```
PSTOPDF=ruby /usr/share/texmf/scripts/context/ruby/pstopdf.rb

all: intercalated.pdf explicacion.html sources.pdf 3.675mm.mask.pdf

frames=1-80x160.pgm 2-80x160.pgm 3-80x160.pgm 4-80x160.pgm
5-80x160.pgm 6-80x160.pgm 7-80x160.pgm 8-80x160.pgm 9-80x160.pgm
colorframes=1.ppm 2.ppm 3.ppm 4.ppm 5.ppm 6.ppm 7.ppm 8.ppm 9.ppm
sources=Makefile intercalate.py pscut.py pnm.py intercalate.py
simplify.py explicacion 3.675mm.mask.ps
sourcesps=$(foreach source,$(sources),$(source).source.ps)

%.pdf: %.ps
	$(PSTOPDF) $<

intercalated.pgm: $(frames)
	./intercalate.py $(frames) > $@

intercalated.ppm: $(colorframes)
	./intercalate.py $(colorframes) > $@

%-80x160.jpg: %.jpg
	convert -geometry 80x160 -colorspace Gray $< $@

%.pgm: %.jpg
	convert $< $@

%.ppm: %.jpg
	convert $< $@

# 300 mm × 450 mm, minus 3 mm on each side except the top
# (* (/ 72 25.4) 3) = 8.503937007874017
# (* (/ 72 25.4) 294) = 833.3858267716537
# (* (/ 72 25.4) 447) = 1267.0866141732286
%.ps: %.pgm pscut.py
	./pscut.py 833.4x1267.0+8.5+8.5 .25 .75 < $< > $@

explicacion.html: explicacion
	python -m markdown $< > $@

# paps only supports one input file
%.source.horked.ps: %
	paps --header $< > $@

# enscript doesn't support Unicode
#enscript -C -p- -2 -G -r $(sources) > $@

# paps output files are hairy enough that psjoin can't handle them
# properly, so we convert them to PDF and back.
%.source.ps: %.source.horked.pdf
	pdftops $< $@

sources.1up.ps: $(sourcesps)
	psjoin $(sourcesps) > $@
```



```
# pstopdf fucks up on this filename:
3.675mm.mask.ps.source.horked.pdf:
	paps --header 3.675mm.mask.ps >
3.675mm.mask.ps.source.horked.ps
	$(PSTOPDF) 3.675mm.mask.ps.source.horked.ps
	mv 3.675mm.mask.source.horked.pdf.pdf $@

# Override default rule invoking pstopdf because it doesn't work for
# some reason
sources.1up.pdf: sources.1up.ps
	ps2pdf $< $@

sources.ps: sources.1up.pdf
	pdftops $< - | psnup -4 > $@
```



```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import itertools
import sys

import pnm

def intercalate(sequences):
    return (item for t in itertools.izip(*sequences) for item in t)

def intercalate_files(outfile, infiles, read_file, write_file):
    w, h, pixstreams = None, None, []
    for infile in infiles:
        nw, nh, pixels = read_file(infile)
        if w is None:
            w, h = nw, nh
        else:
            assert (w, h) == (nw, nh), ((w, h), (nw, nh))
        pixstreams.append(pixels)

    write_file(outfile, w * len(pixstreams), h, intercalate
(pixstreams))

def intercalate_pnms(outfile, infiles_pnm):
    pnm_type = infiles_pnm[0].read(3)
    if pnm_type == 'P5\n':
        read_pnm, write_pnm = pnm.read_pgm, pnm.write_pgm
    elif pnm_type == 'P6\n':
        read_pnm, write_pnm = pnm.read_ppm, pnm.write_ppm
    else:
        raise UnknownMagicNumber(pnm_type)
    SEEK_CUR = 1
    infiles_pnm[0].seek(-3, SEEK_CUR)
    intercalate_files(outfile, infiles_pnm, read_pnm, write_pnm)

class UnknownMagicNumber(Exception): pass

def main():
    intercalate_pnms(sys.stdout, map(open, sys.argv[1:]))

if __name__ == '__main__':
    main()
```



```
#!/usr/bin/python
# -*- coding: utf-8 -*-
"""Generate a cutting path to approximate a grayscale image.

Invoke this program as follows:

    pscut.py WxH+X+Y minheight maxheight < image.pgm

with a pnmtools Portable Graymap as input and with numbers replacing
the parameters W, H, X, Y, minheight, and maxheight.  It generates a
PostScript output file on stdout which represents the image in
question as a cutting path for a laser cutter which will approximate
the image by varying the width of some horizontal slits, which are
placed in an area of dimensions W×H starting at the coordinates
(X, Y).

