R(I))"; and move it to the stratum after R(I). We also need to retract all assertions from other rules that are constrained to be in S(R(II)) or later and move them as well.
• When a newly inferred fact I permits the inference of another newly inferred fact II, that means that the rule R(II) by which II was inferred should be at a stratum greater than *or equal to* the stratum of the rule R(I) by which I was inferred. So we add an inequality S(R(II)) ≥ S(R(I)) to a topological sorting database on that basis, if it is not already there. This may involve moving R(II) to the current stratum, may involve moving other rules constrained to be in a greater than or equal stratum to the current stratum, and may require moving other rules to higher strata.
• Retraction chaining from retraction to either assertion or retraction is handled by the shotgun approach of retracting everything from a

rule that is being promoted to a stratum after the current one, because any rule that transitively depended on a rule being thus promoted will also be promoted and thus all its assertions also retracted.
• Upon encountering circular dependencies with a negation in the chain, throw up hands and report them to the user.

This clearly isn't the most efficient mechanism, since we may waste substantial work on a chain of reasoning that must be later retracted when it was discovered to depend on a rule that was applied too early, but I think it might be adequately efficient in practice.

Aggregate formulas

Also you can add aggregate functions:

```
'X' has line item 'Y'
'Y' costs 'Z'
----
X totals =total(Z)
```

where implicitly we are quantifying over all Y (or, I guess, reducing over all Y) because Y does not occur in the rule's conclusion outside of an aggregate.

Aggregates include =count(), =sum(), =total(), =mean(), =stdev(), =max(), =min(), =product(), =list() (which separates items with commas) and =any(), which just picks one of the values in some unspecified way.

=argmax() and =argmin() are aggregates taking *two* arguments: the first is the thing to be returned, while the second is the thing to be maximized or minimized. So, for example, to get the material that provides the lowest cost, you say =argmin(material, cost).

This kind of aggregation involves a form of negation similar to the above. Suppose you want to count the children of each node in a tree:

```
'Child' is a child of 'parent'
---
Parent has =count(child) children
```

We may at some point have deduced that X has 3 children, and then later infer a fourth child of X. The result involves not only adding "X has 4 children" to the database of known facts, but also *retracting* "X has 3 children". So something like stratification is necessary to provide the aggregation feature, at least without a lot of wasted work, possible nondeterminism (where the result depends on what order the rules were applied in), and possible nontermination. (See the section below about dynamical systems, though, for other sources of nontermination.)

Scalar formulas

And of course you can have other formulas as well:

```
'C' is a cylinder
C has radius 'r'
```

```
C has height 'h'
---
C has volume =(pi r² h)
C has surface area =(2 pi r² + 2 pi r h)

'Something' is made of 'unobtainium'
Unobtainium has density 'D'
Something has volume 'v'
----
Something has mass =(D*v)
```

Here because `D` and `v` occur outside of an aggregate function you are not aggregating over all the densities and volumes. If it happens that some object has two volumes and is made of two materials, each of which have two densities, then the system will deduce eight masses for it.

Abbreviation

It's probably better to abbreviate this:

```
'C':
    is a cylinder
    has:
        radius 'r'
        height 'h'
---
C has:
    volume =(pi r² h)
    surface area =(2 pi r² + 2 pi r h)

'It':
    is made of 'unobtainium'
    has volume 'v'
Unobtainium has density 'D'
----
It has mass =(D*v)
```

Goal seek

In this form the system is sort of unidirectional; it can infer the volume of a cylinder from its radius and height, but it can't infer its radius from its volume and height. Spreadsheets use a special "goal seek" interaction for this; you identify which cells are "design variables" the optimizer can twiddle and which cell you want to give a given value, and it twiddles the design variables to approximate it as closely as possible. This could be supported syntactically, using the same syntactic distinction between variables and constants as in premises:

```
c1:
    is a cylinder
    has height 32cm
*seek*:
    c1 has:
```

```
volume 1 m³
radius 'r'
```

This doesn't give you the whole bidirectional power of constraint solvers, but it's very simple to use and implement, and perfectly adequate for many computations I do in Derctuo.

Libraries

You probably want to be able to import library modules so that you don't have to explain things like cylinders and densities in every database. Probably the best way to do this, in a textual system, is to stick a line in the file saying something like

```
:use circuits geometry shapes
```

But this should probably be at the end of the file.

Existentials

If you say that some object is a cylinder, you probably don't want the system to posit a material from which it is made. But there might be cases where you do want such deductions:

```
'x' is a car
----
'something' is the steering wheel of x
```

Here we have a free variable in the conclusion of the rule, with the meaning that the system is entitled to make up an object to fill that role if nothing else turns up. This involves a sort of negation.

Quantities

Above I've talked about quantities like `32cm` and `1 m³`, which have units and are expressed in Unicode notation. This is very valuable for a lot of the calculations I'm doing. I'm not sure if you can implement that within the system or what.

You know what else would be very valuable? Intervals. `32cm±5cm`. `1-1.5m³`. And gradients: when a value is computed by a formula from some given data, it would be useful to see what its gradient is in terms of those givens. Computing the gradient is of course also very useful for "goal seek".

UI affordances

Tabular output can go beyond the simple column-per-variable default; you can, for example, specify a sort key, change the order of columns, change the formatting of columns, pivot one or more variables to be the column headers for the others, hide columns, etc. In an interactive system, you could add rows to the table as a form of data entry.

A non-interactive system can be implemented that just reads in a text file and spews out an augmented version of it.

It's probably useful to see all the inferred facts, as well as which

given facts and rules were used to infer each inferred fact. In rules with a single conclusion, there's a one-to-one correpondence between table rows (*pace* pivoting) and inferred facts, but if there are multiple conclusions there may be more than one.

Filtering this list of inferences down to a usable list might be a challenge. Interactively, too, we might want to know why a given conclusion was *not* reached from a given rule: which of the premises failed to hold true? This kind of "why *not*" debugging is usually easy in functional programs but very difficult in imperative programs; it seems like it would be pretty difficult to incorporate non-interactively in a "program listing", but you could supply it as a separate batch-mode command similar to goal-seek.

Multiple words and nesting

All of the above is entirely without nesting, and the lack of nesting is one of the great UI benefits of logic programming in general. But sometimes you do need nesting in order to be able to correctly reason about complicated propositions, especially without existentials.

A really simple approach, which doesn't go far, is to allow variables to match arbitrary sequences of words instead of single words:

```
Bob Smith is a person
Mary Smith is a person

'Someone' is a person
----
Someone has skin

'John' 'Doe' is a person
'Richard' Doe is a person
---
John Doe is related to Richard Doe
```

From this we can infer, among other things, that Mary Smith has skin and Mary Smith is related to Bob Smith.

The simplest possible approach to nesting would be *not* allowing variables to match arbitrary sequences of words *containing unmatched parentheses*. That way you could use parentheses to supply arbitrary nesting structure.

It might be desirable for a variable to *not* match multiple words by default. This is partly a usability question that ought to be studied by studying users.

Regexps

If you're trying to apply this kind of tool to parsing text that it wasn't intended for, it might be convenient to specify a regex to constrain the matches.

```
start 'year/\d+/'-'month/\d+/'-'day/\d+/'
---
Session began 'day'.'month'.'year'
```

Denesting

The abbreviation facility above suggests writing:

```
I should buy:
    red peppers
    bananas
    apples
```

to create three facts. But what if we're trying to assimilate something like

```
I should buy red peppers, bananas, apples
```

Then maybe it would be useful to be able to write a rule with a premise like

```
I should buy 'food...'
```

to match three times food=red peppers, food=bananas, food=apples, vaguely similar to Scheme `syntax-rules`.

Syntax

Especially for a textual system, syntax is important for UI. Some alternatives to the strawman syntax above:

• Alternative syntax for variables. Above I've only stuck sigils on variables to indicate their variable nature the first time they occur, but it might be worthwhile to use the sigil every time for readability; Tcl and bash seem to suffer in usability compared to PHP and Perl's more universal sigil usage. Here are some possible alternatives:

```
'It' is made of 'unobtainium'   # example above
«It» is made of «unobtainium»   # much harder to type but safer
<It> is made of <unobtainium>   # common metavariable syntax in grammars
`It` is made of `unobtainium`   # e.g., SQL
"It" is made of "unobtainium"   # also SQL but less weird; harder to
                                # type than '' but safer with contractions
It is made of Unobtainium       # Alain Colmerauer's Prolog, without punctuation
IT is made of UNOBTAINIUM       # variant
it IS MADE OF unobtainium       # variant, often used informally for, e.g., SQL
it Is Made Of unobtainium.      # Darius Bacon's Pythological
It$ is made of unobtainium$     # BASIC
$It is made of $unobtainium     # Perl/PHP, taken from BASIC and sh; also Tcl
@It is made of @unobtainium     # Perl variant
.It is made of .unobtainium     # minimal line noise variant
:It is made of :unobtainium     # Logo/Smalltalk/Ruby(?) params, almost as calm
It :is :made :of unobtainium    # Lisp/Ruby keywords/symbols
It 'is 'made 'of unobtainium    # Lisp quoted symbols
,It is made of ,unobtainium     # Lisp quasiquoted
?It is made of ?unobtainium     # Lisp-family logic languages; N3
It? is made of unobtainium?     # variant
¿It? is made of ¿unobtainium?   # Spanish variant
{It} is made of {unobtainium}   # various templating languages
                                # including Python .format and f''
```

```
#{It} is made of #{unobtainium} # Ruby's equivalent
[It] is made of [unobtainium]   # easier to type on standard keyboard than {}
(It) is made of (unobtainium)   # the remaining ASCII nesting delimiters
%It% is made of %unobtainium%   # MS-DOS batch
%It is made of %unobtainium     # variant
¤It is made of ¤unobtainium     # variant
|It| is made of |unobtainium|   # more little-used delimiters
It* is made of unobtainium*     # asterisk connotes reference
It† is made of unobtainium†     # though daggers connote it HARDER
It... is made of unobtainium... # ellipses connote indefiniteness
It_ is made of unobtainium_     # Mathematica
```

In a multi-font system, we could imagine writing *It* is made of *unobtainium*, It is made of unobtainium, **It** is made of **unobtainium**, or It is made of unobtainium instead. (Note that if you're viewing this on GitLab some of the formatting in this paragraph gets mangled by their buggy Markdown parser.)

• Alternative syntax for deduction. The line of dashes echoes the sequent calculus but it's kind of heavyweight, and how many dashes do you use, anyway? Does it matter? And then there's the question of how far its scope extends (above, to the first blank line). And should the premises come before the conclusion, as above, or after it? Here is the original and some strawman alternatives:

```
{Alice} is {Carol}'s parent
{Carol} is {Bill}'s ancestor
----
{Alice} is {Bill}'s ancestor

{Alice} is {Bill}'s ancestor :-
    {Alice} is {Carol}'s parent
    {Carol} is {Bill}'s ancestor

{Alice} is {Bill}'s ancestor?
    {Alice} is {Carol}'s parent
    {Carol} is {Bill}'s ancestor

if:
    {Alice} is {Carol}'s parent
    {Carol} is {Bill}'s ancestor
then:
    {Alice} is {Bill}'s ancestor

{A} is {C}'s parent; {C} is {B}'s ancestor |- {A} is {B}'s ancestor

:A is :C's parent; :C is :B's ancestor { :A is :B's ancestor }

{Alice} is {Carol}'s parent
{Carol} is {Bill}'s ancestor
:. {Alice} is {Bill}'s ancestor

{Alice} is {Carol}'s parent
{Carol} is {Bill}'s ancestor
=> {Alice} is {Bill}'s ancestor
```

Although I like the one with ":.", the closest ASCII equivalent of "∴", I think the last one with "=>", due to deltab, is better. They both avoid spurious visual suggestions of nesting, it's compact, and there's only one way to do (each of) them. The `premises { conclusion }` idea, also due to deltab, is also very nice, but like the turnstile `|-` it clashes somewhat with the overall line-oriented style.

• Alternative syntax for formulas. I think most formulas will probably be fairly simple affairs, so it's nice to be able to introduce them with just a single character instead of nested delimiters; `=total(cost)` beats `[total(cost)]` on visual noise. And the `=` syntax is familiar from Excel, having replaced Visicalc's `@` syntax (also used in Lotus 1-2-3, though with one less period for ranges): `+B1-SUM(C2...C8)`. Still, you could imagine other syntaxes. ES5 template strings use `${2 * a + b}`.

Queries and reporting

Every set of premises is a query whose answer is a table, but it might be more useful to be able to include only some of these tables in a formatted output report.

Inequalities

For a lot of calculations I care a lot about inequalities: the weight is less than the weight capacity, the temperature is less than the melting point, the change in Gibbs free energy is negative, the absolute pressure is positive, and so on. It seems like the best way to incorporate these inequalities into rules is simply as a special sort of premise that is handled specially; rather than being matched against known or inferred facts, it is checked once the relevant variables are instantiated:

```
{The body} is:
    made of {stuff}
    at atmospheric pressure
{Stuff} has:
    melting point {M}
    boiling point {B}
=> {The body} is liquid from {M} to {B}

{The body}:
    is liquid from {M} to {B}
    has temperature {T}
{M} < {T} < {B}
=> {The body} is liquid

{The body}:
    is liquid from {M} to {_}
    has temperature {T}
{T} < {M}
=> {The body} is solid

{The body}:
    is liquid from {_} to {B}
    has temperature {T}
{T} > {B}
```

```
=> {The body} is gaseous

{The body}:
    is liquid from {M} to {_}
    is slushy
=> {The body} has temperature {M}
```

For point-valued scalar real quantities, these inequalities are trichotomous: either {T} < {M}, {T} == {M}, or {T} > {M}. But for interval-valued quantities, this may not be the case; if the temperature of some water is known to be between -4° and +4°, we cannot conclude either that it is liquid or solid. (Or gaseous, of course; a more powerful modal reasoning system than what I'm proposing could demonstrate that it is not gaseous, and that if the temperature is >0°, it is liquid.)

Equalities can not only be *checked* in this way, but if we permit formulas in premises, they can also *instantiate variables*, which is potentially useful as a way of factoring out formulas. However, this also has potentially complex interactions with interval-valued variables, since knowing that two masses are both in the range (100 g, 200 g) does not demonstrate that they are equal. (And of course this already pops up with the equality testing implicit in multiple occurrences.)

Frames and modal reasoning

All of the above puts both rules and facts in a sort of global tuple space or string space. But the system is clearly capable of expressing logical consequences: if the temperature is 329°, then the wax is liquid. We could imagine "creating a frame" that contains some additional facts (and perhaps omits others), and looking to see what can be newly inferred from those facts.

If you have some way to export computed values from these frames, this very quickly gives you something like Bicicleta, only with logic programming rather than functional formulas.

Arrays and dynamical systems

Computers are great at iteration. You can integrate a system of ordinary differential equations with Euler's method in Python in a minute or two of programming, using a fraction of a second of CPU time:

```
$ time python
...
>>> x0, y0 = 100, 0
>>> x, y = x0, y0
>>> for t in range(1000):
...   x, y = x + .01 * y - .01 * x,  y - .02 *x - .01 * y
...
>>> x, y
(-0.0006995268370926552, -0.006688316539762943)

real    1m11.032s
user    0m0.072s
```

```
sys     0m0.056s
```

Systems like Modelica include robust numerical ODE solvers using much more efficient, higher-precision methods.

But even without getting into dynamical systems, iteration is pretty useful. It turns out that the system as described above can handle this example with no problem:

```
At time {t} of {n} we are at ({x}, {y})
{t} < {n}

=> At time =({t} + 1) of {n} we are at (=({x} + .01 * {y} - .01 * {x}), =({y} - .o
o02 * {x} - .01 * {y}))

At time {n} of {n} we are at ({x}, {y})
=> Our final position is ({x}, {y})

At time 0 of 1000 we are at (100, 0)
```

Given how easy this is, it might be worthwhile to have an implicit loop limit you can override to avoid accidental looping. Like, in a sense this is just transitive closure, right?

This kind of explicit indexing can be used for things that aren't iterative, too, but rather data-parallel:

```
{It} emits {x} watts per square meter per nm in band {λ}
The CIE photopic perceptual weight of band {λ} is {w}
=> {It} emits =mean({x} * {w}) lumens
```

This is easy to distinguish from the dangerous case above because the derivation chain doesn't go through the same rule over and over again. However, I'm not entirely sure how to tell the difference between the (safe) transitive closure of a finite relation produced in some other way, and the (unsafe) looping case above. So it isn't obvious how to implement something like Turner's Total Functional Programming.

There's a practical concern for how to get these arrays of data into the system as a large set of assertions like the example above in the first place. But the solution for that doesn't have to be elegant; it just has to be practical.

Consider this Numerical Python example, where I wanted to see the number of RC time constants needed to decay past a number of candidate thresholds:

```
>>> [round(i, 2) for i in -log((arange(99)+1)/100.0)]
[4.61, 3.91, 3.51, 3.22, 3.0, 2.81, 2.66, 2.53, 2.41, 2.3, 2.21,
2.12, 2.04, 1.97, 1.9, 1.83, 1.77, 1.71, 1.66, 1.61, 1.56, 1.51,
1.47, 1.43, 1.39, 1.35, 1.31, 1.27, 1.24, 1.2, 1.17, 1.14, 1.11,
1.08, 1.05, 1.02, 0.99, 0.97, 0.94, 0.92, 0.89, 0.87, 0.84, 0.82,
0.8, 0.78, 0.76, 0.73, 0.71, 0.69, 0.67, 0.65, 0.63, 0.62, 0.6,
0.58, 0.56, 0.54, 0.53, 0.51, 0.49, 0.48, 0.46, 0.45, 0.43, 0.42,
0.4, 0.39, 0.37, 0.36, 0.34, 0.33, 0.31, 0.3, 0.29, 0.27, 0.26,
0.25, 0.24, 0.22, 0.21, 0.2, 0.19, 0.17, 0.16, 0.15, 0.14, 0.13,
```

```
0.12, 0.11, 0.09, 0.08, 0.07, 0.06, 0.05, 0.04, 0.03, 0.02, 0.01]
```

This is awkward to do in Numpy because Numpy doesn't have output format control, so I had to resort to a regular Python list comprehension.

The Scheme `syntax-rules` macro system includes an interesting "..." construct, permitting you to rewrite, for example (foo (as a...) (bs b...)) to (bar (b a)...), without providing full list-processing capabilities. That example would rewrite (foo (as 1 2 3) (bs x y z)) to (bar (x 1) (y 2) (z 3)), for example. You could imagine supporting a similar but more limited "..." construct in patterns in order to be able to input array data more easily.

Fuck RDF N3 syntax, seriously

Consider this example from EYE, in RDF N3:

```
@prefix log: <http://www.w3.org/2000/10/swap/log#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix : <http://www.agfa.com/w3c/euler/socrates#>.

:Socrates a :Man.
:Man rdfs:subClassOf :Mortal.

{?A rdfs:subClassOf ?B. ?S a ?A} => {?S a ?B}.
```

This looks like a bunch of fucking line noise. I think it's dramatically more understandable to express it as follows; the result, "Socrates is a mortal", is even more understandable than the over a page of line noise generated by EYE.

```
Socrates is a man
Every man is a mortal

Every {X} is {Y}
{Z} is a {X}
=> {Z} is {Y}
```

Also, the *input* is not only more readable, but also 85 characters instead of 317 characters, almost four times smaller.

(The more flexible syntax might be not only more readable, but also permit subtle bugs. Suppose the template above had said `Every {X} is {Y}.`, with a period at the end; then it would unintentionally fail to match.)

N3 is, of course, capable of expressing enormously more powerful forms of inference than that, including anonymous entities and so on. But I don't think that's an excuse for it to read like line noise.

Naming

Names considered: Heef Jumbus, Spungot (file extension: .spug), Facdotum, Axiopolis, Expressum, Shuntence, Nollidge, Ret-o'-Rick, Infoflow, Conceptium, Polythink, Knecksus, Mirrorgation, The

Mind Machine, Databog, Thinksluice, Monad's Revenge, Itshift, Cherry-go-Round, Connectionalismus, Meshotron, Cogtionary, Equationsheet, Mathbox, Logivox, Neotinker, The Bitsmith's Forge, Omnilathe, Spinfluence, Elementalis, Wonderiensis, Bowdos, Monkin, Fleuf, Trurbus, Ploomish, Grufty, Rencum, Dolus, Mujimbo, Stooshiong, Treebus, Widgity, Cleophlembic, Entrisculi, Quimbrus, or Factpool? Thanks to sbp for the cromulentisimo suggestions.

Topics

Rosining chips

Kragen Javier Sitaker, 02020-11-08 (2 minutes)

In Peter Laackmann and Marcus Janke's "Uncaging Microchips" talk, at 30'20" they presented an approach I hadn't heard of for removing epoxy, the most common encapsulant, from a microchip package. By heating ordinary rosin or colophony to its boiling point of 320–360° with a heat gun (which also is not a temperature I knew rosin would withstand without charring) you can dissolve the *cured epoxy package* in under 20', then clean it with acetone at 40°, though I suspect alcohol might work as well, since what must be removed at that point is mostly colophony. Reportedly it smells terrible and leaves the chip unusable because it loses the bond wires.

They also mention that chloroform, dimethyl formamide, and DCM can swell or dissolve epoxy, so that you can "brush it away", as they say, although I would rather not be around any of those; and you can use a CNC milling or grinding machine with micron precision; and you can burn it with a laser, especially a 10-micron infrared laser to which the silicon is transparent.

I suspect you could probably burn off the epoxy with a non-thermal oxygen plasma as well; the epoxy's reaction products with oxygen will be gaseous at room temperature, while the reaction products with the bulk components of the chip — aluminum, silicon, copper, silica, hafnia — either don't exist or are solid. Maybe a non-thermal steam plasma would also work, because although silane is a gas at room temperature, it's not very stable. And of course ionization of air generates oxides of nitrogen, which are of course well known as a way to decapsulate epoxy-encapulsated chips; the talk above says you usually need several grams of them. See the note on cold plasma (p. 560) for more.

The rosining process is pretty interesting to me not only for seeing the chip — for example for reverse engineering — but also for the possibility of converting a packaged chip into a WLCSP, since WLCSPs are usually hard to buy, especially in quantity 1. The chip would need to survive its rosining, but I don't think 360° for 20' is enough to cause substantial dopant diffusion; I think it's just a question of replacing the broken bond wires.

Topics

Cold plasma

Kragen Javier Sitaker, 02020-11-08 (updated 02020-11-24)
(14 minutes)

I was reading about cold plasmas after learning about rosining chips (p. 559). They're already used in many applications (1 ppm ionization is becoming commonly used for sterilizing in medicine, for example) but I was thinking about a few more, including a possible low-temperature mostly electrolytic route to rutile reduction that doesn't have the low cathode efficiency problems that plague traditional molten-salt electrolysis.

Ionized air oxidation

I'm pretty sure a cold plasma of air can eat epoxy, and I suspect that a cold plasma of just oxygen could do so. Ozone is mostly known to attack olefin double bonds, which by itself would not be sufficient to eat epoxy, but I suspect ozone and the other reactive oxygen species in the cold oxygen plasma would be sufficient.

Ionizing air generates oxides of nitrogen, which tend to not only oxidize things (more aggressively even than ozone) but also to nitrate them. I suspect this would be fine for decapping chips; aluminum, silicon, and copper are all relatively inert to such materials. However, at temperatures below 100°, zirconia may be vulnerable, and perhaps thus also hafnia. Typical arcs in electric arc furnaces can reach a few hundred ppm of oxides of nitrogen, and rarely go below a few tens of ppm. This suggests that to produce the ten grams or so needed to decap a chip you'd need to run tens of kg of air through your plasma pencil, most of which of course won't react, so you'd probably need tonnes. This seems slow but perhaps a feasible approach.

(Of course if you have a drop of water on the object, that will facilitate the attack of oxides of nitrogen on many materials, including copper, zirconia, and hafnia; though, perhaps, at first, before the water has absorbed much of the gas, it might have a protective effect instead.)

I've written before about using an air cold plasma with simply a glow discharge for selective functionalization, for example for selective electroless metal plating or selective wetting of otherwise nonreactive surfaces.

A cold plasma of steam would be easier to produce than that of oxygen, and it would have a similar oxidizing effect on some materials. However, I think the hydrogen ions would prevent other materials, such as many metals, from oxidizing. This could be used in some cases for more selective oxidation.

Ionized hydrogen reduction

A potentially more interesting application of cold plasmas is reduction with *just* hydrogen. It's been routine since at least WWII to anneal the iron powder in hydrogen before using it in powder metallurgy, to reduce the oxide film from its surface and enable it to cold-weld and sinter. Similarly experiments have successfully

extracted oxygen from lunar regolith simulant by reducing it with hydrogen. Surely a similar effect can be achieved with a cold hydrogen plasma.

For example, you could pack a powder bed with a powder of an iron oxide, infuse it with hydrogen at above 100° to drive out all the air, insert an iron electrode into it, and apply a high-frequency high voltage to the electrode. This would produce a hydrogen plasma in contact with some of the powder, and some of the iron oxide would be reduced, producing steam, which would diffuse away from the electrode and be replaced by fresh hydrogen to continue the process. The iron electrode would thus gradually grow dendritically through the oxide powder, though not converting all of it. Some flow of hydrogen would be useful to flush out the steam and prevent it from re-oxidizing the iron; alternatively some kind of desiccant such as calcined alabaster could sequester the water, or a more easily oxidized metal could reduce it.

Ellingham diagram by Wikipedia user DerSilberspiegel, CC-BY-SA 4.0

Looking at an Ellingham diagram with hydrogen to steam ratios, it seems that for hydrogen reduction of magnetite to hematite, the equilibrium favors reduction as long as there is less than about ten thousand parts of steam to one part of hydrogen, almost independent of temperature over the usual ranges; and for hydrogen reduction of iron oxide to iron, about 10× as much hydrogen as steam is needed, again almost independent of temperature. Unless I'm reading this diagram wrong.

Copper, cobalt, and nickel seem similarly simple. Other metals are a bit less so; zinc seems to require a bit more hydrogen than water, and that only at 1100°. Getting down to below zinc's melting point requires a million-to-one hydrogen-to-steam ratio, and getting down below 200° requires a trillion to one. At silicon's melting point of 1460° (XXX Pyrolysis 3-D Printing (p. 242) says 1414°, maybe this is the wrong temp) we should be able to reduce it with hydrogen at ten thousand times the concentration of steam, and at a trillion to one this reduces to a balmy 600°. Rutile at its melting point above 2300° can be reduced with hydrogen at about 3000 times the concentration of steam, but at a more comfortable 1000° it's a billion to one, and at a trillion to one we're down to about 600°, a bit higher than silex. (Damned Tuftean plots with no grid lines.) Even at these trillion-to-one levels, sapphire doesn't yield to hydrogen's persuasion until past 800°, magnesia alba until 1000°, and lime until 1200°.

(Maybe in some cases you'd get the hydride of the metal rather than the metal itself.)

Actually, lime is particularly interesting in this connection because, as in the Pidgeon process, it can combine with silex into the stabler larnite. This reaction can work as a sort of desiccant, in effect allowing metallic silicon (or ferrosilicon) to reduce most metals, even very active ones like magnesium, again as in the Pidgeon process. Facilitating this reaction with ionized hydrogen rather than hellish temperatures seems potentially useful.

Ferrosilicon is normally obtained by carbothermal reduction. I

wonder if you could go further, passing a cold plasma of hydrogen contaminated with steam over carbon at some more everyday temperature, to increase its hydrogen-steam ratio before passing it over the metal powder again? You could use the carbon itself as an electrode to ionize the water.

A more quotidian approach to removing water would be to cool the hydrogen–steam mixture, pass it over a garden-variety low-temperature desiccant such as alabaster, muriate of lime, or quicklime, and then heat it up again to the reaction temperature before reionizing it. Carrying out the cooling and heating steps with countercurrent heat exchangers or regenerators would eliminate their unnecessary energy consumption.

In such a case, where does the energy come from? We apparently have hydrogen circulating in a closed loop at constant pressure to reduce an oxide, say silex or rutile, to its base metal, which entails adding energy to it — if we burn the metal we will get the energy back. The desiccants are losing energy by being hydrated, but not nearly enough to reduce the metals. The mystery is solved, though, when we observe that we must continually add hydrogen to the system if we are to prevent its pressure from dropping. The energy to reduce the metals was the chemical potential energy in the hydrogen.

In the case of the desiccants, one is left to wonder what to make the heat exchangers out of if the reaction gas is corrosive at only 800° to even sapphire and tends to oxidize metals. Zirconia, perhaps, or some kind of highly refractory carbide or nitride: BN, TiN, WC, HfC, the usual suspects. Or maybe just aluminum, if it's countercurrent: pipes with enough water vapor will be lined with amorphous sapphire the way aluminum normally is, while pipes without will eventually just be bare aluminum, but neither will corrode. Other metals that form similar passivation coatings serve just as well at higher temperatures that aluminum can't handle.

So, as an example, maybe you can reduce a packed bed of powdered rutile at, say, 800°, in a hydrogen atmosphere at atmospheric pressure, by applying a high-frequency electrical charge through an electrode to ionize the hydrogen, while maintaining the hydrogen very dry (better than 100 billion to one ratio to the water) by passing it through a desiccant such as alabaster at a lower temperature such as room temperature, say 20°, with a countercurrent heat exchanger in between the rutile chamber and the alabaster chamber to maintain them efficiently at different temperatures, plus some additional heating to maintain the rutile at its high temperature and some additional heatsinking to maintain the desiccant at its low temperature, while continually adding new hydrogen to the system to replace the hydrogen absorbed as water in the desiccant. The energy to reduce the rutile comes principally from the hydrogen, whether that is produced by electrolysis or, for example, from natural gas.

Hmm, shit, *that* desiccant probably can't get the hydrogen *that* dry. So you probably need a somewhat higher temperature. Or maybe if you cool the desiccant more, or use a more aggressive desiccant, you can get down to those levels. Obvious candidates include sodium (as in the Hunter process), calcium (as in the Kroll process), ferrosilicon

with lime (as in the Pidgeon process), and of course lithium, magnesium, or aluminum.

Solid magnesium and aluminum have the annoying problem of forming an adherent solid oxide film when oxidized, preventing them from reducing any further water; as outlined in Petrovic and Thomas's 2008 "Reaction of Aluminum with Water to Produce Hydrogen: A Study of Issues Related to the Use of Aluminum for On-Board Vehicular Hydrogen Storage", approaches to solving this problem for aluminum include "hydroxide promoters such as NaOH, oxide promoters such as Al_2O_3, and salt promoters such as NaCl", but I think all of those are necessarily in aqueous solution, and would thus produce too much contaminating steam of their own. Other approaches might include maintaining the desiccant metal *molten*, with a layer of flux salts on top to keep the oxides molten, and bubbling the gas through it to deoxidize it. Magnesium melts at 650°, aluminum at 660°, their eutectic of about 65% aluminum at a pleasant 437°, but as explained above, even at these temperatures, not much oxygen will escape to reoxidize the hydrogen. Alternatively, you could disrupt the oxide layer with plasma, perhaps using the very same hydrogen gas or perhaps using a more conventional sputtering gas like argon, which would of course then be mixed in with the hydrogen.

In this form, you're essentially performing an aluminothermic (or magnesiothermic) reduction of rutile, but with hydrogen acting as a catalyst (instead of, as previously, a fuel.)

Higher pressures would tend to increase the reaction rate, but I don't think they'll affect the equilibrium of the metal reduction much, because both hydrogen and steam are gaseous, with the same number of moles as the resulting water. If you instead used a reducing gas with a different number of hydrogens, such as methane, ammonia, maybe hydrazine if it can stand the heat (silane can't), nitric oxide, or even plain nitrogen, you might be able to use pressure to shift the equilibrium. (Probably by reducing the pressure.) This might allow you to tolerate a larger percentage of water vapor in the system before the metal stopped reducing.

However, higher pressures *do* beneficially affect the relationship between the oxide equilibrium and a desiccant equilibrium! That's because at a certain temperature, a certain percentage of desiccant being hydrated corresponds to a certain partial pressure of steam, and this is unaffected by the partial pressure of hydrogen. So, by increasing the pressure, you proportionally increase the proportion of hydrogen in the desiccated gas.

Zircon, zirconia, coltan, and molybdenite could likely be reduced in the same way.

Flue gas decontamination

If you have an unlimited supply of sufficiently dry hydrogen, the unused hydrogen contaminated with steam could be disposed of by simply passing it through a flame. Ozone and oxides of nitrogen are perhaps not as easy, though natron water should serve to some degree, and automotive catalytic converters are a common solution to precisely this problem. Olefins such as ethylene, propylene, or hexene

would probably also eliminate ozone and oxides of nitrogen, and could then be burned with impunity. 2-methyl-2-butene is used in such a way as a free radical scavenger.

Diesel engines commonly spray urea water ("diesel exhaust fluid" or "AdBlue" or "Azul 32") into the exhaust to eliminate nitrogen oxides instead of using a catalytic converter. This also eliminates ozone. Urea is nontoxic and quite cheap; the solution is sold by the ten-liter bottle. The final resulting gas mix still contains nitrogen dioxide, though.

Topics

Cutting steel with steam

Kragen Javier Sitaker, 02020-11-11 (1 minute)

In my note on cold plasma (p. 560) I wrote about the possibilities of using cold plasma to slowly reduce metal, dissolve away epoxy, and so on. But it's also very useful to be able to *oxidize* metal, especially iron, for example to cut it. And, as I found there, hydrogen can reduce iron from FeO at almost any temperature as long as there's about 10× as much hydrogen as steam. But that *also* means that *steam* can *oxidize* iron to FeO at almost any temperature *until* 90% of the steam has been reduced by the iron.

This suggests the use of steam as a substitute for oxygen in an oxy-acetylene cutting torch.

That's probably a dumb idea because probably you lose more than 79% of the heat of oxidizing the iron to splitting up the water. FeO's melting point is 1377°, its enthalpy of formation is XXX ????, and water's enthalpy of formation is -285 kJ/mol.

Topics

- Materials (p. 788) (51 notes)
- Contrivances (p. 790) (44 notes)
- Manufacturing (p. 799) (17 notes)
- Steel (p. 858) (4 notes)

Improvised humidity sensors with PET dielectric spectroscopy

Kragen Javier Sitaker, 02020-11-11 (3 minutes)

Humidity sensors are useful. Dielectric spectroscopy is easy with modern microcontrollers. PET bottles are widely available. PET is quite hygroscopic, and this alters its permittivity. We can make humidity sensors by dielectric spectroscopy of PET bottles.

Küchler, Farber, and Franck (2020, "Humidity and Temperature Effects on the Dielectric Properties of PET Film") found 7–10% variation in permittivity in 10°C PET films across the 1Hz–1kHz range when relative humidity varied from 0% to 80%; dry PET film had a permittivity magnitude of about $3.18–3.20\varepsilon_0$, while PET film exposed to air at 80% humidity was in the $3.40–3.45\varepsilon_0$ range, depending on frequency. Temperature also influenced the measurement; temperatures from 0°–65° were all about the same, but the permittivity started to soar, especially at lower frequencies, at 85° and above. However, even at the low temperatures where the permittivity magnitude was effectively unchanged, the loss angle varied dramatically with temperature, especially between 10 Hz and 10 kHz.

The frequency at which the peak loss occurred varied even more dramatically with humidity than did the permittivity itself, varying from about 100 mHz at 35% up to 100kHz at 80%; however, the curve they plot seems somewhat irregular, though roughly exponential, and it's not clear whether this is due to measurement imprecision or to it having a complex shape. The detection of such a peak may thus permit a more precise measurement of humidity.

Such a sensor might drift over time, but it should at least be good enough for a rough measurement of humidity and temperature, and it can easily be made from garbage.

How slow would it be? cloudevil on ##electronics pointed out that this might be a problem. The paper actually explains Fick's Law for diffusion and did weight gain measurements. They were using 23-micron boPET and seem to have reached steady state after something like 10'. This bottle I have here is more like 400–450 µm: I cut out a piece, folded it in half four times, and it was about 7 mm thick, giving about 440 µm per layer. Extrapolating quadratically gives a time to steady state of 40 hours. This bottle is only partly biaxially oriented, and probably less crystalline, so permeability might be higher. But it seems like it would be useful to use thinner plastic to get faster response. This wasn't a pressure bottle, but it's a bit thicker than water bottles, and much thicker than chip bags.

This lubricating graphite powder I got at the hardware store (48 g bottle labeled as 60 g) doesn't want to stick to the PET; I'm not sure if abrasive will help or if stronger measures like plasma are necessary. A bit of scrubbing by hand with toothpaste was not sufficient.

Topics

Printf tracebacks

Kragen Javier Sitaker, 02020-11-11 (2 minutes)

I watched a video demo of a new Visual Studio Code plugin for Autodesk Fusion 360 "postprocessor" plugins, which are evidently written in JS. These emit G-Code — I think they might take G-Code as input, but I'm not sure. The video demo showed a feature similar to Bret Victor's famous tree-landscape-drawing demo, in which by clicking on a piece of G-Code in the output, you would immediately jump to the line of source code that emitted it.

This is a super cool feature, and I realized that I really want this for debugging printfs in general: I want to be able to click on a debug log message and get a stack trace of the program as it was at the moment the log message was emitted. Perl's Carp module has provided such a facility in a purely textual form for a long time, and Purify and Valgrind have provided it for memory allocation, but I want to be able to do it for any output, especially debugging output.

Moreover, I especially want to be able to do this for immediate-mode GUIs; I want to be able to jump from a GUI control on the screen into the code that painted it, and see the stack trace as it was at the moment that control was painted. This is actually maybe *easier* to provide than the purely textual version of this feature, because if the feature is still there on the screen when I click on it, and the program is still running, that stack is *actually running* at the moment that my click is delivered.

Delivering this functionality for screenshots and recorded sessions would be harder.

Topics

Random synchronous motor

Kragen Javier Sitaker, 02020-11-11 (2 minutes)

I just ripped apart a microwave, and one of the things I got out of it was a turntable motor. This turns out to be a 5-watt, 5 RPM, synchronous 240VAC gearmotor. This is one of the few occasions where it really doesn't matter which way the motor turns, so in fact they used a synchronous motor that turns in a randomly different direction each time, depending I suppose on where it was when it stopped.

Even if you couldn't sense its position well enough to predict which direction it would turn at startup, you still might be able to use it for motion control with closed-loop feedback, using the following scheme: when you start up the motor, if it's going the wrong way, wait a random fraction of a rotation (of the motor, not the gearbox output shaft), turn it off, and, after enough time for it to stop, turn it on again.

You could take this simple control scheme one more meta level to flagellate bacterium behavior: while the conditions for a machine continue to improve, run the motor continuously; while conditions remain the same or get worse, run the motor intermittently. Connect the motor to a mechanism such that, in one direction of rotation, the machine moves in a straight line in whatever direction it's pointed, but in the other direction of rotation, the machine wanders around randomly. When the motor is running intermittently, it will sometimes go in a straight line, and sometimes wander, but once it happens to be moving in a straight line that improves the situation, it will continue.

Unfortunately, such a mobile machine would probably be battery-powered, and running a synchronous AC motor off DC requires about as much circuitry as the above feedback scheme would take to implement.

Topics

- Ghettobotics (p. 797) (18 notes)
- Motors
- Hardware

Specular photogrammetry

Kragen Javier Sitaker, 02020-11-11 (3 minutes)

I was watching a porn video yesterday, and as the model poured mineral oil all over her body, I was struck by the thought that the specular reflections of the room that were appearing in the oil contained enough information to reconstruct a fairly precise three-dimensional model of the surface of her body, particularly given two simultaneous images from different points of view, or if she were to rotate without deforming.

The forward problem is relatively straightforward: you have a surface, the surface has some Lambertian texture, some Phong exponent, and some percentage specularity, and the surface is in some environment with some lighting and reflecting some scene around it, in front of and occluding part of that same scene; and the surface has some orientation in space that is changing. Given all these parameters, it's straightforward, if somewhat expensive, to do the computation to ray-trace a photorealistic image.

By solving the inverse problem through iterative methods, and in particular methods based on the *difference between* corresponding points on the surface at different rotations, you can *estimate* the surface, the texture, the Phong exponent, the specularity, the scene, the lighting, and the orientation. Generally each part of the scene is reflected in several places on the surface. Most of these parameters are of low dimensionality or effectively so; a small number of spherical harmonics, for example, suffice to approximate Lambertian lighting fairly precisely, and of course the lighting is itself part of the scene. Only the surface geometry, the texture, and the scene are of high dimensionality, and given a few frames of video, they are amply overdetermined.

Spilling some water on my mate and observing the sparkly reflection around the powdered yerba, I am reminded that the Phong specular blurriness exponent is generally taken to be an approximation of surface microfaceting, and one of the major effects of such wetting is to make such microfacets larger, so you can actually see them individually. This allows you to track them from frame to frame, even if the surface's Lambertian texture is too uniform.

If you have two different linearly polarized cameras, you can use Brewster's angle to additionally estimate the refractive index of the surface gloss, and this polarization data gives you an additional measurement of the angle and magnitude of the surface normal, as projected on the focal plane. This should serve to improve surface reconstruction further.

To date, specular reflection has been a major obstacle to photogrammetry, handled only in special cases (like flat reflecting mirrors placed in a scene) or not at all; the standard advice is, to accurately scan the geometry of a highly reflective object, cover it in paper tape or cornstarch. This approach, if it works, would turn that advice on its head — you might find yourself wetting objects, or pouring mineral oil on people, to get a more precise 3-D model of

them.

Topics

A compact textual format for interchange of electronic circuit designs

Kragen Javier Sitaker, 02020-11-11 (updated 02020-11-26) (1 minute)

Consider `5V-{4n7F-2k2||2.2mH(2%)-npn(B=50)[1k-G]}-G`. This describes an analog circuit, kind of a stupid circuit, but a circuit, with five components. In Falstad's circuit simulator's save format, such a circuit would take about 190 bytes, but here it takes 42. Moreover you could sort of imagine that such a representation provides a sort of key command interface; it takes me about 20 seconds to type it, and that's tremendously faster than I think anyone can click through all the stuff in KiCad or Falstad's simulator or LTSpice to do the same thing.

I think it's probably worthwhile building something that simultaneously maintains this representation and a 2-D schematic representation.

(For human readability it might be better to say "gnd" rather than "G".)

Topics

Dictionary data structures for tiny memories

Kragen Javier Sitaker, 02020-11-12 (3 minutes)

Instead of using a hashtable you can use binary search on an array, which wastes no space and permits ordered traversal. This is slow to insert into when the array gets big. If you only have 64K of RAM it doesn't get *that* slow; if you have 16384 entries then the worst case is moving 16384 entries, and building the whole 16384-entry thing incrementally takes an expected 67 million entries moved of work.

But it gets a lot faster if you have a side file with, say, 128 entries, which you maintain sorted and then merge into the main array whenever it gets full. Filling memory that way requires doing an expected 2k entries moved to build each side file, and then iterating over the whole array to do the merge, which takes on average copying 8k entries, so 10k entries moved in all; doing this the requisite 128 times requires 1.3 million entries moved to fill RAM. This is slightly more complex and wastes 128 entries more RAM than the single-array approach but is 50 times faster. Also instead of requiring worst-case 14 probes to find an entry it requires 21. You need slack space for 256 entries because merging requires you to copy one of your merge inputs. (Supposedly there's an in-place mergesort but I don't understand it.)

The standard log-structured merge tree approach where you have a 1-entry side file merged into a 2-entry side file merged into a 4-entry side file, etc., starts to run into problems when you don't have more RAM to merge in. A possible solution to this problem is to use quicksort or introsort instead of doing large merges, at the cost of some extra complexity and slowness; merging 16384 entries takes 1 step per entry, but quicksorting them takes expected about 20 steps per entry, and heapsort is even slower.

A sort of library-sort approach might help: divide your sorted dictionary into hashtable-like buckets, each big enough to hold, say, 8 items, binary-search the buckets, and then linear-search the selected bucket. This allows some slack space within each bucket, so inserting an entry is usually very fast; when a bucket fills up, you can sort the entire array (sorting the slack-space items to the end, say; perhaps first compact, then do an insertion sort) and redistribute evenly into either the same number of buckets, thus redistributing the slack space, or a larger number, dramatically increasing it. There's a tradeoff between wasted slack space and frequency of resizing, a tradeoff which eases with larger bucket sizes at the cost of search time.

Topics

Adiabatic separation

Kragen Javier Sitaker, 02020-11-12 (updated 02020-11-14)
(14 minutes)

Some novel material separation techniques.

Thermoacoustics basics

Adiabatic compression and expansion of gases changes their temperature as well as their pressure, so every sound wave generates local temperature oscillations. Sound waves in air can reach a few megahertz over meter-like distances and much higher amplitudes than we are commonly used to, up to nearly an atmosphere even at atmospheric pressure, and higher amplitudes still at higher pressures. Let's consider a sinusoid.

Normally the oscillation of pressure, volume, and temperature is almost perfectly in phase; especially at high frequencies, the hot compressed gas only has time to lose a tiny amount of heat through radiation or conduction before it's being decompressed and thus cooled, and once cold it only has time to *gain* a tiny amount of heat by absorbing radiation or conduction before the cycle starts again. In a traveling wave, 90° out of phase to this oscillation of volume (and, to a very good approximation, temperature and pressure, XXX no that's exactly backwards), is an oscillation of displacement. Assuming no overall movement, a parcel of air passes its average position when it is either maximally small or maximally large. Suppose the wave is moving to the right. When it is maximally small, the parcel is moving at its peak velocity to the right; when it is maximally large, it is moving at its peak velocity to the left. Halfway in between, when its pressure is changing fastest, its velocity is zero, but its displacement is at its maximum.

That is, the velocity oscillation in the direction of wave travel is 180° out of phase with the volume oscillation (and, to an excellent approximation, pressure and temperature, XXX no that's exactly backwards); the velocity oscillation in the direction opposite wave travel is in perfect phase with the volume oscillation. The displacement oscillation lags the velocity oscillation by 90°: the displacement in the direction of wave travel is at its maximum when the volume is at its average level and is expanding most rapidly, and at its minimum (moving at maximum speed in the *opposite* direction from wave travel) when the volume is at its average level and is *contracting* most rapidly.

Because the displacement is almost exactly 90° out of phase with the temperature, there is no tendency to transfer heat preferentially in either direction. When the parcel is at its extremal positions it is at its average temperature, and when it is at its extremal temperature it is at its average position. So the average temperature of the parcel at each position is the same.

Now, if we have some liquid or solid particles suspended in the air, such as fine dust, they will move with it — effectively increasing its density by a bit — but will not be subject to the same adiabatic heating

and cooling, since their volume changes with pressure orders of magnitude less than the air's does, basically a rounding error in this context. They *will* change temperature, but only because they are exchanging heat with the air they are suspended in. This will cause the temperature oscillation to lag the pressure variation a little, with the result that the volume variation (PV = nRT, thus P = nRT/V) will not be precisely 180° out of phase with either of them, but a little bit in between. This means that the temperature oscillation is no longer precisely in quadrature with the displacement oscillation, and so there is a tendency to pump heat in one direction or the other.

This is the basis of two major thermoacoustic effects. Typically, instead of using dust, they use solid objects the gas can flow past, which makes the heat pumping substantially more useful. This allows the moving gas to leave the heat behind. The process is reversible, so it can also be used as a heat engine, essentially a Stirling engine where the mass of the air itself acts as a piston. In either case, there is a temperature gradient along the "regenerator". Many thermoacoustic machines using these two effects are known.

Regular gas chromatography

Gas chromatography separates gases by mixing them into a mobile phase that passes through a stationary phase (normally a liquid supported on an inert solid) which preferentially adsorbs some of the gases, slowing their passage. Consequently the different gases arrive at the other end of the chromatography column at different times. This is usually used for analysis rather than purification; it's kind of inherently a batch process.

Thermoacoustic mixture separation

But perhaps we can use this same thermoacoustic effect to get a sort of continuous-flow gas chromatography or rapid fractional distillation. When the stationary phase (for example, a powder bed or a liquid on the surface of a powder bed) is at its peak temperature, it will tend to free the gases adsorbed onto or absorbed into it; if this is immediately followed by movement in a given direction, all those adsorbed gases will tend to move in that direction. Then, at the lower temperature in the other part of the acoustic cycle, all the gases will move back in the opposite direction — but those that are more strongly adsorbed will do so less. This cycle can happen at kilohertz to megahertz frequencies, so even a small difference can be used. I suspect that by feeding in gas in the center of such a column, you should be able to get one gas out one end of the tube and another gas out the other. The net direction of movement for each gas will depend on the degree to which its adsorption or absorption varies across the temperature range reached by the sound wave, and on the average axial flow in the tube, which is controlled by the difference in the amounts of gas drawn off at each end.

An inefficient version of this, without the packing, was discovered by accident in 2000 at Los Alamos.

If gas is drawn off at several points along the length of the column then the average flow rate through the column will change at each such point. I think this permits the removal of certain components,

giving a sort of horizontal thermoacoustic version of fractional distillation.

Liquid separation

But what if you want to do this with liquid chromatography instead? You can't rapidly heat and cool the liquid or the solid stationary phase by running sound waves through them; they aren't compressible enough. But there are other possibilities.

The most obvious one is that if you're doing TLC, thin-layer chromatography, you can heat and cool your TLC layer by heating and cooling the plate. But there are other more interesting possibilities.

Fractal recuperator reciprocating liquid purification

I've previously written about synthetic *retia mirabilia*, fractal recuperator-style heat exchangers where the heat-exchange capillary surface, which separates the hot and cold reservoirs, is convoluted like the surface of a cauliflower in order to maximize its area. Such a device could be used directly for this kind of separation.

Consider it to have four main spaces: A1, A2, B1, and B2. A1 is connected to A2 by capillaries through the cauliflower surface, and B1 is similarly connected to B2 by separate capillaries through the cauliflower surface. The A and B spaces are not connected by mass flow at all, but intimately exchange heat through the capillary walls. A1 and B1 are always cold; A2 and B2 are always hot. So there is a thermal gradient along the capillaries. B1 initially contains the liquid mixture we would like to separate, and A contains some arbitrary fluid, either liquid or gas. The cycle goes as follows:

- We add, say, ten times the volume of the capillaries to A1. This makes the capillaries cold and causes hot liquid to come out the A2 spout. Because the B capillaries become cold too, more components of the liquid adsorb to them.
- We add, say, a tenth of the volume of the capillaries to B1. This moves some liquid into B2, from the ends of the capillaries.
- We add the same amount of liquid as in step 1 again, but now to A2 instead. This makes the capillaries hot and causes cold liquid to come out the A1 spout. Because the B capillaries are now hot, less components adsorb to them.
- We add, say, a twentieth of the volume of the capillaries to B2. This moves some liquid into B1 from the ends of the now-hot capillaries.

So, components of the B liquid that are more than twice as mobile at the hot temperature tend to move from B2 to B1, while components that are less than twice as mobile tend to move from B1 to B2. And on average the liquid goes through twenty of these "fractional distillation" cycles in the capillaries before making it all the way from B1 to B2. By adjusting the ratio of the amounts in steps 2 and 4, we can adjust "twice" to whatever ratio we want; by adjusting the amount added in step 2 we can adjust the number of cycles and thus the purification.

It probably isn't possible to do this at macroscopic scales at more than a kilohertz or two, and maybe much less, which is a big disadvantage compared to the thermoacoustic approach above.

An H-bridge

Another alternative approach is an H-bridge, like the motor control circuit. Here you have five tubes forming the shape of a capital H. The horizontal tube is your packed column. Fluid moves both left and right in it, but in the vertical tubes it only ever moves down. The vertical tubes have valves. The left vertical tube is connected to a source of cold liquid at the top, and the right vertical tube to a source of hot liquid. First you open the upper left and lower right valves, causing cold liquid to flow down, flow rightward through the packed column, and then flow down the right bottom leg. Then you close these valves and open the other two, so hot liquid flows down the right top leg, leftward through the packed column, and down the left bottom leg.

This works somewhat similarly to the *rete mirabile* approach described above, but requires much stronger adsorption or absorption onto the stationary phase to be of any use, because a useful amount of temperature change has to move through the column faster than the eluent. It has the advantage, however, that it does not require exotic fabrication technology.

(It also may be reasonable to add the materials to be separated in the center of the column, as above.)

The Hot Chocolate Effect

It may be possible to use thermoacoustic techniques with liquids by filling the liquids with bubbles. As observed in the Hot Chocolate Effect, even fairly small admixtures of bubbles give the liquid compressibility of the same order of magnitude as the gas in question, but because the mixture's density is still the same order of magnitude as the liquid, sound-wave speeds are extremely slow, and displacements are extremely small for a given sonic power level. It seems like this might make these methods less effective with bubbly liquids.

Pressure-swing solidification

An alternative mechanism for varying the mobility of ingredients of a mixture is to use the change in the pressure (rather than the temperature) out of phase with the displacement. With notable exceptions like boric acid and water, most liquids reduce in volume when they solidify, so they tend to solidify under higher pressure and liquefy under less pressure. So by repeatedly moving a fluid to the right, compressing it, moving it to the left, and rarefying it, you can get separation of the components that more easily solidify under pressure. (It will work for liquids like boric acid as well, just in reverse.) This will probably work a lot better at near-gigapascal pressures and pressure swings, which is a very loud sound indeed.

One advantage of this version of the approach are that the stationary phase and the mobile phase can be chosen to have nearly the same acoustic impedance, which is not feasible with gases (except

perhaps at extraordinary pressures), which means that interfaces will scatter the sound less, so the sound will be attenuated less. Another advantage is that liquids and solids can transmit much higher frequencies of sound than gases can.

The key challenge in this variant is probably going to be getting enough displacement to outrun diffusion.

Packed columns?

All these "packed columns" might be better as a honeycomb of narrow parallel tubes, a configuration already commonly used for catalyst support, in part to reduce acoustic losses. You could imagine that it would also reduce turbulence losses, but if your passages are wide enough to permit turbulence, your system probably has bigger problems.

Topics

The rep-2 cuboid

Kragen Javier Sitaker, 02020-11-13 (5 minutes)

A4 paper is a rep-2 rectangle: by putting two sheets of A4 paper next to each other, you get a larger sheet that's the same shape as A4 if you turn it 90°, but twice as big. The whole A0/A1/A2 etc. system is designed that way. In the A-size papers, you're never more than √2 away from the ideal size for your application. If you add the B-size papers, which have √2 area relation to the A-size papers, you're never more than $\sqrt[4]{2}$ away.

I'm thinking about how to pack together boxes to make a portable electronics lab (see Ghettobotics Nonshopping List (p. 516)) and it occurred to me that it would be nice to have boxes with volumes that were powers of 2. That way, a small number of box designs would cover several orders of magnitude, and I could always "buddy-system" two boxes of one size together to fit into a space the next size up. It's an attempt to minimize space fragmentation in the toolbox.

One way (maybe the only way) to make a rep-2 box in three dimensions is to make the sides in the ratio of $\sqrt[3]{2}$ to one another; for example, 100 mm × 126 mm × 159 mm. Then the next size up is 126 mm × 159 mm × 200 mm, for example. These ratios are correct to within about a sixth of a percent.

To approximate a 200-mℓ box, reasonable values are 46 mm × 58 mm × 74 mm. A list of mm dimensions covering a wider range, produced by rounding and exponentiation, is [12, 15, 18, 23, 29, 37, 46, 58, 74, 93, 117, 147, 186, 234]; the resulting box volumes in mℓ are [3.24, 6.21, 12.006, 24.679, 49.358, 98.716, 197.432, 399.156, 805.194, 1599.507, 3199.014]. There's clearly some approximation in there; you can put together two 12×15×18 boxes into a 15×18×24 box, a millimeter over; two 15×18×23 boxes make an 18×23×30 box, slightly over 18×23×29, and so on.

Perhaps a more reasonable approach is to just start with some small dimensions and double them exactly. For example, [18, 24, 29, 36, 48, 58, 72, 96, 116, 144, 192, 232] mm gives us [12.528, 25.056, 50.112, 100.224, 200.448, 400.896, 801.792, 1603.584, 3207.168] mℓ. I probably only really need the first seven of those sizes, and they're actually closer to the ideal volumes than the ones given above, although their ratios are a little more imperfect.

There's no real need to have 200 mℓ be on the list, though. I could just look for the best triplet under about 35, which turns out to have only about 1% error from the real cube root of 2:

```
>>> min(((a, b, c) for a in range(1, 36) for b in range(1, a) for c in range(1, b
o
o)),
    key=lambda (a, b, c): max(abs((a/float(b))**3 - 2), abs((b/float(c))**3 - 2))
o
o)
(24, 19, 15)
```

We can cut that 24 in half: [12, 15, 19, 24, 30, 38, 48, 60, 76, 96, 120, 152] mm, giving [3.42, 6.84, 13.68, 27.36, 54.72, 109.44, 218.88, 437.76, 875.52] mℓ.

After cutting two 12×15×19 boxes, a 15×19×24 box, a 19×24×30 box, an a 24×30×38 box out of cardboard, I conclude that probably at the smallest sizes it makes more sense to use paper envelopes, as I am for resistors already. The 24×30×38 box, 27.4 mℓ, is about the smallest one that it makes sense to make as a separate box. And around that size, 27×34×43 has more precise $\sqrt[3]{2}$ proportions, erring by +0.4% in the 34:43 proportion and -0.05% in the 27:34 proportion.

On that basis, the dimensions should be [27, 34, 43, 54, 68, 86, 108, 136, 172, 216, 272, 344] mm and [39.474, 78.948, 157.896, 315.792, 631.584, 1263.168, 2526.336, 5052.672, 10105.344] mℓ. I probably won't need anything bigger than the 1.26-ℓ box! So the sizes are:

- 27×34×43 mm: 39.47 mℓ; call it "one bix"
- 34×43×54: 79 mℓ, two bixes
- 43×54×68: 158 mℓ, four bixes
- 54×68×86: 316 mℓ, eight bixes
- 68×86×108: 632 mℓ, 16 bixes
- 86×108×136: 1263 mℓ, 32 bixes

So with six box sizes I should be able to cover pretty much the whole portable-lab size spectrum, with boxes always within $\sqrt{2}$ of the ideal volume, and packing together nicely. The 40-ℓ toolchest I was spitballing works out to about 1013 bixes, so rounding it up to 1024 is probably more pleasant. It won't be bix-shaped itself.

Topics

Mica composites

Kragen Javier Sitaker, 02020-11-14 (3 minutes)

Mycalex is a composite of mica and glass, one of Dan Gelbart's favorite ceramics for its cheapness and ease of machining, though it's much less popular now than it was in the 1940s. (Machinable glass-ceramics that precipitate mica crystals during heat treatment were invented in the 1970s and may be responsible for some of this). But mica is used as a filler in many composite materials.

I dissected a broken microwave last weekend and found that the window protecting the magnetron from spattering food seems to be a mica composite; it's slightly translucent to light, but when heated with a butane torch to orange heat, it remained intact and barely burned. It did turn black and outgas a little, enough to blister the surface a bit. Wikipedia says phlogopite can withstand 900°, though some other micas only survive to 500°, so this seems likely to be phlogopite-based "mica paper", as the USGS calls it. It isn't pure phlogopite, because that wouldn't turn black. Many vendors offer sheets of "mica" on MercadoLibre for microwave repairs, suggesting to cut them to the correct size with scissors or a razor knife.

I think such sheets (or sheets of pure mica) were the traditional form on which wire resistors were zigzagged, from which we get the schematic symbol.

WP also points out that it's used in drywall mud, presumably for mechanical strength, and as a filler in paint and plastics, where it has many benefits, including increasing strength and dimensional stability. Dry-ground mica is dull, wet-ground mica is sparkly.

It occurs to me that including ground mica in "Starlite" might help it retain strength when it's being charred. (And maybe foam up better, too.) Similarly, it seems that it might help keep alabaster from crumbling when dehydrated; see the note on plaster foam (p. 453). The alabaster would still remain solid to a higher temperature than the mica, but it becomes very friable when dehydrated (at under 200°); retaining substantial strength to 900° could be very valuable.

Ceramics-supply vendors on MercadoLibre sell finely ground mica for about US$2 per kg, but of course they don't tell you which mica it is. Calcining microwave-oven window panels would be a sure way to get refractory mica, but it's rather expensive by comparison. The USGS says scrap and flake mica costs US$120–165/tonne at wholesale ; this is an interestingly low price because it means that alabaster mixed with mica is still cheaper than muriate of lime; see the note on desiccant climate control (p. 489). If the admixture of mica were under about 40%, the mixture would cost the same order of magnitude as the alabaster alone.

Topics

Improvised display options for embedded hardware development

Kragen Javier Sitaker, 02020-11-16 (updated 02020-11-17)
(16 minutes)

It's 02020-11-15. Four days ago my options for microcontroller output were limited to one four-digit LED display I'd painstakingly desoldered from a microwave, some other LEDs I'd painstakingly desoldered from other random discarded electronic equipment, and the microwave's beeper. Now the covid quarantine is over and I've been able to visit my apartment (I've spent the last 7½ months at my girlfriend's apartment) and now I have a number of additional options.

display	w(mm)	h(mm)	type	shows	color	pins	conn
DeV96	50	18	LED	4 0-9	red?	14	SIP 2.54mm
microwave	45	12	LED	4 0-9	green	14	SIP 2.54mm
breadboard	50	18	LED	4 0-9	red	36	DIP 2.54mm
Kenko	52	12	LCD	8 0-9	grey	≈32	flex 1.26mm
Kadio	60	18	LCD	12 a-z?	grey	≈75	flex 0.8mm
				+12 0-9			
Franklin	53	18	LCD	25 a-z?	grey	≈105	flex 0.5mm

Some of these are sort of package deals with a keyboard and maybe a battery and case, which tempts me to rebrain them (see Rebraining (p. 597)), while others are just displays.

The LED displays are suitable for PWM fading; not sure if this will work with the LCDs. Nobody ever does PWM fading of 7-segment LED displays, which is going to rock. Also I think LED displays would look much better shining through cloth, another thing nobody ever does. These LCDs are all reflective LCDs, so they would have the best readability in daylight.

Cellphone screens are pretty interesting, especially SPI Nokia screens, although I don't happen to have any at the moment.

DeV96

A guy at a hackerspace gifted me this display he'd made on perfboard, evidently on 01996-10-13. It has an 8-key keypad, apparently configured as a 2×4 matrix with diodes, and five BC548B transistors onboard; four of them are hooked up to turn on a particular digit of the 7-segment LED display. I traced the wiring out before, but I forget if you can fully control the display from offboard with the 14-pin header he's provided, or if it's partly under control of the keyboard.

The 7-segment displays are helpfully in a socket, so not only can you replace them if you burn them out, but you can easily transplant them into other things. Each has a decimal point.

This may be the easiest option to use, since it already has

current-limiting resistors (probably sized for 5V) and common-electrode transistors with current-limiting resistors on it. The individual segments are driven from offboard, so you need to source or sink (source I think) enough current to light them up. It's also one of the largest displays, which will make it easy to read. It's probably not using high-brightness LEDs, which means it won't be as bright as some other options, especially if run off 2V or 3V.

I suspect 12 pins of the 14 pins, plus power and ground, would be enough to drive the display, and it would probably work down to 2V.

Microwave

The front-panel display from the microwave oven I ripped apart has four 7-segment digits without decimal points, a colon with separately illuminable dots, and ten miscellaneous microwave-oven-related ideographic indicators above and below. The digits are slightly smaller than the DeV96 digits, but the component as a whole is much smaller, and it lacks any onboard resistors or transistors. There are nine cathodes (?) for the individual segments of a digit, then five common anodes (?) to select the digit; one of the five "digits" is the colon and two of the miscellaneous indicators, while each of the other digits controls two more miscellaneous indicators. So you can control all the digits proper with 11 pins.

The digit segments light up faintly green on the milliamp or two from my multimeter's diode-test mode, but are presumably intended for a much higher current. I could probably stress one of the colon dots until it blows to get an idea of what they could stand, but 20mA is probably safe, and close to the limit of what my microcontrollers can provide anyway.

It's labeled GAL9801-OI on one side, but GAL-something was the microwave's model number, and no datasheets seem to be forthcoming.

I was able to get the microwave's somewhat yellowed front panel off in one piece; it has a flat-flex cable (dated "11/18/99") which went into a 12-pin cable-pinch connector on the mainboard (where the display was soldered), and I was able to desolder that connector, which has the same breadboard-friendly 2.54mm pin spacing as the display. Its polarized front window of course can fit the display in it. It has 17 membrane-keyboard buttons labeled with microwave-related things and connected in some kind of matrix with resistive ink on flat-flex to the 12-pin connector. I was able to detect a press on one of them with the multimeter; it read as about 120 Ω.

I'm not sure, but I think these are "bright green" 3-volt LEDs rather than the older 1.7-volt-or-so older green type. So running the display at 20 mA per segment would probably cost 420 mW worst-case. But at an equivalent brightness it probably costs less power than the DeV96 display.

I could very reasonably tack the display in place inside the front panel with some silicone or something, then mount a control board and even power supply inside, repurposing the shitty membrane microwave keyboard for some nobler purpose, perhaps covering it

with labels. The 7 segment-selector pins could presumably be shared with at least 6 and probably 7 of the pins for the keyboard, so you'd probably only need 16 pins between the keyboard and the display. The 18-GPIO ATTiny2313 has enough pins for that, but not to do much else, and no ADC, which would be useful for many of the things I want to show on the display. But it can blow or suck 40 mA per I/O pin, so it could drive the display at a dimly readable intensity without any external hardware at all, and probably very brightly if supplied with four per-digit high-side transistors.

If I'm wrong and they're actually low-side transistors, it might be able to drive the transistors with its internal pullup resistors! 50kΩ worst-case at 5 volts gives us 100 µA, and with an unremarkable β of 300, that would give us... 30 mA. Which is worse than what it can do itself. *sad trombone* It might happen to be higher than that, or you might be able to use a darlington. But then you can also get transistors with integrated base resistors.

An STM32 or CKS32 is more appealing in many ways because it has a kickass ADC and a lot more I/O pins, but the STM32 can only source or sink 25 mA per pin, and can only run at 3.3 V. (I think the CKS32 is the same, but I have a harder time reading the datasheet.) And I only have two, so if I blow one up, I will be a lot sadder than if I lose a 2313 or a few ATTiny45s.

As for 45s, I have dozens, and it might be possible to gang several of them up to drive the display, maybe using two wires for I²C ("TWI" according to Atmel) and leaving four GPIOs per chip, or three GPIOs plus an ADC. Four of them would be enough to drive the display and read the keyboard, though you'd probably still want external drive transistors for the display digits.

Breadboard

At my house I picked up a breadboard that has two 18-pin two-digit 7-segment red displays in it, probably from 2006. They're labeled LDD5111-11. These have decimal points and need external current limiting. They're probably the largest digits I have available. The datasheet, from 1996, says they have a surprisingly rational pinout with 16 individual per-segment anodes and two common cathodes. You could gang up the corresponding per-segment pins to get a 12-pin interface, but you'd probably want to drive them at more than 40 mA per digit, so you probably still want external low-side drive transistors for the common cathodes.

The datasheet ("Jameco Part Number 24723", "Ligitek", for a whole family of such displays) says these are red GaP LEDs at 697 nm with a 90-nm bandwidth and a 1.7–2.8 V forward voltage drop, with 2.1 V being typical; 0.5 millicandela minimum at 10 mA, 0.8 millicandela typical; either 40 mA or 15 mA maximum current "per chip" (or 200 mA or 60 mA pulsed at a 10% duty cycle), and either 110 mW or 45 mW maximum power per chip; and 10 µA reverse current "absolute maximum". I don't know which power rating to use; the higher rating is "SR", and the lower is "H", but I suspect it's the lower one, because that's the one that is dimmer at 10 mA.

