et

Claude Heiland-Allen

2017–2019

Contents

1 appveyor.sh

```
# Configure the environment
MSYSTEM=MINGW64
source /etc/profile || true # a terrible, terrible workaround for msys2 ↙
    ↳ brokenness

# Don't set -e until after /etc/profile is sourced
set -ex
cd $APPVEYOR_BUILD_FOLDER

export GHCVERSION="8.0.2"
export PATH="/opt/ghc-${GHCVERSION}/bin:/opt/ghc-${GHCVERSION}/lib/bin:/opt/ghc-↙
    ↳ ${GHCVERSION}/mingw/bin:/opt/ghc-${GHCVERSION}/mingw/x86_64-w64-mingw32/↙
    ↳ bin:/usr/local/bin:${PATH}"

case "$1" in
    "prepare")
        # Bring msys up-to-date
        # However, we current don't do this: generally one must restart all
```

```
        # msys2 processes when updating the msys2 runtime, which this may do. We can't
        # easily do this and therefore do simply don't update.
        #pacman --noconfirm -Syuu

        # Install basic build dependencies
        pacman --noconfirm -S --needed git tar bsdtar binutils autoconf make xz curl libtool automake python python2 p7zip patch mingw-w64-$(uname -m)-tools-git

        # Install build dependencies
        mkdir -p /opt
        wget -q -O - https://downloads.haskell.org/~ghc/8.0.2/ghc-8.0.2-x86_64-unknown-mingw32-win10.tar.xz | tar -xJ -C /opt
        mkdir -p /usr/local/bin
        wget -q -O - https://www.haskell.org/cabal/release/cabal-install-2.0.0.1/cabal-install-2.0.0.1-x86_64-unknown-mingw32.zip | bsdtar -xzf- -C /usr/local/bin
        cabal update
        git clone https://code.mathr.co.uk/long-double.git
        git clone https://code.mathr.co.uk/rounded.git
        ;;

    "build")
        cabal install -j --prefix=/usr/local --disable-profiling --disable-documentation --constraint="transformers-compat +five" et.cabal rounded/rounded.cabal long-double/long-double.cabal
        et-cli out.png 640 360 0 0 2 256 1 0 0 1
        ;;

    "test")
        mkdir /tmp/et-windows
        cp out.png /tmp/et-windows/
        cp `which et-cli` /tmp/et-windows/
        cp `dirname $(which ghc)`/../mingw/bin/libgmp-10.dll /tmp/et-windows/
        cp `dirname $(which ghc)`/../mingw/bin/libwinpthread-1.dll /tmp/et-windows/
        7z a et-windows.zip /tmp/et-windows
        ;;

    *)
        echo "$0: unknown mode $1"
        exit 1
        ;;
esac
```

2 appveyor.yml

```
version: "0.0.{build}"

build:
  verbosity: normal

environment:
  matrix:
  - COMPILER: msys2
    PLATFORM: x64
```

```
      MSYS2_ARCH: x86_64
      MSYS2_DIR: msys64
      MSYSTEM: MINGW64
      BIT: 64

os: Visual Studio 2015
deploy: off

install:
  - cmd: |
      SET "PATH=C:\%MSYS2_DIR%\%MSYSTEM%\bin;C:\%MSYS2_DIR%\usr\bin;%PATH%"
      bash appveyor.sh prepare
build_script:
  - bash appveyor.sh build
  - bash appveyor.sh test

artifacts:
  - path: et-windows.zip
    name: et for Windows
    type: zip
```

3 ChangeLog.md

```
# Revision history for et

## 0  -- YYYY-mm-dd

* First version. Released on an unsuspecting world.
```

4 cli/Main.hs

```
{-
et -- escape time fractals
Copyright (C) 2018,2019 Claude Heiland-Allen

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU Affero General Public License as
published by the Free Software Foundation, either version 3 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 Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License
along with this program.  If not, see <https://www.gnu.org/licenses/>.
-}

module Main (main) where

import Control.Exception (Exception, catch, throwIO)
import Control.Monad (forM_, unless, when)
import Data.Word (Word8)
import Foreign (peek, poke, malloc, free, sizeOf)
import Foreign.C.Types (CInt)
```

```
import System.Environment (getArgs, getProgName)
import System.Exit (exitSuccess, exitFailure)
import System.IO (hPutStrLn, hPutBuf, withBinaryFile, IOMode(WriteMode), stderr)
import Control.Concurrent.Async (async, wait)
import qualified Data.Vector.Storable.Mutable as VM
import qualified Data.Vector.Storable as V
import qualified Data.ByteString.Lazy as BS
import Codec.Picture (generateImage, PixelYA8(..), PixelRGB8(..), PixelRGBA8(..)
    ↪ )
import Codec.Picture.Png (encodePngWithMetadata)
import Codec.Picture.Metadata (Keys(Software, Comment, ColorSpace), singleton, ↙
    ↪ ColorSpace(SRGB))

import qualified Data.Binary as B
import qualified Codec.Picture as J
import qualified Codec.Picture.Metadata as J
import qualified Codec.Picture.Png.Internal.Metadata as J

import Numeric.Rounded.Simple (scaleFloat', exponent')
import System.Directory (createDirectoryIfMissing)
import System.Environment.XDG.BaseDir (getUserCacheDir)
import System.FilePath ((</>))
import Fractal.EscapeTime.Runtime.Compiler (compileAndLoad)
import Fractal.EscapeTime.Runtime.Loader (close)
import Fractal.EscapeTime.Runtime.Metadata as M
import Fractal.EscapeTime.Runtime.Render
import Fractal.EscapeTime.Runtime.Render.Transform (transform)

saveRAW :: Bool
saveRAW = False

main :: IO ()
main = do
  args <- getArgs
  case args of
    [outfile,sw,sh,infile] -> do
      w <- readIO sw
      h <- readIO sh
      raw <- BS.readFile infile
      case (J.lookup J.Comment . J.extractMetadatas . B.decode $ raw) >>= M.↙
          ↪ fromString of
        Nothing -> hPutStrLn stderr "error: could not parse metadata from input ↙
            ↪ file"
        Just m -> main' outfile w h m
    [outfile,sw,sh,sre,sim,sr,sn,saa,sab,sba,sbb,sd,se,sp,sq,formula,sweight] ->↙
        ↪ do
      w <- readIO sw
      h <- readIO sh
      View a b r <- readViewIO sre sim sr
      maxiters <- readIO sn
      aa <- readIO saa
      ab <- readIO sab
      ba <- readIO sba
      bb <- readIO sbb
      d <- readIO sd
      e <- readIO se
      p <- readIO sp
```

```
      q <- readIO sq
      weight <- readIO sweight
      let m = Metadata (lines formula) p q d e a b r (aa, ab, ba, bb) maxiters ↙
          ↳ weight
      main' outfile w h m
    _ -> do
      prog <- getProgName
      hPutStrLn stderr $ "usage: " ++ prog ++ " out.png width height input-with-↙
          ↳ metadata.png"
      hPutStrLn stderr $ "usage: " ++ prog ++ " out.png width height real imag ↙
          ↳ radius maxiters aa ab ba bb d e p q formula weight"
  exitFailure

main' :: FilePath -> Int -> Int -> Metadata -> IO ()
main' outfile w h m@(Metadata formula p q d e a b r (aa, ab, ba, bb) maxiters ↙
    ↳ weight) = do
  case transform aa ab ba bb of
    Nothing -> hPutStrLn stderr "error: singular transform"
    Just tr -> do
      progress "compiling..."
      cacheDir <- getUserCacheDir "et"
      let libDir = cacheDir </> "lib"
      createDirectoryIfMissing True libDir
      result <- compileAndLoad libDir (unlines formula) p q
      case result of
        Left msg -> hPutStrLn stderr msg
        Right et -> do
          let er = 10000
          bufr <- buffer w h 4 (-1)
          out <- if saveRAW then Left <$> buffer w h 1 0 else Right <$> buffer w↙
              ↳ h 3 0
          let prog = Nothing
          progress $ "rendering ..."
          (_cancelR, waitR) <- render et er maxiters d e (View a b r) bufr tr ↙
              ↳ prog
          okR <- waitR
          unless okR exitFailure
          case out of
            Left raw -> do
              progress "extracting ..."
              (_cancelC, waitC) <- extract bufr raw
              okC <- waitC
              unless okC exitFailure
              progress "exporting ..."
              writeRAW outfile raw
            Right ppm -> do
              progress "colouring ..."
              (_cancelC, waitC) <- colour bufr ppm (realToFrac weight)
              okC <- waitC
              unless okC exitFailure
              progress "exporting ..."
              let comment = M.toString m
              writePNG comment outfile ppm
          progress "unloading..."
          close et
          exitSuccess
```

```
{-
-- zoom in animation:
forM_ ([exponent' r - ceiling (logBase 2 (fromIntegral h)) .. 0] `zip` [0..]) $ \(ex, fr) -> do
  let outfile = (++ ".png") . reverse . take 4 . (++ "000") . reverse . show $ fr
      r' = scaleFloat' (-ex) r
  ...
-}

progress :: String -> IO ()
progress = hPutStrLn stderr

writeRAW :: FilePath -> Buffer Float -> IO ()
writeRAW file (Buffer w h c v) = do
  withBinaryFile file WriteMode $ \hd -> do
    VM.unsafeWith v $ \p -> do
      hPutBuf hd p (w * h * c * sizeOf (0 :: Float))

writePNG :: String -> FilePath -> Buffer Word8 -> IO ()
writePNG comment outfile (Buffer w h 1 v) = do
  u <- V.freeze v
  let image = generateImage (\x y -> let ix = (y * w + x) * 1 in (u V.! (ix + 0))) w h
  BS.writeFile outfile (encodePngWithMetadata (mconcat [singleton Software "et-0", singleton Comment comment, singleton ColorSpace SRGB]) image)
writePNG comment outfile (Buffer w h 2 v) = do
  u <- V.freeze v
  let image = generateImage (\x y -> let ix = (y * w + x) * 2 in PixelYA8 (u V.! (ix + 0)) (u V.! (ix + 1))) w h
  BS.writeFile outfile (encodePngWithMetadata (mconcat [singleton Software "et-0", singleton Comment comment, singleton ColorSpace SRGB]) image)
writePNG comment outfile (Buffer w h 3 v) = do
  u <- V.freeze v
  let image = generateImage (\x y -> let ix = (y * w + x) * 3 in PixelRGB8 (u V.! (ix + 0)) (u V.! (ix + 1)) (u V.! (ix + 2))) w h
  BS.writeFile outfile (encodePngWithMetadata (mconcat [singleton Software "et-0", singleton Comment comment, singleton ColorSpace SRGB]) image)
writePNG comment outfile (Buffer w h 4 v) = do
  u <- V.freeze v
  let image = generateImage (\x y -> let ix = (y * w + x) * 4 in PixelRGBA8 (u V.! (ix + 0)) (u V.! (ix + 1)) (u V.! (ix + 2)) (u V.! (ix + 3))) w h
  BS.writeFile outfile (encodePngWithMetadata (mconcat [singleton Software "et-0", singleton Comment comment, singleton ColorSpace SRGB]) image)
writePNG _ _ _ = fail "writePNG: bad number of channels"

colour :: Buffer Float -> Buffer Word8 -> Float -> IO (IO Bool, IO Bool)
colour (Buffer sw sh sc@4 output) (Buffer iw ih ic@3 image) weight | sw == iw && sh == ih = do
  runningP <- malloc
  poke runningP (1 :: CInt)
  a <- async $ Control.Exception.catch (do
    forM_ [ 0 .. sh - 1 ] $ \j -> forM_ [ 0 .. sw - 1 ] $ \i -> do
      running <- peek runningP
      when (running == 0) $ throwIO Cancelled
      de <- VM.read output ((j * sw + i) * sc + 0)
      let grey x = (x, x, x)
```

```
175             (r, g, b)
                  | de > 0 = grey (round . (\z -> if isNaN z || isInfinite z || z < 0 ↙
                      ↳ then 0 else z) $ 255 * tanh (clamp (2 + log de * weight) 0 8))
                  | de == 0 = (0, 0, 0)
                  | de < 0 = (0, 0, 0)
          VM.write image (((j * iw + i) * ic + 0) r
180       VM.write image (((j * iw + i) * ic + 1) g
          VM.write image (((j * iw + i) * ic + 2) b
        return True
        ) (\Cancelled -> return False)
      return  ( do{ poke runningP 0 ; r <- wait a ; free runningP ; return r }
185           , do{                   r <- wait a ; free runningP ; return r }
              )
    colour _ _ _ = return (return False, return False)

    extract :: Buffer Float -> Buffer Float -> IO (IO Bool, IO Bool)
190 extract (Buffer sw sh sc@4 output) (Buffer iw ih ic@1 image) | sw == iw && sh ==↙
      ↳ ih = do
      runningP <- malloc
      poke runningP (1 :: CInt)
      a <- async $ Control.Exception.catch (do
        forM_ [ 0 .. sh - 1 ] $ \j -> forM_ [ 0 .. sw - 1 ] $ \i -> do
195       running <- peek runningP
          when (running == 0) $ throwIO Cancelled
          de <- VM.read output (((j * sw + i) * sc + 0)
          VM.write image (((j * iw + i) * ic + 0) de
        return True
200     ) (\Cancelled -> return False)
      return  ( do{ poke runningP 0 ; r <- wait a ; free runningP ; return r }
              , do{                   r <- wait a ; free runningP ; return r }
              )
    extract _ _ = return (return False, return False)
205
    data Cancelled = Cancelled deriving Show
    instance Exception Cancelled

    clamp :: Ord a => a -> a -> a -> a
210 clamp x lo hi = lo `max` x `min` hi
```

5 et.cabal

```
    name:                et
    version:             0
    synopsis:            escape-time fractals
    description:
 5    Escape-time fractals powered by a formula compiler.  Graphics are visualized
      using distance estimation, normalized iteration count, and/or atom-domain
      colouring.  Navigation is enhanced by Newton-Raphson zooming: a single action
      can bring you to a mini-set or an embedded Julia.

10  homepage:            https://mathr.co.uk/et/
    license:             AGPL-3
    license-file:        LICENSE.md
    author:              Claude Heiland-Allen
    maintainer:          claude@mathr.co.uk
15  copyright:           (c) 2018,2019 Claude Heiland-Allen
    category:            Graphics
```

```
build-type:            Simple
extra-source-files:    ChangeLog.md
cabal-version:         >=1.10
data-dir: src
data-files:
  et.png
  formula.h
  LICENSE.md

flag gtk
  description: Build GTK frontend
  default: True

library
  hs-source-dirs: src
  exposed-modules:
    Fractal.EscapeTime.Prelude
    Fractal.EscapeTime.Algorithms.R2
    Fractal.EscapeTime.Algorithms.R2.DomainSize
    Fractal.EscapeTime.Algorithms.R2.Misiurewicz
    Fractal.EscapeTime.Algorithms.R2.Newton
    Fractal.EscapeTime.Algorithms.R2.PeriodJSK
    Fractal.EscapeTime.Algorithms.R2.PeriodTri
    Fractal.EscapeTime.Algorithms.R2.Perturbation
    Fractal.EscapeTime.Algorithms.R2.Plain
    Fractal.EscapeTime.Algorithms.R2.Reference
    Fractal.EscapeTime.Algorithms.R2.Size
    Fractal.EscapeTime.Algorithms.R2.Skew
    Fractal.EscapeTime.Compiler
    Fractal.EscapeTime.Compiler.CodeGen
    Fractal.EscapeTime.Compiler.CodeGen.Control
    Fractal.EscapeTime.Compiler.CodeGen.Core
    Fractal.EscapeTime.Compiler.CodeGen.Expr
    Fractal.EscapeTime.Compiler.CodeGen.OpCode
    Fractal.EscapeTime.Compiler.Expr
    Fractal.EscapeTime.Compiler.Expr.AST
    Fractal.EscapeTime.Compiler.Expr.Deriv
    Fractal.EscapeTime.Compiler.Expr.Optimize
    Fractal.EscapeTime.Compiler.Expr.Perturb
    Fractal.EscapeTime.Compiler.Expr.Pretty
    Fractal.EscapeTime.Compiler.Expr.Simplify
    Fractal.EscapeTime.Compiler.Parser
    Fractal.EscapeTime.Compiler.R2
    Fractal.EscapeTime.Formulas
    Fractal.EscapeTime.Formulas.Types
    Fractal.EscapeTime.Runtime
    Fractal.EscapeTime.Runtime.Compiler
    Fractal.EscapeTime.Runtime.Loader
    Fractal.EscapeTime.Runtime.Loader.Load
    Fractal.EscapeTime.Runtime.Loader.Raw
    Fractal.EscapeTime.Runtime.Metadata
    Fractal.EscapeTime.Runtime.Render
    Fractal.EscapeTime.Runtime.Render.Buffer
    Fractal.EscapeTime.Runtime.Render.Coord
    Fractal.EscapeTime.Runtime.Render.Dither
    Fractal.EscapeTime.Runtime.Render.Perturb
    Fractal.EscapeTime.Runtime.Render.Plain
```

```
      Fractal.EscapeTime.Runtime.Render.Progressive
      Fractal.EscapeTime.Runtime.Render.Transform
      Fractal.EscapeTime.Runtime.Render.View
      Paths_et
    default-language: Haskell2010
    other-extensions:
      DataKinds
      DeriveDataTypeable
      FlexibleContexts
      FlexibleInstances
      ForeignFunctionInterface
      RecordWildCards
      ScopedTypeVariables
    build-depends:
      async >=2.1.1 && <2.3,
      atomic-primops >=0.8 && <0.9,
      base >=4.9 && <4.14,
      containers >=0.5 && <0.7,
      directory >=1.3 && <1.4,
      filepath >=1.4 && <1.5,
      long-double >=0.1 && <0.2,
      mtl >=2.2 && <2.3,
      parsec >=3.1 && <3.2,
      process >=1.4 && <1.7,
      pureMD5 >=2.1 && <2.2,
      rounded >=0.2 && <0.3 || >=1.0 && <1.1,
      text >=1.2 && <1.3,
      uniplate >=1.6 && <1.7,
      vector >=0.12 && <0.13,
      vector-algorithms >=0.7 && <0.9,
      vector-mmap >=0.0.3 && <0.1,
      xdg-basedir >=0.2 && <0.3
    if os(windows)
      build-depends: Win32 >=2.3 && <2.8
      hs-source-dirs: win32
    else
      build-depends: unix >=2.7 && <2.8
      hs-source-dirs: unix
    build-tools: hsc2hs
    ghc-options: -Wall
    c-sources: src/wrap.c
    include-dirs: src

  executable et-cli
    hs-source-dirs: cli
    main-is: Main.hs
    default-language: Haskell2010
    build-depends:
      async,
      base,
      binary,
      bytestring >=0.10 && <0.11,
      directory,
      et,
      filepath,
      JuicyPixels >=3.3.3.2 && <3.4,
      rounded,
```

```
    temporary >=1.2.1 && <1.4,
    vector,
    xdg-basedir
  ghc-options: -Wall -threaded -eventlog -rtsopts -with-rtsopts=-N

executable et-gtk
  if flag(gtk)
    buildable: True
  else
    buildable: False
  hs-source-dirs: gtk
  main-is: Main.hs
  default-language: Haskell2010
  other-extensions:
    OverloadedLabels
    OverloadedStrings
    PatternSynonyms
    RecordWildCards
  build-depends:
    async,
    base,
    binary >=0.8 && <0.9,
    bytestring >=0.10 && <0.11,
    directory,
    et,
    filepath,
    gi-cairo >=1.0.22 && <1.1,
    gi-gdk >=3.0 && <3.1,
    gi-glib >=2.0 && <2.1,
    gi-gtk >=3.0.20 && <4,
    gl >= 0.8 && < 0.9,
    haskell-gi-base >=0.21 && <0.24,
    JuicyPixels >=3.3.3.2 && <3.4,
    JuicyPixels-extra >=0.3 && <0.4,
    random >=1.0 && <1.2,
    rounded,
    temporary >=1.2.1 && <1.4,
    text,
    time >=1.8 && <1.10,
    vector,
    xdg-basedir,
    xdg-userdirs >=0.1 && <0.2
  ghc-options: -Wall -threaded -rtsopts -with-rtsopts=-N

executable et-kf
  hs-source-dirs: kf
  main-is: Main.hs
  other-modules:
    Perturbation
    Reference
  default-language: Haskell2010
  build-depends:
    async,
    base,
    containers,
    et
  ghc-options: -Wall -threaded -rtsopts "-with-rtsopts=-N4 -A64M"
```

6 .gitignore

```
-/
bin/et
lib/*.c
.cabal-sandbox/
cabal.sandbox.config
dist/
dist-newstyle/
zoom/zoom
*.ghc.environment.*
*.png
*.ppm
*.jpg
*.so
*.stamp
*.hi
*.o
```

7 gtk/Main.hs

```
{-
et -- escape time fractals
Copyright (C) 2018,2019 Claude Heiland-Allen

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU Affero General Public License as
published by the Free Software Foundation, either version 3 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 Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License
along with this program.  If not, see <https://www.gnu.org/licenses/>.
-}

{-# LANGUAGE OverloadedLabels #-}
{-# LANGUAGE OverloadedStrings #-}
{-# LANGUAGE PatternSynonyms #-}
{-# LANGUAGE RecordWildCards #-}

module Main (main) where

import Control.Exception (catch, SomeException)
import Control.Monad (replicateM, when)
import Data.Bits ((.|.), bit)
import Data.IORef (IORef, newIORef, readIORef, writeIORef, atomicModifyIORef)
import Data.Word (Word8)
import Text.Read (readMaybe)
import Foreign (alloca, allocaArray, malloc, free, with, withArray, peek, ↙
    ↳ peekArray, poke, pokeElemOff, castPtr, nullPtr)
import Foreign.C (withCString, peekCString)
import Foreign.ForeignPtr (mallocForeignPtrBytes, withForeignPtr)
import System.Exit (exitFailure)
```

```
import System.IO (hPutStrLn, stderr)
import System.Random (randomIO, randomRIO)

import Control.Concurrent (forkIO)
import Control.Concurrent.Async (async, wait)
import qualified Data.ByteString.Lazy as BS
import Data.Text (Text)
import qualified Data.Text as T
import qualified Data.Text.IO as T
import Data.Vector.Storable.Mutable (unsafeWith)
import qualified Data.Vector.Storable.Mutable as V
import Numeric.Rounded.Simple (Rounded, RoundingMode(TowardNearest), precision, ↙
    ↳ exponent', encodeFloat', precRound, fromDouble, add_, sub_, mul_, exp_, ↙
    ↳ log_, show')

import Data.Time (getZonedTime, formatTime, defaultTimeLocale)
import System.Directory (createDirectoryIfMissing)
import System.Environment.XDG.BaseDir (getUserCacheDir)
import System.Environment.XDG.UserDir (getUserDir)
import System.FilePath ((</>), takeDirectory)

import GI.GLib (timeoutAdd, pattern PRIORITY_DEFAULT)
import GI.Gdk (keyvalName, EventMask(EventMaskScrollMask, ↙
    ↳ EventMaskPointerMotionMask), ScrollDirection(ScrollDirectionUp, ↙
    ↳ ScrollDirectionDown), threadsAddIdle)
import GI.Gtk hiding (main)
import qualified GI.GLib as GLib
import qualified GI.Gdk as Gdk
import qualified GI.Gtk as Gtk
import Data.GI.Base
import GI.Gdk.Objects.GLContext (GLContext, gLContextGetUseEs, ↙
    ↳ gLContextMakeCurrent, gLContextClearCurrent)
import Graphics.GL

import qualified Data.Binary as B
import qualified Codec.Picture as J
import qualified Codec.Picture.Metadata as J
import qualified Codec.Picture.Extra as J
import qualified Codec.Picture.Png.Internal.Metadata as J

import Fractal.EscapeTime.Formulas.Types (R2(R2))
import Fractal.EscapeTime.Runtime.Compiler (compileAndLoad)
import Fractal.EscapeTime.Runtime.Loader (ET(..))
import Fractal.EscapeTime.Runtime.Render (render)
import Fractal.EscapeTime.Runtime.Render.View
import Fractal.EscapeTime.Runtime.Render.Buffer
import Fractal.EscapeTime.Runtime.Render.Transform
import Fractal.EscapeTime.Runtime.Render.Progressive (Progressive, ↙
    ↳ cacheProgressive)
import Fractal.EscapeTime.Runtime.Metadata as M
import Paths_et (getDataFileName, version)

catchAny :: IO a -> (SomeException -> IO a) -> IO a
catchAny = Control.Exception.catch

toGTK :: IO () -> IO ()
toGTK action = do
```

```
    Gdk.threadsAddIdle GLib.PRIORITY_DEFAULT $ do
      action
      return False
  return ()

data GUI = GUI
  { tmpDir :: FilePath
  , imagePrefix :: FilePath
  , sequenceNumber :: Int
  , formula :: ET
  , formulaName :: String
  , formulaSource :: [String]
  , formulaER :: Double
  , formulaP :: Int
  , formulaQ :: Int
  , iterations :: Int
  , location :: View
  , transformation :: Transform
  , renderBuffer :: Buffer Float
  , progress :: Maybe Progressive
  , imageWidth :: Int
  , imageHeight :: Int
  , displaySize :: Double
  , srcx0 :: Double
  , srcy0 :: Double
  , srcx1 :: Double
  , srcy1 :: Double
  , log2weight :: Double
  , cancel :: IO Bool
  , glarea :: GLArea
  , gl :: GUIGL
  }

data GUIGL = GUIGL
  { fbo, fbo2 :: GLuint
  , blit_vao :: GLuint
  , blit_p :: GLuint
  , blit_u_gamma :: GLint
  , bound_u :: GLint
  , colourize_vao :: GLuint
  , colourize_p :: GLuint
  , weight_u :: GLint
  , tex :: GLuint
  , ftex, ftex2 :: GLuint
  }
defaultGUIGL :: GUIGL
defaultGUIGL = GUIGL 0 0 0 0 (-1) (-1) 0 0 (-1) 0 0 0

defaultGUI :: IO (Either String GUI)
defaultGUI = do
  let name = "Burning Ship"
      source = ["z := (|x| + i |y|)^p + c"]
      er = 10000
      p = 2
      q = 1
      w = 1024
      h = 576
```

```
        maxiters = 256
        view = defaultView
        tr = identity
    cacheDir <- getUserCacheDir "et"
    picturesDir <- getUserDir "PICTURES"
    now <- getZonedTime
    let libDir = cacheDir </> "lib"
        locale = defaultTimeLocale
        imgPrefix = picturesDir </> "et" </> ("et-" ++ formatTime locale "%Y%m%dT%↙
            ↳ H%M%S%z" now ++ "-")
    createDirectoryIfMissing True libDir
    m <- compileAndLoad libDir (unlines (name:source)) p q
    case m of
      Left msg -> return (Left msg)
      Right et -> do
        buf <- buffer w h 4 (-1)
        mprog <- cacheProgressive w h
        let d_ = 1
            e_ = 0
        (cancelR, _waitR) <- render et er maxiters d_ e_ view buf tr mprog
        return $ Right GUI
          { tmpDir = libDir
          , imagePrefix = imgPrefix
          , sequenceNumber = 1
          , formula = et
          , formulaName = name
          , formulaSource = source
          , formulaER = er
          , formulaP = p
          , formulaQ = q
          , iterations = maxiters
          , location = view
          , transformation = tr
          , renderBuffer = buf
          , progress = mprog
          , imageWidth = w
          , imageHeight = h
          , displaySize = 1
          , srcx0 = 0
          , srcy0 = 0
          , srcx1 = fromIntegral w
          , srcy1 = fromIntegral h
          , log2weight = 0
          , cancel = cancelR
          , glarea = error "glarea"
          , gl = defaultGUIGL
          }

cancelRender :: IORef GUI -> IO ()
cancelRender guiR = do
  cancel' <- atomicModifyIORef guiR $ \g -> (g{ cancel = return False }, cancel ↙
      ↳ g)
  cancel'
  return ()

startRender :: IORef GUI -> IO ()
startRender guiR = do
```

```
  cancelRender guiR
  gui <- readIORef guiR
  let View x y r = location gui
      d_ = 1
      e_ = 0
  print (show' x)
  print (show' y)
  print (show' r)
  (cancelR, _waitR) <- render (formula gui) (formulaER gui) (iterations gui) d_ ↙
      ↳ e_ (location gui) (renderBuffer gui) (transformation gui) (progress gui)
  atomicModifyIORef guiR $ \g -> (g{ cancel = cancelR }, ())

type MenuBarDescr = [(Text, [(Text, Maybe (IO ()))])]

createMenuBar :: MenuBarDescr -> IO Gtk.MenuBar
createMenuBar descr
    = do bar <- menuBarNew
         mapM_ (createMenu bar) descr
         return bar
    where
      createMenu bar (name,items)
          = do menu <- menuNew
               item <- menuItemNewWithLabelOrMnemonic name
               menuItemSetSubmenu item (Just menu)
               menuShellAppend bar item
               mapM_ (createMenuItem menu) items
      createMenuItem menu (name,action)
          = do item <- menuItemNewWithLabelOrMnemonic name
               menuShellAppend menu item
               case action of
                 Just act -> onMenuItemActivate item act
                 Nothing  -> onMenuItemActivate item (return ())
      menuItemNewWithLabelOrMnemonic name
          | '_' `elem` T.unpack name = menuItemNewWithMnemonic name
          | otherwise                = menuItemNewWithLabel name

menuBarDescr :: IORef GUI -> Window -> MenuBarDescr
menuBarDescr guiR w
    = [ ("_File",    [ ("_Open", Just (fileOpen guiR w))
{-
                     , ("_Save", Nothing)
                     , ("Save _As", Nothing)
-}
                     , ("_Quit", Just mainQuit)
                     ]
        )
{-
      , ("_Edit",    [ ("_Undo", Nothing)
                     , ("_Redo", Nothing)
                     , ("_Copy", Nothing)
                     , ("_Paste", Nothing)
                     ]
        )
-}
      , ("_Image",   [ ("Save _As", Just (imageSaveAs guiR w))
                     , ("_Size", Just (imageSize guiR w))
{-
```

```
                        , ("_Colouring", Nothing)
-}
                        ]
          )
        , ("Fracta_l",[ ("_Formula", Just (formulaUI guiR w))
{-
                        , ("_Location", Nothing)
                        , ("_Iterations", Nothing)
                        , ("_Transform", Nothing)
-}
                        ]
          )
        , ("_Help",    [ ("_About", Just $ about w)
                        ]
          )
        ]

debug_program :: GLuint -> String -> IO ()
debug_program program name = do
  if program /= 0
  then do
    linked <- with GL_FALSE $ \p -> glGetProgramiv program GL_LINK_STATUS p >> ↙
        ↳ peek p
    when (linked /= GL_TRUE) $ hPutStrLn stderr $ name ++ ": OpenGL shader ↙
        ↳ program link failed"
    len <- with 0 $ \p -> glGetProgramiv program GL_INFO_LOG_LENGTH p >> peek p
    s <- allocaArray (fromIntegral len + 1) $ \p -> do
      glGetProgramInfoLog program len nullPtr p
      pokeElemOff p (fromIntegral len) 0
      peekCString p
    when (not (null s)) $ hPutStrLn stderr $ name ++ ": OpenGL shader program ↙
        ↳ info log\n" ++ s
  else do
    hPutStrLn stderr $ name ++ ": OpenGL shader program creation failed"

debug_shader :: GLuint -> GLenum -> String -> IO ()
debug_shader shader typ name = do
  let tname = case typ of
        GL_VERTEX_SHADER -> "vertex"
        GL_FRAGMENT_SHADER-> "fragment"
        _ -> "unknown"
  if shader /= 0
  then do
    compiled <- with GL_FALSE $ \p -> glGetShaderiv shader GL_COMPILE_STATUS p ↙
        ↳ >> peek p
    when (compiled /= GL_TRUE) $ hPutStrLn stderr $ name ++ ": OpenGL " ++ tname↙
        ↳  ++ " shader compile failed"
    len <- with 0 $ \p -> glGetShaderiv shader GL_INFO_LOG_LENGTH p >> peek p
    s <- allocaArray (fromIntegral len + 1) $ \p -> do
      glGetShaderInfoLog shader len nullPtr p
      pokeElemOff p (fromIntegral len) 0
      peekCString p
    when (not (null s)) $ hPutStrLn stderr $ name ++ ": OpenGL " ++ tname ++ " ↙
        ↳ shader info log\n" ++ s
  else do
    hPutStrLn stderr $ name ++ ": OpenGL " ++ tname ++ " shader creation failed"

```

```
compile_shader :: GLuint -> GLenum -> String -> String -> IO ()
compile_shader program typ name source = do
  shader <- glCreateShader typ
  withCString source $ \p -> with p $ \pp -> glShaderSource shader 1 pp nullPtr
  glCompileShader shader
  debug_shader shader typ name
  glAttachShader program shader
  glDeleteShader shader

compile_program :: String -> String -> String -> IO GLuint
compile_program name vert frag = do
  program <- glCreateProgram
  compile_shader program GL_VERTEX_SHADER name vert
  compile_shader program GL_FRAGMENT_SHADER name frag
  withCString "vertexID" $ \s -> glBindAttribLocation program 0 s
  glLinkProgram program
  debug_program program name
  return program

blit_frag, blit_vert, simple_frag, simple_vert :: String

blit_vert = unlines
  [ "#version 130"
  , "uniform vec4 bounds;"
  , "in float vertexID;"
  , "out vec2 texCoord;"
  , "void main() {"
  , "  if (vertexID == 0.0) { gl_Position = vec4(-1.0, -1.0, 0.0, 1.0); texCoord
    ↪ = bounds.xy; } else"
  , "  if (vertexID == 1.0) { gl_Position = vec4( 1.0, -1.0, 0.0, 1.0); texCoord
    ↪ = bounds.zy; } else"
  , "  if (vertexID == 2.0) { gl_Position = vec4(-1.0,  1.0, 0.0, 1.0); texCoord
    ↪ = bounds.xw; } else"
  , "                       { gl_Position = vec4( 1.0,  1.0, 0.0, 1.0); texCoord
    ↪ = bounds.zw; }"
  , "}"
  ]

blit_frag = unlines
  [ "#version 130"
  , "uniform sampler2D tex;"
  , "uniform bool gamma;"
  , "in vec2 texCoord;"
  , "out vec4 fragColor;"
  , "float sRGB(float c) {"
  , "  c = clamp(c, 0.0, 1.0);"
  , "  const float a = 0.055;"
  , "  if (c <= 0.0031308)"
  , "    return 12.92 * c;"
  , "  else"
  , "    return (1.0 + a) * pow(c, 1.0 / 2.4) - a;"
  , "}"
  , "void main() {"
  , "  vec4 t = texture(tex, texCoord);"
  , "  if (gamma) {"
  , "    t.r = sRGB(t.r);"
  , "    t.g = sRGB(t.g);"
```

```
    , "    t.b = sRGB(t.b);"
    , "  }"
    , "  fragColor = t;"
    , "}"
    ]

simple_vert = unlines
    [ "#version 130"
    , "in float vertexID;"
    , "out vec2 texCoord;"
    , "void main() {"
    , "  if (vertexID == 0.0) { gl_Position = vec4(-1.0, -1.0, 0.0, 1.0); texCoord = vec2(0.0, 1.0); } else"
    , "  if (vertexID == 1.0) { gl_Position = vec4( 1.0, -1.0, 0.0, 1.0); texCoord = vec2(1.0, 1.0); } else"
    , "  if (vertexID == 2.0) { gl_Position = vec4(-1.0,  1.0, 0.0, 1.0); texCoord = vec2(0.0, 0.0); } else"
    , "                       { gl_Position = vec4( 1.0,  1.0, 0.0, 1.0); texCoord = vec2(1.0, 0.0); }"
    , "}"
    ]

simple_frag = unlines
    [ "#version 130"
    , "uniform sampler2D tex;"
    , "uniform highp float weight;"
    , "in vec2 texCoord;"
    , "out vec4 fragColor;"
    , "// http://lolengine.net/blog/2013/07/27/rgb-to-hsv-in-glsl"
    , "vec3 hsv2rgb(vec3 c) {"
    , "  vec4 K = vec4(1.0, 2.0 / 3.0, 1.0 / 3.0, 3.0);"
    , "  vec3 p = abs(fract(c.xxx + K.xyz) * 6.0 - K.www);"
    , "  return c.z * mix(K.xxx, clamp(p - K.xxx, 0.0, 1.0), c.y);"
    , "}"
    , "void main() {"
    , "  vec2 dx = dFdx(texCoord);"
    , "  vec2 dy = dFdy(texCoord);"
    , "  vec4 me = texture(tex, texCoord);"
    , "  vec4 a = texture(tex, texCoord + dx);"
    , "  vec4 b = texture(tex, texCoord + dy);"
    , "  vec4 c = texture(tex, texCoord + dx + dy);"
    , "  float de = me.x;"
    , "  float s = /*de > 0.0 ? (me.y != c.y || a.y != b.y) ? 1.0 : 0.25 :*/ 0.0;"
    , "  float h = max(max(me.y, c.y), max(a.y, b.y)) / 24.618033988749893;"
    , "  h -= floor(h);"
    , "  float v = de > 0.0 ? tanh(clamp(2.0 + log(de / weight), 0.0, 8.0)) : 0.0;"
    , "  if (isnan(de) || isinf(de)) v = 0.0;"
    , "  bool glitch = de < 0.0;"
    , "  fragColor = vec4(glitch ? vec3(1.0, 0.0, 0.0) : hsv2rgb(vec3(h,s,v)), glitch ? 0.0 : 1.0);"
    , "}"
    ]

doRealize :: IORef GUI -> IO ()
doRealize guiR = do
  gui <- readIORef guiR
```

```
410     gLAreaMakeCurrent $ glarea gui
        e <- gLAreaGetError $ glarea gui
        case e of
          Just err -> return ()
          _ -> do
415         context <- gLAreaGetContext $ glarea gui
            es <- gLContextGetUseEs context
            if es
            then do
              return ()
420         else do
              -- snapshot texture
              ftex <- with 0 $ \p -> glGenTextures 1 p >> peek p
              glActiveTexture GL_TEXTURE1
              glBindTexture GL_TEXTURE_2D ftex
425           glTexImage2D GL_TEXTURE_2D 0 GL_SRGB8_ALPHA8 (fromIntegral (imageWidth ↙
                  ↳ gui)) (fromIntegral (imageHeight gui)) 0 GL_RGBA GL_UNSIGNED_BYTE ↙
                  ↳ nullPtr
              glTexParameteri GL_TEXTURE_2D GL_TEXTURE_MIN_FILTER ↙
                  ↳ GL_LINEAR_MIPMAP_LINEAR
              glTexParameteri GL_TEXTURE_2D GL_TEXTURE_MAG_FILTER GL_LINEAR
              glTexParameteri GL_TEXTURE_2D GL_TEXTURE_WRAP_S GL_CLAMP_TO_EDGE
              glTexParameteri GL_TEXTURE_2D GL_TEXTURE_WRAP_T GL_CLAMP_TO_EDGE
430           glGenerateMipmap GL_TEXTURE_2D
              -- snapshot texture
              ftex2 <- with 0 $ \p -> glGenTextures 1 p >> peek p
              glActiveTexture GL_TEXTURE1
              glBindTexture GL_TEXTURE_2D ftex2
435           glTexImage2D GL_TEXTURE_2D 0 GL_SRGB8_ALPHA8 (fromIntegral (imageWidth ↙
                  ↳ gui)) (fromIntegral (imageHeight gui)) 0 GL_RGBA GL_UNSIGNED_BYTE ↙
                  ↳ nullPtr
              glTexParameteri GL_TEXTURE_2D GL_TEXTURE_MIN_FILTER ↙
                  ↳ GL_LINEAR_MIPMAP_LINEAR
              glTexParameteri GL_TEXTURE_2D GL_TEXTURE_MAG_FILTER GL_LINEAR
              glTexParameteri GL_TEXTURE_2D GL_TEXTURE_WRAP_S GL_CLAMP_TO_EDGE
              glTexParameteri GL_TEXTURE_2D GL_TEXTURE_WRAP_T GL_CLAMP_TO_EDGE
440           glGenerateMipmap GL_TEXTURE_2D
              -- snapshot fbo
              old_fbo <- with 0 $ \p -> glGetIntegerv GL_DRAW_FRAMEBUFFER_BINDING p >>↙
                  ↳  peek p
              fbo <- with 0 $ \p -> glGenFramebuffers 1 p >> peek p
              glBindFramebuffer GL_DRAW_FRAMEBUFFER fbo
445           glFramebufferTexture2D GL_DRAW_FRAMEBUFFER GL_COLOR_ATTACHMENT0 ↙
                  ↳ GL_TEXTURE_2D ftex2 0
              glClearColor 0 0 0 0
              glClear GL_COLOR_BUFFER_BIT
              glBindFramebuffer GL_DRAW_FRAMEBUFFER (fromIntegral old_fbo)
              glBindTexture GL_TEXTURE_2D 0
450           -- snapshot fbo
              fbo2 <- with 0 $ \p -> glGenFramebuffers 1 p >> peek p
              glBindFramebuffer GL_DRAW_FRAMEBUFFER fbo2
              glFramebufferTexture2D GL_DRAW_FRAMEBUFFER GL_COLOR_ATTACHMENT0 ↙
                  ↳ GL_TEXTURE_2D ftex 0
              glClearColor 0 0 0 0
455           glClear GL_COLOR_BUFFER_BIT
              glBindFramebuffer GL_DRAW_FRAMEBUFFER (fromIntegral old_fbo)
              glBindTexture GL_TEXTURE_2D 0
```

```
        -- image texture
        tex <- with 0 $ \p -> glGenTextures 1 p >> peek p
        glActiveTexture GL_TEXTURE0
        glBindTexture GL_TEXTURE_2D tex
        glTexImage2D GL_TEXTURE_2D 0 GL_RGBA32F (fromIntegral $ imageWidth gui) ↙
            ↳ (fromIntegral $ imageHeight gui) 0 GL_RGBA GL_FLOAT nullPtr
        glTexParameteri GL_TEXTURE_2D GL_TEXTURE_MIN_FILTER GL_NEAREST
        glTexParameteri GL_TEXTURE_2D GL_TEXTURE_MAG_FILTER GL_NEAREST
        glTexParameteri GL_TEXTURE_2D GL_TEXTURE_WRAP_S GL_MIRRORED_REPEAT
        glTexParameteri GL_TEXTURE_2D GL_TEXTURE_WRAP_T GL_MIRRORED_REPEAT
        glBindTexture GL_TEXTURE_2D 0
        vbo <- with 0 $ \p -> glGenBuffers 1 p >> peek p
        glBindBuffer GL_ARRAY_BUFFER vbo
        withArray [0 .. 3 :: Word8] $ \p -> glBufferData GL_ARRAY_BUFFER 4 (↙
            ↳ castPtr p) GL_STATIC_DRAW
        -- blit shader
        blit_vao <- with 0 $ \p -> glGenVertexArrays 1 p >> peek p
        glBindVertexArray blit_vao
        blit_p <- compile_program "blit" blit_vert blit_frag
        glUseProgram blit_p
        withCString "tex" $ \s -> glGetUniformLocation blit_p s >>= \l -> ↙
            ↳ glUniform1i l 1
        bound_u <- withCString "bounds" $ \s -> glGetUniformLocation blit_p s
        blit_u_gamma <- withCString "gamma" $ \s -> glGetUniformLocation blit_p ↙
            ↳ s
        glVertexAttribPointer 0 1 GL_UNSIGNED_BYTE GL_FALSE 0 nullPtr
        glEnableVertexAttribArray 0
        glBindVertexArray 0
        glUseProgram 0
        -- colourize shader
        colourize_vao <- with 0 $ \p -> glGenVertexArrays 1 p >> peek p
        glBindVertexArray colourize_vao
        colourize_p <- compile_program "colourize" simple_vert simple_frag
        glUseProgram colourize_p
        weight_u <- withCString "weight" $ \s -> glGetUniformLocation ↙
            ↳ colourize_p s
        glVertexAttribPointer 0 1 GL_UNSIGNED_BYTE GL_FALSE 0 nullPtr
        glEnableVertexAttribArray 0
        glBindVertexArray 0
        glUseProgram 0
        atomicModifyIORef guiR (\g -> (g{ gl = GUIGL{..} }, ()))
        return ()

doUnrealize :: IORef GUI -> IO ()
doUnrealize guiR = do
  gui <- readIORef guiR
  gLAreaMakeCurrent $ glarea gui
  e <- gLAreaGetError $ glarea gui
  case e of
    Just err -> return ()
    _ -> do
      return ()

doRender :: IORef GUI -> GLContext -> IO Bool
doRender guiR ctx = do
  gui@GUI{ gl = GUIGL{..} } <- readIORef guiR
  e <- gLAreaGetError $ glarea gui
```

```
510     case e of
          Just err -> return False
          _ -> do
            old_fbo <- with 0 $ \p -> glGetIntegerv GL_DRAW_FRAMEBUFFER_BINDING p >> ↙
                ↳ peek p
            glBindFramebuffer GL_DRAW_FRAMEBUFFER fbo
515         glViewport 0 0 (fromIntegral $ imageWidth gui) (fromIntegral $ imageHeight↙
                ↳  gui)

            glClearColor 0 0 0 0
            glClear GL_COLOR_BUFFER_BIT
            glDisable GL_DEPTH_TEST
520         glEnable GL_FRAMEBUFFER_SRGB
            glEnable GL_BLEND
            glBlendFunc GL_SRC_ALPHA GL_ONE_MINUS_SRC_ALPHA

            glBindVertexArray blit_vao
525         glUseProgram blit_p
            glUniform1i blit_u_gamma 0
            glActiveTexture GL_TEXTURE1
            glBindTexture GL_TEXTURE_2D ftex
            let w = fromIntegral (imageWidth gui)
530             h = fromIntegral (imageHeight gui)
            glUniform4f bound_u (realToFrac $ srcx0 gui / w) (realToFrac $ srcy0 gui /↙
                ↳  h) (realToFrac $ srcx1 gui / w) (realToFrac $ srcy1 gui / h)
            glDrawArrays GL_TRIANGLE_STRIP 0 4
            glBindTexture GL_TEXTURE_2D 0
            glBindVertexArray 0
535         glUseProgram 0

            glBindVertexArray colourize_vao
            glUseProgram colourize_p
            glActiveTexture GL_TEXTURE0
540         glBindTexture GL_TEXTURE_2D tex
            unsafeWith (contents $ renderBuffer gui) $ \ptr ->
              glTexSubImage2D GL_TEXTURE_2D 0 0 0 (fromIntegral $ imageWidth gui) (↙
                  ↳ fromIntegral $ imageHeight gui) GL_RGBA GL_FLOAT (castPtr ptr)
            glUniform1f weight_u (realToFrac $ 2 ** log2weight gui)
            glDrawArrays GL_TRIANGLE_STRIP 0 4
545         glBindTexture GL_TEXTURE_2D 0
            glBindVertexArray 0
            glUseProgram 0

            glActiveTexture GL_TEXTURE1
550         glBindTexture GL_TEXTURE_2D ftex2
            glGenerateMipmap GL_TEXTURE_2D

            glBindFramebuffer GL_FRAMEBUFFER (fromIntegral old_fbo)
            glBindVertexArray blit_vao
555         glUseProgram blit_p
            glUniform1i blit_u_gamma 1
            glActiveTexture GL_TEXTURE1
            glBindTexture GL_TEXTURE_2D ftex
            let w = round $ displaySize gui * (fromIntegral $ imageWidth gui)
560             h = round $ displaySize gui * (fromIntegral $ imageHeight gui)
            glViewport 0 0 w h
            glUniform4f bound_u 0 0 1 1
```

```
        glDrawArrays GL_TRIANGLE_STRIP 0 4
        glBindTexture GL_TEXTURE_2D 0
        glBindVertexArray 0
        glUseProgram 0

        atomicModifyIORef guiR $ \g -> (g
          { srcx0 = 0
          , srcx1 = fromIntegral (imageWidth gui)
          , srcy0 = 0
          , srcy1 = fromIntegral (imageHeight gui)
          , gl = (gl g)
            { fbo = fbo2
            , fbo2 = fbo
            , ftex = ftex2
            , ftex2 = ftex
            }
          }, ())
        glFlush
        e <- glGetError
        when (e /= 0) $ hPutStrLn stderr $ "OpenGL error " ++ show e
        return True

fromDouble' :: Double -> Rounded
fromDouble' = fromDouble TowardNearest 53

doZoom :: IORef GUI -> Double -> Double -> Double -> Double -> IO ()
doZoom guiR ex ey f g = do
  gui <- readIORef guiR
  let u = (ex / (displaySize gui * w) - 0.5) * 2 * w / h
      v = (ey / (displaySize gui * h) - 0.5) * 2
      vv = 1 - v
      (s, t) = apply (transformation gui) (u, v)
      dx = radius (location gui) .* fromDouble' s
      dy = radius (location gui) .* fromDouble' t
      r = radius (location gui) .* fromDouble' f
      prec = max 53 $ 53 - exponent' r
      (.+) = add_ TowardNearest prec
      (.-) = sub_ TowardNearest 53
      (.*) = mul_ TowardNearest 53
      cx = centerX (location gui) .+ (fromDouble' (1 - g) .* dx)
      cy = centerY (location gui) .+ (fromDouble' (1 - g) .* dy)
      x = ex / displaySize gui
      y = ey / displaySize gui
      yy = h - y
      w = fromIntegral (imageWidth gui)
      h = fromIntegral (imageHeight gui)
      (srcx0', srcx1', srcy0', srcy1')
        | g /= 0.0 = (g * (0 - x) + x, g * (w - x) + x, g * (0 - yy) + yy, g * (↙
            ↳ h - yy) + yy)
        | otherwise = (0 - (w / 2 - x), w - (w / 2 - x), 0 - (h / 2 - yy), h - (↙
            ↳ h / 2 - yy))
  atomicModifyIORef guiR $ \g -> (g{ location = View cx cy r, srcx0 = srcx0', ↙
      ↳ srcx1 = srcx1', srcy0 = srcy0', srcy1 = srcy1' }, ())

doNewton :: IORef GUI -> Bool -> IO ()
doNewton guiR julia = do
  cancelRender guiR
```

```
    gui <- readIORef guiR
    runningP <- malloc
    poke runningP 1
    a <- async $ catchAny (do
      let View vx vy vr = location gui
          d_ = 1
          e_ = 0
      mp <- period_jsk (formula gui) (iterations gui) (formulaER gui * formulaER ↙
          ↳ gui) d_ e_ vx vy vr (getTransform $ transformation gui) runningP
      case mp of
        Just p | p > 0 -> do
          print ("period", p)
          let iprec = precision vx * 4
              a0 = precRound TowardNearest iprec vx
              b0 = precRound TowardNearest iprec vy
              maxSteps = 16
          mn <- newton (formula gui) maxSteps p d_ e_ a0 b0 runningP
          case mn of
            Just (R2 a b) -> do
              print ("newton", show' a, show' b)
              ms <- size (formula gui) p d_ e_ a b runningP
              case ms of
                Just (s0, k@(a11, a12, a21, a22)) -> do
                  print ("size", show' s0)
                  print ("skew", k)
                  let pow_ r p x y = exp_ r p (mul_ r p y (log_ r p x))
                      s | julia = pow_ TowardNearest 53 s0 (fromDouble' d)
                        | otherwise = precRound TowardNearest 53 s0
                      n = degree (formula gui)
                      d = (n + 1) * (n - 1) / (n * n)
                      ur = mul_ TowardNearest 53 s (fromDouble' 4)
                      sprec = max 53 (53 - exponent' ur)
                      ux = precRound TowardNearest sprec a
                      uy = precRound TowardNearest sprec b
                      view = View ux uy ur
                  case transform a11 a12 a21 a22 of
                    Just t -> do
                      print ("transform", getTransform t)
                      threadsAddIdle PRIORITY_DEFAULT $ do
                        atomicModifyIORef guiR $ \g -> (g{ location = view, ↙
                            ↳ transformation = t }, ())
                        startRender guiR
                        return False
                      return True
                    _ -> return False
                _ -> return False
            _ -> return False
        _ -> return False
      ) (\_ -> return False)
    atomicModifyIORef guiR $ \g -> (g{ cancel = do{ poke runningP 0 ; r <- wait a ↙
        ↳ ; free runningP ; return r } }, ())

doSkew :: IORef GUI -> Bool -> Bool -> IO ()
doSkew guiR useDZ useIT = do
  cancelRender guiR
  gui <- readIORef guiR
  runningP <- malloc
```

```
  poke runningP 1
  a <- async $ catchAny (do
    let View vx vy vr = location gui
        d_ = 1
        e_ = 0
    ms <- skew (formula gui) (iterations gui) {-(formulaER gui * formulaER gui)↙
        ↳ -} d_ e_ vx vy useDZ runningP
    case ms of
      Just k@(a11, a12, a21, a22) -> do
        print ("skew", k, a11 * a22 - a12 * a21)
        case if useIT then transform a11 a21 a12 a22 else transform a11 a12 a21 ↙
            ↳ a22 of
          Just t -> do
            print ("transform", getTransform t)
            threadsAddIdle PRIORITY_DEFAULT $ do
              atomicModifyIORef guiR $ \g -> (g{ transformation = t }, ())
              startRender guiR
              return False
            return True
          _ -> return False
      _ -> return False
  ) (\_ -> return False)
  atomicModifyIORef guiR $ \g -> (g{ cancel = do{ poke runningP 0 ; r <- wait a ↙
      ↳ ; free runningP ; return r } }, ())

doRotate :: IORef GUI -> Double -> IO ()
doRotate guiR theta = do
  cancelRender guiR
  gui <- readIORef guiR
  let c = cos theta
      s = sin theta
      (a11, a12, a21, a22) = getTransform (transformation gui)
      Just b = transform (c * a11 + s *  a12) (-s * a11 + c * a12) (c * a21 + s ↙
          ↳ * a22) (-s * a21 + c * a22)
  print ("transform", getTransform b)
  atomicModifyIORef guiR $ \g -> (g{ transformation = b }, ())

imageSaveAs :: IORef GUI -> Window -> IO ()
imageSaveAs guiR w = do
  c <- fileChooserNativeNew (Just "Save Image") (Just w) FileChooserActionSave ↙
      ↳ Nothing Nothing
  fileChooserSetDoOverwriteConfirmation c True
  f <- fileFilterNew
  fileFilterAddPattern f "*.png"
  fileFilterSetName f (Just "PNG Image")
  fileChooserAddFilter c f
  setNativeDialogModal c True
  r <- nativeDialogRun c
  when (r == fromIntegral (fromEnum ResponseTypeAccept)) $ do
    mpath <- fileChooserGetFilename c
    case mpath of
      Nothing -> return ()
      Just path -> doImageSave guiR path
  nativeDialogDestroy c

doImageSave :: IORef GUI -> String -> IO ()
doImageSave guiR path = do
```

```
    gui <- readIORef guiR
    fptr <- mallocForeignPtrBytes (imageWidth gui * imageHeight gui * 3)
    context <- gLAreaGetContext $ glarea gui
    gLContextMakeCurrent context
    glBindFramebuffer GL_READ_FRAMEBUFFER (fbo2 (gl gui))
    glPixelStorei GL_PACK_ALIGNMENT 1
    withForeignPtr fptr $ \ptr ->
      glReadPixels 0 0 (fromIntegral $ imageWidth gui) (fromIntegral $ imageHeight↙
          ↪ gui) GL_RGB GL_UNSIGNED_BYTE (castPtr ptr)
    gLContextClearCurrent
    let img :: J.Image J.PixelRGB8
        img = J.flipVertically $ J.imageFromUnsafePtr (imageWidth gui) (↙
            ↪ imageHeight gui) fptr
        View a b r = location gui
        comment = M.toString (Metadata
          (formulaName gui : formulaSource gui)
          (formulaP gui)
          (formulaQ gui)
          1
          0
          a
          b
          r
          (getTransform $ transformation gui)
          (iterations gui)
          (2 ** log2weight gui) )
        meta = J.insert J.ColorSpace J.SRGB . J.insert J.Comment comment $ J.↙
            ↪ singleton J.Software "et"
        png = J.encodePngWithMetadata meta img
    BS.writeFile path png

fileOpen :: IORef GUI -> Window -> IO ()
fileOpen guiR w = do
    c <- fileChooserNativeNew (Just "Open File") (Just w) FileChooserActionOpen ↙
        ↪ Nothing Nothing
    f <- fileFilterNew
    fileFilterAddPattern f "*.png"
    fileFilterSetName f (Just "PNG Image")
    fileChooserAddFilter c f
    setNativeDialogModal c True
    r <- nativeDialogRun c
    when (r == fromIntegral (fromEnum ResponseTypeAccept)) $ do
      mpath <- fileChooserGetFilename c
      case mpath of
        Nothing -> return ()
        Just path -> doImageLoadMeta guiR path
    nativeDialogDestroy c

doImageLoadMeta :: IORef GUI -> String -> IO ()
doImageLoadMeta guiR path = do
    raw <- BS.readFile path
    case (J.lookup J.Comment . J.extractMetadatas . B.decode $ raw) >>= M.↙
        ↪ fromString of
      Nothing -> return ()
      Just m -> do
        gui <- readIORef guiR
        e <- compileAndLoad (tmpDir gui) (unlines (metadataFormula m)) (metadataP ↙
```

```
            ↳ m) (metadataQ m)
775       case e of
            Left msg -> hPutStrLn stderr msg >> return ()
            Right et -> do
              cancelRender guiR
              old_et <- atomicModifyIORef guiR $ \g -> (g
780                 { formulaName = head (metadataFormula m) -- FIXME
                    , formulaSource = tail (metadataFormula m) -- FIXME
                    , formulaP = metadataP m
                    , formulaQ = metadataQ m
                    , location = View (metadataA m) (metadataB m) (metadataR m)
785                 , transformation = case (case metadataT m of (a, b, c, d) -> ↙
                      ↳ transform a b c d) of Just t -> t -- FIXME
                    , iterations = metadataN m
                    , log2weight = logBase 2 $ metadataDEWeight m
                    , formula = et
                   }, formula g)
790           close old_et
              startRender guiR
              return ()

    imageSize :: IORef GUI -> Window -> IO ()
795 imageSize guiR mainWindow = do
      gui <- readIORef guiR
      dialog <- dialogNew
      windowSetTransientFor dialog (Just mainWindow)
      dialogAddButton dialog "_Ok" (fromIntegral (fromEnum ResponseTypeAccept))
800   dialogAddButton dialog "_Cancel" (fromIntegral (fromEnum ResponseTypeReject))
      box <- dialogGetContentArea dialog
      grid <- gridNew
      eWidth  <- entryNew
      eHeight <- entryNew
805   eSize   <- entryNew
      set eWidth  [ #inputPurpose := InputPurposeNumber, #widthChars := 6, #text := ↙
          ↳ T.pack (show (imageWidth gui)) ]
      set eHeight [ #inputPurpose := InputPurposeNumber, #widthChars := 6, #text := ↙
          ↳ T.pack (show (imageHeight gui)) ]
      set eSize   [ #inputPurpose := InputPurposeNumber, #widthChars := 6, #text := ↙
          ↳ T.pack (show (displaySize gui)) ]
      lWidth  <- labelNewWithMnemonic $ Just "Image _Width"
810   lHeight <- labelNewWithMnemonic $ Just "Image _Height"
      lSize   <- labelNewWithMnemonic $ Just "Display _Size"
      labelSetMnemonicWidget lWidth  $ Just eWidth
      labelSetMnemonicWidget lHeight $ Just eHeight
      labelSetMnemonicWidget lSize   $ Just eSize
815   gridAttach grid lWidth  0 0 1 1
      gridAttach grid eWidth  1 0 1 1
      gridAttach grid lHeight 0 1 1 1
      gridAttach grid eHeight 1 1 1 1
      gridAttach grid lSize   0 2 1 1
820   gridAttach grid eSize   1 2 1 1
      containerAdd box grid
      widgetShowAll dialog
      r <- dialogRun dialog
      when (r == fromIntegral (fromEnum ResponseTypeAccept)) $ do
825     tw <- entryGetText eWidth
        th <- entryGetText eHeight
```

```
      ts <- entryGetText eSize
      case readMaybe (T.unpack tw) of
        Just w | w > 0 -> case readMaybe (T.unpack th) of
          Just h | h > 0 -> case readMaybe (T.unpack ts) of
            Just s | s > 0 -> doImageSize guiR w h s
            _ -> return ()
          _ -> return ()
        _ -> return ()
    widgetDestroy dialog

doImageSize :: IORef GUI -> Int -> Int -> Double -> IO ()
doImageSize guiR iw ih s = do
  gui <- readIORef guiR
  when (imageWidth gui /= iw || imageHeight gui /= ih) $ do
    -- stop rendering
    cancelRender guiR
    gui <- readIORef guiR
    -- resize textures
    context <- gLAreaGetContext $ glarea gui
    gLContextMakeCurrent context
    glActiveTexture GL_TEXTURE1
    glBindTexture GL_TEXTURE_2D (ftex (gl gui))
    glTexImage2D GL_TEXTURE_2D 0 GL_SRGB8_ALPHA8 (fromIntegral iw) (fromIntegral↙
        ↪ ih) 0 GL_RGBA GL_UNSIGNED_BYTE nullPtr
    glBindTexture GL_TEXTURE_2D (ftex2 (gl gui))
    glTexImage2D GL_TEXTURE_2D 0 GL_SRGB8_ALPHA8 (fromIntegral iw) (fromIntegral↙
        ↪ ih) 0 GL_RGBA GL_UNSIGNED_BYTE nullPtr
    glBindTexture GL_TEXTURE_2D 0
    glActiveTexture GL_TEXTURE0
    glBindTexture GL_TEXTURE_2D (tex (gl gui))
    glTexImage2D GL_TEXTURE_2D 0 GL_RGBA32F (fromIntegral iw) (fromIntegral ih) ↙
        ↪ 0 GL_RGBA GL_FLOAT nullPtr
    glBindTexture GL_TEXTURE_2D 0
    -- update renderer
    buf <- buffer iw ih 4 (-1)
    mprog <- cacheProgressive iw ih
    atomicModifyIORef guiR $ \g -> (g
      { imageWidth = iw
      , imageHeight = ih
      , renderBuffer = buf
      , progress = mprog
      , srcx0 = 0
      , srcy0 = 0
      , srcx1 = fromIntegral iw
      , srcy1 = fromIntegral ih
      }, ())
  -- resize GUI
  gl <- atomicModifyIORef guiR $ \g -> (g
    { displaySize = s
    }, glarea g)
  let ww = round (s * fromIntegral iw)
      wh = round (s * fromIntegral ih)
  widgetSetSizeRequest gl ww wh
  -- restart rendering
  when (imageWidth gui /= iw || imageHeight gui /= ih) $ do
    startRender guiR

