prismatic

Claude Heiland-Allen

2015–2018

Contents

1 cc/chromaticity-diagram.cc

```
#include <cmath>
#include <cstdint>
#include <cstdio>
#include <cstdlib>
using namespace std;

#include <png.h>
#include <arpa/inet.h>

inline bool xyz2srgb1
  ( const float &x, const float &y, const float &z
  ,       float &r,       float &g,       float &b
```

```
  ) {
  // white point
  float xw = x * 0.9505f;
  float yw = y;//1.0000f;
  float zw = z * 1.0890f;
  // linear rgb
  float rl =  3.2406f * xw + -1.5372f * yw + -0.4986f * zw;
  float gl = -0.9689f * xw +  1.8758f * yw +  0.0415f * zw;
  float bl =  0.0557f * xw + -0.2040f * yw +  1.0570f * zw;
  // check gammut
  bool  ok
    =  0.0f <= rl && rl <= 1.0f
    && 0.0f <= gl && gl <= 1.0f
    && 0.0f <= bl && bl <= 1.0f;
  r = rl;
  g = gl;
  b = bl;
  return ok;
}

inline void xyz2srgb2
  ( const float &x, const float &y, const float &z
  ,       float &r,       float &g,       float &b
  , const float &o, const float &s
  ) {
  // scale
  float rl = (x + o) * s;
  float gl = (y + o) * s;
  float bl = (z + o) * s;
  // clamp
  float rc = rl < 0.0f ? 0.0f : rl > 1.0f ? 1.0f : rl;
  float gc = gl < 0.0f ? 0.0f : gl > 1.0f ? 1.0f : gl;
  float bc = bl < 0.0f ? 0.0f : bl > 1.0f ? 1.0f : bl;
  // gamma
  float rg
    = rc <= 0.0031308f
    ? 12.92f * rc
    : 1.055f * pow(rc, 0.41666666f) - 0.055f;
  float gg
    = gc <= 0.0031308f
    ? 12.92f * gc
    : 1.055f * pow(gc, 0.41666666f) - 0.055f;
  float bg
    = bc <= 0.0031308f
    ? 12.92f * bc
    : 1.055f * pow(bc, 0.41666666f) - 0.055f;
  // output
  r = rg;
  g = gg;
  b = bg;
}

void write_png_err(png_structp png, png_const_charp msg) {
  fprintf(stderr, "error: png: %s\n", msg);
  jmp_buf *jmp = (jmp_buf *) png_get_error_ptr(png);
  if (jmp) { longjmp(*jmp, 1); } else { abort(); }
}
```

```
70
bool write_png(FILE *out, int width, int height, png_bytepp rows) {
  jmp_buf jmp;
  png_structp png = png_create_write_struct(PNG_LIBPNG_VER_STRING, &jmp, ↙
      ↳ write_png_err, 0);
  if (! png) { return false; }
75  png_infop info = png_create_info_struct(png);
  if (! info) { png_destroy_write_struct(&png, 0); return false; }
  if (setjmp(jmp)) { png_destroy_write_struct(&png, &info); return false; }
  png_init_io(png, out);
  png_set_compression_level(png, Z_BEST_COMPRESSION);
80  png_set_IHDR
    ( png, info
    , width, height, 16
    , PNG_COLOR_TYPE_RGBA
    , PNG_INTERLACE_ADAM7
85  , PNG_COMPRESSION_TYPE_DEFAULT
    , PNG_FILTER_TYPE_DEFAULT
    );
  png_set_sRGB(png, info, PNG_sRGB_INTENT_ABSOLUTE);
  png_write_info(png, info);
90  png_write_image(png, rows);
  png_write_end(png, info);
  png_destroy_write_struct(&png, &info);
  return true;
}
95
int main(int argc, char **argv) {
  const char *stem = "chromaticity";
  if (argc > 1) { stem = argv[1]; }
  int width = 1024;
100 int height = 1024;
  int depth = 100;
  // allocate image
  int bytes = width * height * 4 * sizeof(uint16_t);
  uint16_t *image = (uint16_t *) calloc(bytes, 1);
105 if (! image) {
    fprintf(stderr, "error: memory allocation failed: %d bytes\n", bytes);
    return 1;
  }
  bytes = height * sizeof(uint16_t *);
110 uint16_t **rows = (uint16_t **) calloc(bytes, 1);
  if (! rows) {
    fprintf(stderr, "error: memory allocation failed: %d bytes\n", bytes);
    return 1;
  }
115 for (int j = 0; j < height; ++j) {
    rows[j] = image + j * width * 4;
  }
  for (int k = 0; k < depth; ++k) {
    float cyy = (k + 0.5f) / depth;
120   #pragma omp parallel for schedule(static)
    for (int j = 0; j < height; ++j) {
      float cy = (height - (j + 0.5f)) / height - 0.05f;
      for (int i = 0; i < width; ++i) {
        float cx = (i + 0.5f) / width - 0.05f;
125     float x = cyy * cx / cy;
```

```
          float y = cyy;
          float z = cyy * (1.0f - cx - cy) / cy;
          float rl, gl, bl;
          bool ok = xyz2srgb1(x, y, z, rl, gl, bl);
          float r = 0.0f, g = 0.0f, b = 0.0f, a = 0.0f;
          if (ok) {
            xyz2srgb2(rl, gl, bl, r, g, b, 0.0f, 1.0f);
            a = 1.0f;
          }
          uint16_t rs = fminf(fmaxf(65535.0f * r, 0), 65535);
          uint16_t gs = fminf(fmaxf(65535.0f * g, 0), 65535);
          uint16_t bs = fminf(fmaxf(65535.0f * b, 0), 65535);
          uint16_t as = fminf(fmaxf(65535.0f * a, 0), 65535);
          image[4 * (width * j + i) + 0] = htons(rs);
          image[4 * (width * j + i) + 1] = htons(gs);
          image[4 * (width * j + i) + 2] = htons(bs);
          image[4 * (width * j + i) + 3] = htons(as);
        }
      }
      // write image
      char fname[100];
      snprintf(fname, 100, "%s-%02d5.png", stem, k);
      FILE *out = fopen(fname, "wb");
      if (! out) { return 1; }
      write_png(out, width, height, (png_bytepp) rows);
      fclose(out);
    }
    return 0;
}
```

2 cc/D50_XYZ.icc

(application/vnd.iccprofile; charset=binary)

3 cc/d50_xyz.s

```
.section .rodata
.global d50_xyz_icc
d50_xyz_icc:
.incbin "D50_XYZ.icc"
```

4 cc/D65_XYZ.icc

(application/vnd.iccprofile; charset=binary)

5 cc/d65_xyz.s

```
.section .rodata
.global d65_xyz_icc
d65_xyz_icc:
.incbin "D65_XYZ.icc"
.global d65_xyz_icc_end
d65_xyz_icc_end:
.global d65_xyz_icc_len
.set d65_xyz_icc_len, d65_xyz_icc_end - d65_xyz_icc
```

6 cc/format.cc

```
#include <cstdio>
using namespace std;

void write_header_intensity(FILE *f, int width, int height, float wavelength) {
  fprintf
    ( f
    , "prismatic\nintensity %e nm\nsize %d %d 1\nformat f32le\n\f"
    , wavelength
    , width
    , height
    );
}

void write_header_xyz(FILE *f, int width, int height) {
  fprintf
    ( f
    , "prismatic\nxyz\nsize %d %d 3\nformat f32le\n\f"
    , width
    , height
    );
}

void write_header_srgb(FILE *f, int width, int height) {
  fprintf
    ( f
    , "prismatic\nsrgb\nsize %d %d 3\nformat f32le\n\f"
    , width
    , height
    );
}

void write_data_intensity(FILE *f, int count, float *data) {
  fwrite(data, count * sizeof(*data), 1, f);
}

void write_data_xyz(FILE *f, int count, float *data) {
  fwrite(data, count * 3 * sizeof(*data), 1, f);
}

void write_data_srgb(FILE *f, int count, float *data) {
  fwrite(data, count * 3 * sizeof(*data), 1, f);
}

bool read_header_intensity(FILE *f, int *width, int *height, float *wavelength) ↙
    ↳ {
  if (3 != fscanf
    ( f
    , "prismatic\nintensity %e nm\nsize %d %d 1\nformat f32le"
    , wavelength
    , width
    , height
    )) { return false; }
  if (fgetc(f) != '\n') { return false; }
  if (fgetc(f) != '\f') { return false; }
  return true;
```

```
}

bool read_header_xyz(FILE *f, int *width, int *height) {
  if (2 != fscanf
    ( f
    , "prismatic\nxyz\nsize %d %d 3\nformat f32le"
    , width
    , height
    )) { return false; }
  if (fgetc(f) != '\n') { return false; }
  if (fgetc(f) != '\f') { return false; }
  return true;
}

bool read_header_srgb(FILE *f, int *width, int *height) {
  if (2 != fscanf
    ( f
    , "prismatic\nsrgb\nsize %d %d 3\nformat f32le"
    , width
    , height
    )) { return false; }
  if (fgetc(f) != '\n') { return false; }
  if (fgetc(f) != '\f') { return false; }
  return true;
}

bool read_data_intensity(FILE *f, int count, float *data) {
  return 1 == fread(data, count * sizeof(*data), 1, f);
}

bool read_data_xyz(FILE *f, int count, float *data) {
  return 1 == fread(data, count * 3 * sizeof(*data), 1, f);
}

bool read_data_srgb(FILE *f, int count, float *data) {
  return 1 == fread(data, count * 3 * sizeof(*data), 1, f);
}
```

7 cc/glass.h

```
#ifndef GLASS_H
#define GLASS_H 1

inline float glass_n(float wavelength) {
  float l2 = 1.0e-6f * wavelength * wavelength;
  return sqrt(1.0f + l2 *
    ( 1.34533359f / (l2 - 0.00997743871f)
    + 0.209073176f / (l2 - 0.0470450767f)
    + 0.937357162f / (l2 - 111.886764f)
    ));
}

const float glass_lk[13][2] =
  { { 1000.0f * 0.390f, 2.8886E-08f }
  , { 1000.0f * 0.400f, 1.9243E-08f }
  , { 1000.0f * 0.405f, 1.6869E-08f }
  , { 1000.0f * 0.420f, 1.2087E-08f }
```

```
  , { 1000.0f * 0.436f, 9.7490E-09f }
  , { 1000.0f * 0.460f, 8.8118E-09f }
  , { 1000.0f * 0.500f, 4.7818E-09f }
  , { 1000.0f * 0.546f, 3.4794E-09f }
  , { 1000.0f * 0.580f, 3.6961E-09f }
  , { 1000.0f * 0.620f, 3.9510E-09f }
  , { 1000.0f * 0.660f, 6.3120E-09f }
  , { 1000.0f * 0.700f, 4.4608E-09f }
  , { 1000.0f * 1.060f, 6.7549E-09f }
  };

inline float glass_k(float wavelength) {
  for (int i0 = 0; i0 < 12; ++i0) {
    int i1 = i0 + 1;
    float l0 = glass_lk[i0][0];
    float l1 = glass_lk[i1][0];
    if (l0 <= wavelength && wavelength <= l1) {
      float f1 = (wavelength - l0) / (l1 - l0);
      float f0 = 1.0f - f1;
      float k0 = glass_lk[i0][1];
      float k1 = glass_lk[i1][1];
      return f0 * k0 + f1 * k1;
    }
  }
  return 1.0e-6f;
}

#endif
```

8 cc/glass.sh

```
#!/bin/sh
cat "${1}/database/glass/schott/F2.yml" |
sed 's/\r//g' |
grep coefficients |
(
  read spam c1 c2 c3 c4 c5 c6 c7
  cat << EOF
#ifndef GLASS_H
#define GLASS_H 1

inline float glass_n(float wavelength) {
  float l2 = 1.0e-6f * wavelength * wavelength;
  return sqrt(1.0f + l2 *
    ( ${c2}f / (l2 - ${c3}f)
    + ${c4}f / (l2 - ${c5}f)
    + ${c6}f / (l2 - ${c7}f)
    ));
}

const float glass_lk[13][2] =
EOF
)
cat "${1}/database/glass/schott/F2.yml" |
sed 's/\r//g' |
tail -n+18 |
head -n 13 |
```

```
(
  sep="{"
  while read l k
  do
    echo "  ${sep} { 1000.0f * ${l}f, ${k}f }"
    sep=","
  done
)
cat <<EOF
  };

inline float glass_k(float wavelength) {
  for (int i0 = 0; i0 < 12; ++i0) {
    int i1 = i0 + 1;
    float l0 = glass_lk[i0][0];
    float l1 = glass_lk[i1][0];
    if (l0 <= wavelength && wavelength <= l1) {
      float f1 = (wavelength - l0) / (l1 - l0);
      float f0 = 1.0f - f1;
      float k0 = glass_lk[i0][1];
      float k1 = glass_lk[i1][1];
      return f0 * k0 + f1 * k1;
    }
  }
  return 1.0e-6f;
}

#endif
EOF
```

