# gmndl

Claude Heiland-Allen

2010–2017

# Contents



## 1 Address.hs

```
{-

    gmndl -- Mandelbrot Set explorer
    Copyright (C) 2010,2011,2014  Claude Heiland-Allen <claude@mathr.co.uk>

    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 program; if not, write to the Free Software Foundation, Inc.,
    51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.

-}

module Address(Address(..), angledInternalAddress, externalAngles, rayEnd,
    ↪ parameter, parse, pretty) where

import Prelude hiding (isNaN)

import Control.Monad (guard)
import Control.Monad.Identity (Identity())
import Data.Char (digitToInt)
```

```
import Data.List (genericDrop, genericLength, genericTake, unfoldr)
import Data.Maybe (listToMaybe)
import Data.Ratio ((%), numerator, denominator)
import Data.Vec (NearZero())
import Text.Parsec (ParsecT(), choice, digit, eof, many, many1, sepBy, string, try)
import Text.Parsec.Prim (runP)

import Complex (Complex((:+)), mkPolar, Turbo)
import MuAtom (refineNucleus)

isNaN x = not (x == x)

double :: Rational -> Rational
double angle = wrap (2 * angle)

wrap :: Rational -> Rational
wrap angle
  | frac < 0 = frac + 1
  | otherwise = frac
  where
    _i :: Integer
    (_i, frac) = properFraction angle

data Knead = Zero | One | Star
  deriving (Eq, Show)

knead :: Rational -> [Knead]
knead angle
  | angle == 0 || angle == 1 = [Star]
  | otherwise = (++[Star]) . takeWhile (/= Star) . map k . iterate double $ angle
  where
    k a
      | a `elem` [ angle / 2 , (angle + 1) / 2 ] = Star
      | angle / 2 < a && a < (angle + 1) / 2     = One
      | a < angle / 2 || (angle + 1) / 2 < a     = Zero

period :: Rational -> Integer
period angle = genericLength (knead angle)

internalAddress :: [Knead] -> [Integer]
internalAddress v = iA 1 [Star]
  where
    iA sk vk
      | sk == genericLength v = [ ]
      | otherwise = sk' : iA sk' vk'
      where
        sk' = (+) 1 . genericLength . takeWhile id $ zipWith (==) (cycle v) (cycle vk)
        vk' = genericTake sk' v

orbit :: Eq a => (a -> Maybe a) -> a -> [a]
orbit f x = x : unfoldr (fmap both . f) x
  where
    both z = (z, z)
```

```
rho :: [Knead] -> Integer -> Maybe Integer
rho v r = listToMaybe . concat $ zipWith3 f [r+1 .. 1000000] (zipWith (flip ↙
    ↳ const) v (genericDrop r (cycle v))) v
  where
    f k a b
      | a /= b = [k]
      | otherwise = []

denominators :: [Knead] -> [Integer]
denominators v = zipWith f a (tail a)
  where
    a = internalAddress v
    f sk sk1
      | sk `elem` orbit (rho v) r = (sk1 - r) `div` sk + 1
      | otherwise                 = (sk1 - r) `div` sk + 2
      where
        r | sk1 `mod` sk == 0 = sk
          | otherwise = sk1 `mod` sk

numerators :: Rational -> [Integer] -> [Integer] -> [Integer]
numerators angle = zipWith f
  where
    f qk sk = genericLength . filter (<= angle) $ [ wrap $ 2^(i * sk) * angle | ↙
      ↳ i <- [0 .. qk - 2] ]

data Address = P Integer | S Integer Rational Address
  deriving (Eq, Ord, Show)

angledInternalAddress :: Rational -> Address
angledInternalAddress angle = foldr (\(s, pq) a -> S s pq a) (P (last ss)) (zip ↙
    ↳ ss rs)
  where
    rs = zipWith (%) ns ds
    ns = numerators angle ds ss
    ds = denominators ks
    ss = internalAddress ks
    ks = knead angle

externalAngles :: Address -> Maybe (Rational, Rational)
externalAngles = externalAngles' 1 (0, 1)

externalAngles' :: Integer -> (Rational, Rational) -> Address -> Maybe (Rational↙
    ↳ , Rational)
externalAngles' p0 lohi a0@(P p)
  | p0 /= p = case wakees lohi p of
      [lh] -> externalAngles' p lh a0
      _ -> Nothing
  | otherwise = Just lohi
externalAngles' p0 lohi a0@(S p r a)
  | p0 /= p = case wakees lohi p of
      [lh] -> externalAngles' p lh a0
      _ -> Nothing
  | otherwise = do
      let num = numerator r
          den = denominator r
          q = p * den
          ws = wakees lohi q
```

```
            nums = [ num' | num' <- [ 1.. den - 1 ], let r' = num' % den, ↙
                ↳ denominator r' == den ]
            nws, nnums :: Integer
            nws = genericLength ws
            nnums = genericLength nums
        guard (nws == nnums)
        i <- genericElemIndex num nums
        lh <- safeGenericIndex ws (i :: Integer)
        externalAngles' q lh a

wakees :: (Rational, Rational) -> Integer -> [(Rational, Rational)]
wakees (lo, hi) q =
  let gaps (l, h) n
        | n == 0 = [(l, h)]
        | n > 0 = let gs = gaps (l, h) (n - 1)
                      cs = candidates n gs
                  in  accumulate cs gs
      candidates n gs =
        let den = 2 ^ n - 1
        in  [ r
            | (l, h) <- gs
            , num <- [ ceiling (l * fromInteger den)
                     .. floor (h * fromInteger den) ]
            , let r = num % den
            , l < r, r < h
            , period r == n
            ]
      accumulate [] ws = ws
      accumulate (l : h : lhs) ws =
        let (ls, ms@((ml, _):_)) = break (l `inside`) ws
            (_s, (_, rh):rs) = break (h `inside`) ms
        in  ls ++ [(ml, l)] ++ accumulate lhs ((h, rh) : rs)
      inside x (l, h) = l < x && x < h
  in  chunk2 . candidates q . gaps (lo, hi) $ (q - 1)

chunk2 :: [t] -> [(t, t)]
chunk2 [] = []
chunk2 (x:y:zs) = (x, y) : chunk2 zs

genericElemIndex :: (Eq a, Integral b) => a -> [a] -> Maybe b
genericElemIndex _ [] = Nothing
genericElemIndex e (f:fs)
  | e == f = Just 0
  | otherwise = (1 +) `fmap` genericElemIndex e fs

safeGenericIndex :: Integral b => [a] -> b -> Maybe a
safeGenericIndex [] _ = Nothing
safeGenericIndex (x:xs) i
  | i < 0 = Nothing
  | i > 0 = safeGenericIndex xs (i - 1)
  | otherwise = Just x

safeLast :: [a] -> Maybe a
safeLast [] = Nothing
safeLast xs = Just (last xs)

radius :: (Real r, Floating r) => r
```

```
radius = 2 ** 24

sharpness :: Int
sharpness = 4

limit :: Int
limit = 64

distance :: Int
distance = 64

ray :: (Real r, Floating r, Turbo r) => Rational -> [Complex r]
ray angle = map fst . iterate (step angle) $ (mkPolar radius (2 * pi * ↙
    ↳ fromRational angle), (0, 0))

step :: (Real r, Floating r, Turbo r) => Rational -> (Complex r, (Int, Int)) -> ↙
    ↳ (Complex r, (Int, Int))
step angle (c, (k0, j0))
  | j > sharpness = step angle (c, (k0 + 1, 0))
  | otherwise = (c', (k0, j0 + 1))
  where
    k = k0 + 1
    j = j0 + 1
    m = (k - 1) * sharpness + j
    r = radius ** ((1/2) ** (fromIntegral m / fromIntegral sharpness))
    t = mkPolar (r ** (2 ** fromIntegral k0)) ((2 ** fromIntegral k0) * 2 * pi *↙
        ↳ fromRational angle)
    c' = iterate n c !! limit
    n z = z - (cc - t) / dd
     where
      (cc, dd) = ncnd k
      ncnd 1 = (z, 1)
      ncnd i = let (nc, nd) = ncnd (i - 1) in (nc * nc + z, 2 * nc * nd + 1)

rayEnd :: (Real r, Floating r, Turbo r) => Rational -> Maybe (Complex r)
rayEnd = safeLast . takeWhile (\(r:+i) -> not (isNaN r || isNaN i)) . take (↙
    ↳ sharpness * distance) . ray

parameter :: (NearZero r, Real r, Floating r, Turbo r) => Address -> Maybe (r, r↙
    ↳ , r)
parameter a = do
  (lo, hi) <- externalAngles a
  c1 <- rayEnd lo
  c2 <- rayEnd hi
  let c = 0.5 * (c1 + c2)
  return $ refineNucleus (addressPeriod a) c

addressPeriod :: Address -> Integer
addressPeriod (P p) = p
addressPeriod (S _ _ a) = addressPeriod a

parse :: String -> Maybe Address
parse s = case runP parser () "" s of
  Left _ -> Nothing
  Right a -> Just a

data Token = Number Integer | Fraction Integer Integer
```

```
245 type Parse t = ParsecT String () Identity t

    parser :: Parse Address
    parser = do
      ts <- pTokens
250   accum 1 ts
      where
        accum p [] = return $ P p
        accum _ [Number n] = return $ P n
        accum _ (Number n : ts@(Number _ : _)) = do
255       a <- accum n ts
          return $ S n (1%2) a
        accum _ (Number n : Fraction t b : ts) = do
          a <- accum (n * b) ts
          return $ S n (t%b) a
260     accum p (Fraction t b : ts) = do
          a <- accum (p * b) ts
          return $ S p (t % b) a

    pTokens :: Parse [Token]
265 pTokens = do
      _ <- pOptionalSpace
      ts <- pToken `sepBy` pSpace
      _ <- pOptionalSpace
      eof
270   return ts

    pToken :: Parse Token
    pToken = choice [ try pFraction, pNumber ]

275 pFraction :: Parse Token
    pFraction = do
      Number top <- pNumber
      _ <- pOptionalSpace
      _ <- string "/"
280   _ <- pOptionalSpace
      Number bottom <- pNumber
      guard $ top < bottom
      return $ Fraction top bottom

285 pNumber :: Parse Token
    pNumber = do
      n <- foldl (\x y -> 10 * x + y) 0 `fmap` map (toInteger . digitToInt) `fmap` ↙
          ↳ many1 digit
      guard $ 0 < n
      return $ Number n
290
    pSpace :: Parse [String]
    pSpace = many1 (string " ")

    pOptionalSpace :: Parse [String]
295 pOptionalSpace = many (string " ")

    pretty :: Address -> String
    pretty (P p) = show p
    pretty (S p r a) = show p ++ " " ++ show (numerator r) ++ "/" ++ show (↙
```

```
    ↪ denominator r) ++ " " ++ pretty a
```

