dr1

Claude Heiland-Allen

2010–2019

Contents

1 AUTHORS

Claude Heiland-Allen <claude@mathr.co.uk>

2 cathedral-algorithms/audio.sh

```
#!/bin/bash
STEMS=$(ls -1 out | grep ".txt$" | sed "s|.txt$||")
SOURCES=$(for STEM in ${STEMS} ; do echo ${STEM} ; done | sed "s|--.*||" | sort
    ↪ | uniq)
for SOURCE in ${SOURCES}
do
  SINKS=$(for STEM in ${STEMS} ; do echo ${STEM} ; done | grep "^${SOURCE}--")
```

```
  WARMUP=$(for STEM in ${STEMS} ; do echo ${STEM} ; done | grep "_-_${SOURCE}$" ↙
      ↘ | head -n 1)
  for SINK in ${SINKS}
  do
    pd -noprefs -noaudio -nogui -stderr -r 48000 -open "cathedral.pd" -send "↙
        ↘ SETUP out/${WARMUP}.txt out/${SINK}.txt out/${SINK}.wav"
  done
done
```

3 cathedral-algorithms/Cathedral.hs

```
module Main(main) where

import Graphics.UI.GLUT hiding (Green, rotate, Render)
import Data.IORef
import Data.List(sortBy)
import Data.Ord(comparing)
import Control.Monad(liftM, when)
import System.Random(getStdGen, randoms)
import Data.Time(UTCTime(UTCTime), fromGregorian, secondsToDiffTime, ↙
    ↘ getCurrentTime, diffUTCTime)
import System.IO(withBinaryFile, Handle(), IOMode(ReadMode, WriteMode), ↙
    ↘ openBinaryFile, hClose, hPutStrLn, hGetBuf)
import System.FilePath((</>))
import System.Exit(exitFailure, exitSuccess)
import Foreign.Marshal.Alloc(allocaBytes)

import Snapshot

record :: Bool
record = True

segments :: Int
segments = 2

winWidth :: GLsizei
winWidth  = 1044

winHeight :: GLsizei
winHeight =  336

count :: Int
count = 25

chunk :: Int -> [a] -> [[a]]
chunk _ [] = []
chunk n ls = let (xs, ys) = splitAt n ls in xs : chunk n ys

data Cell = Green | White | Orange
  deriving (Read, Show, Eq, Ord, Enum, Bounded)

child :: (Cell, Bool) -> ((Cell, Bool), (Cell, Bool))
child (Green,  b) = if b then ((Orange, not b), (White,  not b))
                         else ((White,  not b), (Orange, not b))
child (White,  b) = if b then ((Green,  not b), (Orange, not b))
                         else ((Orange, not b), (Green,  not b))
child (Orange, b) = if b then ((White,  not b), (Green,  not b))
```

```
                          else ((Green,  not b), (White,  not b))

rotate :: (Cell, Bool) -> (Cell, Bool)
rotate t@(_, b) = let (c, _) = fst (child t) in (c, b)

fromBits :: [Bool] -> GLdouble
fromBits = foldr (\b s -> (if b then 0.5 else 0) + 0.5 * s) 0

data World = World
  { wTop :: (Cell, Bool)
  , wBits :: [Bool]
  , wDelta :: GLdouble
  }

world0 :: World
world0 = World
  { wTop = (Green, False)
  , wBits = repeat False
  , wDelta = 0
  }

data Render = Render
  { rPlan :: [(World, FilePath)]
  , rVideo :: Maybe Handle
  , rScore :: Maybe Handle
  , rCorners :: (GLdouble, GLdouble, GLdouble, GLdouble)
  , rTex :: Cell -> TextureObject
  , rFrames :: Int
  , rTotal :: Int
  , rNow :: GLdouble
  , rTime :: GLdouble
  }

render0 :: Render
render0 = Render
  { rPlan = []
  , rVideo = Nothing
  , rScore = Nothing
  , rCorners = (0, 0, 0, 0)
  , rTex = error "rTex"
  , rFrames = 0
  , rTotal = 0
  , rNow = 0
  , rTime = 0
  }

reshape :: IORef Render -> Size -> IO ()
reshape renderRef vp@(Size w h) = do
  let cs@(x0,x1,y0,y1) = let v = fromIntegral w / fromIntegral h
                         in (-v/8, v/8, 0, 0.25)
  writeIORef renderRef. (\rr -> rr{ rCorners = cs }) =<< readIORef renderRef
  viewport $= (Position 0 0, vp)
  matrixMode $= Projection
  loadIdentity
  ortho x0 x1 y0 y1 (-1) 1
  matrixMode $= Modelview 0
  loadIdentity
```

```
  postRedisplay Nothing

start :: IORef Render -> IO ()
start renderRef = do
  rr <- readIORef renderRef
  when (null (rPlan rr)) exitSuccess
  let (_, nn) = head (rPlan rr)
  case (rVideo rr, rScore rr) of
    (Nothing, Nothing) -> do
      hv <- openBinaryFile ("out" </> nn ++ ".ppm") WriteMode
      hs <- openBinaryFile ("out" </> nn ++ ".txt") WriteMode
      writeIORef renderRef rr
        { rVideo = Just hv
        , rScore = Just hs
        , rFrames = 0
        , rTotal = 0
        , rNow = 0
        , rTime = 0
        }
    _ -> error "start"

stop :: IORef Render -> IO ()
stop renderRef = do
  rr <- readIORef renderRef
  when (not . null . rPlan $ rr) $ do
    case (rVideo rr, rScore rr) of
      (Just hv, Just hs) -> do
        hClose hv
        hClose hs
        writeIORef renderRef rr
          { rPlan  = tail (rPlan rr)
          , rVideo = Nothing
          , rScore = Nothing
          }
      _ -> error "stop"

step' :: World -> World
step' ww = ww
  { wTop = (if head (wBits ww) then snd else rotate . fst) (child (wTop ww))
  , wBits = tail (wBits ww)
  , wDelta = fromBits . take 64 . tail . wBits $ ww
  }

step :: IORef Render -> IO ()
step renderRef = do
  rr <- readIORef renderRef
  let (ww, nn):pp = rPlan rr
      rr' = if rFrames rr + 1 == count
            then rr{ rPlan = (step' ww, nn) : pp
                   , rFrames = 0
                   , rTime = 0
                   , rTotal = rTotal rr + 1
                   }
            else rr{ rFrames = rFrames rr + 1
                   , rTime = fromIntegral (rFrames rr + 1) / fromIntegral count
                   }
  writeIORef renderRef rr'
```

```
    when (rTotal rr' == 60) $ do
      stop renderRef
      start renderRef

simulateNRT :: Int -> IORef Render -> IO ()
simulateNRT mspf renderRef = do
  step renderRef
  postRedisplay Nothing
  addTimerCallback mspf $ simulateNRT mspf renderRef

simulateRT :: Int -> IORef Render -> IO ()
simulateRT = simulateNRT {- mspf renderRef = do
  rr <- readIORef renderRef
  tnow <- utcr
  let t = tnow - rNow rr
  when (t >= 1) $ step renderRef
  rr' <- readIORef renderRef
  writeIORef renderRef rr'{ rTime = tnow - rNow rr' }
  postRedisplay Nothing
  addTimerCallback mspf $ simulateRT mspf renderRef
-}

display :: IORef Render -> IO ()
display renderRef = do
  r <- readIORef renderRef
  let w = fst . head . rPlan $ r
  d <- if record then return (rTime r) else subtract (rNow r) `liftM` utcr
  let t = wTop w
      delta = wDelta w
      ds = 2 ** d
      x0 = (delta - 2) * ds
      x1 = (delta + 2) * ds
      y0 =  0
      y1 =  4 * ds
      (cx0, cx1, cy0, cy1) = rCorners r
  clear [ColorBuffer]
  blend $= Enabled
  blendFunc $= (SrcAlpha, OneMinusSrcAlpha)
  texture Texture2D $= Enabled
  drawTiles (rTex r) cx0 cy0 cx1 cy1 x0 y0 x1 y1 t
  textureBinding Texture2D $= Nothing
  texture Texture2D $= Disabled
  swapBuffers
  when record $ do
    let Just hv = rVideo r
        Just hs = rScore r
    hSnapshot hv (Position 0 0) (Size winWidth winHeight)
    when (rFrames r == 0) $ do
      let scorePart (x, c) = "new " ++ show x ++ " " ++ show (fromEnum c) ++ ", "
          s = concatMap scorePart . take 9 . sortBy (comparing (abs . fst)) . getTileRow x0 y0 x1 y1 t $ 8
      hPutStrLn hs (s ++ "done;")

getTileRow :: GLdouble -> GLdouble -> GLdouble -> GLdouble -> (Cell, Bool) -> Int -> [(GLdouble, Cell)]
getTileRow x0 y0 x1 y1 t n | n > 0 = let x2 = (x0 + x1) / 2
```

```
                                        y2 = (y0 + y1) / 2
                                        (l, r) = child t
                                        m = n - 1
                                    in getTileRow x0 y0 x2 y2 l m ++
                                       getTileRow x2 y0 x1 y2 r m
                            | otherwise = let x = (x0 + x1) / 2
                                              (c, _) = t
                                          in [(x,c)]

drawTiles :: (Cell -> TextureObject) -> GLdouble -> GLdouble -> GLdouble -> ↙
    ↳ GLdouble -> GLdouble -> GLdouble -> GLdouble -> GLdouble -> (Cell, Bool) ↙
    ↳ -> IO ()
drawTiles tex cx0 cy0 cx1 cy1 x0 y0 x1 y1 t@(c, _) = do
  let outside = or [ x1 < cx0, cx1 < x0, y1 < cy0, cy1 < y0 ]
  when (not outside) $ do
    textureBinding Texture2D $= Just (tex c)
    renderPrimitive Quads $ do
      let tc, v :: GLdouble -> GLdouble -> IO ()
          tc x y = texCoord $ TexCoord2 x y
          v  x y = vertex   $ Vertex2   x y
      color $ Color3 1 1 (1::GLdouble)
      tc 0 1 >> v x0 y0
      tc 0 0 >> v x0 y1
      tc 1 0 >> v x1 y1
      tc 1 1 >> v x1 y0
    let small = abs (((x1 - x0) * (y1 - y0)) / ((cx1 - cx0) * (cy1 - cy0))) < ↙
        ↳ 0.00001
    when (not small) $ do
      let x2 = (x0 + x1) / 2
          y2 = (y0 + y1) / 2
          (l, r) = child t
      drawTiles tex cx0 cy0 cx1 cy1 x0 y0 x2 y2 l
      drawTiles tex cx0 cy0 cx1 cy1 x2 y0 x1 y2 r

textures :: TextureObject -> TextureObject -> TextureObject -> Cell -> ↙
    ↳ TextureObject
textures g _ _ Green  = g
textures _ w _ White  = w
textures _ _ o Orange = o

main :: IO ()
main = do
  let w = winWidth
      h = winHeight
      v = fromIntegral w / fromIntegral h
      cs = (-v/8, v/8, 0, 0.25)
      mspf = 1000 `div` count
  initialWindowSize $= Size w h
  initialDisplayMode $= [RGBAMode, DoubleBuffered]
  _ <- getArgsAndInitialize
  _ <- createWindow "Cathedral Algorithms"
  green  <- loadTexture "green.rgba"
  white  <- loadTexture "white.rgba"
  orange <- loadTexture "orange.rgba"
  g <- getStdGen
  tnow <- utcr
  let bits = randoms g
```

```
      numbers = take segments (chunk 60 bits)
      plan =
        [ (world, name)
        | c1 <- [ Green, White, Orange]
        , n1 <- numbers
        , n2 <- numbers
        , let n = n1 ++ n2
        , let world = world0
                      { wTop = (c1, False)
                      , wBits = cycle n
                      , wDelta = fromBits . take 64 . cycle $ n
                      }
        , let c2 = fst . wTop . (!! 60) . iterate step' $ world
        , let name = show (fromEnum c1) ++ "_" ++ map bit n1 ++ "---"
                ++ show (fromEnum c2) ++ "_" ++ map bit n2
        ]
      bit b = if b then '1' else '0'
  renderRef <- newIORef render0
    { rPlan = plan
    , rTex = textures green white orange
    , rCorners = cs
    , rNow = tnow
    }
  start renderRef
  displayCallback $= display renderRef
  reshapeCallback $= Just (reshape renderRef)
  addTimerCallback mspf $ (if record then simulateNRT else simulateRT) mspf ↙
      ↳ renderRef
  mainLoop

loadTexture :: FilePath -> IO TextureObject
loadTexture f = do
  withBinaryFile f ReadMode $ \h -> do
    let bytes = 1024 * 1024 * 4
    allocaBytes bytes $ \pixels -> do
      bytes' <- hGetBuf h pixels bytes
      when (bytes' /= bytes) exitFailure
      [tex] <- genObjectNames 1
      texture Texture2D $= Enabled
      textureBinding Texture2D $= Just tex
      build2DMipmaps Texture2D RGBA' 1024 1024
        (PixelData RGBA UnsignedByte pixels)
      textureWrapMode Texture2D S $= (Repeated, ClampToEdge)
      textureWrapMode Texture2D T $= (Repeated, ClampToEdge)
      textureBinding Texture2D $= Nothing
      texture Texture2D $= Disabled
      return tex

-- these two functions copied from hosc/Sound.OpenSoundControl.Time
utc_base :: UTCTime
utc_base = UTCTime (fromGregorian 1970 1 1) (secondsToDiffTime 0)
utcr :: IO GLdouble
utcr = do t <- getCurrentTime ; return (realToFrac (diffUTCTime t utc_base))
```

4 cathedral-algorithms/cathedral.pd

```
#N canvas 48 0 810 672 10;
```

```
#X obj 60 291 catch~ \$0-left;
#X obj 164 290 catch~ \$0-right;
#X msg 371 593 clear;
#X obj 308 474 list prepend \$0;
#X obj 308 441 route new done;
#X msg 347 512 loadbang;
#X obj 462 98 + 1;
#X obj 431 97 f 0;
#N canvas 0 0 450 300 \$0-arches-0 0;
#X restore 17 14 pd \$0-arches-0;
#N canvas 0 0 450 300 \$0-arches-1 0;
#X restore 17 34 pd \$0-arches-1;
#N canvas 0 0 450 300 \$0-arches-2 0;
#X restore 17 54 pd \$0-arches-2;
#N canvas 0 0 450 300 \$0-arches-3 0;
#X restore 17 74 pd \$0-arches-3;
#N canvas 0 0 450 300 \$0-arches-4 0;
#X restore 17 94 pd \$0-arches-4;
#N canvas 0 0 450 300 \$0-arches-5 0;
#X restore 17 114 pd \$0-arches-5;
#N canvas 361 299 450 300 \$0-arches-6 0;
#X restore 17 134 pd \$0-arches-6;
#N canvas 0 0 450 300 \$0-arches-7 0;
#X restore 17 154 pd \$0-arches-7;
#X obj 371 637 s;
#X obj 411 464 f 0;
#X obj 448 464 + 1;
#X obj 376 571 t b a;
#N canvas 0 0 450 300 \$0-arches-8 0;
#X restore 17 174 pd \$0-arches-8;
#N canvas 0 0 450 300 \$0-arches-9 0;
#X restore 17 194 pd \$0-arches-9;
#N canvas 0 0 450 300 \$0-arches-10 0;
#X restore 117 14 pd \$0-arches-10;
#N canvas 0 0 450 300 \$0-arches-11 0;
#X restore 117 34 pd \$0-arches-11;
#N canvas 0 0 450 300 \$0-arches-12 0;
#X restore 117 54 pd \$0-arches-12;
#N canvas 0 0 450 300 \$0-arches-13 0;
#X restore 117 74 pd \$0-arches-13;
#N canvas 0 0 450 300 \$0-arches-14 0;
#X restore 117 94 pd \$0-arches-14;
#N canvas 0 0 450 300 \$0-arches-15 0;
#X restore 117 114 pd \$0-arches-15;
#N canvas 0 0 450 300 \$0-arches-16 0;
#X restore 117 134 pd \$0-arches-16;
#N canvas 0 0 450 300 \$0-arches-17 0;
#X restore 117 154 pd \$0-arches-17;
#N canvas 0 0 450 300 \$0-arches-18 0;
#X restore 117 174 pd \$0-arches-18;
#N canvas 0 0 450 300 \$0-arches-19 0;
#X restore 117 194 pd \$0-arches-19;
#N canvas 0 0 450 300 \$0-arches-20 0;
#X restore 227 14 pd \$0-arches-20;
#N canvas 0 0 450 300 \$0-arches-21 0;
#X restore 227 34 pd \$0-arches-21;
#N canvas 0 0 450 300 \$0-arches-22 0;
```

```
#X restore 227 54 pd \$0-arches-22;
#N canvas 0 0 450 300 \$0-arches-23 0;
#X restore 227 74 pd \$0-arches-23;
#N canvas 0 0 450 300 \$0-arches-24 0;
#X restore 227 94 pd \$0-arches-24;
#N canvas 0 0 450 300 \$0-arches-25 0;
#X restore 227 114 pd \$0-arches-25;
#N canvas 0 0 450 300 \$0-arches-26 0;
#X restore 227 134 pd \$0-arches-26;
#N canvas 0 0 450 300 \$0-arches-27 0;
#X restore 227 154 pd \$0-arches-27;
#N canvas 0 0 450 300 \$0-arches-28 0;
#X restore 227 174 pd \$0-arches-28;
#N canvas 0 0 450 300 \$0-arches-29 0;
#X restore 227 194 pd \$0-arches-29;
#X obj 448 486 mod 30;
#X obj 113 260 send~ \$0-phase;
#X msg 308 540 obj 10 10 organ-pipe \$1 \$2 \$3;
#X obj 113 235 phasor~ 8;
#X obj 164 347 expr~ tanh($v1);
#X obj 61 347 expr~ tanh($v1);
#X obj 584 619 soundfiler;
#X obj 584 570 list append \$0-l \$0-r;
#X obj 584 543 symbol;
#X obj 61 385 tabwrite~ \$0-l;
#X obj 164 384 tabwrite~ \$0-r;
#X obj 164 322 *~ 0.125;
#X obj 61 321 *~ 0.125;
#X msg 584 593 write -wave -bytes 2 \$1 \$2 \$3;
#X obj 548 81 list append;
#X obj 344 60 delay 1000;
#X msg 357 84 \; pd dsp 1;
#X obj 344 121 delay 1000;
#X obj 431 149 print progress;
#X obj 427 411 textfile;
#X msg 517 411 0;
#X msg 582 150 read \$1 \, rewind;
#X obj 548 101 unpack s s s;
#X obj 497 483 delay 1000;
#X obj 497 574 delay 1000;
#X msg 497 600 \; pd quit;
#X obj 472 433 spigot 1;
#X msg 548 130 read \$1 \, rewind;
#X obj 363 272 textfile;
#X obj 344 171 metro 1000;
#X obj 363 243 until;
#X msg 363 220 10;
#X msg 408 226 \; pd dsp 0;
#X msg 299 223 \; pd dsp 1;
#X msg 453 276 0;
#X obj 408 298 spigot 1;
#X obj 408 320 metro 1000;
#X obj 427 390 until;
#X msg 426 368 10;
#X msg 467 375 \; pd dsp 0;
#X msg 367 372 \; pd dsp 1;
#X obj 343 197 t b b b b;
```

```
#X obj 408 347 t b b b b;
#X obj 411 510 makefilename pd-\$0-arches-%d;
#X obj 431 11 receive SETUP;
#X text 525 12 SETUP: warmup.txt render.txt render.wav;
#X text 417 644 (GPL) 2010 Claude Heiland-Allen <claude@mathr.co.uk>
;
#X obj 449 129 print setup;
#X obj 431 75 metro 12500;
#X obj 107 472 table \$0-l 2.88e+06;
#X obj 106 495 table \$0-r 2.88e+06;
#X obj 189 425 print~;
#X obj 431 32 t b b a;
#X obj 122 420 dac~;
#X connect 0 0 54 0;
#X connect 1 0 53 0;
#X connect 2 0 16 0;
#X connect 3 0 44 0;
#X connect 4 0 3 0;
#X connect 4 1 5 0;
#X connect 5 0 16 0;
#X connect 6 0 7 1;
#X connect 7 0 6 0;
#X connect 7 0 60 0;
#X connect 17 0 18 0;
#X connect 17 0 85 0;
#X connect 18 0 42 0;
#X connect 19 0 2 0;
#X connect 19 1 16 1;
#X connect 42 0 17 1;
#X connect 44 0 16 0;
#X connect 45 0 43 0;
#X connect 46 0 52 0;
#X connect 46 0 95 1;
#X connect 46 0 93 0;
#X connect 47 0 51 0;
#X connect 47 0 95 0;
#X connect 49 0 55 0;
#X connect 50 0 49 0;
#X connect 53 0 46 0;
#X connect 54 0 47 0;
#X connect 55 0 48 0;
#X connect 56 0 64 0;
#X connect 56 0 89 0;
#X connect 57 0 58 0;
#X connect 57 0 59 0;
#X connect 59 0 71 0;
#X connect 61 0 4 0;
#X connect 61 1 68 0;
#X connect 62 0 68 1;
#X connect 62 0 78 0;
#X connect 63 0 61 0;
#X connect 64 0 69 0;
#X connect 64 1 63 0;
#X connect 64 2 50 1;
#X connect 65 0 66 0;
#X connect 65 0 50 0;
#X connect 66 0 67 0;
```

```
#X connect 68 0 62 0;
#X connect 68 0 65 0;
#X connect 69 0 70 0;
#X connect 70 0 4 0;
#X connect 70 1 77 0;
#X connect 71 0 83 0;
#X connect 72 0 70 0;
#X connect 73 0 72 0;
#X connect 76 0 77 1;
#X connect 76 0 71 0;
#X connect 77 0 76 0;
#X connect 77 0 78 0;
#X connect 77 0 51 0;
#X connect 77 0 52 0;
#X connect 78 0 84 0;
#X connect 79 0 61 0;
#X connect 80 0 79 0;
#X connect 83 0 75 0;
#X connect 83 1 73 0;
#X connect 83 2 17 0;
#X connect 83 3 74 0;
#X connect 84 0 82 0;
#X connect 84 1 80 0;
#X connect 84 2 17 0;
#X connect 84 3 81 0;
#X connect 85 0 19 0;
#X connect 86 0 94 0;
#X connect 90 0 7 0;
#X connect 90 0 93 0;
#X connect 94 0 90 0;
#X connect 94 1 57 0;
#X connect 94 2 56 0;
```

5 cathedral-algorithms/encode_audio.sh

```
#!/bin/bash
for WAV in $@
do
  twolame -b 224 "${WAV}"
done
```

6 cathedral-algorithms/encode_video.sh

```
#!/bin/bash
for PPM in $@
do
  M2V="${PPM%ppm}m2v"
  ppmtoy4m -S 444 -F 25:1 <"${PPM}" |
  y4mscaler -I sar=1/1 -O preset=dvd_wide -O yscale=1/1 |
  mpeg2enc -f 8 -q 3 -b 8000 -B 768 -D 10 -g 9 -G 15 -P -R 2 -o "${M2V}"
done
```

7 cathedral-algorithms/.gitignore

```
Cathedral
MakeDVD
```

```
interleave31
*.hi
*.o
*.rgba
*.stamp
transitions.dot
transitions.png
out
```

8 cathedral-algorithms/interleave31.c

```
#include <stdio.h>

int main(int argc, char **argv) {
  if (argc != 3) {
    return 1;
  }
  FILE *in3 = fopen(argv[1], "rb");
  if (in3) {
    FILE *in1 = fopen(argv[2], "rb");
    if (in1) {
      for (int i = 0; i < 1024*1024; ++i) {
        putchar(getc(in3));
        putchar(getc(in3));
        putchar(getc(in3));
        putchar(getc(in1));
      }
      fclose(in1);
    } else {
      return 1;
    }
    fclose(in3);
  } else {
    return 1;
  }
  return 0;
}
```

9 cathedral-algorithms/MakeDVD.hs

```
import System.Environment(getArgs)
import Data.Ord(comparing)
import Data.List(sortBy)
import Data.Map((!))
import qualified Data.Map as M
import qualified Data.Set as S

default(Int)

dvd :: [String] -> String
dvd mpegs =
  unlines
    [ "<dvdauthor jumppad=\"no\"><vmgm>"
    , "<fpc>{ g1 = random(" ++ show (M.size nodes) ++ "); jump titleset 1 menu ↙
      ↪ entry root; }</fpc>"
    , "</vmgm><titleset><menus>"
```

```
    , unlines . map menu . sortBy (comparing snd) . M.toList $ nodes
    , rootMenu (M.size nodes)
    , "</menus><titles>"
    , unlines . map pgc  . sortBy (comparing snd) . M.toList $ trans
    , "</titles></titleset></dvdauthor>"
    ]
  where
    mpeg f t = f ++ "--" ++ t ++ ".mpeg"
    froms  = map (take 62          ) mpegs
    tos    = map (take 62 . drop 65) mpegs
    edges  = zip froms tos
    source = M.fromList (zip mpegs froms)
    sink   = M.fromList (zip mpegs tos)
    nodes  = M.fromList $ zip (S.toList (S.fromList (M.elems source) `S.union` S.fromList (M.elems sink))) [1..]
    trans  = M.fromList $ zip mpegs [1..]
    pgc (vob, n) =
      unlines
        [ "<pgc><!-- " ++ show n ++ " -->"
        , "<pre>{ g1 = " ++ show (nodes ! (sink ! vob)) ++ "; }</pre>"
        , "<vob file=\"" ++ vob ++ "\" />"
        , "<post>call menu entry root;</post>"
        , "</pgc>"
        ]
    menu (from, n) =
      unlines
        [ "<pgc><!-- " ++ show n ++ " -->"
        , "<pre>{ "
        , "g0 = random(" ++ show m ++ ");"
        , switch "g0" m [1 .. m] (map snd to) $ \t ->
            "jump title " ++ show (trans ! mpeg from t) ++ ";"
        , " }</pre><vob file=\"menu.mpeg\" /></pgc>"
        ]
      where
        m = length to
        to = filter ((from ==) . fst) edges
    rootMenu n =
      unlines
        [ "<pgc entry=\"root\">"
        , "<pre>{ "
        , switch "g1" n [1 .. n] [1 .. n] $ \t ->
            "jump menu " ++ show t ++ ";"
        , " }</pre><vob file=\"menu.mpeg\" /></pgc>"
        ]
    switch r k xs ts f
      | k >  1 = unlines
                   [ "if (" ++ r ++ " lt " ++ show (head xs2) ++ ") {"
                   , switch r k2 xs1 ts1 f
                   , "} else {"
                   , switch r (k - k2) xs2 ts2 f
                   , "}"
                   ]
      | k == 1 = f (head ts)
      | otherwise = error "empty switch"
      where
        k2 = k `div` 2
        (xs1, xs2) = splitAt k2 xs
```

```
        (ts1, ts2) = splitAt k2 ts

main :: IO ()
main = do
  mpegs <- getArgs
  putStrLn $ dvd mpegs
```

10 cathedral-algorithms/Makefile

```
all:
        echo "choose number of segments in Cathedral.hs and run 'make DVD'"

clean:
        -rm -f Cathedral Cathedral.hi Cathedral.o Snapshot.hi Snapshot.o green.rgba orange.rgba white.rgba interleave31 MakeDVD MakeDVD.o MakeDVD.hi dvd.stamp m2v.stamp mp2.stamp mpeg.stamp ppm.stamp txt.stamp wav.stamp transitions.png transitions.dot

Cathedral: Cathedral.hs Snapshot.hs
        ghc -O2 -Wall --make Cathedral.hs

interleave31: interleave31.c
        gcc -std=c99 -O2 -Wall -pedantic -s -o interleave31 interleave31.c

%.rgba: tile.svg interleave31 svg2rgba.sh
        ./svg2rgba.sh $*

MakeDVD: MakeDVD.hs
        ghc -O2 -Wall --make MakeDVD.hs

ppm.stamp txt.stamp: Cathedral green.rgba orange.rgba white.rgba
        ./Cathedral
        touch ppm.stamp
        touch txt.stamp

wav.stamp: txt.stamp audio.sh cathedral.pd organ-pipe.pd organ-stop.pd
        ./audio.sh
        touch wav.stamp

m2v.stamp: ppm.stamp encode_video.sh
        ./encode_video.sh out/*.ppm
        touch m2v.stamp

mp2.stamp: wav.stamp encode_audio.sh
        ./encode_audio.sh out/*.wav
        touch mp2.stamp

mpeg.stamp: m2v.stamp mp2.stamp multiplex.sh
        ./multiplex.sh out/*.m2v
        touch mpeg.stamp

transitions.png: txt.stamp transitions.sh
        ./transitions.sh

out/menu.mpeg: mpeg.stamp
        ls -1 out/ | grep '.mpeg$$' | head -n 1 | xargs -I MPEG ln -s MPEG out/menu.mpeg
```

```
45
    out/dvd.xml: MakeDVD mpeg.stamp out/menu.mpeg
            ls -1 out/ | grep '.mpeg$$' | grep -v "^menu" | xargs ./MakeDVD > out/↙
                ↳ dvd.xml

    dvd.stamp: out/dvd.xml
50          cd out && rm -rf dvd && mkdir dvd && VIDEO_FORMAT="PAL" dvdauthor -o dvd↙
                ↳  -x dvd.xml
            touch dvd.stamp

    out/dvd.iso: dvd.stamp
            cd out/dvd && genisoimage -dvd-video -o ../dvd.iso .
55
    DVD: out/dvd.iso transitions.png
            cp transitions.png out/dvd.png
            cp transitions.dot out/dvd.dot
            ls -1sh out/dvd.iso
```

11 cathedral-algorithms/multiplex.sh

```
#!/bin/bash
for M2V in $@
do
  MP2="${M2V%m2v}mp2"
5   MPEG="${M2V%m2v}mpeg"
  mplex -f 8 -V -o "${MPEG}" "${M2V}" "${MP2}"
done
```

12 cathedral-algorithms/organ-pipe.pd

```
#N canvas 0 0 365 576 10;
#X obj 23 466 throw~ \$1-left;
#X obj 184 462 throw~ \$1-right;
#X obj 64 414 cos~;
5 #X obj 64 435 *~;
#X obj 183 434 *~;
#X obj 198 372 -~ 0.25;
#X obj 198 413 cos~;
#X obj 111 157 vline~;
10 #X obj 65 21 loadbang;
#X obj 65 47 spigot 1;
#X msg 129 48 0;
#X obj 65 145 vline~;
#X obj 13 532 switch~;
15 #X obj 65 81 list append \$2 \$3;
#X obj 65 106 unpack f f;
#X obj 80 197 expr~ pow(2 \, $v1);
#X obj 65 260 *~;
#X obj 12 82 f 1;
20 #X obj 157 155 + 4;
#X obj 64 369 clip~ -0.25 0.25;
#X obj 198 391 clip~ -0.25 0.25;
#X msg 111 127 0 \, 60 60000;
#X obj 157 176 send \$0-harmonic;
25 #X obj 202 291 send~ \$0-pitch;
#X obj 104 246 catch~ \$0-stop;
```

```
#X obj 201 10 organ-stop \$0 \$1 0;
#X obj 202 226 +~ 8;
#X obj 118 504 change 1;
#X obj 118 460 env~ 8192;
#X obj 201 30 organ-stop \$0 \$1 1;
#X obj 201 50 organ-stop \$0 \$1 2;
#X obj 201 70 organ-stop \$0 \$1 3;
#X obj 201 90 organ-stop \$0 \$1 4;
#X obj 201 110 organ-stop \$0 \$1 5;
#X obj 201 130 organ-stop \$0 \$1 6;
#X obj 201 150 organ-stop \$0 \$1 7;
#X obj 104 266 *~;
#X obj 65 342 +~ 0.125;
#X obj 105 287 expr~ tanh($v1);
#X obj 106 308 *~;
#X obj 202 203 *~ -1;
#X obj 142 223 /~ 64;
#X obj 203 258 max~ 0;
#X obj 118 482 > 10;
#X obj 143 266 sqrt~;
#X connect 2 0 3 0;
#X connect 2 0 28 0;
#X connect 3 0 0 0;
#X connect 4 0 1 0;
#X connect 5 0 20 0;
#X connect 6 0 4 1;
#X connect 6 0 28 0;
#X connect 7 0 15 0;
#X connect 7 0 40 0;
#X connect 8 0 9 0;
#X connect 9 0 10 0;
#X connect 9 0 13 0;
#X connect 9 0 21 0;
#X connect 9 0 17 0;
#X connect 10 0 9 1;
#X connect 11 0 16 0;
#X connect 13 0 14 0;
#X connect 14 0 11 0;
#X connect 14 1 18 0;
#X connect 15 0 16 1;
#X connect 15 0 41 0;
#X connect 16 0 37 0;
#X connect 17 0 12 0;
#X connect 18 0 22 0;
#X connect 19 0 2 0;
#X connect 20 0 6 0;
#X connect 21 0 7 0;
#X connect 24 0 36 0;
#X connect 26 0 42 0;
#X connect 27 0 12 0;
#X connect 28 0 43 0;
#X connect 36 0 38 0;
#X connect 37 0 5 0;
#X connect 37 0 19 0;
#X connect 38 0 39 0;
#X connect 39 0 3 1;
#X connect 39 0 4 0;
```

```
#X connect 40 0 26 0;
#X connect 41 0 36 1;
#X connect 41 0 44 0;
#X connect 42 0 23 0;
#X connect 43 0 27 0;
#X connect 44 0 39 1;
```

13 cathedral-algorithms/organ-stop.pd

```
#N canvas 375 194 450 300 10;
#X obj 67 19 receive~ \$2-phase;
#X obj 223 20 receive \$1-harmonic;
#X obj 68 46 *~ 0;
#X obj 223 103 *;
#X obj 223 42 t b f;
#X obj 223 62 f 1;
#X obj 223 82 << \$3;
#X obj 68 73 cos~;
#X obj 67 229 *~;
#X obj 100 65 receive~ \$1-pitch;
#X obj 100 88 -~ \$3;
#X obj 99 113 *~;
#X obj 83 155 sig~ 1;
#X obj 83 176 -~;
#X obj 67 264 throw~ \$1-stop;
#X obj 82 206 max~ 0;
#X obj 99 135 *~ 0.125;
#X connect 0 0 2 0;
#X connect 1 0 4 0;
#X connect 2 0 7 0;
#X connect 3 0 2 1;
#X connect 4 0 5 0;
#X connect 4 1 3 1;
#X connect 5 0 6 0;
#X connect 6 0 3 0;
#X connect 7 0 8 0;
#X connect 8 0 14 0;
#X connect 9 0 10 0;
#X connect 10 0 11 0;
#X connect 10 0 11 1;
#X connect 11 0 16 0;
#X connect 12 0 13 0;
#X connect 13 0 15 0;
#X connect 15 0 8 1;
#X connect 16 0 13 1;
```

14 cathedral-algorithms/README

```
0.  prerequisites:
        make
        bash (ls,rm,cp,ln,mkdir,touch,sed,grep,xargs,head,tail,sort,uniq)
        gcc
        ghc-6.12 (or later, and "cabal install GLUT")
        puredata-0.42-5 (or later)
        twolame
        rsvg-convert
```

```
        pngtopnm
        ppmtoy4m
        y4mscaler
        mpeg2enc
        mplex
        dvdauthor
        genisoimage
        graphviz

1.  edit Cathedral.hs to set segments (1 to 5 are sane values)
2.  $ ln -s /filesystem/with/lots/of/space/ ./out
3.  $ make DVD
4.  go outside and enjoy the real world for a bit
5.  $ vlc out/dvd.iso
```

15 cathedral-algorithms/Snapshot.hs

```
module Snapshot (hSnapshot, writeSnapshot, snapshotWith) where

import Control.Monad(forM_)
import System.IO(Handle())
import Graphics.UI.GLUT(
  readPixels,
  Position,
  Size(Size),
  PixelData(PixelData),
  PixelFormat(RGB),
  DataType(UnsignedByte))
import Foreign.Marshal.Alloc(allocaBytes)
import Foreign.Ptr(plusPtr)
import qualified Data.ByteString.Internal as BSI
import qualified Data.ByteString as BS

-- save a screenshot to a handle as binary PPM
snapshotWith :: (BS.ByteString -> IO b) -> Position -> Size -> IO b
snapshotWith f p0 vp@(Size vw vh) = do
  let fi q = fromIntegral q
      p6 = "P6\n" ++ show vw ++ " " ++ show vh ++ " 255\n"
  allocaBytes (fi (vw*vh*3)) $ \ptr -> do
    readPixels p0 vp $ PixelData RGB UnsignedByte ptr
    px <- BSI.create (fi $ vw * vh * 3) $ \d -> forM_ [0..vh-1] $ \y ->
      BSI.memcpy
        (d`plusPtr`fi(y*vw*3))
        (ptr`plusPtr`fi ((vh-1-y)*vw*3))
        (fi(vw*3))
    f $ BS.pack (map (toEnum . fromEnum) p6) `BS.append` px

hSnapshot :: Handle -> Position -> Size -> IO ()
hSnapshot h = snapshotWith (BS.hPutStr h)

writeSnapshot :: FilePath -> Position -> Size -> IO ()
writeSnapshot f = snapshotWith (BS.writeFile f)
```

16 cathedral-algorithms/svg2rgba.sh

```
#!/bin/bash
```

```
COLOUR="${1}"
SVGFILE="${1}.svg"
PNGFILE="${1}.png"
PPMFILE="${1}.ppm"
PGMFILE="${1}.pgm"
RGBFILE="${1}.rgb"
AFILE="${1}.a"
RGBAFILE="${1}.rgba"
sed "s|\$COLOUR|${COLOUR}|" < tile.svg > "${SVGFILE}"
rsvg-convert     "${SVGFILE}" > "${PNGFILE}"
pngtopnm         "${PNGFILE}" > "${PPMFILE}"
pngtopnm -alpha "${PNGFILE}" > "${PGMFILE}"
tail -c 3145728 "${PPMFILE}" > "${RGBFILE}"
tail -c 1048576 "${PGMFILE}" > "${AFILE}"
./interleave31 "${RGBFILE}" "${AFILE}" > "${RGBAFILE}"
rm -f "${SVGFILE}" "${PNGFILE}" "${PPMFILE}" "${PGMFILE}" "${RGBFILE}" "${AFILE}
    ↳ }"
```

17 cathedral-algorithms/tile.svg

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN" "http://www.w3.org/Graphics/SVG
    ↳ /1.1/DTD/svg11.dtd">
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/
    ↳ xlink"
  width="1024" height="1024" viewbox="0 0 1024 1024"
><title>Cathedral Algorithms</title
><desc>one tile of many</desc
><g><path
    stroke="red"
    stroke-width="16"
    fill="$COLOUR"
    d="M 256 1024 A 768 768 0 0 1 512 451.5665977600538 A 768 768 0 0 1 768 1024 L
        ↳  768 1280 L 256 1280 L 256 1024 Z"
/></g
></svg>
```

18 cathedral-algorithms/transitions.sh

```
#!/bin/bash
(
  echo 'digraph G {'
  echo '  node[label="",shape="circle",style="filled"];'
  edges=$(ls -1 out | grep ".txt$" | sed "s|.txt$||")
  nodes=$(for edge in ${edges} ; do echo ${edge} ; done | sed "s|_-_.*$||" |
      ↳ sort | uniq)
  for node in ${nodes}
  do
    colour=$(echo ${node} | sed "s|_.*$||")
    case ${colour} in
      0)
        echo "node[fillcolor=\"green\"]; node_${node};"
        ;;
      1)
        echo "node[fillcolor=\"white\"]; node_${node};"
        ;;
```

```
      2)
        echo "node[fillcolor=\"orange\"]; node_${node};"
        ;;
    esac
  done
  for edge in ${edges}
  do
    source=$(echo ${edge} | sed "s|_-_.*$||")
    sink=$(echo ${edge} | sed "s|^.*_-_||")
    echo "node_${source} -> node_${sink};"
  done
  echo '}'
) > transitions.dot
circo -Tpng < transitions.dot > transitions.png
```

19 chladni-plate/chladniplate.c

```
// ./chladniplate # rt
// ./chladniplate input.wav | ffmpeg  -i input.wav -framerate 60 -i - -pix_fmt ↙
    ↪ yuv420p -profile:v high -level:v 4.1 -crf:v 18 -b:a 256k output.mkv

#include <complex.h>
#include <math.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <fftw3.h>
#include <sndfile.h>
#include <jack/jack.h>
#include <GL/glew.h>
#include <GLFW/glfw3.h>

// calculation dimensions
#define W 320
#define H 180

// upscaling factor
#define AA 4

// RGB image data
static unsigned char g[H*AA][W*AA][3];

typedef float vec3[3];
typedef float vec4[4];
typedef float _Complex cmat4[4][4];

// construct a cubic interpolation vector
static inline void cubic(vec4 *o, float x) {
  float xx = x * x;
  (*o)[0] = 0.5f * (x * ((2 - x) * x - 1));
  (*o)[1] = 0.5f * (xx * (3 * x - 5) + 2);
  (*o)[2] = 0.5f * (x * ((4 - 3 * x) * x + 1));
  (*o)[3] = 0.5f * xx * (x - 1);
}
```

```
40  // row * matrix * column
    static inline float _Complex vmv(vec4 *l, cmat4 *m, vec4 *r) {
      float _Complex s = 0;
      for (int i = 0; i < 4; ++i) {
        for (int j = 0; j < 4; ++j) {
45        s += (*l)[i] * (*m)[i][j] * (*r)[j];
        }
      }
      return s;
    }
50
    // linear blending
    static inline void mix(vec3 *o, vec3 *a, vec3 *b, float x) {
      float x1 = 1 - x;
      (*o)[0] = (*a)[0] * x1 + x * (*b)[0];
55    (*o)[1] = (*a)[1] * x1 + x * (*b)[1];
      (*o)[2] = (*a)[2] * x1 + x * (*b)[2];
    }

    // avoid array out of bounds
60  static inline int clamp(int i, int lo, int hi) {
      if (i < lo) { return lo; }
      if (i > hi) { return hi; }
      return i;
    }
65
    // speed of sound
    static const float c = 330; // m s^-1

    // dimension of plate
70  static const float L = 10; // m

    // attenuation
    #define A 0.5

75  // radius (-2 / log10(A) + 1) ?
    #define R 8

    // number of FFT bins
    #define B 16384
80
    static const float pi = 3.141592653589793;

    // compute phasor at a pixel
    static inline float _Complex phasor(float x, float y, float _Complex w)
85  {
      double _Complex s = 0;
      for (int i = -R; i <= R; ++i)
      {
        for (int j = -R; j <= R; ++j)
90      {
          float dx = x + (float) i * W / H;
          float dy = y + j;
          float d = sqrtf(dx * dx + dy * dy);
          s += cexpf(d * w);
95      }
      }
```

```
  return s;
}

// sound file handle
static SNDFILE *f = 0;

// sample rate (set from input file)
static float SR = 44100;

// video framerate
static float FPS = 60;

// number of peak sinusoids to plate
#define PEAKS 12

// FFT hop length
static int HOP = 735;

// number of channels in sound file
static int CHANNELS = 2;

// window for FFT analysis
static float window[B];

// open sound file and initialize parameters
static bool open_input(const char *file)
{
  SF_INFO info = { 0, 0, 0, 0, 0, 0 };
  f = sf_open(file, SFM_READ, &info);
  if (! f) return false;
  SR = info.samplerate;
  CHANNELS = info.channels;
  HOP = roundf(SR / FPS);
  return info.channels > 0;
}

// read and window a frame of input ready for FFT
static bool read_input(float *fft_in)
{
  float q[B][CHANNELS];
  if (B == sf_readf_float(f, &q[0][0], B))
  {
    sf_seek(f, HOP - B, SEEK_CUR);
    for (int i = 0; i < B; ++i)
    {
      fft_in[i] = q[i][0] * window[i];
    }
    return true;
  }
  return false;
}

// close the input file
static void close_input(void)
{
  if (f)
  {
```

```
      sf_close(f);
      f = 0;
    }
  }

  // output PPM to stdout
  static void output(void)
  {
    fprintf(stdout, "P6\n%d %d\n255\n", W * AA, H * AA);
    fwrite(&g[0][0][0], H * AA * W * AA * 3, 1, stdout);
    fflush(stdout);
  }

  // tau function for FFT peak estimator
  static float tau(float x)
  {
    return 1.f/4 * logf(3*x*x + 6*x + 1) - sqrtf(6)/24 * logf((x + 1 - sqrtf(2.f/3)) / (x + 1 + sqrtf(2.f/3)));
  }

  // FFT peak estimator
  static float quinn2(int k, float _Complex *X)
  {
    float x = crealf(X[k]) * crealf(X[k]) + cimagf(X[k]) * cimagf(X[k]);
    float ap = (crealf(X[k + 1]) * crealf(X[k]) + cimagf(X[k + 1]) * cimagf(X[k])) / x;
    float dp = -ap / (1 - ap);
    float am = (crealf(X[k - 1]) * crealf(X[k]) + cimagf(X[k - 1]) * cimagf(X[k])) / x;
    float dm = am / (1 - am);
    float d = (dp + dm) / 2 + tau(dp * dp) - tau(dm * dm);
    return k + d;
  }

  // search for largest peak and zero it
  static float peak(float _Complex *X, float _Complex *r_out)
  {
    float ma = 0.0;
    int k = 0;
    float _Complex mx = 0;
    for (int b = 1; b < B/2; ++b)
    {
      float x = crealf(X[b]);
      float y = cimagf(X[b]);
      float r = x * x + y * y;
      if (r > ma)
      {
        ma = r;
        k = b;
        mx = X[b];
      }
    }
    if (k == 0)
    {
      *r_out = 0;
      return 0;
    }
```

