gruff-examples

Claude Heiland-Allen

2011–2016

Contents

1 .gitignore

```
dist
-
```

2 gruff-examples.cabal

```
Name:                gruff-examples
Version:             0.4
Synopsis:            Mandelbrot Set examples using ruff and gruff
Description:
  Some example scripts, including a converter from old versions of gruff
  file formats to the current file format.

License:             GPL-2
License-file:        LICENSE
Author:              Claude Heiland-Allen
Maintainer:          claude@mathr.co.uk
Category:            Graphics

Build-type:          Simple
Cabal-version:       >=1.6
```

```
Flag mpfr
  description: use 'hmpfr' for higher precision floating point
  default: False

Executable gruff-convert
  Hs-source-dirs:       src
  Main-is:              gruff-convert.hs
  Other-modules:        Convert.Common
                        Convert.Gruff
                        Convert.Gruff1
                        Convert.Gruff2a
                        Convert.Gruff2
  Build-depends:        base >= 4 && < 5,
                        filepath,
                        ruff >= 0.4 && < 0.5,
                        gruff >= 0.4 && < 0.5
  GHC-options:          -Wall -threaded -rtsopts

Executable gruff-labels
  Hs-source-dirs:       src
  Main-is:              gruff-labels.hs
  Build-depends:        base >= 4 && < 5,
                        containers,
                        gruff >= 0.4 && < 0.5,
                        ruff >= 0.4 && < 0.5
  GHC-options:          -Wall -threaded -rtsopts

Executable gruff-octopus
  Hs-source-dirs:       src
  Main-is:              gruff-octopus.hs
  Other-modules:        Number
  Build-depends:        base >= 4 && < 5,
                        gruff >= 0.4 && < 0.5,
                        ruff >= 0.4 && < 0.5,
                        qd >= 1 && < 2,
                        qd-vec >= 1 && < 2,
                        Vec >= 1 && < 2
  if (flag(mpfr))
    Build-depends:      hmpfr >= 0.3.2 && < 0.4
    CPP-options:        -DHAVE_MPFR
    CC-options:         -DHAVE_MPFR
  GHC-options:          -Wall -threaded -rtsopts

Executable gruff-fives
  Hs-source-dirs:       src
  Main-is:              gruff-fives.hs
  Other-modules:        Number
  Build-depends:        base >= 4 && < 5,
                        gruff >= 0.4 && < 0.5,
                        ruff >= 0.4 && < 0.5,
                        qd >= 1 && < 2,
                        qd-vec >= 1 && < 2,
                        Vec >= 1 && < 2
  if (flag(mpfr))
    Build-depends:      hmpfr >= 0.3.2 && < 0.4
    CPP-options:        -DHAVE_MPFR
    CC-options:           -DHAVE_MPFR
  GHC-options:            -Wall -threaded -rtsopts

Executable gruff-patterns
  Hs-source-dirs:         src
  Main-is:                gruff-patterns.hs
  Other-modules:          Number
  Build-depends:          base >= 4 && < 5,
                          gruff >= 0.4 && < 0.5,
                          ruff >= 0.4 && < 0.5,
                          qd >= 1 && < 2,
                          qd-vec >= 1 && < 2,
                          Vec >= 1 && < 2
  if (flag(mpfr))
    Build-depends:        hmpfr >= 0.3.2 && < 0.4
    CPP-options:          -DHAVE_MPFR
    CC-options:           -DHAVE_MPFR
  GHC-options:            -Wall -threaded -rtsopts

Executable gruff-randoms
  Hs-source-dirs:         src
  Main-is:                gruff-randoms.hs
  Other-modules:          Number
  Build-depends:          base >= 4 && < 5,
                          gruff >= 0.4 && < 0.5,
                          ruff >= 0.4 && < 0.5,
                          qd >= 1 && < 2,
                          qd-vec >= 1 && < 2,
                          Vec >= 1 && < 2,
                          random >= 1.0 && < 1.1
  if (flag(mpfr))
    Build-depends:        hmpfr >= 0.3.2 && < 0.4
    CPP-options:          -DHAVE_MPFR
    CC-options:           -DHAVE_MPFR
  GHC-options:            -Wall -threaded -rtsopts

Executable gruff-whn
  Hs-source-dirs:         src
  Main-is:                gruff-whn.hs
  Other-modules:          Number
  Build-depends:          base >= 4 && < 5,
                          gruff >= 0.4 && < 0.5,
                          ruff >= 0.4 && < 0.5,
                          qd >= 1 && < 2,
                          qd-vec >= 1 && < 2,
                          Vec >= 1 && < 2,
                          data-memocombinators >= 0.4 && < 0.5
  if (flag(mpfr))
    Build-depends:        hmpfr >= 0.3.2 && < 0.4
    CPP-options:          -DHAVE_MPFR
    CC-options:           -DHAVE_MPFR
  GHC-options:            -Wall -threaded -rtsopts

Executable gruff-zoom
  Hs-source-dirs:         src
  Main-is:                gruff-zoom.hs
  Build-depends:          base >= 4 && < 5,
                      gruff >= 0.4 && < 0.5
  GHC-options:        -Wall -threaded -rtsopts


source-repository head
  type:     git
  location: http://code.mathr.co.uk/gruff-examples.git

source-repository this
  type:     git
  location: http://code.mathr.co.uk/gruff-examples.git
  tag:      v0.4
```