9 cc/lin2012xyz2e_1_7sf.h

```
#ifndef LIN2012XYZ2E_1_7SF_H
#define LIN2012XYZ2E_1_7SF_H

const int   wavelength_min = 390;
const int   wavelength_max = 830;
const float tristimulus_curve[wavelength_max - wavelength_min + 1][3] =
  { { 3.769647E-03f, 4.146161E-04f, 1.847260E-02f }
  , { 4.532416E-03f, 5.028333E-04f, 2.221101E-02f }
  , { 5.446553E-03f, 6.084991E-04f, 2.669819E-02f }
  , { 6.538868E-03f, 7.344436E-04f, 3.206937E-02f }
  , { 7.839699E-03f, 8.837389E-04f, 3.847832E-02f }
  , { 9.382967E-03f, 1.059646E-03f, 4.609784E-02f }
  , { 1.120608E-02f, 1.265532E-03f, 5.511953E-02f }
  , { 1.334965E-02f, 1.504753E-03f, 6.575257E-02f }
  , { 1.585690E-02f, 1.780493E-03f, 7.822113E-02f }
  , { 1.877286E-02f, 2.095572E-03f, 9.276013E-02f }
  , { 2.214302E-02f, 2.452194E-03f, 1.096090E-01f }
  , { 2.601285E-02f, 2.852216E-03f, 1.290077E-01f }
  , { 3.043036E-02f, 3.299115E-03f, 1.512047E-01f }
  , { 3.544325E-02f, 3.797466E-03f, 1.764441E-01f }
  , { 4.109640E-02f, 4.352768E-03f, 2.049517E-01f }
  , { 4.742986E-02f, 4.971717E-03f, 2.369246E-01f }
  , { 5.447394E-02f, 5.661014E-03f, 2.725123E-01f }
  , { 6.223612E-02f, 6.421615E-03f, 3.117820E-01f }
  , { 7.070048E-02f, 7.250312E-03f, 3.547064E-01f }
```

```
    , { 7.982513E-02f, 8.140173E-03f, 4.011473E-01f }
    , { 8.953803E-02f, 9.079860E-03f, 4.508369E-01f }
    , { 9.974848E-02f, 1.005608E-02f, 5.034164E-01f }
    , { 1.104019E-01f, 1.106456E-02f, 5.586361E-01f }
30  , { 1.214566E-01f, 1.210522E-02f, 6.162734E-01f }
    , { 1.328741E-01f, 1.318014E-02f, 6.760982E-01f }
    , { 1.446214E-01f, 1.429377E-02f, 7.378822E-01f }
    , { 1.566468E-01f, 1.545004E-02f, 8.013019E-01f }
    , { 1.687901E-01f, 1.664093E-02f, 8.655573E-01f }
35  , { 1.808328E-01f, 1.785302E-02f, 9.295791E-01f }
    , { 1.925216E-01f, 1.907018E-02f, 9.921293E-01f }
    , { 2.035729E-01f, 2.027369E-02f, 1.051821E+00f }
    , { 2.137531E-01f, 2.144805E-02f, 1.107509E+00f }
    , { 2.231348E-01f, 2.260041E-02f, 1.159527E+00f }
40  , { 2.319245E-01f, 2.374789E-02f, 1.208869E+00f }
    , { 2.403892E-01f, 2.491247E-02f, 1.256834E+00f }
    , { 2.488523E-01f, 2.612106E-02f, 1.305008E+00f }
    , { 2.575896E-01f, 2.739923E-02f, 1.354758E+00f }
    , { 2.664991E-01f, 2.874993E-02f, 1.405594E+00f }
45  , { 2.753532E-01f, 3.016909E-02f, 1.456414E+00f }
    , { 2.838921E-01f, 3.165145E-02f, 1.505960E+00f }
    , { 2.918246E-01f, 3.319038E-02f, 1.552826E+00f }
    , { 2.989200E-01f, 3.477912E-02f, 1.595902E+00f }
    , { 3.052993E-01f, 3.641495E-02f, 1.635768E+00f }
50  , { 3.112031E-01f, 3.809569E-02f, 1.673573E+00f }
    , { 3.169047E-01f, 3.981843E-02f, 1.710604E+00f }
    , { 3.227087E-01f, 4.157940E-02f, 1.748280E+00f }
    , { 3.288194E-01f, 4.337098E-02f, 1.787504E+00f }
    , { 3.349242E-01f, 4.517180E-02f, 1.826609E+00f }
55  , { 3.405452E-01f, 4.695420E-02f, 1.863108E+00f }
    , { 3.451688E-01f, 4.868718E-02f, 1.894332E+00f }
    , { 3.482554E-01f, 5.033657E-02f, 1.917479E+00f }
    , { 3.494153E-01f, 5.187611E-02f, 1.930529E+00f }
    , { 3.489075E-01f, 5.332218E-02f, 1.934819E+00f }
60  , { 3.471746E-01f, 5.470603E-02f, 1.932650E+00f }
    , { 3.446705E-01f, 5.606335E-02f, 1.926395E+00f }
    , { 3.418483E-01f, 5.743393E-02f, 1.918437E+00f }
    , { 3.390240E-01f, 5.885107E-02f, 1.910430E+00f }
    , { 3.359926E-01f, 6.030809E-02f, 1.901224E+00f }
65  , { 3.324276E-01f, 6.178644E-02f, 1.889000E+00f }
    , { 3.280157E-01f, 6.326570E-02f, 1.871996E+00f }
    , { 3.224637E-01f, 6.472352E-02f, 1.848545E+00f }
    , { 3.156225E-01f, 6.614749E-02f, 1.817792E+00f }
    , { 3.078201E-01f, 6.757256E-02f, 1.781627E+00f }
70  , { 2.994771E-01f, 6.904928E-02f, 1.742514E+00f }
    , { 2.909776E-01f, 7.063280E-02f, 1.702749E+00f }
    , { 2.826646E-01f, 7.238339E-02f, 1.664439E+00f }
    , { 2.747962E-01f, 7.435960E-02f, 1.629207E+00f }
    , { 2.674312E-01f, 7.659383E-02f, 1.597360E+00f }
75  , { 2.605847E-01f, 7.911436E-02f, 1.568896E+00f }
    , { 2.542749E-01f, 8.195345E-02f, 1.543823E+00f }
    , { 2.485254E-01f, 8.514816E-02f, 1.522157E+00f }
    , { 2.433039E-01f, 8.872657E-02f, 1.503611E+00f }
    , { 2.383414E-01f, 9.266008E-02f, 1.486673E+00f }
80  , { 2.333253E-01f, 9.689723E-02f, 1.469595E+00f }
    , { 2.279619E-01f, 1.013746E-01f, 1.450709E+00f }
    , { 2.219781E-01f, 1.060145E-01f, 1.428440E+00f }
```

```
    , { 2.151735E-01f, 1.107377E-01f, 1.401587E+00f }
    , { 2.075619E-01f, 1.155111E-01f, 1.370094E+00f }
    , { 1.992183E-01f, 1.203122E-01f, 1.334220E+00f }
    , { 1.902290E-01f, 1.251161E-01f, 1.294275E+00f }
    , { 1.806905E-01f, 1.298957E-01f, 1.250610E+00f }
    , { 1.707154E-01f, 1.346299E-01f, 1.203696E+00f }
    , { 1.604471E-01f, 1.393309E-01f, 1.154316E+00f }
    , { 1.500244E-01f, 1.440235E-01f, 1.103284E+00f }
    , { 1.395705E-01f, 1.487372E-01f, 1.051347E+00f }
    , { 1.291920E-01f, 1.535066E-01f, 9.991789E-01f }
    , { 1.189859E-01f, 1.583644E-01f, 9.473958E-01f }
    , { 1.090615E-01f, 1.633199E-01f, 8.966222E-01f }
    , { 9.951424E-02f, 1.683761E-01f, 8.473981E-01f }
    , { 9.041850E-02f, 1.735365E-01f, 8.001576E-01f }
    , { 8.182895E-02f, 1.788048E-01f, 7.552379E-01f }
    , { 7.376817E-02f, 1.841819E-01f, 7.127879E-01f }
    , { 6.619477E-02f, 1.896559E-01f, 6.725198E-01f }
    , { 5.906380E-02f, 1.952101E-01f, 6.340976E-01f }
    , { 5.234242E-02f, 2.008259E-01f, 5.972433E-01f }
    , { 4.600865E-02f, 2.064828E-01f, 5.617313E-01f }
    , { 4.006154E-02f, 2.121826E-01f, 5.274921E-01f }
    , { 3.454373E-02f, 2.180279E-01f, 4.948809E-01f }
    , { 2.949091E-02f, 2.241586E-01f, 4.642586E-01f }
    , { 2.492140E-02f, 2.307302E-01f, 4.358841E-01f }
    , { 2.083981E-02f, 2.379160E-01f, 4.099313E-01f }
    , { 1.723591E-02f, 2.458706E-01f, 3.864261E-01f }
    , { 1.407924E-02f, 2.546023E-01f, 3.650566E-01f }
    , { 1.134516E-02f, 2.640760E-01f, 3.454812E-01f }
    , { 9.019658E-03f, 2.742490E-01f, 3.274095E-01f }
    , { 7.097731E-03f, 2.850680E-01f, 3.105939E-01f }
    , { 5.571145E-03f, 2.964837E-01f, 2.948102E-01f }
    , { 4.394566E-03f, 3.085010E-01f, 2.798194E-01f }
    , { 3.516303E-03f, 3.211393E-01f, 2.654100E-01f }
    , { 2.887638E-03f, 3.344175E-01f, 2.514084E-01f }
    , { 2.461588E-03f, 3.483536E-01f, 2.376753E-01f }
    , { 2.206348E-03f, 3.629601E-01f, 2.241211E-01f }
    , { 2.149559E-03f, 3.782275E-01f, 2.107484E-01f }
    , { 2.337091E-03f, 3.941359E-01f, 1.975839E-01f }
    , { 2.818931E-03f, 4.106582E-01f, 1.846574E-01f }
    , { 3.649178E-03f, 4.277595E-01f, 1.720018E-01f }
    , { 4.891359E-03f, 4.453993E-01f, 1.596918E-01f }
    , { 6.629364E-03f, 4.635396E-01f, 1.479415E-01f }
    , { 8.942902E-03f, 4.821376E-01f, 1.369428E-01f }
    , { 1.190224E-02f, 5.011430E-01f, 1.268279E-01f }
    , { 1.556989E-02f, 5.204972E-01f, 1.176796E-01f }
    , { 1.997668E-02f, 5.401387E-01f, 1.094970E-01f }
    , { 2.504698E-02f, 5.600208E-01f, 1.020943E-01f }
    , { 3.067530E-02f, 5.800972E-01f, 9.527993E-02f }
    , { 3.674999E-02f, 6.003172E-01f, 8.890075E-02f }
    , { 4.315171E-02f, 6.206256E-01f, 8.283548E-02f }
    , { 4.978584E-02f, 6.409398E-01f, 7.700982E-02f }
    , { 5.668554E-02f, 6.610772E-01f, 7.144001E-02f }
    , { 6.391651E-02f, 6.808134E-01f, 6.615436E-02f }
    , { 7.154352E-02f, 6.999044E-01f, 6.117199E-02f }
    , { 7.962917E-02f, 7.180890E-01f, 5.650407E-02f }
    , { 8.821473E-02f, 7.351593E-01f, 5.215121E-02f }
    , { 9.726978E-02f, 7.511821E-01f, 4.809566E-02f }
```

```
140     , { 1.067504E-01f, 7.663143E-01f, 4.431720E-02f }
        , { 1.166192E-01f, 7.807352E-01f, 4.079734E-02f }
        , { 1.268468E-01f, 7.946448E-01f, 3.751912E-02f }
        , { 1.374060E-01f, 8.082074E-01f, 3.446846E-02f }
        , { 1.482471E-01f, 8.213817E-01f, 3.163764E-02f }
145     , { 1.593076E-01f, 8.340701E-01f, 2.901901E-02f }
        , { 1.705181E-01f, 8.461711E-01f, 2.660364E-02f }
        , { 1.818026E-01f, 8.575799E-01f, 2.438164E-02f }
        , { 1.931090E-01f, 8.682408E-01f, 2.234097E-02f }
        , { 2.045085E-01f, 8.783061E-01f, 2.046415E-02f }
150     , { 2.161166E-01f, 8.879907E-01f, 1.873456E-02f }
        , { 2.280650E-01f, 8.975211E-01f, 1.713788E-02f }
        , { 2.405015E-01f, 9.071347E-01f, 1.566174E-02f }
        , { 2.535441E-01f, 9.169947E-01f, 1.429644E-02f }
        , { 2.671300E-01f, 9.269295E-01f, 1.303702E-02f }
155     , { 2.811351E-01f, 9.366731E-01f, 1.187897E-02f }
        , { 2.954164E-01f, 9.459482E-01f, 1.081725E-02f }
        , { 3.098117E-01f, 9.544675E-01f, 9.846470E-03f }
        , { 3.241678E-01f, 9.619834E-01f, 8.960687E-03f }
        , { 3.384319E-01f, 9.684390E-01f, 8.152811E-03f }
160     , { 3.525786E-01f, 9.738289E-01f, 7.416025E-03f }
        , { 3.665839E-01f, 9.781519E-01f, 6.744115E-03f }
        , { 3.804244E-01f, 9.814106E-01f, 6.131421E-03f }
        , { 3.940988E-01f, 9.836669E-01f, 5.572778E-03f }
        , { 4.076972E-01f, 9.852081E-01f, 5.063463E-03f }
165     , { 4.213484E-01f, 9.863813E-01f, 4.599169E-03f }
        , { 4.352003E-01f, 9.875357E-01f, 4.175971E-03f }
        , { 4.494206E-01f, 9.890228E-01f, 3.790291E-03f }
        , { 4.641616E-01f, 9.910811E-01f, 3.438952E-03f }
        , { 4.794395E-01f, 9.934913E-01f, 3.119341E-03f }
170     , { 4.952180E-01f, 9.959172E-01f, 2.829038E-03f }
        , { 5.114395E-01f, 9.980205E-01f, 2.565722E-03f }
        , { 5.280233E-01f, 9.994608E-01f, 2.327186E-03f }
        , { 5.448696E-01f, 9.999930E-01f, 2.111280E-03f }
        , { 5.618898E-01f, 9.997557E-01f, 1.915766E-03f }
175     , { 5.790137E-01f, 9.989839E-01f, 1.738589E-03f }
        , { 5.961882E-01f, 9.979123E-01f, 1.577920E-03f }
        , { 6.133784E-01f, 9.967737E-01f, 1.432128E-03f }
        , { 6.305897E-01f, 9.957356E-01f, 1.299781E-03f }
        , { 6.479223E-01f, 9.947115E-01f, 1.179667E-03f }
180     , { 6.654866E-01f, 9.935534E-01f, 1.070694E-03f }
        , { 6.833782E-01f, 9.921156E-01f, 9.718623E-04f }
        , { 7.016774E-01f, 9.902549E-01f, 8.822531E-04f }
        , { 7.204110E-01f, 9.878596E-01f, 8.010231E-04f }
        , { 7.394495E-01f, 9.849324E-01f, 7.273884E-04f }
185     , { 7.586285E-01f, 9.815036E-01f, 6.606347E-04f }
        , { 7.777885E-01f, 9.776035E-01f, 6.001146E-04f }
        , { 7.967750E-01f, 9.732611E-01f, 5.452416E-04f }
        , { 8.154530E-01f, 9.684764E-01f, 4.954847E-04f }
        , { 8.337389E-01f, 9.631369E-01f, 4.503642E-04f }
190     , { 8.515493E-01f, 9.571062E-01f, 4.094455E-04f }
        , { 8.687862E-01f, 9.502540E-01f, 3.723345E-04f }
        , { 8.853376E-01f, 9.424569E-01f, 3.386739E-04f }
        , { 9.011588E-01f, 9.336897E-01f, 3.081396E-04f }
        , { 9.165278E-01f, 9.242893E-01f, 2.804370E-04f }
195     , { 9.318245E-01f, 9.146707E-01f, 2.552996E-04f }
        , { 9.474524E-01f, 9.052333E-01f, 2.324859E-04f }
```

```
      , { 9.638388E-01f, 8.963613E-01f, 2.117772E-04f }
      , { 9.812596E-01f, 8.883069E-01f, 1.929758E-04f }
      , { 9.992953E-01f, 8.808462E-01f, 1.759024E-04f }
200   , { 1.017343E+00f, 8.736445E-01f, 1.603947E-04f }
      , { 1.034790E+00f, 8.663755E-01f, 1.463059E-04f }
      , { 1.051011E+00f, 8.587203E-01f, 1.335031E-04f }
      , { 1.065522E+00f, 8.504295E-01f, 1.218660E-04f }
      , { 1.078421E+00f, 8.415047E-01f, 1.112857E-04f }
205   , { 1.089944E+00f, 8.320109E-01f, 1.016634E-04f }
      , { 1.100320E+00f, 8.220154E-01f, 9.291003E-05f }
      , { 1.109767E+00f, 8.115868E-01f, 8.494468E-05f }
      , { 1.118438E+00f, 8.007874E-01f, 7.769425E-05f }
      , { 1.126266E+00f, 7.896515E-01f, 7.109247E-05f }
210   , { 1.133138E+00f, 7.782053E-01f, 6.507936E-05f }
      , { 1.138952E+00f, 7.664733E-01f, 5.960061E-05f }
      , { 1.143620E+00f, 7.544785E-01f, 5.460706E-05f }
      , { 1.147095E+00f, 7.422473E-01f, 5.005417E-05f }
      , { 1.149464E+00f, 7.298229E-01f, 4.590157E-05f }
215   , { 1.150838E+00f, 7.172525E-01f, 4.211268E-05f }
      , { 1.151326E+00f, 7.045818E-01f, 3.865437E-05f }
      , { 1.151033E+00f, 6.918553E-01f, 3.549661E-05f }
      , { 1.150002E+00f, 6.791009E-01f, 3.261220E-05f }
      , { 1.148061E+00f, 6.662846E-01f, 2.997643E-05f }
220   , { 1.144998E+00f, 6.533595E-01f, 2.756693E-05f }
      , { 1.140622E+00f, 6.402807E-01f, 2.536339E-05f }
      , { 1.134757E+00f, 6.270066E-01f, 2.334738E-05f }
      , { 1.127298E+00f, 6.135148E-01f, 2.150221E-05f }
      , { 1.118342E+00f, 5.998494E-01f, 1.981268E-05f }
225   , { 1.108033E+00f, 5.860682E-01f, 1.826500E-05f }
      , { 1.096515E+00f, 5.722261E-01f, 1.684667E-05f }
      , { 1.083928E+00f, 5.583746E-01f, 1.554631E-05f }
      , { 1.070387E+00f, 5.445535E-01f, 1.435360E-05f }
      , { 1.055934E+00f, 5.307673E-01f, 1.325915E-05f }
230   , { 1.040592E+00f, 5.170130E-01f, 1.225443E-05f }
      , { 1.024385E+00f, 5.032889E-01f, 1.133169E-05f }
      , { 1.007344E+00f, 4.895950E-01f, 1.048387E-05f }
      , { 9.895268E-01f, 4.759442E-01f, 0.000000E+00f }
      , { 9.711213E-01f, 4.623958E-01f, 0.000000E+00f }
235   , { 9.523257E-01f, 4.490154E-01f, 0.000000E+00f }
      , { 9.333248E-01f, 4.358622E-01f, 0.000000E+00f }
      , { 9.142877E-01f, 4.229897E-01f, 0.000000E+00f }
      , { 8.952798E-01f, 4.104152E-01f, 0.000000E+00f }
      , { 8.760157E-01f, 3.980356E-01f, 0.000000E+00f }
240   , { 8.561607E-01f, 3.857300E-01f, 0.000000E+00f }
      , { 8.354235E-01f, 3.733907E-01f, 0.000000E+00f }
      , { 8.135565E-01f, 3.609245E-01f, 0.000000E+00f }
      , { 7.904565E-01f, 3.482860E-01f, 0.000000E+00f }
      , { 7.664364E-01f, 3.355702E-01f, 0.000000E+00f }
245   , { 7.418777E-01f, 3.228963E-01f, 0.000000E+00f }
      , { 7.171219E-01f, 3.103704E-01f, 0.000000E+00f }
      , { 6.924717E-01f, 2.980865E-01f, 0.000000E+00f }
      , { 6.681600E-01f, 2.861160E-01f, 0.000000E+00f }
      , { 6.442697E-01f, 2.744822E-01f, 0.000000E+00f }
250   , { 6.208450E-01f, 2.631953E-01f, 0.000000E+00f }
      , { 5.979243E-01f, 2.522628E-01f, 0.000000E+00f }
      , { 5.755410E-01f, 2.416902E-01f, 0.000000E+00f }
      , { 5.537296E-01f, 2.314809E-01f, 0.000000E+00f }
```

```
    , { 5.325412E-01f, 2.216378E-01f, 0.000000E+00f }
    , { 5.120218E-01f, 2.121622E-01f, 0.000000E+00f }
    , { 4.922070E-01f, 2.030542E-01f, 0.000000E+00f }
    , { 4.731224E-01f, 1.943124E-01f, 0.000000E+00f }
    , { 4.547417E-01f, 1.859227E-01f, 0.000000E+00f }
    , { 4.368719E-01f, 1.778274E-01f, 0.000000E+00f }
    , { 4.193121E-01f, 1.699654E-01f, 0.000000E+00f }
    , { 4.018980E-01f, 1.622841E-01f, 0.000000E+00f }
    , { 3.844986E-01f, 1.547397E-01f, 0.000000E+00f }
    , { 3.670592E-01f, 1.473081E-01f, 0.000000E+00f }
    , { 3.497167E-01f, 1.400169E-01f, 0.000000E+00f }
    , { 3.326305E-01f, 1.329013E-01f, 0.000000E+00f }
    , { 3.159341E-01f, 1.259913E-01f, 0.000000E+00f }
    , { 2.997374E-01f, 1.193120E-01f, 0.000000E+00f }
    , { 2.841189E-01f, 1.128820E-01f, 0.000000E+00f }
    , { 2.691053E-01f, 1.067113E-01f, 0.000000E+00f }
    , { 2.547077E-01f, 1.008052E-01f, 0.000000E+00f }
    , { 2.409319E-01f, 9.516653E-02f, 0.000000E+00f }
    , { 2.277792E-01f, 8.979594E-02f, 0.000000E+00f }
    , { 2.152431E-01f, 8.469044E-02f, 0.000000E+00f }
    , { 2.033010E-01f, 7.984009E-02f, 0.000000E+00f }
    , { 1.919276E-01f, 7.523372E-02f, 0.000000E+00f }
    , { 1.810987E-01f, 7.086061E-02f, 0.000000E+00f }
    , { 1.707914E-01f, 6.671045E-02f, 0.000000E+00f }
    , { 1.609842E-01f, 6.277360E-02f, 0.000000E+00f }
    , { 1.516577E-01f, 5.904179E-02f, 0.000000E+00f }
    , { 1.427936E-01f, 5.550703E-02f, 0.000000E+00f }
    , { 1.343737E-01f, 5.216139E-02f, 0.000000E+00f }
    , { 1.263808E-01f, 4.899699E-02f, 0.000000E+00f }
    , { 1.187979E-01f, 4.600578E-02f, 0.000000E+00f }
    , { 1.116088E-01f, 4.317885E-02f, 0.000000E+00f }
    , { 1.047975E-01f, 4.050755E-02f, 0.000000E+00f }
    , { 9.834835E-02f, 3.798376E-02f, 0.000000E+00f }
    , { 9.224597E-02f, 3.559982E-02f, 0.000000E+00f }
    , { 8.647506E-02f, 3.334856E-02f, 0.000000E+00f }
    , { 8.101986E-02f, 3.122332E-02f, 0.000000E+00f }
    , { 7.586514E-02f, 2.921780E-02f, 0.000000E+00f }
    , { 7.099633E-02f, 2.732601E-02f, 0.000000E+00f }
    , { 6.639960E-02f, 2.554223E-02f, 0.000000E+00f }
    , { 6.206225E-02f, 2.386121E-02f, 0.000000E+00f }
    , { 5.797409E-02f, 2.227859E-02f, 0.000000E+00f }
    , { 5.412533E-02f, 2.079020E-02f, 0.000000E+00f }
    , { 5.050600E-02f, 1.939185E-02f, 0.000000E+00f }
    , { 4.710606E-02f, 1.807939E-02f, 0.000000E+00f }
    , { 4.391411E-02f, 1.684817E-02f, 0.000000E+00f }
    , { 4.091411E-02f, 1.569188E-02f, 0.000000E+00f }
    , { 3.809067E-02f, 1.460446E-02f, 0.000000E+00f }
    , { 3.543034E-02f, 1.358062E-02f, 0.000000E+00f }
    , { 3.292138E-02f, 1.261573E-02f, 0.000000E+00f }
    , { 3.055672E-02f, 1.170696E-02f, 0.000000E+00f }
    , { 2.834146E-02f, 1.085608E-02f, 0.000000E+00f }
    , { 2.628033E-02f, 1.006476E-02f, 0.000000E+00f }
    , { 2.437465E-02f, 9.333376E-03f, 0.000000E+00f }
    , { 2.262306E-02f, 8.661284E-03f, 0.000000E+00f }
    , { 2.101935E-02f, 8.046048E-03f, 0.000000E+00f }
    , { 1.954647E-02f, 7.481130E-03f, 0.000000E+00f }
    , { 1.818727E-02f, 6.959987E-03f, 0.000000E+00f }
```

```
      , { 1.692727E-02f, 6.477070E-03f, 0.000000E+00f }
      , { 1.575417E-02f, 6.027677E-03f, 0.000000E+00f }
      , { 1.465854E-02f, 5.608169E-03f, 0.000000E+00f }
      , { 1.363571E-02f, 5.216691E-03f, 0.000000E+00f }
315   , { 1.268205E-02f, 4.851785E-03f, 0.000000E+00f }
      , { 1.179394E-02f, 4.512008E-03f, 0.000000E+00f }
      , { 1.096778E-02f, 4.195941E-03f, 0.000000E+00f }
      , { 1.019964E-02f, 3.902057E-03f, 0.000000E+00f }
      , { 9.484317E-03f, 3.628371E-03f, 0.000000E+00f }
320   , { 8.816851E-03f, 3.373005E-03f, 0.000000E+00f }
      , { 8.192921E-03f, 3.134315E-03f, 0.000000E+00f }
      , { 7.608750E-03f, 2.910864E-03f, 0.000000E+00f }
      , { 7.061391E-03f, 2.701528E-03f, 0.000000E+00f }
      , { 6.549509E-03f, 2.505796E-03f, 0.000000E+00f }
325   , { 6.071970E-03f, 2.323231E-03f, 0.000000E+00f }
      , { 5.627476E-03f, 2.153333E-03f, 0.000000E+00f }
      , { 5.214608E-03f, 1.995557E-03f, 0.000000E+00f }
      , { 4.831848E-03f, 1.849316E-03f, 0.000000E+00f }
      , { 4.477579E-03f, 1.713976E-03f, 0.000000E+00f }
330   , { 4.150166E-03f, 1.588899E-03f, 0.000000E+00f }
      , { 3.847988E-03f, 1.473453E-03f, 0.000000E+00f }
      , { 3.569452E-03f, 1.367022E-03f, 0.000000E+00f }
      , { 3.312857E-03f, 1.268954E-03f, 0.000000E+00f }
      , { 3.076022E-03f, 1.178421E-03f, 0.000000E+00f }
335   , { 2.856894E-03f, 1.094644E-03f, 0.000000E+00f }
      , { 2.653681E-03f, 1.016943E-03f, 0.000000E+00f }
      , { 2.464821E-03f, 9.447269E-04f, 0.000000E+00f }
      , { 2.289060E-03f, 8.775171E-04f, 0.000000E+00f }
      , { 2.125694E-03f, 8.150438E-04f, 0.000000E+00f }
340   , { 1.974121E-03f, 7.570755E-04f, 0.000000E+00f }
      , { 1.833723E-03f, 7.033755E-04f, 0.000000E+00f }
      , { 1.703876E-03f, 6.537050E-04f, 0.000000E+00f }
      , { 1.583904E-03f, 6.078048E-04f, 0.000000E+00f }
      , { 1.472939E-03f, 5.653435E-04f, 0.000000E+00f }
345   , { 1.370151E-03f, 5.260046E-04f, 0.000000E+00f }
      , { 1.274803E-03f, 4.895061E-04f, 0.000000E+00f }
      , { 1.186238E-03f, 4.555970E-04f, 0.000000E+00f }
      , { 1.103871E-03f, 4.240548E-04f, 0.000000E+00f }
      , { 1.027194E-03f, 3.946860E-04f, 0.000000E+00f }
350   , { 9.557493E-04f, 3.673178E-04f, 0.000000E+00f }
      , { 8.891262E-04f, 3.417941E-04f, 0.000000E+00f }
      , { 8.269535E-04f, 3.179738E-04f, 0.000000E+00f }
      , { 7.689351E-04f, 2.957441E-04f, 0.000000E+00f }
      , { 7.149425E-04f, 2.750558E-04f, 0.000000E+00f }
355   , { 6.648590E-04f, 2.558640E-04f, 0.000000E+00f }
      , { 6.185421E-04f, 2.381142E-04f, 0.000000E+00f }
      , { 5.758303E-04f, 2.217445E-04f, 0.000000E+00f }
      , { 5.365046E-04f, 2.066711E-04f, 0.000000E+00f }
      , { 5.001842E-04f, 1.927474E-04f, 0.000000E+00f }
360   , { 4.665005E-04f, 1.798315E-04f, 0.000000E+00f }
      , { 4.351386E-04f, 1.678023E-04f, 0.000000E+00f }
      , { 4.058303E-04f, 1.565566E-04f, 0.000000E+00f }
      , { 3.783733E-04f, 1.460168E-04f, 0.000000E+00f }
      , { 3.526892E-04f, 1.361535E-04f, 0.000000E+00f }
365   , { 3.287199E-04f, 1.269451E-04f, 0.000000E+00f }
      , { 3.063998E-04f, 1.183671E-04f, 0.000000E+00f }
      , { 2.856577E-04f, 1.103928E-04f, 0.000000E+00f }
```

```
      , { 2.664108E-04f, 1.029908E-04f, 0.000000E+00f }
      , { 2.485462E-04f, 9.611836E-05f, 0.000000E+00f }
370   , { 2.319529E-04f, 8.973323E-05f, 0.000000E+00f }
      , { 2.165300E-04f, 8.379694E-05f, 0.000000E+00f }
      , { 2.021853E-04f, 7.827442E-05f, 0.000000E+00f }
      , { 1.888338E-04f, 7.313312E-05f, 0.000000E+00f }
      , { 1.763935E-04f, 6.834142E-05f, 0.000000E+00f }
375   , { 1.647895E-04f, 6.387035E-05f, 0.000000E+00f }
      , { 1.539542E-04f, 5.969389E-05f, 0.000000E+00f }
      , { 1.438270E-04f, 5.578862E-05f, 0.000000E+00f }
      , { 1.343572E-04f, 5.213509E-05f, 0.000000E+00f }
      , { 1.255141E-04f, 4.872179E-05f, 0.000000E+00f }
380   , { 1.172706E-04f, 4.553845E-05f, 0.000000E+00f }
      , { 1.095983E-04f, 4.257443E-05f, 0.000000E+00f }
      , { 1.024685E-04f, 3.981884E-05f, 0.000000E+00f }
      , { 9.584715E-05f, 3.725877E-05f, 0.000000E+00f }
      , { 8.968316E-05f, 3.487467E-05f, 0.000000E+00f }
385   , { 8.392734E-05f, 3.264765E-05f, 0.000000E+00f }
      , { 7.853708E-05f, 3.056140E-05f, 0.000000E+00f }
      , { 7.347551E-05f, 2.860175E-05f, 0.000000E+00f }
      , { 6.871576E-05f, 2.675841E-05f, 0.000000E+00f }
      , { 6.425257E-05f, 2.502943E-05f, 0.000000E+00f }
390   , { 6.008292E-05f, 2.341373E-05f, 0.000000E+00f }
      , { 5.620098E-05f, 2.190914E-05f, 0.000000E+00f }
      , { 5.259870E-05f, 2.051259E-05f, 0.000000E+00f }
      , { 4.926279E-05f, 1.921902E-05f, 0.000000E+00f }
      , { 4.616623E-05f, 1.801796E-05f, 0.000000E+00f }
395   , { 4.328212E-05f, 1.689899E-05f, 0.000000E+00f }
      , { 4.058715E-05f, 1.585309E-05f, 0.000000E+00f }
      , { 3.806114E-05f, 1.487243E-05f, 0.000000E+00f }
      , { 3.568818E-05f, 1.395085E-05f, 0.000000E+00f }
      , { 3.346023E-05f, 1.308528E-05f, 0.000000E+00f }
400   , { 3.137090E-05f, 1.227327E-05f, 0.000000E+00f }
      , { 2.941371E-05f, 1.151233E-05f, 0.000000E+00f }
      , { 2.758222E-05f, 1.080001E-05f, 0.000000E+00f }
      , { 2.586951E-05f, 1.013364E-05f, 0.000000E+00f }
      , { 2.426701E-05f, 9.509919E-06f, 0.000000E+00f }
405   , { 2.276639E-05f, 8.925630E-06f, 0.000000E+00f }
      , { 2.136009E-05f, 8.377852E-06f, 0.000000E+00f }
      , { 2.004122E-05f, 7.863920E-06f, 0.000000E+00f }
      , { 1.880380E-05f, 7.381539E-06f, 0.000000E+00f }
      , { 1.764358E-05f, 6.929096E-06f, 0.000000E+00f }
410   , { 1.655671E-05f, 6.505136E-06f, 0.000000E+00f }
      , { 1.553939E-05f, 6.108221E-06f, 0.000000E+00f }
      , { 1.458792E-05f, 5.736935E-06f, 0.000000E+00f }
      , { 1.369853E-05f, 5.389831E-06f, 0.000000E+00f }
      , { 1.286705E-05f, 5.065269E-06f, 0.000000E+00f }
415   , { 1.208947E-05f, 4.761667E-06f, 0.000000E+00f }
      , { 1.136207E-05f, 4.477561E-06f, 0.000000E+00f }
      , { 1.068141E-05f, 4.211597E-06f, 0.000000E+00f }
      , { 1.004411E-05f, 3.962457E-06f, 0.000000E+00f }
      , { 9.446399E-06f, 3.728674E-06f, 0.000000E+00f }
420   , { 8.884754E-06f, 3.508881E-06f, 0.000000E+00f }
      , { 8.356050E-06f, 3.301868E-06f, 0.000000E+00f }
      , { 7.857521E-06f, 3.106561E-06f, 0.000000E+00f }
      , { 7.386996E-06f, 2.922119E-06f, 0.000000E+00f }
      , { 6.943576E-06f, 2.748208E-06f, 0.000000E+00f }
```

```
425     , { 6.526548E-06f, 2.584560E-06f, 0.000000E+00f }
        , { 6.135087E-06f, 2.430867E-06f, 0.000000E+00f }
        , { 5.768284E-06f, 2.286786E-06f, 0.000000E+00f }
        , { 5.425069E-06f, 2.151905E-06f, 0.000000E+00f }
        , { 5.103974E-06f, 2.025656E-06f, 0.000000E+00f }
430     , { 4.803525E-06f, 1.907464E-06f, 0.000000E+00f }
        , { 4.522350E-06f, 1.796794E-06f, 0.000000E+00f }
        , { 4.259166E-06f, 1.693147E-06f, 0.000000E+00f }
        , { 4.012715E-06f, 1.596032E-06f, 0.000000E+00f }
        , { 3.781597E-06f, 1.504903E-06f, 0.000000E+00f }
435     , { 3.564496E-06f, 1.419245E-06f, 0.000000E+00f }
        , { 3.360236E-06f, 1.338600E-06f, 0.000000E+00f }
        , { 3.167765E-06f, 1.262556E-06f, 0.000000E+00f }
        , { 2.986206E-06f, 1.190771E-06f, 0.000000E+00f }
        , { 2.814999E-06f, 1.123031E-06f, 0.000000E+00f }
440     , { 2.653663E-06f, 1.059151E-06f, 0.000000E+00f }
        , { 2.501725E-06f, 9.989507E-07f, 0.000000E+00f }
        , { 2.358723E-06f, 9.422514E-07f, 0.000000E+00f }
        , { 2.224206E-06f, 8.888804E-07f, 0.000000E+00f }
        , { 2.097737E-06f, 8.386690E-07f, 0.000000E+00f }
445     , { 1.978894E-06f, 7.914539E-07f, 0.000000E+00f }
        , { 1.867268E-06f, 7.470770E-07f, 0.000000E+00f }
        , { 1.762465E-06f, 7.053860E-07f, 0.000000E+00f }
        };

450 #endif
```