```
    else
    {
      *r_out = mx;
      float p = quinn2(k, X);
//    X[k - 1] = 0;
      X[k    ] = 0;
//    X[k + 1] = 0;
      return p;
    }
  }

  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 len = 0;
      glGetProgramiv(program, GL_INFO_LOG_LENGTH, &len);
      char *s = calloc(1, len + 1);
      if (! s) abort();
      glGetProgramInfoLog(program, len, 0, s);
      s[len] = 0;
      if (*s)
        fprintf(stderr, "%s: OpenGL shader program info log\n%s\n", name, s);
      free(s);
    }
    else
    {
      fprintf(stderr, "%s: OpenGL shader program creation failed\n", name);
    }
  }

  void debug_shader(GLuint shader, GLenum typ, const char *name)
  {
    const char *tname = "unknown";
    if (typ == GL_VERTEX_SHADER) tname = "vertex";
    if (typ == GL_FRAGMENT_SHADER) tname = "fragment";
    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 len = 0;
      glGetShaderiv(shader, GL_INFO_LOG_LENGTH, &len);
      char *s = calloc(1, len + 1);
      if (! s) abort();
      glGetShaderInfoLog(shader, len, 0, s);
      s[len] = 0;
      if (*s) fprintf(stderr, "%s: OpenGL %s shader info log\n%s\n", name, tname, ↙
          ↳ s);
      free(s);
    }
    else
```

```
  {
    fprintf(stderr, "%s: OpenGL %s shader creation failed\n", name, tname);
  }
}

void compile_shader(GLuint program, GLenum typ, const char *name, const char *source)
{
  GLuint shader = glCreateShader(typ);
  glShaderSource(shader, 1, &source, 0);
  glCompileShader(shader);
  debug_shader(shader, typ, name);
  glAttachShader(program, shader);
  glDeleteShader(shader);
}

GLuint compile_program(const char *name, const char *vert, const char *frag)
{
  GLuint program = glCreateProgram();
  compile_shader(program, GL_VERTEX_SHADER, name, vert);
  compile_shader(program, GL_FRAGMENT_SHADER, name, frag);
  glLinkProgram(program);
  debug_program(program, name);
  return program;
}

const char *chladni_vert =
"#version 330 core\n"
"void main() {\n"
"  if (gl_VertexID == 0) { gl_Position = vec4(-1.0, -1.0, 0.0, 1.0); } else\n"
"  if (gl_VertexID == 1) { gl_Position = vec4( 1.0, -1.0, 0.0, 1.0); } else\n"
"  if (gl_VertexID == 2) { gl_Position = vec4(-1.0,  1.0, 0.0, 1.0); } else\n"
"                        { gl_Position = vec4( 1.0,  1.0, 0.0, 1.0); }\n"
"}\n"
;

const char *chladni_frag =
"#version 330 core\n"
"#extension GL_ARB_explicit_uniform_location : require\n"
"#define PEAKS 12\n"
"layout (location = 0) uniform vec2 size;\n"
"layout (location = 1) uniform int sides;\n"
"layout (location = 2) uniform vec4 peaks[PEAKS];\n"
"vec2 cmul(vec2 a, vec2 b) { return vec2(a.x * b.x - a.y * b.y, a.x * b.y + a.y * b.x); }\n"
"vec2 cdiv(vec2 a, vec2 b) { return cmul(a, vec2(b.x, -b.y)) / dot(b, b); }\n"
"vec2 cexp(vec2 t) { return exp(t.x) * vec2(cos(t.y), sin(t.y)); }\n"
"vec2 phasor(vec2 q, vec2 w) {\n"
"  vec2 s = vec2(0.0);\n"
"  if (sides == 4) {\n"
"    for (int i = -8; i <= 8; ++i) {\n"
"      float dx = q.x + float(i) * size.x / size.y;\n"
"      for (int j = -8; j <= 8; ++j) {\n"
"        float dy = q.y + float(j);\n"
"        float d = sqrt(dx * dx + dy * dy);\n"
"        s += cexp(d * w);\n"
"      }\n"
```

```
"    }\n"
"  }\n"
"  if (sides == 6) {\n"
"    for (int j = -8; j <= 8; ++j) {\n"
"      float dy0 = q.y + float(j);\n"
"      for (int i = -8; i <= 8; ++i) {\n"
"        float dx = q.x + float(i) * sqrt(3.0)/2.0;\n"
"        float dy = dy0 + (1 == (i & 1) ? 0.5 : 0.0);\n"
"        float d = sqrt(dx * dx + dy * dy);\n"
"        s += cexp(d * w);\n"
"      }\n"
"    }\n"
"  }\n"
"  return s;\n"
"}\n"
"vec2 phasors(vec2 q) {\n"
"  vec2 s = vec2(0.0);\n"
"  for (int p = 0; p < PEAKS; ++p) {\n"
"    vec2 a = peaks[p].xy;\n"
"    vec2 w = peaks[p].zw;\n"
"    s += length(a) * phasor(q, w);\n"
"  }\n"
"  return s;\n"
"}\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 q = gl_FragCoord.xy / size - vec2(0.5);\n"
"  q.x *= size.x / size.y;\n"
"  vec2 s = cdiv(phasors(q), normalize(phasors(vec2(0.0))));\n"
"  float hue = atan(s.y, s.x) / (2.0 * 3.141592653);\n"
"  hue -= floor(hue);\n"
"  float sat = clamp(2.0 - length(s), 0.0, 1.0);\n"
"  float val = clamp(length(s), 0.0, 1.0);\n"
"  gl_FragColor = vec4(hsv2rgb(vec3(hue, sat, val)), 1.0);\n"
"}\n"
;

void keycb(GLFWwindow *win, int key, int scancode, int action, int mods)
{
  (void) key;
  (void) scancode;
  (void) action;
  (void) mods;
  glfwSetWindowShouldClose(win, GL_TRUE);
}

static int wpeaks = 0;
static vec4 peaks[2][PEAKS];

static float abuffer[B];
static int awptr = 0;
static int acount = 0;
```

```
static float *fft_in;
static float _Complex *fft_out;
static fftwf_plan plan;

static jack_client_t *client;
static jack_port_t *input;

static int processcb(jack_nframes_t nframes, void *arg)
{
  (void) arg;
  jack_default_audio_sample_t *in = jack_port_get_buffer(input, nframes);
  for (jack_nframes_t i = 0; i < nframes; ++i)
  {
    abuffer[awptr++] = in[i];
    if (awptr >= B) awptr = 0;
    if (++acount == HOP)
    {
      acount = 0;
      for (int j = 0; j < B; ++j)
      {
        fft_in[j] = window[j] * abuffer[(awptr + j) % B];
      }
      fftwf_execute(plan);
      static float r0 = 0;
      for (int k = 0; k < PEAKS; ++k)
      {
        float _Complex r = 0;
        float bin = peak(fft_out, &r);
        r0 = fmaxf(r0, cabsf(r));
        float f = SR * bin / B;
        float wavelength = c / f / L;
        float _Complex w = logf(A) + I * 2 * pi / wavelength;
        if (r0 > 0) r = 0.25 * r / r0;
        peaks[wpeaks][k][0] = crealf(r);
        peaks[wpeaks][k][1] = cimagf(r);
        peaks[wpeaks][k][2] = crealf(w);
        peaks[wpeaks][k][3] = cimagf(w);
      }
      wpeaks = 1 - wpeaks;
    }
  }
  return 0;
}

// program entry point
int main(int argc, char **argv)
{
  // prepare FFT
  fft_in = fftwf_malloc(B * sizeof(*fft_in));
  fft_out = fftwf_malloc(B * sizeof(*fft_out));
  plan = fftwf_plan_dft_r2c_1d(B, fft_in, fft_out, FFTW_PATIENT | ↙
      ↳ FFTW_PRESERVE_INPUT);
  for (int i = 0; i < B; ++i)
  {
    window[i] = (1 - cos(2 * pi * (i + 0.5f) / B)) / B;
  }
  // process audio
```

```
430     if (argc == 1)
        {
          if (!(client = jack_client_open("chladni", JackNoStartServer, 0))) {
            fprintf(stderr, "jack server not running?\n");
            return 1;
435       }
          jack_set_process_callback(client, processcb, 0);
          // mono processing
          input = jack_port_register(client, "input_1", JACK_DEFAULT_AUDIO_TYPE, ↙
              ↳ JackPortIsInput, 0);
          if (jack_activate(client)) {
440         fprintf (stderr, "cannot activate JACK client");
            return 1;
          }
        }
        else
445     {
          fprintf(stderr, "%s: non-realtime mode not yet re-implemented\n", argv[0]);
          return 1;
        }
        // visualize
450     glfwInit();
        glfwWindowHint(GLFW_CLIENT_API, GLFW_OPENGL_API);
        glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
        glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
        glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
455     glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
        glfwWindowHint(GLFW_DECORATED, GL_TRUE);
        glfwWindowHint(GLFW_RESIZABLE, GL_FALSE);
        GLFWwindow *win = glfwCreateWindow(W * AA, H * AA, "chladniplate", 0, 0);
        glfwMakeContextCurrent(win);
460     glewExperimental = GL_TRUE;
        glewInit();
        glGetError(); // discard common error from glew
        glfwSetInputMode(win, GLFW_CURSOR, GLFW_CURSOR_HIDDEN);
        glfwSetKeyCallback(win, keycb);
465     GLuint vao;
        glGenVertexArrays(1, &vao);
        glBindVertexArray(vao);
        glUseProgram(compile_program("chladni", chladni_vert, chladni_frag));
        GLuint tex;
470     glGenTextures(1, &tex);
        glBindTexture(GL_TEXTURE_2D, tex);
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, W, H, 0, GL_RGB, GL_UNSIGNED_BYTE, 0);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
475     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
        GLuint fbo;
        glGenFramebuffers(1, &fbo);
        glBindFramebuffer(GL_FRAMEBUFFER, fbo);
480     glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, ↙
            ↳ tex, 0);
        glUniform2f(0, W, H);
        glUniform1i(1, 6);
        while (1)//read_input(fft_in))
        {
```

```
485       glUniform4fv(2, PEAKS, &peaks[1-wpeaks][0][0]);
          glViewport(0, 0, W, H);
          glBindFramebuffer(GL_FRAMEBUFFER, fbo);
          glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);
          glBindFramebuffer(GL_DRAW_FRAMEBUFFER, 0);
490   //      glViewport(0, 0, W * AA, H * AA);
          glBlitFramebuffer(0, 0, W, H, 0, 0, W * AA, H * AA, GL_COLOR_BUFFER_BIT, ↙
              ↪ GL_LINEAR);
           //glReadPixels(0, 0, W * AA, H * AA, GL_RGB, GL_UNSIGNED_BYTE, &g[0][0][0])↙
               ↪ ;
          glfwSwapBuffers(win);
          //output();
495       glfwPollEvents();
          if (glfwWindowShouldClose(win)) {
            break;
          }
          GLenum e;
500       while ((e = glGetError()))
          {
            fprintf(stderr, "%d\n", e);
          }
        }
505     // cleanup
        fftwf_destroy_plan(plan);
        fftwf_free(fft_out);
        fftwf_free(fft_in);
        close_input();
510     return 0;
    }
```

20 chladni-plate/chladniplate-eigensystem.m

```
    % tested with GNU Octave, version 4.4.1-rc2

    % number of modes to compute
    e = 25;
5
    % height of plate
    m = 256;

    % width of plate
10  n = 256;

    % kernel (harmonic operator, laplacian)
    %kr = 1;
    %kernel = [ 0,1,0; 1,-4,1; 0,1,0 ];
15  % kernel (biharmonic operator, laplacian of laplacian)
    kr = 2;
    kernel = [ 0,0,1,0,0; 0,2,-8,2,0; 1,-8,20,-8,1; 0,2,-8,2,0; 0,0,1,0,0 ];

    % compute number of non-zero values
20  nz = 0;
    for i = 1 : n
      for j = 1 : m
        k = zeros(2 * kr + 1, 2 * kr + 1);
        for di = -kr : kr
25        for dj = -kr : kr
```

```
          kk = kernel(di + kr + 1, dj + kr + 1);
          ii = di;
          jj = dj;
          if (i + ii < 1)
            ii = -ii;
          endif
          if (i + ii > n)
            ii = -ii;
          endif
          if (j + jj < 1)
            jj = -jj;
          endif
          if (j + jj > m)
            jj = -jj;
          endif
          if (ii != di || jj != dj)
            %kk = -kk;
          endif
          k(ii + kr + 1, jj + kr + 1) = k(ii + kr + 1, jj + kr + 1) + kk;
        endfor
      endfor
      for ii = 1 : 2 * kr + 1
        for jj = 1 : 2 * kr + 1
          if (k(ii, jj) != 0)
            nz = nz + 1;
          endif
        endfor
      endfor
    endfor
  endfor

  pz = nz / (n * m)^2;
  disp(sprintf("sparsity: %f%%", 100 * pz));

  % compute sparse matrix
  t = zeros(nz, 3);
  s = 1;
  for i = 1 : n
    for j = 1 : m
      k = zeros(2 * kr + 1, 2 * kr + 1);
      for di = -kr : kr
        for dj = -kr : kr
          kk = kernel(di + kr + 1, dj + kr + 1);
          ii = di;
          jj = dj;
          if (i + ii < 1)
            ii = -ii;
          endif
          if (i + ii > n)
            ii = -ii;
          endif
          if (j + jj < 1)
            jj = -jj;
          endif
          if (j + jj > m)
            jj = -jj;
          endif
```

```
          if (ii != di || jj != dj)
            %kk = -kk;
          endif
          k(ii + kr + 1, jj + kr + 1) = k(ii + kr + 1, jj + kr + 1) + kk;
        endfor
      endfor
      for ii = -kr : kr
        for jj = -kr : kr
          if (k(ii + kr + 1, jj + kr + 1) != 0)
            t(s, 1) = (i - 1) * m + j;
            t(s, 2) = (i - 1 + ii) * m + j + jj;
            t(s, 3) = k(ii + kr + 1, jj + kr + 1);
            s = s + 1;
          endif
        endfor
      endfor
    endfor
  endfor
  s = spconvert(t);
  disp("constructed matrix");

  % solve the eigen system
  opts.maxit = 5000;
  [v,ls,f] = eigs(s, e, "sm", opts);
  l = diag(ls);
  disp("solved eigen system");

  % save the raw data
  fid = fopen(sprintf("chladni-%d-%d.dat", m, n), "w", "native");
  fwrite(fid, l, "float32", "native");
  fwrite(fid, v, "float32", "native");
  fclose(fid);
  disp("saved raw data");
```

21 chladni-plate/chladniplate-visualize.c

```
// gcc -std=c99 -Wall -pedantic -Wextra -O3 -fopenmp -march=native modes.c -lm

#include <math.h>
#include <stdio.h>

// number of modes
#define E 25

// height of plate
#define M 256

// width of plate
#define N 256

// supersizing
#define A 2

// eigen values
float l[E];

// eigen vectors
```

```
float v[M][N];

// supersized vector
float z[M*A][N*A];

// RGB image
unsigned char g[M*A][N*A];

typedef float vec3[3];
typedef float vec4[4];
typedef float mat4[4][4];

// construct a cubic interpolation vector
inline void cubic(vec4 *o, float x) {
  float xx = x * x;
  (*o)[0] = 0.5f * (x * ((2 - x) * x - 1));
  (*o)[1] = 0.5f * (xx * (3 * x - 5) + 2);
  (*o)[2] = 0.5f * (x * ((4 - 3 * x) * x + 1));
  (*o)[3] = 0.5f * xx * (x - 1);
}

// row * matrix * column
inline float vmv(vec4 *l, mat4 *m, vec4 *r) {
  float s = 0;
  for (int i = 0; i < 4; ++i) {
    for (int j = 0; j < 4; ++j) {
      s += (*l)[i] * (*m)[i][j] * (*r)[j];
    }
  }
  return s;
}

// linear blending
inline void mix(vec3 *o, vec3 *a, vec3 *b, float x) {
  float x1 = 1 - x;
  (*o)[0] = (*a)[0] * x1 + x * (*b)[0];
  (*o)[1] = (*a)[1] * x1 + x * (*b)[1];
  (*o)[2] = (*a)[2] * x1 + x * (*b)[2];
}

// avoid array out of bounds
inline int clamp(int i, int lo, int hi) {
  if (i < lo) { return lo; }
  if (i > hi) { return hi; }
  return i;
}

// main program
int main() {

  // read eigen values
  fread(l, E * sizeof(float), 1, stdin);
  for (int k = 0; k < E; ++k) {

    // read eigen vector
    fread(v, M * N * sizeof(float), 1, stdin);
```

```
    // upscale with bicubic interpolation
80  #pragma omp parallel for schedule(static, 1)
    for (int x = 0; x < A; ++x) {
      float xf = (x + 0.5f) / A;
      vec4 xc;
      cubic(&xc, xf);
85    for (int y = 0; y < A; ++y) {
        float yf = (y + 0.5f) / A;
        vec4 yc;
        cubic(&yc, yf);
        for (int i = 0; i < M; ++i) {
90        int i0 = clamp(i - 1, 0, M - 1);
          int i1 = clamp(i    , 0, M - 1);
          int i2 = clamp(i + 1, 0, M - 1);
          int i3 = clamp(i + 2, 0, M - 1);
          for (int j = 0; j < N; ++j) {
95          int j0 = clamp(j - 1, 0, N - 1);
            int j1 = clamp(j    , 0, N - 1);
            int j2 = clamp(j + 1, 0, N - 1);
            int j3 = clamp(j + 2, 0, N - 1);
            mat4 m =
100           { { v[i0][j0], v[i0][j1], v[i0][j2], v[i0][j3] }
              , { v[i1][j0], v[i1][j1], v[i1][j2], v[i1][j3] }
              , { v[i2][j0], v[i2][j1], v[i2][j2], v[i2][j3] }
              , { v[i3][j0], v[i3][j1], v[i3][j2], v[i3][j3] }
              };
105         z[i * A + x][j * A + y] = vmv(&xc, &m, &yc);
          }
        }
      }
    }
110
    // colourize in groups of 2x2 pixels
    #pragma omp parallel for
    for (int x = 0; x < M * A; ++x) {
      int x1 = x - 1;
115   if (x == 0) {
        x1 = x + 1;
      }
      for (int y = 0; y < N * A; ++y) {
        int y1 = y - 1;
120     if (y == 0) {
          y1 = y + 1;
        }
        float z00 = z[x][y];
        float z01 = z[x][y1];
125     float z10 = z[x1][y];
        float z11 = z[x1][y1];
        // contours
        float mi = fminf(fminf(fminf(z00, z01), z10), z11);
        float ma = fmaxf(fmaxf(fmaxf(z00, z01), z10), z11);
130     float si = (mi > 0) - (0 > mi);
        float sa = (ma > 0) - (0 > ma);
        int s = si == sa;
        g[x][y] = s * 255;
      }
135 }
```

```
      // save as PPM
      char filename[100];
      snprintf(filename, 100, "chladni_%.18f_%02d.pgm", l[k], k);
      FILE *f = fopen(filename, "wb");
      fprintf(f, "P5\n%d %d\n255\n", N * A, M * A);
      fwrite(g, M * A * N * A, 1, f);
      fclose(f);
    }

  return 0;
}
```

22 chladni-plate/.gitignore

```
chladniplate
chladniplate-visualize
*.mkv
*.wav
*.ogg
*.flac
*.png
*.dat
*.pdf
```

23 chladni-plate/Makefile

```
chladniplate: chladniplate.c
	gcc -std=c99 -Wall -Wextra -pedantic -O3 -march=native -ffast-math -↙
	    ↳ fopenmp -fPIC -o chladniplate chladniplate.c -lm -lsndfile -↙
	    ↳ lfftw3f -lGL -lglfw -lGLEW -ljack
```

24 COPYING

GNU AFFERO GENERAL PUBLIC LICENSE
Version 3, 19 November 2007

Copyright (C) 2007 Free Software Foundation, Inc. <http://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 <http://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 <http://www.gnu.org/licenses/>.

25 cypi/cypi.c

```
/*
    cypi -- audio-visual drone for OpenGL and JACK
    Copyright (C) 2010,2018  Claude Heiland-Allen <claude@mathr.co.uk>

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU 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 <http://www.gnu.org/licenses/>.
*/

/* standard C library */
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

/* OpenGL library */
#include <GL/glew.h>
#include <GL/glut.h>

/* JACK library */
#include <jack/jack.h>

#define twopi 6.283185307179586
#define CYPI_SIZE 1024
#define CYPI_COUNT 48

/* non-realtime mode */
static int cypi_nrt;

/* JACK state */
static struct {
  /* JACK handles */
```

```
  jack_client_t *client;
  jack_port_t *port[2];
  /* audio buffers */
  unsigned int index;
  jack_default_audio_sample_t buffer[2][2][CYPI_SIZE];
  /* DC blocker filters */
  jack_default_audio_sample_t r[2], xn1[2], yn1[2];
  /* nrt things */
  FILE *fl;
  FILE *fr;
} cypi_jack;

/* OpenGL state */
static struct {
  /* frame count */
  int frame;
  unsigned int frameR;
  /* window */
  int winw, winh;
  float scale;
  /* frame buffer */
  GLuint fbo;
  /* texture */
  unsigned int tex;
  int texw, texh;
  /* shader */
  GLcharARB *src;
  GLhandleARB prog;
  GLhandleARB frag;
  GLcharARB *src2;
  GLhandleARB prog2;
  GLhandleARB frag2;
  /* uniforms */
  GLint number;
  GLint amount;
  GLfloat numberm[16];
  GLfloat amountv[4];
  GLint tex2;
  GLint fade2;
  /* nrt things */
  unsigned char *buffer;
  FILE *fv;
  int frame2;
  int frame3;
  int frameT;
  int frameC;
} cypi_gl;

/* synchronisation between JACK callback thread and OpenGL main thread */
static volatile int cypi_which = 0;

/* GLUT display callback */
static void cypi_display() {
  /* modulation amounts based on frame count */
  float z = 0.5 - 0.5 * cos(twopi * cypi_gl.frame / CYPI_SIZE);
  for (int i = 0; i < 4; ++i) {
    cypi_gl.amountv[i] = z;
```

```
100     }
        /* set up view */
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(-1, 1, -1, 1);
105     glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        glViewport(0, 0, cypi_gl.texw, cypi_gl.texh);
        glClear(GL_COLOR_BUFFER_BIT);
        glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, cypi_gl.fbo);
110     /* generate a modulation texture */
        glFramebufferTexture2DEXT(
          GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT, GL_TEXTURE_2D, cypi_gl.tex, 0
        );
        glUseProgramObjectARB(cypi_gl.prog);
115     glUniformMatrix4fvARB(cypi_gl.number, 1, 0, &cypi_gl.numberm[0]);
        glUniform4fvARB(cypi_gl.amount, 1, &cypi_gl.amountv[0]);
        glBegin(GL_QUADS); {
          glColor4f(1,1,1,1);
          glTexCoord4f(0, 0, 0, 0); glVertex2f(-1, -1);
120       glTexCoord4f(1, 0, 1, 0); glVertex2f( 1, -1);
          glTexCoord4f(1, 1, 1, 1); glVertex2f( 1,  1);
          glTexCoord4f(0, 1, 0, 1); glVertex2f(-1,  1);
        } glEnd();
        glUseProgramObjectARB(0);
125     /* read horizontal/vertical scanlines into JACk back buffers */
        glReadPixels(
          0, cypi_gl.frame % cypi_gl.texh, cypi_gl.texw, 1, GL_RED, GL_FLOAT,
          cypi_jack.buffer[0][1-cypi_which]
        );
130     glReadPixels(
          cypi_gl.frame % cypi_gl.texw, 0, 1, cypi_gl.texh, GL_RED, GL_FLOAT,
          cypi_jack.buffer[1][1-cypi_which]
        );
        glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);
135     /* fill the window tiled with the generated texture */
        glViewport(0, 0, cypi_gl.winw, cypi_gl.winh);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(-1, 1, -1, 1);
140     glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        glBindTexture(GL_TEXTURE_2D, cypi_gl.tex);
        glUseProgramObjectARB(cypi_gl.prog2);
        glUniform1iARB(cypi_gl.tex2, 0);
145     float c = cypi_gl.frameR / (float) (CYPI_SIZE/2);
        c = fmin(fmax(c, 0.0), 1.0);
        float c2 = ((CYPI_COUNT+1) * CYPI_SIZE - cypi_gl.frameR) / (float) (CYPI_SIZE↙
            ↳ /2);
        c2 = fmin(fmax(c2, 0.0), 1.0);
        glUniform1fARB(cypi_gl.fade2, c * c2);
150     glBegin(GL_QUADS); {
          float s = cypi_gl.scale;
          float tx0 = (1.0 * cypi_gl.texw - cypi_gl.winw / s) / (1.0 * cypi_gl.texw);
          float ty0 = (2.0 * cypi_gl.texh - cypi_gl.winh / s) / (2.0 * cypi_gl.texh);
          float tx1 = (1.0 * cypi_gl.texw + cypi_gl.winw / s) / (1.0 * cypi_gl.texw);
155       float ty1 = (2.0 * cypi_gl.texh + cypi_gl.winh / s) / (2.0 * cypi_gl.texh);
```

```
      glTexCoord4f(tx0, ty0, 0, 0); glVertex2f(-1, -0.5);
      glTexCoord4f(tx1, ty0, 1, 0); glVertex2f( 1, -0.5);
      glTexCoord4f(tx1, ty1, 1, 1); glVertex2f( 1,  0.5);
      glTexCoord4f(tx0, ty1, 0, 1); glVertex2f(-1,  0.5);
    } glEnd();
    glUseProgramObjectARB(0);
    /* display */
    glutSwapBuffers();
    /* report errors */
    int status = glGetError();
    if(status) fprintf(stderr, "GL: error %08x\n", status);
    /* increment/wrap frame count and randomize modulators */
    if (++cypi_gl.frame == CYPI_SIZE) {
      cypi_gl.frame = 0;
      if (++cypi_gl.frameC > CYPLCOUNT) {
        for (int i = 0; i < 16; ++i) {
          cypi_gl.numberm[i] = 0;
        }
      } else {
        for (int i = 0; i < 16; ++i) {
          int r = rand() % 17;
          cypi_gl.numberm[i] = r - 8;
//          fputc("0123456789abcdefg"[r], stderr);
        }
//        fputc('\n', stderr);
      }
    }
    cypi_gl.frameR++;
    if (cypi_nrt) {
      /* save audio buffer */
      if (cypi_gl.frame2 == 0) {
        fwrite(cypi_jack.buffer[0][1-cypi_which], CYPI_SIZE * sizeof(float), 1, ↙
            ↳ cypi_jack.fl);
        fwrite(cypi_jack.buffer[1][1-cypi_which], CYPI_SIZE * sizeof(float), 1, ↙
            ↳ cypi_jack.fr);
      }
      /* save video buffer */
      if (cypi_gl.frame3 == 0) {
        glReadPixels(0, 0, cypi_gl.winw, cypi_gl.winh, GL_RGB, GL_UNSIGNED_BYTE, ↙
            ↳ cypi_gl.buffer);
        const char *header = "P6\n1280 720 255\n";
        fwrite(header, strlen(header), 1, cypi_gl.fv);
        fwrite(cypi_gl.buffer, cypi_gl.winw * cypi_gl.winh * 3, 1, cypi_gl.fv);
      }
      cypi_gl.frame2 = (cypi_gl.frame2 + 1) % 2;
      cypi_gl.frame3 = (cypi_gl.frame3 + 1) % 3;
      cypi_gl.frameT++;
    }
    if (cypi_gl.frameR == (CYPLCOUNT+1) * CYPI_SIZE) exit(0);
}

/* GLUT reshape callback */
static void cypi_reshape(int w, int h) {
  cypi_gl.winw = w;
  cypi_gl.winh = h;
}
```

```
210 /* GLUT idle callback: render as fast as possible */
    static void cypi_idle() {
      glutPostRedisplay();
    }

215 /* JACK audio processing callback */
    static int cypi_process(jack_nframes_t nframes, void *arg) {
      /* get JACK buffers */
      jack_default_audio_sample_t *out[2];
      out[0] = (jack_default_audio_sample_t *)
220     jack_port_get_buffer(cypi_jack.port[0], nframes);
      out[1] = (jack_default_audio_sample_t *)
        jack_port_get_buffer(cypi_jack.port[1], nframes);
      /* copy buffers to JACK */
      for (int i = 0; i < nframes; ++i) {
225     for (int c = 0; c < 2; ++c) {
          jack_default_audio_sample_t xn =
            cypi_jack.buffer[c][cypi_which][(cypi_jack.index + i) % CYPI_SIZE];
          cypi_jack.yn1[c] =
            xn - cypi_jack.xn1[c] + cypi_jack.r[c] * cypi_jack.yn1[c];
230       cypi_jack.xn1[c] = xn;
          out[c][i] = cypi_jack.yn1[c];
        }
      }
      /* increment/wrap write counter, if done then swap buffers */
235   cypi_jack.index = (cypi_jack.index + nframes) % CYPI_SIZE;
      if (cypi_jack.index == 0) {
        cypi_which = 1 - cypi_which;
      }
      return 0;
240 }

    /* JACK sample rate callback: currently unused */
    static int cypi_srate(jack_nframes_t nframes, void *arg) {
      return 0;
245 }

    /* JACK error callback */
    static void cypi_error(const char *desc) {
      fprintf(stderr, "JACK error: %s\n", desc);
250 }

    /* JACK shutdown callback */
    static void cypi_shutdown(void *arg) {
      exit(1);
255 }

    /* exit callback */
    static void cypi_atexit(void) {
      jack_client_close(cypi_jack.client);
260 }

    /* main program */
    int main(int argc, char **argv) {
      /* bad source of pseudorandomness */
265   unsigned int t = time(NULL);
      printf("%08X\n", t);
```

```
  srand(t);
  /* the CyPi fragment shader source code */
  cypi_gl.src =
    "uniform mat4 number;\n"
    "uniform vec4 amount;\n"
    "const float twopi = 6.283185307179586;\n"
    "void main() {\n"
    "  vec4 p = gl_TexCoord[0];\n"
    "  vec4 q = number * p;\n"
    "  vec4 x = -cos(twopi * (p + amount * sin(twopi * q)*0.5+0.5))*0.5+0.5;\n"
    "  float z = x.x * x.y * x.z * x.w;\n"
    "  gl_FragColor = vec4(z,z,z,1.0);\n"
    "}\n";
  cypi_gl.src2 =
    "#version 130\n"
    "uniform sampler2D tex;\n"
    "uniform float fade;\n"
    "const float twopi = 6.283185307179586;\n"
    "void main() {\n"
    "  vec4 p = gl_TexCoord[0];\n"
    "  float a = tanh((clamp(1.0 - cos(p.z * twopi), 0.0, 1.0)) * 8.0);\n"
    "  float b = tanh((clamp(1.0 - cos(p.w * twopi), 0.0, 1.0)) * 8.0);\n"
    "  gl_FragColor = texture2D(tex, p.xy) * a * b * fade;\n"
    "}\n";
  /* set up OpenGL variables */
  cypi_gl.frame = CYPI_SIZE/2;
  cypi_gl.frameC = 0;
  cypi_gl.winw = 1280;
  cypi_gl.winh = 720;
  cypi_gl.scale = 1280.0 / 1920.0;
  cypi_gl.texw = CYPI_SIZE;
  cypi_gl.texh = CYPI_SIZE;
  /* initialize OpenGL */
  glutInitWindowSize(cypi_gl.winw, cypi_gl.winh);
  glutInit(&argc, argv);
  cypi_nrt = argc > 1;
  glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE);
  glutCreateWindow("CyPi");
  glewInit();
  glClearColor(0.0, 0.0, 0.0, 1.0);
  glShadeModel(GL_FLAT);
  /* allocate frame buffer */
  glGenFramebuffersEXT(1, &cypi_gl.fbo);
  /* allocate texture */
  glGenTextures(1, &cypi_gl.tex);
  glEnable(GL_TEXTURE_2D);
  glBindTexture(GL_TEXTURE_2D, cypi_gl.tex);
  glTexImage2D(
    GL_TEXTURE_2D, 0, GL_RGBA32F_ARB, cypi_gl.texw, cypi_gl.texh, 0, GL_RGBA,
    GL_FLOAT, 0
  );
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
  /* compile shaders */
  GLint success;
  cypi_gl.prog = glCreateProgramObjectARB();
  cypi_gl.frag = glCreateShaderObjectARB(GL_FRAGMENT_SHADER_ARB);
```

```
    glShaderSourceARB(cypi_gl.frag, 1, (const GLcharARB **) &cypi_gl.src, 0);
325 glCompileShaderARB(cypi_gl.frag);
    glAttachObjectARB(cypi_gl.prog, cypi_gl.frag);
    glLinkProgramARB(cypi_gl.prog);
    glGetObjectParameterivARB(cypi_gl.prog, GL_OBJECT_LINK_STATUS_ARB, &success);
    if (!success) exit(1);
330 cypi_gl.prog2 = glCreateProgramObjectARB();
    cypi_gl.frag2 = glCreateShaderObjectARB(GL_FRAGMENT_SHADER_ARB);
    glShaderSourceARB(cypi_gl.frag2, 1, (const GLcharARB **) &cypi_gl.src2, 0);
    glCompileShaderARB(cypi_gl.frag2);
    glAttachObjectARB(cypi_gl.prog2, cypi_gl.frag2);
335 glLinkProgramARB(cypi_gl.prog2);
    glGetObjectParameterivARB(cypi_gl.prog2, GL_OBJECT_LINK_STATUS_ARB, &success);
    if (!success) exit(1);
    /* access uniforms */
    cypi_gl.number = glGetUniformLocationARB(cypi_gl.prog, "number");
340 cypi_gl.amount = glGetUniformLocationARB(cypi_gl.prog, "amount");
    cypi_gl.tex2   = glGetUniformLocationARB(cypi_gl.prog2, "tex");
    cypi_gl.fade2  = glGetUniformLocationARB(cypi_gl.prog2, "fade");
    /* set up callbacks */
    glutDisplayFunc(cypi_display);
345 glutReshapeFunc(cypi_reshape);
    glutIdleFunc(cypi_idle);
    /* initialize DC blocking filters */
    cypi_jack.r[0] = 0.995;
    cypi_jack.xn1[0] = 0;
350 cypi_jack.yn1[0] = 0;
    cypi_jack.r[1] = 0.995;
    cypi_jack.xn1[1] = 0;
    cypi_jack.yn1[1] = 0;
    if (cypi_nrt) {
355   cypi_gl.fv   = fopen("video.ppm",   "wb");
      cypi_jack.fl = fopen("audio_l.raw", "wb");
      cypi_jack.fr = fopen("audio_r.raw", "wb");
      cypi_gl.buffer = malloc(cypi_gl.winw * cypi_gl.winh * 3);
      cypi_gl.frame2 = 0;
360   cypi_gl.frame3 = 0;
      cypi_gl.frameT = 0;
    } else {
      cypi_gl.fv = 0;
      cypi_jack.fl = 0;
365   cypi_jack.fr = 0;
      cypi_gl.buffer = 0;
      /* set up JACK */
      jack_set_error_function(cypi_error);
      if (!(cypi_jack.client = jack_client_open("cypi", 0, 0))) {
370     fprintf (stderr, "jack server not running?\n");
        return 1;
      }
      atexit(cypi_atexit);
      jack_set_process_callback(cypi_jack.client, cypi_process, 0);
375   jack_set_sample_rate_callback(cypi_jack.client, cypi_srate, 0);
      jack_on_shutdown(cypi_jack.client, cypi_shutdown, 0);
      /* create ports */
      cypi_jack.port[0] = jack_port_register(
        cypi_jack.client, "output_1", JACK_DEFAULT_AUDIO_TYPE, JackPortIsOutput, 0
380   );
```

```
    cypi_jack.port[1] = jack_port_register(
      cypi_jack.client, "output_2", JACK_DEFAULT_AUDIO_TYPE, JackPortIsOutput, 0
    );
    /* activate audio */
    if (jack_activate(cypi_jack.client)) {
      fprintf (stderr, "cannot activate JACK client");
      return 1;
    }
    /* must be activated before connecting JACK ports */
    const char **ports;
    if ((ports = jack_get_ports(
      cypi_jack.client, NULL, NULL, JackPortIsPhysical | JackPortIsInput
    ))) {
      /* connect up to two physical playback ports */
      int i = 0;
      while (ports[i] && i < 2) {
        if (jack_connect(
          cypi_jack.client, jack_port_name(cypi_jack.port[i]), ports[i]
        )) {
          fprintf(stderr, "cannot connect output port\n");
        }
        i++;
      }
      free(ports);
    }
  }
  /* activate video */
  glutMainLoop();
  return 0; /* never reached */
}

/* EOF */
```

26 cypi/Makefile

```
cypi: cypi.c
	gcc -std=c99 -Wall -pedantic -O3 -s -o cypi cypi.c -lGL -lGLU -lglut -↙
	    ↳ lGLEW `pkg-config --cflags --libs jack` -lm

# RENDERING/ENCODING NOTES
#
# mkfifo video.ppm
#
# cypi nrt &
# cat video.ppm |
# ppmtoy4m -S 444 -F 30:1 |
# y4mscaler -I sar=1/1 -O preset=ATSC_720p |
# mpeg2enc -f 12 -a 3 -l high -o cypi.m2v
#
# audacity
#   import raw 32bit float audio_?.raw 46080Hz sample rate
#   dc offset remover
#   make left/right
#   make stereo
#   chop length
#   fade out end
#   export 16bit wav
```

```
#
# twolame -b 224 cypi.wav
#
# mplex -f 3 -V -o cypi.mpeg cypi.m2v cypi.mp2
```

27 cypi/README

```
cypi -- audio-visual drone for OpenGL and JACK

make && ./cypi
```

28 drip/dline.h

```
#ifndef DLINE_H
#define DLINE_H 1

template <int N>
struct DLINE {
  double d[N];
  DLINE() { memset(d, 0, sizeof(d)); };
};

template <int N>
double dline(DLINE<N> &d, double x) {
  double y = 0.0001 * d.d[N-1];
  for (int i = N - 1; i > 0; --i) {
    d.d[i] = 0.9999 * d.d[i] + 0.0001 * d.d[i - 1];
  }
  d.d[0] = 0.9999 * d.d[0] + x;
  return y;
}

#endif
```

29 drip/drip.h

```
#ifndef DRIP_H
#define DRIP_H 1

#include <math.h>

struct DRIP {
  float mass;
  float position;
  float velocity;
  DRIP() : mass(0.375), position(2), velocity(0) { };
};

float drip(DRIP &d, float dmass, double &energy_out) {
  float dt = 1.0 / 4800.0; // 0.001;
//  float momentum = d.mass * d.velocity;
  float density = 1;
  float radius = 0.916;
  float area = 3.141592653589793 * radius * radius;
  float flow = dmass / density / area;
  float mass = d.mass + dmass;
```

```
  float velocity = d.velocity; // momentum / mass;
//  if (! isfinite(velocity)) { velocity = 0; }
  float gravity = 1 * mass;
  float spring = -fmax(0, 52.5 - 11.4 * mass) * d.position;
  float push = dmass * (flow - velocity);
  float resistance = -0.05 * velocity;
  float force = gravity + spring + push + resistance;
  float acceleration = force / mass;
  velocity += acceleration * dt;
  float position = d.position += velocity * dt;
  d.mass = mass;
  d.position = position;
  d.velocity = velocity;
  if (d.position > 5.5) {
    float m = 0.8 * d.mass - 0.3;
    float v = d.velocity;
    d.mass = d.mass - m;
    d.position = d.position - 5.5 + 2;
    d.velocity = 0;
    energy_out = 0.5 * m * v * v;
    return m;
  } else {
    energy_out = 0;
    return 0;
  }
}

#endif
```

30 drip/drip.jack-rack

(application/gzip; charset=binary)

31 drip/drips.h

```
#ifndef DRIPS_H
#define DRIPS_H 1

#include <math.h>

template <int N>
struct DRIPS {
  float mass[N];
  float position[N];
  float velocity[N];
  DRIPS() {
    for (int i = 0; i < N; ++i)
    {
      mass[i] = rand() / (float) RAND_MAX * 5;
      position[i] = rand() / (float) RAND_MAX * 8 - 2.5;
      velocity[i] = (rand() / (float) RAND_MAX * 2 - 1) * 10;
    }
  };
};

template <int N>
```

```
void drips(DRIPS<N> &d, float dmass, int L) {
  float dt = 1.0 / 4800.0;
  float density = 1;
  float radius = 0.916;
  float area = 3.141592653589793 * radius * radius;
  float flow = dmass / (density * area);
  #pragma omp parallel for
  for (int i = 0; i < N; ++i)
  {
    for (int j = 0; j < L; ++j)
    {
      float mass = d.mass[i] + dmass;
      float velocity = d.velocity[i];
      float gravity = 1 * mass;
      float spring = -fmax(0, 52.5 - 11.4 * mass) * d.position[i];
      float push = dmass * (flow - velocity);
      float resistance = -0.05 * velocity;
      float force = gravity + spring + push + resistance;
      float acceleration = force / mass;
      velocity += acceleration * dt;
      float position = d.position[i] + velocity * dt;
      d.mass[i] = mass;
      d.position[i] = position;
      d.velocity[i] = velocity;
      if (d.position[i] > 5.5) {
        float m = 0.8 * d.mass[i] - 0.3;
        d.mass[i] = d.mass[i] - m;
        d.position[i] = d.position[i] - 5.5 + 2;
        d.velocity[i] = 0;
      }
    }
  }
}

#endif
```

32 drip/event.h

```
#ifndef EVENT_H
#define EVENT_H 1

struct EVENT {
  unsigned long int timestamp;
  double mass;
  double energy;
  int voice;
  EVENT() : timestamp(0), mass(0), energy(0), voice(0) { };
};

struct EVENTS {
  EVENT e[8192];
  int head;
  EVENTS() : head(0) { };
};

void event(EVENTS &event, unsigned long int timestamp, double mass, double ↙
    ↳ energy, int voice) {
```

```
  int h = (event.head - 1 + 8192) % 8192;
  event.e[h].timestamp = timestamp;
  event.e[h].mass = mass;
  event.e[h].energy = energy;
  event.e[h].voice = voice;
  event.head = h;
}

#endif
```

33 drip/main.cc

```
#include <stdio.h>

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

#include <jack/jack.h>

#include <time.h>

#include "drip.h"
#include "pluck.h"
#include "dline.h"
#include "event.h"

#include "drips.h"

struct PLOP
{
  float ramp, rampi, freqi, amp, phase, ramp2, rampi2, env, osc;
  int rampn;
};

float plop(PLOP &p, float mass, float energy)
{
  if (mass > 0)
  {
    float t = 1000 * mass / 8;
    p.ramp = 0;
    p.rampi = 1 / (t / 7 / 1000 * 48000);
    p.rampn = 144; // 3ms
    p.freqi = (t / 5 + 400) / 48000;
    p.amp = t / 1000;
  }
  p.ramp += p.rampi;
  p.ramp = fminf(1, p.ramp);
  p.rampn -= 1;
  if (p.rampn == 0) {
    p.ramp2 = 0;
    p.rampi2 = p.rampi;
  }
  p.ramp2 += p.rampi2;
  p.ramp2 = fminf(1, p.ramp2);
  float amp = 1 - p.ramp;
  amp *= amp;
```

```
    amp *= amp;
    // lop 12
    amp *= 0.01;
    amp += 0.99 * p.env;
    p.env = amp;
    amp *= p.amp;
    amp *= sinf(2.0 * 3.141592653589793 * (p.phase += p.freqi));
    p.phase -= floorf(p.phase);
    // lop 1000
    amp *= 0.1;
    amp += 0.9 * p.osc;
    p.osc = amp;
    return amp;
  }

  struct AUDIO {
    DRIP tap;
    PLOP p6;
  #if 0
    PLUCK<960/6 + 1> p6l;
    PLUCK<960/6 - 1> p6r;
  //#if 0
    DLINE<1> d6;
    DRIP t6;
    PLUCK<960/5 - 1> p5l;
    PLUCK<960/5 + 1> p5r;
    DLINE<1> d5;
    DRIP t5;
    PLUCK<960/4 + 1> p4l;
    PLUCK<960/4 - 1> p4r;
    DLINE<1> d4;
    DRIP t4;
    PLUCK<960/3 - 1> p3l;
    PLUCK<960/3 + 1> p3r;
    DLINE<1> d3;
    DRIP t3;
    PLUCK<960/2 + 1> p2l;
    PLUCK<960/2 - 1> p2r;
    DLINE<1> d2;
    DRIP t2;
    PLUCK<960/1 - 1> p1l;
    PLUCK<960/1 + 1> p1r;
  #endif
  //  DLINE<480/1> d1;
  //  PLANT p
  };

  jack_client_t *client;
  jack_port_t *port[2];
  AUDIO a;
  volatile int dflow = 0;
  volatile unsigned long int count = 0;
  volatile double FLOW = 0;

  EVENTS e;

  double rms = 0;
```

```
double gain = 0;

