incidents

Claude Heiland-Allen

2011–2019

Contents

1 code/boat.c

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

```c
#include <cairo/cairo.h>

#include "boat.h"
#include "boat.frag.c"
#include "boat.vert.c"

struct boat *boat_init(struct boat *boat) {
  if (! boat) { return 0; }
  if (! shader_init(&boat->shader, boat_vert, boat_frag)) { return 0; }
  shader_uniform(boat, tex);
  shader_uniform(boat, boatX);
  shader_uniform(boat, amplitude);
  shader_uniform(boat, frequency);
  shader_uniform(boat, level);
  shader_uniform(boat, phase);
  shader_uniform(boat, size);
  shader_uniform(boat, stroke);
  shader_uniform(boat, fill);
  boat->value.tex = 0;
  boat->value.boatX = 0;
  boat->value.amplitude = 0;
  boat->value.frequency = 0;
  boat->value.level = 0;
  boat->value.phase = 0;
  boat->value.size = 1;
  boat->value.stroke.v[0] = 0;
  boat->value.stroke.v[1] = 0;
  boat->value.stroke.v[2] = 0;
  boat->value.stroke.v[3] = 1;
  boat->value.fill.v[0] = 1;
  boat->value.fill.v[1] = 1;
  boat->value.fill.v[2] = 1;
  boat->value.fill.v[3] = 1;
  glGenTextures(1, &boat->tex);
  glBindTexture(GL_TEXTURE_2D, boat->tex);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
  for (int size = 4096, level = 0; size > 0; size >>= 1, level += 1) {
    cairo_surface_t *surface = cairo_image_surface_create (CAIRO_FORMAT_ARGB32, size, size);
    cairo_t *cr = cairo_create(surface);
    cairo_set_source_rgba(cr, 0, 0, 0, 0);
    cairo_paint(cr);
    if (size > 16) {
      cairo_save(cr);
      cairo_translate(cr, size/2.0, size/2.0);
      cairo_scale(cr, size/128.0, size/128.0);
#define M(x,y) cairo_move_to(cr, x, y);
#define L(x,y) cairo_line_to(cr, x, y);
#define Z cairo_close_path(cr);
      M(-18,-5)L(-2,-5)L(-2,-27)L(14,-15)L(-2,-7)L(-2,-5)L(20,-5)L(12,3)L(-12,3)L(-18,-5)Z
#undef M
#undef L
```

```
  #undef Z
      cairo_restore(cr);
      cairo_set_line_width(cr, 4);
      cairo_set_source_rgba(cr, 1, 1, 1, 1);
      cairo_fill_preserve(cr);
      cairo_set_source_rgba(cr, 0, 0, 0, 1);
      cairo_stroke(cr);
    }
    cairo_destroy(cr);
    cairo_surface_flush(surface);
    unsigned char *pixels = cairo_image_surface_get_data(surface);
    int stride = cairo_image_surface_get_stride(surface);
    if (stride != size * 4) {
      exit(42);
    }
    glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
    glTexImage2D(GL_TEXTURE_2D, level, GL_RGBA, size, size, 0, GL_RGBA, ↙
        ↳ GL_UNSIGNED_BYTE, pixels);
    cairo_surface_destroy(surface);
  }
  glBindTexture(GL_TEXTURE_2D, 0);
  return boat;
}

void boat_use(struct boat *boat) {
  if (boat) {
    glUseProgram(boat->shader.program);
    glBindTexture(GL_TEXTURE_2D, boat->tex);
    shader_updatei(boat, tex);
    shader_updatef(boat, boatX);
    shader_updatef(boat, amplitude);
    shader_updatef(boat, frequency);
    shader_updatef(boat, level);
    shader_updatef(boat, phase);
    shader_updatef(boat, size);
    shader_updatef4(boat, stroke);
    shader_updatef4(boat, fill);
  } else {
    glBindTexture(GL_TEXTURE_2D, 0);
    glUseProgram(0);
  }
}
```

2 code/boat.frag

```
/* boat image */
uniform sampler2D tex;

/* boat position */
uniform float boatX;

/* wave shape */
uniform float amplitude;
uniform float frequency;
uniform float level;
uniform float phase;
```

```
  /* boat size */
  uniform float size;

  /* appearance */
  uniform vec4 stroke;
  uniform vec4 fill;

  const vec4 transparent = vec4(0.0);

  void main(void) {
    vec2 p = gl_TexCoord[0].xy;
    float t = frequency * boatX + phase;
    float boatY = amplitude * sin(t) + level;
    float boatT = atan(amplitude * frequency * cos(t), 2.0);
    mat2 boatM = mat2(cos(boatT), sin(boatT), sin(boatT), -cos(boatT));
    vec2 q = boatM * (p - vec2(boatX, boatY));
    vec4 c = texture2D(tex, size * q + vec2(0.5));
    gl_FragColor = vec4(mix(stroke.rgb, fill.rgb, c.rgb / (c.a > 0.0 ? c.a : 1.0))
        , c.a);
  }
```

3 code/boat.h

```
  #ifndef BOAT_H
  #define BOAT_H 1

  #include "shader.h"
  #include "vector.h"

  struct boat { struct shader shader;
    struct { GLint tex, boatX, amplitude, frequency, level, phase, size,
        stroke, fill; } uniform;
    struct { float tex, boatX, amplitude, frequency, level, phase, size; struct
        vec4 stroke, fill; } value;
    GLuint tex;
  };

  struct boat *boat_init(struct boat *boat);
  void boat_use(struct boat *boat);

  #endif
```

4 code/boat.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="64" height="64" viewbox="0 0 64 64" version="1.1"
  ><path transform="translate(31,31)"
    d="M -18,-5 L -2,-5 -2,-27 14,-15 -2,-7 -2,-5  20,-5 12,3 -12,3 -18,-5 Z"
    stroke-linejoin="round" stroke-width="0.5" stroke="black" fill="white"
  /></svg>
```

5 code/boat.vert

```
void main(void) {
  gl_Position = ftransform();
  gl_TexCoord[0] = gl_MultiTexCoord0;
}
```

6 code/colour.c

```
#include "colour.h"

const struct vec4 red    = {{ 231/255.0,  13/255.0, 100/255.0, 255/255.0 }};
const struct vec4 orange = {{ 255/255.0, 200/255.0, 103/255.0, 255/255.0 }};
const struct vec4 green  = {{ 153/255.0, 255/255.0,  53/255.0, 255/255.0 }};
const struct vec4 blue   = {{   3/255.0,  71/255.0, 148/255.0, 255/255.0 }};
const struct vec4 white  = {{ 255/255.0, 255/255.0, 255/255.0, 255/255.0 }};
const struct vec4 black  = {{   0/255.0,   0/255.0,   0/255.0, 255/255.0 }};
```

7 code/colour.h

```
#ifndef COLOUR_H
#define COLOUR_H 1

#include "vector.h"

const struct vec4 red, orange, green, blue, white, black;

#endif
```

8 code/Colour.hs

```
module Colour (Colour, red, orange, green, blue, white, black, blend) where

-- with 3 elements [r,g,b]
type Colour = [Double]

-- base colour palette
red, orange, green, blue, white, black :: Colour
red    = map (/ 255) [ 231,  13, 100 ]
orange = map (/ 255) [ 255, 200, 103 ]
green  = map (/ 255) [ 153, 255,  53 ]
blue   = map (/ 255) [   3,  71, 148 ]
white  = map (/ 255) [ 255, 255, 255 ]
black  = map (/ 255) [   0,   0,   0 ]

-- linear mixing of colours
blend :: Double -> Colour -> Colour -> Colour
blend k = zipWith (\a b -> a * (1 - k) + k * b)
```

9 code/dinghy.c

```
/*
gcc \
  -std=c99 -Wall -Wextra -pedantic \
```

```
  -O3 -march=native -fopenmp \
  -o dinghy dinghy.c \
  -lglfw -lGL -lGLU -lGLEW -lm
*/

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

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

#define RECORD
#undef  RECORD

#define P 10
#define M 16
#define N 12
#define B 11
#define D 3

typedef GLfloat R;

const int width = 1280, height = 720;
const R stiffness  = 50
      , repulsion  = 1
      , density    = 5
      , pressure   = 1
      , gravity    = 10
      , bouyancy   = 20
      , resistance =  0.996
      , dt         =  0.004
      , pi         =  3.141592653589793;

typedef struct {
  R R[P];
  R r[P];
  R m[P][M][N];
  R x[P][M][N][D];
  R v[P][M][N][D];
  R l[P][M][N][B][B];
  R n[P][M][N][D];
  R fog[P][M][N];
} S;

typedef struct {
  R m[P][M];
  R x[P][M][D];
  R f[P][M][N][D];
  R g[P][M][D];
  R x0[P][D];
  R m0[P];
} T;

typedef struct {
```

```
  GLuint vboX;
  GLuint vboN;
  GLuint vboF;
  GLuint vboC;
  GLuint vboI;
  GLuint vao;
  GLuint i[P][M][N][2][3];
  R c[P][4];
} G;

R rnd(void) {
  return rand() / (R) RAND_MAX;
}

void multmm(R *m, R *n, R *o) {
  R t[D][D];
  for (int i = 0; i < D; ++i) {
    for (int j = 0; j < D; ++j) {
      t[i][j] = 0;
      for (int k = 0; k < D; ++k) {
        t[i][j] += m[i * D + k] * n[k * D + j];
      }
    }
  }
  for (int i = 0; i < D; ++i) {
    for (int j = 0; j < D; ++j) {
      o[i * D + j] = t[i][j];
    }
  }
}

void multmv(R *m, R *v, R *o) {
  R t[D];
  for (int i = 0; i < D; ++i) {
    t[i] = 0;
    for (int j = 0; j < D; ++j) {
      t[i] += m[i * D + j] * v[j];
    }
  }
  for (int i = 0; i < D; ++i) {
    o[i] = t[i];
  }
}

void hsv2rgb(float h, float s, float v, float *rp, float *gp, float *bp) {
  float i, f, p, q, t, r, g, b;
  int ii;
  if (s == 0.0) { r = v; g = v; b = v; } else {
    h = h - floor(h);
    h = h * 6.0;
    i = floor(h); ii = (int) i;
    f = h - i;
    p = v*(1.0 - s);
    q = v*(1.0 - (s*f));
    t = v*(1.0 - (s*(1.0 - f)));
    switch(ii) {
      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;
      case 5: default: r = v; g = p; b = q; break;
    }
  }
  *rp = r;
  *gp = g;
  *bp = b;
}

void initG(S *s, G *g) {
  glGenBuffers(1, &g->vboX);
  glGenBuffers(1, &g->vboN);
  glGenBuffers(1, &g->vboF);
  glGenBuffers(1, &g->vboC);
  glGenBuffers(1, &g->vboI);
  glGenVertexArrays(1, &g->vao);
  glBindVertexArray(g->vao);
  glBindBuffer(GL_ARRAY_BUFFER, g->vboX);
  glBufferData(GL_ARRAY_BUFFER, P * M * N * D * sizeof(R), 0, GL_STREAM_DRAW);
  glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, 0);
  glEnableVertexAttribArray(0);
  glBindBuffer(GL_ARRAY_BUFFER, g->vboN);
  glBufferData(GL_ARRAY_BUFFER, P * M * N * D * sizeof(R), 0, GL_STREAM_DRAW);
  glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, 0, 0);
  glEnableVertexAttribArray(2);
  glBindBuffer(GL_ARRAY_BUFFER, g->vboF);
  glBufferData(GL_ARRAY_BUFFER, P * M * N * sizeof(R), 0, GL_STREAM_DRAW);
  glFogCoordPointer(GL_FLOAT, 0, 0);
  for (int p = 0; p < P; ++p) {
    hsv2rgb(0.38196601125010515 * p, 1, 1, &g->c[p][0], &g->c[p][1], &g->c[p
        ↳ ][2]);
    g->c[p][3] = 1;
  }
  glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, g->vboI);
  for (int p = 0; p < P; ++p) {
    for (int m = 0; m < M; ++m) {
      int mm = (m + 1 + M) % M;
      for (int n = 0; n < N; ++n) {
        int nn = (n + 1 + N) % N;
        g->i[p][m][n][0][0] = (&s->x[p][m ][n ][0] - &s->x[0][0][0][0]) / D;
        g->i[p][m][n][0][1] = (&s->x[p][mm][n ][0] - &s->x[0][0][0][0]) / D;
        g->i[p][m][n][0][2] = (&s->x[p][m ][nn][0] - &s->x[0][0][0][0]) / D;
        g->i[p][m][n][1][0] = (&s->x[p][mm][nn][0] - &s->x[0][0][0][0]) / D;
        g->i[p][m][n][1][1] = (&s->x[p][m ][nn][0] - &s->x[0][0][0][0]) / D;
        g->i[p][m][n][1][2] = (&s->x[p][mm][n ][0] - &s->x[0][0][0][0]) / D;
      }
    }
  }
  glBufferData(GL_ELEMENT_ARRAY_BUFFER, P * M * N * 2 * D * sizeof(GLuint), &g->
      ↳ i[0][0][0][0][0], GL_STATIC_DRAW);
}