```

```
supersampling :: IORef GUI -> Double -> IO ()
supersampling guiR desiredSupersampling = do
  gui <- readIORef guiR
  let currentDisplaySize  = displaySize $ gui
      currentWindowWidth  = (currentDisplaySize *) . fromIntegral . imageWidth  ↙
          ↳ $ gui
      currentWindowHeight = (currentDisplaySize *) . fromIntegral . imageHeight ↙
          ↳ $ gui
      desiredDisplaySize  = recip desiredSupersampling
      desiredImageWidth   = round $ currentWindowWidth  * desiredSupersampling
      desiredImageHeight  = round $ currentWindowHeight * desiredSupersampling
  doImageSize guiR desiredImageWidth desiredImageHeight desiredDisplaySize

formulaUI :: IORef GUI -> Window -> IO ()
formulaUI guiR mainWindow = do
  gui <- readIORef guiR
  window <- windowNew WindowTypeToplevel
  windowSetTransientFor window (Just mainWindow)
  windowSetModal window True
  grid <- gridNew
  eName <- entryNew
  eSource <- textViewNew
  eP <- entryNew
  eQ <- entryNew
  eMessage <- textViewNew
  set eName [ #text := T.pack (formulaName gui) ]
  set eSource [ #editable := True ]
  sBuf <- get eSource #buffer
  set sBuf [ #text := T.pack (unlines (formulaSource gui)) ]
  set eP [ #text := T.pack (show $ formulaP gui), #inputPurpose := ↙
      ↳ InputPurposeNumber ]
  set eQ [ #text := T.pack (show $ formulaQ gui), #inputPurpose := ↙
      ↳ InputPurposeNumber ]
  set eMessage [ #editable := False ]
  mBuf <- get eMessage #buffer
  set mBuf [ #text := "..." ]
  lName <- labelNewWithMnemonic $ Just "_Name"
  lSource <- labelNewWithMnemonic $ Just "_Source"
  lP <- labelNewWithMnemonic $ Just "_P"
  lQ <- labelNewWithMnemonic $ Just "_Q"
  bRandom <- buttonNewWithMnemonic "_Random"
  bCompile <- buttonNewWithMnemonic "_Compile"
  lMessage <- labelNew $ Just "Messages"
  labelSetMnemonicWidget lName $ Just eName
  labelSetMnemonicWidget lSource $ Just eSource
  labelSetMnemonicWidget lP $ Just eP
  labelSetMnemonicWidget lQ $ Just eQ
  gridAttach grid lName    0 0 1 1
  gridAttach grid eName    1 0 1 1
  gridAttach grid lSource  0 1 1 1
  gridAttach grid eSource  1 1 1 1
  gridAttach grid lP       0 2 1 1
  gridAttach grid eP       1 2 1 1
  gridAttach grid lQ       0 3 1 1
  gridAttach grid eQ       1 3 1 1
  gridAttach grid bRandom  1 4 1 1
  gridAttach grid bCompile 1 5 1 1
```

```
    gridAttach grid lMessage 0 6 1 1
935 gridAttach grid eMessage 1 6 1 1
    #add window grid

    on bRandom #clicked $ do
      let stanza = do
940         x <- pure "x"
            y <- pure "y"
            b <- randomIO
            (x, y) <- pure $ if b then (y, x) else (x, y)
            b <- randomIO
945         x <- pure $ if b then "|" ++ x ++ "|" else x
            b <- randomIO
            y <- pure $ if b then "|" ++ y ++ "|" else y
            b <- randomIO
            x <- pure $ if b then "(-" ++ x ++ ")" else x
950         b <- randomIO
            y <- pure $ if b then "(-" ++ y ++ ")" else y
            pure $ "z := (" ++ x ++ " + i " ++ y ++ ")^p + c"
      n <- randomRIO (1, 4)
      s <- unlines <$> replicateM n stanza
955   ebuf <- get eSource #buffer
      set ebuf [ #text := T.pack s ]

    on bCompile #clicked $ do
      ebuf <- get eMessage #buffer
960   set ebuf [ #text := "...compiling..." ]
      name <- T.unpack <$> entryGetText eName
      sbuf <- get eSource #buffer
      msource <- get sbuf #text
      let source = case msource of
965                  Nothing -> []
                     Just t -> lines (T.unpack t)
      sp <- T.unpack <$> entryGetText eP
      sq <- T.unpack <$> entryGetText eQ
      gui <- readIORef guiR
970   case readMaybe sp of
        Just p -> case readMaybe sq of
          Just q -> do
           set bCompile [ #sensitive := False ]
           _ <- forkIO $ do
975         e <- compileAndLoad (tmpDir gui) (unlines (name:source)) p q
            toGTK $ do
             set bCompile [ #sensitive := True ]
             case e of
              Left msg -> do
980             set ebuf [ #text := T.pack msg ]
              Right et -> do
                cancelRender guiR
                old_et <- atomicModifyIORef guiR $ \g -> (g
                  { formulaName = name
985               , formulaSource = source
                  , formulaP = p
                  , formulaQ = q
                  , formula = et
                  }, formula g)
990             close old_et
```

```
                set ebuf [ #text := "compiled" ]
                startRender guiR
            return ()
          _ -> do
995         set ebuf [ #text := "failed to parse number: q" ]
        _ -> do
          set ebuf [ #text := "failed to parse number: p" ]
    widgetShowAll window

1000 about :: Window -> IO ()
    about w = do
      a <- new AboutDialog []
      agpl30 <- getDataFileName "LICENSE.md" >>= T.readFile
      set a
1005    [ #authors := ["Claude Heiland-Allen <claude@mathr.co.uk>"]
        , #comments := "Escape-time fractals."
        , #copyright := "(c) 2018,2019 Claude Heiland-Allen"
    --    , #licenseType := LicenseCustom -- LicenseAgpl30 -- needs too-recent GTK ↙
        ↳ library
        , #license := agpl30
1010    , #programName := "et"
        , #version := "0~git"
        , #website := "https://mathr.co.uk/et"
        , #websiteLabel := "mathr.co.uk/et"
        ]
1015  windowSetTransientFor a (Just w)
      dialogRun a
      widgetDestroy a
      return ()

1020 main :: IO ()
    main = do
      Gtk.init Nothing

      egui <- defaultGUI
1025  gui <- case egui of
        Left msg -> hPutStrLn stderr msg >> exitFailure
        Right gui -> return gui
      guiR <- newIORef gui

1030  getDataFileName "et.png" >>= windowSetDefaultIconFromFile
      window <- windowNew WindowTypeToplevel
      box <- boxNew OrientationVertical 0
      #add window box
      menuBar <- createMenuBar (menuBarDescr guiR window)
1035  setWindowTitle window "et"
      #add box menuBar

      on window #keyPressEvent $ \event -> do
        name <- event `get` #keyval >>= keyvalName
1040    case name of
          Just "Escape" -> cancelRender guiR >> return True
          Just "F5" -> startRender guiR >> return True
          Just "s" -> do
            (prefix, seqNum) <- atomicModifyIORef guiR $ \g -> (g{ sequenceNumber = ↙
                ↳ sequenceNumber g + 1 }, (imagePrefix g, sequenceNumber g))
1045        let seqStr = reverse . take 4 . (++ "0000") . reverse . show $ seqNum
```

```
            path = prefix ++ seqStr ++ ".png"
            picturesDir = takeDirectory path
          createDirectoryIfMissing True picturesDir
          doImageSave guiR path
          return True
        Just "1" -> supersampling guiR 1 >> return True
        Just "2" -> supersampling guiR 2 >> return True
        Just "3" -> supersampling guiR 3 >> return True
        Just "4" -> supersampling guiR 4 >> return True
        Just "5" -> supersampling guiR 5 >> return True
        Just "6" -> supersampling guiR 6 >> return True
        Just "7" -> supersampling guiR 7 >> return True
        Just "8" -> supersampling guiR 8 >> return True
        Just "equal" -> do
          atomicModifyIORef guiR $ \g -> (g{ iterations = min (bit 30) (iterations g * 2) }, ())
          startRender guiR
          return True
        Just "minus" -> do
          atomicModifyIORef guiR $ \g -> (g{ iterations = max 1 (iterations g `div` 2) }, ())
          startRender guiR
          return True
        Just "Home" -> do
          atomicModifyIORef guiR $ \g -> (g{ location = defaultView }, ())
          startRender guiR
          return True
        Just "Page_Up" -> do
          gui <- readIORef guiR
          let ex = displaySize gui * fromIntegral (imageWidth gui) / 2
              ey = displaySize gui * fromIntegral (imageHeight gui) / 2
          doZoom guiR ex ey 0.5 0.5
          startRender guiR
          return True
        Just "Page_Down" -> do
          gui <- readIORef guiR
          let ex = displaySize gui * fromIntegral (imageWidth gui) / 2
              ey = displaySize gui * fromIntegral (imageHeight gui) / 2
          doZoom guiR ex ey 2.0 2.0
          startRender guiR
          return True
        Just "KP_Subtract" -> do
          gui <- readIORef guiR
          let ex = displaySize gui * fromIntegral (imageWidth gui) / 2
              ey = displaySize gui * fromIntegral (imageHeight gui) / 2
          doZoom guiR ex ey (0.5**0.1) (0.5**0.1)
          startRender guiR
          return True
        Just "KP_Add" -> do
          gui <- readIORef guiR
          let ex = displaySize gui * fromIntegral (imageWidth gui) / 2
              ey = displaySize gui * fromIntegral (imageHeight gui) / 2
          doZoom guiR ex ey (2.0**0.1) (2.0**0.1)
          startRender guiR
          return True
        Just "Left" -> do
          gui <- readIORef guiR
```

```
          let ex = displaySize gui * fromIntegral (imageWidth gui) / 2
              ey = displaySize gui * fromIntegral (imageHeight gui) / 2
          doZoom guiR (-ex) ey 1.0 0.0
          startRender guiR
          return True
        Just "Right" -> do
          gui <- readIORef guiR
          let ex = displaySize gui * fromIntegral (imageWidth gui) / 2
              ey = displaySize gui * fromIntegral (imageHeight gui) / 2
          doZoom guiR (3 * ex) ey 1.0 0.0
          startRender guiR
          return True
        Just "Up" -> do
          gui <- readIORef guiR
          let ex = displaySize gui * fromIntegral (imageWidth gui) / 2
              ey = displaySize gui * fromIntegral (imageHeight gui) / 2
          doZoom guiR ex (-ey) 1.0 0.0
          startRender guiR
          return True
        Just "Down" -> do
          gui <- readIORef guiR
          let ex = displaySize gui * fromIntegral (imageWidth gui) / 2
              ey = displaySize gui * fromIntegral (imageHeight gui) / 2
          doZoom guiR ex (3 * ey) 1.0 0.0
          startRender guiR
          return True
        Just "KP_4" -> do
          gui <- readIORef guiR
          let ex = displaySize gui * fromIntegral (imageWidth gui) / 2
              ey = displaySize gui * fromIntegral (imageHeight gui) / 2
          doZoom guiR (ex-ex/10) ey 1.0 0.0
          startRender guiR
          return True
        Just "KP_6" -> do
          gui <- readIORef guiR
          let ex = displaySize gui * fromIntegral (imageWidth gui) / 2
              ey = displaySize gui * fromIntegral (imageHeight gui) / 2
          doZoom guiR (ex+ex/10) ey 1.0 0.0
          startRender guiR
          return True
        Just "KP_8" -> do
          gui <- readIORef guiR
          let ex = displaySize gui * fromIntegral (imageWidth gui) / 2
              ey = displaySize gui * fromIntegral (imageHeight gui) / 2
          doZoom guiR ex (ey-ey/10) 1.0 0.0
          startRender guiR
          return True
        Just "KP_2" -> do
          gui <- readIORef guiR
          let ex = displaySize gui * fromIntegral (imageWidth gui) / 2
              ey = displaySize gui * fromIntegral (imageHeight gui) / 2
          doZoom guiR ex (ey+ey/10) 1.0 0.0
          startRender guiR
          return True
        Just "j" -> doNewton guiR True  >> return True
        Just "k" -> doSkew   guiR False False >> return True
        Just "l" -> doSkew   guiR True  False >> return True
```

```
        Just "semicolon" ->
                    doSkew   guiR False True  >> return True
        Just "apostrophe" ->
                    doSkew   guiR True  True  >> return True
        Just "m" -> doNewton guiR False >> return True
        Just "u" -> do
          atomicModifyIORef guiR $ \g -> (g{ transformation = identity }, ())
          startRender guiR
          return True
        Just "9" -> do
          atomicModifyIORef guiR $ \g -> (g{ log2weight = log2weight g - 0.25 }, ↙
              ↳ ())
          return True
        Just "0" -> do
          atomicModifyIORef guiR $ \g -> (g{ log2weight = log2weight g + 0.25 }, ↙
              ↳ ())
          return True
        Just "comma" -> do
          doRotate guiR (-5 * pi / 180)
          startRender guiR
          return True
        Just "period" -> do
          doRotate guiR ( 5 * pi / 180)
          startRender guiR
          return True
        Just n -> print ("unhandled key", n) >> return False
        _ -> return False

    gui <- readIORef guiR
    glarea' <- new GLArea []
    set glarea' [ #autoRender := False, #hasAlpha := True ]
    widgetSetSizeRequest glarea' (round (displaySize gui * fromIntegral (↙
        ↳ imageWidth gui))) (round (displaySize gui * fromIntegral (imageHeight ↙
        ↳ gui)))
    atomicModifyIORef guiR $ \g -> (g{ glarea = glarea'}, ())
    eventBox <- new EventBox []
    set eventBox [ #aboveChild := True ]
    widgetAddEvents eventBox [ EventMaskScrollMask, EventMaskPointerMotionMask ]
    #add eventBox glarea'
    #add box eventBox
    on glarea' #realize $ doRealize guiR
    on glarea' #unrealize $ doUnrealize guiR
    on glarea' #render $ doRender guiR

    on eventBox #scrollEvent $ \event -> do
      ex <- event `get` #x
      ey <- event `get` #y
      d <- event `get` #direction
      case d of
        ScrollDirectionUp   -> doZoom guiR ex ey 0.5 0.5 >> startRender guiR >> ↙
            ↳ return True
        ScrollDirectionDown -> doZoom guiR ex ey 2.0 2.0 >> startRender guiR >> ↙
            ↳ return True
        _ -> return False

    on eventBox #buttonPressEvent $ \event -> do
      b <- event `get` #button
```

```
      ex <- event `get` #x
      ey <- event `get` #y
      case b of
        1 -> doZoom guiR ex ey 0.5 0.5 >> startRender guiR >> return True
        2 -> doZoom guiR ex ey 1.0 0.0 >> startRender guiR >> return True
        3 -> doZoom guiR ex ey 2.0 2.0 >> startRender guiR >> return True
        _ -> return False

    statusBar <- boxNew OrientationHorizontal 5
    lmousexl <- labelNew $ Just "X"
    lmousex <- labelNew Nothing
    lmouseyl <- labelNew $ Just "Y"
    lmousey <- labelNew Nothing
    ldistancel <- labelNew $ Just "D"
    ldistance <- labelNew Nothing
    lperiodl <- labelNew $ Just "P"
    lperiod <- labelNew Nothing
    literationl <- labelNew $ Just "N"
    literation <- labelNew Nothing
    #add statusBar lmousexl
    #add statusBar lmousex
    #add statusBar lmouseyl
    #add statusBar lmousey
    #add statusBar ldistancel
    #add statusBar ldistance
    #add statusBar lperiodl
    #add statusBar lperiod
    #add statusBar literationl
    #add statusBar literation
    #add box statusBar

    on eventBox #motionNotifyEvent $ \event -> do
      ex <- event `get` #x
      ey <- event `get` #y
      gui <- readIORef guiR
      let i = round (ex / displaySize gui)
          j = round (ey / displaySize gui)
          buf = renderBuffer gui
      when (0 <= i && i < width buf && 0 <= j && j < height buf) $ do
        de <- V.read (contents buf) ((width buf * j + i) * channels buf + 0)
        np <- V.read (contents buf) ((width buf * j + i) * channels buf + 1)
        ni <- V.read (contents buf) ((width buf * j + i) * channels buf + 2)
        nf <- V.read (contents buf) ((width buf * j + i) * channels buf + 3)
        let n = ni + nf
            p = round np
            ti  = T.pack (show i)
            tj  = T.pack (show j)
            tde = T.pack (show de)
            tp  = T.pack (show p)
            tn  = T.pack (show n)
        set lmousex [ #label := ti ]
        set lmousey [ #label := tj ]
        set ldistance [ #label := tde ]
        set lperiod [ #label := tp ]
        set literation [ #label := tn ]
      return True
```

```
  set window [ #resizable := False ]
  onWidgetDestroy window mainQuit
  widgetShowAll window
  timeoutAdd PRIORITY_DEFAULT 200 $ #queueRender glarea' >> return True
  Gtk.main
```

8 kf/formulas.et

```
Mandelbrot
  z := z^p + c

Burning Ship
  z := (|x| + i |y|)^p + c

Buffalo
  w := z^p
  z := (|u| + i |v|) + c

Celtic
  w := z^p
  z := (|u| + i v) + c

Mandelbar
  z := (x - i y)^p + c

Mandelbar Celtic
  w := (x - i y)^2
  z := (|u| + i v) + c

Perpendicular Mandelbrot
  z := (|x| - i y)^2 + c

Perpendicular Burning Ship
  z := (x - i |y|)^2 + c

Perpendicular Celtic
  w := (|x| - i y)^2
  z := (|u| + i v) + c

Perpendicular Buffalo
  w := (x - i |y|)^2
  z := (|u| + i v) + c

Cubic Quasi Burning Ship
  z := (|x| (x^2 - 3 y^2) - i |y (3 x^2 - y^2)|) + c

Cubic Partial BS Real
  z := (|x| (x^2 - 3 y^2) + i y (3 x^2 - y^2)) + c

Cubic Partial BS Imag
  z := (x (x^2 - 3 y^2) + i |y| (3 x^2 - y^2)) + c

Cubic Flying Squirrel (Buffalo Imag)
  z := (x (x^2 - 3 y^2) + i |y (3 x^2 - y^2)|) + c

Cubic Quasi Perpendicular
  z := (|x| (x^2 - 3 y^2) - i y |3 x^2 - y^2|) + c
```

```
50
    4th Burning Ship Partial Imag
      z := (x + i |y|)^4 + c

    4th Burning Ship Partial Real
55    z := (|x| + i y)^4 + c

    4th Burning Ship Partial Real Mbar
      z := (|x| - i y)^4 + c

60  4th Celtic Burning Ship Partial Imag
      w := (x + i |y|)^4
      z := (|u| + i v) + c

    4th Celtic Burning Ship Partial Real
65    w := (|x| + i y)^4
      z := (|u| + i v) + c

    4th Celtic Burning Ship Partial Real Mbar
      w := (|x| - i |y|)^4
70    z := (|u| + i v) + c

    4th Buffalo Partial Imag
      w := z^4
      z := (u + i |v|) + c
75
    4th Celtic Mbar
      w := (x - i y)^4
      z := (|u| + i v) + c

80  4th False Quasi Perpendicular
      z := ((x^4 + y^4 - 6 x^2 y^2) - i 4 x y |x^2 - y^2|) + c

    4th False Quasi Heart
      z := ((x^4 + y^4 - 6 x^2 y^2) + i 4 x y |x^2 - y^2|) + c
85
    4th Celtic False Quasi Perpendicular
      z := (|x^4 + y^4 - 6 x^2 y^2| - i 4 x y |x^2 - y^2|) + c

    4th Celtic False Quasi Heart
90    z := (|x^4 + y^4 - 6 x^2 y^2| + i 4 x y |x^2 - y^2|) + c

    5th Burning Ship Partial
      z := (|x| + i y)^5 + c

95  5th Burning Ship Partial Mbar
      z := (|x| - i y)^5 + c

    5th Celtic Mbar
      w := (x - i y)^5
100   z := (|u| + i v) + c

    5th Quasi Burning Ship (BS/Buffalo Hybrid)
      w := (|x| + i y)^5
      z := (u - i |v|) + c
105
    5th Quasi Perpendicular
```

```
  z := (|x| (x^4 + 5 y^4 - 10 x^2 y^2) - i y (|5 x^4 + y^4 - 10 x^2 y^2|)) + c

5th Quasi Heart
  z := (|x| (x^4 + 5 y^4 - 10 x^2 y^2) + i y (|5 x^4 + y^4 - 10 x^2 y^2|)) + c

SimonBrot 4th
  z := z^2 (|x| + i |y|)^2 + c

4th Imag Quasi Perpendicular / Heart
  z := ((x^4 + y^4 - 6 x^2 y^2) + i 4 x |y (x^2 - y^2)|) + c

4th Real Quasi Perpendicular
  z := ((x^4 + y^4 - 6 x^2 y^2) - i 4 y |x (x^2 - y^2)|) + c

4th Real Quasi Heart
  z := ((x^4 + y^4 - 6 x^2 y^2) + i 4 y |x (x^2 - y^2)|) + c

4th Celtic Imag Quasi Perpendicular / Heart
  z := (|x^4 + y^4 - 6 x^2 y^2| + i 4 x |y (x^2 - y^2)|) + c

4th Celtic Real Quasi Perpendicular
  z := (|x^4 + y^4 - 6 x^2 y^2| - i 4 y |x (x^2 - y^2)|) + c

4th Celtic Real Quasi Heart
  z := (|x^4 + y^4 - 6 x^2 y^2| + i 4 y |x (x^2 - y^2)|) + c

SimonBrot 6th
  z := z^3 (|x| + i |y|)^3 + c

HPDZ Buffalo
  z := (((x^2 - y^2) - |x|) + i (|2xy| - |y|)) + c

TheRedshiftRider 1: a*z^2+z^3+c
  z := (f z^2 + z^3) + c

TheRedshiftRider 2: a*z^2-z^3+c
  z := (f z^2 - z^3) + c

TheRedshiftRider 3: 2*z^2-z^3+c
  z := (2 z^2 - z^3) + c

TheRedshiftRider 4: a*z^2+z^4+c
  z := (f z^2 + z^4) + c

TheRedshiftRider 5: a*z^2-z^4+c
  z := (f z^2 - z^4) + c

TheRedshiftRider 6: a*z^2+z^5+c
  z := (f z^2 + z^5) + c

TheRedshiftRider 7: a*z^2-z^5+c
  z := (f z^2 - z^5) + c

TheRedshiftRider 8: a*z^2+z^6+c
  z := (f z^2 + z^6) + c

TheRedshiftRider 9: a*z^2-z^6+c
```

```
    z := (f z^2 - z^6) + c

SimonBrot2 4th
    w := z^2
    z := w (|u| + i |v|) + c

General Quadratic Minus
    z := ((x^2 - y^2) + i (2 d x y + e x^2)) + c

General Quadratic Plus
    z := ((x^2 + y^2) + i (2 d x y + e x^2)) + c

Mothbrot 2nd 1x1
    z := z^1 (|x| + i |y|)^1 + c

Mothbrot 3rd 1x2
    z := z^1 (|x| + i |y|)^2 + c

Mothbrot 3rd 2x1
    z := z^2 (|x| + i |y|)^1 + c

Mothbrot 4th 1x3
    z := z^1 (|x| + i |y|)^3 + c

Mothbrot 4th 2x2 (aka SimonBrot 4th)
    z := z^2 (|x| + i |y|)^2 + c

Mothbrot 4th 3x1
    z := z^3 (|x| + i |y|)^1 + c

Mothbrot 5th 1x4
    z := z^1 (|x| + i |y|)^4 + c

Mothbrot 5th 2x3
    z := z^2 (|x| + i |y|)^3 + c

Mothbrot 5th 3x2
    z := z^3 (|x| + i |y|)^2 + c

Mothbrot 5th 4x1
    z := z^4 (|x| + i |y|)^1 + c

Mothbrot 6th 1x5
    z := z^1 (|x| + i |y|)^5 + c

Mothbrot 6th 2x4 (Simon's Mothbrot)
    z := z^2 (|x| + i |y|)^4 + c

Mothbrot 6th 3x3 (aka SimonBrot 6th)
    z := z^3 (|x| + i |y|)^3 + c

Mothbrot 6th 4x2
    z := z^4 (|x| + i |y|)^2 + c

Mothbrot 6th 5x1
    z := z^5 (|x| + i |y|)^1 + c
```

9 kf/Main.hs

```
{-
et -- escape time fractals
Copyright (C) 2018 Claude Heiland-Allen

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU Affero General Public License as
published by the Free Software Foundation, either version 3 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 Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License
along with this program.  If not, see <https://www.gnu.org/licenses/>.
-}

import Control.Monad (forM_, when)
import System.Environment (getArgs)
import Data.Set (Set)
import qualified Data.Set as S
import Control.Concurrent.Async (forConcurrently_)

import Fractal.EscapeTime.Formulas.Types (R2(..), R2Formula)
import Fractal.EscapeTime.Compiler.Parser (formulas)
import Fractal.EscapeTime.Compiler.Expr.AST (Expr(EVar), NType(NFloat), ↙
    ↪ interpret)
import Fractal.EscapeTime.Compiler.Expr.Deriv (ddx)
import Fractal.EscapeTime.Compiler.Expr.Optimize (superoptimize, metric)
import Fractal.EscapeTime.Compiler.Expr.Perturb (perturb)
import Fractal.EscapeTime.Compiler.Expr.Pretty (pretty)
import Fractal.EscapeTime.Algorithms.R2 (period_tri, period_jsk, newton, size, ↙
    ↪ skew)

import Perturbation (perturbation)
import Reference (reference)

type Formula = R2 (Expr String) -> R2 (Expr String) -> R2 (Expr String) -> R2 (↙
    ↪ Expr String)

main :: IO ()
main = do
  [dir] <- getArgs
  e <- formulas <$> getContents
  case e of
    Left msg -> error msg
    Right fs -> do

      forConcurrently_ (zip [0..] fs) $ \(n, (name, vars, f)) -> do
        let q = 0
        if S.member "p" vars
        then forConcurrently_ [2 .. if n == 0 then 5{-10-} else 5] $ \p -> case ↙
            ↪ f p q of
          [g] -> main' dir n name (Just p) g
```

```
    else let p = 0 in case f p q of
      [g] -> main' dir n name Nothing g

writeFile (dir ++ "/combo5_addstrings.c") . unlines $
  [ "#include <windows.h>"
  , "void combo5_addstrings(HWND hWnd, const int IDC_COMBO5)"
  , "{"
  ] ++
  [ "  SendDlgItemMessage(hWnd,IDC_COMBO5,CB_ADDSTRING,0,(LPARAM)" ++ show name ++ "); // " ++ show n
  | (n, (name, _, _)) <- zip [0..] fs
  ] ++
  [ "}"
  ]

writeFile (dir ++ "/update_power_dropdown_for_fractal_type.c") . unlines $
  [ "#include <windows.h>"
  , "void update_power_dropdown_for_fractal_type(HWND hWnd, const int IDC_COMBO3, const int m_nFractalType, const int m_nPower)"
  , "{"
  , "  SendDlgItemMessage(hWnd,IDC_COMBO3,CB_RESETCONTENT,0,0);"
  , "  int selected = 0;"
  , "  int ix = 0;"
  , "  switch (m_nFractalType)"
  , "  {"
  ] ++
  [ unlines
    [ "    case " ++ show n ++ ": // " ++ name
    , "    {"
    , unlines . concat $
      [ [ "      if (" ++ show power ++ " == m_nPower) selected = ix;"
        , "      SendDlgItemMessage(hWnd,IDC_COMBO3,CB_ADDSTRING,0,(LPARAM)" ++ show (show power) ++ ");"
        , "      ix++;"
        ]
      | power <- powers_of n vars f
      ]
    , "      break;"
    , "    }"
    ]
  | (n, (name, vars, f)) <- zip [0..] fs
  ] ++
  [ "  }"
  , "  SendDlgItemMessage(hWnd,IDC_COMBO3,CB_SETCURSEL,selected,0);"
  , "  EnableWindow(GetDlgItem(hWnd,IDC_COMBO3), ix > 1);"
  , "}"
  ]

writeFile (dir ++ "/validate_power_for_fractal_type.c") . unlines $
  [ "int validate_power_for_fractal_type(const int m_nFractalType, const int m_nPower)"
  , "{"
  , "  switch (m_nFractalType)"
  , "  {"
  ] ++
  [ unlines
    [ "    case " ++ show n ++ ": // " ++ name
```

```
105             , "       switch (m_nPower)"
                , "       {"
                , "         default:"
                , unlines $
                  [ "         case " ++ show power ++ ": return " ++ show power ++ ";"
110               | power <- powers_of n vars f
                  ]
                , "       }"
                , "       break;"
                ]
115           | (n, (name, vars, f)) <- zip [0..] fs
              ] ++
              [ "     default:"
              , "       return 2;"
              , "   }"
120           , "}"
              ]

        forM_ (zip [0..] fs) $ \(n, (name, vars, f)) -> when (n > -1) $ do
          putStrLn$" <group type=\"" ++ show n ++ "\" name=\"" ++ name ++ "\">"
125       case f 0 0 of
            [g] -> do
              let Just p = compute_degree g
                  glitch = 0.1 ** ([7,6,5,4,4,3,3,2,2] !! (fromIntegral p - 2))
                  -- variable
130               v = EVar NFloat
                  aar = v "Ar"
                  aai = v "Ai"
                  ccr = v "Cr"
                  cci = v "Ci"
135               xxr = v "Xr"
                  xxi = v "Xi"
                  cr = v "cr"
                  ci = v "ci"
                  xr = v "xr"
140               xi = v "xi"
                  xxxr = v "Xxr"
                  xxxi = v "Xxi"
                  cccr = v "Ccr"
                  ccci = v "Cci"
145               daa = v "daa"
                  dab = v "dab"
                  dba = v "dba"
                  dbb = v "dbb"
                  dxa = v "dxa"
150               dxb = v "dxb"
                  dya = v "dya"
                  dyb = v "dyb"
                  optimize' = superoptimize 10000 100
                  -- reference
155               R2 xxrn0 xxin0 = g (R2 aar aai) (R2 ccr cci) (R2 xxr xxi)
                  xxrn = optimize' xxrn0
                  xxin = optimize' xxin0
                  -- perturbation
                  vs =
160                 [ ("Ar", (aar, 0))
                    , ("Ai", (aai, 0))
```

```
                    , ("Cr", (ccr, cr))
                    , ("Ci", (cci, ci))
                    , ("Xr", (xxr, xr))
                    , ("Xi", (xxi, xi))
                    ]
                xrn = optimize' $ perturb vs xxrn
                xin = optimize' $ perturb vs xxin
                -- derivative
                R2 xxxrn0 xxxin0 = g (R2 aar aai) (R2 cccr ccci) (R2 xxxr xxxi)
                xxxrn = optimize' xxxrn0
                xxxin = optimize' xxxin0
                dxan = optimize' $ ddx d cccr xxxrn
                dxbn = optimize' $ ddx d ccci xxxrn
                dyan = optimize' $ ddx d cccr xxxin
                dybn = optimize' $ ddx d ccci xxxin
                d v u
                  | u == xxxr && v == cccr = dxa
                  | u == xxxr && v == ccci = dxb
                  | u == xxxi && v == cccr = dya
                  | u == xxxi && v == ccci = dyb
                  | u == cccr && v == cccr = daa
                  | u == cccr && v == ccci = dab
                  | u == ccci && v == cccr = dba
                  | u == ccci && v == ccci = dbb
                  | u == v = 1
                  | otherwise = 0
            putStrLn$"    <formula power=\"" ++ show p ++ "\" glitch=\"" ++ show↙
                ↳ glitch ++ "\">"
            putStrLn "      <reference t='R'>"
            putStrLn$"        Xrn = " ++ pretty xxrn ++ "; //" ++ show (metric ↙
                ↳ xxrn)
            putStrLn$"        Xin = " ++ pretty xxin ++ "; //" ++ show (metric ↙
                ↳ xxin)
            putStrLn "      </reference>"
            putStrLn "      <perturbation t='R'>"
            putStrLn$"        xrn = " ++ pretty xrn ++ "; //" ++ show (metric ↙
                ↳ xrn)
            putStrLn$"        xin = " ++ pretty xin ++ "; //" ++ show (metric ↙
                ↳ xin)
            putStrLn "      </perturbation>"
            putStrLn "      <derivative t='M'>"
            putStrLn$"        dxan = " ++ pretty dxan ++ "; //" ++ show (metric ↙
                ↳ dxan)
            putStrLn$"        dxbn = " ++ pretty dxbn ++ ": //" ++ show (metric ↙
                ↳ dxbn)
            putStrLn$"        dyan = " ++ pretty dyan ++ "; //" ++ show (metric ↙
                ↳ dyan)
            putStrLn$"        dybn = " ++ pretty dybn ++ "; //" ++ show (metric ↙
                ↳ dybn)
            putStrLn "      </derivative>"
            putStrLn "    </formula>"
        putStrLn "  </group>"
        putStrLn ""

powers_of :: Int -> Set String -> (Int -> Int -> [Formula]) -> [Integer]
powers_of n vars f =
  if S.member "p" vars
```

```
      then [2 .. if n == 0 then 10 else 5]
      else case compute_degree (case f 0 0 of [g] -> g :: Formula) of Just p -> [p↙
          ↳ ]

compute_degree :: Formula -> Maybe Integer
compute_degree f = do
  let er :: Double
      er = recip 10000
      v = EVar NFloat
      vs = [("a", 0), ("b", 0), ("d", 1), ("e", 0), ("x", er), ("y", 0)]
      R2 fx fy = f (R2 (v "d") (v "e") ) (R2 (v "a") (v "b")) (R2 (v "x") (v "y↙
          ↳ "))
  x <- interpret vs fx
  y <- interpret vs fy
  let z = sqrt $ x * x + y * y
      p = round $ log z / log er
  return $ max p 2

main' :: String -> Int -> String -> Maybe Int -> Formula -> IO ()
main' dir index human_name mp f = do
  let p = case mp of
        Nothing -> case compute_degree f of
          Just q -> fromInteger q
        Just q -> q
      symbol_name = "formula_" ++ show index ++ case mp of Nothing -> "" ; Just ↙
          ↳ q -> "_" ++ show q
      file_name = dir ++ "/" ++ symbol_name ++ ".c"
      source = "#define et " ++ symbol_name ++ "\n" ++ compile index p ↙
          ↳ human_name [(p, f)]
  writeFile file_name source

compile :: Int -> Int -> String -> [(Int, R2Formula (Expr String))] -> String
compile typ powr name fs =
  let header = "#include \"formula.h\"\n"
      refe = reference False typ powr fs
      refd = reference True  typ powr fs
      pert = perturbation False typ powr fs
      perd = perturbation True  typ powr fs
      per1 = period_tri fs
      per2 = period_jsk fs
      newt = newton fs
      siz' = size fs
      ske' = skew fs
      --dsiz = domain_size fs
      degr = exp $ sum (map (log . fromIntegral . fst) fs) / fromIntegral (↙
          ↳ length fs) :: Double
      footer = "FORMULA(" ++ show name ++ ",\"(source code unknown)\"," ++ show ↙
          ↳ (if isNaN degr || isInfinite degr then 0 else degr) ++ ")\n"
  in  header ++ refe ++ refd ++ pert ++ perd ++ per1 ++ per2 ++ newt ++ siz' ++ ↙
      ↳ ske' ++ footer
```

10 kf/Perturbation.hs

```
{-
et -- escape time fractals
Copyright (C) 2018 Claude Heiland-Allen
```

```
{-
    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU Affero General Public License as
    published by the Free Software Foundation, either version 3 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 Affero General Public License for more details.

    You should have received a copy of the GNU Affero General Public License
    along with this program.  If not, see <https://www.gnu.org/licenses/>.
-}

module Perturbation (perturbation) where

import Prelude hiding (read)

import Fractal.EscapeTime.Compiler
import Fractal.EscapeTime.Formulas.Types hiding (i)

fts :: [FType]
fts = [FDouble,FLongDouble,FFloatExp]

perturbation :: Bool -> Int -> Int -> [(Int, R2Formula (Expr String))] -> String
perturbation derivs typ powr fs@[(_, f)] = concat . flip map fts $ \t -> runCompile_ t . function NInt ("perturbation" ++ (if derivs then "D" else "") ++ fSuffix t)
  ([(NInt, "m_nFractalType"), (NInt, "m_nPower")
  , (NFloatPtr, "m_db_dxr"), (NFloatPtr, "m_db_dxi"), (NFloatPtr, "m_db_z")
  , (NIntPtr, "antal"), (NFloatPtr, "test1"), (NFloatPtr, "test2"), (NIntPtr, "bGlitch")
  , (NFloat, "m_nBailout2"), (NInt, "nMaxIter"), (NInt, "m_bNoGlitchDetection")
  , (NFloat, "g_real"), (NFloat, "g_imag")
  , (NFloat, "g_FactorAR"), (NFloat, "g_FactorAI")
  , (NFloatPtr, "Dr"), (NFloatPtr, "Di")
  , (NFloat, "dbD0r"), (NFloat, "dbD0i")
  ] ++ if derivs then [ (NFloatPtr, "dzc"), (NFloatPtr, "dci") ] else []) $ \ret -> do
    let m_nFractalType = EVar NInt "m_nFractalType"
        m_nPower = EVar NInt "m_nPower"
        m_db_dxr = EVar NFloatPtr "m_db_dxr"
        m_db_dxi = EVar NFloatPtr "m_db_dxi"
        m_db_z = EVar NFloatPtr "m_db_z"
        antal0 = EVar NIntPtr "antal"
        test10 = EVar NFloatPtr "test1"
        test20 = EVar NFloatPtr "test2"
        bGlitch0 = EVar NIntPtr "bGlitch"
        m_nBailout2 = EVar NFloat "m_nBailout2"
        nMaxIter = EVar NInt "nMaxIter"
        m_bNoGlitchDetection = EVar NInt "m_bNoGlitchDetection"
        g_real = EVar NFloat "g_real"
        g_imag = EVar NFloat "g_imag"
        d_@_g_FactorAR = EVar NFloat "g_FactorAR"
        e_@_g_FactorAI = EVar NFloat "g_FactorAI"
        dR0 = EVar NFloatPtr "Dr"
        dI0 = EVar NFloatPtr "Di"
```

```
        dbD0r = EVar NFloat "dbD0r"
        dbD0i = EVar NFloat "dbD0i"
        dzc = EVar NFloatPtr "dzc"
        dci = EVar NFloatPtr "dci"
    _ <- if_ (EEqual m_nFractalType (fromIntegral typ) `EAnd` EEqual m_nPower (↙
      ↳ fromIntegral powr)) (do
      antal <- int
      test1 <- float
      test2 <- float
      antal .= read antal0 0
      test1 .= read test10 0
      test2 .= read test20 0
      xr <- float
      xi <- float
      xxr <- float
      xxi <- float
      xr .= read dR0 0
      xi .= read dI0 0
      xxr .= 0
      xxi .= 0
      mderivs <- if not derivs then return Nothing else do
        dxa <- float
        dxb <- float
        dya <- float
        dyb <- float
        dxa .=  read dzc 0
        dxb .= -read dzc 1
        dya .=  read dzc 1
        dyb .=  read dzc 0
        dxan <- float
        dxbn <- float
        dyan <- float
        dybn <- float
        daa <- float
        dab <- float
        dba <- float
        dbb <- float
        daa .= read dci 0
        dab .= read dci 1
        dba .= read dci 2
        dbb .= read dci 3
        return $ Just (dxa, dxb, dya, dyb, dxan, dxbn, dyan, dybn, daa, dab, dba↙
            ↳ , dbb)
      k <- int
      k .= 0
      _ <- while_ (antal `ELess` nMaxIter) $ do
        rXr <- float
        rXi <- float
        rXz <- float
        rXr .= read m_db_dxr antal
        rXi .= read m_db_dxi antal
        rXz .= read m_db_z antal
        xxr .= rXr + xr
        xxi .= rXi + xi
        test2 .= test1
        test1 .= g_real * xxr * xxr + g_imag * xxi * xxi
        _ <- if_ (ELess test1 rXz) (do
```

```
      writeI bGlitch0 0 1
      if typ == 0 && powr == 2
        then comment "FIXME knighty glitch detection: if (↙
            ↳ type_0_power_2_pixel_has_glitched(cr, ci, xr, xi, Xr, Xi, dxa ↙
            ↳ / h, dya / h, e, h)) { // FIXME matrix derivatives"
        else return ()
      _ <- if_ (EEqual 0 m_bNoGlitchDetection) break_ (return ())
      if typ == 0 && powr == 2
        then comment "FIXME knighty glitch detection: }"
        else return ()
      return ()) (return ())
    _ <- if_ (ELess m_nBailout2 test1) break_ (return ())
    xrn <- float
    xin <- float
    -- do formula
    let R2 fx0 fy0 = f (R2 d' e') (R2 a' b') (R2 x' y')
        d' = EVar NFloat "d"
        e' = EVar NFloat "e"
        a' = EVar NFloat "a"
        b' = EVar NFloat "b"
        x' = EVar NFloat "x"
        y' = EVar NFloat "y"
        a0 = EVar NFloat "A0"
        b0 = EVar NFloat "B0"
        aa = EVar NFloat "aa"
        bb = EVar NFloat "bb"
        vs =
          [ ("d", (d_, 0))
          , ("e", (e_, 0))
          , ("a", (a0, dbD0r))
          , ("b", (b0, dbD0i))
          , ("x", (rXr, xr))
          , ("y", (rXi, xi))
          ]
        fx = perturb vs fx0
        fy = perturb vs fy0
    comment $ "fx   = " ++ pretty fx
    comment $ "fy   = " ++ pretty fy
    xrn .= fx
    xin .= fy
    case mderivs of
      Nothing -> return ()
      Just (dxa, dxb, dya, dyb, dxan, dxbn, dyan, dybn, daa, dab, dba, dbb) ↙
          ↳ -> do
        let R2 gxx gyy = f (R2 d' e') (R2 aa bb) (R2 x' y')
            fxx = optimize gxx
            fyy = optimize gyy
            fdxa = optimize $ ddx d aa fxx
            fdxb = optimize $ ddx d bb fxx
            fdya = optimize $ ddx d aa fyy
            fdyb = optimize $ ddx d bb fyy
            d v u
              | u == xr && v == aa = dxa
              | u == xr && v == bb = dxb
              | u == xi && v == aa = dya
              | u == xi && v == bb = dyb
              | u == aa && v == aa = daa
```

```
                    | u == aa && v == bb = dab
                    | u == bb && v == aa = dba
                    | u == bb && v == bb = dbb
                    | u == v = 1
                    | otherwise = 0
              comment $ "fdxa = " ++ pretty fdxa
              comment $ "fdxb = " ++ pretty fdxb
              comment $ "fdya = " ++ pretty fdya
              comment $ "fdyb = " ++ pretty fdyb
              dxan .= fdxa
              dxbn .= fdxb
              dyan .= fdya
              dybn .= fdyb
          -- continue loop
          xr .= xrn
          xi .= xin
          case mderivs of
            Nothing -> return ()
            Just (dxa, dxb, dya, dyb, dxan, dxbn, dyan, dybn, _daa, _dab, _dba, ↙
                ↳ _dbb) -> do
              dxa .= dxan
              dxb .= dxbn
              dya .= dyan
              dyb .= dybn
          antal .= antal + 1
        writeI antal0 0 antal
        write test10 0 test1
        write test20 0 test2
        write dR0 0 xr
        write dI0 0 xi
        case mderivs of
          Nothing -> return ()
          Just (dxa, dxb, dya, dyb, _dxan, _dxbn, _dyan, _dybn, _daa, _dab, _dba, ↙
              ↳ _dbb) -> do
            dr <- float
            di <- float
            sqrttest1 <- float
            sqrttest1 .= sqrt test1
            dr .= (xxr * dxa + xxi * dya) / sqrttest1
            di .= (xxr * dxb + xxi * dyb) / sqrttest1
            write dzc 0 dr
            write dzc 1 di
        ret .= 1) (ret .= 0)
    return_ ret
```

11 kf/Reference.hs

```
{-
et -- escape time fractals
Copyright (C) 2018 Claude Heiland-Allen

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU Affero General Public License as
published by the Free Software Foundation, either version 3 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 Affero General Public License for more details.

15  You should have received a copy of the GNU Affero General Public License
    along with this program.  If not, see <https://www.gnu.org/licenses/>.
    -}

    module Reference (reference) where
20
    import Prelude hiding (read)

    import Fractal.EscapeTime.Compiler
    import Fractal.EscapeTime.Formulas.Types hiding (i)
25
    fts :: [FType]
    fts = [FDouble,FLongDouble,FFloatExp]

    reference :: Bool -> Int -> Int -> [(Int, R2Formula (Expr String))] -> String
30  reference derivs typ powr fs@[(_, f)] = concat . flip map fts $ \t -> ↙
        ↘ runCompile_ t . function NInt ("reference" ++ (if derivs then "D" else "")↙
        ↘ ++ fSuffix t)
      ([(NInt, "m_nFractalType"), (NInt, "m_nPower")
      , (NFloatPtr, "m_db_dxr"), (NFloatPtr, "m_db_dxi"), (NFloatPtr, "m_db_z")
      , (NVolatileIntPtr, "m_bStop"), (NIntPtr, "m_nRDone")
      , (NIntPtr, "m_nGlitchIter"), (NIntPtr, "m_nMaxIter")
35    , (NReal, "Cr0"), (NReal, "Ci0")
      , (NFloat, "g_SeedR"), (NFloat, "g_SeedI")
      , (NFloat, "g_FactorAR"), (NFloat, "g_FactorAI")
      , (NFloat, "terminate"), (NFloat, "g_real"), (NFloat, "g_imag")
      , (NInt, "m_bGlitchLowTolerance")
40    , (NIntPtr, "antal"), (NFloatPtr, "test1"), (NFloatPtr, "test2")
      ] ++ if derivs then [ (NFloatPtr, "dzc"), (NFloatPtr, "dci") ] else []) $ \ret↙
          ↘ -> do
        early "mpfr_prec_t prec=mpfr_get_prec(Cr0);\n"
        let m_nFractalType = EVar NInt "m_nFractalType"
            m_nPower = EVar NInt "m_nPower"
45          m_db_dxr = EVar NFloatPtr "m_db_dxr"
            m_db_dxi = EVar NFloatPtr "m_db_dxi"
            m_db_z = EVar NFloatPtr "m_db_z"
            m_bStop = EVar NVolatileIntPtr "m_bStop"
            m_nRDone = EVar NVolatileIntPtr "m_nRDone"
50          m_nGlitchIter = EVar NVolatileIntPtr "m_nGlitchIter"
            m_nMaxIter = EVar NVolatileIntPtr "m_nMaxIter"
            cr0 = EVar NReal "Cr0"
            ci0 = EVar NReal "Ci0"
            g_SeedR = EVar NFloat "g_SeedR"
55          g_SeedI = EVar NFloat "g_SeedI"
            d_@_g_FactorAR = EVar NFloat "g_FactorAR"
            e_@_g_FactorAI = EVar NFloat "g_FactorAI"
            terminate = EVar NFloat "terminate"
            g_real = EVar NFloat "g_real"
60          g_imag = EVar NFloat "g_imag"
            m_bGlitchLowTolerance = EVar NInt "m_bGlitchLowTolerance"
            antal = EVar NVolatileIntPtr "antal"
            test1 = EVar NFloatPtr "test1"
            test2 = EVar NFloatPtr "test2"
```

```
65          dzc = EVar NFloatPtr "dzc"
            dci = EVar NFloatPtr "dci"
        _ <- if_ (EEqual m_nFractalType (fromIntegral typ) `EAnd` EEqual m_nPower (↙
          ↳ fromIntegral powr)) (do
          stored <- int
          stored .= 0
70        old_absval <- float
          old_absval .= 0
          abs_val <- float
          abs_val .= 0
          nMaxIter <- int
75        nMaxIter .= read m_nMaxIter 0
          writeI m_nGlitchIter 0 (nMaxIter + 1)
          glitch <- float
          glitch .= realToFrac (0.1 ** (([7,7,7,6,5,4,4,3,3] ++ repeat 2) !! powr))
          _ <- if_ (m_bGlitchLowTolerance) (glitch .= sqrt glitch) (return ())
80        a@cr <- real
          cr .= cr0
          b@ci <- real
          ci .= ci0
          x@xr <- real
85        xr .= g_SeedR
          y@xi <- real
          xi .= g_SeedI
          xrd <- float
          xrd .= xr
90        xid <- float
          xid .= xi
          xrn <- real
          xin <- real
          mderivs <- if not derivs then return Nothing else do
95          daa <- float
            dab <- float
            dba <- float
            dbb <- float
            daa .= read dci 0
100         dab .= read dci 1
            dba .= read dci 2
            dbb .= read dci 3
            dxa <- float
            dxb <- float
105         dya <- float
            dyb <- float
            dxa .= 0
            dxb .= 0
            dya .= 0
110         dyb .= 0
            dxan <- float
            dxbn <- float
            dyan <- float
            dybn <- float
115         let af = EVar NFloat "af_unused"
                bf = EVar NFloat "bf_unused"
            return (Just (daa, dab, dba, dbb, dxa, dxb, dya, dyb, dxan, dxbn, dyan, ↙
              ↳ dybn, af, bf))
          i <- int
          i .= 0
```

```
120         k <- int
            k .= 0

            while_ (ELess i nMaxIter `EAnd` EEqual 0 (read m_bStop 0)) $ do
                -- do formula
125             let R2 fx0 fy0 = f (R2 d_ e_) (R2 a b) (R2 x y)
                    fx = optimize fx0
                    fy = optimize fy0
                comment $ "fx   = " ++ pretty fx
                comment $ "fy   = " ++ pretty fy
130             xrn .= fx
                xin .= fy
                case mderivs of
                  Nothing -> return ()
                  Just (daa, dab, dba, dbb, dxa, dxb, dya, dyb, dxan, dxbn, dyan, dybn↙
                      ↪ , af, bf) -> do
135                 let R2 ffx0 ffy0 = f (R2 d_ e_) (R2 af bf) (R2 xrd xid)
                        ffx = optimize ffx0
                        ffy = optimize ffy0
                        fdxa = optimize $ ddx d af ffx
                        fdxb = optimize $ ddx d bf ffx
140                     fdya = optimize $ ddx d af ffy
                        fdyb = optimize $ ddx d bf ffy
                        d v u
                          | u == xrd && v == af = dxa
                          | u == xrd && v == bf = dxb
145                       | u == xid && v == af = dya
                          | u == xid && v == bf = dyb
                          | u == af  && v == af = daa
                          | u == af  && v == bf = dab
                          | u == bf  && v == af = dba
150                       | u == bf  && v == bf = dbb
                          | u == v = 1
                          | otherwise = 0
                    comment $ "fdxa = " ++ pretty fdxa
                    comment $ "fdxb = " ++ pretty fdxb
155                 comment $ "fdya = " ++ pretty fdya
                    comment $ "fdyb = " ++ pretty fdyb
                    dxan .= fdxa
                    dxbn .= fdxb
                    dyan .= fdya
160                 dybn .= fdyb
                -- continue loop
                xr .= xrn
                xi .= xin
                case mderivs of
165               Nothing -> return ()
                  Just (_daa, _dab, _dba, _dbb, dxa, dxb, dya, dyb, dxan, dxbn, dyan, ↙
                      ↪ dybn, _af, _bf) -> do
                    dxa .= dxan
                    dxb .= dxbn
                    dya .= dyan
170                 dyb .= dybn
                writeI m_nRDone 0 (read m_nRDone 0 + 1)
                xrd .= xr
                xid .= xi
                old_absval .= abs_val
```

```
175             abs_val .= g_real * xrd * xrd + g_imag * xid * xid
                xz <- float
                xz .= abs_val * glitch
                write m_db_dxr i xrd
                write m_db_dxi i xid
180             write m_db_z   i xz
                _ <- if_ (4 `ELessEqual` abs_val)
                  (if_ (EEqual 4 terminate `EAnd` EEqual stored 0)
                    (do
                      stored .= 1
185                   writeI antal 0 i
                      write test1 0 abs_val
                      write test2 0 old_absval
                    ) (return ())
                  ) (return ())
190             _ <- if_ (terminate `ELessEqual` abs_val)
                  (do
                    _ <- if_ (ELess 4 terminate `EAnd` EEqual stored 0)
                        (do
                          stored .= 1
195                       writeI antal 0 i
                          write test1 0 abs_val
                          write test2 0 old_absval
                        ) (return ())
                    _ <- if_ (nMaxIter `EEqual` read m_nMaxIter 0)
200                     (do
                          nMaxIter .= i + 3
                          _ <- if_ (read m_nMaxIter 0 `ELess` nMaxIter) (nMaxIter .= ↙
                              ↳ read m_nMaxIter 0) (return ())
                          writeI m_nGlitchIter 0 nMaxIter
                        ) (return ())
205                 return ()
                  ) (return ())
                i .= i + 1
          case mderivs of
            Nothing -> return ()
210         Just (_daa, _dab, _dba, _dbb, dxa, dxb, dya, dyb, _dxan, _dxbn, _dyan, ↙
                ↳ _dybn, _af, _bf) -> do
              dr <- float
              di <- float
              sqrttest1 <- float
              sqrttest1 .= sqrt (read test1 0)
215           dr .= (xrd * dxa + xid * dya) / sqrttest1;
              di .= (xrd * dxb + xid * dyb) / sqrttest1;
              write dzc 0 dr
              write dzc 1 di
        ret .= 1) (ret .= 0)
220   return_ ret
```

12 LICENSE.md

```
### GNU AFFERO GENERAL PUBLIC LICENSE

Version 3, 19 November 2007

5 Copyright (C) 2007 Free Software Foundation, Inc.
<https://fsf.org/>
```

```
    Everyone is permitted to copy and distribute verbatim copies of this
    license document, but changing it is not allowed.
10
    ### Preamble

    The GNU Affero General Public License is a free, copyleft license for
    software and other kinds of works, specifically designed to ensure
15  cooperation with the community in the case of network server software.

    The licenses for most software and other practical works are designed
    to take away your freedom to share and change the works. By contrast,
    our General Public Licenses are intended to guarantee your freedom to
20  share and change all versions of a program--to make sure it remains
    free software for all its users.

    When we speak of free software, we are referring to freedom, not
    price. Our General Public Licenses are designed to make sure that you
25  have the freedom to distribute copies of free software (and charge for
    them 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.

30  Developers that use our General Public Licenses protect your rights
    with two steps: (1) assert copyright on the software, and (2) offer
    you this License which gives you legal permission to copy, distribute
    and/or modify the software.

35  A secondary benefit of defending all users' freedom is that
    improvements made in alternate versions of the program, if they
    receive widespread use, become available for other developers to
    incorporate. Many developers of free software are heartened and
    encouraged by the resulting cooperation. However, in the case of
40  software used on network servers, this result may fail to come about.
    The GNU General Public License permits making a modified version and
    letting the public access it on a server without ever releasing its
    source code to the public.

45  The GNU Affero General Public License is designed specifically to
    ensure that, in such cases, the modified source code becomes available
    to the community. It requires the operator of a network server to
    provide the source code of the modified version running there to the
    users of that server. Therefore, public use of a modified version, on
50  a publicly accessible server, gives the public access to the source
    code of the modified version.

    An older license, called the Affero General Public License and
    published by Affero, was designed to accomplish similar goals. This is
55  a different license, not a version of the Affero GPL, but Affero has
    released a new version of the Affero GPL which permits relicensing
    under this license.

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

    ### TERMS AND CONDITIONS
```

0. Definitions.

"This License" refers to version 3 of the GNU Affero General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an

implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey 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; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

- a) The work must carry prominent notices stating that you modified it, and giving a relevant date.
- b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".
- c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.
- d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

- a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.
- b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.
- c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.
- d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.
- e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected

to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

- a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
- b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
- c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or
- d) Limiting the use for publicity purposes of names of licensors or authors of the material; or
- e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or
- f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is

reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted

by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting

any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If 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 convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Remote Network Interaction; Use with the GNU General Public License.