The parameters minheight and maxheight are fractional numbers
specifying the fraction of the vertical slit spacing that the slit is
allowed to take up.  If minheight is too low, your laser cutter may
set the material on fire as it runs the laser over the same place
twice; if it is too high, you lose contrast.  If maxheight is too
high, not only may you have a fire, but also there will be
interruptions of the solid material between the slits, so the final
piece will not hang together.  If maxheight is too low, you lose
brightness.

Normally the idea is that you're looking at light through the slits,
so the slits are wider for brighter pixels.  If instead you're looking
at a dark thing, use a maxheight less than minheight.

"""
import sys

from pnm import read_pgm
import simplify

# This prologue consists of bits pasted from laserboot/cut-7/
heckballs.ps
prologue = """%!PS-Adobe-2.0
%%BoundingBox: 0 0 {width} {height}
%%Pages: 1
%%BeginProlog
/inch 72 def
/mm_size inch 25.4 div def
/bdef { bind def } bind def
/inches { inch mul } bdef
/mm { mm_size mul } bdef

/cutting { 1 0 0 setrgbcolor } bdef

% Measurement by redefining the path-construction operators (the ones
% I'm using, anyway.)
```

```
/orig-rlineto /rlineto load def
/orig-rmoveto /rmoveto load def
/orig-lineto /lineto load def
/orig-moveto /moveto load def
% Calculate distance from x1 y1 to x2 y2
/dist { 3 2 roll sub  3 1 roll sub  dup mul exch  dup mul add  sqrt }
bdef
/total-points 0 def             % Total number of points drawn
/total-distance 0 def           % Total distance drawn
/increment-points { /total-points total-points 1 add def } bdef
/mark-point { currentpoint /lasty exch def /lastx exch def } bdef
/measure {
    /total-distance  lastx lasty currentpoint dist  total-distance
add  def
} bdef
/moveto { orig-moveto increment-points } bdef
/rmoveto { orig-rmoveto increment-points } bdef
/lineto { mark-point  orig-lineto  measure increment-points } bdef
/rlineto { mark-point orig-rlineto measure increment-points } bdef
/measurement-report {
    (Totals: ) print
    total-points (      ) cvs print
    ( vertices; ) print
    total-distance 1 mm div (       ) cvs print
    ( mm drawn.\nEstimated cutting time ) print
    total-points .06 mul total-distance 24 mm div add 5 add dup
    (       ) cvs print
    ( seconds, estimated Max58 cost AR$) print
    0.40 mul (       ) cvs print
    (.\n) print
} bdef

%%EndProlog
%%Page: 1 1
save

.001 setlinewidth
cutting
"""

epilogue = """
measurement-report
restore
showpage
%%EOF
"""

def pscut(geometry, minheight, maxheight, infile_pgm, output_canvas):
    w, h, pixels = read_pgm(infile_pgm)
    scale = (maxheight - minheight) / 255
    transform = geometry.containing((w, h))
    minseen, maxseen = None, None
```



```
    for row in range(h):
        x, y = transform.transform((0, row))
        output_canvas.moveto((x, y))
        for col in range(w):
            pix = pixels.next()
            height = minheight + pix * scale
            # This is unnecessarily inefficient, but that probably
            # doesn't matter ;)
            coordinates = transform.transform((col, row + height))
            output_canvas.lineto(coordinates)
            if minseen is None:
                minseen = maxseen = coordinates[1] - y
            else:
                minseen = min(coordinates[1] - y, minseen)
                maxseen = max(coordinates[1] - y, maxseen)

        output_canvas.lineto(transform.transform((w-1, row)))
        output_canvas.closepath()

    debug("Heights ranged from %s to %s points." % (minseen, maxseen))
    debug("(%s-%s mm)" % (minseen * 25.4 / 72, maxseen * 25.4 / 72))

def debug(s):
    sys.stderr.write(s + "\n")

def fill_pscut(geometry, minheight, maxheight, infile_pgm,
output_canvas):
    #output_canvas.newpath()
    pscut(geometry, minheight, maxheight, infile_pgm, output_canvas)
    output_canvas.stroke()

class PsCanvas:
    def __init__(self, stream, precision=2):
        self.stream = stream
        self.precision = precision # Number of decimal digits to round
to

    def newpath(self):
        self.say("newpath")

    def say(self, s):
        self.stream.write(s)
        self.stream.write('\n')

    def lineto(self, (x, y)):
        self.say('%s %s lineto' % (self.coord(x), self.coord(y)))

    def moveto(self, (x, y)):
        self.say('%s %s moveto' % (self.coord(x), self.coord(y)))

    def coord(self, n):
        return '%s' % round(n, self.precision)
```



```
    def closepath(self):
        self.say("closepath")

    def fill(self):
        self.say("fill")

    def stroke(self):
        self.say("stroke")

    def eofill(self):
        self.say("eofill")

class SimplifyingCanvasDecorator:
    def __init__(self, canvas, precision=0.1): # about 35 microns
        self.canvas = canvas
        self.precision = precision # in PostScript points
        self.current_poly = None

    def newpath(self):
        self.current_poly = None
        self.canvas.newpath()

    def lineto(self, p):
        self.current_poly.append(p)

    def moveto(self, p):
        self.flush_current_poly()
        self.current_poly = [p]

    def closepath(self):
        self.current_poly.append(self.current_poly[0])

    def fill(self):
        self.flush_current_poly()
        self.canvas.fill()

    def stroke(self):
        self.flush_current_poly()
        self.canvas.stroke()

    def eofill(self):
        self.flush_current_poly()
        self.canvas.eofill()

    def flush_current_poly(self):
        if self.current_poly is None:
            return

        poly = simplify.simplify(self.current_poly, self.precision)
        self.canvas.moveto(poly[0])
        for p in poly[1:]:
            # XXX closepath won't have the right linejoin
            self.canvas.lineto(p)
```



```
        self.current_poly = None

class Geometry:
    def __init__(self, spec):
        wh, x, y = spec.split('+')
        self.x, self.y = map(float, (x, y))
        self.w, self.h = map(float, wh.split('x'))

    def containing(self, (ncols, nrows)):
        row_height = self.h / nrows
        col_width = self.w / ncols
        return HomogeneousLinearTransform((self.x, self.y + self.h),
                                          (col_width, -row_height))

class HomogeneousLinearTransform:
    def __init__(self, (x, y), (dx, dy)):
        self.x, self.y, self.dx, self.dy = x, y, dx, dy

    def transform(self, (x, y)):
        return (self.x + self.dx * x, self.y + self.dy * y)

def main():
    geometry, minheight, maxheight = sys.argv[1:]
    geometry = Geometry(geometry)
    # XXX this geometry calculation is super shitty; it should be
    # 300×600 mm but instead it's a totally different thing
    print prologue.replace("{width}", str(int(round(geometry.w +
2*geometry.x)))).replace("{height}", str(int(round(geometry.h +
geometry.y))))
    fill_pscut(geometry,
               float(minheight),
               float(maxheight),
               sys.stdin,
               SimplifyingCanvasDecorator(PsCanvas(sys.stdout),
precision=0.5))
    print epilogue

if __name__ == '__main__':
    main()
```

```
import itertools

def read_pgm(infile_pgm):
    assert infile_pgm.readline() == 'P5\n'
    w, h = map(int, infile_pgm.readline().split())
    assert infile_pgm.readline() == '255\n'

    def pixels():
        for i, c in enumerate(iter(lambda: infile_pgm.read(1), '')):
            yield ord(c)
        assert i == w * h - 1, (i, w * h)

    return w, h, pixels()

def write_pgm(outfile, w, h, pixels):
    outfile.write('P5\n%d %d\n255\n' % (w, h))
    outfile.writelines(itertools.imap(chr, pixels))

def read_ppm(infile_ppm):
    assert infile_ppm.readline() == 'P6\n'
    w, h = map(int, infile_ppm.readline().split())
    assert infile_ppm.readline() == '255\n'

    def pixels():
        for i, ccc in enumerate(iter(lambda: infile_ppm.read(3), '')):
            yield tuple(map(ord, ccc))
        assert i == w * h - 1, (i, w * h)

    return w, h, pixels()

def write_ppm(outfile, w, h, pixels):
    outfile.write('P6\n%d %d\n255\n' % (w, h))
    outfile.writelines(chr(c) for pix in pixels for c in pix)
```



```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import itertools
import sys

import pnm

def intercalate(sequences):
    return (item for t in itertools.izip(*sequences) for item in t)

def intercalate_files(outfile, infiles, read_file, write_file):
    w, h, pixstreams = None, None, []
    for infile in infiles:
        nw, nh, pixels = read_file(infile)
        if w is None:
            w, h = nw, nh
        else:
            assert (w, h) == (nw, nh), ((w, h), (nw, nh))
        pixstreams.append(pixels)

    write_file(outfile, w * len(pixstreams), h, intercalate
(pixstreams))

def intercalate_pnms(outfile, infiles_pnm):
    pnm_type = infiles_pnm[0].read(3)
    if pnm_type == 'P5\n':
        read_pnm, write_pnm = pnm.read_pgm, pnm.write_pgm
    elif pnm_type == 'P6\n':
        read_pnm, write_pnm = pnm.read_ppm, pnm.write_ppm
    else:
        raise UnknownMagicNumber(pnm_type)
    SEEK_CUR = 1
    infiles_pnm[0].seek(-3, SEEK_CUR)
    intercalate_files(outfile, infiles_pnm, read_pnm, write_pnm)

class UnknownMagicNumber(Exception): pass

def main():
    intercalate_pnms(sys.stdout, map(open, sys.argv[1:]))

if __name__ == '__main__':
    main()
```



```
# -*- coding: utf-8 -*-
"""Ramer–Douglas–Peucker polyline path simplification algorithm.