void init(S *s) {
  memset(s, 0, sizeof(*s));
```

```
  for (int p = 0; p < P; ++p) {
    s->R[p] = 2 * rnd() + 2;
    s->r[p] = s->R[p] / (2 * rnd() + 2);
    R m1 = density * 2 * pi * s->R[p] * s->r[p] * s->r[p] / (M * N);
    R p1[D] = { 4 * (rnd() - 0.5), 4 * (rnd() - 0.5), -64 * rnd() };
    R u = rnd() * 2 * pi;
    R v = rnd() * 2 * pi;
    R w = rnd() * 2 * pi;
    R mu[D][D] = { { cos(u), sin(u), 0 }, { -sin(u), cos(u), 0 }, { 0, 0, 1 } };
    R mv[D][D] = { { cos(v), 0, sin(v) }, { 0, 1, 0 }, { -sin(v), 0, cos(v) } };
    R mw[D][D] = { { 1, 0, 0 }, { 0, cos(w), sin(w) }, { 0, -sin(w), cos(w) } };
    R o[D][D];
    multmm(&mu[0][0], &mv[0][0], &o[0][0]);
    multmm(&mw[0][0], &o[0][0], &o[0][0]);
    R a = 0;
    R b = 0;
    R c = 0;
    R oo[D][D] = { { 0, -c, b }, { c, 0, -a }, { -b, a, 0 } };
    for (int m = 0; m < M; ++m) {
      R a = m * 2 * pi / M;
      for (int n = 0; n < N; ++n) {
        R b = n * 2 * pi / N;
        s->m[p][m][n] = m1;
        s->x[p][m][n][0] = cos(a) * (s->R[p] + s->r[p] * cos(b));
        s->x[p][m][n][1] = sin(a) * (s->R[p] + s->r[p] * cos(b));
        s->x[p][m][n][2] =                     s->r[p] * sin(b);
        multmv(&oo[0][0], &s->x[p][m][n][0], &s->v[p][m][n][0]);
        multmv(&o[0][0], &s->x[p][m][n][0], &s->x[p][m][n][0]);
        multmv(&o[0][0], &s->v[p][m][n][0], &s->v[p][m][n][0]);
        for (int d = 0; d < D; ++d) {
          s->x[p][m][n][d] += p1[d];
        }
      }
    }
    for (int m = 0; m < M; ++m) {
      for (int n = 0; n < N; ++n) {
        for (int i = -B/2; i <= B/2; ++i) {
          for (int j = -B/2; j <= B/2; ++j) {
            int mm = (m + i + M) % M;
            int nn = (n + j + N) % N;
            if (i == 0 && j == 0) {
              nn = (n + N / 2) % N;
            }
            R l = 0;
            for (int d = 0; d < D; ++d) {
              R dl = s->x[p][m][n][d] - s->x[p][mm][nn][d];
              l += dl * dl;
            }
            s->l[p][m][n][i+B/2][j+B/2] = sqrt(l);
          }
        }
      }
    }
  }
}

void step(S *s, T *t) {
```

```
230     /* zero temporary space */
        memset(t, 0, sizeof(*t));
        /* intra-mesh spring forces */
        #pragma omp parallel for
        for (int p = 0; p < P; ++p) {
235       for (int m = 0; m < M; ++m) {
            for (int n = 0; n < N; ++n) {
              for (int i = -B/2; i <= B/2; ++i) {
                for (int j = -B/2; j <= B/2; ++j) {
                  int mm = (m + i + M) % M;
240               int nn = (n + j + N) % N;
                  if (i == 0 && j == 0) {
                    nn = (n + N / 2) % N;
                  }
                  R dx[D];
245               R ldx = 0;
                  for (int d = 0; d < D; ++d) {
                    dx[d] = s->x[p][mm][nn][d] - s->x[p][m][n][d];
                    ldx += dx[d] * dx[d];
                  }
250               if (!(ldx > 0)) { continue; }
                  ldx = sqrt(ldx);
                  R k = stiffness * (s->l[p][m][n][i+B/2][j+B/2] - ldx) / ldx;
                  for (int d = 0; d < D; ++d) {
                    t->f[p][m][n][d] -= dx[d] * k;
255               }
                }
              }
            }
          }
260     }
        /* mesh center circle */
        #pragma omp parallel for
        for (int p = 0; p < P; ++p) {
          for (int m = 0; m < M; ++m) {
265         for (int n = 0; n < N; ++n) {
              for (int d = 0; d < D; ++d) {
                t->x[p][m][d] += s->x[p][m][n][d] * s->m[p][m][n];
                t->x0[p][d] += s->x[p][m][n][d] * s->m[p][m][n];
              }
270           t->m[p][m] += s->m[p][m][n];
              t->m0[p] += s->m[p][m][n];
            }
            for (int d = 0; d < D; ++d) {
              t->x[p][m][d] /= t->m[p][m];
275           t->x0[p][d] /= t->m0[p];
            }
          }
        }
        /* inflate */
280     #pragma omp parallel for
        for (int p = 0; p < P; ++p) {
          for (int m = 0; m < M; ++m) {
            for (int n = 0; n < N; ++n) {
              R dx[D];
285           R ldx = 0;
              for (int d = 0; d < D; ++d) {
```

```
            dx[d] = s->x[p][m][n][d] - t->x[p][m][d];
            ldx += dx[d] * dx[d];
          }
          ldx = sqrt(ldx);
          for (int d = 0; d < D; ++d) {
            t->f[p][m][n][d] += pressure * dx[d] / ldx;
          }
        }
      }
    }
    /* inter mesh forces */
    #pragma omp parallel for
    for (int p = 0; p < P; ++p) {
      for (int q = p + 1; q < P; ++q) {
        R l = 0;
        for (int d = 0; d < D; ++d) {
          l += (t->x0[p][d] - t->x0[q][d]) * (t->x0[p][d] - t->x0[q][d]);
        }
        l = sqrt(l);
        if (l < s->R[p] + s->r[p] + s->R[q] + s->r[q] + 0.01) {
          for (int mp = 0; mp < M; ++mp) {
            for (int mq = 0; mq < M; ++mq) {
              R dx[D];
              R ldx = 0;
              for (int d = 0; d < D; ++d) {
                dx[d] = t->x[p][mp][d] - t->x[q][mq][d];
                ldx += dx[d] * dx[d];
              }
              if (!(ldx > 0)) { continue; }
              ldx = sqrt(ldx);
              R k = pow(fmax(0.01, ldx - (s->r[p] + s->r[q])), -repulsion) / ldx;
              for (int d = 0; d < D; ++d) {
                R f = dx[d] * k;
                t->g[p][mp][d] += f;
                t->g[q][mq][d] -= f;
              }
            }
          }
        }
      }
    }
    /* normals */
    #pragma omp parallel for
    for (int p = 0; p < P; ++p) {
      for (int m = 0; m < M; ++m) {
        for (int n = 0; n < N; ++n) {
          R l = 0;
          for (int d = 0; d < D; ++d) {
            s->n[p][m][n][d] = s->x[p][m][n][d] - t->x[p][m][d];
            l += s->n[p][m][n][d] * s->n[p][m][n][d];
          }
          l = sqrt(l);
          for (int d = 0; d < D; ++d) {
            s->n[p][m][n][d] /= l;
          }
        }
      }
    }
```

```
    }
    /* accumulate */
    #pragma omp parallel for
    for (int p = 0; p < P; ++p) {
      for (int m = 0; m < M; ++m) {
        for (int n = 0; n < N; ++n) {
          for (int d = 0; d < D; ++d) {
            R f = t->f[p][m][n][d] + t->g[p][m][d];
            f -= 0.2 * s->x[p][m][n][d];
            R a = f / s->m[p][m][n];
            if (d == D - 1) {
              a -= gravity + bouyancy * fmin(0, s->x[p][m][n][d]);
            }
            s->v[p][m][n][d] += a * dt;
            s->v[p][m][n][d] *= resistance;
            s->x[p][m][n][d] += s->v[p][m][n][d] * dt;
          }
          s->fog[p][m][n] = fmax(-s->x[p][m][n][D-1], 0);
        }
      }
    }
}

void display(GLFWwindow *window, S *s, G *g) {
  R x[D] = { 0, 0, 0 };
  for (int p = 0; p < P; ++p) {
    for (int m = 0; m < M; ++m) {
      for (int n = 0; n < N; ++n) {
        for (int d = 0; d < D; ++d) {
          x[d] += s->x[p][m][n][d];
        }
      }
    }
  }
  for (int d = 0; d < D; ++d) {
    x[d] /= P * M * N;
  }
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glBindBuffer(GL_ARRAY_BUFFER, g->vboX);
  glBufferSubData(GL_ARRAY_BUFFER, 0, P * M * N * D * sizeof(R), &s->x↙
      ↳ [0][0][0][0]);
  glBindBuffer(GL_ARRAY_BUFFER, g->vboN);
  glBufferSubData(GL_ARRAY_BUFFER, 0, P * M * N * D * sizeof(R), &s->n↙
      ↳ [0][0][0][0]);
  glBindBuffer(GL_ARRAY_BUFFER, g->vboF);
  glBufferSubData(GL_ARRAY_BUFFER, 0, P * M * N * 1 * sizeof(R), &s->fog↙
      ↳ [0][0][0]);
  glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, g->vboI);

  // draw regular
  glEnable(GL_FOG);
  glEnableClientState(GL_FOG_COORD_ARRAY);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  GLfloat light_position[] = { 30.0, 10.0, 20.0, 1.0 };
  glLightfv(GL_LIGHT0, GL_POSITION, light_position);
  GLfloat spot_direction[] = { -3.0, -1.0, -2.0 };
```

```
    glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, spot_direction);
    gluLookAt(30, 100, 20, x[0], x[1], x[2], 0, 0, 1);
    glClear(GL_DEPTH_BUFFER_BIT);
    for (int p = 0; p < P; ++p) {
      glMaterialfv(GL_FRONT, GL_SPECULAR, &g->c[p][0]);
      glMaterialfv(GL_FRONT, GL_AMBIENT, &g->c[p][0]);
      glDrawElements(GL_TRIANGLES, M * N * 2 * 3, GL_UNSIGNED_INT, ((char *)0) + p↙
          ↪ * M * N * 2 * 3 * sizeof(GLuint));
    }
    glDisableClientState(GL_FOG_COORD_ARRAY);
    glDisable(GL_FOG);

    // draw reflection
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    GLfloat light_position2[] = { 30.0, 10.0, 20.0, 1.0 };
    glLightfv(GL_LIGHT0, GL_POSITION, light_position2);
    GLfloat spot_direction2[] = { -3.0, -1.0, -2.0 };
    glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, spot_direction2);
    gluLookAt(30, 100, 20, x[0], x[1], x[2], 0, 0, 1);
    glScalef(1, 1, -1);
    GLdouble clip[4] = { 0, 0, 1, 0 };
    glClipPlane(GL_CLIP_PLANE0, clip);
    glEnable(GL_CLIP_PLANE0);
    for (int p = 0; p < P; ++p) {
      glMaterialfv(GL_FRONT, GL_SPECULAR, &g->c[p][0]);
      glMaterialfv(GL_FRONT, GL_AMBIENT, &g->c[p][0]);
      glDrawElements(GL_TRIANGLES, M * N * 2 * 3, GL_UNSIGNED_INT, ((char *)0) + p↙
          ↪ * M * N * 2 * 3 * sizeof(GLuint));
    }
    glDisable(GL_CLIP_PLANE0);

    // draw surface
    glDisable(GL_LIGHTING);
    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
    glEnable(GL_TEXTURE_2D);
    glBegin(GL_QUADS);
    glColor4f(0.1, 0.1, 0.2, 0.75);
    glNormal3f(0,0,1);
    glTexCoord2f(-250,-250);
    glVertex3f(-500,-500,0);
    glTexCoord2f(-250, 250);
    glVertex3f(-500, 500,0);
    glTexCoord2f( 250, 250);
    glVertex3f( 500, 500,0);
    glTexCoord2f( 250,-250);
    glVertex3f( 500,-500,0);
    glEnd();
    glDisable(GL_TEXTURE_2D);
    glDisable(GL_BLEND);
    glEnable(GL_LIGHTING);
    glfwSwapBuffers(window);
  }

