raymond

Claude Heiland-Allen

2018–2019

Contents

1 examples/Balls.frag

```
#version 330 compatibility

/*
Raymond - a physics-inspired ray tracer for Fragmentarium
Copyright (C) 2018  Claude Heiland-Allen
License GPL3+ <http://www.gnu.org/licenses/>
*/

#include "Raymond-sRGB.frag"
#include "Raymond.frag"

uint hash(uint a)
{
  return hash_burtle_9(a);
}

Ray camera(Random PRNG, Camera C, vec2 coord, out float intensity)
{
  return pinhole(PRNG, C, -coord, pinhole_uniforms(), intensity);
}

float raytrace(Random PRNG, Ray V, out Hit h)
{
  return raytrace_de(PRNG, V, raytrace_de_uniforms(), h);
}

vec3 film(Random PRNG, float wavelength, float intensity)
{
  return xyz2rgb(observer(wavelength) * intensity);
}

float screen(vec4 tex, float wavelength)
{
  return 0.0;
}

vec4 light(Random PRNG, vec3 from, vec3 dir)
{
  return vec4(0.0);
}


#define SCENE(hit,surface,scene_tag) \
hit scene(scene_tag tag, Random PRNG, Ray V) \
{ \
  surface G = Plane(tag, Identity(), V, Z, 0.0); \
  return Union6 \
    ( Light(Invert(Sphere(tag, Scale(50.0), V)), V, \
        D65(V.wavelength) * 5.0 * pow(V.origin.z / 50.0, 8.0)) \
    , Glass  (srand(PRNG, 1), Sphere(tag, Translate(vec3(0.0,-4.5,1.0)), V), V) \
    , Quartz (srand(PRNG, 2), Sphere(tag, Translate(vec3(0.0,-1.5,1.0)), V), V) \
    , Water  (srand(PRNG, 3), Sphere(tag, Translate(vec3(0.0, 1.5,1.0)), V), V) \
```

```
    , Diffuse(srand(PRNG, 4), Sphere(tag, Translate(vec3(0.0, 4.5,1.0)), V), V, ↙
      ↳ 0.5) \
    , Checkerboard \
      ( V.origin \
      , Diffuse(srand(PRNG, 5), G, V, 0.3) \
      , Diffuse(srand(PRNG, 6), G, V, 0.7) \
      ) \
    ); \
}

SCENE(Hit,Surface,Scene_HIT)
#if 1
// fast
SCENE(float,float,Scene_DE)
#else
// slow
float scene(Scene_DE tag, Random PRNG, Ray V)
{
  Scene_HIT HIT;
  return scene(HIT, PRNG, V).surface.de;
}
#endif

#preset Default
FOV = 0.575
Eye = 8.703398,3.894242,3.333333
Target = 1.140564,0.6462541,0.3816794
Up = -0.3986355,-0.3137904,0.7589191
Steps = 100
Depth = 25
Acne = -16.51064
Aperture = 0.0299999
Size = 35
Wavelengths = 300,780
Exposure = 2
#endpreset
```

2 examples/Bubbles.frag

```
#version 330 compatibility

/*
Raymond - a physics-inspired ray tracer for Fragmentarium
Copyright (C) 2018  Claude Heiland-Allen
License GPL3+ <http://www.gnu.org/licenses/>
*/

#include "Raymond-sRGB.frag"
#include "Raymond.frag"

#group Bubble
uniform float Thickness; slider[100.0,1000.0,10000.0]

// render as lights instead of bubbles
const bool DEBUG = false;

uint hash(uint a)
```

```
{
  return hash_burtle_9(a);
}

Ray camera(Random PRNG, Camera C, vec2 coord, out float intensity)
{
  return pinhole(PRNG, C, -coord, pinhole_uniforms(), intensity);
}

float raytrace(Random PRNG, Ray V, out Hit h)
{
  return raytrace_de(PRNG, V, raytrace_de_uniforms(), h);
}

vec3 film(Random PRNG, float wavelength, float intensity)
{
  return xyz2rgb(observer(wavelength) * intensity);
}

float screen(vec4 tex, float wavelength)
{
  return CRT(sRGB2linear(tex.rgb), wavelength);
}

vec4 light(Random PRNG, vec3 from, vec3 dir)
{
  return vec4(0.0);
}


vec4 bubble(vec3 origin, int n)
{
  vec3 lo = floor(origin);
  vec3 md = vec3(lo) + vec3((ivec3(n) & ivec3(1, 2, 4)) >> ivec3(0, 1, 2));
  // pseudo-random hash of integer grid coordinates
  Random PRNG2;
  PRNG2.seed = hash(md);
  PRNG2.index = 0u;
  Random PRNG3;
  PRNG3.seed = hash(md.xy);
  PRNG3.index = 0u;
  vec4 posr;
  posr.xyz = uniform3(PRNG2) - vec3(0.5) + md;
  posr.w = uniform1(PRNG3) * 0.125 * (0.9 - smoothstep(4.0, 6.0, length(posr.xyz
      ↘ )));
  return posr;
}

// https://en.wikipedia.org/wiki/Cubic_Hermite_spline#
    ↘ Interpolation_on_the_unit_interval_with_matched_derivatives_at_endpoints
vec4 bubble4(vec3 origin, int n, float t)
{
  const mat4 M = mat4
    (  0.0,  2.0, 0.0,  0.0
    , -1.0,  0.0, 1.0,  0.0
    ,  2.0, -5.0, 4.0, -1.0
    , -1.0, 3.0, -3.0,  1.0
```

```
    ) / 2.0;
  vec4 u = vec4(1.0, t, t * t, t * t * t);
  mat4 p = mat4
    ( bubble(origin - Z      , n)
    , bubble(origin          , n)
    , bubble(origin + Z      , n)
    , bubble(origin + Z * 2.0, n)
    );
  return u * transpose(M) * transpose(p);
}

#define SCENE(hit,surface,scene_tag) \
hit scene(scene_tag tag, Random PRNG, Ray V) \
{ \
  float t = time / 5.0; \
  float FPS = 25.0; \
  float shutter = 0.25 / FPS; \
  t += shutter * halton1(srand(PRNG, 5)); \
  t -= floor(t); \
  hit h = SkyBox(tag, Identity(), V); \
  for (int n = 0; n < 8; ++n) \
  { \
    for (int m = -1; m < 2; ++m) \
    { \
      vec4 posr = bubble4(V.origin - float(m) * Z, n, t); \
      if (posr.w > 0.0) \
      { \
        float thick = Thickness * clamp(1.0 - (V.origin.z - posr.z) / posr.w, ↙
            ↳ 0.01, 1.99); \
        vec2 nk = Water_nk(V.wavelength); \
        surface ball = Sphere(tag, compose(Scale(posr.w), Translate(posr.xyz)), ↙
            ↳ V); \
        h = Union(h, DEBUG ? Light(ball, V, D65(V.wavelength)) : SoapBubble \
          ( srand(PRNG, 4 + n * 3 + m + 1) \
          , ball \
          , V \
          , thick \
          , nk \
          )); \
      } \
    } \
  } \
  return h; \
}

SCENE(Hit,Surface,Scene_HIT)
#if 1
// 16fps / 3.8fps / 1.9fps
// fast
SCENE(float,float,Scene_DE)
#else
// 15fps / 3.8fps / 1.9fps
// slow
float scene(Scene_DE tag, Random PRNG, Ray V)
{
  Scene_HIT HIT;
  return scene(HIT, PRNG, V).surface.de;
```

```
}
#endif

#preset Default
Exposure = 100
Steps = 100
FOV = 1
Eye = 0,0,0
Target = 0,1,1
Up = 0,0,1
Background = ../images/Berlin.equi.jpg
Distance = 100
Depth = 10
Acne = -16
Aperture = 0.035,0.02
Size = 35
Wavelengths = 300,780
Thickness = 1000
#endpreset
```