I think "per chip" means "per LED" rather than "per digit" or

"per package" because 15 mA per digit would be 2 mA per LED, just too low.

This may be the easiest option of all, as well as the most readable, indoors anyway.

So, to run these manually off 5 volts, I probably want 8 per-segment resistors of 220–2200Ω, and maybe 55Ω when driving them with a microcontroller.

A little testing shows that the forward voltage at 14.5 mA is about 2.1 V, and they aren't terribly bright at that current. I may try burning out a decimal point in one to see how bright they go.

Kenko KK-9835TS

I have a talking Kenko KK-9835TS desk clock/calculator from 2007, which has an extremely abrasion-resistant transparent keyboard, an 8-digit 7-segment LCD display, a dynamic speaker with a plastic cone, and lots and lots of wasted space inside. It runs off two AA batteries. It's about 155×120×32 mm and weighs 143 g. The mainboard is of course a chip-on-board with a blob of epoxy over it.

This display probably has the largest LCD digits of the displays I have and would thus be optimal for daylight readability, and the flex cable to the LCD has a very comfortable 1.26-mm "pin spacing" between its 32 or so lines. The slightly mushy 26-key keyboard is also connected via a flat-flex cable, making it ideal for rebraining. The keys are easily replaced (I've turned the digits upside down) but it might be even more interesting to reformat them by taking them out, grinding the opaque backing off the back, and replacing it with a custom-printed sticker.

Driving 32 LCD lines in software probably requires 32 GPIOs and a certain amount of attention to properly alternating the polarity so you don't electrochemically degrade the display. But it doesn't require much voltage or current.

Kadio KD-82TL

I have a Kadio KD-82TL copy of a Casio scientific calculator from 2002, which I've written about previously in Dercuano. It has a 50-key very mushy keyboard and a two-line display, one line of which is something like 12 5×7-pixel bitmaps (used for displaying formulas and programs) and the other line of which is something like 12 7-segment digits with decimal points. (It also has 11 indicators for calculator modes.) It's about 85×165×30 mm and weighs 139 g with the cover on.

This is also very appealing for rebraining; again, both the keyboard and the screen are connected to the brainboard with flat-flex cables, and it runs off a couple of AA batteries, which I have removed because they had run down. It comes with a protective shield that slides over the face to protect it from damage. There is less wasted space inside than on the Kenko.

Driving the LCD seems potentially a lot more challenging than on the Kenko: there are twice as many pins, and they are less than a millimeter apart. Worse, I seem to have torn the flat-flex cable off the LCD edge where it was connected, and so I need to figure out

how to re-establish contact.

Franklin TES-118A

This alphabetic LCD display is in a "Franklin® model TES-118A Spanish-English translator, ISBN 1-56712-689-8" Beatrice and I bought in 2005. The whole device runs off a CR-2032 coin cell, measures about 105×70×15 mm, weighs 64 g, and has a pushbutton-latching domed clamshell cover that covers everything but the power button and the screen, which is recessed so far into a hole in the clamshell that it still hasn't broken, though it's a little scratched. Under the clamshell cover is a very mushy rubber QWERTY keyboard with arrow keys, five ideographic keys for different "applications", and six miscellaneous buttons: menu, power, clear, back, space, and "conj". Since I don't have a CR-2032 handy, I'm not sure how many pixels are on the display, but it's intended to display a few words at a time.

Opening the case was a little tricky; the clamshell unclipped from two steel rods acting as hinge pivots, three Philips screws under the battery cover came out, and then the two halves of the bottom of the case unclipped with fingernail pressure — at which point the tiny hinge pivots and plastic clamshell latch button fall out. I was glad to have a styrofoam sandwich tray underneath it to catch the parts, but still had to shake some out of my clothes.

This is unbelievably appealing for rebraining: a pocket computer ruggedized to survive being sat on, with a daylight-readable tiny LCD that can run on microwatts, with a space for a coin cell! But it would be very challenging: the LCD connector not only has many pins (about 100) but also the keyboard is on the same PCB as the chip-on-board epoxy brainblob. And there's not a lot of wasted space; from the dimensions above you can see that it's about 0.6 g/cc. So the most feasible approach might be to grind away the chip-on-board and wire up the new brain to the now-floating connections, rather than try to use the flat-flex connector.

There's actually a bit of extra space under a silver-covered "keyboard template" which snaps over the keyboard. Like a millimeter or three. Hard to use but it's about 15% of the device's total volume.

Topics

Capacitor meter

Kragen Javier Sitaker, 02020-11-16 (updated 02020-12-03)
(26 minutes)

The project in Multimeter Metrology (p. 502) is a bit large to tackle all at once. So I think it would be good to start with a manageable subset. In particular what I want is something to use for measuring capacitors in the 47 pF to 1000 µF range to within about 1%.

A basic design

If you put a string of two resistors between analog switchable pins A and B of a microcontroller, and a capacitor between the junction between the resistors and ground, then you have the following configurations available:

• pin A output high, pin B input: capacitor charges through resistor A, being continuously watched;
• pin A output low, pin B input: capacitor discharges through resistor A
• pin A input, pin B output high: capacitor charges through resistor B

• pin A input, pin B output low: capacitor charges through resistor B
• pin A high, pin B low: capacitor charges to Ra/(Ra+Rb), can be measured later
• pin A low, pin B high: capacitor charges to Rb/(Ra+Rb)
• pin A low, pin B low: capacitor discharges through Ra||Rb
• pin A high, pin B high: capacitor charges to Vcc through Ra||Rb
• pin A input, pin B input: capacitor discharges through its internal leakage resistance

All of these measurements are essentially ratiometric, so we don't care about the power-supply or reference voltage as long as it isn't too noisy or so high the capacitor blows up (which a double-layer capacitor might.)

Ideally, for component ID, we could go through all 9 measurements in 50 ms or so for capacitors anywhere in range. If Ra and Rb are some distance apart, like an order of magnitude or so, they can provide different "scales"; but if they're *too* far apart, then several of the cases won't be meaningfully different. How far apart is "too far apart" depends in part on the ADC's precision.

I can measure the resistances to an error of about 0.1% with my existing multimeters (in Multimeter Metrology (p. 502) I found that they were about 0.03% apart). This doesn't tell me their temperature coefficients until I heat them up or cool them down and measure them again.

If the resistance is too low, the capacitor will charge all the way to its destination voltage too quickly to get a reasonable number of samples, so higher sampling rates mean we can get by with lower resistances. If the resistance is too high, the capacitor won't charge or discharge enough to measure reliably within the measurement

interval, so higher sampling precision means we can get by with higher resistances. Also, though, higher resistances make life harder for the input circuits.

Larger numbers of resistors and pins could be deployed to get more measurements.

I'm going to look at a few candidate chips.

ATTiny45

I have 18 surface-mount ATTiny45s and 19 DIP ATTiny45s left over from 2006. They run at 20 MHz on an external crystal, or up to 16 MHz on a PLL from their internal oscillators, and have six GPIO pins, 4K of Flash, 256 bytes of RAM, four single-ended ADC channels with a 10-bit ADC, an analog comparator, and supposedly two differential ADC channels with switchable 20× gain (although I wonder if maybe that last feature is actually only in the ATTiny45/V). The I/O pins have a selectable pullup resistor of 20–50 kΩ.

The internal RC oscillator is only calibrated to ±10%, but the datasheet says you can calibrate it to ±1% at a given temperature, voltage, and frequency. This would seem to eat the whole error budget right out the gate. But an external crystal eats up two of the six GPIO pins.

Like most AVRs, at its maximum precision of 10 bits, it can only do 15 ksps, but it can run about 5× faster if you're willing to accept more error. It recommends an input impedance of 10 kΩ or less and says its sample-and-hold capacitor is 14 pF, which makes it sound like it's going to be pretty hard to measure down below 100 pF.

At 15 ksps we would get 45 samples in 5 ms. If we were interested in the *average* voltage, this would give us about a 12½-bit measurement, which is plenty precise enough for our 1% objective. Probably measuring the charge or discharge rate will give us a 12-bit (0.1% precision) measurement, which is still plenty. (I haven't done the analysis rigorously using the various error numbers from the datasheet but I think these are in the ballpark.)

How fast of a time constant can we tolerate? A measurement stops being 1% accurate once it drops below 100 counts or so, about a tenth of full range, so we have about ln 10 = 2.3 time constants, maybe a bit more, to get our measurements. (I'm glossing over the fact that what we need to be 1% accurate is not the voltage but the timing, which I don't think matters.) I think we need at least 3 samples, probably more, say 5 to be safe. So we need the RC time constant to be at least 2 samples, 133 µs, when we're using at least one of the resistors.

Getting that at 47 pF would require a 2.8-megohm resistor, which is inconvenient not only because my multimeters can't measure that high (I'd have to do a four-wire measurement with two multimeters) but also because the ATTiny45 datasheet specifies input leakage current of up to 1 µA, though typically below 0.05 µA, and claims the analog input resistance is typically 100 MΩ. So compromising with 470 kΩ for the low-capacitance-range resistor might be a better idea. (I suspect this might still be too high for the inputs to work reliably with.)

How low of a charge/discharge rate can we tolerate? By the same

handwaving arguments above, I think we want to ensure that the charging/discharging makes it up to 200 counts, about 20% of full range, within the 5-ms measurement window. ln 80% ≈ -.22, so that's about 0.22 time constants, so τ can be up to 5 ms/0.22 ≈ 22 ms. This is disappointingly close to the minimal τ of 133 µs above, only 165 times longer. If we want to get to 22 ms with 1000 µF, then we would need a 22-Ω high-capacitance-range resistor, 20000 times smaller than the low-capacitance-range resistor. This would allow us to meet our performance objective for 1000 µF, but at the cost of precision in the midrange.

This situation probably calls for compromising like crazy to reach acceptable performance. Here are some candidate compromises:

• Use the internal pullup (on the 22-Ω pin) as an alternative medium-range pullup resistor. This requires some kind of calibration procedure to find out what the internal pullup's E-I curve looks like (maybe not a straight line) and how it varies with temperature.
• Use a third pin with a third external resistor. But keep in mind the chip only has six GPIOs!
• Use a longer measurement time for large capacitances so you can use a larger resistor. 470Ω would give you a time constant of 470 ms with 1000 µF, so you could reach the desired performance at 100 ms per measurement rather than 5 ms.
• Accept some performance degradation at the ends of the range.

However, there are some tactics not yet investigated:

• Maybe I'm being too pessimistic in some of my calculations.
• Can the differential-input mode with its selectable 20× amplifier help? Maybe that would allow us to double the number of samples we can take for small capacitances, by turning the amplifier on when the signal gets low.
• Maybe if the capacitor is charging/discharging too fast, we could charge it to way outside the ADC range and measure how long it takes to discharge down into the range. For example, we could charge it to 5 volts and use the 1.1V internal voltage reference as full-scale. The 20× amp might help there too. This obviously only works for
• You can configure the ATTiny45's PWM output to generate PWM levels at up to the system clock speed. If running off a 16MHz clock you could get three PWM levels at 8MHz (0%, 50%, 100%), four at 5⅓MHz, five at 4MHz, and so on. Even at 1600 kHz you have 11 levels. I'm not really sure how this helps though.
• Maybe a higher sampling rate at the expense of precision would be good when the capacitance is small.
• Maybe even if the capacitor finishes charging or discharging in two or three samples, by repeating the measurement dozens of times we can get a precise enough average.
• For fast charging/discharging, the ATTiny45 also has an analog comparator, which can be configured as an interrupt source. IIRC AVR interrupt latency can vary by up to two clock cycles, so this would give you timing information precise to ±62.5 ns on the RC oscillator or ±50 ns on a quartz crystal. This would require an additional external voltage divider and I/O pin to provide the

threshold voltage, but it gives you 1000 times tighter timing than the ADC.

Marcel Post's postwiki article goes into some detail on the ADC on these chips. (The ATTiny85 is the version with twice as much RAM and Flash.)

It seems like it would be pretty difficult to meet the targeted performance on the ATTiny45, but maybe possible.

(Actually I think the interrupt latency is not an issue because I think there's an option to latch a timer value automatically when the comparator fires, and I think you can run the timers at the chip's full clock speed, but I need to check those out.)

Time-domain sensing is a better option here

A much easier thing to do on this chip would be to do *only* the comparator time interval measurement against a fixed threshold, using a single pullup resistor from Vcc to one comparator pin and a divider from Vcc to ground across the other comparator pin. If you can calibrate at startup or at regular intervals against a standard NP0 calibration capacitor then you ought to be able to compensate for the internal oscillator's vagaries. We can choose whatever threshold we want, such as Vcc/2. Then, if we want charging a capacitor to our threshold to take at least 10 clock cycles at 16 MHz, we need a time constant of at least 902 ns; if it's 47 pF we need at least 22 kΩ. In 50 ms we can take 80,000 measurements, perhaps alternately of the DUT and a reference capacitor, and average them, giving about 11 bits of precision.

As the capacitor under test gets bigger, the measurements get slower, until at 3.3 µF our time constant is 73 ms, so it takes the whole 50 ms to hit the comparator threshold. This gives us a nice high-precision measurement of 19.6 bits but it starts to get slow. At 1000 µF it would take 15 seconds!

A possible solution to this problem is to generate the reference voltage threshold with PWM instead of a voltage divider, thus using an external RC filter instead of two external resistors. At 160 kHz, you can get 99 voltage theshold levels, taking times ranging from 4.61 time constants to 0.01 time constants to cross them. So we can deal with a time constant from 100× longer than our measurement period (50 ms) down to 4.61× shorter than our shortest acceptable time (spitballed as 625 ns above). This means 136 ns (47 pF · 2.9 kΩ) to 5 seconds (1000 µF · 5.0 kΩ). That's close enough to be reasonable.

(Alternatively the PWM can feed one of the ends of the DUT and we can use a low fixed divider for the threshold.)

We probably want to RC-filter the PWM enough to get 1% ripple or less, which is easy if it's just providing a reference voltage, which means *the filter's* RC time constant should be at least 625 µs; but we want it to be fast enough to be able to switch "ranges" while trying to get the discharge time reasonable, at most a few milliseconds. The filter's time constant (and thus the capacitance and resistance) is not critical to precision, but it needs to have low enough leakage current that its voltage doesn't sag by more than 1% during a 6.25 µs PWM cycle, but anything with such high leakage current that its own leakage time constant is under 625 µs would be too poor to be sold as

a capacitor. On top of this, we have the ATTiny's < 1 µA input leakage current; at 22 nF and 6.25 µs this would be a 280 µV voltage shift, which would amount to a 1% error on a 30-mV reference voltage, so the filter capacitor should be bigger than 22 nF. A 1 µF capacitor and a 2.2 kΩ resistor would give a 2.2-ms time constant and worst-case 6.25 µV drift from the leakage current.

If the RC filter is feeding the DUT the problem of its design becomes much more difficult.

Atmel app note AVR400, document doc0942.pdf describes a similar design, using an AT90S1200 (lacking an ADC entirely) with a crystal and an RC circuit to measure a voltage to 6 bits of precision. In that case the "reference voltage" is the thing to measure and the RC circuit just provides a sawtooth to compare it to.

The analog comparator in the ATTiny45 can take its negative input from any of the four ADC input pins, which enables us to switch between the standard/calibration capacitor and the DUT on each measurement cycle, which reduces the time available for decalibration to be produced by a temperature shift, a supply voltage shift, a clock speed shift, or resistor aging. A temperature shift will also change the value of the standard capacitor, though the NP0/C0G tempco is limited to 30 ppm/K, and it may be possible to use the ATTiny45's on-chip temperature sensor to compensate for that.

The AVR TransistorTester manual mentions that the offset voltage of the AVR's analog comparator limits its accuracy on low-value capacitors.

So the final circuit is something like PWM1-2k2-AIN0-1µF-gnd; Vcc-2k2-ADC0-DUT-gnd; Vcc-2k2-ADC1-1nF(NP0)-gnd. That is, the reference-voltage pin is connected to the output of a single-pole RC low-pass filter from PWM1 to ground, and the multiplexed inputs ADC0 and ADC1 are connected to similar RC filters that are "filtering" just V_{CC}. Some external protection diodes and a protection resistor might be useful on the DUT terminals to reduce the risk of damage from connecting a precharged capacitor.

ATTiny2313

The time-domain design is also applicable to the ATTiny2313, which has PWM outputs and an analog comparator but no ADC at all; it has only 128 bytes of RAM. But it has 18 GPIOs instead of 6. I have a tube of 16 ATTiny2313 SOICs left over from 2006.

The ATTiny2313**A** is "the picoPower version". Amazingly, both versions of the device are still in production, though the manufacturer Atmel is dead.

However, the ATTiny2313 has one serious drawback compared to the ATTiny45 for this purpose: its analog comparator doesn't have multiplexed inputs, so it *always* compares pin 12 and pin 13 (AIN0 and AIN1) (or pin 10 and pin 11 in the MLF/VQFN packages I don't have). So it can't recalibrate to a calibration capacitor evey measurement cycle. So getting reasonable precision on the ATTiny2313 (better than 1%, maybe better than 10%) would probably require using a crystal with that design.

A simpler design

A possibly better design, at least for the ATTiny2313, is gnd–1kR–{1kR–AIN0 || 10nF(NP0)–GPIO1 || DUT–GPIO2}, with AIN1 connected to a filtered PWM output as before. With GPIO2 tristated, we can toggle GPIO1 to charge and discharge the 10nF NP0/C0G capacitor through the 1kΩ ground resistor, and observe the charging process as a falling voltage on AIN0. When we're discharging it we may be in part discharging through the clamping diodes of GPIO2 and AIN0 (and AIN0's protection resistor), and also the voltage on AIN0 is negative, so only the charge time is visible.

At 16 MHz the time constant for the specified values is 160 cycles, which is plenty, and maybe a bit generous.

So then we tristate GPIO1 and do the same charge–discharge process with the device under test, allowing us to observe its time constant with the same 1kΩ resistor. If it's 47 pF the time constant is 47 ns (0.75 cycles), which is a bit on the low side, and if it's 1000 µF the time constant is a full second, so we're stuck measuring it against high thresholds like $0.9V_{CC}$ (0.105 τ, 105 ms) and $0.99V_{CC}$ (0.01005 τ, 10.05 ms). But both of these values seem reasonable.

The current decay curve through the GPIO pins depends on the capacitance (and, a little, on the supply voltage), but initially it's 5 mA, assuming $V_{CC} = 5$ V, which is a safe limit and probably won't even heat up the chip much. Heating up the chip might be bad because it might be local and so unbalance the analog comparator, introducing error. Heating up the resistor might be a bigger concern — the peak power there is 25 mW, and for large DUTs it will dissipate essentially 25 mW all the time — but most of that heating effect will be canceled out by alternately measuring the DUT and our reference capacitance.

Since we're only measuring the τ_1/τ_2 ratio, by charging up to the same max voltage through the same resistor measured against the same voltage thresholds with the same clock, our measurement should be independent of slow changes in any of these.

Checking out the datasheet (doc8246.pdf, 8246B-AVR-09/11). The microcontroller only has 128 bytes of RAM (p. 1), including the return/interrupt stack, plus the 32-byte register file (p. 11), which is mapped at addresses 0 through 0x1f, while the data SRAM is mapped into the 128 bytes at 0x60 to 0xe0 (p. 17, clearly incorrect), and also there are three spare bytes in the I/O register space, GPIOR[012] (p. 17); all together, this is enough for a few state variables but not much of a buffer of past samples. The 1024 instructions of Flash may be a bit of a pinch but should be doable. It has two timers (p. 6), which is enough to generate PWM from one while using the other to measure the charging time.

There's a clock prescaler CLKPR (p. 32) which can divide the master clock by any power of 2 from 1 to 256, and a separate CKDIV8 "fuse" which is initially programmed (p. 33), which means the ATTiny2313 runs at 1 MHz by default (p. 27). The internal RC oscillator is 8MHz (p. 19) so you only get 40% of the chip's potential clock speed without a crystal. (There are also 4 MHz and 128 kHz internal oscillators, selectable via CKSEL (p. 27).)

The analog comparator (p. 168) is specified as having less than

40 mV of offset voltage and typically less than 10 mV, which is pretty reasonable — it's better than 0.1% of 5 volts. And it's specified as having an input leakage current of -50 to 50 nA, which is a lot better than I expected. (p. ???)

For sourcing and sinking current it looks like the output impedance is in the neighborhood of 60Ω (charts on p. 242) or 25Ω when running on 5V (pp. 243–4). It can do 20 mA per pin.

There's an "input capture unit" that can be configured to latch the timer value when triggered by the analog comparator on at least timer 1, the 16-bit timer (p. 92). This seems like a much better option than using interrupts, which is four clock cycles, minimum, plus normally a three-cycle jump, and possibly finishing a multi-cycle instruction that was in progress when the interrupt fired: 7–9 cycles of latency. The worst part there is the two cycles of jitter, which will make hash of any data about fast RC time constants.

There's an implication that there's a clock prescaler specific to timer 1 (p. 94, where it says it doesn't apply to the optional noise canceler, which adds four cycles of latency).

There's also a third timer on the chip, the watchdog timer, which always runs at 128kHz and "can be configured to generate an interrupt instead of a reset" (p. 43) by setting the WDIE bit in WDTCSR (p. 45). Aside from the usage they suggest — waking from power-down — this could be useful for doing regular tasks. I don't think you can read its counter value, though, just reset it.

Datasheet questions:

- what's the timer precision?
- what's the comparator noise? hysteresis?
- can we maybe use the pullup to see if the cap is too big?
- what's the input impedance of the pins? might we have error from that?
- what's the input clamp diode rated for?

Interestingly I think this approach also would work for light sensing with LEDs; when I did http://canonical.org/~kragen/light_sensing in 2006 this approach didn't occur to me, and consequently my light sensing on the ATTiny2313 was extremely slow.

STM32/CKS32

I have two Blue Pills, an STM32 and a CKS32. These feature a 12-bit 1Msps ADC, 128KiB of Flash, and 20KiB of SRAM, a shitload of pins, and run at 72MHz. This suggests that we ought to be able to do the whole curve tracing thing.

If we again want to get at least 5 samples before the signal drops below 100 counts, well, 100 counts is 1/40.96 of full scale, so 3.7 time constants instead of 2.3, so our time constant needs to be at least 1.35 µs, a hundred times faster than the AVR ADC, attainable at 47 pF with a 29-kΩ resistor. And if we again want to make sure we get at least 200 counts of change within the 5-ms window, that's only 4.9% of full scale, about 0.05006 time constants, so our time constant can be up to 99.8 ms, which I'll just irresponsibly round to

100 ms — which still requires a 100-Ω high-capacitance-scale resistor.

So even with these more powerful chips, the two resistors are a factor of 290 apart, which doesn't give you much benefit from all the cute tricks at the beginning of this document; but now they cover the center of the range to adequate precision as well.

ATTiny12

The 8-pin ATTiny45 or its larger version the ATTiny85 sell for US$1.50 to US$2 on MercadoLibre here, but there's a vendor that sells the deprecated ATTiny12 (also 8-pin) for US$0.30 or so, so it'd be interesting to see if it might be usable for this kind of thing. It only has 1 KiB of program memory, enough for 512 instructions, runs at only 8 MHz (or less in some configurations, like the ATTiny12V-1), and has no RAM other than its 32 8-bit registers, 64 bytes of EEPROM, and a 3-level hardware return stack. It has no ADC, but it does have the analog comparator — without multiplexed inputs, as in the ATTiny2313, but with interrupts. It has a single timer, but without hardware PWM; you could use it to time the discharge or recharge curve, but you'd have to do PWM in software, where interrupts would screw it up. The internal RC oscillator runs at 1.2 MHz instead of 8 MHz, so you need to use an external crystal to get higher speed or consistent speed, which of course eats up two of the 5 GPIO pins.

I feel like this chip would be pretty difficult to get anything done on due to its extremely limited resources. Maaybe you could get it to work for measuring a capacitor, but I'm not sure how.

7-segment LED displays

I'm thinking a 4-digit 7-segment LED display is probably sufficient for a 1%-error meter. Possible capacitance displays might look like any of these:

```
.001F 100µ 10µ0 1µ00 100n 10n0 1n00 100p 10p0
```

A "µ" on a 7-segment display can look like a backward 4.

Topics

Rebraining

Kragen Javier Sitaker, 02020-11-16 (updated 02020-12-06)
(12 minutes)

One of the things I've been wanting to do is to make an autonomous personal computing device, since one doesn't seem to be forthcoming from the economy. It seems likely the easiest way to do this is to get an existing device with the right peripherals (power, input, and output), remove its CPU and RAM, and implant a new microcontroller.

None of the photos below are mine, so I cannot release them to the public domain; they are included for the purpose of factual commentary, including links to their sources.

Miscellaneous

Scientific calculators seem particularly promising here, but I should probably start with something easier, like a large four-banger. See the note on screens (p. 584) for some details.

The *most* interesting possibility here is probably a cellphone, maybe a flip phone for lower battery usage — these tend to have low-power sunlight-readable screens, rechargeable batteries, USB connections, usable keyboards, speakers, and microphones, as well as some radio stuff. Old Nokia displays in particular are SPI and so relatively easy to connect up to (easier than directly driving an LCD), though you suffer some display latency as a result, and they only use a couple of microwatts. A broken Nokia 1110 goes for AR$500 (US$4) on MercadoLibre; a lot containing a 3200, a 9300, and a 2651 go for AR$1600 (US$10); a lot containing eight broken cellphones from that epoch, including two QWERTY phones, goes for AR$2000 (US$14); a broken QWERTY Nokia Asha 303 goes for AR$900 (US$7). (The key words on Mercado Libre are nokia reparar.) But this is probably a much more advanced project.

I also have a solar garden light that seems like an appealing possibility, though perhaps to dissect for parts; see Garden Light Panel (p. 612) for details. I was disappointed to learn that the panel is only 40–80 mW rather than the 300 mW I was hoping for, probably because it's amorphous, but that's still enough to be useful.

Toys

There's a "Lionel's Smart Tablet infantil" on MercadoLibre right now for sale by "chiquiplanet" for AR$3000 (US$20); it has what appears to be a large passive-matrix LCD screen that seems to be monochrome and about 80×16 non-square pixels, and has a gaudy 46-key ABCDEF membrane keyboard:

It runs on AA batteries and appears to have a reasonably robust construction, although it's not marketed for kids under 3, which makes me wonder if it's aimed at kids with intellectual disabilities. It's 250 mm × 190 mm, and looks to be about 15 mm thick, which seems like it would be a lot of space for adding a solar panel.

For AR$3500 (US$23) there's a toy "bilingual Disney Cars computer" sold by El Mundo del Juguete, which looks like it maybe has a higher resolution display and also runs on AA batteries; it has a QWERTY keyboard and a mouse:

And the old standard "9999 in 1 brick game" from 2006 or earlier is still selling, at AR$790 (US$5), which I think is the same price I paid in 2006:

This thing has pixels on the LCD screen that are literally the Tetris squares; I think that means it's 10×20 pixels. The standard chip-on-board hardware provides you with a number of different games (with a "difficulty" parameter to multiply out to the 9999 number) within that context.

Kinderland sells, for US$10, a keychain-sized clone of the Pac-Man arcade machine that runs on two AAA batteries:

This doesn't have much of a keyboard, but it does have lots of space to add one, and it has what appears to be a thumb-sized backlit LCD (20 mm square) with a resolution on the order of an NTSC TV. The advertisement explains several times that it's a completely functional replica. Several buyers marvel at how well it works. Tim Schuerewegen says he reprogrammed a similar but somewhat larger game console just by reprogramming its SPI Flash ROM.

I don't know what kind of electronic interface the Pac-Man LCD takes but I imagine it's pretty power-hungry.

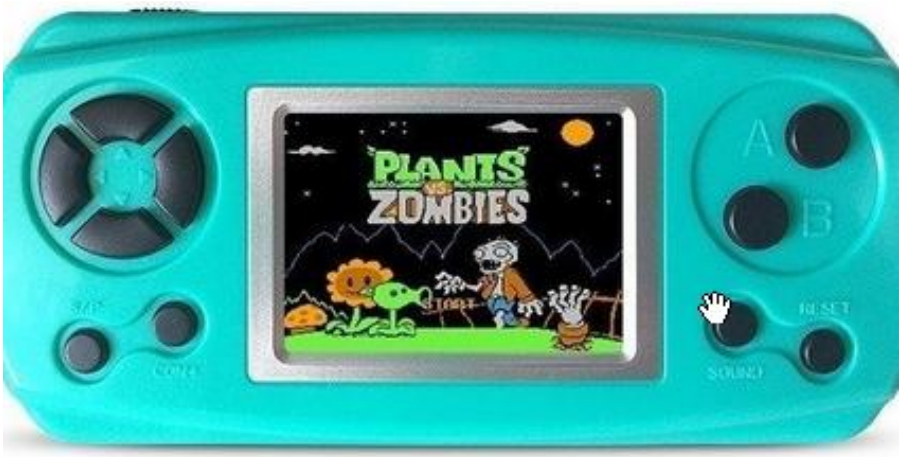
A more modern version of the "brick game" is the "GC-26 168-in-1 portable console", which is a sort of Gameboy form factor (78mm × 117mm × 24mm) with a 2.8-inch backlit color LCD and a volume knob, for US$12.50:

The resolution looks pretty decent, like at least 640×480; probably it's precisely NTSC or PAL resolution:

This has composite video output, charges over USB, includes an 800mAh lithium battery, an 8-bit CPU, 8 megabytes containing 168 ROMs, and probably no programmability. Hackaday did a teardown of one under the brand "Weikin" last year; they found it had a 128 mebibit NOR Flash chip you could probably reprogram. Some dude on YouChube reviewed the games and found some problems in the composite video output; he also took it apart and fixed a video problem.

There are a variety of branded variants of this device, including the Level-UP RetroBoy ("5 hour", "600 mAh" battery), the NogaNet

Pocky, and the Sup. A smaller-scale version is sold for US$8 as the 351-gram Seisa SY-891 or SY-888A Game Player Digital Pocket System, with 328 games on a 2.2-inch screen:



Telephony

Various kinds of telephony devices have displays, keyboards, buttons, and sometimes radios.

For US$9.50 there's a caller-ID display for sale:

This evidently has about 20 digits of 7-segment reflective LCD plus some specific indicators, and a compartment for two AAA batteries.

For US$19 there's a cordless phone with a backlit LCD (which the vendor fraudulently claims is an LED display), an Alcatel Versatis E100:

This gets you a phone keypad with some menu buttons, a handset for voice communication, a rechargeable battery (supposedly 400-mAh and 7-hour talk time) and recharging base, what looks like a 12-character 14-segment alphanumeric display, and of course the short-range license-free radio, probably 900MHz with a 6kHz bandwidth.

Other radios

Someone is selling a Motorola one-way pager for US$8 with what looks like a 20×4 character dot-matrix character display:

I'm not sure if text pager service still exists (and neither is the seller), but presumably this device comes with keyboard and screen intact, and Motorola pagers were famously durable. It probably doesn't have a rechargeable battery, but rather a AAA battery carrier.

There are various walkie-talkies on sale (called "handy" apparently), like this Baofeng BF-T12 for US$10 from DigitalStore:

This is a push-to-talk walkie-talkie transmitting on 16 25-kHz channels of the 400–470 MHz band, with a 1500-mAh USB-rechargeable battery that claims 8-hour battery life, a two-digit 7-segment LCD, earphone and microphone jacks and a volume knob,

claiming a 6–10 km range in the country, or 1–5 km in the city.

In the US, using most of this band requires operator licensing or is even reserved for emergency services, although two of the channels are FRS channels (up to 500 mW, while this radio is supposedly 2 W), but 95% of users ignore this, even though they're potentially subject to 5-figure fines. Also the US FCC restricts these radios to transmitting on 12.5-MHz channels since 2013 ("narrowbanding"). Here in Argentina at least some of this band is exclusively assigned to individual licensees but I'm not sure what the legal status of the rest of the band is.

MP3 players

Someone's selling what seems to be a 2GB S1 MP3 USB stick for US$8:

This is about the same price it went for in 2007, the last time I bought one. It has a reflective dot-matrix passive-matrix LCD screen with on the order of 128×32 monochrome pixels, 2 GB of storage, an iPod-style conductive volume control, four buttons, and a USB A female connector to plug it in with. If I'm right that it's an S1 MP3, there are a couple of open-source firmwares for them (SourceForge, the former s1mp3.de, Wladston's former s1mp3.org) that added features, so you might be able to program it instead of rebraining it. But they've been abandoned since 2009, and the CPU is just a 24-MHz Z80.

Someone else is selling their SanDisk Sansa 4GB MP3 player for the more reasonable price of US$3.30; it has a MicroSD slot and a backlit LCD that looks to be something like 320×240... and a dead battery:

There are also "MP4 players" that have, typically, larger backlit color screens for viewing videos, which have apparently not been totally replaced by cellphones; some cost US$30 or more and have Bluetooth, FM radio transmission, and so on.

Plain screens

You can get a used "digital picture frame" for US$23, basically a backlit 7-inch color LCD with what looks like resolution of about 640×480:

Presumably this sucks battery like nobody's business and is hard to rebrain.

Measuring instruments

MercadoLibre has an entire category of "balanzas para valija", suitcase scales, like this US$3.30 specimen:

This includes a large, easy-to-read 3-digit 7-segment LCD, three buttons, and presumably some kind of strain gauge and battery.

There are similar displays on digital meat thermometers (US$2.50), digital calipers (US$10), and digital countertop scales (US$10).

There are also indoor/outdoor digital weather thermometers with LCD displays like this 3-digit 30mm-tall TPM-10 for AR$600 (US$4) which tend to have fairly large displays:

Some of them are larger and have more digits. And the temperature sensor itself is potentially useful; simply rebraining such a thermometer without drilling any further holes could give you a logging Wi-Fi thermometer.

Clocks and watches

One digital alarm clock sells for US$3.40 and also has an LCD display with quite a number of 7-segment and 14-segment characters.

And for larger digits there's a 45-mm-tall LCD alarm clock Planeta Zenok sells for US$6.50, with somewhat fewer digits:

For US$2 you can get a MegaCuper LCD calculator watch, though most competitors are much more expensive:

Remote controls

There are lots of remote controls that include a shitty keyboard, an infrared LED for communication, and a reflective LCD display, mostly those for air conditioners; for example, those of the LG split air conditioner Akb73756210 (US$5), the LG BGH Arcool Ar809 air conditioner (US$5), the Samsung AR807 air conditioner (US$4.50), and the Sanyo Y512f2 air conditioner (US$4.50). The LCD displays typically have a few 7-segment or 14-segment digits or character positions and a bunch of special-purpose indicators: fan, snowflake, and so on.

Topics

Oscilloscope superresolution via compressed sensing?

Kragen Javier Sitaker, 02020-11-17 (1 minute)

Can compressed sensing make a better oscilloscope?

The STM32 has a 1Msps 12-bit ADC, and there are oscilloscope projects using it. But a decent oscilloscope has at least 20MHz of bandwidth, and the Miniscope has 461kHz, so it's about 2.3% of a decent oscilloscope.

Actually though the STM32F103C8T6 used in that project and in the Blue Pill has *two* such ADCs. If you apply them both to the same input, though, you won't get any more information because (IIRC) they sample in sync. This is ideal if you're trying to measure the voltage-current characteristics of some device but suboptimal if you want to measure a single signal faster. You could perhaps put an analog filter on one of the inputs in order to phase-shift some signal components.

But what if you can fire the ADCs, or an external sample-and-hold circuit feeding them, at effectively random times? Then you could sample the signal in a time-domain basis that's incoherent with respect to its frequency-domain content. Then maybe you could use a standard ℓ_1-minimizing basis-pursuit algorithm to look for a sparse frequency-domain signal that explains what you saw?

It seems like that might be enough to get you an effective 20MHz or so of bandwidth, though of course only for signals that really *are* sparse.

Topics

A solar panel from an LED garden light

Kragen Javier Sitaker, 02020-11-17 (updated 02020-12-01)
(5 minutes)

There are these solar garden lights for sale for US$1.15 or so which contain a Ni-Cd battery, a monocrystalline solar panel to recharge it, an LED, and a bit of circuitry to turn the LED on only at night. The battery in one of mine had died, so I took it out long ago, so I just thought I'd do a little poorly-controlled MPPT measurement in a spot of sunlight that comes in through the window here.

I disconnected the panel from the circuitry and hooked it up to some resistors instead: 117Ω, 156Ω, 454Ω, and 988Ω, measuring respectively 0.60 V, 0.75 V, 1.91 V, and 2.09 V across the loads, and 2.25 V at open circuit (loaded only by the meter), giving respectively 5.1 mA, 4.8 mA, 4.2 mA, and 2.1 mA, and 3.1 mW, 3.6 mW, 8.0 mW, and 4.4 mW, suggesting that in this partial sun the panel's maximum power point is somewhere around 8 mW.

The panel is about 38 mm square, so in full sunlight it would receive about 1.4 W of irradiation, and at the 21% efficiency expected for monocrystalline panels it could produce 300 mW. So 8 mW is pretty low, and I should maybe repeat the experiment while actually going outside and getting direct sunlight — maybe the spot of light through the window was dimmer than direct sunlight by an order of magnitude or more. It's also possible that this isn't really a monocrystalline cell but rather an amorphous cell. Either way, it was nice to get this 8-mW lower bound, because that's already enough for a pretty decent personal computer these days.

(I was using one of the shitty multimeters mentioned in the note on multimeter metrology (p. 502), which ought to give all these figures an error somewhere around 1% or a bit lower.)

Even when loaded by only the meter, the panel's voltage sagged when in indirect sunlight, down to hundreds of millivolts, suggesting that it would not be usable for indoor energy harvesting.

Trying to convert the above to a Thevenin equivalent gives internal resistances for the panel of 320Ω, 310Ω, 82Ω, and 83Ω respectively, suggesting that either the sunlight conditions changed as I was taking readings or the panel is very far from being ohmic.

As a rebraining (p. 597) candidate the garden light is somewhat appealing: it has a built-in energy-harvesting power source, and the cylinder containing it, the battery cartridge, and the electronics is mostly empty, easy to open (three screws), 30 mm high, and 70 mm in diameter. This provides lots of potential space for stuffing electronics into. What it lacks is much in the way of I/O devices, possessing only a white LED.

A potentially more appealing approach is to carefully remove the PV cell and graft it into something else, maybe something easier to carry.

In really full sun, I got 2.47 V open-circuit and 2.10 V across a

105Ω load, thus 20 mA, 42 mW. This suggests I'm actually not at full max power and could benefit from going to a lower load impedance: the load voltage is more than half the open-circuit voltage. If the panel had an ohmic internal resistance dropping 370 mV at 20 mA, it would be 18.5Ω.

However, I think we can be reasonably sure that, although that "internal resistance" won't remain constant, it won't go *down*. Which means that the maximum power output available in this sunlight would be (½ 2.47 V)²/18.5Ω = 82.4 mW. So I can squeeze at most another factor of 2 out with lower impedance, so this panel is at best 5.7% efficient. It must be an amorphous panel, not monocrystalline like I thought.

On MercadoLibre there's a 22mW 22mm×7mm panel for US$4 (crystalline) and a few 1000mW panels in the range of 110mm×60mm for US$3 or so (also crystalline). But there are also 310-watt panels for US$82, which is US$0.25 per peak watt rather than US$3. And there are (supposedly) solar calculators under US$1, but those have much smaller cells (when they're not totally fake). So I think this is largely a question of transaction costs, and these garden lights are probably the cheapest way to get solar cells in the 40–60mW range.

Topics

Representing E12 electronic component values musically

Kragen Javier Sitaker, 02020-11-17 (updated 02020-12-26)
(16 minutes)

Resistors, capacitors, and sometimes even inductors are conventionally manufactured to have a set of "preferred values" in each order of magnitude: the E3 series, 10 22 47; the E6 series, which intercalates 15 33 68; the E12 series, which additionally intercalates 12 18 29 39 55 82; and longer series. So it's common to see a 220-ohm resistor or a 220-µF capacitor, but you'll very rarely see a 200-ohm resistor or a 200-µF capacitor.

Despite appearances, these values are fairly evenly spaced — just in exponential space. $82 \div 68 \approx 1.206$; $29 \div 22 \approx 1.318$; $18 \div 15 = 1.2$; and so on. The intervals all approximate $10^{1/12} \approx 1.2115$.

Component identification — knowing whether this resistor you have is a 2k2, a 22k, or just broken — is a big challenge for salvaging electronics. Especially for us old colorblind humans. Usually it's good enough to know which E12 value something is. Microcontrollers that can easily distinguish components are easy to come by, but getting their output into a useful form requires some kind of screen (p. 584), and those are comparatively rarer and more finicky to interface with.

The humans can easily distinguish notes of the 12-tone equal temperament scale traditionally used for Chinese music (though this is far easier when presented as intervals rather than as bare notes), and there's a pleasing perceptual correspondence between the 12 tones in an octave and the 12 E12 values in a decade. And speakers are cheap and easy; sometimes, as in surface-mount MLCCs using piezoelectric X7R dielectrics and similar, they're even included by accident. Auditory output also has real advantages for high temporal resolution that the humans can perceive.

So I propose that a component-identifying smart-tweezer program should map measured component values to the musical scale in this way and alternately play a couple of notes through a speaker, separated by the appropriate interval. One of the notes can be something like a flute playing a standard frequency, like 261.62557 Hz for A440 middle C (C_4 except in MIDI), to give a point of reference, while the other plays after it as a string instrument or something, with each decimal order of magnitude of component values mapped to one musical octave. That is, represent them with a fundamental frequency of $k\, 2^{\log_{10} v}$, where v is the value to represent and k is some proportionality constant.

Psychoacoustics

The humans can easily hear the eight octaves from 32.70 Hz up to 8372 Hz, while the next octave and a half up is mostly audible only to larval humans; the next octave down from 32.70 Hz to 16.35 Hz is audible in its pure form only if it is very loud, but if provided with

rich harmonic content, it is clearly comprehensible even when the fundamental is too low to hear or even not reproducible by the speakers. Their hearing sensitivity is about 1 Hz for complex tones up to 500 Hz and about 0.6% above 1000 Hz; between 1kHz and 2kHz their perception of frequency is least distorted by amplitude, while between 2kHz and 5kHz they are most able to detect sounds. This suggests that it's probably best to stick to the lower octaves as much as possible, even down below 20 Hz, since their harmonics will populate the most sensitive regions of hearing more densely.

However, we start to lose relative precision once we go below 500 Hz; at 500 Hz the frequency precision is about 0.2%, at 250 Hz only about 0.4%, at 100 Hz only about 1%, at 65.41 Hz about 1.5%, and at 20 Hz only about 5%. After being transformed through the inverse of the representation function above, these perceptual imprecisions are respectively 0.7%, 1.3%, 3.4%, 5.2%, and 14%. At higher frequencies errors climb again but not so high.

Resistors, with high resistances at high frequencies

Resistors generally used by the humans range from 1Ω to 1MΩ, six orders of magnitude apart, with most being in the 100Ω–100kΩ range, only three orders of magnitude apart. Generally low-value resistors are used for precise measurement (linear conversion between voltage and current), and current limiting, while high-value resistors are used for less-precise applications like pullups, pulldowns, capacitor bleeders, and protection. This suggests maybe positioning 1Ω around 110 Hz (A_2, A below small C or A above deep C), giving the following correspondences:

- 10 mΩ: 27.5 Hz, A_0, lowest key on a normal 88-key piano
- 100 mΩ: 55 Hz, A_1, A below low C
- 1 Ω: 110 Hz, A_2, A below small C
- 10 Ω: 220 Hz, A_3, A below middle C
- 100 Ω: 440 Hz, A_4, A above middle C
- 1 kΩ: 880 Hz, A_5
- 10 kΩ: 1760 Hz, A_6
- 100 kΩ: 3520 Hz, A_7, highest A key on a normal 88-key piano
- 1 MΩ: 7040 Hz, A_8

This of course gives $k = 110$ Hz.

This assignment is a compromise between not "wasting" the lowest octaves on little-used low resistances that require Kelvin probing to measure accurately, assigning the best precision to values in the 1–100 Ω range where it often matters the most, and not assigning megohm values to totally inaudible frequencies.

Resistances above a few megohms might be best represented by some additional gimmick, like using a different musical instrument.

Capacitances, and a better resistance scale

Capacitances are trickier because they span a wider range; common capacitors are in the 47 pF to 470 µF range, though up to 22000 µF is not unheard of — though anything above 1 µF probably isn't very

precise, because the piezoelectric and electrolytic technologies used at those higher capacitances aren't very precise. (And then there are supercaps, up to 100 F — that is, 100 000 000 000 pF.) In the 47 pF–4.7 μF range, though, we have only five orders of magnitude, which seems quite manageable.

It's unclear whether to use high frequencies for high capacitances (like, microfarads) or low capacitances (picofarads). Arguments in favor of using them for *low* capacitances include:

• In real life smaller capacitances do things at higher frequencies. 0.01 μF is 1 kΩ at about 16 kHz; 100 pF doesn't drop to 1 kΩ until 1.6 MHz. Whether in an LC tank, an RC oscillator, or an RC filter, using a smaller capacitor means your frequencies go up.
• Also, high capacitances tend to be physically larger, and, psychologically, larger things should make deeper sounds, while tinier things make tinnier sounds.

Arguments in favor of using high frequencies for *high* capacitances include:

• Precision is more important for low capacitances, which are largely used for tuning things, than for high capacitances, which are more used for bypassing things. If we put 47 pF at 880 Hz so that perceptual precision is best around 100 pF, and go down from there, then at 0.047 μF we're already at 110 Hz, and by 4.7 μF we're at 27.5 Hz. What on Earth would we do with 470 μF? The noise of an idling motorcycle without a muffler, at 6.875 Hz? And we'd be wasting a couple of audible octaves that correspond to capacitances too small to measure reliably under ordinary conditions, because they're swamped by parasitics.
• You'll be able to transfer your numerical ear learning from resistance to capacitance more easily. ("Oh, that C# is 2.2kΩ. That means it's 0.022 μF.")
• Higher resistances move RC oscillators and filters to higher frequencies too, so what gives? (Higher resistances move RL circuits to lower frequencies, but nobody uses RL circuits.)
• The difference in response frequencies of capacitors is vastly understated by the frequencies you hear.

The rebuttal arguments in favor of using high frequencies for low capacitances include:

• Maybe high resistances should be represented as low frequencies too, both because they slow down RC circuits and because we think of higher resistances as being "larger", so maybe they should speak in deeper voices too? (Also, *really* high resistances, like 10 MΩ and up, tend to be used with high voltages, so they *are* physically large, both for power dissipation and creepage allowance. Still, lots of physically large resistors I have here are 10-Ω current-sensing shunts or 25-Ω heating elements, not multi-megohm high-voltage protection resistors.)

So I propose this scale for capacitances:

• 10 000 μF: 1.71875 Hz, A_{-4}, 103 beats per minute, *moderato* or *allegretto* musical tempo, hunt-and-peck 17 words per minute typing

speed; largest common electrolytic capacitor
• 1000 μF: 3.4375 Hz, A_{-3}, 206 beats per minute, *prestissimo* musical tempo, beginner touch-typist 34 wpm typing speed
• 100 μF: 6.875 Hz, A_{-2}, 412½ RPM, too fast to be a musical tempo but too slow to be an idling diesel truck engine, respectable typing speed of 69 wpm
• 10 μF: 13.75 Hz, A_{-1}, Harley engine idling too slow at 825 RPM, near the 800-RPM bottom limit permitted by the stock EFI, risking running your starter battery down and also engine damage from insufficient oil pressure; 138 wpm, maximum human typing speed
• 1 μF: 27.5 Hz, A_0, lowest key on a normal 88-key piano; smallest common electrolytic capacitor; largest common film capacitor; largest common ceramic capacitor; microwave oven capacitor
• 0.1 μF: 55 Hz, A_1, A below low C; smallest common tantalum capacitor
• 0.01 μF: 110 Hz, A_2, A below small C
• 0.001 μF: 220 Hz, A_3, A below middle C; smallest common film capacitor
• 100 pF: 440 Hz, A_4, A above middle C
• 10 pF: 880 Hz, A_5; smallest common ceramic capacitor
• 1 pF: 1760 Hz, A_6, but you probably have parasitic capacitances larger than this; the manual for the famous AVR TransistorTester says, "Capacitors with value below 25pF are usually not detecte[d]."

So, for example, 120 pF would be about $G\#_4$, 415.30 Hz, one half-step deeper than A_4, because it's one step higher on the E12 scale; 150 pF would be about G_4 (392.00 Hz), and 220 pF would be about F_4 (349.23 Hz). These are approximate because the E12 scale is approximate: 120 pF would be more accurately 416.50 Hz, 150 pF more accurately 389.44 Hz, and 220 pF more accurately 347.03 Hz. So 150 pF, for example, is flat of G_4 by 11.3 cents, a difference audible to trained musicians and possibly somewhat painful, but hard for the untrained ear to detect, though probably possible in this region of best sensitivity. A perfect G_4 would be closer to 146.8 pF, an error of -2.2% from the nominal value.

Correspondingly, I propose this one for resistances, equating a microfarad to a megohm, a nanofarad to a kilohm, and a picofarad to an ohm, as if we were interested in an electrical signal of 160 kHz:

• 10 GΩ: 1.71875 Hz, A_{-4}, 103 bpm, *allegretto*
• 1 GΩ: 3.4375 Hz, A_{-3}, 206 bpm, *prestissimo*
• 100 MΩ: 6.875 Hz, A_{-2}, 412½ RPM
• 10 MΩ: 13.75 Hz, A_{-1}, 825 RPM
• 1 MΩ: 27.5 Hz, A_0, lowest key on a normal 88-key piano
• 100 kΩ: 55 Hz, A_1, A below low C
• 10 kΩ: 110 Hz, A_2, A below small C
• 1 kΩ: 220 Hz, A_3, A below middle C
• 100 Ω: 440 Hz, A_4, A above middle C
• 10 Ω: 880 Hz, A_5
• 1 Ω: 1760 Hz, A_6; at this point you need Kelvin connections for precision measurement
• 100 mΩ: 3520 Hz, A_7, highest A key on a normal 88-key piano
• 10 mΩ: 7040 Hz, A_8; at this point you need Kelvin probing for any measurement at all

Note that this also solves the problem of what to do with arbitrarily high resistances, although you have to be careful you aren't swamping the whole audio spectrum with harmonics from a spurious detection of a 100-GΩ resistor that's really leakage through your probe insulation or something.

It might be reasonable to do, say, a square wave for capacitance and a repeatedly plucked Karplus–Strong string (or two, slightly offset in frequency, like a piano) for resistance, so that you can play them at the same time if you're doing an ESR measurement. Using different envelopes will help the humans hear them as two separate sounds rather than one sound with a discordant timbre.

Inductors

Now we're faced with another consistency dilemma: do we represent large inductances with low pitches or with high pitches?

In favor of low pitches:

- Large inductances, like large capacitances, are used at low frequencies. A 60-Hz fluorescent light ballast might be an entire henry. A millihenry has a kilohm of reactance at 160 kHz; at 16 MHz you only need 10 μH to get to 1kΩ.
- Inductors with large inductances are physically large, so should ring deeply, like a church bell, not like a jingle bell. Also they tend to have heavy magnetic cores.
- It would be numerically consistent with the systems suggested above for capacitances and resistances.
- Of course in LC circuits increasing the inductance lowers the frequency.

In favor of high pitches:

- Steal underwear.
- ???
- Profit!
- In some metaphorical or Steinmetzian sense a large inductance is the opposite of a large capacitance.

I think the underwear gnomes lose this one.

If we try to use the same 160-kHz signal frequency to figure out where to set the equivalence, we get this:

- 10 000 H: 1.71875 Hz, A_{-4}, 103 bpm, *allegretto*
- 1000 H: 3.4375 Hz, A_{-3}, 206 bpm, *prestissimo*
- 100 H: 6.875 Hz, A_{-2}, 412½ RPM; largest common inductor
- 10 H: 13.75 Hz, A_{-1}, 825 RPM
- 1 H: 27.5 Hz, A_0, lowest key on a normal 88-key piano, fluorescent light ballast inductance
- 100 mH: 55 Hz, A_1, A below low C
- 10 mH: 110 Hz, A_2, A below small C
- 1 mH: 220 Hz, A_3, A below middle C
- 100 μH: 440 Hz, A_4, A above middle C, maybe an air-core coil for an RF circuit
- 10 μH: 880 Hz, A_5
- 1 μH: 1760 Hz, A_6

• 100 nH: 3520 Hz, A_7, highest A key on a normal 88-key piano; smallest common inductor
• 10 nH: 7040 Hz, A_8, typical axial lead parasitic inductance
• 1 nH: A_9, 14.08 kHz, out of my hearing range, a wire of 1 mm length and 1 mm radius

This seems reasonable.

It's probably worthwhile to use a different instrument again for inductor detection, perhaps an FM-synthesized bell sound or something.

Topics

Microcontroller inventory

Kragen Javier Sitaker, 02020-11-18 (updated 02020-11-28)
(4 minutes)

I have several microcontrollers: an Arduino Duemilanove with an ATMega328, a couple of Blue Pills (one with an STM32 and one with a CKS32; see notes on STM32duino (p. 530)), two or three ATMega8s, 18 surface-mount ATTiny45s, 19 DIP ATTiny45s, two loose ATMega328s, 16 ATTiny2313s, an Arduino Nano (or is that a Teensy?), two Raspberry Pi Zeros, a Raspberry Pi 2 Model B 1.1 from 2014, and a Raspberry Pi from 2012 which I think is a Pi 1 B. 66 computers in all.

But right now the Duemilanove is the only one I have successfully programmed!

ATTiny45

It's apparently straightforward to rig up Arduino to program an ATTiny45; you use the ArduinoISP sketch, wire up the ATTiny45 pin 1 (/reset) to Arduino pin 10, ATTiny45 pin 5 (MOSI) to Arduino pin 11, ATTiny45 pin 6 (MISO) to Arduino pin 12, and ATTiny45 pin 7 (SCK) to Arduino pin 13, and of course V_{CC} and ground on pins 8 and 4, and then I guess avrdude can program it. There's an updated Arduino support library for Arduino 1.6 at https://raw.githubusercontent.com/damellis/attiny/ide-1.6.x-boards-manager/package_damellis_attiny_index.json, but probably just avr-gcc is fine most of the time. And there's an Arduino Create Hub page about this too. For ArduinoISP you probably want to hook up a cap to stop the Arduino from resetting.

Luna also gave me a PC with a parallel port, which might work with avrdude without ArduinoISP.

The ATTiny45 is not super powerful and has only 8 pins, but it does have an ADC and support multiplexing the analog comparator between pins.

ATTiny2313

The ATTiny2313 is the first AVR I programmed actually, using a parallel port, using Limor Fried's minipov2. It doesn't have an ADC at all, and its analog comparator isn't multiplexable between pins, but it has a somewhat less cripplingly small number of pins: 17 GPIOs instead of 6, 18 if you push it.

Arduino support is available at http://drazzy.com/package_drazzy.com_index.json. You hook up pin 1 (reset) to pin 10 of the Arduino, just as with the ATTiny45, and similarly pins 17, 18, and 19 (DI, DO, SCK) to pins 11, 12, and 13 of the Arduino. (I'm guessing they're "DI" and "DO" because the 2313 doesn't have SPI hardware.) TinyDebugSerial is supposed to make the serial port work.

This last link also gives a somewhat mangled command line for using ArduinoISP or something called "TinyISP" directly from

avrdude.

Avrdude, when using ArduinoISP

```
avrdude -P COM20 -b 19200 -p t2313 -c avrisp
```

Upload using TinyISP

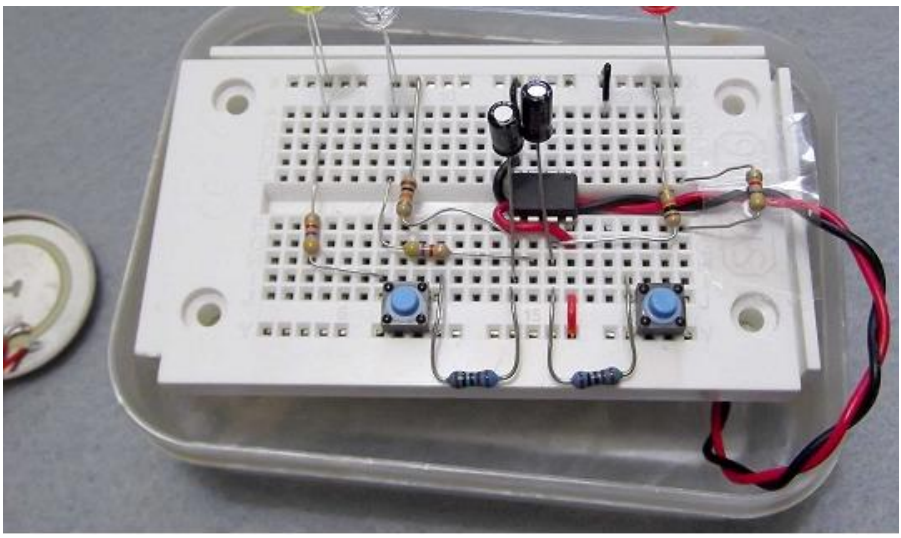
```
-p attiny2313 -c stk500v1 -P COM3 -b19200 -Uflash:w:Blink.hex:i
```

The best documentation for avrdude is still Limor Fried's.

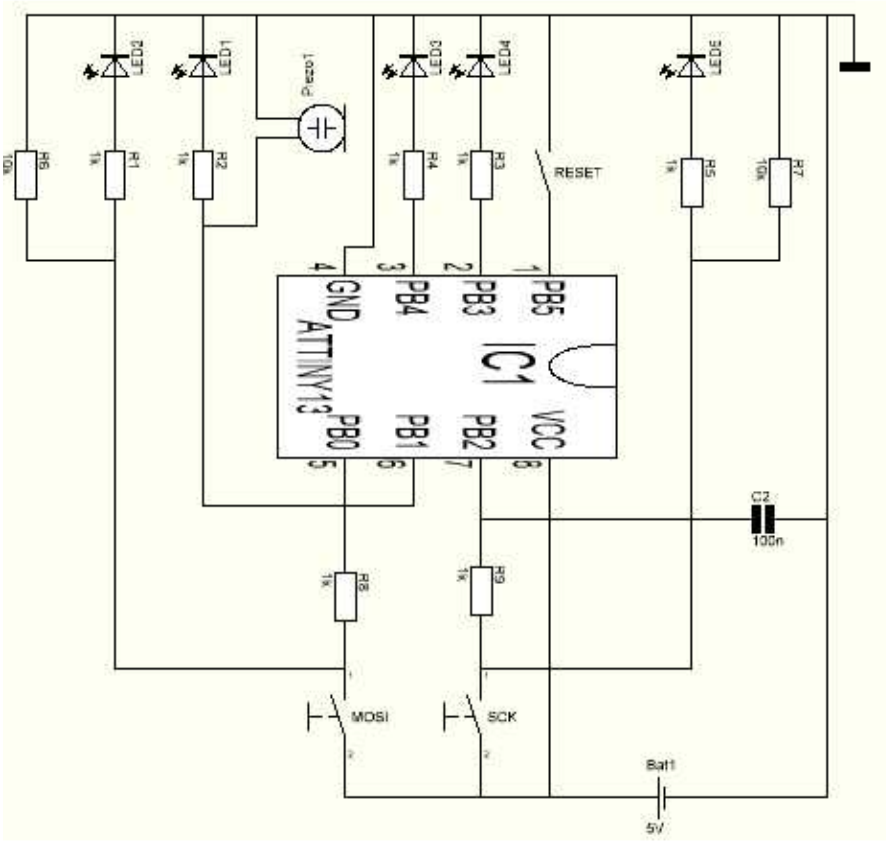
There are a few different options for serial data on these tiny machines actually. This is very useful because it provides a way to debug, and also to use these devices as sensors before having a screen to display the results on, and also a way to use them for continuous data acquisition or later download from EEPROM data. And because it's reasonable to use a crystal on it, you can meet RS-232 timing requirements!

Programming AVRs with pushbuttons

Heinz D. wrote an ATTiny13 tutorial in German which starts out by attaching switches to the reset, SCK, and MOSI lines of an ATTiny13 and programming it with them.



Here's Heinz D.'s schematic:



The ATTiny2313 datasheet has "serial programming timing" on p. 205:

Figure 22-6. Serial Programming Timing

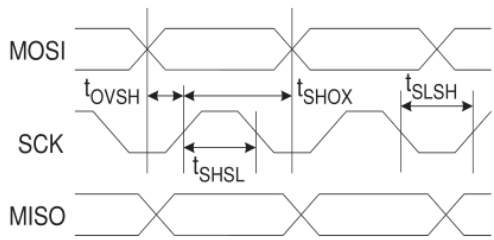
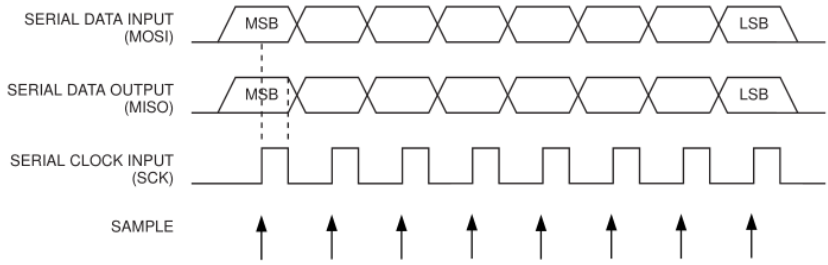


Figure 22-7. Serial Programming Waveform



This doesn't really explain how you get into serial programming mode, or what data gets input or output, though. It does mention the protocol a bit on p. 29: "If the RSTDISBL fuse is programmed, this start-up time will be increased to 14CK + 4 ms to ensure programming mode can be entered." So apparently you program it during reset? Surely I'll eventually find the full procedure.

Topics

Keyboard object environment

Kragen Javier Sitaker, 02020-11-19 (13 minutes)

I'm going through a list of part numbers and snarfing datasheets for them. I've written a googling script for this, but I still end up copying and pasting URLs, typing "wget " or "links " or "mupdf ", tab-completing filenames that were just output, and so on.

I feel like the ideal thing here would be to have my keyboard *focus* be on some *object*, such as an URL or a downloaded file, and have a menu of keystrokes for *operations* or for changing the focus. This doesn't have to be modal in the Bravo sense — the command keystrokes could all be control keys, for example.

The operations I'm currently going through are:

- search an existing collection of datasheets for a part number
- google datasheets for a part number
- look at the SERP results
- select an URL from the SERP
- run `links` or `wget` on it
- navigate in `links` and maybe save a file
- delete the resulting file
- open it in `mupdf` (faster) or `xpdf` (shows outlines but often has trouble with non-ASCII) or `evince` (handles Chinese in outlines better, and prefetches)
- rename it

It would be really nice to have an environment where I could implement new object types, or add new operations to existing object types, and have them immediately visible and instantly invocable.

For integrating existing programs like `links` or `catdoc` into the environment, it would be useful to run them inside a container so that the environment can observe what new files they create and reify them in the UI. But it might be good enough to just rescan the directory after they exit.

(You could implement such an environment with multitouch too, of course.)

A nested stack-based keyboard interaction design

The idea here is that you are standing on top of a linear "operand stack" of objects, some of which correspond to filesystem files, and you can use keyboard commands to invoke operations provided by the objects, or to navigate and manipulate your environment. Generally objects are not destroyed, just moved to a second "discard stack" from which you can easily recover them; nor are they duplicated.

The discard stack is displayed upside down above the operand stack.

The basic navigation commands are:

• ^N or downarrow: discard the top of stack by moving it to the top of the discard stack
• alt-^N or alt-downarrow: discard the object *under* the top of the stack by moving it to the top of the discard stack
• uparrow or ^P: recover the object at the top of the discard stack by moving it to the top of the operand stack
• alt-uparrow or alt-^P: recover the object at the top of the discard stack by moving it *under* the top of the operand stack.

It will be seen that these four operations are sufficient to move your focus to any object and to permute the objects arbitrarily, though inconveniently. You can also type to create new string objects or edit the text in existing string objects; if your focus is on something that doesn't handle text, the environment will create a new string object for you in this case. If your focus is on a string and you want to create a new blank string to type into, you can press Enter.

For more efficient structuring, you can create a new "frame" by typing ^O. Frames can contain other objects. They support some additional key commands:

• Alt-^F or alt-rightarrow: make the frame gobble up the thing below it on the operand stack, moving it from that operand stack to the bottom of the frame's operand stack.
• Alt-^B or alt-leftarrow: move the item at the bottom of the frame's operand stack into the current operand stack, under it.

These two provide a convenient way to gather together a group of related objects to move them together to some other place and deposit them there. But frames also contain a whole new operand stack and discard stack, a whole universe within a universe, allowing you to enter them and navigate around within them:

• TAB: cycle the frame's display on the stack between one-line, expanded, and fully expanded versions
• ^F or rightarrow: *enter* the frame to abide within it a while.

The environment also has a ^B or leftarrow command to get out of the current frame, returning to wherever it lives.

There is an important distinction here between modeless universal commands available everywhere — ^B, ^N, ^P, alt-^N, alt-^P, ^O, typing text, and a few others to be mentioned later — and per-object-type operations only available on a particular kind of object such as a frame.

So, so far we have, apparently, an outliner. But each object type can both affect our view of the stack and provide additional operations. The simplest case of this is provided by strings, which implicitly highlight matches for their contents when you are focused on them, slightly graying out other parts of the stacks that don't match them. But they also have an operation that hides non-matching parts entirely, toggled by the TAB key! And, when this is activated, the ^N, ^P, alt-^N, and alt-^P commands leap past the hidden items as if they weren't even there; so by creating a search string and activating it with TAB, you can fold space and ride the string as a swift vessel to your destination.

(Hmm, at this point I think it's probably better to have a separate

"kill frame" accessed with ^K and ^Y, so that you don't leave the search string beached at your destination. Then our two stacks merge into a single buffer that you move around in, rather than being two stacks.)

Such navigation is facilitated by further commands attached to alt-0, alt-1, and so on up to alt-9, displayed next to applicable items. In the case of a search, these just move your focus to the specified item, but other types of objects can instead use them for other purposes.

The datasheet example doesn't involve ever combining two objects together in a single operation, so it is not a good source of such operations; my original example in Dercuano was numerical calculation. In the datasheet example, though, it's just fanout: string search term → Google SERP list → choose a link → open, rename, or delete the file. You might choose the same link more than once, so you probably don't want the links to get consumed when you invoke operations on them (like "open in `links`").

Actually, maybe there is an opportunity: represent the SERP as a sequence of 14 or so items, the first few of which are "wget", "links", "firefox", and "close SERP", and the others of which are the various URLs:

```
wget <-
links
firefox
close SERP

http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-0896-FPGA-AT40K05-10-20-40-o
oDatasheet.pdf
https://www.microchip.com/AT40K20
https://datasheetspdf.com/datasheet/AT40K20.html
https://datasheetspdf.com/pdf/865245/ATMELCorporation/AT40K20/1

https://www.mouser.com/ProductDetail/Microchip-Technology-Atmel/AT40K20-2BQJ?qs=No
ob%252BVlYz%2F1KKCyOZ25ZvByg%3D%3D
https://www.digikey.com/number/en/microchip-technology/150/AT40K20/9619
http://www.digchip.com/datasheets/parts/datasheet/054/AT40K20-2AQC.php
https://www.trustedparts.com/en/part/microchip/AT40K20-2BQJ

https://www.application-datasheet.com/pdf/microchip-technology/519015/at40k20-2aj
oi.html
```

So initially your focus will be on, say, "wget", and you can just select a link with alt-0 or whatever to invoke "wget" on the first link, creating another item for the file which you can then open with, say, "mupdf" or "xpdf". The "wget" object isn't just a string; it knows that only URLs are acceptable fodder for it.