#ifdef RECORD
void capture(unsigned char *b) {
```

```
    glReadPixels(0, 0, width, height, GL_RGB, GL_UNSIGNED_BYTE, b);
    printf("P6\n%d %d\n255\n", width, height);
    fflush(stdout);
    fwrite(b, width * height * 3, 1, stdout);
    fflush(stdout);
  }
  #endif

  int main(int argc, char **argv) {
    (void) argc;
    (void) argv;
    S s;
    T t;
    G g;
  #ifdef RECORD
    unsigned char b[width * height * 3];
  #endif
    srand(time(0));
    glfwInit();
    glfwWindowHint(GLFW_SAMPLES, 64);
    GLFWwindow *window = glfwCreateWindow(width, height, "dinghy", 0, 0);
    glfwMakeContextCurrent(window);
    glewExperimental = GL_TRUE;
    glewInit();
    glGetError(); // discard error from GLEW
    glMatrixMode(GL_PROJECTION);
    gluPerspective(20,  width * 1.0 / height,  10,  200);
    glEnable(GL_DEPTH_TEST);
    glShadeModel(GL_SMOOTH);
    glEnable(GL_LIGHTING);
    GLfloat mat_shininess[] = { 10.0 };
    glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
    glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, 60.0);
    glEnable(GL_LIGHT0);
    glClearColor(0.1, 0.1, 0.2, 1.0);
    glFogi(GL_FOG_COORD_SRC, GL_FOG_COORD);
    glFogf(GL_FOG_DENSITY, 0.05);
    GLfloat fog[4] = { 0.1, 0.1, 0.2, 1.0 };
    glFogfv(GL_FOG_COLOR, fog);
    GLuint tex;
    glGenTextures(1, &tex);
    glBindTexture(GL_TEXTURE_2D, tex);
    GLfloat img[2][2][3] = { { { 1, 1, 1 }, { 0, 0 ,0 } }, { { 0, 0, 0 }, { 1, 1, ↙
        ↳ 1 } } };
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, 2, 2, 0, GL_RGB, GL_FLOAT, img);
    glGenerateMipmap(GL_TEXTURE_2D);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR)↙
        ↳ ;
    initG(&s, &g);
    int frame = 0;
    while (! glfwWindowShouldClose(window)) {
      if (frame % (8 * 25) == 0) { init(&s); }
      if (frame++ == (2 * 60 + 38) * 25) { exit(0); }
      for (int i = 0; i < 20; ++i) {
        step(&s, &t);
      }
```

```
    display(window, &s, &g);
#ifdef RECORD
    capture(&b[0]);
#endif
    glfwPollEvents();
  }
  return 0;
}
```

10 code/Font.hs

```
module Font (glyph, string) where

import Control.Monad (forM_)
import Graphics.Rendering.Cairo

-- renders a character filling a 2x2 box
glyph :: Char -> Render ()
glyph 'a' = moveTo 0 2 >> lineTo 0 1 >> arc 1 1 1 pi 0 >> lineTo 2 2 >> moveTo 0 1 >> lineTo 2 1
glyph 'b' = moveTo 0 1 >> lineTo 1.5 1 >> arcNegative 1.5 0.5 0.5 (pi/2) (-pi/2) >> lineTo 0 0 >> lineTo 0 2 >> lineTo 1.5 2 >> arcNegative 1.5 1.5 0.5 (pi/2) (-pi/2)
glyph 'c' = moveTo 2 0 >> lineTo 1 0 >> arcNegative 1 1 1 (-pi/2) (pi/2) >> lineTo 2 2
glyph 'd' = moveTo 1 0 >> lineTo 0 0 >> lineTo 0 2 >> lineTo 1 2 >> arcNegative 1 1 1 (pi/2) (-pi/2)
glyph 'e' = moveTo 2 0 >> lineTo 0 0 >> lineTo 0 2 >> lineTo 2 2 >> moveTo 0 1 >> lineTo 1 1
glyph 'f' = moveTo 2 0 >> lineTo 0 0 >> lineTo 0 2 >> moveTo 0 1 >> lineTo 1 1
glyph 'g' = moveTo 2 0 >> lineTo 1 0 >> arcNegative 1 1 1 (-pi/2) 0 >> lineTo 1 1
glyph 'h' = moveTo 0 0 >> lineTo 0 2 >> moveTo 0 1 >> lineTo 2 1 >> moveTo 2 0 >> lineTo 2 2
glyph 'i' = moveTo 0 0 >> lineTo 2 0 >> moveTo 1 0 >> lineTo 1 2 >> moveTo 0 2 >> lineTo 2 2
glyph 'j' = moveTo 0 0 >> lineTo 2 0 >> moveTo 0 2 >> arcNegative 0 1 1 (pi/2) 0 >> lineTo 1 0
glyph 'k' = moveTo 0 0 >> lineTo 0 2 >> moveTo 0 1 >> lineTo 1 1 >> lineTo 2 0 >> moveTo 1 1 >> lineTo 2 2
glyph 'l' = moveTo 0 0 >> lineTo 0 2 >> lineTo 2 2
glyph 'm' = moveTo 0 2 >> lineTo 0 0 >> lineTo 1 1 >> lineTo 2 0 >> lineTo 2 2
glyph 'n' = moveTo 0 2 >> lineTo 0 0 >> lineTo 2 2 >> lineTo 2 0
glyph 'o' = moveTo 1 0 >> arc 1 1 1 (-pi/2) (pi/2) >> arc 1 1 1 (pi/2) (-pi/2)
glyph 'p' = moveTo 0 1 >> lineTo 1.5 1 >> arcNegative 1.5 0.5 0.5 (pi/2) (-pi/2) >> lineTo 0 0 >> lineTo 0 2
glyph 'q' = moveTo 1 0 >> arc 1 1 1 (-pi/2) (pi/2) >> arc 1 1 1 (pi/2) (-pi/2) >> moveTo 1 1 >> lineTo 2 2
glyph 'r' = moveTo 0 1 >> lineTo 1.5 1 >> arcNegative 1.5 0.5 0.5 (pi/2) (-pi/2) >> lineTo 0 0 >> lineTo 0 2 >> moveTo 1 1 >> lineTo 2 2
glyph 's' = moveTo 2 0 >> lineTo 0.5 0 >> arcNegative 0.5 0.5 0.5 (-pi/2) (pi/2) >> lineTo 1.5 1 >> arc 1.5 1.5 0.5 (-pi/2) (pi/2) >> lineTo 0 2
glyph 't' = moveTo 0 0 >> lineTo 2 0 >> moveTo 1 0 >> lineTo 1 2
glyph 'u' = moveTo 0 0 >> lineTo 0 1 >> arcNegative 1 1 1 pi 0 >> lineTo 2 0
glyph 'v' = moveTo 0 0 >> lineTo 0 1 >> lineTo 1 2 >> lineTo 2 1 >> lineTo 2 0
glyph 'w' = moveTo 0 0 >> lineTo 0 2 >> lineTo 1 1 >> lineTo 2 2 >> lineTo 2 0
glyph 'x' = moveTo 0 0 >> lineTo 2 2 >> moveTo 2 0 >> lineTo 0 2
```

```
glyph 'y' = moveTo 0 0 >> lineTo 1 1 >> lineTo 2 0 >> moveTo 1 1 >> lineTo 1 2
glyph 'z' = moveTo 0 0 >> lineTo 2 0 >> lineTo 0 2 >> lineTo 2 2
glyph _ = return ()

-- renders a string centered vertically filling horizontally a 1x1 box
string :: String -> Render ()
string str = do
  save
  let n = fromIntegral . length $ str
      s = 0.5 / n
  scale s s
  translate 1 n
  forM_ str $ \c -> save >> scale (sqrt 0.5) (sqrt 0.5) >> translate (-1) (-1) >> glyph c >> restore >> translate 2 0
  setSourceRGB 0 0 0
  setLineWidth 0.5
  strokePreserve
  setSourceRGB 1 1 1
  setLineWidth 0.25
  stroke
  restore
```

11 code/.gitignore

```
incidentsatsea
*.o
*.frag.c
*.vert.c
dinghy
```

12 code/main.c

```
// for popen, pclose
#define _POSIX_C_SOURCE 2

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

#include <GL/glew.h>
#include <GL/glut.h>

#include <jack/jack.h>

#include "boat.h"
#include "colour.h"
#include "vector.h"
#include "wave.h"

const double pi = 3.141592653589793;

const double waveSpeed = 556.0618996853933; // 2 * pi * 4425 / 50 / 3;
const double waveZoom = 1.272019649514069;//1.189207115002721; // 1.4142135623730951; // sqrt(2);sqrt $ (sqrt 5 + 1) / 2
```

```
25  struct vec4 darkRed, darkBlue;

    struct {
      struct boat boat;
      struct wave wave;
30    int width, height, winwidth, winheight, frame, frames, record, ready, bufsize;
      char *buffer;
      double time;
      jack_client_t *jack;
      FILE *eca;
35  } data;

    void keyboard(unsigned char c, int x, int y) {
      if (! data.record) {
        exit(0);
40    }
    }

    void timer(int t) {
      glutTimerFunc(data.record ? 1 : 10, timer, t + 1);
45    glutPostRedisplay();
    }

    void reshape(int w, int h) {
      data.winwidth = w;
50    data.winheight = h;
      data.ready = 1;
    }

    void update(void) {
55    jack_position_t pos;
      /*jack_transport_state_t s = */ jack_transport_query(data.jack, &pos);
      double phase = pos.frame * 25.0 / pos.frame_rate / data.frames;
      data.time = phase - floor(phase);
    }
60
    void display(void) {
      if (data.ready) {
        update();
      }
65    {
        // view
        float x = data.width * 0.5 / data.height;
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
70      gluOrtho2D(-x, x, -0.5, 0.5);
        glViewport(0, 0, data.width, data.height);
        // background
        struct vec4 sky;
        if (data.time < 0.5) {
75        mix(&orange, &darkRed, data.time * 2, &sky);
        } else {
          mix(&darkRed, &darkBlue, data.time * 2 - 1, &sky);
        }
        glClearColor(sky.v[0], sky.v[1], sky.v[2], sky.v[3]);
80      glClear(GL_COLOR_BUFFER_BIT);
```

```
    // perspective
    int boatBias = -8;
    int viewBias =  4;
    double frequency = data.height * pi / data.width * pow(waveZoom, boatBias);
    double amplitude = (1 - cos(2 * pi * data.time)) / 2;
    double z = 1;//pow(8, amplitude);
    double boatFrequency = frequency * pow(waveZoom, -boatBias);
    double viewFrequency = frequency * pow(waveZoom, -viewBias);
    double boatAmplitude = amplitude / boatFrequency;
    double viewAmplitude = amplitude / viewFrequency;
    double boatX = data.width * -0.5 * cos(pi * data.time) / data.height;
    double viewX = 0;
    double boatPhase = data.time * waveSpeed;
    double viewPhase = data.time * waveSpeed;
    double boatY = boatAmplitude * (sin(boatX * boatFrequency + boatPhase) - ↙
        ↳ 0.5);
    double viewY = boatY + viewAmplitude;
    double viewT = -atan2(viewFrequency * viewAmplitude * cos(viewX * ↙
        ↳ viewFrequency + viewPhase), 16);
    double k = - (viewY + boatY) / (boatY - pow(waveZoom, viewBias - boatBias) *↙
        ↳  boatY);
    double tc = cos(viewT);
    double ts = sin(viewT);
    double dx = boatX;
    double dy = boatY;
    double y0 = -0.5 - dy;
    double y1 =  0.5 - dy;
    double tx00 = z * ( tc * (-x - dx) + ts * y0) + dx;
    double ty00 = z * (-ts * (-x - dx) + tc * y0) + dy + 1.0/6;
    double tx10 = z * ( tc * (-x - dx) + ts * y1) + dx;
    double ty10 = z * (-ts * (-x - dx) + tc * y1) + dy + 1.0/6;
    double tx01 = z * ( tc * ( x - dx) + ts * y0) + dx;
    double ty01 = z * (-ts * ( x - dx) + tc * y0) + dy + 1.0/6;
    double tx11 = z * ( tc * ( x - dx) + ts * y1) + dx;
    double ty11 = z * (-ts * ( x - dx) + tc * y1) + dy + 1.0/6;
    // distant waves
    for (int bias = -19; bias < boatBias; bias += 2) {
      data.wave.value.frequency = frequency * pow(waveZoom, -bias) / z;
      data.wave.value.amplitude = amplitude / data.wave.value.frequency;
      data.wave.value.level = (-pow(waveZoom, bias - boatBias) * boatY + boatY) ↙
          ↳ * k - boatY;
      data.wave.value.phase = data.time * waveSpeed;
      data.wave.value.width = z * fmax(1, pow(waveZoom, bias)) / data.height;
      data.wave.value.blend = data.height / 2.0 / z;
      mix(&sky, &blue,  1 - (boatBias - bias) / 12.0, &data.wave.value.fill);
      data.wave.value.fill.v[3] = 0.25;
      mix(&sky, &white, 0.5 - (boatBias - bias) / 24.0, &data.wave.value.stroke)↙
          ↳ ;
      wave_use(&data.wave);
      glBegin(GL_QUADS); {
        glTexCoord2f(tx00, ty00); glVertex2f(-x, -0.5);
        glTexCoord2f(tx01, ty01); glVertex2f( x, -0.5);
        glTexCoord2f(tx11, ty11); glVertex2f( x,  0.5);
        glTexCoord2f(tx10, ty10); glVertex2f(-x,  0.5);
      } glEnd();
      glUseProgram(0);
    }
```