3 examples/Cubism.frag

```
#version 330 compatibility

/*
Raymond - a physics-inspired ray tracer for Fragmentarium
Copyright (C) 2018  Claude Heiland-Allen
License GPL3+ <http://www.gnu.org/licenses/>
*/

#include "Raymond-sRGB.frag"
#include "Raymond.frag"

// animation parameters
const float bpm = 130.0;
const float beats = 512.0;
const float seconds = beats / bpm * 60.0;
const float fps = 25.0;
const float shutter = 0.5;

struct Animation
{
  Transform A, B, C, D;
  float t1, t2;
};

Animation animation(Random PRNG)
{
  float t = (time + shutter / fps * uniform1(PRNG)) / seconds;
  t -= floor(t);
  t *= 2.0;
  // generate quantized noise
  Random r1, r2, r3, r4;
  r1.seed = hash(vec2(mod(floor(t + 0.0), 2.0), 1.0)); r1.index = 0u;
  r2.seed = hash(vec2(mod(floor(t + 1.0), 2.0), 1.0)); r2.index = 0u;
  r3.seed = hash(vec2(mod(floor(t + 0.5), 2.0), 2.0)); r3.index = 0u;
  r4.seed = hash(vec2(mod(floor(t + 1.5), 2.0), 2.0)); r4.index = 0u;
```

```
  // random unit quaternions (orientations)
  vec4 q1, q2, q3, q4;
  q1 = normalize(gaussian4(r1));
  q2 = normalize(gaussian4(r2));
  q3 = normalize(gaussian4(r3));
  q4 = normalize(gaussian4(r4));
  // interpolate
  vec4 p1 = slerp(q1, q2, t - floor(t));
  vec4 p2 = slerp(q3, q4, t + 0.5 - floor(t + 0.5));
  // faster layer on top
  t *= 4.0;
  // generate quantized noise
  r1.seed = hash(vec2(mod(floor(t + 0.0), 8.0), 3.0)); r1.index = 0u;
  r2.seed = hash(vec2(mod(floor(t + 1.0), 8.0), 3.0)); r2.index = 0u;
  r3.seed = hash(vec2(mod(floor(t + 0.5), 8.0), 4.0)); r3.index = 0u;
  r4.seed = hash(vec2(mod(floor(t + 1.5), 8.0), 4.0)); r4.index = 0u;
  // random film thicknesses
  float t1 = mix(100.0, 1000.0, uniform1(r1));
  float t2 = mix(100.0, 1000.0, uniform1(r2));
  float t3 = mix(100.0, 1000.0, uniform1(r3));
  float t4 = mix(100.0, 1000.0, uniform1(r4));
  float a1 = mix(t1, t2, t - floor(t));
  float a2 = mix(t3, t4, t + 0.5 - floor(t + 0.5));
  // return
  Animation a;
  a.A = Rotate(p1);
  a.B = Rotate(p2);
  a.C = Rotate(-p1);
  a.D = Rotate(-p2);
  a.t1 = a1;
  a.t2 = a2;
  return a;
}

Animation Anim;

uint hash(uint a)
{
  return hash_burtle_9(a);
}

Ray camera(Random PRNG, Camera C, vec2 coord, out float intensity)
{
  // initialize animation once
  Anim = animation(srand(PRNG, 1));
  return pinhole(srand(PRNG, 2), C, -coord, pinhole_uniforms(), intensity);
}

float raytrace(Random PRNG, Ray V, out Hit h)
{
  return raytrace_de(PRNG, V, raytrace_de_uniforms(), h);
}

vec3 film(Random PRNG, float wavelength, float intensity)
{
  return xyz2rgb(observer(wavelength) * intensity);
}
```

```
float screen(vec4 tex, float wavelength)
{
  return CRT(sRGB2linear(tex.rgb), wavelength);
}

vec4 light(Random PRNG, vec3 from, vec3 dir)
{
  return vec4(0.0);
}


Hit scene(Scene_HIT tag, Random PRNG, Ray V)
{
  // glitch the ray tracing
  Ray U;
  U = V;
  U.origin    = forwardP(Anim.D, U.origin);
  U.direction = forwardN(Anim.D, U.direction);
  Hit h1 = ThinFilm(MengerSponge(tag, Anim.A, U, 3), V, Anim.t1, 1.0);
  U = V;
  U.origin    = forwardP(Anim.C, U.origin);
  U.direction = forwardN(Anim.C, U.direction);
  Hit h2 = ThinFilm(MengerSponge(tag, Anim.B, U, 3), V, Anim.t2, 1.0);
  return Union3
    ( Light(Invert(Sphere(tag, Scale(100.0), V)), V, D65(V.wavelength))
    , h1
    , h2
    );
}

float scene(Scene_DE tag, Random PRNG, Ray V)
{
  Scene_HIT HIT;
  return scene(HIT, PRNG, V).surface.de;
}

#preset Default
FOV = 0.4
Eye = 0,0,0
Target = 1,0,0
Up = 0,0,1
Exposure = 2
Steps = 3
Depth = 15
Acne = 0
Aperture = 0.1,0.1
Size = 35
Wavelengths = 300,780
#endpreset
```

4 examples/Lettuce.frag

```
#version 330 compatibility

/*
Raymond - a physics-inspired ray tracer for Fragmentarium
```

```
/*
  Copyright (C) 2018  Claude Heiland-Allen
  License GPL3+ <http://www.gnu.org/licenses/>
*/

#include "Raymond-sRGB.frag"
#include "Raymond.frag"

#include "Raymond-Lettuce.frag"

uniform bool Colour; checkbox[true]

// pseudo-random hash
uint hash(uint a)
{
  return hash_burtle_9(a);
}

// pinhole camera
Ray camera(Random PRNG, Camera C, vec2 coord, out float intensity)
{
  return pinhole(PRNG, C, -coord, pinhole_uniforms(), intensity);
}

// distance estimator raytracer
float raytrace(Random PRNG, Ray V, out Hit h)
{
  return raytrace_de(PRNG, V, raytrace_de_uniforms(), h);
}

// CIE observer
vec3 film(Random PRNG, float wavelength, float intensity)
{
  if (Colour)
    return xyz2rgb(observer(wavelength) * intensity);
  else
    return xyz2rgb(vec3(fp4plus(wavelength) * intensity));
}

// sRGB texture using CRT model
float screen(vec4 tex, float wavelength)
{
  return CRT(sRGB2linear(tex.rgb), wavelength);
}

// scene definition
#define SCENE(hit,surface,scene_tag) \
hit scene(scene_tag tag, Random PRNG, Ray V) \
{ \
  return Union4 \
    ( Mirror(Invert(Sphere(tag, Scale(8.0), V)), V, Screen_hump(vec3(400.0, 300.0, 1.0), V.wavelength)) \
    , Light(Sphere(tag, Translate(vec3(2.0, 5.5, 5.0)), V), V, 500.0 * D65(V.wavelength)) \
    , Diffuse(srand(PRNG, 2), Plane(tag, Identity(), V, Z, 0.0), V, Screen_hump(vec3(700.0, 100.0, 1.0), V.wavelength)) \
    , Diffuse(srand(PRNG, 3), Lettuce(tag, Translate(vec3(0.0, 0.0, 2.0)), V, 16), V, Screen_hump(vec3(580.0, 200.0, 1.0), V.wavelength)) \
```

```
    ); \
}
//     , Light(Plane(tag, Identity(), V, -Z, -6.0), V, 1.0 * D65(V.wavelength)) \

vec4 light(Random PRNG, vec3 from, vec3 dir)
{
  vec3 lpos = vec3(2.0, 5.5, 5.0);
  vec3 ldir = lpos - from;
  float t = cos(asin(1.0 / length(ldir)));
  float s = dot(normalize(ldir), normalize(dir));
  if (length(from) >= 7.999)
  {
   if (s > t) return vec4(normalize(dir), 1.0); else return vec4(0.0);
  }
  vec3 to = normalize(gaussian3(srand(PRNG, 5))) + lpos;
  return vec4(normalize(to - from), (1.0 - t) / 2.0);
}

// scene for DE with lighting
SCENE(Hit,Surface,Scene_HIT)

#ifndef DEBUG

// scene for DE only (fast)
SCENE(float,float,Scene_DE)

#else

// reuse DE with lighting (slow)
float scene(Scene_DE tag, Random PRNG, Ray V)
{
  Scene_HIT HIT;
  return scene(HIT, PRNG, V).surface.de;
}

#endif


#preset Default
Exposure = 1
FOV = 1
Eye = 4,2,4
Target = 0.5,0,2
Up = 0,0,1
Background =
Distance = 100
Steps = 100
Depth = 10
MinDist = -16
Acne = -12
Aperture = 0.0,0.0
Size = 1000
Wavelengths = 300,780
Colour = true
#endpreset
```