## 2 Calculate.hs

```
{-# LANGUAGE BangPatterns #-}
{-# LANGUAGE FlexibleContexts #-}

{-
    gmndl -- Mandelbrot Set explorer
    Copyright (C) 2010,2011,2014,2017  Claude Heiland-Allen <claude@mathr.co.uk>

    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 program; if not, write to the Free Software Foundation, Inc.,
    51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.

-}

module Calculate (convert, renderer) where

-- simple helpers
import Control.Monad (when)

-- concurrent renderer with capability-specific scheduling
import Control.Concurrent (MVar, newEmptyMVar, takeMVar, tryTakeMVar, tryPutMVar
    ↪ , threadDelay, forkIO, killThread)
import GHC.Conc (forkOn, numCapabilities)

-- each worker uses a mutable unboxed array of Bool to know which pixels
-- it has already started to render, to avoid pointless work duplication
import Data.Array.IO (IOUArray, newArray, readArray, writeArray, inRange)

-- each worker thread keeps a queue of pixels that it needs to render or
-- to continue rendering later
import Data.PriorityQueue (PriorityQueue, newPriorityQueue, enqueue,
    ↪ enqueueBatch, dequeue)

-- poking bytes into memory is dirty, but it's quick and allows use of
-- other fast functions like memset and easy integration with OpenGL
import Foreign (Word8)

-- higher precision arithmetic using libqd
import Numeric.QD.DoubleDouble (DoubleDouble())
import Numeric.QD.QuadDouble (QuadDouble())

import Complex (Complex((:+)), Turbo(sqr, twice), convert)
```

```
-- some type aliases to shorten things
type B = Word8
type N = Int
type R = Double

{-
-- colour space conversion from HSV [0..1] to RGB [0..1]
-- HSV looks quite 'chemical' to my eyes, need to investigate something
-- better to make it feel more 'natural'
hsv2rgb :: R -> R -> R -> (R, R, R)
hsv2rgb !h !s !v
  | s == 0 = (v, v, v)
  | h == 1 = hsv2rgb 0 s v
  | otherwise =
      let !i = floor (h * 6) `mod` 6 :: N
          !f = (h * 6) - fromIntegral i
          !p = v * (1 - s)
          !q = v * (1 - s * f)
          !t = v * (1 - s * (1 - f))
      in  case i of
            0 -> (v, t, p)
            1 -> (q, v, p)
            2 -> (p, v, t)
            3 -> (p, q, v)
            4 -> (t, p, v)
            5 -> (v, p, q)
            _ -> (0, 0, 0)
-}

hsv2rgb' :: R -> R -> R -> (R, R, R)
hsv2rgb' !h !s !l =
  let !a = 2 * pi * h
      !ca = cos a
      !sa = sin a
      !y = l / 2
      !u = s / 2 * ca
      !v = s / 2 * sa
      !r = y + 1.407 * u
      !g = y - 0.677 * u - 0.236 * v
      !b = y + 1.848 * v
  in  (r, g, b)

-- compute RGB [0..255] bytes from the results of the complex iterations
-- don't need very high precision for this, as spatial aliasing will be
-- much more of a problem in intricate regions of the fractal
colour :: Complex Double -> Complex Double -> N -> (B, B, B)
colour !(zr:+zi) !(dzr:+dzi) !n =
  let -- micro-optimization - there is no log2 function
      !il2 = 1 / log 2
      !zd2 = sqr zr + sqr zi
      !dzd2 = sqr dzr + sqr dzi
      -- normalized escape time
      !d = (fromIntegral n :: R) - log (log zd2 / log escapeR2) * il2
      !dwell = fromIntegral (floor d :: N)
      -- final angle of the iterate
      !finala = atan2 zi zr
      -- distance estimate
```

```
        !de = (log zd2 * il2) * sqrt (zd2 / dzd2)
        !dscale = -log de * il2
        -- HSV is based on escape time, distance estimate, and angle
        !hue = log ((- log de * il2) `max` 1) * il2 / 16 + log d * il2
        !saturation = 0 `max` (log d * il2 / 8) `min` 1
        !value = 0 `max` (1 - dscale / 256) `min` 1
        !h = hue - fromIntegral (floor hue :: N)
        -- adjust saturation to give concentric striped pattern
        !k = dwell / 2
        !satf = if k - fromIntegral (floor k :: N) >= (0.5 :: R) then 0.9 else 1
        -- adjust value to give tiled pattern
        !valf = if finala < 0 then 0.9 else 1
        -- convert to RGB
        (!r, !g, !b) = hsv2rgb' h (satf * saturation) (valf * value)
        -- convert to bytes
        !rr = floor $ 0 `max` (255 * r) `min` 255
        !gg = floor $ 0 `max` (255 * g) `min` 255
        !bb = floor $ 0 `max` (255 * b) `min` 255
    in  (rr, gg, bb)

-- a Job stores a pixel undergoing iterations
data Job c = Job !N !N !(Complex c) !(Complex c) !(Complex c) !N

-- the priority of a Job is how many iterations have been computed:
-- so 'fresher' pixels drop to the front of the queue in the hope of
-- avoiding too much work iterating pixels that will never escape
priority :: Job c -> N
priority !(Job _ _ _ _ _ n) = n

-- add a job to a work queue, taking care not to duplicate work
-- there is no race condition here as each worker has its own queue
addJob :: (Real c, Floating c, Turbo c) => N -> N -> Complex c -> c -> ↙
    ↳ PriorityQueue IO (Job c) -> IOUArray (N,N) Bool -> N -> N -> IO ()
addJob !w !h !c !zradius' todo sync !i !j = do
  already <- readArray sync (j, i)
  when (not already) $ do
    writeArray sync (j, i) True
    enqueue todo $! Job i j (coords w h c zradius' i j) 0 0 0

-- spawns a new batch of workers to render an image
-- returns an action that stops the rendering
renderer' :: (Turbo c, Real c, Floating c) => MVar () -> ((N,N),(N,N)) -> (N -> ↙
    ↳ N -> B -> B -> B -> IO ()) -> Complex c -> c -> IO (IO ())
renderer' done rng output !c !zradius' = do
  wdog <- newEmptyMVar
  workerts <- mapM (\w -> forkOn w $ worker wdog rng c zradius' output w) [ 0 ..↙
      ↳  workers - 1 ]
  watcher <- forkIO $ do
    () <- takeMVar wdog
    let loop = do
          threadDelay 10000000 -- 10 seconds
          m <- tryTakeMVar wdog
          case m of
            Nothing -> mapM_ killThread workerts >> tryPutMVar done () >> return↙
                ↳  ()
            Just () -> loop
    loop
```

```
  return $ killThread watcher >> mapM_ killThread workerts

-- compute the Complex 'c' coordinate for a pixel in the image
coords :: (Real c, Floating c, Turbo c) => N -> N -> Complex c -> c -> N -> N -> Complex c
coords !w !h !c !zradius' !i !j = c + ( (fromIntegral (i - w`div`2) * k)
                                      :+(fromIntegral (h`div`2 - j) * k))