105
static int processcb(jack_nframes_t nframes, void *arg) {
  (void) arg;
  jack_default_audio_sample_t *out[2];
  out[0] = (jack_default_audio_sample_t *)
110   jack_port_get_buffer(port[0], nframes);
  out[1] = (jack_default_audio_sample_t *)
    jack_port_get_buffer(port[1], nframes);
  double flow = FLOW;
  for (unsigned int k = 0; k < nframes; ++k) {
115   double t = count / (48000.0 * 60.0 * 60.0);
    if (dflow) {
      flow = t < 1 ? t : 0;
      count++;
    }
120   flow /= 4800;
    double mass = flow, energy = 0;
    double audiol = 0, audior = 0;
    double mass0 = mass;
    mass = 0;
125   for (int i = 0; i < 10; ++i)
    {
      double energy1;
       mass += drip(a.tap, mass0, energy1);
       energy += energy1;
130  }
    if (0 < mass) { event(e, count, mass, energy, 6); }
    float am = plop(a.p6, mass, energy);
    audiol += am;
    audior += am;
135 #if 0
    audiol += pluck(a.p6l, energy);
    audior += pluck(a.p6r, energy);
    mass = dline(a.d6, mass);
    mass = drip(a.t6, mass, energy);
140   if (0 < mass) { event(e, count, mass, energy, 5); }
    audiol += pluck(a.p5l, energy);
    audior += pluck(a.p5r, energy);
    mass = dline(a.d5, mass);
    mass = drip(a.t5, mass, energy);
145   if (0 < mass) { event(e, count, mass, energy, 4); }
    audiol += pluck(a.p4l, energy);
    audior += pluck(a.p4r, energy);
    mass = dline(a.d4, mass);
    mass = drip(a.t4, mass, energy);
150   if (0 < mass) { event(e, count, mass, energy, 3); }
    audiol += pluck(a.p3l, energy);
    audior += pluck(a.p3r, energy);
    mass = dline(a.d3, mass);
    mass = drip(a.t3, mass, energy);
155   if (0 < mass) { event(e, count, mass, energy, 2); }
    audiol += pluck(a.p2l, energy);
    audior += pluck(a.p2r, energy);
    mass = dline(a.d2, mass);
    mass = drip(a.t2, mass, energy);
```

```
      if (0 < mass) { event(e, count, mass, energy, 1); }
      audiol += pluck(a.p1l, energy);
      audior += pluck(a.p1r, energy);
#endif
      //mass = dline(a.d1, mass);
      //plant(a.p, mass);
      rms = rms * 0.9999 + 0.0001 * (audiol * audiol + audior * audior);
      gain = gain * 0.9999 + 0.0001 * 0.25 / sqrt(rms);
      if (! isfinite(gain)) { gain = 0; }
      gain = fmin(1, gain);
      out[0][k] = gain * audiol;
      out[1][k] = gain * audior;
    }
    if (dflow)
      FLOW = flow;
    return 0;
}

void keycb(GLFWwindow *window, unsigned int key) {
  switch (key) {
    case 'q': glfwSetWindowShouldClose(window, GL_TRUE); break;
    case ' ': dflow = 1 - dflow; break;
    case '\'': FLOW -= 1; break;
    case '[': FLOW -= 0.1; break;
    case '-': FLOW -= 0.01; break;
    case '=': FLOW += 0.01; break;
    case ']': FLOW += 0.1; break;
    case '#': FLOW += 1; break;
  }
  FLOW = fmax(0, FLOW);
  fprintf(stderr, "%.16f\t%.16f\r", FLOW, gain);
}

const char *drip_vert =
"#version 330 core\n"
"void main() {\n"
"  vec4 pos[4] = vec4[4](vec4(-1.0, -1.0, 0.0, 1.0)\n"
"                       ,vec4(-1.0,  1.0, 0.0, 1.0)\n"
"                       ,vec4( 1.0, -1.0, 0.0, 1.0)\n"
"                       ,vec4( 1.0,  1.0, 0.0, 1.0)\n"
"                       );\n"
"  gl_Position = pos[gl_VertexID];\n"
"}\n"
;

const char *drip_frag =
"#version 330 core\n"
"#extension GL_ARB_explicit_uniform_location : require\n"
"layout (location = 0) uniform float dmass;\n"
"layout (location = 1) uniform sampler2D prev;\n"
"layout (location = 0) out vec3 next;\n"
"void main()\n"
"{\n"
"  float dt = 1.0 / 4800.0;\n"
"  float density = 1;\n"
"  float radius = 0.916;\n"
"  float area = 3.141592653589793 * radius * radius;\n"
```

```
    "  float flow = dmass / (density * area);\n"
    "  vec3 d = texelFetch(prev, ivec2(gl_FragCoord.xy), 0).xyz;\n"
    "  for (int j = 0; j < 1024; ++j)\n"
220 "  {\n"
    "      float mass = d.x + dmass;\n"
    "      float velocity = d.z;\n"
    "      float gravity = 1.0 * mass;\n"
    "      float spring = -max(0, 52.5 - 11.4 * mass) * d.y;\n"
225 "      float push = dmass * (flow - velocity);\n"
    "      float resistance = -0.05 * velocity;\n"
    "      float force = gravity + spring + push + resistance;\n"
    "      float acceleration = force / mass;\n"
    "      velocity += acceleration * dt;\n"
230 "      float position = d.y + velocity * dt;\n"
    "      d.x = mass;\n"
    "      d.y = position;\n"
    "      d.z = velocity;\n"
    "      if (d.y > 5.5) {\n"
235 "        float m = 0.8 * d.x - 0.3;\n"
    "        d.x = d.x - m;\n"
    "        d.y = d.y - 5.5 + 2;\n"
    "        d.z = 0;\n"
    "      }\n"
240 "  }\n"
    "  next = d;\n"
    "}\n"
    ;

245 const char *drop_vert =
    "#version 330 core\n"
    "#extension GL_ARB_explicit_uniform_location : require\n"
    "layout (location = 0) uniform sampler2D prev;\n"
    "layout (location = 1) uniform sampler2D next;\n"
250 "out vec4 colour;\n"
    "void main()\n"
    "{\n"
    "  int i = (gl_VertexID / 2) % textureSize(prev, 0).x;\n"
    "  int j = (gl_VertexID / 2) / textureSize(prev, 0).x;\n"
255 "  bool even = (gl_VertexID & 1) == 0;\n"
    "  vec3 vprev = texelFetch(prev, ivec2(i, j), 0).xyz;\n"
    "  vec3 vnext = texelFetch(next, ivec2(i, j), 0).xyz;\n"
    "  vec3 vertex = even ? vprev : vnext;\n"
    "  gl_Position = vec4((vertex.x - 4.0) / 3.5, -vertex.z / 16.0, 0.0, 1.0);\n"
260 "  float h = (vertex.y + 2.0) / 8.0;\n"
    "  colour = vec4(h, even ? 0.0 : 1.0 - h, even ? 1.0 - h : h, vprev.x < vnext.x ↙
    ↳ ? 0.001 : 0.0);\n"
    "}\n"
    ;

265 const char *drop_frag =
    "#version 330 core\n"
    "in vec4 colour;\n"
    "layout (location = 0) out vec4 c;\n"
    "void main()\n"
270 "{\n"
    "  c = colour;\n"
    "}\n"
```

```
;

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 len = 0;
    glGetProgramiv(program, GL_INFO_LOG_LENGTH, &len);
    char *s = (char *) calloc(1, len + 1);
    if (! s) abort();
    glGetProgramInfoLog(program, len, 0, s);
    s[len] = 0;
    if (*s)
      fprintf(stderr, "%s: OpenGL shader program info log\n%s\n", name, s);
    free(s);
  }
  else
  {
    fprintf(stderr, "%s: OpenGL shader program creation failed\n", name);
  }
}

void debug_shader(GLuint shader, GLenum typ, const char *name)
{
  const char *tname = "unknown";
  if (typ == GL_VERTEX_SHADER) tname = "vertex";
  if (typ == GL_FRAGMENT_SHADER) tname = "fragment";
  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 len = 0;
    glGetShaderiv(shader, GL_INFO_LOG_LENGTH, &len);
    char *s = (char *) calloc(1, len + 1);
    if (! s) abort();
    glGetShaderInfoLog(shader, len, 0, s);
    s[len] = 0;
    if (*s) fprintf(stderr, "%s: OpenGL %s shader info log\n%s\n", name, tname, ↙
        ↳ s);
    free(s);
  }
  else
  {
    fprintf(stderr, "%s: OpenGL %s shader creation failed\n", name, tname);
  }
}

void compile_shader(GLuint program, GLenum typ, const char *name, const char *↙
    ↳ source)
{
  GLuint shader = glCreateShader(typ);
```

```
  glShaderSource(shader, 1, &source, 0);
  glCompileShader(shader);
  debug_shader(shader, typ, name);
  glAttachShader(program, shader);
330   glDeleteShader(shader);
}

GLuint compile_program(const char *name, const char *vert, const char *frag)
{
335   GLuint program = glCreateProgram();
  compile_shader(program, GL_VERTEX_SHADER, name, vert);
  compile_shader(program, GL_FRAGMENT_SHADER, name, frag);
  glLinkProgram(program);
  debug_program(program, name);
340   return program;
}


int main(int argc, char **argv) {
345   (void) argc;
  (void) argv;
  if (! glfwInit()) {
    return 1;
  }
350   glfwWindowHint(GLFW_RESIZABLE, GL_FALSE);
  GLFWwindow *window = glfwCreateWindow(1280, 720, "drip", 0, 0);
  if (! window) {
    return 1;
  }
355   glfwSetCharCallback(window, keycb);
  glfwMakeContextCurrent(window);
  glewInit();

  if (!(client = jack_client_open("drip", JackOptions(0), 0))) {
360     return 1;
  }
  jack_set_process_callback(client, processcb, 0);
  port[0] = jack_port_register(
    client, "output_1", JACK_DEFAULT_AUDIO_TYPE, JackPortIsOutput, 0
365   );
  port[1] = jack_port_register(
    client, "output_2", JACK_DEFAULT_AUDIO_TYPE, JackPortIsOutput, 0
  );
  if (jack_activate(client)) {
370     fprintf(stderr, "cannot activate JACK client");
    return 1;
  }
  const char **ports;
  if ((ports = jack_get_ports(
375     client, NULL, NULL, JackPortIsPhysical | JackPortIsInput
  ))) {
    int i = 0;
    while (ports[i] && i < 2) {
      if (jack_connect(
380       client, jack_port_name(port[i]), ports[i]
      )) {
        fprintf(stderr, "cannot connect output port\n");
```

```
        }
        i++;
385   }
      free(ports);
    }

    GLuint tex, fbo;
390   glGenTextures(1, &tex);
    glActiveTexture(GL_TEXTURE2);
    glBindTexture(GL_TEXTURE_2D, tex);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA32F, 1280, 720, 0, GL_RGBA, GL_FLOAT, 0)↙
        ↳ ;
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
395   glGenFramebuffers(1, &fbo);
    glBindFramebuffer(GL_FRAMEBUFFER, fbo);
    glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, ↙
        ↳ tex, 0);
    glClearColor(0, 0, 0, 1);
    glClear(GL_COLOR_BUFFER_BIT);
400

    GLuint tex2[2], fbo2[2];
    glGenTextures(2, &tex2[0]);
    glActiveTexture(GL_TEXTURE0);
405   glBindTexture(GL_TEXTURE_2D, tex2[0]);
#define TEX_SIZE 1024
{
    float init[TEX_SIZE*TEX_SIZE][3];
    for (int i = 0; i < TEX_SIZE * TEX_SIZE; ++i)
410   {
        init[i][0] = rand() / (float) RAND_MAX * 5;
        init[i][1] = rand() / (float) RAND_MAX * 8 - 2.5;
        init[i][2] = (rand() / (float) RAND_MAX * 2 - 1) * 10;
    }
415   glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB32F, TEX_SIZE, TEX_SIZE, 0, GL_RGB, ↙
        ↳ GL_FLOAT, init);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
    glActiveTexture(GL_TEXTURE1);
    glBindTexture(GL_TEXTURE_2D, tex2[1]);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB32F, TEX_SIZE, TEX_SIZE, 0, GL_RGB, ↙
        ↳ GL_FLOAT, init);
420   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
}
    glGenFramebuffers(2, &fbo2[0]);
    glBindFramebuffer(GL_FRAMEBUFFER, fbo2[0]);
    glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, ↙
        ↳ tex2[0], 0);
425   glBindFramebuffer(GL_FRAMEBUFFER, fbo2[1]);
    glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, ↙
        ↳ tex2[1], 0);
    int which = 0;
    glClearColor(0, 0, 0, 1);
    glClear(GL_COLOR_BUFFER_BIT);
430

    GLuint pdrip = compile_program("drip", drip_vert, drip_frag);
    GLuint pdrop = compile_program("drop", drop_vert, drop_frag);
```

```
    glDisable(GL_DEPTH_TEST);
435 glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE);

        float colours[6][3] =
          { { 1, 1, 0 }
440       , { 0, 1, 0 }
          , { 0, 1, 1 }
          , { 0, 0, 1 }
          , { 1, 0, 1 }
          , { 1, 0, 0 }
445       };


    dflow = 1;

450 unsigned long int speed = 48000;
    unsigned long int last5 = 0;
    unsigned long int frame = 0;

    double mass_mean = 3;
455 double mass_stddev = 10;
    double energy_mean = 5;
    double energy_stddev = 10;

//  DRIPS<1920 * 108> dv = DRIPS<1920 * 108>();
460
    do {
//    fprintf(stderr, "%ld\n", last5);
//    glClear(GL_COLOR_BUFFER_BIT);
      unsigned long int now = count;
465   int head = e.head;

      glViewport(0, 0, TEX_SIZE, TEX_SIZE);
      glDisable(GL_BLEND);
      glBindFramebuffer(GL_FRAMEBUFFER, fbo2[1-which]);
470   glUseProgram(pdrip);
      glUniform1f(0, FLOW);
      glUniform1i(1, which);
      glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);

475   glViewport(0, 0, 1280, 720);
      glEnable(GL_BLEND);
      glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
      glBindFramebuffer(GL_FRAMEBUFFER, fbo);
      glUseProgram(0);
480   glLoadIdentity();
      glOrtho(-1, 1, -1, 1, -1, 1);
      glBegin(GL_QUADS);
        glColor4f(0, 0, 0, 0.2);
        glVertex2f(-1, -1);
485     glVertex2f(-1,  1);
        glVertex2f( 1,  1);
        glVertex2f( 1, -1);
        glColor4f(1, 1, 1, 1);
      glEnd();
490
```

```
    glBlendFunc(GL_SRC_ALPHA, GL_ONE);
    glUseProgram(pdrop);
    glUniform1i(0, which);
    glUniform1i(1, 1 - which);
    glDrawArrays(GL_LINES, 0, TEX_SIZE * TEX_SIZE * 2);

    which = 1 - which;
#if 0

/*
    auto odv = dv;
    drips(dv, FLOW / 10, 200);
*/
    glLoadIdentity();
    glOrtho(-16, 16, 1, 6, -1, 1);
    glBegin(GL_LINES);
    for (int i = 0; i < 1920 * 108; ++i)
    {
      if (odv.mass[i] > dv.mass[i]) continue;
      float a = 0.005;
      float h = (odv.position[i] + 2) / 8;
      glColor4f(h, 0, 1 - h, a);
      glVertex2f(odv.velocity[i], odv.mass[i]);
      h = (dv.position[i] + 2) / 8;
      glColor4f(h, 1 - h, h, a);
      glVertex2f(dv.velocity[i], dv.mass[i]);
    }
    glEnd();
#endif

    glBindFramebuffer(GL_DRAW_FRAMEBUFFER, 0);
    glBlitFramebuffer(0, 0, 1280, 720, 0, 0, 1280, 720, GL_COLOR_BUFFER_BIT, ↙
        ↳ GL_NEAREST);

#if 0
//    glBlendFunc(GL_SRC_ALPHA, GL_ONE);
    glLoadIdentity();
    double aspect = 16.0 / 9.0;
    glOrtho(-aspect, aspect, -1, 1, -1, 1);
    glBegin(GL_QUADS);
    int setlast5 = 0;
    for (int i = 0; i < 8192; ++i) {
      int j = (head + i) % 8192;
      unsigned long int timestamp = e.e[j].timestamp;
      if (timestamp + 48000 * 60 < now) {
        break;
      }
      if (timestamp == 0) {
        break;
      }
      double mass = e.e[j].mass;
      double energy = e.e[j].energy;
      int voice = (e.e[j].voice + 5) % 6;
      if (voice == 5)
      {
        if (setlast5 == 1)
```

```
          {
            speed = last5 - timestamp;
            setlast5 = 2;
          }
          if (setlast5 == 0)
          {
            last5 = timestamp;
            setlast5 = 1;
          }
        }
        double angle = fmod((last5 - (double) timestamp) / (1.0 * speed), 1);
        angle = fmod(1 + angle, 1);
        double angle0 = angle - 0.005 * gain * sqrt(mass);
        double angle1 = angle + 0.005 * gain * sqrt(mass);
        double radius = pow(0.9, (now - timestamp) / (1.0 * speed));
        double dradius = sqrt(1 + 0.05 * gain * sqrt(energy));
        double radius0 = radius * dradius;
        double radius1 = radius;// / dradius;
        float alpha = (1024 / (65536 * (angle1 - angle0) * (radius0 - radius1)));
//fprintf(stderr, "%4d %d %e %e\t%e %e\t%e %e\t%e %ld\n", i, voice, mass, energy
    ↪ , angle0, angle1, radius0, radius1, alpha, speed);
        float x00 = radius0 * cos(2 * 3.141592653589793 * angle0);
        float y00 = radius0 * sin(2 * 3.141592653589793 * angle0);
        float x01 = radius0 * cos(2 * 3.141592653589793 * angle1);
        float y01 = radius0 * sin(2 * 3.141592653589793 * angle1);
        float x11 = radius1 * cos(2 * 3.141592653589793 * angle1);
        float y11 = radius1 * sin(2 * 3.141592653589793 * angle1);
        float x10 = radius1 * cos(2 * 3.141592653589793 * angle0);
        float y10 = radius1 * sin(2 * 3.141592653589793 * angle0);
        glColor4f(colours[voice][0], colours[voice][1], colours[voice][2], alpha);
#if 1
        glVertex2f(x00, y00);
        glVertex2f(x01, y01);
        glVertex2f(x11, y11);
        glVertex2f(x10, y10);
#else
        glVertex2f(angle0, radius0);
        glVertex2f(angle1, radius0);
        glVertex2f(angle1, radius1);
        glVertex2f(angle0, radius1);
#endif
    }
    glEnd();
#endif
#if 0
    glLoadIdentity();
    glOrtho(mass_mean - 3 * mass_stddev, mass_mean + 3 * mass_stddev,
        ↪ energy_mean - 3 * energy_stddev, energy_mean + 3 * energy_stddev, -1,
        ↪ 1);
    glBegin(GL_LINES);
    float previous[6][2];
    memset(previous, 0, sizeof(previous));
    double mass0 = 0, mass1 = 0, mass2 = 0;
    double energy0 = 0, energy1 = 0, energy2 = 0;
    for (int i = 0; i < 8192; ++i)
    {
      int j = (head + i) % 8192;
```

```
        unsigned long int timestamp = e.e[j].timestamp;
        if (timestamp + 48000 * 60 < now) {
          break;
        }
        if (timestamp == 0) {
          break;
        }
        double mass = e.e[j].mass;
        double energy = e.e[j].energy;
        int voice = (e.e[j].voice + 5) % 6;
        if (voice != 5) continue;
        double w = pow(1 - (now - timestamp) / (60.0 * 48000), 4);
        mass0 += w;
        mass1 += w * mass;
        mass2 += w * mass * mass;
        energy0 += w;
        energy1 += w * energy;
        energy2 += w * energy * energy;
        if (previous[voice][0])
        {
          glColor4f(colours[voice][0], colours[voice][1], colours[voice][2], w);
          glVertex2f(previous[voice][0], previous[voice][1]);
          glVertex2f(mass, energy);
        }
        previous[voice][0] = mass;
        previous[voice][1] = energy;
      }
      glEnd();
      if (mass0 > 1)
      {
        mass_mean *= 0.9;
        mass_mean += 0.1 * mass1 / mass0;
        mass_stddev *= 0.9;
        mass_stddev += 0.1 * sqrt(mass0 * mass2 - mass1 * mass1) / mass0;
      }
      if (energy0 > 1)
      {
        energy_mean *= 0.9;
        energy_mean += 0.1 * energy1 / energy0;
        energy_stddev *= 0.9;
        energy_stddev += 0.1 * sqrt(energy0 * energy2 - energy1 * energy1) / ↙
            ↳ energy0;
      }
#endif
      glfwSwapBuffers(window);
      glfwPollEvents();
    } while (! glfwWindowShouldClose(window));
    glfwTerminate();
    return 0;
}
```

34 drip/Makefile

```
drip: main.cc drip.h pluck.h dline.h event.h drips.h
        g++ -std=c++11 -Wall -Wextra -pedantic -O3 -march=native -fopenmp -o ↙
            ↳ drip main.cc -lGLEW -lGL -lGLU -lglfw -ljack
```

35 drip/pluck.h

```
#ifndef PLUCK_H
#define PLUCK_H 1

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

template <int P>
struct PLUCK {
  int w;
  int i;
  float z[2][P];
  PLUCK() : w(0), i(0) {
    memset(&z[0][0], 0, 2 * P * sizeof(z[0][0]));
  };
};

template <int P>
float pluck(PLUCK<P> &p, float amp) {
  if (amp > 0) {
    for (int j = 0; j < P; ++j) {
      p.z[p.w][j] = (480.0 / P) * 1e-2 * sqrt(amp) * (rand() / (double) RAND_MAX
          ↪ - 0.5);
    }
    p.i = 0;
  }
  if (p.i >= P) {
    for (int j = 0; j < P; ++j) {
      p.z[1 - p.w][j] = pow(0.999, 480.0 / P) / 3.0 * (p.z[p.w][j] + p.z[p.w][(j
          ↪ + 1)% P] + p.z[p.w][(j + P - 1)% P]);
    }
    p.i = 0;
    p.w = 1 - p.w;
  }
  return p.z[p.w][p.i++];
}

#endif
```

36 drip/README

```
a tap

drips

landing near the inside edge
of a metal bowl suspended in mid air
which goes "ting" when a drip lands

the drips flow down into the bowl
spreading out as they go
from sharp spikes of mass (impulse train)
becoming overlapping humps (near-Gaussian)
and eventually flat
```

```
the bowl has a small hole in the middle

when enough liquid has accumulated
from the flow of the drips into the bowl
to overcome surface tension
the liquid drips through the hole

underneath the hole in the bowl
is another bowl with a hole
which goes "ting" when a drip lands
near its inside edge

this goes on for some time

each bowl larger than the last
with a deeper "ting"

the final bowl drips into a plant pot

it grows and flowers
```

37 .gitignore

```
cypi/cypi
stringthing/stringthing
drip/drip
*~
*.wav
*.mp4
```

38 harmonic-protocol/COPYING.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/>.
```

39 harmonic-protocol/.gitignore

```
harmonic-protocol
harmonic-protocol.html
harmonic-protocol.js
harmonic-protocol.wasm
*.gz
```

40 harmonic-protocol/harmonic-protocol.c

```
/*
Harmonic Protocol (AGPL3+) 2019 Claude Heiland-Allen <claude@mathr.co.uk>
License: https://www.gnu.org/licenses/agpl-3.0.en.html
*/

#include <SDL2/SDL.h>
#include <SDL2/SDL_audio.h>

#ifdef __EMSCRIPTEN__
#include <emscripten.h>
#else
#include <unistd.h>
#endif

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <tgmath.h>
#include <time.h>

typedef float sample;
static float SR = 48000;
static int TET = 12;
static int DELTA = 7;
static int NOTES = (72 - 24) / 12 * 12;
#define MAXNOTES 512
#define pi 3.141592653589793
#define twopi (2 * pi)

static inline sample clamp(sample x, sample lo, sample hi) {
  return fmin(fmax(x, lo), hi);
}

static inline sample mix(sample x, sample y, sample t) {
  return (1 - t) * x + t * y;
}
```

```
static inline sample noise() {
  return 2 * (rand() / (sample) RAND_MAX - (sample)0.5);
}

static inline sample mtof(sample f) {
  return (sample)8.17579891564 * exp((sample)0.0577622650 * fmin(f, 1499));
}

#define log10overten 0.23025850929940458
#define tenoverlog10 4.3429448190325175

static inline sample dbtorms(sample f) {
  return exp((sample)0.5 * (sample)log10overten * (fmin(f, 870) - 100));
}

static inline sample rmstodb(sample f) {
  return 100 + 2 * (sample)tenoverlog10 * log(f);
}

// based on pd's [lop~] [hip~]

typedef struct { double y; } LOP;

static sample lop(LOP *s, sample x, sample hz) {
  double c = clamp(twopi * hz / SR, 0, 1);
  return s->y = mix(x, s->y, 1 - c);
}

typedef struct { double y; } HIP;

static sample hip(HIP *s, sample x, sample hz) {
  double c = clamp(1 - twopi * hz / SR, 0, 1);
  double n = (1 + c) / 2;
  double y = x + c * s->y;
  double o = n * (y - s->y);
  s->y = y;
  return o;
}

// one-pole one-zero low pass filter designed in the Z plane

typedef struct {
  double a;
  double b;
  sample x;
  double y;
} LOP1;

static LOP1 *lop1_(LOP1 *s, sample hz)
{
  const double w = twopi * clamp(fabs(hz / SR), 0, (sample)0.5);
  s->a = (1 - sin(w)) / cos(w);
  s->b = (1 - s->a) / 2;
  return s;
}
```

```
static inline sample lop1(LOP1 *s, sample x)
{
  const double y = s->b * (x + s->x) + s->a * s->y;
  s->x = x;
  s->y = y;
  return y;
}

typedef struct { double b0, b1, b2, a1, a2, y1, y2; sample x1, x2; } BIQUAD;

static inline sample biquad(BIQUAD *bq, sample x0) {
  double b0 = bq->b0, b1 = bq->b1, b2 = bq->b2, a1 = bq->a1, a2 = bq->a2;
  double x1 = bq->x1, x2 = bq->x2, y1 = bq->y1, y2 = bq->y2;
  double y0 = b0 * x0 + b1 * x1 + b2 * x2 - a1 * y1 - a2 * y2;
  bq->y2 = y1;
  bq->y1 = y0;
  bq->x2 = x1;
  bq->x1 = x0;
  return y0;
}

static BIQUAD *bandpass(BIQUAD *bq, sample hz, sample q) {
  double w0 = hz * twopi / SR;
  double a = fabs(sin(w0) / (2 * q));
  double c = cos(w0);
  double b0 = a, b1 = 0, b2 = -a;
  double a0 = 1 + a, a1 = -2 * c, a2 = 1 - a;
  bq->b0 = b0 / a0;
  bq->b1 = b1 / a0;
  bq->b2 = b2 / a0;
  bq->a1 = a1 / a0;
  bq->a2 = a2 / a0;
  return bq;
}

typedef struct { HIP hip[2]; LOP lop[3]; } COMPRESS;

static inline void compress(sample out[2], COMPRESS *s, sample hiphz, sample ↙
    ↳ lophz1, sample lophz2, sample db, const sample in[2]) {
  sample h[2] =
    { hip(&s->hip[0], in[0], hiphz)
    , hip(&s->hip[1], in[1], hiphz)
    };
  h[0] *= h[0];
  h[1] *= h[1];
  h[0] = lop(&s->lop[0], h[0], lophz1);
  h[1] = lop(&s->lop[1], h[1], lophz1);
  sample env = lop(&s->lop[2], sqrtf(fmax(0, h[0] + h[1])), lophz2);
  sample env0 = env;
  env = rmstodb(env);
  if (env > db) {
    env = db + (env - db) / 4;
  } else {
    env = db;
  }
  env = (sample)0.25 * dbtorms(env) / dbtorms((100 - db) / 4 + db);
  sample gain = env / env0;
```

```
    if (isnan(gain) || isinf(gain)) { gain = 0; }
    gain = clamp(gain, 0, (sample)1.0e6);
    out[0] = tanhf(in[0] * gain);
    out[1] = tanhf(in[1] * gain);
  }

  // https://en.wikipedia.org/wiki/Cubic_Hermite_spline#↙
     ↳ Interpolation_on_the_unit_interval_without_exact_derivatives

  typedef struct { int length, woffset; } DELAY;

  static void delwrite(DELAY *del, sample x0) {
    float *buffer = (float *) (del + 1);
    int l = del->length;
    l = (l > 0) ? l : 1;
    int w = del->woffset;
    buffer[w++] = x0;
    if (w >= l) { w -= l; }
    del->woffset = w;
  }

  static sample delread4(DELAY *del, sample ms) {
    float *buffer = (float *) (del + 1);
    int l = del->length;
    l = (l > 0) ? l : 1;
    int w = del->woffset;
    sample d = ms / (sample) 1000 * SR;
    int d1 = floor(d);
    int d0 = d1 - 1;
    int d2 = d1 + 1;
    int d3 = d1 + 2;
    sample t = d - d1;
    d0 = (0 < d0 && d0 < l) ? d0 : 0;
    d1 = (0 < d1 && d1 < l) ? d1 : d0;
    d2 = (0 < d2 && d2 < l) ? d2 : d1;
    d3 = (0 < d3 && d3 < l) ? d3 : d2;
    int r0 = w - d0;
    int r1 = w - d1;
    int r2 = w - d2;
    int r3 = w - d3;
    r0 = r0 < 0 ? r0 + l : r0;
    r1 = r1 < 0 ? r1 + l : r1;
    r2 = r2 < 0 ? r2 + l : r2;
    r3 = r3 < 0 ? r3 + l : r3;
    sample y0 = buffer[r0];
    sample y1 = buffer[r1];
    sample y2 = buffer[r2];
    sample y3 = buffer[r3];
    sample a0 = -t*t*t + 2*t*t - t;
    sample a1 = 3*t*t*t - 5*t*t + 2;
    sample a2 = -3*t*t*t + 4*t*t + t;
    sample a3 = t*t*t - t*t;
    return (a0 * y0 + a1 * y1 + a2 * y2 + a3 * y3) / 2;
  }


  typedef struct {
```

```
  int reloaded;
  BIQUAD hp_band[MAXNOTES][2];
  LOP1 hp_rms[MAXNOTES][2];
  DELAY hp_del1; float hp_del1buf[192000];
  DELAY hp_del2; float hp_del2buf[192000];
  sample hp_filtered[MAXNOTES][2];
  COMPRESS hp_compress;
} S;

static S state;

static int go(S *s, float *out) {
  sample hz1 = 1;
  sample hz2 = hz1 / 32;

  if (s->reloaded) {
    s->hp_del1.length = 192000;
    s->hp_del2.length = 192000;
    NOTES = (72 - 24) / 12 * TET;
    double f1 = 1.0;
    double f2 = pow(2, 1.0 / TET);
    float Q = sqrt(f2 * f1) / (f2 - f1);
    for (int note = 0; note < NOTES; ++note)
    {
      for (int c = 0; c < 2; ++c)
      {
        bandpass(&s->hp_band[note][c], mtof(24 + note * 12.0 / TET), Q);
        lop1_(&s->hp_rms[note][c], hz2);
      }
    }
    s->reloaded = 0;
  }
  {
    sample band[TET][2];
    memset(band, 0, sizeof(band));
    sample input[2] =
      { delread4(&s->hp_del1, 1000./hz1) + 1e-8 * noise()
      , delread4(&s->hp_del2, 1000./hz1) + 1e-8 * noise()
      };
    for (int note = 0; note < NOTES; ++note)
    {
      for (int c = 0; c < 2; ++c)
      {
        sample b = s->hp_filtered[note][c] = biquad(&s->hp_band[note][c], input[↙
            ↳ c]);
        band[note % TET][c] += lop1(&s->hp_rms[note][c], b * b);
      }
    }
    sample ma = 0;
    for (int i = 0; i < TET; ++i)
    {
      for (int c = 0; c < 2; ++c)
      {
        band[i][c] = sqrtf(band[i][c]);
        ma = fmax(band[i][c], ma);
      }
    }
```

```
    ma /= 8;
    for (int i = 0; i < TET; ++i)
    {
      for (int c = 0; c < 2; ++c)
      {
        band[i][c] /= ma;
      }
    }
    sample output[2] = { 0, 0 };
    for (int note = 0; note < NOTES; ++note)
    {
      for (int c = 0; c < 2; ++c)
      {
        output[c] += tanhf(band[(note + DELTA) % TET][c] * s->hp_filtered[note][c]);
      }
    }
    sample o[2] = { output[0], output[1] };
    sample a = twopi * 1. / 12;
    sample co = cos(a);
    sample si = sin(a);
    output[0] = (co * o[0] - si * o[1]);
    output[1] = (si * o[0] + co * o[1]);
    compress(output, &s->hp_compress, 5, 10, 25, 48, output);
    delwrite(&s->hp_del1, output[0]);
    delwrite(&s->hp_del2, output[1]);
    out[0] = output[0];
    out[1] = output[1];
  }
  return 0;
}

static void audio(void *userdata, Uint8 *stream, int len)
{
  (void) userdata;
  float *b = (float *) stream;
  int m = len / sizeof(float) / 2;
  int k = 0;
  for (int i = 0; i < m; ++i)
  {
    float out[2];
    go(&state, out);
    b[k++] = out[0];
    b[k++] = out[1];
  }
}

#ifdef __EMSCRIPTEN__
static void main1(void)
{
  // nop
}
#endif

int main(int argc, char **argv)
{
  if (argc > 1)
```

```
  {
    TET = atoi(argv[1]);
    if (! (1 <= TET && TET <= 128))
    {
      printf("error: %d-EDO out of range\n", TET);
      return 1;
    }
  }
  if (argc > 2)
  {
    DELTA = atoi(argv[2]);
    DELTA %= TET;
    DELTA += TET;
    DELTA %= TET;
    if (DELTA == 0)
    {
      printf("error: /%d out of range\n", DELTA);
      return 1;
    }
  }
  else
  {
    DELTA = round(log2(1.5) * TET);
  }
  srand(time(0));
  memset(&state, 0, sizeof(state));
  state.reloaded = 1;
  // initialize SDL2 audio
  SDL_Init(SDL_INIT_AUDIO);
  SDL_AudioSpec want, have;
  want.freq = SR;
  want.format = AUDIO_F32;
  want.channels = 2;
  want.samples = 4096;
  want.callback = audio;
  SDL_AudioDeviceID dev = SDL_OpenAudioDevice(NULL, 0, &want, &have, ↙
      ↳ SDL_AUDIO_ALLOW_ANY_CHANGE);
  if (have.freq > 192000 || have.format != AUDIO_F32 || have.channels != 2)
  {
    printf("want: %d %d %d %d\n", want.freq, want.format, want.channels, want.↙
        ↳ samples);
    printf("have: %d %d %d %d\n", have.freq, have.format, have.channels, have.↙
        ↳ samples);
    printf("error: bad audio parameters\n");
  }
  else
  {
    printf("Harmonic Protocol (AGPL3+) 2019 Claude Heiland-Allen\n");
    printf("%d-EDO/%d\n", TET, DELTA);
    SR = have.freq;
    // start audio processing
    SDL_PauseAudioDevice(dev, 0);
#ifdef __EMSCRIPTEN__
    emscripten_set_main_loop(main1, 60, 1);
#else
    while (1) sleep(1);
#endif
```

```
  }
  return 0;
}
```

41 harmonic-protocol/index.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta content="text/html; charset=utf-8" http-equiv="Content-Type">
    <meta name="description" content="Feedback amplifying tones according to the level 7 semitones away." />
    <title>Harmonic Protocol</title>
    <style>
      body { width: 80ex; margin: auto; }
      #spinner{height:30px;width:30px;margin:0;margin-top:20px;margin-left:20px;display:inline-block;vertical-align:top;-webkit-animation:rotation .8s linear infinite;-moz-animation:rotation .8s linear infinite;-o-animation:rotation .8s linear infinite;animation:rotation .8s linear infinite;border-left:5px solid #ebebeb;border-right:5px solid #ebebeb;border-bottom:5px solid #ebebeb;border-top:5px solid #787878;border-radius:100%;background-color:#bdd72e}
      @-webkit-keyframes rotation{from{-webkit-transform:rotate(0)}to{-webkit-transform:rotate(360deg)}}
      @-moz-keyframes rotation{from{-moz-transform:rotate(0)}to{-moz-transform:rotate(360deg)}}
      @-o-keyframes rotation{from{-o-transform:rotate(0)}to{-o-transform:rotate(360deg)}}
      @keyframes rotation{from{transform:rotate(0)}to{transform:rotate(360deg)}}
      #progress{height:20px;width:300px}
    </style>
  </head>
  <body>
    <h1>Harmonic Protocol</h1>
    <h2>Listen</h2>
    <p>(web audio powered by <a href="https://emscripten.org">emscripten</a>)</p>
    <p>
      <progress hidden="hidden" id="progress" max="100" value="0"></progress>
      <div id="spinner"></div>
    </p>
    <p><input type="button" id="btn-audio" value="&#x1F507; Pause" onclick="toggleAudio()" /></p>
    <p id="status">Downloading...</p>
    <textarea id="output" rows="3" cols="80"></textarea>
    <p id="audiobutton">
    <script>window.onhashchange = function() { window.location.reload(); };</script>
    <script>
        var audioBtn = document.getElementById('btn-audio');

        // An array of all contexts to resume on the page
        const audioContexList = [];
        (function() {
            // A proxy object to intercept AudioContexts and
            // add them to the array for tracking and resuming later
```

```
            self.AudioContext = new Proxy(self.AudioContext, {
                construct(target, args) {
                    const result = new target(...args);
                    audioContexList.push(result);
                    if (result.state == "suspended") audioBtn.value = "&#x1F508;↙
                        ↳ Play";
                    return result;
                }
            });
        })();

        function toggleAudio() {
            var resumed = false;
            audioContexList.forEach(ctx => {
                if (ctx.state == "suspended") { ctx.resume(); resumed = true; }
                else if (ctx.state == "running") ctx.suspend();
            });

            if (resumed) audioBtn.value = "&#x1F507; Pause";
            else audioBtn.value = "&#x1F508; Play";
        }
    </script>
    <script>
      var arguments = window.location.hash.substr(1).split('/');
      var statusElement = document.getElementById("status"),
          progressElement = document.getElementById("progress"),
          spinnerElement = document.getElementById("spinner"),
          Module = {
            arguments:arguments,
            preRun:[],
            postRun:[],
            print:function(){var t=document.getElementById("output");return t&&(↙
                ↳ t.value="",function(e){1<arguments.length&&(e=Array.prototype↙
                ↳ .slice.call(arguments).join(" ")),console.log(e),t&&(t.value+=↙
                ↳ e+"\n",t.scrollTop=t.scrollHeight)}}(),
            printErr:function(e){1<arguments.length&&(e=Array.prototype.slice.↙
                ↳ call(arguments).join(" ")),console.error(e)},
            setStatus:function(e){if(Module.setStatus.last||(Module.setStatus.↙
                ↳ last={time:Date.now(),text:""}),e!==Module.setStatus.last.text↙
                ↳ ){var t=e.match(/([^(]+)\((\d+(\.\d+)?)\/(\d+)\)/),n=Date.now↙
                ↳ ();t&&n-Module.setStatus.last.time<30||(Module.setStatus.last.↙
                ↳ time=n,Module.setStatus.last.text=e,t?(e=t[1],progressElement.↙
                ↳ value=100*parseInt(t[2]),progressElement.max=100*parseInt(t↙
                ↳ [4]),progressElement.hidden=!1,spinnerElement.hidden=!1):(↙
                ↳ progressElement.value=null,progressElement.max=null,↙
                ↳ progressElement.hidden=!0,e||(spinnerElement.style.display="↙
                ↳ none")),statusElement.innerHTML=e)}},
            totalDependencies:0,
            monitorRunDependencies:function(e){this.totalDependencies=Math.max(↙
                ↳ this.totalDependencies,e),Module.setStatus(e?"Preparing... ↙
                ↳ ("+(this.totalDependencies-e)+"/"+this.totalDependencies+")":"↙
                ↳ All downloads complete.")}
          };
      Module.setStatus("Downloading..."),window.onerror=function(e){Module.↙
          ↳ setStatus("Exception thrown, see JavaScript console"),spinnerElement↙
          ↳ .style.display="none",Module.setStatus=function(e){e&&Module.↙
          ↳ printErr("[post-exception status] "+e)}}
```

```
    </script>
    <script async src=harmonic-protocol.js></script>
    <h2>Process</h2>
    <p>Stereo feedback loop 1 second long, with left and right blended
80  with a rotation matrix (angle 2PI/12).</p>
    <p>Inside the loop, compute RMS level per step per channel via
    a bank of biquad bandpass filters over 4 octaves upwards from
    MIDI note 24, accumulated modulo #-EDO. Low pass filter for the RMS at
    1/32 Hz.</p>
85  <p>Scale each individual step by the energy of the octave
    accumulation /# steps away (pick a direction). Distort each band
    using tanh(). Apply strong dynamic range compression to the stereo
    mix to normalize peak levels to ~1.</p>
    <h2>Source</h2>
90  <p><a href="harmonic-protocol.c" title="Harmonic Protocol source code">↙
        ↳ harmonic-protocol.c</a></p>
    <p><a href="Makefile" title="Harmonic Protocol build system">Makefile</a></p↙
        ↳ >
    <h2>License</h2>
    <p><a href="https://www.gnu.org/licenses/agpl-3.0.en.html" title="GNU Affero↙
        ↳  General Public License version 3 or greater">AGPL3+</a></p>
    <h2>Author</h2>
95  <p><a href="https://mathr.co.uk" title="mathr.co.uk">mathr.co.uk</a></p>
  </body>
</html>
```

42 harmonic-protocol/Makefile

```
harmonic-protocol: harmonic-protocol.c
	gcc -std=c99 -Wall -Wextra -pedantic -O3 -march=native -ffast-math -o ↙
	    ↳ harmonic-protocol harmonic-protocol.c `sdl2-config --cflags --↙
	    ↳ libs` -lm

harmonic-protocol.html: harmonic-protocol.c
5	emcc -O3 -Os -o harmonic-protocol.html harmonic-protocol.c --closure 1 -↙
	    ↳ s USE_SDL=2
	gzip -9 -k -f harmonic-protocol.js
	gzip -9 -k -f harmonic-protocol.wasm
```

43 interstellar-interference/.gitignore

```
*.pd_linux
```

44 interstellar-interference/interstellar-interference2.pd

```
#N canvas 0 92 661 622 10;
#X obj 201 450 hip~ 5;
#X obj 201 477 expr~ tanh($v1);
#X obj 227 518 dac~;
5 #X obj 64 201 world_light;
#X obj 215 91 nbody 6;
#X msg 150 156 positions;
#X obj 202 5 bang~;
#X msg 202 54 distances;
10 #X msg 59 16 create;
#X msg 59 43 destroy;
```

```
#X obj 37 17 tgl 15 0 empty empty empty 17 7 0 10 -262144 -1 -1 1 1
;
#X obj 37 76 gemwin 50;
#X msg 119 44 lighting 1;
#X obj 120 20 loadbang;
#X obj 36 100 gemhead;
#X obj 281 111 nb-link~ \$0 1 2;
#X obj 231 416 *~ 0;
#X msg 278 416 1;
#X msg 179 563 stop;
#X obj 320 449 hip~ 5;
#X obj 320 476 expr~ tanh($v1);
#X obj 321 416 *~ 0;
#X msg 274 440 0;
#X obj 143 594 writesf~ 2;
#X obj 281 231 nb-link~ \$0 2 4;
#X obj 281 211 nb-link~ \$0 2 3;
#X obj 281 131 nb-link~ \$0 1 3;
#X obj 281 151 nb-link~ \$0 1 4;
#X obj 281 171 nb-link~ \$0 1 5;
#X obj 281 191 nb-link~ \$0 1 6;
#X obj 281 251 nb-link~ \$0 2 5;
#X obj 281 271 nb-link~ \$0 2 6;
#X obj 281 291 nb-link~ \$0 3 4;
#X obj 281 311 nb-link~ \$0 3 5;
#X obj 281 331 nb-link~ \$0 3 6;
#X obj 281 351 nb-link~ \$0 4 5;
#X obj 281 371 nb-link~ \$0 4 6;
#X obj 281 391 nb-link~ \$0 5 6;
#X msg 278 56 step 5;
#X msg 337 86 g \$1;
#X obj 337 63 f 10;
#X floatatom 470 236 5 0 0 0 - - -;
#X obj 202 28 t b b b;
#X obj 413 183 timer;
#X floatatom 414 236 5 0 0 0 - - -;
#X obj 413 208 / 1000;
#X msg 452 116 bang;
#X obj 472 143 metro 1000;
#X obj 473 186 f;
#X msg 389 17 16;
#X obj 57 180 t a a a b;
#X obj 150 201 t b a a;
#X obj 106 270 translateXYZ 0 0 -16;
#X obj 149 180 separator;
#X obj 395 66 * 1;
#X msg 399 39 1;
#X msg 433 42 0.9999;
#X msg 503 41 1.0001;
#X obj 450 -1 bng 15 250 50 0 empty empty empty 17 7 0 10 -262144 -1
-1;
#X obj 146 518 spigot;
#X obj 198 523 tgl 15 0 empty empty empty 17 7 0 10 -262144 -1 -1 1
1;
#X obj 467 72 moses 16;
#X obj 468 93 moses 0.25;
#X msg 353 18 0.25;
```