5 examples/Mandelbrot.frag

```
#version 330 compatibility

/*
Raymond - a physics-inspired ray tracer for Fragmentarium
Copyright (C) 2018  Claude Heiland-Allen
License GPL3+ <http://www.gnu.org/licenses/>
*/

#include "Raymond-sRGB.frag"
#include "Raymond.frag"

#group Mandelbrot
// maximum period for atoms and domains
uniform int Period; slider[1,1,128]
// minimum size of atoms and domains
uniform float MinSize; slider[-24,-10,1];
// maximum iteration count for exterior
uniform int Iterations; slider[1,1,1000];
// thickness of exterior fringe
uniform float Thickness; slider[-24,-10,1];

// standard preamble

uint hash(uint a)
{
  return hash_burtle_9(a);
}

Ray camera(Random PRNG, Camera C, vec2 coord, out float intensity)
{
  return pinhole(PRNG, C, -coord, pinhole_uniforms(), intensity);
}

float raytrace(Random PRNG, Ray V, out Hit h)
{
  return raytrace_de(PRNG, V, raytrace_de_uniforms(), h);
}

vec3 film(Random PRNG, float wavelength, float intensity) {
  return xyz2rgb(observer(wavelength) * intensity);
}

float screen(vec4 tex, float wavelength)
{
  return CRT(sRGB2linear(tex.rgb), wavelength);
}

vec4 light(Random PRNG, vec3 from, vec3 dir)
{
  return vec4(0.0);
}

// Mandelbrot numerics

// https://mathr.co.uk/blog/2018-11-17_newtons_method_for_periodic_points.html
```

```
vec2 Mandelbrot_nucleus(vec2 c0, int period)
{
  vec2 c = c0;
  for (int j = 0; j < 16; ++j)
  {
    vec4 G = cConst(1.0);
    vec4 z = cConst(0.0);
    c0 = c;
    for (int l = 1; l <= period; ++l)
    {
      z = cSqr(z) + cVar(c);
      if (l < period && period % l == 0)
        G = cMul(z, G);
    }
    G = cDiv(z, G);
    c = c0 - cDiv(G.xy, G.zw);
    if (! (c != c0)) break;
  }
  return c;
}

// https://mathr.co.uk/blog/2016-12-24_deriving_the_size_estimate.html
vec2 Mandelbrot_size(vec2 c, int period)
{
  vec2 l = vec2(1.0, 0.0);
  vec2 b = vec2(1.0, 0.0);
  vec2 z = vec2(0.0, 0.0);
  for (int i = 1; i < period; ++i)
  {
    z = cSqr(z) + c;
    l = 2.0 * cMul(z, l);
    b = b + cDiv(vec2(1.0, 0.0), l);
  }
  return cDiv(vec2(1.0, 0.0), cMul(b, cSqr(l)));
}

// https://mathr.co.uk/blog/2013-04-01↙
    ↳ _interior_coordinates_in_the_mandelbrot_set.html
// https://mathr.co.uk/blog/2015-01-26_newtons_method_for_misiurewicz_points.↙
    ↳ html
vec4 Mandelbrot_attractor(vec2 c, vec2 z_initial, int period)
{
  vec4 z = cVar(z_initial);
  vec2 dz = vec2(0.0);
  for (int j = 0; j < 16; ++j)
  {
    vec4 z0 = z;
    vec4 G = cConst(1.0);
    for (int l = 1; l <= period; ++l)
    {
      z = cSqr(z) + cConst(c);
      if (l < period && period % l == 0)
        G = cMul(z - z0, G);
    }
    dz = z.zw;
    G = cDiv(z - z0, G);
    z = cVar(z0.xy - cDiv(G.xy, G.zw));
```

```
    if (! (z != z0)) break;
  }
  return vec4(z.xy, dz);
}

// trace ray from a nucleus, more stable than trying to do it one jump
vec4 Mandelbrot_attractor_ray(vec2 nucleus, vec2 target, int period)
{
  const int steps = 16;
  vec4 P = vec4(0.0);
  for (int ray = 1; ray <= steps; ++ray)
  {
    float t = float(ray) / float(steps);
    P = Mandelbrot_attractor(mix(nucleus, target, t), P.xy, period);
  }
  return P;
}

// https://mathr.co.uk/blog/2013-12-10_atom_domain_size_estimation.html
float Mandelbrot_domain_size(vec2 c, int p)
{
  vec4 z = cVar(c);
  float abszq = length(z.xy);
  for (int q = 2; q <= p; ++q) {
    z = cSqr(z) + cVar(c);
    float abszp = length(z.xy);
    if (abszp < abszq && q < p) {
      abszq = abszp;
    }
  }
  return abszq / length(z.zw);
}

// main DE function

vec3 Mandelbrot(vec3 object, float bg)
{
  // FIXME should be able to take uniforms as arguments
  int maxPeriod = Period;
  float minSize = pow(2.0, MinSize);
  int maxIters = Iterations;
  float thickness = pow(2.0, Thickness);
  // no transformations supported
  vec2 c = object.xy;
  vec2 z = vec2(0.0, 0.0);
  // derivative for exterior distance
  vec2 dc = vec2(0.0, 0.0);
  // minimums for atom domains
  vec2 mz = vec2(0.0, 0.0);
  float mr2 = 1.0 / 0.0;
  // calculate 3x (DE,material,bubblethicknessfactor) for CSG operations
  vec3 back = vec3(bg, 0.0, 0.0);
  if (bg < 0.0)
    return back;
  vec3 far = vec3(1.0/0.0, 1.0, 0.0);
  vec3 de_nucleus  = far;
  vec3 de_domain   = far;
```

```
    vec3 de_exterior = far;
    int escaped = 0;
    int period = 1;
    for (; period <= maxPeriod; ++period)
    {
      dc = 2.0 * cMul(z, dc) + vec2(1.0, 0.0);
      z = cSqr(z) + c;
      float r2 = dot(z, z);
      if (r2 < 1.5 * mr2)
      {
        vec2 nucleus = Mandelbrot_nucleus(c, period);
        if (period > 1)
        {
          vec2 atom = cDiv(z, mz);
          float S = Mandelbrot_domain_size(nucleus, period);
          if (S > minSize)
          {
            // make a DE-traceable shell out of two surfaces inside each other
            float a = length(vec3(atom, object.z / S));
            vec3 d1 = vec3( (a - 1.01) * S, 3.0, clamp(object.z / S + 1.0, 0.01, ↙
                ↳ 1.99));
            vec3 d2 = vec3(-(a - 0.99) * S, 4.0, 0.0);
            float d = max(d1.x, d2.x);
            if (d < de_domain.x) de_domain = d1.x > d2.x ? d1 : d2;
          }
        }
        if (true)
        {
          vec2 atom = Mandelbrot_attractor_ray(nucleus, c, period).zw;
          float S = 0.5 * length(Mandelbrot_size(nucleus, period));
          if (S > minSize)
          {
            float d = (length(vec3(atom, object.z / S)) - 1.0) * S;
            if (d < de_nucleus.x) de_nucleus = vec3(d, 2.0, 0.0);
          }
        }
        if (r2 < mr2)
        {
          mz = z;
          mr2 = r2;
        }
        if (de_nucleus.x < 0.0)
        {
          escaped = -1;
          break;
        }
      }
      if (r2 > 65536.0)
      {
        escaped = 1;
        break;
      }
    }
    if (escaped == 0)
    {
      for (; period <= maxIters; ++period)
      {
```

```
      dc = 2.0 * cMul(z, dc) + vec2(1.0, 0.0);
      z = cSqr(z) + c;
      float r2 = dot(z, z);
      if (r2 > 65536.0)
      {
        escaped = 1;
        break;
      }
    }
  }
  if (escaped == 1)
  {
    float d = length(z) * log(length(z)) / length(dc);
    d = length(vec2(d, object.z)) - thickness;
    if (d < de_exterior.x) de_exterior = vec3(d, 1.0, 0.0);
  }
  // disjoint union(b, a - b) = union(b, intersection(a, invert(b)))
  // this hopefully fixes unsightly issues at atom/domain intersections
  if (isnan(de_exterior.x)) de_exterior = back;
  if (isnan(de_nucleus.x)) de_nucleus = back;
  if (isnan(de_domain.x)) de_domain = back;
  vec3 solid  = de_exterior.x < de_nucleus.x ? de_exterior : de_nucleus;
  vec3 bubble = -solid.x > de_domain.x ? vec3(-solid.x, solid.y, solid.z) : ↙
      ↪ de_domain;
  vec3 mandel = solid.x < bubble.x ? solid : bubble;
  vec3 result = mandel.x < back.x ? mandel : back;
  return result;
}

// delta normal calculation

#define DELTANORMAL(DE,E,p) normalize(vec3 \
  ( DE(p + vec3(E,0.0,0.0)) - DE(p - vec3(E,0.0,0.0)) \
  , DE(p + vec3(0.0,E,0.0)) - DE(p - vec3(0.0,E,0.0)) \
  , DE(p + vec3(0.0,0.0,E)) - DE(p - vec3(0.0,0.0,E)) \
  ))

// main entry points

Hit scene(Scene_HIT tag, Random PRNG, Ray V)
{
//  return Union(Light(Sphere(tag, Identity(), V), V, 1.0), SkyBox(tag, Identity↙
    ↪ (), V));
  Scene_DE DE;
  vec3 object = V.origin;
  Transform T = Identity();
  float bg = SkyBox(DE, T, V);
  vec3 de = Mandelbrot(object, bg);
  if (! (de.y > 0.0)) return SkyBox(tag, T, V);
  const float epsilon = 1.0e-3; // FIXME
#define M(p) Mandelbrot(p, SkyBox(DE, T, Ray((p), V.direction, V.wavelength, V.↙
    ↪ index))).x
  vec3 normal = DELTANORMAL(M, epsilon, object);
  // surface
  Surface S;
  S.position = object;
  S.normal = normal;
```

```
  S.de = de.x;
  vec2 water = Water_nk(V.wavelength);
  // FIXME the magic numbers to chose materials from DE are annoyment
  if (de.y == 1.0) return Light(S, V, 10.0);
  if (de.y == 2.0) return Glass(PRNG, S, V);
  if (de.y == 3.0) return SoapBubble(PRNG, S, V, 1000.0 * de.z, water);
  if (de.y == 4.0) return Transmit(S, V);
  return SkyBox(tag, T, V);
}

#if 1

// fast
float scene(Scene_DE tag, Random PRNG, Ray V)
{
  return Mandelbrot(V.origin, SkyBox(tag, Identity(), V)).x;
}

#else

// slow
float scene(Scene_DE tag, Random PRNG, Ray V)
{
  Scene_HIT HIT;
  return scene(HIT, PRNG, V).surface.de;
}

#endif

#preset LQ
Exposure = 1
TrigIter = 5
TrigLimit = 1.1
FOV = 0.8
Eye = -1.5,-2,-1
Target = -0.75,0,0
Up = 0,0,1
Steps = 100
Depth = 3
MinDist = -16
Acne = -12
Aperture = 0.01,0.01
Size = 35
Wavelengths = 300,780
Background = ../images/Berlin.equi.jpg
Distance = 100
Period = 2
MinSize = -10
Iterations = 100
Thickness = -10
#endpreset

#preset Default
FOV = 1
Eye = -1.5,-2,-1
Target = -0.75,0,0
Up = 0,0,1
```

```
Exposure = 1
TrigIter = 5
TrigLimit = 1.1
DebugDepth = false
DebugNormal = false
DebugBounce = false
Background = ../images/Berlin.equi.jpg
Distance = 100
Steps = 100
Depth = 10
MinDist = -16
Acne = -12
Aperture = 0.01,0.01
Size = 35
Wavelengths = 300,780
Period = 12
MinSize = -10
Iterations = 100
Thickness = -10
#endpreset

#preset Card
TrigIter = 5
TrigLimit = 1.10000000000000009
Exposure = 1
FOV = 1
Eye = -1.5,-2,-1
Target = -0.75,0,0
Up = 0,0,1
DebugDepth = false
DebugNormal = false
DebugBounce = false
Background = ../images/Grid.equi.png Locked
Distance = 100
Steps = 100
Depth = 3
MinDist = -16
Acne = -12
Aperture = 0.01,0.021
Size = 35
Wavelengths = 300,780
Period = 2
MinSize = -10
Iterations = 100
Thickness = -10
#endpreset
```