  where !k = zradius' / (fromIntegral $ (w `div` 2) `min` (h `div` 2))

-- the worker thread enqueues its border and starts computing iterations
worker :: (Turbo c, Real c, Floating c) => MVar () -> ((N,N),(N,N)) -> Complex c -> c -> (N -> N -> B -> B -> B -> IO ()) -> N -> IO ()
worker wdog rng@((y0,x0),(y1,x1)) !c !zradius' output !me = do
  sync <- newArray rng False
  queue <- newPriorityQueue priority
  let addJ = addJob w h c zradius' queue sync
      js = filter mine (border w h)
      w = x1 - x0 + 1
      h = y1 - y0 + 1
  mapM_ (flip (writeArray sync) True) js
  enqueueBatch queue (map (\(j,i) -> Job i j (coords w h c zradius' i j) 0 0 0) js)
  compute wdog rng addJ output queue
  where mine (_, i) = i `mod` workers == me -- another dependency on spread

-- the compute engine pulls pixels from the queue until there are no
-- more, and calculates a batch of iterations for each
compute :: (Turbo c, Real c, Floating c) => MVar () -> ((N,N),(N,N)) -> (N -> N -> IO ()) -> (N -> N -> B -> B -> B -> IO ()) -> PriorityQueue IO (Job c) -> IO ()
compute wdog rng addJ output queue = do
  mjob <- dequeue queue
  case mjob of
    Just (Job i j c z dz n) -> do
      let -- called when the pixel escapes
          done' !(zr:+zi) !(dzr:+dzi) !n' = {-# SCC "done" #-} do
            _ <- tryPutMVar wdog ()
            let (r, g, b) = colour (convert zr :+ convert zi) (convert dzr :+ convert dzi) n'
            output i j r g b
            -- a wavefront of computation spreads to neighbouring pixels
            sequence_
              [ addJ x y
              | u <- spreadX
              , v <- spreadY
              , let x = i + u
              , let y = j + v
              , inRange rng (y, x)
              ]
          -- called when the pixel doesn't escape yet
          todo' !z' !dz' !n' = {-# SCC "todo" #-} {- output i j 255 0 0 >> -} enqueue queue $! Job i j c z' dz' n'
      calculate c limit z dz n done' todo'
      compute wdog rng addJ output queue
    Nothing -> return () -- no pixels left to render, so finish quietly

-- the raw z->z^2+c calculation engine
```

```
-- also computes the derivative for distance estimation calculations
-- this function is crucial for speed, too much allocation will slooow
-- everything down severely
calculate :: (Turbo c, Real c, Floating c) => Complex c -> N -> Complex c -> ↙
    ↳ Complex c -> N -> (Complex c -> Complex c -> N -> IO ()) -> (Complex c -> ↙
    ↳ Complex c -> N -> IO ()) -> IO ()
calculate !c !m0 !z0 !dz0 !n0 done todo = go m0 z0 dz0 n0
  where
    go !m !z@(zr:+zi) !dz !n
      | not (sqr zr + sqr zi < er2) = done z dz n
      | m <= 0 = todo z dz n
      | otherwise = go (m - 1) (sqr z + c) (let !zdz = z * dz in twice zdz + 1) ↙
          ↳ (n + 1)
    !er2 = convert escapeR2

-- dispatch to different instances of renderer depending on required precision
-- if zoom is low, single precision Float is ok, but as soon as pixel spacing
-- gets really small, it's necessary to increase it
-- it's probably not even worth using Float - worth benchmarking this and
-- also the DD and QD types (which cause a massively noticeable slowdown)
renderer :: (Real c, Floating c) => MVar () -> ((N,N),(N,N)) -> (N -> N -> B -> ↙
    ↳ B -> B -> IO ()) -> Complex c -> c -> IO (IO ())
renderer done rng output !c !zradius'
  | zoom' < 20  = {-# SCC "rF"  #-} renderer' done rng output (f c :: Complex ↙
      ↳ Float       ) (g zradius')
  | zoom' < 50  = {-# SCC "rD"  #-} renderer' done rng output (f c :: Complex ↙
      ↳ Double      ) (g zradius')
  | zoom' < 100 = {-# SCC "rDD" #-} renderer' done rng output (f c :: Complex ↙
      ↳ DoubleDouble) (g zradius')
  | otherwise   = {-# SCC "rQD" #-} renderer' done rng output (f c :: Complex ↙
      ↳ QuadDouble  ) (g zradius')
  where f !(cr :+ ci) = convert cr :+ convert ci
        g !x = convert x
        zoom' = - logBase 2 (zradius' / (fromIntegral $ w `min` h))
        ((x0,y0), (x1, y1)) = rng
        w = x1 - x0 + 1
        h = y1 - y0 + 1

-- start rendering pixels from the edge of the image
-- the Mandelbrot Set and its complement are both simply-connected
-- discounting spatial aliasing any point inside the boundary that is
-- in the complement is 'reachable' from a point on the boundary that
-- is also in the complement - probably some heavy math involved to
-- prove this though
-- note: this implicitly depends on the spread values below - it's
-- necessary for each interlaced subimage (one per worker) to have
-- at least a one pixel deep border
border :: N -> N -> [(N, N)]
border !w !h = concat $
  [ [ (j, i) | i <- [ 0 .. w - 1 ], j <- [ 0 ] ]
  , [ (j, i) | j <- [ 0 .. h - 1 ], i <- [ 0 .. workers - 1 ] ]
  , [ (j, i) | j <- [ 0 .. h - 1 ], i <- [ w - workers .. w - 1 ] ]
  , [ (j, i) | i <- [ 0 .. w - 1 ], j <- [ h - 1 ] ]
  ]

-- which neighbours to activate once a pixel has escaped
-- there are essentially two choices, with x<->y swapped
```

```
    -- choose greater X spread because images are often wider than tall
    -- other schemes wherein the spread is split in both directions
    -- might benefit appearance with large worker count, but too complicated
    spreadX, spreadY :: [ N ]
265 spreadX = [ -workers, 0, workers ]
    spreadY = [ -1, 0, 1 ]

    -- number of worker threads
    -- use as many worker threads as capabilities, with the workers
270 -- distributed 1-1 onto capabilities to maximize CPU utilization
    workers :: N
    workers = numCapabilities

    -- iteration limit per pixel
275 -- at most this many iterations are performed on each pixel before it
    -- is shunted to the back of the work queue
    -- this should be tuneable to balance display updates against overheads
    limit :: N
    limit = (2^(13::N)-1)
280
    -- escape radius for fractal iteration calculations
    -- once the complex iterate exceeds this, it's never coming back
    -- theoretically escapeR = 2 would work
    -- but higher values like this give a significantly smoother picture
285 escapeR, escapeR2 :: R
    escapeR = 65536
    escapeR2 = escapeR * escapeR
```