The idea is that if you have a polygonal path to approximate with
lines to a given precision, you can do it by building up the
approximation from a subset of the original points by recursive
subdivision.  The recursion terminates when the precision is
satisfied; otherwise the path is split into two subpaths at the point
that is furthest from the possibly degenerate line segment between the
path endpoints.

This is intended to save cutting time on the laser cutter.

"""
import numpy

def test():
    test_lsdist()
    test_simplify()

def ok(a, b):
    assert a==b, (a, b)

def test_lsdist():
    ok(list(lsdist([0, 1], [0, 1],
                   [0, 1, 0,      1,      1, 4, -3,],
                   [0, 1, 1,      0,      9, 5, -4,])),
                   [0, 0, .5**.5, .5**.5, 8, 5,  5])
    # Degenerate segment test.
    ok(list(lsdist([1, 1], [0, 0],
                   [0, 1, 0,      1,      1, 4,       -3,],
                   [0, 1, 1,      0,      9, 5,       -4,])),
                   [1, 1, 2**.5,  0,      9, 34**.5,  4*2**.5])

def lsdist(ex, ey, xs, ys):
    """Returns the distances of the points at coordinates in xs and ys
from seg.

    ex and ey are the endpoint x and y arrays, which should have
length 2.
    """
    return _lsdist(numpy.asarray(ex),
                   numpy.asarray(ey),
                   numpy.asarray(xs),
                   numpy.asarray(ys))

def _lsdist(ex, ey, xs, ys):
    assert len(ex) == len(ey) == 2
    assert len(xs) == len(ys)
    # Calculate min distance from endpoints.  This will be used for
    # the points where the endpoint is the closest part of the line
    # segment.
    delta = numpy.subtract.outer
```



```
    dx2 = delta(ex, xs)**2
    dy2 = delta(ey, ys)**2
    r2 = dx2 + dy2
    minr = r2.min(axis=0)**0.5

    # Project points onto line segment to see which ones are closer to
    # some point in the middle.
    sx, sy = ex[0], ey[0]           # start x, y
    vx, vy = ex[1] - sx, ey[1] - sy # vector x and y

    # Without this, the degenerate segment case gets zero divisions
    # and incorrectly returns NaNs.
    if vx == vy == 0:
        return minr

    dx, dy = (xs - sx), (ys - sy) # Δx, Δy from the segment start
    dot = vx * dx + vy * dy       # Dot products

    # The magnitude of the projection onto the segment is the delta
    # vector magnitude times the cosine of its angle.  The dot product
    # has an additional factor of the segment length in it.
    vmag2 = vx*vx + vy*vy
    within = (0 <= dot) & (dot <= vmag2)