6 examples/SkyBox.frag

```
#version 330 compatibility

/*
Raymond - a physics-inspired ray tracer for Fragmentarium
Copyright (C) 2018  Claude Heiland-Allen
License GPL3+ <http://www.gnu.org/licenses/>
*/
```

```
#include "Raymond-sRGB.frag"
#include "Raymond.frag"

#group Bubble

// average thickness of bubble's soap film in nm
uniform float Thickness; slider[0.0,1000.0,10000.0]

// pseudo-random hash
uint hash(uint a)
{
  return hash_burtle_9(a);
}

Transform Spin;

// pinhole camera
Ray camera(Random PRNG, Camera C, vec2 coord, out float intensity)
{
  float t = pi - 2.0 * pi * time / 15.0;
  Spin = Rotate(vec4(Z * sin(t/2.0), cos(t/2.0)));
  return pinhole(PRNG, C, -coord, pinhole_uniforms(), intensity);
}

// distance estimator raytracer
float raytrace(Random PRNG, Ray V, out Hit h)
{
  return raytrace_de(PRNG, V, raytrace_de_uniforms(), h);
}

// CIE observer
vec3 film(Random PRNG, float wavelength, float intensity)
{
  return xyz2rgb(observer(wavelength) * intensity);
}

// skybox sRGB texture using CRT model
float screen(vec4 tex, float wavelength)
{
  return CRT(sRGB2linear(tex.rgb), wavelength);
}

vec4 light(Random PRNG, vec3 from, vec3 dir)
{
  return vec4(0.0);
}


// scene definition
#define SCENE(hit,surface,scene_tag) \
hit scene(scene_tag tag, Random PRNG, Ray V) \
{ \
  return Union \
    ( SkyBox(tag, Spin, V) \
    , SoapBubble \
      ( PRNG \
      , Sphere(tag, Identity(), V) \
```

```
        , V \
        , Thickness * (1.0 - V.origin.z) \
        , Water_nk(V.wavelength) \
        ) \
    ); \
}

// scene for DE with lighting
SCENE(Hit,Surface,Scene_HIT)

#ifndef DEBUG

// scene for DE only (fast)
SCENE(float,float,Scene_DE)

#else

// reuse DE with lighting (slow)
float scene(Scene_DE tag, Random PRNG, Ray V)
{
  Scene_HIT HIT;
  return scene(HIT, PRNG, V).surface.de;
}

#endif


#preset Default
FOV = 1
Eye = 0,2.5,0
Target = 0,0,0
Up = 0,0,1
Exposure = 10
Distance = 100
Steps = 100
Depth = 10
MinDist = -16
Acne = -16
Size = 120
Aperture = 0.25,1
Wavelengths = 300,780
Thickness = 1000
#endpreset

#preset Berlin
Exposure = 1
Background = ../images/Berlin.equi.jpg
#endpreset

#preset Paris
Exposure = 2
Background = ../images/Paris.equi.jpg
#endpreset

#preset Grid
Exposure = 2
Background = ../images/Grid.equi.png
```

```
#endpreset
```

7 examples/SoapBubble.frag

```
#version 330 compatibility

/*
Raymond - a physics-inspired ray tracer for Fragmentarium
Copyright (C) 2018  Claude Heiland-Allen
License GPL3+ <http://www.gnu.org/licenses/>
*/

#include "Raymond-sRGB.frag"
#include "Raymond.frag"

#group Soap
uniform float Thickness; slider[100.0,1000.0,10000.0]

uint hash(uint a)
{
  return hash_burtle_9(a);
}

Ray camera(Random PRNG, Camera C, vec2 coord, out float intensity)
{
//  C.origin.z = 1.0 + 0.5 * sin(2.0 * pi * time / 4.0);
  return pinhole(PRNG, C, -coord, pinhole_uniforms(), intensity);
}

float raytrace(Random PRNG, Ray V, out Hit h)
{
  return raytrace_de(PRNG, V, raytrace_de_uniforms(), h);
}

vec3 film(Random PRNG, float wavelength, float intensity)
{
  return xyz2rgb(observer(wavelength) * intensity);
}

float screen(vec4 tex, float wavelength)
{
  return CRT(sRGB2linear(tex.rgb), wavelength);
}

vec4 light(Random PRNG, vec3 from, vec3 dir)
{
  return vec4(0.0);
}

#define SCENE(hit,surface,scene_tag) \
hit scene(scene_tag tag, Random PRNG, Ray V) \
{ \
  surface Ground = Plane(tag, Identity(), V, Z, 0.0); \
  return Union3 \
    ( Light(Invert(Sphere(tag, Scale(8.0), V)), V, D65(V.wavelength) * pow(V.↙
        ↳ origin.z / 8.0, 16.0) * 32.0) \
```

```
    , SoapBubble(srand(PRNG, 1), Sphere(tag, Translate(vec3(0.0, 0.0, 1.0)), V),↙
      ↪ V, Thickness * (2.0 - V.origin.z), Water_nk(V.wavelength)) \
    , Checkerboard \
      ( V.origin \
      , Diffuse(srand(PRNG, 2), Ground, V, 0.5) \
      , Diffuse(srand(PRNG, 2), Ground, V, 0.2) \
      ) \
    ); \
}

SCENE(Hit,Surface,Scene_HIT)
#if 1
// fast
SCENE(float,float,Scene_DE)
#else
// slow
float scene(Scene_DE tag, Random PRNG, Ray V)
{
  Scene_HIT HIT;
  return scene(HIT, PRNG, V).surface.de;
}
#endif


#preset Default
Exposure = 1
FOV = 0.4
Eye = 0,6,0.25
Target = 0,0,1
Up = 0,0,1
Steps = 300
Depth = 10
Acne = -16
Aperture = 0.035
Size = 35
Wavelengths = 300,780
#endpreset
```

8 examples/SpongeAndBulb.frag

```
#version 330 compatibility

/*
Raymond - a physics-inspired ray tracer for Fragmentarium
Copyright (C) 2018  Claude Heiland-Allen
License GPL3+ <http://www.gnu.org/licenses/>
*/

#include "Raymond-sRGB.frag"
#include "Raymond.frag"

uniform bool Colour; checkbox[true]

// pseudo-random hash
uint hash(uint a)
{
  return hash_burtle_9(a);
```

```
}

// pinhole camera
Ray camera(Random PRNG, Camera C, vec2 coord, out float intensity)
{
  return pinhole(PRNG, C, -coord, pinhole_uniforms(), intensity);
}

// distance estimator raytracer
float raytrace(Random PRNG, Ray V, out Hit h)
{
  return raytrace_de(PRNG, V, raytrace_de_uniforms(), h);
}

// CIE observer
vec3 film(Random PRNG, float wavelength, float intensity)
{
  if (Colour)
    return xyz2rgb(observer(wavelength) * intensity);
  else
    return xyz2rgb(vec3(fp4plus(wavelength) * intensity));
}

// sRGB texture using CRT model
float screen(vec4 tex, float wavelength)
{
  return CRT(sRGB2linear(tex.rgb), wavelength);
}

// scene definition
#define SCENE(hit,surface,scene_tag) \
hit scene(scene_tag tag, Random PRNG, Ray V) \
{ \
  return Union5 \
    ( Mirror(Invert(Sphere(tag, Scale(8.0), V)), V, Screen_hump(vec3(400.0, 300.0, 1.0), V.wavelength)) \
    , Light(Sphere(tag, Translate(vec3(2.0, 5.5, 5.0)), V), V, 500.0 * D65(V.wavelength)) \
    , Diffuse(srand(PRNG, 2), Plane(tag, Identity(), V, Z, 0.0), V, Screen_hump(vec3(700.0, 100.0, 1.0), V.wavelength)) \
    , Diffuse(srand(PRNG, 3), MengerSponge(tag, Translate(vec3(0.0, 0.0, 1.0)), V, 6), V, Screen_hump(vec3(580.0, 200.0, 1.0), V.wavelength)) \
    , Diffuse(srand(PRNG, 4), Mandelbulb(tag, compose(Rotate(vec4(sin(pi/4.0) * normalize(vec3(1.0)), cos(pi/4.0))), Translate(vec3(0.0, 0.0, 2.0 + sqrt(8.0)/3.0))), V, 8, 10), V, 1.0) \
    ); \
}
//     , Light(Plane(tag, Identity(), V, -Z, -6.0), V, 1.0 * D65(V.wavelength)) \

vec4 light(Random PRNG, vec3 from, vec3 dir)
{
  vec3 lpos = vec3(2.0, 5.5, 5.0);
  vec3 ldir = lpos - from;
  float t = cos(asin(1.0 / length(ldir)));
  float s = dot(normalize(ldir), normalize(dir));
  if (length(from) >= 7.999)
  {
```

```
    if (s > t) return vec4(normalize(dir), 1.0); else return vec4(0.0);
  }
  vec3 to = normalize(gaussian3(srand(PRNG, 5))) + lpos;
  return vec4(normalize(to - from), (1.0 - t) / 2.0);
}

// scene for DE with lighting
SCENE(Hit,Surface,Scene_HIT)

#ifndef DEBUG

// scene for DE only (fast)
SCENE(float,float,Scene_DE)

#else

// reuse DE with lighting (slow)
float scene(Scene_DE tag, Random PRNG, Ray V)
{
  Scene_HIT HIT;
  return scene(HIT, PRNG, V).surface.de;
}

#endif


#preset Default
Exposure = 1
FOV = 1
Eye = 4,2,4
Target = 0.5,0,2
Up = 0,0,1
Background =
Distance = 100
Steps = 100
Depth = 10
MinDist = -16
Acne = -12
Aperture = 0.0,0.0
Size = 1000
Wavelengths = 300,780
Colour = true
#endpreset
```