3 LICENSE

```
            GNU GENERAL PUBLIC LICENSE
               Version 2, June 1991

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

                    Preamble

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

  When we speak of free software, we are referring to freedom, not
price.  Our General Public Licenses are designed to make sure that you
have the freedom to distribute copies of free software (and charge for
this service if you wish), that you receive source code or can get it
if you want it, that you can change the software or use pieces of it
in new free programs; and that you know you can do these things.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

The source code for a work means the preferred form of the work for
making modifications to it.  For an executable work, complete source
```

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

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

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

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

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

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

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

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

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

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

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

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

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

NO WARRANTY

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

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING

```
WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR
REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES,
INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING
OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED
TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY
YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER
PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE
POSSIBILITY OF SUCH DAMAGES.

                     END OF TERMS AND CONDITIONS

        How to Apply These Terms to Your New Programs

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

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

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

    This program is free software; you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation; either version 2 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License along
    with this program; if not, write to the Free Software Foundation, Inc.,
    51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.

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

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

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

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

You should also get your employer (if you work as a programmer) or your
school, if any, to sign a "copyright disclaimer" for the program, if
necessary.  Here is a sample; alter the names:
  Yoyodyne, Inc., hereby disclaims all copyright interest in the program
  `Gnomovision' (which makes passes at compilers) written by James Hacker.

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

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

4 Setup.hs

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

5 src/Convert/Common.hs

```haskell
module Convert.Common (readMay, image, image', Colour(Colour), Image()) where
import Data.List (sortBy)
import Data.Ord (comparing)
import Fractal.GRUFF
import Fractal.RUFF.Types.Complex(Complex((:+)))

image :: Rational -> Rational -> Double -> Image
image = image' 512 288 4 (red, black, white)
  where
    red   = Colour 1 0 0
    black = Colour 0 0 0
    white = Colour 1 1 1

image' :: Int -> Int -> Double -> (Colour, Colour, Colour) -> Rational -> ↙
    ↳ Rational -> Double -> Image
image' w h ss (ci, cb, ce) re im sz = Image
  { imageWindow = Window{ width = w, height = h, supersamples = ss }
  , imageViewport = Viewport{ aspect = fromIntegral w / fromIntegral h, orient =↙
      ↳ 0 }
  , imageLocation = Location{ center = re :+ im, radius = sz }
  , imageColours = Colours{ colourInterior = ci, colourBoundary = cb, ↙
      ↳ colourExterior = ce }
  , imageLabels = []
  , imageLines = []
  }

readMay :: Read a => String -> Maybe a
readMay s = case sortBy (comparing (length . snd)) . filter (all whiteSpace . ↙
    ↳ snd) . reads $ s of
  (a, _):_ -> Just a
  _ -> Nothing
  where
    whiteSpace ' '  = True
    whiteSpace '\t' = True
    whiteSpace '\n' = True
    whiteSpace '\r' = True
    whiteSpace  _   = False
```

6 src/Convert/Gruff1.hs

```
module Convert.Gruff1 (gruff1) where
import Convert.Common (readMay, image, Image)
import Numeric (readSigned, readFloat)

data AngledInternalAddress
  = Unangled Integer
  | Angled Integer Angle AngledInternalAddress
  deriving Read

type Angle = Rational

newtype R = R Rational

instance Read R where
  readsPrec _ = map (\(x, s) -> (R x, s)) . readParen False (readSigned readFloat)

data Gruff1 = Gruff1
  { gAddress :: Maybe AngledInternalAddress
  , gIsland :: Maybe AngledInternalAddress
  , gChild :: Maybe [Angle]
  , gLowerAngle :: Maybe Angle
  , gUpperAngle :: Maybe Angle
  , gReal :: Maybe R
  , gImag :: Maybe R
  , gSize :: Maybe R
  , gHueShift :: Maybe R
  , gHueScale :: Maybe R
  }
  deriving Read

gruff1 :: String -> Maybe Image
gruff1 s = convert =<< readMay s

convert :: Gruff1 -> Maybe Image
convert Gruff1{ gReal = Just (R re), gImag = Just (R im), gSize = Just (R sz) } = Just $ image re im (fromRational sz)
convert _ = Nothing
```

7 src/Convert/Gruff2a.hs