Notwithstanding any other provision of this License, if you modify the Program, your modified version must prominently offer all users interacting with it remotely through a computer network (if your version supports such interaction) an opportunity to receive the Corresponding Source of your version by providing access to the Corresponding Source from a network server at no charge, through some standard or customary means of facilitating copying of software. This Corresponding Source shall include the Corresponding Source for any work covered by version 3 of the GNU General Public License that is incorporated pursuant to the following paragraph.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the work with which it is combined will remain governed by version 3 of the GNU General Public License.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU Affero 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 that a certain numbered version of the GNU Affero General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU Affero General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU Affero General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different
permissions. However, no additional obligations are imposed on any
author or copyright holder as a result of your choosing to follow a
later version.

15. Disclaimer of Warranty.

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.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING
WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR
CONVEYS 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.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided
above cannot be given local legal effect according to their terms,
reviewing courts shall apply local law that most closely approximates
an absolute waiver of all civil liability in connection with the
Program, unless a warranty or assumption of liability accompanies a
copy of the Program in return for a fee.

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 state
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 Affero General Public License as
635     published by the Free Software Foundation, either version 3 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
640     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
        GNU Affero General Public License for more details.

        You should have received a copy of the GNU Affero General Public License
        along with this program.  If not, see <https://www.gnu.org/licenses/>.
645
    Also add information on how to contact you by electronic and paper
    mail.

    If your software can interact with users remotely through a computer
650 network, you should also make sure that it provides a way for users to
    get its source. For example, if your program is a web application, its
    interface could display a "Source" link that leads users to an archive
    of the code. There are many ways you could offer source, and different
    solutions will be better for different programs; see section 13 for
655 the specific requirements.

    You should also get your employer (if you work as a programmer) or
    school, if any, to sign a "copyright disclaimer" for the program, if
    necessary. For more information on this, and how to apply and follow
660 the GNU AGPL, see <https://www.gnu.org/licenses/>.
```

13 Makefile

```
    # et -- escape time fractals
    # Copyright (C) 2018 Claude Heiland-Allen
    #
    # This program is free software: you can redistribute it and/or modify
 5  # it under the terms of the GNU Affero General Public License as
    # published by the Free Software Foundation, either version 3 of the
    # License, or (at your option) any later version.
    #
    # This program is distributed in the hope that it will be useful,
10  # but WITHOUT ANY WARRANTY; without even the implied warranty of
    # MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    # GNU Affero General Public License for more details.
    #
    # You should have received a copy of the GNU Affero General Public License
15  # along with this program.  If not, see <https://www.gnu.org/licenses/>.

    CABAL ?= cabal
    MAKE ?= make
    C99 ?= gcc -std=c99 -Wall -Wextra -pedantic -fPIC -O3 -I./src/
20
    EXE_SUFFIX ?=
    LIB_SUFFIX ?= .so

    FORMULAS := $(patsubst %.c,%$(LIB_SUFFIX),$(wildcard lib/*.c))
25
    all: bin/et$(EXE_SUFFIX) $(FORMULAS)
```

```
formulas: formulas.stamp

bin/et$(EXE_SUFFIX): src/et.c
	$(C99) -fopenmp -o $@ $< -lmpfr -lgmp -lglfw -lGLEW -lGL -lm -ldl

lib/%$(LIB_SUFFIX): lib/%.c
	$(C99) -shared -o $@ $<

formulas.stamp: .cabal-sandbox/bin/et src/formulas.et
	.cabal-sandbox/bin/et < src/formulas.et ./lib && touch formulas.stamp

.cabal-sandbox/bin/et: et.cabal .cabal-sandbox
	$(CABAL) install

.cabal-sandbox:
	$(CABAL) sandbox init

.PHONY: all formulas
```

14 metric/bot.sh

```
#!/bin/bash
ow=1280
oh=1280
aa=4
tw=$((64 * aa))
th=$((64 * aa))
n=2048
running=1
while (( $running ))
do
  x="$(ghci -e "import System.Random" -e "randomRIO (-2, 2 :: Double)")"
  y="$(ghci -e "import System.Random" -e "randomRIO (-2, 2 :: Double)")"
  r=4
  fn1="$(( RANDOM % 4 ))"
  formula=Formula
  for i in $(seq 0 $((fn1)))
  do
    sx="x"
    if (( $RANDOM & 1 ))
    then
      sx="|${sx}|"
    fi
    if (( $RANDOM & 1 ))
    then
      sx="(-${sx})"
    fi
    sy="y"
    if (( $RANDOM & 1 ))
    then
      sy="|${sy}|"
    fi
    if (( $RANDOM & 1 ))
    then
      sy="(-${sy})"
    fi
    if (( $RANDOM & 1 ))
```

```
    then
      st=$sx
      sx=$sy
      sy=$st
    fi
    f="z:=(${sx}+i${sy})^p+c"
    formula="$(echo -e "${formula}\n ${f}")"
  done
  echo "$formula"
  p="$((2 + (RANDOM % 4)))"
  echo "p = $p"
  q=2
  weight="0.25"
  maxdepth="$((15 + (RANDOM % 20)))"
  stem="etbot-$(date --iso=s)"
  mkdir -p "${stem}"
  for depth in $(seq -w 0 ${maxdepth})
  do
    et-cli "${stem}/${depth}.png" "$((tw*2))" "$((th*2))" "$x" "$y" "$r" "$n" 1 ↙
        ↳ 0 0 1 1 0 "$p" "$q" "$formula" "$weight" > /dev/null 2>/dev/null
    totalscore=0
    for j in $(seq 0 4)
    do
      for i in $(seq 0 4)
      do
        convert "${stem}/${depth}.png" -colorspace RGB -crop "$((tw))x$((th))+$↙
            ↳ ((tw/4 * i))+$((th/4 * j))" -geometry "$((tw/aa))x$((th/aa))" -↙
            ↳ colorspace sRGB "${stem}/tmp.png"
        score="$(pngtopnm < "${stem}/tmp.png" | ppmtopgm | pnmdepth 255 | ./et-↙
            ↳ metric 2>/dev/null)"
        rm "${stem}/tmp.png"
        echo "$score $i $j"
      done
    done > "${stem}/${depth}.txt"
    cat "${stem}/${depth}.txt" | sort -g -k 1,1 | grep -v nan | tail -n 1 > "${↙
        ↳ stem}/${depth}.pick"
    read score i j < "${stem}/${depth}.pick"
    x="$(ghci -e "$x + ($i / 4 - 0.5) * $r :: Double")"
    y="$(ghci -e "$y + ($j / 4 - 0.5) * $r :: Double")"
    r="$(ghci -e "$r / 2 :: Double")"
    echo "${depth}/${maxdepth} ${score} $i $j"
  done
  ok="$(ghci -e "$score > 0.5")"
  if [ "x$ok" = "xTrue" ]
  then
    et-cli "${stem}/big.png" "$((ow * aa))" "$((oh * aa))" "$x" "$y" "$r" 4096 1↙
        ↳  0 0 1 1 0 "$p" "$q" "$formula" "$weight" > /dev/null
    convert "${stem}/big.png" -colorspace RGB -geometry "${ow}x${oh}" -↙
        ↳ colorspace sRGB "${stem}.png"
    running=0
  else
    echo "not good enough, try again"
  fi
done
```

15 metric/Makefile

```
et-metric: metric.c
	gcc -std=c99 -Wall -Wextra -pedantic -O3 -march=native -fopenmp -o et-metric metric.c -lfftw3f -lm
```

16 metric/metric.c

```
#include <complex.h>
#include <math.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <fftw3.h>
#define wisdomfile "/run/shm/et-metric.fftw"

#if 1
static int cmp_alpha(const void *a, const void *b)
{
  const float *p = a;
  const float *q = b;
  float x = *p;
  float y = *q;
  return (x > y) - (y > x);
}
#endif

static int cmp_image(const void *a, const void *b)
{
  const uint8_t *p = a;
  const uint8_t *q = b;
  int x = *p;
  int y = *q;
  return (x > y) - (y > x);
}

static float linear(float srgb)
{
  float s = srgb / 255.0f;
  float l;
  if (s < 0.04045f)
  {
    l = s / 12.92f;
  }
  else
  {
    l = powf((s + 0.055f) / 1.055f, 2.4f);
  }
  return l;
}

int main(int argc, char **argv)
{
  (void) argc;
  (void) argv;
  int boxes = 4;
  while (! feof(stdin))
```

```
  {
    int64_t width, height;
    if (2 != fscanf(stdin, "P5\n%ld %ld\n255", &width, &height))
    {
      if (feof(stdin))
      {
        break;
      }
      fprintf(stderr, "ERROR: failed to parse PGM header from standard input\n");
      return 1;
    }
    if (fgetc(stdin) != '\n')
    {
      fprintf(stderr, "ERROR: junk after PGM header\n");
      return 1;
    }
    uint8_t *image = malloc(width * height);
    if (! image)
    {
      fprintf(stderr, "ERROR: failed to allocate memory for PGM image data\n");
      return 1;
    }
    if (1 != fread(image, width * height, 1, stdin))
    {
      fprintf(stderr, "ERROR: failed to read PGM image data\n");
      free(image);
      return 1;
    }

#if 0
    float _Complex *ibuf = fftwf_alloc_complex(width * height);
    if (! ibuf)
    {
      fprintf(stderr, "ERROR: failed to allocate FFT input buffer\n");
      free(image);
      return 1;
    }
    float _Complex *obuf = fftwf_alloc_complex(width * height);
    if (! obuf)
    {
      fprintf(stderr, "ERROR: failed to allocate FFT output buffer\n");
      fftwf_free(ibuf);
      free(image);
      return 1;
    }
    fftwf_import_wisdom_from_filename(wisdomfile);
    fftwf_plan plan = fftwf_plan_dft_2d(width, height, ibuf, obuf, FFTW_FORWARD,
        FFTW_DESTROY_INPUT | FFTW_EXHAUSTIVE);
    fftwf_export_wisdom_to_filename(wisdomfile);
    float windowj[height];
    float windowi[width];
    #pragma omp parallel for schedule(static)
    for (int64_t j = 0; j < height; ++j)
    {
      windowj[j] = (1 - cos(2 * 3.141592653589793 / height * (j + 0.5)));
    }
```

```
    #pragma omp parallel for schedule(static)
    for (int64_t i = 0; i < width; ++i)
    {
      windowi[i] = (1 - cos(2 * 3.141592653589793 / width * (i + 0.5)));
    }
    #pragma omp parallel for schedule(static)
    for (int64_t j = 0; j < height; ++j)
    {
      for (int64_t i = 0; i < width; ++i)
      {
        int64_t k = j * width + i;
        ibuf[k] = windowi[i] * windowj[j] * (image[k] / 255.0f - 0.5f);
      }
    }
    fftwf_execute(plan);
    fftwf_destroy_plan(plan);
    fftwf_free(ibuf);
    int rBins = 6;
    int tBins = 24;
    double trHistogram[tBins][rBins];
    memset(trHistogram, 0, sizeof(trHistogram));
    float Rx = width/2;
    float Ry = height/2;
    float R = sqrt(Rx * Rx + Ry * Ry);
    float rFactor = rBins / log(R);
    double Esum = 0;
    #pragma omp parallel for schedule(static) reduction(+:Esum)
    for (int64_t j = 0; j < height; ++j)
    {
      float fy = j > height/2 ? j - height : j;
      for (int64_t i = 0; i < width; ++i)
      {
        float fx = i > width/2 ? i - width : i;
        float r = sqrtf(fx * fx + fy * fy);
        float t = atan2f(fy, fx) / 6.283185307179586; t -= floorf(t);
        float _Complex s = obuf[j * width + i];
        float x = creal(s);
        float y = cimag(s);
        float E = sqrtf(x * x + y * y);
        int rBin = fminf(fmaxf(floorf(logf(r + 1.0e-30f) * rFactor), 0), rBins -↙
            ↳ 1);
        int tBin = fminf(fmaxf(floorf(t * tBins), 0), tBins - 1);
        if (r > 0)
        {
          #pragma omp atomic
          trHistogram[tBin][rBin] += E;
        }
        Esum += r > 0 ? E : 0;
      }
    }
    fftwf_free(obuf);
#if 0
    // earth movers distance
    double rEMDs = 0;
    for (int j = 0; j < tBins; ++j)
    {
      double rEMD = 0;
```

```
      for (int i = 0; i < rBins; ++i)
      {
        rEMD += trHistogram[j][i] - trHistogram[(j+(tBins/4))%tBins][i];
        rEMDs += fabs(rEMD);
      }
    }
    double isotropy = 1.0 / (1 + rEMDs / Esum);
#endif
    // deviation from flatness
    double Eavg = Esum / (tBins * rBins);
    double isotropy = 0;
    for (int t = 0; t < tBins; ++t)
    {
      for (int r = 0; r < rBins; ++r)
      {
        double b = trHistogram[t][r] - Eavg;
        isotropy += b * b;
      }
    }
    isotropy /= tBins * rBins;
    isotropy = sqrt(isotropy);
    isotropy /= width * height;
    isotropy = 1 / (1 + isotropy);
#endif

#if 1
  int ndirs = 36; // 5deg accuracy
  double directions[ndirs];
  memset(directions, 0, sizeof(directions));
  static int kernel[5][5] =
{ {  -5,  -4,  0,   4,   5 }
, {  -8, -10,  0,  10,   8 }
, { -10, -20,  0,  20,  10 }
, {  -8, -10,  0,  10,   8 }
, {  -5,  -4,  0,   4,   5 }
};

  double gsum = 0;
  #pragma omp parallel for schedule(static) reduction(+:gsum)
  for (int64_t j = 2; j < height - 2; ++j)
  {
    for (int64_t i = 2; i < width - 2; ++i)
    {
      int mx = 0;
      int my = 0;
      for (int q = -2; q <= 2; ++q)
      {
        for (int p = -2; p <= 2; ++p)
        {
          int m = image[(j + q) * width + (i + p)];
          mx += kernel[2 + q][2 + p] * m;
          my += kernel[2 + p][2 + q] * m;
        }
      }
      float gx = mx * (1.0f / (255.f * 20.f));
      float gy = my * (1.0f / (255.f * 20.f));
      float g = gx * gx + gy * gy;
```

```
220         gsum += g;
            float gdir = atan2f(gy, gx) / 3.141592653589793f;
            gdir -= floorf(gdir);
            int dir = gdir * ndirs;
            #pragma omp atomic
225         directions[dir] += g;
          }
        }
        for (int dir = 0; dir < ndirs; ++dir)
          directions[dir] /= gsum;
230     double directionality = 0;
        for (int dir = 0; dir < ndirs; ++dir)
        {
          double d = directions[dir] - 1.0 / ndirs;
          directionality += d * d;
235     }
    #endif


    #if 1
240     //
        float *alpha = malloc(sizeof(float) * width * height);
        if (! alpha)
        {
          fprintf(stderr, "ERROR: failed to allocate memory for dimension data\n");
245       free(image);
          return 1;
        }
    #endif

250     double sum = 0;
        double black = 0;
        double white = 0;
        #pragma omp parallel for schedule(static) reduction(+:black) reduction(+:⌟
            ↳ white) reduction(+:sum)
        for (int64_t i = 0; i < width * height; ++i)
255     {
          sum += linear(image[i]);
          black += (image[i] < 4);
          white += (image[i] > 251);
        }
260     double mean = sum / (width * height);
        black /= (width * height);
        white /= (width * height);
        double gray = 1 - black - white;

265 #if 1
        #pragma omp parallel for schedule(static)
        for (int64_t j = 0; j < height; ++j)
        {
          for (int64_t i = 0; i < width; ++i)
270       {
            // compute local fractal dimension by simple linear regression
            // finds slope of log/log graph of box size vs contained measure
            // note: this is likely sRGB, perhaps it should be linearized?
            double x[boxes], y[boxes];
275         for (int box = 0; box < boxes; ++box)
```

```
        {
          int box_radius = (2 << box) - 1;
          double measure = 0;
          int count = 0;
          for (int64_t jj = j - box_radius; jj <= j + box_radius; ++jj)
          {
            if (jj < 0) continue;
            if (jj >= height) continue;
            for (int64_t ii = i - box_radius; ii <= i + box_radius; ++ii)
            {
              if (ii < 0) continue;
              if (ii >= width) continue;
              measure += image[jj * width + ii];
              count += 1;
            }
          }
          x[box] = log2(count);
          y[box] = log2(measure + count); // prevent log(0)
        }
        double sx = 0, sy = 0;
        for (int box = 0; box < boxes; ++box)
        {
          sx += x[box];
          sy += y[box];
        }
        sx /= boxes;
        sy /= boxes;
        double cov = 0, var = 0;
        for (int box = 0; box < boxes; ++box)
        {
          cov += (x[box] - sx) * (y[box] - sy);
          var += (x[box] - sx) * (x[box] - sx);
        }
        double beta = cov / var;
        double dimension = fmin(fmax(fabs(beta), 0), 2);
        alpha[j * width + i] = dimension;
      }
    }
#endif

    // compute deviation from flat greyscale histogram
    qsort(image, width * height, sizeof(uint8_t), cmp_image);
    double gray_deviation = 0;
    #pragma omp parallel for schedule(static) reduction(+:gray_deviation)
    for (int64_t i = 0; i < width * height; ++i)
    {
      double ideal = (i + 0.5) / (width * height);
      double actual = linear(image[i]);
      double dev = actual - ideal;
      gray_deviation += dev * dev;
    }
    gray_deviation /= width * height;
    gray_deviation = sqrt(gray_deviation);
    free(image);

#if 1
    // compute width of central 90th percentile excluding 0 2
```

```
    qsort(alpha, width * height, sizeof(float), cmp_alpha);
    int64_t lo0 = 0;
    for (int64_t i = 0; i < width * height; ++i)
    {
      if (alpha[i] == alpha[0]) lo0 = i; else break;
    }
    int64_t hi2 = width * height - 1;
    for (int64_t i = width * height - 1; i >= 0; --i)
    {
      if (alpha[i] == alpha[width * height - 1]) hi2 = i; else break;
    }
    int64_t lo = lo0 + 0.05 * (hi2 - lo0);
    int64_t hi = lo0 + 0.95 * (hi2 - lo0);
    double alpha_width = alpha[hi] - alpha[lo];
    double beta_width = (hi2 - lo0) / (double) (width * height);
    double gamma = 1 / (1 + 3 * 4 * pow(mean - 0.5, 2));
    free(alpha);
#endif

//    double score = /*alpha_width * beta_width * gamma */ (1 - gray_deviation) ↙
    ↪ * (1 - black) * (1 - white) * (1 - directionality);// * isotropy;//pow(↙
    ↪ alpha_width * beta_width * gamma, 0.25) * pow(1 - gray_deviation, 2) * pow↙
    ↪ (1 - black, 2) * pow(1 - white, 2) * pow(isotropy, 4);
    double score = alpha_width * (1 - black) * (1 - white) * (1 - gray_deviation↙
      ↪ ) * (1 - directionality);
#if 0
    fprintf(stderr, "alpha   \t%e\n", alpha_width);
    fprintf(stderr, "beta    \t%e\n", beta_width);
    fprintf(stderr, "gamma   \t%e\n", gamma);
    fprintf(stderr, "black   \t%e\n", 1 - black);
    fprintf(stderr, "white   \t%e\n", 1 - white);
    fprintf(stderr, "gray    \t%e\n", 1 - gray_deviation);
#endif
//    fprintf(stderr, "isotropy\t%e\n", isotropy);
    fprintf(stdout, "%.7e\n", score);
  }
  return 0;
}
```

17 NOTES.md

```
# et notes

## gerrit's glitch condition for R2 formulas

gerrit "Re: Perturbation theory" (Reply #102, Last Edit: 2017-12-27, 04:56:09)
<https://fractalforums.org/fractal-mathematics-and-new-theories/28/perturbation-↙
    ↪ theory/487/msg3212#msg3212>

Here's a general analysis, to implement you'll have to get the actual formulas
for the whatever fractal and plug them in.

Write the perturbed iteration as

    w := f(w,X,c)

where the actual orbit is z+w, X is reference orbit, and c is the delta c.
```

w, c, X are now just real 2-vectors so there is no advantage to think in terms of complex numbers. Using index notation:

    wi(n+1)=fi(w,X,c)

where iteration is denoted by (n+1), and if absent it's understood to be at iteration n. w, X, c with no indices are vectors, so w=(w1,w2). Any f without argument is supposed to be at w,X,c.
Define 3 2X2 matrices in index notation:

    Mij=dfi/dcj
    Kij=dfi/dXj
    Lij=dfi/dwj

Error in wi(n+1) is obtained by replacing every variable pi with pi+|pi|e, and linearizing in e. I think the error in c is not relevant but I'll keep it just in case.

The shift (error) in w(n+1) is now

    Dwi(n+1)=(|Lij||wj|+|Kij||Xj|+|Mij||cj|)e (1)

where I use Einstein summation convention, meaning you have to sum over every index that is repeated in a product (so qiri is shorthand for Sum_i=1^2piqi).

The relation between a shift in w and a shift in c is

    Dwi(n+1)=Jij(n+1)Dcj, with Jij=dwi/dcj, so
    Dci=(J(n+1))-1ijDwj(n+1) (2)

Error condition is now just |Dc|<h.

For any particular formula you have to work out formulas for all these matrices, reducing eq (2) to iteration n using the iteration formula, substitute (1) in (2) and you'll get a condition.

Hope I got everything right.

Edit: I don't think you need advice on implementation issues, but my approach would be to decide on a canonical form for f(), applicable to all 50 fractals, then use some symbolic software to generate the Jacobians, inverses, substitutions, and so on.

If you could make this "online" so you can enter you own formula, that would be probably considered great (Ultra Fractal style), though I personally don't care about all these "artificial" fractals.

realflow100's hybrid technique

Loop with some Mandlebrot and some Burning Ship or other. Suggested syntax for formula compiler:

```
z := z^2 + c
z := (|x| + i |y|)^2 + c
z := z^2 + c
z := z^2 + c
```

18 README.md

```
# et -- escape-time fractals

Escape-time fractals powered by a formula compiler.  Graphics are visualized
using distance estimation, normalized iteration count, and/or atom-domain
colouring.  Navigation is enhanced by Newton-Raphson zooming: a single action
can bring you to a mini-set or an embedded Julia.

<https://mathr.co.uk/et>

## Quick Start

### Prerequisites

    sudo apt-get install \
      build-essential \
      ghc \
      cabal-install \
      libmpfr-dev \
      libgirepository1.0-dev \
      libwebkit2gtk-4.0-dev \
      libgtksourceview-3.0-dev

### Build

    cabal v2-install et.cabal --overwrite-policy=always

### Run

    et-gtk
    et-cli
    et-kf < kf/formulas.et /path/to/output

## et-gtk

### Mouse Controls

- Left

  Zoom in.

- Right

  Zoom out.

- Middle

  Center view.

- Scroll

  Zoom.

### Keyboard Controls

- Escape
```

```
  Stop rendering.

- F5

  Restart rendering.

- 1 2 3 4 5 6 7 8

  Set supersampling to NxN.

- 9 0

  Adjust colouring weight.

- - =

  Adjust maximum iteration count.

- s

  Save timestamped image to ~/Pictures/et/ or similar.

- m

  Zoom to mini-set and auto-skew.

- j

  Zoom to embedded Julia set and auto-skew.

- k l

  Auto-skew.

- u

  Reset skew.

- , .

  Rotate.

- (cursor keys)

  Translate by large amounts.

- 4 2 6 8 (numeric keypad)

  Translate by small amounts.

- PageUp PageDown

  Zoom by large amounts.

- - + (numeric keypad)
```

```
  Zoom by small amounts.

- Home

  Reset view.

## Legal

et -- escape time fractals
Copyright (C) 2018,2019 Claude Heiland-Allen

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU Affero General Public License as
published by the Free Software Foundation, either version 3 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 Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License
along with this program.  If not, see <https://www.gnu.org/licenses/>.
```

19 Setup.hs

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

20 src/et.c

```
/*
et -- escape time fractals
Copyright (C) 2018 Claude Heiland-Allen

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU Affero General Public License as
published by the Free Software Foundation, either version 3 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 Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License
along with this program.  If not, see <https://www.gnu.org/licenses/>.
*/

#define _GNU_SOURCE
#define _POSIX_C_SOURCE 199309L

#include <assert.h>
#include <math.h>
#include <stdbool.h>
#include <stdio.h>
```

```
#include <stdlib.h>
#include <string.h>
#include <time.h>

#include <pthread.h>

#include <mpfr.h>

#include <GL/glew.h>
#include <GLFW/glfw3.h>

#include <dlfcn.h>

#define ET_MAIN 1
#include "formula.h"
#undef ET_MAIN

static struct formula *formula = 0;

// http://www.burtleburtle.net/bob/hash/integer.html
static uint32_t wang_hash(uint32_t a)
{
    a = (a ^ 61) ^ (a >> 16);
    a = a + (a << 3);
    a = a ^ (a >> 4);
    a = a * 0x27d4eb2d;
    a = a ^ (a >> 15);
    return a;
}

static double dither(uint32_t x, uint32_t y, uint32_t c)
{
  return wang_hash(c + wang_hash(y + wang_hash(x))) / (double) (0x100000000LL) -↙
      ↳ 0.5;
}

struct et_pixel
{
  uint16_t x, y; uint8_t w, h;
};

static int et_pixel_cmp(const void *a, const void *b, void *c)
{
  const struct et_pixel *s = a;
  const struct et_pixel *t = b;
  const int *r = c;
  double x = r[0];
  double y = r[1];
  double dax = s->x - x;
  double day = s->y - y;
  double dbx = t->x - x;
  double dby = t->y - y;
  double da = dax * dax + day * day;
  double db = dbx * dbx + dby * dby;
  if (da < db) return -1;
  if (da > db) return 1;
  return 0;
```

```
}

struct et
{
  int width;
  int height;
  int maxiters;
  double escape_radius;
  double escape_radius_2;
  mpfr_t centerx;
  mpfr_t centery;
  mpfr_t radius;
  float *output;
  pthread_t thread;
  int volatile active;
  int volatile running;
  int volatile done_pixels;
  struct et_pixel *pixels;
  int volatile npixel;
};

static struct et *et_new(int width, int height)
{
  struct et *et = calloc(1, sizeof(struct et));
  assert(et);
  et->width = width;
  et->height = height;
  et->maxiters = 1 << 8;
  et->escape_radius = 10000;
  et->escape_radius_2 = et->escape_radius * et->escape_radius;
  mpfr_init2(et->centerx, 53);
  mpfr_init2(et->centery, 53);
  mpfr_init2(et->radius, 53);
  mpfr_set_d(et->centerx, 0, MPFR_RNDN);
  mpfr_set_d(et->centery, 0, MPFR_RNDN);
  mpfr_set_d(et->radius, 2, MPFR_RNDN);
  et->output = calloc(1, 4 * width * height * sizeof(float));
  assert(et->output);

  et->pixels = malloc(width * height * sizeof(struct et_pixel));
  assert(et->pixels);
  int center[2] = { width >> 1, height >> 1 };
  int step = 1 << 7;
  int ix = 0;
  int begin = ix;
  for (int y = 0; y < height; y += step)
    for (int x = 0; x < width; x += step)
    {
      et->pixels[ix].x = x;
      et->pixels[ix].y = y;
      et->pixels[ix].w = step;
      et->pixels[ix].h = step;
      ix++;
    }
  int end = ix;
  qsort_r(&et->pixels[begin], end - begin, sizeof(struct et_pixel), et_pixel_cmp
    ↳ , center);
```

```
  for (; step > 1; step >>= 1)
  {
    begin = ix;
    for (int y = 0;  y < height;  y += step)
      for (int x = step >> 1; x < width; x += step)
      {
        et->pixels[ix].x = x;
        et->pixels[ix].y = y;
        et->pixels[ix].w = step >> 1;
        et->pixels[ix].h = step;
        ix++;
      }
    end = ix;
    qsort_r(&et->pixels[begin], end - begin, sizeof(struct et_pixel), ↙
        ↳ et_pixel_cmp, center);
    begin = ix;
    for (int y = step >> 1; y < height; y += step)
      for (int x = 0; x < width; x += step >> 1)
      {
        et->pixels[ix].x = x;
        et->pixels[ix].y = y;
        et->pixels[ix].w = step >> 1;
        et->pixels[ix].h = step >> 1;
        ix++;
      }
    end = ix;
    qsort_r(&et->pixels[begin], end - begin, sizeof(struct et_pixel), ↙
        ↳ et_pixel_cmp, center);
  }
  assert(ix == width * height);
  return et;
}

static void et_delete(struct et *et)
{
  assert(et);
  assert(et->output);
  assert(et->pixels);
  free(et->output);
  free(et->pixels);
  free(et);
}

static void output_pixelf(struct et *et, int npixel, float *glitch, float g, ↙
    ↳ float out0, float out1, float out2, float out3)
{
  int width = et->width;
  int height = et->height;
  int ii = et->pixels[npixel].x;
  int jj = et->pixels[npixel].y;
  int w = et->pixels[npixel].w;
  int h = et->pixels[npixel].h;
  float *output = et->output;
  for (int v = jj; v < jj + h && v < height; ++v)
  {
    for (int u = ii; u < ii + w && u < width; ++u)
    {
```

```
      int kk = v * width + u;
      if (glitch[kk] < 0)
      {
        output[4 * kk + 0] = out0;
        output[4 * kk + 1] = out1;
        output[4 * kk + 2] = out2;
        output[4 * kk + 3] = out3;
      }
    }
  }
  glitch[jj * width + ii] = g;
}

static void output_pixel(struct et *et, int npixel, double *glitch, double g, ↙
    ↳ float out0, float out1, float out2, float out3)
{
  int width = et->width;
  int height = et->height;
  int ii = et->pixels[npixel].x;
  int jj = et->pixels[npixel].y;
  int w = et->pixels[npixel].w;
  int h = et->pixels[npixel].h;
  float *output = et->output;
  for (int v = jj; v < jj + h && v < height; ++v)
  {
    for (int u = ii; u < ii + w && u < width; ++u)
    {
      int kk = v * width + u;
      if (glitch[kk] < 0)
      {
        output[4 * kk + 0] = out0;
        output[4 * kk + 1] = out1;
        output[4 * kk + 2] = out2;
        output[4 * kk + 3] = out3;
      }
    }
  }
  glitch[jj * width + ii] = g;
}

static void output_pixell(struct et *et, int npixel, long double *glitch, long ↙
    ↳ double g, float out0, float out1, float out2, float out3)
{
  int width = et->width;
  int height = et->height;
  int ii = et->pixels[npixel].x;
  int jj = et->pixels[npixel].y;
  int w = et->pixels[npixel].w;
  int h = et->pixels[npixel].h;
  float *output = et->output;
  for (int v = jj; v < jj + h && v < height; ++v)
  {
    for (int u = ii; u < ii + w && u < width; ++u)
    {
      int kk = v * width + u;
      if (glitch[kk] < 0)
      {
```

```
          output[4 * kk + 0] = out0;
          output[4 * kk + 1] = out1;
          output[4 * kk + 2] = out2;
          output[4 * kk + 3] = out3;
        }
      }
    }
    glitch[jj * width + ii] = g;
  }

  static void *et_worker(void *threadarg)
  {
    struct et *et = threadarg;
    assert(et);

    int width = et->width;
    int height = et->height;
    int n = et->maxiters;
    long double er2 = et->escape_radius_2;
    long double h = mpfr_get_ld(et->radius, MPFR_RNDN) * 2.0 / height;
    float *output = et->output;
    assert(output);
    int e = 0;
    frexpl(h, &e);
    int i0 = width / 2;
    int j0 = height / 2;
    if (! (h > 0))
    {
      et->active = false;
      return 0;
    }

    if (0 && e > -21) // PLAIN FLOAT
    {

      float a = mpfr_get_flt(et->centerx, MPFR_RNDN);
      float b = mpfr_get_flt(et->centery, MPFR_RNDN);
      float *glitch = calloc(1, width * height * sizeof(float));
      assert(glitch);
      #pragma omp parallel for
      for (int j = 0; j < height; ++j)
      {
        for (int i = 0; i < width; ++i)
        {
          int k = j * width + i;
          glitch[k] = -1;
          output[4 * k + 0] = -1;
          output[4 * k + 1] = -1;
          output[4 * k + 2] = -1;
          output[4 * k + 3] = -1;
        }
      }
      et->npixel = 0;
      #pragma omp parallel for
      for (int j = 0; j < height; ++j)
      {
        for (int i = 0; i < width && et->running; ++i)
```

```
      {
305     int npixel;
        #pragma omp atomic capture
        npixel = et->npixel++;
        int ii = et->pixels[npixel].x;
        int jj = et->pixels[npixel].y;
310     float x = ((ii + dither(ii, jj, 0)) - (i0 + dither(i0, j0, 0))) * h + a;
        float y = ((jj + dither(ii, jj, 1)) - (j0 + dither(i0, j0, 1))) * h + b;
        float out[4];
        if (formula->plainf(n, er2, h, x, y, out, &et->running))
        {
315       output_pixelf(et, npixel, glitch, 0, out[0], out[1], out[2], out[3]);
          #pragma omp atomic
          et->done_pixels++;
        }
      }
320   }
      free(glitch);
      et->active = false;
      return 0;

325 }
    else if (e > -50) // PLAIN DOUBLE
    {

      double a = mpfr_get_d(et->centerx, MPFR_RNDN);
330   double b = mpfr_get_d(et->centery, MPFR_RNDN);
      double *glitch = calloc(1, width * height * sizeof(double));
      assert(glitch);
      #pragma omp parallel for
      for (int j = 0; j < height; ++j)
335   {
        for (int i = 0; i < width; ++i)
        {
          int k = j * width + i;
          glitch[k] = -1;
340       output[4 * k + 0] = -1;
          output[4 * k + 1] = -1;
          output[4 * k + 2] = -1;
          output[4 * k + 3] = -1;
        }
345   }
      et->npixel = 0;
      #pragma omp parallel for
      for (int j = 0; j < height; ++j)
      {
350     for (int i = 0; i < width && et->running; ++i)
        {
          int npixel;
          #pragma omp atomic capture
          npixel = et->npixel++;
355       int ii = et->pixels[npixel].x;
          int jj = et->pixels[npixel].y;
          double x = ((ii + dither(ii, jj, 0)) - (i0 + dither(i0, j0, 0))) * h + a↙
              ↳ ;
          double y = ((jj + dither(ii, jj, 1)) - (j0 + dither(i0, j0, 1))) * h + b↙
              ↳ ;
```

```
          double out[4];
360       if (formula->plain(n, er2, h, x, y, out, &et->running))
          {
            output_pixel(et, npixel, glitch, 0, out[0], out[1], out[2], out[3]);
            #pragma omp atomic
            et->done_pixels++;
365       }
        }
      }
      free(glitch);
      et->active = false;
370   return 0;

    }
    else if (e > -60) // PLAIN LONG DOUBLE
    {
375
      long double a = mpfr_get_ld(et->centerx, MPFR_RNDN);
      long double b = mpfr_get_ld(et->centery, MPFR_RNDN);
      long double *glitch = calloc(1, width * height * sizeof(long double));
      assert(glitch);
380   #pragma omp parallel for
      for (int j = 0; j < height; ++j)
      {
        for (int i = 0; i < width; ++i)
        {
385       int k = j * width + i;
          glitch[k] = -1;
          output[4 * k + 0] = -1;
          output[4 * k + 1] = -1;
          output[4 * k + 2] = -1;
390       output[4 * k + 3] = -1;
        }
      }
      et->npixel = 0;
      #pragma omp parallel for
395   for (int j = 0; j < height; ++j)
      {
        for (int i = 0; i < width && et->running; ++i)
        {
          int npixel;
400       #pragma omp atomic capture
          npixel = et->npixel++;
          int ii = et->pixels[npixel].x;
          int jj = et->pixels[npixel].y;
          long double x = ((ii + dither(ii, jj, 0)) - (i0 + dither(i0, j0, 0))) * ↙
              ↳ h + a;
405       long double y = ((jj + dither(ii, jj, 1)) - (j0 + dither(i0, j0, 1))) * ↙
              ↳ h + b;
          long double out[4];
          if (formula->plainl(n, er2, h, x, y, out, &et->running))
          {
            output_pixell(et, npixel, glitch, 0, out[0], out[1], out[2], out[3]);
410         #pragma omp atomic
            et->done_pixels++;
          }
        }
```

```
      }
      free(glitch);
      et->active = false;
      return 0;

    }
    else if (0 && e > -120) // PERTURB FLOAT
    {

      mpfr_t a, b, r;
      int prec = 53 - mpfr_get_exp(et->radius);
      prec = prec < 53 ? 53 : prec;
      mpfr_init2(a, prec);
      mpfr_init2(b, prec);
      mpfr_init2(r, 53);
      mpfr_set(a, et->centerx, MPFR_RNDN);
      mpfr_set(b, et->centery, MPFR_RNDN);
      mpfr_set(r, et->radius, MPFR_RNDN);
      float *xyzs = malloc(n * 3 * sizeof(float));
      assert(xyzs);
      float *glitch = calloc(1, width * height * sizeof(float));
      assert(glitch);
      #pragma omp parallel for
      for (int j = 0; j < height; ++j)
      {
        for (int i = 0; i < width; ++i)
        {
          glitch[j * width + i] = -1;
          output[4 * (j * width + i) + 0] = -1;
          output[4 * (j * width + i) + 1] = -1;
          output[4 * (j * width + i) + 2] = -1;
          output[4 * (j * width + i) + 3] = -1;
        }
      }

      bool glitched = true;
      int ref = 0;
      do
      {

        glitched = false;
        int m = formula->referencef(n, er2, a, b, xyzs, &et->running);
        et->npixel = 0;
        #pragma omp parallel for
        for (int j = 0; j < height; ++j)
        {
          for (int i = 0; i < width && et->running; ++i)
          {
            int npixel;
            #pragma omp atomic capture
            npixel = et->npixel++;
            int ii = et->pixels[npixel].x;
            int jj = et->pixels[npixel].y;
            int k = jj * width + ii;
            if (glitch[k] < 0)
            {
              // ignore isolated small glitches
```

```
                int g = 0;
                float out[4];
                for (int v = -1; v <= 1; ++v)
                {
475               for (int u = -1; u <= 1; ++u)
                  {
                    if (u && v) continue;
                    if ((! u) && (! v)) continue;
                    if (ii + u < 0) continue;
480                 if (ii + u >= width) continue;
                    if (jj + v < 0) continue;
                    if (jj + v >= height) continue;
                    int kk = (jj + v) * width + (ii + u);
                    g += (glitch[kk] < 0);
485                 out[0] = 1e-30;
                    out[1] = output[4 * kk + 1];
                    out[2] = output[4 * kk + 2];
                    out[3] = output[4 * kk + 3];
                  }
490             }
                if (g == 0)
                {
                  output[4 * k + 0] = out[0];
                  output[4 * k + 1] = out[1];
495               output[4 * k + 2] = out[2];
                  output[4 * k + 3] = out[3];
                  glitch[k] = 0;
                  #pragma omp atomic
                  et->done_pixels++;
500             }
                else
                {
                  float x = ((ii + dither(ii, jj, 0)) - (i0 + dither(i0, j0, 0))) * ↙
                      ↪ h;
                  float y = ((jj + dither(ii, jj, 1)) - (j0 + dither(i0, j0, 1))) * ↙
                      ↪ h;
505               if (formula->perturbationf(m, n, er2, h, x, y, xyzs, out, &et->↙
                      ↪ running))
                  {
                    if (out[0] < 0)
                    {
                      // the pixel was glitched
510                   if (ii == i0 && jj == j0)
                      {
                        // the reference pixel was glitched, how can that be?
                        output_pixelf(et, npixel, glitch, 0, 1e-30, 0, 0, 0);
                        #pragma omp atomic
515                     et->done_pixels++;
                      }
                      else
                      {
                        output_pixelf(et, npixel, glitch, out[0], -1, 0, 0, 0);
520                     glitched = true;
                      }
                    }
                    else
                    {
```

```
525                     if (isnanf(out[0]) || isinff(out[0]))
                        {
                          out[0] = 1e-30;
                        }
                        output_pixelf(et, npixel, glitch, 0, out[0], out[1], out[2], ↙
                            ↳ out[3]);
530                     #pragma omp atomic
                        et->done_pixels++;
                      }
                    }
                  }
535             }
              }
            }

            if (glitched && et->running)
540         {
              float ma = -1.0/0.0;
              int mi = width/2;
              int mj = height/2;
              for (int j = 0; j < height; ++j)
545           {
                for (int i = 0; i < width; ++i)
                {
                  int k = j * width + i;
                  if (glitch[k] < 0 && ma < glitch[k])
550               {
                    ma = glitch[k];
                    mi = i;
                    mj = j;
                  }
555             }
              }
              float dx = ((mi + dither(mi, mj, 0)) - (i0 + dither(i0, j0, 0))) * h;
              float dy = ((mj + dither(mi, mj, 1)) - (j0 + dither(i0, j0, 1))) * h;
              mpfr_set_flt(r, dx, MPFR_RNDN);
560           mpfr_add(a, a, r, MPFR_RNDN);
              mpfr_set_flt(r, dy, MPFR_RNDN);
              mpfr_add(b, b, r, MPFR_RNDN);
              i0 = mi;
              j0 = mj;
565         }
            fprintf(stderr, "\r\t%8d (%d)", width * height - et->done_pixels, ++ref);

          } while (glitched && et->running);
          free(glitch);
570       free(xyzs);
          mpfr_clear(a);
          mpfr_clear(b);
          mpfr_clear(r);
          et->active = false;
575       return 0;

        }
        else if (e > -1020) // PERTURB DOUBLE
        {
580
```

```
      mpfr_t a, b, r;
      int prec = 53 - mpfr_get_exp(et->radius);
      prec = prec < 53 ? 53 : prec;
      mpfr_init2(a, prec);
585   mpfr_init2(b, prec);
      mpfr_init2(r, 53);
      mpfr_set(a, et->centerx, MPFR_RNDN);
      mpfr_set(b, et->centery, MPFR_RNDN);
      mpfr_set(r, et->radius, MPFR_RNDN);
590   double *xyzs = malloc(n * 3 * sizeof(double));
      assert(xyzs);
      double *glitch = calloc(1, width * height * sizeof(double));
      assert(glitch);
      #pragma omp parallel for
595   for (int j = 0; j < height; ++j)
      {
        for (int i = 0; i < width; ++i)
        {
          glitch[j * width + i] = -1;
600       output[4 * (j * width + i) + 0] = -1;
          output[4 * (j * width + i) + 1] = -1;
          output[4 * (j * width + i) + 2] = -1;
          output[4 * (j * width + i) + 3] = -1;
        }
605   }

      bool glitched = true;
      int ref = 0;
      do
610   {

        glitched = false;
        int m = formula->reference(n, er2, a, b, xyzs, &et->running);
        et->npixel = 0;
615     #pragma omp parallel for
        for (int j = 0; j < height; ++j)
        {
          for (int i = 0; i < width && et->running; ++i)
          {
620         int npixel;
            #pragma omp atomic capture
            npixel = et->npixel++;
            int ii = et->pixels[npixel].x;
            int jj = et->pixels[npixel].y;
625         int k = jj * width + ii;
            if (glitch[k] < 0)
            {
              // ignore isolated small glitches
              int g = 0;
630           double out[4];
              for (int v = -1; v <= 1; ++v)
              {
                for (int u = -1; u <= 1; ++u)
                {
635               if (u && v) continue;
                  if ((! u) && (! v)) continue;
                  if (ii + u < 0) continue;
```

```
            if (ii + u >= width) continue;
            if (jj + v < 0) continue;
            if (jj + v >= height) continue;
            int kk = (jj + v) * width + (ii + u);
            g += (glitch[kk] < 0);
            out[0] = 1e-30;
            out[1] = output[4 * kk + 1];
            out[2] = output[4 * kk + 2];
            out[3] = output[4 * kk + 3];
          }
        }
        if (g == 0)
        {
          output[4 * k + 0] = out[0];
          output[4 * k + 1] = out[1];
          output[4 * k + 2] = out[2];
          output[4 * k + 3] = out[3];
          glitch[k] = 0;
          #pragma omp atomic
          et->done_pixels++;
        }
        else
        {
          double x = ((ii + dither(ii, jj, 0)) - (i0 + dither(i0, j0, 0))) *↙
              ↪ h;
          double y = ((jj + dither(ii, jj, 1)) - (j0 + dither(i0, j0, 1))) *↙
              ↪ h;
          if (formula->perturbation(m, n, er2, h, x, y, xyzs, out, &et->↙
              ↪ running))
          {
            if (out[0] < 0)
            {
              // the pixel was glitched
              if (ii == i0 && jj == j0)
              {
                // the reference pixel was glitched, how can that be?
                output_pixel(et, npixel, glitch, 0, 1e-30, 0, 0, 0);
                #pragma omp atomic
                et->done_pixels++;
              }
              else
              {
                output_pixel(et, npixel, glitch, out[0], -1, 0, 0, 0);
                glitched = true;
              }
            }
            else
            {
              if (isnan(out[0]) || isinf(out[0]))
              {
                out[0] = 1e-30;
              }
              output_pixel(et, npixel, glitch, 0, out[0], out[1], out[2], ↙
                  ↪ out[3]);
              #pragma omp atomic
              et->done_pixels++;
            }
```

```
                }
              }
            }
          }
        }

        if (glitched && et->running)
        {
          double ma = -1.0/0.0;
          int mi = width/2;
          int mj = height/2;
          for (int j = 0; j < height; ++j)
          {
            for (int i = 0; i < width; ++i)
            {
              int k = j * width + i;
              if (glitch[k] < 0 && ma < glitch[k])
              {
                ma = glitch[k];
                mi = i;
                mj = j;
              }
            }
          }
          double dx = ((mi + dither(mi, mj, 0)) - (i0 + dither(i0, j0, 0))) * h;
          double dy = ((mj + dither(mi, mj, 1)) - (j0 + dither(i0, j0, 1))) * h;
          mpfr_set_d(r, dx, MPFR_RNDN);
          mpfr_add(a, a, r, MPFR_RNDN);
          mpfr_set_d(r, dy, MPFR_RNDN);
          mpfr_add(b, b, r, MPFR_RNDN);
          i0 = mi;
          j0 = mj;
        }
        fprintf(stderr, "\r\t%8d (%d)", width * height - et->done_pixels, ++ref);

      } while (glitched && et->running);
      free(glitch);
      free(xyzs);
      mpfr_clear(a);
      mpfr_clear(b);
      mpfr_clear(r);
      et->active = false;
      return 0;

    }
    else if (e > -16380) // PERTURB LONG DOUBLE
    {

      mpfr_t a, b, r;
      int prec = 53 - mpfr_get_exp(et->radius);
      prec = prec < 53 ? 53 : prec;
      mpfr_init2(a, prec);
      mpfr_init2(b, prec);
      mpfr_init2(r, 53);
      mpfr_set(a, et->centerx, MPFR_RNDN);
      mpfr_set(b, et->centery, MPFR_RNDN);
      mpfr_set(r, et->radius, MPFR_RNDN);
```

```
    long double *xyzs = malloc(n * 3 * sizeof(long double));
    assert(xyzs);
    long double *glitch = calloc(1, width * height * sizeof(long double));
    assert(glitch);
    #pragma omp parallel for
    for (int j = 0; j < height; ++j)
    {
      for (int i = 0; i < width; ++i)
      {
        glitch[j * width + i] = -1;
        output[4 * (j * width + i) + 0] = -1;
        output[4 * (j * width + i) + 1] = -1;
        output[4 * (j * width + i) + 2] = -1;
        output[4 * (j * width + i) + 3] = -1;
      }
    }

    bool glitched = true;
    int ref = 0;
    do
    {

      glitched = false;
      int m = formula->referencel(n, er2, a, b, xyzs, &et->running);
      et->npixel = 0;
      #pragma omp parallel for
      for (int j = 0; j < height; ++j)
      {
        for (int i = 0; i < width && et->running; ++i)
        {
          int npixel;
          #pragma omp atomic capture
          npixel = et->npixel++;
          int ii = et->pixels[npixel].x;
          int jj = et->pixels[npixel].y;
          int k = jj * width + ii;
          if (glitch[k] < 0)
          {
            // ignore isolated small glitches
            int g = 0;
            long double out[4];
            for (int v = -1; v <= 1; ++v)
            {
              for (int u = -1; u <= 1; ++u)
              {
                if (u && v) continue;
                if ((! u) && (! v)) continue;
                if (ii + u < 0) continue;
                if (ii + u >= width) continue;
                if (jj + v < 0) continue;
                if (jj + v >= height) continue;
                int kk = (jj + v) * width + (ii + u);
                g += (glitch[kk] < 0);
                out[0] = 1e-30;
                out[1] = output[4 * kk + 1];
                out[2] = output[4 * kk + 2];
                out[3] = output[4 * kk + 3];
```

```
805                 }
                  }
                  if (g == 0)
                  {
                    output[4 * k + 0] = out[0];
810                 output[4 * k + 1] = out[1];
                    output[4 * k + 2] = out[2];
                    output[4 * k + 3] = out[3];
                    glitch[k] = 0;
                    #pragma omp atomic
815                 et->done_pixels++;
                  }
                  else
                  {
                    //
820                 long double x = ((ii + dither(ii, jj, 0)) - (i0 + dither(i0, j0, ↙
                        ↳ 0))) * h;
                    long double y = ((jj + dither(ii, jj, 1)) - (j0 + dither(i0, j0, ↙
                        ↳ 1))) * h;
                    if (formula->perturbationl(m, n, er2, h, x, y, xyzs, out, &et->↙
                        ↳ running))
                    {
                      if (out[0] < 0)
825                   {
                        // the pixel was glitched
                        if (ii == i0 && jj == j0)
                        {
                          // the reference pixel was glitched, how can that be?
830                       output_pixell(et, npixel, glitch, 0, 1e-30, 0, 0, 0);
                          #pragma omp atomic
                          et->done_pixels++;
                        }
                        else
835                     {
                          output_pixell(et, npixel, glitch, out[0], -1, 0, 0, 0);
                          glitched = true;
                        }
                      }
840                   else
                      {
                        if (isnanl(out[0]) || isinfl(out[0]))
                        {
                          out[0] = 1e-30;
845                     }
                        output_pixell(et, npixel, glitch, 0, out[0], out[1], out[2], ↙
                            ↳ out[3]);
                        #pragma omp atomic
                        et->done_pixels++;
                      }
850                 }
                  }
                }
              }
            }
855
            if (glitched && et->running)
            {
```

```
          long double ma = -1.0/0.0;
          int mi = width/2;
          int mj = height/2;
          for (int j = 0; j < height; ++j)
          {
            for (int i = 0; i < width; ++i)
            {
              int k = j * width + i;
              if (glitch[k] < 0 && ma < glitch[k])
              {
                ma = glitch[k];
                mi = i;
                mj = j;
              }
            }
          }
          long double dx = ((mi + dither(mi, mj, 0)) - (i0 + dither(i0, j0, 0))) *↙
              ↪ h;
          long double dy = ((mj + dither(mi, mj, 1)) - (j0 + dither(i0, j0, 1))) *↙
              ↪ h;
          mpfr_set_ld(r, dx, MPFR_RNDN);
          mpfr_add(a, a, r, MPFR_RNDN);
          mpfr_set_ld(r, dy, MPFR_RNDN);
          mpfr_add(b, b, r, MPFR_RNDN);
          i0 = mi;
          j0 = mj;
        }
        fprintf(stderr, "\r\t%8d (%d)", width * height - et->done_pixels, ++ref);

      } while (glitched && et->running);
      free(glitch);
      free(xyzs);
      mpfr_clear(a);
      mpfr_clear(b);
      mpfr_clear(r);
      et->active = false;
      return 0;

    }
    else
    {
      et->active = false;
      return 0;
    }
}

static void et_start(struct et *et, const mpfr_t cx, const mpfr_t cy, const ↙
    ↪ mpfr_t r, int maxiters)
{
  mpfr_fprintf(stderr, "Re: %Re\nIm: %Re\nSize: %Re\n", cx, cy, r);
  et->maxiters = maxiters;
  int precision = 53 - mpfr_get_exp(r);
  precision = precision > 53 ? precision : 53;
  mpfr_set_prec(et->centerx, precision);
  mpfr_set_prec(et->centery, precision);
  mpfr_set(et->centerx, cx, MPFR_RNDN);
  mpfr_set(et->centery, cy, MPFR_RNDN);
```

```
  mpfr_set(et->radius, r, MPFR_RNDN);
  memset(et->output, 0, 4 * et->width * et->height * sizeof(float));
  et->done_pixels = 0;
  et->running = true;
  et->active = true;
  pthread_create(&et->thread, 0, et_worker, et);
}

static void et_stop(struct et *et, bool force)
{
  if (force) {
    et->running = false;
  }
  pthread_join(et->thread, 0);
}

static bool et_active(const struct et *et)
{
  return et->active;
}

static const float *et_get_output(const struct et *et)
{
  return et->output;
}

static inline int max(int a, int b) {
  return a > b ? a : b;
}

static const char *envs(const char *name, const char *def) {
  const char *e = getenv(name);
  if (e) {
    return e;
  } else {
    return def;
  }
}

static int envi(const char *name, int def) {
  const char *e = getenv(name);
  if (e) {
    return atoi(e);
  } else {
    return def;
  }
}

static double envd(const char *name, double def) {
  const char *e = getenv(name);
  if (e) {
    return atof(e);
  } else {
    return def;
  }
}
```

```
static int envr(mpfr_t out, const char *name, const char *def) {
  const char *e = getenv(name);
  if (e) {
    return mpfr_set_str(out, e,   10, MPFR_RNDN);
  } else {
    return mpfr_set_str(out, def, 10, MPFR_RNDN);
  }
}

static const char *blit_vert =
  "#version 130\n"
  "uniform vec4 bounds;\n"
  "in float vertexID;\n"
  "out vec2 texCoord;\n"
  "void main() {\n"
  "  if (vertexID == 0.0) { gl_Position = vec4(-1.0, -1.0, 0.0, 1.0); texCoord =↙
      ↪ bounds.xy; } else\n"
  "  if (vertexID == 1.0) { gl_Position = vec4( 1.0, -1.0, 0.0, 1.0); texCoord =↙
      ↪ bounds.zy; } else\n"
  "  if (vertexID == 2.0) { gl_Position = vec4(-1.0,  1.0, 0.0, 1.0); texCoord =↙
      ↪ bounds.xw; } else\n"
  "                       { gl_Position = vec4( 1.0,  1.0, 0.0, 1.0); texCoord =↙
      ↪ bounds.zw; }\n"
  "}\n"
  ;

static const char *blit_frag =
  "#version 130\n"
  "uniform sampler2D tex;\n"
  "in vec2 texCoord;\n"
  "out vec4 fragColor;\n"
  "void main() {\n"
  "  fragColor = texture(tex, texCoord);\n"
  "}\n"
  ;

static const char *simple_vert =
  "#version 130\n"
  "in float vertexID;\n"
  "out vec2 texCoord;\n"
  "void main() {\n"
  "  if (vertexID == 0.0) { gl_Position = vec4(-1.0, -1.0, 0.0, 1.0); texCoord =↙
      ↪ vec2(0.0, 1.0); } else\n"
  "  if (vertexID == 1.0) { gl_Position = vec4( 1.0, -1.0, 0.0, 1.0); texCoord =↙
      ↪ vec2(1.0, 1.0); } else\n"
  "  if (vertexID == 2.0) { gl_Position = vec4(-1.0,  1.0, 0.0, 1.0); texCoord =↙
      ↪ vec2(0.0, 0.0); } else\n"
  "                       { gl_Position = vec4( 1.0,  1.0, 0.0, 1.0); texCoord =↙
      ↪ vec2(1.0, 0.0); }\n"
  "}\n"
  ;

static const char *simple_frag =
  "#version 130\n"
  "uniform sampler2D tex;\n"
  "uniform highp float weight;\n"
  "in vec2 texCoord;\n"
```