10 cc/Makefile

```
# directory containing unzipped refractive index database
# <http://refractiveindex.info/download/database/rii-database-2015-03-11.zip>
RII = "$(HOME)/refractiveindex.info"

5   COMPILE = g++ -std=c++11 -Wall -Wextra -pedantic -O3 -march=native -fopenmp

    all: prismatic prism2xyz xyz2srgb srgb2png

    prismatic: prismatic.cc lin2012xyz2e_1_7sf.h water.h glass.h quartz.h sapphire.h
        ↳ format.cc
10          $(COMPILE) -o prismatic prismatic.cc

    prism2xyz: prism2xyz.cc lin2012xyz2e_1_7sf.h format.cc
            $(COMPILE) -o prism2xyz prism2xyz.cc

15  xyz2png: xyz2png.cc format.cc d50_xyz.s D50_XYZ.icc
            $(COMPILE) -o xyz2png xyz2png.cc d50_xyz.s `pkg-config --cflags --libs
                ↳ libpng`

    xyz2srgb: xyz2srgb.cc format.cc
            $(COMPILE) -o xyz2srgb xyz2srgb.cc
20
    srgb2png: srgb2png.cc format.cc
            $(COMPILE) -o srgb2png srgb2png.cc `pkg-config --cflags --libs libpng`

    chromaticity-diagram: chromaticity-diagram.cc
25          $(COMPILE) -o chromaticity-diagram chromaticity-diagram.cc `pkg-config
                ↳ --cflags --libs libpng`
```

```
#%.h: %.sh
#	./$< >$@ $(RII)

clean:
	-rm -f prismatic prism2xyz xyz2srgb srgb2png xyz2png chromaticity-diagram
```