```
module Convert.Gruff2a (gruff2a) where
import Convert.Common (readMay, image', Colour(Colour), Image())
import Numeric (readSigned, readFloat)

data AngledInternalAddress
  = Unangled Integer
  | Angled Integer Angle AngledInternalAddress
  deriving Read

type Angle = Rational

newtype R = R Rational
```

```haskell
instance Read R where
  readsPrec _ = map (\(x, s) -> (R x, s)) . readParen False (readSigned readFloat)

data Color = Color Int Int Int
  deriving Read

c :: Color -> Colour
c (Color r g b) = Colour (fromIntegral r / m) (fromIntegral g / m) (fromIntegral b / m) where m = 65535

data Gruff2a = Gruff2
  { gAddress :: Maybe AngledInternalAddress
  , gReal :: Maybe R
  , gImag :: Maybe R
  , gSize :: Maybe R
  , gRota :: Maybe Double
  , gColours :: (Color, Color, Color)
  }
  deriving Read

gruff2a :: String -> Maybe Image
gruff2a s = convert =<< readMay s

convert :: Gruff2a -> Maybe Image -- FIXME handle rota?
convert Gruff2
  { gReal = Just (R re), gImag = Just (R im), gSize = Just (R sz)
  , gColours = (ci, cb, ce)
  } = Just $ image' 512 288 4 (c ci, c cb, c ce) re im (fromRational sz)
convert _ = Nothing
```

8 src/Convert/Gruff2.hs

```haskell
module Convert.Gruff2 (gruff2) where
import Convert.Common (readMay, image', Colour(Colour), Image())
import Numeric (readSigned, readFloat)

data AngledInternalAddress
  = Unangled Integer
  | Angled Integer Angle AngledInternalAddress
  deriving Read

type Angle = Rational

newtype R = R Rational

instance Read R where
  readsPrec _ = map (\(x, s) -> (R x, s)) . readParen False (readSigned readFloat)

data Color = Color Int Int Int
  deriving Read

c :: Color -> Colour
c (Color r g b) = Colour (fromIntegral r / m) (fromIntegral g / m) (fromIntegral b / m) where m = 65535
```

```haskell
data Window = Window{ width :: Int, height :: Int, supersamples :: R }
  deriving Read

data Viewport = Viewport{ aspect :: R, orient :: R }
  deriving Read

data Gruff2 = Gruff2
  { gAddress :: Maybe AngledInternalAddress
  , gReal :: Maybe R
  , gImag :: Maybe R
  , gSize :: Maybe R
  , gRota :: Maybe Double
  , gColours :: (Color, Color, Color)
  , gWindow :: Window
  , gViewport :: Viewport
  }
  deriving Read

gruff2 :: String -> Maybe Image
gruff2 s = convert =<< readMay s

convert :: Gruff2 -> Maybe Image -- FIXME handle rota, orient, aspect?
convert Gruff2
  { gReal = Just (R re), gImag = Just (R im), gSize = Just (R sz)
  , gColours = (ci, cb, ce)
  , gWindow = Window{ width = w, height = h, supersamples = R ss }
  } = Just $ image' w h (fromRational ss) (c ci, c cb, c ce) re im (fromRational sz)
convert _ = Nothing
```

9 src/Convert/Gruff.hs

```haskell
module Convert.Gruff (gruff) where
import Convert.Common (readMay, image, Image)
import Numeric (readSigned, readFloat)

data AngledInternalAddress
  = Unangled Integer
  | Angled Integer Angle AngledInternalAddress
  deriving Read

type Angle = Rational

newtype R = R Rational

instance Read R where
  readsPrec _ = map (\(x, s) -> (R x, s)) . readParen False (readSigned readFloat)

data Gruff = Gruff
  { gAddress :: Maybe AngledInternalAddress
  , gIsland :: Maybe AngledInternalAddress
  , gChild :: Maybe [Angle]
  , gLowerAngle :: Maybe Angle
  , gUpperAngle :: Maybe Angle
  , gReal :: Maybe R
  , gImag :: Maybe R
  , gSize :: Maybe R
  }
  deriving Read

gruff :: String -> Maybe Image
gruff s = convert =<< readMay s

convert :: Gruff -> Maybe Image
convert Gruff{ gReal = Just (R re), gImag = Just (R im), gSize = Just (R sz) } =↙
    ↳ Just $ image re im (fromRational sz)
convert _ = Nothing
```

10 src/gruff-convert.hs

```haskell
import Data.Maybe (mapMaybe)
import System.Environment (getArgs)
import System.FilePath ((</>), takeFileName)

import Convert.Common (Image)
import Convert.Gruff  (gruff)
import Convert.Gruff1 (gruff1)
import Convert.Gruff2a(gruff2a)
import Convert.Gruff2 (gruff2)

parsers :: [String -> Maybe Image]
parsers = [gruff2, gruff2a, gruff1, gruff]

main :: IO ()
main = do
  args <- getArgs
  case args of
    odir:files@(_:_) -> mapM_ (main1 odir) files
    _ -> putStrLn "usage: gruff-convert outdir/ *.oldformat.gruff"

main1 :: FilePath -> FilePath -> IO ()
main1 odir file = do
  old <- readFile file
  case mapMaybe ($ old) parsers of
    []    -> putStrLn $ "Error: `" ++ file ++ "' unrecognised"
    [new] -> writeFile (odir </> takeFileName file) (show new)
    _     -> putStrLn $ "Error: `" ++ file ++ "' ambiguous"
```

11 src/gruff-fives.hs