But it might be just as reasonable to give the URL objects methods for wget, links, etc., that are invoked with separate keys. That's not a reasonable alternative for numerical calculation, though.

Numerical calculation with units

Suppose you have some length on top of the stack, 3.1 meters or

something, and you invoke its multiplication operator, which pushes a curried multiply onto the stack. Well, you can multiply 3.1 meters by any quantity: 5 newtons, or 2 meters, or 14.5 square centimeters, or just 3. But not by, say, "your mom". So all the quantities (and none of the strings) on the stack are highlighted and given numbers; you can either use alt-5 or whatever to select the thing you want to multiply by, or you can navigate the multiply up or down to the appropriate place, or in and out of frames, or whatever. And then you can invoke it with whatever operand, the multiply operator disappears, and you get a product.

But if you invoke the addition operator instead, then only other lengths are fair game for addition, and 5 newtons and 14.5 square centimeters are grayed out. Just like your mom.

One thing I really like about this design is that you can very reasonably pop up menus of other things to do with your quantity. You can see a length represented in all the common length units, for example, and if you select one the list of units disappears and the displayed representation changes. And you might reasonably include actions like changing the plot color or numerical display format of a value, as well.

The focused element having the opportunity to change the display of everything else offers a substantial set of possible flexibility. In a numerical calculation, one obvious thing to include is the gradient, but also you can imagine various kinds of objects that change the view in various ways, perhaps previewing effects they could have if invoked.

Quasimodal operator keys

Hmm, if you really want to minimize calculation keystrokes, you could make + or - or * or whatever *quasimodal*: when you press it, the reified operation appears in the buffer and its intended target is highlighted, and if you want to apply it to something else you need to select from a menu or start navigating, because if you don't, it gets applied to the nearest applicable target as soon as you release the key. But I think that's a *different* interaction design.

The preview principle mentioned above would suggest that in such a system, the default second target would have preview operation results displayed next to it. So if you have three numbers with the third one focused, you might have a display like

```
-6
 3  +: 7 -: 1 *: 12 /: 1.33
 4
```

and if you start pressing the * key, you might see

```
-6 0: -24
 3     12
 4      *
```

and if you then release * without pressing 0, then you'd send both the 3 and the 4 to the kill list and end up looking at

```
-6  +: 6 -: 18 *: -72 /: -½              3
12                                       4
```

If you instead pressed o before releasing *, 3 survives, -6 and 4 get killed, and you end up looking at

```
  3  +: -21 -: -27 *: -72 /: -8          -6
-24                                       4
```

If instead of having a kill list you just have the two stacks as I initially suggested, the "killed" numbers might just move down and be unhighlighted:

```
  3  +: -21 -: -27 *: -72 /: -8
-24
 -6
  4
```

A flat list

Note that the ^K/^Y formulation suggested above with a separate "kill frame" eliminates the need to have arbitrary frames and frame navigation to reorganize things easily; you can just vacuum up all the items you want to move to a particular part of the list and then truck them all over there at one in the kill frame. This is probably simpler than the outliner and dual-stack approach.

A filesystem

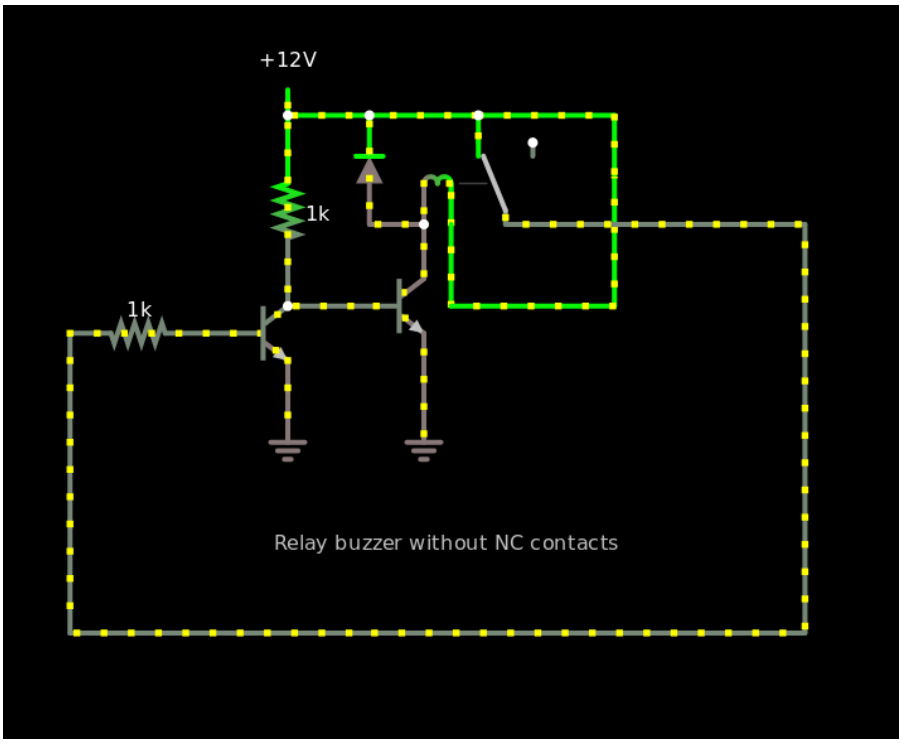
You could imagine applying such a formulation, either with nested frames or with just a working list and a trashcan list, to the problem of a filesystem for a small computer such as a pocket calculator — rather than giving your files names, you might give them positions in the list. This might include executable files — which perhaps can be simply frames, containing lists of steps (as in HP-48 RPL) or lists of methods annotated with control keys.

Topics

Relay buzzer

Kragen Javier Sitaker, 02020-11-23 (2 minutes)

As a test of mostly breadboading, I breadboarded this 12V relay buzzer from a couple of random NPN transistors and a microwave oven relay:




```
$ 1 0.000005 0.010109782498721881 50 5 43
R -48 64 -48 32 0 0 40 12 0 0 0.5
r -48 64 -48 176 0 1000
t -48 176 32 176 0 1 0.6207115498642259 0.6788785101220731 410
t -96 192 -48 192 0 1 -0.02761021024479937 0.6512682998772737 760
r -96 192 -176 192 0 1000
g -48 208 -48 256 0
g 32 192 32 256 0
178 80 128 80 80 0 1 0.2 0.019618688531384095 0.05 1000000 0.02 280
w -48 64 0 64 0
w 64 64 64 80 0
w 64 64 144 64 0
w 144 64 144 176 0
w 32 128 32 160 0
w 48 128 48 176 0
w 48 176 144 176 0
w 80 128 256 128 0
w 256 128 256 368 0
w 256 368 -176 368 0
w -176 192 -176 368 0
d 0 128 0 64 2 default
w 0 128 32 128 0
w 0 64 64 64 0
x -56 319 146 322 4 12 Relay\sbuzzer\swithout\sNC\scontacts
```

Falstad's simulation runs the relay at 200kHz by default, much faster than the relay I have, which is at about 100 Hz.

It was frustrating and slow to build the circuit because:

• I was making jumper wires by sanding varnish off coarse magnet wire with sandpaper between my fingers;
• I didn't know the pinouts of the transistors or the values of the resistors (actually I think one of them may have been 1k5Ω)
• probing the breadboard with probes that don't fit was pretty inconvenient;
• I broke a leg off Q2 during the process, and that took me a while to figure out;
• a multimeter with no audio is a slow way to check out the circuit.

However, once I had it all connected and checked out with the multimeter, I plugged it in and it ran. This was satisfying.

Topics

Geomagnetic energy harvesting is barely feasible at near-kilometer scales

Kragen Javier Sitaker, 02020-11-24 (3 minutes)

Variations in the geomagnetic field penetrate quite deeply into the earth and sea and may be a feasible energy-harvesting source, especially in the case of another Carrington event.

But how feasible is it? The Halloween geomagnetic storm of 2003 provoked planetwide variations of some 10 μT, out of a total 25–65 μT, with larger variations toward the poles, while typical daily variation is about 25 nT, "with variations over a few seconds of typically around 1 nT".

A rate of change of 1 nT (1 nWb/m²) per second is one nanovolt per square meter inside a single-turn inductive loop; even if we have 1000 turns, that's still only a microvolt per square meter. You could imagine stepping that up to a usable 0.4 volts or so with a chain of transformers; three 80:1 step-up transformers would probably serve. If you were trying to get 1 μW, which is a challenging but achievable level for modern energy-harvesting machinery to survive from, you'd need 1 A m² at the 1000 turns level or 1000 A m² at the 1-turn level.

One problem is that you can't get an arbitrarily large amount of power out of an inductor in a varying magnetic field just by winding more turns around it; at some point the current that's being induced will cancel out most of the magnetic field that would otherwise exist, and you'll stop getting more power.

It turns out that the energy of the magnetic field is $\frac{1}{2}B^2/\mu$, so in empty space the difference between 30 nT and 31 nT is 24 pJ/m³, and we probably can't capture more than half of that for impedance-matching reasons, so we're probably limited to a few picowatts per cubic meter. (I don't think using higher-permeability materials helps here; the μ is on the wrong side of that fraction.)

A further complicating factor is that, if you're using conventional conductors, you probably need to use ridiculously thick wires. Suppose your primary coil is 1000 m², the size of a big-box store façade, and you're getting 100 μV out of 100 turns around it. (Remember, unless you're near the poles, it has to be oriented north-south.) To get 1 μW you need 10 mA, and if you want no more than 90% of the energy to be lost in heating the primary coil, you need a voltage drop of no more than 90 μV, 900 μV per turn. So your 130-meter-long coil needs to be no more than 90 mΩ per turn, which requires 3-gauge copper wire (`~awg(2*~circlearea(130 m / copperconductivity 90 milliohm))` in units(1) gives about 3.3), which is is normally used for carrying 150 amps or more and costs abut US$3 per meter.

(I'm not entirely sure but I think you might need to enclose a larger area to grab enough energy. This helps a little with the wire thickness because you can enclose a larger area per unit length of wire, but the

wire is still ridiculously thick.)

Topics

Lava time capsule

Kragen Javier Sitaker, 02020-11-24 (8 minutes)

I was just watching a Long Now Short which described the work of Tobias Kestel and Florian Puschmann, who flouted sacred taboos in 02009 to put time capsules into lava at Kilauea.

It occurred to me that, although putting time capsules into lava is a straightforward procedure (though one with substantial risk of death), making a time capsule that could survive within the lava is much less so. Tin–lead solder melts at 187°, paper and organic polymers char in the 200°–400° range, lead itself melts at 327°, aluminum melts at 660°, ordinary soda-lime glass softens around 900° depending on the precise mix, gold melts at 1064°, copper melts at 1085°, but Kilauea erupts at 1170° and has lava tubes of 1250°. Silicon semiconductors exposed to such a temperature for more than a few seconds will suffer degradation from dopant diffusion, and their aluminum or copper conductors will melt and may bead up.

One obvious approach is to to use fused quartz, which doesn't soften until 1600°. But a possibly more interesting approach is to insulate the time capsule so that it can survive until the lava cools; by sheathing it in a thick layer of refractory firebrick, the heat flux through its surface can be reduced, and by including some phase-change thermal mass within it, the temperature can be maintained at a level that doesn't damage the payload.

The most obvious candidate is water, but as I noted in Desiccant Climate Control (p. 489) and Muriate Thermal Mass (p. 459), alabaster and muriate of lime also offer substantial energy densities; alabaster in particular has the advantage of forming an extremely cheap layer that can serve as insulation once calcined, surviving in solid form until 1460°, at which point it outgasses vitriol (see the note on plaster foam (p. 453).) However, water's enthalpy of vaporization is 2.26 MJ/kg, and its boiling point is only 100° at normal pressures.

It's also important that the time capsule be small enough to be covered entirely by lava, and dense enough that it doesn't float in the lava. This is in tension with the requirement for insulance; foaming things to improve their insulance makes them less dense, and plaster of paris is not very dense to start with (I'm finding conflicting figures of 0.7–2.6 g/cc, but I think pahoehoe is about 2.7–3 g/cc). A reasonable way to resolve this is to counterbalance the less dense alabaster with something denser inside the capsule, like steel (7.9 g/cc, US$1.06/kg), lead (11.34 g/cc, US$2.20/kg), or zinc (7.1 g/cc, US$2.76/kg). I think ʻaʻa may have higher gas content and consequently be lighter.

If we figure on a nominal lava density of 3 g/cc, steel gives us -4.9 g/cc of effective weight (4.9 g/cc of buoyancy) and lead gives us -8.3 g/cc, at prices of respectively US$8.37/liter and US$24.90/liter, giving us prices per effective weight of US$-1.71/kg and US$-3.01/kg respectively. That is, to compensate for a kg of buoyancy due to the low density of alabaster, you'd need US$1.71 worth of steel or US$3.01 worth of lead.

Lead has the potential advantage that it melts at 327° and thus forms an extra protective phase-change thermal mass, though it would be a poor tradeoff for water: 4.77 kJ/mol at 207.2 g/mol gives 23 kJ/kg, 100 times less than what's needed to boil water. However, it has the advantage that, unlike water, it stays in the capsule after changing phase — so it can resist not only the initial injection event but also potential subsequent reheating events. But its melting point is too high to save paper, phenolic circuit boards, or ordinary solder joints.

It's probably also important to protect the insulation from dissolving in the lava — depending on its composition, the lava might react with it. For this purpose it may be best to can the entire time capsule, insulating layer and all, in something that can withstand the lava; a steel can is an obvious choice, since steel is pretty much good to 1400° and doesn't dissolve in lava. But then there's the question of how the steam will escape through the can; it needs to be porous enough for the steam to get *out* but not so porous the lava will get *in*, and moreover it needs to not clog with hardening lava (too much) or the steam will build up inside.

The steam bubbling out through the lava may be sufficient to enlist some of the lava around the capsule as additional insulation, and may also be sufficient to keep open an air passage to the surface, permitting the penetration of circadian air pressure variations (for example, the tidal swings) until, at least, the next lava flow. 'A'a may be more favorable in this sense as well, since much of it cools as open-cell foams; very commonly a lava flow will have a top surface of porous foams such as scoria over a less porous base layer, and it may be ideal to optimize the capsule's specific gravity to float at or near the bottom of the porous layer.

Within the time capsule you can imagine a computing system that communicates through the lava, for example using AM radio (a big loopstick antenna can not only propagate waves through the steel can but also harvest energy from broadcast radio stations as long as those exist) or piezoelectric vibration.

I've previously looked around for mass-market archival media and found that nearly all current electronic storage (Flash, FRAM, MRAM) is only designed for 10-year data retention, occasionally 100-year. So such a time capsule would need an energy source to fight data loss with.

Aside from the obvious, but difficult, approach of using a betavoltaic battery, and aside from energy harvesting from AM radio as mentioned above, other candidate energy sources include: daily thermal cycling; vibration; air pressure variation, like the Atmos clock; or some other source.

I thought about a geomagnetic energy-harvesting system, but it seems that it would either need to run at much lower powers than are currently feasible, only run during violent geomagnetic storms, or be many orders of magnitude larger than is feasible to sink into flowing lava; see Geomagnetic energy harvesting (p. 631).

How thick would the insulation need to be? It depends on how long the lava flow takes to cool down to a temperature where the water stops boiling. Suppose arbitrarily that our insulation is

200 mW/m/K, we have 1000 kg of water for the 1200° lava to boil off, and our surface area is 10 m². Well, `(1000 kg 2.26 MJ/kg) / (1100 K * (200 mW/m/K) * 10 m^2)` gives us about 1.03 million seconds per meter, so if our insulation is 1.03 microns thick, we only last a second, while if it's a meter thick, we last almost 15 days. So to last an hour we need 3.5 mm of insulation, to last 3 hours we need 10.5 mm, to last 8 hours 28 mm, to last 24 hours we need 84 mm, to last 48 hours we need 170 mm, and to last a week we need almost 600 mm. (This is discounting the thermal mass and possible phase changes of the insulation itself, as well as all the thermal mass of the payload.)

I think this demonstrates that this design approach for a survivable lava time capsule is feasible but probably would not fit in your hand.

Topics

Lenticular air bearing

Kragen Javier Sitaker, 02020-11-24 (2 minutes)

A lenticular shape consisting of the intersection of two slightly-overlapping spheres, or a sphere and a half-space (the part cut off a sphere by a flat plane, a plano-convex lens) can be lapped smooth. So can the difference of a sphere (or such a lenticular shape) and a cylinder running through its center; so can a cylinder with two flat faces. These shapes, of which the lenticular shapes are the simplest, have rotational symmetry around a single axis of rotation. So by supporting one of them on an air bearing you can get an air bearing that resists movement in five degrees of freedom, rather than the three or four you get from a planar or endless-cylinder air bearing.

(In practice you probably want to trim the edges of the lens short of being a true knife edge.)

If you're using bearings in pairs rigidly joined by a shaft, as is normal practice, you don't need the lenticular shape; you can just use a sphere. Four bearing pads on a sphere (or lens) are sufficient to support it, but two bearing pads on each of two spheres joined by a rigid shaft — either all four compressing the shaft or all four stretching it — would also be enough. In many applications it would be desirable to tolerate a little axial misalignment in this way. In applications where the axial misalignment is accompanied by uncontrolled axial tension or compression, you might want to use four pads per sphere anyway, just in case the shaft gets put into tension when compression was expected, or vice versa.

It is not necessary that the shaft joining the two spherical sections be coaxial; the axis of rotation will run accurately through the centers of the spheres regardless of where the shaft is. Indeed, the "shaft" could even be a C-shaped thing that goes around the outside of the two trailer park girls. Uh, the two balls.

Topics

Machine readable microcontroller output

Kragen Javier Sitaker, 02020-11-26 (9 minutes)

Suppose I have a super minimal microcontroller that generates some data and I later want that data somewhere else. Like an ATTiny2313, say (US$0.67 in quantity 1 from Digi-Key, US$55 for 100, but not a smart buy in 2020). What are my options?

UART

UARTs can be used with serial ports where those are present. But typically they require a quartz crystal to meet the timing requirements of RS-232; even the ceramic resonators used on low-end Arduinos are insufficient. Lots of times I want to run off the ±10% RC resonators in these computers, which can be maybe trimmed to ±1%, or maybe not.

If you can get a well-controlled computer to send you some data first at what it thinks is an accurate baud rate, you may be able to examine its timing to recalibrate your clock.

But many modern computers omit serial ports.

SD cards

SD cards support an SPI interface through which you might be able to write files onto the filesystem. To deal with RAM as small as the ATTiny2313's (128 bytes!) you probably need to read the same sector from the SD card more than once if you're not sure where you're looking in it. This requires at least four pins (SCK, MISO, MOSI, /SS) and might be possible to pull off.

Then you can disconnect the SD card from the microcontroller and plug it into wherever you want to read the data.

Pretending to be an SD card

I'm not sure if this is practical, since SD cards normally support other interfaces that are hairier than just SPI, but maybe so. Maybe I should investigate further. Lots of computers have SD card interfaces on them now and they're easy to buy.

PS/2 keyboard

The AT/PS/2 keyboard/mouse interface is a two-wire open-collector bus with separate data and clock lines. (Plus +5V at up to 100mA and ground.) Chapweske reports success using PIC internal pullups and setting pins to output 0 to pull them low. Fortunately the *peripheral* always generates the clock signal, so in theory you ought to be able to use a weird clock rate, and in theory the clock speed should be 10–16.7 kHz. That is, 12.9 MHz ±23% — even the uncalibrated RC oscillator can beat *that*! And you have to check every 10 ms or less to see if you need to generate a clock to receive a host-to-peripheral communication. Other than that it's a

matter of sending 11-bit packets, each a byte with some start and stop bits.

So you could imagine acting as a keyboard and typing the data to a computer when plugged in. In theory you're not supposed to hotplug AT and PS/2 keyboards but I've never burned up a motherboard yet doing it.

There's even a PS2Keyboard library in Arduino from PJRC that already implements the protocol, but it's the other way around — it's so you can plug a keyboard into your Arduino!

Unfortunately many modern computers have replaced the PS/2 interface with USB, which is much harder to bitbang.

Low-speed USB bitbanging

The well-known GPL V-USB library supports bitbanging low-speed (1.5 Mbps) USB on "any AVR microcontroller with at least 2 kB of Flash memory, 128 bytes RAM and a clock rate of at least 12 MHz" and claims to even support being clocked from a 12.8-MHz or 16.5-MHz internal RC oscillator — but not the ATTiny2313's 7.3–9.1-MHz (??) internal oscillator. Also, it would occupy most of an ATTiny2313: "Only about 1150 to 1400 bytes code size."

And there's a similar project called 16FUSB for PICs.

Atmel's app note AVR291 on the ATMega32U4RC, which has USB hardware, explains that low-speed USB requires 1.5MHz ±1.5% (or, on p. 9, ±1%, but Silicon Labs says it's 1.5%), and that this precision is within the capability of the AVR family's oscillator calibration.

Bitbanging a USB interface offers the perhaps more appealing possibility of offering a USB mass storage interface.

Centronics parallel port, Raspberry Pi GPIO, and other GPIO

A Centronics parallel port like the fuchsia one on the back of this old tower here used to be common, but isn't now. But if you have one, it's easy enough to bitbang SPI over it, or over the GPIO pins on a Raspberry Pi, or an Arduino, or whatever. I mean that's how the ArduinoISP sketch works.

Speaker modem

If the amount of data to be transmitted is not too great (and in the case of the ATTiny2313 you can't store more than a couple of K on the chip; larger micros might have 32K or 168K or something; but more typical is maybe 128 bytes) then maybe you could use a really simple speaker modem, if the microcontroller can either transmit radio or has a speaker built in. It likely doesn't have to be able to travel over POTS phone service, so use of frequencies about 3kHz is fine; maybe frequency-shift keying between 3520 Hz and 5280 Hz would work, for example. Three cycles at 5280 Hz is 568.2 µs, and so is two cycles at 3520 Hz. If this were half the bit interval ("MSK"), this would give 3520 baud. Some parity bits would probably be

worthwhile. You could use a square wave and it would still work fine if the volume was high enough.

It would sound terrible, though. If you instead used 17000 Hz and 19125 Hz, which are still within the range of most microphones, you'd have 9 half-waves of one against 8 half-waves of the other, each taking 235 µs, giving 4250 baud. And it would be inaudible except to very young people.

If you packetized the data into packets with 1 header byte, 7 data bytes, and 2 parity bytes, then you'd get almost 372 bytes per second.

Although wireless, this approach would often work even more neatly if you could plug the microcontroller device into a microphone jack, like the Danger camera or the Square card reader. Typical microphone jacks provide a 5-volt "bias" or "PiP" or "IEC 61938" voltage on the ring electrode for electret microphone preamps. Reputedly this is typically in the 340 µA to 2.5 mA range because of a 2kΩ–10kΩ impedance. The Google Android device specification for the 3.5-mm headphone jack says the "mic bias voltage" should be 1.8–2.9 volts and that the "mic bias resistance" is "flexible" but later says something I don't understand about "2.9V mic bias applied through 2.2 kOhm resistor". That would be 660 µA and thus a maximum output power of just under a milliwatt.

Wirelessness and many-to-many nature may be advantages in some circumstances. You could imagine several sensors that all log data over the air in a room at random times or on demand, and one or more data loggers that demodulate that data and log it.

LED communication

The actuator most easily accessible to a microcontroller, apart from simple wires, is an LED. LEDs themselves typically support astoundingly high data rates, up into the megabits per second. But most computers that might be able to observe the LEDs cannot observe them very fast, perhaps with a camera running at 30 fps, 60 fps, or a slightly higher rate.

You might be able to get 2 or 3 bits of data per frame of video by modulating the apparent brightness of a single LED, but that's only about 180 baud in the best common case. At 180 baud a 128-byte message would take almost six seconds: inconvenient but workable for some applications.

If you're just after the "wireless" part rather than the "connecting to big computers" part, you could dragoon one microcontroller into feeding photodiode or phototransistor data into the big computer so it can demodulate it, after being transmitted by one or many small computers.

It's a shame IrDA worked so badly and was abandoned!

Topics

Muldiv

Kragen Javier Sitaker, 02020-11-26 (1 minute)

In Forth and some other contexts, there's a */ operation which multiplies by a ratio in integer arithmetic, avoiding overflow typically by using a double-precision intermediate product. So even if you're using a 16-bit Forth, `23082 7 8 */` should give you 20196, which only differs from the correct answer 20196¾ by ¾, rather than 3812, which is wildly wrong but what you would get if you truncated to 16 bits after the multiplication.

It occurred to me to wonder if you can do the division and the multiplication at the same time, if you're doing this in hardware, thus overlapping the multiplication with the division. Division can generate a quotient one bit at a time, and those quotient bits can be used to control a garden-variety shift-and-add multiplier.

Topics

AVR OSCCAL probably won't give you an FM radio

Kragen Javier Sitaker, 02020-11-26 (2 minutes)

Hmm, I just realized that the AVR family OSCCAL oscillator calibration register can adjust their internal oscillator frequency, normally 8MHz or 16MHz, by fine degrees. The 12th harmonic of an 8MHz square wave would be 96MHz, which is within the FM radio band (87.5 MHz–108.0 MHz). The maximum permitted deviation from a nominal carrier frequency is ±75 kHz, which would be ±780 ppm at 96 MHz, ±860 ppm at 87.5 MHz, and ±690 ppm at 108 MHz. So, if "fine degrees" is about 1900 ppm or less (0.2%), then I ought to be able to transmit at least impulses and square waves over FM radio in this way. In fact, since FM radio stations have about 100 kHz of bandwidth after demodulation, that would be sufficient to generate PWM audio by spending different percentages of the time at each different wave.

In the ATTiny2313 OSCCAL is 7 bits, selecting one of 128 frequencies in order to calibrate down from a nominal ±10%, so the situation doesn't look that great. Maybe a better plan is the thing I'd wanted to try previously: put a resistor on the AVR's V_{CC} pin and modulate its *current consumption* in order to affect its operating frequency an therefore the frequency of the waves generated.

Also there's an ominous note in the datasheet saying that the processor needs to remain in RESET while OSCCAL is being written.

On p. 200 the datasheet says that with user calibration the oscillator can range from 7.3–9.1 MHz, with ±2% error at the given voltage and temperature. On p. 230 we have a chart suggesting that OSCCAL is capable of moving the frequency anywhere from 3 MHz or so up to 12 MHz! It looks like an approximate exponential, too, suggesting that each step of OSCCAL is about 1.1% of frequency variation. So that idea probably won't work, unless there are two adjacent OSCCAL values that are anomalously close.

Topics

A field-programmable RTL array: a more efficient alternative to FPGAs?

Kragen Javier Sitaker, 02020-11-26 (updated 02020-11-27)
(11 minutes)

What if, instead of individual registered LUTs, you programmed a synchronous register-transfer-level machine whose individual units were more like an ALU, or a GreenArrays x18 core, than like a gate?

Inspirational computing systems

I'm thinking of the TECS NAND-to-Tetris Hack CPU, based on the DG Nova and the PDP-8, where individual bits in ALU instructions control various aspects of the ALU instruction; and about the kinds of register machines used for Bresenham line drawing and circle drawing (Appendix E of Ivan Sutherland's SKETCHPAD dissertation, republished as UCAM-CL-TR-574, describes these as "incremental computers" and proposes using them to draw general conic sections); and about the very simple register machines provided by AVR peripherals, for example auto-incrementing timers that are connected to digital comparators that automatically reset them, or analog comparators that can register a timer value when they observe a transition; or Babbage's Difference Engine, which tabulates a new value of an arbitrary polynomial on every clock cycle, simply by adding to each register the difference from the previous register; and so on.

In the Hack, the ALU field has 6 control bits, zx, nx, zy, ny, o, and no, and the ALU is basically just six muxes:

```
a <- zx ? 0 : x
b <- zy ? 0 : y
c <- nx ? ~a : a
d <- ny ? ~b : b
e <- o ? c+d : c&d
output <- no ? ~e : e
```

This gives you 64 ALU instructions, many of which are unimplemented in the official Hack simulator, including NAND, NOR, AND, OR, addition, two's-complement subtraction, two's-complement negation, one's-complement negation, abjunction, X+Y+1, and a number of others. The PDP-8's similar setup includes bits for bitwise rotation and byte swapping, which among other things makes division much easier to implement.

The FPRTLA

Now, in a "FPRTLA", instead of routing individual bits around your machine, you could route bytes, or words of 4-64 bits — an 8-bit byte suffices to crossbar four inputs to four outputs with each output derived from a single input, and those inputs and outputs can be full

words.

You could imagine that each cell in a rectangular array of cells, for example, might be configured with, for example, such a 6-bit ALU opcode determining its output word as a function of its two inputs, a bit determining whether its output is registered or combinational, and 6 bits selecting its two inputs from among eight available inputs (including its own output). This might suffice for very simple computation, but so far it's lacking the conditional routing and conditional increment abilities needed for things like packet routing and Bresenham lines.

One simple way to provide that would be to provide *two* configuration words for each cell, and some kind of rule to choose between them, perhaps based on a single bit from somewhere.

Without the possibility of combinational output, you could implement such a machine much more densely in a bit-serial format — each 4×4 crossbar would literally have only 8 data wires going in and out of it, plus 8 control wires. This would of course be much slower.

If such a 4×4 crossbar had 16 control wires going in instead of 8, it could perhaps implicitly perform a wired-AND or wired-OR, for example by using open-drain transistors and pullup resistors; this might eliminate some of the necessity for ALU operations.

An interesting question is what the minimal uniform unit cell for such a machine might be. It takes two or more inputs of some variable word size, generates one output, and has some configuration data. What's the simplest state machine that gives us a given form of universality?

"Incremental computers"?

In Sutherland's thesis, in perhaps the first proposal for a GPU, which would later return to him as the "Wheel of Reincarnation" paper, he said:

> In the course of the work with Sketchpad it has become all too clear that the spot-by-spot display now in use [is] too slow for comfortable observation of reasonable size drawings. Moreover, having the central machine compute and store all the spots for the display is a waste of general purpose capacity...
>
> *The technology of incremental computers is well developed* [emphasis Derctuo], but so far as I know, no one has yet applied them directly to the problem of computer display systems. Basically the incremental computer works by adding one register to another successively and detecting any overflows or underflows which may be generated. Certain registers are incremented conditionally on the result of overflow or underflow generation.

He goes on to explain something similar to the Bresenham line drawing algorithm, using an "X increment" register for the fractional part to be added on each iteration to an "X remainder" register, with an "X scope" register being incremented on overflows, we can increment "X scope" on average every 1/"X remainder" clock cycles. Essentially "X scope" and "X remainder" are the integer and fractional parts of a fixed-point number (in Sutherland's case, binary), and "X scope" is used to control the X position of a CRT beam. A similar Y arrangement is used for the Y beam. The Clock of the Long Now does its calculations the same way, bit-serially, using mechanical binary computation.

This is labeled "Figure E.1. DDA for drawing lines," and it turns out that this is an abbreviation for "Digital Differential Analyser", the term more commonly used in the US at the time; "incremental computer" was the term used in Britain, according to Charles Philip Care's 2008 Ph.D. dissertation, "From analogy-making to modelling: the history of analog computing as a modelling technology." The differential analyzer was the major invention of Vannevar Bush, who also invented hypertext and headed the atomic bomb research program; it was a mechanical contrivance that performed "numerical" integration of ordinary differential equations, originally conceived by Kelvin but not built successfully until Bush managed it in 1931.

(Care's dissertation, though I disagree with most of its conclusions and think its author misunderstands fundamental aspects of the technologies he is attempting to chronicle, is one of the few pieces of research to investigate the distinction between "analog" and "digital" in a serious way. The fact that the conclusions he arrives at are wrong is, by comparison, less significant. And it's one of the few places that will tell you what "incremental computer" meant at the time of Sutherland's dissertation.)

Presumably thanks to Sutherland, nowadays the term "digital differential analyzer" commonly refers to the algorithm he was describing rather than the family of hardware that could implement such algorithms inexpensively.

Sutherland also mentions the remarkable leapfrog-integration property that is also true of Minsky's circle-drawing algorithm:

> Theory and simulation show that just as in the incremental equation used for generating circles (see Chapter V), the latest value of increment must be used if the curve is to close. Therefore, the additions cannot all occur at once; the order shown in Figure E.2 by the numbers 1–4 next to the adders makes the circles and ellipses close. In a serial device it is possible to do the four additions in just two add times by having only a one bit time delay between the two additions for each coordinate, i.e., (?+) just before (+).

And indeed his figure E.2 does present an RTL diagram for something similar to Minsky's algorithm. (For "serial device" read "bit-serial adder".) As I read his variant, it works as follows:

```
xremainder += xincrement
if carry:
    yincrement += ycurvature
    yscope++
elif borrow:
    yincrement -= ycurvature
    yscope--

yremainder += yincrement
if carry:
    xincrement += xcurvature
    xscope++
elif borrow:
    xincrement -= xcurvature
    xscope--
```

This looks like a lot of code but it's really just four arithmetic

operations within the feedback oscillator, two of which are conditional, and two conditional operations driven by the oscillator. (There's an extension to arbitrary conics.)

He says this sort of conditional addition/subtraction goes beyond "the usual practice in incremental computers", which he says is just a conditional increment or decrement (the ordinary sort of carry propagation). However, Wikipedia seems to describe what he's doing as "the basic DDA integrator".

Sutherland points out that this configuration allows us to produce not just circles and ellipses (as is usual for Minsky's algorithm), but also straight lines (by setting the curvature numbers to 0) and hyperbolas (if the curvature numbers are of the same sign). I think it also has a guarantee that it produces densely packed pixels, without space between them, which is harder to achieve with Minsky's algorithm. (To guarantee that it doesn't dawdle on the same point, I think you can prescale the increment values until one of them is essentially 1, e.g., 0.1111111111 if you're using 10-bit fractions.)

If we again treat the xscope.xremainder and yscope.yremainder pairs as fixed-point numbers with the "remainder" being the fractional part, then we can see that this is not quite Minsky's algorithm, because it has four registers instead of two (if we treat the curvatures as constants), and, more surprisingly, it runs *backwards*! The frequency of carries (or borrows) gives us the *derivative* of the variable being thus incremented. So if in some interval of time X increases (or decreases) past 12 integer values, then in that interval of time Y's derivative will be increased (or decreased) by ycurvature 12 times. So Y's second derivative is thus approximated by ycurvature times X's first derivative, and *mutatis mutandis*.

This requires less hardware than the more straightforward approach of X -= k × Y; Y += k × X because that requires a multiplier. By encoding the derivative in this way as a neuron-like spike train, we can multiply by repeated addition instead.

(Spike-train circuitry is experiencing some renewed interest nowadays, both to reduce power consumption and as delta-sigma circuits for higher stochastic DSP speed — the multiplication of two random spike trains is their AND, and their average can be obtained by a multiplexer driven from a random bitstream.)

I think the commonplace circle midpoint algorithm can also be cast into such a form, but triggering additions and subtractions from comparisons rather than carries. It would be interesting to see if there's a stepwise transformation to show the equivalence — or non-equivalence! — of the two circle-drawing algorithms.

Sutherland reports success at the time at building a bit-serial DDA machine at the time using some 36-bit delay lines and some 20-MHz logic chips, plotting a display point every 900 ns.

Topics

Hardware queuing

Kragen Javier Sitaker, 02020-11-26 (updated 02020-12-16)
(11 minutes)

Thinking about hardware multithreading, blocking, waiting, the Padauk chips, etc., it occurred to me that it might be reasonable to build a CPU with a hardware work queue, or several, instead of interrupt handlers and sleeping. (A totally different thought stream involving many of the same features is in the transaction-per-call note (p. 722).)

The CPU should have a priority queue of runnable instructions

The idea is basically hardware multithreading, really: each work queue item is a machine architectural state (PC, SP, and if applicable accumulators, index registers, status flags, MMU tags, etc.), and the CPU works through the highest-priority nonempty run queue, interleaving instruction by instruction. Normally whenever you execute an instruction you enqueue the following instruction on the same priority run queue, so threads of the same priority are run in round-robin fashion, instruction by instruction, but there's a mindelay field in the instruction encoding which forces the thread to pause for 1-4 cycles. So each queue item is tagged with the earliest cycle when it is runnable. This allows interleaved realtime tasks of the same priority to maintain precisely the same timing regardless of how many other tasks of that priority are running at any given time, as long as there aren't too many of them.

(Perhaps the delayed tasks should be initially stored in a separate queue, or perhaps each queue should practice EDF scheduling, although that seems potentially tricky to do in hardware.)

Interrupts are handled by enqueuing a new top-half task at high priority, either on a special interrupt queue or on the regular queue. In a sense an interrupt handler is just a thread that is usually asleep, but usually we don't allocate it hardware registers to store its state between invocations.

There's an instruction to terminate a task, making its hardware state available for forking, and a Redcode-like SPL instruction to fork.

If you have a hardware watchpoint facility, you could provide an instruction to sleep a thread until a given memory location is written to, that location becoming one of the hardware watchpoints. The store units simply need to check their destination address against the list of watchpoints and conditionally awaken a task. As an alternative to external interrupts, you could simply sleep a thread until an I/O port is "written" by the outside world.

The Tera MTA included a facility for a thread to preallocate a set of threads it was going to spawn off, which could atomically succeed or fail; if it succeeded, subsequent thread-spawning instructions from that preallocation were guaranteed to succeed. (And I think that if

you didn't use it, or exceeded the allocated capacity, they were guaranteed to fail.)

If trying to spawn a thread with SPL when all the thread slots are full is treated as a sort of processor exception, one of the possible ways to handle it is to invoke an "OS" that maybe "swaps" one or more threads to RAM, saving their architectural state — the usual OS context-switch code, that is. It wouldn't even have to save the full architectural state if the SPL instruction normally clobbers some of your registers, so it could be as efficient as a cooperative context-switch.

Another possible response to spawning a thread when all slots (at the requested priority level) are full is to enqueue the starting of that thread for later — the thread once running is real-time, but starting it is best-effort, much like in the old circuit-switched telephone system.

A slightly alternate design: A b A c A b A d ... static timeslots

Hmm, what happens if we have run a mindelay-4 instruction from thread A, then a mindelay-3 instruction from thread B? One or the other is going to miss their deadline!

Here's an alternative: we have, say, 4 real-time "queues", A B C, each of which has up to one real-time task, and a fourth hardware task D, a best-effort task. A runs every other cycle (or idles the machine), B runs every 4th cycle, and C and D run every 8th cycle; the sequence is A B A C A B A D, A B A C A B A D... forever — except that if any real-time task is idle in a given cycle, because of having run a sleep instruction, best-effort task D is instead run. D may be a round-robin alternation between various best-effort tasks, either in hardware or in software, and may have its own real-time task that preempts its best-effort task.

In this way, every instruction in queue A has mindelay a multiple of 2, every instruction in queue B has mindelay a multiple of 4, every instruction in queue C has mindelay a multiple of 8, and they are all guaranteed to be reawakened at the cycle-exact time they request. A thread-local architectural register can provide a shift to the mindelay field, so that for example if the field is 2 bits then the possible values of mindelay with shift 1 are 2, 4, 6, and 8; the possible values of mindelay with shift 2 are 4, 8, 12, and 16; and the possible values of mindelay with shift 3 are 8, 16, 24, and 32. This shift is in effect the priority of the thread; a thread running with shift 3 can run in queue A, B, or C, while a thread with shift 2 can only run in queues A or B. So the machine in this form can run up to three threads with shift 3 or less, two threads with shift 2 or less, and one thread with shift 1.

In this form it would be an error to try to change your shift to 1, or launch a shift-1 thread, if there's already a thread running in queue A. If the machine is running three real-time threads with shift 3, it will only be running real-time threads ⅜ of the time, ceding the others to the best-effort thread, but if it is running two real-time threads with shift 2, it will be running real-time threads half the time. If it is running a shift-1 thread, a shift-2 thread, and a shift-3 thread, then it will be running real-time threads ⅞ of the time.

(Actually I guess thread D could also be running a shift-3 real-time thread, if it's not a best-effort thread; it might make sense to make the best-effort thread be a separate thing.)

In theory a shift-3 thread doesn't care whether it's in queue A, B, or C, because in any case it gets one out of every 8 cycles. There might be phase offset questions it cares about.

With this approach, spawning an interrupt thread seems like it will necessarily have worse interrupt latency (both worst-case and jitter) than more traditional approaches, because where do you spawn the interrupt handler thread? What shift do you run it at? If you spawn it at shift 1 in task A, you still have at least 2 cycles of latency, but in order to occasionally do that, you have to *never* run *anything* other than interrupt handlers at shift 1. If you spawn it at shift 2, you can run one other task at either shift 1 or shift 2, but not both; and moreover it will take 1–4 cycles before its time slot comes around, and each additional instruction of handler adds 4 cycles more. For purposes of reliable scheduling, effectively the interrupt handler is a task that always exists — you have to allocate it a task slot.

(The ATTiny2313 datasheet says the AVR interrupt execution response is four clock cycles, minimum, plus normally a three-cycle jump, and possibly finishing a multi-cycle instruction that was in progress when the interrupt fired: 7–9 cycles, plus 4 more cycles if it was in sleep mode, plus the wakeup time. So maybe this isn't actually so bad.)

A perhaps more interesting approach is to have, say, 8 time slots that are rigidly round-robined among in this way, but to run a task at shift 1 or shift 2, you must allocate respectively 4 or 2 slots to it, thus diminishing the total number of real-time tasks by, respectively, 3 or 1. This is sort of like buddy-system malloc: two shift-3 time slots can Voltron into a shift-2 time slot, and two shift-2 time slots can Voltron into a shift-1 time slot.

In the simple form, there isn't an allocation policy that permits full usage but makes fragmentation impossible — a simple adversarial allocation sequence guarantees maximal fragmentation:

- x0 = allocate(shift=1)
- x1 = allocate(shift=1)
- x0.free()
- x00 = allocate(shift=2)
- x01 = allocate(shift=2)
- x00.free()
- x000 = allocate(shift=3)
- x001 = allocate(shift=3)

and so on until all 8 time slots are allocated with known buddy-pairings. Then you can deallocate half the time slots (x001, x011, x101, and x111) in a way that doesn't permit any of them to be coalesced, because their buddies are still allocated; this permits no shift-2 or shift-1 tasks to start, even though the machine is still half idle.

Above it was suggested that *launching* threads might not be a real-time task, even if *running* them is. A different take on that is that launching a thread can shift you into a different timeslot, causing a

phase shift in whatever waveforms you're embroiled in — so in the above adversarial example, whatever shift-3 thread tries to launch a shift-2 task can get reassigned to a different shift-3 timeslot in order to defragment a shift-2 timeslot. That doesn't, however, allow you to launch a shift-1 timeslot.

Another angle on that approach is suggested by Redcode: perhaps the SPL instruction divides your previous timeslot between the two child threads, which I think is actually what the Padauk microcontrollers do in real life — initially the processor runs one thread at 8 MHz, but when it splits into two, each runs at 4 MHz. In the above terminology, SPL raises your "shift" by 1 and spawns a child thread with the same new "shift". This suggests that the way to deallocate is to run a MERGE or WAIT instruction which waits until the other thread executes DIE or HALT or whatever and then drops your "shift" by 1. This incarnation of SPL requires no failure handling but may be somewhat inflexible.

The actual timeslot array in the processor might be something like a 3-bit counter and 8 3-bit pointers, each pointing to one of 8 register sets.

Topics

Foam electro-etching and related techniques

Kragen Javier Sitaker, 02020-11-26 (updated 02020-12-31)
(10 minutes)

In electro-etching, an electrolyte selectively removes metal from a metal surface by anodic dissolution; typically a vinyl mask is applied to the surface to shield some areas, although of course a conventional photolithographic resist like SU-8 epoxy or PMMA can also be used, and painted-on coatings such as Sharpie are also frequently used. This can produce deep etching fairly quickly with high current.

If the surface is first uniformly electroplated (or otherwise coated) with a differently reactive metal, even very shallow electro-etching ought to be able to produce dramatic visual effects by selectively removing the plating, followed by a subsequent treatment such as acid etching, etching with alum, bluing, toning with sulfur, or possibly even autocatalytic "electroless" plating. This ought to enable strongly nonlinear threshold effects as well: where the plating is completely removed, the reactive surface is the substrate, and where it is not completely removed, the reactive surface is the plating.

In cases where the underlying substrate is more reactive than the plating, it ought to be possible to use further uniform electro-etching at a carefully controlled voltage in between the (modified, not standard) electrode potentials of the two materials to selectively remove the substrate material where it is exposed, thus deepening the initially patterned etch.

This is all prequel to suggesting that electro-etching or electroplating with a foam of soap bubbles, as from dishwashing liquid, should make a *freaking awesome* pattern. The air in the bubbles would play the role of the vinyl resist. Thanks to sbp for the idea.

A variant of this commonly happens in a variety of electrolytic processes (anodization, electro-etching, electroplating, electroforming, batteries, and so on) where the bubbles form from electrolysis of the liquid; generally this is considered a nuisance, since the bubbles spawn at unpredictable places, and in batteries "depolarizers" like manganese dioxide are used to counter it. But it might also provide an interesting artistic texture.

In addition to soap bubbles, there are several other surface-patterning approaches that come to mind.

Stamping patterns onto the surface of metal with a conductive rubber stamp (graphite-filled or copper-filled, say) and electrolyte "ink" is another possible form of electrolytic rapid patterning of metal surfaces.

Earlier I'd suggested selective electro-etching or electrodeposition with one or many moving electrodes very close to a metal workpiece as a way to produce precise surface contours, or similarly electrolytic anodization as a way to precisely produce colors. The above-suggested methods of "developing" an extremely thin initial etch or plating with nonlinear effects should enable this process to

pattern a surface orders of magnitude faster, either by selectively etching away part of a surface coating, by selectively depositing plating, or both. (See also the note on ECM engraving (p. 780).

Another possible way to selectively electroplate a surface is with localized laser heating; for example, in a standard acid blue vitriol electroplating solution, even a 5-watt blue laser has been reported to produce this effect by locally heating the solution and thus slightly shifting the electrode potentials.

The more common way to modify a metal surface with a laser is of course to heat it up in the air, which, depending on the degree of heating, can oxidize it, explode tiny holes in it that expose fresh metal, or both. The oxide layer may also be usable as a selective resist. If the laser heating is carried out in a reducing atmosphere such as hydrogen, carbon monoxide, acetylene, or vitriolic air, it could simply remove the oxide, exposing raw metal, rather than depositing it.

By using selective corona or other glow discharge, for example from carbon fibers, platinum electrodes, or sharp aluminum wires, rather than a laser, we could gain a number of other advantages. We could easily pattern the surface at scales well below the wavelength of light, limited only by the diffusion of the plasma, which in turn is largely limited by the precision with which we can control the distance from the tooltip to the substrate. If we are reducing a surface oxide coating, we can use much smaller amounts of reducing gases (or dielectric liquids), and using above-atmospheric or below-atmospheric pressures may be more practical than they would be with a laser. By giving the workpiece a negative charge, we can encourage anions from the plasma to smash into it, reducing lateral plasma diffusion, and the anions can be more reactive than non-ionized molecules would be. (Butane gas, for example, is fairly inert, but a butane plasma will contain all kinds of hydrocarbon free radicals.) This will also tend to vaporize the tooltip electrode faster than a glow discharge would; the electrode can contribute other helpful materials to the mix, including in particular metals for vapor deposition.

These processes, too, can sharpen the boundaries between surface regions using the same kind of differential deposition-then-removal process described earlier for electrolytic processing; for example, first reduce the surface oxide coating everywhere, then selectively deposit it in some places, then selectively remove it in others to steepen its boundaries, and then apply some other reaction, specific to either the oxide or the underlying metal, to use the pattern thus deposited. As another example, you could selectively deposit aluminum in an argon atmosphere by plasma-vaporizing it, then use an oxidizing atmosphere to selectively oxidize areas where you don't want the aluminum.

Using a cold plasma pencil instead of just a glow discharge may permit more flexibility, for example by allowing a higher degree of ionization than a glow discharge can achieve, or allowing short-lived ionized species to decay. But it probably can't achieve as fine precision.

Another way to pattern a surface by local heating is by resistance heating, like a spot welder does. At short distances you can invoke

field electron emission (20–40 V/μm, lower with a low-work-function coating) or thermionic emission to liberate electrodes from your "write head" with which to bombard the surface. (At short distances at atmospheric pressures there isn't enough gas to sustain an avalanche discharge.) This is actually the same process described above for generating plasma, but with a different purpose, of heating the surface rather than generating ions, so the current is in the opposite direction. By pulsing the discharge, greater peak temperatures can be achieved at a given average power, changing the attainable reaction products. This heating can provoke many of the same kinds of reactions as described above. Also, despite what I said above, this current direction is probably better for sputtering atoms off the tooltip electrode.

For thus sputtering metal onto a non-conductive substrate you might want to use two separate electrodes. I suspect such sputtering at atmospheric pressure should be feasible at very small scales.

Local heating and reaction is most precisely attainable with focused electron beams or focused ion beams, but these of course require hard vacuum and thus cannot be used to provoke reactions with gases or volatile liquids, nor reactions that produce much of them. Many semiconductor photoresists are routinely patterned in this way.

Semiconductor etching processes offer further possibilities for amplifying surface patterning, including not only the acid etching mentioned above but also mass anisotropic etching with reactive ion plasmas which react selectively with the exposed substrate.

If you have patterned a metal surface in such a way, you could etch away the substrate metal underneath it — for example, etching steel with alum, or aluminum with lye — to get a very thin foil of the deposited pattern. I understand that Drexler prototyped a solar-sail material in a way similar to this, but you could also use the resulting perforated metal foil as a photolithography mask. A three-layer technique may be the best solution here: first a massive, rigid, etchable substrate; then a uniform thin foil of microns up to hundreds of microns, which is also etchable, but resists at least one etchant that attacks the substrate; then a "resist" mask, perhaps of metal or metal oxide, deposited on top and patterned with submicron thickness. Once the "resist" is patterned, you etch the foil away where it is exposed by the resist; once the foil has been etched all the way through, you switch etchants and etch away the substrate while leaving the foil unharmed.

If you start by depositing a *thin film* of a resist on the surface, you can selectively remove it more easily than the thick films discussed above. Langmuir–Blodgett films of poly(N-alkylmethacrylamides) are already used as UV photoresist for photolithography, but the other patterning techniques described above can also be used with Langmuir–Blodgett films. That includes not only LB films of that photoresist, but also of a variety of inert surface coating materials, and also the opposite — surface coating materials that functionalize an otherwise inert substrate to react with materials it will be exposed to later, or that eventually react with a surface coating that is already present, for example of oxide.

Topics

Using C99 compound literals unjustifiably

Kragen Javier Sitaker, 02020-11-27 (6 minutes)

ISO C99, and GCC since earlier, support so-called compound literals. I've written a number of C functions that look more or less like this:

```
static struct point point_at(int x, int y)
{
    struct point p = {.x = x, .y = y};
    return p;
}
```

The compound literal syntax allows you to write this more simply:

```
static struct point point_at(int x, int y)
{
    return (struct point){.x = x, .y = y};
}
```

However, in simple cases like this, it often eliminates the need for the function altogether, if it existed only for brevity; you can just say `typedef struct point point; ... mp = (point){3, 4};` rather than `mp = point_at(3, 4);`.

This is of course even more useful for arguments of functions; consider:

```
static point delta(point a, point b)
{
    return (point) {.x = a.x - b.x, .y = a.y - b.y};
}

static int distsq(point a, point b)
{
    point d = delta(a, b);
    return d.x*d.x + d.y*d.y;
}

...
printf("%d\n", distsq((point){2, -1}, (point){5, 3}));
```

It's a shame that C doesn't have top-down type inference for this context, so we can't write `distsq({2, -1}, {5, 3})` and have the compiler infer the `point` type. Even OCaml fails us here — because its type inference is bottom-up, to infer types in such cases it requires the field names of record types to be unique, like 1970s C, rather than scoping them within a single record type: as SoftTimur pointed out on Stack Overflow, this is an error in OCaml:

```
type name =
```

```
    { r0: int; r1: int; c0: int; c1: int;
      typ: dtype;
      uid: uid (* key *) }

and func =
    { name: string;
      typ: dtype;
      params: var list;
      body: block }
```

This is in the OCaml FAQ.

(There's an interesting niche open for a language that uses structural subtyping like OCaml's object types and polymorphic variants, but for records with an open set of field names — the kind of thing people do in JS or Lua or with JSON, but with static type checking. I think OCaml didn't have subtyping at all at the time records were added, and its use is still controversial.)

Getting back to C, one of the more interesting uses for so-called designated initializers for struct fields (`.x =...`) is *optional values*. If you have a struct initializer with any initialized fields in it, then *all* the fields of the struct are initialized — even if its storage class is `auto`, the unspecified ones are initialized to 0, as in Java or Golang! So regardless of how many fields are in a `struct foo` you can initialize them *all* to 0 by saying something like:

```
struct foo x = {0};
```

(I think this may be illegal in some versions of C if the first member of `struct foo` is some kind of aggregate, and I think it applies to arrays as well, and I think the requirement to have at least one initialized field has been removed in recent versions of C so `struct foo x = {};` works too, but I'm not sure of any of those.)

So if you have a struct with a large number of fields, you can specify that you want to initialize one or two of them:

```
struct image_transforms t = { .premultiply_alpha = TRUE, .max_depth = 8 };
```

This kind of thing is especially useful to give named arguments to functions with a large number of optional arguments; maybe somewhere there's a `transform_image(&t, ...);` function you're going to invoke. Of course, you always could have designed the interface with a bunch of functions:

```
image_transform_p t = new_image_transform();
if (!t) return 0;
it_set_premultiply_alpha(t, TRUE);
it_set_max_depth(t, 8);
```

But this has several drawbacks compared to the designated-initializer approach. It's more code. It introduces runtime failure into what could have been statically allocated memory, or statically-space-verified stack-allocated memory. It takes time at runtime to execute the function calls, although initializing

stack-allocated memory also takes time at runtime. For statically-allocated objects, you need to somehow run the initialization code at startup, before anything uses the object. It can't be used inside an expression — it forces the caller into an imperative style. And the implementor of the calling interface must write or macro-expand a large number of setter functions.

With compound literals with designated initializers, you get a sort of verbose named-argument syntax:

```
transform_image(&(struct image_transforms){
    .premultiply_alpha = TRUE, .max_depth = 8});
```

This is not a terribly efficient way to get named arguments, though; since this struct has automatic storage duration, if you have 48 fields in the struct, the compiler has to emit code to initialize the other 46, too.

The astonishing thing, though, is that in C, all of these compound literals with automatic storage duration last to the end of their enclosing scope, while in C++ they're treated as temporaries and disappear rather quickly. This means you can build up arbitrarily complex nested structures this way, like Lisp. Consider this expression of the S combinator in the λ calculus:

```
typedef struct ulc
{
    const char *var;
    struct ulc *rator, *rand, *body;
} ulc;

...
ulc s = { "x", .body = &(ulc) {
        "y", .body = &(ulc) {
            "z", .body = &(ulc) {
                .rator = &(ulc) {
                    .rator = &(ulc) { "x" }, .rand = &(ulc) { "z" }},
                .rand = &(ulc) {
                    .rator = &(ulc) { "y" }, .rand = &(ulc) { "z" }}}}}};
```

Here a λ-abstraction is represented by an `ulc` having a non-null `body` and a non-null `var`, a variable is represented by having a null `body` and a non-null `var`, and an application of an operator to an operand is represented by having a null `var`.

This represents some kind of argument about the merits of these language features but I am not sure whether it is in favor or in opposition.

Topics

- Ocaml

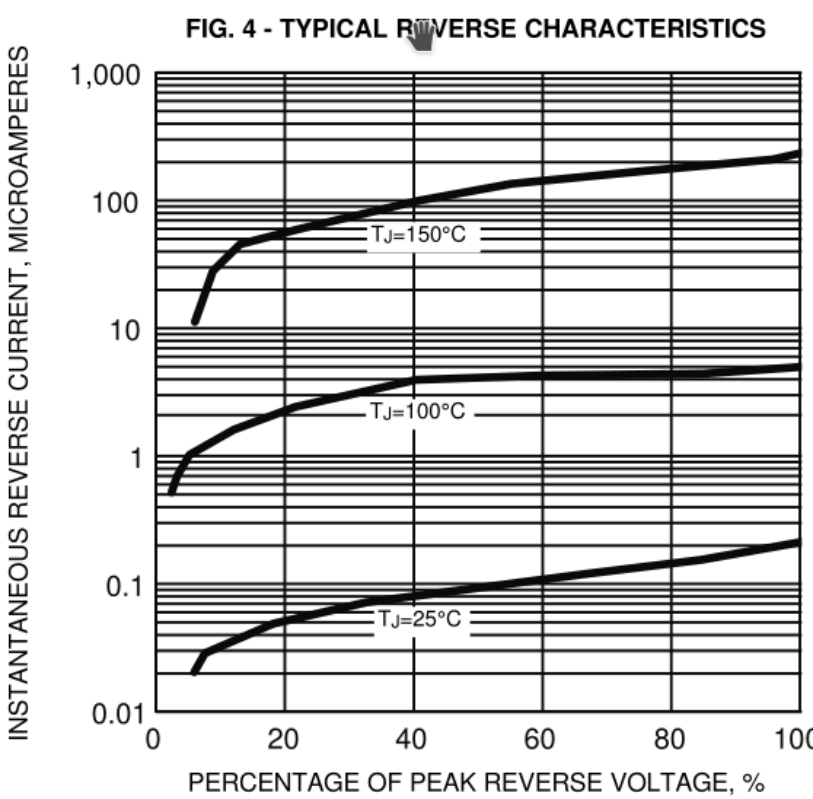
A reverse-biased diode thermometer

Kragen Javier Sitaker, 02020-11-27 (9 minutes)

It seems like, of everyday electronic phenomena, one of those with the largest variation by temperature is the reverse leakage in semiconductor diodes. As explained in the note on thermistors (p. 431), the resistivity of copper changes by about 3900 ppm/K, which is the basis of low-temperature "resistance temperature detectors", and tungsten's by about 4500 (though not consistently; it has phase changes). Carbon film resistors have a usually negative poorly controlled temperature coefficient of resistance; Panasonic, for example, spec their ERDS1, ERDS2, and ERDS25 carbon-film resistors at -150–-1000 ppm/K.

NP0/C0G ceramic capacitors are specified to change their capacitance by under 1% over their temperature range and by under 30 ppm/K, while film capacitors are typically ±200–650 ppm/K, depending on material and humidity. (Thermocouples can in theory be arbitrarily precise but they measure a temperature *difference*, and they tend to drift.)

But feast your eyes on *this* plot of reverse leakage from General Semiconductor's datasheet for 1N4001/4/7 diodes:



Over the 125° temperature range plotted here, they're claiming the diode's reverse leakage varies by a factor of 500–1500, depending on the voltage. That's about *5–6% per degree*, or to put it another way, 55

000 ppm/K.

So if you can measure the current of the diode to within 1% while maintaining it at a relatively constant voltage, you can measure its temperature to within 200 mK. Silicon-junction signal diodes like the 1N4148 are qualitatively similar, but the currents are about an order of magnitude lower than the big rectum fryers; this plot from Diodes Inc.'s 1N4148 datasheet plots leakage versus temperature at a fixed voltage, rather than leakage versus voltage at various fixed temperatures:

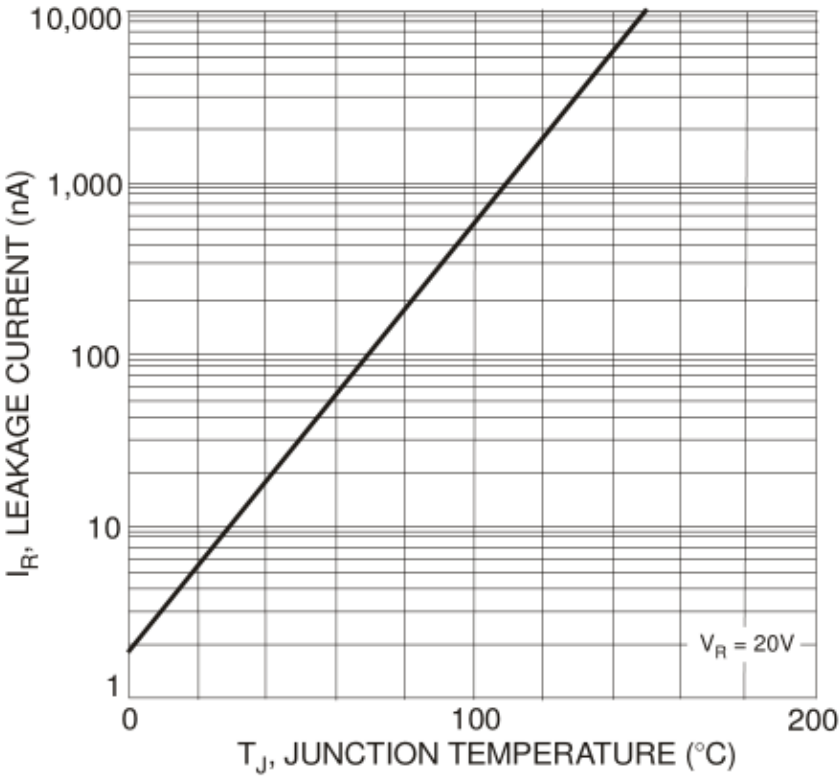


Fig. 2 Leakage Current vs. Junction Temperature

And some other diodes are even *better*! Sunmate's datasheet for their 1N5823 Schottkys, for example, *also* claims a typical reverse current ranging over three orders of magnitude, but over only 100°, amounting to over 7% of current variation per degree, and furthermore at low reverse voltages and more reasonable currents:

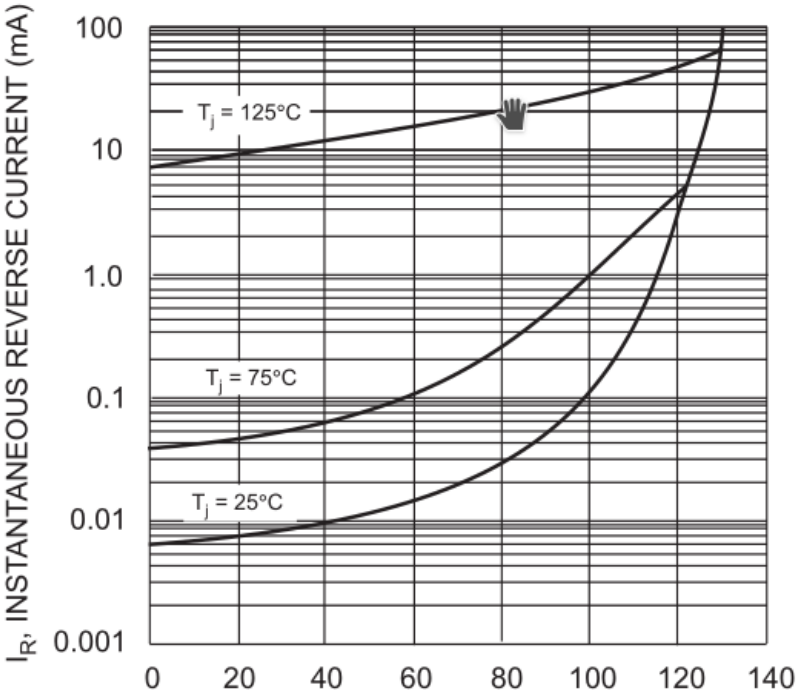


Fig. 5 Typical Reverse Characteristics

This diode is rated for 1500 pF typical capacitance at near-zero voltage, but a milliamp amounts to about 0.7 V/µs at that capacitance, so you'd need to be pretty quick to measure it discharging its own junction capacitance — you'd probably want to use some external capacitance, which might be at a different temperature.

By putting several such diodes in parallel you can increase the leakage current, which will help to make it easier to measure, and perhaps also average out some variation among them.

Crystals

Typical ADCs can't digitize a current, just a voltage, and the voltage is subject to a typically fairly large reference-voltage error (see Multimeter Metrology (p. 502) for lots on the difficulties of measuring voltages). To measure a current, you have to somehow convert it to a voltage. One way is to use a precise capacitor and measure the voltage change over time. But then you need to measure time.

With an ordinary crystal oscillator ("SPXO") you ought to be able to measure the discharge rate with error of better than ±100 ppm over a standard -40°–+105° temperature range, or ±10 ppm if you temperature-compensate it.

Is that right? Digi-Key reports that their most-stocked crystal oscillator is the Abracon ABS05-32.768KHZ-9-T, which they sell for US$0.67–1.16, depending on quantity, from the 728,528 they have in stock. (Hopefully we can assume that it's a relatively typical part due to being so popular.) It's a surface-mount 32.768-kHz crystal, 1.6mm × 1.0mm × 0.5mm, whose error is specified as ±20 ppm at 25°, a temperature coefficient -0.02 to -0.04 "ppm/T²" with a "turn-over

temperature" of 20°–30°, and aging of ±3 ppm in the first year if kept at 25° ±3°. If we assume that "ppm/T²" means ppm/K² for the squared difference of the temperature from the turnover point, where the frequency reaches its max, then at 10° or 40° we might have ΔT = 20K (if the turnover temperature is, respectively, 30° or 20°), thus 400K² and as much as 16 ppm reduction in frequency, or ±8 ppm over that temperature range, plus the ±20 ppm initial error and the ±3 ppm aging error, totaling ±31 ppm over that range, or ±11 ppm if we initially calibrate it. Of this ±11 ppm error, the thermal part reaches 1.6 ppm/K in the worst case at 10° or 40°, so if we can measure our temperature to within 2° we can compensate down to ±6 ppm. Much below that, we start running into tricky issues of things like thermal hysteresis.

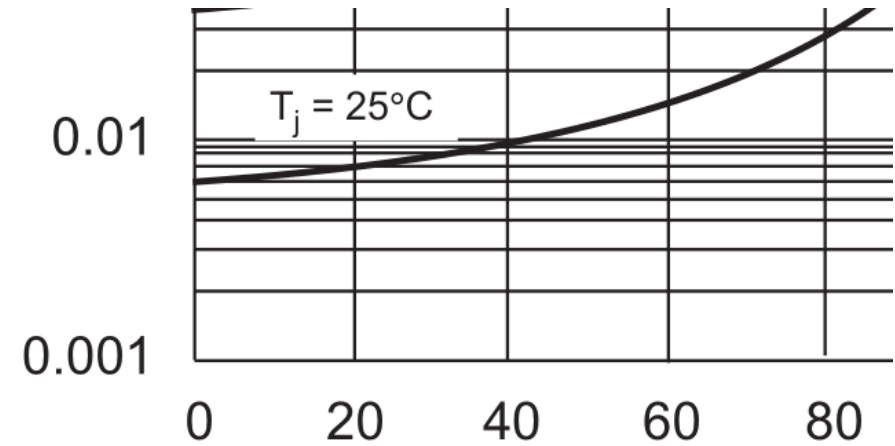
To look at this another way, a 1° error in measuring the crystal's temperature for thermal compensation, a 0.7% temperature error (7000 ppm), produces at worst a 1.6 ppm timing error in the +10°–+40° range.