```
#X obj 450 20 t b b b b b;
#X msg 534 7 0 \, 1;
#X msg 272 14 randomize \, positions;
#X obj 35 130 t a a;
#X obj 89 480 spigot;
#X obj 140 481 tgl 15 0 empty empty empty 17 7 0 10 -262144 -1 -1 1
1;
#X obj 37 503 spigot;
#X obj 35 529 pix_write;
#X msg 169 452 1;
#X msg 109 77 dimen 1280 720;
#X obj 105 225 pix_snap2tex 0 0 1280 720;
#X obj 106 329 rectangle 16 9;
#X obj 421 328 GEMglEnable GL_BLEND;
#X obj 421 307 GEMglBlendFunc GL_SRC_ALPHA GL_ONE;
#X obj 420 349 GEMglDisable GL_DEPTH_TEST;
#X obj 420 264 GEMglBlendFunc GL_SRC_ALPHA GL_ONE_MINUS_SRC_ALPHA;
#X obj 421 283 GEMglEnable GL_DEPTH_TEST;
#X obj 56 443 nb-body~ \$0 6 6;
#X obj 56 403 nb-body~ \$0 6 4;
#X obj 56 343 nb-body~ \$0 6 1;
#X obj 56 423 nb-body~ \$0 6 5;
#X obj 56 363 nb-body~ \$0 6 2;
#X obj 56 383 nb-body~ \$0 6 3;
#X obj 107 290 rotateXYZ 0 0 0.2;
#X obj 105 248 colorRGB 0.99 0.99 0.99 1;
#X obj 57 157 scale 0.25;
#X obj 107 311 scale 3.6;
#X msg 337 573 auto 1 \, file o/i 0;
#X msg 104 542 open o/a.wav \, start;
#X msg 105 503 0;
#X connect 0 0 1 0;
#X connect 1 0 2 0;
#X connect 1 0 23 0;
#X connect 4 0 83 2;
#X connect 4 1 15 2;
#X connect 5 0 4 0;
#X connect 6 0 42 0;
#X connect 7 0 4 0;
#X connect 8 0 11 0;
#X connect 9 0 11 0;
#X connect 10 0 11 0;
#X connect 12 0 11 0;
#X connect 13 0 12 0;
#X connect 13 0 73 0;
#X connect 14 0 67 0;
#X connect 15 0 26 0;
#X connect 15 1 26 1;
#X connect 15 2 26 2;
#X connect 16 0 0 0;
#X connect 17 0 16 1;
#X connect 17 0 21 1;
#X connect 18 0 23 0;
#X connect 19 0 20 0;
#X connect 20 0 2 1;
#X connect 20 0 23 1;
#X connect 21 0 19 0;
```

```
#X connect 22 0 16 1;
#X connect 22 0 21 1;
#X connect 24 0 30 0;
#X connect 24 1 30 1;
#X connect 24 2 30 2;
#X connect 25 0 24 0;
#X connect 25 1 24 1;
#X connect 25 2 24 2;
#X connect 26 0 27 0;
#X connect 26 1 27 1;
#X connect 26 2 27 2;
#X connect 27 0 28 0;
#X connect 27 1 28 1;
#X connect 27 2 28 2;
#X connect 28 0 29 0;
#X connect 28 1 29 1;
#X connect 28 2 29 2;
#X connect 29 0 25 0;
#X connect 29 1 25 1;
#X connect 29 2 25 2;
#X connect 30 0 31 0;
#X connect 30 1 31 1;
#X connect 30 2 31 2;
#X connect 31 0 32 0;
#X connect 31 1 32 1;
#X connect 31 2 32 2;
#X connect 32 0 33 0;
#X connect 32 1 33 1;
#X connect 32 2 33 2;
#X connect 33 0 34 0;
#X connect 33 1 34 1;
#X connect 33 2 34 2;
#X connect 34 0 35 0;
#X connect 34 1 35 1;
#X connect 34 2 35 2;
#X connect 35 0 36 0;
#X connect 35 1 36 1;
#X connect 35 2 36 2;
#X connect 36 0 37 0;
#X connect 36 1 37 1;
#X connect 36 2 37 2;
#X connect 37 0 16 0;
#X connect 37 1 21 0;
#X connect 38 0 4 0;
#X connect 39 0 4 0;
#X connect 40 0 39 0;
#X connect 40 0 48 1;
#X connect 40 0 54 0;
#X connect 42 0 7 0;
#X connect 42 1 38 0;
#X connect 42 2 40 0;
#X connect 43 0 45 0;
#X connect 45 0 44 0;
#X connect 46 0 43 0;
#X connect 46 0 47 0;
#X connect 47 0 43 1;
#X connect 47 0 48 0;
```

```
#X connect 48 0 41 0;
#X connect 49 0 40 1;
#X connect 50 0 77 0;
#X connect 50 1 3 0;
#X connect 50 2 53 0;
#X connect 50 3 5 0;
#X connect 51 0 74 0;
#X connect 51 1 74 0;
#X connect 51 2 79 0;
#X connect 52 0 87 0;
#X connect 53 0 51 0;
#X connect 54 0 40 1;
#X connect 54 0 61 0;
#X connect 55 0 54 1;
#X connect 56 0 54 1;
#X connect 57 0 54 1;
#X connect 58 0 64 0;
#X connect 59 0 92 0;
#X connect 60 0 59 1;
#X connect 61 0 62 0;
#X connect 61 1 56 0;
#X connect 63 0 40 1;
#X connect 64 0 46 0;
#X connect 64 0 59 0;
#X connect 64 0 72 0;
#X connect 64 0 17 0;
#X connect 64 1 57 0;
#X connect 64 2 63 0;
#X connect 64 3 66 0;
#X connect 64 4 65 0;
#X connect 65 0 11 0;
#X connect 66 0 4 0;
#X connect 67 0 70 0;
#X connect 67 1 89 0;
#X connect 68 0 70 1;
#X connect 69 0 68 1;
#X connect 70 0 71 0;
#X connect 72 0 68 0;
#X connect 73 0 11 0;
#X connect 74 0 88 0;
#X connect 76 0 78 0;
#X connect 77 0 76 0;
#X connect 78 0 83 0;
#X connect 79 0 80 0;
#X connect 80 0 83 1;
#X connect 82 0 84 0;
#X connect 82 1 84 1;
#X connect 82 2 84 2;
#X connect 83 0 85 0;
#X connect 83 1 85 1;
#X connect 83 2 85 2;
#X connect 84 0 81 0;
#X connect 84 1 81 1;
#X connect 84 2 81 2;
#X connect 85 0 86 0;
#X connect 85 1 86 1;
#X connect 85 2 86 2;
```

```
#X connect 86 0 82 0;
#X connect 86 1 82 1;
#X connect 86 2 82 2;
#X connect 87 0 90 0;
#X connect 88 0 52 0;
#X connect 89 0 50 0;
#X connect 90 0 75 0;
#X connect 91 0 71 0;
#X connect 92 0 23 0;
#X connect 93 0 70 1;
#X connect 93 0 18 0;
```

45 interstellar-interference/interstellar-interference.pd

```
#N canvas -87 92 661 622 10;
#X obj 201 450 hip~ 5;
#X obj 201 477 expr~ tanh($v1);
#X obj 227 518 dac~;
#X obj 64 201 world_light;
#X obj 215 91 nbody 6;
#X msg 150 156 positions;
#X obj 202 5 bang~;
#X msg 202 54 distances;
#X msg 59 16 create;
#X msg 59 43 destroy;
#X obj 37 17 tgl 15 0 empty empty empty 17 7 0 10 -262144 -1 -1 0 1
;
#X obj 37 76 gemwin 50;
#X msg 119 44 lighting 1;
#X obj 120 20 loadbang;
#X obj 36 100 gemhead;
#X obj 281 111 nb-link~ \$0 1 2;
#X obj 231 416 *~ 0;
#X msg 278 416 1;
#X msg 179 563 stop;
#X obj 320 449 hip~ 5;
#X obj 320 476 expr~ tanh($v1);
#X obj 321 416 *~ 0;
#X msg 274 440 0;
#X obj 143 594 writesf~ 2;
#X obj 281 231 nb-link~ \$0 2 4;
#X obj 281 211 nb-link~ \$0 2 3;
#X obj 281 131 nb-link~ \$0 1 3;
#X obj 281 151 nb-link~ \$0 1 4;
#X obj 281 171 nb-link~ \$0 1 5;
#X obj 281 191 nb-link~ \$0 1 6;
#X obj 281 251 nb-link~ \$0 2 5;
#X obj 281 271 nb-link~ \$0 2 6;
#X obj 281 291 nb-link~ \$0 3 4;
#X obj 281 311 nb-link~ \$0 3 5;
#X obj 281 331 nb-link~ \$0 3 6;
#X obj 281 351 nb-link~ \$0 4 5;
#X obj 281 371 nb-link~ \$0 4 6;
#X obj 281 391 nb-link~ \$0 5 6;
#X msg 278 56 step 5;
#X msg 337 86 g \$1;
#X obj 337 63 f 10;
```

```
#X floatatom 470 236 5 0 0 0 - - -;
#X obj 202 28 t b b b;
#X obj 413 183 timer;
#X floatatom 414 236 5 0 0 0 - - -;
#X obj 413 208 / 1000;
#X msg 452 116 bang;
#X obj 472 143 metro 1000;
#X obj 473 186 f;
#X obj 57 157 scale 0.125;
#X msg 389 17 16;
#X obj 57 180 t a a a b;
#X obj 150 201 t b a a;
#X obj 105 248 colorRGB 1 1 1 1;
#X obj 106 270 translateXYZ 0 0 -16;
#X obj 149 180 separator;
#X obj 395 66 * 1;
#X obj 56 343 nb-body~ \$0 6 1;
#X obj 107 290 rotateXYZ 0 0 0.3;
#X msg 399 39 1;
#X obj 56 363 nb-body~ \$0 6 2;
#X obj 56 423 nb-body~ \$0 6 5;
#X obj 56 443 nb-body~ \$0 6 6;
#X obj 56 383 nb-body~ \$0 6 3;
#X obj 56 403 nb-body~ \$0 6 4;
#X msg 433 42 0.9999;
#X msg 503 41 1.0001;
#X obj 450 -1 bng 15 250 50 0 empty empty empty 17 7 0 10 -262144 -1
-1;
#X obj 146 518 spigot;
#X obj 198 523 tgl 15 0 empty empty empty 17 7 0 10 -262144 -1 -1 1
1;
#X obj 467 72 moses 16;
#X obj 468 93 moses 0.25;
#X msg 353 18 0.25;
#X msg 108 543 open /data/tmp/i-i.wav \, start;
#X obj 105 225 pix_snap2tex 0 0 788 576;
#X msg 109 77 dimen 788 576;
#X obj 106 320 rectangle 67 49;
#X obj 450 20 t b b b b b;
#X msg 534 7 0 \, 1;
#X msg 272 14 randomize \, positions;
#X obj 35 130 t a a;
#X obj 89 480 spigot;
#X obj 140 481 tgl 15 0 empty empty empty 17 7 0 10 -262144 -1 -1 1
1;
#X obj 37 503 spigot;
#X obj 35 529 pix_write;
#X msg 337 573 auto 1 \, file /data/tmp/i-i/i 0;
#X msg 169 452 1;
#X connect 0 0 1 0;
#X connect 1 0 2 0;
#X connect 1 0 23 0;
#X connect 4 0 57 2;
#X connect 4 1 15 2;
#X connect 5 0 4 0;
#X connect 6 0 42 0;
#X connect 7 0 4 0;
```

```
#X connect 8 0 11 0;
#X connect 9 0 11 0;
#X connect 10 0 11 0;
#X connect 12 0 11 0;
#X connect 13 0 12 0;
#X connect 13 0 75 0;
#X connect 14 0 80 0;
#X connect 15 0 26 0;
#X connect 15 1 26 1;
#X connect 15 2 26 2;
#X connect 16 0 0 0;
#X connect 17 0 16 1;
#X connect 17 0 21 1;
#X connect 18 0 23 0;
#X connect 19 0 20 0;
#X connect 20 0 2 1;
#X connect 20 0 23 1;
#X connect 21 0 19 0;
#X connect 22 0 16 1;
#X connect 22 0 21 1;
#X connect 24 0 30 0;
#X connect 24 1 30 1;
#X connect 24 2 30 2;
#X connect 25 0 24 0;
#X connect 25 1 24 1;
#X connect 25 2 24 2;
#X connect 26 0 27 0;
#X connect 26 1 27 1;
#X connect 26 2 27 2;
#X connect 27 0 28 0;
#X connect 27 1 28 1;
#X connect 27 2 28 2;
#X connect 28 0 29 0;
#X connect 28 1 29 1;
#X connect 28 2 29 2;
#X connect 29 0 25 0;
#X connect 29 1 25 1;
#X connect 29 2 25 2;
#X connect 30 0 31 0;
#X connect 30 1 31 1;
#X connect 30 2 31 2;
#X connect 31 0 32 0;
#X connect 31 1 32 1;
#X connect 31 2 32 2;
#X connect 32 0 33 0;
#X connect 32 1 33 1;
#X connect 32 2 33 2;
#X connect 33 0 34 0;
#X connect 33 1 34 1;
#X connect 33 2 34 2;
#X connect 34 0 35 0;
#X connect 34 1 35 1;
#X connect 34 2 35 2;
#X connect 35 0 36 0;
#X connect 35 1 36 1;
#X connect 35 2 36 2;
#X connect 36 0 37 0;
```

```
#X connect 36 1 37 1;
#X connect 36 2 37 2;
#X connect 37 0 16 0;
#X connect 37 1 21 0;
#X connect 38 0 4 0;
#X connect 39 0 4 0;
#X connect 40 0 39 0;
#X connect 40 0 48 1;
#X connect 40 0 56 0;
#X connect 42 0 7 0;
#X connect 42 1 38 0;
#X connect 42 2 40 0;
#X connect 43 0 45 0;
#X connect 45 0 44 0;
#X connect 46 0 43 0;
#X connect 46 0 47 0;
#X connect 47 0 43 1;
#X connect 47 0 48 0;
#X connect 48 0 41 0;
#X connect 49 0 51 0;
#X connect 50 0 40 1;
#X connect 51 0 57 0;
#X connect 51 1 3 0;
#X connect 51 2 55 0;
#X connect 51 3 5 0;
#X connect 52 0 74 0;
#X connect 52 1 74 0;
#X connect 52 2 57 1;
#X connect 53 0 54 0;
#X connect 54 0 58 0;
#X connect 55 0 52 0;
#X connect 56 0 40 1;
#X connect 56 0 70 0;
#X connect 57 0 60 0;
#X connect 57 1 60 1;
#X connect 57 2 60 2;
#X connect 58 0 76 0;
#X connect 59 0 56 1;
#X connect 60 0 63 0;
#X connect 60 1 63 1;
#X connect 60 2 63 2;
#X connect 61 0 62 0;
#X connect 61 1 62 1;
#X connect 61 2 62 2;
#X connect 63 0 64 0;
#X connect 63 1 64 1;
#X connect 63 2 64 2;
#X connect 64 0 61 0;
#X connect 64 1 61 1;
#X connect 64 2 61 2;
#X connect 65 0 56 1;
#X connect 66 0 56 1;
#X connect 67 0 77 0;
#X connect 68 0 73 0;
#X connect 69 0 68 1;
#X connect 70 0 71 0;
#X connect 70 1 65 0;
```

```
#X connect 72 0 40 1;
#X connect 73 0 23 0;
#X connect 74 0 53 0;
#X connect 75 0 11 0;
#X connect 77 0 46 0;
#X connect 77 0 68 0;
#X connect 77 0 86 0;
#X connect 77 0 17 0;
#X connect 77 1 66 0;
#X connect 77 2 72 0;
#X connect 77 3 79 0;
#X connect 77 4 78 0;
#X connect 78 0 11 0;
#X connect 79 0 4 0;
#X connect 80 0 83 0;
#X connect 80 1 49 0;
#X connect 81 0 83 1;
#X connect 82 0 81 1;
#X connect 83 0 84 0;
#X connect 85 0 84 0;
#X connect 86 0 81 0;
```

46 interstellar-interference/Makefile

```
nbody.pd_linux: nbody.c
	gcc -O3 -std=c99 -Wall -pedantic -shared -s -o nbody.pd_linux nbody.c -↙
	    ↳ lm -I$(HOME)/opt/include -fPIC
```

47 interstellar-interference/nb-body~.pd

```
#N canvas 0 0 556 397 10;
#X obj 22 18 inlet;
#X obj 28 265 outlet;
#X obj 23 47 t a a;
#X obj 70 46 separator;
#X obj 287 17 loadbang;
#X obj 236 18 inlet;
#X obj 70 104 translate;
#X obj 70 158 color;
#X msg 178 203 20;
#X msg 177 103 1;
#X obj 174 170 hsv2rgb;
#X obj 175 148 pack f 1 1;
#X obj 175 128 / \$2;
#X obj 130 73 route \$3;
#X obj 287 40 f \$3;
#X obj 233 284 outlet;
#X obj 235 41 t a a;
#X obj 248 179 phasor~;
#X obj 249 208 -~ 0.5;
#X obj 251 233 *~ 2;
#X obj 249 258 s~ \$1-source-l-\$3;
#X obj 378 179 phasor~;
#X obj 379 208 -~ 0.5;
#X obj 381 233 *~ 2;
#X obj 379 258 s~ \$1-source-r-\$3;
```

```
#X obj 403 38 loadbang;
#X obj 402 67 f \$3;
#X obj 403 123 / \$2;
#X obj 356 127 * -1;
#X obj 402 97 - 3.5;
#X obj 403 147 * 1;
#X obj 287 71 * 60;
#X obj 287 92 + 360;
#X obj 135 311 curve 2;
#X obj 150 261 list;
#X obj 175 230 t a b a;
#X obj 126 21 inlet;
#X obj 103 259 t a a;
#X obj 98 331 outlet;
#X obj 136 288 color;
#X msg 198 311 width 2;
#X obj 107 151 hsv2rgb;
#X msg 107 172 \$1 \$2 \$3 0.5;
#X msg 167 189 \$1 \$2 \$3 1;
#X obj 70 202 sphere 0.5;
#X obj 70 221 sphere 1;
#X obj 69 240 sphere 2;
#X obj 80 131 pack f 1 0.25;
#X connect 0 0 2 0;
#X connect 2 0 1 0;
#X connect 2 1 3 0;
#X connect 3 0 6 0;
#X connect 4 0 9 0;
#X connect 4 0 8 0;
#X connect 4 0 14 0;
#X connect 4 0 40 0;
#X connect 5 0 16 0;
#X connect 6 0 7 0;
#X connect 7 0 44 0;
#X connect 8 0 44 2;
#X connect 8 0 45 2;
#X connect 8 0 46 2;
#X connect 9 0 6 1;
#X connect 10 0 43 0;
#X connect 11 0 10 0;
#X connect 12 0 11 0;
#X connect 12 0 47 0;
#X connect 13 0 6 2;
#X connect 13 0 35 0;
#X connect 14 0 12 0;
#X connect 14 0 31 0;
#X connect 16 0 15 0;
#X connect 16 1 13 0;
#X connect 17 0 18 0;
#X connect 18 0 19 0;
#X connect 19 0 20 0;
#X connect 21 0 22 0;
#X connect 22 0 23 0;
#X connect 23 0 24 0;
#X connect 25 0 26 0;
#X connect 26 0 29 0;
#X connect 27 0 30 0;
```

```
#X connect 28 0 17 1;
#X connect 29 0 27 0;
#X connect 30 0 21 1;
#X connect 30 0 28 0;
#X connect 31 0 32 0;
#X connect 32 0 17 0;
#X connect 32 0 21 0;
#X connect 34 0 33 1;
#X connect 35 0 34 1;
#X connect 35 1 34 0;
#X connect 35 2 33 2;
#X connect 36 0 37 0;
#X connect 37 0 38 0;
#X connect 37 1 39 0;
#X connect 39 0 33 0;
#X connect 40 0 33 0;
#X connect 41 0 42 0;
#X connect 42 0 7 1;
#X connect 43 0 39 1;
#X connect 44 0 45 0;
#X connect 45 0 46 0;
#X connect 47 0 41 0;
```

48 interstellar-interference/nb-link~.pd

```
#N canvas 0 0 653 394 10;
#X obj 15 14 inlet~;
#X obj 16 267 outlet~;
#X obj 459 100 route \$2;
#X obj 458 126 route \$3;
#X obj 439 37 inlet;
#X obj 436 278 outlet;
#X obj 368 164 swap 100;
#X obj 369 185 -;
#X obj 368 207 dbtorms;
#X obj 368 117 log;
#X obj 370 96 + 1;
#X obj 30 197 *~;
#X obj 367 253 vline~;
#X obj 368 230 pack f f;
#X obj 472 192 timer;
#X obj 439 64 t a a;
#X obj 459 152 t a b b;
#X obj 29 87 samphold~;
#X obj 53 156 samphold~;
#X obj 107 108 *~ 16;
#X obj 110 130 wrap~;
#X obj 103 63 *~ 16;
#X obj 106 85 wrap~;
#X obj 28 176 *~;
#X obj 90 13 r~ \$1-source-l-\$3;
#X obj 28 40 r~ \$1-source-l-\$2;
#X obj 155 34 inlet~;
#X obj 156 287 outlet~;
#X obj 170 217 *~;
#X obj 169 107 samphold~;
#X obj 193 176 samphold~;
```

```
#X obj 247 128 *~ 16;
#X obj 250 150 wrap~;
#X obj 243 83 *~ 16;
#X obj 246 105 wrap~;
#X obj 168 196 *~;
#X obj 168 60 r~ \$1-source-r-\$2;
#X obj 230 33 r~ \$1-source-r-\$3;
#X obj 369 142 * 25;
#X connect 0 0 1 0;
#X connect 2 0 3 0;
#X connect 3 0 16 0;
#X connect 4 0 15 0;
#X connect 6 0 7 0;
#X connect 6 1 7 1;
#X connect 7 0 8 0;
#X connect 8 0 13 0;
#X connect 9 0 38 0;
#X connect 10 0 9 0;
#X connect 11 0 1 0;
#X connect 12 0 11 1;
#X connect 12 0 28 1;
#X connect 13 0 12 0;
#X connect 14 0 13 1;
#X connect 15 0 5 0;
#X connect 15 1 2 0;
#X connect 16 0 10 0;
#X connect 16 1 14 0;
#X connect 16 2 14 1;
#X connect 17 0 23 0;
#X connect 18 0 23 1;
#X connect 19 0 20 0;
#X connect 20 0 18 1;
#X connect 21 0 22 0;
#X connect 22 0 17 1;
#X connect 23 0 11 0;
#X connect 24 0 18 0;
#X connect 24 0 21 0;
#X connect 25 0 17 0;
#X connect 25 0 19 0;
#X connect 26 0 27 0;
#X connect 28 0 27 0;
#X connect 29 0 35 0;
#X connect 30 0 35 1;
#X connect 31 0 32 0;
#X connect 32 0 30 1;
#X connect 33 0 34 0;
#X connect 34 0 29 1;
#X connect 35 0 28 0;
#X connect 36 0 29 0;
#X connect 36 0 31 0;
#X connect 37 0 30 0;
#X connect 37 0 33 0;
#X connect 38 0 6 0;
```

49 interstellar-interference/nbody.c

```
#include <math.h>
```

```
#include <stdlib.h>

#include <m_pd.h>

t_class *nbody_class;

struct vec3 {
  double x;
  double y;
  double z;
};

struct body {
  double m;
  struct vec3 p;
  struct vec3 v;
  struct vec3 a;
};

struct nbody {
  t_object pd;
  t_outlet *o1;
  t_outlet *o2;
  unsigned int n;
  double g;
  double h;
  struct body *bs;
};

void nbody_g(struct nbody *self, t_float g) {
  self->g = g;
}

void nbody_step(struct nbody *self, t_float sf) {
  int s = floor(sf);
  if (s < 1) s = 1;
  for (int k = 0; k < s; ++k) {
    for (int i = 0; i < self->n; ++i) {
      struct vec3 *pi = &self->bs[i].p;
      struct vec3 f = { 0, 0, 0 };
      for (int j = 0; j < self->n; ++j) {
        if (i != j) {
          struct vec3 *pj = &self->bs[j].p;
          struct vec3 d = { pj->x - pi->x, pj->y - pi->y, pj->z - pi->z };
          double d2 = d.x * d.x + d.y * d.y + d.z * d.z;
          double k = self->g * self->bs[j].m / d2;
          f.x += k * d.x; f.y += k * d.y; f.z += k * d.z;
        }
      }
      self->bs[i].a.x = f.x; self->bs[i].a.y = f.y; self->bs[i].a.z = f.z;
    }
    double h = self->h;
    for (int i = 0; i < self->n; ++i) {
      struct body *b = &self->bs[i];
      b->v.x += b->a.x * h; b->v.y += b->a.y * h; b->v.z += b->a.z * h;
      b->p.x += b->v.x * h; b->p.y += b->v.y * h; b->p.z += b->v.z * h;
    }
```

```
  }
}

void nbody_randomize(struct nbody *self) {
  double m = 0;
  struct vec3 c = { 0, 0, 0 };
  for (int i = 0; i < self->n; ++i) {
    struct body *b = &self->bs[i];
    b->m = 1;
    b->p.x = 20.0 * ((double) rand() / (double) RAND_MAX - 0.5);
    b->p.y = 20.0 * ((double) rand() / (double) RAND_MAX - 0.5);
    b->p.z = 20.0 * ((double) rand() / (double) RAND_MAX - 0.5);
    b->v.x = 0; b->v.y = 0; b->v.z = 0;
    b->a.x = 0; b->a.y = 0; b->a.z = 0;
    m += b->m;
    c.x += b->m * b->p.x; c.y += b->m * b->p.y; c.z += b->m * b->p.z;
  }
  c.x /= m; c.y /= m; c.z /= m;
  for (int i = 0; i < self->n; ++i) {
    struct body *b = &self->bs[i];
    b->p.x -= c.x; b->p.y -= c.y; b->p.z -= c.z;
  }
}

void nbody_positions(struct nbody *self) {
  t_atom atoms[4];
  for (int i = 0; i < self->n; ++i) {
    struct vec3 *p = &self->bs[i].p;
    SETFLOAT(&atoms[0], i+1);
    SETFLOAT(&atoms[1], p->x);
    SETFLOAT(&atoms[2], p->y);
    SETFLOAT(&atoms[3], p->z);
    outlet_list(self->o1, &s_list, 4, atoms);
  }
}

void nbody_distances(struct nbody *self) {
  t_atom atoms[3];
  for (int i = 0; i < self->n; ++i) {
    struct vec3 *pi = &self->bs[i].p;
    for (int j = 0; j < self->n; ++j) {
      if (i < j) {
        struct vec3 *pj = &self->bs[j].p;
        struct vec3 d = { pj->x - pi->x, pj->y - pi->y, pj->z - pi->z };
        SETFLOAT(&atoms[0], i+1);
        SETFLOAT(&atoms[1], j+1);
        SETFLOAT(&atoms[2], sqrt(d.x * d.x + d.y * d.y + d.z * d.z));
        outlet_list(self->o2, &s_list, 3, atoms);
      }
    }
  }
}

struct nbody *nbody_new(t_floatarg n) {
  struct nbody *self = pd_new(nbody_class);
  self->o1 = outlet_new(&self->pd, &s_list);
  self->o2 = outlet_new(&self->pd, &s_list);
```

```
  self->n = floor(n);
  if (self->n < 1) self->n = 1;
  self->g = 1;
  self->h = 0.001;
  self->bs = malloc(self->n * sizeof(struct body));
  nbody_randomize(self);
  return self;
}

void nbody_free(struct nbody *self) {
  if (self && self->bs) {
    free(self->bs);
  }
}

extern void nbody_setup() {
  nbody_class = class_new(gensym("nbody"), nbody_new, nbody_free, sizeof(struct nbody), 0, A_DEFFLOAT, 0);
  if (!nbody_class) return;
  class_addmethod(nbody_class, nbody_distances, gensym("distances"), 0);
  class_addmethod(nbody_class, nbody_positions, gensym("positions"), 0);
  class_addmethod(nbody_class, nbody_randomize, gensym("randomize"), 0);
  class_addmethod(nbody_class, nbody_step, gensym("step"), A_FLOAT, 0);
  class_addmethod(nbody_class, nbody_g, gensym("g"), A_FLOAT, 0);
}
```

50 meshwalk/.gitignore

```
meshwalk-1.0
meshwalk-2.0
meshwalk-3.0
meshwalk-3.1
meshwalk-3.2
*.aux
*.log
*.pdf
*.out
*.ppm
*.mov
*.mp4
```

51 meshwalk/Makefile

```
all: meshwalk-1.0 meshwalk-2.0 meshwalk-3.0 meshwalk-3.1 meshwalk-3.2

clean:
	-rm meshwalk-1.0 meshwalk-2.0 meshwalk-3.0 meshwalk-3.1 meshwalk-3.2

.PHONY: all clean

meshwalk-1.0: meshwalk-1.0.c
	gcc -std=c99 -Wall -Wextra -pedantic -O3 -march=native \
	    -o meshwalk-1.0 meshwalk-1.0.c -lGLEW -lGL -lglfw -lm

meshwalk-2.0: meshwalk-2.0.c
	gcc -std=c99 -Wall -Wextra -pedantic -Ofast -ffast-math -march=native \
```

```
	-o meshwalk-2.0 meshwalk-2.0.c \
	-ljack -lsndfile -lGLEW -lGL -lglfw -lm

meshwalk-3.0: meshwalk-3.0.c
	gcc -std=c99 -Wall -Wextra -pedantic \
	-Ofast -ffast-math -march=native -fopenmp \
	-o meshwalk-3.0 meshwalk-3.0.c \
	-ljack -lsndfile -lGLEW -lGL -lglfw -lm

meshwalk-3.1: meshwalk-3.1.c
	gcc -std=c99 -Wall -Wextra -pedantic \
	-Ofast -ffast-math -march=native -fopenmp \
	-o meshwalk-3.1 meshwalk-3.1.c \
	-ljack -lsndfile -lGLEW -lGL -lglfw -lm

meshwalk-3.2: meshwalk-3.2.c
	gcc -std=c99 -Wall -Wextra -pedantic \
	-Ofast -ffast-math -march=native -fopenmp \
	-o meshwalk-3.2 meshwalk-3.2.c \
	-ljack -lsndfile -lGLEW -lGL -lglfw -lm
```

52 meshwalk/meshwalk-1.0.c

```
/*
gcc -std=c99 -Wall -Wextra -pedantic -O3 -march=native \
    -o meshwalk-1.0 meshwalk-1.0.c -lGLEW -lGL -lglfw -lm
for i in $(seq 180) ; do pngtopnm < meshwalk-1.0-title.png ; done |
ffmpeg -r 60 -i - -vf fade=in:30:30,fade=out:120:30 meshwalk-1.0-title-%04d.ppm
for i in $(seq 180) ; do pngtopnm < meshwalk-1.0-credits.png ; done |
ffmpeg -r 60 -i - -vf fade=in:30:30,fade=out:120:30 meshwalk-1.0-credits-%04d.↙
    ↪ ppm
( cat meshwalk-1.0-title-0*.ppm ; ./meshwalk-1.0 ; cat meshwalk-1.0-credits-0*.↙
    ↪ ppm ) |
ffmpeg -r 60 -i - -i silence.wav \
    -pix_fmt yuv420p -profile:v high -level:v 4.1 -tune:v animation \
    -movflags +faststart meshwalk-1.0.mov
*/

#include <math.h>
#include <stdio.h>
#include <stdlib.h>

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

#define pi 3.141592653589793

static void debug_program(GLuint program) {
  GLint status = 0;
  glGetProgramiv(program, GL_LINK_STATUS, &status);
  GLint length = 0;
  glGetProgramiv(program, GL_INFO_LOG_LENGTH, &length);
  char *info = 0;
  if (length) {
    info = malloc(length + 1);
    info[0] = 0;
    glGetProgramInfoLog(program, length, 0, info);
```

```
    info[length] = 0;
  }
  if ((info && info[0]) || ! status) {
    fprintf(stderr, "program link info:\n%s", info ? info : "(no info log)");
  }
  if (info) {
    free(info);
  }
}

static void debug_shader(GLuint shader, GLenum type, const char *source) {
  GLint status = 0;
  glGetShaderiv(shader, GL_COMPILE_STATUS, &status);
  GLint length = 0;
  glGetShaderiv(shader, GL_INFO_LOG_LENGTH, &length);
  char *info = 0;
  if (length) {
    info = malloc(length + 1);
    info[0] = 0;
    glGetShaderInfoLog(shader, length, 0, info);
    info[length] = 0;
  }
  if ((info && info[0]) || ! status) {
    const char *type_str = "unknown";
    switch (type) {
      case GL_VERTEX_SHADER: type_str = "vertex"; break;
      case GL_FRAGMENT_SHADER: type_str = "fragment"; break;
      case GL_COMPUTE_SHADER: type_str = "compute"; break;
    }
    fprintf
      ( stderr
      , "%s shader compile info:\n%s\nshader source:\n%s"
      , type_str
      , info ? info : "(no info log)"
      , source ? source : "(no source)"
      );
  }
  if (info) {
    free(info);
  }
}

static GLuint vertex_fragment_shader(const char *vert, const char *frag) {
  GLuint program = glCreateProgram();
  {
    GLuint shader = glCreateShader(GL_VERTEX_SHADER);
    glShaderSource(shader, 1, &vert, 0);
    glCompileShader(shader);
    debug_shader(shader, GL_VERTEX_SHADER, vert);
    glAttachShader(program, shader);
    glDeleteShader(shader);
  }
  {
    GLuint shader = glCreateShader(GL_FRAGMENT_SHADER);
    glShaderSource(shader, 1, &frag, 0);
    glCompileShader(shader);
    debug_shader(shader, GL_FRAGMENT_SHADER, frag);
```

```
90      glAttachShader(program, shader);
        glDeleteShader(shader);
      }
      glLinkProgram(program);
      debug_program(program);
95    return program;
    }

    #define W (16 * 8)
    #define H ( 9 * 8)
100
    // positions (even store average, odd store edge offsets)
    static double g[H][W];
    // velocities (odd for edges)
    static double dg[H][W];
105
    // OpenGL triangles: one quad per even coordinate, convexity guaranteed
    static GLfloat t[H/2 + 2][W/2 + 2][2][2][3][3];

    // http://burtleburtle.net/bob/hash/integer.html
110 static uint32_t burtle_hash(uint32_t a)
    {
        a = (a+0x7ed55d16) + (a<<12);
        a = (a^0xc761c23c) ^ (a>>19);
        a = (a+0x165667b1) + (a<<5);
115     a = (a+0xd3a2646c) ^ (a<<9);
        a = (a+0xfd7046c5) + (a<<3);
        a = (a^0xb55a4f09) ^ (a>>16);
        return a;
    }
120
    // pseudo-random uniform number in [0,1)
    static double uniform(uint32_t x, uint32_t y, uint32_t c)
    {
      return
125     burtle_hash(x +
        burtle_hash(y +
        burtle_hash(c))) /
        (double) (0x100000000LL);
    }
130
    // randomize positions, no velocity
    static void init()
    {
      for (int j = 0; j < H; ++j)
135   for (int i = 0; i < W; ++i)
      {
        g[j][i] = 0.25 + 0.5 * uniform(i, j, -1);
        dg[j][i] = 0;
      }
140 }

    // compute average deviation from even coordinates
    static void even()
    {
145   for (int j = 0; j < H; ++j)
      for (int i = 0; i < W; ++i)
```

```
  if (((i & 1) == 0) && ((j & 1) == 0))
  {
    int ip = (i + 1) % W;
    int jp = (j + 1) % H;
    int im = (i - 1 + W) % W;
    int jm = (j - 1 + H) % H;
    double xp =     g[j][ip];
    double xm = 1 - g[j][im];
    double yp =     g[jp][i];
    double ym = 1 - g[jm][i];
    double m = 0.25 * (xp + xm + yp + ym);
    g[j][i] = m;
  }
}

// update deviation of even coordinates
static void odd(int k)
{
  // integration time constant
  const double dt = 0.1;
  // friction proportional to velocity
  const double friction = 0.005;
  // elastic collisions at the end points of the intervals
  const double elastic = 0.125;
  // time-varying forcing of the spring constant gives interesting effects
  const double spring = 0.1 + 0.02 * sin(2 * pi * k / 60.0);

  for (int j = 0; j < H; ++j)
  for (int i = 0; i < W; ++i)
  if (((i + j) & 1) == 1) // edges
  {
    // get current position
    double a = g[j][i];
    // get aveage of endpoints
    double ap, am;
    if (i & 1) // horizontal
    {
      int ip = (i + 1) % W;
      int im = (i - 1 + W) % W;
      ap = g[j][ip];
      am = g[j][im];
    }
    else // vertical
    {
      int jm = (j - 1 + H) % H;
      int jp = (j + 1) % H;
      ap = g[jp][i];
      am = g[jm][i];
    }
    double target = 0.5 * (ap + 1 - am);
    // get current velocity
    double d = dg[j][i];
    // compute force
    double f = spring * (target - a) - friction * d;
    // integrate position
    a = a + dt * d;
    // handle far out of bounds
```

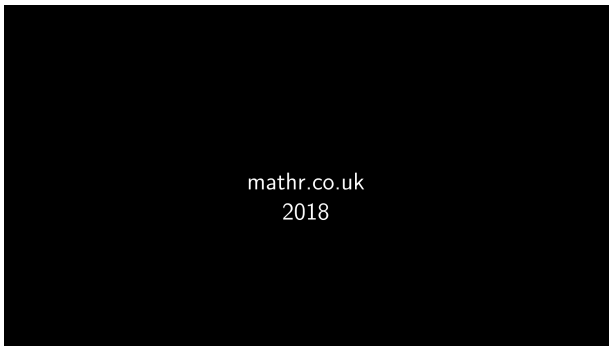
```
      if (a >  2) { a = 0.5; d = 0; }
      if (a < -1) { a = 0.5; d = 0; }
      // handle elastic reflections at end points
      if (a > 1) { a = 1 - (a - 1); d = - elastic * d; }
      if (a < 0) { a = 0 - (a - 0); d = - elastic * d; }
      // store new position
      g[j][i] = a;
      // integrate velocity
      dg[j][i] = d + dt * f;
    }
  }

  // update triangle positions for OpenGL
  static void tris()
  {
    for (int j0 = -1; j0 <= H; ++j0)
    for (int i0 = -1; i0 <= W; ++i0)
    {
      int i = (i0 + W) % W;
      int j = (j0 + H) % H;
      if ((i & 1) == (j & 1)) // even coordinates own a quad
      {
        int x = i0 >> 1;
        int y = j0 >> 1;
        int j1 = y + 1;
        int i1 = x + 1;
        int f = j & 1; // quad type (coordinates both even vs both odd)
        int c = f | ((i0 ^ j0) & 2); // colour
        t[j1][i1][f][0][0][0] = x + f;
        t[j1][i1][f][0][0][1] = y;
        t[j1][i1][f][0][0][2] = c;
        t[j1][i1][f][0][1][0] = x + 1;
        t[j1][i1][f][0][1][1] = y + f;
        t[j1][i1][f][0][1][2] = c;
        t[j1][i1][f][0][2][0] = x + f;
        t[j1][i1][f][0][2][1] = y + 1;
        t[j1][i1][f][0][2][2] = c;
        t[j1][i1][f][1][0][2] = c;
        t[j1][i1][f][1][1][2] = c;
        t[j1][i1][f][1][2][0] = x;
        t[j1][i1][f][1][2][1] = y + f;
        t[j1][i1][f][1][2][2] = c;
        t[j1][i1][f][0][0][  f] += g[(j     ) % H][(i + 1) % W];
        t[j1][i1][f][0][1][1 - f] += g[(j + 1) % H][(i + 2) % W];
        t[j1][i1][f][0][2][  f] += g[(j + 2) % H][(i + 1) % W];
        t[j1][i1][f][1][2][1 - f] += g[(j + 1) % H][(i     ) % W];
        t[j1][i1][f][1][0][0] = t[j1][i1][f][0][0][0];
        t[j1][i1][f][1][0][1] = t[j1][i1][f][0][0][1];
        t[j1][i1][f][1][1][0] = t[j1][i1][f][0][2][0];
        t[j1][i1][f][1][1][1] = t[j1][i1][f][0][2][1];
      }
    }
  }

  // GLFW keyboard callback
  static void keycb
    ( GLFWwindow *window
```

```
    , int key
    , int scancode
    , int action
    , int mods
265 )
{
  (void) scancode;
  (void) mods;
  if (action == GLFW_PRESS)
270 {
    switch (key)
    {
      case GLFW_KEY_Q:
      case GLFW_KEY_ESCAPE:
275     glfwSetWindowShouldClose(window, GL_TRUE);
        break;
    }
  }
}
280
// entry point
extern int main()
{
  // start OpenGL 3.3 core profile 1920x1080 full screen
285 int w = 1920;
  int h = 1080;
  glfwInit();
  glfwWindowHint(GLFW_CLIENT_API, GLFW_OPENGL_API);
  glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
290 glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
  glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
  glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
  glfwWindowHint(GLFW_RESIZABLE, GL_FALSE);
  glfwWindowHint(GLFW_DECORATED, GL_FALSE);
295 glfwWindowHint(GLFW_SAMPLES, 32);
  GLFWwindow *window = glfwCreateWindow(w, h, "meshwalk", 0, 0);
  glfwMakeContextCurrent(window);
  glfwSetKeyCallback(window, keycb);
  glewExperimental = GL_TRUE;
300 glewInit();
  glGetError(); // discard common error from glew
  // set up vertex array object
  glEnable(GL_MULTISAMPLE);
  glClearColor(0, 0, 0.5, 1);
305 GLuint vao;
  glGenVertexArrays(1, &vao);
  glBindVertexArray(vao);
  // set up buffer
  GLuint vbo;
310 glGenBuffers(1, &vbo);
  glBindBuffer(GL_ARRAY_BUFFER, vbo);
  glBufferData(GL_ARRAY_BUFFER, sizeof(t), 0, GL_DYNAMIC_DRAW);
  // set up shader
  const char *s_vertex =
315   "#version 330 core\n"
    "uniform mat4 MVP;\n"
    "layout (location = 0) in vec3 position;\n"
```

```
    "flat out int c;\n"
    "void main()\n"
    "{\n"
    "  c = int(position.z);\n"
    "  gl_Position = MVP * vec4(position.xy, 0.0, 1.0);\n"
    "}\n"
    ;
  const char *s_fragment =
    "#version 330 core\n"
    "flat in int c;\n"
    "layout (location = 0) out vec4 colour;\n"
    "void main()\n"
    "{\n"
    "  vec3 blue = vec3(0.0, 0.5, 1.0);\n"
    "  vec3 white = vec3(1.0);\n"
    "  vec3 red = vec3(1.0, 0.2, 0.0);\n"
    "  vec3 black = vec3(0.0);\n"
    "  colour = vec4(vec3[4](blue, white, red, black)[c], 1.0);\n"
    "}\n"
  ;
  GLuint display = vertex_fragment_shader(s_vertex, s_fragment);
  glUseProgram(display);
  GLfloat m[4][4] = // ortho2d
    { { 4.0 / W, 0, 0, 0 }
    , { 0, 4.0 / H, 0, 0 }
    , { 0, 0, -1, 0 }
    , { -1, -1, 0, 1 }
    };
  GLint u_MVP = glGetUniformLocation(display, "MVP");
  glUniformMatrix4fv(u_MVP, 1, GL_FALSE, &m[0][0]);
  glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, 0);
  glEnableVertexAttribArray(0);
  // main loop
  unsigned char *video = malloc(w * h * 3);
  init();
  glfwPollEvents();
  for (int k = 0; !glfwWindowShouldClose(window); ++k)
  {
    // step
    even();
    odd(k);
    if (k > 0x3FFFF)
    {
      tris();
      // draw
      glBufferSubData(GL_ARRAY_BUFFER, 0, sizeof(t), t);
      glClear(GL_COLOR_BUFFER_BIT);
      glDrawArrays(GL_TRIANGLES, 0, (H/2 + 2) * (W/2 + 2) * 2 * 2 * 3);
      // download output frame
      glReadPixels(0, 0, w, h, GL_RGB, GL_UNSIGNED_BYTE, video);
      // output PPM to stdout (flipped vertically)
      printf("P6\n%d %d\n255\n", w, h);
      fwrite(video, w * h * 3, 1, stdout);
      // redisplay
      glfwSwapBuffers(window);
      glfwPollEvents();
      if (k > 0x3FFFF + 60 * 234)
```

```
            break;
        }
      }
      // cleanup
      free(video);
      glfwDestroyWindow(window);
      glfwTerminate();
      return 0;
}
```

53 meshwalk/meshwalk-1.0-credits.png



54 meshwalk/meshwalk-1.0-title-and-credits.tex

```
\documentclass{article}
\usepackage{lmodern}

\begin{document}

\centering
\Huge
\sf
meshwalk-1.0

\Large
mathr.co.uk

2018

\end{document}
```

55 meshwalk/meshwalk-1.0-title.png

56 meshwalk/meshwalk-2.0.c

```
/*
  meshwalk -- audiovisual drone
  Copyright (C) 2018  Claude Heiland-Allen <claude@mathr.co.uk>

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

  You should have received a copy of the GNU General Public License
  along with this program.  If not, see <https://www.gnu.org/licenses/>.
*/
/*
# build (or use `make`)
gcc -std=c99 -Wall -Wextra -pedantic -Ofast -ffast-math -march=native \
    -o meshwalk-2.0 meshwalk-2.0.c \
    -ljack -lsndfile -lGLEW -lGL -lglfw -lm

# realtime mode with JACK audio, press Q to exit
./meshwalk-2.0

# render
for i in $(seq 180) ; do pngtopnm < meshwalk-2.0-title.png ; done |
ffmpeg -r 60 -i - -vf fade=in:30:30,fade=out:120:30 meshwalk-2.0-title-%04d.ppm
for i in $(seq 180) ; do pngtopnm < meshwalk-2.0-credits.png ; done |
ffmpeg -r 60 -i - -vf fade=in:30:30,fade=out:120:30 meshwalk-2.0-credits-%04d.ppm
(
  cat meshwalk-2.0-title-0*.ppm
  ./meshwalk-2.0 -nrt
  cat meshwalk-2.0-credits-0*.ppm
) |
ffmpeg -framerate 60 -i - -codec:v png meshwalk-2.0_v.mov
ffmpeg -i meshwalk-2.0_v.mov -i meshwalk-2.0_a.mov \
```