```
    "out vec4 fragColor;\n"
    "// http://lolengine.net/blog/2013/07/27/rgb-to-hsv-in-glsl\n"
    "vec3 hsv2rgb(vec3 c) {\n"
    "  vec4 K = vec4(1.0, 2.0 / 3.0, 1.0 / 3.0, 3.0);\n"
    "  vec3 p = abs(fract(c.xxx + K.xyz) * 6.0 - K.www);\n"
    "  return c.z * mix(K.xxx, clamp(p - K.xxx, 0.0, 1.0), c.y);\n"
    "}\n"
    "void main() {\n"
    "  vec2 dx = dFdx(texCoord);\n"
    "  vec2 dy = dFdy(texCoord);\n"
    "  vec4 me = texture(tex, texCoord);\n"
    "  vec4 a = texture(tex, texCoord + dx);\n"
    "  vec4 b = texture(tex, texCoord + dy);\n"
    "  vec4 c = texture(tex, texCoord + dx + dy);\n"
    "  float de = me.x;\n"
    "  if (isnan(de) || isinf(de)) de = 0.0;\n"
    "  float s = de > 0.0 ? (me.y != c.y || a.y != b.y) ? 1.0 : 0.25 : 0.0;\n"
    "  float h = max(max(me.y, c.y), max(a.y, b.y)) / 24.618033988749893;\n"
    "  h -= floor(h);\n"
    "  float v = de > 0.0 ? tanh(clamp(de / weight, 0.0, 8.0)) : 1.0;\n"
    "  bool glitch = de < 0.0;\n"
    "  fragColor = vec4(glitch ? vec3(1.0, 0.0, 0.0) : hsv2rgb(vec3(h,s,v)), ↙
      ↪ glitch ? 0.0 : 1.0);\n"
    "}\n"
    ;

struct state_s {
  GLFWwindow *window;
  struct et *context;

  bool should_stop;
  bool should_restart;
  bool should_redraw;
  bool should_save_now;
  bool should_save_when_done;
  bool should_view_morph;
  bool should_view_morph_julia;

  GLuint fbo;

  GLuint blit_vao;
  GLuint blit_p;
  GLint bounds_u;

  GLuint colourize_vao;
  GLuint colourize_p;
  bool show_lines;
  GLint show_lines_u;
  double log2weight;
  GLint weight_u;

  int width;
  int height;

  int precision;
  mpfr_t centerx;
  mpfr_t centery;
```

```
  mpfr_t radius;
  int maxiters;
  double escape_radius_2;

  double srcx0;
  double srcy0;
  double srcx1;
  double srcy1;

  uint8_t *ppm;
  int starttime;
  int saved;
};
typedef struct state_s state_t;
state_t state_d;

static void refresh_callback(void *user_pointer);

static void button_handler(GLFWwindow *window, int button, int action, int mods)↙
    ↳ {
  state_t *state = glfwGetWindowUserPointer(window);
  if (action == GLFW_PRESS) {
    double srcx0 = 0, srcy0 = 0, srcx1 = state->width, srcy1 = state->height;
    double x = 0, y = 0;
    glfwGetCursorPos(window, &x, &y);
    mpfr_t cx, cy, r;
    mpfr_init2(cx, state->precision);
    mpfr_init2(cy, state->precision);
    mpfr_init2(r, 53);
    mpfr_set(cx, state->centerx, MPFR_RNDN);
    mpfr_set(cy, state->centery, MPFR_RNDN);
    mpfr_set(r, state->radius, MPFR_RNDN);
    double w = state->width;
    double h = state->height;
    double yy = h - y;
    double dx = 2 * ((x + 0.5) / w - 0.5) * (w / h);
    double dy = 2 * ((y + 0.5) / h - 0.5);
    mpfr_t ddx, ddy;
    mpfr_init2(ddx, 53);
    mpfr_init2(ddy, 53);
    mpfr_mul_d(ddx, r, dx, MPFR_RNDN);
    mpfr_mul_d(ddy, r, dy, MPFR_RNDN);
    switch (button) {
      case GLFW_MOUSE_BUTTON_LEFT:
        mpfr_mul_2si(ddx, ddx, -1, MPFR_RNDN);
        mpfr_mul_2si(ddy, ddy, -1, MPFR_RNDN);
        mpfr_add(cx, cx, ddx, MPFR_RNDN);
        mpfr_add(cy, cy, ddy, MPFR_RNDN);
        mpfr_mul_2si(r, r, -1, MPFR_RNDN);
        state->precision += 1;
        mpfr_set_prec(state->centerx, state->precision);
        mpfr_set_prec(state->centery, state->precision);
        mpfr_set(state->centerx, cx, MPFR_RNDN);
        mpfr_set(state->centery, cy, MPFR_RNDN);
        mpfr_set(state->radius, r, MPFR_RNDN);
        state->should_restart = true;
        srcx0 = 0.5 * (srcx0 - x) + x;
```

```
1130            srcx1 = 0.5 * (srcx1 - x) + x;
                srcy0 = 0.5 * (srcy0 - yy) + yy;
                srcy1 = 0.5 * (srcy1 - yy) + yy;
                break;
            case GLFW_MOUSE_BUTTON_RIGHT:
1135            mpfr_sub(cx, cx, ddx, MPFR_RNDN);
                mpfr_sub(cy, cy, ddy, MPFR_RNDN);
                mpfr_mul_2si(r, r, 1, MPFR_RNDN);
                state->precision -= 1;
                mpfr_set_prec(state->centerx, state->precision);
1140            mpfr_set_prec(state->centery, state->precision);
                mpfr_set(state->centerx, cx, MPFR_RNDN);
                mpfr_set(state->centery, cy, MPFR_RNDN);
                mpfr_set(state->radius, r, MPFR_RNDN);
                state->should_restart = true;
1145            srcx0 = 2.0 * (srcx0 - x) + x;
                srcx1 = 2.0 * (srcx1 - x) + x;
                srcy0 = 2.0 * (srcy0 - yy) + yy;
                srcy1 = 2.0 * (srcy1 - yy) + yy;
                break;
1150            case GLFW_MOUSE_BUTTON_MIDDLE:
                mpfr_add(cx, cx, ddx, MPFR_RNDN);
                mpfr_add(cy, cy, ddy, MPFR_RNDN);
                mpfr_set(state->centerx, cx, MPFR_RNDN);
                mpfr_set(state->centery, cy, MPFR_RNDN);
1155            state->should_restart = true;
                srcx0 -= state->width / 2 - x;
                srcx1 -= state->width / 2 - x;
                srcy0 -= state->height / 2 - yy;
                srcy1 -= state->height / 2 - yy;
1160            break;
        }
        mpfr_clear(cx);
        mpfr_clear(cy);
        mpfr_clear(r);
1165    mpfr_clear(ddx);
        mpfr_clear(ddy);
        glBindFramebuffer(GL_READ_FRAMEBUFFER, 0);
        glBindFramebuffer(GL_DRAW_FRAMEBUFFER, state->fbo);
        glBlitFramebuffer(0, 0, state->width, state->height, 0, 0, state->width, ↙
            ↳ state->height, GL_COLOR_BUFFER_BIT, GL_NEAREST);
1170    glBindFramebuffer(GL_DRAW_FRAMEBUFFER, 0);

        state->srcx0 = srcx0;
        state->srcy0 = srcy0;
        state->srcx1 = srcx1;
1175    state->srcy1 = srcy1;
    }
    (void) mods;
    }

1180 static void key_handler(GLFWwindow *window, int key, int scancode, int action, ↙
        ↳ int mods) {
    state_t *state = glfwGetWindowUserPointer(window);
    double srcx0 = 0, srcy0 = 0, srcx1 = state->width, srcy1 = state->height;
    double x = state->width/2, y = state->height/2;
    bool capture = false;
```

```
1185    if (action == GLFW_PRESS) {
          switch (key) {
            case GLFW_KEY_Q:
              glfwSetWindowShouldClose(window, GL_TRUE);
              break;
1190        case GLFW_KEY_ESCAPE:
              state->should_stop = true;
              break;
            case GLFW_KEY_J:
              state->should_view_morph = true;
1195          state->should_view_morph_julia = true;
              break;
            case GLFW_KEY_M:
              state->should_view_morph = true;
              state->should_view_morph_julia = false;
1200          break;
            case GLFW_KEY_C:
              glBindFramebuffer(GL_FRAMEBUFFER, state->fbo);
              glClear(GL_COLOR_BUFFER_BIT);
              glBindFramebuffer(GL_FRAMEBUFFER, 0);
1205          glClear(GL_COLOR_BUFFER_BIT);
              state->should_redraw = true;
              break;
            case GLFW_KEY_S:
              if (mods & GLFW_MOD_SHIFT) {
1210            state->should_save_now = true;
              } else {
                state->should_save_when_done = true;
              }
              break;
1215        case GLFW_KEY_9:
              state->log2weight -= 0.25;
              state->should_redraw = true;
              fprintf(stderr, "W = %g\n", state->log2weight);
              break;
1220        case GLFW_KEY_0:
              state->log2weight += 0.25;
              state->should_redraw = true;
              fprintf(stderr, "W = %g\n", state->log2weight);
              break;
1225        case GLFW_KEY_MINUS:
              state->maxiters >>= 1;
              state->maxiters = state->maxiters < 16 ? 16 : state->maxiters;
              fprintf(stderr, "N = %d\n", state->maxiters);
              state->should_restart = true;
1230          break;
            case GLFW_KEY_EQUAL:
              state->maxiters <<= 1;
              state->maxiters = state->maxiters > 1 << 30 ? 1 << 30 : state->maxiters;
              fprintf(stderr, "N = %d\n", state->maxiters);
1235          state->should_restart = true;
              break;
            case GLFW_KEY_PAGE_UP:
              mpfr_mul_2si(state->radius, state->radius, -1, MPFR_RNDN);
              state->precision += 1;
1240          mpfr_prec_round(state->centerx, state->precision, MPFR_RNDN);
              mpfr_prec_round(state->centery, state->precision, MPFR_RNDN);
```

```
            state->should_restart = true;
            srcx0 = 0.5 * (srcx0 - x) + x;
            srcx1 = 0.5 * (srcx1 - x) + x;
            srcy0 = 0.5 * (srcy0 - y) + y;
            srcy1 = 0.5 * (srcy1 - y) + y;
            capture = true;
            state->should_restart = true;
            break;
          case GLFW_KEY_PAGE_DOWN:
            mpfr_mul_2si(state->radius, state->radius, 1, MPFR_RNDN);
            state->precision -= 1;
            mpfr_prec_round(state->centerx, state->precision, MPFR_RNDN);
            mpfr_prec_round(state->centery, state->precision, MPFR_RNDN);
            state->should_restart = true;
            srcx0 = 2.0 * (srcx0 - x) + x;
            srcx1 = 2.0 * (srcx1 - x) + x;
            srcy0 = 2.0 * (srcy0 - y) + y;
            srcy1 = 2.0 * (srcy1 - y) + y;
            capture = true;
            state->should_restart = true;
            break;
        }
      }
      if (capture)
      {
        glBindFramebuffer(GL_READ_FRAMEBUFFER, 0);
        glBindFramebuffer(GL_DRAW_FRAMEBUFFER, state->fbo);
        glBlitFramebuffer(0, 0, state->width, state->height, 0, 0, state->width, ↙
            ↳ state->height, GL_COLOR_BUFFER_BIT, GL_NEAREST);
        glBindFramebuffer(GL_DRAW_FRAMEBUFFER, 0);
        state->srcx0 = srcx0;
        state->srcy0 = srcy0;
        state->srcx1 = srcx1;
        state->srcy1 = srcy1;
      }

    (void) scancode;
    }

    static void refresh_callback(void *user_pointer) {
      state_t *state = user_pointer;
      assert(state);
      glBindVertexArray(state->blit_vao);
      glUseProgram(state->blit_p);
      glUniform4f(state->bounds_u, state->srcx0 / state->width, state->srcy0 / state↙
          ↳ ->height, state->srcx1 / state->width, state->srcy1 / state->height);
      glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);
      glBindVertexArray(state->colourize_vao);
      glUseProgram(state->colourize_p);
      glActiveTexture(GL_TEXTURE0);
      glTexSubImage2D(GL_TEXTURE_2D, 0, 0, 0, state->width, state->height, GL_RGBA, ↙
          ↳ GL_FLOAT, et_get_output(state->context));
      glUniform1i(state->show_lines_u, state->show_lines);
      glUniform1f(state->weight_u, exp2f(state->log2weight));
      glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);
      glfwSwapBuffers(state->window);
    }
```

```
static void handle_view_morph(struct et *context, state_t *state) {
  (void) context;
  if (state->should_view_morph) {
    state->should_view_morph = false;
    volatile int running = 1;
    int p = formula->period(state->maxiters, state->escape_radius_2, state->↙
        ↪ centerx, state->centery, state->radius, &running);
    if (p > 0)
    {
      fprintf(stderr, "p = %d\n", p);
      int precision = mpfr_get_prec(state->centerx);
      mpfr_t a, b, r, d;
      mpfr_init2(a, 4 * precision);
      mpfr_init2(b, 4 * precision);
      mpfr_init2(r, 53);
      mpfr_init2(d, 53);
      mpfr_set(a, state->centerx, MPFR_RNDN);
      mpfr_set(b, state->centery, MPFR_RNDN);
      bool ok = formula->newton(64, p, a, b, &running);
      if (ok)
      {
        ok = formula->size(p, a, b, r, &running);
        if (ok)
        {
          mpfr_fprintf(stderr, "s = %Re\n", r);
          if (state->should_view_morph_julia)
          {
            double n = formula->degree;
            mpfr_set_d(d, (n + 1) * (n - 1) / (n * n), MPFR_RNDN);
            mpfr_pow(r, r, d, MPFR_RNDN);
          }
          mpfr_mul_si(r, r, 4, MPFR_RNDN);
          mpfr_fprintf(stderr, "r = %Re\n", r);
          int e = mpfr_get_exp(r);
          int prec = max(53, 53 - e);
          state->precision = prec;
          mpfr_set_prec(state->centerx, prec);
          mpfr_set_prec(state->centery, prec);
          mpfr_set(state->centerx, a, MPFR_RNDN);
          mpfr_set(state->centery, b, MPFR_RNDN);
          mpfr_set(state->radius, r, MPFR_RNDN);
          state->should_restart = true;
        }
      }
      mpfr_clear(a);
      mpfr_clear(b);
      mpfr_clear(r);
      mpfr_clear(d);
    }
  }
}

void debug_program(GLuint program, const char *name) {
  if (program) {
    GLint linked = GL_FALSE;
    glGetProgramiv(program, GL_LINK_STATUS, &linked);
```

```
    if (linked != GL_TRUE) {
      fprintf(stderr, "%s: OpenGL shader program link failed\n", name);
    }
    GLint length = 0;
    glGetProgramiv(program, GL_INFO_LOG_LENGTH, &length);
    char *buffer = (char *) malloc(length + 1);
    glGetProgramInfoLog(program, length, NULL, buffer);
    buffer[length] = 0;
    if (buffer[0]) {
      fprintf(stderr, "%s: OpenGL shader program info log\n", name);
      fprintf(stderr, "%s\n", buffer);
    }
    free(buffer);
  } else {
    fprintf(stderr, "%s: OpenGL shader program creation failed\n", name);
  }
}

void debug_shader(GLuint shader, GLenum type, const char *name) {
  const char *tname = 0;
  switch (type) {
    case GL_VERTEX_SHADER:   tname = "vertex";   break;
    case GL_GEOMETRY_SHADER: tname = "geometry"; break;
    case GL_FRAGMENT_SHADER: tname = "fragment"; break;
    default:                 tname = "unknown";  break;
  }
  if (shader) {
    GLint compiled = GL_FALSE;
    glGetShaderiv(shader, GL_COMPILE_STATUS, &compiled);
    if (compiled != GL_TRUE) {
      fprintf(stderr, "%s: OpenGL %s shader compile failed\n", name, tname);
    }
    GLint length = 0;
    glGetShaderiv(shader, GL_INFO_LOG_LENGTH, &length);
    char *buffer = (char *) malloc(length + 1);
    glGetShaderInfoLog(shader, length, NULL, buffer);
    buffer[length] = 0;
    if (buffer[0]) {
      fprintf(stderr, "%s: OpenGL %s shader info log\n", name, tname);
      fprintf(stderr, "%s\n", buffer);
    }
    free(buffer);
  } else {
    fprintf(stderr, "%s: OpenGL %s shader creation failed\n", name, tname);
  }
}

void compile_shader(GLint program, GLenum type, const char *name, const GLchar *↙
    ↳ source) {
  GLuint shader = glCreateShader(type);
  glShaderSource(shader, 1, &source, NULL);
  glCompileShader(shader);
  debug_shader(shader, type, name);
  glAttachShader(program, shader);
  glDeleteShader(shader);
}
```

```
GLint compile_program(const char *name, const GLchar *vert, const GLchar *frag) ↙
    ↳ {
  GLint program = glCreateProgram();
  if (vert) { compile_shader(program, GL_VERTEX_SHADER  , name, vert); }
  if (frag) { compile_shader(program, GL_FRAGMENT_SHADER, name, frag); }
  glBindAttribLocation(program, 0, "vertexID");
  glLinkProgram(program);
  debug_program(program, name);
  return program;
}


bool read_ppm_comment(state_t *state, const char *filename)
{
  FILE *file = fopen(filename, "rb");
  if (! file) return false;
  bool ok = false;
  char *sx = 0, *sy = 0, *sz = 0;
  if ((ok = (3 == fscanf(file, "P6\n# Re: %ms\n# Im: %ms\n# Size: %ms\n", &sx, &↙
      ↳ sy, &sz))))
  {
    if ((ok = (0 == mpfr_set_str(state->radius, sz, 0, MPFR_RNDN))))
    {
      int prec = 53 - mpfr_get_exp(state->radius);
      if (prec < 53) prec = 53;
      mpfr_set_prec(state->centerx, prec);
      mpfr_set_prec(state->centery, prec);
      ok &= (0 == mpfr_set_str(state->centerx, sx, 0, MPFR_RNDN));
      ok &= (0 == mpfr_set_str(state->centery, sy, 0, MPFR_RNDN));
    }
  }
  if (sx) free(sx);
  if (sy) free(sy);
  if (sz) free(sz);
  fclose(file);
  return ok;
}

void save_screenshot(state_t *state, bool to_stdout)
{
  FILE *file = stdout;
  if (! to_stdout)
  {
    char filename[100];
    snprintf(filename, 100, "%08x_%04d.ppm", state->starttime, state->saved++);
    file = fopen(filename, "wb");
    fprintf(stderr, "saving: '%s'\n", filename);
  }
  glReadPixels(0, 0, state->width, state->height, GL_RGB, GL_UNSIGNED_BYTE, ↙
      ↳ state->ppm);
  if (to_stdout)
    fprintf(file, "P6\n%d %d\n255\n", state->width, state->height);
  else
    mpfr_fprintf(file, "P6\n# Re: %Re\n# Im: %Re\n# Size: %Re\n%d %d\n255\n", ↙
        ↳ state->centerx, state->centery, state->radius, state->width, state->↙
        ↳ height);
  for (int y = state->height - 1; y >= 0; --y) {
```

```
1460      fwrite(state->ppm + y * state->width * 3, state->width * 3, 1, file);
        }
        fflush(file);
        if (! to_stdout)
        {
1465      fclose(file);
        }
      }

      extern int main(int argc, char **argv) {
1470    state_t *state = &state_d;
        memset(state, 0, sizeof(*state));
        state->starttime = time(0);

        const char *name = envs("formula", "Mandelbrot");
1475    int p = envi("p", 2);
        int q = envi("q", 1);
        char libso[1024];
        snprintf(libso, 1000, "./lib/%s_p%d_q%d.so", name, p, q);
        void *library = dlopen(libso, RTLD_NOW);
1480    assert(library);
        formula = dlsym(library, "et");
        assert(formula);
        assert(formula->magic == MAGIC);
        assert(formula->ssize == SIZE);
1485    assert(formula->version == VERSION);

        int width = envi("width", 640);
        int height = envi("height", 360);
        int maxiters = envi("maxiters", 1 << 8);
1490    double escape_radius = envd("escaperadius", 10000);
        int precision = envi("precision", 53);
        int zoom_start = envi("zoom_start", 0);
        int zoom_count = envi("zoom_count", 0);
        double weight = envd("weight", 0);
1495
        mpfr_init2(state->radius, 53);
        envr(state->radius, "radius", "2.0");

        int e = max(53, 53 - mpfr_get_exp(state->radius));
1500    if (e > precision) {
          fprintf(stderr, "WARNING: increasing precision to %d\n", e);
          precision = e;
        }

1505    mpfr_init2(state->centerx, precision);
        mpfr_init2(state->centery, precision);
        envr(state->centerx, "real", "0.0");
        envr(state->centery, "imag", "0.0");

1510    glfwInit();
        glfwWindowHint(GLFW_CLIENT_API, GLFW_OPENGL_API);
        glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 2);
        glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 1);
        glfwWindowHint(GLFW_RESIZABLE, GL_FALSE);
1515    state->window = glfwCreateWindow(width, height, "et", 0, 0);
        glfwMakeContextCurrent(state->window);
```

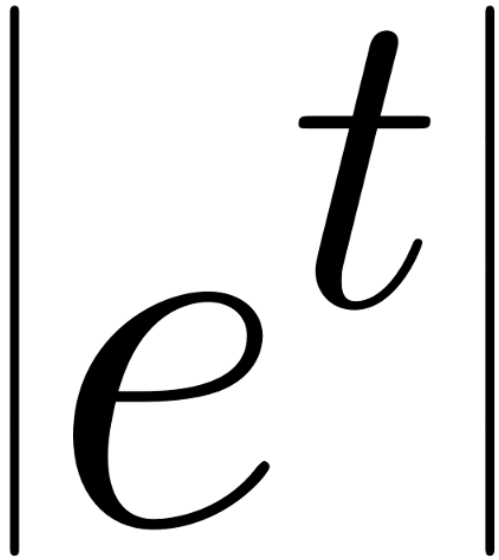
```
    glewExperimental = GL_TRUE;
    glewInit();
    glGetError(); // discard common error from glew

    glClearColor(1.0, 0.0, 0.0, 1.0);
    glDisable(GL_DEPTH_TEST);
    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

    GLuint ftex;
    glGenTextures(1, &ftex);
    glActiveTexture(GL_TEXTURE1);
    glBindTexture(GL_TEXTURE_2D, ftex);
//  printf("%dx%d GL_RGBA\n", width, height);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width, height, 0, GL_RGBA, ↙
        ↳ GL_UNSIGNED_BYTE, 0);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
    glGenFramebuffers(1, &state->fbo);
    glBindFramebuffer(GL_FRAMEBUFFER, state->fbo);
    glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, ↙
        ↳ ftex, 0);
    glClear(GL_COLOR_BUFFER_BIT);
    glBindFramebuffer(GL_FRAMEBUFFER, 0);
    glClear(GL_COLOR_BUFFER_BIT);

    GLuint tex;
    glGenTextures(1, &tex);
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, tex);
//  printf("%dx%d GL_RGBA32F\n", width, height);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA32F, width, height, 0, GL_RGBA, GL_FLOAT↙
        ↳ , 0);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_MIRRORED_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_MIRRORED_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);

    GLuint vbo;
    GLubyte vbo_data[4] = { 0, 1, 2, 3 };
    glGenBuffers(1, &vbo);
    glBindBuffer(GL_ARRAY_BUFFER, vbo);
    glBufferData(GL_ARRAY_BUFFER, sizeof(vbo_data), vbo_data, GL_STATIC_DRAW);

    glGenVertexArrays(1, &state->blit_vao);
    glBindVertexArray(state->blit_vao);
    state->blit_p = compile_program("blit", blit_vert, blit_frag);
    glUseProgram(state->blit_p);
    glUniform1i(glGetUniformLocation(state->blit_p, "tex"), 1);
    state->bounds_u = glGetUniformLocation(state->blit_p, "bounds");
    GLint attr = glGetAttribLocation(state->blit_p, "vertexID");
    glVertexAttribPointer(attr, 1, GL_UNSIGNED_BYTE, GL_FALSE, 0, 0);
    glEnableVertexAttribArray(attr);
    glBindVertexArray(0);

```

```
    glGenVertexArrays(1, &state->colourize_vao);
    glBindVertexArray(state->colourize_vao);
    state->colourize_p = compile_program("colourize", simple_vert, simple_frag);
    glUseProgram(state->colourize_p);
    state->weight_u = glGetUniformLocation(state->colourize_p, "weight");
    attr = glGetAttribLocation(state->colourize_p, "vertexID");
    glVertexAttribPointer(attr, 1, GL_UNSIGNED_BYTE, GL_FALSE, 0, 0);
    glEnableVertexAttribArray(attr);

    state->ppm = malloc(width * height * 3);
    assert(state->ppm);

    state->should_stop = false;
    state->should_restart = false;
    state->should_redraw = false;
    state->should_save_now = false;
    state->should_save_when_done = false;
    state->width = width;
    state->height = height;
    state->precision = precision;
    state->log2weight = weight;
    state->maxiters = maxiters;
    state->escape_radius_2 = escape_radius * escape_radius;
    state->srcx0 = 0;
    state->srcy0 = 0;
    state->srcx1 = state->width;
    state->srcy1 = state->height;

    state->context = et_new(width, height);

    glfwSetWindowUserPointer(state->window, state);

    if (argc > 1)
    {
      for (int a = 1; a < argc; ++a)
      {
        if (read_ppm_comment(state, argv[a]))
        {
          fprintf(stderr, "START '%s'\n", argv[a]);
          et_start(state->context, state->centerx, state->centery, state->radius, ↙
              ↳ state->maxiters);
          et_stop(state->context, false);
          refresh_callback(state);
          save_screenshot(state, false);
        }
      }
    } else if (zoom_count) {
      mpfr_set_d(state->radius, 256, MPFR_RNDN);
      mpfr_div_2exp(state->radius, state->radius, zoom_start, MPFR_RNDN);
      for (int z = zoom_start; z < zoom_count; ++z) {
        glfwPollEvents();
        if (glfwWindowShouldClose(state->window)) {
          break;
        }
        fprintf(stderr, "%8d FRAME\n", z);
        et_start(state->context, state->centerx, state->centery, state->radius, ↙
            ↳ state->maxiters);
```

```
      et_stop(state->context, false);
      refresh_callback(state);
      save_screenshot(state, true);
      mpfr_div_2exp(state->radius, state->radius, 1, MPFR_RNDN);
    }

  } else {
    glfwSetMouseButtonCallback(state->window, button_handler);
    glfwSetKeyCallback(state->window, key_handler);

    bool first = true;
    while (! glfwWindowShouldClose(state->window)) {
      int e = glGetError();
      if (e)
        fprintf(stderr, "E: %d\n", e);

      state->should_restart = false;
//    fprintf(stderr, "start\n");
      if (! first) {
        et_stop(state->context, true);
      }
      first = false;
      et_start(state->context, state->centerx, state->centery, state->radius, ↙
          ↳ state->maxiters);

//    fprintf(stderr, "wait_timeout\n");
      while (et_active(state->context)) {
        struct timespec delta = { 0, 33333333 };
        nanosleep(&delta, 0);
//      fprintf(stderr, "refresh\n");
        glActiveTexture(GL_TEXTURE0);
//      printf("%dx%d\n", state->width, state->height);
        glTexSubImage2D(GL_TEXTURE_2D, 0, 0, 0, state->width, state->height, ↙
            ↳ GL_RGBA, GL_FLOAT, et_get_output(state->context));
        refresh_callback(state);

        glfwPollEvents();
        if (glfwWindowShouldClose(state->window)) {
          break;
        }

        if (state->should_save_now) {
          state->should_save_now = false;
          save_screenshot(state, false);
        }

        handle_view_morph(state->context, state);

        if (state->should_restart) {
          state->should_restart = false;
//        fprintf(stderr, "start\n");
          et_stop(state->context, true);
          et_start(state->context, state->centerx, state->centery, state->radius↙
              ↳ , state->maxiters);
        }
//      fprintf(stderr, "wait_timeout\n");
        if (state->should_stop) {
```

```
1680            et_stop(state->context, true);
                state->should_stop = false;
              }
            }

1685        if (glfwWindowShouldClose(state->window)) {
              break;
            }

            refresh_callback(state);
1690
            while (! state->should_restart) {

              if (state->should_save_now || state->should_save_when_done) {
                state->should_save_now = false;
1695            state->should_save_when_done = false;
                save_screenshot(state, false);
              }

              glfwWaitEvents();
1700          if (glfwWindowShouldClose(state->window)) {
                break;
              }

      //        if (state->should_redraw) {
1705  //          state->should_redraw = false;
      //          fprintf(stderr, "refresh\n");
              refresh_callback(state);
    //          glfwSwapBuffers(state->window);
      //        }
1710
              handle_view_morph(state->context, state);

            }

1715      }

      //  fprintf(stderr, "delete\n");
          et_stop(state->context, true);
        }
1720

        et_delete(state->context);
        free(state->ppm);
        glfwDestroyWindow(state->window);
1725    glfwTerminate();

        return 0;
      (void) argc;
      (void) argv;
1730  }
```

21 src/et.png



22 src/formula.h

```
/*
et -- escape time fractals
Copyright (C) 2018,2019 Claude Heiland-Allen

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU Affero General Public License as
published by the Free Software Foundation, either version 3 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 Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License
along with this program.  If not, see <https://www.gnu.org/licenses/>.
*/

#ifndef ET_FORMULA_H
#define ET_FORMULA_H 1

#include <assert.h>
#include <math.h>
#include <mpfr.h>

#define abs(x) ((x)<0?-(x):(x))
#define sgn(x) (((x)>=0)-((x)<0))

/*
typedef double compensated[2];
```

```
static inline void c_set_d(compensated k, double a)
{
  k[0] = a;
  k[1] = 0;
}

static inline void c_set_i(compensated k, int a)
{
  k[0] = a;
  k[1] = 0;
}

static inline void c_set_c(compensated k, const compensated a)
{
  k[0] = a[0];
  k[1] = a[1];
}

static inline void c_add_dd(compensated k, double a, double b)
{
  double x = a + b;
  double z = x - a;
  double y = (a - (x - z)) + (b - z);
  k[0] = x;
  k[1] = y;
}

static inline void c_split_d(compensated k, double a)
{
  static const double magic = 134217729;
  double c = magic * a;
  double x = c - (c - a);
  double y = a - x;
  k[0] = x;
  k[1] = y;
}

static inline void c_add_cc(compensated k, const compensated ab, const ↙
    ↳ compensated cd)
{
  compensated k1; c_add_dd(k1, ab[0], cd[0]);
  compensated k2; c_add_dd(k2, k1[1], cd[1]);
  compensated k3; c_add_dd(k3, ab[1], k2[0]);
  compensated k4; c_add_dd(k4, k1[0], k3[0]);
                  c_add_dd(k, k4[0], k2[1] + k3[1] + k4[1]);
}

static inline void c_add_ci(compensated k, const compensated ab, int c)
{
  compensated cd;
  c_set_i(cd, c);
  c_add_cc(k, ab, cd);
}

static inline void c_add_ic(compensated k, int a, const compensated cd)
{
```

```
  compensated ab;
  c_set_i(ab, a);
  c_add_cc(k, ab, cd);
}

static inline void c_mul2_c(compensated k, const compensated ab)
{
  k[0] = ab[0] * 2;
  k[1] = ab[1] * 2;
}

static inline void c_mul_dd(compensated k, double a, double b)
{
  compensated k1, k2;
  c_split_d(k1, a);
  c_split_d(k2, b);
  double x = a * b;
  double a1 = k1[0];
  double a2 = k1[1];
  double b1 = k2[0];
  double b2 = k2[1];
  k[0] = x;
  k[1] = a2*b2 - (((x - a1*b1) - a2*b1) - a1*b2);
}

static inline void c_mul_cc(compensated k, const compensated ab, const ↙
    ↳ compensated cd)
{
  compensated k1,k2,k3,k4,k5,k6,k7,k8;
  c_mul_dd(k1, ab[0], cd[0]);
  c_mul_dd(k2, ab[1], cd[0]);
  c_mul_dd(k3, ab[0], cd[1]);
  c_add_dd(k4, k1[0], k2[0]);
  c_add_dd(k5, k3[0], k4[0]);
  c_add_dd(k6, k1[1], k4[1]);
  c_add_dd(k7, k5[1], k6[0]);
  c_add_dd(k8, k5[0], k7[0]);
  c_add_dd(k, k8[0], ab[1] * cd[1] + k2[1] + k3[1] + k6[1] + k7[1] + k8[1]);
}

static inline void c_mul_ic(compensated k, int a, const compensated cd)
{
  // FIXME optimize
  compensated ab;
  c_set_i(ab, a);
  c_mul_cc(k, ab, cd);
}

static inline void c_mul_ci(compensated k, const compensated ab, int c)
{
  // FIXME optimize
  compensated cd;
  c_set_i(cd, c);
  c_mul_cc(k, ab, cd);
}

static inline void c_inv_c(compensated k, const compensated ab)
```

```
{
  c_add_dd(k, 1 / ab[0], -ab[1] / (ab[0] * ab[0]));
}

static inline void c_div_cc(compensated k, const compensated ab, const ↙
    ↳ compensated cd)
{
  // FIXME accuracy
  compensated k1;
  c_inv_c(k1, cd);
  c_mul_cc(k, ab, k1);
}

static inline void c_sqr_d(compensated k, double a)
{
  compensated k1;
  c_split_d(k1, a);
  double x = a * a;
  double a1 = k1[0];
  double a2 = k1[1];
  k[0] = x;
  k[1] = a2*a2 - ((x - a1*a1) - 2*(a2*a1));
}

static inline void c_sqr_c(compensated k, const compensated ab)
{
  // FIXME optimize
  c_mul_cc(k, ab, ab);
}

static inline void c_sqrt_c(compensated k, const compensated m)
{
  compensated z0; c_set_d(z0, sqrt(m[0]));
  compensated w0; c_div_cc(w0, m, z0);
  compensated z1; c_add_cc(z1, z0, w0); z1[0] /= 2; z1[1] /= 2;
  compensated w1; c_div_cc(w1, m, z1);
  compensated z2; c_add_cc(z2, z1, w1); z2[0] /= 2; z2[1] /= 2;
  compensated w2; c_div_cc(w2, m, z2);
  compensated z3; c_add_cc(z3, z2, w2); z3[0] /= 2; z3[1] /= 2;
  compensated w3; c_div_cc(w3, m, z3);
  compensated z4; c_add_cc(z4, z3, w3); z4[0] /= 2; z4[1] /= 2;
  compensated w4; c_div_cc(w4, m, z4);
                  c_add_cc(k, z4, w4); k[0] /= 2; k[1] /= 2;
}

static inline void c_floor_c(compensated k, const compensated ab)
{
  c_add_dd(k, floor(ab[0]), floor(ab[1]));
}

static inline void c_neg_c(compensated k, const compensated ab)
{
  k[0] = -ab[0];
  k[1] = -ab[1];
}

static inline void c_abs_c(compensated k, const compensated ab)
```

```
{
  if (ab[0] < 0) c_neg_c(k, ab); else c_set_c(k, ab);
}

static inline void c_log1p_c(compensated k, const compensated ab)
{
  // FIXME accuracy
  c_set_d(k, log1p(ab[0]));
}

#define i_sgn_c(o,a) (o)=sgn((a)[0])

static inline void c_read_Ci(compensated k, const compensated *p, int i)
{
  k[0] = p[i][0];
  k[1] = p[i][1];
}

static inline void v_write_Cic(compensated *p, int i, const compensated k)
{
  p[i][0] = k[0];
  p[i][1] = k[1];
}

#define b_lt_cc(o,a,b) (o)=(((a)[0]==(b)[0])?((a)[1]<(b)[1]):((a)[0]<(b)[0]))
*/

#define b_read_Vi(a,b,c) (a)=(b)[(c)]
#define v_write_Fif(b,c,d) (b)[(c)]=(d)
#define v_write_Did(b,c,d) (b)[(c)]=(d)
#define v_write_Lil(b,c,d) (b)[(c)]=(d)
#define b_and_bb(a,b,c) (a)=(b)&&(c)
#define b_or_bb(a,b,c) (a)=(b)||(c)
#define b_set_b(a,b) (a)=(b)
#define b_set_i(a,b) (a)=(b)
#define i_set_i(a,b) (a)=(b)
#define i_neg_i(a,b) (a)=-(b)
#define i_add_ii(a,b,c) (a)=(b)+(c)
#define i_mul_ii(a,b,c) (a)=(b)*(c)
#define i_ifte_bii(a,c,t,f) (a)=(c)?(t):(f)

#define f_set_i(a,b) (a)=(b)
#define f_set_f(a,b) (a)=(b)
#define f_sqr_f(a,b) (a)=(b)*(b)
#define f_neg_f(a,b) (a)=(b)
#define f_abs_f(a,b) (a)=abs((b))
#define i_sgn_f(a,b) (a)=sgn((b))
#define f_mul2_f(a,b) (a)=(b)+(b)
#define f_add_if(a,b,c) (a)=(b)+(c)
#define f_add_ff(a,b,c) (a)=(b)+(c)
#define f_mul_fi(a,b,c) (a)=(b)*(c)
#define f_mul_if(a,b,c) (a)=(b)*(c)
#define f_mul_ff(a,b,c) (a)=(b)*(c)
#define f_read_Fi(a,b,c) (a)=(b)[(c)]
#define f_diffabs_ff(a,b,c) (a)=diffabsf((b),(c))
#define f_sqrt_f(a,b) (a)=sqrtf((b))
#define f_log1p_f(a,b) (a)=log1pf((b))
```

```
#define f_floor_f(a,b) (a)=floorf((b))
#define f_inv_f(a,b) (a)=1.0f/(b)

#define d_read_Di(a,b,c) (a)=(b)[(c)]
#define d_set_i(a,b) (a)=(b)
#define d_set_d(a,b) (a)=(b)
#define d_neg_d(a,b) (a)=-(b)
#define d_abs_d(a,b) (a)=abs((b))
#define i_sgn_d(a,b) (a)=sgn((b))
#define d_sqr_d(a,b) (a)=(b)*(b)
#define d_mul2_d(a,b) (a)=(b)+(b)
#define d_add_di(a,b,c) (a)=(b)+(c)
#define d_add_id(a,b,c) (a)=(b)+(c)
#define d_add_dd(a,b,c) (a)=(b)+(c)
#define d_mul_di(a,b,c) (a)=(b)*(c)
#define d_mul_id(a,b,c) (a)=(b)*(c)
#define d_mul_dd(a,b,c) (a)=(b)*(c)
#define d_log1p_d(a,b) (a)=log1p((b))
#define d_sqrt_d(a,b) (a)=sqrt((b))
#define d_floor_d(a,b) (a)=floor((b))
#define d_diffabs_dd(a,b,c) (a)=diffabs((b),(c))
#define i_sgn_d(a,b) (a)=sgn((b))
#define d_inv_d(a,b) (a)=1.0/(b)
#define d_inv_i(a,b) (a)=1.0/(b)

#define l_read_Li(a,b,c) (a)=(b)[(c)]
#define l_set_i(a,b) (a)=(b)
#define l_set_l(a,b) (a)=(b)
#define l_neg_l(a,b) (a)=-(b)
#define l_abs_l(a,b) (a)=abs((b))
#define i_sgn_l(a,b) (a)=sgn((b))
#define l_sqr_l(a,b) (a)=(b)*(b)
#define l_mul2_l(a,b) (a)=(b)+(b)
#define l_add_li(a,b,c) (a)=(b)+(c)
#define l_add_il(a,b,c) (a)=(b)+(c)
#define l_add_ll(a,b,c) (a)=(b)+(c)
#define l_mul_li(a,b,c) (a)=(b)*(c)
#define l_mul_il(a,b,c) (a)=(b)*(c)
#define l_mul_ll(a,b,c) (a)=(b)*(c)
#define l_log1p_l(a,b) (a)=log1pl((b))
#define l_sqrt_l(a,b) (a)=sqrtl((b))
#define l_floor_l(a,b) (a)=floorl((b))
#define l_diffabs_ll(a,b,c) (a)=diffabsl((b),(c))
#define i_sgn_l(a,b) (a)=sgn((b))
#define l_inv_l(a,b) (a)=1.0l/(b)
#define l_inv_i(a,b) (a)=1.0l/(b)

#define f_set_r(a,b) (a)=mpfr_get_d((b),MPFR_RNDN)
#define d_set_r(a,b) (a)=mpfr_get_d((b),MPFR_RNDN)
#define l_set_r(a,b) (a)=mpfr_get_ld((b),MPFR_RNDN)
#define r_mul2_r(a,b) mpfr_mul_2ui((a),(b),1,MPFR_RNDN)
#define r_neg_r(a,b) mpfr_neg((a),(b),MPFR_RNDN)
#define r_sqr_r(a,b) mpfr_sqr((a),(b),MPFR_RNDN)
#define r_inv_r(a,b) mpfr_ui_div((a),1,(b),MPFR_RNDN)
#define r_abs_r(a,b) mpfr_abs((a),(b),MPFR_RNDN)
#define r_set_r(a,b) mpfr_set((a),(b),MPFR_RNDN)
#define r_add_rr(a,b,c) mpfr_add((a),(b),(c),MPFR_RNDN)
```

```
#define r_add_ir(a,b,c) mpfr_add_si((a),(c),(b),MPFR_RNDN)
#define r_add_ri(a,b,c) mpfr_add_si((a),(b),(c),MPFR_RNDN)
#define r_mul_rr(a,b,c) mpfr_mul((a),(b),(c),MPFR_RNDN)
#define r_mul_ir(a,b,c) mpfr_mul_si((a),(c),(b),MPFR_RNDN)
#define r_mul_ri(a,b,c) mpfr_mul_si((a),(b),(c),MPFR_RNDN)
#define r_mul_dr(a,b,c) mpfr_mul_d((a),(c),(b),MPFR_RNDN)
#define r_mul_rd(a,b,c) mpfr_mul_d((a),(b),(c),MPFR_RNDN)
#define i_sgn_r(a,b) (a)=sgn(mpfr_sgn((b)))
#define r_sqrt_r(a,b) mpfr_sqrt((a),(b),MPFR_RNDN)
#define r_log1p_r(a,b) mpfr_log1p((a),(b),MPFR_RNDN)
#define r_expm1_r(a,b) mpfr_expm1((a),(b),MPFR_RNDN)

#define b_lt_ii(a,b,c) (a)=(b)<(c)
#define b_lt_ff(a,b,c) (a)=(b)<(c)
#define b_lt_dd(a,b,c) (a)=(b)<(c)
#define b_lt_ll(a,b,c) (a)=(b)<(c)
#define b_lt_rr(a,b,c) (a)=mpfr_less_p((b),(c))
#define b_lt_ri(a,b,c) (a)=mpfr_cmp_si((b),(c))<0
#define b_le_ii(a,b,c) (a)=(b)<=(c)
#define b_le_ff(a,b,c) (a)=(b)<=(c)
#define b_le_dd(a,b,c) (a)=(b)<=(c)
#define b_le_ll(a,b,c) (a)=(b)<=(c)
#define i_eq_ii(a,b,c) (a)=(b)==(c)
#define b_eq_ii(a,b,c) (a)=(b)==(c)
#define b_eq_ff(a,b,c) (a)=(b)==(c)
#define b_eq_dd(a,b,c) (a)=(b)==(c)
#define b_eq_ll(a,b,c) (a)=(b)==(c)
#define b_le_if(a,b,c) (a)=(b)<=(c)
#define b_le_id(a,b,c) (a)=(b)<=(c)
#define b_le_il(a,b,c) (a)=(b)<=(c)


static inline float diffabsf(float c, float d) {
  if (c >= 0.0f) {
    if (c + d >= 0.0f) { return d; }
    else { return -(2.0f * c + d); }
  } else {
    if (c + d > 0.0f) { return 2.0f * c + d; }
    else { return -d; }
  }
}

static inline double diffabs(double c, double d) {
  if (c >= 0.0) {
    if (c + d >= 0.0) { return d; }
    else { return -(2.0 * c + d); }
  } else {
    if (c + d > 0.0) { return 2.0 * c + d; }
    else { return -d; }
  }
}

static inline long double diffabsl(long double c, long double d) {
  if (c >= 0.0L) {
    if (c + d >= 0.0L) { return d; }
    else { return -(2.0L * c + d); }
  } else {
```

```
370       if (c + d > 0.0L) { return 2.0L * c + d; }
          else { return -d; }
        }
      }

375   //typedef int f_plainf(int,float,float*,float,float,float,float,float*,volatile ↙
          ↪ int*);
      typedef int f_plain(int,double,double*,double,double,double,double,double*,↙
          ↪ volatile int*);
      typedef int f_plainl(int,long double,long double*,long double,long double,long ↙
          ↪ double,long double,long double*,volatile int*);
      //typedef int f_plainc(int,compensated,compensated*,compensated,compensated,↙
          ↪ compensated,compensated,compensated*,volatile int*);
      //typedef int f_referencef(int,float,float,float,mpfr_t,mpfr_t,float*,volatile ↙
          ↪ int*);
380   typedef int f_reference(int,double,double,double,mpfr_t,mpfr_t,double*,volatile ↙
          ↪ int*);
      //typedef int f_referencel(int,long double,long double,long double,mpfr_t,mpfr_t↙
          ↪ ,long double*,volatile int*);
      //typedef int f_perturbationf(int,int,float,float*,float,float,float,float,float↙
          ↪ *,float*,volatile int*);
      typedef int f_perturbation(int,int,double,double*,double,double,double,double,↙
          ↪ double*,double*,volatile int*);
      //typedef int f_perturbationl(int,int,long double,long double*,long double,long ↙
          ↪ double,long double,long double,long double*,long double*,volatile int*);
385   typedef int f_period_tri(int,double,double,double,mpfr_t,mpfr_t,mpfr_t,volatile ↙
          ↪ int*);
      typedef int f_period_jsk(int,double,double,double,mpfr_t,mpfr_t,mpfr_t,double*,↙
          ↪ volatile int*);
      typedef int f_newton(int,int,double,double,mpfr_t,mpfr_t,volatile int*);
      typedef int f_misiurewicz(int,int,int,double,double,mpfr_t,mpfr_t,volatile int*)↙
          ↪ ;
      typedef int f_size(int,double,double,mpfr_t,mpfr_t,mpfr_t,double*,volatile int*)↙
          ↪ ;
390   typedef int f_skew(int,double,double,mpfr_t,mpfr_t,int,double*,volatile int*);
      typedef int f_domain_size(int,double,double,mpfr_t,mpfr_t,mpfr_t,volatile int*);

      extern int et_wrap_plainl(f_plainl*,int,long double *,long double *,long double ↙
          ↪ *,long double *,long double *,long double *,long double *,volatile int*);
      //extern int et_wrap_plainc(f_plainc*,int ,compensated*,compensated*,compensated↙
          ↪ *,compensated*,compensated*,compensated*,compensated*,volatile int*);
395   //extern int et_wrap_referencel(f_referencel*,int,long double*,long double*,long↙
          ↪  double*,mpfr_t,mpfr_t,long double*,volatile int*);
      //extern int et_wrap_perturbationl(f_perturbationl*,int,int,long double*,long ↙
          ↪ double*,long double*,long double*,long double*,long double*,long double*,↙
          ↪ long double*,volatile int*);

      #ifndef ET_MAIN
      //static f_plainf plainf;
400   static f_plain plain;
      static f_plainl plainl;
      //static f_plainc plainc;
      //static f_referencef referencef;
      static f_reference reference;
405   //static f_referencel referencel;
      //static f_perturbationf perturbationf;
      static f_perturbation perturbation;
```

```
    //static f_perturbationl perturbationl;
    static f_period_tri period_tri;
410 static f_period_jsk period_jsk;
    static f_newton newton;
    static f_misiurewicz misiurewicz;
    static f_size size;
    static f_skew skew;
415 static f_domain_size domain_size;
    //static const char name[];
    //static const char source[];
    #endif

420 #define MAGIC ((int)(0xC01dCaf3))
    #define SIZE ((int)(sizeof(struct formula)))
    #define VERSION 8

    struct formula
425 {
      int magic;
      int ssize;
      int version;
      const char *name;
430   const char *source;
      double degree;
    //  f_plainf *plainf;
      f_plain *plain;
      f_plainl *plainl;
435 //  f_plainc *plainc;
    //  f_referencef *referencef;
      f_reference *reference;
    //  f_referencel *referencel;
    //  f_perturbationf *perturbationf;
440   f_perturbation *perturbation;
    //  f_perturbationl *perturbationl;
      f_period_tri *period_tri;
      f_period_jsk *period_jsk;
      f_newton *newton;
445   f_misiurewicz *misiurewicz;
      f_size *size;
      f_skew *skew;
      f_domain_size *domain_size;
    };
450
    extern struct formula et;

    #define FORMULA(name,source,degree) \
    struct formula et = \
455 { MAGIC, SIZE, VERSION \
    , name, source, degree \
    , &plain, &plainl \
    , &reference \
    , &perturbation \
460 , &period_tri, &period_jsk \
    , &newton, &misiurewicz \
    , &size, &skew, &domain_size \
    };
```

```
#endif
```

23 src/formulas.et

```
# et -- escape time fractals
# Copyright (C) 2018 Claude Heiland-Allen
#
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU Affero General Public License as
# published by the Free Software Foundation, either version 3 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 Affero General Public License for more details.
#
# You should have received a copy of the GNU Affero General Public License
# along with this program.  If not, see <https://www.gnu.org/licenses/>.