11 cc/prism2xyz.cc

```
#include <cmath>
#include <cstdio>
#include <cstdlib>
using namespace std;

#include "format.cc"
#include "lin2012xyz2e_1_7sf.h"

int main() {
  bool incremental = false;
  float *xyz = 0;
  float *a = 0;
  float *scanline = 0;
  int allocated_width = -1;
  int allocated_height = -1;
  int width = 0;
  int height = 0;
  float wavelength = 0;
  float white_x = 0;
  float white_y = 0;
  float white_z = 0;
  float white_a = 0;
  // for each wavelength
  int w_count = 0;
  while (read_header_intensity(stdin, &width, &height, &wavelength)) {
    fprintf(stderr, "%8d\r", w_count++);
    // allocate if necessary
    if (allocated_width < 0 && allocated_height < 0) {
      if (width > 0 && height > 0) {
        int bytes = width * height * 3 * sizeof(float);
        xyz = (float *) calloc(bytes, 1);
        if (! xyz) {
          fprintf(stderr, "error: memory allocation failed: %d bytes\n", bytes);
          return 1;
        }
        bytes = width * height * sizeof(float);
        a = (float *) calloc(bytes, 1);
        if (! a) {
          fprintf(stderr, "error: memory allocation failed: %d bytes\n", bytes);
          return 1;
        }
        bytes = width * sizeof(float);
        if (incremental) {
          bytes *= 3;
        }
        scanline = (float *) calloc(bytes, 1);
        if (! scanline) {
```

```
          fprintf(stderr, "error: memory allocation failed: %d bytes\n", bytes);
          return 1;
        }
        allocated_width = width;
        allocated_height = height;
      } else {
        fprintf(stderr, "error: bad image size: %d %d\n", width, height);
        return 1;
      }
    } // allocate if necessary
    // check image size
    if (width != allocated_width || height != allocated_height) {
      fprintf
        ( stderr
        , "error: image size mismatch: %d != %d || %d != %d\n"
        , width, allocated_width, height, allocated_height
        );
      return 1;
    }
    // compute colour
    float w  = wavelength - wavelength_min;
    int   i0 = floor(w);
    int   i1 = i0 + 1;
    float f1 = w - i0;
    float f0 = 1 - f1;
    if (f1 == 0) {
      i1 = i0;
    }
    if (! (0 <= i0 && 0 <= i1 && i1 <= wavelength_max - wavelength_min)) {
      fprintf
        ( stderr
        , "warning: skipping wavelength outside range: %f (%d, %d)\n"
        , wavelength, wavelength_min, wavelength_max
        );
      continue;
    }
    auto xyz0 = tristimulus_curve[i0];
    auto xyz1 = tristimulus_curve[i1];
    float x = f0 * xyz0[0] + f1 * xyz1[0];
    float y = f0 * xyz0[1] + f1 * xyz1[1];
    float z = f0 * xyz0[2] + f1 * xyz1[2];
    white_x += x;
    white_y += y;
    white_z += z;
    white_a += 1.0f;
    // for each scanline
    for (int j = 0; j < height; ++j) {
      read_data_intensity(stdin, width, scanline);
      #pragma omp parallel for schedule(static)
      for (int i = 0; i < width; ++i) {
        int k = j * width + i;
        float intensity = scanline[i];
        if (0.0f <= intensity && intensity < 1.0f / 0.0f) {
          xyz[3*k+0] += intensity * x;
          xyz[3*k+1] += intensity * y;
          xyz[3*k+2] += intensity * z;
          a[k] += 1.0f;
```

```
105           }
            }
          } // for each scanline
          if (incremental) {
            // compute white point
110         float w_x = white_x / white_a;
            float w_y = white_y / white_a;
            float w_z = white_z / white_a;
            float s = w_x + w_y + w_z;
            float cx = w_x / s;
115         float cy = w_y / s;
            float cyy = w_y;
            fprintf(stderr, "info: white point: %f %f %f\n", cx, cy, cyy);
            write_header_xyz(stdout, width, height);
            // normalize image
120         for (int j = 0; j < height; ++j) {
              #pragma omp parallel for schedule(static)
              for (int i = 0; i < width; ++i) {
                int k = j * width + i;
                float alpha = a[k];
125             if (alpha > 0.0f) {
                  scanline[3*i+0] = xyz[3*k+0] / (w_x * alpha);
                  scanline[3*i+1] = xyz[3*k+1] / (w_y * alpha);
                  scanline[3*i+2] = xyz[3*k+2] / (w_z * alpha);
                } else {
130               scanline[3*i+0] = 0.0f;
                  scanline[3*i+1] = 0.0f;
                  scanline[3*i+2] = 0.0f;
                }
              }
135           write_data_xyz(stdout, width, scanline);
            }
          } // if incremental
        } // for each wavelength
        if (! incremental) {
140       // compute white point
          float w_x = white_x / white_a;
          float w_y = white_y / white_a;
          float w_z = white_z / white_a;
          float s = w_x + w_y + w_z;
145       float cx = w_x / s;
          float cy = w_y / s;
          float cyy = w_y;
          fprintf(stderr, "info: white point: %f %f %f\n", cx, cy, cyy);
          // normalize image
150       int count = width * height;
          #pragma omp parallel for schedule(static)
          for (int k = 0; k < count; ++k) {
            float alpha = a[k];
            if (alpha > 0.0f) {
155           xyz[3*k+0] /= w_x * alpha;
              xyz[3*k+1] /= w_y * alpha;
              xyz[3*k+2] /= w_z * alpha;
            }
          }
160       // write image
          write_header_xyz(stdout, width, height);
```

```
      write_data_xyz(stdout, count, xyz);
    }
    return 0;
}
```