```
      // boat
      data.boat.value.boatX = boatX;
      data.boat.value.frequency = boatFrequency;
      data.boat.value.amplitude = boatAmplitude;
      data.boat.value.level = -boatY;
      data.boat.value.phase = boatPhase;
      data.boat.value.size = (0.33333 / (1 - fabs(boatX / x)) - 0.0625) / z;
      mix(&sky, &red,   1, &data.boat.value.fill);
      mix(&sky, &white, 0.5, &data.boat.value.stroke);
      boat_use(&data.boat);
      glBegin(GL_QUADS); {
        glTexCoord2f(tx00, ty00); glVertex2f(-x, -0.5);
        glTexCoord2f(tx01, ty01); glVertex2f( x, -0.5);
        glTexCoord2f(tx11, ty11); glVertex2f( x,  0.5);
        glTexCoord2f(tx10, ty10); glVertex2f(-x,  0.5);
      } glEnd();
      boat_use(0);
      // near waves
      for (int bias = boatBias + 1; bias < 4; bias += 2) {
        data.wave.value.frequency = frequency * pow(waveZoom, -bias) / z;
        data.wave.value.amplitude = amplitude / data.wave.value.frequency;
        data.wave.value.level = (-pow(waveZoom, bias - boatBias) * boatY + boatY) ↙
            ↳ * k - boatY;
        data.wave.value.phase = data.time * waveSpeed;
        data.wave.value.width = z * fmax(1, pow(waveZoom, bias)) / data.height;
        data.wave.value.blend = data.height / 2.0 / z;
        struct vec4 tmp;
        float b = (bias - boatBias) / 12.0;
        mix(&blue, &green, b, &tmp);
        mix(&tmp, &black, b, &data.wave.value.fill);
        data.wave.value.fill.v[3] = 0.25;
        mix(&sky, &white, 0.5 - (boatBias - bias) / 24.0, &data.wave.value.stroke)↙
            ↳ ;
        wave_use(&data.wave);
        glBegin(GL_QUADS); {
          glTexCoord2f(tx00, ty00); glVertex2f(-x, -0.5);
          glTexCoord2f(tx01, ty01); glVertex2f( x, -0.5);
          glTexCoord2f(tx11, ty11); glVertex2f( x,  0.5);
          glTexCoord2f(tx10, ty10); glVertex2f(-x,  0.5);
        } glEnd();
        glUseProgram(0);
      }
    }
    glutSwapBuffers();
    { // recording
      if (data.ready && data.record) {
        if (data.frame == data.frames) {
          exit(0);
        }
        fprintf(stdout, "P6\n%d %d 255\n", data.width, data.height);
        glPixelStorei(GL_PACK_ALIGNMENT, 1);
        glReadPixels(0, 0, data.width, data.height, GL_RGB, GL_UNSIGNED_BYTE, data↙
            ↳ .buffer);
        fwrite(data.buffer, data.bufsize, 1, stdout);
      }
    }
    glutReportErrors();
```

```
}

void cleanup(void) {
  if (data.buffer) { free(data.buffer); }
  if (data.eca) { pclose(data.eca); }
}

int main(int argc, char **argv) {
  data.width = 1050;
  data.height = 576;
  data.frames = 4425;
  data.record = 0;
  mix(&red, &black, 1.0/3.0, &darkRed);
  mix(&blue, &black, 2.0/3.0, &darkBlue);
  glutInitWindowSize(data.width, data.height);
  glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE);
  glutInit(&argc, argv);
  glutCreateWindow("Incidents At Sea");
  glewInit();
  glEnable(GL_BLEND);
  glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  boat_init(&data.boat);
  wave_init(&data.wave);
  if (data.record) {
    data.bufsize = data.width * data.height * 3;
    data.buffer = malloc(data.bufsize);
    memset(data.buffer, 128, data.bufsize);
  } else {
    data.buffer = 0;
  }
  data.frame = 0;
  data.ready = 0;
  glClearColor(0.5, 1.0, 0.5, 1);
  glutReshapeFunc(reshape);
  glutDisplayFunc(display);
  glutTimerFunc(1, timer, 0);
  glutKeyboardFunc(keyboard);
  glutReportErrors();
  data.jack = jack_client_open("incidents", 0, 0);
  data.eca = popen("ecasound -G:jack,incidents,send -i:../audio/nightboat.wav -o
      ↪ :jack,system", "w");
  atexit(cleanup);
  glutMainLoop();
  return 0;
}
```

13 code/main.c.orig

```
// for popen, pclose
#define _POSIX_C_SOURCE 2

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

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

#include <GL/glew.h>
#include <GL/glut.h>

#include <jack/jack.h>

#include "boat.h"
#include "colour.h"
#include "vector.h"
#include "wave.h"

const double pi = 3.141592653589793;

const double waveSpeed = 556.0618996853933 * 2; // 2 * pi * 4425 / 50 / 3;
const double waveZoom = 1.272019649514069;//1.189207115002721; // ↙
    ↳ 1.4142135623730951; // sqrt(2);sqrt $ (sqrt 5 + 1) / 2


struct vec4 darkRed, darkBlue;

struct {
  struct boat boat;
  struct wave wave;
  int width, height, winwidth, winheight, frame, frames, record, ready, bufsize;
  char *buffer;
  double time;
  jack_client_t *jack;
  FILE *eca;
} data;

void keyboard(unsigned char c, int x, int y) {
  if (! data.record) {
    exit(0);
  }
}

void timer(int t) {
  glutTimerFunc(data.record ? 1 : 10, timer, t + 1);
  glutPostRedisplay();
}

void reshape(int w, int h) {
  data.winwidth = w;
  data.winheight = h;
  data.ready = 1;
}

void update(void) {
  jack_position_t pos;
  /*jack_transport_state_t s = */ jack_transport_query(data.jack, &pos);
  double phase = pos.frame * 25.0 / pos.frame_rate / data.frames;
  data.time = phase - floor(phase);
}

void display(void) {
```

```
  if (data.ready) {
    update();
  }
  {
    // view
    float x = data.width * 0.5 / data.height;
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-x, x, -0.5, 0.5);
    glViewport(0, 0, data.width, data.height);
    // background
    struct vec4 sky;
    if (data.time < 0.5) {
      mix(&orange, &darkRed, data.time * 2, &sky);
    } else {
      mix(&darkRed, &darkBlue, data.time * 2 - 1, &sky);
    }
    glClearColor(sky.v[0], sky.v[1], sky.v[2], sky.v[3]);
    glClear(GL_COLOR_BUFFER_BIT);
    // perspective
    int boatBias = -8;
    int viewBias =  4;
    double frequency = data.height * pi / data.width * pow(waveZoom, boatBias);
    double amplitude = (1 - cos(2 * pi * data.time)) / 2;
    double z = 1;//pow(8, amplitude);
    double boatFrequency = frequency * pow(waveZoom, -boatBias);
    double viewFrequency = frequency * pow(waveZoom, -viewBias);
    double boatAmplitude = amplitude / boatFrequency;
    double viewAmplitude = amplitude / viewFrequency;
    double boatX = data.width * -0.5 * cos(pi * data.time) / data.height;
    double viewX = 0;
    double boatPhase = data.time * waveSpeed;
    double viewPhase = data.time * waveSpeed;
    double boatY = boatAmplitude * (sin(boatX * boatFrequency + boatPhase) - ↙
        ↳ 0.5);
    double viewY = -boatY - 1 / boatFrequency;//1 - boatAmplitude;//↙
        ↳ viewAmplitude * (sin(viewX * viewFrequency + viewPhase) - 0.5);
    double viewT = -atan2(viewFrequency * viewAmplitude * cos(viewX * ↙
        ↳ viewFrequency + viewPhase), 2);
    double k = - (viewY + boatY) / (boatY - pow(waveZoom, viewBias - boatBias) *↙
        ↳  boatY);
    double tc = cos(viewT);
    double ts = sin(viewT);
    double dx = boatX;
    double dy = boatY;
    double y0 = -0.5 - dy;
    double y1 =  0.5 - dy;
    double tx00 = z * ( tc * (-x - dx) + ts * y0) + dx;
    double ty00 = z * (-ts * (-x - dx) + tc * y0) + dy + amplitude/6;
    double tx10 = z * ( tc * (-x - dx) + ts * y1) + dx;
    double ty10 = z * (-ts * (-x - dx) + tc * y1) + dy + amplitude/6;
    double tx01 = z * ( tc * ( x - dx) + ts * y0) + dx;
    double ty01 = z * (-ts * ( x - dx) + tc * y0) + dy + amplitude/6;
    double tx11 = z * ( tc * ( x - dx) + ts * y1) + dx;
    double ty11 = z * (-ts * ( x - dx) + tc * y1) + dy + amplitude/6;
    // distant waves
    for (int bias = -19; bias < boatBias; bias += 2) {
```

```
115         data.wave.value.frequency = frequency * pow(waveZoom, -bias) / z;
            data.wave.value.amplitude = amplitude / data.wave.value.frequency;
            data.wave.value.level = (-pow(waveZoom, bias - boatBias) * boatY + boatY) ↙
                ↳ * k - boatY;
            data.wave.value.phase = data.time * waveSpeed;
            data.wave.value.width = z * fmax(1, pow(waveZoom, bias)) / data.height;
120         data.wave.value.blend = data.height / 2.0 / z;
            mix(&sky, &blue,  1 - (boatBias - bias) / 12.0, &data.wave.value.fill);
            data.wave.value.fill.v[3] = 0.25;
            mix(&sky, &white, 0.5 - (boatBias - bias) / 24.0, &data.wave.value.stroke)↙
                ↳ ;
            wave_use(&data.wave);
125         glBegin(GL_QUADS); {
              glTexCoord2f(tx00, ty00); glVertex2f(-x, -0.5);
              glTexCoord2f(tx01, ty01); glVertex2f( x, -0.5);
              glTexCoord2f(tx11, ty11); glVertex2f( x,  0.5);
              glTexCoord2f(tx10, ty10); glVertex2f(-x,  0.5);
130         } glEnd();
            glUseProgram(0);
          }
          // boat
          data.boat.value.boatX = boatX;
135       data.boat.value.frequency = boatFrequency;
          data.boat.value.amplitude = boatAmplitude;
          data.boat.value.level = -boatY;
          data.boat.value.phase = boatPhase;
          data.boat.value.size = (0.25 / (1 - fabs(boatX / x)) - 0.0625) / z;
140       mix(&sky, &red,   1, &data.boat.value.fill);
          mix(&sky, &white, 0.5, &data.boat.value.stroke);
          boat_use(&data.boat);
          glBegin(GL_QUADS); {
            glTexCoord2f(tx00, ty00); glVertex2f(-x, -0.5);
145         glTexCoord2f(tx01, ty01); glVertex2f( x, -0.5);
            glTexCoord2f(tx11, ty11); glVertex2f( x,  0.5);
            glTexCoord2f(tx10, ty10); glVertex2f(-x,  0.5);
          } glEnd();
          boat_use(0);
150       // near waves
          for (int bias = boatBias + 1; bias < 4; bias += 2) {
            data.wave.value.frequency = frequency * pow(waveZoom, -bias) / z;
            data.wave.value.amplitude = amplitude / data.wave.value.frequency;
            data.wave.value.level = (-pow(waveZoom, bias - boatBias) * boatY + boatY) ↙
                ↳ * k - boatY;
155         data.wave.value.phase = data.time * waveSpeed;
            data.wave.value.width = z * fmax(1, pow(waveZoom, bias)) / data.height;
            data.wave.value.blend = data.height / 2.0 / z;
            struct vec4 tmp;
            float b = (bias - boatBias) / 12.0;
160         mix(&blue, &green, b, &tmp);
            mix(&tmp, &black, b, &data.wave.value.fill);
            data.wave.value.fill.v[3] = 0.25;
            mix(&sky, &white, 0.5 - (boatBias - bias) / 24.0, &data.wave.value.stroke)↙
                ↳ ;
            wave_use(&data.wave);
165         glBegin(GL_QUADS); {
              glTexCoord2f(tx00, ty00); glVertex2f(-x, -0.5);
              glTexCoord2f(tx01, ty01); glVertex2f( x, -0.5);
```