Mandelbrot
  z := z^p + c

# Mandelbar: do conjugate before power

Mandelbar
  z := (x - i y)^p + c

# Burning Ship: do abs(Re, Im) before power

Burning Ship
  z := (|x| + i |y|)^p + c

Burning Ship Partial Real aka Heart Mandelbrot
  z := (|x| + i y)^p + c

Burning Ship Partial Imag
  z := (x + i |y|)^p + c

Burning Ship Partial Real Mandelbar aka Perpendicular Mandelbrot
  z := (|x| - i y)^p + c

Burning Ship Partial Imag Mandelbar aka Perpendicular Burning Ship
  z := (x - i |y|)^p + c

# Celtic: do abs(Re) after power

Celtic Mandelbrot aka Buffalo Partial Real
  w := z^p
  z := (|u| + i v) + c

Celtic Mandelbar
  w := z^p
  z := (|u| - i v) + c

Celtic Burning Ship
  w := (|x| + i |y|)^p
```

```
    z := (|u| + i v) + c

Celtic Burning Ship Partial Real aka Celtic Heart
  w := (|x| + i y)^p
  z := (|u| + i v) + c

Celtic Burning Ship Partial Imag
  w := (x + i |y|)^p
  z := (|u| + i v) + c

Celtic Burning Ship Partial Real Mandelbar
  w := (|x| - i y)^p
  z := (|u| + i v) + c

# Buffalo: do abs(Re, Im) after power

Buffalo
  w := z^p
  z := (|u| + i |v|) + c

Buffalo Partial Imag
  w := z^p
  z := (u + i |v|) + c

# ?

Orthogonal Mandelbar
  z := c - (x - i y)^p

Perpendicular Celtic
  w := (|x| + i y)^p
  z := (|u| - i v) + c

Perpendicular Buffalo
  w := (x + i |y|)^p
  z := (|u| - i v) + c

Quasi Burning Ship
  w := (|x| + i |y|)^p
  z := (u - i |v|) + c

#-- POWI(RABS(Z^2/1)+I*RABS(IM(Z^2/1));2)+C
Burning Buffalo
  w := z^p
  z := (|u| + i |v|)^q + c

# hybrids

# this syntax would be nice, but isn't implemented yet
#Hybrid M BS M M
## credit: realflow100 on fractalforums.org
#  Mandelbrot
#  Burning Ship
#  Mandelbrot
#  Mandelbrot

Hybrid M BS M M
```

```
  z := z^p + c
  z := (|x| + i |y|)^p + c
  z := z^p + c
  z := z^p + c

Hybrid M BS H
  z := z^p + c
  z := (|x| + i |y|)^p + c
  z := (|x| + i y)^p + c
```

24 src/Fractal/EscapeTime/Algorithms/C.hs

```
module Fractal.EscapeTime.Algorithms.C
  ( module Fractal.EscapeTime.Algorithms.C.Plain
  , module Fractal.EscapeTime.Algorithms.C.Reference
  , module Fractal.EscapeTime.Algorithms.C.Perturbation
  , module Fractal.EscapeTime.Algorithms.C.Period
  , module Fractal.EscapeTime.Algorithms.C.Newton
  , module Fractal.EscapeTime.Algorithms.C.Size
  ) where

import Fractal.EscapeTime.Algorithms.C.Plain
import Fractal.EscapeTime.Algorithms.C.Reference
import Fractal.EscapeTime.Algorithms.C.Perturbation
import Fractal.EscapeTime.Algorithms.C.Period
import Fractal.EscapeTime.Algorithms.C.Newton
import Fractal.EscapeTime.Algorithms.C.Size
```

25 src/Fractal/EscapeTime/Algorithms/R2/DomainSize.hs

```
{-
et -- escape time fractals
Copyright (C) 2018 Claude Heiland-Allen

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU Affero General Public License as
published by the Free Software Foundation, either version 3 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 Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License
along with this program.  If not, see <https://www.gnu.org/licenses/>.
-}

module Fractal.EscapeTime.Algorithms.R2.DomainSize (domain_size) where

import Prelude hiding (read)

import Control.Monad (replicateM_)

import Fractal.EscapeTime.Compiler
import Fractal.EscapeTime.Formulas.Types
```

```
-- FIXME this is broken, don't know what the fixes should be
domain_size :: [(Int, R2Formula (Expr String))] -> String
domain_size fs = runCompile_ FDouble . function NBool "domain_size" [(NInt, "P")↙
    ↳ , (NFloat, "D"), (NFloat, "E"), (NReal, "A"), (NReal, "B"), (NReal, "R"), ↙
    ↳ (NVolatileIntPtr, "RUNNING")] $ \s -> do
  let degree = 2 -- FIXME
  early "mpfr_prec_t prec=mpfr_get_prec(A);\n"
  p <- int
  a <- real
  b <- real
  p .= EVar NInt "P"
  let d_ = EVar NFloat "D"
      e_ = EVar NFloat "E"
  a .= EVar NReal "A"
  b .= EVar NReal "B"
  x <- real
  y <- real
  lxa <- real
  lxb <- real
  lya <- real
  lyb <- real
  bxa <- real
  bxb <- real
  bya <- real
  byb <- real
  -- FIXME initialization?
  x .= a
  y .= b
  lxa .= 1
  lxb .= 0
  lya .= 0
  lyb .= 1
  zq2 <- real
  zq2 .= x * x + y * y
  -- loop one period
  q <- int
  q .= 2
  k <- int
  k .= 0
  running <- bool
  running .= read (EVar NVolatileIntPtr "RUNNING") 0
  while_ (q `ELessEqual` p `EAnd` running) $ do
    -- alloc z, l
    xn <- real
    yn <- real
    lxan <- real
    lxbn <- real
    lyan <- real
    lybn <- real
    -- calc z, l
    let g (_, f) j0 kn = (,) j0 $ do
          let R2 gx gy = f (R2 d_ e_) (R2 a b) (R2 x y)
              fx = optimize gx
              fy = optimize gy
              fdxa = ddx d a fx
              fdxb = ddx d b fx
```

```
                fdya = ddx d a fy
                fdyb = ddx d b fy
                d v u
                  | u == x && v == a = lxa
                  | u == x && v == b = lxb
                  | u == y && v == a = lya
                  | u == y && v == b = lyb
                  | u == v = 1
                  | otherwise = 0
            xn .= fx
            yn .= fy
            lxan .= optimize fdxa
            lxbn .= optimize fdxb
            lyan .= optimize fdya
            lybn .= optimize fdyb
            k .= fromIntegral kn
        _ <- switch_ k $ zipWith3 g fs [0..] ([1 .. length fs - 1] ++ [0])
        x .= xn
        y .= yn
        lxa .= lxan
        lxb .= lxbn
        lya .= lyan
        lyb .= lybn
        -- z2 <- |z|^2
        zp2 <- real
        zp2 .= x * x + y * y
        _ <- if_ ((zp2 `ELess` zq2) `EAnd` (q `ELess` p)) (zq2 .= zp2) (return ())
        -- loop
        q .= q + 1
        running .= read (EVar NVolatileIntPtr "RUNNING") 0
      -- abszq / cabs(dc);
      EVar NReal "R" .= ESqrt NReal (zq2 / ESqrt NReal (lxa * lxa + lxb * lxb + lya ↙
          ↳ * lya + lyb * lyb))
      s .= running
      return_ s
```

26 src/Fractal/EscapeTime/Algorithms/R2.hs

```
{-
et -- escape time fractals
Copyright (C) 2018,2019 Claude Heiland-Allen

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU Affero General Public License as
published by the Free Software Foundation, either version 3 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 Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License
along with this program.  If not, see <https://www.gnu.org/licenses/>.
-}

module Fractal.EscapeTime.Algorithms.R2
```

```
    ( module Fractal.EscapeTime.Algorithms.R2.Plain
    , module Fractal.EscapeTime.Algorithms.R2.Reference
    , module Fractal.EscapeTime.Algorithms.R2.Perturbation
    , module Fractal.EscapeTime.Algorithms.R2.PeriodTri
    , module Fractal.EscapeTime.Algorithms.R2.PeriodJSK
    , module Fractal.EscapeTime.Algorithms.R2.Newton
    , module Fractal.EscapeTime.Algorithms.R2.Misiurewicz
    , module Fractal.EscapeTime.Algorithms.R2.Size
    , module Fractal.EscapeTime.Algorithms.R2.Skew
    , module Fractal.EscapeTime.Algorithms.R2.DomainSize
    ) where

import Fractal.EscapeTime.Algorithms.R2.Plain
import Fractal.EscapeTime.Algorithms.R2.Reference
import Fractal.EscapeTime.Algorithms.R2.Perturbation
import Fractal.EscapeTime.Algorithms.R2.PeriodTri
import Fractal.EscapeTime.Algorithms.R2.PeriodJSK
import Fractal.EscapeTime.Algorithms.R2.Newton
import Fractal.EscapeTime.Algorithms.R2.Misiurewicz
import Fractal.EscapeTime.Algorithms.R2.Size
import Fractal.EscapeTime.Algorithms.R2.Skew
import Fractal.EscapeTime.Algorithms.R2.DomainSize
```

27 src/Fractal/EscapeTime/Algorithms/R2/Misiurewicz.hs

```
{-
et -- escape time fractals
Copyright (C) 2018,2019 Claude Heiland-Allen

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU Affero General Public License as
published by the Free Software Foundation, either version 3 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 Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License
along with this program.  If not, see <https://www.gnu.org/licenses/>.
-}

module Fractal.EscapeTime.Algorithms.R2.Misiurewicz (misiurewicz) where

import Prelude hiding (read)

import Fractal.EscapeTime.Compiler
import Fractal.EscapeTime.Formulas.Types

misiurewicz :: [(Int, R2Formula (Expr String))] -> String
misiurewicz fs = runCompile_ FDouble . function NBool "misiurewicz" [(NInt, "N")↙
    ↳ , (NInt, "Q"), (NInt, "P"), (NFloat, "D"), (NFloat, "E"), (NReal, "A"), (↙
    ↳ NReal, "B"), (NVolatileIntPtr, "RUNNING")] $ \ok -> do
  early "mpfr_prec_t prec=mpfr_get_prec(A);\n"
  n <- int
  q <- int
```

```
    p <- int
    a <- real
    b <- real
    n .= EVar NInt "N"
    q .= EVar NInt "Q"
    p .= EVar NInt "P"
    let d_ = EVar NFloat "D"
        e_ = EVar NFloat "E"
    a .= EVar NReal "A"
    b .= EVar NReal "B"
    qp <- int
    qp .= q + p
    i <- int
    i .= 0
    running <- bool
    running .= read (EVar NVolatileIntPtr "RUNNING") 0
    while_ (i `ELess` n `EAnd` running) $ do
      xq <- real
      yq <- real
      dxaq <- real
      dxbq <- real
      dyaq <- real
      dybq <- real
      x <- real
      y <- real
      dxa <- real
      dxb <- real
      dya <- real
      dyb <- real
      x .= 0
      y .= 0
      dxa .= 0
      dxb .= 0
      dya .= 0
      dyb .= 0
      -- loop one period
      j <- int
      j .= 0
      k <- int
      k .= 0
      while_ (j `ELess` qp `EAnd` running) $ do
        -- save after preperiod
        _ <- if_ (j `EEqual` q) (do
          xq .= x
          yq .= y
          dxaq .= dxa
          dxbq .= dxb
          dyaq .= dya
          dybq .= dyb) (return ())
        -- alloc
        xn <- real
        yn <- real
        dxan <- real
        dxbn <- real
        dyan <- real
        dybn <- real
        -- calc
```

```
      let g (_, f) j0 kn = (,) j0 $ do
            let R2 gx gy = f (R2 d_ e_) (R2 a b) (R2 x y)
                fx = optimize gx
                fy = optimize gy
                fdxa = ddx d a fx
                fdxb = ddx d b fx
                fdya = ddx d a fy
                fdyb = ddx d b fy
                d v u
                  | u == x && v == a = dxa
                  | u == x && v == b = dxb
                  | u == y && v == a = dya
                  | u == y && v == b = dyb
                  | u == v = 1
                  | otherwise = 0
            xn .= fx
            yn .= fy
            dxan .= optimize fdxa
            dxbn .= optimize fdxb
            dyan .= optimize fdya
            dybn .= optimize fdyb
            k .= fromIntegral kn
      _ <- switch_ k $ zipWith3 g fs [0..] ([1 .. length fs - 1] ++ [0])
      -- step
      x .= xn
      y .= yn
      dxa .= dxan
      dxb .= dxbn
      dya .= dyan
      dyb .= dybn
      -- loop
      j .= j + 1
      running .= read (EVar NVolatileIntPtr "RUNNING") 0
    -- subtract preperiod
    x .= x - xq
    y .= y - yq
    dxa .= dxa - dxaq
    dxb .= dxb - dxbq
    dya .= dya - dyaq
    dyb .= dyb - dybq
    -- newton step
    det <- real
    det .= dxa * dyb - dxb * dya
    u <- real
    v <- real
    u .= -( dyb * x - dxb * y) / det
    v .= -(-dya * x + dxa * y) / det
    a .= a + u
    b .= b + v
    -- TODO check convergence
    i .= i + 1
  EVar NReal "A" .= a
  EVar NReal "B" .= b
  ok .= running
  return_ ok
```

28 src/Fractal/EscapeTime/Algorithms/R2/Newton.hs

```
{-
et -- escape time fractals
Copyright (C) 2018 Claude Heiland-Allen

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU Affero General Public License as
published by the Free Software Foundation, either version 3 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 Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License
along with this program.  If not, see <https://www.gnu.org/licenses/>.
-}

module Fractal.EscapeTime.Algorithms.R2.Newton (newton) where

import Prelude hiding (read)

import Fractal.EscapeTime.Compiler
import Fractal.EscapeTime.Formulas.Types

newton :: [(Int, R2Formula (Expr String))] -> String
newton fs = runCompile_ FDouble . function NBool "newton" [(NInt, "N"), (NInt, "P"), (NFloat, "D"), (NFloat, "E"), (NReal, "A"), (NReal, "B"), (NVolatileIntPtr, "RUNNING")] $ \ok -> do
  early "mpfr_prec_t prec=mpfr_get_prec(A);\n"
  n <- int
  p <- int
  a <- real
  b <- real
  n .= EVar NInt "N"
  p .= EVar NInt "P"
  let d_ = EVar NFloat "D"
      e_ = EVar NFloat "E"
  a .= EVar NReal "A"
  b .= EVar NReal "B"
  i <- int
  i .= 0
  running <- bool
  running .= read (EVar NVolatileIntPtr "RUNNING") 0
  while_ (i `ELess` n `EAnd` running) $ do
    x <- real
    y <- real
    dxa <- real
    dxb <- real
    dya <- real
    dyb <- real
    x .= 0
    y .= 0
    dxa .= 0
    dxb .= 0
```

```
    dya .= 0
    dyb .= 0
    -- loop one period
    j <- int
    j .= 0
    k <- int
    k .= 0
    while_ (j `ELess` p `EAnd` running) $ do
      -- alloc
      xn <- real
      yn <- real
      dxan <- real
      dxbn <- real
      dyan <- real
      dybn <- real
      -- calc
      let g (_, f) j0 kn = (,) j0 $ do
            let R2 gx gy = f (R2 d_ e_) (R2 a b) (R2 x y)
                fx = optimize gx
                fy = optimize gy
                fdxa = ddx d a fx
                fdxb = ddx d b fx
                fdya = ddx d a fy
                fdyb = ddx d b fy
                d v u
                  | u == x && v == a = dxa
                  | u == x && v == b = dxb
                  | u == y && v == a = dya
                  | u == y && v == b = dyb
                  | u == v = 1
                  | otherwise = 0
            xn .= fx
            yn .= fy
            dxan .= optimize fdxa
            dxbn .= optimize fdxb
            dyan .= optimize fdya
            dybn .= optimize fdyb
            k .= fromIntegral kn
      _ <- switch_ k $ zipWith3 g fs [0..] ([1 .. length fs - 1] ++ [0])
      -- step
      x .= xn
      y .= yn
      dxa .= dxan
      dxb .= dxbn
      dya .= dyan
      dyb .= dybn
      -- loop
      j .= j + 1
      running .= read (EVar NVolatileIntPtr "RUNNING") 0
    -- newton step
    det <- real
    det .= dxa * dyb - dxb * dya
    u <- real
    v <- real
    u .= -( dyb * x - dxb * y) / det
    v .= -(-dya * x + dxa * y) / det
    a .= a + u
```

```
    b .= b + v
    -- TODO check convergence
    i .= i + 1
  EVar NReal "A" .= a
  EVar NReal "B" .= b
  ok .= running
  return_ ok
```

29 src/Fractal/EscapeTime/Algorithms/R2/PeriodJSK.hs

```
{-
et -- escape time fractals
Copyright (C) 2018 Claude Heiland-Allen

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU Affero General Public License as
published by the Free Software Foundation, either version 3 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 Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License
along with this program.  If not, see <https://www.gnu.org/licenses/>.
-}

module Fractal.EscapeTime.Algorithms.R2.PeriodJSK (period_jsk) where

import Prelude hiding (read)

import Fractal.EscapeTime.Compiler
import Fractal.EscapeTime.Formulas.Types

period_jsk :: [(Int, R2Formula (Expr String))] -> String
period_jsk fs = runCompile_ FDouble . function NInt "period_jsk" [(NInt, "N"), (↙
    ↪ NFloat, "R2"), (NFloat, "D"), (NFloat, "E"), (NReal, "A"), (NReal, "B"), (↙
    ↪ NReal, "S"), (NFloatPtr, "K"), (NVolatileIntPtr, "RUNNING")] $ \i -> do
  early "mpfr_prec_t prec=mpfr_get_prec(A);\n"
  n <- int
  r2 <- float
  a <- real
  b <- real
  s <- real
  n .= EVar NInt "N"
  r2 .= EVar NFloat "R2"
  let d_ = EVar NFloat "D"
      e_ = EVar NFloat "E"
      kk = EVar NFloatPtr "K"
  a .= EVar NReal "A"
  b .= EVar NReal "B"
  s .= EVar NReal "S"
  -- K^{-1}
  ai <- float
  aj <- float
  bi <- float
```

```
    bj <- float
    ai .= read kk 0
    aj .= read kk 1
    bi .= read kk 2
    bj .= read kk 3
    detK <- float
    detK .= ai * bj - aj * bi
    ai1 <- float
    aj1 <- float
    bi1 <- float
    bj1 <- float
    ai1 .=  bj / detK
    aj1 .= -bi / detK
    bi1 .= -aj / detK
    bj1 .=  ai / detK
    -- X
    x <- real
    y <- real
    x .= 0
    y .= 0
    -- J
    xa <- real
    xb <- real
    ya <- real
    yb <- real
    xa .= 0
    xb .= 0
    ya .= 0
    yb .= 0
    -- ...
    xf <- float
    yf <- float
    z2 <- float
    i .= 0
    z2 .= 0
    p <- bool
    p .= 1
    k <- int
    k .= 0
    running <- bool
    running .= read (EVar NVolatileIntPtr "RUNNING") 0
    while_ (ELess i n `EAnd` ELess z2 r2 `EAnd` p `EAnd` running) $ do
      -- allocate next
      xn <- real
      yn <- real
      xan <- real
      xbn <- real
      yan <- real
      ybn <- real
      -- do formula
      let g (_, f) j kn = (,) j $ do
            let R2 fx0 fy0 = f (R2 d_ e_) (R2 a b) (R2 x y)
                fx = optimize fx0
                fy = optimize fy0
                d v u
                  | u == x && v == a = xa
                  | u == x && v == b = xb
```

```
                    | u == y && v == a = ya
                    | u == y && v == b = yb
                    | u == v = 1
                    | otherwise = 0
            -- calc next
            xn .= fx
            yn .= fy
            xan .= optimize (ddx d a fx)
            xbn .= optimize (ddx d b fx)
            yan .= optimize (ddx d a fy)
            ybn .= optimize (ddx d b fy)
            -- step next
            x .= xn
            y .= yn
            xa .= xan
            xb .= xbn
            ya .= yan
            yb .= ybn
            xf .= x
            yf .= y
            z2 .= xf*xf + yf*yf
            k .= fromIntegral kn
      _ <- switch_ k $ zipWith3 g fs [0..] ([1 .. length fs - 1] ++ [0])
      -- J^{-1}
      detJ <- real
      detJ .= xa * yb - xb * ya
      xa1 <- real
      xb1 <- real
      ya1 <- real
      yb1 <- real
      xa1 .=  yb / detJ
      xb1 .= -ya / detJ
      ya1 .= -xb / detJ
      yb1 .=  xa / detJ
      -- (u0 v0) = J^{-1} * (x y)
      u0 <- real
      v0 <- real
      u0 .= xa1 * x + xb1 * y
      v0 .= ya1 * x + yb1 * y
      -- (u1 v1) = s^{-1} K^{-1} * (u0 v0)
      u1 <- real
      v1 <- real
      u1 .= (ai1 * u0 + aj1 * v0) / s
      v1 .= (bi1 * u0 + bj1 * v0) / s
      -- float
      u2 <- float
      v2 <- float
      u2 .= u1
      v2 .= v1
      uv <- float
      uv .= u2 * u2 + v2 * v2
      p .= 1 `ELessEqual` uv
      -- update loop
      i .= i + 1
      running .= read (EVar NVolatileIntPtr "RUNNING") 0
    _ <- if_ (EEqual i n `EOr` ELessEqual r2 z2 `EOr` p) (i .= -1) (return ())
    return_ i
```

30 src/Fractal/EscapeTime/Algorithms/R2/PeriodTri.hs

```
{-
et -- escape time fractals
Copyright (C) 2018 Claude Heiland-Allen

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU Affero General Public License as
published by the Free Software Foundation, either version 3 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 Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License
along with this program.  If not, see <https://www.gnu.org/licenses/>.
-}

module Fractal.EscapeTime.Algorithms.R2.PeriodTri (period_tri) where

import Prelude hiding (read)

import Fractal.EscapeTime.Compiler
import Fractal.EscapeTime.Formulas.Types

period_tri :: [(Int, R2Formula (Expr String))] -> String
period_tri fs = runCompile_ FDouble . function NInt "period_tri" [(NInt, "N"), (NFloat, "R2"), (NFloat, "D"), (NFloat, "E"), (NReal, "A"), (NReal, "B"), (NReal, "H"), (NVolatileIntPtr, "RUNNING")] $ \i -> do
  early "mpfr_prec_t prec=mpfr_get_prec(A);\n"
  n <- int
  r2 <- float
  a <- real
  b <- real
  h <- real
  n .= EVar NInt "N"
  r2 .= EVar NFloat "R2"
  let d_ = EVar NFloat "D"
      e_ = EVar NFloat "E"
  a .= EVar NReal "A"
  b .= EVar NReal "B"
  h .= EVar NReal "H"
  a0 <- real
  b0 <- real
  a1 <- real
  b1 <- real
  a2 <- real
  b2 <- real
  a0 .= a - h
  b0 .= b - h
  a1 .= a + h
  b1 .= b - h
  a2 .= a
  b2 .= b + h
  x0 <- real
```

```
    y0 <- real
    x1 <- real
    y1 <- real
    x2 <- real
    y2 <- real
    x0 .= 0
    y0 .= 0
    x1 .= 0
    y1 .= 0
    x2 .= 0
    y2 .= 0
    xf <- float
    yf <- float
    z2 <- float
    i .= 0
    z2 .= 0
    p <- bool
    p .= 1
    k <- int
    k .= 0
    running <- bool
    running .= read (EVar NVolatileIntPtr "RUNNING") 0
    while_ (ELess i n `EAnd` ELess z2 r2 `EAnd` p `EAnd` running) $ do
      -- allocate next
      xn <- real
      yn <- real
      z2 .= 0
      -- do formula
      let g (_, f) j kn = (,) j $ do
            let R2 f0x f0y = f (R2 d_ e_) (R2 a0 b0) (R2 x0 y0)
                R2 f1x f1y = f (R2 d_ e_) (R2 a1 b1) (R2 x1 y1)
                R2 f2x f2y = f (R2 d_ e_) (R2 a2 b2) (R2 x2 y2)
            xn .= optimize f0x
            yn .= optimize f0y
            x0 .= xn
            y0 .= yn
            xf .= xn
            yf .= yn
            z2 .= z2 + xf*xf + yf*yf
            xn .= optimize f1x
            yn .= optimize f1y
            x1 .= xn
            y1 .= yn
            xf .= xn
            yf .= yn
            z2 .= z2 + xf*xf + yf*yf
            xn .= optimize f2x
            yn .= optimize f2y
            x2 .= xn
            y2 .= yn
            xf .= xn
            yf .= yn
            z2 .= z2 + xf*xf + yf*yf
            k .= fromIntegral kn
      _ <- switch_ k $ zipWith3 g fs [0..] ([1 .. length fs - 1] ++ [0])
      -- check period
      p01 <- int
```

```
    p12 <- int
    p20 <- int
    let crosses_axis x y u v = EIf (signum y `EEqual` signum v) 0 $
          let a' = u - x
              b' = v - y
              s = signum b'
              t = signum (b' * x - a' * y)
          in  s `EEqual` t
    p01 .= crosses_axis x0 y0 x1 y1
    p12 .= crosses_axis x1 y1 x2 y2
    p20 .= crosses_axis x2 y2 x0 y0
    ps <- int
    ps .= p01 + p12 + p20
    p .= EEqual ps 0 `EOr` EEqual ps 2
    -- update loop
    i .= i + 1
    running .= read (EVar NVolatileIntPtr "RUNNING") 0
  _ <- if_ (EEqual i n `EOr` ELessEqual r2 z2 `EOr` p) (i .= -1) (return ())
  return_ i
```

31 src/Fractal/EscapeTime/Algorithms/R2/Perturbation.hs

```
{-
et -- escape time fractals
Copyright (C) 2018,2019 Claude Heiland-Allen

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU Affero General Public License as
published by the Free Software Foundation, either version 3 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 Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License
along with this program.  If not, see <https://www.gnu.org/licenses/>.
-}

module Fractal.EscapeTime.Algorithms.R2.Perturbation (perturbation) where

import Prelude hiding (read)

import Fractal.EscapeTime.Compiler
import Fractal.EscapeTime.Formulas.Types

-- FIXME this needs the degree of each stanza to be the *highest* non-linear power
perturbation :: [(Int, R2Formula (Expr String))] -> String
perturbation fs = concat . flip map [FDouble] $ \t -> runCompile_ t . function (NBool ("perturbation" ++ fSuffix t) [(NInt, "M"), (NInt, "N"), (NFloat, "R2"), (NFloatPtr, "H"), (NFloat, "d"), (NFloat, "e"), (NFloat, "A"), (NFloat, "B"), (NFloatPtr, "XYZS"), (NFloatPtr, "OUT"), (NVolatileIntPtr, "RUNNING")] $ \running -> do
  let m = EVar NInt "M"
      n = EVar NInt "N"
```

```
        r2 = EVar NFloat "R2"
        hp = EVar NFloatPtr "H"
        d_ = EVar NFloat "d"
        e_ = EVar NFloat "e"
        a = EVar NFloat "A"
        b = EVar NFloat "B"
        xyzs = EVar NFloatPtr "XYZS"
        out = EVar NFloatPtr "OUT"
        runningP = EVar NVolatileIntPtr "RUNNING"
        logdegree :: Double
        logdegree = sum (map (log . fromIntegral . fst) fs) / fromIntegral (length
            ↳ fs)
    daa <- float
    dab <- float
    dba <- float
    dbb <- float
    daa .= read hp 0
    dab .= read hp 1
    dba .= read hp 2
    dbb .= read hp 3
    x <- float
    y <- float
    dxa <- float
    dxb <- float
    dya <- float
    dyb <- float
    let a0 = EVar NFloat "A0"
        b0 = EVar NFloat "B0"
        aa = EVar NFloat "aa"
        bb = EVar NFloat "bb"
    x0 <- float
    y0 <- float
    z0 <- float
    xx <- float
    yy <- float
    x .= 0
    y .= 0
    dxa .= 0
    dxb .= 0
    dya .= 0
    dyb .= 0
    i <- int
    z2 <- float
    i .= 0
    x0 .= read xyzs (3 * i + 0)
    y0 .= read xyzs (3 * i + 1)
    z0 .= read xyzs (3 * i + 2)
    xx .= x0 + x
    yy .= y0 + y
    z2 .= xx*xx + yy*yy
    minz2 <- float
    minz2 .= r2
    period <- int
    period .= 0
    unglitched <- bool
    unglitched .= 1
    running .= read runningP 0
```

```
    k <- int
    k .= 0
    da <- float
    db <- float
    _ <- while_ (ELess i m `EAnd` ELess z2 r2 `EAnd` unglitched `EAnd` running) $ ↙
      ↳ do
      -- allocate next
      xn <- float
      yn <- float
      dxan <- float
      dxbn <- float
      dyan <- float
      dybn <- float
      -- do formula
      let g (_, f) j kn = (,) j $ do
            let R2 fx0 fy0 = f (R2 d_ e_) (R2 a' b') (R2 x' y')
                a' = EVar NFloat "a"
                b' = EVar NFloat "b"
                x' = EVar NFloat "x"
                y' = EVar NFloat "y"
                vs =
                  [ ("d", (d_, 0))
                  , ("e", (e_, 0))
                  , ("a", (a0, a))
                  , ("b", (b0, b))
                  , ("x", (x0, x))
                  , ("y", (y0, y))
                  ]
                fx = perturb vs fx0
                fy = perturb vs fy0
                R2 gxx gyy = f (R2 d_ e_) (R2 aa bb) (R2 xx yy)
                fxx = optimize gxx
                fyy = optimize gyy
                fdxa = optimize $ ddx d aa fxx
                fdxb = optimize $ ddx d bb fxx
                fdya = optimize $ ddx d aa fyy
                fdyb = optimize $ ddx d bb fyy
                d v u
                  | u == xx && v == aa = dxa
                  | u == xx && v == bb = dxb
                  | u == yy && v == aa = dya
                  | u == yy && v == bb = dyb
                  | u == aa && v == aa = daa
                  | u == aa && v == bb = dab
                  | u == bb && v == aa = dba
                  | u == bb && v == bb = dbb
                  | u == v = 1
                  | otherwise = 0
            comment $ "fx   = " ++ pretty fx
            comment $ "fy   = " ++ pretty fy
            comment $ "fdxa = " ++ pretty fdxa
            comment $ "fdxb = " ++ pretty fdxb
            comment $ "fdya = " ++ pretty fdya
            comment $ "fdyb = " ++ pretty fdyb
            xn .= fx
            yn .= fy
            dxan .= fdxa
```

```
            dxbn .= fdxb
            dyan .= fdya
            dybn .= fdyb
            k .= fromIntegral kn
        _ <- switch_ k $ zipWith3 g fs [0..] ([1 .. length fs - 1] ++ [0])
        -- update next
        x .= xn
        y .= yn
        dxa .= dxan
        dxb .= dxbn
        dya .= dyan
        dyb .= dybn
        -- update loop
        i .= i + 1
        x0 .= read xyzs (3 * i + 0)
        y0 .= read xyzs (3 * i + 1)
        z0 .= read xyzs (3 * i + 2)
        xx .= x0 + x
        yy .= y0 + y
        z2 .= xx*xx + yy*yy
        running .= read runningP 0
        unglitched .= ELessEqual z0 z2
        if_ (ELess z2 minz2 `EAnd` ELessEqual (i + 4) m) (do
          minz2 .= z2
          period .= i
          det <- float
          det .= dxa * dyb - dxb * dya
          da .= a - (  dyb * x - dya * y) / det
          db .= b - (- dxb * x + dxa * y) / det) (return ())
      de <- float
      nc <- float
      ni <- float
      nf <- float
      nc .= 0
      ni .= 0
      nf .= 0
      _ <- if_ (ELess z2 z0 `EOr` (((ELessEqual r2 z2 `EAnd` ELess m (i + 4)) `EOr` ↙
          ↳ (ELess z2 r2 `EAnd` EEqual i m)) `EAnd` ELess m n))
            (do{ {- glitch -}  ni .= i ; if_ (ELessEqual r2 z2 `EAnd` ELess m (i +↙
                ↳ 4) `EAnd` ELess m n)
              (do{ {- ref escaped -} nf .= negate 2 ; de .= negate minz2 })
              (do  {- pauldelbrot -}
                nf .= negate 3
                de .= negate z2
                det <- float
                det .= dxa * dyb - dxb * dya
                da .= a - (  dyb * x - dya * y) / det
                db .= b - (- dxb * x + dxa * y) / det)
            }) $ if_ (ELess z2 r2) (de .= 0 {- interior -}) $ do {- escaped -}
        u <- float
        v <- float
        u .= xx * dxa + yy * dya
        v .= xx * dxb + yy * dyb
        de .= z2 * log (sqrt z2) / sqrt (u*u + v*v)
        nc .= if isInfinite logdegree || isNaN logdegree || 0 == logdegree then i ↙
            ↳ else i + 1 - log (log (sqrt z2)) / realToFrac logdegree
        ni .= EFloor NFloat nc
```

```
    nf .= nc - ni
  write out 0 de
  write out 1 period
  write out 2 ni
  write out 3 nf
  -- (a, b) - J^-1 (x, y)  -- Newton step to Pauldelbrot glitch center
  write out 4 da
  write out 5 db
  return_ running
```

32 src/Fractal/EscapeTime/Algorithms/R2/Plain.hs

```
{-
et -- escape time fractals
Copyright (C) 2018,2019 Claude Heiland-Allen

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU Affero General Public License as
published by the Free Software Foundation, either version 3 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 Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License
along with this program.  If not, see <https://www.gnu.org/licenses/>.
-}

module Fractal.EscapeTime.Algorithms.R2.Plain (plain) where

import Prelude hiding (read)

import Fractal.EscapeTime.Compiler
import Fractal.EscapeTime.Formulas.Types

-- FIXME this needs the degree of each stanza to be the *highest* non-linear power
plain :: [(Int, R2Formula (Expr String))] -> String
plain fs = concat . flip map [FDouble, FLongDouble] $ \t -> runCompile_ t . function NBool ("plain" ++ fSuffix t) [(NInt, "N"), (NFloat, "R2"), (NFloatPtr, "H"), (NFloat, "D"), (NFloat, "E"), (NFloat, "A"), (NFloat, "B"), (NFloatPtr, "OUT"), (NVolatileIntPtr, "RUNNING")] $ \running -> do
  let n = EVar NInt "N"
      r2 = EVar NFloat "R2"
      hp = EVar NFloatPtr "H"
      d_ = EVar NFloat "D"
      e_ = EVar NFloat "E"
      a = EVar NFloat "A"
      b = EVar NFloat "B"
      out = EVar NFloatPtr "OUT"
      runningP = EVar NVolatileIntPtr "RUNNING"
      logdegree :: Double
      logdegree = sum (map (log . fromIntegral . fst) fs) / fromIntegral (length fs)
  daa <- float
```

```
    dab <- float
    dba <- float
    dbb <- float
    daa .= read hp 0
45  dab .= read hp 1
    dba .= read hp 2
    dbb .= read hp 3
    x <- float
    y <- float
50  dxa <- float
    dxb <- float
    dya <- float
    dyb <- float
    x .= 0
55  y .= 0
    dxa .= 0
    dxb .= 0
    dya .= 0
    dyb .= 0
60  i <- int
    i .= 0
    z2 <- float
    z2 .= 0
    minz2 <- float
65  minz2 .= r2
    period <- int
    period .= 0
    running .= read runningP 0
    k <- int
70  k .= 0
    _ <- while_ (ELess (i + fromIntegral (length fs - 1)) n `EAnd` ELess z2 r2 `↙
      ↳ EAnd` running) $ do
      -- allocate next
      xn <- float
      yn <- float
75    dxan <- float
      dxbn <- float
      dyan <- float
      dybn <- float
      -- do formula
80    let g (_, f) j kn = (,) j $ do
            let R2 gx gy = f (R2 d_ e_) (R2 a b) (R2 x y)
                fx = optimize gx
                fy = optimize gy
                fdxa = optimize $ ddx d a fx
85              fdxb = optimize $ ddx d b fx
                fdya = optimize $ ddx d a fy
                fdyb = optimize $ ddx d b fy
                d v u
                  | u == x && v == a = dxa
90                | u == x && v == b = dxb
                  | u == y && v == a = dya
                  | u == y && v == b = dyb
                  | u == a && v == a = daa
                  | u == a && v == b = dab
95                | u == b && v == a = dba
                  | u == b && v == b = dbb
```

```
                | u == v = 1
                | otherwise = 0
            comment $ "fx   = " ++ pretty fx
            comment $ "fy   = " ++ pretty fy
            comment $ "fdxa = " ++ pretty fdxa
            comment $ "fdxb = " ++ pretty fdxb
            comment $ "fdya = " ++ pretty fdya
            comment $ "fdyb = " ++ pretty fdyb
            xn .= fx
            yn .= fy
            dxan .= fdxa
            dxbn .= fdxb
            dyan .= fdya
            dybn .= fdyb
            k .= fromIntegral kn
      _ <- switch_ k $ zipWith3 g fs [0..] ([1 .. length fs - 1] ++ [0])
      -- update next
      x .= xn
      y .= yn
      dxa .= dxan
      dxb .= dxbn
      dya .= dyan
      dyb .= dybn
      -- update loop
      z2 .= x*x + y*y
      i .= i + 1
      _ <- if_ (ELess z2 minz2) (do{ minz2 .= z2 ; period .= i }) (return ())
      running .= read runningP 0
    de <- float
    nc <- float
    ni <- float
    nf <- float
    nc .= 0
    ni .= 0
    nf .= 0
    _ <- if_ (ELess z2 r2) (de .= 0 {- interior -}) $ do {- escaped -}
      u <- float
      v <- float
      u .= x * dxa + y * dya
      v .= x * dxb + y * dyb
      de .= z2 * log (sqrt z2) / sqrt (u*u + v*v)
      nc .= if isInfinite logdegree || isNaN logdegree || 0 == logdegree then i ↙
          ↳ else i + 1 - log (log (sqrt z2)) / realToFrac logdegree
      ni .= EFloor NFloat nc
      nf .= nc - ni
    write out 0 de
    write out 1 period
    write out 2 ni
    write out 3 nf
    return_ running
```

33 src/Fractal/EscapeTime/Algorithms/R2/Reference.hs

```
{-
et -- escape time fractals
Copyright (C) 2018,2019 Claude Heiland-Allen
```

```
    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU Affero General Public License as
    published by the Free Software Foundation, either version 3 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 Affero General Public License for more details.

    You should have received a copy of the GNU Affero General Public License
    along with this program.  If not, see <https://www.gnu.org/licenses/>.
    -}

module Fractal.EscapeTime.Algorithms.R2.Reference (reference) where

import Prelude hiding (read)

import Fractal.EscapeTime.Compiler
import Fractal.EscapeTime.Formulas.Types hiding (i)

reference :: [(Int, R2Formula (Expr String))] -> String
reference fs = concat . flip map [FDouble] $ \t -> runCompile_ t . function NInt↙
    ↳ ("reference" ++ fSuffix t) [(NInt, "N"), (NFloat, "R2"), (NFloat, "D"), (↙
    ↳ NFloat, "E"), (NReal, "A"), (NReal, "B"), (NFloatPtr, "XYZS"), (↙
    ↳ NVolatileIntPtr, "RUNNING")] $ \i -> do
  early "mpfr_prec_t prec=mpfr_get_prec(A);\n"
  let n = EVar NInt "N"
      r2 = EVar NFloat "R2"
      d_ = EVar NFloat "D"
      e_ = EVar NFloat "E"
      a = EVar NReal "A"
      b = EVar NReal "B"
      xyzs = EVar NFloatPtr "XYZS"
      runningP = EVar NVolatileIntPtr "RUNNING"
  x <- real
  y <- real
  xf <- float
  yf <- float
  x .= 0
  y .= 0
  xf .= x
  yf .= y
  z2 <- float
  i .= 0
  z2 .= 0
  running <- bool
  running .= read runningP 0
  k <- int
  k .= 0
  while_ (ELess i n `EAnd` ELess z2 r2 `EAnd` running) $ do
    -- write output
    write xyzs (3 * i + 0) xf
    write xyzs (3 * i + 1) yf
    write xyzs (3 * i + 2) (z2 / 1024.0^2) -- XXX needs .0, otherwise i_inv_ii
    -- allocate next
    xn <- real
```

```
    yn <- real
    -- do formula
    let g (_, f) j kn = (,) j $ do
          let R2 fx fy = f (R2 d_ e_) (R2 a b) (R2 x y)
          xn .= optimize fx
          yn .= optimize fy
          k .= fromIntegral kn
    _ <- switch_ k $ zipWith3 g fs [0..] ([1 .. length fs - 1] ++ [0])
    -- update next
    x .= xn
    y .= yn
    xf .= x
    yf .= y
    z2 .= xf*xf + yf*yf
    -- update loop
    i .= i + 1
    running .= read runningP 0
  return_ i
```

34 src/Fractal/EscapeTime/Algorithms/R2/Size.hs

```
{-
et -- escape time fractals
Copyright (C) 2018 Claude Heiland-Allen

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU Affero General Public License as
published by the Free Software Foundation, either version 3 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 Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License
along with this program.  If not, see <https://www.gnu.org/licenses/>.
-}

module Fractal.EscapeTime.Algorithms.R2.Size (size) where

import Prelude hiding (read)

import Fractal.EscapeTime.Compiler
import Fractal.EscapeTime.Formulas.Types

-- FIXME this needs the degree of each stanza to be the *lowest* non-linear power
size :: [(Int, R2Formula (Expr String))] -> String
size fs = runCompile_ FDouble . function NBool "size" [(NInt, "P"), (NFloat, "D"), (NFloat, "E"), (NReal, "A"), (NReal, "B"), (NReal, "R"), (NFloatPtr, "M"), (NVolatileIntPtr, "RUNNING")] $ \s -> do
  let degree = exp $ sum (map (log . fromIntegral . fst) fs) / fromIntegral (length fs) :: Double
      deg = degree / (degree - 1)
      deg' = if isNaN deg || isInfinite deg then 0 else deg
      p = EVar NInt "P"
```

```
        d_ = EVar NFloat "D"
        e_ = EVar NFloat "E"
        a = EVar NReal "A"
        b = EVar NReal "B"
        r = EVar NReal "R"
        m = EVar NFloatPtr "M"
        runningP = EVar NVolatileIntPtr "RUNNING"
    early "mpfr_prec_t prec=mpfr_get_prec(A);\n"
    deg <- float
    deg .= realToFrac deg'
    x <- real
    y <- real
    lxa <- real
    lxb <- real
    lya <- real
    lyb <- real
    bxa <- real
    bxb <- real
    bya <- real
    byb <- real
    x .= 0
    y .= 0
    lxa .= 1
    lxb .= 0
    lya .= 0
    lyb .= 1
    bxa .= 1
    bxb .= 0
    bya .= 0
    byb .= 1
    -- loop one period
    j <- int
    j .= 1
    k <- int
    k .= 0
    running <- bool
    running .= read runningP 0
    while_ (j `ELess` p `EAnd` running) $ do
      -- alloc z, l
      xn <- real
      yn <- real
      lxan <- real
      lxbn <- real
      lyan <- real
      lybn <- real
      -- calc z, l
      let g (_, f) j0 kn = (,) j0 $ do
            let R2 gx gy = f (R2 d_ e_) (R2 a b) (R2 x y)
                fx = optimize gx
                fy = optimize gy
                fdxa = ddx d x fx
                fdxb = ddx d y fx
                fdya = ddx d x fy
                fdyb = ddx d y fy
                d v u
                  | u == x && v == x = lxa
                  | u == x && v == y = lxb
```

```
                    | u == y && v == x = lya
                    | u == y && v == y = lyb
                    | u == v = 1
                    | otherwise = 0
              xn .= fx
              yn .= fy
              x .= xn
              y .= yn
              lxan .= optimize fdxa
              lxbn .= optimize fdxb
              lyan .= optimize fdya
              lybn .= optimize fdyb
              lxa .= lxan
              lxb .= lxbn
              lya .= lyan
              lyb .= lybn
              k .= fromIntegral kn
        _ <- switch_ k $ zipWith3 g fs [0..] ([1 .. length fs - 1] ++ [0])
        -- step b
        det <- real
        det .= lxa * lyb - lxb * lya
        bxa .= bxa + lyb / det
        bxb .= bxb - lxb / det
        bya .= bya - lya / det
        byb .= byb + lxa / det
        -- loop
        j .= j + 1
        running .= read runningP 0
      -- l^d b
      l <- real
      beta <- real
      l .= sqrt (abs (lxa * lyb - lxb * lya))
      beta .= sqrt (abs (bxa * byb - bxb * bya))
      llb <- real
      llb .= exp (log l * deg) * beta
      r .= 1 / llb
      byb .= byb / beta
      bxb .= negate bxb / beta
      bya .= negate bya / beta
      bxa .= bxa / beta
      write m 0 byb
      write m 1 bxb
      write m 2 bya
      write m 3 bxa
      s .= running
      return_ s
```

35 src/Fractal/EscapeTime/Algorithms/R2/Skew.hs

```
{-
et -- escape time fractals
Copyright (C) 2018 Claude Heiland-Allen

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU Affero General Public License as
published by the Free Software Foundation, either version 3 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 Affero General Public License for more details.

    You should have received a copy of the GNU Affero General Public License
    along with this program.  If not, see <https://www.gnu.org/licenses/>.
    -}

    {-
    <https://fractalforums.org/fractal-mathematics-and-new-theories/28/automatic-skew-adjustment-for-burning-ship-etc/1777/msg9133#msg9133>
    Re: automatic skew adjustment for Burning Ship etc
    gerrit
    2018-09-05 19:01
    If c0 is your escaping reference location, compute orbit and
    matrices dfdc and dfdz (z = (x0,y0)).
    Skew matrix is then S = inv(dfdc)*dfdz, S -> S/sqrt(abs(det(S)))
    evaluated at escape iteration.
    Then c <-- S*(c-c0)+c0.
    -}

    module Fractal.EscapeTime.Algorithms.R2.Skew (skew) where

    import Prelude hiding (read)

    import Fractal.EscapeTime.Compiler
    import Fractal.EscapeTime.Formulas.Types

    skew :: [(Int, R2Formula (Expr String))] -> String
    skew fs = runCompile_ FDouble . function NBool "skew" [(NInt, "N"), (NFloat, "D"), (NFloat, "E"), (NReal, "A"), (NReal, "B"), (NBool, "DZ"), (NFloatPtr, "M"), (NVolatileIntPtr, "RUNNING")] $ \s -> do
      let n = EVar NInt "N"
          d_ = EVar NFloat "D"
          e_ = EVar NFloat "E"
          a = EVar NReal "A"
          b = EVar NReal "B"
          m = EVar NFloatPtr "M"
          useDZ = EVar NInt "DZ"
          r = 65536 -- FIXME take escape radius as parameter?
          runningP = EVar NVolatileIntPtr "RUNNING"
      early "mpfr_prec_t prec=mpfr_get_prec(A);\n"
      x   <- real
      y   <- real
      dxa <- real
      dxb <- real
      dya <- real
      dyb <- real
      dxx <- real
      dxy <- real
      dyx <- real
      dyy <- real
      x .= a
      y .= b
      dxa .= 1
```

```
    dxb .= 0
    dya .= 0
    dyb .= 1
    dxx .= 1
    dxy .= 0
    dyx .= 0
    dyy .= 1
    -- loop one period
    j <- int
    j .= 1
    k <- int
    k .= 0
    running <- bool
    running .= read runningP 0
    z2 <- real
    z2 .= x * x + y * y
    while_ ((j `ELess` n) `EAnd` (z2 `ELess` r) `EAnd` running) $ do
      -- alloc next
      xn <- real
      yn <- real
      dxan <- real
      dxbn <- real
      dyan <- real
      dybn <- real
      dxxn <- real
      dxyn <- real
      dyxn <- real
      dyyn <- real
      -- step
      let g (_, f) j0 kn = (,) j0 $ do
            let R2 gx gy = f (R2 d_ e_) (R2 a b) (R2 x y)
                fx = optimize gx
                fy = optimize gy
                d v u
                  | u == x && v == a = dxa
                  | u == x && v == b = dxb
                  | u == y && v == a = dya
                  | u == y && v == b = dyb
                  | u == x && v == x = dxx
                  | u == x && v == y = dxy
                  | u == y && v == x = dyx
                  | u == y && v == y = dyy
                  | u == v = 1
                  | otherwise = 0
            -- calc next
            xn .= fx
            yn .= fy
            dxan .= optimize (ddx d a fx)
            dxbn .= optimize (ddx d b fx)
            dyan .= optimize (ddx d a fy)
            dybn .= optimize (ddx d b fy)
            _ <- if_ useDZ (do
              dxxn .= optimize (ddx d x fx)
              dxyn .= optimize (ddx d y fx)
              dyxn .= optimize (ddx d x fy)
              dyyn .= optimize (ddx d y fy)) (return ())
            -- step vars
```

```
            x .= xn
            y .= yn
            dxa .= dxan
            dxb .= dxbn
            dya .= dyan
            dyb .= dybn
            _ <- if_ useDZ (do
              dxx .= dxxn
              dxy .= dxyn
              dyx .= dyxn
              dyy .= dyyn) (return ())
            z2 .= x * x + y * y
            k .= fromIntegral kn
        _ <- switch_ k $ zipWith3 g fs [0..] ([1 .. length fs - 1] ++ [0])
        -- loop
        j .= j + 1
        running .= read runningP 0
      _ <- if_ running (do
        -- s = normalize(dc^-1 dz)
        sii <- real
        sij <- real
        sji <- real
        sjj <- real
        _ <- if_ useDZ (do
          sii .= dyb * dxx - dxb * dyx
          sij .= dyb * dxy - dxb * dyy
          sji .= dxa * dyx - dya * dxx
          sjj .= dxa * dyy - dya * dxy) (do
          sii .=   dyb
          sij .= - dxb
          sji .= - dya
          sjj .=   dxa)
        det <- real
        det .= sqrt (abs (sii * sjj - sij * sji))
        sii .= sii / det
        sij .= sij / det
        sji .= sji / det
        sjj .= sjj / det
        write m 0 sii
        write m 1 sij
        write m 2 sji
        write m 3 sjj) (return ())
      s .= running
      return_ s
```

36 src/Fractal/EscapeTime/Compiler/CodeGen/Control.hs

```
{-
et -- escape time fractals
Copyright (C) 2018 Claude Heiland-Allen

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU Affero General Public License as
published by the Free Software Foundation, either version 3 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 Affero General Public License for more details.

    You should have received a copy of the GNU Affero General Public License
    along with this program.  If not, see <https://www.gnu.org/licenses/>.
    -}

    module Fractal.EscapeTime.Compiler.CodeGen.Control
      ( if_
      , switch_
      , while_
      , function
      , return_
      , break_
      ) where

    import Data.List (intercalate)

    import Control.Monad.RWS (get, put)

    import Fractal.EscapeTime.Compiler.Expr
    import Fractal.EscapeTime.Compiler.CodeGen.Core
    import Fractal.EscapeTime.Compiler.CodeGen.Expr

    if_ :: Expr String -> CompileM a -> CompileM b -> CompileM (a, b)
    if_ test true false = do
      ~b@(EVar _ v) <- bool
      b .= test
      emit $ "if (" ++ v ++ ") {\n"
      x <- true
      emit "} else {\n"
      y <- false
      emit "}\n"
      return (x, y)

    switch_ :: Expr String -> [(Int, CompileM a)] -> CompileM [a]
    switch_ e cs = do
      ~t@(EVar _ v) <- int
      t .= e
      emit $ "switch (" ++ v ++ ") {\n"
      rs <- mapM case_ cs
      emit $ "default: assert(0); /* internal error */ }\n"
      return rs
      where
        case_ (n, a) = do
          emit $ "case " ++ show n ++ ": {\n"
          r <- a
          emit $ "break; }\n"
          return r

    while_ :: Expr String -> CompileM a -> CompileM a
    while_ test body = do
      ~b@(EVar _ v) <- bool
      b .= test
      emit $ "while (" ++ v ++ ") {\n"
      x <- body
```

```
    b .= test
    emit "}\n"
    return x

return_ :: Expr String -> CompileM ()
return_ (EVar _ v) = do
  (nexts, ~(_:spares)) <- get
  late $ "return " ++ v ++ ";\n"
  put (nexts, []:spares)
return_ _ = error "Fractal.EscapeTime.Compile.CodeGen.Control.return_: not a variable"

break_ :: CompileM ()
break_ = emit "break;"

function :: NType -> String -> [(NType, String)] -> (Expr String -> CompileM a) -> CompileM a
function result name args body = bareScope $ do
  rt <- typeName result
  as <- mapM arg args
  early $ "static " ++ rt ++ " " ++ name ++ "(" ++ intercalate "," as ++ ")\n"
  early "{\n"
  retval <- alloc result
  r <- scope $ body retval
  late "}\n"
  return r
  where
    arg (t, v) = do
      at <- typeName t
      return $ at ++ " " ++ v
```

37 src/Fractal/EscapeTime/Compiler/CodeGen/Core.hs

```
{-
et -- escape time fractals
Copyright (C) 2018,2019 Claude Heiland-Allen

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU Affero General Public License as
published by the Free Software Foundation, either version 3 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 Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License
along with this program.  If not, see <https://www.gnu.org/licenses/>.
-}

{-# LANGUAGE FlexibleContexts #-}

module Fractal.EscapeTime.Compiler.CodeGen.Core where

import Prelude hiding (init)
import Data.List (delete)
```

```
25
    import Control.Monad.RWS

    import Fractal.EscapeTime.Compiler.Expr.AST

30  data FType = FSingle | FDouble | FLongDouble | FCompensated | FFloatExp
      deriving (Read, Show, Eq, Ord, Enum, Bounded)

    fTypes :: [FType]
    fTypes = [FSingle, FDouble, FLongDouble, FCompensated {-, FFloatExp -}]
35
    fSuffix :: FType -> String
    fSuffix FSingle = "f"
    fSuffix FDouble = ""
    fSuffix FLongDouble = "l"
40  fSuffix FCompensated = "c"
    fSuffix FFloatExp = "fe"

    type CompileM a = RWS FType (String, String, String) ([String], [[(NType, String↙
        ↳ )]]) a

45  early, late, emit :: String -> CompileM ()
    early s = tell (s, "", "")
    late s = tell ("", "", s)
    emit s = tell ("", s, "")

50  typeName :: NType -> CompileM String
    typeName NBool = return "int" -- C99 stdbool.h has no binding in Foreign.C.Types↙
        ↳  in older base
    typeName NInt = return "int"
    typeName NFloat = do
      b <- ask
55    return $ case b of
        FSingle -> "float"
        FDouble -> "double"
        FLongDouble -> "long double"
        FCompensated -> "compensated"
60      FFloatExp -> "cfloatexp"
    typeName NReal = return "mpfr_t"
    typeName NFloatPtr = do
      t <- typeName NFloat
      return $ t ++ "*"
65  typeName NIntPtr = return "int*"
    typeName NVolatileIntPtr = return "volatile int*"

    declare :: NType -> String -> CompileM (Expr String)
    declare t v = do
70    s <- typeName t
      early $ s ++ " " ++ v ++ ";\n"
      return (EVar t v)

    init :: Expr String -> CompileM ()
75  init (EVar NReal v) = early $ "mpfr_init2(" ++ v ++ ",prec);\n"
    init _ = return ()

    clear :: Expr String -> CompileM ()
    clear (EVar NReal v) = late $ "mpfr_clear(" ++ v ++ ");\n"
```

```
clear _ = return ()

alloc :: NType -> CompileM (Expr String)
alloc t = do
  (~(next:nexts), ~(spare:spares)) <- get
  case lookup t spare of
    Nothing -> do
      put (nexts, spare:spares)
      v <- declare t next
      init v
      clear v
      return v
    Just u -> do
      put (next:nexts, delete (t, u) spare : spares)
      return (EVar t u)

free :: Expr String -> CompileM ()
free (EVar t u) = do
  (nexts, ~(spare:spares)) <- get
  put (nexts, ((t, u):spare):spares)
free _ = error "Fractal.EscapeTime.Compile.CodeGen.Core.free: not a variable"

bool, int, float, real, intPtr, floatPtr :: CompileM (Expr String)
bool = alloc NBool
int = alloc NInt
float = alloc NFloat
real = alloc NReal
intPtr = alloc NIntPtr
floatPtr = alloc NFloatPtr

scope :: CompileM a -> CompileM a
scope a = do
  early "{\n"
  r <- bareScope a
  late "}\n"
  return r

bareScope :: CompileM a -> CompileM a
bareScope a = do
  (nexts, spares) <- get
  put (nexts, []:spares)
  r <- a
  put (nexts,    spares)
  return r

comment :: String -> CompileM ()
comment s = emit $ "// " ++ s ++ "\n"

runCompile :: FType -> CompileM a -> (a, String)
runCompile t a = case runRWS a t (map (('v':) . show) [(0::Integer)..], []) of
  (r, (_, []), (u, v, w)) -> (r, u ++ v ++ w)
  _ -> error "Fractal.EscapeTime.Compiler.CodeGen.Core.runCompile: internal error: bad scope?"

runCompile_ :: FType -> CompileM () -> String
runCompile_ t a = case runCompile t a of ((), w) -> w
```

38 src/Fractal/EscapeTime/Compiler/CodeGen/Expr.hs

```
{-
et -- escape time fractals
Copyright (C) 2018 Claude Heiland-Allen

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU Affero General Public License as
published by the Free Software Foundation, either version 3 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 Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License
along with this program.  If not, see <https://www.gnu.org/licenses/>.
-}

module Fractal.EscapeTime.Compiler.CodeGen.Expr where

import Fractal.EscapeTime.Compiler.CodeGen.Core
import Fractal.EscapeTime.Compiler.CodeGen.OpCode
import Fractal.EscapeTime.Compiler.Expr.AST

read :: Expr String -> Expr String -> Expr String
read = ERead

withVarAs :: NType -> Expr String -> (Expr String -> CompileM a) -> CompileM a
withVarAs s e@(EVar t _) a | s == t = a e
withVarAs s e a = do
  x <- alloc s
  expr x e
  r <- a x
  free x
  return r

withVar :: Expr String -> (Expr String -> CompileM a) -> CompileM a
withVar e@(EVar _ _) a = a e
withVar e a = do
  x <- alloc (eType e)
  expr x e
  r <- a x
  free x
  return r

withBool :: Expr String -> (Expr String -> CompileM a) -> CompileM a
withBool = withVarAs NBool

withInt :: Expr String -> (Expr String -> CompileM a) -> CompileM a
withInt = withVarAs NInt

withFloat :: Expr String -> (Expr String -> CompileM a) -> CompileM a
withFloat = withVarAs NFloat

withReal :: Expr String -> (Expr String -> CompileM a) -> CompileM a
```

```
withReal = withVarAs NReal

withFloatPtr :: Expr String -> (Expr String -> CompileM a) -> CompileM a
withFloatPtr = withVarAs NFloatPtr

withIntPtr :: Expr String -> (Expr String -> CompileM a) -> CompileM a
withIntPtr = withVarAs NIntPtr

withVolatileIntPtr :: Expr String -> (Expr String -> CompileM a) -> CompileM a
withVolatileIntPtr = withVarAs NVolatileIntPtr

write :: Expr String -> Expr String -> Expr String -> CompileM ()
write p a b =
  withFloatPtr p $ \q ->
  withInt a $ \x ->
  withFloat b $ \y ->
  op_write q x y

writeI :: Expr String -> Expr String -> Expr String -> CompileM ()
writeI p a b =
  withIntPtr p $ \q ->
  withInt a $ \x ->
  withInt b $ \y ->
  op_write q x y

(.=) :: Expr String -> Expr String -> CompileM ()
infix 0 .=
v .= e = expr v e

expr :: Expr String -> Expr String -> CompileM ()
expr v (EIf p t f) =
  withBool p $ \x ->
  withVarAs (eType v) t $ \y ->
  withVarAs (eType v) f $ \z ->
  op3 O'ifte v x y z
expr v (ERead p a) =
  withVar p $ \q ->
  withInt a $ \x ->
  op2 O'read v q x
expr v (EFloat a) =
  withFloat a $ \x ->
  op1 O'set v x
expr v (EInt n) = op_lit v n
expr v (ERat r) = op_litf v r
expr v (EPi _) = op0 O'pi v
expr v (EAdd _ a (EMul _ (-1) b)) =
  withVar a $ \x ->
  withVar b $ \y ->
  op2 O'sub v x y
expr v (EAdd _ a b) =
  withVar a $ \x ->
  withVar b $ \y ->
  op2 O'add v x y
expr v (EMul _ a b)
  | a == 1 = expr v b
  | b == 1 = expr v a
  | a == fromInteger (-1) = expr v b >> op1 O'neg v v
```

```
    | b == fromInteger (-1) = expr v a >> op1 O'neg v v
    | a == 2 = expr v b >> op1 O'mul2 v v
    | b == 2 = expr v a >> op1 O'mul2 v v
    | a == fromInteger (-2) = expr v b >> op1 O'mul2 v v >> op1 O'neg v v
    | b == fromInteger (-2) = expr v a >> op1 O'mul2 v v >> op1 O'neg v v
    | a == b = withVar a $ \x -> op1 O'sqr v x
    | otherwise = withVar a $ \x -> withVar b $ \y -> op2 O'mul v x y
expr v (ENeg _ a) = do
  v .= a
  op1 O'neg v v
expr v (EAbs _ a) = do
  v .= a
  op1 O'abs v v
expr v (ESgn  a) =
  withVar a $ \x ->
  op1 O'sgn v x
expr v a@(EVar _ _) =
  op1 O'set v a
expr v (ESqrt _ a) =
  withVar a $ \x ->
  op1 O'sqrt v x
expr v (EExpM1 _ a) =
  withVar a $ \x ->
  op1 O'expm1 v x
expr v (ELog1P _ a) =
  withVar a $ \x ->
  op1 O'log1p v x
expr v (EFloor _ a) =
  withVar a $ \x ->
  op1 O'floor v x
expr v (EInv _ a) =
  withVar a $ \x ->
  op1 O'inv v x

expr v (ESin _ a) =
  withVar a $ \x ->
  op1 O'sin v x
expr v (ECos _ a) =
  withVar a $ \x ->
  op1 O'cos v x
expr v (ETan _ a) =
  withVar a $ \x ->
  op1 O'tan v x
expr v (ESinh _ a) =
  withVar a $ \x ->
  op1 O'sinh v x
expr v (ECosh _ a) =
  withVar a $ \x ->
  op1 O'cosh v x
expr v (ETanh _ a) =
  withVar a $ \x ->
  op1 O'tanh v x
expr v (EASin _ a) =
  withVar a $ \x ->
  op1 O'asin v x
expr v (EACos _ a) =
  withVar a $ \x ->
```

```
    op1 O'acos v x
expr v (EATan _ a) =
  withVar a $ \x ->
  op1 O'atan v x
expr v (EASinh _ a) =
  withVar a $ \x ->
  op1 O'asinh v x
expr v (EACosh _ a) =
  withVar a $ \x ->
  op1 O'acosh v x
expr v (EATanh _ a) =
  withVar a $ \x ->
  op1 O'atanh v x

expr v (EDiffAbs _ a b) =
  withVar a $ \x ->
  withVar b $ \y ->
  op2 O'diffabs v x y
expr v (ELess a b) =
  withVar a $ \x ->
  withVar b $ \y ->
  op2 O'lt v x y
expr v (ELessEqual a b) =
  withVar a $ \x ->
  withVar b $ \y ->
  op2 O'le v x y
expr v (EEqual a b) =
  withVar a $ \x ->
  withVar b $ \y ->
  op2 O'eq v x y
expr v (EAnd a b) =
  withVar a $ \x ->
  withVar b $ \y ->
  op2 O'and v x y
expr v (EOr a b) =
  withVar a $ \x ->
  withVar b $ \y ->
  op2 O'or v x y
```

39 src/Fractal/EscapeTime/Compiler/CodeGen.hs

```
{-
et -- escape time fractals
Copyright (C) 2018 Claude Heiland-Allen

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU Affero General Public License as
published by the Free Software Foundation, either version 3 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 Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License
along with this program.  If not, see <https://www.gnu.org/licenses/>.
```

```
-}

module Fractal.EscapeTime.Compiler.CodeGen
  ( module Fractal.EscapeTime.Compiler.CodeGen.Core
  , module Fractal.EscapeTime.Compiler.CodeGen.Control
  , module Fractal.EscapeTime.Compiler.CodeGen.Expr
  , module Fractal.EscapeTime.Compiler.CodeGen.OpCode
  ) where

import Fractal.EscapeTime.Compiler.CodeGen.Core
import Fractal.EscapeTime.Compiler.CodeGen.Control
import Fractal.EscapeTime.Compiler.CodeGen.Expr
import Fractal.EscapeTime.Compiler.CodeGen.OpCode
```

40 src/Fractal/EscapeTime/Compiler/CodeGen/OpCode.hs

```
{-
et -- escape time fractals
Copyright (C) 2018,2019 Claude Heiland-Allen

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU Affero General Public License as
published by the Free Software Foundation, either version 3 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 Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License
along with this program.  If not, see <https://www.gnu.org/licenses/>.
-}

module Fractal.EscapeTime.Compiler.CodeGen.OpCode
  ( Op0(..), op0
  , Op1(..), op1
  , Op2(..), op2
  , Op3(..), op3
  , op_lit, op_litf
  , op_write
  )
  where

import Control.Monad.RWS (ask)

import Fractal.EscapeTime.Compiler.CodeGen.Core
import Fractal.EscapeTime.Compiler.Expr.AST

typname :: NType -> CompileM String
typname NBool = return "b"
typname NInt = return "i"
typname NIntPtr = return "I"
typname NVolatileIntPtr = return "V"
typname NReal = return "r"
typname NFloat = do
  t <- ask
```

```
    return $ case t of
      FSingle -> "f"
      FDouble -> "d"
      FLongDouble -> "l"
      FCompensated -> "c"
      FFloatExp -> "e"
typname NFloatPtr = do
  t <- ask
  return $ case t of
    FSingle -> "F"
    FDouble -> "D"
    FLongDouble -> "L"
    FCompensated -> "C"
    FFloatExp -> "E"

data Op0
  = O'zero
  | O'inf
  | O'ninf
  | O'nan
  | O'pi
  deriving (Eq, Ord, Enum, Bounded, Read, Show)

op0name :: NType -> Op0 -> CompileM String
op0name to op = do
  to' <- typname to
  return $ to' ++ "_" ++ drop 2 (show op)

op0 :: Op0 -> Expr String -> CompileM ()
op0 op (EVar to v) = do
  name <- op0name to op
  emit $ name ++ "(" ++ v ++ ");\n"

data Op1
  = O'set
  | O'abs
  | O'sgn
  | O'neg
  | O'inv
  | O'mul2
  | O'sqr
  | O'sqrt
  | O'exp
  | O'log
  | O'expm1
  | O'log1p
  | O'sin
  | O'cos
  | O'tan
  | O'sinh
  | O'cosh
  | O'tanh
  | O'asin
  | O'acos
  | O'atan
  | O'asinh
  | O'acosh
```

```
    | O'atanh
    | O'floor
    | O'not
    | O'comp
    deriving (Eq, Ord, Enum, Bounded, Read, Show)

op1name :: NType -> Op1 -> NType -> CompileM String
op1name to op from = do
  to' <- typname to
  from' <- typname from
  return $ to' ++ "_" ++ drop 2 (show op) ++ "_" ++ from'

op1 :: Op1 -> Expr String -> Expr String -> CompileM ()
op1 op (EVar to v) (EVar from x) = do
  name <- op1name to op from
  emit $ name ++ "(" ++ v ++ "," ++ x ++ ");\n"

data Op2
  = O'add
  | O'sub
  | O'mul
  | O'div
  | O'diffabs
  | O'lt
  | O'le
  | O'gt
  | O'ge
  | O'eq
  | O'ne
  | O'and
  | O'or
  | O'xor
  | O'shl
  | O'shr
  | O'read
  deriving (Eq, Ord, Enum, Bounded, Read, Show)

op2name :: NType -> Op2 -> NType -> NType -> CompileM String
op2name to op l r = do
  to' <- typname to
  l' <- typname l
  r' <- typname r
  return $ to' ++ "_" ++ drop 2 (show op) ++ "_" ++ l' ++ r'

op2 :: Op2 -> Expr String -> Expr String -> Expr String -> CompileM ()
op2 op (EVar to v) (EVar l x) (EVar r y) = do
  name <- op2name to op l r
  emit $ name ++ "(" ++ v ++ "," ++ x ++ "," ++ y ++ ");\n"

data Op3
  = O'ifte
  deriving (Eq, Ord, Enum, Bounded, Read, Show)

op3name :: NType -> Op3 -> NType -> NType -> NType -> CompileM String
op3name to op l r c = do
  to' <- typname to
  l' <- typname l
```

```
  r' <- typname r
  c' <- typname c
  return $ to' ++ "_" ++ drop 2 (show op) ++ "_" ++ l' ++ r' ++ c'

op3 :: Op3 -> Expr String -> Expr String -> Expr String -> CompileM ()
op3 op (EVar to v) (EVar l x) (EVar r y) (EVar k z) = do
  name <- op3name to op l r k
  emit $ name ++ "(" ++ v ++ "," ++ x ++ "," ++ y ++ "," ++ z ++ ");\n"

op_lit :: Expr String -> Integer -> CompileM ()
op_lit (EVar NReal v) n = emit $ "mpfr_set_si(" ++ v ++ "," ++ show n ++ ",MPFR_RNDN);\n"
op_lit (EVar NInt v) n = emit $ v ++ "=" ++ show n ++ ";\n"
op_lit (EVar NBool v) n = emit $ v ++ "=" ++ show n ++ ";\n"
op_lit (EVar NFloat v) n = do
  t <- ask
  case t of
    FCompensated -> emit $ "c_set_i(" ++ v ++ "," ++ show n ++ ")" ++ ";\n"
    FFloatExp -> emit $ "e_set_i(" ++ v ++ "," ++ show n ++ ")" ++ ";\n"
    _ -> emit $ v ++ "=" ++ show n ++ ";\n"
op_lit v n = error $ show v ++ " = " ++ show n

op_litf :: Expr String -> Rational -> CompileM ()
op_litf (EVar NReal v) f = emit $ "mpfr_set_d(" ++ v ++ "," ++ show (fromRational f :: Double) ++ ",MPFR_RNDN);\n"
op_litf (EVar NFloat v) f = do
  t <- ask
  case t of
    FCompensated -> emit $ "c_set_d(" ++ v ++ "," ++ show (fromRational f :: Double) ++ ")" ++ ";\n"
    FFloatExp -> emit $ "e_set_d(" ++ v ++ "," ++ show (fromRational f :: Double) ++ ");\n"
    _ -> emit $ v ++ "=" ++ show (fromRational f :: Double) ++ ";\n"

op_write :: Expr String -> Expr String -> Expr String -> CompileM ()
op_write (EVar vt v) (EVar it i) (EVar xt x) = do
  vt' <- typname vt
  it' <- typname it
  xt' <- typname xt
  emit $ "v_write_" ++ vt' ++ it' ++ xt' ++ "(" ++ v ++ "," ++ i ++ "," ++ x ++ ");\n"

op_pi :: Expr String -> CompileM ()
op_pi (EVar NReal v) = emit $ "mpfr_const_pi(" ++ v ++ ",MPFR_RNDN);\n"
op_pi (EVar _     v) = emit $ v ++ "=3.14159265358979323846264338327950288 4L;\n"
```

41 src/Fractal/EscapeTime/Compiler/Expr/AST.hs

```
{-
et -- escape time fractals
Copyright (C) 2018,2019 Claude Heiland-Allen

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU Affero General Public License as
published by the Free Software Foundation, either version 3 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 Affero General Public License for more details.