```haskell
import Data.Maybe (mapMaybe)

import Fractal.GRUFF

import Fractal.RUFF.Mandelbrot.Address (parseAngledInternalAddress)
import Fractal.RUFF.Mandelbrot.Atom (MuAtom(..), findAtom_)
import Fractal.RUFF.Types.Complex (Complex((:+)))

import Number (R)

main :: IO ()
main = defaultMain animation
```

```
animation :: [(Image, FilePath)]
animation = mapMaybe scene score

scene :: String -> Maybe (Image, FilePath)
scene s = do
  m <- findAtom_ =<< parseAngledInternalAddress s
  let cx :+ cy = muNucleus m :: Complex R
      f = filename s
      i = Image
            { imageLocation = Location
                { center = toRational cx :+ toRational cy
                , radius = muSize m * 16
                }
            , imageViewport = Viewport
                { aspect = 1
                , orient = muOrient m - pi / 2
                }
            , imageWindow   = Window
                { width = 512
                , height = 288
                , supersamples = 8
                }
            , imageColours  = Colours
                { colourInterior = Colour 1 0.75 0
                , colourBoundary = Colour 0 0 0
                , colourExterior = Colour 1 1 1
                }
            , imageLabels   = []
            , imageLines    = []
            }
  return (i, f)

filename :: String -> FilePath
filename s = map filechar s ++ ".ppm"
  where
    filechar ' ' = '_'
    filechar '/' = '-'
    filechar  c  =  c

score :: [String]
score =
  [ "1 1/5 5 6 " ++ accum deltas
  | l <- [0 .. 124]
  , let m = l `div` 5
  , let n = l `mod` 5 + 1
  , let deltas = replicate m 5 ++ [n]
  ]
  where accum = unwords . map show . scanl (+) 11
```

12 src/gruff-labels.hs

```
import Fractal.GRUFF

import Fractal.RUFF.Mandelbrot.Address hiding (angles)
import Fractal.RUFF.Mandelbrot.Ray
import Fractal.RUFF.Mandelbrot.Atom
import Fractal.RUFF.Types.Complex
import Fractal.RUFF.Types.Ratio hiding (Rational)

import Control.Monad (ap, replicateM)
import qualified Data.Map as M
import Data.Maybe (mapMaybe)

main :: IO ()
main = defaultMain [(labels, ”gruff-labels.ppm”)]

labels :: Image
labels = Image
  { imageWindow   = Window{ width = 1280, height = 720, supersamples = 8 }
  , imageViewport = Viewport{ aspect =16/9, orient = 0 }
  , imageLocation = Location{ center = (-0.1178) :+ 1.0413, radius = 0.125 }
  , imageColours  = Colours
      { colourInterior = red
      , colourBoundary = black
      , colourExterior = darkgrey
      }
  , imageLabels   = rayLabels ++ atomLabels
  , imageLines    = rayLines
  }

addressSpec :: [String]
addressSpec =
  [ ”1 1/3 ” ++ (unwords . map show . scanl (+) (3 :: Int)) steps
  | n <- [1 .. 3]
  , steps <- replicateM n [1,2,3]
  ]

addresses :: [AngledInternalAddress]
addresses = mapMaybe parseAngledInternalAddress addressSpec

atoms :: [(AngledInternalAddress, MuAtom Double)]
atoms = mapMaybe (\addr -> fmap ((,) addr) (findAtom_ addr)) addresses

angles :: [(AngledInternalAddress, MuAtom Double, Angle, Angle)]
angles = mapMaybe f atoms
  where
    f (addr, mu) = fmap g (externalAngles addr)
      where
        g (lo, hi) = (addr, mu, lo, hi)

atomLabels :: [Label]
atomLabels =
  [ Label
    { labelCoords = toRationalC (muNucleus mu)
    , labelColour = white
    , labelText = prettyAngledInternalAddress addr
    }
  | (addr, mu, _, _) <- angles
  ]

rayLabels :: [Label]
rayLabels =
  [ Label
```

```
    { labelCoords = fst . (!! (sharpness * k)) $ rays M.! a
    , labelColour = lightgrey , labelText = prettyAngle a
    }
  | (_, _, lo, hi) <- angles
  , (a, k) <- [(lo, 10), (hi, 11)]
  ]

rayLines :: [Line]
rayLines =
  [ Line{ lineSegments = rs, lineColour = midgrey }
  | rs <- M.elems rays
  ]

rays :: M.Map Angle [(Complex Rational, Complex Rational)]
rays = M.fromList [ (a, ray a) | (_, _, lo, hi) <- angles, a <- [lo, hi] ]

ray :: Angle -> [(Complex Rational, Complex Rational)]
ray = (zip `ap` tail)
    . map toRationalC
    . take (sharpness * 32)
    . externalRay 1e-8 sharpness (2**24)
    . (\t -> toRational (numerator t) / toRational (denominator t))

sharpness :: Int
sharpness = 8

red, black, darkgrey, midgrey, lightgrey, white :: Colour
red = Colour 1 0 0
black = Colour 0 0 0
darkgrey = Colour 0.25 0.25 0.25
midgrey = Colour 0.5 0.5 0.5
lightgrey = Colour 0.75 0.75 0.75
white = Colour 1 1 1

toRationalC :: Complex Double -> Complex Rational
toRationalC (x :+ y) = toRational x :+ toRational y
```

13 src/gruff-octopus.hs