```
    -codec:v copy -codec:a copy -movflags +faststart \
    meshwalk-2.0.mov
ffmpeg -i meshwalk-2.0.mov \
    -pix_fmt yuv420p -profile:v high -level:v 4.1 -tune:v animation \
    -crf:v 20 -b:a 384k -movflags +faststart \
    meshwalk-2.0.mp4
*/

#include <math.h>
#include <stdio.h>
#include <stdlib.h>

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

#include <jack/jack.h>
#include <sndfile.h>

#define SR 48000
#define FPS 60
#define W (16 * 4)
#define H ( 9 * 4)

#define pi 3.141592653589793

// precondition: 0 <= phase <= 1
static inline float cosf9(float phase) {
  float p = fabsf(4.0f * phase - 2.0f) - 1.0f;
  float p2 = p * p;
  // p in -1 .. 1
  float s
    = 1.5707963267948965580e+00f * p
    - 6.4596271553942852250e-01f * p * p2
    + 7.9685048314861006702e-02f * p * p2 * p2
    - 4.6672571910271187789e-03f * p * p2 * p2 * p2
    + 1.4859762069630022552e-04f * p * p2 * p2 * p2 * p2;
  // compiler figures out optimal simd multiplications
  return s;
}

static inline float wrap(float x) { return x - floor(x); }

static void debug_program(GLuint program) {
  GLint status = 0;
  glGetProgramiv(program, GL_LINK_STATUS, &status);
  GLint length = 0;
  glGetProgramiv(program, GL_INFO_LOG_LENGTH, &length);
  char *info = 0;
  if (length) {
    info = malloc(length + 1);
    info[0] = 0;
    glGetProgramInfoLog(program, length, 0, info);
    info[length] = 0;
  }
  if ((info && info[0]) || ! status) {
    fprintf(stderr, "program link info:\n%s", info ? info : "(no info log)");
  }
```

```
  if (info) {
    free(info);
  }
}

static void debug_shader(GLuint shader, GLenum type, const char *source) {
  GLint status = 0;
  glGetShaderiv(shader, GL_COMPILE_STATUS, &status);
  GLint length = 0;
  glGetShaderiv(shader, GL_INFO_LOG_LENGTH, &length);
  char *info = 0;
  if (length) {
    info = malloc(length + 1);
    info[0] = 0;
    glGetShaderInfoLog(shader, length, 0, info);
    info[length] = 0;
  }
  if ((info && info[0]) || ! status) {
    const char *type_str = "unknown";
    switch (type) {
      case GL_VERTEX_SHADER: type_str = "vertex"; break;
      case GL_FRAGMENT_SHADER: type_str = "fragment"; break;
      case GL_COMPUTE_SHADER: type_str = "compute"; break;
    }
    fprintf
      ( stderr
      , "%s shader compile info:\n%s\nshader source:\n%s"
      , type_str
      , info ? info : "(no info log)"
      , source ? source : "(no source)"
      );
  }
  if (info) {
    free(info);
  }
}

static GLuint vertex_fragment_shader(const char *vert, const char *frag) {
  GLuint program = glCreateProgram();
  {
    GLuint shader = glCreateShader(GL_VERTEX_SHADER);
    glShaderSource(shader, 1, &vert, 0);
    glCompileShader(shader);
    debug_shader(shader, GL_VERTEX_SHADER, vert);
    glAttachShader(program, shader);
    glDeleteShader(shader);
  }
  {
    GLuint shader = glCreateShader(GL_FRAGMENT_SHADER);
    glShaderSource(shader, 1, &frag, 0);
    glCompileShader(shader);
    debug_shader(shader, GL_FRAGMENT_SHADER, frag);
    glAttachShader(program, shader);
    glDeleteShader(shader);
  }
  glLinkProgram(program);
  debug_program(program);
```

```
  return program;
}

// positions (even store average, odd store edge offsets)
static float g[H][W];
// velocities (odd for edges)
static float dg[H][W];

// OpenGL triangles: two quads per even coordinate, convexity guaranteed
static GLfloat t[H/2 + 2][W/2 + 2][2][2][3][3];

// audio oscillators: one per quad, double buffered with linear interpolation
static int a_which = 0;
static float pan  [2][H/2 + 2][W/2 + 2][2];
static float pitch[2][H/2 + 2][W/2 + 2][2];
static float level[2][H/2 + 2][W/2 + 2][2];
static float phase[H/2 + 2][W/2 + 2][2];

// http://burtleburtle.net/bob/hash/integer.html
static uint32_t burtle_hash(uint32_t a)
{
    a = (a+0x7ed55d16) + (a<<12);
    a = (a^0xc761c23c) ^ (a>>19);
    a = (a+0x165667b1) + (a<<5);
    a = (a+0xd3a2646c) ^ (a<<9);
    a = (a+0xfd7046c5) + (a<<3);
    a = (a^0xb55a4f09) ^ (a>>16);
    return a;
}

// pseudo-random uniform number in [0,1)
static float uniform(uint32_t x, uint32_t y, uint32_t c)
{
  return
    burtle_hash(x +
    burtle_hash(y +
    burtle_hash(c))) /
    (float) (0x100000000LL);
}

// randomize positions, no velocity
static void init()
{
  for (int j = 0; j < H; ++j)
  for (int i = 0; i < W; ++i)
  {
    g[j][i] = 0.25 + 0.5 * uniform(i, j, 0);
    dg[j][i] = 0;
  }
  for (int j = 0; j < H/2 + 2; ++ j)
  for (int i = 0; i < W/2 + 2; ++ i)
  {
    phase[j][i][0] = uniform(i, j, 1);
    phase[j][i][1] = uniform(i, j, 2);
  }
}
```

```
210  // compute average deviation from even coordinates
     static void even()
     {
       for (int j = 0; j < H; ++j)
       for (int i = 0; i < W; ++i)
215    if (((i & 1) == 0) && ((j & 1) == 0))
       {
         int ip = (i + 1) % W;
         int jp = (j + 1) % H;
         int im = (i - 1 + W) % W;
220      int jm = (j - 1 + H) % H;
         float xp =     g[j][ip];
         float xm = 1 - g[j][im];
         float yp =     g[jp][i];
         float ym = 1 - g[jm][i];
225      float m = 0.25f * (xp + xm + yp + ym);
         g[j][i] = m;
       }
     }

230  // update deviation of even coordinates
     static void odd(int k)
     {
       // integration time constant
       const float dt = 1.0f/32.0f;
235    // friction proportional to velocity
       const float friction = 0.07f * dt;
       // elastic collisions at the end points of the intervals
       const float elastic = 0.125f;
       // time-varying forcing of the spring constant gives interesting effects
240    const float spring = 0.1f + 0.02f * sinf(2.0f * pi * k / 60.0f);

       for (int j = 0; j < H; ++j)
       for (int i = 0; i < W; ++i)
       if (((i + j) & 1) == 1) // edges
245    {
         // get current position
         float a = g[j][i];
         // get aveage of endpoints
         float ap, am;
250      if (i & 1) // horizontal
         {
           int ip = (i + 1) % W;
           int im = (i - 1 + W) % W;
           ap = g[j][ip];
255        am = g[j][im];
         }
         else // vertical
         {
           int jm = (j - 1 + H) % H;
260        int jp = (j + 1) % H;
           ap = g[jp][i];
           am = g[jm][i];
         }
         float target = 0.5f * (ap + 1.0f - am);
265      // get current velocity
         float d = dg[j][i];
```

```
      // compute force
      float f = spring * (target - a) - friction * d;
      // integrate position
      a = a + dt * d;
      // handle far out of bounds
      if (a >  2) { a = 0.5; d = 0; }
      if (a < -1) { a = 0.5; d = 0; }
      // handle elastic reflections at end points
      if (a > 1) { a = 1 - (a - 1); d = - elastic * d; }
      if (a < 0) { a = 0 - (a - 0); d = - elastic * d; }
      // store new position
      g[j][i] = a;
      // integrate velocity
      dg[j][i] = d + dt * f;
    }
  }

  // update triangle positions for OpenGL
  static void tris()
  {
    int w = 1 - a_which;
    float base_pitch[4] = { 150.0f / SR, 250.0f / SR, 150.0f / SR, 100.0f / SR };
    float gain = 0.5f;
    float volume = gain / sqrtf((H/2 + 2) * (W/2 + 2) * 2);
    for (int j0 = -1; j0 <= H; ++j0)
    for (int i0 = -1; i0 <= W; ++i0)
    {
      int i = (i0 + W) % W;
      int j = (j0 + H) % H;
      if ((i & 1) == (j & 1)) // even coordinates own a quad
      {
        int x = i0 >> 1;
        int y = j0 >> 1;
        int j1 = y + 1;
        int i1 = x + 1;
        int f = j & 1; // quad type (coordinates both even vs both odd)
        int c = f | ((i0 ^ j0) & 2); // colour
        t[j1][i1][f][0][0][0] = x + f;
        t[j1][i1][f][0][0][1] = y;
        t[j1][i1][f][0][0][2] = c;
        t[j1][i1][f][0][1][0] = x + 1;
        t[j1][i1][f][0][1][1] = y + f;
        t[j1][i1][f][0][1][2] = c;
        t[j1][i1][f][0][2][0] = x + f;
        t[j1][i1][f][0][2][1] = y + 1;
        t[j1][i1][f][0][2][2] = c;
        t[j1][i1][f][1][0][2] = c;
        t[j1][i1][f][1][1][2] = c;
        t[j1][i1][f][1][2][0] = x;
        t[j1][i1][f][1][2][1] = y + f;
        t[j1][i1][f][1][2][2] = c;
        float top    = g[(j    ) % H][(i + 1) % W];
        float right  = g[(j + 1) % H][(i + 2) % W];
        float bottom = g[(j + 2) % H][(i + 1) % W];
        float left   = g[(j + 1) % H][(i    ) % W];
        float xpos0, ypos0;
        (void) xpos0;
```

```
      {
325     float x0 = t[j1][i1][f][0][0][0];
        float x1 = t[j1][i1][f][0][1][0];
        float x2 = t[j1][i1][f][0][2][0];
        float x3 = t[j1][i1][f][1][2][0];
        float y0 = t[j1][i1][f][0][0][1];
330     float y1 = t[j1][i1][f][0][1][1];
        float y2 = t[j1][i1][f][0][2][1];
        float y3 = t[j1][i1][f][1][2][1];
        xpos0 = 0.25f * (x0 + x1 + x2 + x3 + 1.0f);
        ypos0 = 0.25f * (y0 + y1 + y2 + y3 + 1.0f);
335   }
      t[j1][i1][f][0][0][  f] += top;
      t[j1][i1][f][0][1][1-f] += right;
      t[j1][i1][f][0][2][  f] += bottom;
      t[j1][i1][f][1][2][1-f] += left;
340   float x0 = t[j1][i1][f][0][0][0];
      float x1 = t[j1][i1][f][0][1][0];
      float x2 = t[j1][i1][f][0][2][0];
      float x3 = t[j1][i1][f][1][2][0];
      float y0 = t[j1][i1][f][0][0][1];
345   float y1 = t[j1][i1][f][0][1][1];
      float y2 = t[j1][i1][f][0][2][1];
      float y3 = t[j1][i1][f][1][2][1];
      float xpos = 0.25f * (x0 + x1 + x2 + x3);
      float ypos = 0.25f * (y0 + y1 + y2 + y3);
350   float d1x = x2 - x0;
      float d2x = x3 - x1;
      float d1y = y2 - y0;
      float d2y = y3 - y1;
      d1x *= d1x;
355   d2x *= d2x;
      d1y *= d1y;
      d2y *= d2y;
      float d1 = d1x + d1y;
      float d2 = d2x + d2y;
360   float areaSquared = d1 * d2;
      t[j1][i1][f][1][0][0] = t[j1][i1][f][0][0][0];
      t[j1][i1][f][1][0][1] = t[j1][i1][f][0][0][1];
      t[j1][i1][f][1][1][0] = t[j1][i1][f][0][2][0];
      t[j1][i1][f][1][1][1] = t[j1][i1][f][0][2][1];
365   pan  [w][j1][i1][f] = xpos / ((W >> 1) + 2) * 0.25f;
      pitch[w][j1][i1][f] = pow(1.5, ypos - ypos0) * base_pitch[c];
      level[w][j1][i1][f] = areaSquared * volume;
    }
  }
370 }

static void audiocb(float *out_left, float *out_right, int nframes)
{
  for (int n = 0; n < nframes; ++n)
375 {
    out_left[n] = out_right[n] = 0;
  }
  int w = a_which;
  int w1 = 1 - w;
380 for (int j0 = -1; j0 <= H; ++j0)
```

```
  {
    float l[nframes], r[nframes];
    for (int n = 0; n < nframes; ++n)
    {
385   l[n] = r[n] = 0;
    }
    for (int i0 = -1; i0 <= W; ++i0)
    {
      int i = (i0 + W) % W;
390   int j = (j0 + H) % H;
      if ((i & 1) == (j & 1)) // even coordinates own a quad
      {
        int x = i0 >> 1;
        int y = j0 >> 1;
395     int j1 = y + 1;
        int i1 = x + 1;
        int f = j & 1; // quad type (coordinates both even vs both odd)
        int c = f | ((i0 ^ j0) & 2); // colour
        float pan_0c = cosf9(       pan[w ][j1][i1][f]);
400     float pan_0s = cosf9(0.25f - pan[w ][j1][i1][f]);
        float pan_1c = cosf9(       pan[w1][j1][i1][f]);
        float pan_1s = cosf9(0.25f - pan[w1][j1][i1][f]);
        float pit_0 = pitch[w ][j1][i1][f];
        float pit_1 = pitch[w1][j1][i1][f];
405     float lev_0 = level[w ][j1][i1][f];
        float lev_1 = level[w1][j1][i1][f];
        for (int n = 0; n < nframes; ++n)
        {
          float lin = (n + 0.5) / nframes;
410       float lin1 = 1 - lin;
          float pan1c  = pan_0c* lin1 + pan_1c* lin;
          float pan1s  = pan_0s* lin1 + pan_1s* lin;
          float pitch1 = pit_0 * lin1 + pit_1 * lin;
          float level1 = lev_0 * lin1 + lev_1 * lin;
415       float p = phase[j1][i1][f] = wrap(phase[j1][i1][f] + pitch1);
          float o = 0;
          switch (c)
          {
            case 0: o = 0.5f - p; break;
420         case 1: o = fabsf(p - 0.5f) - 0.25f; break;
            case 2: o = p - 0.5f; break;
            case 3: o = (p < 0.5f) - 0.5f; break;
          }
          o *= level1;
425       l[n] += o * pan1c;
          r[n] += o * pan1s;
        }
      }
    }
430 for (int n = 0; n < nframes; ++n)
    {
      out_left[n] += l[n];
      out_right[n] += r[n];
    }
435 }
  a_which = 1 - a_which;
}
```

```
  jack_client_t *client;
440 jack_port_t *port[2];
  static int processcb(jack_nframes_t nframes, void *arg) {
    (void) arg;
    jack_default_audio_sample_t *out[2];
    out[0] = jack_port_get_buffer(port[0], nframes);
445 out[1] = jack_port_get_buffer(port[1], nframes);
    audiocb(out[0], out[1], nframes);
    return 0;
  }

450 // GLFW keyboard callback
  static void keycb
    ( GLFWwindow *window
    , int key
    , int scancode
455 , int action
    , int mods
    )
  {
    (void) key;
460 (void) scancode;
    (void) mods;
    if (action == GLFW_PRESS)
      glfwSetWindowShouldClose(window, GL_TRUE);
  }
465
  // entry point
  extern int main(int argc, char **argv)
  {
    (void) argv;
470 int RECORD = argc > 1;
    // start OpenGL 3.3 core profile 1920x1080 full screen
    int w = 1920;
    int h = 1080;
    glfwInit();
475 glfwWindowHint(GLFW_CLIENT_API, GLFW_OPENGL_API);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
    glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
    glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
480 glfwWindowHint(GLFW_RESIZABLE, GL_FALSE);
    glfwWindowHint(GLFW_DECORATED, GL_FALSE);
    GLFWwindow *window = glfwCreateWindow(w, h, "meshwalk", 0, 0);
    glfwMakeContextCurrent(window);
    glfwSetKeyCallback(window, keycb);
485 glewExperimental = GL_TRUE;
    glewInit();
    glGetError(); // discard common error from glew
    // set up vertex array object
    glClearColor(0, 1, 0, 1);
490 GLuint vao;
    glGenVertexArrays(1, &vao);
    glBindVertexArray(vao);
    // set up buffer
    GLuint vbo;
```

```
495     glGenBuffers(1, &vbo);
        glBindBuffer(GL_ARRAY_BUFFER, vbo);
        glBufferData(GL_ARRAY_BUFFER, sizeof(t), 0, GL_DYNAMIC_DRAW);
        // set up framebuffer
        GLuint fbo;
500     glGenFramebuffers(1, &fbo);
        glBindFramebuffer(GL_FRAMEBUFFER, fbo);
        GLuint tex;
        glGenTextures(1, &tex);
        glBindTexture(GL_TEXTURE_2D_MULTISAMPLE, tex);
505     glTexImage2DMultisample(GL_TEXTURE_2D_MULTISAMPLE, 8, GL_RGB, w, h, GL_FALSE);
        glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, ↙
            ↳ GL_TEXTURE_2D_MULTISAMPLE, tex, 0);
        GLenum status = glCheckFramebufferStatus(GL_FRAMEBUFFER);
        if (status != GL_FRAMEBUFFER_COMPLETE)
          fprintf(stderr, "GL FRAMEBUFFER STATUS %d\n", status);
510     // set up shader
        const char *s_vertex =
          "#version 330 core\n"
          "uniform mat4 MVP;\n"
          "layout (location = 0) in vec3 position;\n"
515       "flat out int c;\n"
          "void main()\n"
          "{\n"
          "  c = int(position.z);\n"
          "  gl_Position = MVP * vec4(position.xy, 0.0, 1.0);\n"
520       "}\n"
          ;
        const char *s_fragment =
          "#version 330 core\n"
          "flat in int c;\n"
525       "layout (location = 0) out vec4 colour;\n"
          "void main()\n"
          "{\n"
          "  vec3 blue = vec3(0.0, 0.5, 1.0);\n"
          "  vec3 white = vec3(1.0);\n"
530       "  vec3 red = vec3(1.0, 0.2, 0.0);\n"
          "  vec3 black = vec3(0.0);\n"
          "  colour = vec4(vec3[4](blue, white, red, black)[c], 1.0);\n"
          "}\n"
        ;
535     GLuint display = vertex_fragment_shader(s_vertex, s_fragment);
        glUseProgram(display);
        GLfloat m[4][4] = // ortho2d
          { { 4.0 / W, 0, 0, 0 }
          , { 0, 4.0 / H, 0, 0 }
540       , { 0, 0, -1, 0 }
          , { -1, -1, 0, 1 }
          };
        GLint u_MVP = glGetUniformLocation(display, "MVP");
        glUniformMatrix4fv(u_MVP, 1, GL_FALSE, &m[0][0]);
545     glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, 0);
        glEnableVertexAttribArray(0);
        // prepare
        init();
        SF_INFO info = { 0, 48000, 2, SF_FORMAT_WAV | SF_FORMAT_FLOAT, 0, 0 };
550     SNDFILE *sndfile = 0;
```

```
  unsigned char *video = 0;
  if (RECORD)
  {
    sndfile = sf_open("meshwalk-2.0_a.wav", SFM_WRITE, &info);
    video = malloc(w * h * 3);
    // write 3 seconds silence for title
    float frames[SR/FPS][2];
    for (int i = 0; i < SR/FPS; ++i)
      for (int c = 0; c < 2; ++c)
        frames[i][c] = 0;
    for (int k = 0; k < 3 * FPS; ++k)
      sf_writef_float(sndfile, &frames[0][0], SR/FPS);
  }
  else
  {
    if (!(client = jack_client_open("meshwalk", JackNoStartServer, 0))) {
      fprintf (stderr, "jack server not running?\n");
      return 1;
    }
    jack_set_process_callback(client, processcb, 0);
    /* create ports */
    port[0] = jack_port_register(
      client, "output_1", JACK_DEFAULT_AUDIO_TYPE, JackPortIsOutput, 0
    );
    port[1] = jack_port_register(
      client, "output_2", JACK_DEFAULT_AUDIO_TYPE, JackPortIsOutput, 0
    );
    /* activate audio */
    if (jack_activate(client)) {
      fprintf (stderr, "cannot activate JACK client");
      return 1;
    }
    /* must be activated before connecting JACK ports */
    const char **ports;
    if ((ports = jack_get_ports(
      client, NULL, NULL, JackPortIsPhysical | JackPortIsInput
    ))) {
      /* connect up to two physical playback ports */
      int i = 0;
      while (ports[i] && i < 2) {
        if (jack_connect(
          client, jack_port_name(port[i]), ports[i]
        )) {
          fprintf(stderr, "cannot connect output port\n");
        }
        i++;
      }
      free(ports);
    }
  }
  // main loop
  glfwPollEvents();
  for (int k = 0; !glfwWindowShouldClose(window); ++k)
  {
    // step
    even();
    odd(k);
```

```
    if (k > 0x3FFFF)
    {
      tris();
      glBindFramebuffer(GL_FRAMEBUFFER, fbo);
      glBufferSubData(GL_ARRAY_BUFFER, 0, sizeof(t), t);
      glClear(GL_COLOR_BUFFER_BIT);
      glDrawArrays(GL_TRIANGLES, 0, (H/2 + 2) * (W/2 + 2) * 2 * 2 * 3);
      glBindFramebuffer(GL_DRAW_FRAMEBUFFER, 0);
      glBlitFramebuffer
        ( 0, 0, w, h
        , 0, 0, w, h
        , GL_COLOR_BUFFER_BIT, GL_LINEAR
        );
      if (RECORD)
      {
        float audio[2][SR/FPS];
        audiocb(&audio[0][0], &audio[1][0], SR/FPS);
        float frames[SR/FPS][2];
        for (int i = 0; i < SR/FPS; ++i)
          for (int c = 0; c < 2; ++c)
            frames[i][c] = audio[c][i];
        sf_writef_float(sndfile, &frames[0][0], SR/FPS);
        // download output frame
        glBindFramebuffer(GL_READ_FRAMEBUFFER, 0);
        glReadPixels(0, 0, w, h, GL_RGB, GL_UNSIGNED_BYTE, video);
        // output PPM to stdout (flipped vertically)
        printf("P6\n%d %d\n255\n", w, h);
        fwrite(video, w * h * 3, 1, stdout);
      }
      // redisplay
      glfwSwapBuffers(window);
      glfwPollEvents();
      if (RECORD && k >= 0x3FFFF + (4 * 60 - 6) * FPS)
        break;
      GLuint e;
      while ((e = glGetError())) fprintf(stderr, "GL ERROR %d\n", e);
    }
  }
  // cleanup
  if (RECORD)
  {
    // write 3 seconds silence for credits
    float frames[SR/FPS][2];
    for (int i = 0; i < SR/FPS; ++i)
      for (int c = 0; c < 2; ++c)
        frames[i][c] = 0;
    for (int k = 0; k < 3 * FPS; ++k)
      sf_writef_float(sndfile, &frames[0][0], SR/FPS);
    sf_close(sndfile);
    free(video);
  }
  else
  {
    jack_client_close(client);
  }
  glfwDestroyWindow(window);
  glfwTerminate();
```

```
665     return 0;
}
```

57 meshwalk/meshwalk-2.0-credits.png

58 meshwalk/meshwalk-2.0.tex

```
\documentclass[a4paper]{article}
\usepackage[margin=2cm]{geometry}
\usepackage{lmodern}
\usepackage{listings}
\usepackage[hidelinks]{hyperref}
\usepackage{MnSymbol}

\title{meshwalk-2.0\footnote{\url{https://mathr.co.uk/meshwalk/}}}
\author{Claude Heiland-Allen\footnote{\url{mailto:claude@mathr.co.uk}}}
\date{2018}

\lstset{prebreak=\raisebox{0ex}[0ex][0ex]
        {\ensuremath{\rhookswarrow}}}
\lstset{postbreak=\raisebox{0ex}[0ex][0ex]
        {\ensuremath{\rcurvearrowse\space}}}
\lstset{breaklines=true, breakatwhitespace=true}
\lstset{numbers=left, numberstyle=\scriptsize}

\begin{document}
\maketitle

\lstinputlisting[language=C, breaklines=true, breakatwhitespace=true]{meshwalk
    ↪ -2.0.c}

\end{document}
```

59 meshwalk/meshwalk-2.0-title-and-credits.tex

```
\documentclass{article}
\usepackage{lmodern}

\begin{document}

\centering
\Huge
```

```
\sf
meshwalk-2.0

\Large
mathr.co.uk

2018

\end{document}
```

60 meshwalk/meshwalk-2.0-title.png

61 meshwalk/meshwalk-3.0.c

```
/*
  meshwalk -- audiovisual drone
  Copyright (C) 2018  Claude Heiland-Allen <claude@mathr.co.uk>

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

  You should have received a copy of the GNU General Public License
  along with this program.  If not, see <https://www.gnu.org/licenses/>.
*/
/*
# build (or use `make`)
gcc -std=c99 -Wall -Wextra -pedantic \
    -Ofast -ffast-math -march=native -fopenmp \
    -o meshwalk-3.0 meshwalk-3.0.c \
    -ljack -lsndfile -lGLEW -lGL -lglfw -lm

# realtime mode with JACK audio, press Q to exit
./meshwalk-3.0

# render
(
```

```
    for i in $(seq 180) ; do pngtopnm < meshwalk-3.0-title.png ; done |
    ffmpeg -r 60 -i - -vf fade=in:30:30,fade=out:120:30 -f image2pipe -codec:v ppm -
    ./meshwalk-3.0 -nrt
    for i in $(seq 180) ; do pngtopnm < meshwalk-3.0-credits.png ; done |
    ffmpeg -r 60 -i - -vf fade=in:30:30,fade=out:120:30 -f image2pipe -codec:v ppm -
) |
ffmpeg -framerate 60 -i - -codec:v png -y meshwalk-3.0_v.mkv
ffmpeg -i meshwalk-3.0_v.mkv -i meshwalk-3.0_a.wav \
  -codec:v copy -codec:a copy -movflags +faststart \
  -y meshwalk-3.0.mkv
ffmpeg -i meshwalk-3.0.mkv \
  -pix_fmt yuv420p -profile:v high -level:v 5.1 -tune:v animation \
  -crf:v 20 -b:a 512k -movflags +faststart \
  -y meshwalk-3.0_m.mp4
python2.7 ../../../github.com/google/spatial-media/spatialmedia \
  -i --spatial-audio meshwalk-3.0_m.mp4 meshwalk-3.0.mp4
*/

#include <assert.h>
#include <limits.h>
#include <math.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

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

#include <jack/jack.h>
#include <sndfile.h>

#define pi 3.141592653f

#define SR 48000
#define FPS 60
#define SUB 1
#define OSCILLATORS 4000
#define GAIN (1.0f/8.0f)

#define DEBUG_AMBISONICS 0
#define DEBUG_AMBISONICS_SPIN 0 //(2 * pi / FPS / 12)
#define DEBUG_AMBISONICS_AXES 0

#define DRAW_CLEAR 0
#define DRAW_VORONOI 0
#define DRAW_LINES 0
#define DRAW_POINTS 0
#define DRAW_TRAILS 0

#define R0 (6 / SUB)
#define W (8192 / SUB)
#define H (4096 / SUB)

#define WINW 1024
```

```
85  #define WINH 512

    // note: must be <= 4000 (limit set in fragment shader)
    #define N OSCILLATORS
    #define B 12
90
    // one-pole one-zero low pass filter designed in the Z plane

    typedef float sample;

95  typedef struct {
      sample x, b;
      double y, a;
    } LOP1;

100 static inline LOP1 *lop1_init(LOP1 *s, sample hz)
    {
      const double w = 2 * pi * fminf(fmaxf(fabs(hz / SR), 0), (sample)0.5);
      const double a = (1 - sin(w)) / cos(w);
      const double b = (1 - a) / 2;
105   s->a = a;
      s->b = b;
      s->x = 0;
      s->y = 0;
      return s;
110 }

    static inline sample lop1(LOP1 *s, sample x)
    {
      const double y = s->b * (x + s->x) + s->a * s->y;
115   s->x = x;
      s->y = y;
      return y;
    }


120
    float DISTANCE;
    float Ky;
    float K;
    float sqrtK;
125 const float dt = 1.0f / FPS;
    const float SPEED = 2.0 * sqrtf(1.0f / N);

    GLint u_position;
    GLfloat position[N][4];
130
    GLfloat V[3][3] = { { 1, 0, 0 }, { 0, 1, 0 }, { 0, 0, 1 } };

    float focus[3];

135 const float spin_axes[4][3] =
      { { -1, -1, -1 }
      , {  1,  1, -1 }
      , {  1, -1,  1 }
      , { -1,  1,  1 }
140   };
```

```
int oscillator_targets[OSCILLATORS];

LOP1 output_filters[16];

typedef struct { float f; int i; } index_t;

typedef struct
{
  float x, y, z, dx, dy, dz, ddx, ddy, ddz, speed;
  index_t nbnb[B * B + B];
  int nb[B];
  int count;
} node;

node atom[N];
int which_oscillator_buffer = 0;
index_t oscillators[3][N];

static inline uint32_t hash(uint32_t a)
{
        a = (a+0x7ed55d16u) + (a<<12);
        a = (a^0xc761c23cu) ^ (a>>19);
        a = (a+0x165667b1u) + (a<<5);
        a = (a+0xd3a2646cu) ^ (a<<9);
        a = (a+0xfd7046c5u) + (a<<3);
        a = (a^0xb55a4f09u) ^ (a>>16);
        return a;
}

// GLFW keyboard callback
static void keycb
  ( GLFWwindow *window
  , int key
  , int scancode
  , int action
  , int mods
  )
{
  (void) key;
  (void) scancode;
  (void) mods;
  if (action == GLFW_PRESS)
    glfwSetWindowShouldClose(window, GL_TRUE);
}

volatile int in_audio_cb = 0;
float audio_xyz[N][3];
static void audiocb(float *output[16], int n, int f)
{
  float out[16][n];
  in_audio_cb = 1;
  int source = which_oscillator_buffer;
  for (int o = 0; o < OSCILLATORS && o < N; ++o)
  {
    int i = oscillators[source][o].i;
    oscillators[2][o] = oscillators[source][o];
    float x = atom[i].x;
```

```
      float y = atom[i].y;
      float z = atom[i].z;
      audio_xyz[o][0] = V[0][0] * x + V[0][1] * y + V[0][2] * z;
      audio_xyz[o][1] = V[1][0] * x + V[1][1] * y + V[1][2] * z;
      audio_xyz[o][2] = V[2][0] * x + V[2][1] * y + V[2][2] * z;
    }
    in_audio_cb = 0;
    for (int c = 0; c < 16; ++c)
      for (int t = 0; t < n; ++t)
        out[c][t] = 0;
    for (int o = 0; o < OSCILLATORS && o < N; ++o)
    {
      int i = oscillators[2][o].i;
      float amp = oscillators[2][o].f;
      float x = audio_xyz[o][0];
      float y = audio_xyz[o][1];
      float z = audio_xyz[o][2];
      float factor[16];
      // order 0
      factor[ 0] = 1;
      // order 1
      factor[ 1] = y;
      factor[ 2] = z;
      factor[ 3] = x;
      //  order 2
      factor[ 4] = sqrt(3) * x * y;
      factor[ 5] = sqrt(3) * y * z;
      factor[ 6] = 1/2. * (3 * z * z - 1);
      factor[ 7] = sqrt(3) * x * z;
      factor[ 8] = sqrt(3/4.) * (x * x - y * y);
      // order 3
      factor[ 9] = sqrt(5/8.) * y * (3 * x * x - y * y);
      factor[10] = sqrt(15) * x * y * z;
      factor[11] = sqrt(3/8.) * y * (5 * z * z - 1);
      factor[12] = 1/2. * z * (5 * z * z - 3);
      factor[13] = sqrt(3/8.) * x * (5 * z * z - 1);
      factor[14] = sqrt(15/4.) * z * (x * x - y * y);
      factor[15] = sqrt(5/8.) * x * (x * x - 3 * y * y);
      int period = (720 + ((i % 15)-7)) / ((i % 4) + 1);
      float osc[n];
      for (int t = 0; t < n; ++t)
        osc[t] = hash(hash((f + t) % period) * N + i) / (float) UINT_MAX - 0.5f;
      for (int c = 0; c < 16; ++c)
      {
        factor[c] *= amp;
        for (int t = 0; t < n; ++t)
          out[c][t] += osc[t] * factor[c];
      }
    }
    float level = GAIN;
    for (int c = 0; c < 16; ++c)
      for (int t = 0; t < n; ++t)
        output[c][t] = lop1(&output_filters[c], out[c][t] * level);
  }

  jack_client_t *client;
  jack_port_t *port[16];
```

```
static int processcb(jack_nframes_t nframes, void *arg) {
  (void) arg;
  static jack_nframes_t f = 0;
  jack_default_audio_sample_t *out[16];
  for (int c = 0; c < 16; ++c)
  {
    out[c] = jack_port_get_buffer(port[c], nframes);
    for (jack_nframes_t t = 0; t < nframes; ++t)
      out[c][t] = 0;
  }
  audiocb(out, nframes, f);
  f += nframes;
  return 0;
}

static void debug_program(GLuint program) {
  GLint status = 0;
  glGetProgramiv(program, GL_LINK_STATUS, &status);
  GLint length = 0;
  glGetProgramiv(program, GL_INFO_LOG_LENGTH, &length);
  char *info = 0;
  if (length) {
    info = malloc(length + 1);
    info[0] = 0;
    glGetProgramInfoLog(program, length, 0, info);
    info[length] = 0;
  }
  if ((info && info[0]) || ! status) {
    fprintf(stderr, "program link info:\n%s", info ? info : "(no info log)");
  }
  if (info) {
    free(info);
  }
}

static void debug_shader(GLuint shader, GLenum type, const char *source) {
  GLint status = 0;
  glGetShaderiv(shader, GL_COMPILE_STATUS, &status);
  GLint length = 0;
  glGetShaderiv(shader, GL_INFO_LOG_LENGTH, &length);
  char *info = 0;
  if (length) {
    info = malloc(length + 1);
    info[0] = 0;
    glGetShaderInfoLog(shader, length, 0, info);
    info[length] = 0;
  }
  if ((info && info[0]) || ! status) {
    const char *type_str = "unknown";
    switch (type) {
      case GL_VERTEX_SHADER: type_str = "vertex"; break;
      case GL_FRAGMENT_SHADER: type_str = "fragment"; break;
      case GL_COMPUTE_SHADER: type_str = "compute"; break;
    }
    fprintf
      ( stderr
```

```
      , "%s shader compile info:\n%s\nshader source:\n%s"
      , type_str
315   , info ? info : "(no info log)"
      , source ? source : "(no source)"
      );
  }
  if (info) {
320   free(info);
  }
}

static GLuint vertex_fragment_shader(const char *vert, const char *frag) {
325 GLuint program = glCreateProgram();
  {
    GLuint shader = glCreateShader(GL_VERTEX_SHADER);
    glShaderSource(shader, 1, &vert, 0);
    glCompileShader(shader);
330 debug_shader(shader, GL_VERTEX_SHADER, vert);
    glAttachShader(program, shader);
    glDeleteShader(shader);
  }
  {
335 GLuint shader = glCreateShader(GL_FRAGMENT_SHADER);
    glShaderSource(shader, 1, &frag, 0);
    glCompileShader(shader);
    debug_shader(shader, GL_FRAGMENT_SHADER, frag);
    glAttachShader(program, shader);
340 glDeleteShader(shader);
  }
  glLinkProgram(program);
  debug_program(program);
  return program;
345 }

int cmp_index_f(const void *a, const void *b)
{
  const index_t *p = a;
350 const index_t *q = b;
  const float x = p->f;
  const float y = q->f;
  return (x > y) - (x < y);
}
355
int cmp_index_f_rev(const void *a, const void *b)
{
  return -cmp_index_f(a, b);
}
360
void upload_uniforms(void)
{
  for (int i = 0; i < N; ++i)
  {
365 float x = atom[i].x;
    float y = atom[i].y;
    float z = atom[i].z;
    position[i][0] = V[0][0] * x + V[0][1] * y + V[0][2] * z;
    position[i][1] = V[1][0] * x + V[1][1] * y + V[1][2] * z;
```

```
370       position[i][2] = V[2][0] * x + V[2][1] * y + V[2][2] * z;
          position[i][3] = 1;
        }
        glUniform4fv(u_position, N, &position[0][0]);
      }
375
      unsigned char ppm[4096/SUB][8192/SUB][3];

      void insert_by_distance(int i, int k, float d)
380   {
        int b;
        for (b = 0; b < B && b < atom[i].count; ++b)
        {
          if (d < atom[i].nbnb[b].f) break;
385     }
        if (b < B)
        {
          for (int c = B - 1; c > b; --c)
          {
390         atom[i].nbnb[c] = atom[i].nbnb[c - 1];
          }
          atom[i].nbnb[b].f = d;
          atom[i].nbnb[b].i = k;
          if (atom[i].count < B) atom[i].count += 1;
395     }
      }

      int find_target(float tx, float ty, float tz)
      {
400     float md = -2.0;
        int ix = -1;
        for (int i = 0; i < N; ++i)
        {
          float d = atom[i].x * tx + atom[i].y * ty + atom[i].z * tz;
405       if (d > md)
          {
            md = d;
            ix = i;
          }
410     }
        return ix;
      }

      void init(void)
415   {
        for (int i = 0; i < N; ++i)
        {
          #define U (rand() / (float) RAND_MAX)
          float R;
420         float t = 2 * pi * (i + U) / N;
            float s = 2.0 * (U - 0.5);
          atom[i].x = cosf(t) * cosf(asinf(s));
          atom[i].y = sinf(t) * cosf(asinf(s));
          atom[i].z = s;
425       R = atom[i].x * atom[i].x + atom[i].y * atom[i].y + atom[i].z * atom[i].z;
          R = sqrtf(R);
```

```
    atom[i].x /= R;
    atom[i].y /= R;
    atom[i].z /= R;
    atom[i].dx = 0;
    atom[i].dy = 0;
    atom[i].dz = 0;
    #undef U
  }
  #pragma omp parallel for
  for (int i = 0; i < N; ++i)
  {
    atom[i].count = 0;
    for (int k = 0; k < N; ++k)
    {
      if (i == k) continue;
      float dx = atom[i].x - atom[k].x;
      float dy = atom[i].y - atom[k].y;
      float dz = atom[i].z - atom[k].z;
      float d = dx * dx + dy * dy + dz * dz;
      insert_by_distance(i, k, d);
    }
    assert(atom[i].count == B);
    for (int b = 0; b < B; ++b)
    {
      atom[i].nb[b] = atom[i].nbnb[b].i;
    }
  }
}

static void field_at_atom(float *ddx, float *ddy, float *ddz, int i)
{
  *ddx = 0;
  *ddy = 0;
  *ddz = 0;
  int j = i % 4;
  for (int b = 0; b < B; ++b)
  {
    int k = atom[i].nb[b];
    int l = k % 4;
    float dx = atom[k].x - atom[i].x;
    float dy = atom[k].y - atom[i].y;
    float dz = atom[k].z - atom[i].z;
    float d2 = dx * dx + dy * dy + dz * dz;
    if (d2 == 0) continue;
    float d = sqrtf(d2);
    float F = 0.0;
    F += 1 / (d2 * K);
    if (l != j)
      F -= 1 / (d * sqrtK);
    else
      F *= 2;
    *ddx -= F * dx / d;
    *ddy -= F * dy / d;
    *ddz -= F * dz / d;
  }
  if (isnan(*ddx)) *ddx = 0;
  if (isnan(*ddy)) *ddy = 0;
```

```
  if (isnan(*ddz)) *ddz = 0;
}

void insert_by_index(int i, int k, float d)
{
  for (int b = 0; b < atom[i].count; ++b)
  {
    if (atom[i].nbnb[b].i == k) return;
  }
  if (atom[i].count < B * B + B)
  {
    atom[i].nbnb[atom[i].count].f = d;
    atom[i].nbnb[atom[i].count].i = k;
    atom[i].count += 1;
  }
}

void sort_by_distance(int i)
{
  qsort(&atom[i].nbnb[0], atom[i].count, sizeof(atom[i].nbnb[0]), cmp_index_f);
}

float metric = 0;
void update_nearest_neighbours()
{
  float m = 0.0;
  for (int i = 0; i < N; ++i)
  {
    atom[i].count = 0;
    for (int b = 0; b < B; ++b)
    {
      int j = atom[i].nb[b];
      {
        float dx = atom[i].x - atom[j].x;
        float dy = atom[i].y - atom[j].y;
        float dz = atom[i].z - atom[j].z;
        float d = dx * dx + dy * dy + dz * dz;
        if (i != j)
        insert_by_index(i, j, d);
      }
      for (int c = 0; c < B; ++c)
      {
        int k = atom[j].nb[c];
        if (i == k) continue;
        float dx = atom[i].x - atom[k].x;
        float dy = atom[i].y - atom[k].y;
        float dz = atom[i].z - atom[k].z;
        float d = dx * dx + dy * dy + dz * dz;
        insert_by_index(i, k, d);
      }
    }
    assert(atom[i].count >= B);
    sort_by_distance(i);
    for (int b = 0; b < B; ++b)
      atom[i].nb[b] = atom[i].nbnb[b].i;
    m = fmaxf(m, atom[i].nbnb[B-1].f);
  }
```

```
  metric = m;
}

void update_oscillator_amplitudes(void)
{
  int dest = 1 - which_oscillator_buffer;
#if 0
  int l = 0;
  for (int i = -1; i <= 1; i += 1)
  for (int j = -1; j <= 1; j += 1)
  for (int k = -1; k <= 1; k += 1)
  {
    if (l >= OSCILLATORS) break;
    if (i == 0 && j == 0 && k == 0) continue;
    oscillator_targets[l] = find_target(i, j, k);
    ++l;
  }
  for (int i = 0; i < OSCILLATORS && i < N; ++i)
  {
    oscillators[dest][i].i = oscillator_targets[i];
    oscillators[dest][i].f = powf(atom[oscillator_targets[i]].speed, 8.0);
  }
  qsort(&oscillators[dest][0], OSCILLATORS < N ? OSCILLATORS : N, sizeof(↙
    ↳ oscillators[0][0]), cmp_index_f_rev);
#else
  for (int i = 0; i < N; ++i)
  {
    oscillators[dest][i].i = i;
    oscillators[dest][i].f = powf(atom[i].speed, 8.0);
  }
//  qsort(&oscillators[dest][0], N, sizeof(oscillators[0][0]), cmp_index_f_rev);
#endif
  while (in_audio_cb)
    ; // spin in case audio callback is copying previous data
  which_oscillator_buffer = dest;
//  return atom[oscillators[dest][OSCILLATORS - 1].i].speed;
}

void step(void)
{
  for (int i = 0; i < N; ++i)
    field_at_atom(&atom[i].ddx, &atom[i].ddy, &atom[i].ddz, i);
  for (int i = 0; i < N; ++i)
  {
    atom[i].dx += atom[i].ddx * dt;
    atom[i].dy += atom[i].ddy * dt;
    atom[i].dz += atom[i].ddz * dt;
    // d = normalize(cross(p, cross(p, d)))
    float a1 = atom[i].x;
    float a2 = atom[i].y;
    float a3 = atom[i].z;
    float b1 = atom[i].dx;
    float b2 = atom[i].dy;
    float b3 = atom[i].dz;
    float s1 = a2 * b3 - a3 * b2;
    float s2 = a3 * b1 - a1 * b3;
    float s3 = a1 * b2 - a2 * b1;
```

```
      float t1 = a2 * s3 - a3 * s2;
      float t2 = a3 * s1 - a1 * s3;
      float t3 = a1 * s2 - a2 * s1;
600   float t = -sqrtf(t1 * t1 + t2 * t2 + t3 * t3);
      atom[i].dx = t1 / t;
      atom[i].dy = t2 / t;
      atom[i].dz = t3 / t;
    }
605 for (int i = 0; i < N; ++i)
    {
      float ox = atom[i].x;
      float oy = atom[i].y;
      float oz = atom[i].z;
610   // boost = exp(dot(o, focus));
      float boost = expf(ox * focus[0] + oy * focus[1] + oz * focus[2]);
      float x = atom[i].x + atom[i].dx * dt * SPEED * boost;
      float y = atom[i].y + atom[i].dy * dt * SPEED * boost;
      float z = atom[i].z + atom[i].dz * dt * SPEED * boost;
615   float R = x * x + y * y + z * z;
      R = sqrtf(R);
      atom[i].x = x / R;
      atom[i].y = y / R;
      atom[i].z = z / R;
620 #if 0
      float nx = atom[i].x;
      float ny = atom[i].y;
      float nz = atom[i].z;
      float dx = nx - ox;
625   float dy = ny - oy;
      float dz = nz - oz;
      float s = sqrtf(dx * dx + dy * dy + dz * dz);
      atom[i].speed = tanhf(s / (SPEED * dt));
    #else
630   atom[i].speed = tanhf(boost);
    #endif

    }
    update_nearest_neighbours();
635 update_oscillator_amplitudes();
  }

  void plot(int x, int y, int r, int g, int b)
  {
640 if (0 <= x && x < W && 0 <= y && y < H)
    {
      y = H-1 - y;
      x = (W/2 + W-1 - x) % W;
      ppm[y][x][0] = r;
645   ppm[y][x][1] = g;
      ppm[y][x][2] = b;
    }
  }