9 .gitignore

```
-
```

10 images/berlin.sh

```
#!/bin/sh
# image page: <https://commons.wikimedia.org/wiki/File:Reichstagsufer,_Berlin-↙
    ↪ Mitte,_360x180,_160401,_ako.jpg>
# author: Ansgar Koreng
# license: CC BY-SA 3.0 (DE) <https://creativecommons.org/licenses/by-sa/3.0/de/↙
    ↪ deed.de>
```

```
wget -c https://upload.wikimedia.org/wikipedia/commons/7/71/Reichstagsufer%2C_C_Berlin-Mitte%2C_360x180%2C_160401%2C_ako.jpg
cp Reichstag*.jpg Berlin.equi.jpg
```

11 images/.gitignore

```
*.exr
*.jpg
*.png
```

12 images/grid.sh

```
#!/bin/sh
# image page: <https://dmswart.com/2016/06/28/drawing-a-panorama/>
# author: D. M. Swart
# license: unknown
wget -c https://c2.staticflickr.com/2/1482/26363697850_53505495a2_o_d.png
cp 26363697850_53505495a2_o_d.png Grid.equi.png
```

13 images/paris.sh

```
#!/bin/sh
# image page: https://commons.wikimedia.org/wiki/File:Paris_s%27%C3%A9veille_equirectangular_panorama.jpg
# author: Alexandre Duret-Lutz <https://www.flickr.com/photos/gadl/456185667/>
# license: CC-BY-SA
wget -c https://upload.wikimedia.org/wikipedia/commons/c/c6/Paris_s%27%C3%A9veille_equirectangular_panorama.jpg
cp Paris_s\'éveille_equirectangular_panorama.jpg Paris.equi.jpg
#convert Paris.equi.jpg Paris.equi.exr
#exrenvmap -c -w 2048 Paris.equi.exr Paris.cube.exr
#convert Paris.cube.exr Paris.cube.png
```

14 include/Raymond-Core.frag

```
#donotrun

/*
Raymond - a physics-inspired ray tracer for Fragmentarium
Copyright (C) 2018  Claude Heiland-Allen
License GPL3+ <http://www.gnu.org/licenses/>
*/

// simple 3D camera, API defined by FragM
#camera 3D
#vertex
#group Camera
uniform float FOV; slider[0.0,0.4,2.0] NotLockable
uniform vec3 Eye; slider[(-50,-50,-50),(0,0,-10),(50,50,50)] NotLockable
uniform vec3 Target; slider[(-50,-50,-50),(0,0,0),(50,50,50)] NotLockable
uniform vec3 Up; slider[(0,0,0),(0,1,0),(0,0,0)] NotLockable

uniform vec2 pixelSize;

varying vec3 Forward;
```

```
varying vec3 UpOrtho;
varying vec3 Right;
varying vec2 coord;
varying vec2 texCoord;

void main(void)
{
  Forward = normalize(Target - Eye);
  UpOrtho = normalize(Up - dot(Forward, Up) * Forward);
  Right = normalize(cross(Forward, UpOrtho));
  coord = (gl_ProjectionMatrix * gl_Vertex).xy;
  coord.x *= pixelSize.y / pixelSize.x;
  texCoord = (gl_Vertex.xy + vec2(1.0)) * 0.5;
  gl_Position =  gl_Vertex;
}

#endvertex

varying vec3 Forward;
varying vec3 UpOrtho;
varying vec3 Right;
varying vec2 coord;
varying vec2 texCoord;

uniform float FOV;
uniform vec3 Eye;
uniform vec2 pixelSize;

uniform sampler2D backbuffer;
uniform int subframe;
uniform float time;

#group Debug
uniform bool DebugDepth; checkbox[false]
uniform bool DebugNormal; checkbox[false]
uniform bool DebugBounce; checkbox[false]

void main(void)
{
  // seed pseudo-random number generator based on coordinates
  Random PRNG;
  PRNG.seed = hash(vec4(coord, time, subframe));
  PRNG.index = uint(subframe);
  // encapsulate camera parameters from uniform variables
  Camera C;
  C.origin = Eye;
  C.X = Right;
  C.Y = UpOrtho;
  C.Z = Forward;
  C.fieldOfView = FOV;
  // render image
  float factor = 1.0;
  Ray V = camera(srand(PRNG, 1), C, coord, factor);
  Hit H;
  float I = raytrace(srand(PRNG, 2), V, H);
  vec3 rgb = film(srand(PRNG, 3), V.wavelength, I * factor);
  // accumulate linear RGB values
```

```
  vec4 next = vec4(rgb, 1.0);
  if (DebugDepth)  next.xyz = vec3(distance(V.origin, H.surface.position));
  if (DebugNormal) next.xyz = vec3(0.5 + 0.5 * H.surface.normal);
  if (DebugBounce) next.xyz = vec3(0.5 + 0.5 * H.ray.direction);
  if (! (next == next)) next = vec4(0.0); // NaN check
  if (! (dot(next, next) < 1.0/0.0)) next = vec4(0.0); // Infinity check
  vec4 prev = texture(backbuffer, texCoord);
  gl_FragColor = prev + next;
}
```

15 include/Raymond-D65.frag

```
#donotrun

/*
Raymond - a physics-inspired ray tracer for Fragmentarium
Copyright (C) 2018  Claude Heiland-Allen
License GPL3+ <http://www.gnu.org/licenses/>
*/

// standard illuminant D65

float D65(float nm)
{
const float d65_min = 300.0;
const float d65_max = 780.0;
const float d65_step = 5.0;
#define d65_count 97
const float d65_v[d65_count] = float[d65_count]
( 0.034100
, 1.664300
, 3.294500
, 11.765200
, 20.236000
, 28.644700
, 37.053500
, 38.501100
, 39.948800
, 42.430200
, 44.911700
, 45.775000
, 46.638300
, 49.363700
, 52.089100
, 51.032300
, 49.975500
, 52.311800
, 54.648200
, 68.701500
, 82.754900
, 87.120400
, 91.486000
, 92.458900
, 93.431800
, 90.057000
, 86.682300
, 95.773600
```

```
      , 104.865000
      , 110.936000
      , 117.008000
      , 117.410000
50    , 117.812000
      , 116.336000
      , 114.861000
      , 115.392000
      , 115.923000
55    , 112.367000
      , 108.811000
      , 109.082000
      , 109.354000
      , 108.578000
60    , 107.802000
      , 106.296000
      , 104.790000
      , 106.239000
      , 107.689000
65    , 106.047000
      , 104.405000
      , 104.225000
      , 104.046000
      , 102.023000
70    , 100.000000
      , 98.167100
      , 96.334200
      , 96.061100
      , 95.788000
75    , 92.236800
      , 88.685600
      , 89.345900
      , 90.006200
      , 89.802600
80    , 89.599100
      , 88.648900
      , 87.698700
      , 85.493600
      , 83.288600
85    , 83.493900
      , 83.699200
      , 81.863000
      , 80.026800
      , 80.120700
90    , 80.214600
      , 81.246200
      , 82.277800
      , 80.281000
      , 78.284200
95    , 74.002700
      , 69.721300
      , 70.665200
      , 71.609100
      , 72.979000
100   , 74.349000
      , 67.976500
      , 61.604000
```

```
    , 65.744800
    , 69.885600
105 , 72.486300
    , 75.087000
    , 69.339800
    , 63.592700
    , 55.005400
110 , 46.418200
    , 56.611800
    , 66.805400
    , 65.094100
    , 63.382800
115 );
      float f = (nm - d65_min) / d65_step;
      int i0 = int(floor(f));
      int i1 = i0 + 1;
      float f1 = f - float(i0);
120   if (i0 < 0) return 0.0;
      if (i1 > d65_count - 1) return 0.0;
      return mix(d65_v[i0], d65_v[i1], f1) * 0.01;
    #undef d65_count
    }
```

16 include/Raymond-Defs.frag

```
    #donotrun

    /*
    Raymond - a physics-inspired ray tracer for Fragmentarium
  5 Copyright (C) 2018  Claude Heiland-Allen
    License GPL3+ <http://www.gnu.org/licenses/>
    */

    struct Random  { uint seed, index; };
 10 struct Camera  { vec3 origin, X, Y, Z; float fieldOfView; };
    struct Ray     { vec3 origin, direction; float wavelength; vec2 index; };
    struct Surface { vec3 position; vec3 normal; float de; };
    struct Hit     { Surface surface; Ray ray; float factor; float emit; };

 15 // implement these
    uint hash(uint n);
    Ray camera(Random PRNG, Camera C, vec2 coord, out float intensity);
    float /* intensity */ raytrace(Random PRNG, Ray V, out Hit h);
    vec3 /* linear RGB */ film(Random PRNG, float wavelength, float intensity);
 20 vec4 /* direction, weight */ light(Random PRNG, vec3 from, vec3 dir);

    // perturb a random number generator
    Random srand(Random PRNG, int seed)
    {
 25   PRNG.seed = hash(vec2(hash(PRNG.seed), hash(uint(seed))));
      return PRNG;
    }

    // scene definition
 30
    struct Scene_DE  { int dummy; };
    struct Scene_HIT { int dummy; };
```

```
float scene(Scene_DE  tag, Random PRNG, Ray V);
Hit   scene(Scene_HIT tag, Random PRNG, Ray V);
```