# 3 Complex.hs

```
    {-# LANGUAGE BangPatterns, FlexibleContexts #-}

    {-

 5      gmndl -- Mandelbrot Set explorer
        Copyright (C) 2010,2011,2014  Claude Heiland-Allen <claude@mathr.co.uk>

        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
10      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
15      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 program; if not, write to the Free Software Foundation, Inc.,
20      51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.

    -}

    module Complex where
25
    import Prelude hiding (atan2)
```

```
import Foreign.C (CDouble)

-- higher precision arithmetic using libqd
import Numeric.QD.DoubleDouble (DoubleDouble(DoubleDouble))
import Numeric.QD.QuadDouble (QuadDouble(QuadDouble))
import qualified Numeric.QD.DoubleDouble as DD
import qualified Numeric.QD.QuadDouble as QD
import Numeric.AD.Mode.Reverse (Reverse)
import Data.Reflection (Reifies)
import Numeric.AD.Internal.Reverse (Tape)

-- ugly! but the default realToFrac :: (C)Double -> (C)Double is slooow
import Unsafe.Coerce (unsafeCoerce)

-- don't look! this is really really ugly, and should be benchmarked
-- to see how really necessary it is, or at least made into a type class
convert :: (Real a, Fractional b) => a -> b
convert = realToFrac
{-# NOINLINE convert #-}
convertDouble2CDouble :: Double -> CDouble
convertDouble2CDouble !x = unsafeCoerce x
convertCDouble2Double :: CDouble -> Double
convertCDouble2Double !x = unsafeCoerce x
convertDouble2DoubleDouble :: Double -> DoubleDouble
convertDouble2DoubleDouble !x = convertCDouble2DoubleDouble . ↙
    ↳ convertDouble2CDouble $ x
convertCDouble2DoubleDouble :: CDouble -> DoubleDouble
convertCDouble2DoubleDouble !x = DoubleDouble x 0
convertDoubleDouble2Double :: DoubleDouble -> Double
convertDoubleDouble2Double !(DoubleDouble x _) = convertCDouble2Double x
convertDoubleDouble2CDouble :: DoubleDouble -> CDouble
convertDoubleDouble2CDouble !(DoubleDouble x _) = x
{-# RULES "convert/Double2CDouble" convert = convertDouble2CDouble #-}
{-# RULES "convert/CDouble2Double" convert = convertCDouble2Double #-}
{-# RULES "convert/Double2DoubleDouble" convert = convertDouble2DoubleDouble #-}
{-# RULES "convert/CDouble2DoubleDouble" convert = convertCDouble2DoubleDouble ↙
    ↳ #-}
{-# RULES "convert/DoubleDouble2Double" convert = convertDoubleDouble2Double #-}
{-# RULES "convert/DoubleDouble2CDouble" convert = convertDoubleDouble2CDouble ↙
    ↳ #-}

{-
-- this is ugly too: can't use Data.Complex because the qd bindings do
-- not implement some low-level functions properly, leading to obscure
-- crashes inside various Data.Complex functions...
data Complex c = {-# UNPACK #-} !c :+ {-# UNPACK #-} !c deriving (Read, Show, Eq↙
    ↳ )

-- complex number arithmetic, with extra strictness and cost-centres
instance Num c => Num (Complex c) where
  (!(a :+ b)) + (!(c :+ d)) = {-# SCC "C+" #-} ((a + c) :+ (b + d))
  (!(a :+ b)) - (!(c :+ d)) = {-# SCC "C-" #-} ((a - c) :+ (b - d))
  (!(a :+ b)) * (!(c :+ d)) = {-# SCC "C*" #-} ((a * c - b * d) :+ (a * d + b * ↙
      ↳ c))
  negate !(a :+ b) = (-a) :+ (-b)
  abs x = error $ "Complex.abs: " ++ show x
  signum x = error $ "Complex.signum: " ++ show x
```

```
  fromInteger !x = fromInteger x :+ 0
-}

-- an extra class for some operations that can be made faster for things
-- like DoubleDouble: probably should have given this a better name
class Num c => Turbo c where
  sqr :: c -> c
  sqr !x = x * x
  twice :: c -> c
  twice !x = x + x

-- the default methods are fine for simple primitive types...
instance Turbo Float where
instance Turbo Double where
instance Turbo CDouble where

-- ...and complex numbers
instance (Real c, Floating c, Turbo c) => Turbo (Complex c) where
  sqr !(r :+ i) = (sqr r - sqr i) :+ (twice (r * i))
  twice !(r :+ i) = (twice r) :+ (twice i)

-- use the specific implementations for the higher precision types
instance Turbo DoubleDouble where
  sqr !x = DD.sqr x
  twice !(DoubleDouble a b) = DoubleDouble (twice a) (twice b)

instance Turbo QuadDouble where
  sqr !x = QD.sqr x
  twice !(QuadDouble a b c d) = QuadDouble (twice a) (twice b) (twice c) (twice d)

instance (Reifies s Tape, Num r) => Turbo (Reverse s r) where


data Complex r = !r :+ !r
  deriving (Read, Show, Eq)

instance (Real r, Floating r, Turbo r) => Num (Complex r) where
  (a :+ b) + (x :+ y) = (a + x) :+ (b + y)
  (a :+ b) - (x :+ y) = (a - x) :+ (b - y)
  (a :+ b) * (x :+ y) = (a * x - b * y) :+ (a * y + b * x)
  negate (a :+ b) = negate a :+ negate b
  abs c = magnitude c :+ 0
  signum = normalize
  fromInteger n = fromInteger n :+ 0

instance (Real r, Floating r, Turbo r) => Fractional (Complex r) where
  (a:+b) / (c:+d) = ((a * c + b * d)/m2) :+ ((b * c - a * d)/m2) where m2 = sqr c + sqr d
  fromRational r = fromRational r :+ 0

magnitude :: (Real c, Floating c, Turbo c) => Complex c -> c
magnitude (re:+im) = sqrt $ sqr re + sqr im

cis :: (Real c, Floating c) => c -> Complex c
cis a = cos a :+ sin a
```

```
mkPolar :: (Real c, Floating c) => c -> c -> Complex c
mkPolar r a = (r * cos a) :+ (r * sin a)

phase :: (Real c, Floating c) => Complex c -> c
phase (re:+im) = atan2 im re

normalize :: (Real c, Floating c, Turbo c) => Complex c -> Complex c
normalize z@(re:+im) = let m = magnitude z in (re / m) :+ (im / m)

atan2 :: (Real c, Floating c) => c -> c -> c
atan2 y x
      | x > 0             =  atan (y/x)
      | x == 0 && y > 0   =  pi/2
      | x <  0 && y > 0   =  pi + atan (y/x)
      | x <= 0 && y < 0   = -atan2 (-y) x
      | y == 0 && x < 0   =  pi    -- must be after the previous test on zero y
      | x == 0 && y == 0  =  y     -- must be after the other double zero tests
      | otherwise         =  x + y -- x or y is a NaN, return a NaN (via +)
```

