


```

expand :: Int → Transform
expand n = scale (n', n')
  where n' = fromIntegral n

grid :: Image
grid = cross11 black white 0.1 ∘ pixelate11

cAsTransform :: Double → Transform
cAsTransform k (x, y) = (x + k * dx, y + k * dy)
  where
    (dx, dy) = c (x, y)

c1 :: Image
c1 = grid ∘ expand 5 ∘ cAsTransform 0.05

makeC1 :: IO ()
makeC1 = writePpmAlias 3 "c1.ppm" 8 white (200, 200) square11 c1

```

Chance (C2) This image is based on the results of incorrect code while working on Grid (C1).

```

almostC :: Point → (Double, Double)
almostC (x, y) = (-1 * sin(y') * cos(x' * 2),
                 -1 * sin(x') * cos(y' * 2))
  where
    x' = pi * x
    y' = pi * y

almostCAsTransform :: Transform
almostCAsTransform (x, y) = (x + 0.5 * dx, y + 0.5 * dy)
  where
    (dx, dy) = almostC (x, y)

c2 :: Image
c2 = grid ∘ expand 5 ∘ almostCAsTransform

makeC2 :: IO ()
makeC2 = writePpmAlias 3 "c2.ppm" 8 white (200, 200) square11 c2

```

Dots (C3) I used to tutor physics in a room with a couple of Bridget Riley prints on the wall showing regular patterns of dots.

```

dot11 :: Colour → Colour → Double → Image
dot11 fg bg r p = if dist p < r then fg else bg

dots :: Colour → Colour → Double → Image
dots fg bg r = dot11 fg bg r ∘ pixelate11

dotsBW :: Double → Image
dotsBW = dots black white

c3 :: Image
c3 = dotsBW 0.3 ∘ expand 5 ∘ cAsTransform 0.05

square11' :: Window
square11' = ((-1.1, -1.1), (1.1, 1.1))

```

```
makeC3 :: IO ()
makeC3 = writePpmAlias 3 "c3.ppm" 8 white (200, 200) square11' c3
```

Dots (C4) Change in size emphasises local distortion.

```
c4 :: Image
c4 = dotsBW 0.6 ◦ expand 5 ◦ cAsTransform 0.05

makeC4 :: IO ()
makeC4 = writePpmAlias 3 "c4.ppm" 8 white (200, 200) square11' c4
```

Dots (C5) Remove the translation, but preserve the local deformation.

This is a significant increase in algorithmic complexity. Until now, the C transform has been applied to the main coordinate system only. That has then been pixel-mapped (for want of a better expression — I mean that the region around each point (n,m) in the image where n and m are integers is mapped to the same sub-image) to a grid or dots. Now, the C transform is used to modify the shape within the sub-images. Global image information is "leaking" into lower levels.

```
localExpansion :: Int → Transform → Transform
localExpansion n t (x, y) = shift (dx, dy) $ e $ t (x, y)
  where
    n' = fromIntegral n
    e = scale (n', n')
    e' = scale (1.0 / n', 1.0 / n')
    (ix, iy) = dRoundP $ e (x, y)
    (ix', iy') = e $ t $ e' (ix, iy)
    (dx, dy) = (ix - ix', iy - iy')

c5 :: Image
c5 = dotsBW 0.6 ◦ localExpansion 5 (cAsTransform 0.05)

makeC5 :: IO ()
makeC5 = writePpmAlias 3 "c5.ppm" 8 white (200, 200) square11' c5
```

Dots (C6) Add shading based on the transform. We could pass more global information down to the lowest levels or add shading at a higher level. Because the image is only two colours (before aliasing) the high level approach is probably OK even for more complex work — we can use the two colours to distinguish between ground and motif.