This is maybe a little better than typical, but not that much. Another very popular crystal on Digi-Key is the 20.0000 MHz Citizen HCM492000000ABJT (196,220 in stock) which is described as "±30 ppm", but again that's at 25°, plus another ±50 ppm over the -10°–+60° temperature range and ±5 ppm aging over the first year. They don't specify the temperature coefficient, but if we figure that the frequency curve is, like the other crystal, parabolic with temperature with a maximum at 20°–30°, then that's up to -100 ppm at a ΔT of 40K, so a worst-case "ppm/T²" of -0.0625 ppm/K².

So I think it's reasonable to expect that we can get to ±10 ppm by calibrating and temperature-compensating random crystals, or ±1 ppm by calibrating and ovening them.

Current measurement error attributable to timing error and voltage error

So, suppose the diode leakage current we're measuring varies by 7% per degree, as the Schottky datasheet above shows it doing, and we're timing how fast it discharges a capacitor. Looking more closely at Sunmate's data sheet:



We can see that on the 25° curve it's about 6 µA at 20% of rated peak reverse voltage and about 10 µA at 40%, which is to say, it's nearly proportional to the voltage; below that, the leakage is nearly

constant, and above, it starts to go superlinear. So if we take our discharge-time measurements with bias voltage near this point, errors in our reference voltage will mostly cancel out — the reverse leakage voltage is, locally, nearly ohmic. But errors in our timing *won't* cancel out.

However, remember that a 1° error in temperature is an 0.7% error (7000 ppm), and it takes a 10% error in timing (100 000 ppm) to cause it. So the ±10 ppm timing error we expect, producing a ±10 ppm error in measuring the diode current, works out to a ±0.7 ppm temperature error, about 200 μK. So probably even an uncompensated SPXO with ±100 ppm would only add about 2 mK of error to the temperature measurement.

Properly characterizing the error introduced by an imprecise reference voltage is difficult without better information than the datasheet gives, but handwavingly I guess that a ±2% reference-voltage error around the ohmic voltage will produce a current measurement error of about 2% of ±2%, or ±400 ppm, producing about ±30 ppm temperature error, or 10 millikelvins.

Other candidate temperature transducers

Are there other things that would work better than a silicon Schottky diode?

I've already explored four-wire resistance temperature detectors (p. 431), which have the advantage that their linear E–I relation permits measurements that precisely cancel out any non-drifting errors in the reference voltage, but have 15× smaller responses on an absolute scale.

The other temperature-measurement candidates that come to mind are LED reverse leakage (though apparently that's typically very small and difficult to detect because the bandgap is larger, suggesting that maybe germanium diodes might work better if you can find one), ferroelectric-capacitor dielectric permittivity, and the aforementioned quartz crystal resonant frequency itself, this last not because it varies a lot but because it can be measured to greater precision. Most crystals, though, are cut to have their extremum in resonant frequency happen at 20°–30°, so it's hard to measure them against each other, and we need to be far from the extremum to get a large effect.

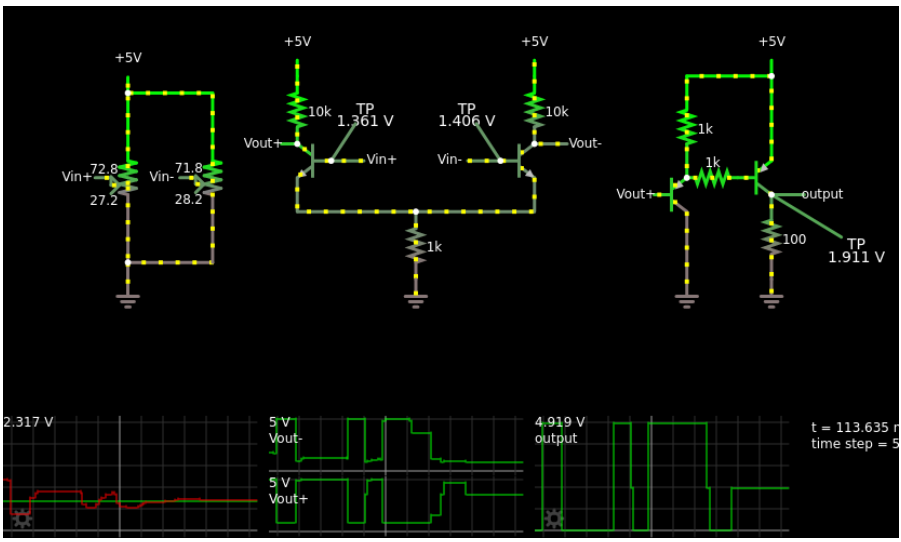
Topics

My very first opamp

Kragen Javier Sitaker, 02020-11-27 (4 minutes)

It's been Norbert Wiener's birthday today, and although he wasn't the inventor of the op-amp, he was the key person in explaining the significance of the op-amp and similar things. So I thought I'd celebrate by building my very first op-amp. If I could.

I got a four-transistor circuit working in Falstad's circuit simulator:




```
$ 1 0.000005 0.06323366621862497 50 5 50
R 176 208 176 176 0 0 40 5 0 0 0.5
174 176 368 176 208 1 100 0.5693 Vin+
g 176 368 176 400 0
207 160 288 128 288 4 Vin\p
207 368 272 416 272 4 Vin\p
t 368 272 336 272 0 1 0.6109572102548784 0.6310546959814354 100
w 336 288 336 320 0
w 336 320 448 320 0
r 448 320 448 384 0 1000
g 448 384 448 400 0
r 336 256 336 192 0 10000
R 336 192 336 160 0 0 40 5 0 0 0.5
207 336 256 304 256 4 Vout\p
w 176 208 256 208 0
w 176 368 256 368 0
174 256 368 256 208 1 100 0.2822 Vin-
207 240 288 208 288 4 Vin-
t 528 272 560 272 0 1 -3.588999998997953 -0.7590165014195449 100
w 560 288 560 320 0
w 560 320 448 320 0
207 528 272 480 272 4 Vin-
R 560 192 560 160 0 0 40 5 0 0 0.5
r 560 256 560 192 0 10000
207 560 256 608 256 4 Vout-
207 688 304 656 304 4 Vout\p
r 784 384 784 304 0 100
```

```
R 784 192 784 160 0 0 40 5 0 0 0.5
g 784 384 784 400 0
207 784 304 832 304 4 output
368 784 304 864 352 0 0
r 752 288 704 288 0 1000
t 752 288 784 288 0 -1 -0.5948957135185378 -0.6972226181259069 100
w 784 272 784 192 0
t 688 304 704 304 0 -1 2.190113987148148 -0.6288987134827844 100
w 784 192 704 192 0
r 704 192 704 288 0 1000
g 704 320 704 400 0
368 528 272 496 224 0 0
368 368 272 400 224 0 0
o 3 1 0 4098 5 0.1 0 2 20 0
o 23 1 0 4098 5 0.1 1 2 23 3 Vout-
o 12 1 0 4098 5 0.1 1 2 12 3 Vout\p
o 28 1 0 4098 5 0.1 2 2 28 3
```

It seems to have open-loop differential gain of at least 100 and outputs down to the negative rail ("single-supply op-amp") but it wastes a ridiculous amount of power in its class-A output stage. Also it has an input offset voltage of about 50 mV and an offset current of a couple of milliamps. So it's not a very *good* op-amp, but it *is* an op-amp.

I feel like it ought to be possible to get the output stage down to one transistor instead of two, but I was having a hard time getting that to work (my output stage was loading down the differential pair too much) and so I just brute-forced it with a PNP emitter follower. The emitter-follower loading is still probably the culprit for the rather poor offset voltage.

I haven't breadboarded it yet, and I had a hard time figuring out why the previous circuit simulation weren't working, and I'd like to see if I can get the output stage down to one transistor.

Topics

Taking screenshots

Kragen Javier Sitaker, 02020-11-27 (updated 02020-12-20)
(14 minutes)

I often take X-Windows screenshots to include in Derctuo, for example the schematic of My Very First Op-Amp (p. 665). Typically this process has looked something like:

• Launch the GIMP. Wait for it to finish booting.
• Take a screenshot with it.
• Crop the screenshot in the GIMP (shift-C), zooming in and out as necessary.
• "Export" the screenshot (ctrl-E).
• Exit the GIMP.
• Confirm that, yes, I really do want to "exit without saving" (ctrl-D, I think).

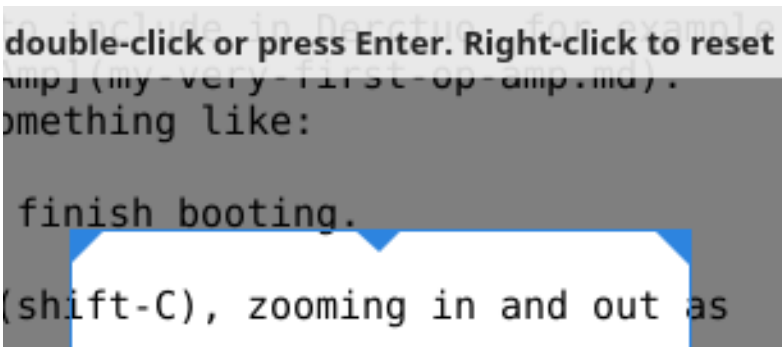
This is very cumbersome, taking about a minute. So I just tried some various different screenshot programs to see what would work better. KDE's "Spectacle" program is wonderful, better than I had thought possible, and now I can invoke it from Emacs with a single keystroke and automatically insert the resulting cropped screenshot into the Markdown document I'm editing.

This has had the rather alarming effect of enabling me to add half a megabyte of images to Derctuo over the last couple of days, which is about the same as the amount of text I've added to it in the last two *months*. But the year 02020 is over in another month, and Derctuo is still less than 10MB out of its 20MB size budget, so maybe I can lighten up a bit.

Using KDE Spectacle from the command line

Boudhayan Gupta's Spectacle is *the shit*. It is *totally awesome*.

I ran `sudo apt install kde-spectacle` because I don't have KDE installed.



`spectacle -rbo somedir/somefile.png` takes a few seconds to start, and then brings up a fullscreen instruction screen, which you can dismiss by starting to drag out a rectangle. But *once the rectangle is there*, you can *interactively resize it*, seeing the cropped image as it will finally appear. Then you can hit Enter to save it to `somefile.png` (`-o`) with no

further interaction (`-b`). (Not even confirmation for overwriting or errors for unsupported file extensions — be careful!)

You can even use Spectacle to take a screenshot of Spectacle's own cropping UI in operation; I think it works by taking a full-screen screenshot when it starts, then covering up the screen with it for this `-r` region cropping UI. In fact, sometimes the cropping UI sort of turns back time a little bit, showing what the screen looked like a second or two earlier.

The whole interaction takes about 15 seconds, even though I'm not running KDE.

This is the ideal mode of operation for invocation from a script; the only GUI interaction is dragging out a box, optionally resizing it, and pressing Enter or the right mouse button.

It has a few problems. One is that it emits some debugging crap to stderr every time you run it:

```
Problem creating accessible interface for:  ScreenClipper(0x9aa3c90)
Make sure to deploy Qt with accessibility plugins.
```

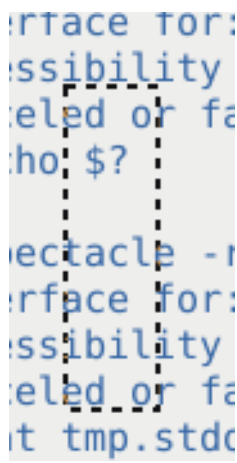
Another problem is that if you cancel the screenshot, or if it fails, you don't get a nonzero exit status; instead you just get another stderr message:

```
ERROR:  "Screenshot capture canceled or failed"
```

A third problem is that it somehow returns the keyboard focus to *the XFCE desktop manager* every time it runs, so you normally have to press alt-tab to get back to one of your windows.

At least four other reasonable options are not as good as Spectacle

It turns out there are several other approaches that are considerably more convenient than what I was doing with the GIMP.



`xfce4-screenshooter -s somedir -r` is nearly as good as Spectacle, and it's even faster (maybe 6 seconds rather than 15 on this netbook), but it doesn't let you interactively adjust the dotted-line crop box shown above before saving — once you release the mouse button, that's it! But then it prompts you for a filename. At least it has the option of

specifying what directory to put it in.

`mate-screenshot -a` is similar, but you have to manually navigate through the filesystem to the right directory with its GUI. Only the first time you run it, though, or when you're switching directories.

`gnome-screenshot -i` is similar, except that you additionally have to select the option to not screenshot the whole screen every fucking time you fucking start it.

Mirage is an image viewer with a relatively accessible "crop" option on its "Edit" menu, as well as a full-screen or full-window screenshot option on its "File" menu; it's a much easier way to crop existing images than the GIMP. Beware, by default it saves the screenshots into directories like `/tmp/mirage-EYotNO`. The cropping functionality is also somewhat suboptimal since the view of the image in the cropping window is teensy (though what's shown on the left side of the screenshot above is the thumbnails of other nearby files). Having cropped the image you can save it over the original with no further confirmation.

Several other options are even worse than the GIMP

KGrab takes forever and doesn't have a crop option. ScreenGrab is impossibly clumsy. On this version of X11, `xwd | xwdtopnm | pnmtopng > foo.png` generates a totally bogus image that looks like it used the wrong pixel format or the wrong part of the screen or something, and of course it also doesn't have a crop option. Kazam brings up a blank gray fullscreen window so you can indicate which part of the screen you want to grab; I think it's really intended for screencasting, with screenshots being an afterthought.

The well-known `scrot` command is worth a mention; it doesn't have a crop option, but `scrot -e 'mirage $f'` will open the full-screen screenshot in Mirage so you can crop it with Mirage, and hopefully not forget to save the cropped version. By default `scrot` generates a filename but you can instead specify one: `scrot foo.png`.

ImageMagick has an `import` command which I think can do cropping — but I think you have to specify the pixel coordinates on the command line, not with the mouse.

Scripting screenshotting in Elisp

The workflow I really want is to be editing a Markdown document in Emacs, press a magic key (Print Screen, which Emacs calls `<print>` or `[print]`, actually works, since, for whatever reason, my XFCE doesn't intercept it), type a filename to store the screenshot in, hide

the Emacs window, spawn off Spectacle to capture the file, unhide the Emacs window, and insert a Markdown inline image tag for the screenshot.

My Elisp is a little rusty, but I managed to get this to work:

```
(defun screenshot-save-to (filename)
  "Interactively crop a screenshot with Spectacle and write to FILENAME.

  This is not an interactive command because it doesn't check
  to see if FILENAME already exists, doesn't hide and
  redisplay the Emacs window, and doesn't append .png if
  FILENAME isn't a PNG or JPEG filename.  `screenshot-make'
  does those things.

  "
  (let ((screenshot-return-value 'unknown-return-value))
    (let ((screenshot-messages
           (with-temp-buffer
             (setq screenshot-return-value
                   (call-process "spectacle" nil t nil
                                 "-rbo" (expand-file-name filename)))
             (buffer-string))))
      (if (or (string-match "ERROR" screenshot-messages)
              (not (eq screenshot-return-value 0)))
          (error "Screenshot failed: %s(return value %s)"
                 screenshot-messages screenshot-return-value))

      (if (not (file-exists-p filename))
          (error "Screenshot supposedly succeeded but %s doesn't exist: %s"
                 filename screenshot-messages)))))

(defmacro with-frame-iconified (&rest body)
  "Iconify the current frame only until BODY completes."
  (declare (indent 0) (debug t))

  `(progn
     (iconify-frame)
     (unwind-protect
         (progn ,@body)
       (make-frame-visible))))

(defun screenshot-make (filename)
  "Take a cropped screenshot.

  If FILENAME doesn't end in .png or .jpeg, this command appends .png.
  When called as a Lisp function, it returns the real filename."
  (interactive "*FScreenshot filename to create: ")

  (if (not (or (string-suffix-p ".png" filename)
               (string-suffix-p ".jpeg" filename)))
      (setq filename (concat filename ".png")))

  (if (file-exists-p filename)
      (if (not (yes-or-no-p (concat filename " already exists; overwrite? ")))
          (error (concat "Not overwriting " filename))))
```

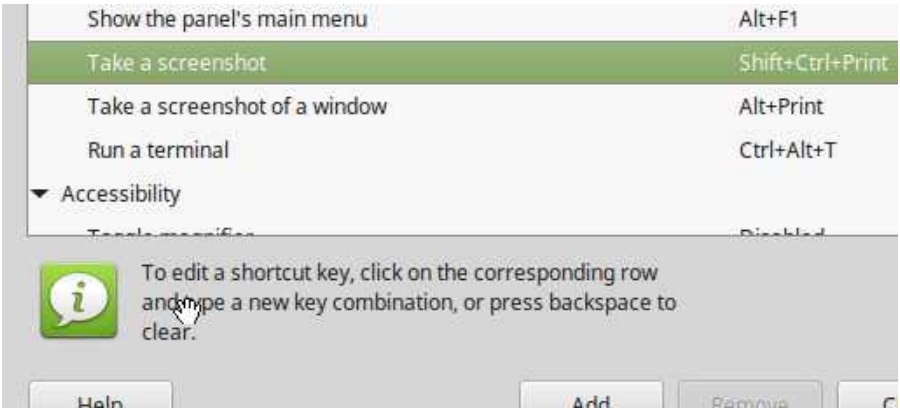
```
  (with-frame-iconified
    (screenshot-save-to filename))

  (let ((file-size (elt (file-attributes filename) 7)))
    (message "Screenshot %s is %dKiB." filename (/ (+ 512 file-size) 1024)))

  filename)

(defun screenshot-insert-preview-line (filename)

  "Insert a newline into the buffer with, if possible, the image FILENAME displayo
oed."
  (interactive "*FImage filename: ")
  (let ((screenshot-image-descriptor
         (create-image (expand-file-name filename) nil nil :margin 4)))
    ;; (message "descriptor %s" screenshot-image-descriptor)
    (if screenshot-image-descriptor
        (progn
          ;; If we just overwrote an image, Emacs might have it
          ;; cached.
          (image-flush screenshot-image-descriptor)
          (insert-image screenshot-image-descriptor "\n"))
      (insert "\n"))))

(defun markdown-insert-screenshot (filename)
  "Crop a screenshot and insert a Markdown inline image in source and buffer."
  (interactive "*FScreenshot filename to create: ")
  (setq filename (screenshot-make filename))

  (let ((basename (file-name-nondirectory filename)))
    (insert (format "\n![(screenshot %s)](%s)\n" basename basename)))

  (screenshot-insert-preview-line filename))

(defun markdown-insert-preview ()
   "Show a preview for the previous Markdown inline image tag if possible.

   Unfortunately this function modifies the buffer.
   "
    (interactive)
    (if (not (re-search-backward "^!\\[.*?\\](\\(.*\\))\n\n"))
        (message "no Markdown inline images found")
        (save-excursion
          (message "found %s" (match-string 1))
          (move-end-of-line 2)
          (delete-char 1)           ; hope this is a newline
          (screenshot-insert-preview-line (match-string 1)))))

 (global-set-key [print] 'markdown-insert-screenshot)
 (global-set-key [C-print] 'markdown-insert-preview)
```

This also displays the image inline in the Emacs buffer! But only until I close and reopen the file (or reboot Emacs), though I can use Ctrl-PrtSc (`markdown-insert-preview`) to re-add them one by one after

reopening the file. Mysteriously the `revert-buffer` command displays the images in the right margin; I suspect this might be a bug in fill-column-indicator.el. A little refactoring might make it possible to scan for such images to add such previews to, but I probably wouldn't want to invoke that automatically every time I opened a file.

This is a pretty nice experience in Emacs as long as the images aren't too big:

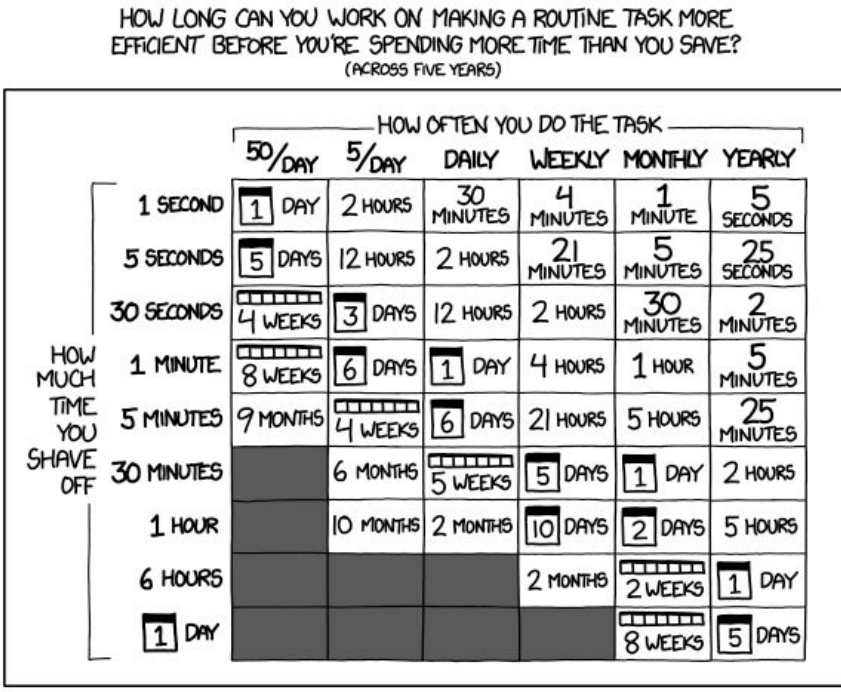
(To make this work with the same key on MATE on my craptop, I opened Main menu → Control Center → Hardware → Keyboard Shortcuts (*not* "Keyboard") → Desktop → Take a screenshot and reconfigured it to "Shift+Ctrl+Print".)



Sometimes the images seem to disappear and be replaced with small squares; maybe Emacs is evicting them from some cache and not putting them back. So far this has been a minor annoyance.

Efficiency

Well, I've spent the last 9 hours on automating screenshots†, so now I can insert a screenshot into my notes in only 45 seconds. I ran through the previous procedure using the GIMP again and it took me 4 minutes and 51 seconds, but I think I was usually able to do it faster than that — I must be getting sleepy, and I couldn't figure out where the GIMP had saved my screenshot. But, suppose it's 3 minutes "saved". Am I being efficient?



According to the comic, yes, as long as I insert several screenshots a week, and keep doing this for five years or more; that seems very likely to be true. That doesn't take into account the good or bad of having the screenshots displayed in the notes as I'm editing them, though, or whether I learned anything useful in the process, or whether these notes are useful to somebody else.

(Is It Worth The Time comic by Randall Munroe, licensed CC-BY-NC 2.5.)

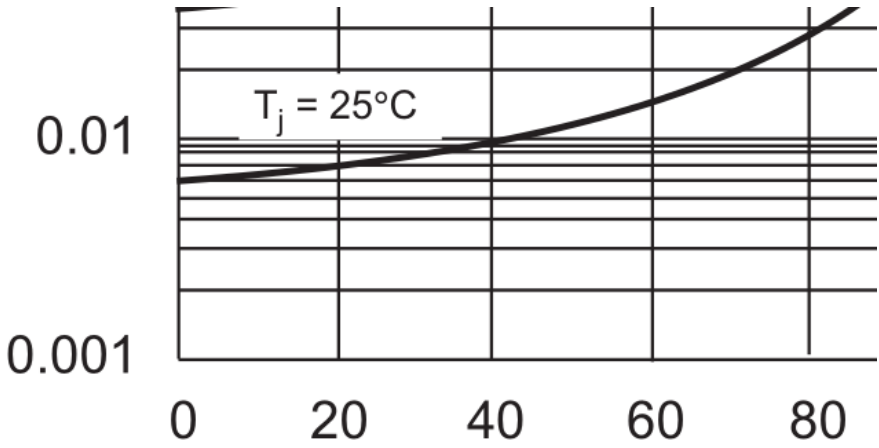
† Actually, I spent some more time on it the next day to add JPEG

support and refactor the code reasonably.

Compression

By default if you ask Spectacle to write a JPEG it writes it with reasonably high but not impeccable quality; in the case of the XKCD comic above Spectacle generates about a 140K PNG (which `pngcrush` reduces to 107K) or a 74K JPEG, while ImageMagick produces an equivalent-quality JPEG with `convert -quality 50 xkcd-time-saved.png xkcd-time-saved.jpeg` at 50K, or an equivalent-size JPEG with impeccable quality at `-quality 80` (73K, monochrome).

On theoretical grounds we would expect PNG to have a better quality/compression tradeoff than JPEG on text and line art, and that is somewhat borne out by experiment. Consider this Schottky-diode-leakage graph from the note on diode thermometers (p. 660):



As a PNG this is 15K, which is pretty good. It's still totally readable at `-quality 1`, where it's 6K, and the JPEG artifacts are not strikingly obvious at `-quality 10`, where it's 10K. But they don't really disappear until `-quality 50`, at which point the JPEG is 18K.

The Mirage screenshot earlier looks pretty much the same in JPEG with `-quality 80` and PNG, but it's 9K in JPEG and 17K in PNG.

I think the conclusion is that I should use JPEG for JPEG things (and just accept Spectacle's reasonable default quality) and use PNG for text and line art. And occasionally I should run `pngcrush` on all my PNGs, which is an easy batch process and therefore doesn't need to be automated in the Elisp interaction above.

Topics

Punk zine look

Kragen Javier Sitaker, 02020-11-28 (6 minutes)

In seeking an aesthetic alternative to the shrink-wrapped Apple Store look so popular in modern UIs, I thought it would be inspiring to look at old punk zines.

Reviewing old material

Chainsaw

Looking at the aesthetic of Chainsaw I see:

- lots of black
- text placed at random angles within irregular cutout shapes overlapping backgrounds
- Xerox halftoning (or Gestetner duplicating) blacking out or whiting out much of the photos
- hand-lettered text with amphetamine style
- photos also at random angles
- shagging pigs on the front cover
- typewriter text because that's what was available
- redrawn title every issue
- use of image as stencil over psychedelic color gradient like from cheap posters
- budget constraints forcing B&W
- cartoon covers
- fluorescent color clash
- interspersed photos of fistulas and rectal examinations from a 1920s medical book

Koleksi

Someone uploaded Koleksi to the Archive. Seems to be a punk zine from Malaysia from 1997 or so. I see:

- pages of typewritten text with frames around them, sometimes broken by pasted-over titles in slightly different font size
- a few different font sizes
- Xerox-saturated B&W photos
- typewriter-overstruck chars
- all caps, ????, !!!!
- sideways text
- typewritten text flowed around images
- letters maybe intentionally vertically displaced using the shift key
- everything in English, nothing in Bahasa

In a 2011 issue of Keeema, a comic about politics in the same collection, I see:

- photos as comic-strip panels with speech balloons
- all Bahasa, almost no English
- imitation logos to make fun of brands and politicians
- lesbians in hijab kissing

• orangutans on a motorcycle
• posterized photos

In Dogged, another zine from the same collection:

• walls of text in sans serif proportional font
• all Bahasa
• handwritten captions superimposed on a xeroxed photo
• line-art comix

Maximum Rocknroll

In an issue from 1992 I see:

• lots and lots of ads
• standard 3-column DTP kind of layout and formatting
• newspaper-like halftoned images
• Xerox halftoned images
• extensive discussion of gender dynamics and rape
• contrasty photos and drawings of naked women (the drawing, repeated many times, seems to be the logo of the band Spitboy)
• columns of text pasted over backgrounds and xeroxed
• pullquotes with poetry

Homocore

Homocore was founded by Tom Jennings of FidoNet and Deke Nihilson, and published 1988–91. The Internet Archive has preserved some even though Tom got sick of the internet's shit and/or sold out.

I see:

• basic DTP one-column layout with bold serif text
• Xeroxed Fred Phelps bigot photos
• basic DTP two-column layout with line drawings
• comix
• basic DTP two-column layout with line drawings and horizontal rules 2

Kill Your Pet Puppy

KYPP is totally online. In the first issue I see:

• angled text blocks cut from other publications and pasted in (a screenshot of sorts)
• running text in hand-lettered thought bubbles
• xeroxed photos of topless women apparently sucking something
• !!!!! (hand-lettered)
• collaged typewriter text at various angles
• Xerox-saturated photos of contributors, all using pseudonyms
• typewritten text on monochrome blue background photo
• photo with hand-drawn speech bubbles stenciling over a psychedelic gradient
• watermarked album names behind typewritten text
• quoted cliché comics with replaced speech bubble text
• bright primary colors mostly on white
• an article about a problematic "new" recreational drug called Tuinal (which turns out to be a brand name for a barbiturate cocktail sold as

a sleeping pill from the 1940s)
• a Xeroxed page from an anarcho-situationist leaflet from 1974 with Stalin imploring you to support the British government
• doublespaced typewritten: 'I'm positive the 70s will be given a real shakedown in years to come, every moment dissected by people intent on rediscovering its secrets. Where will they look? No doubt "punk rock" will be celebrated and chronicled. But what of the lost years of this decade, the first five. Will schoolchildren tronble at the names of Gary Glitter or Slik.'

Generalizing

An overriding theme here is use of whatever is expedient for communication: vanilla DTP layouts, typewriters, handwriting, Xerox machines, Gestetner duplicators, collage, etc. Other common themes include intentionally transgressing taboos, whether small or large; deliberate scruffiness; social criticism; and maximizing intensity rather than seeking or even accepting the sort of "tasteful restraint" that connotes prestige in the cultures these zines were criticizing.

Modern zines commonly use the same design elements, which you can interpret as an homage to the old punk zines, a clichéd imitation of the form which by virtue of being imitative runs counter to the spirit of rebellion and innovation that animated the original, as an attempt to appropriate the credibility and cachet of the original, or just a result of the authors having the same tools at their disposal.

What would we get if we were to apply the maxims of traditional punk zines, as I've described them above, in the current historical context? Typewriters and Xerox machines are, to say the least, not expedient for communication today! Using the internet is expedient, and so is taking photos and screenshots. And they can be in color with very nearly the same ease as black and white. Handwriting and scruffy hand-drawing is still expedient.

Of course when it comes to *documents* we have a Web-native vernacular for such things, or really several now: plain-text mailing lists and their archives; the GeoCities epoch of garish backgrounds, animated GIFs, "under construction" signs, and `<blink>` and `<marquee>` tags; README.md on GitHub; Yahoo Groups; Facebook groups; and so on. But what I'm interested in here is *user interfaces*.

User interfaces are not documents. Their appearance changes over time, and not in a predetermined way — rather, in a way that arises from *interaction* with the user, who is in a significant sense the coauthor of what they are seeing.

What's the equivalent of the typewriter and xerox for user interface design? Hypercard?

Topics

Caching layout

Kragen Javier Sitaker, 02020-12-03 (8 minutes)

Let's consider building up a GUI with table layout, flowable text, graphical objects, and clipping, based on a pipeline from some arbitrary state data, through to a set of nested boxes, to a layout which assigns positions to all these boxes, to pixels to put onto the screen, all in a form that is purely functional and therefore tractable to memoize or cache, without losing unreasonable amounts of efficiency.

First, from the arbitrary state data, we build up a structure of nested boxes. Maybe we iterate over the characters of a text and generate letters from a font, which we stack into hboxes that form words, which we fill into paragraphs which will compute a reasonable set of positions for the word boxes within the resulting paragraph box, which is perhaps a child of a vbox. Maybe somewhere there are some GUI elements that get laid out in a table, and one of them is a dropdown list which is associated with a transient dropdown, which should get composited on top of the whole shebang at a later step. Maybe all of the above is within a scrolling pane of a larger window.

If this stage of the process is comprehensively memoized, then rather than a tree of nested boxes we will have a DAG of nested boxes; perhaps the letter "f" from a given font only results in a single tree node. Also, if it's comprehensively purely functional, it's safe to discard any of these boxes as well, as long as we save the function call that produced them so that we can restart it if we need the box again.

Second, to do the layout, we do three further passes over this DAG of nested boxes.

Pass 2.1 is to compute a corresponding DAG of nested boxes augmented with requested sizes and elasticities. This is intended to be a mostly bottom-up pass, where each resulting node depends on its descendants, but not on the space actually available for it on the screen. (This is easier for Tk-style widgets than for paragraphs!)

Pass 2.2 is to compute the actual positions and sizes of each nested box, which is a top-down process, beginning with the size assigned to the root box. This produces another DAG of nested boxes augmented with (x, y, width, height) geometry information. Each box is given only the width and height assigned to it, and then can follow any policy it likes to assign positions to some or none of its children within itself, whatever is visible. The positions may overlap, in which case the z-order is important.

Pass 2.3 is to add floating boxes by iterating over the tree from the previous pass and giving each visible box an opportunity to produce things such as dropdown lists that should be propagated further up the tree.

Now that we have a layout, we have essentially a scene graph, and we need to rasterize the scene graph. So we make two more passes.

In pass 3.1, we produce a new augmented tree (this is starting to sound like a nanopass compiler framework) that has a pixel buffer associated with each box, a texture buffer which is scaled to the box's visible area, and which contains the graphics from the *background* of

that box, not the children. But the node still has references to its child boxes and their positions relative to it. Some boxes may be translucent, but most are opaque.

Pass 3.2 is rasterizing the actual screen. The standard scan-line rendering algorithm (Wylie et al. 1968) is, I think, to sort all the objects on the screen by their minimum Y-coordinate, then iterate over the scan lines maintaining a currently-visible priority queue (the so-called "active edge table") of the *edges of* the objects intersecting the current scan line according to their *maximum* Y-coordinate, while adding the edges of new objects to the queue when we get to their minimum Y-coordinate. (The original Wylie et al. algorithm calculated an "occupied table" of *polygons* for each scan line.) Within each scan line, we sort the edges of the visible objects by X-coordinate (an insertion sort which does work proportional to the number of new out-of-place edges, times the total number of edges), then iterate over blocks of pixels to sample from the textures of the topmost object in that block, and whatever other objects are visible through it, if alpha-blending is called for. Finding out the topmost object at a given pixel is also a priority-queue problem, this time with Z-order instead of bottom-order.

Unlike all the previous passes, this last pass is not cacheable at a node-by-node level like the others because its results aren't associated with subtrees; a box may be overlapped by some other box that isn't its descendant, and we respond by not trying to sample its texture in the overlapped area. It can make up for this by being fast so we don't need to cache it; remember, it's just sampling from textures, typically one or two samples per pixel, if most boxes really are opaque.

If we want to do this in a real-time system that guarantees responsiveness, we need some kind of cheat for cases where we can't meet our deadline with the correct answer. If there are too many things on the screen to rasterize in time, for example, we could rasterize only certain scanlines, for example, or make everything opaque, or only rasterize the first N objects on each scanline, or the first N objects after the largest X-coordinate that was reached during the last frame. If it's instead one of the tree stages that's taking too long, we might be able to use an old or outdated version of the subtrees we don't have time for, perhaps tagging them in order to gray them out.

To figure out what's worth caching, we might be able to use recency, reference counts on cacheable nodes, and the computation time they took; when we discard a cacheable node, we must move the computation time it took to each of its previous parent nodes.

Some ingenuity may be required for the cache manager to do this quickly and optimally; a straightforward suboptimal solution is to maintain two caches, add each newly memoized item to both caches, and whenever a cache becomes full, empty it in constant time (by resetting an arena pointer); additionally, whenever a cache crosses the halfway-full mark, empty the other cache unless it's nearly full. Lookups need not consult both caches, since one is a perfect superset of the other, but should promote the cached item to both caches.

The result of this approach should be to empty the caches alternately at nearly regular intervals, so that all the items that are

referenced more often than that interval remain always cached, and items that are referenced less often have some probability of remaining cached. In particular, if a parent call is getting reliably cached, then its children and grandchildren will not get referenced, and will be eventually eliminated from the cache, which is in some sense optimal. It may lead to bad worst-case behavior, though.

Topics

Compressed imaging

Kragen Javier Sitaker, 02020-12-06 (3 minutes)

In compressed sensing we sense a signal, for example an image, via some kind of linear basis that's incoherent with respect to some underlying basis in which we expect the signal to be sparse, and then attempt to estimate a sparse signal in that underlying basis that best "explains" our observation. If we are correct in our prior that the signal should be sparse in that underlying basis, this does a great job at reproducing the true signal. (And often we can choose the underlying basis such that when we're wrong about that, it's one of the cases we care less about.)

It occurs to me that you can use this for producing images as well. Consider, for example, a disco-ball sparkle pattern swept over a wall while being illuminated by a rapidly modulated LED (or three). A camera or eye will sum many successive positions of the sparkle pattern together due to the persistence of vision, and the brightness and color of these positions will depend on the brightness of the LED at that moment. These may be sufficiently incoherent with respect to a suitable basis such as the Fourier basis as to be able to sum to an arbitrary visually coherent image.

They may not, though, and the inability of the LED to emit negative light may be a serious limitation here, since it limits the image's dynamic range, potentially rather badly (like to 3:1 or 4:1 rather than the 100:1 of a good LCD or CRT.) Other candidate output devices for such compressed imaging include:

- A rotating sparkling surface illuminated by a time-domain-modulated light, or several, viewed from a single point.
- A piece of sandpaper with grains in random but known positions, rotated over a surface while floating on a cushion of air, then whacked into the surface at precise moments by a hammer at one location or another.
- A sparkle pattern produced by refraction or reflection through two or more random but known optical surfaces, either fixed or in known motion with respect to one another.
- A collection of multi-pointed electrodes swept over a surface with a time-domain-modulated electrical current on each one to deposit and/or remove and/or functionalize material, for example through electroplating and electro-etching, through plasma surface activation, or through vaporizing parts of the surface.

If you use an optimization algorithm in a Fourier-like basis whose objective function selectively neglects phase and precise frequency, you may gain useful degrees of freedom with respect to human vision and audition, among other things: the humans can't hear the phase of the tenth harmonic of a vocal signal, nor see if all the hairs in an area of a closeup photo of a person have been shifted half a hairsbreadth to the right, nor hear the difference between 60Hz and 60.1Hz. This optimization approach is of course also useful for applications like mural design, JPEG compression, and adapting sound reproduction to the resonances of a given listening space.

Topics

A letter-by-letter Hamming code for manual ECC computation

Kragen Javier Sitaker, 02020-12-06 (updated 02020-12-16)
(5 minutes)

Watched 3Blue1Brown's video on Hamming codes recently and a couple of thoughts occurred to me.

First, Hamming codes, like matrix parity codes, are simple enough that you could reasonably compute them by hand, making them a reasonable candidate for archival media.

Second, you can take the same Hamming-code approach over any character code, not just a binary code. For example, rather than computing a (15, 11) Hamming code by adding 4 parity bits to 11 data bits, or a (7, 4) Hamming code by adding 3 parity bits to 4 data bits, you could add 4 "parity" letters to 11 data letters, or 3 "parity" letters to 4 data letters, or indeed 6 "parity" letters to 57 data letters; a variety of "parity" computations are possible but perhaps the simplest is to use a character code assigning numbers 0 to n-1 to the possible letters, and use the sum modulo n. (It's entirely irrelevant what n is, but the decoder needs to know the whole code.) This is optimized for situations in which a whole letter at a time is damaged or lost, rather than single-bit errors.

Third, you can run either variant of the Hamming-code approach along various axes. If your text consists of lines of up to 57 characters, for example, you could add 6 parity characters (or 7, for a SECDED extended Hamming code) to each line, or you could divide it into "pages" of 57 lines and add 6 or 7 parity *lines*, each of whose characters would be computed over the corresponding characters in the other lines. This would enable the recovery of entire missing or erroneous lines.

Fourth, you can combine this with the matrix-parity idea; for example, you could compute an extended Hamming code both horizontally and vertically, allowing you to correct up to one error per line, plus up to one line with two or more errors. This is not the most efficient error-correcting code, but it is very simple, and enables a substantial level of robustness.

If you were using this for archival in practice, you might want to put the "parity" lines and columns at the beginning or end of the data, rather than interspersing them as in the canonical Hamming-code construction.

The ASCII character code has some disadvantages as a code to use in this context, since its last position is an unprintable character (DEL) and so are its first 32 positions, except arguably TAB, CR, LF, and BEL. Also, arguably, space is unprintable; certainly it is especially prone to OCR errors. But if you replace the unprintable characters with printable ones — one option would be "$^{N}_{U}$ $^{S}_{O}$ $^{S}_{T}$ $^{E}_{X}$ $^{E}_{T}$ $^{E}_{O}$ $^{E}_{N}$ $^{A}_{C}$ $^{B}_{E}$ $^{B}_{S}$ $^{H}_{T}$ $^{L}_{F}$ $^{V}_{T}$ $^{F}_{F}$ $^{C}_{R}$ $^{S}_{S}$ $^{S}_{I}$ $^{D}_{L}$ $^{D}_{C}$$_{1}$ $^{D}_{C}$$_{2}$ $^{D}_{C}$$_{3}$ $^{D}_{C}$$_{4}$ $^{N}_{A}$ $^{S}_{Y}$ $^{E}_{T}$ $^{C}_{A}$ $^{E}_{M}$ $^{S}_{U}$ $^{E}_{S}$ $^{F}_{S}$ $^{G}_{S}$ $^{R}_{S}$ $^{U}_{S}$ ␣ //", but many others have been used at different times, including accented letters and extra punctuation — then you would have an

error-correction code people could very plausibly discover by hand in an archival document, for example if microprinted.

TeX OT1

One particularly handy 7-bit all-printable encoding for text is TeX's OT1 font encoding, whose translation into Unicode Wikipedia gives as follows, if I haven't screwed it up:

Γ U+0393
Δ U+0394
Θ U+0398
Λ U+039B
Ξ U+039E
Π U+03A0
Σ U+03A3
Υ U+03A5
Φ U+03A6
Ψ U+03A8
Ω U+03A9
ﬀ U+FB00
ﬁ U+FB01
ﬂ U+FB02
ﬃ U+FB03
ﬄ U+FB04
ı U+0131
ȷ U+0237
` U+0060
´ U+00B4
ˇ U+02C7
˘ U+02D8
¯ U+02C9
˚ U+02DA
¸ U+00B8
ß U+00DF
æ U+00E6
œ U+0153
ø U+00F8
Æ U+00C6
Œ U+0152
Ø U+00D8
̷ U+0337
! U+0021
" U+201D
U+0023
$ U+0024
% U+0025
& U+0026
’ U+2019
(U+0028
) U+0029
* U+002A
+ U+002B
, U+002C
- U+002D
. U+002E

/ U+002F
0 U+0030
1 U+0031
2 U+0032
3 U+0033
4 U+0034
5 U+0035
6 U+0036
7 U+0037
8 U+0038
9 U+0039
: U+003A
; U+003B
¡ U+00A1
= U+003D
¿ U+00BF
? U+003F
@ U+0040
A U+0041
B U+0042
C U+0043
D U+0044
E U+0045
F U+0046
G U+0047
H U+0048
I U+0049
J U+004A
K U+004B
L U+004C
M U+004D
N U+004E
O U+004F
P U+0050
Q U+0051
R U+0052
S U+0053
T U+0054
U U+0055
V U+0056
W U+0057
X U+0058
Y U+0059
Z U+005A
[U+005B
" U+201C
] U+005D
ˆ U+02C6
¨ U+02D9
‘ U+2018
a U+0061
b U+0062
c U+0063
d U+0064
e U+0065

f U+0066
g U+0067
h U+0068
i U+0069
j U+006A
k U+006B
l U+006C
m U+006D
n U+006E
o U+006F
p U+0070
q U+0071
r U+0072
s U+0073
t U+0074
u U+0075
v U+0076
w U+0077
x U+0078
y U+0079
z U+007A
- U+2013
- U+2014
˝ U+02DD
˜ U+02DC
¨ U+00A8

This is intentionally missing some mathematical symbols, namely <, >, {, |, and }, as well as some others rarely used in text like \\ and _. TeX generally sets mathematical symbols in a different font (which makes it a bit strange to include some Greek letters but not enough to actually write Greek), and can stack glyphs one on top of the other, so separate codepoints for letters like áéòč are not needed.

Topics

Majority logic with DRAM sense amps

Kragen Javier Sitaker, 02020-12-09 (30 minutes)

Horowitz & Hill explain that modern DRAM column sense amplifiers consist of a cycle of two CMOS inverters (two CMOS inverters in antiparallel) with a pass transistor across them. As long as the pass transistor is passing voltage, the inverters are held in their metastable state, with their input connected to their output, producing shoot-through current the way CMOS does. But when the pass transistor is opened, the inverters become a latch precariously balanced in its unstable equilibrium state, and the tiniest bit of charge dropped on one node or the other can determine which direction the latch falls.

This is done by connecting one of the bits of memory on the column to the column sense line, and the other to a reference threshold voltage. The capacitor pulls the sense line either up or down by the tiniest bit, and that sets off a positive feedback slippery slope in the newborn metastable flip-flop, which pulls the column sense line all the way down to 0 or all the way up to 1.

This has the salutary effect of fully charging or discharging the capacitor, thus refreshing the bit of memory; so the commonly-repeated line that DRAM has destructive read, like core, is only sort of true — the refreshing action is inherent to the way the sense amplifier works.

It occurred to me that this is a potentially interesting design for McCulloch–Pitts sequential logic, for SRAM, and for op-amp design. Probably there's some flaw in these ideas that explain why they haven't been pursued previously.

McCulloch–Pitts sequential logic

The LGP-30 manual includes the full logic design for the machine; each of its flip-flops (what we would today call latches) has a Boolean function that tells it when to transition to 1 and another Boolean function that tells it when to transition to 0, which are written out in the manual so you can fix it when it breaks. When neither of these functions is true, it retains the same state it had previously, because that is the natural state of a latch. These functions were computed monotonically with diode logic from signals already available; the flip-flops provided not only all the sequentiality needed, but also all the inversion and all the amplification.

In an analogous fashion, you could run three lines to a "neuron" made thus of two inverters and a pass transistor: two input-output lines and a "clock" (in quotes because it's level-triggered, not edge-triggered) to control the pass transistor, which holds the neuron's input-output lines at a metastable and identical level. When the pass transistor is deactivated, whatever differential external drive is present on these input-output lines will be amplified into a Boolean 1 or 0.

In CMOS this neuron is 5 transistors, and the pass transistor is smaller if it's N-channel, but can be P-channel instead to get an active-low "clock".

This neuron has four states:

- Railing high, in which the pass transistor is open, and it drives its positive I/O line high and its negative I/O line low.
- Railing low, in which the pass transistor is open, and it drives its positive I/O line low and its negative I/O line high.
- Erased, in which the pass transistor is closed, and it drives both its positive and negative I/O lines firmly to their equilibrium voltage.
- Sensitive, in which the pass transistor is open, but the I/O lines are still at their equilibrium level; this state is metastable, and at this point it starts transitioning to either high or low, depending on the signed difference in the drive currents impinging upon them from the outside world.

A simple way to use it is to hook its input lines to a weighted-summing node driven by outputs of some set of neurons activated by an earlier clock phase; resistors are one way to accomplish this. For example, given four neurons A, B, C, and D, which have eight input-output lines A, /A ("A inverted"), B, /B, C, /C, D, and /D, you can connect A-10k-D ("pin A to 10-kΩ resistor connected to pin D"), B-10k-D, C-10k-D, /A-10k-/D, /B-10k-/D, and /C-10k-/D. First you clock the A, B, and C neurons, driving them out of their metastable state into railing, while leaving the D neuron in its metastable state. Now the D line, though held metastable by the D neuron, is being driven by A, B, and C to the average of their values; if 1 is 3.3 volts, then they're trying to drive D to either 0, 1.1, 2.2, or 3.3 volts. The /D line is being driven by the inverted signals to either 3.3, 2.2, 1.1, or 0 volts respectively. So, when we finally clock D and let it go to a stable state, it will compute the majority rule of the other neurons. At this point we are free to drive the other neurons back to a metastable state with the clock signal.

At the level of state transitions, this is closely analogous to Merkle's buckling-spring logic, but is not adiabatic or reversible; the resistors are constantly dissipating energy, and so are the push-pull transistors in the metastable neurons. It's similar to the McCulloch–Pitts threshold-based neuron model Turing used to explain the Pilot ACE's logic design, but with a clock.

If we then hook up D in a similar way to drive the inputs of one or more other neurons, a resistive path exists directly from A, B, and C to those other neurons, which complicates our reasoning somewhat. Their drive is somewhat attenuated, because it has to go through two resistors instead of one, but it's there. However, if we suppose the output impedance of the inverters is on the order of 10Ω while the coupling resistors are on the order of 10kΩ, then A, B, and C together can only pull a low D output up by about 10mV, or a high D output down by only about 10mV, if the power-supply voltage Vcc is 3.3V.

However, it may be inconvenient to use such powerful inverters and such weak coupling between neurons, because the coupling needs to be strong enough to reliably overwhelm random variation in the metastable feedback currents once the pass transistor is opened. (Also,

if you're doing this on an IC, large-value resistors are massive space hogs.) We might want the inverters' output drive to only be a little stronger than the coupling resistors. In that case, here are some ways to reduce the extra parasitic coupling described above:

• If we drive A, B, and C back to a metastable state once D is properly latched, their drive will just attenuate D's drive rather than adding uncertainty to it, because they've been erased before we clock the neurons that depend on D. We just have to not clock them back out of the metastable state before we're done using D's results.
• Up to a point, we can use progressively more resistance in successive stages; for example, stage-0 neurons can take their inputs through 10Ω resistors, stage 1 through 100Ω resistors, stage 2 through 1kΩ resistors, and stage 4 through 10kΩ resistors. Obviously this cannot continue indefinitely.
• Instead of driving *both* of D's input/output lines from A, B, and C, we can drive just *one* of them, while maintaining the other at a reference threshold voltage, as is done in DRAM sense amps. For example, instead of wiring A-10k-D, /A-10k-/D, we just wire A-10k-D and connect /D only to neurons that are activated at a later stage, /D-10k-E for example. This way, D mediates all the communication between earlier and later stages, and when it's active, that communication is almost entirely blocked. As D is making its decision, the /D line is pulled toward the equilibrium metastable voltage level by the neurons it will later drive, since they are still being held in the metastable state. This scheme necessarily gives us a single layer of negation at each clock phase; if we want to combine different levels of negation, we need to run wires across phases.
• We can lengthen the bucket brigade. Suppose we have a chain of neurons V:W:X:Y:Z with each one coupled to the next on both rails as described earlier, but with lower resistances: V-100R-W, /V-100R-/W, etc. So if V goes metastable and then transitions into some new state, its influence on Z's input is fighting against not one neuron storing its old state, but three: W, X, and Y.

This parasitic-influence concern is even bigger when we try to get fanout, influencing all of D, E, and F from the output of A — D, E, and F can influence *each other*.

Sometimes it's not necessary to reduce that influence, though.

An interesting to note about this form of logic is that its input-output directionality is entirely determined by the order of clocking. The V:W:X:Y:Z chain described above can copy bits either to the left or to the right, depending on the order of clocking. As Greg Sittler points out, with three-phase clocking, like a stepper motor or an amplifying digital CCD bucket-brigade; we can store one shiftable bit per three neurons. A two-dimensional array wired in such a way can shift bits north, south, east, or west, though at the cost of 9 neurons per bit and 9 potential clock phases.

Although this is already enough for universal computation (with elaborate preprogrammed clocking schemes), coupling between the neurons need not be limited to goofy McCulloch–Pitts weighted sums.

Pass transistors between stages in a bucket brigade can give us

bidirectional shift registers with only two neurons and two pass registers per bit, rather than three neurons and three resistors; we just clock the pass transistors to prevent the bits from going in the direction we don't want. (And, for two-dimensional shifts, the advantage is even larger, requiring 4 neurons and 8 pass transistors per bit.)

Diodes between neurons, rather or in addition to than resistors or pass transistors, permit an input coupling to be stronger in pullup mode or pulldown mode rather than symmetric, which permits more compact logic than if you were to build it without diodes. For example, given A->-B, C->-B, (D-<||E-<)-100R-B (D and E each have a diode down to them from a point which is connected to B through a 100-ohm resistor), F-1kR-B, then when B is in its metastable "receptive" state, it will be pulled up if either A or C is high; if not, either D or E can pull it down; if none of these conditions hold, F drives B.

Naïvely you might think that such diodes would determine the direction of data flow, but of course they don't; this example configuration allows B to strongly pull down A and C whenever B is low or metastable, which may not be desirable. You can fix this thing with tricks like (A->||C->)-100R-B, (D-<||E-<)-200R-B, which still allows a high on either A or C to override D and E.

This limited kind of diode logic is not capable of doing the full universal-up-to-monotonicity logic that traditional diode logic can do.

Another very interesting possible way to couple neurons together is via small capacitors, but this is tricky because it seems to be inherently pretty glitchy. Consider, for example, (A-1pF||B-1pF||C-1pF)-D, where A, B, C, and D are the positive I/O lines of four neurons, and suppose C and D are initially held in the equilibrium state, suppose 1.6V, while A is 3.3V, B is 0. So the capacitors on A and B have been charged up to 1.6V, but in different directions, while the capacitor on C is discharged to 0V. If we then open D's pass transistor to make it sensitive, then it will register whichever of the following events happens *first*:

- A transitions to its equilibrium voltage, producing a 1.6V negative voltage spike. This spike would have a time constant of 10 ps, if the 10Ω-output-impedance hypothesis from earlier holds, if the capacitor were getting discharged through A's output impedance. However, this rapidly drives D to 0, so in fact the voltage across the capacitor rapidly returns to 1.6V in the same direction as before.
- B transitions to its equilibrium voltage, producing a 1.6V positive voltage spike that drives D to 3.3V; this is otherwise equivalent to the case for A.
- C transitions to either 0 or 3.3V, producing a 1.6V positive *or negative* voltage spike which then immediately propagates to D. As before, the coupling capacitor between C and D doesn't change its state of charge.

So a group of such capacitor-coupled neurons that has *just* been sensitized (by opening their internal pass transistors) is like tinder for the least little glitch, which can propagate along it like wildfire in any

direction, being inverted wherever the neighbor connections are via the negative outputs. It's like a set of dominos that can be automatically re-erected. The pulse's propagation is slowed if there are coupling capacitors to the I/O lines of other neurons that are *not* in a sensitive state; these capacitors must be charged or discharged through the series output impedances of the transitioning neurons and the non-transitioning neurons, so the time constant would be about 20 ps under the same assumptions as before.

(This is suddenly far afield of McCulloch–Pitts neurons!)

Such a pulse can be originated by a "destructive read" like that described above on A or B, where by reopening their pass transistors, we get a pulse that tells us what the neuron's state used to be.

One reliability concern: in the case where a coupling capacitor must be charged or discharged because it's driving the input of a gate in a stable (0 or 1) state, then until that happens, a voltage spike is produced in the insensitive neuron that goes beyond the power rails, 1.6V beyond in this case, so the circuitry needs to be designed to not explode under these circumstances.

If the internal pass transistors in this capacitor-coupled system were driven as before by stupid canned clock-phase signals, this system would not be very interesting. However, we can also drive the pass transistors from neuron outputs, so a neuron transitioning to 0 or 1 can make one or more neurons become sensitive or insensitive. It's important in this case that the sensitivity state be determined both when the controlling neuron's output is 0 or 1 and when the neuron is at its equilibrium voltage, whether by altering the threshold voltage of the pass transistor or by some other means. Specifically, I think normally you want the pass transistor to remain open when the controlling neuron is at its "erased" or equilibrium voltage, so that you can make a decision about whether to erase the controlled neuron without actually erasing it in the process.

I think this ability to enable or disable spike propagation in neurons from the state of other neurons is all you need for universal computation, although it may be somewhat complicated by the fact that erasing the neuron's previous state also produces a spike.

You could consider using additional coupling pass transistors in series with the coupling capacitors: A–1pF–Nch–B, say, where Nch is the drain and source of an N-channel MOSFET. However, this capacitor will act like a bit of DRAM: whenever the MOSFET is open, it has a stored charge from the difference between A and B the last time the MOSFET was closed, unless it has been a very long time — tens of microseconds or more, depending on the temperature and physical construction. So, if the voltage difference between A and B isn't the same as it was when it was last opened, closing the pass transistor produces a voltage spike on both A and B. If one or both of them are in a sensitive state, this could cause them to transition!

That behavior might be useful, but when it isn't, we can be sure to only close these inter-neuron pass transistors when both A and B are in their "erased" state; that is, their internal pass transistors are already closed. (It would also be okay if they were in their high or low state, but that is harder to guarantee.)

Note that this means that the capacitor is storing a sort of trit:

when we close the coupling pass transistor, it can produce either a positive pulse, a negative pulse, or no pulse. But the "no pulse" case may not be reliable, since slightly different operating voltages between adjacent neurons might make it a small pulse instead.

The possibility of using coupling pass transistors in this way suggests another way to produce two-neuron-per-bit bidirectional shift registers with these neurons. To shift right, suppose that initially the bits are in the left neuron of each two-neuron cell, the right neuron is erased. The coupling transistor between the neurons is closed, copying the bit onto the capacitor, and then opened. Now we erase the left neuron, so both neurons are erased, but the bit is preserved on the coupling capacitor. Now we sensitize the right neuron, then close the coupling transistor again, thus storing the bit in the right neuron — but inverted! To finish shifting the bit into the next cell, we do the same sequence of steps using the right neuron, the left neuron of the cell to the right, and the coupling capacitor and transistor between the cells rather than within them. A precisely analogous set of steps permits shifting to the left instead.

There might be a way to do this with only one neuron and one coupling capacitor per bit, but it isn't obvious to me what it is.

Diodes may still be useful in this capacitor-coupled world; junction diodes chop a 1.6-volt spike down to only about 1.0 volts, but that's still plenty to knock over the domino, and they may be able to prevent the propagation of back-biased spikes. However, it may be necessary to maintain a DC bias on the diode, or to use a PIN diode, to lower the back-biased diode's capacitance enough to really block back-biased spikes.

I suspect weighting different connections in the capacitor-coupled world with various sizes of capacitor is not going to be useful, because once a neuron goes sensitive, its new state will be determined by the relative *timing* of different pulses — first come, first served! It won't be determined by the relative *sizes* of different pulses unless they arrive in *very* close proximity. So you might still be able to do a sort of priority decoding by varying *delays* rather than *impedances*, and you might be able to provide delays by bypassing some neurons' I/O lines with additional capacitors to ground (or, equivalently, Vcc), thus slowing their transitions.

The power use of these neurons, if built with MOS, is negligible in the High and Low states, but heinous in the Erased and Sensitive states, because we're deliberately provoking shoot-through. However, in the resistor-coupled McCulloch–Pitts version described first, we *also* have constant power use whenever a High I/O line is connected to a Low I/O line, even indirectly (perhaps they are two inputs to a neuron used as a gate). The diode-coupled version has less of this, and the capacitor-coupled version eliminates it entirely, dissipating energy only in Erased and Sensitive states and transitions, just like regular CMOS.

If resistance is interposed in one or both of the positive feedback paths within the neuron, it becomes possible to provoke state transitions without an intermediate Erased state, simply by overpowering the weak feedback drive with a lower-impedance external drive. This potentially permits the use of the technique

described earlier for the LGP-30, with separate Set and Reset logical formulas, which might be coupled to the weakly-feedback-driven node using diodes, so they can only pull it up or down strongly. Rather than interposing resistance, you could just use especially wimpy drive transistors in the inverter, which has the additional flexibility of permitting asymmetric drive capabilities — so, for example, a strong low signal coming in on the I/O line can overpower the feedback and reset the neuron, but a strong high signal cannot, as if that signal were diode-coupled.

SRAM

Normal CMOS SRAM cells have 6 transistors or 8 transistors ("6T" and "8T" respectively). These neurons are 5 transistors. I suspect that adding a pass transistor to a data bus line gives us precisely the standard 6T SRAM cell, but I need to check this out.

Op-amps

So what happens if we take one of these neurons and try to use it as an analog comparator? We'd probably like to have separate input and output signals; let's use a couple of pass transistors and an S&H capacitor on each input to achieve this. We wire IN0-Nch1-(20pF-GND||Nch2-A) and correspondingly IN1-Nch3-(20pF-GND||Nch4-/A) (that is, connect an N-channel MOSFET Nch3 from IN1 to a node connected to ground through a 20-pF capacitor, and to the I/O line /A via another N-channel MOSFET Nch4). To sample, we open the MOSFETs Nch2 and Nch4, then close Nch1 and Nch3 (with make-before-break ordering, probably). Then we wait for the dual 20pF capacitors to ground to charge and discharge (they could probably be a single 20pF capacitor between these nodes, really). Once they're adequately charged (failure to wait long enough will induce excessive hysteresis) we open Nch1 and Nch3, erase the neuron, sensitize the neuron, and simultaneously close Nch2 and Nch4. The differential voltage thus imposed on A and /A will kick it out of the Sensitive state into either High or Low.

At this point we can return to sampling again, while the output state continues to rail in A and /A until the clock says it's time to do another comparison.

An interesting thing about this setup is that its input offset voltage doesn't depend on transistor matching in the usual way, although a mismatch in the equilibrium voltage between the two inverters will provoke some offset — the initial erased voltage will be somewhere in between the two inverters' equilibrium voltages, so one of them will tend to start rising and the other falling. But this offset can be arbitrarily ensmallened by weakening the inverters' output drivers (as before, possibly by putting an output rsistance on them) at the cost of speed, or embiggening the S&H capacitors at the cost of bias current. The R_{on} of the erasing pass transistor will also provoke some imbalance during the erase state — if one inverter has a stronger high-side output drive, it will tend to pull its end of the erasing transistor up, and *mutatis mutandis* if it's the low-side output drive that's stronger. This R_{on} also means that the inverter with the lower

input threshold will tend to have lower output voltage, which further reduces the precision of the initial unstable equilibrium.

The input bias current of this setup is spiky, coming whenever we take a new sample, so we can reduce it arbitrarily low by taking samples less often, a choice which also enables us to completely prevent oscillations above half the sample frequency. If our rails are ±15V, then every time we sample, we briefly spike one input to -15V and the other to +15V. If the input impedance is 1kΩ, then these spikes would decay with a time constant of 20ns. If the inputs are equal (as is usual for an op-amp), these spikes will total 30V and 0.6 nanocoulombs (regardless of what the input impedance is); if we use a sampling frequency of 1MHz, that's up to 0.6 mA of input bias current, which is horrendous, and all of which is balanced and therefore also an input offset current, which is worse. But if we reduce the sampling frequency to 1kHz, we reduce it to 0.6 µA, which is reasonable if not excellent.

We could perhaps reduce the magnitude of these voltage spikes by only drawing from the S&H capacitors until the neuron is safely out of equilibrium, then opening Nch2 and Nch4; 500mV out of equilibrium is probably plenty, and if there's a bit of resistance at Nch2 and Nch4, we might be able to keep the offset voltage imposed on the capacitor down to 50mV or so. That is, once the horse is safely running free, slam the barn door shut as fast as you can. This would drop the input bias current by three orders of magnitude, down to a reasonable 0.6 µA or good 0.6 nA in the example, assuming the input transistors are perfect. (0.6 nA at 30 V would require them to have 50 GΩ of impedance when turned off, which is unlikely.) With a bit of cleverness, you might be able to break the circuit at roughly the point where the sampling caps have been "refreshed" by the amount of charge they lost when they initially unbalanced the neuron.

It might make more sense to just use JFET followers to drive the S&H inputs, but then of course we get back into the question of transistor matching.

So far we only have a clocked differential-output analog comparator whose output spikes down to zero on every clock cycle, a problem that could be masked by a pass transistor, capacitor, and drain follower glommed onto its output, carefully clocked to exclude the spikes. To transform it into an op-amp, we need to filter its output. The *ideal* way to do this might be with a SAW or coax delay-line filter to notch out the clock frequency and all its harmonics, plus a good number of subharmonics. But it's probably more practical to eliminate the spikes in the time domain by the method described above, then use an RC filter to get some crude averaging.

(In some cases it might make more sense to feed the raw railing output to a power amplifier, LC low-pass filter the output, and use *that* for the negative feedback; that way you get a class-D amplifier instead of an op-amp.)

I'm not sure what the impact will be of gate charge injection in the S&H circuit, but it seems like it could be a potential problem.

The potential open-loop gain of this sort of amplifier is ridiculously huge, because by reducing the clock rate the input power can be reduced to almost arbitrarily low levels. It seems plausible that it

could detect an input difference of 1 mV (indeed, if it can't, it's not going to be very useful as an analog op-amp); if it's getting 1 nA of offset current there, it's consuming a picowatt of input power. But a small amplifier of this design could produce an output swing of 30 V at 500 mA, 15 watts, in response to that millivolt input swing. That's 16 orders of magnitude more power, which in some sense corresponds to 8 orders of magnitude more voltage, which is how open-loop gain is normally measured.

Given how commonplace it is to use sample-and-hold circuits, analog comparators, and D flip-flops, this is surely a family of op-amp design that has been tried previously, even if the analog comparator in question wasn't specifically based on the antiparallel-shorted-inverters approach.

Topics

Truth table search

Kragen Javier Sitaker, 02020-12-09 (11 minutes)

If you read down the output column of the NOR truth table, you get 1110, binary 14. The corresponding column of the AND truth table is 0111, binary 7. In C-derived languages, 14 ^ 7 = 9, binary 1001, the output column of XNOR's truth table, and 14 & 7 = 5, binary 0110, XOR. Correspondingly (A NOR B) XOR (A AND B) = A XNOR B, and (A NOR B) AND (A AND B) = A XOR B.

In this way on a modern CPU we can combine truth tables of up to 6 inputs in a single instruction, computing the result of combining two Boolean functions with a Boolean operator. We could imagine building up a database of optimal circuits for all Boolean functions of up to, say, 5 inputs (32 bits per truth table, thus 4'294'967'296 possible functions.) Then, if we want to know how to compute any of these functions, we can just look up the optimal circuit in the database. We might have different databases for different design criteria; for example, the circuit with the smallest number of NAND gates might not be the one with the smallest propagation delay, and if we additionally have AND or XOR gates we can produce smaller circuits in many cases.

It might be reasonable to build such a database for 6 or more inputs if we could exploit some kind of simple normalization. Some functions, such as XOR and "threshold" functions like AND, OR, and majority, don't care if you permute their inputs, but other functions do. For three inputs, for example, ~A & (~B | C) gives truth table 0xd0, but other permutations of the same inputs give truth tables 0xb0, 0xc4, 0x8c, 0x8a, and 0xa2. With five inputs you could have as many as 5! = 120 functions that are equivalent under permutation of inputs, and presumably most possible functions don't have the kind of special symmetry that XOR have, so you could imagine that taking advantage of such input permutations would reduce the database size by two orders of magnitude. Then, before probing the database, you'd have to permute the bits in your truth table into the normal order — a simple criterion would be to take the truth table with the numerically lowest value, in this case 0x8a, (~C & (~B | A)).

Another kind of normalization that would be useful in many cases is a different kind of bit permutation: negating some or all of the inputs. If you negate the most significant bit of the input, for example, you swap the first and second halves of the truth table. In contexts where the negated input is just as easily available as the non-negated input, this negation comes for free. Even when it doesn't come absoutely for free, this may be worth doing, because the cost is not large in many contexts. A counting argument suggests that this reduces the number of possibilities greatly: for 5 inputs we have 2^{32} possible functions. Of these, some depend on all of their inputs, while others depend on 4 inputs or less. The ones with 4 inputs or less can be expressed with a 4-input truth table plus a position (0, 1, 2, 3, or 4) for the ignored bit. There are only 2^{16} 4-input truth tables, and so only $5{\cdot}2^{16}$ 6-input truth tables that ignore a bit; that's less than 2^{19},

which is 1/8192 of the total search space. The ones that depend on all 5 inputs usually (handwaving here!) have $2^5 = 32$ versions equivalent under input negations, and (I think) all have at least 2 equivalent versions. So we should expect a reduction of a factor of at least 2 and I think nearly 32 in the database size by this approach.

