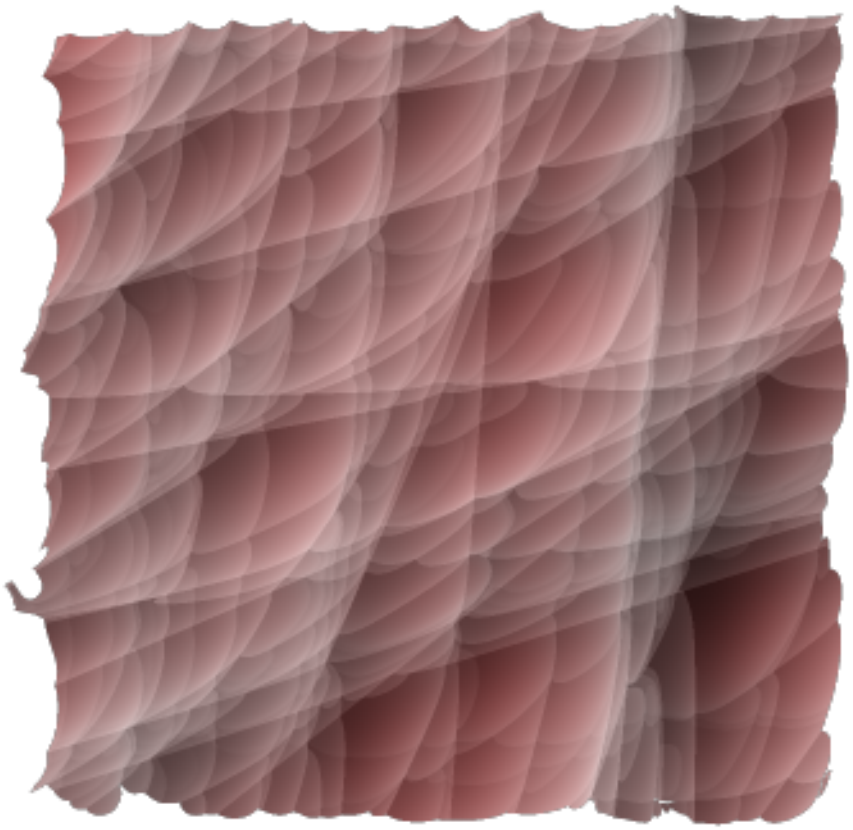
Andrew Cooke

Red Square 1



Introduction

Red Square 1 is an example of what is possible using the Pancito package. I haven't tried to explain why or how the image looks as it does, but I hope I do give enough information for you to see how to make images of your own.

This Document

You can generate the image by typing:

```
ghc -c Pancito
ghc Pancito.o RedSquare1.lhs -o RedSquare1
./RedSquare1
```

where ghc is the Glasgow Haskell Compiler, RedSquare1.lhs is "this" file (actually, you are probably reading a file called RedSquare1.ps or RedSquare1.pdf which are also generated from "this" file using LaTeX, dvips, and ps2pdf, but as I type this I am editing the file RedSquare1.lhs).

That will generate a file called redSquare1.ppm which contains the image. It can then be viewed using xloadimage (for example) or converted to another format using the netpbm package.

Haskell

Red Square 1 is described using the Haskell language and uses the Pancito module and a number of other standard libraries. These are imported here:

```
module Main where

import IO
import Monad
import Random
import Pancito
```

Support

Red Square

The image starts as a simple square on a transparent background.

```
redSquare :: Image
redSquare = windowedImage square11 red transparent
```

Crease Frame

When adding a "crease" to the image, calculations are smiplified by working in a coordinate frame that has one axis lying along the crease with the other perpendicular.

Here xc and yc are coordinates of a point on the crease and theta is the angle the crease makes with the vertical. In the result, the y coordinate lies along the crease.

```
toCreaseFrame :: Point → Double → Point → Point
toCreaseFrame (xc, yc) theta p =
    fromPolar (r, t − theta)
  where (r, t) = toPolar $ shift (−xc, −yc) p

fromCreaseFrame :: Point → Double → Point → Point
fromCreaseFrame c theta p =
    shift c $ fromPolar (r, t + theta)
  where (r, t) = toPolar p
```

Crease To add a crease we add shading and shift the points appropriately.

```
extend :: Double → Double → Double
extend s d = d + s * 0.05 * (exp (-30.0 * (abs d)))

ifVisible :: (Colour → Colour) → Colour → Colour
ifVisible f c = if similar 0.0001 c transparent
                  then transparent
                  else f c

foldFade :: Double → Double → Colour → Colour
foldFade s d =
    if d > 0
      then ifVisible $ darken dd
      else ifVisible $ lighten dd
  where
    dd = s * 0.2 / ((1.0 + (abs d))**7.0)

darkLight :: (Point, Double, Double) → Filter
darkLight (c, theta, s) im p =
    foldFade s x $ im $ fromCreaseFrame c theta (x', y)
  where
    x' = extend s x
    (x, y) = toCreaseFrame c theta p
```

Multiple Creases The creases are generated at random positions and angles.

```
buildParams :: Int → [(Point, Double, Double)]
buildParams n =
    take n (zip3 (zip (randomRs (-1.5, 1.5) gen1)
                      (randomRs (-1.5, 1.5) gen2))
                 (randomRs (-pi/2.0, 0.0) gen3)
                 (map fade [0..n-1]))
  where
    nn = fromIntegral n
    fade x = 0.5 * (1 + sin (pi * fromIntegral x / nn))
    gen' = mkStdGen 1
    (gen1, gen2') = split gen'
    (gen2, gen3) = split gen2'

manyCreases :: Int → Image
manyCreases n =
    foldr darkLight redSquare (buildParams n)
```

Image This defines the command that generates the image when the complied code is executed.

```
frame :: Window
frame = ((-1.3, -1.3), (1.3, 1.3))

main :: IO ()
main = writePpmAlias 3 "redSquare1.ppm" 8 white
         (200, 200) frame (manyCreases 100)
```

Credits Once again, thanks to Conal Elliot for Pan and all the people on the Haskell mailing list and c.l.functional for replying to questions.