12 cc/prismatic.cc

```
// feature test macro requirement for glibc clock_gettime()
#define _POSIX_C_SOURCE 199309L
#include <time.h>

#include "lin2012xyz2e_1_7sf.h"
#include <glm/glm.hpp>
using namespace glm;
using namespace std;
#include "water.h"
#include "glass.h"
#include "quartz.h"
/*
#include "sapphire.h"
*/
#include "format.cc"

#ifndef IMAGE_SIZE
#define IMAGE_SIZE 5
#endif

#ifndef DEPTH_MAX
#define DEPTH_MAX 4
#endif

// { config {
const int   image_size = IMAGE_SIZE;
const int   image_width  = 22 << image_size;
const int   image_height =  7 << image_size;
const int   frames = 1; // 375;
const int   fps = 25;
const float animation_duration = 12.0f;
const int   animation_spin = -1;
const int   animation_push =  1;
/*
const float view_XY = 30.0f;
const float view_XZ =  0.0f;
const float view_XW = 30.0f;
const float view_YZ =  0.0f;
const float view_YW = 30.0f;
const float view_ZW =  0.0f;
*/
const float sphere_size = 2.0f/3.0f;
/*
const float light_size = 0.1f;
*/
const int   depth_max = DEPTH_MAX;

const int   wavelength_start = 556; // peak brightness
const int   wavelength_count = 441; // number of wavelengths to trace
const int   wavelength_wrap = 441;
```

```
const int   wavelength_step = 169;  // important: gcd(wrap, step) = 1
const int   wavelength_lo = wavelength_min;
const int   wavelength_hi = wavelength_lo + wavelength_wrap;
// } config }

const int   spheres = 24;
const int   lights = 1;

// advance rays away from surface intersection
const float acne = 0.0001f;
const float cacne = cos(acne);
const float sacne = sin(acne);

const float pi = 3.141592653589793f;


typedef unsigned char uint8;

/*
// <https://commons.wikimedia.org/wiki/File:Lambert_cylindrical_equal-↙
    ↪ area_projection_SW.jpg>
const int   earth_width  = 2044;
const int   earth_height =  650;
uint8 earth_srgb[earth_height][earth_width][3];

float earth_phase = 0.0f;
*/
float earth(vec4 ray, float wavelength) {
  (void) wavelength;
  vec3 dir(ray);
  {
    float t = pi / 3.0f;
    float c = cos(t);
    float s = sin(t);
    float z = 0.0f;
    float i = 1.0f;
    dir = mat3(c, s, z, -s, c, z, z, z, i) * dir;
  }
  dir = normalize(dir);
  float x = atan(dir[0], dir[2]) / pi;
  float y = dir[1];
  y = asin(y) / pi;
  bool  k = (fmod(12.0f * (x + 1.0f), 1.0f) > 0.5f)
         != (fmod(12.0f * (y + 1.0f), 1.0f) > 0.5f);
  return k ? 1.0f : 1.0f/64.0f;
/*
  int   i = (x / 2.0f + 0.5f + earth_phase) * earth_width;
  int   j = (y / 2.0f + 0.5f) * earth_height;
  i = (i + earth_width) % earth_width;
  if (j < 0) { j = 0; }
  if (j >= earth_height) { j = earth_height - 1; }
  auto e = earth_srgb[j][i];
  // <https://en.wikipedia.org/wiki/SRGB#The_reverse_transformation>
  vec3 rgb(0.0f);
  for (int c = 0; c < 3; ++c) {
    rgb[c] = e[c] / 255.0f;
    if (rgb[c] <= 0.04045f) {
```

```
      rgb[c] /= 12.92;
    } else {
      rgb[c] = pow((rgb[c] + 0.055f) / 1.055f, 2.4f);
    }
  }
  vec3 exyz(transpose(mat3
    ( 0.4124f, 0.3576f, 0.1805f
    , 0.2126f, 0.7152f, 0.0722f
    , 0.0193f, 0.1192f, 0.9505f
    )) * rgb);
  auto stimulus = tristimulus_curve[int(wavelength) - wavelength_min];
  vec3 xyz(stimulus[0], stimulus[1], stimulus[2]);
  return dot(xyz, exyz);
*/
}

/*
vec4  image_xyza[image_height][image_width];
uint8 image_srgb[image_height][image_width][3];
*/
float image_intensity[image_height][image_width];

struct material {
  bool  light;
  float index;
  float emission;
  float transmission;
};

enum material_t
  { material_vacuum = 0
  , material_water
  , material_quartz
  , material_glass
  , material_white
  , material_red
  , material_yellow
  , material_green
  , material_blue
  , material_earth
  , material_count
  };

material materials[material_count];


struct sphere {
  vec4       center;
  float      radius;
  float      sradius;
  float      cradius;
  material_t material;
};

sphere scene[spheres + lights];

inline vec4 cross(const vec4 &a, const vec4 &b, const vec4 &c) {
```

```
  // <http://www.gamedev.net/topic/456301-cross-product-vector-4d/#entry4011062>
  float ax = a[0];
  float ay = a[1];
  float az = a[2];
  float aw = a[3];
  float bx = b[0];
  float by = b[1];
  float bz = b[2];
  float bw = b[3];
  float cx = c[0];
  float cy = c[1];
  float cz = c[2];
  float cw = c[3];
  float dx =  ay*(bz*cw - cz*bw) - az*(by*cw - cy*bw) + aw*(by*cz - cy*bz);
  float dy = -ax*(bz*cw - cz*bw) + az*(bx*cw - cx*bw) - aw*(bx*cz - cx*bz);
  float dz =  ax*(by*cw - cy*bw) - ay*(bx*cw - cx*bw) + aw*(bx*cy - cx*by);
  float dw = -ax*(by*cz - cy*bz) + ay*(bx*cz - cx*bz) - az*(bx*cy - cx*by);
  return vec4(dx, dy, dz, dw);
}

inline mat4 transport(const vec4 &target) {
  vec4 a(1.0f, 0.0f, 0.0f, 0.0f);
  vec4 b(0.0f, 1.0f, 0.0f, 0.0f);
  vec4 c(0.0f, 0.0f, 1.0f, 0.0f);
  vec4 d(target);
  a = normalize(cross(b, c, d));
  b = normalize(cross(c, d, a));
  c = normalize(cross(d, a, b));
  return mat4(a, b, c, d);
}

inline mat4 lookAt(const vec4 &target, const vec4 &eye) {
  return transpose(transport(eye)) * transport(target);
}


inline float dot(const vec4 &a, const vec4 &b) {
  float ax = a[0];
  float ay = a[1];
  float az = a[2];
  float aw = a[3];
  float bx = b[0];
  float by = b[1];
  float bz = b[2];
  float bw = b[3];
  return ax * bx + ay * by + az * bz + aw * bw;
}

inline vec4 reflect(const vec4 &I, const vec4 &N, float dotNI) {
  return I - (2.0f * dotNI) * N;
}

inline vec4 refract(const vec4 &I, const vec4 &N, float eta, float dotNI) {
  float k = 1.0f - eta * eta * (1.0f - dotNI * dotNI);
  if (k < 0.0f) {
    return vec4(0.0f);
  } else {
```

```
    return eta * I - (eta * dotNI + sqrt(k)) * N;
  }
}

// Fresnel equations
// I = incident ray
// N = surface normal
// eta = n1 / n2
// result.x = reflected power
// result.y = transmitted power
inline vec2 fresnel(float dotNI, float eta) {
  float c1 = -dotNI;
  float c2 = 1.0f - (1.0f - c1 * c1) * (eta * eta);
  if (c2 < 0.0f) {
    // total internal reflection
    return vec2(1.0f, 0.0f);
  }
  c2 = sqrt(c2);
  float Rs = (eta * c1 - c2) / (eta * c1 + c2);
  float Rp = (eta * c2 - c1) / (eta * c2 + c1);
  float R = 0.5f * (Rs * Rs + Rp * Rp);
  float T = 1.0f - R;
  return vec2(R, T);
}


// spherical tangent space
// points and directions are unit length
// directions from the pole are equivalent to points on the equator
// result is direction towards "to" orthogonal to "from"
inline vec4 tangent(const vec4 &to, const vec4 &from, float c, float s) {
  return (to - c * from) * (1.0f / s);
}

inline vec4 tangent(const vec4 &to, const vec4 &from, float dotTF) {
  return tangent(to, from, dotTF, sqrt(1.0f - dotTF * dotTF));
}

inline vec4 tangent(const vec4 &to, const vec4 &from) {
  return tangent(to, from, dot(to, from));
}


mat4 initialize_view() {
/*
  float cXY = cos(radians(view_XY));
  float sXY = sin(radians(view_XY));
  float cXZ = cos(radians(view_XZ));
  float sXZ = sin(radians(view_XZ));
  float cXW = cos(radians(view_XW));
  float sXW = sin(radians(view_XW));
  float cYZ = cos(radians(view_YZ));
  float sYZ = sin(radians(view_YZ));
  float cYW = cos(radians(view_YW));
  float sYW = sin(radians(view_YW));
  float cZW = cos(radians(view_ZW));
```

```
  float sZW = sin(radians(view_ZW));
  mat4 mXYZW = mat4
    (  cXY, sXY, 0.0f, 0.0f
    , -sXY, cXY, 0.0f, 0.0f
    , 0.0f, 0.0f,  cZW, sZW
    , 0.0f, 0.0f, -sZW, cZW
    );
  mat4 mXZYW = mat4
    (  cXZ, 0.0f, sXZ, 0.0f
    , 0.0f,  cYW, 0.0f, sYW
    , -sXZ, 0.0f, cXZ, 0.0f
    , 0.0f, -sYW, 0.0f, cYW
    );
  mat4 mXWYZ = mat4
    (  cXW, 0.0f, 0.0f, sXW
    , 0.0f,  cYZ, sYZ, 0.0f
    , 0.0f, -sYZ, cYZ, 0.0f
    , -sXW, 0.0f, 0.0f, cXW
    );
  return mXYZW * mXZYW * mXWYZ;
  float p = sqrt(0.5f);
  float z = 0.0f;
  return lookAt(vec4(z,z,p,p), vec4(z,p,z,p));
*/
  float t = radians(160.0f);
  float c = cos(t);
  float s = sin(t);
  float z = 0.0f;
  float i = 1.0f;
  mat4  m(c,s,z,z,-s,c,z,z,z,z,i,z,z,z,z,i);
  return m * lookAt(vec4(0.5,0.5,0.5,0.5), normalize(vec4(0.1,0.5,-0.1,1.0)));
}

mat4 initialize_animation(float time) {
  float t = 2.0f * pi * time / animation_duration;
  float cSpin = cos(float(animation_spin) * t);
  float sSpin = sin(float(animation_spin) * t);
  float cPush = cos(float(animation_push) * t);
  float sPush = sin(float(animation_push) * t);
  return mat4
    (  cSpin, sSpin, 0.0f, 0.0f
    , -sSpin, cSpin, 0.0f, 0.0f
    , 0.0f, 0.0f,  cPush, sPush
    , 0.0f, 0.0f, -sPush, cPush
    );
}


vec4 lambert_cylindrical_equal_area_projection(float x, float y) {
  float sx = -sin(x);
  float cx = -cos(x);
  float sy = y;
  float cy = sqrt(1.0f - sy * sy);
  return vec4(cy * sx, sy, cy * cx, 0.0f);
}

/*
```

```
335 float simple_index(float wavelength) {
      return 2.5f - 0.75f
        * (wavelength     - wavelength_min)
        / (wavelength_max - wavelength_min);
    }
340

    float simple_spectrum(float wavelength_peak, float quality, float wavelength) {
      float delta = quality
        * (wavelength     - wavelength_peak)
345     / (wavelength_max - wavelength_min);
      return 1.0f / (1.0f + delta * delta);
    }
    */

350 void initialize_materials(float wavelength) {
      materials[material_vacuum].light        = false;
      materials[material_vacuum].index        = 1.0f;
      materials[material_vacuum].emission     = 0.0f;
      materials[material_vacuum].transmission = 1.0f;
355   materials[material_water ].light        = false;
      materials[material_water ].index        = water(wavelength, 0);
      materials[material_water ].emission     = 0.0f;
      materials[material_water ].transmission =
          exp(-4.0f * pi * water(wavelength, 1) / (wavelength * 1.0e-9f));
360   materials[material_glass ].light        = false;
      materials[material_glass ].index        = glass_n(wavelength);
      materials[material_glass ].emission     = 0.0f;
      materials[material_glass ].transmission =
          exp(-4.0f * pi * glass_k(wavelength) / (wavelength * 1.0e-9f));
365   materials[material_quartz].light        = false;
      materials[material_quartz].index        = quartz(wavelength, 0);
      materials[material_quartz].emission     = 0.0f;
      materials[material_quartz].transmission =
          exp(-4.0f * pi * quartz(wavelength, 1) / (wavelength * 1.0e-9f));
370 /*
      materials[material_white ].light        = true;
      materials[material_white ].index        = 1.0f;
      materials[material_white ].emission     = 8.0f;
      materials[material_white ].transmission = 1.0f;
375   materials[material_red   ].light        = true;
      materials[material_red   ].index        = 1.0f;
      materials[material_red   ].emission     = simple_spectrum(675, 2, wavelength);
      materials[material_red   ].transmission = 1.0f;
      materials[material_yellow].light        = true;
380   materials[material_yellow].index        = 1.0f;
      materials[material_yellow].emission     = simple_spectrum(575, 2, wavelength);
      materials[material_yellow].transmission = 1.0f;
      materials[material_green ].light        = true;
      materials[material_green ].index        = 1.0f;
385   materials[material_green ].emission     = simple_spectrum(525, 2, wavelength);
      materials[material_green ].transmission = 1.0f;
      materials[material_blue  ].light        = true;
      materials[material_blue  ].index        = 1.0f;
      materials[material_blue  ].emission     = simple_spectrum(425, 2, wavelength);
390   materials[material_blue  ].transmission = 1.0f;
    */
```

```
  materials[material_earth ].light        = true;
  materials[material_earth ].index        = 1.0f;
  materials[material_earth ].emission     = 0.0f; // magic
  materials[material_earth ].transmission = 0.0f;
}


void initialize_scene(mat4 animation, mat4 view) {
  mat4 rotation(animation * view);
  // at pi/6 the spheres just touch, so sphere_size is in 0..1
  float sphere_radius  = sphere_size * pi / 6.0f;
  float sphere_sradius = sin(sphere_radius);
  float sphere_cradius = cos(sphere_radius);
/*
  // the light must be smaller than the sphere, so light size is in 0..1
  float light_radius = light_size * sphere_radius;
  float light_sradius = sin(light_radius);
  float light_cradius = cos(light_radius);
*/
  int k = 0;
  {
/*
    float p = sqrt(0.5f);
    float z = 0.0f;
*/
    vec4  q = animation
/*
            * inverse(lookAt(vec4(z,z,p,p), vec4(z,p,z,p)))
*/
            * vec4(0.0f, 0.0f, 0.0f, 1.0f);
    scene[k].center   = -q;
    scene[k].radius   = pi / 2.0f;
    scene[k].sradius  = 1.0f;
    scene[k].cradius  = 0.0f;
    scene[k].material = material_earth;
    k = k + 1;
  }
  // 24-cell has the vertices of a cube and a cross
  // three sets of eight vertices, each set in its own colour
  // cube
  for (int sx = -1; sx <= 1; sx += 2) {
  for (int sy = -1; sy <= 1; sy += 2) {
  for (int sz = -1; sz <= 1; sz += 2) {
  for (int sw = -1; sw <= 1; sw += 2) {
    bool odd = ((sx + sy + sz + sw) & 2) == 2;
    vec4 p  = rotation
            * (0.5f * vec4(float(sx), float(sy), float(sz), float(sw)));
    scene[k].center   = p;
    scene[k].radius   = sphere_radius;
    scene[k].sradius  = sphere_sradius;
    scene[k].cradius  = sphere_cradius;
    scene[k].material = odd ? material_quartz : material_glass;
/*
    int l = k + spheres;
    scene[l].center   = p;
    scene[l].radius   = light_radius;
    scene[l].sradius  = light_sradius;
```

```
      scene[l].cradius  = light_cradius;
      scene[l].material = material_white; // material_earth;
    */
      k = k + 1;
    }}}}
    // cross
    for (int sd = 0; sd < 4; sd += 1) {
    for (int sv = -1; sv <= 1; sv += 2) {
      vec4 p = vec4(0.0f);
      p[sd] = float(sv);
      p = rotation * p;
      scene[k].center   = p;
      scene[k].radius   = sphere_radius;
      scene[k].sradius  = sphere_sradius;
      scene[k].cradius  = sphere_cradius;
      scene[k].material = material_water;
    /*
      int l = k + spheres;
      scene[l].center   = p;
      scene[l].radius   = light_radius;
      scene[l].sradius  = light_sradius;
      scene[l].cradius  = light_cradius;
      scene[l].material = material_white; // material_earth;
    */
      k = k + 1;
    }}
  }


  material_t material_at(vec4 p) {
    float cdistance = -2.0;
    material_t material = material_vacuum;
    for (int k = 0; k < spheres + lights; ++k) {
      float cd = dot(p, scene[k].center);
      if (cd > scene[k].cradius && cd > cdistance) {
        material = scene[k].material;
      }
    }
    return material;
  }

  /*
  void image_clear_xyza() {
    #pragma omp parallel for schedule(static)
    for (int image_j = 0; image_j < image_height; ++image_j) {
      for (int image_i = 0; image_i < image_width; ++image_i) {
        image_xyza[image_j][image_i] = vec4(0.0f);
      }
    }
  }

  inline float srgb(float c0) {
    float c = clamp(c0, 0.0f, 1.0f);
    if (c <= 0.0031308f) {
      return 12.92f * c;
    } else {
      return 1.055f * pow(c, 1.0f / 2.4f) - 0.055f;
```

```
  }
}

inline vec3 xyz_to_srgb(const vec3 &xyz) {
  vec3 white(0.9505f, 1.0000f, 1.0890f);
  vec3 rgb = transpose(mat3
    (  3.2406f, -1.5372f, -0.4986f
    , -0.9689f,  1.8758f,  0.0415f
    ,  0.0557f, -0.2040f,  1.0570f
    )) * (xyz * white);
  return vec3(srgb(rgb.r), srgb(rgb.g), srgb(rgb.b));
}

void image_xyza_to_srgb() {
  #pragma omp parallel for schedule(static)
  for (int image_j = 0; image_j < image_height; ++image_j) {
    for (int image_i = 0; image_i < image_width; ++image_i) {
      vec4 xyza(image_xyza[image_j][image_i]);
      vec3 srgb(xyz_to_srgb(vec3(xyza) / xyza.a));
      for (int image_c = 0; image_c < 3; ++image_c) {
        image_srgb[image_j][image_i][image_c] =
            uint8(clamp(255.0f * srgb[image_c], 0.0f, 255.0f));
      }
    }
  }
}

void image_save_ppm() {
  printf("P6\n%d %d\n255\n", image_width, image_height);
  fwrite(&image_srgb[0][0][0], image_width * image_height * 3, 1, stdout);
}
*/

// trace rays
float trace
  ( int depth
  , float wavelength
  , material_t material
  , const vec4 &eye
  , const vec4 &ray
  ) {
  if (depth <= 0) {
    return 0.0f;
  }
  // initialize results
  bool       hit         = false;           // did the ray hit anything
  float      hitcdistance = -1.0f / 0.0f;    // keep only the nearest hit
  float      hitsdistance = 0.0f;
  vec4       hitpos      = vec4(0.0f);      // ray-surface intersection point
  vec4       hitnormal   = vec4(0.0f);      // surface normal at intersection
  material_t hitmaterial = material_vacuum; // material beyond the surface
  int        hitk = -1;
  // check every object in the scene
  for (int k = 0; k < spheres + lights; ++k) {
    vec4  center = scene[k].center;
    // r = sphere radius
    float sr     = scene[k].sradius;
```

```
      float cr      = scene[k].cradius;
      // d = distance from eye to center
565   float cd      = dot(eye, center);
      float sd      = sqrt(clamp(1.0f - cd * cd, 0.0f, 1.0f));
      // early rejection test
      float cdr = cd * cr + sd * sr;
      if (cdr < hitcdistance) {
570     // nearest point on sphere = d - r > nearest hit
        continue;
      }

      // theta = angle subtended by the sphere radius
575   float stheta  = sr / sd;
      // phi = angle between ray and sphere center
      vec4  tcenter = tangent(center, eye, cd, sd);
      float cphi    = dot(ray, tcenter);
      if (length(center + eye) < acne) { cphi = 1.0f; }
580   if (length(center - eye) < acne) { cphi = 1.0f; }

      if (stheta > 1.0f) {
        // find the distance to the intersection
        if (cd > cr) {
585       // d < r
          // sphere surrounds eye
          // ray intersects from the inside
          float a = cr;
          float b = cd;
590       float c = sd * cphi;
          // tan (h/2) = ... / ...
          // cos h = (1 - tan^2(h/2)) / (1 + tan^2(h/2))
          // sin h = 2 tan(h/2) / (1 + tan^2(h/2))
          // ... c + sqrt ... because the other intersection is behind
595       float th2 = (c + sqrt(c * c + b * b - a * a)) / (a + b);
          float t2h2 = th2 * th2;
          float t2h2p1 = 1.0f + t2h2;
          float ch = (1.0f - t2h2) / t2h2p1;
          // check if it's closer
600       if (ch > hitcdistance) {
            float sh = 2.0f * th2 / t2h2p1;
            // update results with new details
            hit          = true;
            hitcdistance = ch;
605         hitsdistance = sh;
            // eye and ray are orthogonal unit vectors
            hitpos       = ch * eye + sh * ray;
            // surface normal points towards sphere center
            hitnormal    =  tangent(center, hitpos, cr, sr);
610         // outside the sphere is vacuum
            hitmaterial  = material_vacuum;
            hitk         = k;
          }
        } else {
615       // sphere surrounds -eye
          // ray intersects from the outside
          float a = cr;
          float b = cd;
          float c = sd * cphi;
```

```
620           // ... c - sqrt ... because the other intersection is further
              float th2 = (c - sqrt(c * c + b * b - a * a)) / (a + b);
              float t2h2 = th2 * th2;
              float t2h2p1 = 1.0f + t2h2;
              float ch = (1.0f - t2h2) / t2h2p1;
625           // check if it's closer
              if (ch > hitcdistance) {
                float sh = 2.0f * th2 / t2h2p1;
                // update results with new details
                hit          = true;
630             hitcdistance = ch;
                hitsdistance = sh;
                // eye and ray are orthogonal unit vectors
                hitpos       = ch * eye + sh * ray;
                // surface normal points away from sphere center
635             hitnormal    = -tangent(center, hitpos, cr, sr);
                // inside the sphere is the sphere's material
                hitmaterial  = scene[k].material;
                hitk         = k;
    /*
640             if (depth == depth_max && k == 0) {
                  fprintf(stderr, "\t\t%f\t%f\n", ch, sh);
                }
    */
              }
645         }

          } else {
            // d > r
            // sphere is disjoint from eye, ray might intersect from outside
            float ctheta2 = 1.0f - stheta * stheta;
            if (cphi >= 0.0f && cphi * cphi > ctheta2) {
              // phi < theta
              // ray does intersect
              // find the distance to the intersection
              float a = cr;
              float b = cd;
              float c = sd * cphi;
              // ... c - sqrt ... because the other intersection is further
              float th2 = (c - sqrt(c * c + b * b - a * a)) / (a + b);
              float t2h2 = th2 * th2;
              float t2h2p1 = 1.0f + t2h2;
              float ch = (1.0f - t2h2) / t2h2p1;
              // check if it's closer
              if (ch > hitcdistance) {
                float sh = 2.0f * th2 / t2h2p1;
                // update results with new details
                hit          = true;
                hitcdistance = ch;
                hitsdistance = sh;
                // eye and ray are orthogonal unit vectors
                hitpos       = ch * eye + sh * ray;
                // surface normal points away from sphere center
                hitnormal    = -tangent(center, hitpos, cr, sr);
                // inside the sphere is the sphere's material
                hitmaterial  = scene[k].material;
                hitk         = k;
```

```
        }
      }
    }
  } // for each object in scene

  float intensity = 0.0f;
  if (hit) {
    // attenuation by the near-side material through which the ray travelled
    // FIXME acos()
    float transmission =
        pow(materials[material].transmission, 2.0 * acos(hitcdistance));
    // FIXME non-physical fading with distance
    float fade = 1.0f;//0.5f * (1.0f + hitcdistance);
    // contribution from this ray
    float factor = transmission * fade;
    // incident ray
    vec4 incident = hitcdistance * ray - hitsdistance * eye;
    // Lambert diffuse
    float dotNI = dot(hitnormal, incident);
    float emit = clamp(-dotNI, 0.0f, 1.0f);
    // accumulate this ray
    float emission = 0.0f;
    if (hitmaterial == material_earth) { // magic
      vec4 middle =
          hitpos - dot(hitpos, scene[hitk].center) * scene[hitk].center;
      emission = earth(middle, wavelength);
    } else {
      emission = materials[hitmaterial].emission;
    }
    intensity += factor * emit * emission;
    if ((! materials[hitmaterial].light) && depth > 1 && factor > 0.0f) {
      float eta = materials[material].index / materials[hitmaterial].index;
      // Fresnel equations for intensity of reflection and refraction
      vec2 f = fresnel(dotNI, eta);
      if (f.x > 0.0f) {
        // reflected ray travels through the near side material
        // incident and hitnormal are both tangent to hitpos,
        //    so too will be the reflected ray
        // incident and hitnormal are both unit,
        //    so too will be the reflected ray
        vec4 ray2 = reflect(incident, hitnormal, dotNI);
        // rotate away from surface
        vec4 eye2 = hitpos * cacne + sacne * ray2;
        ray2      = ray2   * cacne - sacne * hitpos;
        // re-normalize to avoid acne at high depths
        eye2 = normalize(eye2);
        ray2 = normalize(ray2);
        // trace reflected ray
        intensity += factor * f.x *
            trace(depth - 1, wavelength,    material, eye2, ray2);
      }
      if (f.y > 0.0f) {
        // refracted ray travels through the far side material
        // incident and hitnormal are both tangent to hitpos,
        //     so too will be the refracted ray
        // incident and hitnormal are both unit,
        //     so too will be the refracted ray
```

```
        vec4 ray2 = refract(incident, hitnormal, eta, dotNI);
        // rotate away from surface
        vec4 eye2 = hitpos * cacne + sacne * ray2;
        ray2      = ray2   * cacne - sacne * hitpos;
        // re-normalize to avoid acne at high depths
        eye2 = normalize(eye2);
        ray2 = normalize(ray2);
        // trace refracted ray
        intensity += factor * f.y *
            trace(depth - 1, wavelength, hitmaterial, eye2, ray2);
      }
    }
  }

  return intensity;
}

int main(int argc, char **argv) {
  const char *stem = "prismatic";
  if (argc > 1) { stem = argv[1]; }
  struct timespec now;
  clock_gettime(CLOCK_MONOTONIC, &now);
  time_t then = now.tv_sec;
/*
  fprintf(stderr, "# nm water glass quartz\n");
  for (int wavelength = wavelength_min; wavelength <= wavelength_max; wavelength
      ↪ += wavelength_step) {
    fprintf(stderr, "%d %f %f %f\n", wavelength, water(wavelength, 0), glass_n(
        ↪ wavelength), quartz(wavelength, 0));
  }
*/
/*
  FILE *e = fopen("earth.ppm", "rb");
  if (e) {
    fseek(e, 16, SEEK_SET);
    fread(earth_srgb, 3 * earth_width * earth_height, 1, e);
    fclose(e);
  } else {
    return 1;
  }
*/
  vec4 eye(0.0f, 0.0f, 0.0f, 1.0f);
  mat4 view(initialize_view());
  for (int frame = 0; frame < frames; ++frame) {
    fprintf(stderr, "\r%7d ", frame);
    float time = frame / float(fps);
/*
    earth_phase = time / 6.0f;
*/
    mat4 animation(initialize_animation(time));
    initialize_scene(animation, view);
    material_t material(material_at(eye));
/*
    image_clear_xyza();
*/
    char fname[100];
    snprintf(fname, 100, "%s-%06d.prism", stem, frame);
```

```
    FILE *out = fopen(fname, "wb");
    for (int w = 0; w < wavelength_count; ++w) {
      int wavelength
        = wavelength_lo
        + ( (wavelength_start - wavelength_lo + w * wavelength_step)
          % wavelength_wrap);
/*
      auto stimulus = tristimulus_curve[wavelength - wavelength_min];
      vec3 xyz(stimulus[0], stimulus[1], stimulus[2]);
*/
      initialize_materials(wavelength);
      write_header_intensity(out, image_width, image_height, wavelength);
      int progress = 0;
      #pragma omp parallel for schedule(dynamic, 1)
      for (int image_j = 0; image_j < image_height; ++image_j) {
        #pragma omp critical
        {
          clock_gettime(CLOCK_MONOTONIC, &now);
          if (now.tv_sec > then) {
            then = now.tv_sec;
            float percent
              = ( frame    + 0.0f
              + ( w        + 0.0f
              + ( progress + 0.0f
              ) / image_height
              ) / wavelength_count
              ) / frames;
            fprintf
              ( stderr
              , "\r%6d/%6d : %3d/%3d (%3dnm) : %5d/%5d : %7.3f%%"
              , frame + 1, frames
              , w + 1, wavelength_count, wavelength
              , progress + 1, image_height
              , 100 * percent
              );
          }
        }
        float screen_y = 2.0f * ((image_j + 0.5f) / image_height - 0.5f);
        for (int image_i = 0; image_i < image_width; ++image_i) {
          float screen_x = pi * 2.0f * ((image_i + 0.5f) / image_width - 0.5f);
          vec4 ray(lambert_cylindrical_equal_area_projection(screen_x, screen_y)
              ↘ );
          float intensity(trace(depth_max, wavelength, material, eye, ray));
          if (0.0f <= intensity && intensity < 1.0f / 0.0f) {
            image_intensity[image_j][image_i] = intensity;
/*
            intensity *= 3.0f;
            image_xyza[image_j][image_i] += vec4(intensity * xyz, 1.0);
*/
          } else {
            image_intensity[image_j][image_i] = -1.0f;
          }
        }
        #pragma omp atomic
        progress++;
      }
      write_data_intensity(out, image_width * image_height, &image_intensity
```

```
            ↪ [0][0]);
845       fflush(out);
      }
      fclose(out);
/*
      image_xyza_to_srgb();
850   image_save_ppm();
*/
    }
    fprintf(stderr, "\r");
    return 0;
855 }
```