    # Now we would like to calculate the perpendicular distance for
    # the points within the segment.  We've already nearly calculated
    # the projected positions, and we could just subtract those, but
    # then we have to calculate the lengths of the vectors, which
    # involves taking N Pythagorean sums (and 2N multiplications and
    # subtractions).  It's simpler and faster to just compute a sort
    # of cross product, a dot product with a 90° rotated vector.  We
    # only need this for the `within` points, but it's simpler and
    # probably almost as fast to just calculate it for all points.
    perp = numpy.abs(dx * vy - dy * vx) * vmag2**-.5

    return numpy.where(within, perp, minr)

def test_simplify():
    sq = [(0, 0), (0, 1), (1, 1), (1, 0), (0, 0)]
    ok(simplify(sq, 0), sq)
    ok(simplify(sq, 2), [(0, 0), (0, 0)])
    asq = [(0,0), (0,0.5), (0,1), (1,1), (1,0), (0,0)]
    ok(simplify(asq, 0), sq)
    ok(simplify(asq, 0.1), sq)
    osq = [(0,0), (0.1,0.5), (0,1), (1,1), (1,0), (0,0)]
    ok(simplify(osq, 0), osq)
    ok(simplify(osq, 0.1), sq)
    ok(simplify(osq, 0.2), sq)
    ok(simplify(osq, 2), simplify(sq, 2))

def simplify(path, tolerance):
    xs, ys = zip(*path)
    return zip(*_simplify(numpy.array(xs), numpy.array(ys), tolerance))
```

```
def _simplify(xs, ys, tolerance):
    n = len(xs)
    if n <= 2:
        return xs, ys

    ex = xs[0::n-1]
    ey = ys[0::n-1]
    dists = _lsdist(ex, ey, xs, ys)
    i = dists.argmax() # index of point farthest from line between
endpoints
    if dists[i] <= tolerance:
        return ex, ey

    # The intervals being recursively simplified overlap by one
    # vertex, which they are guaranteed to include in their return
    # values.
    xs0, ys0 = _simplify(xs[:i+1], ys[:i+1], tolerance)
    xs1, ys1 = _simplify(xs[i:],   ys[i:],   tolerance)

    return (numpy.concatenate([xs0[:-1], xs1]),
            numpy.concatenate([ys0[:-1], ys1]))


test()
```


Todo el código fuente
del proyecto
está impreso como parte del proyecto,
de la misma forma
que cada célula de la flor
y cada célula de tu cuerpo
contiene una copia entera del código genético del organismo entero.
En el código
se pueden detectar
varias cosas que intenté sumarle
pero que no funcionaron,
de la misma forma que nuestro propio genoma
contiene varias copias
rotas
de virus que infectaron a nuestros parientes,
cosa que supongo que se aplica también a nuestras primas las plantas.

Las flores se encuentran boca abajo
como mi padrastro
cuando lo encontraron
horas después de su infarto.
El motor de su auto todavía corría.
Era fotógrafo.
Hace poco había sacado las fotos de mi boda,
como su regalo a nosotres.
Fotos con flores.

```
<style>
body { text-align: right }
</style>
```



```
<title>explicación de las flores de fibrofácil</title>

<meta charset="utf-8" />
```

Cada línea
intercala los aspectos temporales y espaciales
de las flores.
Si seguimos
una línea
horizontalmente,
atravesamos nueve píxeles correspondientes
separados en el tiempo
antes de llegar al píxel siguiente
espacialmente.
Por eso los patrones parecen repetirse,
como nuestras historias de vida,
porque siguen la historia de un píxel tras otro
por el tiempo,
y los píxeles cercanos
tienen historias cercanas,
como personas en situaciones cercanas
suelen tener historias cercanas.

El material de la imágen
se doblará bajo su propio peso,
doblándose hacia el piso,
poco a poco durante los días de la muestra,
porque las fibras vegetales dentro de sí mismo
soltarán poco a poco los lazos moleculares
que las juntan entre sí,
exactamente como pasó a las flores de las fotos,
exactamente como tus propias fibras sueltan sus lazos moleculares
poco a poco,
mientras vos marchás inexorablemente hacia tu ataúd.

El material impermanente de la obra
hace referencia a las obras de Burning Man,
diseñadas para ser quemadas al fin de una semana,
hechas de madera o este mismo
tablero de fibra de densidad media.
En Burning Man cuestionamos la permanencia del arte,
que típicamente se ocupa de usar materiales
aptos para archival.
Mientras tanto,
el tablero de fibra de densidad media,
especialmente cortado a láser,
liberando el ácido piroleñoso,
contiene su propio destino
de gradualmente oxidar la celulosa,
quemándose sin calor
hasta no quedar nada más que polvo
en pocos años.


```
%!PS-Adobe-2.0
%%BoundingBox: 0 0 595 842
%%Pages: 1
%%BeginProlog

/inch 72 def
/mm_size inch 25.4 div def
/bdef { bind def } bind def
/inches { inch mul } bdef
/mm { mm_size mul } bdef

/box { 2 copy  0 exch rlineto  0 rlineto  neg 0 exch rlineto
closepath  pop } bdef

/spacing 3.675 mm def
/slitwidth 1 300 div inches def
/rectwidth spacing slitwidth sub def

% A4 paper:
/height 297 mm def
/width 210 mm def

%%EndProlog
%%Page: 1 1

save
0 0 moveto
width height box clip

newpath
0 0 moveto

width spacing div cvi 1 add {
    currentpoint
    rectwidth height box
    moveto
    spacing 0 rmoveto
} repeat

fill

restore
showpage
%%EOF
```