That's not a rigorous argument, but it is at least strongly suggestive. Moreover I think these two kinds of normalization are complementary, and we should get at least a factor of 1000 database compression by combining them.

Also, of course, before probing the database we can check to see if we *do* have any don't-care inputs. If so, we can probe a much smaller table, probably in RAM.

I thought about also tabulating all the especially low-cost circuits for a larger number of inputs, but I think it may not be practical. Consider tabulating low-cost circuits with 6 bits of input: you need at least 5 minimal-cost gates, if they're binary gates (or your circuit has less than 6 bits of input), and so at a minimum you have the binary trees on 6 inputs, maybe multiplied by the 5th power of the number of types of gates. I think you exceed the number of possible 5-bit truth tables by a lot rather soon. (But I haven't done the calculation.)

If you have a 32-entry truth table to probe for that contains a few don't-care entries, the brute-force way to handle them is to probe the database for the 2, 4, 8, 16, etc., entries they correspond to, normalizing each possibility in turn. If the database *isn't* normalized in the way I described earlier, you may be able to get some mileage out of contiguity properties of indices: by permuting the truth table so that the don't-care bits are toward the end of your search key, all the entries that could match will be physically close together in the database index. This would permit larger numbers of don't-care entries in the search key without totally losing locality.

As for actually building the database, a simple approach is basically Dijkstra's algorithm, a breadth-first search using a queue: initially enqueue the trivial circuit (for example, for five inputs, a circuit with nets n0=0, n1=1, n2=in0, n3=in1, n4=in2, n5=in3, n6=in4), and upon dequeueing a circuit, do the following:

- compute the truth table of the last net and the cost of the circuit;
- normalize the truth table;
- check to see if the normalized truth table is already in the database with an equal or lower cost, and add it if not;
- compute all possible single-gate extensions of the circuit (e.g., n6 = n2 NAND n4) and enqueue them.

Probably some kind of circuit-normalization step would be useful to avoid enqueuing trivial permutations of already-enqueued or even already-processed circuits. Also, if the cost metric is something more complex than just "number of gates", you might want to use a priority queue (by cost) rather than a regular FIFO queue. To find out when you're done, you can maintain a second database of still-unachieved normalized 5-input truth tables, removing items from it as you find them.

I thought about trying to just do the graph traversal on truth tables,

stored for example as 64-bit integers, rather than circuits — so, going back to my first example, if you knew that computing the truth table 14 takes c(14) = 1 gate, and computing 7 takes c(7) = 1 gate, then you can XOR them together (assuming XOR is one of your primitive gates) and get 9, with cost c(14) + c(7) + 1 = 3 gates. This runs into two problems:

• It doesn't take structure sharing into account, which becomes important for circuits of more than two inputs, so the costs it computes are only upper bounds.
• You have to iterate over the entire database of known truth tables every time you add a new truth table in order to enqueue all the successor circuits, and it isn't obvious how to do that in an efficient way.

So much for tabulating forward-chaining search results for a meet-in-the-middle attack. How can we chain *backwards*, though?

One omnipotent approach, ably explained by Darius Bacon in The Language of Choice, is the binary decision diagram: by choosing a Boolean variable to split the universe in half with first, we reduce the number of possibilities by half. So if we have a database of optimal circuits for all 5-bit-input Boolean functions, and we want a 6-bit Boolean function, we can pick one of the 6 bits to split our truth table in half with, probe the database twice, and combine the results with a MUX.

Moreover, we can quite plausibly do this six times to see if any of the six gives us a better result. On an SSD each probe might take 100 µs, so this might take 1.2 ms, while on spinning rust it might take a second.

This kind of MUX-based approach is probably reasonable up to somewhere around 3 more bits of input, so up to 8 bits of input if we have a table of circuits for all 5-bit functions. It's still fast and guaranteed to work at that point and well beyond, but beyond that I think it's going to usually synthesize circuits that are worse than optimal by an order of magnitude or more.

I'm not sure how else to do backward chaining. Maybe you could consider partitioning the input bits into subsets. You could divide 9 input bits, for example, into 1 bit and 8 others (9 ways), 2 bits and 7 others (36 ways), 3 bits and 6 others (336 ways), or 4 bits and 5 others (4536 ways). It might turn out that some single-bit function of those 5 bits can be combined with the other 4 bits to produce the 512-row truth table we're looking for.

Somehow for backward chaining we need to be searching for *simpler* component functions, truth tables that are functions of fewer inputs. Large blocks of zeroes or ones suggest the possibility of AND or OR with a function of fewer bits — then on the other input of the AND or OR those bits become don't-cares.

Topics

Yablochkov arc cutter

Kragen Javier Sitaker, 02020-12-09 (1 minute)

A Yablochkov candle was an early kind of arc light, consisting of a strip of plaster with two carbon rods running along its edges and shorted together with a fuse wire at one end. By connecting line voltage to the other end of the carbon rods, you vaporize the fuse wire, initiating an arc, which then continues between the ends of the carbon electrodes, burning down the rods and the plaster at the same speed. Later models included some metal powder in the plaster so you could initiate it a second time.

It occurred to me that if you had a hole through the center of the plaster, you could blow air through it to squirt some of the plasma onto things, thus perhaps cutting them; this gives you a plasma cutter suitable for non-metals. And with a modern constant-current supply (or even a ghetto version consisting of a quartz-halogen lightbulb or ten in series with the carbon rods and a voltage source high enough to strike the arc) you maybe don't even need the fuse wire.

Topics

The Language of Choice, and other languages

Kragen Javier Sitaker, 02020-12-09 (updated 02020-12-31) (10 minutes)

I was reading over Darius Bacon's The Language of Choice yesterday, which is an introduction to binary decision diagrams; chatting with Bacon, a couple of ideas came out that I thought I ought to write down.

Eliminating left recursion

(I'm going to ignore tokenization and thus whitespace here, since it would add more heat than light.)

One is that the language generated by the CFG

```
<expr> ::= <var> | <expr> if <expr> else <expr>
```

is context-free, but left-recursive and ambiguous (`a if b else c if d else e` can be parsed as testing `b` first or `d` first). If you want to parse it using a standard linear-time Packrat parser, you need to factor out the left-recursion and resolve the ambiguity. Bacon suggested two ways of doing this, one right-associative as it should be:

```
<expr> ← <var> (if <expr> else <expr>)?
```

and the other left-associative:

```
<expr> ← <var> (if <expr> else <var>)*
```

Of course, once you have a parse tree, it's a simple pattern-matching exercise to construct a parse tree with a different associativity.

(Of course Bacon is aware he didn't invent left-recursion removal; I just had never understood it until he explained it.)

XXX verify these

In this form the language is missing a little bit of expressivity; it can only express an unbranching chain of single-variable choices, and it can't express negation. You can enhance this with nesting to be able to handle arbitrary boolean functions:

```
<expr> ← ("(" <expr> ")" / <var> / <const>) (if <expr> else <expr>)?
<const> ← 0 / 1
```

Memoization sparsity

(See also The sparsity of PEG memoization utility (p. 769).)

I was thinking that this grammar might be a good example of when you benefit from Packrat's memoization, but in fact it's not, once the left recursion is factored out. I'd like to figure out how to work this

out rigorously: how can we know that a given memo-table entry can never be used? Is there a single canonical answer that's practical to compute, or is it more a question of a series of conservative approximations, progressively less simple?

Perhaps in this case we can observe that no retry-after-backtrack of any callsite of `<expr>` will ever invoke `<expr>` a second time. This is a whole-grammar property because it flows through invocations; we could falsify it for both `<expr>` and `<var>` by, for example, adding this rule to the above grammar:

```
<when> ← <expr> when <expr> / <expr>
```

But if we refactor that rule as follows, we regain the desirable memorylessness property:

```
<when> ← <expr> (when <expr>)?
```

PEG cuts

Mizushima, Maeda, and Yamaguchi published a paper in 2010 where they insert a Prolog-style "cut" operator into PEGs, spelled ↑; the construction `x ↑ y / z` is similar to `x y / z`, but although failures before the cut (in `x`) can backtrack to `z`, failures after the cut (in `y`) do not. Analogously in `(x ↑ y)*` if there is a failure in `y` then the whole repetition fails. They first implemented this in *Yapp*, but in the 2010 paper they explain how they inserted cuts and negative lookahead assertions automatically in semantics-preserving locations.

They say `x ↑ y / z` is equivalent to `x y / z` iff there is some string S such that `x` matches some prefix of S and `z` matches some (possibly different) prefix of S. This is an undecidable problem even in some trivial cases, like where `x` matches everything, so that the problem reduces to whether `z` can ever succeed on any input. So they compute a conservative approximation that lets them insert enough cuts to get "mostly sequential space" on "practical grammars", including "a Java PEG and a JSON PEG" but not "an XML PEG."

They also report that their cut-insertion improves error reporting from PEGs.

Redziejowski wrote a 2016 paper "Cut points in PEG" where he uses a second kind of "cut" ↓, which I'm guessing he got from an earlier Mizushima et al. paper: `a ↓ b ↑ c / d` backtracks and retries `d` only if the failure happens within `b`, so a failure after the ordinary cut ↑ doesn't backtrack, but *neither does the failure before the backwards cut ↓*. I'm not yet sure what this is good for.

Cut insertion doesn't avoid sticking things into a memo table that we are never going to need, but it does allow us to discard potential backtracking points off the stack as soon as we know we aren't going to need them, which allows us to safely discard everything to the left of that point.

Empirical testing of cut insertion

Given a candidate criterion for not memoizing a callsite at all, we could test it by memoizing it anyway, then testing the parser on some

input to see which callsites the criterion erred on — those whose memo entries were expected to be useless, but got used anyway. It's maybe also worthwhile to examine which callsites don't need to *probe* the memo table because it's impossible for a memo-table entry to exist, because no preceding attempt to parse the same text could have invoked that nonterminal at that position. I haven't seen any research on this aspect of the problem, but it seems like it would help a lot to reduce both the time usage and space usage of the memo table.

Left-recursive PEGs

There's a trick due to Warth, Douglas, and Millstein, which lets Packrat parse some left-recursive grammars at the expense of their linear-time guarantee, which I think would enable Packrat to handle the original grammar in PEG form:

```
<expr> ← <expr> if <expr> else <expr> / <var>
```

When it detects a left-recursive call, it initially fails; if it finishes parsing the rule successfully, for example with the `<var>` alternative here, it enters the parse result into the memo table, then restarts parsing, this time allowing the left-recursive call to succeed. So, for example, parsing `a if b else c if d else e`, it would initially parse `a` with the second alternative, and then on a second go-round, use that `a` for the left-recursive `<expr>` invocation and succeed in parsing `a if b else c`, which would replace `a` in the memo table; after restarting a second time, the initial left-recursive `<expr>` requirement would be fulfilled with the whole `a if b else c` from the previous attempt, and the whole expression would be parsed as an `<expr>`, though associating incorrectly.

After a third restart, an attempt to parse an `<expr>` that begins with `a if b else c if d else e` fails, because that is followed by end of input, not by `if`. So we keep the former parse.

Essentially, this left-recursion trick converts a Packrat parser locally from a top-down parser into a bottom-up parser.

I think Medeiros, Mascarenhas, and Ierusalemschy have written a 2014 paper on this question.

Alternative Boolean syntax

An earlier draft of Bacon's article, if I recall correctly, chose the symbols `⊢` and `⊣` rather than `0` and `1` for the Boolean values, and used infix rather than prefix syntax, so (if we interpret `⊢` as True and `⊣` as False) `a if b else c` would be written `a {b} c`, which is `a` if `b` evaluates to `⊢` and `c` if `b` evaluates to `⊣`. That is, `a {⊢} c` evaluates to `a`, and `a {⊣} c` evaluates to `c`. So you got nice identities like `{0 {x} 1} = x`, `a {1 {x} 0} b = b {x} a`, and `a {b} a = a`. I really liked this idea, but he dropped it for the final version of the article as making it less accessible.

Laws of Form

The other clever Boolean notation idea is the one from Laws of Form, which is optimized for handwriting rather than strings of characters; in a string-of-characters form we can use nested parentheses, which indicate negation. One interpretation is that in

LoF juxtaposition represents OR and the empty string represents False, so we can write the whole canonical evaluation ruleset as follows:

```
()() -> ()
(()) ->
```

That is, two empty sets of parentheses ("crosses" in LoF jargon) juxtaposed can be rewritten as a single empty set of parentheses, and a parenthesized empty parenthesis can be rewritten as an empty string.

This is essentially the same as the Sheffer stroke or NOR logic, but with a notation that exploits a homeomorphism between the empty string being the identity element in the free monad and falsehood being the identity element for Boolean OR. (LoF itself is silent on the correspondence between its crosses and logic, so you can equally well interpret the empty string as true and juxtaposition as AND.)

While the notation is pleasantly straightforward for explicit evaluation, my attempts to formulate normalization rules such as CNF normalization as tree rewriting rules for LoF notation were not very fruitful — they seemed to come out a great deal less clear than in traditional Boolean logic, and at the time I wasn't able to get them working.

Topics

Scribal Basic: a 1960s language for the 02020s

Kragen Javier Sitaker, 02020-12-12 (updated 02020-12-15)
(32 minutes)

I was thinking about the BASIC systems of my childhood and what a similar system, tentatively called Scribal Basic, might look like now. The overall emphasis was on making them *easy*.

What BASIC was like

One key aspect was the "IDE": you could interrupt the program at any time, type bits of code, inspect some variables (the "PRINT" command had a shortcut spelling "?" for this purpose, so you could type "?X3" to see the value of X3, a feature important enough that FORTH spent the single-character word "?" on it as well), modify the program, and continue from where you left off. So in a sense you were always running "in the debugger", just as with Smalltalk, Lisp, and FORTH.

Microsoft BASIC-80 had a set of vi-like line editing commands which I couldn't understand, because I couldn't see what was going on, so I would modify the program by retyping lines of code from scratch, with the line number. The 8086 version of Microsoft BASIC improved this because it had arrow keys and modeless screen editing, so you could use the arrow keys to move up to a previous output line (from LIST, say), modify it, and hit Enter, which worked like retyping the line. Because you could replace, say, line 30 of the program to say "INPUT X" by typing `30 INPUT X`, this had the effect of changing the program in memory to say what you'd edited that line to say, as if you were WYSIWYG-editing it through a tiny one-line-long window.

Another aspect of BASIC's easiness was that it had no mutable data structures other than variables. It had array variables, but no way to alias them. There were no records, no containment, no foo.bar.baz, no pointers (other than indices you could potentially use to index arrays). Strings were immutable. This seems to have been important to usability; Joel Spolsky (?) reports that the `with` statement, which in VBA (as in Pascal or JS) imports the contents of a record into the variable namespace, was a significant boon to VBA's usability. (I can't find Joel's post now.)

You also didn't have to deal with data types, and I didn't, except for the distinction between strings and numbers, a distinction which Perl and JS eliminate. More advanced BASIC programmers knew that by defining some or most of their variables as integers they could speed up their programs by one or two orders of magnitude, because of course all the floating-point math was done in software, and by default when you wrote `FOR I = 1 TO 10` you were implicitly doing a floating-point addition and comparison every time through the loop.

There was no scoping; all variables were global, declaration was implicit, and as in Perl, initialization was automatic — unassigned

variables had the value 0, or "" if they were strings. Like the lack of types, this was super bug-prone, but it was also simple — you never lost a variable value because it went out of scope, or had two different variables with the same name, or had code that worked at one point but then stopped working due to an unrelated change giving an uninitialized variable a nonsense value.

BASIC-80 had a one-level parser that didn't need whitespace to recognize keywords, but variable names were limited to only two characters. The consequence was that you could write things like `ifx=ythen300elseprintx` and have it parse. This was terrible for readability but probably also improved the language's "easiness" by making it more forgiving: you couldn't break your program by leaving out whitespace. Relatedly, if you wanted to PRINT multiple things on the same line, you were supposed to separate them with ;, but BASIC-80 (and I think even GW-BASIC) didn't enforce that.

There was a strong distinction between the user program and the system, and the temptation was to think that "learning to program" meant memorizing the capabilities the language provided. BASIC's extensibiilty was almost zilch. GOSUB was the limit; if you defined a subroutine to draw a line, for example, you might invoke it as follows:

```
1450 X1=37:Y1=102:X2=128:Y2=17:GOSUB 3000
```

Compare that to the LINE statement added to Microsoft BASIC at some point (it was present on at least the versions for the TRS-80 Color Computer, the IBM PC Jr, and the Zenith Z-100):

```
1450 LINE (37,102)-(128,17)
```

From the perpective of a 7-year-old, which I was when I started programming the CoCo, that's a huge difference. I learned to define subroutines in Logo when I was 5 or 6; I didn't figure out how to transfer that knowledge to BASIC until after learning Pascal (at 11), or maybe even later.

The other problem created by the strong system-code/user-code distinction was that, even if you did implement the code to draw a line in high-level BASIC, it would be so slow that you could see it drawing each individual pixel, even on the IBM PC Jr, which ran at almost 1 MIPS but only about 0.2 Dhrystone MIPS. So BASIC was too slow to be used as other than a "scripting language" in that sense.

However, the great *benefit* provided by the system-code/user-code distinction was that your BASIC program couldn't corrupt the system's structures, at least until you pierced the veil with PEEK and POKE. When you interrupted your program — a common thing to do to fix bugs you'd just noticed, or to recover from an infinite loop — BASIC would tell you you were at some line number or other, not in the middle of some internal Bresenham-point-calculation routine invoked by LINE. On one hand, this did not facilitate studying and extending the system, but on the other hand, it meant that the behavior you saw was always explicable in terms of your program — you didn't *have* to study and extend the underlying system to debug your program.

(Vaguely related thought: transactions simplify debugging and error recovery, at least as long as they don't make debugging impossible. What if every subroutine call were a nested transaction? It would simplify exception handling, too, since rolling back the transaction would eliminate any need for running user-defined cleanup code. See [transaction-per-call.md].)

So, thinking about BASIC and Logo, I wondered what a modern sort of BASIC would look like, one that was as easy as real BASIC in the ways that BASIC was easy, but easier in the ways that BASIC was hard. And even though I've said above that a lot of the important aspects were the IDE — you were always running the program "in the debugger", you could stop and look at variables and fix it and CONTinue, etc. — I'm going to focus on the purely linguistic aspects here.

Syntax and control flow

From my experience with Logo, I don't think the absence of real subroutines with parameters was really a key thing that made BASIC easier. In fact, I think Logo might have made things *easier* by having named subroutines with parameters.

But if you don't have records or other heterogeneous aggregate data types (tuples or whatever) you need some way for a subroutine to return multiple atomic units of data. In BASIC this was easy, because all the variables were global, so you could just change them. Octave (another champion at making programming accessible to nonprogrammers) fixes this by explicitly listing named return values in the procedure header. A different approach is to make parameters implicitly inout, for example by always using pass-by-reference.

Another way to compensate for not having records is by using closures, but closures can be confusing. But Smalltalk-like "blocks" don't have to be confusing; consider Logo's REPEAT 4 [FD :SIDE RT 90] — it's apparently easy enough for kids to use.

So I propose that Scribal Basic should support CLU-like or Ruby-like block arguments, so you can define `repeat` in the language itself, although learners should be able to lock it so they don't end up in the middle of it when they interrupt the program:

```
[repeat n]
for i = 1 to n:
    yield
```

I think Python's indentation-based syntax with colons has been shown to be easier for learners to understand, but if not, you could spell this as follows:

```
[repeat n]
for i = 1 to n
yield
next i
```

You would invoke this with something like the following:

```
repeat 4:
    fd side
    rt 90
```

Or, if the indentation goes away:

```
repeat 4 {
    fd side
    rt 90
}
```

The block argument is invoked with `yield`, and because it's not a first-class object, it becomes a downward funarg; it can't be captured and stored for later use, so resources can be reclaimed with stack discipline. This does require a little bit of trickiness with the calling convention: the block is running in the lexical context where it was defined, with access to variables like `side`, but the activation record of `repeat` is still active, so `fd` and `rt` get their activation records pushed on top of `repeat`'s. And when we reach the end of the block we must "return" into the body of `repeat`, before it returns to the caller of `repeat` which contained the block.

Because all the parameters are passed by reference, you can pass variables to return values in as parameters, especially useful for IMGUI kinds of things. Because these references are also not first-class values, they also cannot be saved for later, so they do not impede stack discipline for resource management.

Nonlocal exits from block arguments are potentially tricky. If you implement GOTO and RETURN in the straightforward way, three potential problems come up:

- You could RETURN from inside a block without giving `repeat` or similar things a chance to clean up their activation records. They won't leak memory in the usual stack implementation — you just restore the stack pointer to what it was on entry to the subroutine you're returning from — but you could imagine wanting, for example, to unlock a lock or restore some graphics context state. So whatever cleanup code is necessary would need to be activated.
- If you can GOTO from within the block to outside the block, not only could you circumvent such cleanups, you could enter the block repeatedly, which means that there would be more than one activation record on the stack of `repeat` or a similar procedure for the block to potentially return into if it ever manages to successfully terminate.
- If you can GOTO from outside the block to within it, you can end up at the end of the block without anyplace to return to.

The simplest way to avoid all these problems is to eliminate GOTO and RETURN from the language.

Blocks that take parameters could be spelled in a few different ways. At one point I was thinking to use [] (otherwise unused in BASIC) for parameters in general, so maybe you could spell a block that takes X and Y parameters as [X, Y] { ... }, for example. Or you could put them after the colon if you use Python-style indented blocks with colons.

But really you don't need parameters for blocks if parameters are passed by reference. You can just use out-parameters for the higher-order subroutine you're invoking:

```
eachpoint x y:
   print "x: " x "y: " y
```

Here `x` and `y` are variables in scope in the caller of `eachpoint` (perhaps local, perhaps global, perhaps parameters) which `eachpoint` can assign to before invoking `yield`.

In PostScript, we define paths by a series of commands: moveto, lineto, closepath, arc, and so on. We can then stroke, fill, or clip to these paths, and IIRC in Display PostScript we could also handle events in those paths. This sort of thing seems like a good use of blocks for passing complex data down the stack. As a simple example, we could define a subroutine that both strokes and fills a path:

```
[strokefill]
yield              ' stroking is default behavior for discoverability

fill:
    yield
```

And to handle a mouse click event inside a path, perhaps we could store the coordinates and a bitmask of buttons via out parameters:

```
[handleclick x y buttons]
getmouse x y buttons         ' get mouse state from intrinsic

checkwithinpath within x y:  ' standard routine for containment check
    yield                    ' delegating path definition to block passed in

if not within:
    buttons = 0            ' zero out the buttons so caller knows to ignore
```

I'm not sure if a single block argument will be adequate, because you probably want to define things like "if this button is being clicked, do such and so", and that is difficult to express with a single block if both the button geometry and its action are represented with blocks. You could do something grody like this:

```
button bletches:
    if bletches = "draw":
        drawmybutton
    else:             ' it got clicked, since that's the other possibiity
        print "mybutton clicked"
```

But some kind of syntax to pass multiple block arguments, or name them, might be better.

Other cases where you might want more than one block argument include clipping (an operation that takes a clipping path and a drawing — though this is handled in PostScript by having the `clip` operator change the current clipping path until the next `grestore`);

looping constructs with a "final" block; exception handling with a handler block; union and difference of paths; and 3-D CSG.

Naming all the block arguments might be a good idea:

```
[handleclick x y buttons &path]
getmouse x y buttons

checkwithinpath within x y {
    path
}

if not within {
    buttons = 0
}
```

This permits the more straightforward formulation:

```
[handleclick x y buttons &path &handler]
getmouse x y buttons

checkwithinpath within x y {
    path
}

if within {
    handler        ' invoke second block argument
}
```

This might be invoked simply with two juxtaposed blocks:

```
handleclick a b c {
    moveto 100 100
    rlineto 0 200
    rlineto -100 0
} {
    print "triangle clicked" a "," b "buttons" c
}
```

In fact, it might be written with them:

```
[handleclick x y buttons &path &handler]
getmouse x y buttons

ifwithin x y {
    path
} {                        ' second block argument to ifwithin is handler
    handler
}
```

But if we're using the Python-like indented-block syntax, you need some kind of keyword to introduce the block:

```
[onclick x y buttons &path then: &handler]
getmouse x y buttons
```

```
ifwithin x y:
    path
then:  ' second block argument to ifwithin is also introduced with "then:"
    handler
```

Argument separation

Logo and Tcl come down firmly on the side of juxtaposed arguments, but BASIC traditionally separates arguments with commas, for user-defined functions like DEF FNA, for built-in functions like INT and RND, and for some commands like SAVE "FOO",A. Other commands separate arguments with semicolons (INPUT "Name";N$) or a combination (PRINT I;"th month", M(I)). PV-WAVE IDL goes further and separates even the function name from the first argument with a comma: F,X,Y.

The argument in favor of separating arguments with commas is redundancy for error reporting; if `rlineto 0 -100` gets interpreted as `rlineto (0-100)` it could be difficult to figure out why you were getting an error or an incorrect effect, or that you need to write `rlineto 0 (-100)` instead.

Smalltalk uses keywords to separate arguments: ary at: pos put: element, which would read a little better as ary at=pos put=element or ary at: pos, put: element. And I'm already considering that maybe Scribal Basic should use such keywords to introduce block arguments, at least after the first one.

For the time being I'm sticking to simple Logo-like juxtaposition of arguments despite infix syntax, but I'm noting this as a potential trouble point.

Data model

I think the unification of strings with numbers as done in Perl, the Bourne shell, and Tcl is a significant improvement in usability, especially for novices, and worth keeping, although it undermines Scribal Basic's claim to be a Basic. Awk and JS also try to do this, but they do it by guessing when something is supposed to be a number and when it's not, and there are some operations that handle the two differently, notably comparisons and, in JS, `+`. I think this is a mistake.

I also think built-in hash tables (as in awk, Perl, Tcl, Lua, and JS) improve usability a lot, even without being first-class values, as they aren't in Tcl, Perl 4, and awk.

The possibility of passing a nonexistent hash table entry as an argument by reference as an argument suggests a lurking danger of autovivification. This can be ameliorated by not making hash tables first-class values, so the question of what to do when reading a("foo")("bar") doesn't arise, and by adopting the Lua convention for existence: a nonexistent hash-table entry is indistinguishable from one to which nil has been assigned. This is bug-prone but probably better than the alternatives in this context.

This way, if someone says `foo bar["baz"]` and bar doesn't have "baz" in it yet, we can safely insert a nil at "baz" into the hash table `bar`

before invoking `foo` with a pointer to that nil, which it can then set to something else if it wants. However, this pretty picture is complicated by the possibility of needing to rehash the table to expand it before `foo` attempts to mutate it.

This can be avoided, rustily, if it's impossible to insert anything else into `bar` in the meantime, for example because no other reference to `bar` is passed to `foo` or used by a block argument of `foo`. Alternatively, we could pass in a writing thunk rather than a raw memory address, or apply a lock to `bar` to prevent insertion until `foo` returns, a lock the insertion routine would have to respect.

If we're going to write subroutines that process arrays of numerical data, we need some way to pass the whole array as an argument. Traditionally in BASIC this is done, inefficiently, by sharing a global array, while in Algol 60 it was done with call-by-name, allowing constructs analogous to the following:

```
[sum i n item total]
total = 0
for i = 1 to n:
    total = total + item
```

which could be invoked as, for example, `sum i 10 a(i)*b(i) dp` to put a dot product into `dp`.

I think call-by-name is a terrible idea, though I'm not clear that it's really that much worse than implicit call-by-reference. But the alternative to call-by-name is to pass entire arrays by reference, which seems like the right choice:

```
[dotproduct n p q total]
total = 0
for i = 1 to n:
    total = total + p(i) * q(i)
```

Much of the language design is aimed at pretty high and highly-predictable efficiency with a simple compiler: stack discipline for variable storage, limited pointers, no records, and so on. But it seems that if you're going to make the language stringly-typed like Tcl, you're going to have to do some type inference to figure out which variables are really numbers. This is going to be complicated by pervasive mutability and call-by-reference, since potentially any function you call with a variable could change the type of that variable. And any time you invoke `yield` (or a named block argument) that block can potentially mutate any global variable or by-reference argument, including changing its type.

But I think in most cases you can infer at least numeric or string nature for variables, and in, out, or inout mode for parameters. Most subroutines won't take block arguments; most parameters won't be modified; etc.

Scoping

Nested scopes are probably a mistake for novice usability, and probably implicit global is the wrong choice — it would render

disastrous the simple `repeat` definition above, which mutates `i` without declaring it local — especially given the possibility to pass things by reference. The usual annoying issues of producing closures in a loop (do they all alias the same underlying variable?) disappear with downward funargs.

But mutable global variables probably are needed. Ruby's solution of prefixing them with `$` seems like the best tradeoff, avoiding the "action at a distance" effect of explicit declarations.

Imaging model

Making graphics was one of the most important aspects of both Logo and BASIC. Even on the H89 I was making ASCII-art graphics. Nearly all I did in Logo was make graphics, a fact which will surprise anyone who's seen my adult drawings. Other people I've talked to about their childhood Logo experience also talked about making graphics. The graphics capability of IBM PC BASIC, Z-BASIC, and GW-BASIC was what I spent all my time on when I programmed those machines. Proce55ing is popular with novice programmers today and mostly focused on making graphics. And James Hague describes his experience learning to program as follows:

...I can talk about the Atari 800 I learned to program on.

Most games didn't use memory-intensive bitmaps, but a gridded character mode. The graphics processor converted each byte to a character glyph as the display was scanned out. By default these glyphs looked like ASCII characters, but you could change them to whatever you wanted, so the display could be mazes or platforms or a landscape, and with multiple colors per character, too. Modify one of the character definitions and all the references to it would be drawn differently next frame, no CPU work involved.

Each row of characters could be pixel-shifted horizontally or vertically via two memory-mapped hardware registers, so you could smoothly scroll through levels without moving any data.

Sprites, which were admittedly only a single color each, were merged with the tiled background as the video chip scanned out the frame. Nothing was ever drawn to a buffer, so nothing needed to be erased. The compositing happened as the image was sent to the monitor. A sprite could be moved by poking values in position registers.

The on-the-fly compositing also checked for overlap between sprites and background pixels, setting bits to indicate collisions. There was no need for even simple rectangle intersection tests in code, given pixel-perfect collision detection at the video processing level.

What I never realized when working with all of these wonderful capabilities, was that to a large extent I was merely scripting the hardware. The one sound and two video processors were doing the heavy lifting: flashing colors, drawing characters, positioning sprites, and reporting collisions. It was more than visuals and audio; I didn't even think about where random numbers came from. Well, that's not true: I know they came from reading memory location 53770 (it was a pseudo-random number generator that updated every cycle).

When I moved to newer systems I found I wasn't nearly the hotshot game coder I thought I was. I had taken for granted all the work that the dedicated hardware handled, allowing me to experiment with game design ideas.

A common observation of kids (and, to a lesser extent, novice adult programmers) is that they quickly pick up how to use the built-in facilities of the environment, but struggle to build their own abstractions for hierarchical composition, even when they aren't using generalization-impaired environments like BASIC-80. Later in the article quoted above, Hague describes his own process of learning to do this when thrust into "the cold expanse of real programming".

So it's really important that you can do this kind of thing in a trivial amount of code in GW-BASIC:

```
10 screen 2:for i=0 to 20:line(i*31,0)-(0,i*9):next
```

This generates a graceful string-art approximation of a quadratic Bézier curve when you RUN it, which is super cool when you're 8. It's only three bytes longer than a minimal Java program that does nothing at all:

```
class X{public static void main(String[]args){}}
```

Further contrast this "hello, world" program with Swing, and consider the level of novice-accessibility it demonstrates:

```
import javax.swing.SwingUtilities;
import javax.swing.JFrame;
import javax.swing.JLabel;

public class HelloWorldSwing {
    public static void main(String[] args) {
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
                public void run() {
                    JFrame frame = new JFrame("HelloWorldSwing");
                    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                    frame.getContentPane().add(new JLabel("hello, world"));
                    frame.pack();
                    frame.setVisible(true);
                }
            });
    }
}
```

That's over an order of magnitude worse than the BASIC line-art program. I don't think you can get it much more compact than that with Swing, and that's appalling.

So I think Scribal Basic, despite the name, would need to make it easy to draw graphics. But I think most people will quickly get frustrated with the limits of 1980s-style opaque line-art polygons in 4 or 16 flat colors, or even 24-bit flat colors, even for cartoons. So I don't think GW-BASIC-style flood-fill and XORing into the framebuffer is really going to get us very far.

I think instead something like the PostScript/PDF/SVG imaging model is better, at least for 2-D graphics, where you build up two-dimensional paths one at a time and apply stroke/fill/eofill/clip operations to them, with alpha-blending and gradients. (The GLSL fragment shader approach is more powerful but more challenging.) There are a lot of ways for people to build graphics for that model: interactive drawing programs, making JPEGs or PNGs or H.264 frames that are then loaded in, rendering from 3-D models, constraint solvers like SKETCHPAD and SolidWorks, and so on.

But I'm going to focus on the most straightforwardly usable imperative programming approach to defining paths, which I think is

either turtle graphics or moveto/lineto with numerical coordinates. It's fairly straightforward to define one in terms of the other; here's the code for turtle graphics in PostScript I used for Heckballs:

```
% Turtle graphics.

/seth { /theta exch def } def
/rt { theta add seth } def
/lt { neg rt } def
/pd { /turtle-pen {rlineto} def } def
/pu { /turtle-pen {rmoveto} def } def
/here { [ /turtle-pen load  theta  currentpoint ] } def
/return { aload pop  moveto  seth  /turtle-pen exch def } def
/fd { dup  theta sin mul  exch theta cos mul  turtle-pen } def
pd  0 seth
```

But of course the target audience for Scribal Basic is people who can't figure out how to write such an adaptor layer, so you'd want it to be part of the base system and one that they don't accidentally get lost in. In Scribal Basic as described so far, it would be something like this:

```
[seth θ] ' set heading
$θ = θ   ' $ to indicate global

[rt θ]       ' turn right
seth $θ + θ * π / 180

[lt θ]       ' turn left
rt -θ

[pd]         ' put turtle's pen down
$pendown = 1 ' can't use higher-order functions like PostScript

[pu]         ' pen up
$pendown = 0

[fd n]           ' go forward n paces
Δx = n * sin($θ)
Δy = n * cos($θ)
if $pendown:
    rlineto Δx Δy
else:
    rmoveto Δx Δy

[saveturtle pen θ x y] ' can't return a heterogeneous array like PostScript
pen = $pendown
θ = $θ
currentpoint x y   ' currentpoint subroutine from primitive model sets x, y

[restoreturtle pen θ x y]
$pendown = pen
seth θ
moveto x y
```

```
[saveexcursion]       ' do some turtle-drawing in a saveexcursion:
saveturtle pen θ x y  ' to return to where you started when you're done
yield
restoreturtle pen θ x y
```

Though this is not as graceful as the PostScript, and a lot longer, I think it's actually easier to read.

The initialization code maybe needs to get invoked somehow, although by using a `$penup` instead of `$pendown` we could get the right defaults from default initialization to zero.

The higher-order function `saveexcursion` suggests using the block facility as a substitute for PostScript's gsave/grestore, which save the current stroke width, color, point, path, and so on, and then restore them. And, as mentioned above, this would also work for `fill`:

```
fillcolor 128 31 45
fill:
    moveto 100 100
    fd 50
    rt 100
    fd 40
```

Aside from the possibility of providing a fourth α argument to things like `fillcolor`, you could *also* use the block facility to do a drawing with a global α. Also, you could use the same approach to provide double-buffering, scaling or rotation or skewing or perspective distortion, drawing on an offscreen canvas that you later composite in, and so on.

Sound

Similarly, sound was always a big deal for motivating programming, but you can do a lot more sound now on a computer than you could in 1985. In the article of James Hague's I mentioned above, he was setting two registers on his Atari 800 to produce a musical tone, but now a cheap soundcard can produce literally any sound a human can hear, if you have a precomputed CD-DA recording of it.

There are existing DSLs for computer music, such as CSound, SuperCollider, ChucK, Sporth, Faust, and Pure Data. Unfortunately I don't have enough experience with them to venture an opinion as to what subset of their capabilities could be reasonably replaced by a Scribal Basic embedded DSL. Some of them, like Sporth, represent sounds as bits of code that execute (conceptually at least) on every sample; this is not harmonious with the way I've conceptualized Scribal Basic, at least so far, although you could do it if you added some kind of threading. Or you could set a global function as the "sound generator", which would be invoked to generate sound samples from some kind of event loop.

At a minimum, I'd think you need support for playing WAV and ogg files (with a built-in mixer), playing MIDI files (with a built-in soundfont), and playing sequences of pitches and durations computed by the program (using the same path as MIDI). But it would be super

cool to be able to do subtractive synthesis, additive synthesis, FM synthesis, distortion, flanging, pitch bending, ADSR envelopes, portamento, tremolo, vibrato, reverb, digital waveguide synthesis, and custom wavetables (from samples or otherwise).

Topics

Hierarchical state space learning

Kragen Javier Sitaker, 02020-12-14 (8 minutes)

(Probably none of this is new and all of it is obvious to those who study such things, but I'm just beginning to learn about the area.)

Suppose you can observe a time-varying set of inputs or stimuli to some system, like a stirred vat of reagents or a circuit or a vibrating string, and a corresponding time-varying set of outputs. How would you go about automatically "learning" the behavior of the system?

If you suppose that the system is approximately linear and time-invariant, then the output vector y_t is given by some direct-coupling matrix D multiplied by the input u_t plus some output matrix C multiplied by the system's internal state x_t: $y_t = Du_t + Cx_t$. Unfortunately we cannot observe x_t directly, and we may not even know what its dimensionality is, and it may be infinite. But by hypothesis it is evolving in time according to some input forcing matrix B and its own square matrix of internal linear relationships A: $x_t = Bu_t + Ax_{t-1}$. And we can linearly superpose effects from stimuli at different points in the past, so $y_t = Du_t + C\Sigma_i A^i B u_{t-i}$, $i > 0$.

If we factor the feedback matrix A with an eigendecomposition $Q\Lambda Q^{-1}$, the eigenvectors Q give us the "vibrational modes" of the system, and the eigenvalues Λ tell us how fast they decay (or, possibly, grow). We can fold the Q and Q^{-1} matrices into the C and B matrices respectively to reduce A to the diagonal matrix of eigenvalues. Some of these eigenvalues may be small, which means that the vibrational modes in question die away very quickly; unless their coupling to the input and output is particularly strong, these can be dropped, reducing the dimensionality of the model, with little effect on the error.

Given some estimates $\hat{A}$, $\hat{B}$, $\hat{C}$, $\hat{D}$, of the matrices A, B, C, and D, we can compute a residual $y_t - (\hat{D}u_t + \hat{C}\Sigma_i \hat{A}^i \hat{B} u_{t-i})$ that tells us how shitty our estimates are for a given time t, and then we can summarize this residual vector over all times t by, for example, summing the squares or absolute values of the residuals to give us an overall residual shittiness to minimize. There are a variety of standard optimization algorithms that may work to minimize this residual.

In particular I think that if we use the sum of the squares of the residuals, ordinary least squares may work: we just take the derivative of the whole residual expression, set it to zero, and figure out what values of A, B, C, and D that gives us. If there are less degrees of freedom in the observations y_t than there are in the four matrices we're trying to estimate, I think the problem is underdetermined. For example, if we have single scalar observations at 100 points in time y_t for a scalar-input system (i.e., u_t is also one-dimensional), and we suppose that the state space x_t is 10-dimensional, then A has 100 degrees of freedom, B has 10, C has 10, and D has 1, so the system isn't fully determined. But if we suppose that it is 9-dimensional, it is now fully determined, and if it's 8-dimensional, it's overdetermined. (In the underdetermined case, we could add extra penalty terms for, for example, norms of the matrices involved, to get the "simplest"

solution in some sense.)

XXX okay smart guy, how can you derive 100 unknowns from a single equation? you add all the squared residuals together, take the derivative, you have a sum of squared residual derivatives, you set it to zero, that's still one equation. ohhh: you take the partial derivative with respect to each design variable ($A^?_{00}$, etc.), and that gives you N equations — but now we're faced with a different problem... I guess I need to go back and brush up on OLS and linear regression!

Another approach is to use gradient descent: if we initially suppose that A is very small, perhaps a single real or complex coefficient, we should be able to iteratively converge very quickly. If we then add dimensions one by one, optimizing after the addition of each new dimension, we ought to be able to converge relatively quickly, as each new dimension is in some sense being used to account for the error in the previous approximation.

Since, as explained above, we can assume without loss of generality that A is a (complex) diagonal matrix (of eigenvalues), the number of design variables we need to optimize doesn't actually increase quadratically with the dimensionality of the state space, as I said it does above — it only increases linearly. For example, if we have 3 scalar inputs, 2 scalar outputs, and 20 hidden variables in the state space, then D is 3×2, A is a 20-item diagonal matrix, B is 3×20, and C is 20×2. I think B and C need to be complex as well, not just A, but D can be purely real.

It should also be possible to use L1 basis pursuit algorithms in order to handle massively underdetermined systems, where we assume an absurdly large number of state-space dimensions, but privilege extremely sparse solutions. This might require abandoning the eigenvalue-diagonal assumption about A, because, although that's the sparsest A can be, it might impose unreasonable constraints of non-sparsity on B and C.

Of course, no real system is perfectly linear, so even without measurement noise, we'll have a nonzero residual signal, but hopefully we'll also have some kind of estimate of the internal state vector of the system at each point in time, which will presumably be uncorrelated with the residual once our optimization has converged. But if we take the outer product of that state vector with itself, we will get a larger matrix some of whose items may have significant correlations with one or another channel of the residuals, and this will give us a second-order correction to apply to our linear predictor. The kernel trick used in support vector machines is, as I understand it, a more efficient generalization of this approach, and can be applied to learn more general system behaviors.

While you could in theory apply this sort of approach to any black-box system at all, it won't work very well for extremely nonlinear systems like flip-flops. For things like that, a hidden Markov model is probably a better sort of model, and that also has efficient algorithms for learning it; you can combine the two models by having different A, B, C, D matrices for different Markov states. But for mostly linear systems, this linear-first approach might have some useful merits.

You can apply this to control systems (automatically tuning the

model for model-predictive control to the plant), simulation (the plucked-string model alluded to earlier), or system identification (guessing what kind of circuit you're looking at), among other things.

Topics

Programming in the debugger

Kragen Javier Sitaker, 02020-12-15 (2 minutes)

Last night I watched some of the demo videos of Jonathan Edwards's most recent investigations in Subtext: one in which the I/O stream is a filtered view of the program's execution (hiding everything that isn't an input or an output); one in which sequences of steps from the edit history can get packaged up into formulas; and one in which you incrementally build up a set of example scenarios as a sort of test suite as you run the program.

It occurred to me that programming in the debugger, Minsky-style, is kind of like programming by demonstration. You have some values in the registers, and you add an instruction to the program, and continue the program. The instruction runs, but then you run off the end of the program and fall back into the debugger, with the results in the registers. Now you add another instruction and run that. Perhaps you add a conditional jump which isn't taken this time, and restart the program from the beginning with a different input, and see where you end up: off the end of the program, or off the conditional jump into nowhere, at which point you can start programming that case.

This is not a style of interaction that modern programming tools support very well, except for spreadsheets. Bret Victor and Jonathan Edwards, among others, have done a number of explorations of what a less limited programming environment in that direction might look like.

Topics

Transaction per call

Kragen Javier Sitaker, 02020-12-15 (updated 02020-12-23)
(69 minutes)

It looks like a new way to use transactional memory can simultaneously improve programming in a large number of very important ways: improved debugging, simplifying some of the hardest parts of JIT compilation, dramatically simplified error handling, fearless concurrency, improved interactive responsiveness (but I repeat myself), modular blocking on input, transparent incrementalization, simple and fast parsing, and enormously faster generative testing and solving of inverse problems.

How does this work?

Suppose that you have an imperative programming language like Daira Hopwood's Noether in which every function call is associated with a new nested transaction, one covering all mutable variables and other effects, and your normal means of handling errors is by rolling back these transactions. What does that give you?

This seems like a way to mostly cut the knot of error handling and responsiveness, without requiring static bounds of worst-case execution time for your entire user interface.

Debugging

Well, one thing it gives you is radical debuggability: because every function call you enter has to save enough information for backtracking if it needs to roll back. The debugger can see this information, and it can restart the function from the beginning as if it had not started running (Hopwood calls this "reversible execution" in hir 2014 Strange Loop presentation, crediting the idea to a 1973 paper by Marvin Zelkowitz; ze claims that Zelkowitz found time overheads of less than a factor of 2 for PL/I, which features pervasive mutability. Zelkowitz seems to have done his 1971 dissertation, "Reversible execution as a diagnostic tool," on the topic, at Cornell, though I could only find a 13-page tech report). This enables efficient granular time-travel debugging, but also, it's potentially useful simply to look at the pending changes so far made by each of the functions on the stack so far.

And implementing edit-and-continue in the debugger becomes substantially easier under some circumstances when you can restart the function you've just edited.

Also, being able to see which transactional variables are being *depended on* at each level in the call stack is also a potential boon to debugging, sort of like `strace` at a per-function level. This could even permit you to produce an interactively explorable dataflow digraph of the call tree; in a standard bubble-and-arrow diagram, dataflow edges might be displayed as connecting to the lowest visible ancestors in the call tree, which you could interactively explode into self and callees. Other forms of aggregation for debugging (grouping together all calls to a particular procedure, or all calls from a particular callsite) might also be insightful.

JIT support

Rolling back to the beginning of the function and re-executing it is also a particularly simple way to support on-stack replacement (whether deoptimization for debuggability, or optimization to get a speedup on a hot loop that might not run again).

For example, if after entering a slow interpreted procedure, the JIT found that it had spent a lot of time in that procedure without finishing, because it contains a long loop. The on-stack replacement problem is that, even if the JIT compiles a fast native-code version of the procedure, the interpreter is still in the middle of running the slow version. To get the benefit of the compilation, it somehow needs to transform a state of the interpreted version (register settings, program counter, etc.) into a corresponding state of the native-code version. Transactions give us the alternative possibility of rolling back from the state of the interpreted version and starting the compiled version from a fresh slate.

Dynamic deoptimization, as in Self, is just the opposite: it requires transforming the current state of the machine-code program into a corresponding state of the source-code program so the programmer can debug it. This is closely related to the time-travel feature described in the previous section.

Error handling

With a transaction per subroutine invocation, error handling becomes substantially easier. Nonlocal exceptions are especially popular in pure functional languages because cleanup while unwinding the stack is unnecessary; by contrast, C++ had so much trouble with this that the STL wasn't exception-safe for several years after it was written! In fact, if I understand correctly, exceptions are still prohibited at Google, because they complicate reasoning about what happens in failure cases — precisely what kinds of states can result. But in such a transactional system, the transaction system takes care of cleaning up any incompletely made changes. So you don't need RAII, destructors, or special failure handling.

The basic nested-transaction feature doesn't require tracking *reads* of transactional variables, the way Haskell's STM does, only *writes*. That's because there's no need to check a transaction for validity when you go to commit it — no other code could have been running concurrently. You only need to buffer the writes to transactional variables so that you can undo them if you have to roll back. (This is a general property of pessimistic synchronization, and this is just the extreme case of it, as explained later.)

This seems to have been Hopwood's primary concern in the design of Noether.

Fearless concurrency and distribution

As Hopwood points out in hir 2014 Strange Loop presentation, logging writes in this way is also what you need for a concurrent or generational garbage collector.

However, if you do additionally track reads of transactional variables, you can use the transaction system for multithreading with

a guarantee of serializability. This is probably costly unless the language is mostly functional, like Clojure or OCaml, and only slightly imperative, because pervasive Python-style mutability would entail logging a huge amount of read traffic to the mutable variables, similar to the overhead of unoptimized reference counting. The per-call transactions would reduce the cost of retrying in most cases.

There's the question of when the threads of such a multithreaded program would *not* be in a transaction, making their transactional mutations visible to other threads. I think the answer is something like Erlang's top-level process loop, where the process evolves by having its top-level procedure make a tail call to itself, and of course when a thread exits successfully.

Such a system would be sort of like the "dynamic typing" equivalent of Rust's fearless concurrency through lifetime checking: your program's non-interference is checked dynamically at run-time, and corrected if necessary, rather than proven at compile-time. But there is a crucial difference: unlike dynamic type checking, it's not just turning a subtle failure into an easier-to-understand failure; it actually *removes the bug*, thus dramatically simplifying the correct code by factoring the hairy concurrency questions out of the application. So, while dynamic typing typically makes code harder to statically prove correct, this kind of dynamic concurrency checking should make code *easier* to statically prove correct.

A significant feature of this kind of concurrency is that it can be nested and physically distributed over a parallel virtual machine: a "master server" node might own the "home location" of all global variables, while a "pool worker" node might (in the optimistic-sync case) start a top-level transaction that reads them from time to time and then in the end sends a commit message listing all the variables it read and all the variables it's writing, which the master can accept or abort. Meanwhile, the pool worker can create non-global transactional variables that exist only inside its transaction, and farm out work to subcontractor subtransactions potentially running on other subcontractor nodes, proxying their reads of transactional variables through the parent transaction's node.

(To avoid ABA problems, probably a monotonically increasing revision number for each transactional variable depended on should be in the commit message, rather than just the value the variable happened to have at the time.)

Worker nodes can maintain a local cache of cached values for global mutable variables. It's okay if the items in the cache get outdated, because the master will reject the commit message for any transaction that has read an outdated value from such a cache — all that's lost is the CPU time wasted doing work that now must be retried. The system would still work properly, though inefficiently, if such rejected commits were the only way to learn about outdated cached values, but a more efficient way for a wide variety of scenarios is to implicitly add an observer to the variable when processing a read-variable message, such that a single cache invalidation notification will be sent to the reader when the previously-read value has been updated, so the reader can invalidate their cache. Since this is an optimization, it's okay if the invalidation messages aren't reliable,

but for most usage scenarios it's best to discard the observer relationship after sending the invalidation message, so at most one invalidation packet (and one current-value packet) is sent per read packet.

The way that works out differs depending on the access pattern. Global variables that are frequently read and almost never updated are almost always globally cached; after each update, the master sends out invalidation messages to nearly all workers, which respond by retrying a lot of in-progress transactions, which immediately send read messages to get the new values of the variable, so it's effectively a two-packet-per-node broadcast of the new value. Global variables that are frequently written and almost never read are also almost never cached, so each write produces almost no invalidation traffic. Global variables that are frequently written and also frequently read unavoidably produce a lot of traffic and also a lot of retried transactions, unless some sort of pessimistic synchronization is used, in which case they instead produce inefficient serialization.

The cases where this caching/invalidation mechanism is insufficient are the extreme cases where either it results in an unacceptable waste of CPU time in transactions that will abort, where it's unacceptable to have to wait for a cache miss to be served from the master server, or where sending a separate copy of a new value of a popular or voluminous variable to every client is unacceptable. The first case can be handled with pessimistic synchronization (see below) while the other two cases can work by supplementing the usual cache-invalidation mechanism with a "push" mechanism that immediately broadcasts new values of popular variables before anyone asks for them, for example using Ethernet multicast.

This scheme also permits proxies which pass through your global transactions to the real master server (or master server cluster), which look just like a real worker to the master server and just like a real master server to the real workers. The proxies answer almost all variable-read queries from their caches, without bothering the real master server, and when they receive a transaction commit message, they simply forward it on to the real master, then relay the COMMITTED or ABORTED response to the real worker. This is analogous to the scenario described above with a worker node farming out jobs in subtransactions to subcontractors. By this means it is possible to scale horizontally in the same way people do with MySQL readslaves.

These proxies, again like a parent transaction, can run consistency validation code on the state of the database after a transaction, aborting the transaction if the consistency checks fail. This is related to the "integrity enforcement" section below.

Sharding the database of global mutable variables across multiple master servers is somewhat problematic, because each transaction needs to commit or abort atomically. The standard consensus protocols for distributed transactions (two-phase commit, three-phase commit, Paxos, Raft, Chandra–Toueg, Mostefaoui–Raynal, ZooKeeper ZAB, in some cases Nakamoto consensus) can be used. For some cases, you could instead add new "global" mutable variables belonging to a proxy described above, which are visible to everyone

sharing the same proxy, in the same way that mutable variables created within a transaction and not exported are visible to subtransactions.

So, as with the single-machine version of the system, it's important to limit the number of writes to global mutable variables, and in particular contention on them. To the extent that you can instead pass around immutable data structures, for example blobs identified by their BLAKE3 hash, you can reduce the work centralized in the master server. Note that this doesn't necessarily mean you want to minimize the *number of variables* that are global and mutable; if you're building a distributed filesystem this way, for example, you could get by with a single global mutable variable for the root of the filesystem (like how a Git HEAD refers to a commit by hash), but every write to the filesystem would invalidate it and force all existing transactions to be restarted. Instead you would probably want at least one mutable variable per file, possibly one per data block, to prevent concurrent transactions from conflicting, even at the expense of increasing the load on the master.

REST and the continuation-based web frameworks exemplified by Paul Graham's Arc and the Smalltalk system Seaside can integrate with such systems in an interesting way. Consider a web server serving up an HTML `<form>` for changing a field in a database record. If the `<form>` contains a hidden "manifest" field that lists all the transactional variables read to produce the page, along with the relevant values of their version counters, then when the form is submitted, the submit handler can check all of these variables to see if they are outdated, and in such a case produce an error page for the user, thus preventing lost-update conflicts where the user's desired change no longer makes sense in light of something else on the page. However, in practice you'd probably want to limit the scope of these dependencies, so that a change to something unrelated (the number of users currently online, the current time) doesn't produce spurious errors.

This "manifest" mechanism, in a sense, permits the protection of (purely optimistic) transactions to be extended all the way out to untrusted browsers, either with no server-side session state in full compliance with the REST model, or by storing the session state in a time-limited variable on the server identified by a continuation ID.

In summary, transactions, especially per-call transactions, enable the single-system-image programming model to be extended with acceptable efficiency across a distributed network of up to a few thousand nodes, including to some extent mutually untrusting actors, unreliable networks, unreliable nodes, heterogeneous software and protocols, high latency, though with a single root of trust ("there is one administrator," in Peter Deutsch's phrase). They would do so by hiding latency with concurrency, avoiding latency and reducing bandwidth with safe caching including proxies, recovering from failures, and automatically retrying transactions safely after node or network failures.

Optimistic vs. pessimistic synchronization defined

(This section is not specific to nested transaction systems, transactional memory systems, or even indeed to transactional systems at all; it applies to all forms of synchronization in software.)

"Optimistic synchronization" means that things don't block each other; instead you allow transactions to run to completion, and if there's a conflict, the first one to commit wins. This guarantees progress and liveness at the potential expense of machine efficiency. "Pessimistic synchronization" is where you use locks to ensure that you don't waste any work on transactions that would have to be rolled back due to write conflicts. Most systems use a mixture rather than purely one or the other.

Pessimistic synchronization is helpful, for example, for interoperating with systems outside the scope of the transactional system, because transactions only roll back (and possibly have to be retried) if they are buggy and try to commit something erroneous. This way, the transaction system avoids imposing any obligation of rollback on such external systems, and the transaction system itself only needs to support rollback for error recovery.

In general, doing pessimistic synchronization safely requires some kind of static analysis of your transaction code to find out what resources it *could possibly* read or write, so that it won't be started until it can acquire all of them. (This lock acquisition can be atomic, but it's sufficient for it to happen in a deterministic order in every transaction to prevent deadlocks; and doing some computation in between lock acquisitions is actually okay.) To be computable, this analysis must be conservative, so in case of doubt, it will delay your transaction until it can guarantee that it will be able to succeed. In the limit, pessimistic synchronization reduces to no synchronization: acquiring a global system lock, as in Noether and other traditional event-loop systems like Monte, Tcl/Tk, Twisted Python, asyncore, or JS.

This kind of static analysis is generally infeasible (for the transactional system to do, at least) in the context where pessimistic synchronization is most appealing: that of interoperation with external non-transactional systems, or systems that otherwise cannot fulfill a commitment to roll back changes. So pessimistic synchronization tends to suffer deadlocks from time to time, even though this is theoretically avoidable.

Aside from the deadlock issue, pessimistic synchronization suffers from an efficiency problem in the multicore era (which, for transaction systems, began with VAXclusters). If your limiting resource is CPU cycles, then to guarantee *efficient* progress, then *pessimistic* synchronization is the ticket: if a transaction read-locks every mutable variable it reads and write-locks every mutable varible it writes, then you never have to retry anything, so then the only way you can go slower than maximum speed is if you have deadlock or run out of work. And this is important — a system making sufficiently slow progress is effectively indistinguishable from a deadlocked system, as anyone will attest after trying to use a desktop Linux system that's thrashing in swap.

However, by never burning a CPU cycle it can't prove will get committed, pessimistic synchronization fails to take advantage of

available CPU resources in uncertain situations, thus conserving energy at the expense of speed.

Both forms of synchronization suffer from low throughput in situations of high contention, and both can get high throughput in situations where non-contention can be detected. So in both cases the best way to get high concurrency is to keep your transactions short. But optimistic synchronization resolves contention with a strong bias in favor of short transactions, while pessimistic synchronization resolves contention with a strong bias in favor of long transactions; it's easy to get into a situation where your pessimistically-synchronized 1000-transaction-per-second system is processing 1 transaction for 30 minutes.

One interesting compromise is granting a limited-time lease on a variable, which prevents any other transaction from altering it during that time. If your transaction commits while holding the lease, you are guaranteed that nobody has written to the variable in the meantime, so if your transaction has to roll back and retry, at least it won't be because of *that* variable. If it commits while holding such leases on all variables it read, it is guaranteed to not have to retry because of any of them. Similarly, you can grant "write-leases" or "write options" ("put options"?), which prevent anyone from taking out a read-lease on the variable during the given time. So if your transaction has an unexpired read lease on every variable it read, and an unexpired write lease on every variable it wrote, it is guaranteed to be able to commit without retrying. In a distributed system that can tolerate node failures, this is the only kind of lock that can ever be granted; otherwise an unreachable node could hold locks forever, blocking some and perhaps eventually all transactions in the system.

The transaction manager doesn't necessarily have to tell the transactions that it's granting them a lease, and if it does, it can choose the expiry date at will. Leases can be purely an optimization to improve throughput in the face of heavy contention by reducing the fraction of CPU wasted on doomed transactions.

Modular blocking

You might think that this approach would preclude I/O anywhere but at some sort of top-level event loop, at least per thread, since I/O is a side effect. It's straightforward to see how you could buffer up output (maybe logging it for debugging in case of an abort) until the top level is reached, but how could you do that for input?

Fortunately *Composable Memory Transactions* has a solution to taking input: if we log reads, as a multithreaded system would, then an input routine such as getchar() would simply `retry` if no input character was waiting. This would abort its transaction, but the transaction system would know that it would simply fail again if no input character was waiting, since it failed by calling `retry` instead of having a read/write conflict or an error. Its caller has the option (as, one supposes, it would have in the case of errors) to handle the retry by moving on to a fallback case, for example reading from a different input stream. If at some point the whole shebang fails, the transaction system can suspend the thread (and do other work, if applicable) until one of the things it had read before `retry`ing changes. (This is the

point where handling diverges from ordinary errors: if the handler for an ordinary error also fails, you just unwind the transaction stack until you terminate the program.)

This provides, in the words of the paper, "a modular form of blocking" — a thread can wait on a condition variable, or an arbitrary Boolean function of various transactional variables, or anything else that can be shoehorned into the transaction system, including input events — and the functions that do such waiting can be made nonblocking by having a fallback that always succeeds, or combined by falling back from one to the other.

As Shae Erisson points out, this could integrate well with modern event-driven I/O systems like Linux's `io_uring`: a thread reading the event source can enqueue events in internal queues, thus inducing other transactions to get retried.

Safe aborting for guaranteed responsiveness

Another benefit provided by pervasive transactionality — and this one wouldn't require either read-logging or nested transactions — is that any task can always be safely aborted, which eliminates the Sophie's Choice we normally face in event-loop systems where we can get either safety from concurrency problems (by running code in the event-loop thread) or guaranteed responsiveness (by running code in another thread). If an event handler is running when another higher-priority event comes in, we can simply peremptorily discard the current transaction, including the dequeuing of its input event, and launch the handler for the higher-priority event. (A classic case of this is repainting the screen in response to an input keystroke when another input keystroke comes in, which will probably require an additional screen repaint.) Or, if we *do* do read logging, we can run one thread for each concurrently executing event handler, retrying executions as necessary.

This kind of abandonment can be constant-time, but only if the buffered writes from the transaction are not written to their home location; as Hopwood points out in hir talk slides, if the writes are written to their home locations, then rolling back a transaction requires undoing all the writes, one by one. An alternative that provides constant-time, effectively instantaneous, abandonment is to only write the writes to their home locations when a (top-level) transaction commits. This requires every read of a transactional variable to check for a buffered write belonging to the current transaction before falling back to the value from the home location.

This same sort of write-log consultation is also needed for concurrency with optimistic synchronization: if some other transaction might be concurrently reading the home location of a transactional variable, it needs to see the previous committed state, not the state that might possibly be committed. (This could be done by instead having all reads of mutable variables check all active undo logs for old values, but that is even worse.) Pessimistic synchronization is a way to avoid this.

This possibility of abandonment through rollback solves one of the knottiest problems in E-style event-loop object-capability systems such as Monte: in a vat shared between code from mutually

untrusting security domains, it is always possible for one security domain to deny service to the other by running an infinite loop. By providing a guaranteed safe way to abort and retry event handlers, such abandonment eliminates this risk, thus enabling closer and more efficient cross-domain collaboration. (However, you still have to do most of the communication between the domains with eventual sends to get this nonblocking benefit, so it may not be more convenient.)

With virtual memory, one common problem for responsiveness is that when the system starts to thrash, responsiveness for the whole user interface goes to hell, because there's no reasonable way to make progress when your threads are blocked on page faults. If, instead, page faults are handled by failing a transaction as needing to retry — just as if it were blocking on input — it should be possible to try many different event handlers, bringing all of their working sets into memory, and allowing whichever ones can make progress to do so without being blocked by the others that are blocked on page faults. This, again, could be done in a single-threaded event-loop system that just uses one transaction per event handler, rather than one transaction per function. (However, it might make things worse rather than better, and of course requires integration with the OS kernel.)

These approaches could even guarantee hard-real-time event response. Hardware interrupts, or software interrupts such as Unix signals, can be handled in this way. If such hard-real-time tasks are to have strictly bounded response times, though, we must render it impossible for other tasks to delay their progress. On a single-threaded computer this is easy — just don't run any other code until the interrupt handler completes. On a multithreaded computer, such as one with multiple processors or multiple hardware threads, it is necessary to use some kind of pessimistic synchronization to prevent any other top-level transaction from committing that could require the interrupt handler task to rollback and retry — this also makes it safe for the interrupt handler to manipulate the outside world without waiting for its transaction to commit first.

Support for optimistic synchronization and running the interrupt handler as a top-level transaction is all that's necessary to get it to *start* running promptly, and then blocking any possibly interfering concurrent transactions (and any other interrupts) is all that's needed to ensure that it can *finish* running promptly without any retries. When the interrupt handler finishes, the changes it commits may or may not cause other transactions (blocked or not) to have to retry. So it isn't always even necessary to *discard* the work in progress to guarantee responsiveness to urgent events in this way. But buffering the writes of uncommitted transactions in a write buffer, rather than logging an undo-log record and updating mutable variables at their home locations, seems to be necessary for optimistic synchronization, and sufficient for constant-time work abandonment.

I'm not quite sure how precisely we can compute "any possibly interfering concurrent transactions" or whether this benefits from static analysis of the interrupt handler. Clearly if another (top-level) transaction tries to write to a variable the interrupt handler has read, it needs to be at least blocked from committing until after the interrupt handler.

Specifically with respect to screen updates, it would be useful to break up the screen repaint into three pieces: a top-half "push" that runs as part of input processing, which takes a small, bounded amount of time to ensure high input handling throughput to recover from overload conditions; a "pull" that runs as part of the vertical blanking interrupt or even the *horizontal* blanking interrupt, which is higher priority than input processing and also takes a bounded amount of time, and whose reason for being is to allow the top-half push to do less work by using a more efficiently updatable in-memory representation (a scenegraph, a display list, a set of sprite positions, a tilemap, etc., of some bounded complexity; see the notes in Scribal Basic (p. 705) about the Atari 800); and a bottom-half push that is scheduled after input processing, can be abandoned and restarted if new input comes in, and can take unbounded time to more elaborately update the structures read by the pull transaction. For example, the bottom-half push might read in text from a disk file after it's newly scrolled into view, or overwrite an approximate 3-D rendering with a more precise one, possibly more than once in multiple different transactions.

In an OLTP database context, you could imagine handling incoming write transactions ("writes", including record updates, insertions, deletions, and schema changes) by appending them to a journal and scheduling additional transactions to update views and indices ("rebuilds", though presumably incremental). Read transactions ("queries") that consulted a view or index would also need to read through whatever part of the journal was not yet accounted for by the view or index in question. An OLTP workload usually won't work with a hard priority system, since totally starving any of writes, rebuilds, or queries due to a high load of the other two would be unacceptable; the relative priorities of writes, queries, and rebuilds could be adjusted through an internal pricing system, in which writes and queries earn "money" by spending CPU time and perhaps IOPS, writes are additionally billed for the expected losses from slower queries, and rebuilds earn "money" by reducing the expected costs of queries, which is at least in part an option value — Black–Scholes may be the right valuation.

A partly completed agoric OLTP transaction would tend to be able to bid higher for resources than one that hadn't started — if its expected completion time is 2 ms and doesn't change during evaluation, and its expected net earnings are 2 simoleons, it can initially bid 1000 simoleons per second, but after running for 1.5 ms, it can bid 4000. But, if that's not high enough because another job with much higher value has arrived, it's "socially optimal" to abort the currently running transaction and handle the higher-value job.

(This same OLTP approach also applies, of course, to updating source code and computing executable views of it with a compiler or groveling over the update log with an interpreter; this could entirely eliminate the JIT pause problem that plagued Self, if Moore's Law hadn't already taken care of that. Yet people still sometimes wait for rebuilds to finish.)

To support simultaneous OLAP operations on the OLTP database, you could simply run your queries on the most recent available indices and views, including precomputed rollup views, without

taking the still-unincorporated journals into account.

Error values

With regard to error handling, it might be best in most cases for aborted functions to return error values rather than automatically propagating. As long as these error values are either handled (inspected to see what the error is, presumably as part of a conditional) or moved to some kind of storage (for later debugging), automatic propagation woud be suppressed, as in Wheat. But if such an error value is ignored (evaluated in void context, or stored in a variable whose lifetime ends without being tested) it would propagate up to the parent function.

These error values can propagate along the program's dataflow graph, like floating-point quiet NaNs; they only leap over to the control-flow graph if they are "leaked" or "dropped".