17 include/Raymond-FP4Plus.frag

```
#donotrun

/*
Raymond - a physics-inspired ray tracer for Fragmentarium
Copyright (C) 2018  Claude Heiland-Allen
License GPL3+ <http://www.gnu.org/licenses/>
*/

// Ilford FP4 Plus black and white film

float fp4plus(float wavelength)
{
const int fp4plus_u[350] = int[350]
( 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
, 0x00, 0x00, 0x00, 0x00, 0x0d, 0x22, 0x35, 0x42, 0x4f, 0x5a
, 0x62, 0x68, 0x6d, 0x71, 0x75, 0x78, 0x7c, 0x7f, 0x81, 0x83
, 0x86, 0x87, 0x89, 0x8b, 0x8d, 0x8f, 0x91, 0x94, 0x95, 0x98
, 0x9b, 0x9c, 0x9f, 0xa1, 0xa3, 0xa5, 0xa7, 0xa9, 0xab, 0xac
, 0xad, 0xaf, 0xb0, 0xb1, 0xb2, 0xb3, 0xb4, 0xb4, 0xb5, 0xb5
, 0xb6, 0xb6, 0xb6, 0xb7, 0xb7, 0xb7, 0xb8, 0xb8, 0xb9, 0xb9
, 0xba, 0xbb, 0xbb, 0xbc, 0xbd, 0xbd, 0xbe, 0xbe, 0xc0, 0xc0
, 0xc2, 0xc2, 0xc3, 0xc4, 0xc5, 0xc6, 0xc7, 0xc8, 0xca, 0xcb
, 0xcc, 0xcd, 0xce, 0xcf, 0xd0, 0xd1, 0xd2, 0xd2, 0xd3, 0xd4
, 0xd4, 0xd5, 0xd5, 0xd6, 0xd6, 0xd7, 0xd7, 0xd7, 0xd8, 0xd8
, 0xd8, 0xd8, 0xd9, 0xd9, 0xda, 0xda, 0xda, 0xda, 0xda, 0xda
, 0xda, 0xda, 0xda, 0xda, 0xda, 0xda, 0xda, 0xda, 0xda, 0xda
, 0xda, 0xda, 0xda, 0xda, 0xda, 0xda, 0xda, 0xd9, 0xd9, 0xd9
, 0xd9, 0xd8, 0xd8, 0xd8, 0xd8, 0xd7, 0xd7, 0xd7, 0xd7, 0xd6
, 0xd6, 0xd6, 0xd6, 0xd5, 0xd5, 0xd5, 0xd5, 0xd5, 0xd5, 0xd5
, 0xd6, 0xd6, 0xd6, 0xd7, 0xd7, 0xd8, 0xd8, 0xda, 0xda, 0xdb
, 0xdc, 0xdd, 0xde, 0xde, 0xdf, 0xe0, 0xe0, 0xe0, 0xe1, 0xe2
, 0xe2, 0xe2, 0xe3, 0xe3, 0xe3, 0xe3, 0xe3, 0xe3, 0xe3, 0xe3
, 0xe3, 0xe3, 0xe3, 0xe3, 0xe2, 0xe2, 0xe3, 0xe3, 0xe3, 0xe3
, 0xe3, 0xe4, 0xe4, 0xe5, 0xe6, 0xe6, 0xe7, 0xe7, 0xe8, 0xe9
, 0xe9, 0xea, 0xea, 0xeb, 0xec, 0xec, 0xed, 0xee, 0xee, 0xef
, 0xef, 0xf0, 0xf1, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf5, 0xf6
, 0xf6, 0xf7, 0xf8, 0xf8, 0xf9, 0xf9, 0xfa, 0xfa, 0xfb, 0xfb
, 0xfb, 0xfc, 0xfc, 0xfc, 0xfd, 0xfd, 0xfd, 0xfe, 0xfe, 0xfe
, 0xfe, 0xfe, 0xfe, 0xfe, 0xfe, 0xfe, 0xfe, 0xfe, 0xfe, 0xfe
, 0xfd, 0xfd, 0xfd, 0xfc, 0xfc, 0xfb, 0xfb, 0xfb, 0xfa, 0xfa
, 0xfa, 0xfa, 0xfa, 0xfa, 0xfa, 0xfa, 0xfa, 0xfa, 0xfa, 0xfa
, 0xfa, 0xf9, 0xf8, 0xf6, 0xf4, 0xf1, 0xee, 0xeb, 0xe8, 0xe5
, 0xe2, 0xde, 0xdb, 0xd7, 0xd3, 0xcf, 0xcb, 0xc7, 0xc2, 0xbd
, 0xb8, 0xb2, 0xad, 0xa7, 0xa1, 0x9b, 0x94, 0x8c, 0x84, 0x7c
, 0x72, 0x68, 0x5c, 0x4f, 0x3d, 0x2e, 0x1c, 0x0d, 0x00, 0x00
, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
);
  float f = wavelength - 350.0;
  int i0 = int(floor(f));
```

```
  int i1 = i0 + 1;
  float f1 = f - float(i0);
  if (i0 < 0) return 0.0;
  if (i1 > 350 - 1) return 0.0;
  return mix(float(fp4plus_u[i0]), float(fp4plus_u[i1]), f1) / float(0xfe);
}
```

18 include/Raymond-Fractal.frag

```
#donotrun

/*
Raymond - a physics-inspired ray tracer for Fragmentarium
Copyright (C) 2018  Claude Heiland-Allen
License GPL3+ <http://www.gnu.org/licenses/>
*/

#if 0

#define MengerSponge_(type) \
MengerSponge(type tag, Transform T, Ray V, int depth) \
{ \
  vec3 p = backwardP(T, V.origin); \
  vec3 q0 = vec3(0.0); \
  float r = 1.5; \
  for (int i = 0; i < depth; ++i) \
  { \
    vec3 q1 = vec3(0.0); \
    float d = 1.0 / 0.0; \
    for (int x = -1; x <= 1; ++x) \
    for (int y = -1; y <= 1; ++y) \
    for (int z = -1; z <= 1; ++z) \
    { \
      if (int(x == 0) + int(y == 0) + int(z == 0) > 1) continue; \
      vec3 q = q0 + vec3(x, y, z) / r; \
      float de = length(p - q); \
      if (de < d) { d = de; q1 = q; } \
    } \
    q0 = q1; \
    r *= 3.0; \
  } \
  return Cube(tag, compose(Translate(q0), T), V, 1.5 / r); \
}
float MengerSponge_(Scene_DE)


Surface MengerSponge_(Scene_HIT)
#undef MengerSponge_

#endif

// delta normal calculation
#define DELTANORMAL(DE,E,p) normalize(vec3 \
  ( DE(p + vec3(E,0.0,0.0)) - DE(p - vec3(E,0.0,0.0)) \
  , DE(p + vec3(0.0,E,0.0)) - DE(p - vec3(0.0,E,0.0)) \
  , DE(p + vec3(0.0,0.0,E)) - DE(p - vec3(0.0,0.0,E)) \
  ))
```

```
float MengerSponge(Scene_DE tag, Transform T, Ray V, int depth)
{
  const float Scale = 3.0;
  const vec3 Offset = vec3(1.0);
  vec3 z = backwardP(T, V.origin);
        int n = 0;
        while (n < depth) {
                z = abs(z);
                if (z.x<z.y){ z.xy = z.yx;}
                if (z.x< z.z){ z.xz = z.zx;}
                if (z.y<z.z){ z.yz = z.zy;}
                z = Scale*z-Offset*(Scale-1.0);
                if( z.z<-0.5*Offset.z*(Scale-1.0))  z.z+=Offset.z*(Scale-1.0);
                n++;
        }
  vec3 d = abs(z) - vec3(1.0);
  float de = length(max(d,0.0)) + min(max(d.x,max(d.y,d.z)),0.0);
  return forwardD(T, de * pow(Scale, float(-n)));
}

Surface MengerSponge(Scene_HIT tag, Transform T, Ray V, int depth)
{
  Scene_DE DE;
  float de = MengerSponge(DE, T, V, depth);
  const float epsilon = 1.0e-3; // FIXME
#define M(p) MengerSponge(DE, T, Ray((p), V.direction, V.wavelength, V.index), ↙
    ↳ depth)
  vec3 normal = DELTANORMAL(M, epsilon, V.origin);
#undef M
  // surface
  return Surface(V.origin, normal, de);
}

float Mandelbulb(Scene_DE tag, Transform T, Ray V, int Power, int depth)
{
  vec3 pos = backwardP(T, V.origin);
        vec3 z = pos;
        const float Bailout = 25.0;
        float dr=1.0;
        float r=length(z);
  int n = 0;
  while (n < depth && r < Bailout) {
// This is a power function taken from the implementation by Enforcer:
// http://www.fractalforums.com/mandelbulb-implementation/realtime-↙
    ↳ renderingoptimisations/
//
// I cannot follow its derivation from spherical coordinates,
// but it does give a nice mandelbrot like object for Power=2
//      z=abs(z);
//void powN2(inout vec3 z, float zr0, inout float dr, float Power) {
        float zr0 = r;
        float zo0 = asin( z.z/zr0 );
        float zi0 = atan( z.y,z.x );
        float zr = pow( zr0, Power-1.0 );
        float zo = zo0 * Power;
        float zi = zi0 * Power;
```

```
        // mermelada's tweak
        // http://www.fractalforums.com/new-theories-and-research/error-estimation-of-distance-estimators/msg102670/?topicseen#msg102670
        const float DerivativeBias = 1.0;
        dr = max(dr*DerivativeBias,zr*dr*Power + 1.0);
        zr *= zr0;
        z  = zr*vec3( cos(zo)*cos(zi), cos(zo)*sin(zi), sin(zo) );
//}
        z += pos;
        r = length(z);
                n++;
        }
        float de = 0.5*log(r)*r/dr;
  return forwardD(T, de);
}

Surface Mandelbulb(Scene_HIT tag, Transform T, Ray V, int power, int depth)
{
  Scene_DE DE;
  float de = Mandelbulb(DE, T, V, power, depth);
  const float epsilon = 1.0e-5; // FIXME
#define M(p) Mandelbulb(DE, T, Ray((p), V.direction, V.wavelength, V.index), power, depth)
  vec3 normal = DELTANORMAL(M, epsilon, V.origin);
#undef M
  // surface
  return Surface(V.origin, normal, de);
}

#undef DELTANORMAL
```