    You should have received a copy of the GNU Affero General Public License
    along with this program.  If not, see <https://www.gnu.org/licenses/>.
    -}

    {-# LANGUAGE DeriveDataTypeable #-}
    module Fractal.EscapeTime.Compiler.Expr.AST where

    import Data.Data
    import Data.Typeable

    import Numeric

    data NType = NBool | NInt | NFloat | NReal | NFloatPtr | NIntPtr | ↙
        ↳ NVolatileIntPtr
      deriving (Read, Show, Eq, Ord, Enum, Bounded, Data, Typeable)

    data Expr a
      -- these ones can be perturbed
      = EInt          Integer
      | ERat          Rational
      | EVar   NType a
      | ENeg   NType (Expr a)
      | EInv   NType (Expr a) -- FIXME handle integer division properly?
      | EAdd   NType (Expr a) (Expr a)
      | EMul   NType (Expr a) (Expr a)
      | EAbs   NType (Expr a)
      | ESin   NType (Expr a)
      | ECos   NType (Expr a)
      | ETan   NType (Expr a)
      | ESinh  NType (Expr a)
      | ECosh  NType (Expr a)
      | ETanh  NType (Expr a)
      | EPi    NType
      -- these can be perturbed in some limited circumstances
      | EExpM1 NType (Expr a)
      | ELog1P NType (Expr a)
      | ESqrt  NType (Expr a)
      -- these ones can't be perturbed
      | EFloat        (Expr a)
      | ESgn          (Expr a)
      | EASin  NType (Expr a)
      | EACos  NType (Expr a)
      | EATan  NType (Expr a)
      | EASinh NType (Expr a)
      | EACosh NType (Expr a)
      | EATanh NType (Expr a)
      | EFloor NType (Expr a)
      | ELess        (Expr a) (Expr a)
      | ELessEqual   (Expr a) (Expr a)
      | EEqual       (Expr a) (Expr a)
      | EOr          (Expr a) (Expr a)
```

```
    | EAnd         (Expr a) (Expr a)
    | ERead       (Expr a) (Expr a)
    | EDiffAbs NType (Expr a) (Expr a)
    | EIf (Expr a) (Expr a) (Expr a)
  deriving (Read, Show, Eq, Ord, Data, Typeable)

interpret :: (Show v, Eq v, Floating a) => [(v, a)] -> Expr v -> Maybe a
-- constant
interpret _ (EInt n) = Just $ fromInteger n
interpret _ (ERat q) = Just $ fromRational q
interpret _ (EPi _) = Just pi
-- variable
interpret vs (EVar _ v) = lookup v vs
-- functor
interpret vs (EAdd _ e f) = (+) <$> interpret vs e <*> interpret vs f
interpret vs (EMul _ e f) = (*) <$> interpret vs e <*> interpret vs f
interpret vs (EInv _ e) = recip <$> interpret vs e
interpret vs (ENeg _ e) = negate <$> interpret vs e
interpret vs (EAbs _ e) = abs <$> interpret vs e
interpret vs (ESin _ e) = sin <$> interpret vs e
interpret vs (ECos _ e) = cos <$> interpret vs e
interpret vs (ETan _ e) = tan <$> interpret vs e
interpret vs (ESinh _ e) = sinh <$> interpret vs e
interpret vs (ECosh _ e) = cosh <$> interpret vs e
interpret vs (ETanh _ e) = tanh <$> interpret vs e
interpret vs (EASin _ e) = asin <$> interpret vs e
interpret vs (EACos _ e) = acos <$> interpret vs e
interpret vs (EATan _ e) = atan <$> interpret vs e
interpret vs (EASinh _ e) = asinh <$> interpret vs e
interpret vs (EACosh _ e) = acosh <$> interpret vs e
interpret vs (EATanh _ e) = atanh <$> interpret vs e
interpret vs (ESqrt _ e) = sqrt <$> interpret vs e
interpret vs (EExpM1 _ e) = expm1 <$> interpret vs e
interpret vs (ELog1P _ e) = log1p <$> interpret vs e
interpret vs (EDiffAbs _ e f) = (\x y -> abs (x + y) - abs x) <$> interpret vs e↙
    ↳ <*> interpret vs f
interpret _ e = error (show e)
-- unimplemented
--interpret _ _ = Nothing

eType :: Expr a -> NType
eType (EInt    _   ) = NInt
eType (ERat    _   ) = NFloat
eType (EVar  t _   ) = t
eType (ENeg  t _   ) = t
eType (EInv  t _   ) = t
eType (EAdd  t _ _ ) = t
eType (EMul  t _ _ ) = t
eType (EAbs  t _   ) = t
eType (ESin  t _   ) = t
eType (ECos  t _   ) = t
eType (ETan  t _   ) = t
eType (ESinh t _   ) = t
eType (ECosh t _   ) = t
eType (ETanh t _   ) = t
eType (EPi   t     ) = t
eType (EExpM1 t _  ) = t
```

```
    eType (ELog1P t _ ) = t
    eType (EFloat _   ) = NFloat
    eType (ESgn   _   ) = NInt
    eType (ESqrt t _  ) = t
125 eType (EASin  t _ ) = t
    eType (EACos  t _ ) = t
    eType (EATan  t _ ) = t
    eType (EASinh t _ ) = t
    eType (EACosh t _ ) = t
130 eType (EATanh t _ ) = t
    eType (EFloor t  _) = t
    eType (ELess    _ _) = NBool
    eType (ELessEqual _ _) = NBool
    eType (EEqual   _ _) = NBool
135 eType (EAnd     _ _) = NBool
    eType (EOr      _ _) = NBool
    eType (ERead (EVar NFloatPtr       _) _) = NFloat
    eType (ERead (EVar NIntPtr         _) _) = NInt
    eType (ERead (EVar NVolatileIntPtr _) _) = NInt
140 eType (ERead _ _) = error $ "Fractal.EscapeTime.Compiler.Expr.AST.eType: ERead ↙
      ↳ not a pointer type"
    eType (EDiffAbs t _ _) = t
    eType (EIf _ t e) = eeType t e

    eeType :: Expr a -> Expr a -> NType
145 eeType a b = case (eType a, eType b) of
      (NReal, _) -> NReal
      (_, NReal) -> NReal
      (NFloat, _) -> NFloat
      (_, NFloat) -> NFloat
150   (NInt, _) -> NInt
      (_, NInt) -> NInt
      _ -> NBool -- FIXME

    instance Num (Expr a) where
155   fromInteger a = EInt a
      a + b = EAdd (eeType a b) a b
      a * b = EMul (eeType a b) a b
      negate a = ENeg (eType a) a
      abs a = EAbs (eType a) a
160   signum a = ESgn a

    instance Fractional (Expr a) where
      fromRational = ERat
      recip x = EInv (case eType x of
165       NReal -> NReal
          _ -> NFloat) x

    instance Floating (Expr a) where
      pi = EPi NReal
170   exp x = expm1 x + 1
      log x = log1p (1 + x)
      sqrt x = ESqrt (eType x) x
      sin x = ESin (eType x) x
      cos x = ECos (eType x) x
175   tan x = ETan (eType x) x
      sinh x = ESinh (eType x) x
```

```
  cosh x = ECosh (eType x) x
  tanh x = ETanh (eType x) x
  asin x = EASin (eType x) x
  acos x = EACos (eType x) x
  atan x = EATan (eType x) x
  asinh x = EASinh (eType x) x
  acosh x = EACosh (eType x) x
  atanh x = EATanh (eType x) x
  expm1 x = EExpM1 (eType x) x
  log1p x = ELog1P (eType x) x

diffabs :: Expr a -> Expr a -> Expr a
diffabs a b = EDiffAbs (eeType a b) a b
```

42 src/Fractal/EscapeTime/Compiler/Expr/Deriv.hs

```
{-
et -- escape time fractals
Copyright (C) 2018 Claude Heiland-Allen

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU Affero General Public License as
published by the Free Software Foundation, either version 3 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 Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License
along with this program.  If not, see <https://www.gnu.org/licenses/>.
-}

module Fractal.EscapeTime.Compiler.Expr.Deriv (ddx) where

import Fractal.EscapeTime.Compiler.Expr.AST

ddx :: Eq a => (Expr a -> Expr a -> Expr a) -> Expr a -> Expr a -> Expr a
ddx _ _ (EInt _) = 0
ddx _ _ (ERat _) = 0
ddx d x a@(EVar _ _) = d x a
ddx d x (ENeg _ a) = negate (ddx d x a)
ddx d x (EInv _ a) = negate (ddx d x a) / a * a
ddx d x (EAdd _ a b) = ddx d x a + ddx d x b
ddx d x (EMul _ a b) = ddx d x a * b + a * ddx d x b
ddx d x (EAbs _ a) = signum a * ddx d x a
ddx d x (ESin _ a) =  cos a * ddx d x a
ddx d x (ECos _ a) = -sin a * ddx d x a
ddx d x (ETan _ a) = ddx d x a / cos a ^ 2
ddx d x (ESinh _ a) = cosh a * ddx d x a
ddx d x (ECosh _ a) = sinh a * ddx d x a
ddx d x (ETanh _ a) = ddx d x a / cosh a ^ 2
ddx _ _ (EPi _) = 0
ddx d x (EExpM1 _ a) = exp a * ddx d x a
ddx d x (ELog1P _ a) = ddx d x a / (1 + a)
ddx d x (EFloat a) = EFloat $ ddx d x a
```

```
ddx _ _ (ESgn  _) = 0
ddx d x (ESqrt _ a) = ddx d x a / (2 * sqrt a)
ddx d x (EASin _ a) =  ddx d x a / sqrt (1 - x * x)
ddx d x (EACos _ a) = -ddx d x a / sqrt (1 - x * x)
ddx d x (EATan _ a) = ddx d x a / (1 + x * x)
ddx d x (EASinh _ a) = ddx d x a / sqrt (x * x + 1)
ddx d x (EACosh _ a) = ddx d x a / sqrt (x * x - 1)
ddx d x (EATanh _ a) = ddx d x a / (1 - x * x)
ddx _ _ (EFloor _ _) = 0
ddx d x (EDiffAbs _ a b) = ddx d x (abs (a + b)) - ddx d x (abs a)
ddx d x (EIf t a b) = EIf t (ddx d x a) (ddx d x b)
ddx _ _ _ = error "Fractal.EscapeTime.Compiler.Expr.Deriv.ddx: cant ↙
    ↳ differentiate"
```

43 src/Fractal/EscapeTime/Compiler/Expr.hs

```
module Fractal.EscapeTime.Compiler.Expr
  ( module Fractal.EscapeTime.Compiler.Expr.AST
  , module Fractal.EscapeTime.Compiler.Expr.Deriv
  , module Fractal.EscapeTime.Compiler.Expr.Optimize
  , module Fractal.EscapeTime.Compiler.Expr.Perturb
  , module Fractal.EscapeTime.Compiler.Expr.Pretty
  , module Fractal.EscapeTime.Compiler.Expr.Simplify
  ) where

import Fractal.EscapeTime.Compiler.Expr.AST
import Fractal.EscapeTime.Compiler.Expr.Deriv
import Fractal.EscapeTime.Compiler.Expr.Optimize
import Fractal.EscapeTime.Compiler.Expr.Perturb
import Fractal.EscapeTime.Compiler.Expr.Pretty
import Fractal.EscapeTime.Compiler.Expr.Simplify
```

44 src/Fractal/EscapeTime/Compiler/Expr/Optimize.hs

```
{-
et -- escape time fractals
Copyright (C) 2018 Claude Heiland-Allen

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU Affero General Public License as
published by the Free Software Foundation, either version 3 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 Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License
along with this program.  If not, see <https://www.gnu.org/licenses/>.
-}

module Fractal.EscapeTime.Compiler.Expr.Optimize (superoptimize, metric) where

import Data.Data
import Data.List (minimumBy, sort)
```

```
import Data.Ord (comparing)
import Data.Set (Set, empty, singleton, unions, size, toList, fromList)
import Data.Generics.Uniplate.Data (contexts)

import Debug.Trace

import Fractal.EscapeTime.Compiler.Expr.AST

step :: (Ord t, Data t) => Set (Expr t) -> Set (Expr t)
step s = fromList [ c y | x <- toList s, (e, c) <- contexts x, let t = rules ↙
    ↳ allRules e, y <- toList t ]

fix f s = traceShow (size s) $ let t = f s in if t == s then s else fix f t

superoptimize :: (Ord t, Data t) => Int -> Int -> Expr t -> Expr t
superoptimize m n = minimumBy (comparing metric) . go n . singleton
  where
    go 0 s = s
    go n s = (\t -> if s == t then t else go (n - 1) t) . topm . step $ s
    topm = fromList . take m . map snd . sort . map (\e -> (metric e, e)) . ↙
        ↳ toList

depth :: Expr t -> Integer
depth (EInt _) = 1
depth (ERat _) = 1
depth (EVar _ _) = 1
depth (ENeg _ a) = 1 + depth a
depth (EInv _ a) = 1 + depth a
depth (EAdd _ a b) = 1 + max (depth a) (depth b)
depth (EMul _ a b) = 1 + max (depth a) (depth b)
depth (EAbs _ a) = 1 + depth a
depth (ESin _ a) = 1 + depth a
depth (ECos _ a) = 1 + depth a
depth (ETan _ a) = 1 + depth a
depth (ESinh _ a) = 1 + depth a
depth (ECosh _ a) = 1 + depth a
depth (ETanh _ a) = 1 + depth a
depth (EPi _) = 1
depth (EExpM1 _ a) = 1 + depth a
depth (ELog1P _ a) = 1 + depth a
depth (EFloat a) = 1 + depth a
depth (ESgn a) = 1 + depth a
depth (ESqrt _ a) = 1 + depth a
depth (EASin _ a) = 1 + depth a
depth (EACos _ a) = 1 + depth a
depth (EATan _ a) = 1 + depth a
depth (EASinh _ a) = 1 + depth a
depth (EACosh _ a) = 1 + depth a
depth (EATanh _ a) = 1 + depth a
depth (EFloor _ a) = 1 + depth a
depth (ELess a b) = 1 + max (depth a) (depth b)
depth (ELessEqual a b) = 1 + max (depth a) (depth b)
depth (EEqual a b) = 1 + max (depth a) (depth b)
depth (EOr a b) = 1 + max (depth a) (depth b)
depth (EAnd a b) = 1 + max (depth a) (depth b)
depth (ERead a b) = 1 + max (depth a) (depth b)
depth (EDiffAbs _ a b) = 1 + max (depth a) (depth b)
```

```
depth (EIf t a b) = 1 + max (depth t) (max (depth a) (depth b))

cost :: Ord t => Expr t -> Integer
cost (EInt 0)     = 100000000
cost (EInt 1)     = 1000000
cost (EInt _) = 1
cost (ERat 0)     = 100000000
cost (ERat 1)     = 1000000
cost (EInt _) = 1
cost (EVar _ _) = 50
cost (ENeg _ a) = 1000000 + cost a
cost (EInv _ a) = 100 + cost a
cost (EAdd _ a b) = 10 + cost a + cost b
cost (EMul _ a b) = (if a == b then 30 else 50) + cost a + cost b
cost (EAbs _ a) = 3 + cost a
cost (ESin _ a) = 1000 + cost a
cost (ECos _ a) = 1000 + cost a
cost (ETan _ a) = 1000 + cost a
cost (ESinh _ a) = 1000 + cost a
cost (ECosh _ a) = 1000 + cost a
cost (ETanh _ a) = 1000 + cost a
cost (EPi _) = 1
cost (EExpM1 _ a) = 1000 + cost a
cost (ELog1P _ a) = 1000 + cost a
cost (EFloat a) = 5 + cost a
cost (ESgn a) = 3 + cost a
cost (ESqrt _ a) = 500 + cost a
cost (EASin _ a) = 1000 + cost a
cost (EACos _ a) = 1000 + cost a
cost (EATan _ a) = 1000 + cost a
cost (EASinh _ a) = 1000 + cost a
cost (EACosh _ a) = 1000 + cost a
cost (EATanh _ a) = 1000 + cost a
cost (EFloor _ a) = 500 + cost a
cost (ELess a b) = 10 + cost a + cost b
cost (ELessEqual a b) = 10 + cost a + cost b
cost (EEqual a b) = 10 + cost a + cost b
cost (EOr a b) = 10 + cost a + cost b
cost (EAnd a b) = 10 + cost a + cost b
cost (ERead a b) = 10 + cost a + cost b
cost (EDiffAbs _ a b) = 10000 + cost a + cost b
cost (EIf t a b) = 10 + cost t + cost a + cost b

metric :: Ord t => Expr t -> (Integer, Integer)
metric a = (cost a, depth a)

type Rule t = Expr t -> Set (Expr t)

rule :: Ord t => Rule t -> Set (Expr t) -> Maybe (Set (Expr t))
rule r s = let t = unions (map r (toList s)) in if s == t then Nothing else Just↙
    ↳ t

rules :: (Ord t, Data t) => [Rule t] -> Rule t
rules rs e = unions (map ($ e) rs)

allRules :: Ord t => [Rule t]
allRules =
```

```
    [ identity
135 , add_id
    , mul_id
    , zero
    , negate_mul
    , distribute
140 , constant
    , abs_mul
    , associate_r
    , associate_l
    , swap
145 , undistribute_i
    , undistribute
    , abs_i
    , mul_signum
    , signum_i
150 , sin_sin
    , cos_cos
    , tan_tan
    , sinh_sinh
    , cosh_cosh
155 , tanh_tanh
    ]

-- identity

160 identity e = singleton e

-- identity

add_id (EAdd _ (EInt 0) b) = singleton b
165 add_id (EAdd _ b (EInt 0)) = singleton b
add_id (EAdd _ (ERat 0) b) = singleton b
add_id (EAdd _ b (ERat 0)) = singleton b
add_id _ = empty

170 mul_id (EMul _ (EInt 1) b) = singleton b
mul_id (EMul _ b (EInt 1)) = singleton b
mul_id (EMul _ (ERat 1) b) = singleton b
mul_id (EMul _ b (ERat 1)) = singleton b
mul_id _ = empty
175
-- zero

zero (ENeg _ (EInt 0)) = singleton 0
zero (EAbs _ (EInt 0)) = singleton 0
180 zero (EMul _ (EInt 0) _) = singleton 0
zero (EMul _ _ (EInt 0)) = singleton 0
zero (EMul _ (ERat 0) _) = singleton 0
zero (EMul _ _ (ERat 0)) = singleton 0
zero _ = empty
185
-- negate

negate_mul (ENeg t a) = singleton $ EMul t (EInt (-1)) a
negate_mul (EMul t (EInt (-1)) a) = singleton $ ENeg t a
190 negate_mul _ = empty
```

```
-- distribute

distribute (EMul t a (EAdd _ b c)) = singleton $ EAdd t (EMul t a b) (EMul t a c)
distribute (EMul t (EAdd _ a b) c) = singleton $ EAdd t (EMul t a c) (EMul t b c)
distribute _ = empty

-- combine constants

constant (ENeg t (EInt a)) = singleton (EInt (negate a))
constant (ENeg t (ERat a)) = singleton (ERat (negate a))
constant (EAdd _ (EInt a) (EInt b)) = singleton $ EInt (a + b)
constant (EAdd t (EInt a) (EAdd _ (EInt b) c)) = singleton $ EAdd t (EInt (a + b)) c
constant (EAdd _ (EInt a) (ERat b)) = singleton $ ERat $ fromInteger a + b
constant (EAdd t (EInt a) (EAdd _ (ERat b) c)) = singleton $ EAdd t (ERat (fromInteger a + b)) c
constant (EAdd _ (ERat a) (ERat b)) = singleton $ ERat (a + b)
constant (EAdd t (ERat a) (EAdd _ (ERat b) c)) = singleton $ EAdd t (ERat (a + b)) c
constant (EMul _ (EInt a) (EInt b)) = singleton $ EInt (a * b)
constant (EMul t (EInt a) (EMul _ (EInt b) c)) = singleton $ EMul t (EInt (a * b)) c
constant (EMul _ (EInt a) (ERat b)) = singleton $ ERat $ fromInteger a * b
constant (EMul t (EInt a) (EMul _ (ERat b) c)) = singleton $ EMul t (ERat (fromInteger a * b)) c
constant (EMul _ (ERat a) (ERat b)) = singleton $ ERat (a * b)
constant (EMul t (ERat a) (EMul _ (ERat b) c)) = singleton $ EMul t (ERat (a * b)) c
constant _ = empty

-- combine abs*abs

abs_mul (EMul t (EAbs _ a) (EAbs _ b))
  | a == b = singleton $ EMul t a b
  | otherwise = singleton $ EAbs t (EMul t a b)
abs_mul (EMul t (EAbs _ a) (EMul _ (EAbs _ b) c))
  | a == b = singleton $ EMul t (EMul t a b) c
  | otherwise = singleton $ EMul t (EAbs t (EMul t a b)) c
abs_mul _ = empty

-- associate

associate_l (EAdd t c (EAdd _ a b)) = singleton $ EAdd t (EAdd t c a) b
associate_l (EMul t c (EMul _ a b)) = singleton $ EMul t (EMul t c a) b
associate_l _ = empty

associate_r (EAdd t (EAdd _ a b) c) = singleton $ EAdd t a (EAdd t b c)
associate_r (EMul t (EMul _ a b) c) = singleton $ EMul t a (EMul t b c)
associate_r _ = empty

-- swap

swap (EMul t a b) = singleton $ EMul t b a
swap (EAdd t a b) = singleton $ EAdd t b a
```

```
swap _ = empty

-- undistribute int

undistribute_i (EAdd _ (EMul _ (EInt a) x) (EAdd _ (EMul _ (EInt b) y) c)) | x ↙
    ↳ == y = singleton $ fromInteger (a + b) * x + c
undistribute_i (EAdd _ x (EAdd _ (EMul _ (EInt b) y) c)) | x == y = singleton $ ↙
    ↳ fromInteger (1 + b) * x + c
undistribute_i (EAdd _ (EMul _ (EInt a) x) (EAdd _ y c)) | x == y = singleton $ ↙
    ↳ fromInteger (a + 1) * x + c
undistribute_i (EAdd _ (EMul _ (EInt a) x) (EMul _ (EInt b) y)) | x == y = ↙
    ↳ singleton $ fromInteger (a + b) * x
undistribute_i (EAdd _ x (EMul _ (EInt b) y)) | x == y = singleton $ fromInteger↙
    ↳  (1 + b) * x
undistribute_i (EAdd _ (EMul _ (EInt a) x) y) | x == y = singleton $ fromInteger↙
    ↳  (a + 1) * x
undistribute_i (EAdd _ x (EAdd _ y c)) | x == y = singleton $ 2 * x + c
undistribute_i (EAdd _ x y) | x == y = singleton $ 2 * x
undistribute_i _ = empty

-- undistribute

undistribute (EAdd _ (EMul _ x1 y1) (EMul _ x2 y2)) | x1 == x2 = singleton $ x1 ↙
    ↳ * (y1 + y2)
undistribute (EAdd _ (EMul _ y1 x1) (EMul _ x2 y2)) | x1 == x2 = singleton $ x1 ↙
    ↳ * (y1 + y2)
undistribute (EAdd _ (EMul _ x1 y1) (EMul _ y2 x2)) | x1 == x2 = singleton $ x1 ↙
    ↳ * (y1 + y2)
undistribute (EAdd _ (EMul _ y1 x1) (EMul _ y2 x2)) | x1 == x2 = singleton $ x1 ↙
    ↳ * (y1 + y2)
undistribute (EAdd _ (EMul _ x1 y1) (EAdd _ (EMul _ x2 y2) c)) | x1 == x2 = ↙
    ↳ singleton $ x1 * (y1 + y2) + c
undistribute (EAdd _ (EMul _ y1 x1) (EAdd _ (EMul _ x2 y2) c)) | x1 == x2 = ↙
    ↳ singleton $ x1 * (y1 + y2) + c
undistribute (EAdd _ (EMul _ x1 y1) (EAdd _ (EMul _ y2 x2) c)) | x1 == x2 = ↙
    ↳ singleton $ x1 * (y1 + y2) + c
undistribute (EAdd _ (EMul _ y1 x1) (EAdd _ (EMul _ y2 x2) c)) | x1 == x2 = ↙
    ↳ singleton $ x1 * (y1 + y2) + c
{-
undistribute (EAdd _ (EDiv _ y1 x1) (EDiv _ y2 x2)) | x1 == x2 = singleton $ (y1↙
    ↳  + y2) / x1
undistribute (EAdd _ (EDiv _ y1 x1) (EDiv _ y2 x2)) | x1 == x2 = singleton $ (y1↙
    ↳  + y2) / x1
undistribute (EAdd _ (EDiv _ y1 x1) (EAdd _ (EDiv _ y2 x2) c)) | x1 == x2 = ↙
    ↳ singleton $ (y1 + y2) / x1 + c
undistribute (EAdd _ (EDiv _ y1 x1) (EAdd _ (EDiv _ y2 x2) c)) | x1 == x2 = ↙
    ↳ singleton $ (y1 + y2) / x1 + c
-}
undistribute _ = empty

-- misc

abs_i (EAbs _ (EMul _ (EInt a) b)) = singleton $ fromInteger (abs a) * abs b
abs_i _ = empty

mul_signum (ESgn (EMul _ a b)) = singleton $ signum a * signum b
mul_signum _ = empty
```

```
signum_i (ESgn (EInt a)) = singleton $ EInt (signum a)
signum_i _ = empty

-- sin + sin

sin_sin (EAdd _ (ESin _ a) (ESin _ b)) = singleton $ 2 * sin ((a + b) / 2) * cos ((a - b) / 2)
sin_sin (EAdd _ (ESin _ a) (EAdd _ (ESin _ b) c)) = singleton $ 2 * sin ((a + b) / 2) * cos ((a - b) / 2) + c
sin_sin _ = empty

-- cos + cos

cos_cos (EAdd _ (ECos _ a) (ECos _ b)) = singleton $ 2 * cos ((a + b) / 2) * cos ((a - b) / 2)
cos_cos (EAdd _ (ECos _ a) (EAdd _ (ECos _ b) c)) = singleton $ 2 * cos ((a + b) / 2) * cos ((a - b) / 2) + c
cos_cos _ = empty

-- tan + tan

tan_tan (EAdd _ (ETan _ a) (ETan _ b)) = singleton $ sin (a + b) / (cos a * cos b)
tan_tan (EAdd _ (ETan _ a) (EAdd _ (ETan _ b) c)) = singleton $ sin (a + b) / (cos a * cos b) + c
tan_tan _ = empty

-- sinh + sinh

sinh_sinh (EAdd _ (ESinh _ a) (ESinh _ b)) = singleton $ 2 * sinh ((a + b) / 2) * cosh ((a - b) / 2)
sinh_sinh (EAdd _ (ESinh _ a) (EAdd _ (ESinh _ b) c)) = singleton $ 2 * sinh ((a + b) / 2) * cosh ((a - b) / 2) + c
sinh_sinh _ = empty

-- cosh + cosh

cosh_cosh (EAdd _ (ECosh _ a) (ECosh _ b)) = singleton $ 2 * cosh ((a + b) / 2) * cosh ((a - b) / 2)
cosh_cosh (EAdd _ (ECosh _ a) (EAdd _ (ECosh _ b) c)) = singleton $ 2 * cosh ((a + b) / 2) * cosh ((a - b) / 2) + c
cosh_cosh _ = empty

-- tanh + tanh

tanh_tanh (EAdd _ (ETanh _ a) (ETanh _ b)) = singleton $ sinh (a + b) / (cosh a * cosh b)
tanh_tanh (EAdd _ (ETanh _ a) (EAdd _ (ETanh _ b) c)) = singleton $ sinh (a + b) / (cosh a * cosh b) + c
tanh_tanh _ = empty
```

45 src/Fractal/EscapeTime/Compiler/Expr/Perturb.hs

```
{-
et -- escape time fractals
Copyright (C) 2018,2019 Claude Heiland-Allen
```

```
{-  This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU Affero General Public License as
    published by the Free Software Foundation, either version 3 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 Affero General Public License for more details.

    You should have received a copy of the GNU Affero General Public License
    along with this program.  If not, see <https://www.gnu.org/licenses/>.
-}

module Fractal.EscapeTime.Compiler.Expr.Perturb where

import Fractal.EscapeTime.Compiler.Expr.AST
import Fractal.EscapeTime.Compiler.Expr.Simplify

widenE :: (Eq a, Show a) => [(a, (Expr a, Expr a))] -> Expr a -> Expr a
widenE _ n@(EInt _) = n
widenE _ n@(ERat _) = n
widenE vs (EVar _ v) = case lookup v vs of Just (x, y) -> x + y ; Nothing -> error $ "widenE " ++ show v
widenE vs (ENeg _ a) = negate (widenE vs a)
widenE vs (EInv _ a) = recip (widenE vs a)
widenE vs (EAdd _ a b) = widenE vs a + widenE vs b
widenE vs (EMul _ a b) = widenE vs a * widenE vs b
widenE vs (EAbs _ a) = abs (widenE vs a)
widenE vs (ESin _ a) = sin (widenE vs a)
widenE vs (ECos _ a) = cos (widenE vs a)
widenE vs (ETan _ a) = tan (widenE vs a)
widenE vs (ESinh _ a) = sinh (widenE vs a)
widenE vs (ECosh _ a) = cosh (widenE vs a)
widenE vs (ETanh _ a) = tanh (widenE vs a)
widenE vs (ESqrt _ a) = sqrt (widenE vs a)
widenE vs n@(EPi _) = n

biggenE :: (Eq a, Show a) => [(a, (Expr a, Expr a))] -> Expr a -> Expr a
biggenE _ n@(EInt _) = n
biggenE _ n@(ERat _) = n
biggenE vs (EVar _ v) = case lookup v vs of Just (x, _) -> x ; Nothing -> error $ "biggenE " ++ show v
biggenE vs (ENeg _ a) = negate (biggenE vs a)
biggenE vs (EInv _ a) = recip (biggenE vs a)
biggenE vs (EAdd _ a b) = biggenE vs a + biggenE vs b
biggenE vs (EMul _ a b) = biggenE vs a * biggenE vs b
biggenE vs (EAbs _ a) = abs (biggenE vs a)
biggenE vs (ESin _ a) = sin (biggenE vs a)
biggenE vs (ECos _ a) = cos (biggenE vs a)
biggenE vs (ETan _ a) = tan (biggenE vs a)
biggenE vs (ESinh _ a) = sinh (biggenE vs a)
biggenE vs (ECosh _ a) = cosh (biggenE vs a)
biggenE vs (ETanh _ a) = tanh (biggenE vs a)
biggenE vs (ESqrt _ a) = sqrt (biggenE vs a)
biggenE _ n@(EPi _) = n
```

```
perturbE :: [(String, (Expr String, Expr String))] -> Expr String -> Expr String
perturbE _ (EInt _) = 0
perturbE _ (ERat _) = 0
perturbE vs a@(EVar _ _) = optimize $ widenE vs a - biggenE vs a
perturbE vs (ENeg _ a  ) = optimize $ negate (perturbE vs a)
perturbE vs (EInv _ a  ) = optimize $ negate (perturbE vs a) / (biggenE vs a * widenE vs a)
perturbE vs (EAdd _ a b) = optimize $ perturbE vs a + perturbE vs b
perturbE vs (EMul _ a b) = optimize $ perturbE vs a * widenE vs b + biggenE vs a * perturbE vs b
perturbE vs (EAbs _ a  ) = optimize $ diffabs (biggenE vs a) (perturbE vs a)
perturbE vs (ESin _ a  ) = optimize $  2 * sin (perturbE vs a * 0.5) * cos ((widenE vs a + biggenE vs a) * 0.5)
perturbE vs (ECos _ a  ) = optimize $ -2 * sin (perturbE vs a * 0.5) * sin ((widenE vs a + biggenE vs a) * 0.5)
perturbE vs (ETan _ a  ) = optimize $ sin (perturbE vs a) / (cos (widenE vs a) * cos (biggenE vs a))
perturbE vs (ESinh _ a ) = optimize $ 2 * sinh (perturbE vs a * 0.5) * cosh ((widenE vs a + biggenE vs a) * 0.5)
perturbE vs (ECosh _ a ) = optimize $ 2 * sinh (perturbE vs a * 0.5) * sinh ((widenE vs a + biggenE vs a) * 0.5)
perturbE vs (ETanh _ a ) = optimize $ sinh (perturbE vs a) / (cosh (widenE vs a) * cosh (biggenE vs a))
perturbE vs (ESqrt _ a ) = optimize $ perturbE vs a / (sqrt (widenE vs a) + sqrt (biggenE vs a))
perturbE _ (EPi _) = 0
perturbE _ e = error $ "perturbE " ++ show e

perturb :: [(String, (Expr String, Expr String))] -> Expr String -> Expr String
perturb vs = optimize . perturbE vs . optimize
```

46 src/Fractal/EscapeTime/Compiler/Expr/Pretty.hs

```
{-
et -- escape time fractals
Copyright (C) 2018 Claude Heiland-Allen

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU Affero General Public License as
published by the Free Software Foundation, either version 3 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 Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License
along with this program.  If not, see <https://www.gnu.org/licenses/>.
-}

module Fractal.EscapeTime.Compiler.Expr.Pretty where

import Data.Ratio (numerator, denominator)

import Fractal.EscapeTime.Compiler.Expr.AST
```

```
pretty :: Expr String -> String
pretty (EInt i) = show i
pretty (ERat i) = "(" ++ show (numerator i) ++ "/" ++ show (denominator i) ++ ")"
pretty (EVar _ v) = v
pretty (ENeg _ e) = "-(" ++ pretty e ++ ")"
pretty (EInv _ e) = "1/(" ++ pretty e ++ ")"
pretty (EAdd _ a b) = "(" ++ pretty a ++ "+" ++ pretty b ++ ")"
pretty (EMul _ a b) = "(" ++ pretty a ++ "*" ++ pretty b ++ ")"
pretty (EAbs _ e) = "abs(" ++ pretty e ++ ")"
pretty (ESin _ e) = "sin(" ++ pretty e ++ ")"
pretty (ECos _ e) = "cos(" ++ pretty e ++ ")"
pretty (ETan _ e) = "tan(" ++ pretty e ++ ")"
pretty (ESinh _ e) = "sinh(" ++ pretty e ++ ")"
pretty (ECosh _ e) = "cosh(" ++ pretty e ++ ")"
pretty (ETanh _ e) = "tanh(" ++ pretty e ++ ")"
pretty (EPi _) = "pi"
pretty (EExpM1 _ e) = "expm1(" ++ pretty e ++ ")"
pretty (ELog1P _ e) = "log1p(" ++ pretty e ++ ")"
pretty (EFloat e) = pretty e
pretty (ESgn   e) = "sgn(" ++ pretty e ++ ")"
pretty (ESqrt _ e) = "sqrt(" ++ pretty e ++ ")"
pretty (EASin _ e) = "asin(" ++ pretty e ++ ")"
pretty (EACos _ e) = "acos(" ++ pretty e ++ ")"
pretty (EATan _ e) = "atan(" ++ pretty e ++ ")"
pretty (EASinh _ e) = "asinh(" ++ pretty e ++ ")"
pretty (EACosh _ e) = "acosh(" ++ pretty e ++ ")"
pretty (EATanh _ e) = "atanh(" ++ pretty e ++ ")"
pretty (EFloor _ e) = "floor(" ++ pretty e ++ ")"
pretty (ELess a b) = "(" ++ pretty a ++ "<" ++ pretty b ++ ")"
pretty (ELessEqual a b) = "(" ++ pretty a ++ "<=" ++ pretty b ++ ")"
pretty (EEqual a b) = "(" ++ pretty a ++ "==" ++ pretty b ++ ")"
pretty (EOr a b) = "(" ++ pretty a ++ "||" ++ pretty b ++ ")"
pretty (EAnd a b) = "(" ++ pretty a ++ "&&" ++ pretty b ++ ")"
pretty (ERead a b) = "(" ++ pretty a ++ "[" ++ pretty b ++ "])"
pretty (EDiffAbs _ a b) = "diffabs(" ++ pretty a ++ "," ++ pretty b ++ ")"
pretty (EIf t a b) = "(" ++ pretty t ++ "?" ++ pretty a ++ ":" ++ pretty b ++ ")"
```

47 src/Fractal/EscapeTime/Compiler/Expr/Simplify.hs

```
along with this program.  If not, see <https://www.gnu.org/licenses/>.
-}

module Fractal.EscapeTime.Compiler.Expr.Simplify where

import Data.List (sort)
import Data.Maybe (fromMaybe)
import Data.Ratio (numerator, denominator)
import Numeric
import qualified Data.Map as M

import Fractal.EscapeTime.Compiler.Expr.AST

-- |int| < |rat| < abs < sgn < mul < add < var
cmpMul :: Ord a => Expr a -> Expr a -> Bool
cmpMul (EInt i) (EInt j) = abs i < abs j
cmpMul EInt{} _ = True
cmpMul _ EInt{} = False
cmpMul (ERat i) (ERat j) = abs i < abs j
cmpMul ERat{} _ = True
cmpMul _ ERat{} = False
cmpMul (EAbs _ a) (EAbs _ b) = cmpMul a b
cmpMul EAbs{} _ = True
cmpMul _ EAbs{} = False
cmpMul (ESgn a) (ESgn b) = cmpMul a b
cmpMul ESgn{} _ = True
cmpMul _ ESgn{} = False
cmpMul (EMul _ a b) (EMul _ x y) | a == x = cmpMul b y | otherwise = cmpMul a x
cmpMul EMul{} _ = True
cmpMul _ EMul{} = False
cmpMul (EAdd _ a b) (EAdd _ x y) | a == x = cmpMul b y | otherwise = cmpMul a x
cmpMul EAdd{} _ = True
cmpMul _ EAdd{} = False
cmpMul a@EVar{} b@EVar{} = a < b
cmpMul EVar{} _ = True
cmpMul _ EVar{} = False
cmpMul a b = a < b

-- var < abs < sgn < mul < add < |int| < |rat|
cmpAdd :: Ord a => Expr a -> Expr a -> Bool
cmpAdd a@EVar{} b@EVar{} = a < b
cmpAdd EVar{} _ = True
cmpAdd _ EVar{} = False
cmpAdd (EAbs _ a) (EAbs _ b) = cmpAdd a b
cmpAdd EAbs{} _ = True
cmpAdd _ EAbs{} = False
cmpAdd (ESgn a) (ESgn b) = cmpAdd a b
cmpAdd ESgn{} _ = True
cmpAdd _ ESgn{} = False
cmpAdd (EMul _ a b) (EMul _ x y) | a == x = cmpAdd b y | otherwise = cmpAdd a x
cmpAdd EMul{} _ = True
cmpAdd _ EMul{} = False
cmpAdd (EAdd _ a b) (EAdd _ x y) | a == x = cmpAdd b y | otherwise = cmpAdd a x
cmpAdd EAdd{} _ = True
cmpAdd _ EAdd{} = False
cmpAdd (EInt i) (EInt j) = abs i < abs j
cmpAdd EInt{} _ = True
```

```
cmpAdd _ EInt{} = False
cmpAdd (ERat i) (ERat j) = abs i < abs j
cmpAdd ERat{} _ = True
cmpAdd _ ERat{} = False
cmpAdd a b = a < b


rule :: Ord a => Expr a -> Maybe (Expr a)

-- identity

rule (EAdd _ (EInt 0) b) = Just b
rule (EAdd _ b (EInt 0)) = Just b
rule (EAdd _ (ERat 0) b) = Just b
rule (EAdd _ b (ERat 0)) = Just b

rule (EMul _ (EInt 1) b) = Just b
rule (EMul _ b (EInt 1)) = Just b
rule (EMul _ (ERat 1) b) = Just b
rule (EMul _ b (ERat 1)) = Just b

-- zero

rule (EMul _ (EInt 0) _) = Just 0
rule (EMul _ _ (EInt 0)) = Just 0

-- negate

rule (ENeg t a) = Just $ EMul t (EInt (-1)) a

-- distribute

rule (EMul t a (EAdd _ b c)) = Just $ EAdd t (EMul t a b) (EMul t a c)
rule (EMul t (EAdd _ a b) c) = Just $ EAdd t (EMul t a c) (EMul t b c)

-- combine constants

rule (EMul _ (EInt a) (EInt b)) = Just $ EInt (a * b)
rule (EMul t (EInt a) (EMul _ (EInt b) c)) = Just $ EMul t (EInt (a * b)) c
rule (EMul _ (EInt a) (ERat b)) = Just $ ERat $ fromInteger a * b
rule (EMul t (EInt a) (EMul _ (ERat b) c)) = Just $ EMul t (ERat (fromInteger a ↙
    ↳ * b)) c
rule (EMul _ (ERat a) (ERat b)) = Just $ ERat (a * b)
rule (EMul t (ERat a) (EMul _ (ERat b) c)) = Just $ EMul t (ERat (a * b)) c

-- combine abs*abs

rule (EMul t (EAbs _ a) (EAbs _ b))
  | a == b = Just $ EMul t a b
  | otherwise = Just $ EAbs t (EMul t a b)

rule (EMul t (EAbs _ a) (EMul _ (EAbs _ b) c))
  | a == b = Just $ EMul t (EMul t a b) c
  | otherwise = Just $ EMul t (EAbs t (EMul t a b)) c

-- associate
```

```
  rule (EAdd t (EAdd _ a b) c) = Just $ EAdd t a (EAdd t b c)
  rule (EMul t (EMul _ a b) c) = Just $ EMul t a (EMul t b c)

  -- sort

  rule (EMul t a (EMul _ b c)) | cmpMul b a = Just $ EMul t b (EMul t a c)
  rule (EAdd t a (EAdd _ b c)) | cmpAdd b a = Just $ EAdd t b (EAdd t a c)
  rule (EMul t a (EAbs s b)) = Just $ EMul t (EAbs s b) a

  -- undistribute int

  rule (EAdd _ (EMul _ (EInt a) x) (EAdd _ (EMul _ (EInt b) y) c)) | x == y = Just↙
      ↪  $ fromInteger (a + b) * x + c
  rule (EAdd _ x (EAdd _ (EMul _ (EInt b) y) c)) | x == y = Just $ fromInteger (1 ↙
      ↪ + b) * x + c
  rule (EAdd _ (EMul _ (EInt a) x) (EAdd _ y c)) | x == y = Just $ fromInteger (a ↙
      ↪ + 1) * x + c

  rule (EAdd _ (EMul _ (EInt a) x) (EMul _ (EInt b) y)) | x == y = Just $ ↙
      ↪ fromInteger (a + b) * x
  rule (EAdd _ x (EMul _ (EInt b) y)) | x == y = Just $ fromInteger (1 + b) * x
  rule (EAdd _ (EMul _ (EInt a) x) y) | x == y = Just $ fromInteger (a + 1) * x

  rule (EAdd _ x (EAdd _ y c)) | x == y = Just $ 2 * x + c
  rule (EAdd _ x y) | x == y = Just $ 2 * x

  -- misc

  rule (EAbs _ (EMul _ (EInt a) b)) = Just $ fromInteger (abs a) * abs b
  rule (ESgn (EMul _ a b)) = Just $ signum a * signum b
  rule (ESgn (EInt a)) = Just $ EInt (signum a)
  rule (ESgn (EAbs _ _)) = Just $ EInt 1

  -- functor

  rule (EAdd _ a b) = case (rule a, rule b) of
    (Nothing, Nothing) -> Nothing
    (ma, mb) -> Just $ fromMaybe a ma + fromMaybe b mb
  rule (EMul _ a b) = case (rule a, rule b) of
    (Nothing, Nothing) -> Nothing
    (ma, mb) -> Just $ fromMaybe a ma * fromMaybe b mb
  rule (EDiffAbs _ a b) = case (rule a, rule b) of
    (Nothing, Nothing) -> Nothing
    (ma, mb) -> Just $ diffabs (fromMaybe a ma) (fromMaybe b mb)
  rule (EInv _ a) = recip <$> rule a
  rule (EAbs _ a) = abs <$> rule a
  rule (ESgn   a) = signum <$> rule a
  rule (ESin _ a) = sin <$> rule a
  rule (ECos _ a) = cos <$> rule a
  rule (ETan _ a) = tan <$> rule a
  rule (ESinh _ a) = sinh <$> rule a
  rule (ECosh _ a) = cosh <$> rule a
  rule (ETanh _ a) = tanh <$> rule a
  rule (ESqrt _ a) = sqrt <$> rule a

  -- sin + sin
```

```
rule (EAdd _ (ESin _ a) (ESin _ b)) = Just $ 2 * sin ((a + b) / 2) * cos ((a - b↙
    ↳ ) / 2)
rule (EAdd _ (ESin _ a) (EAdd _ (ESin _ b) c)) = Just $ 2 * sin ((a + b) / 2) * ↙
    ↳ cos ((a - b) / 2) + c

-- cos + cos

rule (EAdd _ (ECos _ a) (ECos _ b)) = Just $ 2 * cos ((a + b) / 2) * cos ((a - b↙
    ↳ ) / 2)
rule (EAdd _ (ECos _ a) (EAdd _ (ECos _ b) c)) = Just $ 2 * cos ((a + b) / 2) * ↙
    ↳ cos ((a - b) / 2) + c

-- tan + tan

rule (EAdd _ (ETan _ a) (ETan _ b)) = Just $ sin (a + b) / (cos a * cos b)
rule (EAdd _ (ETan _ a) (EAdd _ (ETan _ b) c)) = Just $ sin (a + b) / (cos a * ↙
    ↳ cos b) + c

-- sinh + sinh

rule (EAdd _ (ESinh _ a) (ESinh _ b)) = Just $ 2 * sinh ((a + b) / 2) * cosh ((a↙
    ↳  - b) / 2)
rule (EAdd _ (ESinh _ a) (EAdd _ (ESinh _ b) c)) = Just $ 2 * sinh ((a + b) / 2)↙
    ↳  * cosh ((a - b) / 2) + c

-- cosh + cosh

rule (EAdd _ (ECosh _ a) (ECosh _ b)) = Just $ 2 * cosh ((a + b) / 2) * cosh ((a↙
    ↳  - b) / 2)
rule (EAdd _ (ECosh _ a) (EAdd _ (ECosh _ b) c)) = Just $ 2 * cosh ((a + b) / 2)↙
    ↳  * cosh ((a - b) / 2) + c

-- tanh + tanh

rule (EAdd _ (ETanh _ a) (ETanh _ b)) = Just $ sinh (a + b) / (cosh a * cosh b)
rule (EAdd _ (ETanh _ a) (EAdd _ (ETanh _ b) c)) = Just $ sinh (a + b) / (cosh a↙
    ↳  * cosh b) + c

-- irreducible

rule _ = Nothing

rule1 :: Expr a -> Expr a

rule1 n@(EInt _) = n
rule1 n@(ERat _) = n
rule1 v@(EVar _ _) = 1 * v
rule1 (ENeg _ a) = negate (rule1 a)
rule1 (EInv _ a) = recip (rule1 a)
rule1 (EAdd _ a b) = rule1 a + rule1 b
rule1 (EMul _ a b) = rule1 a * rule1 b
rule1 (EAbs _ a) = abs (rule1 a)
rule1 (ESin _ a) = sin (rule1 a)
rule1 (ECos _ a) = cos (rule1 a)
rule1 (ETan _ a) = tan (rule1 a)
rule1 (ESinh _ a) = sinh (rule1 a)
rule1 (ECosh _ a) = cosh (rule1 a)
```

```
    rule1 (ETanh _ a) = tanh (rule1 a)
230 rule1 n@(EPi _) = n
    rule1 (EExpM1 _ a) = expm1 (rule1 a)
    rule1 (ELog1P _ a) = log1p (rule1 a)
    rule1 (EFloat a) = EFloat (rule1 a)
    rule1 (ESgn   a) = signum (rule1 a)
235 rule1 (ESqrt _ a) = sqrt (rule1 a)
    rule1 (EASin _ a) = asin (rule1 a)
    rule1 (EACos _ a) = acos (rule1 a)
    rule1 (EATan _ a) = atan (rule1 a)
    rule1 (EASinh _ a) = asinh (rule1 a)
240 rule1 (EACosh _ a) = acosh (rule1 a)
    rule1 (EATanh _ a) = atanh (rule1 a)
    rule1 (EFloor t a) = EFloor t (rule1 a)
    rule1 (ELess a b) = rule1 a `ELess` rule1 b
    rule1 (ELessEqual a b) = rule1 a `ELessEqual` rule1 b
245 rule1 (EEqual a b) = rule1 a `EEqual` rule1 b
    rule1 (EOr a b) = rule1 a `EOr` rule1 b
    rule1 (EAnd a b) = rule1 a `EAnd` rule1 b
    rule1 (ERead a b) = a `ERead` rule1 b
    rule1 (EDiffAbs _ a b) = 1 * diffabs (rule1 a) (rule1 b)
250 rule1 (EIf t a b) = EIf (rule1 t) (rule1 a) (rule1 b)

    simplify :: Ord a => Expr a -> Expr a
    simplify e = case rule e of
      Nothing -> e
255   Just f -> simplify f

    type SAtoms t = SSum (SProduct (SAtom t))
    data SSum t = SSum{ getSSum :: [t] } deriving (Eq, Ord)
    data SProduct t = SProduct{ getSProduct :: [t] } deriving (Eq, Ord)
260 data SAtom t
      = SRat    Rational
      | SPi     NType
      | SVar    NType t
      | SSgn          (SAtoms t)
265   | SInv    NType (SAtoms t)
      | SAbs    NType (SAtoms t)
      | SSin    NType (SAtoms t)
      | SCos    NType (SAtoms t)
      | STan    NType (SAtoms t)
270   | SSinh   NType (SAtoms t)
      | SCosh   NType (SAtoms t)
      | STanh   NType (SAtoms t)
      | SExpM1 NType (SAtoms t)
      | SLog1P NType (SAtoms t)
275   | SFloat        (SAtoms t)
      | SSqrt   NType (SAtoms t)
      | SASin   NType (SAtoms t)
      | SACos   NType (SAtoms t)
      | SATan   NType (SAtoms t)
280   | SASinh NType (SAtoms t)
      | SACosh NType (SAtoms t)
      | SATanh NType (SAtoms t)
      | SDiffAbs NType (SAtoms t) (SAtoms t)
      | SRead t Integer
285   deriving (Eq, Ord)
```