For this image, however, global lightening is fine (on a white background, lightening will only affect the foreground).

```
type Mask = Point → Double

lightenM :: Mask → Filter
lightenM s im p = lighten (s p) $ im p

darkenM :: Mask → Filter
darkenM s im p = darken (s p) $ im p

cAsMask :: Double → Double → Mask
```

```

cAsMask z dz p = z + dz * (dist $ c p)

c6 :: Image
c6 = lightenM (cAsMask 0.1 0.3) c5

makeC6 :: IO ()
makeC6 = writePpmAlias 3 "c6.ppm" 8 white (200, 200) square11' c6

```

Dots (C7) Combining shading with Dots (C4), but with the two patterns at right angles.

```

rot90, rot180, rot270 :: Transform
rot90 (x, y) = (y, -x)
rot180 (x, y) = (-x, -y)
rot270 (x, y) = (-y, x)

c7 :: Image
c7 = lightenM (cAsMask 0.1 0.3 o rot90) c4

makeC7 :: IO ()
makeC7 = writePpmAlias 3 "c7.ppm" 8 white (200, 200) square11' c7

```

Crosses (C8) Show the distortion directly.

```

clippedCross11 :: Colour → Colour → Double → Double → Image
clippedCross11 fg bg r w p =
  if dist p < r then cross11 fg bg w p else bg

crosses :: Image
crosses = clippedCross11 black white 0.45 0.15 o pixelate11

c8 :: Image
c8 = crosses o localExpansion 5 (cAsTransform 0.2)

makeC8 :: IO ()
makeC8 = writePpmAlias 5 "c8.ppm" 8 white (200, 200) square11' c8

```

Dots (C9) Overlapping.

```

c9x :: Colour → Image
c9x c =
  dots (opacity 0.5 c) transparent 0.6 o expand 5 o cAsTransform 0.15

c9, c91, c92, c93 :: Image
c91 = c9x black
c92 = c91 o rot90
c93 = c92 o rot90
c9 p = solidify (c91 p + c92 p + c93 p)

solidify :: Colour → Colour
solidify c = if similar 0.01 c transparent then transparent else c * 1.15

makeC9 :: IO ()
makeC9 = writePpmAlias 3 "c9.ppm" 8 white (200, 200) square11' c9

```

Chance (C10) Thanks to a dynamic colour look-up table.

```
greyN :: Double → Colour
greyN n = solid n n n

paint :: Colour → Colour
paint c | similar 0.01 c' white = solid 0.7 0.0 0.2
        | similar 0.01 c' black = solid 0.0 0.0 0.6
        | similar 0.01 c' one   = solid 0.0 0.9 0.0
        | similar 0.01 c' two   = solid 0.7 0.9 0.3
        | otherwise             = white
  where
    c' = opacity 1 (white + c)
    one = greyN 0.137
    two = greyN 0.424

c10 :: Image
c10 = paint ∘ c9

makeC10 :: IO ()
makeC10 = writePpmAlias 3 "c10.ppm" 8 white (200, 200) square11' c10
```

Crosses (C11) Duplicate and window (with background of future images in mind, but also as an image in its own right).

```
crosses' :: Image
crosses' = clippedCross11 black transparent 0.6 0.2 ∘ pixelate11

box :: Window
box = ((-0.32, 0.53), (-0.08, 0.77))

frame :: Window → Double → Filter
frame w f im p = if contains w p then c else darken f c
  where c = white + im p

c11' :: Image → Image
c11' im p = c111 p + c112 p
  where
    c111 = im ∘ expand 5 ∘ cAsTransform 0.2
    c112 = im ∘ shift (0.5, 0.5) ∘ expand 5 ∘ cAsTransform 0.2 ∘ rot90

c11 :: Image
c11 = frame box 0.5 (c11' crosses')

makeC11 :: IO ()
makeC11 = writePpmAlias 3 "c11.ppm" 8 white (200, 200) square11' c11
```