XXX add example

Modal reasoning

Another application of transaction rollback is *code search*, as suggested by Hopwood in hir 2014 talk under the heading "confining side effects", based on Joel Galenson's† CodeHint (which cites the Squeak method finder): is there an existing function in my code base that will convert 4 and 66 into "iv" and "lxvi" respectively? How about a composition of two functions? Or five methods? An obvious way to implement such a query is to just run all the functions, or pairs of functions, and see what you get, but to do this safely you need to prevent the functions from looping infinitely or causing destructive side effects. By running them inside a transaction and killing them if they exceed a time limit, you can test them safely.

(Note, though, that this time limit is a potentially deadly inlet through which nondeterminism could enter the system, causing any computation that depends on such testing to be irreproducible; if it counts something like function calls plus backward control flow transfers and is precise, it's safe, but not if it's counting wall-clock time or clock cycles and/or is checked only irregularly.)

A generalization of this is the ability for a program to reason about code's behavior under conditions that do not presently prevail, simply by running it inside a transaction that is then rolled back. This does require the transaction's rollback notification to contain enough information to tell us what we want to know about the code's behavior, but that's probably a requirement for useful transaction failure messages, anyway.

Given this kind of facility, you could reasonably ask questions such a the following: Which methods would write to some field of this object? Is there any live object on which calling the ".open()" method would read the current user ID? What is the object whose ".destroy()" method would return the highest value?

In the debugger context, this kind of automatic cleanup would allow you to view "speculative" executions as well: the hypothetical flow of values through a piece of code, without the risk of corrupting the "true" state of the program under inspection with a side effect.

† and Philip Reames's, and Rastislav Bodik's, and Björn Hartmann's, and Koushik Sen's CodeHint

Memoization and incrementalization

Suppose the transaction for a procedure invocation is logging all its reads and writes of mutable data; if it additionally logs which procedure it is, any closed-over data, and its input parameters, then it becomes possible to use it for memoization — any call to the same procedure with the same parameters and closure data will necessarily perform the same writes and return the same value, unless either one of those reads is out of date or execution is nondeterministic. So it's valid to just perform those writes and return those results without actually running any of the function's code. This is very similar to a build system like `make`, or to Umut Acar's "Self-Adjusting Computation"; it provides a way to transparently incrementalize a computation, so that it can be efficiently re-executed on slightly modified input. Also, it automatically derives a guaranteed-linear-time Packrat parser from an ordinary exponential-time recursive-descent parser.

Moreover, this caching or memoization is still valid *even if the original memoized computation was a child of a transaction that was rolled back*. That is, even computation that was "discarded" can affect the memo table. (This is the same mechanism that produced the Spectre and Meltdown vulnerabilities in Intel CPUs — it can produce a subliminal leak of information.) This means that we can speculatively pre-cache computations we expect to need in the future.

Incrementalization is an extremely important transformation for a few different reasons:

• By reducing the need for manual state management for efficiency, it can make direct programs much simpler. For example, you could implement a word processor as a view function from document state to view state, a window function from view state to pixel state, and an edit function from (document state, input event) pairs to document state, or perhaps even a function from input histories (keystroke sequences) to rectangles of pixels.
• By making coordinate search practical, it can make many programs "invertible" in practice (in the sense that you can in practice find an input that produces a desired output, not in the sense that such an input exists or is unique), permitting the practical solution of a wide variety of inverse problems. The optimization procedure can randomly alter the program's input, propagating the incremental changes through the incrementalized program, in order to converge on the desired result.
• A special case of the former is generative software testing like that done by Hypothesis or American Fuzzy Lop, where the "desired" output is a crash or assertion failure; this is to some extent how AFL works, but because it can only backtrack chronologically, its strategies for exploring the input space are necessarily limited. Once a failure is found, incrementalization also greatly accelerates the test-case minimization process. Additionally, the introspection provided by the transaction system can be used by the generative testing system to guide its search.

• Another special case, one which might not work out, is superoptimization — search over a space of *programs* for the shortest or fastest program that has the desired effect. This shades into the "code search" application mentioned earlier.

In short, incrementalization reduces the need for explicit caching and makes searching over the space of executions immensely more efficient.

As an example of "invertibility in practice", or "solving inverse problems", you could imagine applying a ray tracer like Peter Stefek's incremental ray tracer to solve photogrammetry or caustic design: by searching for an input 3-dimensional scene that closely approximates a movie taken by a moving camera, you can estimate the geometry of a scene. Mitsuba 2, for example, has demonstrated this using automatic differentiation rather than incrementalization. (As I said in Dercuano, I suspect that integrating reduced affine arithmetic into the caching system might make it possible to do this trick much more effectively by permitting limited errors in the output, so that memo table values can be reused even for slightly changed inputs.)

Above I talked about using transaction scheduling as a way to guarantee responsivity for real-time and OLTP systems, in particular allowing updating of indices and views to be deferred to some degree to improve query responsivity. A simpler, though probably lower performance, design is to compute an index (or a view) as the cached result of a giant computation over one or more entire tables, or even the update log. Then, queries that consult this index will first request the index in a cached subtransaction, made out of smaller subtransactions; normally this will be instant, served from the memo cache, but in other cases will require a partial or full recomputation to bring the index up to date.

So, for example, you might have 99000 data blocks in an append-only table, each containing 10 rows. Each data block is an immutable blob pointed to by a separate mutable variable, and there's another mutable variable that's a list of all 99000 blocks. Every append to the table appends a row to the last data block (by copying the other 9 or less rows into a new block), or if it's full, creates a new mutable variable, points it to a block of one row, and adds it to the list. The index on column FOO is an LSM-tree, consisting of a run of the sorted FOO values (and record numbers) of the first 65536 rows in the update log, a run of 32768 FOO values, a run of 512 FOO values, a run of 128 FOO values, and so on for 32, 16, and 8. So when a new row is added, maybe a new run gets added, or normally the smallest few runs get jiggered around a bit in the next query, but the 65536-item run and the 32768-item run are returned immediately from the cache rather than being recomputed.

This scheme "works" with tables that are being updated "in place" (by replacing immutable data blocks at random offsets by slightly different immutable data blocks) in the sense that queries will never return the wrong answer, but suppose someone updates record 50000. This will invalidate, among other things, one of the leaves under the 65536-item run in the index; if the FOO value has changed, this change will bubble up to recomputing that 65536-item run by merging together two 32768-item runs, the first of which is hopefully

still in the cache despite not having been used in quite a while. This takes some 65536 comparisons, which is not a lot of work in an absolute sense but still about four orders of magnitude larger than what you would hope to see for a single record update. Also when you append record 131072 you are going to have to do 131072 comparisons the next time you run a query that uses that index.

I think you can repair this approach to some degree by storing the table as a segmented journal of changes, maintaining a parallel bitmap or something of liveness markers for those changes, periodically cleaning low-occupancy segments like a log-structured filesystem by copying their remaining live changes to a new segment, and then using an incrementalized version of the LSM-tree-merging code that computes partial merges of soon-to-be-superseded blocks of the LSM tree. But this degree of complexity seems like it kind of loses the appeal of having the transaction caching system do everything for you automatically, and it still doesn't give you the option of having queries grovel over the log of recent changes when churn is too high.

Integrity enforcement

Hopwood also describes the use of such write logging to help with invariant maintenance: the write log tells you which objects have been changed in a transaction and whose state thus ought to be checked for correctness, and transaction rollback gives you the wherewithal to undo the damage. This is of course precisely the "C" in "ACID" in the traditional RDBMS usage of transactions: transactions violating consistency constraints will not be committed. (Ze also suggests automatic failover to alternate implementations in order to either detect the bug more precisely, by using slower invariant checking, or to fail over to an inefficient but trivially-correct implementation of the mutation.)

The incremental computation framework described in the previous section provides an efficient and simple way to do this: before committing, the code in the top-level transaction invokes a procedure which ostensibly verifies all the interesting invariants in the entire part of the system that it knows about, failing otherwise. This procedure invokes many other procedures to check invariants on particular parts of the system; most of these procedures will not have changed their inputs since the last invocation, and thus can succeed instantly simply using the memo table. But those which read transactional variables that have been written to will run for real, giving the transaction a chance to fail.

Relationship with dynamic scoping and graphics contexts

In retained-mode graphics APIs, it's common for graphical properties like fill color, line width, font, and transformation matrices to be implicitly inherited from parents to children in a hierarchically-nested scene graph; CSS properties in HTML and SVG are examples. In immediate-mode graphics APIs, such as PostScript, `<canvas>`, and even TeX, these are typically implemented as a large number of stateful variables instead, whose values are saved and restored using a stack of graphics states, for example using `gsave`

and `grestore` in PS, `.save()` and `.restore()` in `<canvas>`, or {} in TeX. The same set of tricks used for dynamically-scoped variables in Lisp are applicable — shallow binding for best read performance, deep binding for fastest context switching — and indeed such variables were one of the major arguments for retaining "special variables" in Common Lisp and adding `dynamic-wind` to Scheme.

This operation of temporarily obscuring the "global" value of a dynamically-scoped variable with one or more stack layers of "local" variables, then restoring it upon exit from a scope — this is all very closely reminiscent of the process of buffering mutable-cell writes and then discarding them on rollback. But of course you don't normally want to erase everything you've drawn when you restore these graphics parameters, and that's what rolling back a transaction would do. Is there an underlying unifying abstraction that can be applied to both cases?

Optimizing transactions

Zelkowitz's work in 1971 found that adding comprehensive undo logs to PL/I only added about 70% execution time to his PL/I programs (and bloated the programs themselves somewhat); he didn't report on runtime memory usage, which I'd think would often be the more crucial aspect.

Even with read logging and competing with modern compilers, the cost for one transaction per subroutine invocation for a pervasively mutable language like Python might be comparable to CPython's existing interpretation cost. But for many purposes CPython's performance is unsatisfactory.

How can we do better?

Only logging writes for high-priority transactions

As discussed in the section about interrupt handling, if a piece of code is protected from interference by other concurrent transactions — for example, by not allowing any of them to commit — it is guaranteed not to need a retry. So in that case the transaction mechanism is only providing error recovery, and for that the transaction system need only be involved in writes of transactional variables, not reads. This reduces the overhead of the transaction system by about an order of magnitude for these transactions, perhaps to less than a factor of 2 for conventional mutable code. (If we write transactional variables back to their home locations while doing this, we can use normal memory reads to read transactional variables inside the transaction.) Compiling code for latency-sensitive transactions in this way may be a worthwhile optimization.

A more extreme version of this is possible if the high-priority transactions are read-only; see the section below about read-only transactions and time travel.

Reducing the number of mutable variables

As mentioned above in the "fearless concurrency" section, the cost of logging reads and writes ought to be proportionally lower in a language design with many fewer mutable variables, like Haskell, OCaml, or Clojure — though, by the same token, many of the

potential benefits are smaller: for debugging you'll need to use a smart diff for path-copied data structures or other FP-persistent data structures, tail-call looping constructs already provide on-stack replacement, cleanup from exceptions is rarely necessary, and in many cases it's possible to confine bits of code for safe experimentation (for modal reasoning or debugging) using mechanisms the type system or object-capability discipline without resorting to transactions. The benefits for concurrency, I/O composability, and responsiveness remain unchanged, but they pertain to transactional-memory systems in general, not just those with implicit per-invocation nested transactions.

Even in pure or very-nearly-pure functional programming systems, acquiring the benefits of the time-limiting, automatic memoization, and incrementalization features described above requires other kinds of work, such as hash consing and the development of good cache eviction heuristics. This work is needed with or without transactions, and promises to be the lion's share of the job.

However, as mentioned in the filesystem example, reducing the number of mutable variables *too far* will cause unnecessary contention and thus reduce system throughput, either by optimistic concurrency control retrying transactions or by pessimistic concurrency control blocking them. In some cases, we would actually benefit by introducing extra mutable variables to permit higher levels of concurrency.

Aggregation

Aggregation is another common way to reduce the cost of read barriers, write barriers, and dependency tracking for incremental computation and rollback. The idea is that, by agglomerating mutable variables into larger units, we can reduce the work needed to track them, though, as above, reducing our potential concurrency as well. (Maybe this is the same idea under a different name.)

Under `make`, compilers and linkers communicate through the filesystem; if the compiler† changes `psmouse.o`, `make` reinvokes the whole linker with the whole new `psmouse.o`. It doesn't care which parts of `psmouse.o` have changed, and its devil-may-care attitude buys much less dependency-tracking overhead on the compiler's workings in exchange for a less precise incremental recompilation, involving a full relink.

If we try to analyze the `make` example in functional-programming terms, we could say that the compiler mutates the `psmouse.o` entry in a mutable directory to point to a new (immutable) binary string — the new contents of the object file; or that the compiler produces a new state of the filesystem in which the directory is a copy of the old directory except with `psmouse.o` pointing to different contents, and that is in fact more or less how Git implements directories in commits. (Even if you wouldn't normally commit `psmouse.o`.) But another way to analyze it is that the compiler applies a sequence of mutation operations to `psmouse.o`: first truncating it, then appending various blocks of bytes, perhaps even seeking around and backpatching some bytes. What strace shows is somewhere in between:

```
[pid 26311] stat("psmouse.o", {st_mode=S_IFREG|0644, st_size=2352, ...}) = 0
```

```
[pid 26311] lstat("psmouse.o", {st_mode=S_IFREG|0644, st_size=2352, ...}) = 0
[pid 26311] unlink("psmouse.o")          = 0
[pid 26311] open("psmouse.o", O_RDWR|O_CREAT|O_TRUNC, 0666) = 3
[pid 26311] write(3, "\0psmouse.c\0main\0read\0printf\0putc"..., 53) = 53
[pid 26311] lseek(3, 0, SEEK_SET)        = 0

[pid 26311] read(3, "\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 4096) = 1029
[pid 26311] lseek(3, -965, SEEK_CUR)     = 64

[pid 26311] write(3, "UH\211\345H\203\3540dH\213\4%(\0\0\0H\211E\3701\300\307E\320\0\0\0\0\307E"..., 516) = 516
[pid 26311] lseek(3, 0, SEEK_SET)        = 0

[pid 26311] read(3, "\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 4096) = 1029
[pid 26311] lseek(3, -445, SEEK_CUR)     = 584

[pid 26311] write(3, "\24\0\0\0\0\0\0\0\1zR\0\1x\20\1\33\f\7\10\220\1\0\0\34\0\0\0\34\0\0\0"..., 56) = 56
[pid 26311] lseek(3, 0, SEEK_SET)        = 0

[pid 26311] read(3, "\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 4096) = 1029
[pid 26311] lseek(3, 3, SEEK_CUR)        = 1032

[pid 26311] write(3, "7\0\0\0\0\0\0\0\2\0\0\0\n\0\0\0\374\377\377\377\377\377\377\377g\0\0\0\0\0\0\0"..., 384) = 384
[pid 26311] lseek(3, 0, SEEK_SET)        = 0

[pid 26311] read(3, "\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 4096) = 1416
[pid 26311] lseek(3, -776, SEEK_CUR)     = 640

[pid 26311] write(3, "\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\1\0\0\0\4\0\361\377"..., 336) = 336
[pid 26311] lseek(3, 0, SEEK_SET)        = 0

[pid 26311] read(3, "\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 4096) = 1416
[pid 26311] write(3, "\0.symtab\0.strtab\0.shstrtab\0.rela"..., 97) = 97
[pid 26311] lseek(3, 0, SEEK_SET)        = 0

[pid 26311] write(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\1\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 64) = 64
[pid 26311] lseek(3, 0, SEEK_SET)        = 0

[pid 26311] read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\1\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 4096) = 1513
[pid 26311] lseek(3, 7, SEEK_CUR)        = 1520

[pid 26311] write(3, "\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 832) = 832
[pid 26311] close(3)                     = 0
```

From the point of view of the kernel, the compiler† is mutating `psmouse.o` nine or ten times: first it unlinks the old file, then it creates the new one (`O_TRUNC`ing it if it somehow already exists), and then it write()s into it eight times at six different offsets.

But `make` doesn't care about that level of detail; it's content to work with the knowledge that `psmouse.o` has changed. So, for transactional purposes, it's unnecessary to keep noting that `psmouse.o` keeps changing unless we're creating new rollback points; it's adequate to keep a snapshot of its previous state.

We could imagine a filesystem or similar tree structure in which the degree of detail we retain about a transaction's writes varies dynamically: perhaps after we've accumulated a bunch of before-images of sibling "files" that are all being modified at once in a single transaction, we throw up our hands and save a before-image of the whole parent "directory", thus avoiding any further requirement to interpose write barriers on anything within it.

Similarly, if we have a large numerical array we're running a mutation loop over, it's adequate for many purposes to snapshot the whole array before the first mutation, rather than tracking individual mutations on the array. Analogously, array-computation libraries with automatic differentiation like TensorFlow track computational dependencies between entire arrays (vectors, matrices, etc.) rather than individual scalars within them.

The card-marking write barrier developed for Self used a single dirty bit for each chunk of memory (of I think 32 or 64 bytes), keeping the write-barrier data tiny and the write-barrier code fast, at the expense of imposing extra scanning work on the garbage collector. Similarly, for purposes of swapping out individual objects to disk, LOOM (Kaehler & Krasner 1982) used a single dirty bit per Smalltalk object. Logical logging for transactional RDBMS rollback logs typically stores before-images of rows being updated rather than the individual updated fields, and physical logging, used for rapid recovery to checkpoints rather than rolling back individual transactions, instead stores before-images of entire pages. (Terminology varies somewhat between databases.)

And, of course, virtual-memory operating systems typically track memory dirtiness at the granularity of a hardware page — 512 bytes on the VAX and 4096 bytes on most other systems — and handle copy-on-write data at the same granularity. Traditional FORTHs do the same thing, but with 1024-byte blocks.

For our transactional purposes we'd need to do more than set a dirty bit; as with RDBMS logging, we'd need to copy the clean data before modifying it, either into an undo log (as in Noether) or into a buffer of pending writes for the current transaction (to permit optimistic synchronization or constant-time rollback).

In the note on segments and blocks (p. 166) I outlined a virtual-machine system in which the virtual machine has a number of "descriptor registers" which mediate its access to memory, which consists of "segments" and "blocks"; read and write access is checked when a new descriptor is loaded into a descriptor register, while any number of accesses via an already-loaded descriptor can proceed with no further checking. Loading a read/write descriptor register would

potentially trigger a copy of the segment or block to be modified. This is explained in more detail in that note.

†Typically the assembler on Unix, actually.

Eliding unused rollback points

If we're *only* using transactions for error recovery and/or peremptory work discarding for responsiveness (not memoization, multithreading with optimistic synchronization, deoptimization, or debugging, as suggested above), then, when a parent procedure invokes a child procedure at a callsite where failures in the child will necessarily propagate to a failure in the parent, it's not necessary (for execution) to preserve the separate transaction for the child procedure — if the child rolls back, the parent rolls back too. This optimization dramatically reduces the amount of extra work imposed by the transaction system, and in particular something like it is mandatory for systems like Scheme that rely on tail-call optimization for looping.

Local variables and escape analysis

A subroutine can mutate its local variables freely without incurring any transaction overhead, unless those variables are referenceable (something impossible in, for example, Scheme) and references to them have in fact escaped. For example, Pascal-style `var` parameters can enable references to local variables to be passed to callees, but the language guarantees that once the callees return, those references are no longer live.

Plumbing transactions to the user interface, the filesystem, and the network

Depending on what filesystem you're running and how deeply you've been hurt, you might be able to trust the filesystem to honor your transaction boundaries as well, which means that code inside a transaction can read and write the filesystem freely — but the filesystem must give us a way to keep the writes within a transactional bubble, hidden from the rest of the world at first, and perhaps forever. Also, it must give us a way to transactionally validate our reads when we go to commit, if there's a possibility the data we read has been modified in the meantime.

This is potentially useful because it means you can run a transaction that includes multiple programs all communicating through the filesystem. This also potentially means you can use this sort of fearless concurrency in things like shell scripts, avoiding the messy failure cases and concurrency problems that normally plague them.

(If you do this with memoization of program outputs, you have a rather standard build system.)

A network file server can participate in your transactions in the same way as a local filesystem. Indeed, a network server need not be implementing anything very similar to a filesystem; it just needs to be participating in a transactional protocol with you, either arbitrating transaction commits and serialization or faithfully deferring to some such arbitration system. A queueing system is a prime candidate.

If you're willing to embrace the filesystem and networked services as part of your transactions, what about users? In particular, if you can run multiple entire programs inside a giant transaction, you could enable users to create a long-lived transaction that they then have a window into, as a way to experiment with new states they may not want to keep. However, I'm not sure this approach can really deliver a usable user experience of undo and restoration from backups; NixOS has its fans, in part because it offers a much freer model of switching between configurations than simple nested commit/rollback. On the other hand, using this approach for debugging implies that it's possible for users to see inside an uncommitted transaction, at least within the debugger; being able to can copy things out of the transaction history or an uncommitted transaction might be enough.

(Also, see above about the relationship with REST; the system can be extended in a natural way to prevent lost-update errors in web services.)

What about the XPra/NeWS/AJAX problem? Above I talked about using transactions across a distributed network under a single administrator as a near-panacea for problems of distributed programming, a level of optimism that surely will not pan out in real life. XPra provides remote access to GUI applications running on a server somewhere by rendering their GUIs server-side and transmitting the screen updates using a codec such as H.264, but this suffers from both computing-power-bottleneck problems (especially when many users share the same server) and latency problems. NeWS tried to solve this problem by allowing the application author to upload snippets of PostScript code to the window server, which could then react instantly to user interface events and do as much rendering inside the window server as desirable, providing a smart-client/mobile-code solution similar to modern AJAX webapps, but with PostScript as the client-side programming language instead of JS. AJAX is very good at improving responsivity, at least when it doesn't bloat a fucking text chat UI to occupy a gigabyte of RAM, and especially at reducing server load.

Could distributed transactions simplify the task of programming such applications? This is a degenerate case of the network of worker nodes all beating on a single master: one master, which is also a worker, and a second worker for low latency. Maybe they could allow any given code to run transparently on either end of the high-latency connection, or indeed optimistically on both ends of the connection, with the results from the second-to-commit execution being discarded. Transactions that only *read* the database can display their results from the database with the possibility of being out-of-date and needing to be re-executed (this is more or less how Meteor works, although they aren't called "transactions"), while transactions that write to it would have to wait for the server to confirm before reporting success. I don't know, I think there's maybe some potential here, but I don't have it thoroughly thought out.

Is there a connection with hardware transactional memory support that is starting to appear in modern high-end manycore systems? It is in some ways a way to expose the multisocket nature of the system to application software so that it can avoid paying unnecessary

synchronization costs. How would it play with this kind of per-subroutine-call nested transactions?

Reverse-mode automatic differentiation

Implementing this kind of rollback suffers from the same difficulties as reverse-mode automatic differentiation, namely that it needs to keep around all the intermediate values that have been overwritten, or anyway those that were live at a live rollback point. The checkpoints it provides could in fact literally be used as the checkpoints for reverse-mode automatic differentiation, a further crucial technique for solving inverse problems.

First-class transactions

What would it look like to, as Shae Erisson suggested, expose the per-call transactions as first-class objects to the user program? You could imagine, for example, inspecting your current transaction to see what mutable variables it had read or written, or the rolled-back transaction executed by a callee, and this would provide a natural interface for applying the technique to the various problems described above.

For example, the suggested application to REST would require the web framework to be able to generate some kind of serializable identifier for each mutable variable the HTML `<form>` depends on, and also to retrieve those variables given those identifiers when the form is returned. Facilities like those would also allow the proxy code for distributed nodes as described above to be written entirely at user level. As another example, the modal-reasoning question "what variables would this randomly generated code write to if I ran it?" needs to be able to abort the child transaction and then inspect its write log.

Time travel and read-only transactions

A transaction that doesn't write any transactional variables can be safely run at any time, regardless of anything else that's happening. The vertical-blanking-interval pull transaction mentioned above is one example: it might write to video RAM, but probably not to anything within the transaction system. Normally you would like such a transaction to run in the most up-to-date state possible, but the usual ACID serializability requirements don't actually require that; it's perfectly valid to run it with access to some consistent past state, maybe a recent-past state.

In a flat memory space in which transactional variables live at some "home" memory address, doing this without retrying or blocking any write transactions would normally require every read access to a transactional variable to be indirected through the transaction system, so that it could give you the results that were valid at the point in time you've been transported to. Although this is a reasonable cost, and one that most of the above discussion assumes we normally pay in every transaction, it might be nice to avoid it for real-time things like the VBI screen redraw transaction example. Earlier I suggested blocking all other transactions that go to commit until after the real-time transaction is done, but another alternative is to let them

commit, but buffer their writes in an update journal rather than write them back to their home addresses. This allows the read-only real-time transaction to proceed to completion without interacting with the transaction system *at all*, thus running at maximum speed. Other transactions can run in parallel as usual, if you have multiple cores, but their reads are served from the update log, so they can see updates that have happened since the real-time transaction began.

To the extent that past states of the transactional variables are logged and don't suffer linkrot (for example, because logged past states are not included as GC roots) you can also provide a time travel facility to allow not just debuggers but ordinary application programs to inspect past states, by explicitly running read-only transactions at past points — analogous to detached HEAD state in Git.

Thanks

Thanks to sbp, Darius Bacon, Corbin Simpson, CcxWrk, and especially Shae Erisson for many very informative discussions that helped greatly with this note.

Topics

Materials YouTube

Kragen Javier Sitaker, 02020-12-16 (updated 02020-12-17) (1 minute)

You can learn theoretical materials science by doing pencil-and-paper exercises from a book, but for doing it in practice, it's often crucially important to watch other people work through problems — this often allows you to pick up details of technique they wouldn't think to mention.

Some YouTube channels with a lot of information on materials processing:

- NileRed (2.2M subs) & NileBlue
- Applied Science (700k subs)
- Nurdrage (750k subs)
- Cody'sLab (1.9M subs)
- Primitive Technology (10M subs, largely bushcraft)
- How to Make Everything (though usually it omits details crucial to reproducing results; 1.4M subs, autarky-focused)
- MIT OCW Digital Lab Technique Manual, part of MIT OCW (2.8M subs)
- Mrhomescientist (57k subs)
- NightHawkInLight (1.7M subs)
- N2H4 labs
- Extractions&Ire (64k subs)
- Doug's Lab (52k subs)
- Gyzmodium (330 subs)
- Scrap Science (3.7k subs)
- Hydrogen, Time (240 subs)
- UC235 (9.2k subs)
- Chemsurvival (53k subs) (mostly lecture animations)
- Chemplayer, now banned on YT, moved to BitChute
- Neptunium (1.9k subs)
- ytmachx
- Myst32yt (18k subs)

Topics

- Materials (p. 788) (51 notes)
- Practical (p. 810) (12 notes)

Electronics next project

Kragen Javier Sitaker, 02020-12-21 (updated 02020-12-22)
(7 minutes)

I'm suffering from analysis paralysis as I ponder what electronic project to do next to level up (to both level up my skills and my equipment). This is related to the microcontroller inventory (p. 620) and the nonshopping list (p. 516). Here are some 58 projects I could try to do, mostly in random order with five randomly chosen but with a few manually chosen to move to the top list of finalists:

• **house thermometer** (indoor, outdoor, logging)
• **Joule Thief** as a simple battery tester if nothing else
• **7-segment display** with 7 scavenged discrete LEDs in some kind of case with slits and partitions, which can be very large
• **use a speaker as a microphone** to get bidirectional audio on a microcontroller.
• **make the strip of giant LEDs shine** so we can illuminate with it
• **audio amp** driving one or more of the speakers here
• **metronome** with relays
• **low-bandwidth 433MHz license-free SDR** for kilometer-scale robust communication
• **pulse soldering**: a circuit to deliver a calibrated amount of energy at a calibrated power level for, for example, battery pack construction or hair-wire soldering
• **ultra-low-power FM radio transmitter** to listen to music wirelessly
• **display text on a 7-segment LED screen** from an AVR or STM32
• **timer with relay** to turn off the water pump after 20'
• **plant waterer** to keep the plants from dying; this probably involves a microwave oven relay, a washing machine valve, a microcontroller, and some transistors

• **hot air pencil**: thermostatic control of air temperature permits rapid localized heating, very useful for rework and electronics scavenging, as well as preheat for soldering
• **ultrasound measurement** with two piezos and some oil, to see what frequencies I can get through the oil; this might require the Blue Pill
• **basic Magic Kazoo**: you hum into a microphone on one end, and it does pitch ID and synthesizes a musical instrument out the other
• **lightbulb thermometer** using a lightbulb as the temperature sensor
• **read the digital caliper's output port** to get another 20-micron-precision positional feedback source
• **stepper controller**: drive a low-power stepper motor with some transistors and a microcontroller
• **an electromagnet**, pure and simple
• **electrolysis controller** to permit precise control of electroplating, electro-etching, and similar processes
• **copper-wire foam cutter** (see Copper Segelin (p. 538))
• **micro-engraver** with three piezos, or two piezos and electro-etching, or one piezo and several electrodes, as a testbed for micro-engraving ideas
• **stainless steel wool foam cutter**: it's about 0.2Ω per 100mm, which

is in the same ballpark as thinner copper wire for the copper segelín (p. 538), but instead of burning at 300° it can get to over 800°, which is plenty for cutting styrofoam.
• **adjustable switching power supply** using discontinuous conduction mode to get current limiting
• **program the PALs** I scavenged to compute some particular logic function
• **noise-based optical position encoder**: by measuring randomly varying transmissivity or reflectivity along a linear track, it can rapidly determine its precise position
• **read and write SD card from AVR or STM32**: SD cards support SPI, and this enables a lot of projects that otherwise need way too much space.
• **air conditioner remote control** to control the air conditioner with infrared
• **FM radio receiver** to listen to the radio
• **Blue Pill oscilloscope**, only up to a megahertz or two but still better than a sound card
• **ultrasonic vector image transmission** (see audio vector image (p. 513))
• **program my ATMega328s with the Duemilanove**, the ones that are in a tube
• **program the ATTiny2313s** I have a tube full of here so I can use them for things like capacitance measurement
• **humidity sensor** (see PET dielectric spectroscopy (p. 566))
• **make an MPPT to measure a red LED in the sun** to find out how much energy you can harvest from it and at what power level
• **recognize whistling with an AVR** to command household electronics
• **synthesize voice with AVR and speaker** to get output
• **program the ATTiny45s** so I can use them for things like my notebook
• **dielectric spectroscopy**: identify materials, especially including humidity, capacitively through their frequency-dependent permittivity
• **program Blink onto one of the Blue Pills** so I can start using them
• **test the PAL delay line** I scavenged from a VCR
• **Lissajous projector**, scanning a laser pointer with a resonating nutating mirror. Once the mirror has a stable resonance set up, maybe from a speaker, you can tune in a picture by moving a photodiode into the illuminated area to find the overall period, then to corners and across the center to find the phase
• **impedance tomography sensor** to detect touches on a resistive surface with just a few electrodes and a microcontroller
• **emulate a PS/2 keyboard** to get data from microcontrollers onto bigger computers (see machine-readable microcontroller output (p. 637))
• **build a simple circuit with hair wire** to see if it can be made feasible
• **data reception with bidirectional LEDs** as extensively tested for, among other things, PJON
• **linear motor with aluminum sheet** consisting of two or three coils that can levitate and move the sheet; this probably requires some pretty hefty power
• **thermostat-controlled microfurnace** for things like firing pottery

• **current microbalance** to weigh light things by counteracting their weight with a precisely measured electromagnetic force
• **adjustable linear power supply** hanging off an ATX or power-brick power supply so I have an adjustable voltage; this becomes a lot more useful once it has some kind of readout, and ideally current limiting too; if it depends on the upstream power supply for voltage regulation it can be just a trimpot and an emitter follower
• **TransistorTester**, maybe not a super sophisticated one
• **rotational capacitive sensor** made of paper towel tubes and aluminum foil to measure rotational position with some precision, thus providing feedback for DC motors
• **capacitor meter (p. 589)**: with Duemilanove, to begin with
• **test PS/2 keyboard** by programming the host side of the PS/2 protocol into an AVR or STM32
• **decode a printer position sensor signal** of one of the two inkjet carriage assemblies I have here so as to get a 20-µm-precision motion control feedback system
• **4-wire ohmmeter** to measure sub-hectaohm resistances precisely
• **show text on a passive LCD** (ideally 7-segment at first; see the note on screens (p. 584) and rebraining (p. 597))
• **decode the microwave keyboard** so I can use it as a project box with membrane buttons

Topics

Electroforming networks

Kragen Javier Sitaker, 02020-12-22 (3 minutes)

aluminum VLA horn. cloth. soluble insulation

Suppose you want to make a metal sheet that has channels running through it, say for coolant. One possible way to do this is to make the shape of the channels out of wire of steel or aluminum, electroform copper onto them until you have a solid sheet (after a flash of some intermediate metal in the case of steel), and then dissolve the channels out of the copper, using alum solution for steel, or either hydrochloric acid or lye for aluminum.

This is potentially useful for many things, such as heat exchangers, cooling jackets, heated floors, and "labs on a chip" in which a sequence of processes is carried out in tubes within a monolithic device. Another similar application is making coils for induction heating: at high frequencies only the very surface of the coil's metal can be used, and you normally want to run water through it to keep it coool.

In many cases like this, it's useful to have a minimum separation between the channels to prevent fluid flow between them. For example, in a long serpentine coolant channel, a "short circuit" for the fluid could eliminate one or more loops, producing a hot spot due to inadequate coolant flow; and, in an induction coil, unwanted contact between successive coils can produce an *electrical* short circuit, which can produce a hot spot due to a narrow resistive connection.

To provide such a minimum separation, it's potentially useful to form the original wire network using wires bearing a layer of flexible "spacer", similar to the insulation used on electrical wires. Once the wires are all in place, you can remove the spacer, for example by burning it off with fire (if it is an organic chemical or, say, amorphous sulfur); dissolving it off with water (if it is a water-soluble material such as gelatin, albumin, pectin, starch, carboxymethylcellulose, poly(ethylene glycol), poly(acrylic acid), xanthan or guar gum, or polyvinyl alcohol, or held together with a water-soluble binder; mixing some baking powder in may speed this process); melting it off, if it has a low enough melting temperature; degelling it by changing the pH, if it's a pH-dependent gel like poly(acrylic acid) or some carrageenans; etc. Once the spacer is gone, hopefully without the wires sagging much, the electroforming process can begin.

Ideally the wires would themselves be hollow, having a thin open tube through their centers, thus containing within themselves the seed of their own doom — the alum or lye or whatever can attack them enormously more rapidly if it can be run through this pipe, rather than having to diffuse in from the ends.

Of course, copper is not the only metal that can be electroformed, and alternatives to electroforming exist: you may be able to pot the channel wires in resin, for example. But these processes are perhaps less interesting.

Topics

Time-scale material processing

Kragen Javier Sitaker, 02020-12-22 (3 minutes)

We frequently talk about whether something is soluble in water as purely a function of temperature, because of course the equilibrium is a function of temperature (and slightly of pressure, or more than slightly, for gases). But equilibrium is an ideal state never reached; two different materials with the same equilibrium solubility might have very different dissolution rates, and in the absence of seed crystals, one might have a much higher energy barrier than the other to nucleate crystals. Moreover, the growth of crystals after nucleation is initially exponential, then slows down to quadratic, rather than the initially-linear growth you'd expect from the simple Boltzmann energy-barrier picture.

Similar comments pertain to other reactions: the Gibbs free energy determines the reaction's equilibrium, but not the reaction rate, which may be autocatalytic (like crystal growth, but with diffusion) and thus experience exponential growth.

Historically the humans haven't taken much advantage of this in material processing (?), other than in heat treatment of metals, where it's an unavoidable challenge. But most lab techniques involve running the relevant reactions fully to equilibrium over the time of seconds to hours, spanning about four orders of magnitude. Processes that don't happen to an appreciable degree over hours are considered unimportant; processes that happen in less than a second are considered immediate.

It occurs to me that modern electronics and microfluidics each offer us the opportunity to intervene reproducibly in such processes at nanosecond timescales, adding nine more orders of temporal magnitude to our arsenal. If we have two processes in a material mixture, one which runs to completion at a given temperature in 10 microseconds and the other in 10 milliseconds, we can interrupt the proceedings after 10 microseconds when the second process is only 0.1% complete, or perhaps 0.0001%. (Interrupt? For example, by diluting, chilling, or poisoning the interaction.) We commonly do this kind of thing over a longer timescale in cooking: boiling the carrots for five minutes may make them delightfully soft, while boiling them for an hour will make them unpalatably mushy.

The information needed to plan such processes is rarely available in the existing research literature, because workers in the field usually don't care whether a particular material change takes a nanosecond, a millisecond, or a microsecond.

We can alternate between two processes, one which has a yield of 0.01% of a desired product (due to equilibrium, for example) and the other of which removes the product from it, at kilohertz or megahertz frequencies. Of course, there are many existing processes which work this way already without such alternation; the Pidgeon process, for example, produces magnesium through silicothermic reduction of magnesia (produced by calcining dolomite) despite an unpromising equilibrium, because the magnesium boils out of the

reaction and the silicon is taken up by quicklime (also from the dolomite) to form larnite. But there are other processes that cannot be run in this way, for example because they involve ingredients that would have unwanted reactions with one another, or the desired equilibria require vastly different temperatures or pressures.

Topics

Circle-portal GUI II

Kragen Javier Sitaker, 02020-12-22 (updated 02020-12-23)
(4 minutes)

In Dercuano I wrote about a "circle portal GUI" ZUI design consisting entirely of circular windows. Your user interface is a circular viewport onto an infinitely zoomable canvas, which contains other such circular viewports onto other parts of itself, each possessed of only a position and size, a destination position and size and orientation, a z-order, a background color, and a translucency. You can create these portals, duplicate them, move them, move their destination, resize them, resize their destination, change their background color, change their z-order, and follow them — they function as hyperlinks. Also you can "snap" them, destroying them and copying the region of canvas they view onto the place where they are displayed.

By arranging some empty portals with a given background color somewhere, you could create a letterform, and then by making portals that view that letterform in various places you could make copies of the letter. By arranging dozens of such letters together in a font, you could have a way to write text in that font; if you made a "font portal" viewing that font, you could make portals onto the positions of letterforms in that portal, which could also be used to spell text; by changing the destination of your font portal, you could change the font used for that text.

(You'd probably want to have a way to interpose a level of indirection like that after the fact.)

If you wanted to draw something in some changeable colors, you could do that by putting your palette in one place, and making your drawing out of portals onto the palette.

So this single type of object serves as a graphical primitive (since you can set the background color), a limited IFS generator (without shearing or nonuniform scaling, since each transform only supports four degrees of freedom rather than the usual 6), a hypermedia navigation system with live preview, a graphical instancing system capable of some use as a stylesheet, a universal "view source" button, and so on, all through direct manipulation, though a sort of direct manipulation that would probably be super confusing, just due to its hall-of-mirrors nature.

(I implemented a tiny prototype of the system one day, but didn't get far enough to get a good sense for how to use it. I imagine you'd want to add some other graphical primitives and interaction modes; dragging letterforms one by one is maybe not a great way to write text.)

I was thinking today that it would be interesting to do the same thing in 3-D, using spheres instead of circles. The sphere portals could display what was *within* their target volume, like you might imagine looking into a crystal ball; they could display what was visible by looking *through* it (either in a given direction, or in the direction you're looking in); or they could display what would be

reflected from the volume around their target volume, just as a silver ball displays what is reflected from the volume around itself. This last item would have the benefit that the target volume could itself be an object in the world, so the hypertext links in this system could be bidirectional, which might actually help a little bit with the confusion, though it would impede uses like instancing a character from a font. (You could probably get an acceptable approximation by using really small target spheres.)

Topics

Methods for two-dimensional rotation with two or three real multiplies

Kragen Javier Sitaker, 02020-12-23 (updated 02020-12-26)
(14 minutes)

Method K for complex multiplication

Wikipedia says Knuth gives an algorithm for multiplying $(a + bi)(c + di)$ as follows: $(k_1 - k_2) + (k_1 + k_3)i$ where $k_1 = a(c + d)$, $k_2 = (a + b)d$, $k_3 = (b - a)c$, which works out to $(ac - bd) + (ad + bc)i$ as it should. Call this Method K. Method K is interesting because often multiplication is much more expensive than addition or subtraction (for example, it takes much more space in hardware in fixed point, and much more time in multiple precision), and this algorithm requires only three real multiplies, rather than the four required by the more direct approach.

In Unicode matrix form:

$$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ -1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} ac \\ ad \\ bc \\ bd \end{bmatrix} = \begin{bmatrix} k_1 \\ k_2 \\ k_3 \end{bmatrix}$$

$$\begin{bmatrix} 1 & -1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} k_1 \\ k_2 \\ k_3 \end{bmatrix} = \begin{bmatrix} \Re \\ \Im \end{bmatrix}$$

Because

$$\begin{bmatrix} 1 & -1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ -1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} ac \\ ad \\ bc \\ bd \end{bmatrix} =$$

$$\begin{bmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} ac \\ ad \\ bc \\ bd \end{bmatrix}$$

(Incidentally, the version of Method K given in Wikipedia interchanges k_2 and k_3, as well as a and c, and b and d. That's because I reconstructed it from memory and it has an asymmetry with respect

to its arguments that Karatsuba multiplication does not.)

Karatsuba multiplication

A very similar insight underlies Karatsuba multiplication for multiple-precision real numbers; I don't know if Karatsuba was working directly from the above complex algorithm, but in Karatsuba multiplication we obtain $(a + br)(c + dr) = ac + (ad + bc)r + bdr^2$ by calculating $j_0 + (j_2 - j_1 - j_0)r + j_1r^2$, where j_0 is of course ac and j_1 is of course bd, from which we can calculate that j_2 must be $ac + ad + bc + bd = (a + b)(c + d)$ — our third multiply! (And two adds.)

So, for example, with $r = 10$, we can calculate 97×86 as $j_0 = 7{\times}6 = 42$, $j_1 = 9{\times}8 = 72$, $j_2 = (9{+}7){\times}(8{+}6) - j_1 - j_0 = 16{\times}14 - 42 - 72 = 224 - 42 - 72 = 110$, so our answer is $42 + 110{\times}10 + 72{\times}100 = 8342$, which is correct. And recursively subdividing the problem this way gives us Karatsuba's asymptotically faster multiplication algorithm, the first one discovered in millennia.

Applied to the special case $r = i$, Karatsuba multiplication gives us $j_0 - j_1 + (j_2 - j_1 - j_0)i$, so Karatsuba multiplication gives us a slightly different three-real-multiply algorithm for complex multiplication. It uses two adds and three subtracts in addition to the multiplies, while Method K uses three adds and two subtracts — virtually the same computational cost.

Partial evaluation

If we partially evaluate these two algorithms with respect to one of the arguments, say we hold constant the (a, b) argument, Method K eliminates an add and a subtract, because $k_2 = m_0d$ and $k_3 = m_1c$, where the m_i depend only on the constant argument. Partially evaluating Karatsuba multiplication in the same way only eliminates the $a + b$ add. So for complex multiplication by a constant multiplier, Karatsuba costs three multiplies, one add, and three subtracts; Method K costs three multiplies, two adds, and one subtract; and the basic method costs four multiplies, one add, and one subtract, just as for non-constant arguments.

Complex multiplication for rotation and scaling

In the context of computer graphics, complex multiplies are particularly interesting because they perform uniform scaling and rotation in a single operation. So, for example, you can take a vector figure represented as a list of (x, y) points, and scale it by n and rotate it by θ by multiplying each number $(x + yi)$ by a complex constant $(n \cos \theta + ni \sin \theta)$. (Typically you also want translation, which is complex addition if you're still thinking in complex numbers, but there's no advantage in doing so.) If you have the x and y components of various different points in some SIMD registers, this costs you three SIMD multiplies, two SIMD adds, and one SIMD subtract using the partially-evaluated Method K.

Commonly in computer graphics we instead rotate raster images; this can be done by brute force by translating and rotating the screen coordinates of every screen pixel into texture space by the methods

above, but strength-reducing this operation is very advantageous and universally done. Instead of transforming the coordinates of each pixel, we transform the coordinates of a start pixel, the delta (1, 0) to move one pixel to the right, and the delta (0, 1) to move one scan line down. These give us increments we can use to walk around texture space with simple adds. Then we can sample from texture space with a variety of methods — for procedural textures like fragment shaders, we just invoke the procedure with the transformed (x, y) arguments, while for materialized textures we commonly use nearest-neighbor or bilinear sampling, though there are a variety of common tradeoffs between aliasing and computation time.

Paeth's three-shear rotation algorithm

There's another famous algorithm for rotating raster images with three multiplies, though, by Paeth, usually called the "three-shear rotation". It doesn't do any scaling, but its compensating virtue is that, in its usual form, it doesn't lose any pixels — under appropriate circumstances it's perfectly reversible, because you execute it by *shearing* the raster pixels by displacing them some integer number of pixels. This also means that it doesn't require per-pixel sampling operations, even rounding.

The disadvantage of Paeth's three-shear rotation is that it produces a lot of aliasing artifacts because of the constraint of shifting the pixels only by integer amounts. See the note on stochastic fractional delay lines (p. 772) for some approaches to this problem.

Paeth's algorithm for rotating a vector (x, y) consist of the following three steps:

```
x += αy;
y -= βx;
x += αy;
```

(And the shear transformations work by shifting pixels αy pixels to the right in the first step, βx pixels up in the second step, and αy pixels to the right again in the third step. Normally you round these shifts to integers.)

We can represent this calculation with matrix concatenation as follows (here I'm copying my memory of Tobin Fricke's page on the subject):

$$\begin{bmatrix} 1 & \alpha \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ -\beta & 1 \end{bmatrix} \begin{bmatrix} 1 & \alpha \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Let's concatenate out the matrices; first the rightmost ones:

$$\begin{bmatrix} 1 & 0 \\ -\beta & 1 \end{bmatrix} \begin{bmatrix} 1 & \alpha \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & \alpha \\ -\beta & 1-\alpha\beta \end{bmatrix}$$

That is, when we subtract off β of x from y in the second step, the x

we're subtracting already has an α of the original y in it, so the y-to-y item isn't 1. Now the third step:

$$\begin{bmatrix} 1 & \alpha \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & \alpha \\ -\beta & 1-\alpha\beta \end{bmatrix} = \begin{bmatrix} 1 - \alpha\beta & 2\alpha - \alpha^2\beta \\ -\beta & 1 - \alpha\beta \end{bmatrix}$$

So now we end up subtracting off a proportion $\alpha\beta$ of the original x as well as the original y, and adjusting each one by different fractions (respectively $-\beta$ and $2\alpha - \alpha^2\beta$) of the other, fractions which approach 0 if α and β do.

So, to get a matrix for rotation by θ out of this, we need $\cos\theta = 1 - \alpha\beta$ and $\sin\theta = -\beta = \alpha^2\beta - 2\alpha$, which is three equations in two unknowns. We have directly that $\beta = -\sin\theta$, and to calculate α we can observe that $\cos\theta = \sqrt{(1 - \sin^2\theta)} = \sqrt{(1 - \beta^2)} = 1 - \alpha\beta$. Thus $\alpha = (1 - \sqrt{(1 - \beta^2)})/\beta$. (As it turns out, this is $-\tan \frac{1}{2}\theta$, but that's a terrible way to calculate it.)

But our problem was overdetermined, so we still have to check if this gives us $-\beta = \alpha^2\beta - 2\alpha$, so we want to see if $\beta = 2\alpha - \alpha^2\beta$, which becomes

$$\begin{aligned}
&2(1 - \sqrt{(1 - \beta^2)})/\beta - \beta(1 - \sqrt{(1 - \beta^2)})^2/\beta^2 = \\
&2/\beta - 2\sqrt{(1 - \beta^2)}/\beta - (1 - \sqrt{(1 - \beta^2)})^2/\beta = \\
&(2 - 2\sqrt{(1 - \beta^2)} - (1 - \sqrt{(1 - \beta^2)})^2)/\beta = \\
&(2 - 2\sqrt{(1 - \beta^2)} - 1 + 2\sqrt{(1 - \beta^2)} - (1 - \beta^2))/\beta = \\
&(2 - 1 - 1 + \beta^2))/\beta = \\
&\beta.
\end{aligned}$$

So choosing α and β in this way does give us a pure rotation. For example, for $\theta = 10°$, $\beta = -\sin\theta \approx -.174$, $\alpha = (1 - \sqrt{(1 - \beta^2)})/\beta \approx -.0877$. Starting at (100, 0), we proceed to (98.5, 17.4), (93.9, 34.3), (86.5, 50.1). These all have magnitude 100 and angles of respectively 0°, 10°, 20°, and 30°, so it seems to be working, though with a bit of rounding error — the last one should have been ($86.6 = 100\sqrt{¾}$, $50.0 = 100/2$).

Toffoli and Quick in 1997 reported a similar three-shear algorithm for three-dimensional rotation, but I haven't read the paper.

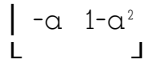
Minsky's circle algorithm and two-shear image rotation

Paeth's algorithm is strikingly similar to the Minsky circle algorithm described in HAKMEM, which computes an approximate rotation of a point around the origin as follows:

```
x += αy;
y -= αx;
# no further steps
```

This is, surprisingly, stable with exact math; the determinant of the resulting matrix is exactly 1:

$$\begin{bmatrix} 1 & \alpha \\ \end{bmatrix}$$



It's usually even stable with approximate math, including integer math. With integer math, even if α is prescaled to be something like 3/32, each of the steps is computationally reversible, like Paeth's rotations; so orbits can't converge, and they can't grow without bound because the determinant is 1, so they must return to the starting point. (For it to be computationally irreversible, you'd need addition and subtraction to round sometimes.) But the circles described by successive iterations are elliptical, which is obvious if you start with, for example, $(x, y, \alpha) = (1, 1, 1)$ — the orbit is (1, 1), (2, -1), (2, -1), (1, -2), (-1, -1), (-2, 1), (-1, 2), and then repeats. For smaller values of α the ellipticity is quite small.

The reversibility property means that if you use this transformation to map a bunch of unique pixel coordinates, the resulting pixel coordinates will still be unique! In fact we can implement this "rotation" on a raster image with two Paeth-like shears, and each of the pixels will describe a Minsky pseudocircle that never collides with any other pixel, and eventually returns to its starting point. The image will be distorted as it rotates (in particular any pixels close enough to the origin that αx and αy round to zero will stay put instead of rotating!) but it will eventually return to its original form. Not having tried it, I suspect that for many parameter values, particularly with the center of rotation well outside the image, the distortions will be imperceptibly small compared to the aliasing of Paeth's algorithm implemented with integer shears.

Both Minsky's and Paeth's algorithm can be thought of as two timesteps of leapfrog integration of a simple harmonic oscillator ($\ddot{x} = -kx$); the difference is that Paeth's algorithm starts halfway in between two timesteps. But it seems like this would imply that you can undo the ellipticity of Minsky's algorithm by picking a different starting position, and in fact you can't.

Paeth locality and run-length slicing

Many people have questioned whether Paeth rotation is still fast on modern machines, quite apart from the aliasing artifacts induced by its standard implementation with integer pixel shifts, because the standard approach uses three passes over the image, thus blowing up your dcache unnecessarily. I think you can probably get enough locality by pipelining and maybe tiling. Suppose you break the image into tiles of 16×16 pixels (768 bytes in fancy shmancy 24-bit color) and your α and β factors are small enough that the shift over the whole image is less than ±16 pixels. Then producing a single output tile involves only pixels from two horizontally adjacent second-stage tiles; each of these involves only pixels from two vertically adjacent first-stage tiles, four in all; and each of these involves only pixels from two horizontally adjacent input tiles, six in all, spread across two rows of tiles.

So if you pipeline the shears you only need enough dcache to hold 32 scan lines of the image, which I think is even true without tiling; if it's the traditional 1024 pixels wide then that's 96 kilobytes in

24-bit color, which is coincidentally exactly the size of my L1D cache on my Pentium N3700 laptop.

We can do better than this with Z-curve or Hilbert-curve ordering over the tiles.

However, for smallish rotations, we can do *much* better; each scan line of the output is made out of concatenated segments from different scan lines of the input.

Consider an input image all white with one horizontal black line. The initial x-shear just moves the black line with the image. Then the y-shear breaks the black line up into something like a Bresenham line; if the y-shear is 16 pixels over a width of 500, for example, it consists of alternating 31-pixel (75%) and 32-pixel (25%) segments, placed on successive scan lines; this stretches the originally-500-pixel line into a line of roughly 500.26 pixels in length. Then the final x-shear may overlap some of these segments, putting two black pixels above one another, or it may not, depending on where the line is positioned vertically.

I think these overlaps are points where, when moving from writing into output scan line m by copying from input scan line n to copying from input scan line $n\pm1$, we also adjust the x-offset from which we are copying.

Larger rotations produce shorter segments which overlap more often, but until you get to a rotation of more than about 20°, the segments are still substantial. You can copy these segments directly from the input image into their place in the output image, rather like run-length-slice line drawing, but sometimes with overlap between the ends of the slices, and copying pixels rather than filling colors.

Topics

Light pen latency

Kragen Javier Sitaker, 02020-12-23 (updated 02020-12-28)
(29 minutes)

There's a lot of recent HCI research on the impact of UI latency on computer usability. The picture is abysmal: keyboard latency is 70–200 ms and touchscreen latency is 70–250; even musical performance systems like the Reactable suffer from 100–300 ms input lag (I counted frames in that video); the experimental tangible programming system at Dynamicland is kneecapped by a latency in the 600–2500 ms range. Meanwhile, people can tell the difference between 1 ms and 2 ms of dragging lag in user tests with a custom-built low-latency stylus tracking system, or detect 10 ms of latency in a different experiment; 10 ms of latency at dragging or drawing is extremely obvious; performance in the simplest touchscreen tasks is hampered by input latency over 25–50 ms; and with 1–3 ms of latency, projection-mapping can convincingly add textures to moving and flexing objects, even when some of the light leaks around the edges or the objects ripple in the wind, though commercially available real-time projection mapping systems let the projection mapping visibly slide around over the moving surface due to their higher latency, thus failing to achieve the desired illusion.

The Microsoft system used above used a custom-built "high-performance stylus system" using Gray-code-modulated light patterns projected at 24kHz with a TI DLP DMD "projector development kit" which costs US$1800; the Ishikawa .* Lab research used both a projector with a custom-built 1000fps tracking mirror setup (using what looks like the kind of galvo rotating mirrors used for laser shows), later a 1000fps custom-built projector, and a custom-built 1000fps camera. Lower-cost DMD devices like the TI DLP4710 chip can only manage 120Hz but still cost US$170.

Is there nothing we can do to do HCI experimentation with low-latency user interfaces without shelling out the big bucks?

Seven possibilities occur to me: light pens, Wacom tablets, theremins, MEMS accelerometers and gyroscopes, inkjet-printer carriage feedback hardware, acoustic surface triangulation, encoded LEDs and photodetectors, and impedance tomography.

Light pens

A light pen — described in Ivan Sutherland's SKETCHPAD dissertation and his 1994 talk about SKETCHPAD, and he seems to have been the first to draw with it, but apparently didn't invent the thing — is a high-temporal-resolution light sensor in a tube. You point it at a CRT screen, and it detects the time when the electron beam illuminates the point the pen is pointed at on the screen. A 6-ns photodiode covering the whole visible spectrum will run you 26¢ in quantity 10 at Digi-Key or 63¢ for a bigger 5-ns one, although some common photodiodes are as slow as 50 ns or 100 ns.

An NTSC TV set runs 29.97 full interlaced frames per second at 525 lines per frame (486 visible), so the electron beam in an NTSC

CRT takes 63.6 µs to sweep across each line, including the 10.9 µs horizontal blanking interval. Typically about a quarter of the screen is illuminated at any given time, as viewed through a high-speed camera, about 4–5 ms (a fourth of a 16.7-ms field), because that's how long the phosphors take to fade (maybe 200 ns P22B, 850 µs P22R, 35 µs P22G). So the brightness at any given point on the screen rises sharply once or twice per 33.3-ms frame, with a rise time limited mostly by the focus of the electron beam (or, at high beam energies, the contagion of cathodoluminescence through the phosphor, or sometimes the rise time of fluorescence in the phosphor, which is on the order of 10 ns), and then fades away exponentially to zero over 4–5 ms. Since each scan line is 52.7 µs, not counting the HBI, 100 ns is a 527th of a scan line, and 5 ns is a 10,540th of a scan line. So any old photodiode would work to get near-single-pixel precision on a light pen driven by a regular NTSC raster. (High-quality analog CRTs could reproduce 400 dark-light cycles across a scan line, so we can consider them to have about 800 pixels per scan line, but NTSC was more limited. The whole NTSC broadcast signal was only 6 MHz in bandwidth, which is 200 kilocycles or 400 kilopixels per 29.97-Hz frame; split across 525 lines, that's only 762 pixels per line, only 52.7/63.6 = 632 of which were outside the HBI.)

However, because each point on the screen is only scanned once every 33.3 ms, your average latency *in the light pen itself* would be 16.7 ms, while the worst-case latency would be 33.3 ms. This is an unpromising place to start.

However, in SKETCHPAD, Sutherland was driving the TX-2's "scope" with two registers that were wired up to (10-bit) DACs; so, by writing to these registers, he could position the electron beam at any position on the screen immediately. To track the light pen, he drew a crosshair around the point where the pen was believed to be pointing, and by detecting which part of the crosshair was stimulating the pen's photodetector (by way of their timing), he could detect when the pen had moved somewhat. The TX-2 was far too small and too slow to handle a raster scan at any kind of reasonable resolution.

Light pens on NTSC CRTs

I find PAL CRT TVs discarded on the sidewalk on a regular basis; some are NTSC-capable as well, and PAL is broadly similar to NTSC. Usually the deflection coils have been recycled before I get to them, destroying the CRT. I'm told that in the US the going rate for a CRT TV is US$25 — paid to the person who hauls it away.

You could rip out the sync and scanning circuitry from a regular NTSC CRT TV and drive its deflection coils from custom circuitry so as to revisit the area around your light pen more often. This won't be quite as simple as the electrostatic deflection used in oscilloscope tubes and the TX-2 "scope" — magnetic deflection coils have substantial inductance, and the vertical scan coil in particular isn't guaranteed to be able to cope with more than a few hundred Hz of bandwidth; a 60-Hz vertical scan with a fast vertical retrace. But you probably don't need it to. (The horizontal deflection coil normally runs at 15.73 kHz, so it should be fine down to deep sub-millisecond latency.)

The particular form of bandwidth limiting is that it the electron beam's vertical position is determined by the coil's current, and the voltage aplied to the coil is proportional to the position's *derivative*, and you can only apply so much voltage before something breaks. So, in theory, there's no difficulty with instantly starting and stopping the vertical scanning motion, though some parasitic capacitances across the coil might cause that until you counteract them; what you can't do is move the electron beam quickly vertically, like probably at more than about one screen height per millisecond. The NTSC vertical blanking interval is 16.67 ms × (525 - 486)/525 ≈ 1.24 ms, but maybe you'll get lucky and get a faster tube.

So you could spend some of the time doing a semi-normal raster scan over the whole display, but periodically, like every 4 ms, take a break from drawing the normal raster image, jump down to where you expect the cursor to be, draw a crosshair or whatever to see if it generates a pulse in the photodiode, and then start drawing raster again. Once every field (you could perhaps increase the interlacing from NTSC's two fields up to three or four fields) you can do this for free; if the cursor isn't too close to the top or bottom of the screen, you can start drawing the raster starting *from* the cursor, moving up or down according to where pixels need painting. So if the cursor is in the middle of the screen, for example, you can paint each field in two halves, one starting from the cursor and going up, the other starting from the cursor and going down. This might save you the return fare from your round-trip ticket. (I'm not convinced it will actually help, though.)

Sometimes, though, especially if the cursor is close to the top or bottom of the screen, you'll have to spend the time to jump up or down to the cursor, then jump back to what you were drawing.

So one scheme is:

- 15.73 kHz horizontal raster scan most of the time.
- 29.97 Hz raster frame rate (33.37 ms per frame).
- 4:1 interlacing, so each field is 8.34 ms, implying a 120 Hz vertical retrace frequency, which is probably feasible but may be challenging.
- Three cursor probes per field: one when it happens to reach the cursor and thus costs no travel time; one from a worst-case vertical distance of ⅓ field away from the cursor and thus 0.41 ms each way, 0.83 ms total; and one from a worst-case vertical distance of ⅔ fields away from the cursor and thus 0.83 ms each way, 1.7 ms total. So in the worst case we spend 2.48 ms of our 8.34 ms field waiting for partial vertical retraces to go to and from the cursor position to probe it.
- Boustrophedon vertical scanning, so we don't have to waste any other time on vertical retrace; to keep the scan lines parallel, when drawing upwards, we scan right to left, but when drawing downwards, left to right.
- The cursor probing proper is restricted to 5% of the height and width of the screen (24 scan lines, each scanned horizontally 5% of the screen width) so it takes about as much time as drawing a single scan line, 60 µs or so. This can also be used to rapidly update images around the cursor to reduce visual feedback latency.
- So, out of the 33.37 ms per frame, we spend 0.72 ms probing for the

cursor 12 times, 9.92 moving the beam vertically to and from where we're tracking the cursor, leaving 22.7 ms to draw pixels. This gives us 357 scan lines of actual data, which is respectably close to the 486 visible scan lines of normal NTSC.

I suspect that boustrophedon horizontal scanning might allow us to use much higher raster scan rates with the same tube, since there's no need for an HBI, but then you have to modulate your data to avoid bright spots where scan lines intersect. Also, the higher raster scan rates would place more demand on the light pen's signal latency and the phosphor's rise time in order to achieve the same horizontal positioning precision.

This would give us 2.8 ms worst case input latency, 1.4 ms average, and almost a quarter of a megapixel. "VGA" resolution, you might say. This is capable of supporting some HCI experiments with about two orders of magnitude better latency than commonly deployed user interface hardware.

A less demanding way to use the device would be to only draw things in a narrow band, like ½ of the screen height, around the cursor. Maybe you could occasionally sneak away to draw something on the outer parts of the display, or maybe you could just leave letterboxing black strips at the top and bottom of the screen.

Light pens with VGA CRTs

I also regularly find discarded computer CRT monitors on the sidewalk, also usually with the deflection yokes having been already recycled by scavengers. These are similar to NTSC TVs, but typically support at least 1024×768 pixels at 72 Hz, implying a horizontal deflection frequency of at least 55 kHz.

This would probably be a bit superior to an NTSC TV, but maybe not as much as you might hope, since the vertical slew rate is still the limiting factor on input latency.

Light pens with projectors

CRT projectors have similar time-domain behavior, so in theory you ought to be able to use a light pen by pointing it at the image from a CRT projector or eidophor in the same way you could use a direct-view CRT. However, CRT projectors were never very common and have become extinct since the 1990s, so it's hard to find them nowadays.

There's a specialized kind of CRT projector called a "flying-spot scanner" which projects a flat light field onto a sheet of paper, then using the time-domain variation in diffuse reflectance to recover the original paper image; this is closely analogous to structured-light 3-D scanning, which identifies the parallax to all the points on an object using the same kind of binary or Gray-code images the Microsoft researchers used for their touchscreen. By the 1970s flying-spot scanners were using specialized low-persistence phosphors to maximize "flicker" and thus the sharpness of the scanned image. (Before the Vidicon tube, such flying-spot illumination was a favored way to take television images.)

Several times in the past it has occurred to me that you could scan a

time-domain-modulated laser across a surface with mirrors to make a flying-spot projector. (Perhaps a planar Kerr cell or Pockels cell with a voltage gradient along its surface across a resistive surface film electrode could provide a faster-response alternative to a moving mirror.) This approach of course would also work with a light pen in the same way as the scanning electron beam from a CRT, but without the problems induced by phosphor persistence and phosphor electron penetration depth. Ordinary laser-show galvos are capable of much faster response than typical CRT vertical deflection yokes.

Ordinary LCD and DMD projectors cannot be used in this way because the high-resolution time-domain signal is respectively absent or not under the control of the computer system. (DMDs control the brightness of their pixels with PWM at some kilohertz, so they would be able to transmit tens of kilobits pe second of data if those PWM signals could be fed in externally.)

A potentially more interesting way to do this would be to use a separate infrared (or green + infrared) tracking laser to track the light pen, so that you would never have to move the tracking laser away from the indicated point, except when you lost it.

Such light pens would probably also usually be indirect pointing devices, where the user relies on projected feedback from the computer system to find out where their "hand" is, although with rear projection (like large multitouch screens use) you could get direct pointing out of them.

Wacom tablets

I can't find good information about the latency of Wacom tablets, although they're popular for experimental musical instrument interfaces. Apparently the wires in the tablet grid switch between transmitting and receiving to the pen every 20 µs, though, and demanding users report higher latency with the USB versions, so I suspect they're submillisecond.

Wacom tablets, unlike touchscreens and stylus screens, are normally indirect pointing devices: what you are looking at is not what you are pointing to. This makes the latency demands less demanding, but it also probably makes people's performance slower, since they don't have a lifetime of experience coordinating their proprioceptive and visual feedback channels to know how far to move their hand.

Theremins

A theremin, invented in 1920, capacitively senses the distance to the user's hand by inducing a small frequency shift in an RF oscillator of a few hundred kHz, whose resonator is a tank circuit including the user as part of the capacitor. A second RF oscillator is calibrated to have nearly the same frequency; by heterodyning the two, an audio difference frequency is produced. Thus a difference that is very small in relative terms — the difference between 100 Hz and 2000 Hz is only 1900 Hz, and on a signal resonating around 400 kHz, that's only a difference of about 0.25%, produced by a difference of about 0.01 pF. But this difference can be easily detected.

If I understand correctly, the theremin's memory, and thus its

maximum possible response latency, is around a millisecond. If you want to use the theremin principle for low-latency gesture detection, though, you probably want to use slightly higher beat frequencies, or directly measure the frequency of the oscillations rather than heterodyning anything, because measuring the frequency of a 440-Hz distorted sine wave in less than a millisecond is, if not impossible, at least unnecessarily difficult.

A theremin is normally also an indirect pointing instrument, but you could imagine projecting an image with any kind of projector — even an LCD projector — and using the theremin to detect where your hand was on the projected image, having calibrated it to the projection setup. Other kinds of screens (LCD screens, OLED screens) would probably overwhelm the theremin signal with electrostatic noise.

MEMS accelerometers and "gyroscopes"

Every new cellphone or tablet computer has one of these MEMS accelerometer chips. I think the ADXL350 is typical of the genre: 3×4 mm, 3200 samples per second, 2 milligee resolution, typically a few tens of milligees offset error. A pointing device containing such a chip could in theory give you hand orientation and movement feedback at 300-µs latency, but of course a cellphone can't manage anything like such low latencies; from Digi-Key these devices cost US$7.38 in quantity 10.

There are also similar "gyroscope" chips that directly detect rotation, as well as "IMU" chips combining both; rotation might actually be a more amenable input modality for pointing at things. The TDK IAM-20380 MEMS gyro costs US$10.32 in quantity 10 and gives you 16-bit readouts on rotation speed around three axes, with 16.4 to 131 counts per (degree per second) depending on which range you have set, about ±2 degrees per second of offset, and 8000-sample-per-second output — but with a built-in low-pass filter, which suggests that possibly the signal is super noisy, and which isn't really documented in the datasheet except to say that it's 5–250 Hz. The specs say it has 0.010 dps/√Hz noise spectral density, which suggests that at 500 Hz (thus 1000 samples/second) you'd expect about 0.2 dps of noise, which sounds pretty tolerable for a low-latency pointing device.

Even the 2 ms latency implied by the 250 ms low-pass filter setting would be a vast improvement over conventional I/O devices.