19 include/Raymond.frag

```
#donotrun
#info Raymond - a physics-inspired ray tracer (C) 2018 Claude Heiland-Allen (License GPL3+)

/*
Raymond - a physics-inspired ray tracer for Fragmentarium
Copyright (C) 2018  Claude Heiland-Allen
License GPL3+ <http://www.gnu.org/licenses/>
*/

const float pi = 3.141592653589793;
const vec3 X = vec3(1.0, 0.0, 0.0);
const vec3 Y = vec3(0.0, 1.0, 0.0);
const vec3 Z = vec3(0.0, 0.0, 1.0);

#include "Complex.frag"

#include "Raymond-Hash.frag"
#include "Raymond-Defs.frag"
#ifndef RaymondNoMain
#include "Raymond-Core.frag"
#endif
#include "Raymond-Random.frag"
#include "Raymond-Halton.frag"
```

```
#include "Raymond-Quaternion.frag"
#include "Raymond-Transform.frag"
#include "Raymond-Surface.frag"
#include "Raymond-Fractal.frag"
#include "Raymond-Material.frag"
#include "Raymond-D65.frag"
#include "Raymond-Glass.frag"
#include "Raymond-Quartz.frag"
#include "Raymond-Water.frag"
#include "Raymond-Observer.frag"
#include "Raymond-FP4Plus.frag"
#include "Raymond-Screen.frag"
#include "Raymond-SkyBox.frag"
#include "Raymond-Trace.frag"
#include "Raymond-Pinhole.frag"
```

20 include/Raymond-Glass.frag

```
#donotrun

/*
Raymond - a physics-inspired ray tracer for Fragmentarium
Copyright (C) 2018  Claude Heiland-Allen
License GPL3+ <http://www.gnu.org/licenses/>
*/

// database/glass/schott/F2.yml

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

float Glass_k(float wavelength) {
  const vec2 glass_lk[13] = vec2[13]
    ( vec2(1000.0 * 0.390, 2.8886e-8)
    , vec2(1000.0 * 0.400, 1.9243e-8)
    , vec2(1000.0 * 0.405, 1.6869e-8)
    , vec2(1000.0 * 0.420, 1.2087e-8)
    , vec2(1000.0 * 0.436, 9.7490e-9)
    , vec2(1000.0 * 0.460, 8.8118e-9)
    , vec2(1000.0 * 0.500, 4.7818e-9)
    , vec2(1000.0 * 0.546, 3.4794e-9)
    , vec2(1000.0 * 0.580, 3.6961e-9)
    , vec2(1000.0 * 0.620, 3.9510e-9)
    , vec2(1000.0 * 0.660, 6.3120e-9)
    , vec2(1000.0 * 0.700, 4.4608e-9)
    , vec2(1000.0 * 1.060, 6.7549e-9)
    );
  if (wavelength < glass_lk[0][0]) return glass_lk[0][1];
  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 k0 = glass_lk[i0][1];
        float k1 = glass_lk[i1][1];
        return mix(k0, k1, f1);
      }
    }
    return glass_lk[12][1];
}

vec2 Glass_nk(float wavelength)
{
  return vec2(Glass_n(wavelength), Glass_k(wavelength));
}

float Glass(Random PRNG, float S, Ray V)
{
  return S;
}

Hit Glass(Random PRNG, Surface S, Ray V)
{
  return Transparent(PRNG, S, V, Glass_nk(V.wavelength));
}
```

21 include/Raymond-Halton.frag

```
#donotrun

/*
Raymond - a physics-inspired ray tracer for Fragmentarium
Copyright (C) 2018  Claude Heiland-Allen
License GPL3+ <http://www.gnu.org/licenses/>
*/

// https://en.wikipedia.org/wiki/Halton_sequence

float halton(uint b, uint i)
{
  float f = 1.0;
  float r = 0.0;
  if (b > 1u)
    while (i > 0u)
    {
      f /= float(b);
      r += f * float(i % b);
      i /= b;
    }
  return r;
}

vec2 halton(uvec2 b, uint i)
{
  return vec2(halton(b.x, i), halton(b.y, i));
}
```

```
vec3 halton(uvec3 b, uint i)
{
  return vec3(halton(b.x, i), halton(b.y, i), halton(b.z, i));
}

vec4 halton(uvec4 b, uint i)
{
  return vec4(halton(b.x, i), halton(b.y, i), halton(b.z, i), halton(b.w, i));
}

float halton(Random PRNG, int b1)
{
  float h = halton(uint(b1), PRNG.index) + uniform1(PRNG);
  return h - floor(h);
}

vec2 halton(Random PRNG, int b1, int b2)
{
  float theta = 2.0 * pi * uniform1(srand(PRNG, 1));
  float c = cos(theta);
  float s = sin(theta);
  float r = 1.0 / (abs(c) + abs(s));
  mat2 m = mat2(c, s, -s, c) / r;
  vec2 h = m * halton(uvec2(b1, b2), PRNG.index) + uniform2(srand(PRNG, 2));
  return h - floor(h);
}

vec3 halton(Random PRNG, int b1, int b2, int b3)
{
  vec3 h = halton(uvec3(b1, b2, b3), PRNG.index) + uniform3(PRNG);
  return h - floor(h);
}

vec4 halton(Random PRNG, int b1, int b2, int b3, int b4)
{
  vec4 h = halton(uvec4(b1, b2, b3, b4), PRNG.index) + uniform4(PRNG);
  return h - floor(h);
}

float halton1(Random PRNG) { return halton(PRNG, 2); }
vec2 halton2(Random PRNG) { return halton(PRNG, 2, 3); }
vec3 halton3(Random PRNG) { return halton(PRNG, 2, 3, 5); }
vec4 halton4(Random PRNG) { return halton(PRNG, 2, 3, 5, 7); }

vec2 haltonDisc(Random PRNG)
{
  return uniformDisc(halton2(PRNG));
}
```

22 include/Raymond-Hash.frag

```
#donotrun

/*
Raymond - a physics-inspired ray tracer for Fragmentarium
Copyright (C) 2018  Claude Heiland-Allen
License GPL3+ <http://www.gnu.org/licenses/>
```

```
*/

// implement this

uint hash(uint a);

// scaling from uint to [0,1)

float uniform01(uint a) { return float(a) / 4294967296.0; }

// hashes of other/larger objects

uint hash(int a) { return hash(uint(a)); }
uint hash(float a) { return hash(floatBitsToUint(a)); }
uint hash(uvec2 a) { return hash(a.x ^ hash(a.y)); }
uint hash(uvec3 a) { return hash(a.x ^ hash(a.yz)); }
uint hash(uvec4 a) { return hash(a.x ^ hash(a.yzw)); }
uint hash(ivec2 a) { return hash(uint(a.x) ^ hash(a.y)); }
uint hash(ivec3 a) { return hash(uint(a.x) ^ hash(a.yz)); }
uint hash(ivec4 a) { return hash(uint(a.x) ^ hash(a.yzw)); }
uint hash(vec2 a) { return hash(floatBitsToUint(a.x) ^ hash(a.y)); }
uint hash(vec3 a) { return hash(floatBitsToUint(a.x) ^ hash(a.yz)); }
uint hash(vec4 a) { return hash(floatBitsToUint(a.x) ^ hash(a.yzw)); }

// http://www.burtleburtle.net/bob/hash/integer.html

uint hash_burtle_1(uint a)
{
  a = (a ^ 61u) ^ (a >> 16u);
  a = a + (a << 3u);
  a = a ^ (a >> 4u);
  a = a * 0x27d4eb2du;
  a = a ^ (a >> 15u);
  return a;
}

uint hash_burtle_2(uint a)
{
  a = (a+0x7ed55d16u) + (a<<12u);
  a = (a^0xc761c23cu) ^ (a>>19u);
  a = (a+0x165667b1u) + (a<<5u);
  a = (a+0xd3a2646cu) ^ (a<<9u);
  a = (a+0xfd7046c5u) + (a<<3u);
  a = (a^0xb55a4f09u) ^ (a>>16u);
  return a;
}

uint hash_burtle_3(uint a)
{
  a -= (a<<6u);
  a ^= (a>>17u);
  a -= (a<<9u);
  a ^= (a<<4u);
  a -= (a<<3u);
  a ^= (a<<10u);
  a ^= (a>>15u);
  return a;
```

```
}

uint hash_burtle_4(uint a)
{
  a += ~(a<<15u);
  a ^=  (a>>10u);
  a +=  (a<<3u);
  a ^=  (a>>6u);
  a += ~(a<<11u);
  a ^=  (a>>16u);
  return a;
}

uint hash_burtle_5(uint a)
{
  a = (a+0x479ab41du) + (a<<8u);
  a = (a^0xe4aa10ceu) ^ (a>>5u);
  a = (a+0x9942f0a6u) - (a<<14u);
  a = (a^0x5aedd67du) ^ (a>>3u);
  a = (a+0x17bea992u) + (a<<7u);
  return a;
}

uint hash_burtle_6(uint a)
{
  a = (a^0xdeadbeefu) + (a<<4u);
  a = a ^ (a>>10u);
  a = a + (a<<7u);
  a = a ^ (a>>13u);
  return a;
}

uint hash_burtle_7(uint a)
{
  a = a ^ (a>>4u);
  a = (a^0xdeadbeefu) + (a<<5u);
  a = a ^ (a>>11u);
  return a;
}

uint hash_burtle_8(uint a)
{
  a = (a+0x479ab41du) + (a<<8u);
  a = (a^0xe4aa10ceu) ^ (a>>5u);
  a = (a+0x9942f0a6u) - (a<<14u);
  a = (a^0x5aedd67du) ^ (a>>3u);
  a = (a+0x17bea992u) + (a<<7u);
  return a;
}

uint hash_burtle_9(uint a)
{
  a = (a+0x7ed55d16u) + (a<<12u);
  a = (a^0xc761c23cu) ^ (a>>19u);
  a = (a+0x165667b1u) + (a<<5u);
  a = (a+0xd3a2646cu) ^ (a<<9u);
  a = (a+0xfd7046c5u) + (a<<3u);
```

```
  a = (a^0xb55a4f09u) ^ (a>>16u);
  return a;
}

uint hash_burtle_10(uint a)
{
  a = (a+0x7fb9b1eeu) + (a<<12u);
  a = (a^0xab35dd63u) ^ (a>>19u);
  a = (a+0x41ed960du) + (a<<5u);
  a = (a+0xc7d0125eu) ^ (a<<9u);
  a = (a+0x071f9f8fu) + (a<<3u);
  a = (a^0x55ab55b9u) ^ (a>>16u);
  return a;
}

uint hash_burtle_11(uint a)
{
  a -= (a<<6u);
  a ^= (a>>17u);
  a -= (a<<9u);
  a ^= (a<<4u);
  a -= (a<<3u);
  a ^= (a<<10u);
  a ^= (a>>15u);
  return a;
}

uint hash_burtle_12(uint a)
{
  a += ~(a<<15u);
  a ^=  (a>>10u);
  a +=  (a<<3u);
  a ^=  (a>>6u);
  a += ~(a<<11u);
  a ^=  (a>>16u);
  return a;
}
```