Crosses (C12) Colour the window.

```
crosses'' :: Image
crosses'' = clippedCross11 ltGrn transparent 0.6 0.2 ∘ pixelate11
  where ltGrn = (solid 0.2 0.9 0.2)

flipX :: Double → Double
flipX z = r - dz
  where
    r = fromIntegral (round z)
```

```

dz = z - r

xNoise :: Transform
xNoise (x, y) = (flipX x, flipX y)

xNoise' :: Int → Transform
xNoise' n = scale (1/n', 1/n') ∘ xNoise ∘ scale (n', n')
  where n' = fromIntegral n

filterBg :: Filter → Colour → Filter
filterBg f c im p = f (colourImage c) p + im p

c12 :: Image
c12 = filterBg fbg bg $ ffg ∘ xNoise' 15 ∘ xNoise' 73
  where
    bg = solid 0.7 0.9 0.3
    fbg = lightenM (cAsMask 0 0.2 ∘ rot180 ∘ xNoise' 5 ∘ xNoise' 13)
    fg = (c11' crosses'' ∘ mapWindow box square11)
    ffg = lightenM (cAsMask 0 0.3 ∘ rot180) fg

makeC12 :: IO ()
makeC12 = writePpmAlias 3 "c12.ppm" 8 white (200, 200) square11' c12

```

Dots (C13) Overlay orange dots.

```

orange :: Colour
orange = solid 0.8 0.4 0

dotsOrange :: Image
dotsOrange = dots orange transparent 0.6 ∘ t
  where t = localExpansion 5 (cAsTransform 0.05)

box' :: Window
box' = ((-1.1, 0.5), (-0.5, 1.1))

c13, c13f :: Image
c13f = lightenM (cAsMask 0.0 0.3) dotsOrange
c13 = \p → c12 p + (c13f $ mapWindow box' square11 p)

makeC13 :: IO ()
makeC13 = writePpmAlias 3 "c13.ppm" 8 white (200, 200) square11' c13

```

Dots (C14) Combine the previous image with the approach in Chance (C2).

```

c14 :: Image
c14 = c13 ∘ cAsTransform 2

makeC14 :: IO ()
makeC14 = writePpmAlias 3 "c14.ppm" 8 white (200, 200) square11 c14

```

Dots (C15) And again. And again. And again.

```

c15 :: Image
c15 p = c13 (head $ drop 10 $ iterate (rot90 ∘ cAsTransform 2) p)

makeC15 :: IO ()
makeC15 = writePpmAlias 3 "c15.ppm" 8 white (200, 200) square11 c15

```

Dots (C16) The limit(?) may be clearer in black and white.

```
c16 :: Image
c16 p = dotsBW 0.6 ◦ expand 5 $
      (head $ drop 20 $ iterate (rot90 ◦ cAsTransform 2) p)

makeC16 :: IO ()
makeC16 = writePpmAlias 3 "c16.ppm" 8 white (200, 200) square11 c16
```

Index Index images.

```
main :: IO ()
main = do writePpmAlias 3 "c1i.ppm" 8 white (100, 100) square11 c1 ;
          writePpmAlias 3 "c2i.ppm" 8 white (100, 100) square11 c2 ;
          writePpmAlias 3 "c3i.ppm" 8 white (100, 100) square11' c3 ;
          writePpmAlias 3 "c4i.ppm" 8 white (100, 100) square11' c4 ;
          writePpmAlias 3 "c5i.ppm" 8 white (100, 100) square11' c5 ;
          writePpmAlias 3 "c6i.ppm" 8 white (100, 100) square11' c6 ;
          writePpmAlias 3 "c7i.ppm" 8 white (100, 100) square11' c7 ;
          writePpmAlias 5 "c8i.ppm" 8 white (100, 100) square11' c8 ;
          writePpmAlias 3 "c9i.ppm" 8 white (100, 100) square11' c9 ;
          writePpmAlias 3 "c10i.ppm" 8 white (100, 100) square11' c10 ;
          writePpmAlias 3 "c11i.ppm" 8 white (100, 100) square11' c11 ;
          writePpmAlias 3 "c12i.ppm" 8 white (100, 100) square11' c12 ;
          writePpmAlias 3 "c13i.ppm" 8 white (100, 100) square11' c13 ;
          writePpmAlias 3 "c14i.ppm" 8 white (100, 100) square11 c14 ;
          writePpmAlias 3 "c15i.ppm" 8 white (100, 100) square11 c15 ;
          writePpmAlias 3 "c16i.ppm" 8 white (100, 100) square11 c16
```

Licencing Conditions This document and code are released under the GPL (see www.gnu.org for full details).

Copyright 2001 Andrew Cooke (Jara Software).