650 void lineL(int x0,int y0, int x1, int y1, int r, int g, int b)
  {
    int dx = x1 - x0;
    int dy = y1 - y0;
```

```
  int yi = 1;
  if (dy < 0)
  {
    yi = -1;
    dy = -dy;
  }
  int D = 2*dy - dx;
  int y = y0;
  for (int x = x0; x <= x1; ++x)
  {
    plot(x, y, r, g, b);
    if (D > 0)
    {
       y = y + yi;
       D = D - 2*dx;
    }
    D = D + 2*dy;
  }
}

void lineH(int x0, int y0, int x1, int y1, int r, int g, int b)
{
  int dx = x1 - x0;
  int dy = y1 - y0;
  int xi = 1;
  if (dx < 0)
  {
    xi = -1;
    dx = -dx;
  }
  int D = 2*dx - dy;
  int x = x0;

  for (int y = y0; y <= y1; ++y)
  {
    plot(x, y, r, g, b);
    if (D > 0)
    {
       x = x + xi;
       D = D - 2*dy;
    }
    D = D + 2*dx;
  }
}

void line(int x0, int y0, int x1, int y1, int r, int g, int b)
{
  if (abs(y1 - y0) < abs(x1 - x0))
    if (x0 > x1)
      lineL(x1, y1, x0, y0, r, g, b);
    else
      lineL(x0, y0, x1, y1, r, g, b);
  else
    if (y0 > y1)
      lineH(x1, y1, x0, y0, r, g, b);
    else
      lineH(x0, y0, x1, y1, r, g, b);
```

```
}

void draw_geodesic(float x0, float y0, float z0, float x1, float y1, float z1, ↙
    ↳ int red, int grn, int blu, int depth)
{
  if (depth == 0)
  {
    int i0 = atan2f(y0, x0) / (2 * pi) * W;
    if (i0 < 0) i0 += W;
    int j0 = acosf(z0) / pi * H;
    int i1 = atan2f(y1, x1) / (2 * pi) * W;
    if (i1 < 0) i1 += W;
    int j1 = acosf(z1) / pi * H;
    if (i0 < W * 0.25 && W * 0.75 < i1)
    {
      line(i0, j0, i1 - W, j1, red, grn, blu);
      line(i0 + W, j0, i1, j1, red, grn, blu);
    }
    else
    if (i1 < W * 0.25 && W * 0.75 < i0)
    {
      line(i0 - W, j0, i1, j1, red, grn, blu);
      line(i0, j0, i1 + W, j1, red, grn, blu);
    }
    else
      line(i0, j0, i1, j1, red, grn, blu);
  }
  else
  {
    float x2 = x0 + x1;
    float y2 = y0 + y1;
    float z2 = z0 + z1;
    float r2 = x2 * x2 + y2 * y2 + z2 * z2;
    float r = sqrtf(r2);
    x2 /= r;
    y2 /= r;
    z2 /= r;
    draw_geodesic(x0, y0, z0, x2, y2, z2, red, grn, blu, depth - 1);
    draw_geodesic(x2, y2, z2, x1, y1, z1, red, grn, blu, depth - 1);
  }
}

void draw_lines_for(int i, int red, int grn, int blu)
{
  for (int b = 0; b < B; ++b)
  {
    int j = atom[i].nb[b];
    draw_geodesic
      ( atom[i].x, atom[i].y, atom[i].z
      , atom[j].x, atom[j].y, atom[j].z
      , red, grn, blu
      , 6
      );
  }
}

void draw_point(int i, int red, int grn, int blu)
```

```
{
  float ax = atom[i].x;
  float ay = atom[i].y;
  float az = atom[i].z;
  float x0 = atan2f(ay, ax) / (2 * pi);
  x0 -= floor(x0);
  x0 *= W;
  float y0 = acosf(az) / pi * H;
  for (int dy = -R0; dy <= R0; ++dy)
  {
    int y = y0 + dy;
    float w = 1 - fabsf((y0 + dy + 0.5f) / H * 2 - 1);
    if (y < 0) continue;
    if (y >= H) continue;
    for (int dx = -R0 / w; dx <= R0 / w; ++dx)
    {
      int x = x0 + dx;
      while (x < 0) x += W;
      while (x >= W) x -= W;
      if (dx * dx * w * w + dy * dy > R0 * R0) continue;
      plot(x, y, red, grn, blu);
    }
  }
}

void draw_lines(int target)
{
  // everything dark blue
  for (int i = 0; i < N; ++i)
    draw_lines_for(i, 0x00, 0x00, 0x80);
  // neighbours bright green
  for (int b = 0; b < B; ++b)
    draw_lines_for(atom[target].nb[b], 0x00, 0xFF, 0x00);
  // target yellow
  draw_lines_for(target, 0xFF, 0xFF, 0x00);
}


void draw_points(int target)
{
  // everything dark blue
  for (int i = 0; i < N; ++i)
    draw_point(i, 0x00, 0x00, 0x80);
  // neighbours of neighbours bright green
  for (int b = 0; b < B; ++b)
  {
    int j = atom[target].nb[b];
    for (int c = 0; c < B; ++c)
      draw_point(atom[j].nb[c], 0x00, 0xFF, 0x00);
  }
  // neighbours yellow
  for (int b = 0; b < B; ++b)
    draw_point(atom[target].nb[b], 0xFF, 0xFF, 0x00);
  // target red
  draw_point(target, 0xFF, 0x00, 0x00);
}
```

```
void draw_trails(void )
{
  const int colour[4][3] = { { 0, 0, 0 }, { 0, 0x80, 0xFF }, { 0xFF, 51, 0 }, { ↙
      ↳ 0xFF, 0xFF, 0xFF } };
  for (int i = 0; i < N; ++i)
  {
    if (i % 4 != 3) continue;
    draw_point(i, colour[i%4][0], colour[i%4][1], colour[i%4][2]);
  }
}

void output(FILE *vidfile)
{
  fprintf(vidfile, "P6\n%d %d\n255\n", W, H);
  fwrite(&ppm[0][0][0], sizeof(ppm[0][0][0]) * 3 * W * H, 1, vidfile);
  fflush(vidfile);
}

// entry point
extern int main(int argc, char **argv)
{
  (void) argv;
  int RECORD = argc > 1;

  for (int o = 0; o < 16; ++o)
    lop1_init(&output_filters[o], 1000);

  glfwInit();
  glfwWindowHint(GLFW_CLIENT_API, GLFW_OPENGL_API);
  glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
  glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
  glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
  glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
  glfwWindowHint(GLFW_RESIZABLE, GL_FALSE);
  glfwWindowHint(GLFW_DECORATED, GL_TRUE);
  GLFWwindow *window = glfwCreateWindow(WINW, WINH, "meshwalk", 0, 0);
  glfwMakeContextCurrent(window);
  glfwSetKeyCallback(window, keycb);
  glewExperimental = GL_TRUE;
  glewInit();
  glGetError(); // discard common error from glew
  // set up vertex array object
  glClearColor(0, 0, 0, 1);
  GLuint vao;
  glGenVertexArrays(1, &vao);
  glBindVertexArray(vao);
  // set up framebuffer
  GLuint fbo[2];
  glGenFramebuffers(2, &fbo[0]);
  GLuint tex[2];
  glGenTextures(2, &tex[0]);
  glBindFramebuffer(GL_FRAMEBUFFER, fbo[0]);
  glActiveTexture(GL_TEXTURE1);
  glBindTexture(GL_TEXTURE_2D_MULTISAMPLE, tex[0]);
  glTexImage2DMultisample(GL_TEXTURE_2D_MULTISAMPLE, 8, GL_RGB, 8192/SUB, 4096/↙
      ↳ SUB, GL_FALSE);
```

```
  glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, ↙
      ↳ GL_TEXTURE_2D_MULTISAMPLE, tex[0], 0);
880 GLenum status = glCheckFramebufferStatus(GL_FRAMEBUFFER);
  if (status != GL_FRAMEBUFFER_COMPLETE)
    fprintf(stderr, "GL FRAMEBUFFER STATUS %d\n", status);
  glBindFramebuffer(GL_FRAMEBUFFER, fbo[1]);
  glActiveTexture(GL_TEXTURE0);
885 glBindTexture(GL_TEXTURE_2D, tex[1]);
  glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, 8192/SUB, 4096/SUB, 0, GL_RGB, ↙
      ↳ GL_UNSIGNED_BYTE, 0);
  glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, ↙
      ↳ tex[1], 0);
  status = glCheckFramebufferStatus(GL_FRAMEBUFFER);
  if (status != GL_FRAMEBUFFER_COMPLETE)
890   fprintf(stderr, "GL FRAMEBUFFER STATUS %d\n", status);
  // set up shader
  const char *s_vertex =
    "#version 330 core\n"
    "out vec2 tc;\n"
895   "const float pi = 3.141592653;\n"
    "void main()\n"
    "{\n"
    "  if (gl_VertexID == 0) { tc = vec2( pi, -pi/2.0); gl_Position = vec4(-1.0,↙
        ↳  -1.0, 0.0, 1.0); } else\n"
    "  if (gl_VertexID == 1) { tc = vec2(-pi, -pi/2.0); gl_Position = vec4( 1.0,↙
        ↳  -1.0, 0.0, 1.0); } else\n"
900   "  if (gl_VertexID == 2) { tc = vec2( pi,  pi/2.0); gl_Position = vec4(-1.0,↙
        ↳   1.0, 0.0, 1.0); } else\n"
    "  if (gl_VertexID == 3) { tc = vec2(-pi,  pi/2.0); gl_Position = vec4( 1.0,↙
        ↳   1.0, 0.0, 1.0); } else\n"
    "  { tc = vec2(0.0); gl_Position = vec4(0.0); }\n"
    "}\n"
    ;
905 const char *s_fragment =
    "#version 330 core\n"
    "in vec2 tc;\n"
    "uniform int N;\n"
    "uniform vec4 position[4000];\n"
910   "layout (location = 0) out vec4 colour;\n"
    "void main()\n"
    "{\n"
    "  vec3 me = vec3(cos(tc.x) * cos(tc.y), sin(tc.x) * cos(tc.y), sin(tc.y));\↙
        ↳ n"
    "  float delta = length(vec2(length(dFdx(me)), length(dFdy(me))));\n"
915   "  float maxdot = -2.0;\n"
    "  int ix = 0;\n"
    "  float maxdot2 = -2.0;\n"
    "  int ix2 = 0;\n"
    "  for (int i = 0; i < N && i < 4000; ++i)\n"
920   "  {\n"
    "    float d = dot(position[i].xyz, me);\n"
    "    if (d > maxdot) { maxdot2 = maxdot; ix2 = ix; maxdot = d; ix = i; }\n"
    "    else\n"
    "    if (d > maxdot2) { maxdot2 = d; ix2 = i; }\n"
925   "  }\n"
    "  vec3 blue = vec3(0.0, 0.5, 1.0);\n"
    "  vec3 white = vec3(1.0);\n"
```

```
    "  vec3 red = vec3(1.0, 0.2, 0.0);\n"
    "  vec3 black = vec3(0.0);\n"
    "  vec3 mine = mix(vec3(0.5), vec3[4](black, blue, red, white)[ix & 3], position[ix].w);\n"
    "  vec3 them = mix(vec3(0.5), vec3[4](black, blue, red, white)[ix2 & 3], position[ix2].w);\n"
    "  float lmine = length(position[ix].xyz - me);\n"
    "  float lthem = length(position[ix2].xyz - me);\n"
    "  float lme = lmine - lthem;\n"
    "  colour = vec4(mix(mine, them, smoothstep(-delta, delta, lme)), 1.0);\n"
    "}\n"
;
GLuint display = vertex_fragment_shader(s_vertex, s_fragment);
glUseProgram(display);
u_position = glGetUniformLocation(display, "position");
GLint u_N = glGetUniformLocation(display, "N");
glUniform1i(u_N, N);
SF_INFO info = { 0, SR, 16, SF_FORMAT_WAV | SF_FORMAT_FLOAT, 0, 0 };
SNDFILE *sndfile = 0;
FILE *vidfile = stdout;
if (RECORD)
{
  sndfile = sf_open("meshwalk-3.0_a.wav", SFM_WRITE, &info);
}
else
{
  if (!(client = jack_client_open("meshwalk", JackNoStartServer, 0))) {
    fprintf (stderr, "jack server not running?\n");
    return 1;
  }
  jack_set_process_callback(client, processcb, 0);
  /* create ports */
  for (int i = 0; i < 16; ++i)
  {
    char portname[100];
    snprintf(portname, 100, "output_%d", i + 1);
    port[i] = jack_port_register(client, portname, JACK_DEFAULT_AUDIO_TYPE, JackPortIsOutput, 0);
  }
  /* activate audio */
  if (jack_activate(client)) {
    fprintf (stderr, "cannot activate JACK client");
    return 1;
  }
}

float frames[SR/FPS][16];
// write titles
if (RECORD)
{
  for (int i = 0; i < SR/FPS; ++i)
    for (int c = 0; c < 16; ++c)
      frames[i][c] = 0;
  for (int f = 0; f < FPS * 3; ++f)
    sf_writef_float(sndfile, &frames[0][0], SR/FPS);
}
init();
```

```
  int dirty = 1;
  double last = glfwGetTime();
  int last_count = 0;
985 memset(&ppm[0][0][0], 0x80, 8192/SUB * 4096/SUB * 3);
  float focus_target[3] = { 0, 0, 0 };
  LOP1 focus_filter[3];
  float fhz = 4.0f * SR / FPS;
  lop1_init(&focus_filter[0], fhz);
990 lop1_init(&focus_filter[1], fhz);
  lop1_init(&focus_filter[2], fhz);
  // 1/2 = g * (1 + (1-g) + (1-g)^2 + ... + (1-g)^(H-1))
  //     = g * (1 - (1 - g)^(H-1)) / (1 - (1 - g))
  //     = 1 - (1 - g)^(H-1)
995 //   g = 1 - (1/2)^(1/(H-1))
  double geiger = 1.0 - pow(0.5, 4.0 / FPS);
  for (int f = 0; ! glfwWindowShouldClose(window); ++f)
  {
    DISTANCE = 1;
1000 Ky = DISTANCE * 2.0f * sqrtf(4.0f / N);
    K = 1.0 / (Ky * Ky);
    sqrtK = sqrtf(K);
    if (rand() / (double) RAND_MAX < geiger)
    {
1005  float R;
      do
      {
        R = 0;
        for (int c = 0; c < 3; ++c)
1010    {
          focus_target[c] = 2.0f * (rand() / (float) RAND_MAX - 0.5f);
          R += focus_target[c] * focus_target[c];
        }
      } while (R > 1);
1015  R = sqrtf(R);
      float G = 0.5 * (1 - cos(2 * pi * (f - (FPS * (1 * 60 + 0))) / (FPS * (4 *↙
          ↳ 60 - 6)))) / R;
      focus_target[0] *= G;
      focus_target[1] *= G;
      focus_target[2] *= G;
1020  }
    focus[0] = lop1(&focus_filter[0], focus_target[0]);
    focus[1] = lop1(&focus_filter[1], focus_target[1]);
    focus[2] = lop1(&focus_filter[2], focus_target[2]);
    if (DEBUG_AMBISONICS_AXES)
1025  {
      int k = (f / FPS) % 6;
      const float targets[6][3] =
        { { 1, 0, 0 }, { -1, 0, 0 } // front, back
        , { 0, 1, 0 }, { 0, -1, 0 } // left, right
1030    , { 0, 0, 1 }, { 0, 0, -1 } // top, bottom
        };
      focus[0] = targets[k][0];
      focus[1] = targets[k][1];
      focus[2] = targets[k][2];
1035  }
    if (DEBUG_AMBISONICS_SPIN != 0)
    {
```

```
        float t = DEBUG_AMBISONICS_SPIN;
        float c = cos(t);
        float s = sin(t);
        float M[3][3] = { { c, s, 0 }, { -s, c, 0}, { 0, 0, 1 } };
        float nV[3][3];
        for (int i = 0; i < 3; ++i)
        for (int j = 0; j < 3; ++j)
        {
          nV[i][j] = 0;
          for (int k = 0; k < 3; ++k)
            nV[i][j] += M[i][k] * V[k][j];
        }
        for (int i = 0; i < 3; ++i)
        for (int j = 0; j < 3; ++j)
          V[i][j] = nV[i][j];
      }
      double now = glfwGetTime();
      if (now - last > 1)
      {
        fprintf(stderr, "\n%f fps\n%f max\n", (f - last_count) / (now - last), sqrtf(metric) / Ky);
        last_count = f;
        last = now;
      }
      if (RECORD && f >= FPS * (60 * 5 - 6))
        break;

      step();

      if ((RECORD && f >= FPS * 60 * 1) || ! RECORD)
      {
        upload_uniforms();
        glBindFramebuffer(GL_FRAMEBUFFER, fbo[1]);
        glClear(GL_COLOR_BUFFER_BIT);
        glViewport(0, 0, 8192/SUB, 4096/SUB);
        glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);
#if 0
        glBindFramebuffer(GL_DRAW_FRAMEBUFFER, fbo[1]);
        glBlitFramebuffer
          ( 0, 0, 8192/SUB, 4096/SUB
          , 0, 0, 8192/SUB, 4096/SUB
          , GL_COLOR_BUFFER_BIT, GL_LINEAR
          );
        // download output frame
        glBindFramebuffer(GL_READ_FRAMEBUFFER, fbo[1]);
#endif
        if (RECORD && f >= FPS * 60 * 1 && DRAW_VORONOI)
        {
          dirty = 0;
          glReadPixels(0, 0, 8192/SUB, 4096/SUB, GL_RGB, GL_UNSIGNED_BYTE, &ppm[0][0][0]);
        }
        else if (dirty && DRAW_CLEAR)
        {
          dirty = 0;
          memset(&ppm[0][0][0], 0, 8192/SUB * 4096/SUB * 3);
        }
```

```
        int target = find_target(1, 0, 0);
        if (DRAW_LINES)
        {
          dirty = 1;
          draw_lines(target);
        }
        if (DRAW_POINTS)
        {
          dirty = 1;
          draw_points(target);
        }
        if (DRAW_TRAILS)
        {
          dirty = 1;
          draw_trails();
        }
        // preview
        glBindFramebuffer(GL_DRAW_FRAMEBUFFER, 0);
        glClear(GL_COLOR_BUFFER_BIT);
        glBlitFramebuffer
          ( 0, 0, 8192/SUB, 4096/SUB
          , 0, (WINH - WINW/2) / 2, WINW, (WINH + WINW/2) / 2
          , GL_COLOR_BUFFER_BIT, GL_LINEAR
          );
        // redisplay
        glfwSwapBuffers(window);
        glfwPollEvents();
        GLuint e;
        while ((e = glGetError())) fprintf(stderr, "GL ERROR %d\n", e);

        // output PPM to stdout (flipped vertically)
        if (RECORD && f >= FPS * 60 * 1)
        {
          fprintf(vidfile, "P6\n%d %d\n255\n", 8192/SUB, 4096/SUB);
          fwrite(&ppm[0][0][0], 8192/SUB * 4096/SUB * 3, 1, vidfile);
          if (f == FPS * 60 * 1)
          {
            FILE *imgfile = popen("pnmtopng -force -interlace -compression 9 > ↙
                ↳ meshwalk-3.0.png", "w");
            fprintf(imgfile, "P6\n%d %d\n255\n", 8192/SUB, 4096/SUB);
            fwrite(&ppm[0][0][0], 8192/SUB * 4096/SUB * 3, 1, imgfile);
            pclose(imgfile);
          }
          float audio[16][SR/FPS];
          float *audiop[16];
          for (int c = 0; c < 16; ++c)
            audiop[c] = &audio[c][0];
          audiocb(audiop, SR/FPS, f * SR/FPS);
          for (int i = 0; i < SR/FPS; ++i)
            for (int c = 0; c < 16; ++c)
              frames[i][c] = audio[c][i];
          sf_writef_float(sndfile, &frames[0][0], SR/FPS);
        }
      }
    }
    if (RECORD)
    {
```

```
    // write credits
    for (int i = 0; i < SR/FPS; ++i)
      for (int c = 0; c < 16; ++c)
        frames[i][c] = 0;
    for (int f = 0; f < FPS * 3; ++f)
      sf_writef_float(sndfile, &frames[0][0], SR/FPS);
    sf_close(sndfile);
  }
  glfwDestroyWindow(window);
  glfwTerminate();
  return 0;
}
```

62 meshwalk/meshwalk-3.0-credits.png

63 meshwalk/meshwalk-3.0.sh

```
#!/bin/bash
set -e
(
  for i in $(seq 180)
  do
    pngtopnm < meshwalk-3.0-title.png
  done |
  ffmpeg -r 60 -i - -vf fade=in:30:30,fade=out:120:30 -f image2pipe -codec:v ppm -
  ./meshwalk-3.0 -nrt
  for i in $(seq 180)
  do
    pngtopnm < meshwalk-3.0-credits.png
  done |
  ffmpeg -r 60 -i - -vf fade=in:30:30,fade=out:120:30 -f image2pipe -codec:v ppm -
) |
ffmpeg -framerate 60 -i - -codec:v png -y meshwalk-3.0_v.mkv
ffmpeg -i meshwalk-3.0_v.mkv -i meshwalk-3.0_a.wav \
  -codec:v copy -codec:a copy \
  -y meshwalk-3.0.mkv
ffmpeg -i meshwalk-3.0.mkv -af pan="4| c0=c0 | c1=c1 | c2=c2 | c3=c3" \
  -pix_fmt yuv420p -profile:v high -level:v 5.1 -tune:v animation \
  -crf:v 20 -b:a 512k -movflags +faststart \
  -y meshwalk-3.0_m.mp4
python2.7 ../../../github.com/google/spatial-media/spatialmedia \
  -i --spatial-audio meshwalk-3.0_m.mp4 meshwalk-3.0.mp4
```

64 meshwalk/meshwalk-3.0-title.png

65 meshwalk/meshwalk-3.0-titles.tex

```
\documentclass{article}
\usepackage[paperwidth=8.192cm,paperheight=4.096cm,margin=0.1cm]{geometry}
\usepackage{lmodern}
\begin{document}
\thispagestyle{empty}
\hspace{0pt}
\vfill
\centering \sf
{\Huge meshwalk-3.0}

{\Large mathr.co.uk}

{\Large 2018}
\vfill
\hspace{0pt}
\end{document}
```

66 meshwalk/meshwalk-3.1.c

```
/*
  meshwalk -- audiovisual drone
  Copyright (C) 2018  Claude Heiland-Allen <claude@mathr.co.uk>

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

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

```
# realtime mode with JACK audio, press Q to exit
./meshwalk-3.1

# render
./meshwalk-3.1.sh
*/

#include <assert.h>
#include <limits.h>
#include <math.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

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

#include <jack/jack.h>
#include <sndfile.h>

#define pi 3.141592653f

#define SR 48000
#define FPS 60
#define SUB 1
#define OSCILLATORS 4000
#define GAIN 0.1f

#define DEBUG_AMBISONICS 0
#define DEBUG_AMBISONICS_SPIN 0 //(2 * pi / FPS / 12)
#define DEBUG_AMBISONICS_AXES 0

#define DRAW_CLEAR 0
#define DRAW_VORONOI 1
#define DRAW_LINES 0
#define DRAW_POINTS 0
#define DRAW_TRAILS 0

#define RECORD_AUDIO 1
#define RECORD_VIDEO 1

#define R0 (6 / SUB)
#define W (8192 / SUB)
#define H (4096 / SUB)

#define WINW 1024
#define WINH 512

// note: must be <= 4000 (limit set in fragment shader)
#define N OSCILLATORS
#define B 12

// one-pole one-zero low pass filter designed in the Z plane

typedef float sample;
```

```
typedef struct {
  sample x, b;
  double y, a;
} LOP1;

static inline LOP1 *lop1_init(LOP1 *s, sample hz)
{
  const double w = 2 * pi * fminf(fmaxf(fabs(hz / SR), 0), (sample)0.5);
  const double a = (1 - sin(w)) / cos(w);
  const double b = (1 - a) / 2;
  s->a = a;
  s->b = b;
  s->x = 0;
  s->y = 0;
  return s;
}

static inline sample lop1(LOP1 *s, sample x)
{
  const double y = s->b * (x + s->x) + s->a * s->y;
  s->x = x;
  s->y = y;
  return y;
}


float DISTANCE;
float Ky;
float K;
float sqrtK;
const float dt = 1.0f / FPS;
const float SPEED = 2.0 * sqrtf(1.0f / N);

GLint u_position;
GLfloat position[N][4];

GLfloat V[3][3] = { { 1, 0, 0 }, { 0, 1, 0 }, { 0, 0, 1 } };

float focus[3];

const float spin_axes[4][3] =
  { { -1, -1, -1 }
  , {  1,  1, -1 }
  , {  1, -1,  1 }
  , { -1,  1,  1 }
  };

int oscillator_targets[OSCILLATORS];

LOP1 output_filters[16];

typedef struct { float f; int i; } index_t;

typedef struct
{
  float x, y, z, dx, dy, dz, ddx, ddy, ddz, speed;
  index_t nbnb[B * B + B];
```

```
  int nb[B];
  int count;
} node;

node atom[N];
int which_oscillator_buffer = 0;
index_t oscillators[3][N];

static inline uint32_t hash(uint32_t a)
{
        a = (a+0x7ed55d16u) + (a<<12);
        a = (a^0xc761c23cu) ^ (a>>19);
        a = (a+0x165667b1u) + (a<<5);
        a = (a+0xd3a2646cu) ^ (a<<9);
        a = (a+0xfd7046c5u) + (a<<3);
        a = (a^0xb55a4f09u) ^ (a>>16);
        return a;
}

// GLFW keyboard callback
static void keycb
  ( GLFWwindow *window
  , int key
  , int scancode
  , int action
  , int mods
  )
{
  (void) key;
  (void) scancode;
  (void) mods;
  if (action == GLFW_PRESS)
    glfwSetWindowShouldClose(window, GL_TRUE);
}

volatile int in_audio_cb = 0;
float audio_xyz[N][3];
static void audiocb(float *output[16], int n, int f)
{
  float out[16][n];
  in_audio_cb = 1;
  int source = which_oscillator_buffer;
  for (int o = 0; o < OSCILLATORS && o < N; ++o)
  {
    int i = oscillators[source][o].i;
    oscillators[2][o] = oscillators[source][o];
    float x = atom[i].x;
    float y = atom[i].y;
    float z = atom[i].z;
    audio_xyz[o][0] = V[0][0] * x + V[0][1] * y + V[0][2] * z;
    audio_xyz[o][1] = V[1][0] * x + V[1][1] * y + V[1][2] * z;
    audio_xyz[o][2] = V[2][0] * x + V[2][1] * y + V[2][2] * z;
  }
  in_audio_cb = 0;
  for (int c = 0; c < 16; ++c)
    for (int t = 0; t < n; ++t)
      out[c][t] = 0;
```

```
  for (int o = 0; o < OSCILLATORS && o < N; ++o)
  {
    int i = oscillators[2][o].i;
    float amp = oscillators[2][o].f;
    float x = audio_xyz[o][0];
    float y = audio_xyz[o][1];
    float z = audio_xyz[o][2];
    float factor[16];
    // order 0
    factor[ 0] = 1;
    // order 1
    factor[ 1] = y;
    factor[ 2] = z;
    factor[ 3] = x;
    //  order 2
    factor[ 4] = sqrt(3) * x * y;
    factor[ 5] = sqrt(3) * y * z;
    factor[ 6] = 1/2. * (3 * z * z - 1);
    factor[ 7] = sqrt(3) * x * z;
    factor[ 8] = sqrt(3/4.) * (x * x - y * y);
    // order 3
    factor[ 9] = sqrt(5/8.) * y * (3 * x * x - y * y);
    factor[10] = sqrt(15) * x * y * z;
    factor[11] = sqrt(3/8.) * y * (5 * z * z - 1);
    factor[12] = 1/2. * z * (5 * z * z - 3);
    factor[13] = sqrt(3/8.) * x * (5 * z * z - 1);
    factor[14] = sqrt(15/4.) * z * (x * x - y * y);
    factor[15] = sqrt(5/8.) * x * (x * x - 3 * y * y);
    int period = (720 + ((i % 15)-7)) / ((i % 4) + 1);
    float osc[n];
    for (int t = 0; t < n; ++t)
      osc[t] = hash(hash((f + t) % period) * N + i) / (float) UINT_MAX - 0.5f;
    for (int c = 0; c < 16; ++c)
    {
      factor[c] *= amp;
      for (int t = 0; t < n; ++t)
        out[c][t] += osc[t] * factor[c];
    }
  }
  float level = GAIN;
  for (int c = 0; c < 16; ++c)
    for (int t = 0; t < n; ++t)
      output[c][t] = lop1(&output_filters[c], out[c][t] * level);
}

jack_client_t *client;
jack_port_t *port[16];
static int processcb(jack_nframes_t nframes, void *arg) {
  (void) arg;
  static jack_nframes_t f = 0;
  jack_default_audio_sample_t *out[16];
  for (int c = 0; c < 16; ++c)
  {
    out[c] = jack_port_get_buffer(port[c], nframes);
    for (jack_nframes_t t = 0; t < nframes; ++t)
      out[c][t] = 0;
  }
```

```
250   audiocb(out, nframes, f);
      f += nframes;
      return 0;
    }

255
    static void debug_program(GLuint program) {
      GLint status = 0;
      glGetProgramiv(program, GL_LINK_STATUS, &status);
      GLint length = 0;
260   glGetProgramiv(program, GL_INFO_LOG_LENGTH, &length);
      char *info = 0;
      if (length) {
        info = malloc(length + 1);
        info[0] = 0;
265     glGetProgramInfoLog(program, length, 0, info);
        info[length] = 0;
      }
      if ((info && info[0]) || ! status) {
        fprintf(stderr, "program link info:\n%s", info ? info : "(no info log)");
270   }
      if (info) {
        free(info);
      }
    }
275
    static void debug_shader(GLuint shader, GLenum type, const char *source) {
      GLint status = 0;
      glGetShaderiv(shader, GL_COMPILE_STATUS, &status);
      GLint length = 0;
280   glGetShaderiv(shader, GL_INFO_LOG_LENGTH, &length);
      char *info = 0;
      if (length) {
        info = malloc(length + 1);
        info[0] = 0;
285     glGetShaderInfoLog(shader, length, 0, info);
        info[length] = 0;
      }
      if ((info && info[0]) || ! status) {
        const char *type_str = "unknown";
290     switch (type) {
          case GL_VERTEX_SHADER: type_str = "vertex"; break;
          case GL_FRAGMENT_SHADER: type_str = "fragment"; break;
          case GL_COMPUTE_SHADER: type_str = "compute"; break;
        }
295     fprintf
          ( stderr
          , "%s shader compile info:\n%s\nshader source:\n%s"
          , type_str
          , info ? info : "(no info log)"
300       , source ? source : "(no source)"
          );
      }
      if (info) {
        free(info);
305   }
    }
```

```
static GLuint vertex_fragment_shader(const char *vert, const char *frag) {
  GLuint program = glCreateProgram();
  {
    GLuint shader = glCreateShader(GL_VERTEX_SHADER);
    glShaderSource(shader, 1, &vert, 0);
    glCompileShader(shader);
    debug_shader(shader, GL_VERTEX_SHADER, vert);
    glAttachShader(program, shader);
    glDeleteShader(shader);
  }
  {
    GLuint shader = glCreateShader(GL_FRAGMENT_SHADER);
    glShaderSource(shader, 1, &frag, 0);
    glCompileShader(shader);
    debug_shader(shader, GL_FRAGMENT_SHADER, frag);
    glAttachShader(program, shader);
    glDeleteShader(shader);
  }
  glLinkProgram(program);
  debug_program(program);
  return program;
}

int cmp_index_f(const void *a, const void *b)
{
  const index_t *p = a;
  const index_t *q = b;
  const float x = p->f;
  const float y = q->f;
  return (x > y) - (x < y);
}

int cmp_index_f_rev(const void *a, const void *b)
{
  return -cmp_index_f(a, b);
}

void upload_uniforms(void)
{
  for (int i = 0; i < N; ++i)
  {
    float x = atom[i].x;
    float y = atom[i].y;
    float z = atom[i].z;
    position[i][0] = V[0][0] * x + V[0][1] * y + V[0][2] * z;
    position[i][1] = V[1][0] * x + V[1][1] * y + V[1][2] * z;
    position[i][2] = V[2][0] * x + V[2][1] * y + V[2][2] * z;
    position[i][3] = 1;
  }
  glUniform4fv(u_position, N, &position[0][0]);
}


unsigned char ppm[4096/SUB][8192/SUB][3];

void insert_by_distance(int i, int k, float d)
```

```
{
  int b;
  for (b = 0; b < B && b < atom[i].count; ++b)
  {
    if (d < atom[i].nbnb[b].f) break;
  }
  if (b < B)
  {
    for (int c = B - 1; c > b; --c)
    {
      atom[i].nbnb[c] = atom[i].nbnb[c - 1];
    }
    atom[i].nbnb[b].f = d;
    atom[i].nbnb[b].i = k;
    if (atom[i].count < B) atom[i].count += 1;
  }
}

int find_target(float tx, float ty, float tz)
{
  float md = -2.0;
  int ix = -1;
  for (int i = 0; i < N; ++i)
  {
    float d = atom[i].x * tx + atom[i].y * ty + atom[i].z * tz;
    if (d > md)
    {
      md = d;
      ix = i;
    }
  }
  return ix;
}

void init(void)
{
  for (int i = 0; i < N; ++i)
  {
    #define U (rand() / (float) RAND_MAX)
    float R;
      float t = 2 * pi * (i + U) / N;
      float s = 2.0 * (U - 0.5);
    atom[i].x = cosf(t) * cosf(asinf(s));
    atom[i].y = sinf(t) * cosf(asinf(s));
    atom[i].z = s;
    R = atom[i].x * atom[i].x + atom[i].y * atom[i].y + atom[i].z * atom[i].z;
    R = sqrtf(R);
    atom[i].x /= R;
    atom[i].y /= R;
    atom[i].z /= R;
    atom[i].dx = 0;
    atom[i].dy = 0;
    atom[i].dz = 0;
    #undef U
  }
  #pragma omp parallel for
  for (int i = 0; i < N; ++i)
```

```
  {
    atom[i].count = 0;
    for (int k = 0; k < N; ++k)
    {
      if (i == k) continue;
      float dx = atom[i].x - atom[k].x;
      float dy = atom[i].y - atom[k].y;
      float dz = atom[i].z - atom[k].z;
      float d = dx * dx + dy * dy + dz * dz;
      insert_by_distance(i, k, d);
    }
    assert(atom[i].count == B);
    for (int b = 0; b < B; ++b)
    {
      atom[i].nb[b] = atom[i].nbnb[b].i;
    }
  }
}

static void field_at_atom(float *ddx, float *ddy, float *ddz, int i)
{
  *ddx = 0;
  *ddy = 0;
  *ddz = 0;
  int j = i % 4;
  for (int b = 0; b < B; ++b)
  {
    int k = atom[i].nb[b];
    int l = k % 4;
    float dx = atom[k].x - atom[i].x;
    float dy = atom[k].y - atom[i].y;
    float dz = atom[k].z - atom[i].z;
    float d2 = dx * dx + dy * dy + dz * dz;
    if (d2 == 0) continue;
    float d = sqrtf(d2);
    float F = 0.0;
    F += 1 / (d2 * K);
    if (l != j)
      F -= 1 / (d * sqrtK);
    else
      F *= 2;
    *ddx -= F * dx / d;
    *ddy -= F * dy / d;
    *ddz -= F * dz / d;
  }
  if (isnan(*ddx)) *ddx = 0;
  if (isnan(*ddy)) *ddy = 0;
  if (isnan(*ddz)) *ddz = 0;
}

void insert_by_index(int i, int k, float d)
{
  for (int b = 0; b < atom[i].count; ++b)
  {
    if (atom[i].nbnb[b].i == k) return;
  }
  if (atom[i].count < B * B + B)
```

```
    {
      atom[i].nbnb[atom[i].count].f = d;
      atom[i].nbnb[atom[i].count].i = k;
      atom[i].count += 1;
    }
  }

  void sort_by_distance(int i)
  {
    qsort(&atom[i].nbnb[0], atom[i].count, sizeof(atom[i].nbnb[0]), cmp_index_f);
  }

  float metric = 0;
  void update_nearest_neighbours()
  {
    float m = 0.0;
    for (int i = 0; i < N; ++i)
    {
      atom[i].count = 0;
      for (int b = 0; b < B; ++b)
      {
        int j = atom[i].nb[b];
        {
          float dx = atom[i].x - atom[j].x;
          float dy = atom[i].y - atom[j].y;
          float dz = atom[i].z - atom[j].z;
          float d = dx * dx + dy * dy + dz * dz;
          if (i != j)
          insert_by_index(i, j, d);
        }
        for (int c = 0; c < B; ++c)
        {
          int k = atom[j].nb[c];
          if (i == k) continue;
          float dx = atom[i].x - atom[k].x;
          float dy = atom[i].y - atom[k].y;
          float dz = atom[i].z - atom[k].z;
          float d = dx * dx + dy * dy + dz * dz;
          insert_by_index(i, k, d);
        }
      }
      assert(atom[i].count >= B);
      sort_by_distance(i);
      for (int b = 0; b < B; ++b)
        atom[i].nb[b] = atom[i].nbnb[b].i;
      m = fmaxf(m, atom[i].nbnb[B-1].f);
    }
    metric = m;
  }

  void update_oscillator_amplitudes(void)
  {
    int dest = 1 - which_oscillator_buffer;
  #if 0
    int l = 0;
    for (int i = -1; i <= 1; i += 1)
    for (int j = -1; j <= 1; j += 1)
```

```
535     for (int k = -1; k <= 1; k += 1)
        {
          if (l >= OSCILLATORS) break;
          if (i == 0 && j == 0 && k == 0) continue;
          oscillator_targets[l] = find_target(i, j, k);
540       ++l;
        }
        for (int i = 0; i < OSCILLATORS && i < N; ++i)
        {
          oscillators[dest][i].i = oscillator_targets[i];
545       oscillators[dest][i].f = powf(atom[oscillator_targets[i]].speed, 8.0);
        }
        qsort(&oscillators[dest][0], OSCILLATORS < N ? OSCILLATORS : N, sizeof(↙
            ↳ oscillators[0][0]), cmp_index_f_rev);
      #else
        for (int i = 0; i < N; ++i)
550     {
          oscillators[dest][i].i = i;
          oscillators[dest][i].f = powf(atom[i].speed, 8.0);
        }
      //  qsort(&oscillators[dest][0], N, sizeof(oscillators[0][0]), cmp_index_f_rev);
555   #endif
        while (in_audio_cb)
          ; // spin in case audio callback is copying previous data
        which_oscillator_buffer = dest;
      //  return atom[oscillators[dest][OSCILLATORS - 1].i].speed;
560   }

      void step(void)
      {
        for (int i = 0; i < N; ++i)
565       field_at_atom(&atom[i].ddx, &atom[i].ddy, &atom[i].ddz, i);
        for (int i = 0; i < N; ++i)
        {
          atom[i].dx += atom[i].ddx * dt;
          atom[i].dy += atom[i].ddy * dt;
570       atom[i].dz += atom[i].ddz * dt;
          // d = normalize(cross(p, cross(p, d)))
          float a1 = atom[i].x;
          float a2 = atom[i].y;
          float a3 = atom[i].z;
575       float b1 = atom[i].dx;
          float b2 = atom[i].dy;
          float b3 = atom[i].dz;
          float s1 = a2 * b3 - a3 * b2;
          float s2 = a3 * b1 - a1 * b3;
580       float s3 = a1 * b2 - a2 * b1;
          float t1 = a2 * s3 - a3 * s2;
          float t2 = a3 * s1 - a1 * s3;
          float t3 = a1 * s2 - a2 * s1;
          float t = -sqrtf(t1 * t1 + t2 * t2 + t3 * t3);
585       atom[i].dx = t1 / t;
          atom[i].dy = t2 / t;
          atom[i].dz = t3 / t;
        }
        for (int i = 0; i < N; ++i)
590     {
```

```
        float ox = atom[i].x;
        float oy = atom[i].y;
        float oz = atom[i].z;
        // boost = exp(dot(o, focus));
595     float boost = expf(ox * focus[0] + oy * focus[1] + oz * focus[2]);
        float x = atom[i].x + atom[i].dx * dt * SPEED * boost;
        float y = atom[i].y + atom[i].dy * dt * SPEED * boost;
        float z = atom[i].z + atom[i].dz * dt * SPEED * boost;
        float R = x * x + y * y + z * z;
600     R = sqrtf(R);
        atom[i].x = x / R;
        atom[i].y = y / R;
        atom[i].z = z / R;
    #if 0
605     float nx = atom[i].x;
        float ny = atom[i].y;
        float nz = atom[i].z;
        float dx = nx - ox;
        float dy = ny - oy;
610     float dz = nz - oz;
        float s = sqrtf(dx * dx + dy * dy + dz * dz);
        atom[i].speed = tanhf(s / (SPEED * dt));
    #else
        atom[i].speed = tanhf(boost);
615 #endif

      }
      update_nearest_neighbours();
      update_oscillator_amplitudes();
620 }

    void plot(int x, int y, int r, int g, int b)
    {
      if (0 <= x && x < W && 0 <= y && y < H)
625   {
        y = H-1 - y;
        x = (W/2 + W-1 - x) % W;
        ppm[y][x][0] = r;
        ppm[y][x][1] = g;
630     ppm[y][x][2] = b;
      }
    }

    void lineL(int x0,int y0, int x1, int y1, int r, int g, int b)
635 {
      int dx = x1 - x0;
      int dy = y1 - y0;
      int yi = 1;
      if (dy < 0)
640   {
        yi = -1;
        dy = -dy;
      }
      int D = 2*dy - dx;
645   int y = y0;
      for (int x = x0; x <= x1; ++x)
      {
```

```
      plot(x, y, r, g, b);
      if (D > 0)
      {
         y = y + yi;
         D = D - 2*dx;
      }
      D = D + 2*dy;
    }
}

void lineH(int x0, int y0, int x1, int y1, int r, int g, int b)
{
  int dx = x1 - x0;
  int dy = y1 - y0;
  int xi = 1;
  if (dx < 0)
  {
    xi = -1;
    dx = -dx;
  }
  int D = 2*dx - dy;
  int x = x0;

  for (int y = y0; y <= y1; ++y)
  {
    plot(x, y, r, g, b);
    if (D > 0)
    {
       x = x + xi;
       D = D - 2*dy;
    }
    D = D + 2*dx;
  }
}

void line(int x0, int y0, int x1, int y1, int r, int g, int b)
{
  if (abs(y1 - y0) < abs(x1 - x0))
    if (x0 > x1)
      lineL(x1, y1, x0, y0, r, g, b);
    else
      lineL(x0, y0, x1, y1, r, g, b);
  else
    if (y0 > y1)
      lineH(x1, y1, x0, y0, r, g, b);
    else
      lineH(x0, y0, x1, y1, r, g, b);
}

void draw_geodesic(float x0, float y0, float z0, float x1, float y1, float z1, ↙
    ↳ int red, int grn, int blu, int depth)
{
  if (depth == 0)
  {
    int i0 = atan2f(y0, x0) / (2 * pi) * W;
    if (i0 < 0) i0 += W;
    int j0 = acosf(z0) / pi * H;
```

```
      int i1 = atan2f(y1, x1) / (2 * pi) * W;
      if (i1 < 0) i1 += W;
      int j1 = acosf(z1) / pi * H;
      if (i0 < W * 0.25 && W * 0.75 < i1)
      {
        line(i0, j0, i1 - W, j1, red, grn, blu);
        line(i0 + W, j0, i1, j1, red, grn, blu);
      }
      else
      if (i1 < W * 0.25 && W * 0.75 < i0)
      {
        line(i0 - W, j0, i1, j1, red, grn, blu);
        line(i0, j0, i1 + W, j1, red, grn, blu);
      }
      else
        line(i0, j0, i1, j1, red, grn, blu);
    }
    else
    {
      float x2 = x0 + x1;
      float y2 = y0 + y1;
      float z2 = z0 + z1;
      float r2 = x2 * x2 + y2 * y2 + z2 * z2;
      float r = sqrtf(r2);
      x2 /= r;
      y2 /= r;
      z2 /= r;
      draw_geodesic(x0, y0, z0, x2, y2, z2, red, grn, blu, depth - 1);
      draw_geodesic(x2, y2, z2, x1, y1, z1, red, grn, blu, depth - 1);
    }
  }

  void draw_lines_for(int i, int red, int grn, int blu)
  {
    for (int b = 0; b < B; ++b)
    {
      int j = atom[i].nb[b];
      draw_geodesic
        ( atom[i].x, atom[i].y, atom[i].z
        , atom[j].x, atom[j].y, atom[j].z
        , red, grn, blu
        , 6
        );
    }
  }

  void draw_point(int i, int red, int grn, int blu)
  {
    float ax = atom[i].x;
    float ay = atom[i].y;
    float az = atom[i].z;
    float x0 = atan2f(ay, ax) / (2 * pi);
    x0 -= floor(x0);
    x0 *= W;
    float y0 = acosf(az) / pi * H;
    for (int dy = -R0; dy <= R0; ++dy)
    {
```

```
      int y = y0 + dy;
      float w = 1 - fabsf((y0 + dy + 0.5f) / H * 2 - 1);
      if (y < 0) continue;
      if (y >= H) continue;
      for (int dx = -R0 / w; dx <= R0 / w; ++dx)
      {
        int x = x0 + dx;
        while (x < 0) x += W;
        while (x >= W) x -= W;
        if (dx * dx * w * w + dy * dy > R0 * R0) continue;
        plot(x, y, red, grn, blu);
      }
    }
  }

  void draw_lines(int target)
  {
    // everything dark blue
    for (int i = 0; i < N; ++i)
      draw_lines_for(i, 0x00, 0x00, 0x80);
    // neighbours bright green
    for (int b = 0; b < B; ++b)
      draw_lines_for(atom[target].nb[b], 0x00, 0xFF, 0x00);
    // target yellow
    draw_lines_for(target, 0xFF, 0xFF, 0x00);
  }


  void draw_points(int target)
  {
    // everything dark blue
    for (int i = 0; i < N; ++i)
      draw_point(i, 0x00, 0x00, 0x80);
    // neighbours of neighbours bright green
    for (int b = 0; b < B; ++b)
    {
      int j = atom[target].nb[b];
      for (int c = 0; c < B; ++c)
        draw_point(atom[j].nb[c], 0x00, 0xFF, 0x00);
    }
    // neighbours yellow
    for (int b = 0; b < B; ++b)
      draw_point(atom[target].nb[b], 0xFF, 0xFF, 0x00);
    // target red
    draw_point(target, 0xFF, 0x00, 0x00);
  }

  void draw_trails(void )
  {
    const int colour[4][3] = { { 0, 0, 0 }, { 0, 0x80, 0xFF }, { 0xFF, 51, 0 }, { ↙
        ↳ 0xFF, 0xFF, 0xFF } };
    for (int i = 0; i < N; ++i)
    {
      if (i % 4 != 3) continue;
      draw_point(i, colour[i%4][0], colour[i%4][1], colour[i%4][2]);
    }
  }
```

```
void output(FILE *vidfile)
{
  fprintf(vidfile, "P6\n%d %d\n255\n", W, H);
  fwrite(&ppm[0][0][0], sizeof(ppm[0][0][0]) * 3 * W * H, 1, vidfile);
  fflush(vidfile);
}

// entry point
extern int main(int argc, char **argv)
{
  (void) argv;
  int RECORD = argc > 1;

  for (int o = 0; o < 16; ++o)
    lop1_init(&output_filters[o], 1000);

  glfwInit();
  glfwWindowHint(GLFW_CLIENT_API, GLFW_OPENGL_API);
  glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
  glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
  glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
  glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
  glfwWindowHint(GLFW_RESIZABLE, GL_FALSE);
  glfwWindowHint(GLFW_DECORATED, GL_TRUE);
  GLFWwindow *window = glfwCreateWindow(WINW, WINH, "meshwalk", 0, 0);
  glfwMakeContextCurrent(window);
  glfwSetKeyCallback(window, keycb);
  glewExperimental = GL_TRUE;
  glewInit();
  glGetError(); // discard common error from glew
  // set up vertex array object
  glClearColor(0, 0, 0, 1);
  GLuint vao;
  glGenVertexArrays(1, &vao);
  glBindVertexArray(vao);
  // set up framebuffer
  GLuint fbo[2];
  glGenFramebuffers(2, &fbo[0]);
  GLuint tex[2];
  glGenTextures(2, &tex[0]);
  glBindFramebuffer(GL_FRAMEBUFFER, fbo[0]);
  glActiveTexture(GL_TEXTURE1);
  glBindTexture(GL_TEXTURE_2D_MULTISAMPLE, tex[0]);
  glTexImage2DMultisample(GL_TEXTURE_2D_MULTISAMPLE, 8, GL_RGB, 8192/SUB, 4096/↙
      ↳ SUB, GL_FALSE);
  glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, ↙
      ↳ GL_TEXTURE_2D_MULTISAMPLE, tex[0], 0);
  GLenum status = glCheckFramebufferStatus(GL_FRAMEBUFFER);
  if (status != GL_FRAMEBUFFER_COMPLETE)
    fprintf(stderr, "GL FRAMEBUFFER STATUS %d\n", status);
  glBindFramebuffer(GL_FRAMEBUFFER, fbo[1]);
  glActiveTexture(GL_TEXTURE0);
  glBindTexture(GL_TEXTURE_2D, tex[1]);
  glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, 8192/SUB, 4096/SUB, 0, GL_RGB, ↙
      ↳ GL_UNSIGNED_BYTE, 0);
```

```
  glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, ↙
      ↳ tex[1], 0);
  status = glCheckFramebufferStatus(GL_FRAMEBUFFER);
  if (status != GL_FRAMEBUFFER_COMPLETE)
    fprintf(stderr, "GL FRAMEBUFFER STATUS %d\n", status);
875 // set up shader
  const char *s_vertex =
    "#version 330 core\n"
    "out vec2 tc;\n"
    "const float pi = 3.141592653;\n"
880 "void main()\n"
    "{\n"
    "  if (gl_VertexID == 0) { tc = vec2( pi, -pi/2.0); gl_Position = vec4(-1.0,↙
        ↳  -1.0, 0.0, 1.0); } else\n"
    "  if (gl_VertexID == 1) { tc = vec2(-pi, -pi/2.0); gl_Position = vec4( 1.0,↙
        ↳  -1.0, 0.0, 1.0); } else\n"
    "  if (gl_VertexID == 2) { tc = vec2( pi,  pi/2.0); gl_Position = vec4(-1.0,↙
        ↳   1.0, 0.0, 1.0); } else\n"
885 "  if (gl_VertexID == 3) { tc = vec2(-pi,  pi/2.0); gl_Position = vec4( 1.0,↙
        ↳   1.0, 0.0, 1.0); } else\n"
    "  { tc = vec2(0.0); gl_Position = vec4(0.0); }\n"
    "}\n"
    ;
  const char *s_fragment =
890   "#version 330 core\n"
    "in vec2 tc;\n"
    "uniform int N;\n"
    "uniform vec4 position[4000];\n"
    "layout (location = 0) out vec4 colour;\n"
895 "void main()\n"
    "{\n"
    "  vec3 me = vec3(cos(tc.x) * cos(tc.y), sin(tc.x) * cos(tc.y), sin(tc.y));\↙
        ↳ n"
    "  float delta = length(vec2(length(dFdx(me)), length(dFdy(me))));\n"
    "  float maxdot = -2.0;\n"
900 "  int ix = 0;\n"
    "  float maxdot2 = -2.0;\n"
    "  int ix2 = 0;\n"
    "  for (int i = 0; i < N && i < 4000; ++i)\n"
    "  {\n"
905 "    float d = dot(position[i].xyz, me);\n"
    "    if (d > maxdot) { maxdot2 = maxdot; ix2 = ix; maxdot = d; ix = i; }\n"
    "    else\n"
    "    if (d > maxdot2) { maxdot2 = d; ix2 = i; }\n"
    "  }\n"
910 "  vec3 blue = vec3(0.0, 0.5, 1.0);\n"
    "  vec3 white = vec3(1.0);\n"
    "  vec3 red = vec3(1.0, 0.2, 0.0);\n"
    "  vec3 black = vec3(0.0);\n"
    "  vec3 mine = mix(vec3(0.5), vec3[4](black, blue, red, white)[ix & 3], ↙
        ↳ position[ix].w);\n"
915 "  vec3 them = mix(vec3(0.5), vec3[4](black, blue, red, white)[ix2 & 3], ↙
        ↳ position[ix2].w);\n"
    "  float lmine = length(position[ix].xyz - me);\n"
    "  float lthem = length(position[ix2].xyz - me);\n"
    "  float lme = lmine - lthem;\n"
    "  colour = vec4(mix(mine, them, smoothstep(-delta, delta, lme)), 1.0);\n"
```

```
920       "}\n"
      ;
      GLuint display = vertex_fragment_shader(s_vertex, s_fragment);
      glUseProgram(display);
      u_position = glGetUniformLocation(display, "position");
925   GLint u_N = glGetUniformLocation(display, "N");
      glUniform1i(u_N, N);
      SF_INFO info = { 0, SR, 16, SF_FORMAT_WAV | SF_FORMAT_FLOAT, 0, 0 };
      SNDFILE *sndfile = 0;
      FILE *vidfile = stdout;
930   if (RECORD)
      {
        sndfile = sf_open("meshwalk-3.1_a.wav", SFM_WRITE, &info);
      }
      else
935   {
        if (!(client = jack_client_open("meshwalk", JackNoStartServer, 0))) {
          fprintf (stderr, "jack server not running?\n");
          return 1;
        }
940     jack_set_process_callback(client, processcb, 0);
        /* create ports */
        for (int i = 0; i < 16; ++i)
        {
          char portname[100];
945       snprintf(portname, 100, "output_%d", i + 1);
          port[i] = jack_port_register(client, portname, JACK_DEFAULT_AUDIO_TYPE, ↙
              ↳ JackPortIsOutput, 0);
        }
        /* activate audio */
        if (jack_activate(client)) {
950       fprintf (stderr, "cannot activate JACK client");
          return 1;
        }
      }

955   float frames[SR/FPS][16];
      // write titles
      if (RECORD)
      {
        for (int i = 0; i < SR/FPS; ++i)
960       for (int c = 0; c < 16; ++c)
            frames[i][c] = 0;
        for (int f = 0; f < FPS * 3; ++f)
          sf_writef_float(sndfile, &frames[0][0], SR/FPS);
      }
965   init();
      int dirty = 1;
      double last = glfwGetTime();
      int last_count = 0;
      memset(&ppm[0][0][0], 0x80, 8192/SUB * 4096/SUB * 3);
970   float focus_target[3] = { 0, 0, 0 };
      LOP1 focus_filter[3];
      float fhz = 4.0f * SR / FPS;
      lop1_init(&focus_filter[0], fhz);
      lop1_init(&focus_filter[1], fhz);
975   lop1_init(&focus_filter[2], fhz);
```

```
    // 1/2 = g * (1 + (1-g) + (1-g)^2 + ... + (1-g)^(H-1))
    //     = g * (1 - (1 - g)^(H-1)) / (1 - (1 - g))
    //     = 1 - (1 - g)^(H-1)
    //   g = 1 - (1/2)^(1/(H-1))
    double geiger = 1.0 - pow(0.5, 4.0 / FPS);
    for (int f = 0; ! glfwWindowShouldClose(window); ++f)
    {
      DISTANCE = 1;
      Ky = DISTANCE * 2.0f * sqrtf(4.0f / N);
      K = 1.0 / (Ky * Ky);
      sqrtK = sqrtf(K);
      if (rand() / (double) RAND_MAX < geiger)
      {
        float R;
        do
        {
          R = 0;
          for (int c = 0; c < 3; ++c)
          {
            focus_target[c] = 2.0f * (rand() / (float) RAND_MAX - 0.5f);
            R += focus_target[c] * focus_target[c];
          }
        } while (R > 1);
        R = sqrtf(R);
        float G = 0.5 * (1 - cos(2 * pi * (f - (FPS * (1 * 60 + 0))) / (FPS * (4 *↙
            ↳ 60 - 6)))) / R;
        focus_target[0] *= G;
        focus_target[1] *= G;
        focus_target[2] *= G;
      }
      focus[0] = lop1(&focus_filter[0], focus_target[0]);
      focus[1] = lop1(&focus_filter[1], focus_target[1]);
      focus[2] = lop1(&focus_filter[2], focus_target[2]);
      if (DEBUG_AMBISONICS_AXES)
      {
        int k = (f / FPS) % 6;
        const float targets[6][3] =
          { { 1, 0, 0 }, { -1, 0, 0 } // front, back
          , { 0, 1, 0 }, { 0, -1, 0 } // left, right
          , { 0, 0, 1 }, { 0, 0, -1 } // top, bottom
          };
        focus[0] = targets[k][0];
        focus[1] = targets[k][1];
        focus[2] = targets[k][2];
      }
      if (DEBUG_AMBISONICS_SPIN != 0)
      {
        float t = DEBUG_AMBISONICS_SPIN;
        float c = cos(t);
        float s = sin(t);
        float M[3][3] = { { c, s, 0 }, { -s, c, 0}, { 0, 0, 1 } };
        float nV[3][3];
        for (int i = 0; i < 3; ++i)
        for (int j = 0; j < 3; ++j)
        {
          nV[i][j] = 0;
          for (int k = 0; k < 3; ++k)
```

```
            nV[i][j] += M[i][k] * V[k][j];
        }
        for (int i = 0; i < 3; ++i)
        for (int j = 0; j < 3; ++j)
          V[i][j] = nV[i][j];
      }
      double now = glfwGetTime();
      if (now - last > 1)
      {
        fprintf(stderr, "\n%f fps\n%f max\n", (f - last_count) / (now - last), sqrtf(metric) / Ky);
        last_count = f;
        last = now;
      }
      if (RECORD && f >= FPS * (60 * 5 - 6))
        break;

      step();

      if ((RECORD && f >= FPS * 60 * 1) || ! RECORD)
      {
#if RECORD_VIDEO
        upload_uniforms();
        glBindFramebuffer(GL_FRAMEBUFFER, fbo[1]);
        glClear(GL_COLOR_BUFFER_BIT);
        glViewport(0, 0, 8192/SUB, 4096/SUB);
        glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);
#if 0
        glBindFramebuffer(GL_DRAW_FRAMEBUFFER, fbo[1]);
        glBlitFramebuffer
          ( 0, 0, 8192/SUB, 4096/SUB
          , 0, 0, 8192/SUB, 4096/SUB
          , GL_COLOR_BUFFER_BIT, GL_LINEAR
          );
        // download output frame
        glBindFramebuffer(GL_READ_FRAMEBUFFER, fbo[1]);
#endif
        if (RECORD && f >= FPS * 60 * 1 && DRAW_VORONOI)
        {
          dirty = 0;
          glReadPixels(0, 0, 8192/SUB, 4096/SUB, GL_RGB, GL_UNSIGNED_BYTE, &ppm[0][0][0]);
        }
        else if (dirty && DRAW_CLEAR)
        {
          dirty = 0;
          memset(&ppm[0][0][0], 0, 8192/SUB * 4096/SUB * 3);
        }
        int target = find_target(1, 0, 0);
        if (DRAW_LINES)
        {
          dirty = 1;
          draw_lines(target);
        }
        if (DRAW_POINTS)
        {
          dirty = 1;
```

```
        draw_points(target);
      }
      if (DRAW_TRAILS)
      {
        dirty = 1;
        draw_trails();
      }
      // preview
      glBindFramebuffer(GL_DRAW_FRAMEBUFFER, 0);
      glClear(GL_COLOR_BUFFER_BIT);
      glBlitFramebuffer
        ( 0, 0, 8192/SUB, 4096/SUB
        , 0, (WINH - WINW/2) / 2, WINW, (WINH + WINW/2) / 2
        , GL_COLOR_BUFFER_BIT, GL_LINEAR
        );
      // redisplay
      glfwSwapBuffers(window);
      glfwPollEvents();
      GLuint e;
      while ((e = glGetError())) fprintf(stderr, "GL ERROR %d\n", e);
#endif

      // output PPM to stdout (flipped vertically)
      if (RECORD && f >= FPS * 60 * 1)
      {
#if RECORD_VIDEO
        fprintf(vidfile, "P6\n%d %d\n255\n", 8192/SUB, 4096/SUB);
        fwrite(&ppm[0][0][0], 8192/SUB * 4096/SUB * 3, 1, vidfile);
        if (f == FPS * 60 * 1)
        {
          FILE *imgfile = popen("pnmtopng -force -interlace -compression 9 > ↙
              ↳ meshwalk-3.1.png", "w");
          fprintf(imgfile, "P6\n%d %d\n255\n", 8192/SUB, 4096/SUB);
          fwrite(&ppm[0][0][0], 8192/SUB * 4096/SUB * 3, 1, imgfile);
          pclose(imgfile);
        }
#endif
#if RECORD_AUDIO
        float audio[16][SR/FPS];
        float *audiop[16];
        for (int c = 0; c < 16; ++c)
          audiop[c] = &audio[c][0];
        audiocb(audiop, SR/FPS, f * SR/FPS);
        for (int i = 0; i < SR/FPS; ++i)
          for (int c = 0; c < 16; ++c)
            frames[i][c] = audio[c][i];
        sf_writef_float(sndfile, &frames[0][0], SR/FPS);
#endif
      }
    }
  }
  if (RECORD)
  {
    // write credits
    for (int i = 0; i < SR/FPS; ++i)
      for (int c = 0; c < 16; ++c)
        frames[i][c] = 0;
```

```
    for (int f = 0; f < FPS * 3; ++f)
      sf_writef_float(sndfile, &frames[0][0], SR/FPS);
    sf_close(sndfile);
  }
  glfwDestroyWindow(window);
  glfwTerminate();
  return 0;
}
```