A cheaper option is the ST L2G2ISTR, which costs US$2.54 in quantity 10 and has only two axes; it has 131–262 counts per (degree per second) and 9090 samples per second. Its target market is "optical image stabilization", so presumably digital cameras use it to tilt their mirrors around, and high sample rates and low latency are obviously a *sine qua non* there. It has a worse offset rating of ±5°/s and a better noise density rate of 0.006°/s/√Hz. It also has a built-in LPF, which goes up to 350 Hz, but it can be disabled with the `LPF_D` bit in the `CTRL_REG3` register; it claims that this LPF imposes 7° of phase delay at 20 Hz, which is about a millisecond of latency.

I suspect that such chips can be scavenged from discarded cellphones. They would of course also be indirect pointing devices.

Inkjet printer feedback strips

Inkjet printer carriages have precise positional feedback with about 20-µm precision, typically using an optical quadrature encoder made of four differential slit photointerruptors with some integrated comparators and a strip of transparent plastic with black stripes printed on it; typically these operate at about 200 mm/s in normal printer operation, suggesting about a 10kHz encoder transition rate. If you could arrange your input device to move such a carriage, you could decode the quadrature signal and probably get submillisecond latency out of that hardware.

Inkjet-printer linear optical encoders might be used as direct or indirect pointing devices.

Acoustic surface triangulation

Last year, in the "Audio Tablet" note in Dercuano, I wrote about using the sound conducted through a surface between a stylus and two or more reference transducers to detect the distance along the surface to the stylus. The idea is that the audio lag time along the paths through the surface tells you what the distance is; then it's just a matter of damping the waves when they reach the edge of the surface so they don't rebound and give you multipath. Typical sound speeds through solids are kilometers per second, which is millimeters per microsecond, so a microsecond or so is about the right level of precision on the lag, and so the acoustic signal needs to be a few MHz, which won't propagate far through air but has no trouble with most solids. You can use pulses, noise signals, or perhaps even just the scratching of the stylus on the surface, though in that case it might lack the requisite MHz-frequency components.

(If you're using lower-frequency acoustic signals, you might not be able to use the time lag, but be forced to use the attenuation, a technique I learned from David H. "n2" Christensen, RIP, PBUH.)

This has the inherent latency of the audio propagation time, which might be up to a millisecond or so depending on how many transducers you're using, plus several microseconds to measure the correlation.

This technique should still work if the surface you're using is a screen displaying an image, whether from front projection, rear projection, or an LCD.

Encoded LEDs and photodetectors

All common LEDs, except the white ones, have submicrosecond response times, so you can modulate them at megabits per second; so do all common photodiodes, although some phototransistors are a bit slower. If you're modulating an LED with some random bit sequence, or even just a sine wave, at hundreds of kilobits per second, you should be able to run a correlation with the signal from such a photodetector (at zero lag, if they're within a meter or two) to measure the strength of the coupling between the LED and the photodetector. The correlation can be done with a simple analog chopper circuit, or digitally to get a window shape closer to the ideal boxcar; if the modulating signal is a simple sine wave, you can even use a simple tuned filter. Since LEDs and photodetectors are

somewhat directional, this coupling strength is a function not only of the distance between the devices but also their relative angles, but (unlike with a laser pointer) it typically doesn't drop off to near-zero until the LED or the photodetector are pointed nearly 90° off-axis.

If you have two such encoded LEDs mounted on an object at different angles, carrying different signals, this gives you two degrees of freedom, which allows you to separate the factor of the coupling due to the orientation of the object from a factor that combines the distance to the object and its closeness to the photodetector's optical axis. Adding two additional photodetectors gives you a total of six coupling constants, one between each photodetector-LED pair, which in theory might be enough to measure the position and orientation of the object in all six degrees of freedom; I suspect you might actually need three LEDs on the object to disambiguate orientations reliably.

Moreover, these three photodetectors are in theory sufficient for any number of objects as long as their optical signals are uncorrelated and the photodetectors don't saturate, or don't saturate much.

Measuring the relevant correlations to the necessary degree of precision should in theory take much less than a millisecond when using signals modulated at hundreds of kHz which are perfectly uncorrelated over millisecond timescales. 250kbps random bitstreams have a bit per 4 µs, so surely over 100 µs their Hamming distance will be quite large. (An even simpler alternative is that each LED could simply transmit its callsign over and over, but I suspect that will tend to perform worse.)

You can use this for either a direct pointing device positioned on a screen, perhaps for a ring worn on the hand, as long as the photodetectors are looking down at the screen from known positions on the same side as the user, or an indirect pointing device in an arbitrary place in space.

I think some virtual-reality gear from the 1990s used this approach with an ultrasonic signal rather than an optical one, thus enabling it to use the 343-µm-per-µs speed of sound in air to get distance information.

Impedance tomography

I've seen some recent papers using "impedance tomography" over a resistive surface under a dielectric layer to detect finger touches on the dielectric layer; a series of eight or so electrodes around the edge are alternately stimulated to measure the pairwise impedance between all pairs of electrodes, which changes when a finger capacitively couples some of the surface to ground, which allows you to approximate the finger position. In theory this could be done very quickly, but the papers I've seen didn't achieve submillisecond latency, so maybe there's some obstacle such as high noise. I suspect this is probably unavoidably an indirect pointing method.

Thanks

Thanks to Brandon Moore and Greg Sittler for the discussion this note arose from.

Topics

The sparsity of PEG memoization utility

Kragen Javier Sitaker, 02020-12-24 (updated 02020-12-28)
(1 minute)

Most PEG callsites can't memoize usefully: either they can't be reached by backtracking, so they can never find a hit in the memo table, or their result can't be used by backtracking, so there's no point in saving their result in the memo table. This should dramatically improve the memory and even time consumption of PEG parsers without affecting their other advantages.

The memoizability of a particular call in a PEG (an attempt to parse a particular nonterminal at a particular position) has two aspects: winkability — the ability to avoid doing any actual parsing by fetching the result from the memo table; and storability — the fact that the call's results, if stored in the memo table, will be used by a later winkable call. Both of these are potentially dependent on the entire source text, both before and after the XXX

Topics

Cheating étendue?

Kragen Javier Sitaker, 02020-12-26 (4 minutes)

One way to cheat conservation of étendue is apparently to use a highly diffusively reflective cavity with only a small pinhole in it to let light out, but in practice I don't think this works out. A large extended light source inside the cavity will tend to make the pinhole as bright per unit area as the light source itself; if the light source flashes only briefly, the light will escape through the pinhole over a longer period of time than the original light source was emitting. In theory this ought to allow you to reduce the étendue of the light source: the light is originally emitted over, say, 100 square millimeters and 4π steradians (isotropically), but then escapes through, say, 0.01 square millimeters and 2π steradians.

For the moment, let's suppose the cavity is perfectly reflective and a perfect Lambertian diffuser; there's no known way to achieve this, but it doesn't violate at least classical optics or classical thermodynamics. (There might be a quantum-physical reason it's impossible, but I don't know of one.)

For straightforward thermodynamic reasons the brightness escaping through the hole is the same as the brightness of the emitting surface: the emitter has to have the same absorptivity and emittivity, so when it turns on, the light level in the cavity rises until it's absorbing the same quantity of light that it's emitting and therefore is in thermodynamic equilibrium; at this point the entire reflective cavity is reflecting light at the same illuminance: however many lux (lm/m^2) the surface of the emitter is, that's how many lux the cavity surface is too. And the hole is the same brightness as the rest of the surface.

So if you flash the emitter for 1 nanosecond, for example, then the light will escape through the hole at the same brightness over a longer period of time, maybe 10 µs: ten thousand times less étendue in exchange for a light pulse ten thousand times longer. Of course, though, there'll be a finite rise time and an exponential falloff to zero, and depending on how the emitter works, it might extract energy from the cavity by non-optical means. (For example, if it's an LED, it would just heat up without emitting light, perhaps also driving a little photocurrent given the opportunity.)

So, in effect, the light in the cavity behaves pretty similar to an incandescent thermal mass, cooling off by emitting light through the pinhole. It's almost effectively just a cavity absorber; we don't gain much from the perfect reflective surface, although the "cooling" is proportional to the remaining energy, rather than the fourth power of the remaining energy as in Stefan–Boltzmann cooling. The emitter rather quickly will become a real incandescent thermal mass, though, and by hypothesis it's much larger than the exit pinhole, so I think even this apparent difference will turn out to be illusory: the brightness in the cavity will follow the brightness of the emitter much more quickly than it decays by escaping through the pinhole.

(You could argue that a Lambertian diffuser doesn't conserve

étendue, and is thus cheating, but (a) you can do an adequate imitation of a Lambertian diffuser with a pockmarked mirror, with lots and lots of little reflective craters covering its surface, and such a mirror *does* conserve étendue; (b) a Lambertian diffuser *increases* étendue, while the cavity system described above seemed interesting because of the possibility of *decreasing* étendue.)

There's also the issue that in fact the most reflective known materials for visible light are in fact only about 95% reflective, so if the pinhole is less than 5% of the surface area of the cavity, the system loses more energy to its walls than through the pinhole. So you really can't get much benefit from this hack with known materials anyway.

Topics

Stochastic fractional delay lines

Kragen Javier Sitaker, 02020-12-26 (9 minutes)

One of the key primitives for constructing delay-line synthesis sounds like Karplus–Strong is the fractional-delay filter, but computing this filter is often computationally more expensive than the rest of the delay line, even if it's a first-order FIR filter. I think a simple stochastic version of this filter is likely to be good enough for many applications and dramatically cheaper to compute.

If, for example, the resonance you want is a lag of 22.32 samples (at CD-DA's 44.1 ksps, that's 1976 Hz, B_6, more than an octave below the top of a piano) then a lag of 22 samples would give you 2004 Hz, about 24 cents sharp — a very conspicuous tone difference to the ear — and a lag of 23 samples would be even worse. Worse, if you're changing the delay over time (for example, for a vibrato), the sudden change in pitch would be even more conspicuous. So you *really need* a fractional-sample delay. As Julius Smith explains, the easiest way to do this is to calculate the lerp $(1 - \eta)\ y(n) + \eta y(n - 1) = y(n) + \eta(y(n - 1) - y(n))$, where η is the desired fractional delay, 0.32 in this case; this requires one multiply per sample. (There's also a feedforward first-order allpass alternative with the same computational cost, less phase distortion, no low-pass filtering loss, but less ability to handle variable delays: $y(n) = \eta(x(n) - y(n - 1)) + x(n - 1)$, with the delay being roughly $(1 - \eta)/(1 + \eta)$.)

But a multiply is much more costly than an add. A different way to achieve the same effect is to *randomly* choose a lag of 22 or 23 samples *on every sample*, with probabilities 0.68 and 0.32. This will give the desired *average* lag of 22.32 samples, but it can be done much more efficiently than a multiply: an 8-bit LFSR and an 8-bit comparator, for example, would suffice, and these take much less circuitry than even an 8×8-bit multiplier, much less an 8×16 multiplier or 8×24.

The resulting phase noise will tend to introduce white noise modulated by high-frequency components and time-domain transients of the signal. This can be somewhat diminished by switching between the lags less often than every sample, perhaps every 4–8 samples. And if the cheaper fractional-delay filter allows you to use a higher sampling rate, that may have a larger effect than this noise.

A different approach to mitigating the white-noise modulation problem is to synthetically add white noise where it wouldn't otherwise be necessary, so that this technique adds even more noise, but consistently rather than at a rate modulated by the signal's maximum slew rate and the momentary value of the lag.

This stochastic-fractional-delay technique can be applied to a variety of other applications: Paeth (and Minsky-circle) rotation of raster images (p. 754), adding vibrato or flanging to an existing audio signal, fractional-delay resonators for tone recognition (including radio-frequency tone recognition), calculating optimal sampling times for clock and data recovery in asynchronous communication, and

phased-array beamforming.

Paeth rotation

In the standard version of Paeth three-shear rotation, shearing is done by shifting rows and columns of pixels by integer numbers of pixels, which approximate fractional-pixel shifts. But the rounding of the shifts produces aliasing artifacts, which can be diminished by fractional-delay filtering. Different tradeoffs are possible here. A single row or column can be shifted by a fractional amount by sampling individual pixels, or runs of pixels, at randomly different "lags" or shift distances, which will tend to fuzz out the rows or columns that have ideal shifts further from any integer, like 33.5 pixels rather than 33.9 or 33.1, similar to how just doing bilinear filtering on them (the lerp fractional-delay approach described above for audio) would fuzz them out, but also adding noise. As before, randomly selecting lags for *runs* of samples (pixels) rather than individual samples will tend to reduce this noise.

(Shearing using an all-pass filter, again as described above for delay-line music synthesis, would eliminate the shift-dependent fuzzing-out low-pass filtering, still costing one multiply per sample. Condat, Van de Ville, and Forster-Heinlein found a way to do this with a more computationally expensive symmetrically reversible all-pass filter in 2007. Also, JOS claims the all-pass approach isn't suitable for "random access" — computing output sample *n* without computing all the *n* output samples before it, which is obviously relevant to tiled rendering of images — because it's recursive; but presumably you can apply the standard prefix-sum algorithm to the recurrence relation to get a more efficient way to do random access.)

Sampling different pixels in the same row (or column) being shifted will result in making zero copies of some of them and two copies of others, thus degrading the image somewhat. A different approach is to generate the random lags on a per-row (or per-column) basis, so all pixels in a row (or column) are shifted horizontally (respectively, vertically) by the same amount, and no pixels are lost or duplicated. This eliminates much of the information loss and also takes the extra work out of the inner loop.

A further step to reduce the information loss and extra work is to make these rounded lags of successive rows (or columns) monotonic: if there's a succession of rows to be shifted by 33.0 pixels, 33.2 pixels, 33.4, 33.6, 33.8, and 34.0, then we pick a random breakpoint among the fractional shifts where we switch from rounding to 33 to rounding to 34. This should still provide enough randomness to break up the most objectionable aliasing patterns.

Vibrato and flanging

Vibrato on a musical instrument alters its pitch slightly in a periodic manner; in most cases a good approximation can be achieved by slightly advancing and retarding the signal, although of course if taken to an extreme this would start swinging the tempo too. Flanging is pretty much by definition just a matter of retarding one copy of the signal by a variable amount. So these techniques are directly applicable.

Resonators for tone recognition

A piano string is a delay-line resonator that can recognize particular audio frequencies (or their harmonics). The same technique can be applied with a Karplus–Strong delay-line resonator in a relatively small amount of memory; also, though, if we have enough memory to store the whole input signal, we can quite reasonably convolve it with a sparse impulse train in order to detect periodic motifs in it at the relevant frequency, whether sinusoidal or of some other form, which allows us to choose a different shape for our temporal window than the exponential rise and decay the constant-space cyclic delay line would seem to limit us to. The various fractional-delay techniques described above can allow us to effectively position these impulses at fractional-sample positions.

This, of course, also works for variable-frequency and radio-frequency tones, for example for FM radio reception.

Clock and data recovery

The simplest form of clock and data recovery is just a question of extracting the phase of a periodic motif of a known frequency and shape from noisy data, and of course there's no guarantee that the period of that frequency has an integer number of samples.

Phased-array beamforming

In phased-array beamforming we sum up the signals received at many different transducers from a given source (or do the equivalent with time and causality reversed). Depending on the direction and sometimes the distance to the source, the delays from the source to these different transducers are different, and this permits us too separate the source's signal from other signals.

If everything is known except the signal, this is just a matter of applying the right lags to the different signals and adding them up. A fast way of applying different fractional-sample lags to different signals is thus useful.

(But it's hard for me to imagine that it can compete with an FX correlator on precision, so you'd probably only do it when the Fourier technique is too expensive.)

Topics

Successive-approximation UI design

Kragen Javier Sitaker, 02020-12-28 (1 minute)

In Ivan Sutherland's 1994 talk about SKETCHPAD, he explains, "The idea was that you draw the drawing first, and then fix it up later," (33'38" into the video) by adding constraints, for example by adding parallelism and equal-length constraints. And this is still pretty similar to modern constraint-driven CAD programs like FreeCAD.

But on modern computers, we could imagine a wider variety of ways of "fixing it up". For example, we could imagine smoothing out a crooked line you've drawn, or converting it into a circle arc, or connecting or disconnecting two lines, or cleaning up some text you've written, or running handwriting recognition on it. Perhaps buttons would pop up next to recently drawn objects to offer you these opportunities.

This kind of successive approximation to the state you want is not nearly as well supported in current drawing software as I think it should be.

Topics

- HCI (human-computer interaction) (p. 801) (17 notes)
- Graphics (p. 814) (10 notes)
- SKETCHPAD (p. 876) (3 notes)

Differential dividing plate

Kragen Javier Sitaker, 02020-12-31 (14 minutes)

A dividing plate has circles of evenly-spaced holes used to measure out precise divisions of the circle for machining purposes such as cutting gear teeth; by using close-fitting physical contact between hard materials with minimal thermal expansion (for example, a hole in a steel plate and a brass dowel pin shoved into it), they can easily achieve precisions far better than we can achieve by eye. "Cliff" aka "Clickspring" has speculated that such objects might date back to Hellenistic times or earlier, since you'd need *some* way to lay out the gear teeth in the Antikythera Mechanism, and a compass (without even a straightedge!) is sufficient measuring equipment to construct them. Modern dividing plates are normally used with a dividing *head*, which gears down the angles by some constant factor to increase the number of possibilities.

But if you had no gears and wanted to minimize the number of holes you had to drill, and thus the opportunities to introduce error, you could get by with a relatively small number of plates stacked on a common axis.

To divide a circle into 6 equal sectors, you can use one plate with two holes 180° apart and a second plate of the same diameter stacked atop it with two holes 60° apart. By aligning each of the four possible pairs of holes in these plates with a dowel pin (several of which seem to have been present in the Antikythera Mechanism, both as gear pivots and as rivets), we achieve four orientations of the top plate relative to the bottom, adding four positions to the two achievable with only the bottom plate. Even if both plates are present, the dowel pin can stick through the top plate, so we can bump our straightedge up against that dowel pin instead of whatever dowel pin we have stuck in the top plate. (The other side of the straightedge might, for example, run through the center of the shaft, as in Clickspring's ingenious construction.)

A 120° plate would work in precisely the same way as the 60° plate. You can think of the 120° plate as giving you the option to either add or subtract 120° from either of the two reference angles (0° and 180°).

With the 180° plate and a 90° plate, again stacked with the same diameter, we can divide the circle into 4 equal sectors. If we then use the 60° (or 120°) plate from the two 180° and two new 90° positions, we can now divide the circle into the other 8 of 12 equal parts.

Adding a fourth plate, again stacked with the same diameter, we could increase this from 12 equal divisions to 36; the possible angles between the two holes on the rim of this fourth plate are 10°, 20°, 40°, 50°, 70°, 80°, 100°, 110°, 130°, 140°, 160°, and 170°.

If at this point we wanted to continue in this balanced-ternary groove, we would add an angle of 3°20' + 10°n for some integer n and get 108 equal divisions of the circle, but for many purposes it would be more useful to be able to divide the circle by multiples of 5, so a plate with *three* holes instead of two (at 0°, 2°, 4°, all plus 10°n, thus allowing us to reach 2°, 4°, 6° (10° - 4°), etc.) would be useful.

Note that at this point we are suffering from symmetry: although there are three positions in which we can position this new plate relative to the previous one, the center position of these three offers us no additional dividing power. We'll come back to the theme of suffering from symmetry below.

So at this point, for 180 equal divisions, we have five plates, four with two holes each (plus the shaft hole) and a fifth with three holes, for a total of 11 holes, or 16 holes if we count the shaft hole. A sixth plate with two holes brings us to 360 equal divisions, 6 plates, and 13 holes. This is substantially simpler than drilling 360 precise holes into a plate. It might be less convenient to use, but when switching between angles you simply leave some of the plate pairs immobile to drop out the factors they contribute, so you can divide the circle by any of the divisors of 360: 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, 18, 20, 24, 30, 36, 40, 45, 60, 72, 90, 120, 180, and 360.

(Perhaps the Babylonians and the Vedic sages stopped there because you can't construct a regular heptagon with a compass and straightedge.)

It might be desirable to use plates of different diameters, stacked like the discs of the Towers of Hanoi, each one referenced to the disc below it with a dowel pin at its rim. At first I thought that to make this work without increasing the number of plates, though, we'd need more holes in each plate, since you can't switch the "input hole" referenced to the previous plate and the "output pin" the next plate (or final output angle) is referenced to, so if there are only two holes in the previous plate, there are only two positions for a given plate, so with only two holes a plate always moves you either clockwise or counterclockwise — you don't get to pick. So I thought you needed three holes per plate, plus the center hole, and one of them can have a reference pin permanently installed in it.

But then I realized you can *flip a plate over* if the reference pin sticks out of both sides. And that way you can either add *or* subtract. We were suffering from the reflection-plane symmetry of the plates without even noticing it!

So, by this method, to get to 360°, you still need six plates, each with a dowel pin permanently pressed into one of its holes, five with two holes and a sixth with three, a total of 13 holes, plus the shaft holes in the center.

Incidentally, the width of the dowel pin itself can be calibrated to give us a particular angle, so we can double the number of angles available by bumping our straightedge up against one side or the other of the dowel pin.

But what if we go back to movable dowel pins all at the same radius, and exploit this new possibility of flipping the plates over?

Our first plate, which I will assume is clamped down to a table or something, has two holes 180° apart, as before. Our second plate now has *three* holes, at -60°, 0°, and +90°. With these two, and the possibility of flipping the second plate, we can reach 0°, 60°, 90°, 150°, 300°, 270°, and 210° from the 0° hole on the first plate, plus 180°, 240°, 270°, 330°, 120°, 90°, and 30° from its 180° hole. So we get to 12 equal divisions of the circle with only 2 plates, 5 holes, and one dowel pin, instead of (as previously) 3 plates, 6 holes, and 2 dowel

pins. But maybe we could do better than this, because of our 14 configurations, only 12 are unique — we can reach 270° and 90° in two different ways.

What do we gain from a third flippable plate with three irregularly spaced holes? We could use, for example, -10°, 0°, and +5°, or perhaps some variant that spaces these out by some multiples of 30°. This gets us to 72 equal divisions of the circle in 3 plates, 8 holes, and two alignment pins. I think we could still do better than this, though, because the 15° increment here doesn't buy us anything.

A fourth flippable plate with holes at -1°, 0°, and +2° gets us to the traditional 360°, in 4 plates, 11 holes, and 3 alignment pins.

We could try to exploit the possibilities inherent in this scheme more fully. Suppose that our second plate, instead of having its holes at 0, -2/12, and +3/12 as before, instead has them at 0, -2/14 and +3/14? As before, this allows us to measure 2, 3, or 5 divisions in either direction from either of our two initial reference holes, which are themselves 7/14 apart. But this doesn't actually work the way we hoped: instead of getting 14 equal divisions, we get only 10 distinct positions, because we have two different ways to reach +2/14 (0 + 2/14 and 7 - 5/14), and simiarly for 5, 7, 9, and 12. By trying to be less clever, and putting the second plate's holes at 0, -1/14, and +2/14, we do in fact achieve an equal division into 14 parts with 2 plates, 5 holes, and 1 dowel pin. If we divide the first plate into thirds instead of halves, and put the second plates holes at 0, -1/21, and +2/21, we can achieve an equal division into 21 parts with 2 plates, 6 holes, and 1 dowel pin. Adding a third plate with holes at 0, -1/147, and +2/147 gives us an equal division of the circle into 147 parts with 3 plates, 9 holes, and 2 dowel pins.

All of this has a flavor rather similar to the note on the 6 Trit Variac (p. 112), but with angles rather than voltages.

If the plates are perfectly round and consistent in diameter, the central shaft is strictly speaking unnecessary: you could line the plates up by the feel of your fingers running over the edges. This is perhaps less implausible than it seems, since we know that lathe technology goes back to Old Kingdom Egypt.

Metals are not the only reasonable materials for such discs, shafts, and pins; jade would work well, as of course would various kinds of concretes and sintered ceramics, perhaps even including fired clay, particularly if foamed to improve its machinability. Granite might also be an option. Glasses such as fused quartz would be more challenging to cut without chipping, but might be feasible.

Tom Lipton of Ox Tools has demonstrated a modern alternative to dividing plates, using two plates each containing an identical circular row of identical bearing balls, which are pressed against one another to give as many divisions of the circle as there are balls in each plate. The plates are constrained to move with the balls, rather than rolling on them as in a ball bearing. These balls are routinely made spherical to submicron tolerances, and the errors that do exist are averaged over the whole row of balls, permitting enormously closer tolerances with this mechanism than with the holes bored in a conventional dividing plate.

A sort of hybrid approach that avoids the use of shafts entirely

would align adjacent pairs of discs with kinematic ball-and-V-groove mounts rather than entire rows of ball bearings or dowel pins. Each disc (perhaps except the bottommost) would have three balls on its bottom side, spaced evenly 120° apart around its rim, and (perhaps except the topmost) six radial V-grooves on its top side, in two sets of three 120°-apart grooves. The angle between the two sets of grooves would determine the contributions of this disc to the angle of the total stackup. So a single disc pair, where the bottom disc's six V-grooves are all 60° from the previous one, could divide the circle into sixths. A third disc, with its V-grooves at 0°, 120°, 240°, 90°, 210°, and 330°, bumps that up to twelfths — but not 24ths, as you might hope. The third disc adds the possibility of incrementing the angle by 90°, but not decrementing it, but since we already had 180°, we don't need it.

There are a couple ways to try to improve that situation before adding more parts or features. If we put V-grooves on both sides of a single disc, we *can* flip it over, giving us the possibility of either adding its angle, or subtracting it, as initially — but without the possibility of zero. If we alternate double-V-groove discs and three-ball discs (with the same three balls protruding from both side of the disc) then we could delete a pair of discs from the stackup to get a 0 angle, but at the expense of changing the stack's thickness, which may be a problem.

Topics

ECM engraving

Kragen Javier Sitaker, 02020-12-31 (5 minutes)

PEMTec claim that with ECM they get surface reproducibility (I think?) down to 30 nanometers. This is an extremely promising figure for micro-engraving of information or machinery on metal surfaces using movable ECM electrodes. They claim to use a process gap of some microns, a salt-water electrolyte, "an exact current pulse", "workpieces with an imaging accuracy in the lower micrometer range", and oscillating die-sink tool electrodes, and get "a surface quality of up to 0.03 micrometers."

(Magnetic impulse engraving is also potentially interesting: a large high-permittivity core brought down to a sharp needle point resting on a copper or aluminum surface, with a high-current pancake-stack coil wrapped around the core and connected through a step-down transformer to a high-voltage source with a fast switch such as a spark gap. This ought to produce, I think, enough force from the eddy currents around the sharp point to plastically indent the softer metal, but I haven't done the math to check this. I guess I could do an experiment, but if it fails, that would only provide evidence that a particular configuration didn't work, and I'd still have to do the math.)

ECM is also a potentially valuable technique for getting sharp metal conical points, flat surfaces, cylindrical surfaces, and spherical surfaces. By rotating the workpiece past an ECM "form tool" electrode you can do "ECM lathing"; if the form tool is a straight edge and also translates parallel to that edge, then small errors in the edge will be smoothed out, somewhat analogous to lapping — but permitting the formation of precise *conical* and *hyperboloid* shapes (depending on whether the edge intersects the axis) as well as cylindrical and flat. For a spherical surface, you want to use a concave circular form tool instead, and rotate it around its center of curvature.

A taut wire may be an adequate straight edge for many ECM purposes, and a taut wire being moved back and forth may be an adequate plane.

For high precision, this is superior to traditional lathing because the forces distorting and heating the workpiece and tool can be made arbitrarily low. Sometimes, though, it may be more desirable to maintain a positive fluid pressure in the gap in order to control the gap between the tool and workpiece more precisely than the position of either can be controlled independently. When this pressure can be spread over a large area, it should produce no *local* distortion, for example in the shape of the surface, only global distortion. However, in this case, only the cylindrical, flat, and spherical shapes achievable by lapping are achievable, not the wider range of shapes achievable on the traditional lathe.

(A different, widely-used electrochemical approach to sharpening is isotropic electrochemical etching; by isotropically eroding the metal by some distance d, any rounded features of radius less than d should in theory shrink to a point. This doesn't produce precise shapes but it

does produce sharp points.)

Plasmas, especially nonthermal plasmas, may be better working fluids for fluid-bearing purposes than traditional liquid electrolytes, particularly if they contain groups such as carbonyl which form low-boiling-point compounds with the workpiece metal. They would permit a much smaller process gap at a given pressure, and plasmas containing oxygen, hydrogen, or fluorine should be able to erode graphite, silicon, silicon carbide, and diamond, although in this case we are perhaps going a bit afield of ECM proper.

With a tool electrode shaped like an air-hockey puck with a needle stuck through it, the fluid-bearing technique will give extremely precise control of the process gap. To engrave precise three-dimensional shapes you still need precise positional control of the other two axes, though; while a kinematic mount consisting of six such fluid bearings able to swivel would achieve this, we wouldn't be cutting inside those bearings, so traditional piezoelectric or galvanometer approaches are probably better.

(Probably EDM is a better fit than ECM for "lathing" and "lapping", since its material removal rate is both higher and has a much sharper falloff with distance from the workpiece, but ECM will make it practical to do this with tungsten, copper, and tungsten-copper alloys, and with plasma, even semiconductors such as graphite.)

This technique should make it possible to produce, among other things, precise sharp-pointed electrodes for uses such as electrochemical engraving and scanning-probe microscopy.

Topics

Electro-etching graded-index optics in porous silicon

Kragen Javier Sitaker, 02020-12-31 (2 minutes)

Ben Krasnow, aka Applied Science, did a wonderful video on fabricating rugate filters by electro-etching heavily-doped P-type silicon wafers (≤10mΩ·cm) at 10–100 mA/cm² in aqueous HF (1:1 — 50% HF, I think w/w) mixed 1:1 with ethanol (50% v/v) as a depolarizer. The silicon superlattice thus anodized onto the surface, layer by layer, has an index of refraction determined by the electrical current density used to porosify it at that moment, and consequently has a butterfly-wing-like spectrum that is the Fourier transform of the time-domain current signal.

This is an astonishing and unique property, and it opens the door to fabricating not only cheap dichroic filters but also, by applying the current in a spatially varying way (for example, by spatially modulated UV light on a photosensitive N-type wafer during etching as Krasnow suggests, or by any of the methods described in the note on foam electro-etching (p. 652)) the fabrication of general graded-index optics and color holograms on the surface of the silicon.

Graded-index optics avoid discontinuous changes of index; if the index changes over many wavelengths rather than a fraction of a wavelength, this entirely avoids the interfacial reflections that produce the stray light that plagues optical systems.

Even more amazingly, Krasnow demonstrates how to separate the microporous silicon lattice thus formed from the silicon substrate, which is opaque to visible light, though somewhat reflective. (I suspect that the useful refractive-index property will disappear for infrared light, for which the substrate is transparent, though Krasnow claims they should work *better* at those frequencies.) By turning the current up high enough, the new layer of microporous silicon being formed underneath the previous layers is so diaphanous that a simple water wash can separate the previously-formed layers from the wafer!

One of several surprising things about this process is that HF doesn't normally etch Si; it's used as a specific wet etch in semiconductor fabrication to remove SiO_2 without attacking the silicon. Krasnow explains that, even without the current, silicon is not totally invulnerable to HF, limiting the time span of this process.

Krasnow also points out that the filter material's transmissivity to blue light cannot reach 100%.

Topics

- Materials (p. 788) (51 notes)
- Digital fabrication (p. 802) (17 notes)
- Optics (p. 843) (5 notes)
- Metamaterials (p. 943) (2 notes)

Electrodeposition welding

Kragen Javier Sitaker, 02020-12-31 (2 minutes)

To connect two pieces of metal together, commonly you weld them. This is versatile and relatively inexpensive if you have a high-power electrical supply, but it tends to stress and distort the metal, also creating a heat-affected zone with a different crystal structure, and it is challenging for some combinations of metals.

But many metals can be electrodeposited fairly easily, notably copper and nickel. What if you instead electrodeposited metal at the junction of two existing metal pieces? You could use an insulated-shaft tool with electrolyte flowing through it, similar to the kind used for EDM small hole drilling (and presumably also ECM drilling), but electrolytically depositing metal and progressively withdrawing from the hole rather than being fed into it. This could permit "welding" "without heat" (< 100°) and thus without distorting the metal. The material deposition rate would surely be much smaller than with traditional welding, but if it can be deposited in precisely the right place, you *need* much less material.

As with ECM, tool oscillation is a potentially valuable approach for increasing flow of electrolyte through the process gap.

Deep eutectic systems and molten salts offer a similar approach for metals that can't be electrodeposited from an aqueous solution. This requires far lower temperature and is consequently far less hazardous than welding, for example, magnesium by traditional methods.

Such "welding" might also be able to compensate for its low material deposition rate by simultaneously depositing material evenly over a very long seam, for example through electrolyte held in blotter paper. Without inserting the electrode into the seam, good penetration would be difficult or impossible to achieve, so you need to bevel the edges so that you can electrodeposit at the bottom of a V-groove.

A place where this technique would really shine would be in replacing "spray welding" for building up worn parts with new metal, at which point it's pretty much just ordinary electroforming.

Topics

Jigsaw blades

Kragen Javier Sitaker, 02020-12-31 (5 minutes)

Jigsaw blades break a lot. In a sense that's because the stroke of the saw is greater than the elastic limit of the sawblade material. But this is entirely avoidable.

If the stroke of the saw is *too* short, it won't cut, because all of the motion will be taken up by the elastic deformation of the sawblade and the workpiece. This is how saws for removing plaster casts avoid cutting skin: their stroke length is shorter than the skin's elastic limit. But the saw blade is typically much, much longer than the distance the chips from the workpiece have to move to detach from the workpiece; typical numbers might be 100 mm and 100 µm. The elastic limit of a hard steel might be ½% permitting about a 250-µm stroke without risk of breaking the blade, which is plenty to cut the workpiece; if this is not long enough, the jigsaw can be built bigger.

It's also necessary for the stroke length to be larger than the tooth size, or each tooth will cut a separate hole, rather than joining the holes together into a slot. A higher movement frequency can be used with smaller teeth and the same total material removal rate; moreover thinner teeth and the elimination of the breakage risk sometimes permit using thinner blades and thus lower total power; but sometimes this undesirably reduces the achievable kerf curvature.

A typical electric jigsaw blade might move 1 m/s at 50 Hz. The speed of sound in steel is about 4 km/s, so a 100-mm-long stretched-tight steel jigsaw blade will move more or less as a rigid body at frequencies below about 40 kHz. Moving 250 µm twice per cycle at 40 kHz would be 20 m/s, so such an ultrasonic jigsaw could probably cut at a higher speed than a regular jigsaw without risk of breaking the blade, at least if there's some way to clear the chips. If the blade is 100 µm square, like one of my beard hairs or these hair-fine copper wires I've been trying to solder with, and has an extra 50% of non-tensile-load-bearing mass of teeth on one side, it weighs 120 µg/mm and so has a total mass of some 12 mg. Accelerating it by 40 m/s in half of a 40 kHz would require 3.2 Mm/s/s, or 330 thousand gees of acceleration, which works out to almost 40 newtons with this mass, thus a stress of 40 MPa at the pulling end, about 4% of the strength of steel.

This kind of sounds like an ultrasonic cheesecutter that can cut through brass, mild steel, glass, bone, fingernails, hard plastics, fired clay, concrete, and maybe wood and granite, but not actual cheese as such, or your skin, or turkey.

Other possibilities to alleviate these compromises include using a blade with omnidirectional teeth (for example a single helical tooth, like a buttress-thread screw), which has no minimum kerf curvature radius and can also be rotated between cutting strokes; mounting hard teeth (whether high-speed steel or something like tungsten carbide) on a softer blade that can stretch further; and force feedback through electronics that stop pulling on the blade when overload is detected; or coupling the saw frame to the jigsaw blade through a

lightweight spring that limits the force over the saw's normal stroke. But I'm kind of excited about this ultrasonic cheesecutter thing.

To actually make it work you probably need synchronized but mechanically weakly coupled pullers at the two ends of the sawblade, like a two-man sawing team, rather than hoping the saw frame will move as a sufficiently solid body. By controlling the amount of slack with some sort of feedback, they ought to be able to keep the tension on the blade relatively constant. Strain gauges in a lightweight saw frame occur to me as one possibility.

The mechanical power going *into* the wire is about 20 m/s · 40 N = 800 W, but almost all of that is being transmitted from one sawblade puller to the other over the wire, then returned 25 µs later; only a small amount of it goes into the workpiece being cut. You'd still probably have to water-cool it.

An interesting feature of this device is that, because it runs at 40 kHz, its cutting action should be uncannily almost silent.

A vibrating engraver or scraper that works at scales, frequencies, and powers like this, rather than the usual 50 Hz or so, would also be very interesting. It could push its hardened tip into the workpiece as per normal.

Topics

Table text

Kragen Javier Sitaker, 02020-12-31 (4 minutes)

"Plain ASCII text files" traditionally means files that could be interpreted directly by an ASR-33, with a CR LF sequence at the end of each line and a fixed-width font. Unix simplified this by eliminating the CRs, and CRT terminals simplified it by eliminating overstrikes. Nowadays we've usually extended this to UTF-8 Unicode text and sometimes ANSI color and other SGR escape sequences, and for a program with a terminal interface, other escape codes.

But this is terribly limiting. Usually we have only a single font size (though the VT100 did support double-width and double-height characters, most modern terminal emulators don't support them, and that's not much of an improvement really) and the font still has to be fixed-width. Otherwise our carefully vertically aligned tables and ASCII art will get mangled by the unknowable font metrics of the user's viewer.

Formatting those fixed-width-font tables and ASCII art isn't easy, either. It takes substantially more code than just spewing out some strings.

Markdown, or some variant thereof, might be a reasonable choice; you could write a Markdown or Markdown-variant terminal program. CommonMark doesn't support tables, but pandoc, GitHub-flavored Markdown, and PHP Markdown Extra do.

It occurs to me that a slight variation on the ordinary Unix interpretation of ASCII or UTF-8 text could work, as well. Suppose you consider a text (any text) to consist of a sequence of tables, where the rows are separated by LF (^J), and the tables are separated by blank lines consisting of two consecutive LFs (^J^J). Thus ordinary paragraphs are single-column tables. Then, instead of treating TAB (^I) to instruct a terminal to move the cursor to the next tab stop, treat it as instructing the terminal to move the cursor to the next *column*; enough space for each column is allocated to hold its contents, which means that text in subsequent rows of the table can expand them, moving previously displayed text to the right.

This form of output is clearly very easy to produce in a program, and it can be reasonably copied and pasted between programs. In fact lots of programs already accept a table in this format as input or produce it as output, under the name TSV, "tab-separated values".

However, no self-respecting programmer would rest content without adding recursion. So if we use the characters ^R and ^T (DC2 and DC4) to begin and end *nested* tables (or blank-line separated sequences of tables), we gain new and exciting abilities:

• We can put a paragraph of text in a table cell, as long as we wrap it beforehand, just by beginning it with ^R and ending it with ^T.
• We can put a header across the top of a whole table by beginning the table with ^R and ending it with ^T, so that the header isn't really part of the table.

• If the vertical layout of the table is well-defined, we can split a table into vertical slices with their own headers by putting each of the vertical slices in its own table cell.
• In general we can do Tk-style packing layout or TeX-style vbox/hbox layout by nesting "tables" each consisting of just one row or just one column.

A document in this format isn't merely *readable*, it's also *editable* at the character level, although deleting a ^T or inserting a ^R may have surprising and exciting results. Incremental relayout is vastly easier than with the CSS box model. And proportional fonts don't inconvenience the table layout in the slightest.

Still, I feel like this is maybe more of a 1995 protocol or format design than a 2025 design.

Topics

Notes concerning "Materials"

• Sodium silicate (p. 86) 02020-06-04 (32 minutes)
• Ultra machining (p. 200) 02020-07-06 (updated 02020-07-18) (5 minutes)
• Pyrolysis 3-D printing (p. 242) 02020-08-02 (updated 02020-11-24) (20 minutes)
• Machine teeth (p. 251) 02020-08-02 (updated 02020-12-31) (8 minutes)
• 3-D printing iron by electrodeposition? (p. 255) 02020-08-15 (11 minutes)
• Peroxide and bleach (p. 259) 02020-08-15 (2 minutes)
• Cyclic fabrication systems (p. 260) 02020-08-17 (updated 02020-09-10) (56 minutes)
• Foil-marking glass (p. 277) 02020-08-18 (4 minutes)
• Inductively-coupled plasma torches (p. 279) 02020-09-10 (5 minutes)
• Oxygen generator rocket (p. 281) 02020-09-10 (1 minute)
• Phosphate precipitation (p. 284) 02020-09-10 (12 minutes)
• Smart plumbing (p. 290) 02020-09-10 (updated 02020-09-12) (11 minutes)
• Inorganic burnout (p. 294) 02020-09-11 (updated 02020-09-12) (18 minutes)
• Micro material sorting (p. 300) 02020-09-12 (2 minutes)
• An index of the 1880 edition of Cooley's Cyclopædia (p. 305) 02020-09-17 (updated 02020-10-23) (9 minutes)
• Copper salts (p. 317) 02020-09-21 (updated 02020-09-23) (8 minutes)
• Hot fabrication (p. 320) 02020-09-21 (updated 02020-09-23) (16 minutes)
• Aluminum-air batteries (p. 326) 02020-09-23 (4 minutes)
• Solar netting (p. 330) 02020-09-23 (9 minutes)
• Mild bases (p. 333) 02020-09-23 (updated 02020-10-01) (3 minutes)

• Magnesium fuel (p. 335) 02020-09-23 (updated 02020-10-09) (13 minutes)
• Ancient batteries (p. 340) 02020-09-23 (updated 02020-12-31) (4 minutes)
• Modern material processing (p. 342) 02020-09-24 (updated 02020-09-26) (8 minutes)
• Materials shopping list (p. 345) 02020-09-25 (updated 02020-12-20) (1 minute)
• Reducing sucrose (p. 354) 02020-09-30 (7 minutes)
• Wang tile chemicals (p. 357) 02020-09-30 (updated 02020-12-31) (2 minutes)
• Scraping Sciencemadness (p. 358) 02020-10-01 (updated 02020-10-05) (4 minutes)
• Lithium fuel (p. 371) 02020-10-04 (7 minutes)
• Globoflexia (p. 374) 02020-10-05 (updated 02020-10-10) (37 minutes)
• Oscillating flexion (p. 440) 02020-10-15 (updated 02020-10-16)

(11 minutes)
• Plaster foam (p. 453) 02020-10-16 (updated 02020-11-08) (8 minutes)
• Muriate thermal mass (p. 459) 02020-10-18 (updated 02020-10-28) (11 minutes)
• Calcium strengthening (p. 463) 02020-10-21 (updated 02020-10-24) (23 minutes)
• Abbe-limited DRO (p. 476) 02020-10-24 (updated 02020-12-31) (11 minutes)
• Desiccant climate control (p. 489) 02020-10-27 (updated 02020-11-24) (31 minutes)
• Foaming infiltration (p. 524) 02020-11-06 (1 minute)
• Pit firing (p. 527) 02020-11-06 (3 minutes)
• Rosining chips (p. 559) 02020-11-08 (2 minutes)
• Cold plasma (p. 560) 02020-11-08 (updated 02020-11-24) (14 minutes)
• Cutting steel with steam (p. 565) 02020-11-11 (1 minute)
• Adiabatic separation (p. 575) 02020-11-12 (updated 02020-11-14) (14 minutes)
• Mica composites (p. 582) 02020-11-14 (3 minutes)
• Lava time capsule (p. 633) 02020-11-24 (8 minutes)
• Foam electro-etching and related techniques (p. 652) 02020-11-26 (updated 02020-12-31) (10 minutes)
• Yablochkov arc cutter (p. 700) 02020-12-09 (1 minute)
• Materials YouTube (p. 744) 02020-12-16 (updated 02020-12-17) (1 minute)
• Electroforming networks (p. 748) 02020-12-22 (3 minutes)
• Time-scale material processing (p. 750) 02020-12-22 (3 minutes)
• ECM engraving (p. 780) 02020-12-31 (5 minutes)
• Electro-etching graded-index optics in porous silicon (p. 782) 02020-12-31 (2 minutes)
• Electrodeposition welding (p. 783) 02020-12-31 (2 minutes)

Notes concerning “Contrivances”

• Optimized finger joints (p. 63) 02020-05-16 (4 minutes)
• Solar furnace CPC (p. 65) 02020-05-16 (12 minutes)
• Slide rule addition (p. 183) 02020-06-22 (3 minutes)
• Trying to drive a speaker with a buck converter (p. 191) 02020-06-29 (4 minutes)
• Retro teletext (p. 229) 02020-07-18 (updated 02020-07-23) (18 minutes)
• Machine teeth (p. 251) 02020-08-02 (updated 02020-12-31) (8 minutes)
• Inductively-coupled plasma torches (p. 279) 02020-09-10 (5 minutes)
• Oxygen generator rocket (p. 281) 02020-09-10 (1 minute)
• The programmable world (p. 289) 02020-09-10 (0 minutes)
• Spark gap logic (p. 309) 02020-09-20 (updated 02020-12-16) (25 minutes)
• Aluminum-air batteries (p. 326) 02020-09-23 (4 minutes)
• Modern material processing (p. 342) 02020-09-24 (updated 02020-09-26) (8 minutes)
• Level shifter (p. 411) 02020-10-08 (updated 02020-10-10) (9 minutes)
• A seamless CMG-driven walker (p. 420) 02020-10-11 (updated 02020-10-12) (6 minutes)
• Rigid glider (p. 422) 02020-10-12 (1 minute)
• VGA oscilloscope? (p. 425) 02020-10-13 (5 minutes)
• Wire machines (p. 427) 02020-10-13 (updated 02020-12-31) (12 minutes)
• Thermistors, resistance temperature detectors, and other thermal sensors (p. 431) 02020-10-14 (updated 02020-11-06) (12 minutes)
• Atkinson differential blower (p. 435) 02020-10-14 (updated 02020-12-31) (10 minutes)
• Oscillating flexion (p. 440) 02020-10-15 (updated 02020-10-16) (11 minutes)
• Minimal cost computer (p. 471) 02020-10-23 (updated 02020-12-01) (12 minutes)
• Abbe-limited DRO (p. 476) 02020-10-24 (updated 02020-12-31) (11 minutes)
• Desiccant climate control (p. 489) 02020-10-27 (updated 02020-11-24) (31 minutes)
• Pit firing (p. 527) 02020-11-06 (3 minutes)
• Swashplate screwdriver (p. 536) 02020-11-06 (1 minute)
• Thermal expansion speaker (p. 537) 02020-11-06 (1 minute)
• Copper segelín (p. 538) 02020-11-06 (updated 02020-11-08) (19 minutes)
• Alien screws (p. 544) 02020-11-06 (updated 02020-11-11) (4 minutes)
• Cutting steel with steam (p. 565) 02020-11-11 (1 minute)
• Improvised humidity sensors with PET dielectric spectroscopy (p. 566) 02020-11-11 (3 minutes)
• Adiabatic separation (p. 575) 02020-11-12 (updated 02020-11-14)

(14 minutes)
• The rep-2 cuboid (p. 580) 02020-11-13 (5 minutes)
• Capacitor meter (p. 589) 02020-11-16 (updated 02020-12-03) (26 minutes)
• Oscilloscope superresolution via compressed sensing? (p. 611) 02020-11-17 (1 minute)
• Relay buzzer (p. 629) 02020-11-23 (2 minutes)
• Lava time capsule (p. 633) 02020-11-24 (8 minutes)
• Lenticular air bearing (p. 636) 02020-11-24 (2 minutes)
• A field-programmable RTL array: a more efficient alternative to FPGAs? (p. 643) 02020-11-26 (updated 02020-11-27) (11 minutes)
• A reverse-biased diode thermometer (p. 660) 02020-11-27 (9 minutes)
• My very first opamp (p. 665) 02020-11-27 (4 minutes)
• Majority logic with DRAM sense amps (p. 687) 02020-12-09 (30 minutes)
• Yablochkov arc cutter (p. 700) 02020-12-09 (1 minute)
• Light pen latency (p. 760) 02020-12-23 (updated 02020-12-28) (29 minutes)
• Differential dividing plate (p. 776) 02020-12-31 (14 minutes)

Notes concerning “Electronics”

• Electronics kit (p. 81) 02020-05-23 (updated 02020-12-20) (14 minutes)
• A 6-bit “variac casero” (p. 112) 02020-06-06 (22 minutes)
• Ghettobotics soldering iron (p. 124) 02020-06-17 (4 minutes)
• Trying to drive a speaker with a buck converter (p. 191) 02020-06-29 (4 minutes)
• Retro teletext (p. 229) 02020-07-18 (updated 02020-07-23) (18 minutes)
• Notable quotes from Steinmetz’s 1892 hysteresis paper (p. 288) 02020-09-10 (2 minutes)
• Spark gap logic (p. 309) 02020-09-20 (updated 02020-12-16) (25 minutes)
• Aluminum-air batteries (p. 326) 02020-09-23 (4 minutes)
• Level shifter (p. 411) 02020-10-08 (updated 02020-10-10) (9 minutes)
• VGA oscilloscope? (p. 425) 02020-10-13 (5 minutes)
• Thermistors, resistance temperature detectors, and other thermal sensors (p. 431) 02020-10-14 (updated 02020-11-06) (12 minutes)
• Minimal cost computer (p. 471) 02020-10-23 (updated 02020-12-01) (12 minutes)
• LED computation? (p. 480) 02020-10-25 (5 minutes)
• Some of the cheapest memory ICs (p. 488) 02020-10-27 (updated 02020-10-30) (1 minute)
• Bluepill aspirations (p. 499) 02020-10-30 (updated 02020-11-01) (9 minutes)
• Multimeter metrology (p. 502) 02020-11-01 (updated 02020-11-27) (23 minutes)
• Guide to finding datasheets and avoiding malicious datasheet SEO sites (p. 509) 02020-11-02 (updated 02020-12-22) (7 minutes)
• Dead bugging (p. 514) 02020-11-04 (3 minutes)
• Ghettobotics nonshopping list (p. 516) 02020-11-04 (updated 02020-12-21) (22 minutes)
• Machine-readable PNG circuit diagram watermarks (p. 529) 02020-11-06 (1 minute)
• Arduino support for STM32 (p. 530) 02020-11-06 (10 minutes)
• Thermal expansion speaker (p. 537) 02020-11-06 (1 minute)
• Copper segelín (p. 538) 02020-11-06 (updated 02020-11-08) (19 minutes)
• Rosining chips (p. 559) 02020-11-08 (2 minutes)
• Improvised humidity sensors with PET dielectric spectroscopy (p. 566) 02020-11-11 (3 minutes)
• A compact textual format for interchange of electronic circuit designs (p. 572) 02020-11-11 (updated 02020-11-26) (1 minute)
• Improvised display options for embedded hardware development (p. 584) 02020-11-16 (updated 02020-11-17) (16 minutes)
• Capacitor meter (p. 589) 02020-11-16 (updated 02020-12-03) (26 minutes)
• Rebraining (p. 597) 02020-11-16 (updated 02020-12-06) (12 minutes)

Notes concerning “Performance”

• A reproducible vector-instruction VM? (p. 20) 02020-04-21 (updated 02020-06-17) (30 minutes)
• Bloomtags: a Bloom-filter tree for efficient and flexible database queries (p. 44) 02020-05-13 (21 minutes)
• Commit log transfer (p. 57) 02020-05-16 (1 minute)
• One pass sort (p. 58) 02020-05-16 (15 minutes)
• Font rendering with all-pass filters (p. 76) 02020-05-18 (7 minutes)
• Monoid prefix sum (p. 105) 02020-06-05 (13 minutes)
• An outline of the design process leading up to the Veskeno virtual machine (p. 126) 02020-06-17 (updated 02020-07-10) (88 minutes)
• Segments and blocks (p. 166) 02020-06-20 (updated 02020-12-16) (51 minutes)
• Migrating app snapshots (p. 204) 02020-07-10 (updated 02020-07-11) (14 minutes)
• Virtual machine setup (p. 209) 02020-07-10 (updated 02020-07-14) (17 minutes)
• Line-numbered ISAM buffers (p. 224) 02020-07-18 (updated 02020-07-23) (14 minutes)
• Retro teletext (p. 229) 02020-07-18 (updated 02020-07-23) (18 minutes)
• Merkle ropes (p. 415) 02020-10-09 (15 minutes)
• Minimal cost computer (p. 471) 02020-10-23 (updated 02020-12-01) (12 minutes)
• Residue number systems (p. 483) 02020-10-26 (2 minutes)
• Dictionary data structures for tiny memories (p. 573) 02020-11-12 (3 minutes)
• Muldiv (p. 641) 02020-11-26 (1 minute)
• Hardware queuing (p. 648) 02020-11-26 (updated 02020-12-16) (11 minutes)
• Caching layout (p. 678) 02020-12-03 (8 minutes)
• The Language of Choice, and other languages (p. 701) 02020-12-09 (updated 02020-12-31) (10 minutes)
• Transaction per call (p. 722) 02020-12-15 (updated 02020-12-23) (69 minutes)
• Methods for two-dimensional rotation with two or three real multiplies (p. 754) 02020-12-23 (updated 02020-12-26) (14 minutes)
• The sparsity of PEG memoization utility (p. 769) 02020-12-24 (updated 02020-12-28) (1 minute)
• Stochastic fractional delay lines (p. 772) 02020-12-26 (9 minutes)

Notes concerning “Mechanical things”

• Ballpoint SPIF (p. 36) 02020-04-25 (7 minutes)
• Optimized finger joints (p. 63) 02020-05-16 (4 minutes)
• Lantern gears (p. 165) 02020-06-20 (updated 02020-06-28) (1 minute)
• The orbital drive and stepped planetary drive (p. 235) 02020-07-28 (updated 02020-08-02) (10 minutes)
• Machine teeth (p. 251) 02020-08-02 (updated 02020-12-31) (8 minutes)
• Modern material processing (p. 342) 02020-09-24 (updated 02020-09-26) (8 minutes)
• A seamless CMG-driven walker (p. 420) 02020-10-11 (updated 02020-10-12) (6 minutes)
• Rigid glider (p. 422) 02020-10-12 (1 minute)
• Wire machines (p. 427) 02020-10-13 (updated 02020-12-31) (12 minutes)
• Atkinson differential blower (p. 435) 02020-10-14 (updated 02020-12-31) (10 minutes)
• Oscillating flexion (p. 440) 02020-10-15 (updated 02020-10-16) (11 minutes)
• Reuleaux (p. 444) 02020-10-15 (updated 02020-10-18) (19 minutes)

• Abbe-limited DRO (p. 476) 02020-10-24 (updated 02020-12-31) (11 minutes)
• Desiccant climate control (p. 489) 02020-10-27 (updated 02020-11-24) (31 minutes)
• Swashplate screwdriver (p. 536) 02020-11-06 (1 minute)
• Alien screws (p. 544) 02020-11-06 (updated 02020-11-11) (4 minutes)
• Lenticular air bearing (p. 636) 02020-11-24 (2 minutes)
• Differential dividing plate (p. 776) 02020-12-31 (14 minutes)
• Jigsaw blades (p. 784) 02020-12-31 (5 minutes)

Notes concerning “Physics”

• Ballpoint SPIF (p. 36) 02020-04-25 (7 minutes)
• Solar furnace CPC (p. 65) 02020-05-16 (12 minutes)
• Long distance radio (p. 216) 02020-07-17 (19 minutes)
• Inductively-coupled plasma torches (p. 279) 02020-09-10 (5 minutes)
• Oxygen generator rocket (p. 281) 02020-09-10 (1 minute)
• Spark gap logic (p. 309) 02020-09-20 (updated 02020-12-16) (25 minutes)
• A digital Dagarti might save your life (p. 328) 02020-09-23 (3 minutes)
• A seamless CMG-driven walker (p. 420) 02020-10-11 (updated 02020-10-12) (6 minutes)
• Oscillating flexion (p. 440) 02020-10-15 (updated 02020-10-16) (11 minutes)
• Fluidic household pumping (p. 456) 02020-10-18 (updated 02020-10-19) (7 minutes)
• Muriate thermal mass (p. 459) 02020-10-18 (updated 02020-10-28) (11 minutes)
• Multimeter metrology (p. 502) 02020-11-01 (updated 02020-11-27) (23 minutes)
• Thermal expansion speaker (p. 537) 02020-11-06 (1 minute)
• Copper segelín (p. 538) 02020-11-06 (updated 02020-11-08) (19 minutes)
• Adiabatic separation (p. 575) 02020-11-12 (updated 02020-11-14) (14 minutes)
• Geomagnetic energy harvesting is barely feasible at near-kilometer scales (p. 631) 02020-11-24 (3 minutes)
• Lava time capsule (p. 633) 02020-11-24 (8 minutes)
• Cheating étendue? (p. 770) 02020-12-26 (4 minutes)

Notes concerning “Ghettobotics”

• Electronics kit (p. 81) 02020-05-23 (updated 02020-12-20) (14 minutes)
• A 6-bit “variac casero” (p. 112) 02020-06-06 (22 minutes)
• Ghettobotics soldering iron (p. 124) 02020-06-17 (4 minutes)
• Wire machines (p. 427) 02020-10-13 (updated 02020-12-31) (12 minutes)
• Thermistors, resistance temperature detectors, and other thermal sensors (p. 431) 02020-10-14 (updated 02020-11-06) (12 minutes)
• Desiccant climate control (p. 489) 02020-10-27 (updated 02020-11-24) (31 minutes)
• Dead bugging (p. 514) 02020-11-04 (3 minutes)
• Ghettobotics nonshopping list (p. 516) 02020-11-04 (updated 02020-12-21) (22 minutes)
• Rosining chips (p. 559) 02020-11-08 (2 minutes)
• Improvised humidity sensors with PET dielectric spectroscopy (p. 566) 02020-11-11 (3 minutes)
• Random synchronous motor (p. 569) 02020-11-11 (2 minutes)
• The rep-2 cuboid (p. 580) 02020-11-13 (5 minutes)
• Mica composites (p. 582) 02020-11-14 (3 minutes)
• Improvised display options for embedded hardware development (p. 584) 02020-11-16 (updated 02020-11-17) (16 minutes)
• Capacitor meter (p. 589) 02020-11-16 (updated 02020-12-03) (26 minutes)
• Rebraining (p. 597) 02020-11-16 (updated 02020-12-06) (12 minutes)
• Electronics next project (p. 745) 02020-12-21 (updated 02020-12-22) (7 minutes)
• Light pen latency (p. 760) 02020-12-23 (updated 02020-12-28) (29 minutes)

Notes concerning “Metrology”

Notes concerning “Manufacturing”

- Ballpoint SPIF (p. 36) 02020-04-25 (7 minutes)
- Ultra machining (p. 200) 02020-07-06 (updated 02020-07-18) (5 minutes)
- 3-D printing iron by electrodeposition? (p. 255) 02020-08-15 (11 minutes)
- Cyclic fabrication systems (p. 260) 02020-08-17 (updated 02020-09-10) (56 minutes)
- Foil-marking glass (p. 277) 02020-08-18 (4 minutes)
- Hot fabrication (p. 320) 02020-09-21 (updated 02020-09-23) (16 minutes)
- Toolpath optimization (p. 347) 02020-09-27 (updated 02020-09-30) (19 minutes)
- Ancient machinists (p. 403) 02020-10-08 (26 minutes)
- Abbe-limited DRO (p. 476) 02020-10-24 (updated 02020-12-31) (11 minutes)
- Hard sticky balls (p. 525) 02020-11-06 (1 minute)
- Cold plasma (p. 560) 02020-11-08 (updated 02020-11-24) (14 minutes)
- Cutting steel with steam (p. 565) 02020-11-11 (1 minute)
- Lenticular air bearing (p. 636) 02020-11-24 (2 minutes)
- Electroforming networks (p. 748) 02020-12-22 (3 minutes)
- Differential dividing plate (p. 776) 02020-12-31 (14 minutes)
- ECM engraving (p. 780) 02020-12-31 (5 minutes)
- Electrodeposition welding (p. 783) 02020-12-31 (2 minutes)

Notes concerning “History”

• Pandemic collapse (p. 69) 02020-05-17 (updated 02020-12-16) (22 minutes)
• One big text file (p. 97) 02020-06-04 (updated 02020-06-06) (20 minutes)
• Lantern gears (p. 165) 02020-06-20 (updated 02020-06-28) (1 minute)
• Segments and blocks (p. 166) 02020-06-20 (updated 02020-12-16) (51 minutes)
• Hacker calendar (p. 185) 02020-06-28 (updated 02020-12-03) (15 minutes)
• Line-numbered ISAM buffers (p. 224) 02020-07-18 (updated 02020-07-23) (14 minutes)
• Retro teletext (p. 229) 02020-07-18 (updated 02020-07-23) (18 minutes)
• Notable quotes from Steinmetz’s 1892 hysteresis paper (p. 288) 02020-09-10 (2 minutes)
• An index of the 1880 edition of Cooley’s Cyclopædia (p. 305) 02020-09-17 (updated 02020-10-23) (9 minutes)
• Ancient batteries (p. 340) 02020-09-23 (updated 02020-12-31) (4 minutes)
• Ancient machinists (p. 403) 02020-10-08 (26 minutes)
• Reuleaux (p. 444) 02020-10-15 (updated 02020-10-18) (19 minutes)

• Multimeter metrology (p. 502) 02020-11-01 (updated 02020-11-27) (23 minutes)
• Mica composites (p. 582) 02020-11-14 (3 minutes)
• A field-programmable RTL array: a more efficient alternative to FPGAs? (p. 643) 02020-11-26 (updated 02020-11-27) (11 minutes)
• Punk zine look (p. 675) 02020-11-28 (6 minutes)
• Yablochkov arc cutter (p. 700) 02020-12-09 (1 minute)

Notes concerning "HCI (human-computer interaction)"

• Pure functional UI (p. 17) 02020-04-21 (4 minutes)
• One big text file (p. 97) 02020-06-04 (updated 02020-06-06) (20 minutes)
• Writing a shopping list in TeX (p. 110) 02020-06-05 (4 minutes)
• Modelica notes (p. 196) 02020-07-06 (updated 02020-07-07) (9 minutes)
• The programmable world (p. 289) 02020-09-10 (0 minutes)
• Globoflexia (p. 374) 02020-10-05 (updated 02020-10-10) (37 minutes)
• Nodebook: autotagging quantities for ad-hoc calculation and example-based end-user programming (p. 392) 02020-10-07 (7 minutes)
• The Spungot sentential database for end-user logic programming (p. 546) 02020-11-06 (updated 02020-12-31) (27 minutes)
• Representing E12 electronic component values musically (p. 614) 02020-11-17 (updated 02020-12-26) (16 minutes)
• Keyboard object environment (p. 624) 02020-11-19 (13 minutes)
• Scribal Basic: a 1960s language for the 02020s (p. 705) 02020-12-12 (updated 02020-12-15) (32 minutes)
• Programming in the debugger (p. 721) 02020-12-15 (2 minutes)
• Transaction per call (p. 722) 02020-12-15 (updated 02020-12-23) (69 minutes)
• Circle-portal GUI II (p. 752) 02020-12-22 (updated 02020-12-23) (4 minutes)
• Light pen latency (p. 760) 02020-12-23 (updated 02020-12-28) (29 minutes)
• Successive-approximation UI design (p. 775) 02020-12-28 (1 minute)
• Table text (p. 786) 02020-12-31 (4 minutes)

Notes concerning “Digital fabrication”

Notes concerning “Algorithms”

• Reversible parsing (p. 40) 02020-05-11 (6 minutes)
• Bloomtags: a Bloom-filter tree for efficient and flexible database queries (p. 44) 02020-05-13 (21 minutes)
• One pass sort (p. 58) 02020-05-16 (15 minutes)
• Font rendering with all-pass filters (p. 76) 02020-05-18 (7 minutes)
• Monoid prefix sum (p. 105) 02020-06-05 (13 minutes)
• Line-numbered ISAM buffers (p. 224) 02020-07-18 (updated 02020-07-23) (14 minutes)
• Globoflexia (p. 374) 02020-10-05 (updated 02020-10-10) (37 minutes)
• Merkle ropes (p. 415) 02020-10-09 (15 minutes)
• Skip list variants (p. 423) 02020-10-12 (4 minutes)
• Specular photogrammetry (p. 570) 02020-11-11 (3 minutes)
• Dictionary data structures for tiny memories (p. 573) 02020-11-12 (3 minutes)
• A field-programmable RTL array: a more efficient alternative to FPGAs? (p. 643) 02020-11-26 (updated 02020-11-27) (11 minutes)
• The Language of Choice, and other languages (p. 701) 02020-12-09 (updated 02020-12-31) (10 minutes)
• Transaction per call (p. 722) 02020-12-15 (updated 02020-12-23) (69 minutes)
• The sparsity of PEG memoization utility (p. 769) 02020-12-24 (updated 02020-12-28) (1 minute)
• Stochastic fractional delay lines (p. 772) 02020-12-26 (9 minutes)

Notes concerning “Pricing”

• Sodium silicate (p. 86) 02020-06-04 (32 minutes)
• Phosphate precipitation (p. 284) 02020-09-10 (12 minutes)
• Hot fabrication (p. 320) 02020-09-21 (updated 02020-09-23) (16 minutes)
• Solar netting (p. 330) 02020-09-23 (9 minutes)
• Muriate thermal mass (p. 459) 02020-10-18 (updated 02020-10-28) (11 minutes)
• Minimal cost computer (p. 471) 02020-10-23 (updated 02020-12-01) (12 minutes)
• Some of the cheapest memory ICs (p. 488) 02020-10-27 (updated 02020-10-30) (1 minute)
• Desiccant climate control (p. 489) 02020-10-27 (updated 02020-11-24) (31 minutes)
• Bluepill aspirations (p. 499) 02020-10-30 (updated 02020-11-01) (9 minutes)
• Multimeter metrology (p. 502) 02020-11-01 (updated 02020-11-27) (23 minutes)
• Mica composites (p. 582) 02020-11-14 (3 minutes)
• A solar panel from an LED garden light (p. 612) 02020-11-17 (updated 02020-12-01) (5 minutes)
• Lava time capsule (p. 633) 02020-11-24 (8 minutes)
• Light pen latency (p. 760) 02020-12-23 (updated 02020-12-28) (29 minutes)

Notes concerning “Microcontrollers”

Notes concerning “Thermodynamics”

- Solar furnace CPC (p. 65) 02020-05-16 (12 minutes)
- Cyclic fabrication systems (p. 260) 02020-08-17 (updated 02020-09-10) (56 minutes)
- Smart plumbing (p. 290) 02020-09-10 (updated 02020-09-12) (11 minutes)
- Hot fabrication (p. 320) 02020-09-21 (updated 02020-09-23) (16 minutes)
- Magnesium fuel (p. 335) 02020-09-23 (updated 02020-10-09) (13 minutes)
- Reducing sucrose (p. 354) 02020-09-30 (7 minutes)
- Lithium fuel (p. 371) 02020-10-04 (7 minutes)
- Atkinson differential blower (p. 435) 02020-10-14 (updated 02020-12-31) (10 minutes)
- Fluidic household pumping (p. 456) 02020-10-18 (updated 02020-10-19) (7 minutes)
- Muriate thermal mass (p. 459) 02020-10-18 (updated 02020-10-28) (11 minutes)
- Desiccant climate control (p. 489) 02020-10-27 (updated 02020-11-24) (31 minutes)
- Copper segelín (p. 538) 02020-11-06 (updated 02020-11-08) (19 minutes)
- Cheating étendue? (p. 770) 02020-12-26 (4 minutes)