# 4 .gitignore

```
.cabal-sandbox
cabal.sandbox.config
dist
```

# 5 gmndl.cabal

```
Name:                gmndl
Version:             0.4.0.4
Synopsis:            Mandelbrot Set explorer using GTK

Description:
                     A Mandelbrot Set explorer.  Multiple render threads
                     use higher precision maths at higher zoom levels.
                     Suggested usage:
                     .
                     @gmndl +RTS -N -qa -RTS --width=640 --height=480@
                     .
                     Left-click to zoom in, right-click to zoom out.  The
                     status bar shows where you are, and the entry field
                     takes an angled internal address, try for example:
                     .
                     @1\/3 1\/2 5 7@

Cabal-version:       >=1.6
License:             GPL-2
License-file:        LICENSE
Author:              Claude Heiland-Allen
Maintainer:          claude@mathr.co.uk
Category:            Graphics
Build-type:          Simple

Executable gmndl
  Main-is:           gmndl.hs
  Other-modules:     Address
                     Calculate
                     Complex
```

```
                    Image
                    MuAtom
                    Roots
  Build-depends:    base >= 4 && < 5,
                    array >= 0.5 && < 0.6,
                    gtk >= 0.12 && < 0.15,
                    gtkglext >= 0.12 && < 0.14,
                    mtl >= 2.2 && < 2.3,
                    OpenGL >= 3.0 && < 3.1,
                    OpenGLRaw >= 3.2 && < 3.3,
                    parsec >= 3.1 && < 3.2,
                    priority-queue >= 0.2 && < 0.3,
                    qd >= 1.0 && < 1.1,
                    ad >= 4.2 && < 4.4,
                    reflection >= 1.5 && < 2.2,
                    Vec >= 1.0 && < 1.1
  GHC-options:      -O2 -Wall -threaded -fno-excess-precision -funbox-strict-
    ↳ fields -rtsopts

source-repository head
  type:     git
  location: https://code.mathr.co.uk/gmndl.git

source-repository this
  type:     git
  location: https://code.mathr.co.uk/gmndl.git
  tag:      v0.4.0.4
```