67 meshwalk/meshwalk-3.1-credits.png

68 meshwalk/meshwalk-3.1.sh

```
#!/bin/bash
set -e
make meshwalk-3.1
(
  for i in $(seq 180)
  do
    pngtopnm < meshwalk-3.1-title.png
  done |
  ffmpeg -r 60 -i - -vf fade=in:30:30,fade=out:120:30 -f image2pipe -codec:v ppm -
  ./meshwalk-3.1 -nrt
  for i in $(seq 180)
  do
    pngtopnm < meshwalk-3.1-credits.png
  done |
  ffmpeg -r 60 -i - -vf fade=in:30:30,fade=out:120:30 -f image2pipe -codec:v ppm -
) |
ffmpeg -framerate 60 -i - -codec:v png -y meshwalk-3.1_v.mkv
ffmpeg -i meshwalk-3.1_v.mkv -i meshwalk-3.1_a.wav \
  -codec:v copy -codec:a copy \
  -y meshwalk-3.1.mkv
ffmpeg -i meshwalk-3.1.mkv -af pan="4.0|c0=c0|c1=c1|c2=c2|c3=c3" \
  -pix_fmt yuv420p -profile:v high -level:v 5.1 -tune:v animation \
  -crf:v 20 -b:a 512k -movflags +faststart \
  -y meshwalk-3.1_m.mp4
python2.7 ../../../github.com/google/spatial-media/spatialmedia \
  -i --spatial-audio meshwalk-3.1_m.mp4 meshwalk-3.1.mp4
```

69 meshwalk/meshwalk-3.1-title.png

70 meshwalk/meshwalk-3.1-titles.tex

```
\documentclass{article}
\usepackage[paperwidth=8.192cm,paperheight=4.096cm,margin=0.1cm]{geometry}
\usepackage{lmodern}
\begin{document}
\thispagestyle{empty}
\hspace{0pt}
\vfill
\centering \sf
{\small meshwalk-3.1}

{\tiny mathr.co.uk}

{\tiny 2018}
\vfill
\hspace{0pt}
\end{document}
```

71 meshwalk/meshwalk-3.2.c

```
/*
  meshwalk -- audiovisual drone
  Copyright (C) 2018  Claude Heiland-Allen <claude@mathr.co.uk>

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

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

```
# realtime mode with JACK audio, press Q to exit
./meshwalk-3.2

# render
./meshwalk-3.2.sh
*/

#include <assert.h>
#include <limits.h>
#include <math.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

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

#include <jack/jack.h>
#include <sndfile.h>

#define pi 3.141592653f

#define SR 48000
#define FPS 30
#define LOD 4
#define OSCILLATORS 4000
#define GAIN 0.1f

#define DEBUG_AMBISONICS 0
#define DEBUG_AMBISONICS_SPIN 0 //(2 * pi / FPS / 12)
#define DEBUG_AMBISONICS_AXES 0

#define DRAW_CLEAR 0
#define DRAW_VORONOI 1
#define DRAW_LINES 0
#define DRAW_POINTS 0
#define DRAW_TRAILS 0

#define RECORD_AUDIO 1
#define RECORD_VIDEO 1

#define R0 4
#define W 8192
#define H 4096

#define WINW 1024
#define WINH 512

// note: must be <= 4000 (limit set in fragment shader)
#define N OSCILLATORS
#define B 12

// one-pole one-zero low pass filter designed in the Z plane

typedef float sample;
```

```
typedef struct {
  sample x, b;
  double y, a;
} LOP1;

static inline LOP1 *lop1_init(LOP1 *s, sample hz)
{
  const double w = 2 * pi * fminf(fmaxf(fabs(hz / SR), 0), (sample)0.5);
  const double a = (1 - sin(w)) / cos(w);
  const double b = (1 - a) / 2;
  s->a = a;
  s->b = b;
  s->x = 0;
  s->y = 0;
  return s;
}

static inline sample lop1(LOP1 *s, sample x)
{
  const double y = s->b * (x + s->x) + s->a * s->y;
  s->x = x;
  s->y = y;
  return y;
}


float DISTANCE;
float Ky;
float K;
float sqrtK;
const float dt = 1.0f / FPS;
const float SPEED = 2.0 * sqrtf(1.0f / N);

GLint u_position;
GLfloat position[N][4];

GLfloat V[3][3] = { { 1, 0, 0 }, { 0, 1, 0 }, { 0, 0, 1 } };

float focus[3];

const float spin_axes[4][3] =
  { { -1, -1, -1 }
  , {  1,  1, -1 }
  , {  1, -1,  1 }
  , { -1,  1,  1 }
  };

int oscillator_targets[OSCILLATORS];

LOP1 output_filters[16];

typedef struct { float f; int i; } index_t;

typedef struct
{
  float x, y, z, dx, dy, dz, ddx, ddy, ddz, speed;
  index_t nbnb[B * B + B];
```

```
  int nb[B];
  int count;
} node;

node atom[N];
int which_oscillator_buffer = 0;
index_t oscillators[3][N];

static inline uint32_t hash(uint32_t a)
{
        a = (a+0x7ed55d16u) + (a<<12);
        a = (a^0xc761c23cu) ^ (a>>19);
        a = (a+0x165667b1u) + (a<<5);
        a = (a+0xd3a2646cu) ^ (a<<9);
        a = (a+0xfd7046c5u) + (a<<3);
        a = (a^0xb55a4f09u) ^ (a>>16);
        return a;
}

// GLFW keyboard callback
static void keycb
  ( GLFWwindow *window
  , int key
  , int scancode
  , int action
  , int mods
  )
{
  (void) key;
  (void) scancode;
  (void) mods;
  if (action == GLFW_PRESS)
    glfwSetWindowShouldClose(window, GL_TRUE);
}

float audio_xyz[N][3];
static void audiocb(float *output[16], int n, int f, int which)
{
  float out[16][n];
  int source = which_oscillator_buffer;
  for (int o = 0; o < OSCILLATORS && o < N; ++o)
  {
    int i = oscillators[source][o].i;
    oscillators[2][o] = oscillators[source][o];
    float x = atom[i].x;
    float y = atom[i].y;
    float z = atom[i].z;
    audio_xyz[o][0] = V[0][0] * x + V[0][1] * y + V[0][2] * z;
    audio_xyz[o][1] = V[1][0] * x + V[1][1] * y + V[1][2] * z;
    audio_xyz[o][2] = V[2][0] * x + V[2][1] * y + V[2][2] * z;
  }
  for (int c = 0; c < 16; ++c)
    for (int t = 0; t < n; ++t)
      out[c][t] = 0;
  for (int o = 0; o < OSCILLATORS && o < N; ++o)
  {
    int i = oscillators[2][o].i;
```

```
      if (which >= 0 && (i & 3) != which) continue;
      float amp = oscillators[2][o].f;
      float x = audio_xyz[o][0];
      float y = audio_xyz[o][1];
      float z = audio_xyz[o][2];
      float factor[16];
      // order 0
      factor[ 0] = 1;
      // order 1
      factor[ 1] = y;
      factor[ 2] = z;
      factor[ 3] = x;
      //  order 2
      factor[ 4] = sqrt(3) * x * y;
      factor[ 5] = sqrt(3) * y * z;
      factor[ 6] = 1/2. * (3 * z * z - 1);
      factor[ 7] = sqrt(3) * x * z;
      factor[ 8] = sqrt(3/4.) * (x * x - y * y);
      // order 3
      factor[ 9] = sqrt(5/8.) * y * (3 * x * x - y * y);
      factor[10] = sqrt(15) * x * y * z;
      factor[11] = sqrt(3/8.) * y * (5 * z * z - 1);
      factor[12] = 1/2. * z * (5 * z * z - 3);
      factor[13] = sqrt(3/8.) * x * (5 * z * z - 1);
      factor[14] = sqrt(15/4.) * z * (x * x - y * y);
      factor[15] = sqrt(5/8.) * x * (x * x - 3 * y * y);
      int period = (720 + ((i % 15)-7)) / ((i % 4) + 1);
      float osc[n];
      for (int t = 0; t < n; ++t)
        osc[t] = hash(hash((f + t) % period) * N + i) / (float) UINT_MAX - 0.5f;
      for (int c = 0; c < 16; ++c)
      {
        factor[c] *= amp;
        for (int t = 0; t < n; ++t)
          out[c][t] += osc[t] * factor[c];
      }
    }
    float level = GAIN;
    for (int c = 0; c < 16; ++c)
      for (int t = 0; t < n; ++t)
        output[c][t] = lop1(&output_filters[c], out[c][t] * level);
}

jack_client_t *client[4];
jack_port_t *port[4][16];
static int processcb(jack_nframes_t nframes, void *arg) {
  int *which = arg;
  static jack_nframes_t f[4] = { 0, 0, 0, 0 };
  jack_default_audio_sample_t *out[16];
  for (int c = 0; c < 16; ++c)
  {
    out[c] = jack_port_get_buffer(port[*which][c], nframes);
    for (jack_nframes_t t = 0; t < nframes; ++t)
      out[c][t] = 0;
  }
  audiocb(out, nframes, f[*which], *which);
  f[*which] += nframes;
```

```
250   return 0;
}


static void debug_program(GLuint program) {
255   GLint status = 0;
  glGetProgramiv(program, GL_LINK_STATUS, &status);
  GLint length = 0;
  glGetProgramiv(program, GL_INFO_LOG_LENGTH, &length);
  char *info = 0;
260   if (length) {
    info = malloc(length + 1);
    info[0] = 0;
    glGetProgramInfoLog(program, length, 0, info);
    info[length] = 0;
265   }
  if ((info && info[0]) || ! status) {
    fprintf(stderr, "program link info:\n%s", info ? info : "(no info log)");
  }
  if (info) {
270   free(info);
  }
}

static void debug_shader(GLuint shader, GLenum type, const char *source) {
275   GLint status = 0;
  glGetShaderiv(shader, GL_COMPILE_STATUS, &status);
  GLint length = 0;
  glGetShaderiv(shader, GL_INFO_LOG_LENGTH, &length);
  char *info = 0;
280   if (length) {
    info = malloc(length + 1);
    info[0] = 0;
    glGetShaderInfoLog(shader, length, 0, info);
    info[length] = 0;
285   }
  if ((info && info[0]) || ! status) {
    const char *type_str = "unknown";
    switch (type) {
      case GL_VERTEX_SHADER: type_str = "vertex"; break;
290   case GL_FRAGMENT_SHADER: type_str = "fragment"; break;
      case GL_COMPUTE_SHADER: type_str = "compute"; break;
    }
    fprintf
      ( stderr
295   , "%s shader compile info:\n%s\nshader source:\n%s"
      , type_str
      , info ? info : "(no info log)"
      , source ? source : "(no source)"
      );
300   }
  if (info) {
    free(info);
  }
}
305
static GLuint vertex_fragment_shader(const char *vert, const char *frag) {
```

```
  GLuint program = glCreateProgram();
  {
    GLuint shader = glCreateShader(GL_VERTEX_SHADER);
    glShaderSource(shader, 1, &vert, 0);
    glCompileShader(shader);
    debug_shader(shader, GL_VERTEX_SHADER, vert);
    glAttachShader(program, shader);
    glDeleteShader(shader);
  }
  {
    GLuint shader = glCreateShader(GL_FRAGMENT_SHADER);
    glShaderSource(shader, 1, &frag, 0);
    glCompileShader(shader);
    debug_shader(shader, GL_FRAGMENT_SHADER, frag);
    glAttachShader(program, shader);
    glDeleteShader(shader);
  }
  glLinkProgram(program);
  debug_program(program);
  return program;
}

int cmp_index_f(const void *a, const void *b)
{
  const index_t *p = a;
  const index_t *q = b;
  const float x = p->f;
  const float y = q->f;
  return (x > y) - (x < y);
}

int cmp_index_f_rev(const void *a, const void *b)
{
  return -cmp_index_f(a, b);
}

void upload_uniforms(void)
{
  for (int i = 0; i < N; ++i)
  {
    float x = atom[i].x;
    float y = atom[i].y;
    float z = atom[i].z;
    position[i][0] = V[0][0] * x + V[0][1] * y + V[0][2] * z;
    position[i][1] = V[1][0] * x + V[1][1] * y + V[1][2] * z;
    position[i][2] = V[2][0] * x + V[2][1] * y + V[2][2] * z;
    position[i][3] = 1;
  }
  glUniform4fv(u_position, N, &position[0][0]);
}


unsigned char ppm[4096][8192][3];

void insert_by_distance(int i, int k, float d)
{
  int b;
```

```
    for (b = 0; b < B && b < atom[i].count; ++b)
365 {
      if (d < atom[i].nbnb[b].f) break;
    }
    if (b < B)
    {
370   for (int c = B - 1; c > b; --c)
      {
        atom[i].nbnb[c] = atom[i].nbnb[c - 1];
      }
      atom[i].nbnb[b].f = d;
375   atom[i].nbnb[b].i = k;
      if (atom[i].count < B) atom[i].count += 1;
    }
  }

380 int find_target(float tx, float ty, float tz)
  {
    float md = -2.0;
    int ix = -1;
    for (int i = 0; i < N; ++i)
385 {
      float d = atom[i].x * tx + atom[i].y * ty + atom[i].z * tz;
      if (d > md)
      {
        md = d;
390     ix = i;
      }
    }
    return ix;
  }
395
  void init(void)
  {
    for (int i = 0; i < N; ++i)
    {
400   #define U (rand() / (float) RAND_MAX)
      float R;
        float t = 2 * pi * (i + U) / N;
        float s = 2.0 * (U - 0.5);
      atom[i].x = cosf(t) * cosf(asinf(s));
405   atom[i].y = sinf(t) * cosf(asinf(s));
      atom[i].z = s;
      R = atom[i].x * atom[i].x + atom[i].y * atom[i].y + atom[i].z * atom[i].z;
      R = sqrtf(R);
      atom[i].x /= R;
410   atom[i].y /= R;
      atom[i].z /= R;
      atom[i].dx = 0;
      atom[i].dy = 0;
      atom[i].dz = 0;
415   #undef U
    }
    #pragma omp parallel for
    for (int i = 0; i < N; ++i)
    {
420   atom[i].count = 0;
```

```
    for (int k = 0; k < N; ++k)
    {
      if (i == k) continue;
      float dx = atom[i].x - atom[k].x;
      float dy = atom[i].y - atom[k].y;
      float dz = atom[i].z - atom[k].z;
      float d = dx * dx + dy * dy + dz * dz;
      insert_by_distance(i, k, d);
    }
    assert(atom[i].count == B);
    for (int b = 0; b < B; ++b)
    {
      atom[i].nb[b] = atom[i].nbnb[b].i;
    }
  }
}

static void field_at_atom(float *ddx, float *ddy, float *ddz, int i)
{
  *ddx = 0;
  *ddy = 0;
  *ddz = 0;
  int j = i % 4;
  for (int b = 0; b < B; ++b)
  {
    int k = atom[i].nb[b];
    int l = k % 4;
    float dx = atom[k].x - atom[i].x;
    float dy = atom[k].y - atom[i].y;
    float dz = atom[k].z - atom[i].z;
    float d2 = dx * dx + dy * dy + dz * dz;
    if (d2 == 0) continue;
    float d = sqrtf(d2);
    float F = 0.0;
    F += 1 / (d2 * K);
    if (l != j)
      F -= 1 / (d * sqrtK);
    else
      F *= 2;
    *ddx -= F * dx / d;
    *ddy -= F * dy / d;
    *ddz -= F * dz / d;
  }
  if (isnan(*ddx)) *ddx = 0;
  if (isnan(*ddy)) *ddy = 0;
  if (isnan(*ddz)) *ddz = 0;
}

void insert_by_index(int i, int k, float d)
{
  for (int b = 0; b < atom[i].count; ++b)
  {
    if (atom[i].nbnb[b].i == k) return;
  }
  if (atom[i].count < B * B + B)
  {
    atom[i].nbnb[atom[i].count].f = d;
```

```
      atom[i].nbnb[atom[i].count].i = k;
      atom[i].count += 1;
    }
  }

  void sort_by_distance(int i)
  {
    qsort(&atom[i].nbnb[0], atom[i].count, sizeof(atom[i].nbnb[0]), cmp_index_f);
  }

  float metric = 0;
  void update_nearest_neighbours()
  {
    float m = 0.0;
    for (int i = 0; i < N; ++i)
    {
      atom[i].count = 0;
      for (int b = 0; b < B; ++b)
      {
        int j = atom[i].nb[b];
        {
          float dx = atom[i].x - atom[j].x;
          float dy = atom[i].y - atom[j].y;
          float dz = atom[i].z - atom[j].z;
          float d = dx * dx + dy * dy + dz * dz;
          if (i != j)
          insert_by_index(i, j, d);
        }
        for (int c = 0; c < B; ++c)
        {
          int k = atom[j].nb[c];
          if (i == k) continue;
          float dx = atom[i].x - atom[k].x;
          float dy = atom[i].y - atom[k].y;
          float dz = atom[i].z - atom[k].z;
          float d = dx * dx + dy * dy + dz * dz;
          insert_by_index(i, k, d);
        }
      }
      assert(atom[i].count >= B);
      sort_by_distance(i);
      for (int b = 0; b < B; ++b)
        atom[i].nb[b] = atom[i].nbnb[b].i;
      m = fmaxf(m, atom[i].nbnb[B-1].f);
    }
    metric = m;
  }

  void update_oscillator_amplitudes(void)
  {
    int dest = 1 - which_oscillator_buffer;
  #if 0
    int l = 0;
    for (int i = -1; i <= 1; i += 1)
    for (int j = -1; j <= 1; j += 1)
    for (int k = -1; k <= 1; k += 1)
    {
```

```
535       if (l >= OSCILLATORS) break;
          if (i == 0 && j == 0 && k == 0) continue;
          oscillator_targets[l] = find_target(i, j, k);
         ++l;
        }
540     for (int i = 0; i < OSCILLATORS && i < N; ++i)
        {
          oscillators[dest][i].i = oscillator_targets[i];
          oscillators[dest][i].f = powf(atom[oscillator_targets[i]].speed, 8.0);
        }
545     qsort(&oscillators[dest][0], OSCILLATORS < N ? OSCILLATORS : N, sizeof(↙
          ↳ oscillators[0][0]), cmp_index_f_rev);
      #else
        for (int i = 0; i < N; ++i)
        {
          oscillators[dest][i].i = i;
550       oscillators[dest][i].f = powf(atom[i].speed, 8.0);
        }
      //  qsort(&oscillators[dest][0], N, sizeof(oscillators[0][0]), cmp_index_f_rev);
      #endif
        which_oscillator_buffer = dest;
555   //  return atom[oscillators[dest][OSCILLATORS - 1].i].speed;
      }

      void step(void)
      {
560     for (int i = 0; i < N; ++i)
          field_at_atom(&atom[i].ddx, &atom[i].ddy, &atom[i].ddz, i);
        for (int i = 0; i < N; ++i)
        {
          atom[i].dx += atom[i].ddx * dt;
565       atom[i].dy += atom[i].ddy * dt;
          atom[i].dz += atom[i].ddz * dt;
          // d = normalize(cross(p, cross(p, d)))
          float a1 = atom[i].x;
          float a2 = atom[i].y;
570       float a3 = atom[i].z;
          float b1 = atom[i].dx;
          float b2 = atom[i].dy;
          float b3 = atom[i].dz;
          float s1 = a2 * b3 - a3 * b2;
575       float s2 = a3 * b1 - a1 * b3;
          float s3 = a1 * b2 - a2 * b1;
          float t1 = a2 * s3 - a3 * s2;
          float t2 = a3 * s1 - a1 * s3;
          float t3 = a1 * s2 - a2 * s1;
580       float t = -sqrtf(t1 * t1 + t2 * t2 + t3 * t3);
          atom[i].dx = t1 / t;
          atom[i].dy = t2 / t;
          atom[i].dz = t3 / t;
        }
585     for (int i = 0; i < N; ++i)
        {
          float ox = atom[i].x;
          float oy = atom[i].y;
          float oz = atom[i].z;
590       // boost = exp(dot(o, focus));
```

```
      float boost = expf(ox * focus[0] + oy * focus[1] + oz * focus[2]);
      float x = atom[i].x + atom[i].dx * dt * SPEED * boost;
      float y = atom[i].y + atom[i].dy * dt * SPEED * boost;
      float z = atom[i].z + atom[i].dz * dt * SPEED * boost;
595   float R = x * x + y * y + z * z;
      R = sqrtf(R);
      atom[i].x = x / R;
      atom[i].y = y / R;
      atom[i].z = z / R;
600 #if 0
      float nx = atom[i].x;
      float ny = atom[i].y;
      float nz = atom[i].z;
      float dx = nx - ox;
605   float dy = ny - oy;
      float dz = nz - oz;
      float s = sqrtf(dx * dx + dy * dy + dz * dz);
      atom[i].speed = tanhf(s / (SPEED * dt));
    #else
610   atom[i].speed = tanhf(boost);
    #endif

    }
    update_nearest_neighbours();
615 update_oscillator_amplitudes();
  }

  void plot(int x, int y, int r, int g, int b)
  {
620 if (0 <= x && x < W && 0 <= y && y < H)
    {
      y = H-1 - y;
      x = (W/2 + W-1 - x) % W;
      ppm[y][x][0] = r;
625   ppm[y][x][1] = g;
      ppm[y][x][2] = b;
    }
  }

630 void lineL(int x0,int y0, int x1, int y1, int r, int g, int b)
  {
    int dx = x1 - x0;
    int dy = y1 - y0;
    int yi = 1;
635 if (dy < 0)
    {
      yi = -1;
      dy = -dy;
    }
640 int D = 2*dy - dx;
    int y = y0;
    for (int x = x0; x <= x1; ++x)
    {
      plot(x, y, r, g, b);
645   if (D > 0)
      {
         y = y + yi;
```

```
        D = D - 2*dx;
      }
      D = D + 2*dy;
    }
  }

  void lineH(int x0, int y0, int x1, int y1, int r, int g, int b)
  {
    int dx = x1 - x0;
    int dy = y1 - y0;
    int xi = 1;
    if (dx < 0)
    {
      xi = -1;
      dx = -dx;
    }
    int D = 2*dx - dy;
    int x = x0;

    for (int y = y0; y <= y1; ++y)
    {
      plot(x, y, r, g, b);
      if (D > 0)
      {
        x = x + xi;
        D = D - 2*dy;
      }
      D = D + 2*dx;
    }
  }

  void line(int x0, int y0, int x1, int y1, int r, int g, int b)
  {
    if (abs(y1 - y0) < abs(x1 - x0))
      if (x0 > x1)
        lineL(x1, y1, x0, y0, r, g, b);
      else
        lineL(x0, y0, x1, y1, r, g, b);
    else
      if (y0 > y1)
        lineH(x1, y1, x0, y0, r, g, b);
      else
        lineH(x0, y0, x1, y1, r, g, b);
  }

  void draw_geodesic(float x0, float y0, float z0, float x1, float y1, float z1, ↙
      ↪ int red, int grn, int blu, int depth)
  {
    if (depth == 0)
    {
      int i0 = atan2f(y0, x0) / (2 * pi) * W;
      if (i0 < 0) i0 += W;
      int j0 = acosf(z0) / pi * H;
      int i1 = atan2f(y1, x1) / (2 * pi) * W;
      if (i1 < 0) i1 += W;
      int j1 = acosf(z1) / pi * H;
      if (i0 < W * 0.25 && W * 0.75 < i1)
```

```
      {
        line(i0, j0, i1 - W, j1, red, grn, blu);
        line(i0 + W, j0, i1, j1, red, grn, blu);
      }
      else
      if (i1 < W * 0.25 && W * 0.75 < i0)
      {
        line(i0 - W, j0, i1, j1, red, grn, blu);
        line(i0, j0, i1 + W, j1, red, grn, blu);
      }
      else
        line(i0, j0, i1, j1, red, grn, blu);
    }
    else
    {
      float x2 = x0 + x1;
      float y2 = y0 + y1;
      float z2 = z0 + z1;
      float r2 = x2 * x2 + y2 * y2 + z2 * z2;
      float r = sqrtf(r2);
      x2 /= r;
      y2 /= r;
      z2 /= r;
      draw_geodesic(x0, y0, z0, x2, y2, z2, red, grn, blu, depth - 1);
      draw_geodesic(x2, y2, z2, x1, y1, z1, red, grn, blu, depth - 1);
    }
}

void draw_lines_for(int i, int red, int grn, int blu)
{
  for (int b = 0; b < B; ++b)
  {
    int j = atom[i].nb[b];
    draw_geodesic
      ( atom[i].x, atom[i].y, atom[i].z
      , atom[j].x, atom[j].y, atom[j].z
      , red, grn, blu
      , 6
      );
  }
}

void draw_point(int i, int red, int grn, int blu)
{
  float ax = atom[i].x;
  float ay = atom[i].y;
  float az = atom[i].z;
  float x0 = atan2f(ay, ax) / (2 * pi);
  x0 -= floor(x0);
  x0 *= W;
  float y0 = acosf(az) / pi * H;
  for (int dy = -R0; dy <= R0; ++dy)
  {
    int y = y0 + dy;
    float w = 1 - fabsf((y0 + dy + 0.5f) / H * 2 - 1);
    if (y < 0) continue;
    if (y >= H) continue;
```

```
      for (int dx = -R0 / w; dx <= R0 / w; ++dx)
      {
        int x = x0 + dx;
        while (x < 0) x += W;
        while (x >= W) x -= W;
        if (dx * dx * w * w + dy * dy > R0 * R0) continue;
        plot(x, y, red, grn, blu);
      }
    }
  }

  void draw_lines(int target)
  {
    // everything dark blue
    for (int i = 0; i < N; ++i)
      draw_lines_for(i, 0x00, 0x00, 0x80);
    // neighbours bright green
    for (int b = 0; b < B; ++b)
      draw_lines_for(atom[target].nb[b], 0x00, 0xFF, 0x00);
    // target yellow
    draw_lines_for(target, 0xFF, 0xFF, 0x00);
  }


  void draw_points(int target)
  {
    // everything dark blue
    for (int i = 0; i < N; ++i)
      draw_point(i, 0x00, 0x00, 0x80);
    // neighbours of neighbours bright green
    for (int b = 0; b < B; ++b)
    {
      int j = atom[target].nb[b];
      for (int c = 0; c < B; ++c)
        draw_point(atom[j].nb[c], 0x00, 0xFF, 0x00);
    }
    // neighbours yellow
    for (int b = 0; b < B; ++b)
      draw_point(atom[target].nb[b], 0xFF, 0xFF, 0x00);
    // target red
    draw_point(target, 0xFF, 0x00, 0x00);
  }

  void draw_trails(void )
  {
    const int colour[4][3] = { { 0, 0, 0 }, { 0, 0x80, 0xFF }, { 0xFF, 51, 0 }, { ↙
        ↳ 0xFF, 0xFF, 0xFF } };
    for (int i = 0; i < N; ++i)
    {
      if (i % 4 != 3) continue;
      draw_point(i, colour[i%4][0], colour[i%4][1], colour[i%4][2]);
    }
  }

  void output(FILE *vidfile)
  {
    fprintf(vidfile, "P6\n%d %d\n255\n", W, H);
```

```
    fwrite(&ppm[0][0][0], sizeof(ppm[0][0][0]) * 3 * W * H, 1, vidfile);
    fflush(vidfile);
  }

  static int global_which[4] = { 0, 1, 2, 3 };

  // entry point
  extern int main(int argc, char **argv)
  {
    (void) argv;
    int RECORD = argc > 1;

    for (int o = 0; o < 16; ++o)
      lop1_init(&output_filters[o], 1000);

    glfwInit();
    glfwWindowHint(GLFW_CLIENT_API, GLFW_OPENGL_API);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
    glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
    glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
    glfwWindowHint(GLFW_RESIZABLE, GL_FALSE);
    glfwWindowHint(GLFW_DECORATED, GL_TRUE);
    GLFWwindow *window = glfwCreateWindow(WINW, WINH, "meshwalk", 0, 0);
    glfwMakeContextCurrent(window);
    glfwSwapInterval(2);
    glfwSetKeyCallback(window, keycb);
    glewExperimental = GL_TRUE;
    glewInit();
    glGetError(); // discard common error from glew
    // set up vertex array object
    glClearColor(0, 0, 0, 1);
    GLuint vao;
    glGenVertexArrays(1, &vao);
    glBindVertexArray(vao);

    // set up framebuffer
    GLuint fbo[LOD + 1];
    glGenFramebuffers(LOD + 1, &fbo[0]);
    GLuint tex;
    glGenTextures(1, &tex);
    glActiveTexture(GL_TEXTURE1);
    glBindTexture(GL_TEXTURE_2D, tex);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA32F, 8192, 4096, 0, GL_RGBA, GL_FLOAT, ↙
        ↳ 0);
    glGenerateMipmap(GL_TEXTURE_2D);
    for (int lod = 0; lod <= LOD; ++lod)
    {
      glBindFramebuffer(GL_FRAMEBUFFER, fbo[lod]);
      glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, ↙
          ↳ tex, lod);
      GLint status = glCheckFramebufferStatus(GL_FRAMEBUFFER);
      if (status != GL_FRAMEBUFFER_COMPLETE)
        fprintf(stderr, "GL FRAMEBUFFER STATUS %d\n", status);
    }
```