13 cc/quartz.h

```
#ifndef QUARTZ_H
#define QUARTZ_H 1

const float quartz_min   = 390;
5 const float quartz_step  =   2;
const int   quartz_count = 221;
const float quartz_nk[quartz_count][2] =
  { { 1.490867f, 0.000009f }
  , { 1.490629f, 0.000007f }
10  , { 1.490396f, 0.000005f }
  , { 1.490165f, 0.000003f }
  , { 1.489938f, 0.000002f }
  , { 1.489714f, 0.000001f }
  , { 1.489494f, 0.000001f }
15  , { 1.489276f, 0.000000f }
  , { 1.489062f, 0.000000f }
  , { 1.488850f, 0.000000f }
  , { 1.488641f, 0.000000f }
  , { 1.488435f, 0.000000f }
20  , { 1.488231f, 0.000000f }
  , { 1.488031f, 0.000000f }
  , { 1.487833f, 0.000000f }
  , { 1.487638f, 0.000000f }
  , { 1.487446f, 0.000000f }
25  , { 1.487256f, 0.000000f }
  , { 1.487069f, 0.000000f }
  , { 1.486884f, 0.000000f }
  , { 1.486702f, 0.000000f }
  , { 1.486522f, 0.000000f }
30  , { 1.486345f, 0.000000f }
  , { 1.486170f, 0.000000f }
  , { 1.485997f, 0.000000f }
  , { 1.485827f, 0.000000f }
  , { 1.485659f, 0.000000f }
35  , { 1.485493f, 0.000000f }
  , { 1.485329f, 0.000000f }
  , { 1.485167f, 0.000000f }
  , { 1.485008f, 0.000000f }
  , { 1.484851f, 0.000000f }
40  , { 1.484695f, 0.000000f }
  , { 1.484542f, 0.000000f }
  , { 1.484390f, 0.000000f }
```

```
    , { 1.484241f, 0.000000f }
    , { 1.484094f, 0.000000f }
    , { 1.483948f, 0.000000f }
    , { 1.483804f, 0.000000f }
    , { 1.483662f, 0.000000f }
    , { 1.483522f, 0.000000f }
    , { 1.483383f, 0.000000f }
    , { 1.483247f, 0.000000f }
    , { 1.483111f, 0.000000f }
    , { 1.482978f, 0.000000f }
    , { 1.482846f, 0.000000f }
    , { 1.482716f, 0.000000f }
    , { 1.482588f, 0.000000f }
    , { 1.482461f, 0.000000f }
    , { 1.482335f, 0.000000f }
    , { 1.482211f, 0.000000f }
    , { 1.482089f, 0.000000f }
    , { 1.481968f, 0.000000f }
    , { 1.481848f, 0.000000f }
    , { 1.481730f, 0.000000f }
    , { 1.481613f, 0.000000f }
    , { 1.481498f, 0.000000f }
    , { 1.481384f, 0.000000f }
    , { 1.481271f, 0.000000f }
    , { 1.481160f, 0.000000f }
    , { 1.481050f, 0.000000f }
    , { 1.480941f, 0.000000f }
    , { 1.480834f, 0.000000f }
    , { 1.480727f, 0.000000f }
    , { 1.480622f, 0.000000f }
    , { 1.480518f, 0.000000f }
    , { 1.480416f, 0.000000f }
    , { 1.480314f, 0.000000f }
    , { 1.480214f, 0.000000f }
    , { 1.480115f, 0.000000f }
    , { 1.480016f, 0.000000f }
    , { 1.479919f, 0.000000f }
    , { 1.479823f, 0.000000f }
    , { 1.479729f, 0.000000f }
    , { 1.479635f, 0.000000f }
    , { 1.479542f, 0.000000f }
    , { 1.479450f, 0.000000f }
    , { 1.479359f, 0.000000f }
    , { 1.479269f, 0.000000f }
    , { 1.479181f, 0.000000f }
    , { 1.479093f, 0.000000f }
    , { 1.479006f, 0.000000f }
    , { 1.478920f, 0.000000f }
    , { 1.478835f, 0.000000f }
    , { 1.478751f, 0.000000f }
    , { 1.478667f, 0.000000f }
    , { 1.478585f, 0.000000f }
    , { 1.478503f, 0.000000f }
    , { 1.478423f, 0.000000f }
    , { 1.478343f, 0.000000f }
    , { 1.478264f, 0.000000f }
    , { 1.478186f, 0.000000f }
```

```
100     , { 1.478108f, 0.000000f }
        , { 1.478032f, 0.000000f }
        , { 1.477956f, 0.000000f }
        , { 1.477881f, 0.000000f }
        , { 1.477807f, 0.000000f }
105     , { 1.477733f, 0.000000f }
        , { 1.477660f, 0.000000f }
        , { 1.477588f, 0.000000f }
        , { 1.477517f, 0.000000f }
        , { 1.477447f, 0.000000f }
110     , { 1.477377f, 0.000000f }
        , { 1.477308f, 0.000000f }
        , { 1.477239f, 0.000000f }
        , { 1.477171f, 0.000000f }
        , { 1.477104f, 0.000000f }
115     , { 1.477038f, 0.000000f }
        , { 1.476972f, 0.000000f }
        , { 1.476907f, 0.000000f }
        , { 1.476842f, 0.000000f }
        , { 1.476778f, 0.000000f }
120     , { 1.476715f, 0.000000f }
        , { 1.476652f, 0.000000f }
        , { 1.476590f, 0.000000f }
        , { 1.476529f, 0.000000f }
        , { 1.476468f, 0.000000f }
125     , { 1.476408f, 0.000000f }
        , { 1.476348f, 0.000000f }
        , { 1.476289f, 0.000000f }
        , { 1.476230f, 0.000000f }
        , { 1.476172f, 0.000000f }
130     , { 1.476115f, 0.000000f }
        , { 1.476058f, 0.000000f }
        , { 1.476001f, 0.000000f }
        , { 1.475946f, 0.000000f }
        , { 1.475890f, 0.000000f }
135     , { 1.475835f, 0.000000f }
        , { 1.475781f, 0.000000f }
        , { 1.475727f, 0.000000f }
        , { 1.475674f, 0.000000f }
        , { 1.475621f, 0.000000f }
140     , { 1.475568f, 0.000000f }
        , { 1.475517f, 0.000000f }
        , { 1.475465f, 0.000000f }
        , { 1.475414f, 0.000000f }
        , { 1.475364f, 0.000000f }
145     , { 1.475314f, 0.000000f }
        , { 1.475264f, 0.000000f }
        , { 1.475215f, 0.000000f }
        , { 1.475166f, 0.000000f }
        , { 1.475118f, 0.000000f }
150     , { 1.475070f, 0.000000f }
        , { 1.475022f, 0.000000f }
        , { 1.474975f, 0.000000f }
        , { 1.474929f, 0.000000f }
        , { 1.474883f, 0.000000f }
155     , { 1.474837f, 0.000000f }
        , { 1.474791f, 0.000000f }
```

```
        , { 1.474746f, 0.000000f }
        , { 1.474702f, 0.000000f }
        , { 1.474657f, 0.000000f }
160     , { 1.474614f, 0.000000f }
        , { 1.474570f, 0.000000f }
        , { 1.474527f, 0.000000f }
        , { 1.474484f, 0.000000f }
        , { 1.474442f, 0.000000f }
165     , { 1.474400f, 0.000000f }
        , { 1.474358f, 0.000000f }
        , { 1.474317f, 0.000000f }
        , { 1.474276f, 0.000000f }
        , { 1.474235f, 0.000000f }
170     , { 1.474195f, 0.000000f }
        , { 1.474155f, 0.000000f }
        , { 1.474115f, 0.000000f }
        , { 1.474076f, 0.000000f }
        , { 1.474037f, 0.000000f }
175     , { 1.473998f, 0.000000f }
        , { 1.473960f, 0.000000f }
        , { 1.473922f, 0.000000f }
        , { 1.473884f, 0.000000f }
        , { 1.473847f, 0.000000f }
180     , { 1.473810f, 0.000000f }
        , { 1.473773f, 0.000000f }
        , { 1.473737f, 0.000000f }
        , { 1.473700f, 0.000000f }
        , { 1.473665f, 0.000000f }
185     , { 1.473629f, 0.000000f }
        , { 1.473594f, 0.000000f }
        , { 1.473559f, 0.000000f }
        , { 1.473524f, 0.000000f }
        , { 1.473489f, 0.000000f }
190     , { 1.473455f, 0.000000f }
        , { 1.473421f, 0.000000f }
        , { 1.473387f, 0.000000f }
        , { 1.473354f, 0.000000f }
        , { 1.473321f, 0.000000f }
195     , { 1.473288f, 0.000000f }
        , { 1.473255f, 0.000000f }
        , { 1.473223f, 0.000000f }
        , { 1.473191f, 0.000000f }
        , { 1.473159f, 0.000000f }
200     , { 1.473127f, 0.000000f }
        , { 1.473096f, 0.000000f }
        , { 1.473065f, 0.000000f }
        , { 1.473034f, 0.000000f }
        , { 1.473003f, 0.000000f }
205     , { 1.472973f, 0.000000f }
        , { 1.472942f, 0.000000f }
        , { 1.472912f, 0.000000f }
        , { 1.472883f, 0.000000f }
        , { 1.472853f, 0.000000f }
210     , { 1.472824f, 0.000000f }
        , { 1.472795f, 0.000000f }
        , { 1.472766f, 0.000000f }
        , { 1.472737f, 0.000000f }
```

```
    , { 1.472709f, 0.000000f }
    , { 1.472680f, 0.000000f }
    , { 1.472652f, 0.000000f }
    , { 1.472625f, 0.000000f }
    , { 1.472597f, 0.000000f }
    , { 1.472570f, 0.000000f }
    , { 1.472542f, 0.000000f }
    , { 1.472515f, 0.000000f }
    , { 1.472489f, 0.000000f }
    , { 1.472462f, 0.000000f }
    , { 1.472436f, 0.000000f }
    , { 1.472409f, 0.000000f }
    , { 1.472383f, 0.000000f }
    , { 1.472357f, 0.000000f }
    , { 1.472332f, 0.000000f }
    };

inline float quartz(float l, int w) {
  float x  = (l - quartz_min) / quartz_step;
  int   i0 = floor(x);
  int   i1 = i0 + 1;
  float f1 = x - i0;
  float f0 = 1 - f1;
  if (i0 < 0) { return quartz_nk[0][w]; }
  if (i1 > quartz_count - 1) { return quartz_nk[quartz_count - 1][w]; }
  float y0 = quartz_nk[i0][w];
  float y1 = quartz_nk[i1][w];
  return f0 * y0 + f1 * y1;
}

#endif
```

14 cc/quartz.sh

```
#!/bin/sh
cat "${1}/database/main/SiO2/Gao.yml" |
tail -n+79 |
head -n 221 |
sed 's/\r//g' |
(
  cat <<EOF
#ifndef QUARTZ_H
#define QUARTZ_H 1

const float quartz_min   = 390;
const float quartz_step  =   2;
const int   quartz_count = 221;
const float quartz_nk[quartz_count][2] =
EOF
  sep="{"
  while read l n k
  do
    echo "  ${sep} { ${n}f, ${k}f }"
    sep=","
  done
  cat <<EOF
  };
```

```
inline float quartz(float l, int w) {
  float x  = (l - quartz_min) / quartz_step;
  int   i0 = floor(x);
  int   i1 = i0 + 1;
  float f1 = x - i0;
  float f0 = 1 - f1;
  if (i0 < 0) { return quartz_nk[0][w]; }
  if (i1 > quartz_count - 1) { return quartz_nk[quartz_count - 1][w]; }
  float y0 = quartz_nk[i0][w];
  float y1 = quartz_nk[i1][w];
  return f0 * y0 + f1 * y1;
}

#endif
EOF
)
```

15 cc/sapphire.h

```
#ifndef SAPPHIRE_H
#define SAPPHIRE_H 1

inline float sapphire(float wavelength) {
  float l2 = 1.0e-6f * wavelength * wavelength;
  return sqrt(1.0f + l2 *
    ( 1.4313493f / (l2 - 0.0726631f*0.0726631f)
    + 0.65054713f / (l2 - 0.1193242f*0.1193242f)
    + 5.3414021f / (l2 - 18.028251f*18.028251f)
    ));
}

#endif
```

16 cc/sapphire.sh

```
#!/bin/sh
cat "${1}/database/main/Al2O3/Malitson-o.yml" |
sed 's/\r//g' |
grep coefficients |
(
  read spam c1 c2 c3 c4 c5 c6 c7
  cat << EOF
#ifndef SAPPHIRE_H
#define SAPPHIRE_H 1

inline float sapphire(float wavelength) {
  float l2 = 1.0e-6f * wavelength * wavelength;
  return sqrt(1.0f + l2 *
    ( ${c2}f / (l2 - ${c3}f*${c3}f)
    + ${c4}f / (l2 - ${c5}f*${c5}f)
    + ${c6}f / (l2 - ${c7}f*${c7}f)
    ));
}

#endif
```

```
EOF
)
```

17 cc/srgb2png.cc

```
#include <cmath>
#include <cstdint>
#include <cstdio>
#include <cstdlib>
using namespace std;

#include "format.cc"

#include <png.h>

void write_png_err(png_structp png, png_const_charp msg) {
  fprintf(stderr, "error: png: %s\n", msg);
  jmp_buf *jmp = (jmp_buf *) png_get_error_ptr(png);
  if (jmp) { longjmp(*jmp, 1); } else { abort(); }
}

bool write_png(FILE *out, int width, int height, png_bytepp rows) {
  jmp_buf jmp;
  png_structp png = png_create_write_struct(PNG_LIBPNG_VER_STRING, &jmp, ↙
      ↳ write_png_err, 0);
  if (! png) { return false; }
  png_infop info = png_create_info_struct(png);
  if (! info) { png_destroy_write_struct(&png, 0); return false; }
  if (setjmp(jmp)) { png_destroy_write_struct(&png, &info); return false; }
  png_init_io(png, out);
  png_set_compression_level(png, Z_BEST_COMPRESSION);
  png_set_IHDR
    ( png, info
    , width, height, 16
    , PNG_COLOR_TYPE_RGB
    , PNG_INTERLACE_ADAM7
    , PNG_COMPRESSION_TYPE_DEFAULT
    , PNG_FILTER_TYPE_DEFAULT
    );
  png_set_sRGB(png, info, PNG_sRGB_INTENT_ABSOLUTE);
  png_write_info(png, info);
  png_write_image(png, rows);
  png_write_end(png, info);
  png_destroy_write_struct(&png, &info);
  return true;
}

int main(int argc, char **argv) {
  const char *stem = "prismatic";
  if (argc > 1) { stem = argv[1]; }
  float *imagef = 0;
  uint16_t *image16 = 0;
  uint16_t **rows = 0;
  int allocated_width = -1;
  int allocated_height = -1;
  int width = 0;
  int height = 0;
```

```
  int frame = 0;
  while (read_header_srgb(stdin, &width, &height)) {
    // allocate if necessary
    if (allocated_width != width && allocated_height != height) {
      if (imagef) { free(imagef); imagef = 0; }
      if (image16) { free(image16); image16 = 0; }
      if (rows) { free(rows); rows = 0; }
      if (width > 0 && height > 0) {
        int bytes = width * height * 3 * sizeof(float);
        imagef = (float *) calloc(bytes, 1);
        if (! imagef) {
          fprintf(stderr, "error: memory allocation failed: %d bytes\n", bytes);
          return 1;
        }
        bytes = width * height * 3 * sizeof(uint16_t);
        image16 = (uint16_t *) calloc(bytes, 1);
        if (! image16) {
          fprintf(stderr, "error: memory allocation failed: %d bytes\n", bytes);
          return 1;
        }
        bytes = height * sizeof(uint16_t *);
        rows = (uint16_t **) calloc(bytes, 1);
        if (! rows) {
          fprintf(stderr, "error: memory allocation failed: %d bytes\n", bytes);
          return 1;
        }
        for (int j = 0; j < height; ++j) {
          rows[j] = image16 + j * width * 3;
        }
      } else {
        fprintf(stderr, "error: bad image size: %d %d\n", width, height);
        return 1;
      }
    } // allocate if necessary
    if (! read_data_srgb(stdin, width * height, imagef)) { return 1; }
    // quantize image
    int count = width * height * 3;
    #pragma omp parallel for schedule(static)
    for (int k = 0; k < count; ++k) {
      // FIXME TODO dither
      uint64_t v = fminf(fmaxf(65535.0f * imagef[k], 0), 65535);
      image16[k] = ((v & 0xff) << 8) | ((v >> 8) & 0xff);
    }
    // write image
    char fname[100];
    snprintf(fname, 100, "%s-%06d.png", stem, frame);
    FILE *out = fopen(fname, "wb");
    if (! out) { return 1; }
    write_png(out, width, height, (png_bytepp) rows);
    fclose(out);
    // advance frame counter
    frame++;
  }
  if (frame == 0) {
    fprintf(stderr, "warning: no images\n");
  }
  return 0;
```

```
}
```

18 cc/water.h

```
#ifndef WATER_H
#define WATER_H 1

const float water_min   = 375;
const float water_step  =  25;
const int   water_count =  20;
const float water_nk[water_count][2] =
  { { 1.341f, 3.50E-9f }
  , { 1.339f, 1.86E-9f }
  , { 1.338f, 1.30E-9f }
  , { 1.337f, 1.02E-9f }
  , { 1.336f, 9.35E-10f }
  , { 1.335f, 1.00E-9f }
  , { 1.334f, 1.32E-9f }
  , { 1.333f, 1.96E-9f }
  , { 1.333f, 3.60E-9f }
  , { 1.332f, 1.09E-8f }
  , { 1.332f, 1.39E-8f }
  , { 1.331f, 1.64E-8f }
  , { 1.331f, 2.23E-8f }
  , { 1.331f, 3.35E-8f }
  , { 1.330f, 9.15E-8f }
  , { 1.330f, 1.56E-7f }
  , { 1.330f, 1.48E-7f }
  , { 1.329f, 1.25E-7f }
  , { 1.329f, 1.82E-7f }
  , { 1.329f, 2.93E-7f }
  };

inline float water(float l, int w) {
  float x  = (l - water_min) / water_step;
  int   i0 = floor(x);
  int   i1 = i0 + 1;
  float f1 = x - i0;
  float f0 = 1 - f1;
  if (i0 < 0) { return water_nk[0][w]; }
  if (i1 > water_count - 1) { return water_nk[water_count - 1][w]; }
  float y0 = water_nk[i0][w];
  float y1 = water_nk[i1][w];
  return f0 * y0 + f1 * y1;
}

#endif
```