Notes concerning “Programming”

• Difficulty estimation of programming tasks (p. 16) 02020-04-20 (2 minutes)
• Bloomtags: a Bloom-filter tree for efficient and flexible database queries (p. 44) 02020-05-13 (21 minutes)
• Monoid prefix sum (p. 105) 02020-06-05 (13 minutes)
• Using Numpy for non-numerical computation: what would a good example be? (p. 193) 02020-06-29 (updated 02020-06-30) (3 minutes)
• Virtual machine setup (p. 209) 02020-07-10 (updated 02020-07-14) (17 minutes)
• A generic universal entity-component simulatorium (p. 223) 02020-07-18 (1 minute)
• The Spungot sentential database for end-user logic programming (p. 546) 02020-11-06 (updated 02020-12-31) (27 minutes)
• Printf tracebacks (p. 568) 02020-11-11 (2 minutes)
• Dictionary data structures for tiny memories (p. 573) 02020-11-12 (3 minutes)
• Using C99 compound literals unjustifiably (p. 656) 02020-11-27 (6 minutes)
• Taking screenshots (p. 667) 02020-11-27 (updated 02020-12-20) (14 minutes)
• The Language of Choice, and other languages (p. 701) 02020-12-09 (updated 02020-12-31) (10 minutes)
• Transaction per call (p. 722) 02020-12-15 (updated 02020-12-23) (69 minutes)

Notes concerning “Math”

• Monoid prefix sum (p. 105) 02020-06-05 (13 minutes)
• Slide rule addition (p. 183) 02020-06-22 (3 minutes)
• Sparse sinc (p. 301) 02020-09-17 (12 minutes)
• Wang tile chemicals (p. 357) 02020-09-30 (updated 02020-12-31) (2 minutes)
• Oscillating flexion (p. 440) 02020-10-15 (updated 02020-10-16) (11 minutes)
• Intervals and gradients (p. 451) 02020-10-16 (4 minutes)
• Residue number systems (p. 483) 02020-10-26 (2 minutes)
• The rep-2 cuboid (p. 580) 02020-11-13 (5 minutes)
• A field-programmable RTL array: a more efficient alternative to FPGAs? (p. 643) 02020-11-26 (updated 02020-11-27) (11 minutes)
• Compressed imaging (p. 681) 02020-12-06 (3 minutes)
• Hierarchical state space learning (p. 718) 02020-12-14 (8 minutes)
• Methods for two-dimensional rotation with two or three real multiplies (p. 754) 02020-12-23 (updated 02020-12-26) (14 minutes)
• Stochastic fractional delay lines (p. 772) 02020-12-26 (9 minutes)

Notes concerning “Systems architecture”

• Pure functional VM (p. 19) 02020-04-21 (1 minute)
• Bitwise reproducibility (p. 39) 02020-04-25 (1 minute)
• Feeds or streams on CCNs (p. 52) 02020-05-16 (15 minutes)
• Commit log transfer (p. 57) 02020-05-16 (1 minute)
• One pass sort (p. 58) 02020-05-16 (15 minutes)
• One big text file (p. 97) 02020-06-04 (updated 02020-06-06) (20 minutes)
• An outline of the design process leading up to the Veskeno virtual machine (p. 126) 02020-06-17 (updated 02020-07-10) (88 minutes)
• Segments and blocks (p. 166) 02020-06-20 (updated 02020-12-16) (51 minutes)
• Migrating app snapshots (p. 204) 02020-07-10 (updated 02020-07-11) (14 minutes)
• A generic universal entity-component simulatorium (p. 223) 02020-07-18 (1 minute)
• Prate thoughts (p. 367) 02020-10-02 (updated 02020-12-30) (12 minutes)
• Transaction per call (p. 722) 02020-12-15 (updated 02020-12-23) (69 minutes)

Notes concerning “Practical”

• Writing a shopping list in TeX (p. 110) 02020-06-05 (4 minutes)
• Importing the WHO’s COVID-19 data into SQLite (p. 202) 02020-07-10 (2 minutes)
• Virtual machine setup (p. 209) 02020-07-10 (updated 02020-07-14) (17 minutes)
• Scraping Sciencemadness (p. 358) 02020-10-01 (updated 02020-10-05) (4 minutes)
• Wire machines (p. 427) 02020-10-13 (updated 02020-12-31) (12 minutes)
• COVID-19 risk and vitamin D (p. 484) 02020-10-27 (updated 02020-10-28) (12 minutes)
• Guide to finding datasheets and avoiding malicious datasheet SEO sites (p. 509) 02020-11-02 (updated 02020-12-22) (7 minutes)
• Ghettobotics nonshopping list (p. 516) 02020-11-04 (updated 02020-12-21) (22 minutes)
• Arduino support for STM32 (p. 530) 02020-11-06 (10 minutes)
• Microcontroller inventory (p. 620) 02020-11-18 (updated 02020-11-28) (4 minutes)
• Taking screenshots (p. 667) 02020-11-27 (updated 02020-12-20) (14 minutes)
• Materials YouTube (p. 744) 02020-12-16 (updated 02020-12-17) (1 minute)

Notes concerning “Security”

• Static hypertext on CCN (p. 51) 02020-05-16 (2 minutes)
• Feeds or streams on CCNs (p. 52) 02020-05-16 (15 minutes)
• Segments and blocks (p. 166) 02020-06-20 (updated 02020-12-16) (51 minutes)
• Migrating app snapshots (p. 204) 02020-07-10 (updated 02020-07-11) (14 minutes)
• A digital Dagarti might save your life (p. 328) 02020-09-23 (3 minutes)
• Secure Scuttlebutt is a cool idea whose realization has fatal flaws (p. 361) 02020-10-02 (updated 02020-11-06) (17 minutes)
• Prate thoughts (p. 367) 02020-10-02 (updated 02020-12-30) (12 minutes)
• DNS Cache Rendezvous: a permissionless signaling channel for bootstrapping end-to-end connections (p. 387) 02020-10-07 (13 minutes)
• Single-bridge Tor deanonymization? (p. 396) 02020-10-07 (4 minutes)
• LOGSL: Lisp object-graph serialization language (p. 398) 02020-10-07 (updated 02020-10-09) (8 minutes)
• Merkle ropes (p. 415) 02020-10-09 (15 minutes)

Notes concerning “Energy”

• Solar furnace CPC (p. 65) 02020-05-16 (12 minutes)
• Fossil geothermal (p. 238) 02020-08-02 (updated 02020-11-13) (12 minutes)
• Oxygen generator rocket (p. 281) 02020-09-10 (1 minute)
• Smart plumbing (p. 290) 02020-09-10 (updated 02020-09-12) (11 minutes)
• Aluminum-air batteries (p. 326) 02020-09-23 (4 minutes)
• Magnesium fuel (p. 335) 02020-09-23 (updated 02020-10-09) (13 minutes)
• Lithium fuel (p. 371) 02020-10-04 (7 minutes)
• Minimal cost computer (p. 471) 02020-10-23 (updated 02020-12-01) (12 minutes)
• Desiccant climate control (p. 489) 02020-10-27 (updated 02020-11-24) (31 minutes)
• A solar panel from an LED garden light (p. 612) 02020-11-17 (updated 02020-12-01) (5 minutes)
• Geomagnetic energy harvesting is barely feasible at near-kilometer scales (p. 631) 02020-11-24 (3 minutes)

Notes concerning “Protocols”

- Static hypertext on CCN (p. 51) 02020-05-16 (2 minutes)
- Feeds or streams on CCNs (p. 52) 02020-05-16 (15 minutes)
- Commit log transfer (p. 57) 02020-05-16 (1 minute)
- Migrating app snapshots (p. 204) 02020-07-10 (updated 02020-07-11) (14 minutes)
- Secure Scuttlebutt is a cool idea whose realization has fatal flaws (p. 361) 02020-10-02 (updated 02020-11-06) (17 minutes)
- Prate thoughts (p. 367) 02020-10-02 (updated 02020-12-30) (12 minutes)
- DNS Cache Rendezvous: a permissionless signaling channel for bootstrapping end-to-end connections (p. 387) 02020-10-07 (13 minutes)
- Single-bridge Tor deanonymization? (p. 396) 02020-10-07 (4 minutes)
- Machine readable microcontroller output (p. 637) 02020-11-26 (9 minutes)
- Transaction per call (p. 722) 02020-12-15 (updated 02020-12-23) (69 minutes)

Notes concerning “Graphics”

• Font rendering with all-pass filters (p. 76) 02020-05-18 (7 minutes)
• Intervals and gradients (p. 451) 02020-10-16 (4 minutes)
• Specular photogrammetry (p. 570) 02020-11-11 (3 minutes)
• A field-programmable RTL array: a more efficient alternative to FPGAs? (p. 643) 02020-11-26 (updated 02020-11-27) (11 minutes)
• Caching layout (p. 678) 02020-12-03 (8 minutes)
• Compressed imaging (p. 681) 02020-12-06 (3 minutes)
• Circle-portal GUI II (p. 752) 02020-12-22 (updated 02020-12-23) (4 minutes)
• Methods for two-dimensional rotation with two or three real multiplies (p. 754) 02020-12-23 (updated 02020-12-26) (14 minutes)
• Stochastic fractional delay lines (p. 772) 02020-12-26 (9 minutes)
• Successive-approximation UI design (p. 775) 02020-12-28 (1 minute)

Notes concerning “Experiment report”

• Ancient batteries (p. 340) 02020-09-23 (updated 02020-12-31) (4 minutes)
• Plaster foam (p. 453) 02020-10-16 (updated 02020-11-08) (8 minutes)
• Calcium strengthening (p. 463) 02020-10-21 (updated 02020-10-24) (23 minutes)
• Bluepill aspirations (p. 499) 02020-10-30 (updated 02020-11-01) (9 minutes)
• Dead bugging (p. 514) 02020-11-04 (3 minutes)
• Ghettobotics nonshopping list (p. 516) 02020-11-04 (updated 02020-12-21) (22 minutes)
• Arduino support for STM32 (p. 530) 02020-11-06 (10 minutes)
• A solar panel from an LED garden light (p. 612) 02020-11-17 (updated 02020-12-01) (5 minutes)
• Relay buzzer (p. 629) 02020-11-23 (2 minutes)
• My very first opamp (p. 665) 02020-11-27 (4 minutes)

Notes concerning “Mathematical optimization”

- Penalized bits (p. 282) 02020-09-10 (3 minutes)
- Toolpath optimization (p. 347) 02020-09-27 (updated 02020-09-30) (19 minutes)
- Intervals and gradients (p. 451) 02020-10-16 (4 minutes)
- OCR with linear optimization (p. 526) 02020-11-06 (1 minute)
- Specular photogrammetry (p. 570) 02020-11-11 (3 minutes)
- Oscilloscope superresolution via compressed sensing? (p. 611) 02020-11-17 (1 minute)
- Compressed imaging (p. 681) 02020-12-06 (3 minutes)
- Truth table search (p. 696) 02020-12-09 (11 minutes)
- Hierarchical state space learning (p. 718) 02020-12-14 (8 minutes)

Notes concerning “Independence”

• Long distance radio (p. 216) 02020-07-17 (19 minutes)
• Cyclic fabrication systems (p. 260) 02020-08-17 (updated 02020-09-10) (56 minutes)
• Micro material sorting (p. 300) 02020-09-12 (2 minutes)
• DNS Cache Rendezvous: a permissionless signaling channel for bootstrapping end-to-end connections (p. 387) 02020-10-07 (13 minutes)
• Minimal cost computer (p. 471) 02020-10-23 (updated 02020-12-01) (12 minutes)
• Desiccant climate control (p. 489) 02020-10-27 (updated 02020-11-24) (31 minutes)
• Rosining chips (p. 559) 02020-11-08 (2 minutes)
• Rebraining (p. 597) 02020-11-16 (updated 02020-12-06) (12 minutes)
• Electronics next project (p. 745) 02020-12-21 (updated 02020-12-22) (7 minutes)

Notes concerning “Facepalm”

• Pandemic collapse (p. 69) 02020-05-17 (updated 02020-12-16) (22 minutes)
• Trying to drive a speaker with a buck converter (p. 191) 02020-06-29 (4 minutes)
• Peroxide and bleach (p. 259) 02020-08-15 (2 minutes)
• Oxygen generator rocket (p. 281) 02020-09-10 (1 minute)
• Wang tile chemicals (p. 357) 02020-09-30 (updated 02020-12-31) (2 minutes)
• Secure Scuttlebutt is a cool idea whose realization has fatal flaws (p. 361) 02020-10-02 (updated 02020-11-06) (17 minutes)
• Ancient machinists (p. 403) 02020-10-08 (26 minutes)
• Microcontroller inventory (p. 620) 02020-11-18 (updated 02020-11-28) (4 minutes)
• Cheating étendue? (p. 770) 02020-12-26 (4 minutes)

Notes concerning “Embedded programming”

• A digital Dagarti might save your life (p. 328) 02020-09-23 (3 minutes)
• Minimal cost computer (p. 471) 02020-10-23 (updated 02020-12-01) (12 minutes)
• Arduino support for STM32 (p. 530) 02020-11-06 (10 minutes)
• Dictionary data structures for tiny memories (p. 573) 02020-11-12 (3 minutes)
• Improvised display options for embedded hardware development (p. 584) 02020-11-16 (updated 02020-11-17) (16 minutes)
• Capacitor meter (p. 589) 02020-11-16 (updated 02020-12-03) (26 minutes)
• Keyboard object environment (p. 624) 02020-11-19 (13 minutes)
• Machine readable microcontroller output (p. 637) 02020-11-26 (9 minutes)
• AVR OSCCAL probably won’t give you an FM radio (p. 642) 02020-11-26 (2 minutes)

Notes concerning “Derctuo”

Notes concerning “Strength of materials”

• Ballpoint SPIF (p. 36) 02020-04-25 (7 minutes)
• Sodium silicate (p. 86) 02020-06-04 (32 minutes)
• Machine teeth (p. 251) 02020-08-02 (updated 02020-12-31) (8 minutes)
• Phosphate precipitation (p. 284) 02020-09-10 (12 minutes)
• Solar netting (p. 330) 02020-09-23 (9 minutes)
• Globoflexia (p. 374) 02020-10-05 (updated 02020-10-10) (37 minutes)
• Oscillating flexion (p. 440) 02020-10-15 (updated 02020-10-16) (11 minutes)
• Jigsaw blades (p. 784) 02020-12-31 (5 minutes)

Notes concerning “Refractory”

• Inductively-coupled plasma torches (p. 279) 02020-09-10 (5 minutes)
• Hot fabrication (p. 320) 02020-09-21 (updated 02020-09-23) (16 minutes)
• Globoflexia (p. 374) 02020-10-05 (updated 02020-10-10) (37 minutes)
• Plaster foam (p. 453) 02020-10-16 (updated 02020-11-08) (8 minutes)
• Calcium strengthening (p. 463) 02020-10-21 (updated 02020-10-24) (23 minutes)
• Pit firing (p. 527) 02020-11-06 (3 minutes)
• Mica composites (p. 582) 02020-11-14 (3 minutes)
• Lava time capsule (p. 633) 02020-11-24 (8 minutes)

Notes concerning “Foaming”

• Sodium silicate (p. 86) 02020-06-04 (32 minutes)
• Cyclic fabrication systems (p. 260) 02020-08-17 (updated 02020-09-10) (56 minutes)
• Inorganic burnout (p. 294) 02020-09-11 (updated 02020-09-12) (18 minutes)
• Globoflexia (p. 374) 02020-10-05 (updated 02020-10-10) (37 minutes)
• Plaster foam (p. 453) 02020-10-16 (updated 02020-11-08) (8 minutes)
• Calcium strengthening (p. 463) 02020-10-21 (updated 02020-10-24) (23 minutes)
• Foaming infiltration (p. 524) 02020-11-06 (1 minute)
• Foam electro-etching and related techniques (p. 652) 02020-11-26 (updated 02020-12-31) (10 minutes)

Notes concerning “The future”

• Pandemic collapse (p. 69) 02020-05-17 (updated 02020-12-16) (22 minutes)
• Ultra machining (p. 200) 02020-07-06 (updated 02020-07-18) (5 minutes)
• Cyclic fabrication systems (p. 260) 02020-08-17 (updated 02020-09-10) (56 minutes)
• The programmable world (p. 289) 02020-09-10 (0 minutes)
• A digital Dagarti might save your life (p. 328) 02020-09-23 (3 minutes)
• Modern material processing (p. 342) 02020-09-24 (updated 02020-09-26) (8 minutes)
• Time-scale material processing (p. 750) 02020-12-22 (3 minutes)

Notes concerning “The STM32 microcontroller family”

- Segments and blocks (p. 166) 02020-06-20 (updated 02020-12-16) (51 minutes)
- Minimal cost computer (p. 471) 02020-10-23 (updated 02020-12-01) (12 minutes)
- Bluepill aspirations (p. 499) 02020-10-30 (updated 02020-11-01) (9 minutes)
- Arduino support for STM32 (p. 530) 02020-11-06 (10 minutes)
- Capacitor meter (p. 589) 02020-11-16 (updated 02020-12-03) (26 minutes)
- Oscilloscope superresolution via compressed sensing? (p. 611) 02020-11-17 (1 minute)
- Microcontroller inventory (p. 620) 02020-11-18 (updated 02020-11-28) (4 minutes)

Notes concerning “Physical computation”

• The programmable world (p. 289) 02020-09-10 (0 minutes)
• Spark gap logic (p. 309) 02020-09-20 (updated 02020-12-16) (25 minutes)
• Wang tile chemicals (p. 357) 02020-09-30 (updated 02020-12-31) (2 minutes)
• LED computation? (p. 480) 02020-10-25 (5 minutes)
• A field-programmable RTL array: a more efficient alternative to FPGAs? (p. 643) 02020-11-26 (updated 02020-11-27) (11 minutes)
• Majority logic with DRAM sense amps (p. 687) 02020-12-09 (30 minutes)
• Truth table search (p. 696) 02020-12-09 (11 minutes)

Notes concerning “File formats”

• An outline of the design process leading up to the Veskeno virtual machine (p. 126) 02020-06-17 (updated 02020-07-10) (88 minutes)
• Nodebook: autotagging quantities for ad-hoc calculation and example-based end-user programming (p. 392) 02020-10-07 (7 minutes)
• LOGSL: Lisp object-graph serialization language (p. 398) 02020-10-07 (updated 02020-10-09) (8 minutes)
• Machine-readable PNG circuit diagram watermarks (p. 529) 02020-11-06 (1 minute)
• The Spungot sentential database for end-user logic programming (p. 546) 02020-11-06 (updated 02020-12-31) (27 minutes)
• A compact textual format for interchange of electronic circuit designs (p. 572) 02020-11-11 (updated 02020-11-26) (1 minute)
• Table text (p. 786) 02020-12-31 (4 minutes)

Notes concerning “Falstad’s circuit simulator”

Notes concerning “Electrolysis”

• 3-D printing iron by electrodeposition? (p. 255) 02020-08-15 (11 minutes)
• Copper salts (p. 317) 02020-09-21 (updated 02020-09-23) (8 minutes)
• Aluminum-air batteries (p. 326) 02020-09-23 (4 minutes)
• Ancient batteries (p. 340) 02020-09-23 (updated 02020-12-31) (4 minutes)
• Foam electro-etching and related techniques (p. 652) 02020-11-26 (updated 02020-12-31) (10 minutes)
• Electroforming networks (p. 748) 02020-12-22 (3 minutes)
• Electrodeposition welding (p. 783) 02020-12-31 (2 minutes)

Notes concerning “Communication”

- Long distance radio (p. 216) 02020-07-17 (19 minutes)
- Audio vector image (p. 513) 02020-11-04 (2 minutes)
- Machine-readable PNG circuit diagram watermarks (p. 529) 02020-11-06 (1 minute)
- Lava time capsule (p. 633) 02020-11-24 (8 minutes)
- Machine readable microcontroller output (p. 637) 02020-11-26 (9 minutes)
- AVR OSCCAL probably won’t give you an FM radio (p. 642) 02020-11-26 (2 minutes)
- A letter-by-letter Hamming code for manual ECC computation (p. 683) 02020-12-06 (updated 02020-12-16) (5 minutes)

Notes concerning “Caching”

• Pure functional UI (p. 17) 02020-04-21 (4 minutes)
• Single output build (p. 79) 02020-05-19 (4 minutes)
• One big text file (p. 97) 02020-06-04 (updated 02020-06-06) (20 minutes)
• Segments and blocks (p. 166) 02020-06-20 (updated 02020-12-16) (51 minutes)
• Migrating app snapshots (p. 204) 02020-07-10 (updated 02020-07-11) (14 minutes)
• Caching layout (p. 678) 02020-12-03 (8 minutes)
• Transaction per call (p. 722) 02020-12-15 (updated 02020-12-23) (69 minutes)

Notes concerning “Self replication”

• Machine teeth (p. 251) 02020-08-02 (updated 02020-12-31) (8 minutes)
• Cyclic fabrication systems (p. 260) 02020-08-17 (updated 02020-09-10) (56 minutes)
• Micro material sorting (p. 300) 02020-09-12 (2 minutes)
• Wire machines (p. 427) 02020-10-13 (updated 02020-12-31) (12 minutes)
• Calcium strengthening (p. 463) 02020-10-21 (updated 02020-10-24) (23 minutes)
• Jigsaw blades (p. 784) 02020-12-31 (5 minutes)

Notes concerning “Radio”

• Electronics kit (p. 81) 02020-05-23 (updated 02020-12-20) (14 minutes)
• Long distance radio (p. 216) 02020-07-17 (19 minutes)
• VGA oscilloscope? (p. 425) 02020-10-13 (5 minutes)
• Lava time capsule (p. 633) 02020-11-24 (8 minutes)
• Machine readable microcontroller output (p. 637) 02020-11-26 (9 minutes)
• AVR OSCCAL probably won’t give you an FM radio (p. 642) 02020-11-26 (2 minutes)

Notes concerning “Nostalgia”

• Electronics kit (p. 81) 02020-05-23 (updated 02020-12-20) (14 minutes)
• Slide rule addition (p. 183) 02020-06-22 (3 minutes)
• Machine-readable PNG circuit diagram watermarks (p. 529) 02020-11-06 (1 minute)
• Machine readable microcontroller output (p. 637) 02020-11-26 (9 minutes)
• Punk zine look (p. 675) 02020-11-28 (6 minutes)
• Scribal Basic: a 1960s language for the 02020s (p. 705) 02020-12-12 (updated 02020-12-15) (32 minutes)

Notes concerning “Minerals”

• Sodium silicate (p. 86) 02020-06-04 (32 minutes)
• Hot fabrication (p. 320) 02020-09-21 (updated 02020-09-23) (16 minutes)
• Muriate thermal mass (p. 459) 02020-10-18 (updated 02020-10-28) (11 minutes)
• Calcium strengthening (p. 463) 02020-10-21 (updated 02020-10-24) (23 minutes)
• Desiccant climate control (p. 489) 02020-10-27 (updated 02020-11-24) (31 minutes)
• Lava time capsule (p. 633) 02020-11-24 (8 minutes)

Notes concerning “LEDs”

• Electronics kit (p. 81) 02020-05-23 (updated 02020-12-20) (14 minutes)
• Abbe-limited DRO (p. 476) 02020-10-24 (updated 02020-12-31) (11 minutes)
• LED computation? (p. 480) 02020-10-25 (5 minutes)
• Machine readable microcontroller output (p. 637) 02020-11-26 (9 minutes)
• Compressed imaging (p. 681) 02020-12-06 (3 minutes)
• Light pen latency (p. 760) 02020-12-23 (updated 02020-12-28) (29 minutes)

Notes concerning “Latency”

• Migrating app snapshots (p. 204) 02020-07-10 (updated 02020-07-11) (14 minutes)
• Virtual machine setup (p. 209) 02020-07-10 (updated 02020-07-14) (17 minutes)
• Hardware queuing (p. 648) 02020-11-26 (updated 02020-12-16) (11 minutes)
• Caching layout (p. 678) 02020-12-03 (8 minutes)
• Transaction per call (p. 722) 02020-12-15 (updated 02020-12-23) (69 minutes)
• Light pen latency (p. 760) 02020-12-23 (updated 02020-12-28) (29 minutes)

Notes concerning “Calculation”

• An outline of the design process leading up to the Veskeno virtual machine (p. 126) 02020-06-17 (updated 02020-07-10) (88 minutes)
• Slide rule addition (p. 183) 02020-06-22 (3 minutes)
• Modelica notes (p. 196) 02020-07-06 (updated 02020-07-07) (9 minutes)
• Nodebook: autotagging quantities for ad-hoc calculation and example-based end-user programming (p. 392) 02020-10-07 (7 minutes)
• Keyboard object environment (p. 624) 02020-11-19 (13 minutes)
• A field-programmable RTL array: a more efficient alternative to FPGAs? (p. 643) 02020-11-26 (updated 02020-11-27) (11 minutes)

Notes concerning “The AVR microcontroller”

• Segments and blocks (p. 166) 02020-06-20 (updated 02020-12-16) (51 minutes)
• Capacitor meter (p. 589) 02020-11-16 (updated 02020-12-03) (26 minutes)
• Microcontroller inventory (p. 620) 02020-11-18 (updated 02020-11-28) (4 minutes)
• Machine readable microcontroller output (p. 637) 02020-11-26 (9 minutes)
• AVR OSCCAL probably won’t give you an FM radio (p. 642) 02020-11-26 (2 minutes)
• Hardware queuing (p. 648) 02020-11-26 (updated 02020-12-16) (11 minutes)

Notes concerning “Waterglass”

- Sodium silicate (p. 86) 02020-06-04 (32 minutes)
- Globoflexia (p. 374) 02020-10-05 (updated 02020-10-10) (37 minutes)
- Plaster foam (p. 453) 02020-10-16 (updated 02020-11-08) (8 minutes)
- Calcium strengthening (p. 463) 02020-10-21 (updated 02020-10-24) (23 minutes)
- Foaming infiltration (p. 524) 02020-11-06 (1 minute)

Notes concerning “Solar”

• Solar furnace CPC (p. 65) 02020-05-16 (12 minutes)
• Long distance radio (p. 216) 02020-07-17 (19 minutes)
• Solar netting (p. 330) 02020-09-23 (9 minutes)
• Desiccant climate control (p. 489) 02020-10-27 (updated 02020-11-24) (31 minutes)
• A solar panel from an LED garden light (p. 612) 02020-11-17 (updated 02020-12-01) (5 minutes)

Notes concerning “Reproducibility”

• Pure functional VM (p. 19) 02020-04-21 (1 minute)
• A reproducible vector-instruction VM? (p. 20) 02020-04-21 (updated 02020-06-17) (30 minutes)
• Bitwise reproducibility (p. 39) 02020-04-25 (1 minute)
• One big text file (p. 97) 02020-06-04 (updated 02020-06-06) (20 minutes)
• An outline of the design process leading up to the Veskeno virtual machine (p. 126) 02020-06-17 (updated 02020-07-10) (88 minutes)

Notes concerning “Optics”

• Solar furnace CPC (p. 65) 02020-05-16 (12 minutes)
• Abbe-limited DRO (p. 476) 02020-10-24 (updated 02020-12-31) (11 minutes)
• Compressed imaging (p. 681) 02020-12-06 (3 minutes)
• Cheating étendue? (p. 770) 02020-12-26 (4 minutes)
• Electro-etching graded-index optics in porous silicon (p. 782) 02020-12-31 (2 minutes)

Notes concerning “Instruction sets”

• Pure functional VM (p. 19) 02020-04-21 (1 minute)
• An outline of the design process leading up to the Veskeno virtual machine (p. 126) 02020-06-17 (updated 02020-07-10) (88 minutes)
• Segments and blocks (p. 166) 02020-06-20 (updated 02020-12-16) (51 minutes)
• Muldiv (p. 641) 02020-11-26 (1 minute)
• Hardware queuing (p. 648) 02020-11-26 (updated 02020-12-16) (11 minutes)

Notes concerning “Incremental computation”

- Single output build (p. 79) 02020-05-19 (4 minutes)
- Segments and blocks (p. 166) 02020-06-20 (updated 02020-12-16) (51 minutes)
- Toolpath optimization (p. 347) 02020-09-27 (updated 02020-09-30) (19 minutes)
- Caching layout (p. 678) 02020-12-03 (8 minutes)
- Transaction per call (p. 722) 02020-12-15 (updated 02020-12-23) (69 minutes)

Notes concerning “Household”

• Smart plumbing (p. 290) 02020-09-10 (updated 02020-09-12) (11 minutes)
• Fluidic household pumping (p. 456) 02020-10-18 (updated 02020-10-19) (7 minutes)
• Muriate thermal mass (p. 459) 02020-10-18 (updated 02020-10-28) (11 minutes)
• COVID-19 risk and vitamin D (p. 484) 02020-10-27 (updated 02020-10-28) (12 minutes)
• The rep-2 cuboid (p. 580) 02020-11-13 (5 minutes)

Notes concerning “Heating”

• Smart plumbing (p. 290) 02020-09-10 (updated 02020-09-12) (11 minutes)
• Fluidic household pumping (p. 456) 02020-10-18 (updated 02020-10-19) (7 minutes)
• Muriate thermal mass (p. 459) 02020-10-18 (updated 02020-10-28) (11 minutes)
• Desiccant climate control (p. 489) 02020-10-27 (updated 02020-11-24) (31 minutes)
• Pit firing (p. 527) 02020-11-06 (3 minutes)

Notes concerning “End-user programming”

• One big text file (p. 97) 02020-06-04 (updated 02020-06-06) (20 minutes)
• Modelica notes (p. 196) 02020-07-06 (updated 02020-07-07) (9 minutes)
• Nodebook: autotagging quantities for ad-hoc calculation and example-based end-user programming (p. 392) 02020-10-07 (7 minutes)
• The Spungot sentential database for end-user logic programming (p. 546) 02020-11-06 (updated 02020-12-31) (27 minutes)
• Programming in the debugger (p. 721) 02020-12-15 (2 minutes)

Notes concerning “Digital signal processing”

- Font rendering with all-pass filters (p. 76) 02020-05-18 (7 minutes)
- Sparse sinc (p. 301) 02020-09-17 (12 minutes)
- Residue number systems (p. 483) 02020-10-26 (2 minutes)
- Machine readable microcontroller output (p. 637) 02020-11-26 (9 minutes)
- Stochastic fractional delay lines (p. 772) 02020-12-26 (9 minutes)

Notes concerning “Debugging”

• Segments and blocks (p. 166) 02020-06-20 (updated 02020-12-16) (51 minutes)
• Printf tracebacks (p. 568) 02020-11-11 (2 minutes)
• Scribal Basic: a 1960s language for the 02020s (p. 705) 02020-12-12 (updated 02020-12-15) (32 minutes)
• Programming in the debugger (p. 721) 02020-12-15 (2 minutes)
• Transaction per call (p. 722) 02020-12-15 (updated 02020-12-23) (69 minutes)

Notes concerning “Control”

• Ultra machining (p. 200) 02020-07-06 (updated 02020-07-18) (5 minutes)
• Toolpath optimization (p. 347) 02020-09-27 (updated 02020-09-30) (19 minutes)
• Abbe-limited DRO (p. 476) 02020-10-24 (updated 02020-12-31) (11 minutes)
• Copper segelín (p. 538) 02020-11-06 (updated 02020-11-08) (19 minutes)
• Hierarchical state space learning (p. 718) 02020-12-14 (8 minutes)

Notes concerning “Composite materials”

• Pyrolysis 3-D printing (p. 242) 02020-08-02 (updated 02020-11-24) (20 minutes)
• Globoflexia (p. 374) 02020-10-05 (updated 02020-10-10) (37 minutes)
• Calcium strengthening (p. 463) 02020-10-21 (updated 02020-10-24) (23 minutes)
• Foaming infiltration (p. 524) 02020-11-06 (1 minute)
• Mica composites (p. 582) 02020-11-14 (3 minutes)

Notes concerning “Archival”

• Tentative outline of a body of knowledge (p. 120) 02020-06-06 (updated 02020-10-28) (10 minutes)
• An outline of the design process leading up to the Veskeno virtual machine (p. 126) 02020-06-17 (updated 02020-07-10) (88 minutes)
• Lava time capsule (p. 633) 02020-11-24 (8 minutes)
• A letter-by-letter Hamming code for manual ECC computation (p. 683) 02020-12-06 (updated 02020-12-16) (5 minutes)
• ECM engraving (p. 780) 02020-12-31 (5 minutes)

Notes concerning “Analog”

• Trying to drive a speaker with a buck converter (p. 191) 02020-06-29 (4 minutes)
• LED computation? (p. 480) 02020-10-25 (5 minutes)
• Capacitor meter (p. 589) 02020-11-16 (updated 02020-12-03) (26 minutes)
• My very first opamp (p. 665) 02020-11-27 (4 minutes)
• Light pen latency (p. 760) 02020-12-23 (updated 02020-12-28) (29 minutes)

Notes concerning “Zirconia”

• Machine teeth (p. 251) 02020-08-02 (updated 02020-12-31) (8 minutes)
• Hot fabrication (p. 320) 02020-09-21 (updated 02020-09-23) (16 minutes)
• Modern material processing (p. 342) 02020-09-24 (updated 02020-09-26) (8 minutes)
• Cold plasma (p. 560) 02020-11-08 (updated 02020-11-24) (14 minutes)

Notes concerning “Ultrasound”

• Audio vector image (p. 513) 02020-11-04 (2 minutes)
• Machine readable microcontroller output (p. 637) 02020-11-26 (9 minutes)
• Light pen latency (p. 760) 02020-12-23 (updated 02020-12-28) (29 minutes)
• Jigsaw blades (p. 784) 02020-12-31 (5 minutes)

Notes concerning “Text editors”

• One big text file (p. 97) 02020-06-04 (updated 02020-06-06) (20 minutes)
• Monoid prefix sum (p. 105) 02020-06-05 (13 minutes)
• Line-numbered ISAM buffers (p. 224) 02020-07-18 (updated 02020-07-23) (14 minutes)
• Table text (p. 786) 02020-12-31 (4 minutes)

Notes concerning “Steel”

- Ballpoint SPIF (p. 36) 02020-04-25 (7 minutes)
- Machine teeth (p. 251) 02020-08-02 (updated 02020-12-31) (8 minutes)
- Cutting steel with steam (p. 565) 02020-11-11 (1 minute)
- Jigsaw blades (p. 784) 02020-12-31 (5 minutes)

Notes concerning “Sensors”

• Ultra machining (p. 200) 02020-07-06 (updated 02020-07-18) (5 minutes)
• Globoflexia (p. 374) 02020-10-05 (updated 02020-10-10) (37 minutes)
• Specular photogrammetry (p. 570) 02020-11-11 (3 minutes)
• Light pen latency (p. 760) 02020-12-23 (updated 02020-12-28) (29 minutes)

Notes concerning “Python”

• A reproducible vector-instruction VM? (p. 20) 02020-04-21 (updated 02020-06-17) (30 minutes)
• Using Numpy for non-numerical computation: what would a good example be? (p. 193) 02020-06-29 (updated 02020-06-30) (3 minutes)
• LOGSL: Lisp object-graph serialization language (p. 398) 02020-10-07 (updated 02020-10-09) (8 minutes)
• Scribal Basic: a 1960s language for the 02020s (p. 705) 02020-12-12 (updated 02020-12-15) (32 minutes)

Notes concerning “Plumbing”

• Smart plumbing (p. 290) 02020-09-10 (updated 02020-09-12) (11 minutes)
• Modern material processing (p. 342) 02020-09-24 (updated 02020-09-26) (8 minutes)
• Atkinson differential blower (p. 435) 02020-10-14 (updated 02020-12-31) (10 minutes)
• Adiabatic separation (p. 575) 02020-11-12 (updated 02020-11-14) (14 minutes)

Notes concerning “Photovoltaic”

• Long distance radio (p. 216) 02020-07-17 (19 minutes)
• Copper salts (p. 317) 02020-09-21 (updated 02020-09-23) (8 minutes)
• Desiccant climate control (p. 489) 02020-10-27 (updated 02020-11-24) (31 minutes)
• A solar panel from an LED garden light (p. 612) 02020-11-17 (updated 02020-12-01) (5 minutes)

Notes concerning “Parsing”

• Reversible parsing (p. 40) 02020-05-11 (6 minutes)
• Monoid prefix sum (p. 105) 02020-06-05 (13 minutes)
• The Language of Choice, and other languages (p. 701) 02020-12-09 (updated 02020-12-31) (10 minutes)
• The sparsity of PEG memoization utility (p. 769) 02020-12-24 (updated 02020-12-28) (1 minute)

Notes concerning “Music”

Notes concerning “Layout”

• One big text file (p. 97) 02020-06-04 (updated 02020-06-06) (20 minutes)
• Line-numbered ISAM buffers (p. 224) 02020-07-18 (updated 02020-07-23) (14 minutes)
• Caching layout (p. 678) 02020-12-03 (8 minutes)
• Table text (p. 786) 02020-12-31 (4 minutes)

Notes concerning “GUIs”

• Pure functional UI (p. 17) 02020-04-21 (4 minutes)
• Caching layout (p. 678) 02020-12-03 (8 minutes)
• Transaction per call (p. 722) 02020-12-15 (updated 02020-12-23) (69 minutes)
• Circle-portal GUI II (p. 752) 02020-12-22 (updated 02020-12-23) (4 minutes)

Notes concerning “Energy harvesting”

- Long distance radio (p. 216) 02020-07-17 (19 minutes)
- Minimal cost computer (p. 471) 02020-10-23 (updated 02020-12-01) (12 minutes)
- Geomagnetic energy harvesting is barely feasible at near-kilometer scales (p. 631) 02020-11-24 (3 minutes)
- Lava time capsule (p. 633) 02020-11-24 (8 minutes)

Notes concerning “Cooling”

• Smart plumbing (p. 290) 02020-09-10 (updated 02020-09-12) (11 minutes)
• Fluidic household pumping (p. 456) 02020-10-18 (updated 02020-10-19) (7 minutes)
• Muriate thermal mass (p. 459) 02020-10-18 (updated 02020-10-28) (11 minutes)
• Desiccant climate control (p. 489) 02020-10-27 (updated 02020-11-24) (31 minutes)

Notes concerning “Coding”

• Long distance radio (p. 216) 02020-07-17 (19 minutes)
• Machine-readable PNG circuit diagram watermarks (p. 529) 02020-11-06 (1 minute)
• Machine readable microcontroller output (p. 637) 02020-11-26 (9 minutes)
• A letter-by-letter Hamming code for manual ECC computation (p. 683) 02020-12-06 (updated 02020-12-16) (5 minutes)

Notes concerning “C”

• A reproducible vector-instruction VM? (p. 20) 02020-04-21 (updated 02020-06-17) (30 minutes)
• Reversible parsing (p. 40) 02020-05-11 (6 minutes)
• Minimal cost computer (p. 471) 02020-10-23 (updated 02020-12-01) (12 minutes)
• Using C99 compound literals unjustifiably (p. 656) 02020-11-27 (6 minutes)

Notes concerning “Book notes”

• Sodium silicate (p. 86) 02020-06-04 (32 minutes)
• Notable quotes from Steinmetz’s 1892 hysteresis paper (p. 288) 02020-09-10 (2 minutes)
• An index of the 1880 edition of Cooley’s Cyclopædia (p. 305) 02020-09-17 (updated 02020-10-23) (9 minutes)
• Reuleaux (p. 444) 02020-10-15 (updated 02020-10-18) (19 minutes)

Notes concerning “Alabaster”

• Plaster foam (p. 453) 02020-10-16 (updated 02020-11-08) (8 minutes)
• Calcium strengthening (p. 463) 02020-10-21 (updated 02020-10-24) (23 minutes)
• Desiccant climate control (p. 489) 02020-10-27 (updated 02020-11-24) (31 minutes)
• Lava time capsule (p. 633) 02020-11-24 (8 minutes)

Notes concerning “Virtual machines”

• Segments and blocks (p. 166) 02020-06-20 (updated 02020-12-16) (51 minutes)
• Migrating app snapshots (p. 204) 02020-07-10 (updated 02020-07-11) (14 minutes)
• Virtual machine setup (p. 209) 02020-07-10 (updated 02020-07-14) (17 minutes)

Notes concerning “Urbit”

Notes concerning “Thermal storage”

• Smart plumbing (p. 290) 02020-09-10 (updated 02020-09-12) (11 minutes)
• Muriate thermal mass (p. 459) 02020-10-18 (updated 02020-10-28) (11 minutes)
• Desiccant climate control (p. 489) 02020-10-27 (updated 02020-11-24) (31 minutes)

Notes concerning "SKETCHPAD"

• A field-programmable RTL array: a more efficient alternative to FPGAs? (p. 643) 02020-11-26 (updated 02020-11-27) (11 minutes)
• Light pen latency (p. 760) 02020-12-23 (updated 02020-12-28) (29 minutes)
• Successive-approximation UI design (p. 775) 02020-12-28 (1 minute)

Notes concerning “Physical system simulation”

- Modelica notes (p. 196) 02020-07-06 (updated 02020-07-07) (9 minutes)
- Toolpath optimization (p. 347) 02020-09-27 (updated 02020-09-30) (19 minutes)
- Hierarchical state space learning (p. 718) 02020-12-14 (8 minutes)

Notes concerning “Ropes (the data structure)”

- Monoid prefix sum (p. 105) 02020-06-05 (13 minutes)
- Line-numbered ISAM buffers (p. 224) 02020-07-18 (updated 02020-07-23) (14 minutes)
- Merkle ropes (p. 415) 02020-10-09 (15 minutes)

Notes concerning “R”

• A reproducible vector-instruction VM? (p. 20) 02020-04-21 (updated 02020-06-17) (30 minutes)
• One big text file (p. 97) 02020-06-04 (updated 02020-06-06) (20 minutes)
• Writing a shopping list in TeX (p. 110) 02020-06-05 (4 minutes)

Notes concerning “Purification”

- Micro material sorting (p. 300) 02020-09-12 (2 minutes)
- Modern material processing (p. 342) 02020-09-24 (updated 02020-09-26) (8 minutes)
- Adiabatic separation (p. 575) 02020-11-12 (updated 02020-11-14) (14 minutes)

Notes concerning “Prolog”

• Reversible parsing (p. 40) 02020-05-11 (6 minutes)
• The Spungot sentential database for end-user logic programming (p. 546) 02020-11-06 (updated 02020-12-31) (27 minutes)
• The Language of Choice, and other languages (p. 701) 02020-12-09 (updated 02020-12-31) (10 minutes)

Notes concerning “Programming by example”

• One big text file (p. 97) 02020-06-04 (updated 02020-06-06) (20 minutes)
• Nodebook: autotagging quantities for ad-hoc calculation and example-based end-user programming (p. 392) 02020-10-07 (7 minutes)
• Programming in the debugger (p. 721) 02020-12-15 (2 minutes)

Notes concerning “Plasma”

• Inductively-coupled plasma torches (p. 279) 02020-09-10 (5 minutes)
• Cold plasma (p. 560) 02020-11-08 (updated 02020-11-24) (14 minutes)
• Yablochkov arc cutter (p. 700) 02020-12-09 (1 minute)

Notes concerning “Parsing expression grammars”

• Reversible parsing (p. 40) 02020-05-11 (6 minutes)
• The Language of Choice, and other languages (p. 701) 02020-12-09 (updated 02020-12-31) (10 minutes)
• The sparsity of PEG memoization utility (p. 769) 02020-12-24 (updated 02020-12-28) (1 minute)

Notes concerning “Merkle graphs”

- Feeds or streams on CCNs (p. 52) 02020-05-16 (15 minutes)
- Prate thoughts (p. 367) 02020-10-02 (updated 02020-12-30) (12 minutes)
- Merkle ropes (p. 415) 02020-10-09 (15 minutes)

Notes concerning “Hypertext”

• Static hypertext on CCN (p. 51) 02020-05-16 (2 minutes)
• One big text file (p. 97) 02020-06-04 (updated 02020-06-06) (20 minutes)
• Circle-portal GUI II (p. 752) 02020-12-22 (updated 02020-12-23) (4 minutes)

Notes concerning “Gradient descent”

- Penalized bits (p. 282) 02020-09-10 (3 minutes)
- OCR with linear optimization (p. 526) 02020-11-06 (1 minute)
- Hierarchical state space learning (p. 718) 02020-12-14 (8 minutes)

Notes concerning “Gearing”

• Lantern gears (p. 165) 02020-06-20 (updated 02020-06-28) (1 minute)
• The orbital drive and stepped planetary drive (p. 235) 02020-07-28 (updated 02020-08-02) (10 minutes)
• Reuleaux (p. 444) 02020-10-15 (updated 02020-10-18) (19 minutes)

Notes concerning "Publish/subscribe feeds"

- Feeds or streams on CCNs (p. 52) 02020-05-16 (15 minutes)
- Commit log transfer (p. 57) 02020-05-16 (1 minute)
- Prate thoughts (p. 367) 02020-10-02 (updated 02020-12-30) (12 minutes)

Notes concerning “Emacs”

• Writing a shopping list in TeX (p. 110) 02020-06-05 (4 minutes)
• Line-numbered ISAM buffers (p. 224) 02020-07-18 (updated 02020-07-23) (14 minutes)
• Taking screenshots (p. 667) 02020-11-27 (updated 02020-12-20) (14 minutes)

Notes concerning “Espacio de César”

• A 6-bit “variac casero” (p. 112) 02020-06-06 (22 minutes)
• Ghettobotics soldering iron (p. 124) 02020-06-17 (4 minutes)
• Ghettobotics nonshopping list (p. 516) 02020-11-04 (updated 02020-12-21) (22 minutes)

Notes concerning “Electrochemical machining”

- Cyclic fabrication systems (p. 260) 02020-08-17 (updated 02020-09-10) (56 minutes)
- ECM engraving (p. 780) 02020-12-31 (5 minutes)
- Electrodeposition welding (p. 783) 02020-12-31 (2 minutes)

Notes concerning “Distributed systems”

- Segments and blocks (p. 166) 02020-06-20 (updated 02020-12-16) (51 minutes)
- Migrating app snapshots (p. 204) 02020-07-10 (updated 02020-07-11) (14 minutes)
- Transaction per call (p. 722) 02020-12-15 (updated 02020-12-23) (69 minutes)

Notes concerning “Digital logic”

- Muldiv (p. 641) 02020-11-26 (1 minute)
- A field-programmable RTL array: a more efficient alternative to FPGAs? (p. 643) 02020-11-26 (updated 02020-11-27) (11 minutes)
- Truth table search (p. 696) 02020-12-09 (11 minutes)

Notes concerning “Desiccants”

• Smart plumbing (p. 290) 02020-09-10 (updated 02020-09-12) (11 minutes)
• Muriate thermal mass (p. 459) 02020-10-18 (updated 02020-10-28) (11 minutes)
• Desiccant climate control (p. 489) 02020-10-27 (updated 02020-11-24) (31 minutes)

Notes concerning “Databases”

• Bloomtags: a Bloom-filter tree for efficient and flexible database queries (p. 44) 02020-05-13 (21 minutes)
• The Spungot sentential database for end-user logic programming (p. 546) 02020-11-06 (updated 02020-12-31) (27 minutes)
• Transaction per call (p. 722) 02020-12-15 (updated 02020-12-23) (69 minutes)

Notes concerning “Crackpots”

- Convincingness (p. 164) 02020-06-20 (1 minute)
- Ancient batteries (p. 340) 02020-09-23 (updated 02020-12-31) (4 minutes)
- Ancient machinists (p. 403) 02020-10-08 (26 minutes)

Notes concerning “Covid”

• Pandemic collapse (p. 69) 02020-05-17 (updated 02020-12-16) (22 minutes)
• Importing the WHO’s COVID-19 data into SQLite (p. 202) 02020-07-10 (2 minutes)
• COVID-19 risk and vitamin D (p. 484) 02020-10-27 (updated 02020-10-28) (12 minutes)

Notes concerning “Constraint satisfaction”

• One big text file (p. 97) 02020-06-04 (updated 02020-06-06) (20 minutes)
• The Spungot sentential database for end-user logic programming (p. 546) 02020-11-06 (updated 02020-12-31) (27 minutes)
• Truth table search (p. 696) 02020-12-09 (11 minutes)

Notes concerning “Concurrency”

• Segments and blocks (p. 166) 02020-06-20 (updated 02020-12-16) (51 minutes)
• Hardware queuing (p. 648) 02020-11-26 (updated 02020-12-16) (11 minutes)
• Transaction per call (p. 722) 02020-12-15 (updated 02020-12-23) (69 minutes)

Notes concerning “Concrete”

• Sodium silicate (p. 86) 02020-06-04 (32 minutes)
• Globoflexia (p. 374) 02020-10-05 (updated 02020-10-10) (37 minutes)
• Calcium strengthening (p. 463) 02020-10-21 (updated 02020-10-24) (23 minutes)

Notes concerning “Ceramic”

• Sodium silicate (p. 86) 02020-06-04 (32 minutes)
• Globoflexia (p. 374) 02020-10-05 (updated 02020-10-10) (37 minutes)
• Calcium strengthening (p. 463) 02020-10-21 (updated 02020-10-24) (23 minutes)

Notes concerning “Build systems”

• Single output build (p. 79) 02020-05-19 (4 minutes)
• One big text file (p. 97) 02020-06-04 (updated 02020-06-06) (20 minutes)
• Transaction per call (p. 722) 02020-12-15 (updated 02020-12-23) (69 minutes)

Notes concerning “Automatic differentiation”

- Segments and blocks (p. 166) 02020-06-20 (updated 02020-12-16) (51 minutes)
- Penalized bits (p. 282) 02020-09-10 (3 minutes)
- Intervals and gradients (p. 451) 02020-10-16 (4 minutes)

Notes concerning “Audio”

• Trying to drive a speaker with a buck converter (p. 191) 02020-06-29 (4 minutes)
• Representing E12 electronic component values musically (p. 614) 02020-11-17 (updated 02020-12-26) (16 minutes)
• Light pen latency (p. 760) 02020-12-23 (updated 02020-12-28) (29 minutes)

Notes concerning “Art”

- Inspiration (p. 439) 02020-10-15 (3 minutes)
- Lava time capsule (p. 633) 02020-11-24 (8 minutes)
- Punk zine look (p. 675) 02020-11-28 (6 minutes)

Notes concerning “Arrays”

• A reproducible vector-instruction VM? (p. 20) 02020-04-21 (updated 02020-06-17) (30 minutes)
• Segments and blocks (p. 166) 02020-06-20 (updated 02020-12-16) (51 minutes)
• Using Numpy for non-numerical computation: what would a good example be? (p. 193) 02020-06-29 (updated 02020-06-30) (3 minutes)

Notes concerning “Arduino”

• Arduino support for STM32 (p. 530) 02020-11-06 (10 minutes)
• Microcontroller inventory (p. 620) 02020-11-18 (updated 02020-11-28) (4 minutes)
• Machine readable microcontroller output (p. 637) 02020-11-26 (9 minutes)

Notes concerning “Archaeology”

Notes concerning “Yttria”

• Hot fabrication (p. 320) 02020-09-21 (updated 02020-09-23) (16 minutes)
• Modern material processing (p. 342) 02020-09-24 (updated 02020-09-26) (8 minutes)

Notes concerning “Web scraping”

• Importing the WHO’s COVID-19 data into SQLite (p. 202) 02020-07-10 (2 minutes)
• Scraping Sciencemadness (p. 358) 02020-10-01 (updated 02020-10-05) (4 minutes)

Notes concerning “Veskeno”

Notes concerning “Utopias”

• Hacker calendar (p. 185) 02020-06-28 (updated 02020-12-03) (15 minutes)
• Cyclic fabrication systems (p. 260) 02020-08-17 (updated 02020-09-10) (56 minutes)

Notes concerning “Transactions”

• Segments and blocks (p. 166) 02020-06-20 (updated 02020-12-16) (51 minutes)
• Transaction per call (p. 722) 02020-12-15 (updated 02020-12-23) (69 minutes)

Notes concerning “Toxicology”

- Copper salts (p. 317) 02020-09-21 (updated 02020-09-23) (8 minutes)
- Mild bases (p. 333) 02020-09-23 (updated 02020-10-01) (3 minutes)

Notes concerning “TeX”

- One big text file (p. 97) 02020-06-04 (updated 02020-06-06) (20 minutes)
- Writing a shopping list in TeX (p. 110) 02020-06-05 (4 minutes)

Notes concerning “Ternary”

- A 6-bit “variac casero” (p. 112) 02020-06-06 (22 minutes)
- Differential dividing plate (p. 776) 02020-12-31 (14 minutes)

Notes concerning “Sparkle”

- Abbe-limited DRO (p. 476) 02020-10-24 (updated 02020-12-31) (11 minutes)
- Compressed imaging (p. 681) 02020-12-06 (3 minutes)

Notes concerning “Sorting”

- One pass sort (p. 58) 02020-05-16 (15 minutes)
- Dictionary data structures for tiny memories (p. 573) 02020-11-12 (3 minutes)

Notes concerning “Small is beautiful”

• Minimal cost computer (p. 471) 02020-10-23 (updated 02020-12-01) (12 minutes)
• Circle-portal GUI II (p. 752) 02020-12-22 (updated 02020-12-23) (4 minutes)

Notes concerning “Scanning probe microscopes”

- Abbe-limited DRO (p. 476) 02020-10-24 (updated 02020-12-31) (11 minutes)
- ECM engraving (p. 780) 02020-12-31 (5 minutes)

Notes concerning “Sapphire”

• Machine teeth (p. 251) 02020-08-02 (updated 02020-12-31) (8 minutes)
• Modern material processing (p. 342) 02020-09-24 (updated 02020-09-26) (8 minutes)

Notes concerning “Rutile”

• Hot fabrication (p. 320) 02020-09-21 (updated 02020-09-23) (16 minutes)
• Cold plasma (p. 560) 02020-11-08 (updated 02020-11-24) (14 minutes)

Notes concerning “Regrettable”

• Guide to finding datasheets and avoiding malicious datasheet SEO sites (p. 509) 02020-11-02 (updated 02020-12-22) (7 minutes)
• Using C99 compound literals unjustifiably (p. 656) 02020-11-27 (6 minutes)

Notes concerning “Quotes”

- Inspiration (p. 439) 02020-10-15 (3 minutes)
- Intervals and gradients (p. 451) 02020-10-16 (4 minutes)

Notes concerning “QEMU”

• Migrating app snapshots (p. 204) 02020-07-10 (updated 02020-07-11) (14 minutes)
• Virtual machine setup (p. 209) 02020-07-10 (updated 02020-07-14) (17 minutes)

Notes concerning “Projectors”

• Compressed imaging (p. 681) 02020-12-06 (3 minutes)
• Light pen latency (p. 760) 02020-12-23 (updated 02020-12-28) (29 minutes)

Notes concerning “Programming languages”

• The Spungot sentential database for end-user logic programming (p. 546) 02020-11-06 (updated 02020-12-31) (27 minutes)
• Scribal Basic: a 1960s language for the 02020s (p. 705) 02020-12-12 (updated 02020-12-15) (32 minutes)

Notes concerning “Prefix sums”

- Monoid prefix sum (p. 105) 02020-06-05 (13 minutes)
- Line-numbered ISAM buffers (p. 224) 02020-07-18 (updated 02020-07-23) (14 minutes)

Notes concerning “Politics”

- Pandemic collapse (p. 69) 02020-05-17 (updated 02020-12-16) (22 minutes)
- Phosphate precipitation (p. 284) 02020-09-10 (12 minutes)

Notes concerning “Pocket furnaces”

- Solar furnace CPC (p. 65) 02020-05-16 (12 minutes)
- Hot fabrication (p. 320) 02020-09-21 (updated 02020-09-23) (16 minutes)

Notes concerning “Pidgeon process”

- Lithium fuel (p. 371) 02020-10-04 (7 minutes)
- Cold plasma (p. 560) 02020-11-08 (updated 02020-11-24) (14 minutes)

Notes concerning “Phosphates”

- Phosphate precipitation (p. 284) 02020-09-10 (12 minutes)
- Calcium strengthening (p. 463) 02020-10-21 (updated 02020-10-24) (23 minutes)

Notes concerning “Paeth rotation”

- Methods for two-dimensional rotation with two or three real multiplies (p. 754) 02020-12-23 (updated 02020-12-26) (14 minutes)
- Stochastic fractional delay lines (p. 772) 02020-12-26 (9 minutes)

Notes concerning “Padauk”

• Minimal cost computer (p. 471) 02020-10-23 (updated 02020-12-01) (12 minutes)
• Hardware queuing (p. 648) 02020-11-26 (updated 02020-12-16) (11 minutes)

Notes concerning “Oscilloscopes”

• VGA oscilloscope? (p. 425) 02020-10-13 (5 minutes)
• Oscilloscope superresolution via compressed sensing? (p. 611) 02020-11-17 (1 minute)

Notes concerning “Octave”

• A reproducible vector-instruction VM? (p. 20) 02020-04-21 (updated 02020-06-17) (30 minutes)
• Sparse sinc (p. 301) 02020-09-17 (12 minutes)

Notes concerning “Numpy”

• A reproducible vector-instruction VM? (p. 20) 02020-04-21 (updated 02020-06-17) (30 minutes)
• Using Numpy for non-numerical computation: what would a good example be? (p. 193) 02020-06-29 (updated 02020-06-30) (3 minutes)

Notes concerning “Muriate of lime”

- Muriate thermal mass (p. 459) 02020-10-18 (updated 02020-10-28) (11 minutes)
- Desiccant climate control (p. 489) 02020-10-27 (updated 02020-11-24) (31 minutes)

Notes concerning “Monoids”

• Monoid prefix sum (p. 105) 02020-06-05 (13 minutes)
• Line-numbered ISAM buffers (p. 224) 02020-07-18 (updated 02020-07-23) (14 minutes)

Notes concerning “Mole people”

- Fossil geothermal (p. 238) 02020-08-02 (updated 02020-11-13) (12 minutes)
- Geomagnetic energy harvesting is barely feasible at near-kilometer scales (p. 631) 02020-11-24 (3 minutes)

Notes concerning “Minsky algorithm”

• A field-programmable RTL array: a more efficient alternative to FPGAs? (p. 643) 02020-11-26 (updated 02020-11-27) (11 minutes)
• Methods for two-dimensional rotation with two or three real multiplies (p. 754) 02020-12-23 (updated 02020-12-26) (14 minutes)

Notes concerning “Metamaterials”

• Abbe-limited DRO (p. 476) 02020-10-24 (updated 02020-12-31) (11 minutes)
• Electro-etching graded-index optics in porous silicon (p. 782) 02020-12-31 (2 minutes)

Notes concerning “Merging”

- One pass sort (p. 58) 02020-05-16 (15 minutes)
- Merkle ropes (p. 415) 02020-10-09 (15 minutes)

Notes concerning “Magnesium”

• Magnesium fuel (p. 335) 02020-09-23 (updated 02020-10-09) (13 minutes)
• Cold plasma (p. 560) 02020-11-08 (updated 02020-11-24) (14 minutes)

Notes concerning “LSM-trees (log-structured merge trees)”

- One pass sort (p. 58) 02020-05-16 (15 minutes)
- Merkle ropes (p. 415) 02020-10-09 (15 minutes)

Notes concerning “The Long Now Foundation”

- Lava time capsule (p. 633) 02020-11-24 (8 minutes)
- A field-programmable RTL array: a more efficient alternative to FPGAs? (p. 643) 02020-11-26 (updated 02020-11-27) (11 minutes)

Notes concerning “Logic”

• The Spungot sentential database for end-user logic programming (p. 546) 02020-11-06 (updated 02020-12-31) (27 minutes)
• The Language of Choice, and other languages (p. 701) 02020-12-09 (updated 02020-12-31) (10 minutes)

Notes concerning “Linux”

Notes concerning “Lime”

• Mild bases (p. 333) 02020-09-23 (updated 02020-10-01) (3 minutes)

• Calcium strengthening (p. 463) 02020-10-21 (updated 02020-10-24) (23 minutes)

Notes concerning “Kafka”

- Feeds or streams on CCNs (p. 52) 02020-05-16 (15 minutes)
- Commit log transfer (p. 57) 02020-05-16 (1 minute)

Notes concerning “The JS language”

- Monoid prefix sum (p. 105) 02020-06-05 (13 minutes)
- Scribal Basic: a 1960s language for the 02020s (p. 705) 02020-12-12 (updated 02020-12-15) (32 minutes)

Notes concerning “Interrupts”

• Hardware queuing (p. 648) 02020-11-26 (updated 02020-12-16) (11 minutes)
• Transaction per call (p. 722) 02020-12-15 (updated 02020-12-23) (69 minutes)

Notes concerning “Immediate-mode GUIs”

- Printf tracebacks (p. 568) 02020-11-11 (2 minutes)
- Scribal Basic: a 1960s language for the 02020s (p. 705) 02020-12-12 (updated 02020-12-15) (32 minutes)

Notes concerning “FPGAs”

• An outline of the design process leading up to the Veskeno virtual machine (p. 126) 02020-06-17 (updated 02020-07-10) (88 minutes)
• A field-programmable RTL array: a more efficient alternative to FPGAs? (p. 643) 02020-11-26 (updated 02020-11-27) (11 minutes)

Notes concerning “FP-persistent data structures”

- Monoid prefix sum (p. 105) 02020-06-05 (13 minutes)
- Line-numbered ISAM buffers (p. 224) 02020-07-18 (updated 02020-07-23) (14 minutes)

Notes concerning “Flying machines”

- Oxygen generator rocket (p. 281) 02020-09-10 (1 minute)
- Rigid glider (p. 422) 02020-10-12 (1 minute)

Notes concerning “Flexures”

• Wire machines (p. 427) 02020-10-13 (updated 02020-12-31) (12 minutes)
• Oscillating flexion (p. 440) 02020-10-15 (updated 02020-10-16) (11 minutes)

Notes concerning “Étendue”

- Solar furnace CPC (p. 65) 02020-05-16 (12 minutes)
- Cheating étendue? (p. 770) 02020-12-26 (4 minutes)

Notes concerning “Errors”

• An outline of the design process leading up to the Veskeno virtual machine (p. 126) 02020-06-17 (updated 02020-07-10) (88 minutes)
• Transaction per call (p. 722) 02020-12-15 (updated 02020-12-23) (69 minutes)

Notes concerning “Epistemology”

- Convincingness (p. 164) 02020-06-20 (1 minute)
- Ancient machinists (p. 403) 02020-10-08 (26 minutes)

Notes concerning “Energy efficiency”

• Muriate thermal mass (p. 459) 02020-10-18 (updated 02020-10-28) (11 minutes)
• A field-programmable RTL array: a more efficient alternative to FPGAs? (p. 643) 02020-11-26 (updated 02020-11-27) (11 minutes)

Notes concerning “Earthships”

• Smart plumbing (p. 290) 02020-09-10 (updated 02020-09-12) (11 minutes)
• Fluidic household pumping (p. 456) 02020-10-18 (updated 02020-10-19) (7 minutes)

Notes concerning “Drying”

• Muriate thermal mass (p. 459) 02020-10-18 (updated 02020-10-28) (11 minutes)
• Desiccant climate control (p. 489) 02020-10-27 (updated 02020-11-24) (31 minutes)

Notes concerning “Docker”

- One big text file (p. 97) 02020-06-04 (updated 02020-06-06) (20 minutes)
- Migrating app snapshots (p. 204) 02020-07-10 (updated 02020-07-11) (14 minutes)

Notes concerning “Corewar”

Notes concerning “Copy on write”

• Segments and blocks (p. 166) 02020-06-20 (updated 02020-12-16) (51 minutes)
• Migrating app snapshots (p. 204) 02020-07-10 (updated 02020-07-11) (14 minutes)

Notes concerning “Copper”

- Copper salts (p. 317) 02020-09-21 (updated 02020-09-23) (8 minutes)
- Mild bases (p. 333) 02020-09-23 (updated 02020-10-01) (3 minutes)

Notes concerning “Compilers”

• An outline of the design process leading up to the Veskeno virtual machine (p. 126) 02020-06-17 (updated 02020-07-10) (88 minutes)
• Transaction per call (p. 722) 02020-12-15 (updated 02020-12-23) (69 minutes)

Notes concerning “Collapse”

- Pandemic collapse (p. 69) 02020-05-17 (updated 02020-12-16) (22 minutes)
- Ancient machinists (p. 403) 02020-10-08 (26 minutes)

Notes concerning “Clusters”

• Segments and blocks (p. 166) 02020-06-20 (updated 02020-12-16) (51 minutes)
• Transaction per call (p. 722) 02020-12-15 (updated 02020-12-23) (69 minutes)

Notes concerning “Chifir”

Notes concerning “Chat”

• Secure Scuttlebutt is a cool idea whose realization has fatal flaws (p. 361) 02020-10-02 (updated 02020-11-06) (17 minutes)
• Prate thoughts (p. 367) 02020-10-02 (updated 02020-12-30) (12 minutes)

Notes concerning "Content-centric networking/named-data networking"

- Static hypertext on CCN (p. 51) 02020-05-16 (2 minutes)
- Feeds or streams on CCNs (p. 52) 02020-05-16 (15 minutes)

Notes concerning “Casting”

• Hot fabrication (p. 320) 02020-09-21 (updated 02020-09-23) (16 minutes)
• Globoflexia (p. 374) 02020-10-05 (updated 02020-10-10) (37 minutes)

Notes concerning “Carborundum”

- Machine teeth (p. 251) 02020-08-02 (updated 02020-12-31) (8 minutes)
- Hot fabrication (p. 320) 02020-09-21 (updated 02020-09-23) (16 minutes)

Notes concerning “Cameras”

• Globoflexia (p. 374) 02020-10-05 (updated 02020-10-10) (37 minutes)
• Specular photogrammetry (p. 570) 02020-11-11 (3 minutes)

Notes concerning “Bootstrapping”

- Minimal cost computer (p. 471) 02020-10-23 (updated 02020-12-01) (12 minutes)
- A field-programmable RTL array: a more efficient alternative to FPGAs? (p. 643) 02020-11-26 (updated 02020-11-27) (11 minutes)

Notes concerning “Bearings”

- Ballpoint SPIF (p. 36) 02020-04-25 (7 minutes)
- Lenticular air bearing (p. 636) 02020-11-24 (2 minutes)

Notes concerning “Batteries”

- Aluminum–air batteries (p. 326) 02020–09–23 (4 minutes)
- Ancient batteries (p. 340) 02020–09–23 (updated 02020–12–31) (4 minutes)

Notes concerning “Basic”

• One big text file (p. 97) 02020-06-04 (updated 02020-06-06) (20 minutes)
• Scribal Basic: a 1960s language for the 02020s (p. 705) 02020-12-12 (updated 02020-12-15) (32 minutes)

Notes concerning “B-trees”

• Line-numbered ISAM buffers (p. 224) 02020-07-18 (updated 02020-07-23) (14 minutes)
• Skip list variants (p. 423) 02020-10-12 (4 minutes)

Notes concerning “Automata theory”

- Monoid prefix sum (p. 105) 02020-06-05 (13 minutes)
- Wang tile chemicals (p. 357) 02020-09-30 (updated 02020-12-31) (2 minutes)

Notes concerning “Assembly language”

• An outline of the design process leading up to the Veskeno virtual machine (p. 126) 02020-06-17 (updated 02020-07-10) (88 minutes)
• Hardware queuing (p. 648) 02020-11-26 (updated 02020-12-16) (11 minutes)