```
    // set up framebuffer 2
    GLuint fbo2;
    glGenFramebuffers(1, &fbo2);
875 GLuint tex2;
    glGenTextures(1, &tex2);
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, tex2);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, 8192, 4096, 0, GL_RGB, GL_UNSIGNED_BYTE↙
        ↳ , 0);
880 glBindFramebuffer(GL_FRAMEBUFFER, fbo2);
    glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, ↙
        ↳ tex2, 0);
    GLint status = glCheckFramebufferStatus(GL_FRAMEBUFFER);
    if (status != GL_FRAMEBUFFER_COMPLETE)
      fprintf(stderr, "GL FRAMEBUFFER STATUS %d\n", status);
885
    // set up voronoi shader
    const char *s_voronoi_vertex =
      "#version 330 core\n"
      "out vec2 tc;\n"
890   "const float pi = 3.141592653;\n"
      "void main()\n"
      "{\n"
      "  if (gl_VertexID == 0) { tc = vec2( pi, -pi/2.0); gl_Position = vec4(-1.0,↙
          ↳  -1.0, 0.0, 1.0); } else\n"
      "  if (gl_VertexID == 1) { tc = vec2(-pi, -pi/2.0); gl_Position = vec4( 1.0,↙
          ↳  -1.0, 0.0, 1.0); } else\n"
895   "  if (gl_VertexID == 2) { tc = vec2( pi,  pi/2.0); gl_Position = vec4(-1.0,↙
          ↳   1.0, 0.0, 1.0); } else\n"
      "  if (gl_VertexID == 3) { tc = vec2(-pi,  pi/2.0); gl_Position = vec4( 1.0,↙
          ↳   1.0, 0.0, 1.0); } else\n"
      "  { tc = vec2(0.0); gl_Position = vec4(0.0); }\n"
      "}\n"
      ;
900 const char *s_voronoi_fragment =
      "#version 330 core\n"
      "in vec2 tc;\n"
      "uniform int N;\n"
      "uniform vec4 position[4000];\n"
905   "layout (location = 0) out vec4 colour;\n"
      "void main()\n"
      "{\n"
      "  vec3 me = vec3(cos(tc.x) * cos(tc.y), sin(tc.x) * cos(tc.y), sin(tc.y));\↙
          ↳ n"
      "  float delta = length(vec2(length(dFdx(me)), length(dFdy(me))));\n"
910   "  float maxdot = -2.0;\n"
      "  int ix = 0;\n"
      "  for (int i = 0; i < N && i < 4000; ++i)\n"
      "  {\n"
      "    float d = dot(position[i].xyz, me);\n"
915   "    if (d > maxdot) { maxdot = d; ix = i; }\n"
      "  }\n"
      "  colour = vec4(position[ix].xyz, ix);\n"
      "}\n"
      ;
920 GLuint p_voronoi = vertex_fragment_shader(s_voronoi_vertex, s_voronoi_fragment↙
        ↳ );
```

```
  glUseProgram(p_voronoi);
  u_position = glGetUniformLocation(p_voronoi, "position");
  GLint u_voronoi_N = glGetUniformLocation(p_voronoi, "N");
  glUniform1i(u_voronoi_N, N);

  // set up refinement shader
  const char *s_refine_vertex =
    "#version 330 core\n"
    "out vec4 tc;\n"
    "const float pi = 3.141592653;\n"
    "void main()\n"
    "{\n"
    "  if (gl_VertexID == 0) { tc = vec4( pi, -pi/2.0, 0.0, 0.0); gl_Position = vec4(-1.0, -1.0, 0.0, 1.0); } else\n"
    "  if (gl_VertexID == 1) { tc = vec4(-pi, -pi/2.0, 1.0, 0.0); gl_Position = vec4( 1.0, -1.0, 0.0, 1.0); } else\n"
    "  if (gl_VertexID == 2) { tc = vec4( pi,  pi/2.0, 0.0, 1.0); gl_Position = vec4(-1.0,  1.0, 0.0, 1.0); } else\n"
    "  if (gl_VertexID == 3) { tc = vec4(-pi,  pi/2.0, 1.0, 1.0); gl_Position = vec4( 1.0,  1.0, 0.0, 1.0); } else\n"
    "  { tc = vec4(0.0); gl_Position = vec4(0.0); }\n"
    "}\n"
    ;
  const char *s_refine_fragment =
    "#version 330 core\n"
    "in vec4 tc;\n"
    "uniform int R;\n"
    "uniform int lod;\n"
    "uniform sampler2D tex;\n"
    "layout (location = 0) out vec4 colour;\n"
    "void main()\n"
    "{\n"
    "  vec3 me = vec3(cos(tc.x) * cos(tc.y), sin(tc.x) * cos(tc.y), sin(tc.y));\n"
    "  ivec2 s = textureSize(tex, lod);\n"
    "  ivec2 loc = ivec2(floor(vec2(s) * tc.zw));\n"
    "  vec4 nearest = texelFetch(tex, loc, lod);\n"
    "  for (int j = -R; j <= R; ++j)\n"
    "    for (int i = -R; i <= R; ++i)\n"
    "    {\n"
    "      ivec2 c = loc + ivec2(i, j);\n"
    "      if (c.x < 0) c.x += s.x;\n"
    "      if (c.x >= s.x) c.x -= s.x;\n"
    "      if (c.y < 0) continue;\n"
    "      if (c.y >= s.y) continue;\n"
    "      vec4 test = texelFetch(tex, c, lod);\n"
    "      if (distance(me, test.xyz) < distance(me, nearest.xyz))\n"
    "        nearest = test;\n"
    "    }\n"
    "  colour = nearest;\n"
    "}\n"
  ;
  GLuint p_refine = vertex_fragment_shader(s_refine_vertex, s_refine_fragment);
  glUseProgram(p_refine);
  GLint u_refine_R = glGetUniformLocation(p_refine, "R");
  GLint u_refine_lod = glGetUniformLocation(p_refine, "lod");
  GLint u_refine_tex = glGetUniformLocation(p_refine, "tex");
```

```
  glUniform1i(u_refine_R, 2);
  glUniform1i(u_refine_tex, 1);

  // set up colourize shader
  const char *s_colour_vertex =
    "#version 330 core\n"
    "out vec4 tc;\n"
    "const float pi = 3.141592653;\n"
    "void main()\n"
    "{\n"
    "  if (gl_VertexID == 0) { tc = vec4( pi, -pi/2.0, 0.0, 0.0); gl_Position = vec4(-1.0, -1.0, 0.0, 1.0); } else\n"
    "  if (gl_VertexID == 1) { tc = vec4(-pi, -pi/2.0, 1.0, 0.0); gl_Position = vec4( 1.0, -1.0, 0.0, 1.0); } else\n"
    "  if (gl_VertexID == 2) { tc = vec4( pi,  pi/2.0, 0.0, 1.0); gl_Position = vec4(-1.0,  1.0, 0.0, 1.0); } else\n"
    "  if (gl_VertexID == 3) { tc = vec4(-pi,  pi/2.0, 1.0, 1.0); gl_Position = vec4( 1.0,  1.0, 0.0, 1.0); } else\n"
    "  { tc = vec4(0.0); gl_Position = vec4(0.0); }\n"
    "}\n"
    ;
  const char *s_colour_fragment =
    "#version 330 core\n"
    "in vec4 tc;\n"
    "uniform int R;\n"
    "uniform int lod;\n"
    "uniform sampler2D tex;\n"
    "layout (location = 0) out vec4 colour;\n"
    "void main()\n"
    "{\n"
    "  vec3 me = vec3(cos(tc.x) * cos(tc.y), sin(tc.x) * cos(tc.y), sin(tc.y));\n"
    "  float delta = length(vec2(length(dFdx(me)), length(dFdy(me))));\n"
    "  ivec2 s = textureSize(tex, lod);\n"
    "  ivec2 loc = ivec2(floor(vec2(s) * tc.zw));\n"
    "  vec4 nearest = texelFetch(tex, loc, lod);\n"
    "  vec4 nearest2 = nearest;\n"
    "  for (int j = -R; j <= R; ++j)\n"
    "    for (int i = -R; i <= R; ++i)\n"
    "    {\n"
    "      ivec2 c = loc + ivec2(i, j);\n"
    "      if (c.x < 0) c.x += s.x;\n"
    "      if (c.x >= s.x) c.x -= s.x;\n"
    "      if (c.y < 0) continue;\n"
    "      if (c.y >= s.y) continue;\n"
    "      vec4 test = texelFetch(tex, c, lod);\n"
    "      if (distance(me, test.xyz) < distance(me, nearest.xyz))\n"
    "      {\n"
    "        nearest2 = nearest;\n"
    "        nearest = test;\n"
    "      }\n"
    "      else\n"
    "      if (distance(me, test.xyz) < distance(me, nearest2.xyz))\n"
    "      {\n"
    "        nearest2 = test;\n"
    "      }\n"
    "    }\n"
```

```
1025      "  int ix = int(nearest.w);\n"
          "  int ix2 = int(nearest2.w);\n"
          "  vec3 blue = vec3(0.0, 0.5, 1.0);\n"
          "  vec3 white = vec3(1.0);\n"
          "  vec3 red = vec3(1.0, 0.2, 0.0);\n"
1030      "  vec3 black = vec3(0.0);\n"
          "  vec3 mine = vec3[4](black, blue, red, white)[ix & 3];\n"
          "  vec3 them = vec3[4](black, blue, red, white)[ix2 & 3];\n"
          "  float lmine = length(nearest.xyz - me);\n"
          "  float lthem = length(nearest2.xyz - me);\n"
1035      "  float lme = lmine - lthem;\n"
          "  colour = vec4(mix(mine, them, smoothstep(-delta, delta, lme)), 1.0);\n"
          "}\n"
          ;
        GLuint p_colour = vertex_fragment_shader(s_colour_vertex, s_colour_fragment);
1040    glUseProgram(p_colour);
        GLint u_colour_R = glGetUniformLocation(p_colour, "R");
        GLint u_colour_lod = glGetUniformLocation(p_colour, "lod");
        GLint u_colour_tex = glGetUniformLocation(p_colour, "tex");
        glUniform1i(u_colour_R, 0);
1045    glUniform1i(u_colour_lod, 0);
        glUniform1i(u_colour_tex, 1);

        SF_INFO info = { 0, SR, 16, SF_FORMAT_WAV | SF_FORMAT_FLOAT, 0, 0 };
        SNDFILE *sndfile = 0;
1050    FILE *vidfile = stdout;
        if (RECORD)
        {
          sndfile = sf_open("meshwalk-3.2_a.wav", SFM_WRITE, &info);
        }
1055    else
        {
          for (int j = 0; j < 4; ++j)
          {
            char clientname[100];
1060        snprintf(clientname, 100, "meshwalk_%d", j + 1);
            if (!(client[j] = jack_client_open(clientname, JackNoStartServer, 0))) {
              fprintf (stderr, "jack server not running?\n");
              return 1;
            }
1065        jack_set_process_callback(client[j], processcb, &global_which[j]);
            /* create ports */
            for (int i = 0; i < 16; ++i)
            {
              char portname[100];
1070          snprintf(portname, 100, "output_%d", i + 1);
              port[j][i] = jack_port_register(client[j], portname, ↙
                  ↳ JACK_DEFAULT_AUDIO_TYPE, JackPortIsOutput, 0);
            }
            /* activate audio */
            if (jack_activate(client[j])) {
1075          fprintf (stderr, "cannot activate JACK client");
              return 1;
            }
          }
        }
1080
```

```
  float frames[SR/FPS][16];
  // write titles
  if (RECORD)
  {
    for (int i = 0; i < SR/FPS; ++i)
      for (int c = 0; c < 16; ++c)
        frames[i][c] = 0;
    for (int f = 0; f < FPS * 3; ++f)
      sf_writef_float(sndfile, &frames[0][0], SR/FPS);
  }
  init();
  int dirty = 1;
  double last = glfwGetTime();
  int last_count = 0;
  memset(&ppm[0][0][0], 0x80, 8192 * 4096 * 3);
  float focus_target[3] = { 0, 0, 0 };
  LOP1 focus_filter[3];
  float fhz = 4.0f * SR / FPS;
  lop1_init(&focus_filter[0], fhz);
  lop1_init(&focus_filter[1], fhz);
  lop1_init(&focus_filter[2], fhz);
  // 1/2 = g * (1 + (1-g) + (1-g)^2 + ... + (1-g)^(H-1))
  //     = g * (1 - (1 - g)^(H-1)) / (1 - (1 - g))
  //     = 1 - (1 - g)^(H-1)
  //   g = 1 - (1/2)^(1/(H-1))
  double geiger = 1.0 - pow(0.5, 4.0 / FPS);
  for (int f = 0; ! glfwWindowShouldClose(window); ++f)
  {
    DISTANCE = 1;
    Ky = DISTANCE * 2.0f * sqrtf(4.0f / N);
    K = 1.0 / (Ky * Ky);
    sqrtK = sqrtf(K);
    if (rand() / (double) RAND_MAX < geiger)
    {
      float R;
      do
      {
        R = 0;
        for (int c = 0; c < 3; ++c)
        {
          focus_target[c] = 2.0f * (rand() / (float) RAND_MAX - 0.5f);
          R += focus_target[c] * focus_target[c];
        }
      } while (R > 1);
      R = sqrtf(R);
      float G = 0.5 * (1 - cos(2 * pi * (f - (FPS * (1 * 60 + 0))) / (FPS * (4 *↙
          ↳ 60 - 6)))) / R;
      focus_target[0] *= G;
      focus_target[1] *= G;
      focus_target[2] *= G;
    }
    focus[0] = lop1(&focus_filter[0], focus_target[0]);
    focus[1] = lop1(&focus_filter[1], focus_target[1]);
    focus[2] = lop1(&focus_filter[2], focus_target[2]);
    if (DEBUG_AMBISONICS_AXES)
    {
      int k = (f / FPS) % 6;
```

```
      const float targets[6][3] =
        { { 1, 0, 0 }, { -1, 0, 0 } // front, back
        , { 0, 1, 0 }, { 0, -1, 0 } // left, right
        , { 0, 0, 1 }, { 0, 0, -1 } // top, bottom
        };
      focus[0] = targets[k][0];
      focus[1] = targets[k][1];
      focus[2] = targets[k][2];
    }
    if (DEBUG_AMBISONICS_SPIN != 0)
    {
      float t = DEBUG_AMBISONICS_SPIN;
      float c = cos(t);
      float s = sin(t);
      float M[3][3] = { { c, s, 0 }, { -s, c, 0}, { 0, 0, 1 } };
      float nV[3][3];
      for (int i = 0; i < 3; ++i)
      for (int j = 0; j < 3; ++j)
      {
        nV[i][j] = 0;
        for (int k = 0; k < 3; ++k)
          nV[i][j] += M[i][k] * V[k][j];
      }
      for (int i = 0; i < 3; ++i)
      for (int j = 0; j < 3; ++j)
        V[i][j] = nV[i][j];
    }
    double now = glfwGetTime();
    if (now - last > 1)
    {
      fprintf(stderr, "\n%f fps\n%f max\n", (f - last_count) / (now - last), ↙
          ↳ sqrtf(metric) / Ky);
      last_count = f;
      last = now;
    }
    if (RECORD && f >= FPS * (60 * 5 - 6))
      break;

    step();

    if ((RECORD && f >= FPS * 60 * 1) || ! RECORD)
    {
#if RECORD_VIDEO

      // voronoi
      glUseProgram(p_voronoi);
      upload_uniforms();
      glBindFramebuffer(GL_FRAMEBUFFER, fbo[LOD]);
      glViewport(0, 0, 8192 >> LOD, 4096 >> LOD);
      glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);

      // refine
      glUseProgram(p_refine);
      for (int lod = LOD; lod >= 1; --lod)
      {
        glBindFramebuffer(GL_FRAMEBUFFER, fbo[lod - 1]);
        glViewport(0, 0, 8192 >> (lod - 1), 4096 >> (lod - 1));
```

```
        glUniform1i(u_refine_lod, lod);
        glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);
    }

    // colourize
    glUseProgram(p_colour);
    glBindFramebuffer(GL_FRAMEBUFFER, fbo2);
    glViewport(0, 0, 8192, 4096);
    glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);

    if (RECORD && f >= FPS * 60 * 1 && DRAW_VORONOI)
    {
      dirty = 0;
      glReadPixels(0, 0, 8192, 4096, GL_RGB, GL_UNSIGNED_BYTE, &ppm[0][0][0]);
    }
    else if (dirty && DRAW_CLEAR)
    {
      dirty = 0;
      memset(&ppm[0][0][0], 0, 8192 * 4096 * 3);
    }
    int target = find_target(1, 0, 0);
    if (DRAW_LINES)
    {
      dirty = 1;
      draw_lines(target);
    }
    if (DRAW_POINTS)
    {
      dirty = 1;
      draw_points(target);
    }
    if (DRAW_TRAILS)
    {
      dirty = 1;
      draw_trails();
    }
    // preview
    glBindFramebuffer(GL_DRAW_FRAMEBUFFER, 0);
    glClear(GL_COLOR_BUFFER_BIT);
    glBlitFramebuffer
      ( 0, 0, 8192, 4096
      , 0, (WINH - WINW/2) / 2, WINW, (WINH + WINW/2) / 2
      , GL_COLOR_BUFFER_BIT, GL_LINEAR
      );
    // redisplay
    glfwSwapBuffers(window);
    glfwPollEvents();
    GLuint e;
    while ((e = glGetError())) fprintf(stderr, "GL ERROR %d\n", e);
#endif

    // output PPM to stdout (flipped vertically)
    if (RECORD && f >= FPS * 60 * 1)
    {
#if RECORD_VIDEO
      fprintf(vidfile, "P6\n%d %d\n255\n", 8192, 4096);
      fwrite(&ppm[0][0][0], 8192 * 4096 * 3, 1, vidfile);
```

```
1250          if (f == FPS * 60 * 1)
              {
                FILE *imgfile = popen("pnmtopng -force -interlace -compression 9 > ↙
                    ↳ meshwalk-3.2.png", "w");
                fprintf(imgfile, "P6\n%d %d\n255\n", 8192, 4096);
                fwrite(&ppm[0][0][0], 8192 * 4096 * 3, 1, imgfile);
1255            pclose(imgfile);
              }
      #endif
      #if RECORD_AUDIO
              float audio[16][SR/FPS];
1260          float *audiop[16];
              for (int c = 0; c < 16; ++c)
                audiop[c] = &audio[c][0];
              audiocb(audiop, SR/FPS, f * SR/FPS, -1);
              for (int i = 0; i < SR/FPS; ++i)
1265            for (int c = 0; c < 16; ++c)
                  frames[i][c] = audio[c][i];
              sf_writef_float(sndfile, &frames[0][0], SR/FPS);
      #endif
            }
1270      }
        }
        if (RECORD)
        {
          // write credits
1275      for (int i = 0; i < SR/FPS; ++i)
            for (int c = 0; c < 16; ++c)
              frames[i][c] = 0;
          for (int f = 0; f < FPS * 3; ++f)
            sf_writef_float(sndfile, &frames[0][0], SR/FPS);
1280      sf_close(sndfile);
        }
        glfwDestroyWindow(window);
        glfwTerminate();
        return 0;
1285    }
```

72 meshwalk/meshwalk-3.2.sh

```
      #!/bin/bash
      set -e
      make meshwalk-3.2
      (
 5      for i in $(seq 180)
        do
          pngtopnm < meshwalk-3.2-title.png
        done |
        ffmpeg -r 60 -i - -vf fade=in:30:30,fade=out:120:30 -f image2pipe -codec:v ppm↙
            ↳ -
10      ./meshwalk-3.2 -nrt
        for i in $(seq 180)
        do
          pngtopnm < meshwalk-3.2-credits.png
        done |
15      ffmpeg -r 60 -i - -vf fade=in:30:30,fade=out:120:30 -f image2pipe -codec:v ppm↙
            ↳ -
```

```
) |
ffmpeg -framerate 60 -i - -codec:v png -y meshwalk-3.2_v.mkv
ffmpeg -i meshwalk-3.1_v.mkv -i meshwalk-3.2_a.wav \
  -codec:v copy -codec:a copy \
  -y meshwalk-3.2.mkv
ffmpeg -i meshwalk-3.2.mkv -af pan="4.0|c0=c0|c1=c1|c2=c2|c3=c3" \
  -pix_fmt yuv420p -profile:v high -level:v 5.1 -tune:v animation \
  -crf:v 20 -b:a 512k -movflags +faststart \
  -y meshwalk-3.2_m.mp4
python2.7 ../../../github.com/google/spatial-media/spatialmedia \
  -i --spatial-audio meshwalk-3.2_m.mp4 meshwalk-3.2.mp4
```

73 README

```
dr1 -- a collection of ambient drone works
```

74 stringthing/Makefile

```
stringthing: stringthing.c
	gcc -std=c99 -Wall -pedantic -Wextra -Wno-unused-parameter -O3 -march=native stringthing.c -o stringthing -lm -ljack -lglut -lGLEW -lGL -lGLU
```

75 stringthing/stringthing.c

```
/*
    stringthing -- pseudo-generative spinning string drone physical model
    Copyright (C) 2011-10-18 Claude Heiland-Allen <claude@mathr.co.uk>

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU 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 <http://www.gnu.org/licenses/>.

*/

#include <ctype.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <jack/jack.h>
#include <GL/glew.h>
#include <GL/glut.h>

#define PI 3.141592653589793
#define DT 0.0625
#define FRICTION 0.9999
```

```
#define GRAVITY 2

static double SR = 48000;
static double HZ;
#define MINN 32
#define MAXN 2048
static int N = 1024;

struct elem {
  double x, y, z, dx, dy, dz;
};
static struct elem string[2][MAXN];
static volatile int w = 0;
static jack_default_audio_sample_t buf[MAXN][2];

static jack_client_t *client;
static jack_port_t *port[2];

static int phase = -1;
static int t = 0;
static double f1 = 0;
static double f2 = 0;
static double p1 = 0;
static double p2 = 0;
static double lo = 0;
static double lh = 0;
static double lb = 0;
static double ro = 0;
static double rh = 0;
static double rb = 0;

static float vstring[MAXN][4];
static int ufudge = 0;
static int uangle = 0;

static int width = 512;
static int height = 288;
static int fullscreen = 0;

static void displaycb(void) {
  int s = w;
  float tot = 0;
  for (int i = 0; i < N; ++i) {
    vstring[i][0] = string[s][i].dx; tot += string[s][i].dx * string[s][i].dx;
    vstring[i][1] = string[s][i].dy; tot += string[s][i].dy * string[s][i].dy;
    vstring[i][2] = string[s][i].dz; tot += string[s][i].dz * string[s][i].dz;
    vstring[i][3] = 0.0;
  }
  tot = sqrt(tot / (3 * N));
  for (int i = 0; i < N; ++i) {
    vstring[i][0] /= tot;
    vstring[i][1] /= tot;
    vstring[i][2] /= tot;
    vstring[i][3] /= tot;
  }
  glTexSubImage1D(GL_TEXTURE_1D, 0, 0, N, GL_RGBA, GL_FLOAT, vstring);
  glUniform1fARB(ufudge, (N - 1) / (float) MAXN);
```

```
  glUniform1fARB(uangle, atan2(vstring[N-1][1], vstring[N-1][0]));
  glBegin(GL_QUADS); {
    float a = width * 1.0 / height;
    float f = N * 1.25 / MAXN;
    float x = a * f;
    float y =     f;
    glTexCoord2f( x, -y); glVertex2f(1, 0);
    glTexCoord2f( x,  y); glVertex2f(1, 1);
    glTexCoord2f(-x,  y); glVertex2f(0, 1);
    glTexCoord2f(-x, -y); glVertex2f(0, 0);
  } glEnd();
  glutSwapBuffers();
  glutReportErrors();
}

static void reshapecb(int w, int h) {
  width = w;
  height = h;
  glViewport(0, 0, width, height);
  glLoadIdentity();
  gluOrtho2D(0, 1, 1, 0);
  glutPostRedisplay();
}

static void timercb(int v) {
  glutPostRedisplay();
  glutTimerFunc(20, timercb, v);
}

static double f1s[256];
static double f2s[256];
static void keyboard1cb(unsigned char key, int x, int y) {
  if (isalpha(key)) {
    if (isupper(key)) {
      f1s[tolower(key)] = f1;
      f2s[tolower(key)] = f2;
    } else {
      f1 = f1s[key];
      f2 = f2s[key];
    }
  } else {
    switch (key) {
    case ' ': { double temp = f1; f1 = f2; f2 = -temp; break; }
    }
  }
}

static void keyboard2cb(int key, int x, int y) {
  switch (key) {
  case GLUT_KEY_F11:
    fullscreen = !fullscreen;
    if (fullscreen) {
      glutFullScreen();
      glutSetCursor(GLUT_CURSOR_NONE);
    } else {
      glutReshapeWindow(512, 288);
      glutSetCursor(GLUT_CURSOR_INHERIT);
```

```
      }
      break;
    }
  }

static void mousecb(int x, int y) {
  double f = (x - width  / 2) * 2.0 / width;
  double d = (height / 2 - y) * 4.0 / height;
  double m = (pow(2, fabs(f)) - 1) * (f < 0 ? -1 : 1);
  double r = pow(2, d);
  f1 = (m * r) / 8.0;
  f2 = (m / r) / 8.0;
}

static double ls = 0;

static int processcb(jack_nframes_t nframes, void *arg) {
  jack_default_audio_sample_t *out[2];
  out[0] = (jack_default_audio_sample_t *)
    jack_port_get_buffer(port[0], nframes);
  out[1] = (jack_default_audio_sample_t *)
    jack_port_get_buffer(port[1], nframes);
  for (unsigned int k = 0; k < nframes; ++k, ++phase) {
    if (phase >= 2 * N) {
      phase = 0;
    }
    if (phase == 0) {
      double s = 0;
      for (int i = 0; i < N; ++i) {
        s += string[w][i].dx * string[w][i].dx
          +  string[w][i].dy * string[w][i].dy
          +  string[w][i].dz * string[w][i].dz;
      }
      s = 4 * sqrt(s / N);
      ls = 0.99 * ls + 0.01 * s;
      if (!(ls > 0)) { ls = 1; }
      double a0 = -atan2(string[w][N-1].dy, string[w][N-1].dx);
      double s0 = sin(a0);
      double c0 = cos(a0);
      for (int j = 0; j < 2 * N; ++j) {
        int i = j >= N ? 2 * N - 1 - j : j;
        double x =  c0 * string[w][i].dx + s0 * string[w][i].dy;
        double y = -s0 * string[w][i].dx + c0 * string[w][i].dy;
        double lv = x * x + string[w][i].dz * string[w][i].dz;
        double rv = y * y + string[w][i].dz * string[w][i].dz;
        double wd = (1 - cos(j * PI / N)) / 2;
        lv = sqrt(lv) / ls;
        rv = sqrt(rv) / ls;
        lo = 0.999 * lo + 0.001 * lv;
        ro = 0.999 * ro + 0.001 * rv;
        lh = lv - lo;
        rh = rv - ro;
        lb = tanh(0.5 * lb + 0.5 * wd * lh);
        rb = tanh(0.5 * rb + 0.5 * wd * rh);
        buf[j][0] = tanh(2 * lb);
        buf[j][1] = tanh(2 * rb);
      }
```

```
      double a1 = 1 / (1 + f1 * f2);
      double a2 = 1 / (1 + f1 * f2);
      for (int tt = 0; tt < 256; ++tt) {
        p1 += f1 * N / SR;
        p2 += f2 * N / SR;
        while (p1 > 1) { p1 -= 1; } while (p1 < 0) { p1 += 1; }
        while (p2 > 1) { p2 -= 1; } while (p2 < 0) { p2 += 1; }
        {
          double fx = a1 * cos(2 * PI * p1) - a2 * sin(2 * PI * p2);
          double fy = a1 * sin(2 * PI * p1) + a2 * cos(2 * PI * p2);
          double fz = 0;
          string[1-w][0].dx = FRICTION * string[w][0].dx + fx * DT;
          string[1-w][0].dy = FRICTION * string[w][0].dy + fy * DT;
          string[1-w][0].dz = FRICTION * string[w][0].dz + fz * DT;
          string[1-w][0].x = string[w][0].x + string[1-w][0].dx * DT;
          string[1-w][0].y = string[w][0].y + string[1-w][0].dy * DT;
          string[1-w][0].z = string[w][0].z + string[1-w][0].dz * DT;
        }
        for (int i = 1; i < N - 1; ++i) {
          double fx = string[w][i-1].x + string[w][i+1].x - 2 * string[w][i].x;
          double fy = string[w][i-1].y + string[w][i+1].y - 2 * string[w][i].y;
          double fz = string[w][i-1].z + string[w][i+1].z - 2 * string[w][i].z
                      - GRAVITY;
          string[1-w][i].dx = FRICTION * string[w][i].dx + fx * DT;
          string[1-w][i].dy = FRICTION * string[w][i].dy + fy * DT;
          string[1-w][i].dz = FRICTION * string[w][i].dz + fz * DT;
          string[1-w][i].x = string[w][i].x + string[1-w][i].dx * DT;
          string[1-w][i].y = string[w][i].y + string[1-w][i].dy * DT;
          string[1-w][i].z = string[w][i].z + string[1-w][i].dz * DT;
        }
        for (int i = N - 1; i < N; ++i) {
          double fx = string[w][i-1].x - string[w][i].x;
          double fy = string[w][i-1].y - string[w][i].y;
          double fz = string[w][i-1].z - string[w][i].z - GRAVITY;
          string[1-w][i].dx = FRICTION * string[w][i].dx + fx * DT;
          string[1-w][i].dy = FRICTION * string[w][i].dy + fy * DT;
          string[1-w][i].dz = FRICTION * string[w][i].dz + fz * DT;
          string[1-w][i].x = string[w][i].x + string[1-w][i].dx * DT;
          string[1-w][i].y = string[w][i].y + string[1-w][i].dy * DT;
          string[1-w][i].z = string[w][i].z + string[1-w][i].dz * DT;
        }
        w = 1 - w;
      }
      ++t;
    }
    out[0][k] = buf[phase][0];
    out[1][k] = buf[phase][1];
  }
  return 0;
}

static int sratecb(jack_nframes_t nframes, void *arg) {
  SR = nframes;
  N = fmin(fmax(SR / HZ, MINN), MAXN);
  return 0;
}
```

```
260 static void errorcb(const char *desc) {
      fprintf(stderr, "JACK error: %s\n", desc);
    }

    static void shutdowncb(void *arg) {
265   exit(1);
    }

    static void exitcb(void) {
      jack_client_close(client);
270 }

    static const char *src =
    "uniform sampler1D tex;\n"
    "uniform sampler2D palette;\n"
275 "uniform float fudge;\n"
    "uniform float angle;\n"
    "void main() {\n"
    "  vec2 q = gl_TexCoord[0].xy;\n"
    "  float l = length(q);\n"
280 "  float a = atan(q.y, q.x) + angle;\n"
    "  float s = sin(a);\n"
    "  float c = cos(a);\n"
    "  mat2 m = mat2(c, s, -s, c);\n"
    "  float r = clamp(l, 0.0, fudge);\n"
285 "  vec2 p = texture1D(tex, r).rg;\n"
    "  vec4 rgba = texture2D(palette, p * m * 0.125 + vec2(0.5));\n"
    "  rgba.rgb *= pow(clamp(r / l, 0.0, 1.0), 2.0);\n"
    "  gl_FragColor = rgba;\n"
    "}\n";
290
    int main(int argc, char **argv) {
      HZ = 0;
      if (argc > 1) {
        HZ = atof(argv[1]);
295   }
      if (! HZ) { HZ = 96000.0 / 1024.0; }
      memset(string, 0, 2 * MAXN * sizeof(struct elem));
      memset(buf, 0, 2 * MAXN * sizeof(jack_default_audio_sample_t));
      for (int i = 0; i < N; ++i) {
300     string[w][i].z = -i;
      }
      memset(f1s, 0, 256 * sizeof(double));
      memset(f2s, 0, 256 * sizeof(double));
      /* initialize OpenGL */
305   glutInitWindowSize(width, height);
      glutInit(&argc, argv);
      glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE);
      glutCreateWindow("stringthing");
      glewInit();
310   glShadeModel(GL_FLAT);
      GLuint texp;
      glActiveTexture(GL_TEXTURE1);
      glGenTextures(1, &texp);
      glEnable(GL_TEXTURE_2D);
315   glBindTexture(GL_TEXTURE_2D, texp);
      unsigned char *bufp = malloc(512 * 512 * 4);
```

```
  double gamma = 1;
  for (int j = 0; j < 512; ++j) {
    for (int i = 0; i < 512; ++i) {
      double x = (i - 256) / 64.0;
      double y = (j - 256) / 64.0;
      double h = atan2(y, x) * 0.15915494309189535;
      double s = fmin(0.25 * sqrt(x * x + y * y), 1.0);
      double v = 0.0625 * (x * x + y * y);
      h -= floor(h);
      h *= 6;
      int k = floor(h);
      double f = h - k;
      double p = v * (1 - s);
      double q = v * (1 - s * f);
      double t = v * (1 - s * (1 - f));
      double r, g, b;
      switch (k) {
      case  0: r = v; g = t; b = p; break;
      case  1: r = q; g = v; b = p; break;
      case  2: r = p; g = v; b = t; break;
      case  3: r = p; g = q; b = v; break;
      case  4: r = t; g = p; b = v; break;
      default: r = v; g = p; b = q; break;
      }
      double z = 0.5 * s / (0.001 + 0.299 * r + 0.587 * g + 0.114 * b);
      double w = pow(256, 1.0/gamma) * 256 * z / (1 + pow(fmax(x * x + y * y, ↙
          ↪ 16), 2));
      bufp[(512 * j + i)*4 + 0] = fmin(fmax(pow(w * r, gamma), 0), 255);
      bufp[(512 * j + i)*4 + 1] = fmin(fmax(pow(w * g, gamma), 0), 255);
      bufp[(512 * j + i)*4 + 2] = fmin(fmax(pow(w * b, gamma), 0), 255);
      bufp[(512 * j + i)*4 + 3] = 1;
    }
  }
  glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, 512, 512, 0, GL_RGBA, GL_UNSIGNED_BYTE↙
      ↪ , bufp);
  free(bufp);
  glGenerateMipmap(GL_TEXTURE_2D);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
  glActiveTexture(GL_TEXTURE0);
  GLuint tex;
  glGenTextures(1, &tex);
  glEnable(GL_TEXTURE_1D);
  glBindTexture(GL_TEXTURE_1D, tex);
  glTexImage1D(GL_TEXTURE_1D, 0, GL_RGBA32F_ARB, MAXN, 0, GL_RGBA, GL_FLOAT, 0);
  glTexParameteri(GL_TEXTURE_1D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
  glTexParameteri(GL_TEXTURE_1D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);

  /* compile shaders */
  GLint success;
  int prog = glCreateProgramObjectARB();
  int frag = glCreateShaderObjectARB(GL_FRAGMENT_SHADER_ARB);
  glShaderSourceARB(frag, 1, (const GLcharARB **) &src, 0);
  glCompileShaderARB(frag);
  glAttachObjectARB(prog, frag);
  glLinkProgramARB(prog);
  glGetObjectParameterivARB(prog, GL_OBJECT_LINK_STATUS_ARB, &success);
```

```
  if (!success) exit(1);
  glUseProgramObjectARB(prog);
  glUniform1iARB(glGetUniformLocationARB(prog, "tex"), 0);
  glUniform1iARB(glGetUniformLocationARB(prog, "palette"), 1);
  ufudge = glGetUniformLocationARB(prog, "fudge");
  uangle = glGetUniformLocationARB(prog, "angle");

  /* set up callbacks */
  glutDisplayFunc(displaycb);
  glutReshapeFunc(reshapecb);
  glutMotionFunc(mousecb);
  glutPassiveMotionFunc(mousecb);
  glutKeyboardFunc(keyboard1cb);
  glutSpecialFunc(keyboard2cb);
  glutTimerFunc(20, timercb, 0);

  /* initialize JACK */
  jack_set_error_function(errorcb);
  if (!(client = jack_client_open("stringthing", 0, 0))) {
    return 1;
  }
  atexit(exitcb);
  jack_set_process_callback(client, processcb, 0);
  jack_set_sample_rate_callback(client, sratecb, 0);
  jack_on_shutdown(client, shutdowncb, 0);
  port[0] = jack_port_register(
    client, "output_1", JACK_DEFAULT_AUDIO_TYPE, JackPortIsOutput, 0
  );
  port[1] = jack_port_register(
    client, "output_2", JACK_DEFAULT_AUDIO_TYPE, JackPortIsOutput, 0
  );
  if (jack_activate(client)) {
    fprintf (stderr, "cannot activate JACK client");
    return 1;
  }
  const char **ports;
  if ((ports = jack_get_ports(
    client, NULL, NULL, JackPortIsPhysical | JackPortIsInput
  ))) {
    int i = 0;
    while (ports[i] && i < 2) {
      if (jack_connect(
        client, jack_port_name(port[i]), ports[i]
      )) {
        fprintf(stderr, "cannot connect output port\n");
      }
      i++;
    }
    free(ports);
  }

  glutMainLoop();
  return 0;
}
```

76 temple/.gitignore

temple

77 temple/Makefile

```
temple: temple.c
	gcc -O3 -std=c99 -Wall -Wextra -pedantic -o temple temple.c -lm -ljack

clean:
	-rm temple

.PHONY: clean
```

78 temple/README

```
port of a pd+gridflow patch from 2010

make

ffmpeg -i somevideofile -f yuv4mpegpipe - | ./temple
```

79 temple/temple.c

```
#define _DEFAULT_SOURCE

#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <jack/jack.h>

static const float pif = 3.141592653589793f;

struct vector {
  float v[3];
};

#define BUCKETS 8

struct histogram {
  float h[BUCKETS];
};

static void clear(struct histogram *h) {
  for (int i = 0; i < BUCKETS; ++i) {
    h->h[i] = 0.0f;
  }
}

static void accum(struct histogram *h, const struct vector *v) {
  int b = ((int) floor(BUCKETS/2.0f * (atan2f(v->v[1], v->v[2]) / pif + 1.0f)) +
      ↪ BUCKETS) % BUCKETS;
  h->h[b] += v->v[0];
}
```

```
static void histy4m(struct histogram *h, const unsigned char *y, const unsigned ↙
    ↳ char *u, const unsigned char *v, int width, int height) {
  clear(h);
  for (int j = 0; j < height/2; ++j) {
    for (int i = 0; i < width/2; ++i) {
      struct vector p;
      p.v[0] = 0.25f * (y[2*j*width + 2*i] + y[2*j*width + 2*i+1] + y[(2*j+1)*↙
          ↳ width + 2*i] + y[(2*j+1)*width + 2*i+1]);
      p.v[1] = u[j*width/2 + i] - 128.0f;
      p.v[2] = v[j*width/2 + i] - 128.0f;
      accum(h, &p);
    }
  }
  for (int i = 0; i < BUCKETS; ++i) {
    h->h[i] /= (width * height * 256.0);
  }
}

#define TABSIZE 1024

struct borgan {
  float table[2][TABSIZE];
  int read[2];
  int write;
  int which;
  float env[TABSIZE];
  float hipcoef;
  float hip[2];
  float lopcoef;
  float lop[2];
  float phase;
  float inc;
};

void borgan_init(struct borgan *b, float sr, float f) {
  memset(b, 0, sizeof(struct borgan));
  for (int i = 0; i < TABSIZE; ++i) {
    b->env[i] = sinf(i * pif / TABSIZE);
  }
  b->hipcoef = 1.0f - 10.0f * 2.0f * pif / sr;
  b->lopcoef = 1000.0f * 2.0f * pif / sr;
  b->inc = TABSIZE * f / sr;
}

float borgan_dsp(struct borgan *b, float vol) {
  float osc1 = b->table[b->which][b->read[0]];
  float osc2 = b->table[b->which][b->read[1]];
  float osc = tanhf(1.5f * (osc1 - osc2 + 0.25f * (rand() / (float) RAND_MAX * ↙
      ↳ 2.0f - 1.0f)));
  float n = 0.5f * (1.0f + b->hipcoef);
  float t = osc + b->hipcoef * b->hip[0];
  osc = n * (t - b->hip[0]);
  b->hip[0] = t;
  t = osc + b->hipcoef * b->hip[1];
  osc = n * (t - b->hip[1]);
  b->hip[1] = t;
  n = 1.0f - b->lopcoef;
```

```
  b->lop[0] = osc = b->lopcoef * osc + n * b->lop[0];
  b->lop[1] = osc = b->lopcoef * osc + n * b->lop[1];
  b->table[1 - b->which][b->write] = b->env[b->write] * osc;
  int i = floorf(b->phase);
  float x = b->phase - i;
  float p = b->table[b->which][(i - 1) & (TABSIZE - 1)];
  float q = b->table[b->which][(i    ) & (TABSIZE - 1)];
  float r = b->table[b->which][(i + 1) & (TABSIZE - 1)];
  float s = b->table[b->which][(i + 2) & (TABSIZE - 1)];
  float out = 0.5f * ((-x*x*x+2.0f*x*x-x) * p + (3.0f*x*x*x-5.0f*x*x+2.0f) * q +↙
      ↳ (-3.0f*x*x*x+4.0f*x*x+x) * r + (x*x*x-x*x) * s);
  b->phase += b->inc;
  if (b->phase >= TABSIZE) {
    b->phase -= TABSIZE;
  }
  b->read[0] = (b->read[0] + 1) & (TABSIZE - 1);
  b->read[1] = (b->read[1] - 2) & (TABSIZE - 1);
  b->write = (b->write + 1) & (TABSIZE - 1);
  if (! b->write) {
    b->which = 1 - b->which;
  }
  return out * vol;
}

struct dsp {
  struct borgan b[2][BUCKETS];
};

void dsp_init(struct dsp *d, float sr) {
  for (int i = 0; i < BUCKETS; ++i) {
    borgan_init(&d->b[0][i], sr, 30.0f * (i + 1));
    borgan_init(&d->b[1][i], sr, 30.0f * (i + 1));
  }
}

void dsp_dsp(struct dsp *d, const struct histogram *h, float *oleft, float *↙
    ↳ oright) {
  float o[2] = { 0.0f, 0.0f };
  for (int i = 0; i < BUCKETS; ++i) {
    o[0] += borgan_dsp(&d->b[0][i], h->h[i]);
    o[1] += borgan_dsp(&d->b[1][i], h->h[i]);
  }
  *oleft = o[0];
  *oright = o[1];
}

struct dsp dsp;
int which = 0;
struct histogram histogram[2];

static jack_client_t *client;
static jack_port_t *port[2];

static int sratecb(jack_nframes_t sr, void *arg) {
  (void) arg;
  dsp_init(&dsp, sr);
  return 0;
```

```
}

static void errorcb(const char *desc) {
  fprintf(stderr, "JACK error: %s\n", desc);
}

static void shutdowncb(void *arg) {
  (void) arg;
  exit(1);
}

static void exitcb(void) {
  jack_client_close(client);
}

static int processcb(jack_nframes_t nframes, void *arg) {
  (void) arg;
  jack_default_audio_sample_t *out[2];
  out[0] = (jack_default_audio_sample_t *)
    jack_port_get_buffer(port[0], nframes);
  out[1] = (jack_default_audio_sample_t *)
    jack_port_get_buffer(port[1], nframes);
  for (unsigned int k = 0; k < nframes; ++k) {
    float l, r;
    dsp_dsp(&dsp, &histogram[which], &l, &r);
    out[0][k] = l;
    out[1][k] = r;
  }
  return 0;
}

int main(int argc, char **argv) {
  (void) argc;
  (void) argv;
  for (int i = 0; i < BUCKETS; ++i) {
    histogram[0].h[i] = 1.0 / BUCKETS;
    histogram[1].h[i] = 1.0 / BUCKETS;
  }
  jack_set_error_function(errorcb);
  if (!(client = jack_client_open("temple", 0, 0))) {
    return 1;
  }
  atexit(exitcb);
  jack_set_process_callback(client, processcb, 0);
  jack_set_sample_rate_callback(client, sratecb, 0);
  jack_on_shutdown(client, shutdowncb, 0);
  port[0] = jack_port_register(
    client, "output_1", JACK_DEFAULT_AUDIO_TYPE, JackPortIsOutput, 0
  );
  port[1] = jack_port_register(
    client, "output_2", JACK_DEFAULT_AUDIO_TYPE, JackPortIsOutput, 0
  );
  if (jack_activate(client)) {
    fprintf (stderr, "cannot activate JACK client");
    return 1;
  }
  const char **ports;
```

```
  if ((ports = jack_get_ports(
200     client, NULL, NULL, JackPortIsPhysical | JackPortIsInput
  ))) {
    int i = 0;
    while (ports[i] && i < 2) {
      if (jack_connect(
205       client, jack_port_name(port[i]), ports[i]
      )) {
        fprintf(stderr, "cannot connect output port\n");
      }
      i++;
210   }
    free(ports);
  }
  int width, height;
  FILE *f = stdin;
215 fscanf(f, "YUV4MPEG2 W%d H%d F25:1 Ip A1:1 C420mpeg2 XYSCSS=420MPEG2\n", &↙
      ↳ width, &height);
  unsigned char *ybuf = malloc(width * height);
  unsigned char *ubuf = malloc(width * height / 4);
  unsigned char *vbuf = malloc(width * height / 4);
  while (EOF != fscanf(f, "FRAME\n")) {
220   putchar('.');
    fflush(stdout);
    fread(ybuf, width * height, 1, f);
    fread(ubuf, width * height / 4, 1, f);
    fread(vbuf, width * height / 4, 1, f);
225   histy4m(&histogram[1 - which], ybuf, ubuf, vbuf, width, height);
    which = 1 - which;
    usleep(40000);
  }
  free(ybuf);
230 free(ubuf);
  free(vbuf);
  return 0;
}
```