```
          glTexCoord2f(tx11, ty11); glVertex2f( x,  0.5);
          glTexCoord2f(tx10, ty10); glVertex2f(-x,  0.5);
        } glEnd();
        glUseProgram(0);
      }
    }
    glutSwapBuffers();
    { // recording
      if (data.ready && data.record) {
        if (data.frame == data.frames) {
          exit(0);
        }
        fprintf(stdout, "P6\n%d %d 255\n", data.width, data.height);
        glPixelStorei(GL_PACK_ALIGNMENT, 1);
        glReadPixels(0, 0, data.width, data.height, GL_RGB, GL_UNSIGNED_BYTE, data↙
            ↳ .buffer);
        fwrite(data.buffer, data.bufsize, 1, stdout);
      }
    }
    glutReportErrors();
  }

  void cleanup(void) {
    if (data.buffer) { free(data.buffer); }
    if (data.eca) { pclose(data.eca); }
  }

  int main(int argc, char **argv) {
    data.width = 1050;
    data.height = 576;
    data.frames = 4425;
    data.record = 0;
    mix(&red, &black, 1.0/3.0, &darkRed);
    mix(&blue, &black, 2.0/3.0, &darkBlue);
    glutInitWindowSize(data.width, data.height);
    glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE);
    glutInit(&argc, argv);
    glutCreateWindow("Incidents At Sea");
    glewInit();
    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    boat_init(&data.boat);
    wave_init(&data.wave);
    if (data.record) {
      data.bufsize = data.width * data.height * 3;
      data.buffer = malloc(data.bufsize);
      memset(data.buffer, 128, data.bufsize);
    } else {
      data.buffer = 0;
    }
    data.frame = 0;
    data.ready = 0;
    glClearColor(0.5, 1.0, 0.5, 1);
```

```
    glutReshapeFunc(reshape);
    glutDisplayFunc(display);
    glutTimerFunc(1, timer, 0);
    glutKeyboardFunc(keyboard);
    glutReportErrors();
    data.jack = jack_client_open("incidents", 0, 0);
    data.eca = popen("ecasound -G:jack,incidents,send -i:../audio/nightboat.wav -o
        ↪ :jack,system", "w");
    atexit(cleanup);
    glutMainLoop();
    return 0;
}
```

14 code/Makefile

```
CC = gcc
CFLAGS = -std=c99 -Wall -pedantic -Wextra -Wno-unused-parameter -O3 -march=
    ↪ native
#CFLAGS += -ggdb
LIBS = -lGLEW -lGL -lGLU -lglut -lcairo -ljack -lpthread -lrt -lm
OBJS = \
  boat.o \
  colour.o \
  main.o \
  shader.o \
  vector.o \
  wave.o
GENC = \
  boat.frag.c \
  boat.vert.c \
  wave.frag.c \
  wave.vert.c

all: incidentsatsea

clean:
	-rm -f $(OBJS) $(GENC)

.SUFFIXES:
.PHONY: all clean

incidentsatsea: $(OBJS)
	$(CC) $(CFLAGS) -o incidentsatsea $(OBJS) $(LIBS)

# C source to object file
%.o: %.c
	$(CC) $(CFLAGS) -o $@ -c $<

# shader source to C source
%.frag.c: %.frag s2c.sh
	./s2c.sh $*_frag < $< > $@
%.vert.c: %.vert s2c.sh
	./s2c.sh $*_vert < $< > $@

# dependencies
boat.o: boat.c boat.h boat.frag.c boat.vert.c shader.h vector.h
colour.o: colour.c colour.h vector.h
```

```
main.o: main.c boat.h colour.h shader.h vector.h wave.h
shader.o: shader.c shader.h
vector.o: vector.c vector.h
wave.o: wave.c wave.h wave.frag.c wave.vert.c shader.h vector.h
```

15 code/menu.hs

```
{-# LANGUAGE NoMonomorphismRestriction #-}

import Control.Monad (forM_, when)
import Graphics.Rendering.Cairo

import Font
import Colour
import Track

width  = 1536
height = 8192

menu = do
  setSourceRGBA 0 0 0 0
  paint
  setLineCap LineCapRound
  setLineJoin LineJoinRound
  save
  scale (width / 3) (width / 3)
  translate (0.5) (-0.5)
  forM_ (titles `zip` [0..]) $ \(track, number) -> do
    save
    translate 1 (1 + fromIntegral number)
    let angle = atan2 1 3
    rotate $ if even number then angle else -angle
    scale 3 3
    translate (-0.5) (-0.5)
    string $ " " ++ track ++ " "
    restore
  restore

main = do
  image <- createImageSurface FormatARGB32 width height
  renderWith image menu
  surfaceWriteToPNG image "menu.png"
```

16 code/nightboat.hs

```
{-# LANGUAGE NoMonomorphismRestriction #-}

import Control.Monad (forM_, when)
import Graphics.Rendering.Cairo

import Colour

width  = 1920
height = 1080
zoom = height / 576
weight =    4 * zoom
```

```
frames = 4425
peak   = 2495 / frames
step   = 2734 -- frames `gcd` step == 1

boatX0 = 10 * zoom
waveSpeed = 2 * pi * frames / 50
waveZoom = 3 ** (1 / 7)

filename f = (reverse .  take 6 . (++ repeat '0') . reverse . show $ f) ++ ".png"

darkRed  = blend (1/3) red black
darkBlue = blend (2/3) blue black

renderFrame image f = renderWith image $ do
  let _boatColour@[boatR,boatG,boatB] = red
      skyColour@[skyR,skyG,skyB]
        | f < 0.5   = blend (f * 2    ) orange  darkRed
        | otherwise = blend (f * 2 - 1) darkRed darkBlue
      seaColours = reverse $ map ($ blue)
        [ blend (1/4) black . blend (1/4) green
        , blend (2/4) black . blend (2/4) green
        , blend (3/4) black . blend (3/4) green
        , blend (5/6) skyColour
        , blend (4/6) skyColour
        , blend (3/6) skyColour
        , blend (2/6) skyColour
        , blend (1/6) skyColour
        ]
      (earlySea, lateSea) = splitAt 5 ([-9,-7 ..] `zip` seaColours)
      waveSeparation = height * magic / 4
      vanishingPoint = waveSeparation * waveZoom ** 7
      magic = cos (pi * (f - 0.5))
      --magic = let x = (2 * f - 1) * pi / 3 in (1 - signum x * (1 - 1 / cos x)) / 2
      seaLevel = (height + 2 * weight) * (1 - magic / 2) - weight
      boatX = (width + 2 * boatX0) * (1 - cos (pi * f)) / 2 - boatX0
      boatY = waveLevel 0 + waveY 0 boatX
      boatSize = sin (pi * f) ^ 2
      waveSize = sin (pi * f)
      waveHeight bias = (max 0 $ height / 4 * (1 - ((f - peak) / (1 - peak))^2)) * waveZoom ** bias
      waveFreq = 4 / height
      waveY bias x = vanishingPoint * (waveZoom ** bias - 1) + waveSize * waveHeight bias * cos ((x - (width / 2)) * waveFreq * waveZoom ** negate bias + f * waveSpeed)
      waveLevel bias = seaLevel - 0.5 * waveY bias boatX + boatSize * 100 * zoom
      boatAngle = atan2 (waveY 0 (boatX + 1) - waveY 0 (boatX - 1)) 2
      drawWave (bias, [seaR, seaG, seaB]) = do
        setSourceRGB seaR seaG seaB
        moveTo (- 2 * weight) (waveLevel bias + waveY bias (- 2 * weight))
        forM_ [-weight .. width + weight] $ \x -> do
          let y = waveLevel bias + waveY bias x
          lineTo x y
        lineTo (width + 2 * weight) (waveLevel bias + waveY bias (width + 2 * weight))
        lineTo (width + 2 * weight) (height + 2 * weight)
```

```
        lineTo (      - 2 * weight) (height + 2 * weight)
        lineTo (      - 2 * weight) (waveLevel bias + waveY bias (      - 2 * ↙
            ↳ weight))
        closePath
        fillPreserve
        setSourceRGB 0 0 0
        stroke
    setLineWidth (2 * zoom)
    setSourceRGB skyR skyG skyB
    paint
    forM_ earlySea drawWave
    when (boatSize > 0) $ do
      save
      setSourceRGB boatR boatG boatB
      translate boatX boatY
      rotate boatAngle
      scale (boatSize * zoom) (boatSize * zoom)
      moveTo (-180) (-50)
      forM_ [(-20,-50),(-20,-270),(140,-150),(-20,-70),(-20,-50),(200,-50)↙
          ↳ ,(120,30),(-120,30),(-180,-50)] $ \(x,y) -> do
        lineTo x y
      closePath
      fillPreserve
      restore
      setSourceRGB 0 0 0
      stroke
    forM_ lateSea drawWave

main = do
  image <- createImageSurface FormatARGB32 width height
  forM_ [ 0 .. frames - 1 ] $ \f -> do
    let g = (f * step) `mod` frames
    renderFrame image (fromIntegral g / frames)
    surfaceWriteToPNG image (filename g)
```

17 code/s2c.sh

```
#!/bin/bash
echo "/* machine-generated file, do not edit */"
echo "static const char $1[] ="
sed 's|\\|\\\\|' |
sed 's|^|"|' |
sed 's|$|\\n"|'
echo ";"
```

18 code/semaphore.hs

```
letters = ['a'..'z'] `zip` (
  [1,2,3,4, 0,0,0,2, 1,4,1,1, 1,1,2,2, 2,2,2,3, 3,4,5,5, 3,7] `zip`
  [0,0,0,0, 5,6,7,1, 3,6,4,5, 6,7,3,4, 5,6,7,4, 5,7,6,7, 6,6] )
numbers = ['0'..'9'] `zip` (
  [1, 1,2,3,4, 0,0,0,2, 1] `zip`
  [4, 0,0,0,0, 5,6,7,1, 3] )
space = (0,0)
numeralsMode = (4,5)
lettersMode = (4,6)
```

19 code/shader.c

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

#include "shader.h"

//====================================================================
// print a shader object's debug log
void shader_debug(GLhandleARB obj) {
  int infologLength = 0;
  int maxLength;
  if (glIsShader(obj)) {
    glGetShaderiv(obj, GL_INFO_LOG_LENGTH, &maxLength);
  } else {
    glGetProgramiv(obj, GL_INFO_LOG_LENGTH, &maxLength);
  }
  char *infoLog = malloc(maxLength);
  if (!infoLog) {
    return;
  }
  if (glIsShader(obj)) {
    glGetShaderInfoLog(obj, maxLength, &infologLength, infoLog);
  } else {
    glGetProgramInfoLog(obj, maxLength, &infologLength, infoLog);
  }
  if (infologLength > 0) {
    fprintf(stderr, "%s\n", infoLog);
  }
  free(infoLog);
}

//====================================================================
// generic shader initialization
struct shader *shader_init(
  struct shader *shader, const char *vert, const char *frag
) {
  if (! shader) { return 0; }
  shader->linkStatus     = 0;
  shader->vertexSource   = vert;
  shader->fragmentSource = frag;
  if (shader->vertexSource || shader->fragmentSource) {
    shader->program = glCreateProgramObjectARB();
    if (shader->vertexSource) {
      shader->vertex =
        glCreateShaderObjectARB(GL_VERTEX_SHADER_ARB);
      glShaderSourceARB(shader->vertex,
        1, (const GLcharARB **) &shader->vertexSource, 0
      );
      glCompileShaderARB(shader->vertex);
      shader_debug(shader->vertex);
      glAttachObjectARB(shader->program, shader->vertex);
    }
    if (shader->fragmentSource) {
      shader->fragment =
        glCreateShaderObjectARB(GL_FRAGMENT_SHADER_ARB);
      glShaderSourceARB(shader->fragment,
```

```
        1, (const GLcharARB **) &shader->fragmentSource, 0
      );
      glCompileShaderARB(shader->fragment);
      shader_debug(shader->fragment);
      glAttachObjectARB(shader->program, shader->fragment);
    }
    glLinkProgramARB(shader->program);
    glGetObjectParameterivARB(shader->program,
      GL_OBJECT_LINK_STATUS_ARB, &shader->linkStatus
    );
    if (! shader->linkStatus) {
      shader_debug(shader->program);
      return 0;
    }
  } else {
    return 0;
  }
  return shader;
}
```

