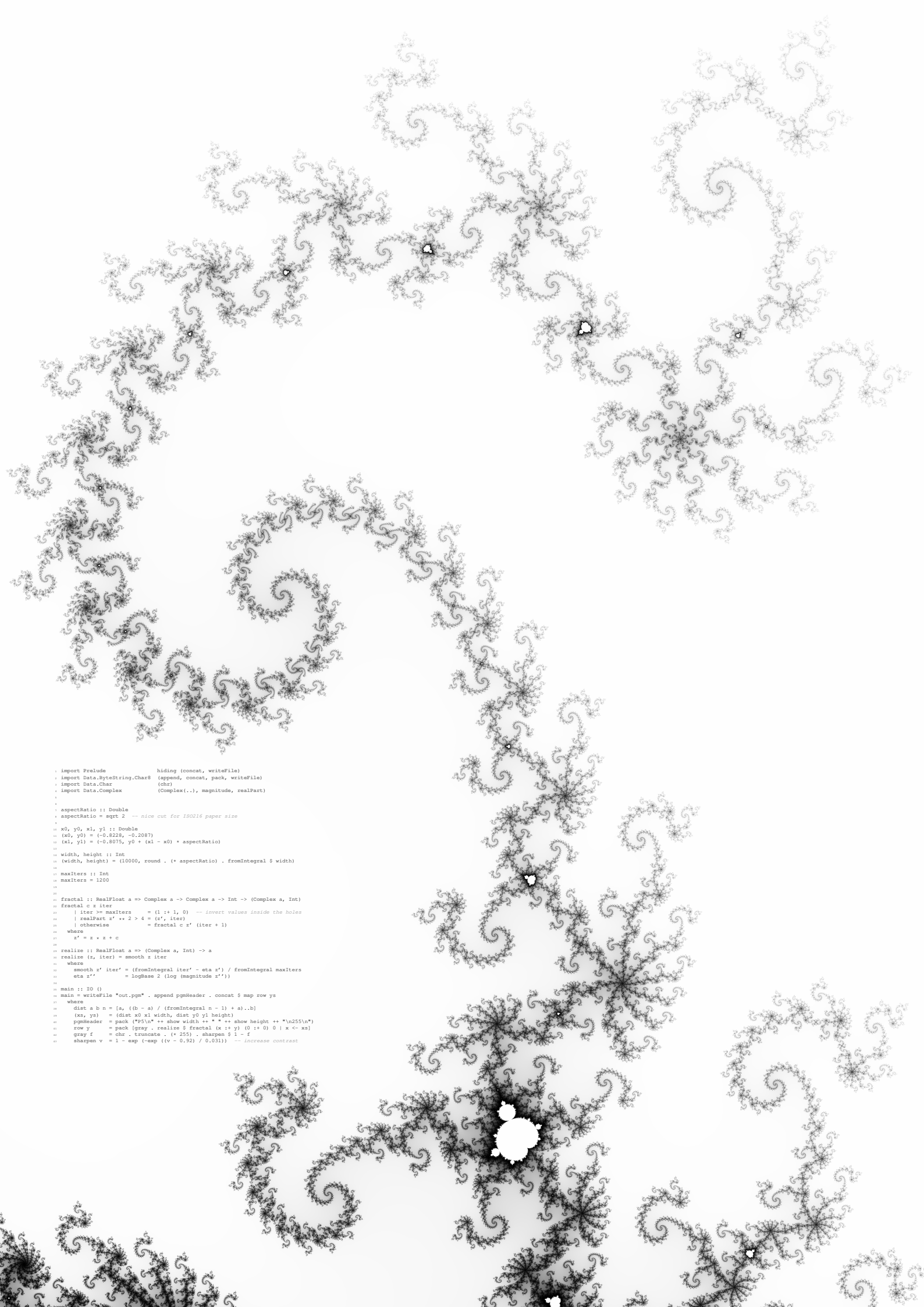

```haskell
import Prelude              hiding (concat, writeFile)
import Data.ByteString.Char8  (append, concat, pack, writeFile)
import Data.Char              (chr)
import Data.Complex           (Complex(..), magnitude, realPart)


aspectRatio :: Double
aspectRatio = sqrt 2  -- nice cut for ISO216 paper size

x0, y0, x1, y1 :: Double
(x0, y0) = (-0.8228, -0.2087)
(x1, y1) = (-0.8075, y0 + (x1 - x0) * aspectRatio)

width, height :: Int
(width, height) = (10000, round . (* aspectRatio) . fromIntegral $ width)

maxIters :: Int
maxIters = 1200


fractal :: RealFloat a => Complex a -> Complex a -> Int -> (Complex a, Int)
fractal c z iter
    | iter >= maxIters     = (1 :+ 1, 0)  -- invert values inside the holes
    | realPart z' ** 2 > 4 = (z', iter)
    | otherwise            = fractal c z' (iter + 1)
  where
    z' = z * z + c

realize :: RealFloat a => (Complex a, Int) -> a
realize (z, iter) = smooth z iter
  where
    smooth z' iter' = (fromIntegral iter' - eta z') / fromIntegral maxIters
    eta z''         = logBase 2 (log (magnitude z''))

main :: IO ()
main = writeFile "out.pgm" . append pgmHeader . concat $ map row ys
  where
    dist a b n = [a, ((b - a) / (fromIntegral n - 1) + a)..b]
    (xs, ys)   = (dist x0 x1 width, dist y0 y1 height)
    pgmHeader  = pack ("P5\n" ++ show width ++ " " ++ show height ++ "\n255\n")
    row y      = pack [gray . realize $ fractal (x :+ y) (0 :+ 0) 0 | x <- xs]
    gray f     = chr . truncate . (* 255) . sharpen $ 1 - f
    sharpen v  = 1 - exp (-exp ((v - 0.92) / 0.031))  -- increase contrast
```