```
    atoms :: Show t => Expr t -> SAtoms t
    atoms (EInt      n) = SSum [SProduct [SRat (fromInteger n)]]
    atoms (ERat      n) = SSum [SProduct [SRat n]]
290 atoms (EVar    t v) = SSum [SProduct [SRat 1, SVar    t v]]
    atoms (EPi     t  ) = SSum [SProduct [SRat 1, SPi     t  ]]
    atoms (ENeg    _ a) = SSum [SProduct (SRat (-1) : getSProduct ax) | ax <- getSSum↙
        ↳ (atoms a)]
    atoms (EInv    t a) = SSum [SProduct [SRat 1, SInv    t (atoms a)]]
    atoms (ESgn      a) = SSum [SProduct [SRat 1, SSgn      (atoms a)]]
295 atoms (EAbs    t a) = SSum [SProduct [SRat 1, SAbs    t (atoms a)]]
    atoms (ESin    t a) = SSum [SProduct [SRat 1, SSin    t (atoms a)]]
    atoms (ECos    t a) = SSum [SProduct [SRat 1, SCos    t (atoms a)]]
    atoms (ETan    t a) = SSum [SProduct [SRat 1, STan    t (atoms a)]]
    atoms (ESinh   t a) = SSum [SProduct [SRat 1, SSinh   t (atoms a)]]
300 atoms (ECosh   t a) = SSum [SProduct [SRat 1, SCosh   t (atoms a)]]
    atoms (ETanh   t a) = SSum [SProduct [SRat 1, STanh   t (atoms a)]]
    atoms (EExpM1  t a) = SSum [SProduct [SRat 1, SExpM1  t (atoms a)]]
    atoms (ELog1P  t a) = SSum [SProduct [SRat 1, SLog1P  t (atoms a)]]
    atoms (EFloat    a) = SSum [SProduct [SRat 1, SFloat    (atoms a)]]
305 atoms (ESqrt   t a) = SSum [SProduct [SRat 1, SSqrt   t (atoms a)]]
    atoms (EASin   t a) = SSum [SProduct [SRat 1, SASin   t (atoms a)]]
    atoms (EACos   t a) = SSum [SProduct [SRat 1, SACos   t (atoms a)]]
    atoms (EATan   t a) = SSum [SProduct [SRat 1, SATan   t (atoms a)]]
    atoms (EASinh  t a) = SSum [SProduct [SRat 1, SASinh  t (atoms a)]]
310 atoms (EACosh  t a) = SSum [SProduct [SRat 1, SACosh  t (atoms a)]]
    atoms (EATanh  t a) = SSum [SProduct [SRat 1, SATanh  t (atoms a)]]
    atoms (EDiffAbs t a b) = SSum [SProduct [SRat 1, SDiffAbs t (atoms a) (atoms b)↙
        ↳ ]]
    atoms (EAdd _ a b) = SSum (getSSum (atoms a) ++ getSSum (atoms b))
    atoms (EMul _ a b) = SSum [SProduct $ [SRat 1] ++ getSProduct ax ++ getSProduct ↙
        ↳ bx | ax <- getSSum (atoms a), bx <- getSSum (atoms b) ]
315 atoms (ERead (EVar NFloatPtr p) (EInt n)) = SSum [SProduct [SRead p n]]

    orderS :: Ord t => SAtoms t -> SAtoms t
    orderS = SSum . sort . map orderP . getSSum
    orderP :: Ord t => SProduct (SAtom t) -> SProduct (SAtom t)
320 orderP = SProduct . sort . map order . getSProduct
    order :: Ord t => SAtom t -> SAtom t
    order n@(SRat _) = n
    order n@(SPi _) = n
    order n@(SVar _ _) = n
325 order (SSgn t) = SSgn $ orderS t
    order (SInv    s t) = SInv    s $ orderS t
    order (SAbs    s t) = SAbs    s $ orderS t
    order (SSin    s t) = SSin    s $ orderS t
    order (SCos    s t) = SCos    s $ orderS t
330 order (STan    s t) = STan    s $ orderS t
    order (SSinh   s t) = SSinh   s $ orderS t
    order (SCosh   s t) = SCosh   s $ orderS t
    order (STanh   s t) = STanh   s $ orderS t
    order (SExpM1  s t) = SExpM1  s $ orderS t
335 order (SLog1P  s t) = SLog1P  s $ orderS t
    order (SFloat    t) = SFloat    $ orderS t
    order (SSqrt   s t) = SSqrt   s $ orderS t
    order (SASin   s t) = SASin   s $ orderS t
    order (SACos   s t) = SACos   s $ orderS t
```

```
340 order (SATan  s t) = SATan  s $ orderS t
    order (SASinh s t) = SASinh s $ orderS t
    order (SACosh s t) = SACosh s $ orderS t
    order (SATanh s t) = SATanh s $ orderS t
    order (SDiffAbs s a b) = SDiffAbs s (orderS a) (orderS b)
345 order n@(SRead _ _) = n

    fromS :: SAtoms t -> Expr t
    fromS = sum . map fromP . getSSum
    fromP :: SProduct (SAtom t) -> Expr t
350 fromP = product . map from . getSProduct
    from :: SAtom t -> Expr t
    from (SRat n)
      | denominator n == 1 = EInt (numerator n)
      | otherwise = ERat n
355 from (SPi t) = EPi t
    from (SInv  s t) = EInv  s $ fromS t
    from (SVar t v) = EVar t v
    from (SSgn t) = ESgn $ fromS t
    from (SAbs  s t) = EAbs  s $ fromS t
360 from (SSin  s t) = ESin  s $ fromS t
    from (SCos  s t) = ECos  s $ fromS t
    from (STan  s t) = ETan  s $ fromS t
    from (SSinh s t) = ESinh s $ fromS t
    from (SCosh s t) = ECosh s $ fromS t
365 from (STanh s t) = ETanh s $ fromS t
    from (SExpM1 s t) = EExpM1 s $ fromS t
    from (SLog1P s t) = ELog1P s $ fromS t
    from (SFloat  t) = EFloat  $ fromS t
    from (SSqrt s t) = ESqrt s $ fromS t
370 from (SASin  s t) = EASin  s $ fromS t
    from (SACos  s t) = EACos  s $ fromS t
    from (SATan  s t) = EATan  s $ fromS t
    from (SASinh s t) = EASinh s $ fromS t
    from (SACosh s t) = EACosh s $ fromS t
375 from (SATanh s t) = EATanh s $ fromS t
    from (SDiffAbs s a b) = EDiffAbs s (fromS a) (fromS b)
    from (SRead p n) = ERead (EVar NFloatPtr p) (EInt n)

    collapseS :: Ord t => SAtoms t -> SAtoms t
380 collapseS (SSum bs) = SSum
      [ SProduct (SRat v : qs)
      | (qs, v) <- filter ((/= 0) . snd) . M.toList . M.fromListWith (+) $
          [ (ps, u) | x <- map collapseP bs, let SProduct (SRat u : ps) = x ]
      ]
385 collapseP :: Ord t => SProduct (SAtom t) -> SProduct (SAtom t)
    collapseP (SProduct (SRat a : SRat b : cs)) = collapseP (SProduct (SRat (a * b) ↙
      ↳ : cs))
    collapseP (SProduct (SRat a : cs)) = SProduct (case getSProduct (collapseP (↙
      ↳ SProduct cs)) of (SRat b : ds) ->  SRat (a * b) : ds)
    collapseP (SProduct cs) = SProduct (SRat 1 : map collapse cs)
    collapse :: Ord t => SAtom t -> SAtom t
390 collapse (SRat n) = SRat n
    collapse (SPi t) = SPi t
    collapse (SInv  s t) = SInv  s $ collapseS t
    collapse (SVar t v) = SVar t v
    collapse (SSgn t) = SSgn $ collapseS t
```

```
collapse (SAbs   s t) = SAbs   s $ collapseS t
collapse (SSin   s t) = SSin   s $ collapseS t
collapse (SCos   s t) = SCos   s $ collapseS t
collapse (STan   s t) = STan   s $ collapseS t
collapse (SSinh  s t) = SSinh  s $ collapseS t
collapse (SCosh  s t) = SCosh  s $ collapseS t
collapse (STanh  s t) = STanh  s $ collapseS t
collapse (SExpM1 s t) = SExpM1 s $ collapseS t
collapse (SLog1P s t) = SLog1P s $ collapseS t
collapse (SFloat   t) = SFloat   $ collapseS t
collapse (SSqrt  s t) = SSqrt  s $ collapseS t
collapse (SASin  s t) = SASin  s $ collapseS t
collapse (SACos  s t) = SACos  s $ collapseS t
collapse (SATan  s t) = SATan  s $ collapseS t
collapse (SASinh s t) = SASinh s $ collapseS t
collapse (SACosh s t) = SACosh s $ collapseS t
collapse (SATanh s t) = SATanh s $ collapseS t
collapse (SDiffAbs s a b) = SDiffAbs s (collapseS a) (collapseS b)
collapse (SRead p n) = SRead p n

optimize :: Expr String -> Expr String
optimize = simplify . rule1 . fromS . collapseS . orderS . atoms . simplify . ↙
    ↳ rule1

{-
undistribute :: Expr String -> Maybe (Expr String)
undistribute (EAdd _ (EMul _ x1 y1) (EMul _ x2 y2)) | x1 == x2 = Just $ x1 * (y1↙
    ↳ + y2)
undistribute (EAdd _ (EMul _ y1 x1) (EMul _ x2 y2)) | x1 == x2 = Just $ x1 * (y1↙
    ↳ + y2)
undistribute (EAdd _ (EMul _ x1 y1) (EMul _ y2 x2)) | x1 == x2 = Just $ x1 * (y1↙
    ↳ + y2)
undistribute (EAdd _ (EMul _ y1 x1) (EMul _ y2 x2)) | x1 == x2 = Just $ x1 * (y1↙
    ↳ + y2)
undistribute (EAdd _ (EDiv _ y1 x1) (EDiv _ y2 x2)) | x1 == x2 = Just $ (y1 + y2↙
    ↳ ) / x1
undistribute _ = Nothing
-}
```

48 src/Fractal/EscapeTime/Compiler.hs

```
{-
et -- escape time fractals
Copyright (C) 2018 Claude Heiland-Allen

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU Affero General Public License as
published by the Free Software Foundation, either version 3 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 Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License
along with this program.  If not, see <https://www.gnu.org/licenses/>.
```

```
-}

module Fractal.EscapeTime.Compiler
  ( module Fractal.EscapeTime.Compiler.CodeGen
  , module Fractal.EscapeTime.Compiler.Expr
  ) where

import Fractal.EscapeTime.Compiler.CodeGen
import Fractal.EscapeTime.Compiler.Expr
```

49 src/Fractal/EscapeTime/Compiler/Parser.hs

```
{-
et -- escape time fractals
Copyright (C) 2018 Claude Heiland-Allen

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU Affero General Public License as
published by the Free Software Foundation, either version 3 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 Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License
along with this program.  If not, see <https://www.gnu.org/licenses/>.
-}

{-# LANGUAGE FlexibleContexts #-}

module Fractal.EscapeTime.Compiler.Parser (Formula, formulas) where

import Data.Char (isSpace)
import Data.Monoid ((<>))
import Data.Set (Set)
import qualified Data.Set as S

import Text.Parsec (parse, ParseError, (<|>), try, choice, many)
import Text.Parsec.Char (string)
import Text.Parsec.Combinator (chainl1)

import Fractal.EscapeTime.Formulas.Types (R2(..))
import Fractal.EscapeTime.Compiler.Expr.AST hiding (interpret)

type Formula a = Int -> Int -> [R2 a -> R2 a -> R2 a -> R2 a]

data Assignment = LetW PExpr | SetZ PExpr

interpretA :: (Eq a, Floating a) => [Assignment] -> Either String (Set String, ↙
    ↳ Formula a)
interpretA [] = return (S.empty, \p q -> [])
interpretA (LetW e : SetZ f : gs) = do
  _ <- letW e 1 1 (sqrt 2 `R2` sqrt 3) (sqrt 5 `R2` sqrt 7) (sqrt 31 `R2` sqrt ↙
      ↳ 37) $ setZ f 1 1 (sqrt 11 `R2` sqrt 13) (sqrt 17 `R2` sqrt 19) (sqrt 23 ↙
      ↳ `R2` sqrt 29)
```

```
        -- by parametricity, if this succeeds so will the case below
  (g', g) <- interpretA gs
  return (g' <> variables e <> variables f, \p q -> (\a c z -> case letW e p q a c z $ setZ f p q a c z of Right z' -> z') : g p q)
interpretA (SetZ f : gs) = do
  _ <- setZ f 1 1 (sqrt 2 `R2` sqrt 3) (sqrt 5 `R2` sqrt 7) (sqrt 11 `R2` sqrt 13) (sqrt 17 `R2` sqrt 19)
        -- by parametricity, if this succeeds so will the case below
  (g', g) <- interpretA gs
  return (g' <> variables f, \p q -> (\a c z -> case let w = 0 in setZ f p q a c z w of Right z' -> z') : g p q)

letW :: (Eq a, Floating a) => PExpr -> Int -> Int -> R2 a -> R2 a -> R2 a -> (R2 a -> Either String (R2 a)) -> Either String (R2 a)
letW ex p q (R2 d e) (R2 a b) (R2 x y) f
  = f =<< interpret
            [ ("p", fromIntegral p)
            , ("q", fromIntegral q)
            , ("i", R2 0 1)
            , ("a", R2 a 0)
            , ("b", R2 b 0)
            , ("c", R2 a b)
            , ("d", R2 d 0)
            , ("e", R2 e 0)
            , ("f", R2 d e)
            , ("x", R2 x 0)
            , ("y", R2 y 0)
            , ("z", R2 x y)
            ] ex

setZ :: (Eq a, Floating a) => PExpr -> Int -> Int -> R2 a -> R2 a -> R2 a -> R2 a -> Either String (R2 a)
setZ ex p q (R2 d e) (R2 a b) (R2 x y) (R2 u v)
  = interpret
            [ ("p", fromIntegral p)
            , ("q", fromIntegral q)
            , ("i", R2 0 1)
            , ("a", R2 a 0)
            , ("b", R2 b 0)
            , ("c", R2 a b)
            , ("d", R2 d 0)
            , ("e", R2 e 0)
            , ("f", R2 d e)
            , ("x", R2 x 0)
            , ("y", R2 y 0)
            , ("z", R2 x y)
            , ("u", R2 u 0)
            , ("v", R2 v 0)
            , ("w", R2 u v)
            ] ex

formulas :: (Eq a, Floating a) => String -> Either String [(String, Set String, Formula a)]
formulas s = sequence
  [ (\r -> case r of Left e -> Left (name ++ ":" ++ show e) ; Right (vs, x) -> Right (name, vs, x))
  . (\r -> case r of Left e -> Left (show e) ; Right x -> interpretA x)
```

```
    . sequence
    . map (uncurry assignment)
    . zip [1..]
    . map (filter (not . isSpace))
    $ as
    | name : as <-
      paras
    . map stripComment
    . lines
    $ s
    ]

stripComment :: String -> String
stripComment = takeWhile ('#' /=)

paras :: [String] -> [[String]]
paras s = case dropWhile null s of
  [] -> []
  s' -> p : paras s'' where ~(p, s'') = break null s'

assignment :: Int -> String -> Either ParseError Assignment
assignment n s = case s of
  'w':':':'=':t -> LetW <$> parse expr (show n) t
  'z':':':'=':t -> SetZ <$> parse expr (show n) t
  _ -> parse (fail $ "unexpected: " ++ s) (show n) s

expr   = do{ string "-"; PNeg <$> factor }
    <|>  do{ a <- factor;
             do { string "+"; b <- expr; return $ PAdd a b } <|>
             do { string "-"; b <- expr; return $ PAdd a (PNeg b) } <|>
             return a }

factor = do{ a <- base; try (do{ string "/"; b <- base ; return $ PMul a (PInv b↙
    ↳ ) })
                    <|> try (do{ b <- factor; return $ PMul a b })
                    <|> return a }

base   = do{ a <- sub; do{ string "^"; b <- sub; return $ PPow a b } <|> return ↙
    ↳ a }

sub    = do{ string "("; a <- expr; string ")"; return a }
    <|>  do{ string "|"; a <- expr; string "|"; return $ PFunc "abs" a }
    <|>  do{ f <- function ; string "("; a <- expr ; string ")" ; return (f a) }
    <|>  variable
    <|>  number

-- note: no function name should be constructible from any variable names

number = do { n <- digit ; ns <- many ((string "0" >> return '0') <|> digit) ; ↙
    ↳ return $ PInt (read (n : ns)) }
  where digit = choice [string [d] >> return d | d <- "123456789"]

variable = choice $ [var "pi"] ++ [var [x] | x <- "pqiabcdefuvwxyz"]
  where var s = try (string s >> return (PVar s))

function = choice . map func . words $ "abs sqrt exp log sin cos tan sinh cosh ↙
    ↳ tanh asin acos atan asinh acosh atanh"
```

```
  where func s = try (string s >> return (PFunc s))

data PExpr
  = PVar String
  | PInt Integer
  | PNeg PExpr
  | PInv PExpr
  | PAdd PExpr PExpr
  | PMul PExpr PExpr
  | PPow PExpr PExpr
  | PFunc String PExpr
  deriving Show

variables :: PExpr -> Set String
variables (PVar s) = S.singleton s
variables (PInt _) = S.empty
variables (PNeg e) = variables e
variables (PInv e) = variables e
variables (PAdd e f) = variables e <> variables f
variables (PMul e f) = variables e <> variables f
variables (PPow e f) = variables e <> variables f
variables (PFunc _ e) = variables e

class Pow t where pow :: t -> t -> t
instance Pow Double where pow = (**)
instance (Eq a, Fractional a) => Pow (R2 a) where
  pow z n = case [ i | i <- [-100..100], fromInteger i == n ] of -- FIXME
    [] -> 0
    (j:_) -> z ^^ j

interpret :: (Pow t, Floating t) => [(String, t)] -> PExpr -> Either String t
-- interpret variables
interpret vs (PVar "pi") = Right pi
interpret vs (PVar u) = maybe (Left u) Right (lookup u vs)
interpret vs (PInt n) = Right (fromInteger n)
-- applicative traversal
interpret vs (PNeg e) = negate <$> interpret vs e
interpret vs (PInv e) = recip <$> interpret vs e
interpret vs (PAdd e f) = (+) <$> interpret vs e <*> interpret vs f
interpret vs (PMul e f) = (*) <$> interpret vs e <*> interpret vs f
interpret vs (PPow e f) = pow <$> interpret vs e <*> interpret vs f
interpret vs (PFunc "abs" e) = abs <$> interpret vs e
interpret vs (PFunc "sqrt" e) = sqrt <$> interpret vs e
interpret vs (PFunc "exp" e) = exp <$> interpret vs e
interpret vs (PFunc "log" e) = log <$> interpret vs e
interpret vs (PFunc "sin" e) = sin <$> interpret vs e
interpret vs (PFunc "cos" e) = cos <$> interpret vs e
interpret vs (PFunc "tan" e) = tan <$> interpret vs e
interpret vs (PFunc "sinh" e) = sinh <$> interpret vs e
interpret vs (PFunc "cosh" e) = cosh <$> interpret vs e
interpret vs (PFunc "tanh" e) = tanh <$> interpret vs e
interpret vs (PFunc "asin" e) = asin <$> interpret vs e
interpret vs (PFunc "acos" e) = acos <$> interpret vs e
interpret vs (PFunc "atan" e) = atan <$> interpret vs e
interpret vs (PFunc "asinh" e) = asinh <$> interpret vs e
interpret vs (PFunc "acosh" e) = acosh <$> interpret vs e
interpret vs (PFunc "atanh" e) = atanh <$> interpret vs e
```

```
interpret vs e = Left (show e)
```

50 src/Fractal/EscapeTime/Compiler/R2.hs

```
{-
et -- escape time fractals
Copyright (C) 2018,2019 Claude Heiland-Allen

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU Affero General Public License as
published by the Free Software Foundation, either version 3 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 Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License
along with this program.  If not, see <https://www.gnu.org/licenses/>.
-}

module Fractal.EscapeTime.Compiler.R2 (compile) where

import Fractal.EscapeTime.Compiler.Expr.AST ( Expr() )
import Fractal.EscapeTime.Formulas.Types ( R2Formula )
import Fractal.EscapeTime.Algorithms.R2

compile :: [(Int, R2Formula (Expr String))] -> String
compile fs =
  let header = "#include \"formula.h\"\n"
      plai = plain fs
      refe = reference fs
      pert = perturbation fs
      per1 = period_tri fs
      per2 = period_jsk fs
      newt = newton fs
      misi = misiurewicz fs
      siz' = size fs
      ske' = skew fs
      dsiz = domain_size fs
      degr = exp $ sum (map (log . fromIntegral . fst) fs) / fromIntegral (↙
          ↳ length fs) :: Double
      name = "(unknown)"
      source = "(unknown)"
      footer = "FORMULA(" ++ show name ++ "," ++ show source ++ "," ++ show (if ↙
          ↳ isNaN degr || isInfinite degr then 0 else degr) ++ ")\n"
  in  header ++ plai ++ refe ++ pert ++ per1 ++ per2 ++ newt ++ misi ++ siz' ++ ↙
      ↳ ske' ++ dsiz ++ footer
```

51 src/Fractal/EscapeTime/Formulas.hs

```
{-
et -- escape time fractals
Copyright (C) 2018 Claude Heiland-Allen
```

```
    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU Affero General Public License as
    published by the Free Software Foundation, either version 3 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 Affero General Public License for more details.

    You should have received a copy of the GNU Affero General Public License
    along with this program.  If not, see <https://www.gnu.org/licenses/>.
    -}

    module Fractal.EscapeTime.Formulas
      ( module Fractal.EscapeTime.Formulas.Types
      ) where

    import Fractal.EscapeTime.Formulas.Types
```

52 src/Fractal/EscapeTime/Formulas/Types.hs

```
    {-
    et -- escape time fractals
    Copyright (C) 2018 Claude Heiland-Allen

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU Affero General Public License as
    published by the Free Software Foundation, either version 3 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 Affero General Public License for more details.

    You should have received a copy of the GNU Affero General Public License
    along with this program.  If not, see <https://www.gnu.org/licenses/>.
    -}

    module Fractal.EscapeTime.Formulas.Types where

    data R2 a = R2 a a deriving (Read, Show, Eq, Ord)

    instance (Eq a, Num a) => Num (R2 a) where
      fromInteger n = R2 (fromInteger n) 0
      R2 a b + R2 x y = R2 (a + x) (b + y)
      R2 a b - R2 x y = R2 (a - x) (b - y)
      R2 a b * R2 x y = R2 (a * x - b * y) (a * y + b * x)
      negate (R2 a b) = R2 (negate a) (negate b)
      abs (R2 x _) = R2 (abs x) 0
    --  abs _ = error "Fractal.EscapeTime.Formulas.Types.R2.Num.abs: not real"
      signum (R2 x _) = R2 (signum x) 0
    --  signum _ = error "Fractal.EscapeTime.Formulas.Types.R2.Num.signum: not real"

    instance (Eq a, Fractional a) => Fractional (R2 a) where
      fromRational n = R2 (fromRational n) 0
```

```
  c / z = let R2 u v = c * conj z ; w = norm z in R2 (u / w) (v / w)

instance (Eq a, Floating a) => Floating (R2 a) where
  pi = R2 pi 0
  exp (R2 a b) = let r = exp a ; c = cos b ; s = sin b in R2 (r * c) (r * s)
  log z = R2 (log (norm z) / 2) (arg z)

  sqrt z@(R2 x y) = R2 (sqrt $ (magz + x) / 2) (signum' y * (sqrt $ (magz - x) /↙
      ↳ 2))
    where magz = mag z ; signum' 0 = 1 ; signum' w = signum w

  sin (R2 x y) = R2 (sin x * cosh y) ( cos x * sinh y)
  cos (R2 x y) = R2 (cos x * cosh y) (-sin x * sinh y)

  sinh (R2 x y) = R2 (sinh x * cos y) (cosh x * sin y)
  cosh (R2 x y) = R2 (cosh x * cos y) (sinh x * sin y)

  asin z = -i * log (i * z + sqrt (1 - z * z))
  acos z = -i * log (z + i * sqrt (1 - z * z))
  atan z = (log (1 + i * z) - log (1 - i * z)) / (2 * i)

  asinh z = log (z + sqrt (z * z + 1))
  acosh z = 2 * log (sqrt ((z + 1)/2) + sqrt ((z - 1)/2))
  atanh z = log ((1 + z) / (1 - z)) / 2

  -- FIXME expm1, log1p, ...

i :: Num a => R2 a
i = R2 0 1

conj :: Num a => R2 a -> R2 a
conj (R2 a b) = R2 a (negate b)

norm :: Num a => R2 a -> a
norm (R2 x y) = x * x + y * y

mag :: Floating a => R2 a -> a
mag = sqrt . norm

arg :: (Eq a, Floating a) => R2 a -> a
arg 0 = 0
arg (R2 x 0) = (pi - signum x * pi) / 2
arg (R2 x y) = 2 * atan (y / (sqrt (x * x + y * y) + x))

type R2Formula a = R2 a -> R2 a -> R2 a -> R2 a
```

53 src/Fractal/EscapeTime/Prelude.hs

```
{-
et -- escape time fractals
Copyright (C) 2018,2019 Claude Heiland-Allen

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU Affero General Public License as
published by the Free Software Foundation, either version 3 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 Affero General Public License for more details.

    You should have received a copy of the GNU Affero General Public License
    along with this program.  If not, see <https://www.gnu.org/licenses/>.
    -}

{-# LANGUAGE DataKinds #-}
{-# LANGUAGE RankNTypes #-}

module Fractal.EscapeTime.Prelude
  ( module Fractal.EscapeTime.Prelude
  , module Fractal.EscapeTime.Compiler.Expr
  , module Fractal.EscapeTime.Formulas.Types
  , module Fractal.EscapeTime.Runtime.Compiler
  , module Fractal.EscapeTime.Runtime.Loader
  ) where

import Fractal.EscapeTime.Compiler.Expr
import Fractal.EscapeTime.Formulas.Types
import Fractal.EscapeTime.Runtime.Compiler
import Fractal.EscapeTime.Runtime.Loader hiding (readLib)

import qualified Numeric.Rounded as R
import Numeric.Rounded.Simple hiding (simplify)

import Data.Char (toUpper)

[a,b,d,e,x,y,aa,bb,dd,ee,xx,yy] = map (var.(:[])) "abdexyABDEXY"
[daa,dab,dba,dbb,dxa,dxb,dya,dyb,dxx,dxy,dyx,dyy] = map var $ words "daa dab dba
    ↪ dbb dxa dxb dya dyb dxx dxy dyx dyy"
c = R2 a b
f = R2 d e
z = R2 x y
cc = R2 aa bb
ff = R2 dd ee
zz = R2 xx yy
var = EVar NFloat
vars = map vp "abxy" ++ map v0 "ef"
  where
    vp c = ([c], (EVar NFloat [toUpper c], EVar NFloat [c]))
    v0 c = ([c], (EVar NFloat [toUpper c], EInt 0))
dby [v] [u] | u `elem` "abxy" && v `elem` "abxy" = var ['d',u,v]
dby v u = EInt $ if v == u then 1 else 0

diff by = ddx d (EVar NFloat by)
  where
    d (EVar _ v) (EVar _ u) = dby v u
    d v u = if v == u then 1 else 0

toR :: Double -> Rounded
toR = fromDouble R.TowardNearest 53

fromR :: Rounded -> Double
fromR r = reifyRounded r (R.toDouble :: forall p . R.Precision p => R.Rounded 'R
```

```
    ↳ .TowardNearest p -> Double)

{-
      tmp <- getCanonicalTemporaryDirectory
      dir <- createTempDirectory tmp "et"

      r <- compileAndLoad dir ("Burning Ship\n  z := (|x| + i |y|)^p + c"]) 2 1
      case r of
        Left msg -> hPutStrLn stderr msg
        Right et -> do
          (c, s, t) <- do
            Just (R2 a b) <- with 1 $ newton et 50 3 0 0 (toR 1) (toR (-2))
            Just (s, t) <- with 1 $ size bs 3 0 0 a b
            pure (R2 (fromR a) (fromR b), fromR s, t)
-}
```

54 src/Fractal/EscapeTime/Runtime/Compiler.hs

```
{-
et -- escape time fractals
Copyright (C) 2018,2019 Claude Heiland-Allen

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU Affero General Public License as
published by the Free Software Foundation, either version 3 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 Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License
along with this program.  If not, see <https://www.gnu.org/licenses/>.
-}

module Fractal.EscapeTime.Runtime.Compiler (compileAndLoad) where

import Prelude
import Control.Exception (catch, SomeException)
import System.Exit (ExitCode(..))
import Data.Text.Lazy (pack)
import Data.Text.Lazy.Encoding (encodeUtf8)
import Data.Digest.Pure.MD5 (md5)
import System.FilePath ((</>))
import System.Process (readProcessWithExitCode)

import Fractal.EscapeTime.Compiler.Expr.AST (Expr(EVar), NType(NFloat))
import Fractal.EscapeTime.Compiler.Parser as R2
import qualified Fractal.EscapeTime.Compiler.R2 as R2
import Fractal.EscapeTime.Formulas.Types
import Fractal.EscapeTime.Runtime.Loader (ET, readLib)
import Fractal.EscapeTime.Runtime.Loader.Load (load, libext)
import Paths_et (getDataDir)

catchAny :: IO a -> (SomeException -> IO a) -> IO a
catchAny = Control.Exception.catch
```

```
40
libraryName :: Show a => FilePath -> a -> FilePath
libraryName dir = (dir </>) . (++ ("-et" ++ libext)) . show . md5 . encodeUtf8 .↙
    ↳  pack . show

invokeGCC :: String -> FilePath -> IO (ExitCode, String, String)
45 invokeGCC source target = do
  include <- getDataDir
  writeFile "/tmp/et.c" source
  readProcessWithExitCode "gcc"
    [ "-std=c99", "-Wall", "-Wextra", "-pedantic"
50  , "-fPIC", "-O3", "-march=native", "-shared"
    , "-I" ++ include
    , "-o", target
    , "-xc", "-"
    , "-lmpfr"
55  ] source

compileAndLoad :: FilePath -> String -> Int -> Int -> IO (Either String ET)
compileAndLoad dir input p q = catchAny (case R2.formulas input of
  Left e -> return (Left e)
60 Right [] -> return (Left "no formula defined")
  Right (_:_:_) -> return (Left "more than one formula defined")
  Right [(_,_,f)] -> do
    let fpq = f p q
        target = libraryName dir [ fpqs (R2 (EVar NFloat "d") (EVar NFloat "e"))↙
            ↳  (R2 (EVar NFloat "a") (EVar NFloat "b")) (R2 (EVar NFloat "x") (↙
            ↳ EVar NFloat "y")) | fpqs <- fpq ]
65  r <- load target
    case r of
      Right p -> readLib p
      Left _ -> do
        let source = R2.compile $ map ((,) (p * q)) fpq -- FIXME compute degree
70      (exitCode, stdout, stderr) <- invokeGCC source target
        case exitCode of
          ExitSuccess -> do
            s <- load target
            case s of
75            Right p -> readLib p
              Left e -> return (Left e)
          ExitFailure n -> return (Left $ "compile failed with return code " ++ ↙
            ↳ show n ++ "\nstdout:\n" ++ stdout ++ "\nstderr:\n" ++ stderr ++ ↙
            ↳ "\n")) $ \e -> return (Left ("exception: " ++ show e))
```

55 src/Fractal/EscapeTime/Runtime.hs

```
{-
et -- escape time fractals
Copyright (C) 2018,2019 Claude Heiland-Allen

5 This program is free software: you can redistribute it and/or modify
it under the terms of the GNU Affero General Public License as
published by the Free Software Foundation, either version 3 of the
License, or (at your option) any later version.

10 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 Affero General Public License for more details.

    You should have received a copy of the GNU Affero General Public License
    along with this program.  If not, see <https://www.gnu.org/licenses/>.
-}

module Fractal.EscapeTime.Runtime
  ( module Fractal.EscapeTime.Runtime.Compiler
  , module Fractal.EscapeTime.Runtime.Loader
  , module Fractal.EscapeTime.Runtime.Metadata
  , module Fractal.EscapeTime.Runtime.Render
  ) where

import Fractal.EscapeTime.Runtime.Compiler
import Fractal.EscapeTime.Runtime.Loader
import Fractal.EscapeTime.Runtime.Metadata
import Fractal.EscapeTime.Runtime.Render
```

56 src/Fractal/EscapeTime/Runtime/Loader.hs

```
{-
et -- escape time fractals
Copyright (C) 2018,2019 Claude Heiland-Allen

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU Affero General Public License as
    published by the Free Software Foundation, either version 3 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 Affero General Public License for more details.

    You should have received a copy of the GNU Affero General Public License
    along with this program.  If not, see <https://www.gnu.org/licenses/>.
-}

{-# LANGUAGE RecordWildCards #-}

module Fractal.EscapeTime.Runtime.Loader
  ( Output
  , Plain
  , Reference
  , Perturbation
  , PeriodTri
  , PeriodJSK
  , Newton
  , Misiurewicz
  , Size
  , Skew
  , DomainSize
  , ET(..)
  , readLib
  , version
  ) where
```

```
import Foreign
import Foreign.C
import Data.Coerce (coerce)
import Numeric.LongDouble (LongDouble)
--import Numeric.Compensated (Compensated)
import Numeric.MPFR.Types (MPFR)
import Numeric.Rounded.Simple (Rounded, withInRounded, withOutRounded, ↙
    ↳ withInOutRounded)
import Data.Vector.Storable.Mutable (IOVector, unsafeWith)
import qualified Data.Vector.Storable.Mutable as V

import qualified Fractal.EscapeTime.Runtime.Loader.Raw as Raw
import Fractal.EscapeTime.Formulas.Types (R2(..))
import Fractal.EscapeTime.Runtime.Loader.Load (load)

type Output t = (t, t, t, t)
type Plain t = Int -> t -> (t, t, t, t) -> t -> t -> t -> t -> Ptr CInt -> IO (↙
    ↳ Maybe (Output t))
type Reference t = IOVector t -> t -> t -> t -> Rounded -> Rounded -> Ptr CInt ↙
    ↳ -> IO (Maybe Int)
type Perturbation t = IOVector t -> Int -> t -> (t, t, t, t) -> t -> t -> t -> t↙
    ↳  -> Ptr CInt -> IO (Maybe (Output t, (t, t)))
type PeriodTri = Int -> Double -> Double -> Double -> Rounded -> Rounded -> ↙
    ↳ Rounded -> Ptr CInt -> IO (Maybe Int)
type PeriodJSK = Int -> Double -> Double -> Double -> Rounded -> Rounded -> ↙
    ↳ Rounded -> (Double, Double, Double, Double) -> Ptr CInt -> IO (Maybe Int)
type Newton = Int -> Int -> Double -> Double -> Rounded -> Rounded -> Ptr CInt ↙
    ↳ -> IO (Maybe (R2 Rounded))
type Misiurewicz = Int -> Int -> Int -> Double -> Double -> Rounded -> Rounded ↙
    ↳ -> Ptr CInt -> IO (Maybe (R2 Rounded))
type Size = Int -> Double -> Double -> Rounded -> Rounded -> Ptr CInt -> IO (↙
    ↳ Maybe (Rounded, (Double, Double, Double, Double)))
type Skew = Int -> Double -> Double -> Rounded -> Rounded -> Bool -> Ptr CInt ->↙
    ↳  IO (Maybe (Double, Double, Double, Double))
type DomainSize = Int -> Double -> Double -> Rounded -> Rounded -> Ptr CInt -> ↙
    ↳ IO (Maybe Rounded)

data ET = ET
  { close :: IO ()
  , name :: String
  , source :: String
  , degree :: Double
--  , plainf :: Plain Float
  , plain  :: Plain Double
  , plainl :: Plain LongDouble
--  , plainc :: Plain (Compensated Double)
--  , referencef :: Reference Float
  , reference  :: Reference Double
--  , referencel :: Reference LongDouble
--  , perturbationf :: Perturbation Float
  , perturbation  :: Perturbation Double
--  , perturbationl :: Perturbation LongDouble
  , period_tri :: PeriodTri
  , period_jsk :: PeriodJSK
  , newton :: Newton
  , misiurewicz :: Misiurewicz
```

```
    , size :: Size
    , skew :: Skew
    , domain_size :: DomainSize
    }

{-
wrapPlainF :: Raw.Plain CFloat -> Plain Float
wrapPlainF f n r2 (h1, h2, h3, h4) d e a b running = withArray (coerce [h1, h2, ↙
    ↳ h3, h4]) $ \hp -> withArray [0,0,0,0] $ \out -> do
  ok <- f (fromIntegral n) (coerce r2) hp (coerce d) (coerce e) (coerce a) (↙
      ↳ coerce b) out running
  if ok /= 0
  then do
    [de,p,ni,nf] <- coerce <$> peekArray 4 out
    return $ Just (de, p, ni, nf)
  else return Nothing
-}

wrapPlain  :: Raw.Plain CDouble -> Plain Double
wrapPlain  f n r2 (h1, h2, h3, h4) d e a b running = withArray (coerce [h1, h2, ↙
    ↳ h3, h4]) $ \hp -> withArray [0,0,0,0] $ \out -> do
  ok <- f (fromIntegral n) (coerce r2) hp (coerce d) (coerce e) (coerce a) (↙
      ↳ coerce b) out running
  if ok /= 0
  then do
    [de,p,ni,nf] <- coerce <$> peekArray 4 out
    return $ Just (de, p, ni, nf)
  else return Nothing

wrapPlainL :: (CInt -> Ptr LongDouble -> Ptr LongDouble -> Ptr LongDouble -> Ptr↙
    ↳  LongDouble -> Ptr LongDouble -> Ptr LongDouble -> Ptr LongDouble -> Ptr ↙
    ↳ CInt -> IO CInt) -> Plain LongDouble
wrapPlainL f n r2 (h1, h2, h3, h4) d e a b running = withArray [h1, h2, h3, h4] ↙
    ↳ $ \hp -> withArray [0,0,0,0] $ \out -> with r2 $ \r2p -> with d $ \dp -> ↙
    ↳ with e $ \ep -> with a $ \ap -> with b $ \bp -> do
  ok <- f (fromIntegral n) r2p hp dp ep ap bp out running
  if ok /= 0
  then do
    [de,p,ni,nf] <- peekArray 4 out
    return $ Just (de, p, ni, nf)
  else return Nothing

{-
wrapPlainC :: (CInt -> Ptr (Compensated Double) -> Ptr (Compensated Double) -> ↙
    ↳ Ptr (Compensated Double) -> Ptr (Compensated Double) -> Ptr (Compensated ↙
    ↳ Double) -> Ptr (Compensated Double) -> Ptr (Compensated Double) -> Ptr ↙
    ↳ CInt -> IO CInt) -> Plain (Compensated Double)
wrapPlainC f n r2 (h1, h2, h3, h4) d e a b running = withArray [h1, h2, h3, h4] ↙
    ↳ $ \hp -> withArray [0,0,0,0] $ \out -> with r2 $ \r2p -> with d $ \dp -> ↙
    ↳ with e $ \ep -> with a $ \ap -> with b $ \bp -> do
  ok <- f (fromIntegral n) r2p hp dp ep ap bp out running
  if ok /= 0
  then do
    [de,p,ni,nf] <- peekArray 4 out
    return $ Just (de, p, ni, nf)
  else return Nothing
```

```
wrapReferenceF :: Raw.Reference CFloat -> Reference Float
wrapReferenceF f v r2 d e a b running = unsafeWith v $ \vp -> withInRounded a $ ↙
    ↳ \ap -> withInRounded b $ \bp -> do
  n <- f (fromIntegral (V.length v `div` 3)) (coerce r2) (coerce d) (coerce e) ↙
      ↳ ap bp (castPtr vp) running
  ok <- peek running
  if ok /= 0
    then return (Just $ fromIntegral n)
    else return Nothing
-}

wrapReference  :: Raw.Reference CDouble -> Reference Double
wrapReference  f v r2 d e a b running = unsafeWith v $ \vp -> withInRounded a $ ↙
    ↳ \ap -> withInRounded b $ \bp -> do
  n <- f (fromIntegral (V.length v `div` 3)) (coerce r2) (coerce d) (coerce e) ↙
      ↳ ap bp (castPtr vp) running
  ok <- peek running
  if ok /= 0
    then return (Just $ fromIntegral n)
    else return Nothing

{-
wrapReferenceL :: (CInt -> Ptr LongDouble -> Ptr LongDouble -> Ptr LongDouble ->↙
    ↳  Ptr MPFR -> Ptr MPFR -> Ptr LongDouble -> Ptr CInt -> IO CInt) -> ↙
    ↳ Reference LongDouble
wrapReferenceL f v r2 d e a b running = unsafeWith v $ \vp -> withInRounded a $ ↙
    ↳ \ap -> withInRounded b $ \bp -> with r2 $ \r2p -> with d $ \dp -> with e $↙
    ↳  \ep -> do
  n <- f (fromIntegral (V.length v `div` 3)) r2p dp ep ap bp vp running
  ok <- peek running
  if ok /= 0
    then return (Just $ fromIntegral n)
    else return Nothing

wrapPerturbationF :: Raw.Perturbation CFloat -> Perturbation Float
wrapPerturbationF f v m r2 (h1, h2, h3, h4) d e a b running = unsafeWith v $ \vp↙
    ↳  -> withArray (map coerce [h1, h2, h3, h4]) $ \hp -> withArray ↙
    ↳ [0,0,0,0,0,0] $ \out -> do
  ok <- f (fromIntegral m) (fromIntegral (V.length v `div` 3)) (coerce r2) hp (↙
      ↳ coerce d) (coerce e) (coerce a) (coerce b) (castPtr vp) out running
  if ok /= 0
  then do
    [de,p,ni,nf,u,v] <- coerce <$> peekArray 6 out
    return $ Just ((de, p, ni, nf),(u,v))
  else return Nothing
-}

wrapPerturbation  :: Raw.Perturbation CDouble -> Perturbation Double
wrapPerturbation  f v m r2 (h1, h2, h3, h4) d e a b running = unsafeWith v $ \vp↙
    ↳  -> withArray (map coerce [h1, h2, h3, h4]) $ \hp -> withArray ↙
    ↳ [0,0,0,0,0,0] $ \out -> do
  ok <- f (fromIntegral m) (fromIntegral (V.length v `div` 3)) (coerce r2) hp (↙
      ↳ coerce d) (coerce e) (coerce a) (coerce b) (castPtr vp) out running
  if ok /= 0
  then do
    [de,p,ni,nf,u,v] <- coerce <$> peekArray 6 out
    return $ Just ((de, p, ni, nf),(u,v))
```

```
    else return Nothing

{-
wrapPerturbationL :: (CInt -> CInt -> Ptr LongDouble -> Ptr LongDouble -> Ptr ↙
    ↘ LongDouble -> Ptr LongDouble -> Ptr LongDouble -> Ptr LongDouble -> Ptr ↙
    ↘ LongDouble -> Ptr LongDouble -> Ptr CInt -> IO CInt) -> Perturbation ↙
    ↘ LongDouble
wrapPerturbationL f v m r2(h1, h2, h3, h4) d e a b running = unsafeWith v $ \vp ↙
    ↘ -> withArray [h1, h2, h3, h4] $ \hp -> withArray [0,0,0,0,0,0] $ \out -> ↙
    ↘ with r2 $ \r2p -> with d $ \dp -> with e $ \ep -> with a $ \ap -> with b $↙
    ↘  \bp -> do
  ok <- f (fromIntegral m) (fromIntegral (V.length v `div` 3)) r2p hp dp ep ap ↙
      ↘ bp vp out running
  if ok /= 0
  then do
    [de,p,ni,nf,u,v] <- peekArray 6 out
    return $ Just ((de, p, ni, nf),(u,v))
  else return Nothing
-}

wrapPeriodTri :: Raw.PeriodTri -> PeriodTri
wrapPeriodTri f n r2 d e a b h running = withInRounded a $ \ap -> withInRounded ↙
    ↘ b $ \bp -> withInRounded h $ \hp -> do
  p <- f (fromIntegral n) (coerce r2) (coerce d) (coerce e) ap bp hp running
  ok <- peek running
  if p > 0 && ok /= 0
    then return (Just $ fromIntegral p)
    else return Nothing

wrapPeriodJSK :: Raw.PeriodJSK -> PeriodJSK
wrapPeriodJSK f n r2 d e a b h (k0, k1, k2, k3) running = withInRounded a $ \ap ↙
    ↘ -> withInRounded b $ \bp -> withInRounded h $ \hp -> withArray (coerce [k0↙
    ↘ , k1, k2, k3]) $ \kp -> do
  p <- f (fromIntegral n) (coerce r2) (coerce d) (coerce e) ap bp hp kp running
  ok <- peek running
  if p > 0 && ok /= 0
    then return (Just $ fromIntegral p)
    else return Nothing

wrapNewton :: Raw.Newton -> Newton
wrapNewton f n p d e a b running = do
  (a', (b', ok)) <- withInOutRounded a $ \ap -> withInOutRounded b $ \bp -> do
    f (fromIntegral n) (fromIntegral p) (coerce d) (coerce e) ap bp running
  if ok /= 0
    then return (Just (R2 a' b'))
    else return Nothing

wrapMisiurewicz :: Raw.Misiurewicz -> Misiurewicz
wrapMisiurewicz f n q p d e a b running = do
  (a', (b', ok)) <- withInOutRounded a $ \ap -> withInOutRounded b $ \bp -> do
    f (fromIntegral n) (fromIntegral q) (fromIntegral p) (coerce d) (coerce e) ↙
        ↘ ap bp running
  if ok /= 0
    then return (Just (R2 a' b'))
    else return Nothing

wrapSize :: Raw.Size -> Size
```

```
wrapSize f p d e a b running = do
  let prec = 53
  withArray [0,0,0,0] $ \sp -> do
    (r, ok) <- withOutRounded prec $ \rp -> withInRounded a $ \ap -> ↙
        ↳ withInRounded b $ \bp -> do
      f (fromIntegral p) (coerce d) (coerce e) ap bp rp sp running
    if ok /= 0
      then do
        [w,x,y,z] <- coerce <$> peekArray 4 sp
        return (Just (r, (w, x, y, z)))
      else return Nothing

wrapSkew :: Raw.Skew -> Skew
wrapSkew f n d e a b usedz running = do
  withArray [0,0,0,0] $ \sp -> do
    ok <- withInRounded a $ \ap -> withInRounded b $ \bp -> do
      f (fromIntegral n) (coerce d) (coerce e) ap bp (if usedz then 1 else 0) sp↙
          ↳ running
    if ok /= 0
      then do
        [w,x,y,z] <- coerce <$> peekArray 4 sp
        return (Just (w, x, y, z))
      else return Nothing

wrapDomainSize :: Raw.DomainSize -> DomainSize
wrapDomainSize f p d e a b running = do
  let prec = 53
  (r, ok) <- withOutRounded prec $ \rp -> withInRounded a $ \ap -> withInRounded↙
      ↳ b $ \bp -> do
    f (fromIntegral p) (coerce d) (coerce e) ap bp rp running
  if ok /= 0
    then return (Just r)
    else return Nothing

readLib :: (IO (), Ptr Raw.S_Formula) -> IO (Either String ET)
readLib (close, p) = do
    s <- peek p
    if Raw.magic s == magic && Raw.ssize s == fromIntegral (sizeOf s) && Raw.↙
        ↳ version s == version
    then do
      name <- peekCString (Raw.name s)
      source <- peekCString (Raw.source s)
      let degree = coerce (Raw.degree s)
--          plainf = wrapPlainF (Raw.mkPlainF $ Raw.plainf s)
          plain  = wrapPlain  (Raw.mkPlain  $ Raw.plain  s)
          plainl = wrapPlainL (Raw.mkPlainL $ Raw.plainl s)
--          plainc = wrapPlainC (Raw.mkPlainC $ Raw.plainc s)
--          referencef = wrapReferenceF (Raw.mkReferenceF $ Raw.referencef s)
          reference  = wrapReference  (Raw.mkReference  $ Raw.reference  s)
--          referencel = wrapReferenceL (Raw.mkReferenceL $ Raw.referencel s)
--          perturbationf = wrapPerturbationF (Raw.mkPerturbationF $ Raw.↙
    ↳ perturbationf s)
          perturbation  = wrapPerturbation  (Raw.mkPerturbation  $ Raw.↙
              ↳ perturbation  s)
--          perturbationl = wrapPerturbationL (Raw.mkPerturbationL $ Raw.↙
    ↳ perturbationl s)
          period_tri = wrapPeriodTri (Raw.mkPeriodTri $ Raw.period_tri s)
```

```
            period_jsk = wrapPeriodJSK (Raw.mkPeriodJSK $ Raw.period_jsk s)
            newton = wrapNewton (Raw.mkNewton $ Raw.newton s)
            misiurewicz = wrapMisiurewicz (Raw.mkMisiurewicz $ Raw.misiurewicz s)
            size = wrapSize (Raw.mkSize $ Raw.size s)
            skew = wrapSkew (Raw.mkSkew $ Raw.skew s)
            domain_size = wrapDomainSize (Raw.mkDomainSize $ Raw.domain_size s)
        return (Right (ET{..}))
    else close >> (return $ Left "incompatible library found")

magic :: CInt
magic = 0xC01dCaf3

version :: CInt
version = 8
```

57 src/Fractal/EscapeTime/Runtime/Loader/Raw.hsc

```
{-
et -- escape time fractals
Copyright (C) 2018,2019 Claude Heiland-Allen

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU Affero General Public License as
published by the Free Software Foundation, either version 3 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 Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License
along with this program.  If not, see <https://www.gnu.org/licenses/>.
-}

{-# LANGUAGE ForeignFunctionInterface #-}
{-# LANGUAGE RecordWildCards #-}

#include "formula.h"

module Fractal.EscapeTime.Runtime.Loader.Raw where

import Foreign
import Foreign.C.Types

import Numeric.MPFR.Types (MPFR)
import Numeric.LongDouble (LongDouble)
--import Numeric.Compensated (Compensated)

type Plain t = CInt -> t -> Ptr t -> t -> t -> t -> t -> Ptr t -> Ptr CInt -> IO↙
    ↳ CInt
type WrapPlain t = F_PlainP t -> CInt -> Ptr t -> Ptr t -> Ptr t -> Ptr t -> Ptr t -> Ptr↙
    ↳ t -> Ptr t -> Ptr t -> Ptr CInt -> IO CInt
type Reference t = CInt -> t -> t -> t -> Ptr MPFR -> Ptr MPFR -> Ptr t -> Ptr ↙
    ↳ CInt -> IO CInt
--type WrapReferenceL = F_ReferenceL -> CInt -> Ptr LongDouble -> Ptr LongDouble↙
    ↳ -> Ptr LongDouble -> Ptr MPFR -> Ptr MPFR -> Ptr LongDouble -> Ptr CInt ↙
```

```
    ↳ -> IO CInt
type Perturbation t = CInt -> CInt -> t -> Ptr t -> t -> t -> t -> t -> Ptr t ->↙
    ↳  Ptr t -> Ptr CInt -> IO CInt
--type WrapPerturbationL = F_PerturbationL -> CInt -> CInt -> Ptr LongDouble -> ↙
    ↳ Ptr LongDouble -> Ptr LongDouble -> Ptr LongDouble -> Ptr LongDouble -> ↙
    ↳ Ptr LongDouble -> Ptr LongDouble -> Ptr LongDouble -> Ptr CInt -> IO CInt
type PeriodTri = CInt -> CDouble -> CDouble -> CDouble -> Ptr MPFR -> Ptr MPFR ↙
    ↳ -> Ptr MPFR -> Ptr CInt -> IO CInt
type PeriodJSK = CInt -> CDouble -> CDouble -> CDouble -> Ptr MPFR -> Ptr MPFR ↙
    ↳ -> Ptr MPFR -> Ptr CDouble -> Ptr CInt -> IO CInt
type Newton = CInt -> CInt -> CDouble -> CDouble -> Ptr MPFR -> Ptr MPFR -> Ptr ↙
    ↳ CInt -> IO CInt
type Misiurewicz = CInt -> CInt -> CInt -> CDouble -> CDouble -> Ptr MPFR -> Ptr↙
    ↳  MPFR -> Ptr CInt -> IO CInt
type Size = CInt -> CDouble -> CDouble -> Ptr MPFR -> Ptr MPFR -> Ptr MPFR -> ↙
    ↳ Ptr CDouble -> Ptr CInt -> IO CInt
type Skew = CInt -> CDouble -> CDouble -> Ptr MPFR -> Ptr MPFR -> CInt -> Ptr ↙
    ↳ CDouble -> Ptr CInt -> IO CInt
type DomainSize = CInt -> CDouble -> CDouble -> Ptr MPFR -> Ptr MPFR -> Ptr MPFR↙
    ↳  -> Ptr CInt -> IO CInt

type F_PlainP t = FunPtr (Plain t)
--type F_PlainF = FunPtr (Plain CFloat)
type F_Plain  = FunPtr (Plain CDouble)
type F_PlainL = FunPtr (Plain LongDouble)
--type F_PlainC = FunPtr (Plain (Compensated Double))
--type F_ReferenceF = FunPtr (Reference CFloat)
type F_Reference  = FunPtr (Reference CDouble)
--type F_ReferenceL = FunPtr (Reference LongDouble)
--type F_PerturbationF = FunPtr (Perturbation CFloat)
type F_Perturbation  = FunPtr (Perturbation CDouble)
--type F_PerturbationL = FunPtr (Perturbation LongDouble)
type F_PeriodTri = FunPtr PeriodTri
type F_PeriodJSK = FunPtr PeriodJSK
type F_Newton = FunPtr Newton
type F_Misiurewicz = FunPtr Misiurewicz
type F_Size = FunPtr Size
type F_Skew = FunPtr Skew
type F_DomainSize = FunPtr DomainSize

--foreign import ccall safe "dynamic" mkPlainF :: F_PlainF -> Plain CFloat
foreign import ccall safe "dynamic" mkPlain  :: F_Plain  -> Plain CDouble
foreign import ccall safe "formula.h et_wrap_plainl" mkPlainL :: WrapPlain ↙
    ↳ LongDouble
--foreign import ccall safe "formula.h et_wrap_plainc" mkPlainC :: WrapPlain (↙
    ↳ Compensated Double)
--foreign import ccall safe "dynamic" mkReferenceF :: F_ReferenceF -> Reference ↙
    ↳ CFloat
foreign import ccall safe "dynamic" mkReference  :: F_Reference  -> Reference ↙
    ↳ CDouble
--foreign import ccall safe "formula.h et_wrap_referencel" mkReferenceL :: ↙
    ↳ WrapReferenceL
--foreign import ccall safe "dynamic" mkPerturbationF :: F_PerturbationF -> ↙
    ↳ Perturbation CFloat
foreign import ccall safe "dynamic" mkPerturbation  :: F_Perturbation  -> ↙
    ↳ Perturbation CDouble
--foreign import ccall safe "formula.h et_wrap_perturbationl" mkPerturbationL ::↙
```

```
    ↳ WrapPerturbationL
foreign import ccall safe "dynamic" mkPeriodTri :: F_PeriodTri -> PeriodTri
foreign import ccall safe "dynamic" mkPeriodJSK :: F_PeriodJSK -> PeriodJSK
foreign import ccall safe "dynamic" mkNewton :: F_Newton -> Newton
foreign import ccall safe "dynamic" mkMisiurewicz :: F_Misiurewicz -> ↙
    ↳ Misiurewicz
foreign import ccall safe "dynamic" mkSize    :: F_Size    -> Size
foreign import ccall safe "dynamic" mkSkew    :: F_Skew    -> Skew
foreign import ccall safe "dynamic" mkDomainSize    :: F_DomainSize    -> ↙
    ↳ DomainSize

data S_Formula = S_Formula
  { magic :: CInt
  , ssize :: CInt
  , version :: CInt
  , name :: Ptr CChar
  , source :: Ptr CChar
  , degree :: CDouble
--  , plainf :: F_PlainF
  , plain  :: F_Plain
  , plainl :: F_PlainL
--  , plainc :: F_PlainC
--  , referencef :: F_ReferenceF
  , reference  :: F_Reference
--  , referencel :: F_ReferenceL
--  , perturbationf :: F_PerturbationF
  , perturbation  :: F_Perturbation
--  , perturbationl :: F_PerturbationL
  , period_tri :: F_PeriodTri
  , period_jsk :: F_PeriodJSK
  , newton :: F_Newton
  , misiurewicz :: F_Misiurewicz
  , size :: F_Size
  , skew :: F_Skew
  , domain_size :: F_DomainSize
  }

instance Storable S_Formula where

  sizeOf _ = (#size struct formula)

  alignment _ = (#alignment struct formula)

  peek p = do
    magic <- (#peek struct formula, magic) p
    ssize <- (#peek struct formula, ssize) p
    version <- (#peek struct formula, version) p
    name <- (#peek struct formula, name) p
    source <- (#peek struct formula, source) p
    degree <- (#peek struct formula, degree) p
--    plainf <- (#peek struct formula, plainf) p
    plain  <- (#peek struct formula, plain ) p
    plainl <- (#peek struct formula, plainl) p
--    plainc <- (#peek struct formula, plainc) p
--    referencef <- (#peek struct formula, referencef) p
    reference  <- (#peek struct formula, reference ) p
--    referencel <- (#peek struct formula, referencel) p
```

```
    --    perturbationf <- (#peek struct formula, perturbationf) p
        perturbation  <- (#peek struct formula, perturbation ) p
    --    perturbationl <- (#peek struct formula, perturbationl) p
        period_tri <- (#peek struct formula, period_tri) p
        period_jsk <- (#peek struct formula, period_jsk) p
        newton <- (#peek struct formula, newton) p
        misiurewicz <- (#peek struct formula, misiurewicz) p
        size <- (#peek struct formula, size) p
        skew <- (#peek struct formula, skew) p
        domain_size <- (#peek struct formula, domain_size) p
        return $ S_Formula{..}

      poke p S_Formula{..} = do
        (#poke struct formula, magic) p magic
        (#poke struct formula, ssize) p ssize
        (#poke struct formula, version) p version
        (#poke struct formula, name) p name
        (#poke struct formula, source) p source
        (#poke struct formula, degree) p degree
    --    (#poke struct formula, plainf) p plainf
        (#poke struct formula, plain ) p plain
        (#poke struct formula, plainl) p plainl
    --    (#poke struct formula, plainc) p plainc
    --    (#poke struct formula, referencef) p referencef
        (#poke struct formula, reference ) p reference
    --    (#poke struct formula, referencel) p referencel
    --    (#poke struct formula, perturbationf) p perturbationf
        (#poke struct formula, perturbation ) p perturbation
    --    (#poke struct formula, perturbationl) p perturbationl
        (#poke struct formula, period_tri) p period_tri
        (#poke struct formula, period_jsk) p period_jsk
        (#poke struct formula, newton) p newton
        (#poke struct formula, misiurewicz) p misiurewicz
        (#poke struct formula, size) p size
        (#poke struct formula, skew) p skew
        (#poke struct formula, domain_size) p domain_size
```

58 src/Fractal/EscapeTime/Runtime/Metadata.hs

```
{-
et -- escape time fractals
Copyright (C) 2018,2019 Claude Heiland-Allen

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU Affero General Public License as
published by the Free Software Foundation, either version 3 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 Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License
along with this program.  If not, see <https://www.gnu.org/licenses/>.
-}
```