19 cc/water.sh

```
#!/bin/sh
cat "${1}/database/main/H2O/Hale.yml" |
tail -n+17 |
head -n 20 |
sed 's/\r//g' |
(
```

```
  cat <<EOF
#ifndef WATER_H
#define WATER_H 1

const float water_min   = 375;
const float water_step  =  25;
const int   water_count =  20;
const float water_nk[water_count][2] =
EOF
  sep="{"
  while read l n k
  do
    echo "  ${sep} { ${n}f, ${k}f }"
    sep=","
  done
  cat <<EOF
  };

inline float water(float l, int w) {
  float x  = (l - water_min) / water_step;
  int   i0 = floor(x);
  int   i1 = i0 + 1;
  float f1 = x - i0;
  float f0 = 1 - f1;
  if (i0 < 0) { return water_nk[0][w]; }
  if (i1 > water_count - 1) { return water_nk[water_count - 1][w]; }
  float y0 = water_nk[i0][w];
  float y1 = water_nk[i1][w];
  return f0 * y0 + f1 * y1;
}

#endif
EOF
)
```

20 cc/xyz2png.cc

```
#include <cmath>
#include <cstdint>
#include <cstdio>
#include <cstdlib>
using namespace std;

#include "format.cc"

#include <png.h>
#include <arpa/inet.h>

#define d50_xyz_icc_len 528
extern const uint8_t d50_xyz_icc[d50_xyz_icc_len];

void write_png_err(png_structp png, png_const_charp msg) {
  fprintf(stderr, "error: png: %s\n", msg);
  jmp_buf *jmp = (jmp_buf *) png_get_error_ptr(png);
  if (jmp) { longjmp(*jmp, 1); } else { abort(); }
}

```

```
bool write_png(FILE *out, int width, int height, png_bytepp rows) {
  jmp_buf jmp;
  png_structp png = png_create_write_struct(PNG_LIBPNG_VER_STRING, &jmp, ↙
      ↳ write_png_err, 0);
  if (! png) { return false; }
  png_infop info = png_create_info_struct(png);
  if (! info) { png_destroy_write_struct(&png, 0); return false; }
  if (setjmp(jmp)) { png_destroy_write_struct(&png, &info); return false; }
  png_init_io(png, out);
  png_set_compression_level(png, Z_BEST_COMPRESSION);
  png_set_IHDR
    ( png, info
    , width, height, 16
    , PNG_COLOR_TYPE_RGB
    , PNG_INTERLACE_ADAM7
    , PNG_COMPRESSION_TYPE_DEFAULT
    , PNG_FILTER_TYPE_DEFAULT
    );
  png_set_iCCP(png, info, (png_charp) "D50 XYZ", 0, (png_charp) d50_xyz_icc, ↙
      ↳ d50_xyz_icc_len);
  png_write_info(png, info);
  png_write_image(png, rows);
  png_write_end(png, info);
  png_destroy_write_struct(&png, &info);
  return true;
}

int main(int argc, char **argv) {
  const char *stem = "prismatic";
  if (argc > 1) { stem = argv[1]; }
  float *f32 = 0;
  uint16_t *u16 = 0;
  uint16_t **row = 0;
  int allocated_width = -1;
  int allocated_height = -1;
  int width = 0;
  int height = 0;
  int frame = 0;
  // read image
  while (read_header_xyz(stdin, &width, &height)) {
    // allocate if necessary
    if (allocated_width != width && allocated_height != height) {
      if (f32) { free(f32); f32 = 0; }
      if (u16) { free(u16); u16 = 0; }
      if (row) { free(row); row = 0; }
      if (width > 0 && height > 0) {
        int bytes = width * height * 3 * sizeof(float);
        f32 = (float *) calloc(bytes, 1);
        if (! f32) {
          fprintf(stderr, "error: memory allocation failed: %d bytes\n", bytes);
          return 1;
        }
        bytes = width * height * 3 * sizeof(uint16_t);
        u16 = (uint16_t *) calloc(bytes, 1);
        if (! u16) {
          fprintf(stderr, "error: memory allocation failed: %d bytes\n", bytes);
          return 1;
```

```
        }
        bytes = height * sizeof(uint16_t *);
        row = (uint16_t **) calloc(bytes, 1);
        if (! row) {
          fprintf(stderr, "error: memory allocation failed: %d bytes\n", bytes);
          return 1;
        }
        for (int j = 0; j < height; ++j) {
          row[j] = u16 + j * width * 3;
        }
      } else {
        fprintf(stderr, "error: bad image size: %d %d\n", width, height);
        return 1;
      }
    } // allocate if necessary
    int count = width * height;
    if (! read_data_xyz(stdin, count, f32)) { break; }
    // transform colour space
    const float d50[3] = { 0.9642f, 1.0000f, 0.8249f };
    #pragma omp parallel for schedule(static)
    for (int k = 0; k < count; ++k) {
      f32[3*k+0] *= d50[0];
      f32[3*k+1] *= d50[1];
      f32[3*k+2] *= d50[2];
    }
    int count3 = count * 3;
    float ma = -1.0f / 0.0f;
    #pragma omp parallel for schedule(static) reduction(max: ma)
    for (int k = 0; k < count3; ++k) {
      ma = fmaxf(ma, f32[k]);
    }
    fprintf(stderr, "maximum at D50: %e\n", ma);
    // quantize to 16bit (FIXME TODO dither)
    float s = 65535.0f;// / ma;
    #pragma omp parallel for schedule(static)
    for (int k = 0; k < count3; ++k) {
      uint16_t u = fminf(fmaxf(s * f32[k], 0.0f), 65535.0f);
      u16[k] = htons(u);
    }
    // write image
    char fname[100];
    snprintf(fname, 100, "%s-%06d.png", stem, frame);
    FILE *out = fopen(fname, "wb");
    if (! out) { return 1; }
    write_png(out, width, height, (png_bytepp) row);
    fclose(out);
    frame++;
  }
  if (frame == 0) {
    fprintf(stderr, "warning: no images\n");
  }
  return 0;
}
```

21 cc/xyz2srgb.cc

```
#include <cmath>
```

```
#include <cstdio>
#include <cstdlib>
using namespace std;

#include "format.cc"

inline bool xyz2srgb1
  ( const float &x, const float &y, const float &z
  ,       float &r,       float &g,       float &b
  ) {
  // white point
  float xw = x * 0.9505f;
  float yw = y;//1.0000f;
  float zw = z * 1.0890f;
  // linear rgb
  float rl =  3.2406f * xw + -1.5372f * yw + -0.4986f * zw;
  float gl = -0.9689f * xw +  1.8758f * yw +  0.0415f * zw;
  float bl =  0.0557f * xw + -0.2040f * yw +  1.0570f * zw;
  // check gammut
  bool  ok
    =  0.0f <= rl && rl <= 1.0f
    && 0.0f <= gl && gl <= 1.0f
    && 0.0f <= bl && bl <= 1.0f;
  r = rl;
  g = gl;
  b = bl;
  return ok;
}

inline void xyz2srgb2
  ( const float &x, const float &y, const float &z
  ,       float &r,       float &g,       float &b
  , const float &o, const float &s
  ) {
  // scale
  float rl = (x + o) * s;
  float gl = (y + o) * s;
  float bl = (z + o) * s;
  // clamp
  float rc = rl < 0.0f ? 0.0f : rl > 1.0f ? 1.0f : rl;
  float gc = gl < 0.0f ? 0.0f : gl > 1.0f ? 1.0f : gl;
  float bc = bl < 0.0f ? 0.0f : bl > 1.0f ? 1.0f : bl;
  // gamma
  float rg
    = rc <= 0.0031308f
    ? 12.92f * rc
    : 1.055f * pow(rc, 0.41666666f) - 0.055f;
  float gg
    = gc <= 0.0031308f
    ? 12.92f * gc
    : 1.055f * pow(gc, 0.41666666f) - 0.055f;
  float bg
    = bc <= 0.0031308f
    ? 12.92f * bc
    : 1.055f * pow(bc, 0.41666666f) - 0.055f;
  // output
  r = rg;
```

```
  g = gg;
  b = bg;
}

int cmp_float(const void *a, const void *b) {
  const float *p = (const float *) a;
  const float *q = (const float *) b;
  float x = *p;
  float y = *q;
  if (x < y) return -1;
  if (x > y) return  1;
  return 0;
}

int main() {
  const float lopercentile = 0.004;
  const float hipercentile = 0.999;
  float *image = 0;
  float *image2 = 0;
  int allocated_width = -1;
  int allocated_height = -1;
  int width = 0;
  int height = 0;
  int frame = 0;
  // read image
  while (read_header_xyz(stdin, &width, &height)) {
    // allocate if necessary
    if (allocated_width != width && allocated_height != height) {
      if (image) { free(image); image = 0; }
      if (image2) { free(image2); image2 = 0; }
      if (width > 0 && height > 0) {
        int bytes = width * height * 3 * sizeof(float);
        image = (float *) calloc(bytes, 1);
        if (! image) {
          fprintf(stderr, "error: memory allocation failed: %d bytes\n", bytes);
          return 1;
        }
        image2 = (float *) calloc(bytes, 1);
        if (! image2) {
          fprintf(stderr, "error: memory allocation failed: %d bytes\n", bytes);
          return 1;
        }
      } else {
        fprintf(stderr, "error: bad image size: %d %d\n", width, height);
        return 1;
      }
    } // allocate if necessary
    int count = width * height;
    if (! read_data_xyz(stdin, count, image)) { break; }
    // transform colour space
    int out_of_gamut = 0;
    #pragma omp parallel for schedule(static)
    for (int k = 0; k < count; ++k) {
      float x = image[3*k+0];
      float y = image[3*k+1];
      float z = image[3*k+2];
      float r, g, b;
```

```
        bool ok = xyz2srgb1(x, y, z, r, g, b);
        image2[3*k+0] = image[3*k+0] = r;
        image2[3*k+1] = image[3*k+1] = g;
        image2[3*k+2] = image[3*k+2] = b;
        if (! ok) {
          #pragma omp atomic
          out_of_gamut++;
        }
      }
      // diagnostics
      if (out_of_gamut) {
        fprintf
          ( stderr
          , "warning: %f%% outside sRGB gamut\n"
          , out_of_gamut * 100.0 / count
          );
      }
      qsort(image2, count * 3, sizeof(float), cmp_float);
      float lo = image2[int(ceil(count * 3 * lopercentile))    ];
      float hi = image2[int(ceil(count * 3 * hipercentile)) - 1];
      lo = fminf(lo, 0.0f);
      fprintf
        ( stderr
        , "info: rescaling linear RGB from (%f, %f) to (0, 1);\n"
        , lo, hi
        );
      float o = -lo;
      float s = 1.0f / (hi - lo);
      #pragma omp parallel for schedule(static)
      for (int k = 0; k < count; ++k) {
        float x = image[3*k+0];
        float y = image[3*k+1];
        float z = image[3*k+2];
        float r, g, b;
        xyz2srgb2(x, y, z, r, g, b, o, s);
        image[3*k+0] = r;
        image[3*k+1] = g;
        image[3*k+2] = b;
      }
      // write image
      write_header_srgb(stdout, width, height);
      write_data_srgb(stdout, count, image);
      fflush(stdout);
      frame++;
    }
    if (frame == 0) {
      fprintf(stderr, "warning: no images\n");
    }
    return 0;
}
```

22 .gitignore

```
cc/prismatic
cc/*/
cc/chromaticity-diagram
cc/prism2xyz
```

```
cc/srgb2png
cc/xyz2png
cc/xyz2srgb
cc/*.prism
cc/*.srgb
cc/*.png
cc/*.xyz
cc/*.o
```

23 glsl/prismatic-2d.frag

```
#donotrun

#buffer RGBA32F
#buffershader "prismatic-buffer.frag"

#vertex

#group Camera
uniform vec2 Center; slider[(-10,-10),(0,0),(10,10)] NotLockable
uniform float Zoom; slider[0,1,2] NotLockable

uniform vec2 pixelSize;

varying vec2 coord;
varying vec2 viewCoord;

void main() {
  float ar = pixelSize.y/pixelSize.x;
  gl_Position = gl_Vertex;
  viewCoord = gl_Vertex.xy;
  coord = ((gl_ProjectionMatrix*gl_Vertex).xy * vec2(ar, 1.0)) / Zoom + Center;
}

#endvertex

varying vec2 coord;
varying vec2 viewCoord;

// forward declarations
void init();
vec3 color(vec2 z);

uniform sampler2D backbuffer;

void main() {
  init();
  vec4 prev = texture2D(backbuffer, (viewCoord + vec2(1.0)) / 2.0);
  vec4 new = vec4(color(coord.xy), 1.0);
  if (new != new) { new = vec4(0.0); } // NaN check
  gl_FragColor = prev + vec4(new.rgb * new.a, new.a);
}
```

24 glsl/prismatic-buffer.frag

```
#donotrun
```

```
#vertex

varying vec2 coord;

void main() {
  gl_Position = gl_Vertex;
  coord = (gl_ProjectionMatrix * gl_Vertex).xy;
}

#endvertex

varying vec2 coord;

uniform sampler2D frontbuffer;

uniform float Gain;

float srgb(float c0) {
  float c = clamp(c0, 0.0, 1.0);
  if (c <= 0.0031308) {
    return 12.92 * c;
  } else {
    return 1.055 * pow(c, 1.0 / 2.4) - 0.055;
  }
}

vec3 srgb(vec3 xyz) {
  vec3 rgb = xyz * mat3
    ( 3.2406, -1.5372, -0.4986
    , -0.9689, 1.8758, 0.0415
    , 0.0557, -0.2040, 1.0570
    );
  return vec3(srgb(rgb.r), srgb(rgb.g), srgb(rgb.b));
}

void main() {
  vec2 pos = (coord+vec2(1.0))/2.0;
  vec4 tex = texture2D(frontbuffer, pos);
  vec3 xyz = exp2(Gain) * tex.xyz / tex.a;
  gl_FragColor = vec4(srgb(xyz), 1.0);
}
```