# 6 gmndl.hs

```
{-

    gmndl -- Mandelbrot Set explorer
    Copyright (C) 2010,2011,2014  Claude Heiland-Allen <claude@mathr.co.uk>

    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 program; if not, write to the Free Software Foundation, Inc.,
    51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.

-}

module Main (main) where

import Prelude hiding (isNaN)

import Control.Concurrent (forkIO, newEmptyMVar, takeMVar, putMVar)
import Control.Monad (forever)
```

```
-- some simple helpers
import Data.List (isPrefixOf)

-- the dependency on mtl is just for this!
import Control.Monad.Trans (liftIO)

-- the main program thread needs to store some thread-local state
import Data.IORef (newIORef, readIORef, writeIORef)

-- build the interface with GTK to allow more fancy controls later
import Graphics.UI.Gtk

-- use OpenGL to display frequently update images on a textured quad
import Graphics.UI.Gtk.OpenGL
import qualified Graphics.Rendering.OpenGL as GL
import Graphics.Rendering.OpenGL ((($=))

-- need a hack to ensure correct qd operation
import Numeric.QD.QuadDouble (QuadDouble())
import Foreign (nullPtr)

import Complex (Complex((:+)))

-- mu-atom properties
import Calculate
import qualified Image
import Address (parse, parameter)

isNaN x = not (x == x)

-- the state we need for everything
data GMNDL
  = Invalid
  | GMNDL
      { center :: Complex QuadDouble
      , zradius :: QuadDouble
      , image :: Image.Image
      , stop :: IO ()
      }

-- command line arguments: currently only initial window dimensions
data Args = Args{ aWidth :: Int, aHeight :: Int, aOversample :: Int, aRe :: ↙
    ↳ QuadDouble, aIm :: QuadDouble, aZr :: QuadDouble }

-- and the defaults are suitable for PAL DVD rendering, if that should
-- come to pass in the future
defaultArgs :: Args
defaultArgs = Args{ aWidth = 788, aHeight = 576, aOversample = 1, aRe = 0, aIm =↙
    ↳  0, aZr = 2 }

-- braindead argument parser: latest argument takes priority
-- probably should use Monoid instances for this stuff
combineArgs :: Args -> String -> Args
combineArgs a0 s
  | "--width="  `isPrefixOf` s = a0{ aWidth  = read $ "--width="  `dropPrefix` s↙
      ↳  }
```

```
    | "-w="       `isPrefixOf` s = a0{ aWidth  = read $ "-w="       `dropPrefix` s }
    | "--height=" `isPrefixOf` s = a0{ aHeight = read $ "--height=" `dropPrefix` s }
    | "-h="       `isPrefixOf` s = a0{ aHeight = read $ "-h="       `dropPrefix` s }
    | "--aa="     `isPrefixOf` s = a0{ aOversample = read $ "--aa=" `dropPrefix` s }
    | "--re="     `isPrefixOf` s = a0{ aRe = read $ "--re=" `dropPrefix` s }
    | "--im="     `isPrefixOf` s = a0{ aIm = read $ "--im=" `dropPrefix` s }
    | "--zr="     `isPrefixOf` s = a0{ aZr = read $ "--zr=" `dropPrefix` s }
    | otherwise = a0

-- this is a bit silly, especially with the duplicated string literals..
dropPrefix :: String -> String -> String
dropPrefix p s = drop (length p) s

-- the main program!
main :: IO ()
main = do
  args <- foldl combineArgs defaultArgs `fmap` unsafeInitGUIForThreadedRTS
  let width = aWidth args
      height = aHeight args
      oversample = aOversample args
      rng = ((0, 0), (oversample * height - 1, oversample * width - 1))
  _ <- initGL
  glconfig <- glConfigNew [ GLModeRGBA, GLModeDouble ]
  canvas <- glDrawingAreaNew glconfig
  widgetSetSizeRequest canvas width height
  window <- windowNew
  eventb <- eventBoxNew
  vbox <- vBoxNew False 0
  status <- vBoxNew False 0
  statusRe <- entryNew
  statusIm <- entryNew
  statusZr <- entryNew
  ratios <- entryNew
  boxPackStart vbox eventb PackGrow 0
  boxPackStart vbox status PackGrow 0
  boxPackStart vbox ratios PackGrow 0
  boxPackStart status statusRe PackGrow 0
  boxPackStart status statusIm PackGrow 0
  boxPackStart status statusZr PackGrow 0
  let -- update the status bar
      updateStatus re im zr = do
        entrySetText statusRe (show re)
        entrySetText statusIm (show im)
        entrySetText statusZr (show zr)
  set window [ containerBorderWidth := 0, containerChild := vbox,
      windowResizable := False ]
  set eventb [ containerBorderWidth := 0, containerChild := canvas ]
  -- initial state is invalid because...
  sR <- newIORef Invalid
  done <- newEmptyMVar
  let -- restart the renderer
      restart :: IO ()
      restart = do
```

```
        g <- readIORef sR
        stop g
        Image.clear (image g)
        stop' <- renderer done rng (Image.plot (image g)) (center g) (zradius g)
        writeIORef sR $! g{ stop = stop' }
        let re :+ im = center g
        updateStatus re im (zradius g)
    -- ...need to initialize OpenGL stuff etc in this callback
    _ <- onRealize canvas $ {-# SCC "cbRz" #-} withGLDrawingArea canvas $ \_ -> do
      GL.matrixMode $= GL.Projection
      GL.loadIdentity
      GL.ortho 0.0 1.0 0.0 1.0 (-1.0) 1.0
      GL.drawBuffer $= GL.BackBuffers
      GL.texture GL.Texture2D $= GL.Enabled
      i <- Image.new (oversample * width) (oversample * height)
      writeIORef sR $! GMNDL{ image = i, center = aRe args :+ aIm args, zradius = ↙
          ↳ aZr args, stop = return () }
      restart
    -- when the mouse button is pressed, center and zoom in or out
    _ <- eventb `on` buttonPressEvent $ {-# SCC "cbEv" #-} tryEvent $ do
      b <- eventButton
      (x, y) <- eventCoordinates
      liftIO $ do
        g <- readIORef sR
        let w2 = fromIntegral width  / 2
            h2 = fromIntegral height / 2
            p  = convert x :+ convert (-y)
            s  = (zradius g / (w2 `min` h2)) :+ 0
            c  = center g + (p - (w2 :+ (-h2))) * s
            zradius' = zradius g * delta
            delta | b == LeftButton = 0.5
                  | b == RightButton = 2
                  | otherwise = 1
        writeIORef sR $! g{ center = c, zradius = zradius' }
        restart
    -- when typing in the coordinate boxes, zoom to the new place
    _ <- statusRe `onEntryActivate` do
      s <- entryGetText statusRe
      liftIO $ do
        g <- readIORef sR
        case safeRead s of
          Just re -> do
            let _ :+ im = center g
            writeIORef sR $! g{ center = re :+ im }
            restart
          Nothing -> return ()
    _ <- statusIm `onEntryActivate` do
      s <- entryGetText statusIm
      liftIO $ do
        g <- readIORef sR
        case safeRead s of
          Just im -> do
            let re :+ _ = center g
            writeIORef sR $! g{ center = re :+ im }
            restart
          Nothing -> return ()
    _ <- statusZr `onEntryActivate` do
```

```
      s <- entryGetText statusZr
      liftIO $ do
        g <- readIORef sR
        case safeRead s of
          Just zradius' -> do
            writeIORef sR $! g{ zradius = zradius' }
            restart
          Nothing -> return ()
    -- when pressing return in the ratios list, zoom to that mu-atom
    muQueue <- newEmptyMVar
    _ <- forkIO . forever $ do
      qs <- takeMVar muQueue
      case parameter =<< parse qs of
        Nothing -> postGUISync $ do
          _ <- ratios `widgetSetSensitive` True
          return ()
        Just (cr, ci, radius) -> do
          let zradius' = radius * 3
          cr `seq` ci `seq` zradius' `seq` postGUISync $ do
            g <- readIORef sR
            if isNaN cr || isNaN ci
              then writeIORef sR $! g{ center = c0, zradius = zradius0 }
              else writeIORef sR $! g{ center = cr :+ ci, zradius = zradius' }
            _ <- ratios `widgetSetSensitive` True
            restart
    _ <- ratios `onEntryActivate` do
      s <- entryGetText ratios
      _ <- ratios `widgetSetSensitive` False
      g <- readIORef sR
      stop g
      putMVar muQueue s
    -- time to draw the image: upload to the texture and draw a quad
    _ <- onExpose canvas $ {-# SCC "cbEx" #-} \_ -> do
      withGLDrawingArea canvas $ \glwindow -> do
        GMNDL{ image = i } <- readIORef sR
        Image.upload i
        Image.draw i
        glDrawableSwapBuffers glwindow
      return True
    -- need an exit strategy
    _ <- onDestroy window mainQuit
    -- make sure the expose callback gets called regularly (5fps)
    _ <- timeoutAdd (widgetQueueDraw canvas >> return True) 200
    -- and we're off!
    widgetShowAll window
    mainGUI

-- initial center coordinates
-- using the maximum precision available from the start for this makes
-- sure that nothing weird happens when precision gets close to the edge
c0 :: Complex QuadDouble
c0 = 0

-- initial zoom level
-- the initial zoom level should probably depend on initial image size
zradius0 :: QuadDouble
zradius0 = 2
```

```
safeRead :: Read a => String -> Maybe a
safeRead s = case reads s of
  [(a, "")] -> Just a
  _ -> Nothing
```