```
{-# LANGUAGE RecordWildCards #-}

module Fractal.EscapeTime.Runtime.Metadata
  ( Metadata(..)
  , toString
  , fromString
  ) where

import Text.Read (readMaybe)
import Numeric.Rounded.Simple (Rounded, RoundingMode(TowardNearest), read', show↙
    ↳ ', exponent')

data Metadata = Metadata
  { metadataFormula :: [String]
  , metadataP, metadataQ :: Int
  , metadataD, metadataE :: Double
  , metadataA, metadataB, metadataR :: Rounded
  , metadataT :: (Double, Double, Double, Double)
  , metadataN :: Int
  , metadataDEWeight :: Double
  }

toString :: Metadata -> String
toString Metadata{..} =
  unlines metadataFormula ++ "\n" ++
  "p=" ++ show metadataP ++ "\n" ++
  "q=" ++ show metadataQ ++ "\n" ++
  "d=" ++ show metadataD ++ "\n" ++
  "e=" ++ show metadataE ++ "\n" ++
  "a=" ++ show' metadataA ++ "\n" ++
  "b=" ++ show' metadataB ++ "\n" ++
  "r=" ++ show' metadataR ++ "\n" ++
  "t=" ++ show metadataT ++ "\n" ++
  "n=" ++ show metadataN ++ "\n" ++
  "\n" ++
  "de=" ++ show metadataDEWeight ++ "\n"

fromString :: String -> Maybe Metadata
fromString s =
  case break null (lines s) of
    (metadataFormula, "":('p':'=':sp):('q':'=':sq):('d':'=':sd):('e':'=':se):('a↙
        ↳ ':'=':sa):('b':'=':sb):('r':'=':sr):('t':'=':st):('n':'=':sn):"":('d↙
        ↳ ':'e':'=':sde):[]) -> do
      metadataP <- readMaybe sp
      metadataQ <- readMaybe sq
      metadataD <- readMaybe sd
      metadataE <- readMaybe se
      metadataR <- Just $ read' TowardNearest 53 sr -- FIXME
      let prec = max 53 (53 - exponent' metadataR)
      metadataA <- Just $ read' TowardNearest prec sa -- FIXME
      metadataB <- Just $ read' TowardNearest prec sb -- FIXME
      metadataT <- readMaybe st
      metadataN <- readMaybe sn
      metadataDEWeight <- readMaybe sde
      return Metadata{..}
    _ -> Nothing
```

59 src/Fractal/EscapeTime/Runtime/Render/Buffer.hs

```
{-
et -- escape time fractals
Copyright (C) 2018 Claude Heiland-Allen

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU Affero General Public License as
published by the Free Software Foundation, either version 3 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 Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License
along with this program.  If not, see <https://www.gnu.org/licenses/>.
-}

module Fractal.EscapeTime.Runtime.Render.Buffer where

import qualified Data.Vector.Storable.Mutable as V

data Buffer t = Buffer{ width, height, channels :: Int, contents :: V.IOVector t }

buffer :: V.Storable a => Int -> Int -> Int -> a -> IO (Buffer a)
buffer w h c i = do
  v <- V.replicate (w * h * c) i
  return (Buffer w h c v)
```

60 src/Fractal/EscapeTime/Runtime/Render/Coord.hs

```
{-
et -- escape time fractals
Copyright (C) 2018,2019 Claude Heiland-Allen

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU Affero General Public License as
published by the Free Software Foundation, either version 3 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 Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License
along with this program.  If not, see <https://www.gnu.org/licenses/>.
-}

module Fractal.EscapeTime.Runtime.Render.Coord (Coord, coord, UnCoord, unCoord) where

import Fractal.EscapeTime.Runtime.Render.Dither
```

```
import Fractal.EscapeTime.Runtime.Render.Transform

type Coord t = Transform -> t -> Int -> Int -> Int -> Int -> (t, t)
type UnCoord t = Transform -> t -> Int -> Int -> t -> t -> Maybe (Int, Int)

coord :: RealFrac t => Dither -> Coord t
coord d t px w h i j =
  let (u, v) = dither d i j
      (x, y) = apply t (u - fromIntegral w / 2, v - fromIntegral h / 2)
  in  (px * realToFrac x, px * realToFrac y)

unCoord :: RealFrac t => UnCoord t
unCoord t px w h x0 y0 =
  let x = realToFrac (x0 / px)
      y = realToFrac (y0 / px)
      (u, v) = apply (inverse t) (x, y)
      i = u + fromIntegral w / 2
      j = v + fromIntegral h / 2
      limit = 2^24
  in  if abs i > limit || abs j > limit then Nothing else Just (round i, round j↙
      ↳ )
```

61 src/Fractal/EscapeTime/Runtime/Render/Dither.hs

```
{-
et -- escape time fractals
Copyright (C) 2018 Claude Heiland-Allen

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU Affero General Public License as
published by the Free Software Foundation, either version 3 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 Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License
along with this program.  If not, see <https://www.gnu.org/licenses/>.
-}

module Fractal.EscapeTime.Runtime.Render.Dither (Dither(..), dither, ↙
    ↳ defaultDither) where

import Control.Monad.Identity (runIdentity)
import Data.Word (Word32)
import Data.Bits (xor, shiftL, shiftR)

data Dither = Dither{ ditherGauss :: Bool, ditherSeed :: Int, ditherRadius :: ↙
    ↳ Double }

defaultDither :: Dither
defaultDither = Dither False 1 1

-- http://www.burtleburtle.net/bob/hash/integer.html
burtleHash :: Word32 -> Word32
```

```
burtleHash a = runIdentity $ do
  a <- pure $ (a+0x7ed55d16) + (a`shiftL`12);
  a <- pure $ (a`xor`0xc761c23c) `xor` (a`shiftR`19);
  a <- pure $ (a+0x165667b1) + (a`shiftL`5);
  a <- pure $ (a+0xd3a2646c) `xor` (a`shiftL`9);
  a <- pure $ (a+0xfd7046c5) + (a`shiftL`3);
  a <- pure $ (a`xor`0xb55a4f09) `xor` (a`shiftR`16);
  pure a

uniform :: Word32 -> Word32 -> Word32 -> Double
uniform x y c = fromIntegral (burtleHash (x + burtleHash (y + burtleHash c))) / ↙
    ↳ 0x100000000

boxMuller :: Double -> Double -> Double -> (Double, Double)
boxMuller s u v =
  let r | u < 0 && u < 1 = s * sqrt (-2 * log u)
        | otherwise = 0
      t = 2 * pi * v
  in  (r * cos t, r * sin t)

dither :: Dither -> Int -> Int -> (Double, Double)
dither (Dither gauss seed radius) x y =
  let u = uniform (fromIntegral x) (fromIntegral y) (fromIntegral (2 * seed) + ↙
      ↳ 0)
      v = uniform (fromIntegral x) (fromIntegral y) (fromIntegral (2 * seed) + ↙
          ↳ 1)
      (dx, dy)  | gauss = boxMuller (0.5 * radius) u v
                | otherwise = (radius * (u - 0.5), radius * (v - 0.5))
  in  (fromIntegral x + dx, fromIntegral y + dy)
```

62 src/Fractal/EscapeTime/Runtime/Render.hs

```
{-
et -- escape time fractals
Copyright (C) 2018,2019 Claude Heiland-Allen

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU Affero General Public License as
published by the Free Software Foundation, either version 3 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 Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License
along with this program.  If not, see <https://www.gnu.org/licenses/>.
-}

module Fractal.EscapeTime.Runtime.Render
  ( module Fractal.EscapeTime.Runtime.Render.View
  , module Fractal.EscapeTime.Runtime.Render.Buffer
  , render
  ) where

import Data.Vector.Storable.Mutable (set)
```

```
--import Numeric.Compensated (Compensated)
import Numeric.LongDouble (LongDouble)
import Numeric.Rounded.Simple (exponent')

import Fractal.EscapeTime.Runtime.Render.View
import Fractal.EscapeTime.Runtime.Render.Buffer
import Fractal.EscapeTime.Runtime.Render.Progressive
import Fractal.EscapeTime.Runtime.Render.Dither (defaultDither)
import Fractal.EscapeTime.Runtime.Render.Transform (Transform)
import Fractal.EscapeTime.Runtime.Render.Coord (coord, unCoord)
import qualified Fractal.EscapeTime.Runtime.Render.Plain as Plain
import qualified Fractal.EscapeTime.Runtime.Render.Perturb as Perturb
import Fractal.EscapeTime.Runtime.Loader (ET)

render :: ET -> Double -> Int -> Double -> Double -> View -> Buffer Float -> ↙
    ↳ Transform -> Maybe Progressive -> IO (IO Bool, IO Bool)
render et er maxiters d e v@(View _ _ r) b@(Buffer _ h _ c) tr prog = do
  set c (-1)
  case exponent' r - ceiling (logBase 2 (fromIntegral h :: Double)) of
    e
--      | e >    -21 -> print "f" >> Plain.render   (realToFrac er :: Float ↙
    ↳            ) et maxiters (realToFrac d) (realToFrac e) v b tr (coord ↙
    ↳ defaultDither) prog
      | e >    -50 -> print "d" >> Plain.render   (realToFrac er :: Double ↙
        ↳           ) et maxiters (realToFrac d) (realToFrac e) v b tr (coord↙
        ↳  defaultDither) prog
      | e >    -61 -> print "l" >> Plain.render   (realToFrac er :: LongDouble ↙
        ↳         ) et maxiters (realToFrac d) (realToFrac e) v b tr (coord ↙
        ↳ defaultDither) prog
--      | e >   -103 -> print "c" >> Plain.render   (realToFrac er :: ↙
    ↳ Compensated Double) et maxiters (realToFrac d) (realToFrac e) v b tr (↙
    ↳ coord defaultDither) prog
--      | e >   -120 -> print "pf" >> Perturb.render (realToFrac er :: Float ↙
    ↳            ) et maxiters (realToFrac d) (realToFrac e) v b tr (coord ↙
    ↳ defaultDither, unCoord) prog
      | e >  -1020 -> print "pd" >> Perturb.render (realToFrac er :: Double ↙
        ↳           ) et maxiters (realToFrac d) (realToFrac e) v b tr (coord ↙
        ↳ defaultDither, unCoord) prog
--      | e > -16380 -> print "pl" >> Perturb.render (realToFrac er :: ↙
    ↳ LongDouble       ) et maxiters (realToFrac d) (realToFrac e) v b tr (coord↙
    ↳  defaultDither, unCoord) prog
    _ -> return (return False, return False)
```

63 src/Fractal/EscapeTime/Runtime/Render/Perturb.hs

```
{-
et -- escape time fractals
Copyright (C) 2018,2019 Claude Heiland-Allen

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU Affero General Public License as
published by the Free Software Foundation, either version 3 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 Affero General Public License for more details.

15  You should have received a copy of the GNU Affero General Public License
    along with this program.  If not, see <https://www.gnu.org/licenses/>.
    -}

    module Fractal.EscapeTime.Runtime.Render.Perturb where
20
    import Control.Exception (Exception, catch, throwIO)
    import Control.Monad (forM_, unless, when)
    import Data.IORef (newIORef, readIORef, writeIORef, atomicModifyIORef')
    import Data.List (sort, sortBy)
25  import Data.Ord (comparing)
    import Data.Atomics.Counter (newCounter, incrCounter)
    import Foreign
    import GHC.Conc (getNumCapabilities)
    import Control.Concurrent.Async (async, wait, replicateConcurrently)
30  import Numeric.LongDouble (LongDouble)
    import Numeric.Rounded.Simple (RoundingMode(TowardNearest), precRound, exponent'
        ↳ , encodeFloat', decodeFloat', add_, sub_, precision)
    import qualified Data.Vector.Storable as S
    import qualified Data.Vector.Storable.Mutable as SM

35  import qualified Fractal.EscapeTime.Runtime.Loader as ET
    import Fractal.EscapeTime.Runtime.Loader (ET)
    import Fractal.EscapeTime.Runtime.Render.View
    import Fractal.EscapeTime.Runtime.Render.Buffer
    import Fractal.EscapeTime.Runtime.Render.Coord
40  import Fractal.EscapeTime.Runtime.Render.Transform
    import Fractal.EscapeTime.Runtime.Render.Progressive
    import Fractal.EscapeTime.Formulas.Types (R2(..))

    class (Storable t, RealFloat t, Show t) => Perturb t where
45    reference :: ET -> ET.Reference t
      perturbation :: ET -> ET.Perturbation t

    --instance Perturb Float where reference = ET.referencef ; perturbation = ET.
        ↳ perturbationf
    instance Perturb Double where reference = ET.reference ; perturbation = ET.
        ↳ perturbation
50  --instance Perturb LongDouble where reference = ET.referencel ; perturbation =
        ↳ ET.perturbationl

    render :: Perturb t => t -> ET -> Int -> t -> t -> View -> Buffer Float ->
        ↳ Transform -> (Coord t, UnCoord t) -> Maybe Progressive -> IO (IO Bool, IO
        ↳ Bool)
    render er et maxiters d_ e_ (View x0hi y0hi r0hi) buf@(Buffer w h _c@4 _) tr (
        ↳ coord', unCoord') mprog = do
      runningP <- malloc
55    poke runningP 1
      a <- async $ Control.Exception.catch (do
        let prec = max 53 $ 53 - exponent' r0hi
            count = w * h
        orbit <- SM.new (maxiters * 3)
60      done <- SM.replicate count False
        glitch_refescaped  <- SM.replicate count (-1 `asTypeOf` er)
        glitch_pauldelbrot <- SM.replicate count (-1 `asTypeOf` er)
```

```
    glitched_refescapedR <- newIORef True
    glitched_pauldelbrotR <- newIORef True
    referenceR <- newIORef (precRound TowardNearest prec x0hi, precRound ↙
        ↳ TowardNearest prec y0hi, w `div` 2, h `div` 2)
    let r0 = uncurry encodeFloat $ decodeFloat' r0hi
        px = 2 * r0 / fromIntegral h
        (aa, ab, ba, bb) = getTransform tr
        pxt = (px * realToFrac aa, px * realToFrac ab, px * realToFrac ba, px * ↙
            ↳ realToFrac bb)
        active = do
          running <- peek runningP
          glitched_refescaped <- readIORef glitched_refescapedR
          glitched_pauldelbrot <- readIORef glitched_pauldelbrotR
          return (running /= 0 && (glitched_refescaped || glitched_pauldelbrot))
    whileM_ active $ do
     writeIORef glitched_refescapedR False
     writeIORef glitched_pauldelbrotR False
     with (0::Int) $ \refescapedP -> with (0::Int) $ \pauldelbrotP -> do
      (a, b, i0, j0) <- readIORef referenceR
      print (i0, j0)
      mm <- reference et orbit (er * er) d_ e_ a b runningP
      case mm of
        Just iters -> do
          let (da0, db0) = coord' tr px w h i0 j0
          npixelR <- newCounter 0
          threads <- getNumCapabilities
          let worker pb0@(pb, _) ge0@(ge, _) = do
                npixel <- subtract 1 `fmap` incrCounter 1 npixelR
                if npixel < count
                  then do
                    let Pixel (i, j, _, _) = case mprog of
                                               Just prog -> prog S.! npixel
                                               Nothing -> case npixel `divMod` w of
                                                 (j, i) -> Pixel (fromIntegral i, ↙
                                                     ↳ fromIntegral j, 0, 0)
                        (a0, b0) = coord' tr px w h (fromIntegral i) (↙
                            ↳ fromIntegral j)
                        da = a0 - da0
                        db = b0 - db0
                    gr <- SM.read glitch_refescaped npixel
                    gp <- SM.read glitch_pauldelbrot npixel
                    let g = min gr gp
                    if g < 0
                      then do -- pixel needs computing
                        -- BEGIN HACK
                        -- ignore isolated glitches in 3x3 neighbourhood
                        -- speeds up high resolution rendering a great deal
                        -- at cost of slightly inaccurate images
                        -- doesn't work for interactive mode because the vectors↙
                            ↳  are not in image order...
                        isolated <- case mprog of
                          Just _ -> return False
                          Nothing -> do
                            let is = [fromIntegral i - 1 .. fromIntegral i + 1]
                                js = [fromIntegral j - 1 .. fromIntegral j + 1]
                                ns = [ j * w + i | j <- js, 0 <= j, j < h, i <- ↙
                                    ↳ is, 0 <= i, i < w ]
```

```
                                grs <- mapM (SM.read glitch_refescaped) ns
                                gps <- mapM (SM.read glitch_pauldelbrot) ns
                                let (lo, hi) = span (< 0) . sort $ zipWith min grs ↙
                                    ↳ gps
                                    isolated = length lo <= 1
                                return isolated
                              mout <-
                                if isolated
                                  then pure (Just ((0, 0, 0, 0), (0, 0)))
                                  else
                                    perturbation et orbit iters (er * er) pxt d_ e_ da↙
                                        ↳ db runningP
                              -- END HACK
                              case mout of
                                Nothing -> throwIO Cancelled
                                Just ((de, p, ni, nf), (u, v)) -> do
                                  if de < 0
                                  then do -- glitched
                                    if (fromIntegral i == i0 && fromIntegral j == j0)
                                    then do -- reference pixel
                                      output mprog done buf npixel True (if nf == ↙
                                          ↳ negate 2 then 1e30 else 1e-30, realToFrac ↙
                                          ↳ p, 0, 0)
                                      SM.write glitch_refescaped  npixel 0
                                      SM.write glitch_pauldelbrot npixel 0
                                      worker pb0 ge0
                                    else if nf == negate 2
                                      then do -- reference escaped
                                        output mprog done buf npixel False (-1, ↙
                                            ↳ realToFrac p, 0, 0)
                                        SM.write glitch_refescaped npixel de
                                        SM.write glitch_pauldelbrot npixel p
                                        poke refescapedP 1
                                        worker pb0 (if de > ge then (de, ((u, v), (i, ↙
                                            ↳ j))) else ge0)
                                      else do -- pauldebrot glitch , nf == negate 3
                                        output mprog done buf npixel False (-1, ↙
                                            ↳ realToFrac p, 0, 0)
                                        SM.write glitch_refescaped npixel 0
                                        SM.write glitch_pauldelbrot npixel de
                                        poke pauldelbrotP 1
                                        worker (if de > pb then (de, ((u, v), (i, j)))↙
                                            ↳  else pb0) ge0
                                  else do -- non glitched
                                      output mprog done buf npixel True (realToFrac de↙
                                          ↳ , realToFrac p, realToFrac ni, realToFrac ↙
                                          ↳ nf)
                                      SM.write glitch_refescaped  npixel 0
                                      SM.write glitch_pauldelbrot npixel 0
                                      worker pb0 ge0
                            else do
                              worker pb0 ge0
                        else do
                          return (pb0, ge0)
                    rs <- replicateConcurrently threads $ worker (-1/0, ((0, 0), (0, 0))) ↙
                        ↳ (-1/0, ((0, 0), (0, 0)))
                    g <- peek refescapedP
```

```
            when (g /= 0) $ writeIORef glitched_refescapedR True
            g <- peek pauldelbrotP
            when (g /= 0) $ writeIORef glitched_pauldelbrotR True
            whenM_ active $ do
              let (pb, ((u0, v0), (i0, j0))):_ = sortBy (flip (comparing fst)) (↙
                  ↳ map fst rs)
                  (ge, ((u1, v1), (i1, j1))):_ = sortBy (flip (comparing fst)) (↙
                      ↳ map snd rs)
                  mdxy
                    | pb > -1/0 = Just ((u0, v0), (i0, j0))
                    | ge > -1/0 = Just ((u1, v1), (i1, j1))
                    | otherwise = Nothing
              case mdxy of
                Just ((u, v), (ii, jj)) -> do
                  let a' = add_ TowardNearest prec a $ uncurry (encodeFloat' ↙
                      ↳ TowardNearest 53) (decodeFloat u)
                      b' = add_ TowardNearest prec b $ uncurry (encodeFloat' ↙
                          ↳ TowardNearest 53) (decodeFloat v)
                  case unCoord tr px w h (u - da0) (v - db0) of
                    -- Just (i', j') -> writeIORef referenceR (a', b', i', j') -- ↙
                        ↳ FIXME doesn't work properly, rethink
                    _ -> do
--                    print ("internal warning: far glitch?", ii, jj, u, v)
                      let (u, v) = coord' tr px w h (fromIntegral ii) (↙
                          ↳ fromIntegral jj)
                          dx = u - da0
                          dy = v - db0
                          a' = add_ TowardNearest prec a $ uncurry (encodeFloat' ↙
                              ↳ TowardNearest 53) (decodeFloat dx)
                          b' = add_ TowardNearest prec b $ uncurry (encodeFloat' ↙
                              ↳ TowardNearest 53) (decodeFloat dy)
                      writeIORef referenceR (a', b', fromIntegral ii, fromIntegral↙
                          ↳  jj)
                _ -> print "internal error: no glitch?"
          _ -> throwIO Cancelled
        return True
        ) (\Cancelled -> return False)
    return  ( do{ poke runningP 0 ; r <- wait a ; free runningP ; return r }
            , do{                   r <- wait a ; free runningP ; return r }
            )
render _ _ _ _ _ _ _ _ _ _ _ = return (return False, return False)

taxicab :: Perturb t => Int -> Int -> Maybe Progressive -> SM.IOVector t -> IO (↙
    ↳ Int, Int, Word32)
taxicab w h mprog glitch = do
  let count = w * h
  distance <- SM.replicate count (maxBound :: Word32)
  forM_ [0 .. count - 1] $ \npixel -> do
    let Pixel (i, j, _, _) = case mprog of
                               Just prog -> prog S.! npixel
                               Nothing -> case npixel `divMod` w of
                                 (j, i) -> Pixel (fromIntegral i, fromIntegral j, 0, ↙
                                     ↳ 0)
        k = fromIntegral j * w + fromIntegral i
    g <- SM.read glitch npixel
    unless (g < 0) $ SM.write distance k 0
  forM_ [0 .. h - 1] $ \j -> do -- par
```

```
      forM_ [1 .. w - 1] $ \i -> do
        let ii = i - 1
            k = j * w + i
            kk = j * w + ii
        dkk <- SM.read distance kk
        dk  <- SM.read distance k
        when (dkk < dk) $ SM.write distance k (dkk + 1)
      forM_ [w - 2, w - 3 .. 0] $ \i -> do
        let ii = i + 1
            k = j * w + i
            kk = j * w + ii
        dkk <- SM.read distance kk
        dk  <- SM.read distance k
        when (dkk < dk) $ SM.write distance k (dkk + 1)
    forM_ [0 .. w - 1] $ \i -> do -- par
      forM_ [1 .. h - 1] $ \j -> do
        let jj = j - 1
            k = j * w + i
            kk = jj * w + i
        dkk <- SM.read distance kk
        dk  <- SM.read distance k
        when (dkk < dk) $ SM.write distance k (dkk + 1)
      forM_ [h - 2, h - 3 .. 0] $ \j -> do
        let jj = j + 1
            k = j * w + i
            kk = jj * w + i
        dkk <- SM.read distance kk
        dk  <- SM.read distance k
        when (dkk < dk) $ SM.write distance k (dkk + 1)
    miR <- newIORef 0
    mjR <- newIORef 0
    mdR <- newIORef 0
    forM_ [0 .. h - 1] $ \j -> do
      forM_ [0 .. w - 1] $ \i -> do
        let k = j * w + i
        md <- readIORef mdR
        d <- SM.read distance k
        when (d > md) $ do
          writeIORef miR i
          writeIORef mjR j
          writeIORef mdR d
    i <- readIORef miR
    j <- readIORef mjR
    d <- readIORef mdR
    return (i, j, d)

data Cancelled = Cancelled deriving Show
instance Exception Cancelled

whenM_, whileM_ :: IO Bool -> IO () -> IO ()

whenM_ test action = do
  ok <- test
  when ok action

whileM_ test action = whenM_ test $ do
    action
```

```
    whileM_ test action
```

64 src/Fractal/EscapeTime/Runtime/Render/Plain.hs

```
{-
et -- escape time fractals
Copyright (C) 2018,2019 Claude Heiland-Allen

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU Affero General Public License as
published by the Free Software Foundation, either version 3 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 Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License
along with this program.  If not, see <https://www.gnu.org/licenses/>.
-}

{-# LANGUAGE DataKinds #-}
{-# LANGUAGE FlexibleInstances #-}
{-# LANGUAGE ScopedTypeVariables #-}

module Fractal.EscapeTime.Runtime.Render.Plain where

import Control.Exception (Exception, catch, throwIO)
import Control.Monad (when)
import Foreign
import GHC.Conc (getNumCapabilities)
import Control.Concurrent.Async (async, wait, replicateConcurrently_)
import Data.Atomics.Counter (newCounter, incrCounter)
import Numeric.LongDouble (LongDouble)
--import Numeric.Compensated (Compensated)
import Numeric.Rounded (RoundingMode(TowardNearest), Precision)
import qualified Numeric.Rounded as R
import Numeric.Rounded.Simple (Rounded, reifyRounded)
import qualified Data.Vector.Storable as S
import qualified Data.Vector.Storable.Mutable as SM
import qualified Data.Vector.Unboxed as U
import qualified Data.Vector.Unboxed.Mutable as UM

import qualified Fractal.EscapeTime.Runtime.Loader as ET
import Fractal.EscapeTime.Runtime.Loader (ET)
import Fractal.EscapeTime.Runtime.Render.View
import Fractal.EscapeTime.Runtime.Render.Buffer
import Fractal.EscapeTime.Runtime.Render.Coord
import Fractal.EscapeTime.Runtime.Render.Transform
import Fractal.EscapeTime.Runtime.Render.Progressive

class (Storable t, RealFrac t) => Plain t where plain :: ET -> ET.Plain t
--instance Plain Float  where plain = ET.plainf
instance Plain Double where plain = ET.plain
instance Plain LongDouble where plain = ET.plainl
--instance Plain (Compensated Double) where plain = ET.plainc
```

```
render :: Plain t => t -> ET -> Int -> t -> t -> View -> Buffer Float -> Transform -> Coord t -> Maybe Progressive -> IO (IO Bool, IO Bool)
render er et maxiters d_ e_ (View x0hi y0hi r0hi) buf@(Buffer w h 4 _) tr coord' mprog = do
  runningP <- malloc
  poke runningP 1
  a <- async $ Control.Exception.catch (do
    let x0 = recodeFloat x0hi `asTypeOf` er
        y0 = recodeFloat y0hi
        r0 = recodeFloat r0hi
        px = 2 * r0 / fromIntegral h
        (aa, ab, ba, bb) = getTransform tr
        pxt = (px * realToFrac aa, px * realToFrac ab, px * realToFrac ba, px * realToFrac bb)
        count = w * h
    done <- SM.replicate count False
    npixelR <- newCounter 0
    threads <- getNumCapabilities
    let worker = do
          npixel <- subtract 1 `fmap` incrCounter 1 npixelR
          when (npixel < count) $ do
            let Pixel (i, j, _, _) = case mprog of
                                       Just prog -> prog S.! npixel
                                       Nothing -> case npixel `divMod` w of (j, i) -> Pixel (fromIntegral i, fromIntegral j, 0, 0)
                (a0, b0) = coord' tr px w h (fromIntegral i) (fromIntegral j)
                a = a0 + x0
                b = b0 + y0
            m <- plain et maxiters (er * er) pxt d_ e_ a b runningP
            case m of
              Nothing -> throwIO Cancelled
              Just (de, p, ni, nf) -> output mprog done buf npixel True (realToFrac de, realToFrac p, realToFrac ni, realToFrac nf)
            worker
    replicateConcurrently_ threads worker
    return True
    ) (\Cancelled -> return False)
  return  ( do{ poke runningP 0 ; r <- wait a ; free runningP ; return r }
          , do{                   r <- wait a ; free runningP ; return r }
          )
render _ _ _ _ _ _ _ _ _ _ _ = return (return False, return False)

data Cancelled = Cancelled deriving Show
instance Exception Cancelled

recodeFloat :: forall t . Fractional t => Rounded -> t
recodeFloat a = reifyRounded a (realToFrac :: Precision p => R.Rounded TowardNearest p -> t)
```

65 src/Fractal/EscapeTime/Runtime/Render/Progressive.hs

```
{-
et -- escape time fractals
Copyright (C) 2018,2019 Claude Heiland-Allen
```

```
    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU Affero General Public License as
    published by the Free Software Foundation, either version 3 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 Affero General Public License for more details.

    You should have received a copy of the GNU Affero General Public License
    along with this program.  If not, see <https://www.gnu.org/licenses/>.
    -}

module Fractal.EscapeTime.Runtime.Render.Progressive where

import Prelude hiding (read)
import Control.Exception (catch, IOException)
import Control.Monad (forM_, unless)
import Data.STRef (newSTRef, readSTRef, writeSTRef)
import Data.Bits (bit, shiftR)
import Data.Ord (comparing)
import Data.Word (Word8, Word16)

import Foreign.Storable (Storable(..))
import qualified Data.Vector.Storable.Mutable as SV
import Data.Vector.Storable (Vector, (!), createT)
import Data.Vector.Storable.Mutable (IOVector, new, read, write, slice)
import Data.Vector.Algorithms.Merge (sortBy)
import Data.Vector.Storable.MMap (writeMMapVector, unsafeMMapVector)
import System.Directory (createDirectoryIfMissing)
import System.Environment.XDG.BaseDir (getUserCacheDir)
import System.FilePath ((</>))

import Fractal.EscapeTime.Runtime.Render.Buffer (Buffer(Buffer))

newtype Pixel = Pixel (Word16, Word16, Word8, Word8)

instance Storable Pixel where
  sizeOf _ = 6
  alignment _ = 2
  peek p = do
    i <- peekByteOff p 0
    j <- peekByteOff p 2
    w <- peekByteOff p 4
    h <- peekByteOff p 5
    return $ Pixel (i, j, w, h)
  poke p (Pixel (i, j, w, h)) = do
    pokeByteOff p 0 i
    pokeByteOff p 2 j
    pokeByteOff p 4 w
    pokeByteOff p 5 h

type Progressive = Vector Pixel

calculateProgressive :: Int -> Int -> Maybe Progressive
calculateProgressive width height
```

```
  | 0 < width  && width  <= 65536 &&
    0 < height && height <= 65536 = createT $ do
      v <- new (width * height)
      let fill i x0 y0 sx sy w h = do
            ix <- newSTRef i
            forM_ [y0, y0 + sy .. height - 1] $ \y -> do
              forM_ [x0, x0 + sx .. width - 1] $ \x -> do
                j <- readSTRef ix
                write v j (Pixel (fromIntegral x, fromIntegral y, fromIntegral w
                    , fromIntegral h))
                writeSTRef ix $! j + 1
            readSTRef ix
          sort i j = sortBy (comparing distance) (slice i (j - i) v)
          cx = fromIntegral width / 2
          cy = fromIntegral height / 2
          distance :: Pixel -> Double
          distance (Pixel (x, y, _, _)) =
            let dx = fromIntegral x - cx
                dy = fromIntegral y - cy
            in  dx * dx + dy * dy
          bit7 = bit 7
      i <- pure 0
      j <- fill i 0 0 bit7 bit7 bit7 bit7
      sort i j
      ix <- newSTRef j
      forM_ (map bit [7, 6 .. 1]) $ \step -> do
        let step1 = step `shiftR` 1 :: Int
        i <- readSTRef ix
        j <- fill i 0 step1 step step step1 step
        sort i j
        k <- fill j step1 0 step step1 step1 step1
        sort j k
        writeSTRef ix k
      n <- readSTRef ix
      return $ if n == width * height then Just v else Nothing
  | otherwise = Nothing

cacheProgressive :: Int -> Int -> IO (Maybe Progressive)
cacheProgressive width height = do
  cacheDir <- getUserCacheDir "et"
  let progDir = cacheDir </> "progressive"
  createDirectoryIfMissing True progDir
  let filePath = progDir </> (show width ++ "x" ++ show height)
      readVector = Just <$> unsafeMMapVector filePath Nothing
      catchIO a b = do
        let f :: IOException -> IO (Maybe Progressive)
            f _ = b
        a `catch` f
  readVector `catchIO` do
    case calculateProgressive width height of
      Nothing -> return Nothing
      Just v -> do
        writeMMapVector filePath v
        readVector `catchIO` return (Just v)

output :: Maybe (Vector Pixel) -> IOVector Bool -> Buffer Float -> Int -> Bool
    -> (Float, Float, Float, Float) -> IO ()
```

```
output (Just prog) done (Buffer width height channels@4 out) npixel d0 (o0, o1, ↙
    ↳ o2, o3) = do
  let Pixel (i, j, w, h) = prog ! npixel
      ii = fromIntegral i
      jj = fromIntegral j
  forM_ [jj .. ((jj + fromIntegral h) `min` height) - 1] $ \v -> do
    forM_ [ii .. ((ii + fromIntegral w) `min` width) - 1] $ \u -> do
      let kk = v * width + u
      d <- read done kk
      unless d $ do
        let k = channels * kk
        SV.write out (k + 0) o0
        SV.write out (k + 1) o1
        SV.write out (k + 2) o2
        SV.write out (k + 3) o3
  write done (jj * width + ii) d0
output Nothing done (Buffer width height channels@4 out) npixel d0 (o0, o1, o2, ↙
    ↳ o3) = do
  let kk = npixel
  d <- read done kk
  unless d $ do
    let k = channels * kk
    SV.write out (k + 0) o0
    SV.write out (k + 1) o1
    SV.write out (k + 2) o2
    SV.write out (k + 3) o3
  write done kk d0
output _ _ _ _ _ _ = fail "Fractal.EscapeTime.Runtime.Render.Progressive.output: ↙
    ↳  bad channel count"
```

66 src/Fractal/EscapeTime/Runtime/Render/Transform.hs

```
{-
et -- escape time fractals
Copyright (C) 2018 Claude Heiland-Allen

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU Affero General Public License as
published by the Free Software Foundation, either version 3 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 Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License
along with this program.  If not, see <https://www.gnu.org/licenses/>.
-}

module Fractal.EscapeTime.Runtime.Render.Transform
  ( Transform()
  , transform
  , getTransform
  , identity
  , rotation
  , scaleX
```

```
    , shearX
    , inverse
    , compose
    , apply
    , Point
    , fromPoints
    ) where

import Data.Word (Word32)
import Data.Bits (xor, shiftL, shiftR)

type Point = (Double, Double)

newtype Transform = Transform (Double, Double, Double, Double)

transform :: Double -> Double -> Double -> Double -> Maybe Transform
transform a b c d =
  let det = a * d - b * c
      s = recip . sqrt . abs $ det
      t@(sa, sb, sc, sd) = (s * a, s * b, s * c, s * d)
      f x = not (isNaN x) && not (isInfinite x)
  in  if all f [sa, sb, sc, sd] then Just (Transform t) else Nothing

getTransform :: Transform -> (Double, Double, Double, Double)
getTransform (Transform t) = t

identity :: Transform
identity = Transform (1, 0, 0, 1)

rotation :: Double -> Transform
rotation t = let c = cos t ; s = sin t in Transform (c, -s, s, c)

scaleX :: Double -> Maybe Transform
scaleX x = transform x 0 0 1

shearX :: Double -> Maybe Transform
shearX x = transform 1 x 0 1

inverse :: Transform -> Transform
inverse (Transform (a, b, c, d)) = Transform (d, -c, -b, a)

compose :: Transform -> Transform -> Transform
compose (Transform (a, b, c, d)) (Transform (x, y, z, w))
  = Transform (a * x + b * z, a * y + b * w, c * x + d * z, c * y + d * w)

apply :: Transform -> Point -> Point
apply (Transform (a, b, c, d)) (x, y) = (a * x + b * y, c * x + d * y)

fromPoints :: Point -> Point -> Point -> Maybe Transform
fromPoints (ox, oy) (ax, ay) (bx, by) = transform (ax - ox) (ay - oy) (bx - ox) ↙
    ↳ (by - oy)
```

67 src/Fractal/EscapeTime/Runtime/Render/View.hs

```
{-
et -- escape time fractals
Copyright (C) 2018 Claude Heiland-Allen
```

```
   This program is free software: you can redistribute it and/or modify
   it under the terms of the GNU Affero General Public License as
   published by the Free Software Foundation, either version 3 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 Affero General Public License for more details.

   You should have received a copy of the GNU Affero General Public License
   along with this program.  If not, see <https://www.gnu.org/licenses/>.
-}

{-# LANGUAGE DataKinds #-}

module Fractal.EscapeTime.Runtime.Render.View where

import System.IO.Unsafe (unsafePerformIO)
import Numeric.Rounded
import qualified Numeric.Rounded.Simple as S

data View = View{ centerX, centerY, radius :: S.Rounded }

readViewIO :: String -> String -> String -> IO View
readViewIO sx sy sr = do
  r <- readIO sr
  let prec = max 53 $ 53 - exponent r
  reifyPrecision prec $ \p -> do
    let readViewIO2 :: Precision p => Rounded TowardNearest p -> Rounded ↙
        ↪ TowardNearest p -> Rounded TowardNearest Double -> proxy p -> IO View
        readViewIO2 x' y' r' _ = return $ View (S.simplify x') (S.simplify y') (↙
            ↪ S.simplify r')
    x <- readIO sx
    y <- readIO sy
    readViewIO2 x y r p

defaultView :: View
defaultView = unsafePerformIO $ readViewIO "0" "0" "2"
```

68 src/LICENSE.md

```
### GNU AFFERO GENERAL PUBLIC LICENSE

Version 3, 19 November 2007

Copyright (C) 2007 Free Software Foundation, Inc.
<https://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this
license document, but changing it is not allowed.

### Preamble

The GNU Affero General Public License is a free, copyleft license for
software and other kinds of works, specifically designed to ensure
```

cooperation with the community in the case of network server software.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, our General Public Licenses are intended to guarantee your freedom to share and change all versions of a program--to make sure it remains free software for all its users.

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 them 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.

Developers that use our General Public Licenses protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License which gives you legal permission to copy, distribute and/or modify the software.

A secondary benefit of defending all users' freedom is that improvements made in alternate versions of the program, if they receive widespread use, become available for other developers to incorporate. Many developers of free software are heartened and encouraged by the resulting cooperation. However, in the case of software used on network servers, this result may fail to come about. The GNU General Public License permits making a modified version and letting the public access it on a server without ever releasing its source code to the public.

The GNU Affero General Public License is designed specifically to ensure that, in such cases, the modified source code becomes available to the community. It requires the operator of a network server to provide the source code of the modified version running there to the users of that server. Therefore, public use of a modified version, on a publicly accessible server, gives the public access to the source code of the modified version.

An older license, called the Affero General Public License and published by Affero, was designed to accomplish similar goals. This is a different license, not a version of the Affero GPL, but Affero has released a new version of the Affero GPL which permits relicensing under this license.

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

TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU Affero General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable

work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey 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; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

- a) The work must carry prominent notices stating that you modified it, and giving a relevant date.
- b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".
- c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.
- d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

- a) Convey the object code in, or embodied in, a physical product

(including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

- b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.
- c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.
- d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.
- e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User

Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

- a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
- b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal

Notices displayed by works containing it; or
- c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or
- d) Limiting the use for publicity purposes of names of licensors or authors of the material; or
- e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or
- f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under

this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free

patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If 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 convey a

covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Remote Network Interaction; Use with the GNU General Public License.

Notwithstanding any other provision of this License, if you modify the Program, your modified version must prominently offer all users interacting with it remotely through a computer network (if your version supports such interaction) an opportunity to receive the Corresponding Source of your version by providing access to the Corresponding Source from a network server at no charge, through some standard or customary means of facilitating copying of software. This Corresponding Source shall include the Corresponding Source for any work covered by version 3 of the GNU General Public License that is incorporated pursuant to the following paragraph.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the work with which it is combined will remain governed by version 3 of the GNU General Public License.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU Affero 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 that a certain numbered version of the GNU Affero General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU Affero General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU Affero General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

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.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS 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.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

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 state 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 Affero General Public License as published by the Free Software Foundation, either version 3 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 Affero General Public License for more details.

```
    You should have received a copy of the GNU Affero General Public License
    along with this program.  If not, see <https://www.gnu.org/licenses/>.

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

If your software can interact with users remotely through a computer
network, you should also make sure that it provides a way for users to
get its source. For example, if your program is a web application, its
interface could display a "Source" link that leads users to an archive
of the code. There are many ways you could offer source, and different
solutions will be better for different programs; see section 13 for
the specific requirements.

You should also get your employer (if you work as a programmer) or
school, if any, to sign a "copyright disclaimer" for the program, if
necessary. For more information on this, and how to apply and follow
the GNU AGPL, see <https://www.gnu.org/licenses/>.
```

69 src/wrap.c

```
/*
et -- escape time fractals
Copyright (C) 2018,2019 Claude Heiland-Allen

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU Affero General Public License as
published by the Free Software Foundation, either version 3 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 Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License
along with this program.  If not, see <https://www.gnu.org/licenses/>.
*/

#include "formula.h"

extern int et_wrap_plainl(f_plainl*z,int a,long double*b,long double*c,long ↙
    ↘ double*d,long double*e,long double*d_,long double*e_,long double*f,↙
    ↘ volatile int*g)
{
  return z(a,*b,c,*d,*e,*d_,*e_,f,g);
}

/*
extern int et_wrap_plainc(f_plainc*z,int a,compensated*b,compensated*c,↙
    ↘ compensated*d,compensated*e,compensated*d_,compensated*e_,compensated*f,↙
    ↘ volatile int*g)
{
  return z(a,*b,c,*d,*e,*d_,*e_,f,g);
}
```

```
extern int et_wrap_referencel(f_referencel*z,int a,long double*b,long double*c_,↙
    ↳ long double*d_,mpfr_t c,mpfr_t d,long double*e,volatile int*f)
{
  return z(a,*b,*c_,*d_,c,d,e,f);
}

extern int et_wrap_perturbationl(f_perturbationl*z,int a,int b,long double*c,↙
    ↳ long double*d,long double*e,long double*f,long double*e_,long double*f_,↙
    ↳ long double*g,long double*h,volatile int*i)
{
  return z(a,b,*c,d,*e,*f,*e_,*f_,g,h,i);
}
*/
```

70 unix/Fractal/EscapeTime/Runtime/Loader/Load.hs

```
{-
et -- escape time fractals
Copyright (C) 2018 Claude Heiland-Allen

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU Affero General Public License as
published by the Free Software Foundation, either version 3 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 Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License
along with this program.  If not, see <https://www.gnu.org/licenses/>.
-}

module Fractal.EscapeTime.Runtime.Loader.Load (load, libext) where

import Foreign (Ptr, castFunPtrToPtr, nullPtr)
import Control.Exception (catch, SomeException)
import System.Posix.DynamicLinker (dlopen, dlsym, dlerror, dlclose, RTLDFlags(↙
    ↳ RTLD_NOW))

import Fractal.EscapeTime.Runtime.Loader.Raw (S_Formula)

libext :: String
libext = ".so"

load :: FilePath -> IO (Either String (IO (), Ptr S_Formula))
load path = catchAny (do
  dl <- dlopen path [RTLD_NOW]
  p <- castFunPtrToPtr <$> dlsym dl "et"
  if p == nullPtr
  then Left <$> dlerror
  else return $ Right (dlclose dl, p)) (\e -> return (Left ("exception: " ++ ↙
      ↳ show e)))

catchAny :: IO a -> (SomeException -> IO a) -> IO a
catchAny = Control.Exception.catch
```

71 win32/Fractal/EscapeTime/Runtime/Loader/Load.hs

```
{-
et -- escape time fractals
Copyright (C) 2018 Claude Heiland-Allen

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU Affero General Public License as
published by the Free Software Foundation, either version 3 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 Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License
along with this program.  If not, see <https://www.gnu.org/licenses/>.
-}

module Fractal.EscapeTime.Runtime.Loader.Load (load, libext) where

import Foreign (Ptr, castPtr, nullPtr)
import Control.Exception (catch, SomeException)
import System.Win32 (loadLibrary, getProcAddress, freeLibrary, getLastError)

import Fractal.EscapeTime.Runtime.Loader.Raw (S_Formula)

libext :: String
libext = ".dll"

load :: FilePath -> IO (Either String (IO (), Ptr S_Formula))
load path = catchAny (do
  dl <- loadLibrary path
  p <- castPtr <$> getProcAddress dl "et"
  if p == nullPtr
  then do
    e <- getLastError
    return $ Left ("LoadLibrary/GetProcAddress error " ++ show e)
  else return $ Right (freeLibrary dl, p)) (\e -> return (Left ("exception: " ++ show e)))

catchAny :: IO a -> (SomeException -> IO a) -> IO a
catchAny = Control.Exception.catch
```

72 zoom/Makefile

```
zoom: zoom.c
	gcc -O3 -march=native -funroll-loops -o zoom zoom.c -lGL -lglut -lGLEW -lm -lGLU
```

73 zoom/zoom.c

```
/*
et -- escape time fractals
Copyright (C) 2018,2019 Claude Heiland-Allen
```

```
    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU Affero General Public License as
    published by the Free Software Foundation, either version 3 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 Affero General Public License for more details.

    You should have received a copy of the GNU Affero General Public License
    along with this program.  If not, see <https://www.gnu.org/licenses/>.

    this file is based on extra/zoom.c from:

    mightymandel -- GPU-based Mandelbrot Set explorer
    Copyright (C) 2012,2013,2014,2015 Claude Heiland-Allen
    License GPL3+ http://www.gnu.org/licenses/gpl.html
    */

    #include <assert.h>
    #include <math.h>
    #include <stdio.h>
    #include <stdlib.h>
    #include <string.h>

    #include <GL/glew.h>
    #include <GL/glut.h>

    // image sizes
    static int IWIDTH = 0;
    static int IHEIGHT = 0;
    static int OWIDTH = 0;
    static int OHEIGHT = 0;
    static int WWIDTH = 0;
    static int WHEIGHT = 0;

    // speed of zooming
    static int IFRAMES = 0;
    static int OFPS = 25;
    static double OLENGTH = 0;
    static double increment = 0;

    // motion blur
    static double shutter = 0.0;
    static int motion_count = 1;

    // image buffers
    static float *ibuffer[2] = { 0, 0 };
    static unsigned char *ybuffer = 0;

    static char oheader[1024];

    // variables
    static double phase = 0;
    static int which = 0;
```

```
// textures
enum texid {
  texi11a = 0,
  texi11b = 1,
  tex11a,
  texo,
  // total number of textures
  texCount
};
static GLuint tex[texCount];

// frame buffer
static GLuint fbo = 0;
static GLuint fbo2 = 0;

static int combine_prog  = 0;
static int combine_tex0  = 0;
static int combine_tex1  = 0;
static int combine_phase = 0;
static int combine_alpha = 0;

static int draw_prog     = 0;
static int draw_tex      = 0;
static int draw_inverse  = 0;

// read raw image data
static int read_image(void) {
  return 1 == fread(ibuffer[which], IHEIGHT * IWIDTH * sizeof(float), 1, stdin);
}

// upload an image to the GPU
static void upload_image(int t, int w, int h) {
  glActiveTexture(GL_TEXTURE0 + t);
  glTexSubImage2D(GL_TEXTURE_2D, 0, 0, 0, w, h, GL_RED, GL_FLOAT, ibuffer[which
      ↳ ]);
}

// download Y data from the GPU
static void download_y(void) {
  glActiveTexture(GL_TEXTURE0 + texo);
  glGetTexImage(GL_TEXTURE_2D, 0, GL_RED, GL_UNSIGNED_BYTE, ybuffer);
//  glReadPixels(0, 0, OWIDTH, OHEIGHT, GL_RED, GL_UNSIGNED_BYTE, ybuffer);
}

// write planar Y
static int write_image(void) {
  if (1 != fwrite("FRAME\n", 6, 1, stdout)) { return 0; }
  if (1 != fwrite(ybuffer, OWIDTH * OHEIGHT    , 1, stdout)) { return 0; }
  return 1;
}

static const char *combine_frag_src =
"#version 130\n"
"uniform sampler2D tex0;\n"
"uniform sampler2D tex1;\n"
"uniform float phase;\n"
```

```
    "uniform float alpha;\n"
    "\n"
    "void main() {\n"
120 "  vec2 de = vec2(texture(tex0, 0.5 * (gl_TexCoord[0].xy - vec2(0.5)) + vec2(0.5)).r, texture(tex1, gl_TexCoord[0].xy).r);\n"
    "  de *= pow(vec2(2.0), vec2(phase + 1.0, phase));\n"
    "  vec2 g = tanh(clamp(vec2(4.0) + log2(max(de, vec2(1e-10))), 0.0, 8.0));\n"
    "  float y = mix(g.x, g.y, phase);\n"
    "  gl_FragColor = vec4(vec3(y), alpha);\n"
125 "}\n"
    ;

    static const char *draw_frag_src =
    "#version 130\n"
130 "uniform sampler2D tex;\n"
    "uniform bool inverse;\n"
    "\n"
    "void main() {\n"
    "  vec4 c = texture(tex, gl_TexCoord[0].xy);\n"
135 "  gl_FragColor = inverse ? vec4(vec3(1.0) - c.rgb, c.a) : c;\n"
    "}\n"
    ;

    // display callback
140 static void display(void) {
      int w = IWIDTH;
      int h = IHEIGHT;
      int motion = 0;
      glViewport(0, 0, w, h);
145   glBindFramebuffer(GL_FRAMEBUFFER, fbo);
      glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, tex[tex11a], 0);
      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
      glUseProgramObjectARB(combine_prog);
      for (motion = 0; motion < motion_count; ++motion) {
150     double motion_alpha = 1.0 / (motion + 1.0);
        while (phase >= 1) {
          // read next image
          if (read_image()) {
            upload_image(which, w, h);
155         which = 1 - which;
            phase -= 1;
          } else {
            fprintf(stderr, "couldn't read image\n");
            exit(0);
160       }
        }
        if (motion <= shutter * motion_count) {
          // scale and blend
          glUniform1iARB(combine_tex0, which);
165       glUniform1iARB(combine_tex1, 1 - which);
          glUniform1fARB(combine_phase, phase);
          glUniform1fARB(combine_alpha, motion_alpha);
          double x = 0.5*pow(0.5, phase);
          double y = 0.5*pow(0.5, phase);
170       glBegin(GL_QUADS); {
            glTexCoord2f(0.5 - x, 0.5 + y); glVertex2f(0, 1);
```

```
          glTexCoord2f(0.5 + x, 0.5 + y); glVertex2f(1, 1);
          glTexCoord2f(0.5 + x, 0.5 - y); glVertex2f(1, 0);
          glTexCoord2f(0.5 - x, 0.5 - y); glVertex2f(0, 0);
        } glEnd();
      }
      // advance
      phase += increment / motion_count;
    }
    glUseProgramObjectARB(0);
    // download output
    glActiveTexture(GL_TEXTURE0 + tex11a);
    glGenerateMipmap(GL_TEXTURE_2D);
    glBindFramebuffer(GL_FRAMEBUFFER, fbo2);
    glViewport(0, 0, OWIDTH, OHEIGHT);
    glUseProgramObjectARB(draw_prog);
    glBegin(GL_QUADS); {
      glTexCoord2f(0, 1); glVertex2f(0, 1);
      glTexCoord2f(1, 1); glVertex2f(1, 1);
      glTexCoord2f(1, 0); glVertex2f(1, 0);
      glTexCoord2f(0, 0); glVertex2f(0, 0);
    } glEnd();
    glUseProgramObjectARB(0);
    glBindFramebuffer(GL_FRAMEBUFFER, 0);
    download_y();
    write_image();
    // display current output
    glBindFramebuffer(GL_READ_FRAMEBUFFER, fbo2);
    glBindFramebuffer(GL_DRAW_FRAMEBUFFER, 0);
    glViewport(0, 0, WWIDTH, WHEIGHT);
    glClear(GL_COLOR_BUFFER_BIT);
    glBlitFramebuffer(0, 0, OWIDTH, OHEIGHT, 0, 0, WWIDTH, WHEIGHT, ↙
        ↳ GL_COLOR_BUFFER_BIT, GL_LINEAR);
    glutSwapBuffers();
    glutReportErrors();
}

// allocate an RGB texture
static void textureb(int tid, int w, int h, int r) {
  glActiveTexture(GL_TEXTURE0 + tid);
  glBindTexture(GL_TEXTURE_2D, tex[tid]);
  glTexImage2D(GL_TEXTURE_2D, 0, GL_R32F, w, h, 0, GL_RED, GL_FLOAT, 0);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, r ? ↙
      ↳ GL_LINEAR_MIPMAP_LINEAR : GL_LINEAR);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
}

// initialize a shader
static int shader(const char *src) {
  int p = glCreateProgramObjectARB();
#if 0
  int v = glCreateShaderObjectARB(GL_VERTEX_SHADER_ARB);
  glShaderSourceARB(v, 1, (const GLcharARB **) &shader_vert_src, 0);
  glCompileShaderARB(v);
  glAttachObjectARB(p, v);
#endif
```

```
  int f = glCreateShaderObjectARB(GL_FRAGMENT_SHADER_ARB);
  glShaderSourceARB(f, 1, (const GLcharARB **) &src, 0);
  glCompileShaderARB(f);
  glAttachObjectARB(p, f);
  glLinkProgramARB(p);
  int success;
  glGetObjectParameterivARB(p, GL_OBJECT_LINK_STATUS_ARB, &success);
  if (!success) { fprintf(stderr, "failed to compile shader\n%s\n", src); exit(1); }
  return p;
}

// GLUT timer callback
static void timerf(int x) {
  glutTimerFunc(10, timerf, x+1);
  glutPostRedisplay();
}

// main program
extern int main(int argc, char **argv) {
  if (! (argc >= 5)) {
    fprintf(stderr, "usage: %s iwidth iheight iframes olength [ofps [oshutter [header [inverse]]]] < stream.ppm > stream.y4m\n", argv[0]);
    return 1;
  }
  IWIDTH = atoi(argv[1]);
  IHEIGHT = atoi(argv[2]);
  IFRAMES = atoi(argv[3]);
  OWIDTH = 1920;
  OHEIGHT = 1080;
  OLENGTH = atof(argv[4]);
  OFPS = 25;
  if (argc >= 6) {
    OFPS = atoi(argv[5]);
  }
  if (argc >= 7) {
    shutter = fmin(fmax(atof(argv[6]), 0.0), 1.0);
  }
  int header = 1;
  if (argc >= 8) {
    header = atoi(argv[7]);
  }
  int inverse = 0;
  if (argc >= 9) {
    inverse = atoi(argv[8]);
  }
  increment = (IFRAMES - 1.0) / (OLENGTH * OFPS);
  /*
  zoom0 = 0.5 ** phase
  zoom1 = 0.5 ** (phase + increment / motion_count)
  texcoord = (width/2 + width/2 * zoom , height/2 + height/2 * zoom)
  assume phase = 0
  texcoord0 = (width, height)
  texcoord1 = (width, height) * (1 + 0.5 ** (increment / motion_count)) / 2
  texcoord1 - texcoord2 = (width, height) * [(1 + 0.5 ** (increment / motion_count)) / 2 - 1]
  | tex1 - tex2 | = sqrt(w^2+h^2) * [(1 + 0.5 ** (inc / mot)) / 2 - 1]
```

```
    | tex1 - tex2 | == 1  for smooth motion blur
    2*[1/sqrt(w^2+h^2) + 1] - 1 = 0.5 ** (inc / mot)
    2/sqrt(w^2 + h^2) + 1 = 0.5**(inc/mot)
    log (2 / sqrt (w^2 + h^2) + 1) = (inc / mot) log 0.5
    mot = inc * log 0.5 / log (2 / sqrt(w^2 + h^2) + 1)
    */
    motion_count = fmax(1.0, ceil(fabs(increment * log(0.5) / log(2.0 / sqrt(↙
        ↳ OWIDTH * OWIDTH + OHEIGHT * OHEIGHT) + 1.0))));
    fprintf(stderr, "zoom: motion_count = %d (%d)\n", motion_count, (int) ceil(↙
        ↳ shutter * motion_count));
    ibuffer[0] = calloc(1, IWIDTH * IHEIGHT * sizeof(float));
    ibuffer[1] = calloc(1, IWIDTH * IHEIGHT * sizeof(float));
    ybuffer = malloc(OWIDTH * OHEIGHT);
    snprintf(oheader, 1024, "YUV4MPEG2 W%d H%d F%d:1 Ip A1:1 Cmono\n", OWIDTH, ↙
        ↳ OHEIGHT, OFPS);
    // initialize
    double aspect = OWIDTH / (double) OHEIGHT;
    WWIDTH = 640;
    WHEIGHT = 640 / aspect;
    glutInitWindowSize(WWIDTH, WHEIGHT);
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE);
    glutCreateWindow("zoom");
    glewInit();
    // shaders
    combine_prog  = shader(combine_frag_src);
    combine_tex0  = glGetUniformLocationARB(combine_prog, "tex0");
    combine_tex1  = glGetUniformLocationARB(combine_prog, "tex1");
    combine_phase = glGetUniformLocationARB(combine_prog, "phase");
    combine_alpha = glGetUniformLocationARB(combine_prog, "alpha");
    draw_prog     = shader(draw_frag_src);
    draw_tex      = glGetUniformLocationARB(draw_prog, "tex");
    draw_inverse  = glGetUniformLocationARB(draw_prog, "inverse");
    glUseProgramObjectARB(draw_prog);
    glUniform1iARB(draw_tex, tex11a);
    glUniform1iARB(draw_inverse, inverse);
    glUseProgramObjectARB(0);
    // universal view
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0, 1, 0, 1);
    // textures
    glGenTextures(texCount, tex);
    int w = IWIDTH;
    int h = IHEIGHT;
    textureb(texi11a, w * 1, h * 1, 0);
    textureb(texi11b, w * 1, h * 1, 0);
    textureb(tex11a, w * 1, h * 1, 1);
    textureb(texo, OWIDTH, OHEIGHT, 0);
    // frame buffer
    glGenFramebuffers(1, &fbo);
    glGenFramebuffers(1, &fbo2);
    glBindFramebuffer(GL_FRAMEBUFFER, fbo2);
    glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, ↙
        ↳ tex[texo], 0);
```

```
  glBindFramebuffer(GL_FRAMEBUFFER, 0);
  // initialize state
  which = 0;
  phase = 2;
  // write output header
  if (header)
  {
    if (1 != fwrite(oheader, strlen(oheader), 1, stdout)) { fprintf(stderr, "
        ↳ failed to write header\n"); return 1; }
  }
  // start processing frames
  glutDisplayFunc(display);
  glutTimerFunc(10, timerf, 0);
  glutReportErrors();
  glutMainLoop();
  return 0;
}
```