25 glsl/prismatic.frag

```
#version 330 compatibility

#include "prismatic-2d.frag"

uniform int subframe;
uniform float time;

const float pi = 3.141592653589793;


#group Animation
```

```
uniform float Duration; slider[1,15,300]
uniform int Spin; slider[-15,-1,15]
uniform int Push; slider[-15,2,15]

#group Rotation

uniform float XY; slider[-180,0,180]
uniform float XZ; slider[-180,30,180]
uniform float XW; slider[-180,45,180]
uniform float YZ; slider[-180,-15,180]
uniform float YW; slider[-180,60,180]
uniform float ZW; slider[-180,30,180]

#group RayTrace

// XYZ tristimulus response curves
uniform sampler2D Spectrum; file[xyz_390nm-830nm.png]
const float mincolour = 0.39; // violet
const float maxcolour = 0.83; // red

uniform float Gain; slider[-10,0,10]

uniform int Depth; slider[1,3,6]
uniform int Count; slider[1,10,500]

uniform float Sphere; slider[0,0.66666,1]
uniform float Light; slider[0,0.1,1]

#group Material

uniform vec3 TransmissionWavelength; slider[(0.39,0.39,0.39),(0.45,0.58,0.65)
    ↳ ,(0.83,0.83,0.83)]
uniform vec3 TransmissionQuality; slider[(0.0,0.0,0.0),(8.0,8.0,8.0)
    ↳ ,(64.0,64.0,64.0)]
uniform vec3 TransmissionAttenuation; slider[(0.0,0.0,0.0),(3.0,3.0,3.0)
    ↳ ,(16.0,16.0,16.0)]
uniform vec3 DiffuseWavelength; slider[(0.39,0.39,0.39),(0.45,0.58,0.65)
    ↳ ,(0.83,0.83,0.83)]
uniform vec3 DiffuseQuality; slider[(0.0,0.0,0.0),(4.0,4.0,4.0),(64.0,64.0,64.0)
    ↳ ]
uniform vec3 DiffuseIntensity; slider[(0.0,0.0,0.0),(1.0,1.0,1.0)
    ↳ ,(10.0,10.0,10.0)]

// wavelength of current subframe
float colourTheta = 0.75 * (float(subframe % Count) + 0.5) / float(Count);
float colour = mix(mincolour, maxcolour, colourTheta);

// corresponding XYZ
vec3 basecolour = texture(Spectrum, vec2(colourTheta, 0.5)).xyz;

float refractionIndex = 2.5 - colourTheta;

// trace rays this deep
const int ndepth = 6;
const int ncontext = 1 << (1 + ndepth);
```

```
// advance rays away from surface intersection
const float acne = 0.001;


// spheres at the vertices of a 24-cell with lights at their centers
const int nspheres = 24;
const int nlights = nspheres;
struct sphere {
  vec4  center;
  float sradius; // sin(radius)
  float index;   // refractive index
  int   colour;  // wavelength in micrometers, or 0 for white
};
sphere scene[nspheres + nlights];


// no recursion in shaders, so emulate it with a queue
struct context {
  vec4  eye;       // ray start point
  vec4  ray;       // ray direction vector
  float factor;    // contribution of this ray to the total
  int   material; // what the ray is travelling through
  int   depth;     // pseudo-recursion depth
};
context queue[ncontext];
int qread  = 0;
int qwrite = 0;

// check if the queue is empty
bool empty() { return !(qread < qwrite); }

// append to the queue
void enqueue(vec4 eye, vec4 ray, float factor, int material, int depth) {
  if (depth < Depth && depth < ndepth && qwrite < ncontext) {
    queue[qwrite].eye = eye;
    queue[qwrite].ray = ray;
    queue[qwrite].factor = factor;
    queue[qwrite].material= material;
    queue[qwrite].depth = depth;
    qwrite = qwrite + 1;
  }
}


float attenuation(int material) {
  float c = 0.0;
  switch (material) { // work around NVIDIA driver bug where dynamic indexing ↙
      ↳ takes forever to link
    // zero at material colour (perfect transmission), higher elsewhere
    case 0:
      c = (TransmissionWavelength[0] - colour) / (maxcolour - mincolour);
      return TransmissionAttenuation[0] * (1.0 - 1.0 / (1.0 + ↙
          ↳ TransmissionQuality[0] * TransmissionQuality[0] * c * c));
    case 1:
      c = (TransmissionWavelength[1] - colour) / (maxcolour - mincolour);
      return TransmissionAttenuation[1] * (1.0 - 1.0 / (1.0 + ↙
          ↳ TransmissionQuality[1] * TransmissionQuality[1] * c * c));
```

```
      case 2:
        c = (TransmissionWavelength[2] - colour) / (maxcolour - mincolour);
        return TransmissionAttenuation[2] * (1.0 - 1.0 / (1.0 + ↙
            ↳ TransmissionQuality[2] * TransmissionQuality[2] * c * c));
      // empty space doesn't attenuate
      default:
        return 0.0;
    }
}


float emission(int material) {
  float c = 0.0;
  switch (material) { // work around NVIDIA driver bug where dynamic indexing ↙
      ↳ takes forever to link
    // peak at material colour, lower elsewhere
    case 0:
      c = (DiffuseWavelength[0] - colour) / (maxcolour - mincolour);
      return DiffuseIntensity[0] / (1.0 + DiffuseQuality[0] * DiffuseQuality[0] ↙
          ↳ * c * c);
    case 1:
      c = (DiffuseWavelength[1] - colour) / (maxcolour - mincolour);
      return DiffuseIntensity[1] / (1.0 + DiffuseQuality[1] * DiffuseQuality[1] ↙
          ↳ * c * c);
    case 2:
      c = (DiffuseWavelength[2] - colour) / (maxcolour - mincolour);
      return DiffuseIntensity[2] / (1.0 + DiffuseQuality[2] * DiffuseQuality[2] ↙
          ↳ * c * c);
    default:
      // lights are bright
      return 16.0;
  }
}


// spherical tangent space
// points and directions are unit length
// directions from the pole are equivalent to points on the equator
// result is direction towards "to" orthogonal to "from"
vec4 tangent(vec4 to, vec4 from) {
  return normalize(to - dot(to, from) * from);
}

// Beer-Lambert law
float transmit(float attenuation, float pathlength) {
  return exp(-attenuation * pathlength);
}

// Fresnel equations
// I = incident ray
// N = surface normal
// eta = n1 / n2
// result.x = reflected power
// result.y = transmitted power
vec2 fresnel(vec4 I, vec4 N, float eta) {
  float c1 = -dot(I, N);
  float c2 = 1.0 - (1.0 - c1 * c1) * (eta * eta);
```

```
    if (c2 < 0.0) {
      // total internal reflection
      return vec2(1.0, 0.0);
    }
    c2 = sqrt(c2);
    float Rs = (eta * c1 - c2) / (eta * c1 + c2);
    float Rp = (eta * c2 - c1) / (eta * c2 + c1);
    float R = 0.5 * (Rs * Rs + Rp * Rp);
    float T = 1.0 - R;
    return vec2(R, T);
}


// trace rays
float trace() {
    // accumulated from all rays
    float intensity = 0.0;
    // while there are rays in the queue
    for (int count = 0; count < ncontext; ++count) {
      if (empty()) { break; }

      // dequeue a ray
      vec4  eye      = queue[qread].eye;
      vec4  ray      = queue[qread].ray;
      float factor   = queue[qread].factor;
      int   material = queue[qread].material;
      int   depth    = queue[qread].depth;
      qread = qread + 1;
      // initialize results
      bool  hit         = false;      // did the ray hit anything
      bool  hitlight    = false;      // if so, did it hit a light
      float hitdistance = 1.0 / 0.0;  // keep only the nearest hit
      vec4  hitpos      = vec4(0.0);  // ray-surface intersection point
      vec4  hitnormal   = vec4(0.0);  // surface normal at intersection
      int   hitmaterial = -1;        // material beyond the surface
      float eta         = 1.0;        // ratio of refraction indices
      float emit        = 0.0;        // emission from surface

      // check every object in the scene
      for (int k = 0; k < nspheres + nlights; ++k) {
        vec4  center  = scene[k].center;
        // r = sphere radius
        float sr      = scene[k].sradius;
        float cr      = sqrt(1.0 - sr * sr);
        // d = distance from eye to center
        float cd      = dot(eye, center);
        float sd      = sqrt(1.0 - cd * cd);
        // theta = angle subtended by the sphere radius
        float stheta  = sr / sd;
        // phi = angle between ray and sphere center
        vec4  tcenter = tangent(center, eye);
        float cphi    = dot(ray, tcenter);

        if (stheta > 1.0) {
          // d < r
          // sphere surrounds eye, ray intersects from the inside
          // ignore lights
```

```
      if (k < nspheres) {
        // find the distance to the intersection
        // ... c + sqrt ... because the other intersection is behind
        float a = cr;
        float b = cd;
        float c = sd * cphi;
        float h = 2.0 * atan(c + sqrt(c * c + b * b - a * a), a + b);
        // check if it's closer
        if (0.0 < h && h < hitdistance) {
          // psi = angle between ray and surface normal
          float sphi = sqrt(1.0 - cphi * cphi);
          float spsi = sphi / stheta;
          float cpsi = sqrt(1.0 - spsi * spsi);
          // update results with new details
          hit         = true;
          hitlight    = false;
          hitdistance = h;
          // eye and ray are orthogonal unit vectors
          hitpos      = cos(h) * eye + sin(h) * ray;
          // surface normal points towards sphere center
          hitnormal   = tangent(center, hitpos);
          // outside the sphere is vacuum
          hitmaterial = -1;
          eta         = scene[k].index / 1.0;
          // Lambertian diffuse reflection
          emit        = emission(scene[k].colour) * clamp(cpsi, 0.0, 1.0);
        }
      }

    } else {
      // d > r
      // sphere is disjoint from eye, ray might intersect from outside
      float ctheta = sqrt(1.0 - stheta * stheta);
      if (cphi > ctheta) {
        // phi < theta
        // ray does intersect
        // find the distance to the intersection
        // ... c - sqrt ... because the other intersection is further
        float a = cr;
        float b = cd;
        float c = sd * cphi;
        float h = 2.0 * atan(c - sqrt(c * c + b * b - a * a), a + b);
        // check if it's closer
        if (0.0 < h && h < hitdistance) {
          // psi = angle between ray and surface normal
          float sphi = sqrt(1.0 - cphi * cphi);
          float spsi = sphi / stheta;
          float cpsi = sqrt(1.0 - spsi * spsi);
          // update results with new details
          hit         = true;
          hitlight    = nspheres <= k;
          hitdistance = h;
          // eye and ray are orthogonal unit vectors
          hitpos      = cos(h) * eye + sin(h) * ray;
          // surface normal points away from sphere center
          hitnormal   = -tangent(center, hitpos);
          // inside the sphere is the sphere's material
```

```
            hitmaterial = scene[k].colour;
285         // outside the sphere is vacuum
            eta         = 1.0 / scene[k].index;
            // Lambertian diffuse reflection
            emit        = emission(scene[k].colour) * clamp(cpsi, 0.0, 1.0);
          }
290     }
      }
    } // for each object in scene

    if (hit) {
295   // attenuation by the near-side material through which the ray travelled
      float transmission = transmit(attenuation(material), hitdistance);
      // non-physical fading with distance
      float fade = 0.5 * (1.0 + cos(hitdistance));
      // contribution from this ray
300   float factor2 = factor * transmission * fade;
      // accumulate this ray
      intensity += factor2 * emit;
      if (! hitlight) {
        // incident ray
305     vec4 incident = tangent(ray, hitpos);
        // parallel transport past the equator
        if (hitdistance > pi / 2.0) { incident = -incident; }
        // Fresnel equations for intensity of reflection and refraction
        vec2 f = fresnel(incident, hitnormal, eta);
310     if (f.x > 0.0) {
          // reflected ray travels through the near side material
          vec4 ray2 = tangent(normalize(reflect(incident, hitnormal)), hitpos);
          vec4 eye2 = normalize(hitpos + acne * ray2);
          ray2 = tangent(ray2, eye2);
315       enqueue(eye2, ray2, factor2 * f.x, material, depth + 1);
        }
        if (f.y > 0.0) {
          // refracted ray travels through the far side material
          vec4 ray2 = tangent(normalize(refract(incident, hitnormal, eta)),
              ↪ hitpos);
320       vec4 eye2 = normalize(hitpos + acne * ray2);
          ray2 = tangent(ray2, eye2);
          enqueue(eye2, ray2, factor2 * f.y, hitmaterial, depth + 1);
        }
      }
325 }

  } // while there are rays in the queue
  return intensity;
}
330

// base rotation transform
float cXY = cos(radians(XY));
float sXY = sin(radians(XY));
335 float cXZ = cos(radians(XZ));
float sXZ = sin(radians(XZ));
float cXW = cos(radians(XW));
float sXW = sin(radians(XW));
float cYZ = cos(radians(YZ));
```

```
340   float sYZ = sin(radians(YZ));
      float cYW = cos(radians(YW));
      float sYW = sin(radians(YW));
      float cZW = cos(radians(ZW));
      float sZW = sin(radians(ZW));
345   mat4 mXYZW = mat4
        ( cXY, sXY, 0.0, 0.0
        , -sXY, cXY, 0.0, 0.0
        , 0.0, 0.0, cZW, sZW
        , 0.0, 0.0, -sZW, cZW
350     );
      mat4 mXZYW = mat4
        ( cXZ, 0.0, sXZ, 0.0
        , 0.0, cYW, 0.0, sYW
        , -sXZ, 0.0, cXZ, 0.0
355     , 0.0, -sYW, 0.0, cYW
        );
      mat4 mXWYZ = mat4
        ( cXW, 0.0, 0.0, sXW
        , 0.0, cYZ, sYZ, 0.0
360     , 0.0, -sYZ, cYZ, 0.0
        , -sXW, 0.0, 0.0, cXW
        );
      mat4 m0 = mXYZW * mXZYW * mXWYZ;

365   // animation rotation transform
      float timeR = 2.0 * pi * time / Duration;
      float cSpin = cos(float(Spin) * timeR);
      float sSpin = sin(float(Spin) * timeR);
      float cPush = cos(float(Push) * timeR);
370   float sPush = sin(float(Push) * timeR);
      mat4 m1= mat4
        ( cSpin, sSpin, 0.0, 0.0
        , -sSpin, cSpin, 0.0, 0.0
        , 0.0, 0.0, cPush, sPush
375     , 0.0, 0.0, -sPush, cPush
        );

      // combined transform
      mat4 m = m1 * m0;
380


      // initialize
      void init() {
385     // initialize scene
        // sphere radius
        float sr = sin(Sphere * pi / 6.0);
        // light radius
        float lr = sin(Light * Sphere * pi / 6.0);
390     // non-physical index of refraction dependent on wavelength
        float idx = refractionIndex;
        // 24-cell has the vertices of a cube and a cross
        // three sets of eight vertices, each set in its own colour
        int k = 0;
395     // cube
        for (int sx = -1; sx <= 1; sx += 2) {
```

```
    for (int sy = -1; sy <= 1; sy += 2) {
    for (int sz = -1; sz <= 1; sz += 2) {
    for (int sw = -1; sw <= 1; sw += 2) {
      bool odd = ((sx + sy + sz + sw) & 2) == 2;
      vec4 p = m * (0.5 * vec4(float(sx), float(sy), float(sz), float(sw)));
      scene[k].center  = p;
      scene[k].sradius = sr;
      scene[k].index   = idx;
      scene[k].colour  = odd ? 0 : 1;
      scene[k+nspheres].center  = p;
      scene[k+nspheres].sradius = lr;
      scene[k+nspheres].index   = 1.0;
      scene[k+nspheres].colour  = -1;
      k = k + 1;
    }}}}
    // cross
    for (int sd = 0; sd < 4; sd += 1) {
    for (int sv = -1; sv <= 1; sv += 2) {
      vec4 p = vec4(0.0);
      p[sd] = float(sv);
      p = m * p;
      scene[k].center  = p;
      scene[k].sradius = sr;
      scene[k].index   = idx;
      scene[k].colour  = 2;
      scene[k+nspheres].center  = p;
      scene[k+nspheres].sradius = lr;
      scene[k+nspheres].index   = 1.0;
      scene[k+nspheres].colour  = -1;
      k = k + 1;
    }}
  }


  // compute the colour for pixel
  vec3 color(vec2 screen) {
    // https://en.wikipedia.org/wiki/Lambert_cylindrical_equal-area_projection
    if (abs(screen.y) >= 1.0) { return vec3(0.0); }
    vec4 eye = vec4(0.0, 0.0, 0.0, 1.0);
    vec4 ray = normalize(vec4
      ( sqrt(1.0 - screen.y * screen.y) * sin(screen.x)
      , screen.y
      , sqrt(1.0 - screen.y * screen.y) * cos(screen.x)
      , 0.0
      ));

    // compute the material at the eye
    float neardist= -2.0;
    int neark = 0;
    for (int k = 0; k < nspheres; ++k) {
      float d = dot(eye, scene[k].center);
      if (d > neardist) {
        neardist = d;
        neark = k;
      }
    }
    int material = -1;
```

```
  float sr = scene[neark].sradius;
  if (neardist > sqrt(1.0 - sr * sr)) {
    material = scene[neark].colour;
  }

  // compute the resulting CIE XYZ colour
  enqueue(eye, ray, 1.0, material, 0);
  float intensity = trace();
  return intensity * basecolour;
}
```

26 glsl/xyz_390nm-830nm.png

27 README.md

```
prismatic
=========

Physics-based ray-tracing in curved space.

Usage
-----

    ./prismatic
    ./prism2xyz < prismatic-000000.prism > prismatic-000000.xyz
    ./xyz2srgb < prismatic-000000.xyz > prismatic-000000.srgb
    ./srgb2png < prismatic-000000.srgb
    display prismatic-000000.png

References
----------

Projection:
<http://paulbourke.net/geometry/transformationprojection/> section cubic to/from↙
    ↳ spherical map
<https://en.wikipedia.org/wiki/Lambert_cylindrical_equal-area_projection>
<https://en.wikipedia.org/wiki/Spherical_coordinate_system#Cartesian_coordinates↙
    ↳ >

Ray-surface intersection distance:
<http://www.wolframalpha.com/input/?i=solve+a+%3D+b+cos%28x%29+%2B+c+sin%28x%29>
<https://en.wikipedia.org/wiki/List_of_trigonometric_identities#Double-↙
    ↳ angle_formulae>

Reflection and refraction:
<https://en.wikipedia.org/wiki/Lambertian_reflectance>
<https://en.wikipedia.org/wiki/Fresnel_equations#Power_or_intensity_equations>
<https://en.wikipedia.org/wiki/Snell%27s_law#Vector_form>
<https://www.opengl.org/sdk/docs/man4/html/reflect.xhtml>
<https://www.opengl.org/sdk/docs/man4/html/refract.xhtml>
<https://en.wikipedia.org/wiki/Kramers%E2%80%93Kronig_relations>
```

```
Absorption and Emission:
<https://en.wikipedia.org/wiki/Beer%E2%80%93Lambert_law>
<http://en.wikipedia.org/wiki/Refractive_index#Complex_refractive_index>
<https://en.wikipedia.org/wiki/Absorption_spectroscopy>
<https://en.wikipedia.org/wiki/Emission_spectrum>

Colour, wavelength, CIE XYZ, sRGB:
<http://cvrl.ioo.ucl.ac.uk> section database / CMFs
<https://en.wikipedia.org/wiki/SRGB#Specification_of_the_transformation>
<https://en.wikipedia.org/wiki/Illuminant_D65>
<https://en.wikipedia.org/wiki/CMYK_color_model>

Materials:
<http://refractiveindex.info/>
<http://en.wikipedia.org/wiki/Corundum>
<http://en.wikipedia.org/wiki/Sapphire>
<http://hypertextbook.com/facts/2007/GaryChang.shtml>
<http://www.jewelinfo4u.com/Ruby_Identification.aspx>
<http://webbook.nist.gov/chemistry/>
<http://webbook.nist.gov/chemistry/uv-vis/>
<https://en.wikipedia.org/wiki/Crown_glass_%28optics%29>
<https://en.wikipedia.org/wiki/Flint_glass>


Benchmarks
----------

Fixed size (4), varying depth:

    real       user       sys
1 0m01.911s 0m06.948s 0m0.016s
2 0m05.534s 0m20.909s 0m0.012s
3 0m12.855s 0m48.483s 0m0.028s
4 0m26.562s 1m42.266s 0m0.040s
5 0m54.689s 3m30.053s 0m0.116s
6 1m51.469s 7m02.346s 0m0.172s

Fixed depth (3), varying size:

1 0m00.294s 00m00.980s 0m0.008s
2 0m00.806s 00m03.096s 0m0.004s
3 0m03.333s 00m12.277s 0m0.024s
4 0m12.893s 00m48.647s 0m0.020s
5 0m50.431s 03m12.780s 0m0.076s
6 3m18.849s 12m46.232s 0m0.356s


Optimisations
-------------

Phase 1:

baseline port from GLSL

real    0m43.529s
user    0m43.519s
sys     0m0.000s
```

```
early escape, less trig

real    0m33.588s
user    0m33.478s
sys     0m0.080s

tangent transport

real    0m26.727s
user    0m26.718s
sys     0m0.004s

Phase 2:

baseline

real    1m1.375s
user    2m15.520s
sys     0m0.028s

omp schedule pragmas

real    0m33.954s
user    2m14.184s
sys     0m0.028s
```