```
import Data.Maybe (mapMaybe)
import Numeric.QD (QuadDouble)
import Numeric.QD.Vec ()

import Fractal.GRUFF

import Fractal.RUFF.Mandelbrot.Address (parseAngledInternalAddress)
import Fractal.RUFF.Mandelbrot.Atom (MuAtom(..), findAtom_)
import Fractal.RUFF.Types.Complex (Complex((:+)))

main :: IO ()
main = defaultMain animation

animation :: [(Image, FilePath)]
animation = mapMaybe scene (score `zip` [0..])

scene :: (String, Int) -> Maybe (Image, FilePath)
scene (s, n) = do
```

```
  m <- findAtom_ =<< parseAngledInternalAddress s
  let cx :+ cy = muNucleus m :: Complex QuadDouble
      f = filename n
      i = Image
            { imageLocation = Location
                { center = toRational cx :+ toRational cy
                , radius = muSize m * 16
                }
            , imageViewport = Viewport
                { aspect = 1
                , orient = muOrient m - pi / 2
                }
            , imageWindow   = Window
                { width = 256
                , height = 256
                , supersamples = 8
                }
            , imageColours  = Colours
                { colourInterior = Colour 1 0 0
                , colourBoundary = Colour 0 0 0
                , colourExterior = Colour 1 1 1
                }
            , imageLabels   = []
            , imageLines    = []
            }
  return (i, f)

filename :: Int -> FilePath
filename n = (reverse . take 2 . (++ "00") . reverse . show) n ++ ".ppm"

score :: [String]
score =
  [ "1 " ++ show k ++ "/29 " ++ (unwords . map show) [ 30 .. 38 :: Int ]
  | k <- [ 1 .. 28 ] ++ [ 27, 26 .. 2 :: Int ]
  ]
```

14 src/gruff-patterns.hs

```
import Control.Monad (replicateM)
import Data.Maybe (mapMaybe)
import Data.Ratio (numerator, denominator, (%))
import System.Environment (getArgs)

import Fractal.GRUFF

import Fractal.RUFF.Mandelbrot.Address (parseAngledInternalAddress)
import Fractal.RUFF.Mandelbrot.Atom (MuAtom(..), findAtom_)
import Fractal.RUFF.Types.Complex (Complex((:+)))

import Number (R)

main :: IO ()
main = do
  args <- getArgs
  case map reads args of
    [[(num, "")], [(den, "")], [(depth, "")]] ->
      defaultMain (animation (num % den) (fromIntegral depth))
    _ -> putStrLn "usage: gruff-patterns num den depth | gruff"

animation :: Rational -> Int -> [(Image, FilePath)]
animation r d = mapMaybe scene (score r d)

scene :: String -> Maybe (Image, FilePath)
scene s = do
  m <- findAtom_ =<< parseAngledInternalAddress s
  let cx :+ cy = muNucleus m :: Complex R
      f = filename s
      i = Image
            { imageLocation = Location
                { center = toRational cx :+ toRational cy
                , radius = muSize m * 16
                }
            , imageViewport = Viewport
                { aspect = 1
                , orient = muOrient m - pi / 2
                }
            , imageWindow   = Window
                { width = 256
                , height = 256
                , supersamples = 16
                }
            , imageColours  = Colours
                { colourInterior = Colour 1 0 0
                , colourBoundary = Colour 0 0 0
                , colourExterior = Colour 1 1 1
                }
            , imageLabels   = []
            , imageLines    = []
            }
  return (i, f)

filename :: String -> FilePath
filename s = map filechar s ++ ".ppm"
  where
    filechar ' ' = '_'
    filechar '/' = '-'
    filechar  c  =  c

score :: Rational -> Int -> [String]
score r n =
  [ "1 " ++ nr ++ "/" ++ dr ++ " " ++ accum deltas
  | deltas <- replicateM n [1 .. denominator r - 1]
  ]
  where
    nr = show (numerator r)
    dr = show (denominator r)
    accum = unwords . map show . scanl (+) (denominator r)
```

15 src/gruff-randoms.hs

```
import GHC.Conc (numCapabilities)
import Control.Concurrent (forkIO, Chan, newChan, getChanContents, writeChan)
import Control.Monad (forM_)
import Data.Function (on)
```