# 7 Image.hs

```
{-# LANGUAGE ForeignFunctionInterface #-}
{-# LANGUAGE PatternSynonyms #-}

{-

    gmndl -- Mandelbrot Set explorer
    Copyright (C) 2010,2011,2014,2017  Claude Heiland-Allen <claude@mathr.co.uk>

    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 program; if not, write to the Free Software Foundation, Inc.,
    51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.

-}

module Image (Image(), new, clear, plot, upload, with, draw, putPPM, hPutPPM) ↙
    ↳ where

-- poking bytes into memory is dirty, but it's quick and allows use of
-- other fast functions like memset and easy integration with OpenGL
import Foreign (castPtr, mallocBytes, nullPtr, plusPtr, pokeByteOff, Ptr, Word8)
import Foreign.C (CInt(..), CSize(..))
import qualified Graphics.Rendering.OpenGL as GL
import Graphics.Rendering.OpenGL (($=))
import Graphics.GL (glGenerateMipmap, pattern GL_TEXTURE_2D)
import System.IO (Handle, hPutBuf, hPutStr, stdout)

data Image
  = Image
      { pixels :: Ptr Word8
      , width :: Int
      , height :: Int
      , size :: Int
      , texture :: GL.TextureObject
      }

-- these images are RGB only
channels :: Int
channels = 3
```

```
-- create a new image
-- note: this needs an OpenGL context
new :: Int -> Int -> IO Image
new w h = do
  p <- mallocBytes $ w * h * channels
  [t] <- GL.genObjectNames 1
  let s = roundUp2 (w `max` h)
      i = Image{ pixels = p, width = w, height = h, size = s, texture = t }
      dim = GL.TextureSize2D (fromIntegral s) (fromIntegral s)
      img = GL.PixelData GL.RGB GL.UnsignedByte nullPtr
  with i $ do
    GL.texImage2D GL.Texture2D GL.NoProxy 0 GL.RGB' dim 0 img
    GL.textureFilter GL.Texture2D $= ((GL.Linear', Just GL.Linear'), GL.Linear')
    GL.textureWrapMode GL.Texture2D GL.S $= (GL.Repeated, GL.ClampToEdge)
    GL.textureWrapMode GL.Texture2D GL.T $= (GL.Repeated, GL.ClampToEdge)
  clear i
  upload i
  return i

-- clear with white
clear :: Image -> IO ()
clear i = do
  let bytes = width i * height i * channels
  _ <- memset (castPtr (pixels i)) 255 (fromIntegral bytes)
  return ()

-- plot a pixel
plot :: Image -> Int -> Int -> Word8 -> Word8 -> Word8 -> IO ()
plot i x y r g b = do
  let p = pixels i `plusPtr` ((y * width i + x) * channels)
  pokeByteOff p 0 r
  pokeByteOff p 1 g
  pokeByteOff p 2 b

-- upload an image to its texture
upload :: Image -> IO ()
upload i = do
  let pos = GL.TexturePosition2D 0 0
      dim = GL.TextureSize2D (fromIntegral $ width i) (fromIntegral $ height i)
      img = GL.PixelData GL.RGB GL.UnsignedByte (pixels i)
  with i $ do
    GL.texSubImage2D GL.Texture2D 0 pos dim img
    glGenerateMipmap GL_TEXTURE_2D

-- use a texture
-- FIXME TODO preserve the previous texture binding instead of clearing
with :: Image -> IO a -> IO a
with i act = do
  GL.textureBinding GL.Texture2D $= Just (texture i)
  r <- act
  GL.textureBinding GL.Texture2D $= Nothing
  return r

-- draw textured unit quad
draw :: Image -> IO ()
draw i = do
  let v :: GL.GLfloat -> GL.GLfloat -> GL.GLfloat -> GL.GLfloat -> IO ()
```

```
      v tx ty vx vy = GL.texCoord (GL.TexCoord2 tx ty) >> GL.vertex (GL.Vertex2 ↙
          ↳ vx vy)
      sx = fromIntegral (width  i) / fromIntegral (size i)
      sy = fromIntegral (height i) / fromIntegral (size i)
  with i $ GL.renderPrimitive GL.Quads $ do
    v 0  sy 0 0
    v 0   0 0 1
    v sx  0 1 1
    v sx sy 1 0

-- save as PPM
putPPM :: Image -> IO ()
putPPM = hPutPPM stdout

hPutPPM :: Handle -> Image -> IO ()
hPutPPM h i = do
  hPutStr h ("P6\n" ++ show (width i) ++ " " ++ show (height i) ++ " 255\n")
  hPutBuf h (pixels i) (width i * height i * channels)

-- round up to nearest power of two
-- this will probably explode when n gets large, but it's only used
-- for OpenGL texture dimensions so you'll run out of memory first
roundUp2 :: Int -> Int
roundUp2 n = head . dropWhile (< n) . iterate (2*) $ 1

-- import standard C library memset for clearing images efficiently
-- previous implementation used pokeArray ... (replicate ...) ...
-- which had a nasty habit of keeping the list around in memory
foreign import ccall unsafe "string.h memset"
  c_memset :: Ptr Word8 -> CInt -> CSize -> IO (Ptr Word8)
memset :: Ptr Word8 -> Word8 -> CSize -> IO (Ptr Word8)
memset p w s = c_memset p (fromIntegral w) s
```

# 8 LICENSE

GNU GENERAL PUBLIC LICENSE
Version 2, June 1991

```
 Copyright (C) 1989, 1991 Free Software Foundation, Inc.,
 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.

                      Preamble

  The licenses for most software are designed to take away your
freedom to share and change it.  By contrast, the GNU General Public
License is intended to guarantee your freedom to share and change free
software--to make sure the software is free for all its users.  This
General Public License applies to most of the Free Software
Foundation's software and to any other program whose authors commit to
using it.  (Some other Free Software Foundation software is covered by
the GNU Lesser General Public License instead.)  You can apply it to
your programs, too.

  When we speak of free software, we are referring to freedom, not
price.  Our General Public Licenses are designed to make sure that you
```

have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's

source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein.

You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free

programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the program's name and a brief idea of what it does.>
Copyright (C) <year> <name of author>

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 program; if not, write to the Free Software Foundation, Inc.,

```
    51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this
when it starts in an interactive mode:

    Gnomovision version 69, Copyright (C) year name of author
    Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
    This is free software, and you are welcome to redistribute it
    under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate
parts of the General Public License.  Of course, the commands you use may
be called something other than `show w' and `show c'; they could even be
mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your
school, if any, to sign a "copyright disclaimer" for the program, if
necessary.  Here is a sample; alter the names:

  Yoyodyne, Inc., hereby disclaims all copyright interest in the program
  `Gnomovision' (which makes passes at compilers) written by James Hacker.

  <signature of Ty Coon>, 1 April 1989
  Ty Coon, President of Vice

This General Public License does not permit incorporating your program into
proprietary programs.  If your program is a subroutine library, you may
consider it more useful to permit linking proprietary applications with the
library.  If this is what you want to do, use the GNU Lesser General
Public License instead of this License.
```