20 code/shader.h

```
#ifndef SHADER_H
#define SHADER_H 1

#include <GL/glew.h>

//=========================================================
// generic shader data
struct shader {
  GLint linkStatus;
  GLhandleARB program;
  GLhandleARB fragment;
  GLhandleARB vertex;
  const GLcharARB *fragmentSource;
  const GLcharARB *vertexSource;
};

//=========================================================
// generic shader uniform location access macro
#define shader_uniform(self,name) \
  (self)->uniform.name = \
    glGetUniformLocationARB((self)->shader.program, #name)

//=========================================================
// generic shader uniform update access macro (integer)
#define shader_updatei(self,name) \
  glUniform1iARB((self)->uniform.name, (self)->value.name)

//=========================================================
// generic shader uniform update access macro (float)
#define shader_updatef(self,name) \
  glUniform1fARB((self)->uniform.name, (self)->value.name)

//=========================================================
// generic shader uniform update access macro (float4)
#define shader_updatef4(self,name) \
```

```
  glUniform4fARB((self)->uniform.name, (self)->value.name.v[0], (self)->value.↙
      ↳ name.v[1], (self)->value.name.v[2], (self)->value.name.v[3])

//============================================================================
// generic shader initialization
struct shader *shader_init(
  struct shader *shader, const char *vert, const char *frag
);

#endif
```

21 code/Track.hs

```
module Track (titles, durations) where

-- track titles
titles :: [String]
titles =
  [ "nightboat"
  , "semaphore"
  , "prepare to be boarded"
  , "no quarter"
  , "sonar"
  , "subterfuge"
  , "bulkhead collapse"
  , "man overboard"
  , "inflatable"
  , "aeronautics"
  , "engine failure"
  , "freefall"
  , "cloctopus"
  , "fisheye"
  , "coda"
  , "bubblicious"
  ]

-- track durations in frames
durations :: [Double]
durations = []
```

22 code/vector.c

```
#include "vector.h"

void mix(const struct vec4 *x, const struct vec4 *y, float t, struct vec4 *o) {
  float s = 1 - t;
  o->v[0] = x->v[0] * s + t * y->v[0];
  o->v[1] = x->v[1] * s + t * y->v[1];
  o->v[2] = x->v[2] * s + t * y->v[2];
  o->v[3] = x->v[3] * s + t * y->v[3];
}
```

23 code/vector.h

```
#ifndef VECTOR_H
#define VECTOR_H 1
```

```
struct vec4 {
  float v[4];
};

void mix(const struct vec4 *x, const struct vec4 *y, float t, struct vec4 *o);

#endif
```

24 code/wave.c

```
#include <math.h>

#include "wave.h"
#include "wave.frag.c"
#include "wave.vert.c"

struct wave *wave_init(struct wave *wave) {
  if (! wave) { return 0; }
  if (! shader_init(&wave->shader, wave_vert, wave_frag)) { return 0; }
  shader_uniform(wave, amplitude);
  shader_uniform(wave, frequency);
  shader_uniform(wave, level);
  shader_uniform(wave, phase);
  shader_uniform(wave, width);
  shader_uniform(wave, blend);
  shader_uniform(wave, stroke);
  shader_uniform(wave, fill);
  wave->value.amplitude = 0;
  wave->value.frequency = 0;
  wave->value.level = 0;
  wave->value.phase = 0;
  wave->value.width = 0;
  wave->value.blend = 0;
  wave->value.stroke.v[0] = 0;
  wave->value.stroke.v[1] = 0;
  wave->value.stroke.v[2] = 0;
  wave->value.stroke.v[3] = 1;
  wave->value.fill.v[0] = 1;
  wave->value.fill.v[1] = 1;
  wave->value.fill.v[2] = 1;
  wave->value.fill.v[3] = 1;
  return wave;
}

void wave_use(struct wave *wave) {
  glUseProgram(wave->shader.program);
  shader_updatef(wave, amplitude);
  shader_updatef(wave, frequency);
  shader_updatef(wave, level);
  shader_updatef(wave, phase);
  shader_updatef(wave, width);
  shader_updatef(wave, blend);
  shader_updatef4(wave, stroke);
  shader_updatef4(wave, fill);
}
```

25 code/wave.frag

```
/*

    stroking a path with a constant perpendicular width

    approximate sinusoid at P by a straight line tangent

              P     drop a vertical line from P to Q
             /|     PRO  is a right angle
          O/_|R     POQ  is a right angle
           /\ |      |RO| is one unit
          /  \|      |OQ| is 'width'
         /    Q     PRO  is similar to POQ

    |PQ|:|OQ| = |PO|:|RO| -> |PQ| = width * sqrt(1^2 + dy^2)

    gnuplot example

        set size square
        set xrange [-pi:pi]
        set yrange [-pi:pi]
        plot sin(x),sin(x)+0.5*sqrt(1+cos(x)*cos(x)),sin(x)-0.5*sqrt(1+cos(x)*
            cos(x))

*/

/* wave shape */
uniform float amplitude;
uniform float frequency;
uniform float level;
uniform float phase;

/* appearance */
uniform float width;
uniform float blend;
uniform vec4 stroke;
uniform vec4 fill;

const vec4 transparent = vec4(0.0);

void main(void) {
  vec2 p = gl_TexCoord[0].xy;
  float t = frequency * p.x + phase;
  float y = amplitude * sin(t) + level;
  float dy = amplitude * cos(t) * frequency;
  float delta = width * sqrt(1.0 + dy * dy);
  float y0 = y - delta;
  float y1 = y + delta;
  vec4 c;
  if (p.y < y0) {
    c = mix(fill, stroke, clamp(1.0 - blend * (y0 - p.y), 0.0, 1.0));
  } else if (p.y < y1) {
    c = stroke;
  } else {
    float a = clamp(blend * (p.y - y1), 0.0, 1.0);
    c = vec4(stroke.rgb, 1.0 - a);
```

```
  }
  gl_FragColor = c;
}
```

26 code/wave.h

```
#ifndef WAVE_H
#define WAVE_H 1

#include "shader.h"
#include "vector.h"

struct wave { struct shader shader;
  struct { GLint amplitude, frequency, level, phase, width, blend,
      ↘ stroke, fill; } uniform;
  struct { float amplitude, frequency, level, phase, width, blend; struct vec4
      ↘ stroke, fill; } value;
};

struct wave *wave_init(struct wave *wave);
void wave_use(struct wave *wave);

#endif
```

27 code/wave.vert

```
void main(void) {
  gl_Position = ftransform();
  gl_TexCoord[0] = gl_MultiTexCoord0;
}
```

28 design/boat.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="42" height="34" viewbox="0 0 42 34" version="1.1"
><path transform="translate(20,29)"
  d="M -18,-5 L -2,-5 -2,-27 14,-15 -2,-7 -2,-5  20,-5 12,3 -12,3 -18,-5 Z"
  stroke-linejoin="round" stroke-width="1" stroke="rgb(0,0,0)" fill="rgb
      ↘ (231,13,100)"
/></svg>
```

29 design/menu-1.svg

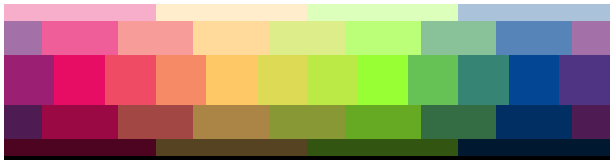
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN" "http://www.w3.org/Graphics/SVG
    ↘ /1.1/DTD/svg11.dtd">
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/
    ↘ xlink"
 width="1024" height="4096" viewbox="0 0 1024 4096" version="1.1"
><defs>
<path id="a" d="M 0,2 L 0,1 A 1,1 0 0 1 2,1 L 2,2 M 0,1 L 2,1" />
```

```
<path id="b" d="M 0,1 L 1,1 A 0.5,0.5 0 0 0 1,0 L 0,0 0,2 1,2 A 0.5,0.5 0 0 0 ↙
    ↳ 1,1" />
<path id="c" d="M 2,0 L 1,0 A 1,1 0 0 0 1,2 L 2,2" />
<path id="d" d="M 1,0 L 0,0 0,2 1,2 A 1,1 0 0 0 1,0" />
<path id="e" d="M 2,0 L 0,0 0,2 2,2 M 0,1 L 1,1" />
<path id="f" d="M 2,0 L 0,0 0,2 M 0,1 L 1,1" />
<path id="g" d="M 2,0 L 1,0 A 1,1 0 1 0 2,1 L 1,1" />
<path id="h" d="M 0,0 L 0,2 M 0,1 L 2,1 M 2,0 L 2,2" />
<path id="i" d="M 0,0 L 2,0 M 1,0 L 1,2 M 0,2 L 2,2" />
<path id="j" d="M 0,0 L 2,0 M 0,2 A 0.5,0.5 0 0 0 1,1 L 1,0" />
<path id="k" d="M 0,0 L 0,2 M 0,1 L 1,1 0,2 M 1,1 L 2,2" />
<path id="l" d="M 0,0 L 0,2 2,2" />
<path id="m" d="M 0,2 L 0,0 1,1 2,0 2,2" />
<path id="n" d="M 0,2 L 0,0 2,2 2,0" />
<path id="o" d="M 1,0 A 1,1 0 0 0 1,2 A 1,1 0 0 0 1,0" />
<path id="p" d="M 0,1 L 1,1 A 0.5,0.5 0 0 0 1,0 L 0,0 0,2" />
<path id="q" d="M 1,0 A 1,1 0 0 0 1,2 A 1,1 0 0 0 1,0 M 1,1 L 2,2" />
<path id="r" d="M 0,1 L 1,1 A 0.5,0.5 0 0 0 1,0 L 0,0 0,2 M 1,1 L 2,2" />
<path id="s" d="M 2,0 L 1,0 A 0.5,0.5 0 0 0 1,1 A 0.5,0.5 0 0 1 1,2 L 0,2" />
<path id="t" d="M 0,0 L 2,0 M 1,0 L 1,2" />
<path id="u" d="M 0,0 L 0,1 A 1,1 0 0 0 2,1 L 2,0" />
<path id="v" d="M 0,0 L 0,1 1,2 2,1 2,0" />
<path id="w" d="M 0,0 L 0,2 1,1 2,2 0,2" />
<path id="x" d="M 0,0 L 2,2 M 2,0 L 0,2" />
<path id="y" d="M 0,0 L 1,1 2,0 M 1,1 L 1,2" />
<path id="z" d="M 0,0 L 2,0 0,2 2,2" />
<svg id="nightboat" viewBox="0.625 0.625 26.75 2.75" preserveAspectRatio="none">
<use x="1" y="1" xlink:href="#n" />
<use x="4" y="1" xlink:href="#i" />
<use x="7" y="1" xlink:href="#g" />
<use x="10" y="1" xlink:href="#h" />
<use x="13" y="1" xlink:href="#t" />
<use x="16" y="1" xlink:href="#b" />
<use x="19" y="1" xlink:href="#o" />
<use x="22" y="1" xlink:href="#a" />
<use x="25" y="1" xlink:href="#t" />
</svg>
<svg id="semaphore" viewBox="0.625 0.625 26.75 2.75" preserveAspectRatio="none">
<use x="1" y="1" xlink:href="#s" />
<use x="4" y="1" xlink:href="#e" />
<use x="7" y="1" xlink:href="#m" />
<use x="10" y="1" xlink:href="#a" />
<use x="13" y="1" xlink:href="#p" />
<use x="16" y="1" xlink:href="#h" />
<use x="19" y="1" xlink:href="#o" />
<use x="22" y="1" xlink:href="#r" />
<use x="25" y="1" xlink:href="#e" />
</svg>
<svg id="prepare" viewBox="0.625 0.625 62.75 2.75" preserveAspectRatio="none">
<use x="1" y="1" xlink:href="#p" />
<use x="4" y="1" xlink:href="#r" />
<use x="7" y="1" xlink:href="#e" />
<use x="10" y="1" xlink:href="#p" />
<use x="13" y="1" xlink:href="#a" />
<use x="16" y="1" xlink:href="#r" />
<use x="19" y="1" xlink:href="#e" />
```

```
    <use x="25" y="1" xlink:href="#t" />
    <use x="28" y="1" xlink:href="#o" />
65
    <use x="34" y="1" xlink:href="#b" />
    <use x="37" y="1" xlink:href="#e" />

    <use x="43" y="1" xlink:href="#b" />
70  <use x="46" y="1" xlink:href="#o" />
    <use x="49" y="1" xlink:href="#a" />
    <use x="52" y="1" xlink:href="#r" />
    <use x="55" y="1" xlink:href="#d" />
    <use x="58" y="1" xlink:href="#e" />
75  <use x="61" y="1" xlink:href="#d" />
    </svg>
    <svg id="noquarter" viewBox="0.625 0.625 29.75 2.75" preserveAspectRatio="none">
    <use x="1" y="1" xlink:href="#n" />
    <use x="4" y="1" xlink:href="#o" />
80
    <use x="10" y="1" xlink:href="#q" />
    <use x="13" y="1" xlink:href="#u" />
    <use x="16" y="1" xlink:href="#a" />
    <use x="19" y="1" xlink:href="#r" />
85  <use x="22" y="1" xlink:href="#t" />
    <use x="25" y="1" xlink:href="#e" />
    <use x="28" y="1" xlink:href="#r" />
    </svg>
    <svg id="sonar" viewBox="0.625 0.625 14.75 2.75" preserveAspectRatio="none">
90  <use x="1" y="1" xlink:href="#s" />
    <use x="4" y="1" xlink:href="#o" />
    <use x="7" y="1" xlink:href="#n" />
    <use x="10" y="1" xlink:href="#a" />
    <use x="13" y="1" xlink:href="#r" />
95  </svg>
    <g id="tracks">
    <svg preserveAspectRatio="none" x="0" width="100%" y="0"                    height↙
        ↳ ="269.0181176224174" ><use xlink:href="#nightboat" /></svg>
    <svg preserveAspectRatio="none" x="0" width="100%" y="269.0181176224174"  height↙
        ↳ ="189.47699765114226"><use xlink:href="#semaphore" /></svg>
    <svg preserveAspectRatio="none" x="0" width="100%" y="458.49511527355963" height↙
        ↳ ="121.21135231369968"><use xlink:href="#prepare"    /></svg>
100 <svg preserveAspectRatio="none" x="0" width="100%" y="579.7064675872593"  height↙
        ↳ ="57.29656947083377" ><use xlink:href="#noquarter" /></svg>
    <svg preserveAspectRatio="none" x="0" width="100%" y="637.0030370580931"  height↙
        ↳ ="79.54111997127511" ><use xlink:href="#sonar"     /></svg>
    </g>
    </defs>
    <g fill="none" stroke="black"           stroke-width="0.7"  stroke-linejoin="↙
        ↳ round" stroke-linecap="round"><use xlink:href="#tracks" /></g>
105 <g fill="none" stroke="rgb(153,255,53)" stroke-width="0.25" stroke-linejoin="↙
        ↳ round" stroke-linecap="round"><use xlink:href="#tracks" /></g>
    </svg>
```