```haskell
import Data.List (nub, nubBy, unfoldr)
import System.Environment (getArgs)
import System.Random (newStdGen, RandomGen, random, randomR, split)

import Fractal.GRUFF

import Fractal.RUFF.Mandelbrot.Address
  ( AngledInternalAddress(..), angledToList, angledFromList
  , prettyAngledInternalAddress
  )
import Fractal.RUFF.Mandelbrot.Atom (MuAtom(..), findAtom_)
import Fractal.RUFF.Types.Complex (Complex((:+)))
import Fractal.RUFF.Types.Ratio (denominator, (%))

import Number (R)

main :: IO ()
main = do
  args <- getArgs
  case args of
    [ns] -> case reads ns of
      [(n, "")] -> do
        gs <- unfoldr (Just . split) `fmap` newStdGen
        ch <- newChan
        forM_ ([0..] `zip` take numCapabilities gs) $ forkIO . worker ch
        let unique = nubBy ((==) `on` snd)
            f ((i, _), a) = (i, toFileName (prettyAngledInternalAddress a))
        defaultMain . take n . map f . unique =<< getChanContents ch
      _ -> usage
    _ -> usage
  where
    usage = putStrLn "usage: gruff-randoms count | gruff"

toFileName :: String -> String
toFileName = (++ ".ppm") . map toFileChar

toFileChar :: Char -> Char
toFileChar '/' = '-'
toFileChar ' ' = '_'
toFileChar  c  =  c

type Message = ((Image, FilePath), AngledInternalAddress)

worker :: RandomGen g => Chan Message -> (Int, g) -> IO ()
worker ch (w, g) =
  mapM_ (uncurry $ work ch w) . zip [0..] . nub . randomAddresses $ g

work :: Chan Message -> Int -> Int -> AngledInternalAddress -> IO ()
work ch w n a = case scene n a of
  Nothing -> return ()
  Just (i, f) -> writeChan ch ((i, show w ++ "_" ++ f), a)

scene :: Int -> AngledInternalAddress -> Maybe (Image, FilePath)
scene n a = do
  a' <- (angledFromList . angledToList) a
  m <- findAtom_ a'
  let cx :+ cy = muNucleus m :: Complex R
```

```
      f = filename n
      i = Image
            { imageLocation = Location
                { center = toRational cx :+ toRational cy
                , radius = muSize m * 32
                }
            , imageViewport = Viewport
                { aspect = 1
                , orient = muOrient m - pi / 2
                }
            , imageWindow    = Window
                { width = 512
                , height = 512
                , supersamples = 8
                }
            , imageColours   = Colours
                { colourInterior = Colour 1 0 0
                , colourBoundary = Colour 0 0 0
                , colourExterior = Colour 1 1 1
                }
            , imageLabels    = []
            , imageLines     = []
            }
  return (i, f)

filename :: Int -> FilePath
filename n = (reverse . take 4 . (++ ”0000”) . reverse . show) n ++ ”.ppm”

randomAddresses :: RandomGen g => g -> [AngledInternalAddress]
randomAddresses g = let (g', a) = randomAddress g in  a : randomAddresses g'

randomAddress :: RandomGen g => g -> (g, AngledInternalAddress)
randomAddress g = randomAddress' g 16 2 1

randomAddress' ::
  RandomGen g => g -> Int -> Integer -> Integer -> (g, AngledInternalAddress)
randomAddress' g0 size _den per | size == 0 || per > 100 = (g0, Unangled (↙
    ↪ fromInteger per))
randomAddress' g0 size  den per
  | coin < (0.125 :: Double) && den' > 2 =
      if per' > 200
        then (g6, Unangled (fromInteger per))
        else Angled (fromInteger per) angle ‘fmap‘ randomAddress' g6 (size - 1) ↙
            ↪ den' per'
  | otherwise = Angled (fromInteger per) (1 % 2) ‘fmap‘ randomAddress' g6 (size ↙
      ↪ - 1) den per2
  where
    (coin, g1) = random g0
    (rand, g2) = random g1
    (numr, g3) = randomR (1, denr - 1) g2
    (poff, g4) = randomR (1, den - 1)  g3
    (per', g5) = randomR (perMin, perMax) g4
    (per'', g6) = randomR (perMin', perMax') g5
    per2 = if den > 2 then per + poff else per''
    denr = floor (31 * rand * rand + 2 :: Double)
    angle = numr % denr
    den' = denominator angle
    perMin = per * (den' - 1) - 1
    perMax = (per + 1) * den' - 1
    perMin' = per + 1
    perMax' = per * 2
```

16 src/gruff-raytrace.hs