# 9 MuAtom.hs

```
{-# LANGUAGE BangPatterns, RecordWildCards, Rank2Types, FlexibleContexts #-}

{-

    gmndl -- Mandelbrot Set explorer
    Copyright (C) 2010,2011,2014  Claude Heiland-Allen <claude@mathr.co.uk>

    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 program; if not, write to the Free Software Foundation, Inc.,
    51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.

-}
```

```
module MuAtom (muAtom, refineNucleus) where

import Data.Ratio (numerator, denominator)
import Data.List (genericIndex, genericSplitAt)
import Data.Vec (NearZero())
import Numeric.QD (QuadDouble())
import Roots (root2, root4, lift, FF)
import Complex (Complex((:+)), magnitude, phase, cis, mkPolar, normalize, Turbo)

type N = Integer
type Q = Rational
type R = QuadDouble
type C = Complex R

-- a Mandelbrot Set mu-atom

data Atom = Atom{ nucleus :: !C, period :: !N, root :: !C, cardioid :: !Bool }
  deriving Show

continent :: Atom
continent = Atom 0 1 1 True

-- finding bond points

fdf :: (Integral i, Num c) => i -> c -> c -> (c, c)
fdf !n !z !c = let (fzs, fz:_) = genericSplitAt n $ iterate (\w -> w * w + c) z
               in  (fz, 2 ^ n * product fzs)  -- [ f i z c | i <- [0 .. n - 1] ]

bondIter :: (Real r, Floating r) => Integer -> Complex r -> FF [] [] r
bondIter !n !(br:+bi) [x0, x1, x2, x3] =
  let !z = x0:+x1
      !c = x2:+x3
      !b = lift br :+ lift bi
      (!fz, !dfz) = fdf n z c
      !(y0 :+ y1) =  fz - z -- f n z c - z
      !(y2 :+ y3) = dfz - b -- df n z c - (lift br :+ lift bi)
  in  [y0, y1, y2, y3]
bondIter _ _ _ = error "MuAtom.bondIter: internal error"

-- finding nucleus

l :: (Integral i, Num c) => i -> c -> c
l !n !c = (`genericIndex` n) . iterate (\z -> z * z + c) $ 0

nucleusIter :: (Real r, Floating r) => Integer -> FF [] [] r
nucleusIter !n [x0, x1] =
  let !c = x0 :+ x1
      !(y0 :+ y1) = l n c
  in  [y0, y1]
nucleusIter _ _ = error "MuAtom.nucleusIter: internal error"


refineNucleus :: (NearZero r, Real r, Floating r, Turbo r) => Integer -> Complex↙
    ↳ r -> (r, r, r)
refineNucleus p root@(gr :+ gi) =
  let eps = 1e-20 -- FIXME
```

```
        [cr, ci] = root2 eps (nucleusIter p) [gr, gi]
        [_, _, b0r, b0i] = root4 eps (bondIter p ( 1)) [cr, ci, cr, ci]
        [_, _, b1r, b1i] = root4 eps (bondIter p (-1)) [cr, ci, cr, ci]
        bond0 = b0r :+ b0i
        bond1 = b1r :+ b1i
        r = magnitude (bond1 - bond0)
    in  (cr, ci, r)

-- finding descendants

muChild :: Atom -> Q -> Atom
muChild !Atom{..} !address =
  let -- some properties of the parent and its relation to the child
      !size = magnitude (root - nucleus)
      !address' = fromIntegral (numerator address) / fromIntegral (denominator ↙
          ↳ address)
      !angle = 2 * pi * address'
      !(bar :+ bai) = cis angle
      !bondAngle = bar :+ bai
      !child = period * denominator address
      -- perturb from the stable nucleus to help ensure convergence to the bond ↙
          ↳ point
      !_initial@(ir :+ ii) = nucleus + mkPolar (size / 2) (phase (root - nucleus↙
          ↳ ) + angle)
      [_, _, br, bi] = {-# SCC "bond" #-} root4 eps (bondIter period bondAngle) ↙
          ↳ [ ir, ii, ir, ii ]
      !bondPoint = br :+ bi
      -- estimate where the nucleus will be
      !radiusEstimate
        | cardioid  = size / m2 * sin (pi * address')
        | otherwise = size / m2
        where m2 = fromIntegral (denominator address) ^ (2 :: N)
      !deltaEstimate = bondPoint - nucleus
      !_guess@(gr :+ gi) = bondPoint + (radiusEstimate :+ 0) * normalize ↙
          ↳ deltaEstimate
      -- refine the nucleus estimate
      [cr, ci] = {-# SCC "nucleus" #-} root2 eps (nucleusIter child) [gr, gi]
      !childNucleus = cr :+ ci
      eps = radiusEstimate / 10000000
  in  Atom childNucleus child bondPoint False

muChildren :: Atom -> [Q] -> [Atom]
muChildren !a [] = [a]
muChildren !a (q:qs) = let b = muChild a q in a : muChildren b qs

-- interface to the outside world

muAtom :: [Q] -> (R, R, R, N)
muAtom qs =
  let Atom{..} = last $ muChildren continent qs
      r :+ i = nucleus
      s = magnitude (nucleus - root)
      p = period
  in (r, i, s, p)
```

# 10 Roots.hs

```
{-# LANGUAGE BangPatterns, Rank2Types, ScopedTypeVariables, FlexibleContexts, ↙
    ↳ NoMonomorphismRestriction #-}

{-

    gmndl -- Mandelbrot Set explorer
    Copyright (C) 2010,2011,2014  Claude Heiland-Allen <claude@mathr.co.uk>

    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 program; if not, write to the Free Software Foundation, Inc.,
    51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.

-}

module Roots (root2, root4, FF, lift) where

import Prelude hiding (zipWith)
import Data.Maybe (fromJust)
import Data.Functor ((<$>))
import Data.Vec (toList, solve, fromList, matFromLists, zipWith, Vec2, Mat22, ↙
    ↳ Vec4, Mat44, NearZero(nearZero))
import Data.Reflection (Reifies)
import Numeric.AD (jacobian', auto)
import Numeric.AD.Mode.Reverse (Reverse)
import Numeric.AD.Internal.Reverse (Tape)
import Numeric.QD (QuadDouble())

lift = auto
type FF f g a = forall s. Reifies s Tape => f (Reverse s a) -> g (Reverse s a)

root2' :: forall r . (Fractional r, NearZero r, Ord r) => r -> FF [] [] r -> ↙
    ↳ Vec2 r -> Vec2 r
root2' eps f !x = go x
  where
    jf = jacobian' f
    go x0 =
      let (ys, js) = unzip $ jf (toList x0)
          y = fromList (negate <$> ys) :: Vec2 r
          j = matFromLists js :: Mat22 r
          dx = fromJust $ solve j y
          x1 = zipWith (+) x0 dx
      in  if all (not . (> eps)) (abs <$> ys)
            then  x0
            else  if all (not . (> eps)) (abs <$> toList dx)
                    then x1
                    else go x1
```

```
root2 :: (Fractional r, NearZero r, Ord r) => r -> FF [] [] r -> [r] -> [r]
root2 eps f = toList . root2' eps f . fromList

root4' :: forall r . (Fractional r, NearZero r, Ord r) => r -> FF [] [] r -> ↙
    ↳ Vec4 r -> Vec4 r
root4' eps f !x = go x
  where
    jf = jacobian' f
    go x0 =
      let (ys, js) = unzip $ jf (toList x0)
          y = fromList (negate <$> ys) :: Vec4 r
          j = matFromLists js :: Mat44 r
          dx = fromJust $ solve j y
          x1 = zipWith (+) x0 dx
      in  if all (not . (> eps)) (abs <$> ys)
            then  x0
            else  if all (not . (> eps)) (abs <$> toList dx)
                    then x1
                    else go x1

root4 :: (Fractional r, NearZero r, Ord r) => r -> FF [] [] r -> [r] -> [r]
root4 eps f = toList . root4' eps f . fromList

instance NearZero QuadDouble where
  nearZero x = not (abs x > 1e-60) -- NearZero Double has 1e-14
```

# 11 Setup.hs

```
import Distribution.Simple
main = defaultMain
```