30 design/palette.png



31 README

```
Needs audio files from:

http://archive.org/details/ClaudiusMaximus--Incidents_At_Sea

resampled and trimmed to 48kHz, thus:

audio/nightboat.flac: FLAC audio bitstream data, 16 bit, stereo, 48 kHz, 8496000 samples
audio/nightboat.wav:  RIFF (little-endian) data, WAVE audio, Microsoft PCM, 16 bit, stereo 48000 Hz


Needs ecasound and jackd at 48000Hz.
Remember to stop/rewind transport between tweaks/reruns.

cd code && make && ./incidentsatsea

GPL license, NO WARRANTY

--
http://mathr.co.uk
claude@mathr.co.uk
```

32 todo/cover.hs

```
{-# LANGUAGE NoMonomorphismRestriction #-}

import Control.Monad (forM_)
import Data.Char (toLower)
import Graphics.Rendering.Cairo

-- 600dpi
width  = 6496
height = 4370

red    = map (/255) [231,  13, 100]
orange = map (/255) [255, 200, 103]
green  = map (/255) [153, 255,  53]
blue   = map (/255) [  3,  71, 148]
black  = map (/255) [  0,   0,   0]

blend k = zipWith (\a b -> a * (1 - k) + k * b)

tracks =
  [ "nightboat"
```

```
    , "semaphore"
    , "prepare to be boarded"
    , "no quarter"
    , "sonar"
    , "subterfuge"
    , "bulkhead collapse"
    , "man overboard"
    , "inflatable"
    , "aeronautics"
    , "engine failure"
    , "freefall"
    , "cloctopus"
    , "fisheye"
    , "coda"
    , "bubblicious"
    ]

durations = [ 4390, 3092, 1978, 935, 1298, 8200, 2965, 4995, 3965, 5213, 3996, 4765, 6113, 7892, 4318, 2667 ]
duration = sum durations

(trackWidths, trackHeights) = unzip $ zipWith f tracks durations
  where
    f name dur =
      let area = dur / duration
          aspect = fromIntegral (length name)
          -- area = w * h, w = aspect h -> area = aspect (h^2) -> h = sqrt (area / aspect)
      in (sqrt (aspect * area), sqrt (area / aspect))

listingWidth = maximum trackWidths
listingHeight = sum trackHeights

glyph 'a' = moveTo 0 2 >> lineTo 0 1 >> arc 1 1 1 pi 0 >> lineTo 2 2 >> moveTo 0 1 >> lineTo 2 1
glyph 'b' = moveTo 0 1 >> lineTo 1.5 1 >> arcNegative 1.5 0.5 0.5 (pi/2) (-pi/2) >> lineTo 0 0 >> lineTo 0 2 >> lineTo 1.5 2 >> arcNegative 1.5 1.5 0.5 (pi/2) (-pi/2)
glyph 'c' = moveTo 2 0 >> lineTo 1 0 >> arcNegative 1 1 1 (-pi/2) (pi/2) >> lineTo 2 2
glyph 'd' = moveTo 1 0 >> lineTo 0 0 >> lineTo 0 2 >> lineTo 1 2 >> arcNegative 1 1 1 (pi/2) (-pi/2)
glyph 'e' = moveTo 2 0 >> lineTo 0 0 >> lineTo 0 2 >> lineTo 2 2 >> moveTo 0 1 >> lineTo 1 1
glyph 'f' = moveTo 2 0 >> lineTo 0 0 >> lineTo 0 2 >> moveTo 0 1 >> lineTo 1 1
glyph 'g' = moveTo 2 0 >> lineTo 1 0 >> arcNegative 1 1 1 (-pi/2) 0 >> lineTo 1 1
glyph 'h' = moveTo 0 0 >> lineTo 0 2 >> moveTo 0 1 >> lineTo 2 1 >> moveTo 2 0 >> lineTo 2 2
glyph 'i' = moveTo 0 0 >> lineTo 2 0 >> moveTo 1 0 >> lineTo 1 2 >> moveTo 0 2 >> lineTo 2 2
glyph 'j' = moveTo 0 0 >> lineTo 2 0 >> moveTo 0 2 >> arcNegative 0 1 1 (pi/2) 0 >> lineTo 1 0
glyph 'k' = moveTo 0 0 >> lineTo 0 2 >> moveTo 0 1 >> lineTo 1 1 >> lineTo 2 0 >> moveTo 1 1 >> lineTo 2 2
glyph 'l' = moveTo 0 0 >> lineTo 0 2 >> lineTo 2 2
glyph 'm' = moveTo 0 2 >> lineTo 0 0 >> lineTo 1 1 >> lineTo 2 0 >> lineTo 2 2
```

```
65  glyph 'n' = moveTo 0 2 >> lineTo 0 0 >> lineTo 2 2 >> lineTo 2 0
    glyph 'o' = moveTo 1 0 >> arc 1 1 1 (-pi/2) (pi/2) >> arc 1 1 1 (pi/2) (-pi/2)
    glyph 'p' = moveTo 0 1 >> lineTo 1.5 1 >> arcNegative 1.5 0.5 0.5 (pi/2) (-pi/2)↙
        ↳  >> lineTo 0 0 >> lineTo 0 2
    glyph 'q' = moveTo 1 0 >> arc 1 1 1 (-pi/2) (pi/2) >> arc 1 1 1 (pi/2) (-pi/2) ↙
        ↳ >> moveTo 1 1 >> lineTo 2 2
    glyph 'r' = moveTo 0 1 >> lineTo 1.5 1 >> arcNegative 1.5 0.5 0.5 (pi/2) (-pi/2)↙
        ↳  >> lineTo 0 0 >> lineTo 0 2 >> moveTo 1 1 >> lineTo 2 2
70  glyph 's' = moveTo 2 0 >> lineTo 0.5 0 >> arcNegative 0.5 0.5 0.5 (-pi/2) (pi/2)↙
        ↳  >> lineTo 1.5 1 >> arc 1.5 1.5 0.5 (-pi/2) (pi/2) >> lineTo 0 2
    glyph 't' = moveTo 0 0 >> lineTo 2 0 >> moveTo 1 0 >> lineTo 1 2
    glyph 'u' = moveTo 0 0 >> lineTo 0 1 >> arcNegative 1 1 1 pi 0 >> lineTo 2 0
    glyph 'v' = moveTo 0 0 >> lineTo 0 1 >> lineTo 1 2 >> lineTo 2 1 >> lineTo 2 0
    glyph 'w' = moveTo 0 0 >> lineTo 0 2 >> lineTo 1 1 >> lineTo 2 2 >> lineTo 2 0
75  glyph 'x' = moveTo 0 0 >> lineTo 2 2 >> moveTo 2 0 >> lineTo 0 2
    glyph 'y' = moveTo 0 0 >> lineTo 1 1 >> lineTo 2 0 >> moveTo 1 1 >> lineTo 1 2
    glyph 'z' = moveTo 0 0 >> lineTo 2 0 >> lineTo 0 2 >> lineTo 2 2
    glyph _ = return ()

80  string xs = do
      save
      let n = fromIntegral . length $ xs
          s = 0.5 / n
      scale s s
85    translate 1 1
      forM_ xs $ \c -> save >> scale (sqrt 0.5) (sqrt 0.5) >> translate (-1) (-1) >>↙
          ↳  glyph c >> restore >> translate 2 0
      setSourceRGB 0 0 0
      setLineWidth 0.5
      strokePreserve
90    setSourceRGB 1 1 1
      setLineWidth 0.25
      stroke
      restore
      translate 0 (1 / n)
95
    track (name, w, y) = do
      save
      translate (listingWidth / listingHeight - w) y
      scale w w
100   string name
      restore

    menu mayday = do
      setSourceRGBA 0 0 0 0
105   paint
      setLineCap LineCapRound
      setLineJoin LineJoinRound
      save
      scale height height
110   forM_ (zip3 tracks (map (/ listingHeight) trackWidths) (scanl (+) 0 (map (/ ↙
          ↳ listingHeight) trackHeights))) track
      restore

    main = do
      image <- createImageSurface FormatARGB32 width (round height)
115   mayday <- readFile "mayday.txt"
```

```
  renderWith image (menu mayday)
  surfaceWriteToPNG image "cover.png"
```