```
import Control.Monad (ap)
import Data.Ratio ((%))
import System.Environment (getArgs)
import Fractal.GRUFF

main :: IO ()
main = do
  [scount, snum, sden] <- getArgs
  let count = read scount
      num = read snum
      den = read sden
      angle = num % den
      callback = RayTraceForwardCallback $ \continue c _ ->
        let x :+ y = toComplex c
        in  (if magnitudeSquared c < 4 then ((toDFloat x :+ toDFloat y) :) else ↙
          ↳ id)
             (rayTraceForward continue callback)
      ray = rayTraceForward (rayTraceForwardStart (ExternalAngle angle)) ↙
        ↳ callback
  defaultMain . map scene . zip [0..] . (zip`ap`tail) . take count $ ray

scene (n, (cx0 :+ cy0, cx1 :+ cy1)) =
  withDFloat cx0 $ \x0' -> withDFloat cy0 $ \y0' ->
    withDFloat cx1 $ \x1' -> withDFloat cy1 $ \y1 ->
      let x0 = auto x0'
          y0 = auto y0'
          x1 = auto x1'
          c0 = x0 :+ y0
          c1 = x1 :+ y1
          d = c1 - c0
          center' = 0.5 * (c0 + c1)
          radius' = 0.5 `min` (128 * auto (magnitude d) :: F24)
          orient' = phase (fmap realToFrac d :: Complex Double) - pi / 2
      in  (Image{ imageLocation = Location
                    { center = fmap toRational center'
                    , radius = toRational radius'
                    }
                , imageViewport = Viewport
                    { aspect = 512/288
                    , orient = orient'
                    }
                , imageWindow    = Window
                    { width = 512
                    , height =  288
                    , supersamples = 0.5
                    }
                , imageColours  = Colours
                    { colourInterior = Colour 1 0 0
                    , colourBoundary = Colour 0 0 0
                    , colourExterior = Colour 1 1 1
                    }
                ,  imageLabels    = []
                ,  imageLines     = []
                }
          , filename n)

filename n = (reverse . take 5 . (++ "00000") . reverse . show) n ++ ".ppm"
```

17 src/gruff-whn.hs

```
{-

Music video for:

B1t Crunch3r vs Killeralien vs Phonetic System
White Hole Nocturne (Feat. Jay Cotton)
Planet Terror Records planet015 #03

speed up audio by 2.0408% from 140bpm to 142.857bpm
video at 25fps

-}
import Data.Maybe (mapMaybe)
import Data.MemoCombinators (list, char)
import System.Environment (getArgs)

import Fractal.GRUFF

import Fractal.RUFF.Mandelbrot.Address (parseAngledInternalAddress)
import Fractal.RUFF.Mandelbrot.Atom (MuAtom(..), findAtom_)
import Fractal.RUFF.Types.Complex (Complex((:+)))

import Number (R)

data Quality = Preview | P288 | P576 | P720 | P1080 deriving Read

main :: IO ()
main = do
  args <- getArgs
  let q = case args of
            [q'] -> case reads q' of
              [(q'', "")] -> q''
              _ -> Preview
            _ -> Preview
  defaultMain (animation q)

window :: Quality -> Window
window Preview = Window{ width =  512, height =  288, supersamples =  1    }
window P288    = Window{ width =  512, height =  288, supersamples = 14    }
window P576    = Window{ width = 1080, height =  576, supersamples =  3.5  }
window P720    = Window{ width = 1280, height =  720, supersamples =  2.25 }
window P1080   = Window{ width = 1920, height = 1080, supersamples =  1    }

animation :: Quality -> [(Image, String)]
animation q = mapMaybe (scene q) (score `zip` [0..])

scene :: Quality -> (String, Int) -> Maybe (Image, FilePath)
scene q (s, n) = do
  m <- findMu s
  let cx :+ cy = muNucleus m
      f = filename n
      i = Image
            { imageLocation = Location
                { center = toRational cx :+ toRational cy
                , radius = muSize m * 8
                }
            , imageViewport = Viewport
                { aspect = aspectQ q
                , orient = muOrient m - pi / 2
                }
            , imageWindow   = window q
            , imageColours  = Colours
                { colourInterior = Colour 1 0 0
                , colourBoundary = Colour 0 0 0
                , colourExterior = Colour 1 1 1
                }
            , imageLabels   = []
            , imageLines    = []
            }
  return (i, f)

findMu :: String -> Maybe (MuAtom R)
findMu = list char findMu'

findMu' :: String -> Maybe (MuAtom R)
findMu' s = do
  a <- parseAngledInternalAddress s
  findAtom_ a

aspectQ :: Quality -> Double
aspectQ q = let w = window q in fromIntegral (width w) / fromIntegral (height w)

filename :: Int -> String
filename n = (reverse . take 4 . (++ "0000") . reverse . show) n ++ ".ppm"

kick1, snare1, kick2, snare2, kick3, snare3 :: Int -> [String]
bass3 :: Int -> Int -> [String]
kick1  n = [ "1 2 "     ++ (unwords . map show . take m . scanl (+) (3 :: Int) .↙
    ↳ repeat) 1 | m <- [n, n - 1 .. 1] ]
snare1 n = [ "1 2 "     ++ (unwords . map show . take (2 * m) . filter (\x -> x ↙
    ↳ `mod` 3 /= 0)) [(3 :: Int) ..] | m <- [n, n - 1 .. 1] ]
kick2  n = [ "1 2 "     ++ (unwords . map show . take m . scanl (+) (5 :: Int) .↙
    ↳ repeat) 2 | m <- [n, n - 1 .. 1] ]
snare2 n = [ "1 2 3 "   ++ (unwords . map show . take (3 * m) . filter (\x -> x ↙
    ↳ `mod` 4 /= 0)) [(4 :: Int) ..] | m <- [n, n - 1 .. 1] ]
kick3  n = [ "1 2 4 8 " ++ (unwords . map show . take m . scanl (+) (10:: Int) .↙
    ↳ repeat) 4 | m <- [n, n - 1 .. 1] ]
snare3 n = [ "1 2 3 4 " ++ (unwords . map show . take (4 * m) . filter (\x -> x ↙
    ↳ `mod` 5 /= 0)) [(5 :: Int) ..] | m <- [n, n - 1 .. 1] ]
bass3 n k= [ "1 2 4 "   ++ show k ++ "/7 " ++ (unwords . map show . take m) [(23↙
    ↳ :: Int) ..] | m <- [n, n - 1 .. 1] ]

score :: [String]
score = concat $
```

```
    [ kick1  21
    , snare1 32
    , kick1  20
    , snare1 11
    , kick1   8
    , kick1   8
    , kick1   5
    , snare1 32
    , kick1  20
    , snare1 11
    ] ++
    [ kick2  21
    , snare2 32
    , kick2  20
    , snare2 11
    , kick2  21
    , snare2 32
    , kick2  10
    , snare2  8
    , snare2  8
    , snare2  5
    ] ++
    [ kick3  21
    , snare3 32
    , kick3  15
    , snare3  5
    , snare3 11
    , kick3   4
    , bass3   4 1
    , kick3   8
    , kick3   5
    , snare3 32
    , kick3  16
    , bass3   5 2
    , snare3  4
    , bass3   3 3
    , snare3  3
    ] ++
    [ kick3  21
    , snare1 32
    , kick2  20
    , snare3 11
    , kick1   8
    , kick2   5
    , bass3   3 4
    , kick3   5
    , snare2  6
    , bass3  26 5
    , kick2  16
    , bass3   5 6
    , snare3  5
    , snare2  3
    , snare1  2
    ]
```

18 src/gruff-zoom.hs

```
import System.Environment (getArgs)

import Fractal.GRUFF

main :: IO ()
main = do
  [fn, nfs] <- getArgs
  i0 <- read ‘fmap‘ readFile fn
  let nf = read nfs
      r0 = 4
      r1 = radius (imageLocation i0)
      dr = r1 / r0
      zoom f =
        let t = fromIntegral f / fromIntegral (nf - 1)
            r = r0 * dr ** t
        in  i0{ imageLocation = (imageLocation i0){ radius = r } }
      name = (++ ".ppm") . reverse . take 8 . (++ "00000000") . reverse . show
  defaultMain [ (zoom f, name f) | f <- [0 .. nf - 1] ]
```

19 src/Number.hs

```
{-# LANGUAGE CPP, GeneralizedNewtypeDeriving #-}
module Number (R) where

import Data.Vec (NearZero)

#ifdef HAVE_PRECISION

#else
#ifdef HAVE_MPFR

import Data.Vec (nearZero)
import Control.Monad (guard)
import Data.Ratio (numerator, denominator)
import Numeric (readSigned)
import Data.Number.MPFR (MPFR, RoundMode(Near, Up), Precision, getPrec, int2w, ↙
    ↪ fromIntegerA, stringToMPFR_, toString)
import Data.Number.MPFR.Instances.Near ()

#else

import Numeric.QD (QuadDouble)
import Numeric.QD.Vec ()

#endif
#endif

#ifdef HAVE_MPFR

instance NearZero MPFR where
  nearZero x = let p = getPrec x in not (abs x > int2w Up p 1 (4 - fromIntegral ↙
      ↪ p))

newtype R = R MPFR
  deriving (Eq, Ord, Floating, Real, RealFrac, NearZero)

instance Num R where
```

```
  R a + R b = R (a + b)
  R a * R b = R (a * b)
  R a - R b = R (a - b)
  negate (R a) = R (negate a)
  abs (R a) = R (abs a)
  signum (R a) = R (signum a)
  fromInteger i = R (fromIntegerA Near bits i)

instance Fractional R where
  R a / R b = R (a / b)
  recip (R a) = R (recip a)
  fromRational r = R (fromIntegerA Near bits (numerator r) / fromIntegerA Near ↙
      ↪ bits (denominator r))

instance Read R where
  readsPrec _ = readParen False . readSigned $ \s -> do
    (f, r) <- lex s
    let (n, k) = stringToMPFR_ Near bits 10 f
    guard (k == 0)
    return (R n, r)

instance Show R where
  show (R m) = toString (ceiling $ (2::Double) + log 2 / log 10 * fromIntegral (↙
      ↪ getPrec m)) m

bits :: Precision
bits = 1000

#else

newtype R = R QuadDouble
  deriving (Eq, Ord, Num, Fractional, Floating, Real, RealFrac, NearZero)

instance Show R where
  show (R m) = show m

instance Read R where
  readsPrec p = map (\(m, s) -> (R m, s)) . readsPrec p

#endif
```