33 todo/diagonals.hs

```
{-# LANGUAGE NoMonomorphismRestriction #-}

import Control.Monad (forM_, when)
import Data.List (zip4)
import Graphics.Rendering.Cairo

width  = 2 * pi * height
height = 576

red    = map (/255) [231,  13, 100]
orange = map (/255) [255, 200, 103]
green  = map (/255) [153, 255,  53]
blue   = map (/255) [  3,  71, 148]
black  = map (/255) [  0,   0,   0]

blend k = zipWith (\a b -> a * (1 - k) + k * b)

tracks =
  [ "nightboat"
  , "semaphore"
  , "prepare to be boarded"
  , "no quarter"
  , "sonar"
  , "subterfuge"
  , "bulkhead collapse"
  , "man overboard"
  , "inflatable"
  , "aeronautics"
  , "engine failure"
  , "freefall"
  , "cloctopus"
  , "fisheye"
  , "coda"
  , "bubblicious"
  ]

durations = [ 4390, 3092, 1978, 935, 1298, 8200, 2965, 4995, 3965, 5213, 3996, ↙
    ↳ 4765, 6113, 7892, 4318, 2667 ]
duration = sum durations

(trackWidths, trackHeights) = unzip $ zipWith f tracks durations
  where
    f name dur =
      let area = aspect^3 -- recip $ dur / duration
          aspect = fromIntegral (length name)
          -- area = w * h, w = aspect h -> area = aspect (h^2) -> h = sqrt (area↙
              ↳ / aspect)
      in (sqrt (aspect * area), sqrt (area / aspect))

horizontal = sum . map fst . filter (even . snd) . (`zip` [0..]) $ trackWidths
vertical   = sum . map fst . filter (odd  . snd) . (`zip` [0..]) $ trackWidths
incline = atan2 vertical' horizontal'
```

```
  horizontal' = sum . map fst . filter (even . snd) . (`zip` [0..]) $ trackHeights
  vertical'   = sum . map fst . filter (odd  . snd) . (`zip` [0..]) $ trackHeights
  diagonal = sqrt $ horizontal'^2 + vertical'^2

string xs = do
  save
  let n = fromIntegral . length $ xs
      s = 0.5 / n
  scale s s
  translate 1 n
  forM_ xs $ \c -> save >> scale (sqrt 0.5) (sqrt 0.5) >> translate (-1) (-1) >> glyph c >> restore >> translate 2 0
  setSourceRGB 0 0 0
  setLineWidth 0.5
  strokePreserve
  setSourceRGB 1 1 1
  setLineWidth 0.25
  stroke
  restore

menu = do
  setSourceRGBA 0 0 0 0
  paint
  setLineCap LineCapRound
  setLineJoin LineJoinRound
  save
  translate 0 (height * 3 / 4)
  let s = width / diagonal
  scale s s
  rotate (-incline)
  forM_ (zip4 [0..] tracks trackWidths trackHeights) $ \(number, track, w, h) -> do
    save
    scale h h
    rotate $ if even number then 0 else pi/2
    translate 0 (-0.5)
    string $ " " ++ track ++ " "
    restore
    if even number then translate h 0 else translate 0 h
  restore

main = do
  image <- createImageSurface FormatARGB32 (round width) height
  renderWith image menu
  surfaceWriteToPNG image "menu.png"


glyph 'a' = moveTo 0 2 >> lineTo 0 1 >> arc 1 1 1 pi 0 >> lineTo 2 2 >> moveTo 0 1 >> lineTo 2 1
glyph 'b' = moveTo 0 1 >> lineTo 1.5 1 >> arcNegative 1.5 0.5 0.5 (pi/2) (-pi/2) >> lineTo 0 0 >> lineTo 0 2 >> lineTo 1.5 2 >> arcNegative 1.5 1.5 0.5 (pi/2) (-pi/2)
glyph 'c' = moveTo 2 0 >> lineTo 1 0 >> arcNegative 1 1 1 (-pi/2) (pi/2) >> lineTo 2 2
glyph 'd' = moveTo 1 0 >> lineTo 0 0 >> lineTo 0 2 >> lineTo 1 2 >> arcNegative 1 1 1 (pi/2) (-pi/2)
```

```
glyph 'e' = moveTo 2 0 >> lineTo 0 0 >> lineTo 0 2 >> lineTo 2 2 >> moveTo 0 1 >> lineTo 1 1
glyph 'f' = moveTo 2 0 >> lineTo 0 0 >> lineTo 0 2 >> moveTo 0 1 >> lineTo 1 1
glyph 'g' = moveTo 2 0 >> lineTo 1 0 >> arcNegative 1 1 1 (-pi/2) 0 >> lineTo 1 1
glyph 'h' = moveTo 0 0 >> lineTo 0 2 >> moveTo 0 1 >> lineTo 2 1 >> moveTo 2 0 >> lineTo 2 2
glyph 'i' = moveTo 0 0 >> lineTo 2 0 >> moveTo 1 0 >> lineTo 1 2 >> moveTo 0 2 >> lineTo 2 2
glyph 'j' = moveTo 0 0 >> lineTo 2 0 >> moveTo 0 2 >> arcNegative 0 1 1 (pi/2) 0 >> lineTo 1 0
glyph 'k' = moveTo 0 0 >> lineTo 0 2 >> moveTo 0 1 >> lineTo 1 1 >> lineTo 2 0 >> moveTo 1 1 >> lineTo 2 2
glyph 'l' = moveTo 0 0 >> lineTo 0 2 >> lineTo 2 2
glyph 'm' = moveTo 0 2 >> lineTo 0 0 >> lineTo 1 1 >> lineTo 2 0 >> lineTo 2 2
glyph 'n' = moveTo 0 2 >> lineTo 0 0 >> lineTo 2 2 >> lineTo 2 0
glyph 'o' = moveTo 1 0 >> arc 1 1 1 (-pi/2) (pi/2) >> arc 1 1 1 (pi/2) (-pi/2)
glyph 'p' = moveTo 0 1 >> lineTo 1.5 1 >> arcNegative 1.5 0.5 0.5 (pi/2) (-pi/2) >> lineTo 0 0 >> lineTo 0 2
glyph 'q' = moveTo 1 0 >> arc 1 1 1 (-pi/2) (pi/2) >> arc 1 1 1 (pi/2) (-pi/2) >> moveTo 1 1 >> lineTo 2 2
glyph 'r' = moveTo 0 1 >> lineTo 1.5 1 >> arcNegative 1.5 0.5 0.5 (pi/2) (-pi/2) >> lineTo 0 0 >> lineTo 0 2 >> moveTo 1 1 >> lineTo 2 2
glyph 's' = moveTo 2 0 >> lineTo 0.5 0 >> arcNegative 0.5 0.5 0.5 (-pi/2) (pi/2) >> lineTo 1.5 1 >> arc 1.5 1.5 0.5 (-pi/2) (pi/2) >> lineTo 0 2
glyph 't' = moveTo 0 0 >> lineTo 2 0 >> moveTo 1 0 >> lineTo 1 2
glyph 'u' = moveTo 0 0 >> lineTo 0 1 >> arcNegative 1 1 1 pi 0 >> lineTo 2 0
glyph 'v' = moveTo 0 0 >> lineTo 0 1 >> lineTo 1 2 >> lineTo 2 1 >> lineTo 2 0
glyph 'w' = moveTo 0 0 >> lineTo 0 2 >> lineTo 1 1 >> lineTo 2 2 >> lineTo 2 0
glyph 'x' = moveTo 0 0 >> lineTo 2 2 >> moveTo 2 0 >> lineTo 0 2
glyph 'y' = moveTo 0 0 >> lineTo 1 1 >> lineTo 2 0 >> moveTo 1 1 >> lineTo 1 2
glyph 'z' = moveTo 0 0 >> lineTo 2 0 >> lineTo 0 2 >> lineTo 2 2
glyph _ = return ()
```

34 todo/mayday.txt

```
. . . - - - . . .
MAYDAY MAYDAY MAYDAY
THIS IS
MAXIMUS MAXIMUS MAXIMUS
MAYDAY MAXIMUS
POSITION 51 50 NORTH 12 62 EAST
WE ARE LOST IN FOG
WITH DAMAGED INSTRUMENTS
REQUIRE DIVINE INTERVENTION
WE ARE 1 PERSON ABOARD OVER
. . . - - - . . .
x
```

35 todo/packed.hs

```
{-# LANGUAGE NoMonomorphismRestriction #-}

import Control.Monad (forM_)
import Data.Char (toLower)
import Graphics.Rendering.Cairo
```

```
width  = 2167
height = width * sum (map (recip . fromIntegral . length) tracks)

red    = map (/255) [231,  13, 100]
orange = map (/255) [255, 200, 103]
green  = map (/255) [153, 255,  53]
blue   = map (/255) [  3,  71, 148]
black  = map (/255) [  0,   0,   0]

blend k = zipWith (\a b -> a * (1 - k) + k * b)

tracks =
  [ "nightboat"
  , "semaphore"
  , "prepare to be boarded"
  , "no quarter"
  , "sonar"
  , "subterfuge"
  , "bulkhead collapse"
  , "man overboard"
  , "inflatable"
  , "aeronautics"
  , "engine failure"
  , "freefall"
  , "cloctopus"
  , "fisheye"
  , "coda"
  , "bubblicious"
  ]

lengths = [ 4390, 3092, 1978, 935, 1298, 8200, 2965, 4995, 3965, 5213, 3996, ↙
    ↳ 4765, 6113, 7892, 4318, 2667 ]

glyph 'a' = moveTo 0 2 >> lineTo 0 1 >> arc 1 1 1 pi 0 >> lineTo 2 2 >> moveTo 0↙
    ↳  1 >> lineTo 2 1
glyph 'b' = moveTo 0 1 >> lineTo 1.5 1 >> arcNegative 1.5 0.5 0.5 (pi/2) (-pi/2)↙
    ↳  >> lineTo 0 0 >> lineTo 0 2 >> lineTo 1.5 2 >> arcNegative 1.5 1.5 0.5 (↙
    ↳ pi/2) (-pi/2)
glyph 'c' = moveTo 2 0 >> lineTo 1 0 >> arcNegative 1 1 1 (-pi/2) (pi/2) >> ↙
    ↳ lineTo 2 2
glyph 'd' = moveTo 1 0 >> lineTo 0 0 >> lineTo 0 2 >> lineTo 1 2 >> arcNegative ↙
    ↳ 1 1 1 (pi/2) (-pi/2)
glyph 'e' = moveTo 2 0 >> lineTo 0 0 >> lineTo 0 2 >> lineTo 2 2 >> moveTo 0 1 ↙
    ↳ >> lineTo 1 1
glyph 'f' = moveTo 2 0 >> lineTo 0 0 >> lineTo 0 2 >> moveTo 0 1 >> lineTo 1 1
glyph 'g' = moveTo 2 0 >> lineTo 1 0 >> arcNegative 1 1 1 (-pi/2) 0 >> lineTo 1 ↙
    ↳ 1
glyph 'h' = moveTo 0 0 >> lineTo 0 2 >> moveTo 0 1 >> lineTo 2 1 >> moveTo 2 0 ↙
    ↳ >> lineTo 2 2
glyph 'i' = moveTo 0 0 >> lineTo 2 0 >> moveTo 1 0 >> lineTo 1 2 >> moveTo 0 2 ↙
    ↳ >> lineTo 2 2
glyph 'j' = moveTo 0 0 >> lineTo 2 0 >> moveTo 0 2 >> arcNegative 0 1 1 (pi/2) 0↙
    ↳  >> lineTo 1 0
glyph 'k' = moveTo 0 0 >> lineTo 0 2 >> moveTo 0 1 >> lineTo 1 1 >> lineTo 2 0 ↙
    ↳ >> moveTo 1 1 >> lineTo 2 2
glyph 'l' = moveTo 0 0 >> lineTo 0 2 >> lineTo 2 2
```

```
glyph 'm' = moveTo 0 2 >> lineTo 0 0 >> lineTo 1 1 >> lineTo 2 0 >> lineTo 2 2
glyph 'n' = moveTo 0 2 >> lineTo 0 0 >> lineTo 2 2 >> lineTo 2 0
glyph 'o' = moveTo 1 0 >> arc 1 1 1 (-pi/2) (pi/2) >> arc 1 1 1 (pi/2) (-pi/2)
glyph 'p' = moveTo 0 1 >> lineTo 1.5 1 >> arcNegative 1.5 0.5 0.5 (pi/2) (-pi/2)
    ↪ >> lineTo 0 0 >> lineTo 0 2
glyph 'q' = moveTo 1 0 >> arc 1 1 1 (-pi/2) (pi/2) >> arc 1 1 1 (pi/2) (-pi/2)
    ↪ >> moveTo 1 1 >> lineTo 2 2
glyph 'r' = moveTo 0 1 >> lineTo 1.5 1 >> arcNegative 1.5 0.5 0.5 (pi/2) (-pi/2)
    ↪ >> lineTo 0 0 >> lineTo 0 2 >> moveTo 1 1 >> lineTo 2 2
glyph 's' = moveTo 2 0 >> lineTo 0.5 0 >> arcNegative 0.5 0.5 0.5 (-pi/2) (pi/2)
    ↪ >> lineTo 1.5 1 >> arc 1.5 1.5 0.5 (-pi/2) (pi/2) >> lineTo 0 2
glyph 't' = moveTo 0 0 >> lineTo 2 0 >> moveTo 1 0 >> lineTo 1 2
glyph 'u' = moveTo 0 0 >> lineTo 0 1 >> arcNegative 1 1 1 pi 0 >> lineTo 2 0
glyph 'v' = moveTo 0 0 >> lineTo 0 1 >> lineTo 1 2 >> lineTo 2 1 >> lineTo 2 0
glyph 'w' = moveTo 0 0 >> lineTo 0 2 >> lineTo 1 1 >> lineTo 2 2 >> lineTo 2 0
glyph 'x' = moveTo 0 0 >> lineTo 2 2 >> moveTo 2 0 >> lineTo 0 2
glyph 'y' = moveTo 0 0 >> lineTo 1 1 >> lineTo 2 0 >> moveTo 1 1 >> lineTo 1 2
glyph 'z' = moveTo 0 0 >> lineTo 2 0 >> lineTo 0 2 >> lineTo 2 2
glyph _ = return ()

string xs = do
  save
  let n = fromIntegral . length $ xs
      s = 0.5 / n
  scale s s
  translate 1 1
  forM_ xs $ \c -> save >> scale (sqrt 0.5) (sqrt 0.5) >> translate (-1) (-1) >>
    ↪ glyph c >> restore >> translate 2 0
  setSourceRGB 0 0 0
  setLineWidth 0.5
  strokePreserve
  setSourceRGB 1 1 1
  setLineWidth 0.25
  stroke
  restore
  translate 0 (1 / n)

menu = do
  setSourceRGBA 0 0 0 0
  paint
  setLineCap LineCapRound
  setLineJoin LineJoinRound
  save
  scale width width
  forM_ tracks string
  restore

main = do
  image <- createImageSurface FormatARGB32 width (round height)
  renderWith image menu
  surfaceWriteToPNG image "menu.png"
```