23 include/Raymond-Lettuce.frag

```
#donotrun

/*
Raymond - a physics-inspired ray tracer for Fragmentarium
Copyright (C) 2018,2019  Claude Heiland-Allen
License GPL3+ <http://www.gnu.org/licenses/>

Lettuce: <https://fractalforums.org/f/28/t/2996/msg16077#msg16077>
*/

#define DELTA(N,E,p) vec3 \
  ( N(p + vec3(E,0.0,0.0)) - N(p - vec3(E,0.0,0.0)) \
  , N(p + vec3(0.0,E,0.0)) - N(p - vec3(0.0,E,0.0)) \
  , N(p + vec3(0.0,0.0,E)) - N(p - vec3(0.0,0.0,E)) \
  )

float Lettuce(vec3 p, int maxiter)
```

```
{
  float a = p.x;
  float b = p.y;
  float c = p.z;
  float len2 = dot(p, p);
  float rlen;
  int i = 0;
  while ((len2 < 16.0) && (i < maxiter)) {
    i++;
    rlen = 1.0/sqrt(len2);
    p *= rlen;  // normalize vector to unit length => project onto sphere
    // find X-related iso-plane: polar projection onto unit circle
    float Kx = 2.0*p.x*(1.0 - p.y)/((p.y - 2.0)*p.y + p.x*p.x + 1.0);
    float Ky = 1.0 - 2.0*((p.y - 2.0)*p.y + 1.0) /
      ((p.y - 2.0)*p.y + p.x*p.x + 1.0);
    // doubled point
    float K2x = -2.0*Kx*Ky;
    float K2y = -(Ky*Ky - Kx*Kx);
    // two more doublings (for total power eight)
    Kx = -2.0*K2x*K2y;
    Ky = -(K2y*K2y - K2x*K2x);
    K2x = -2.0*Kx*Ky;
    K2y = -(Ky*Ky - Kx*Kx);
    // (relevant) normal vector coordinates of doubled point plane
    float n1x = K2y - 1.0;
    float n1y = -K2x;
    // find Z-related iso-plane: polar projection onto unit circle
    float Kz = 2.0*p.z*(1.0 - p.y)/((p.y - 2.0)*p.y + p.z*p.z + 1.0);
    Ky = 1.0 - 2.0*((p.y - 2.0)*p.y + 1.0)/((p.y - 2.0)*p.y + p.z*p.z + 1.0);
    // doubled point
    float K2z = -2.0*Kz*Ky;
    K2y = -(Ky*Ky - Kz*Kz);
    // two more doublings (for total power eight)
    Kz = -2.0*K2z*K2y;
    Ky = -(K2y*K2y - K2z*K2z);
    K2z = -2.0*Kz*Ky;
    K2y = -(Ky*Ky - Kz*Kz);
    // (relevant) normal vector coordinates of doubled point plane
    float n2y = -K2z;
    float n2z = K2y - 1.0;
    // compute position of doubled point as intersection of planes and sphere
    // solved ray parameter
    float nt = 2.0*(n1x*n1x*n2z*n2z)/((n1x*n1x + n1y*n1y)*n2z*n2z
                  + n1x*n1x*n2y*n2y);
    // doubled point position
    p.y = 1.0 - nt;
    p.x = n1y*(1.0 - p.y)/n1x;
    p.z = n2y*(1.0 - p.y)/n2z;
    // raise original length to the power, then add constant
    // p *= len2;
    p *= len2*len2*len2*len2;  // for 8th power
    p += vec3(a,b,c);
    len2 = dot(p, p);
  }
  if (len2 < 16.0) {
    return float(maxiter);
  }
```

```
  return float(i) + 1.0 - log(sqrt(len2)) / log(8.0);
}

float Lettuce(Scene_DE tag, Transform T, Ray V, int maxiter)
{
  vec3 origin = backwardP(T, V.origin);
  const float epsilon = 1.0e-4; // FIXME
#define M(p) Lettuce(p, maxiter)
  vec3 gradient = DELTA(M, epsilon, origin) / (2.0 * epsilon);
#undef M
  float de = 0.25 * 1.0 / (log(8.0) * length(gradient));
  return forwardD(T, de);
}

Surface Lettuce(Scene_HIT tag, Transform T, Ray V, int maxiter)
{
  vec3 origin = backwardP(T, V.origin);
  const float epsilon = 1.0e-4; // FIXME
#define M(p) Lettuce(p, maxiter)
  vec3 gradient = DELTA(M, epsilon, origin) / (2.0 * epsilon);
#undef M
  float de = 0.25 * 1.0 / (log(8.0) * length(gradient));
  vec3 normal = normalize(-gradient); // FIXME normal of escape field, not of
      ↪ distance?
  // surface
  return Surface(forwardP(T, origin), forwardN(T, normal), forwardD(T, de));
}

#undef DELTA
```

24 include/Raymond-Material.frag

```
#donotrun

/*
Raymond - a physics-inspired ray tracer for Fragmentarium
Copyright (C) 2018  Claude Heiland-Allen
License GPL3+ <http://www.gnu.org/licenses/>
*/

// CSG union: minimum by de

Hit Union(in Hit a)
{
  return a;
}

Hit Union(in Hit a, in Hit b)
{
  if (a.surface.de < b.surface.de) return a; else return b;
}

Hit Union(in Hit a, in Hit b, in Hit c)
{
  return Union(Union(a, b), c);
}

```

```
Hit Union(in Hit a, in Hit b, in Hit c, in Hit d)
{
  return Union(Union(Union(a, b), c), d);
}

Hit Union(in Hit a, in Hit b, in Hit c, in Hit d, in Hit e)
{
  return Union(Union(Union(Union(a, b), c), d), e);
}

Hit Union(in Hit a, in Hit b, in Hit c, in Hit d, in Hit e, in Hit f)
{
  return Union(Union(Union(Union(Union(a, b), c), d), e), f);
}

Hit Union(in Hit a, in Hit b, in Hit c, in Hit d, in Hit e, in Hit f, in Hit g)
{
  return Union(Union(Union(Union(Union(Union(a, b), c), d), e), f), g);
}

// CSG intersection: maximum by de

Hit Intersection(in Hit a, in Hit b)
{
  if (a.surface.de > b.surface.de) return a; else return b;
}

Hit Intersection(in Hit a, in Hit b, in Hit c)
{
  return Intersection(Intersection(a, b), c);
}

Hit Intersection(in Hit a, in Hit b, in Hit c, in Hit d)
{
  return Intersection(Intersection(Intersection(a, b), c), d);
}

Hit Intersection(in Hit a, in Hit b, in Hit c, in Hit d, in Hit e)
{
  return Intersection(Intersection(Intersection(Intersection(a, b), c), d), e);
}

Hit Intersection(in Hit a, in Hit b, in Hit c, in Hit d, in Hit e, in Hit f)
{
  return Intersection(Intersection(Intersection(Intersection(Intersection(a, b),↙
      ↳ c), d), e), f);
}

// virtual surface (transmits everything)

float Transmit(float S, Ray V)
{
  return S;
}

Hit Transmit(Surface S, Ray V)
{
```

```
  V.origin = S.position;
  Hit h;
  h.surface = S;
  h.ray = V;
  h.factor = 1.0;
  h.emit = 0.0;
  return h;
}

// opaque emissive material

float Light(float S, Ray V, float brightness)
{
  return S;
}

Hit Light(Surface S, Ray V, float brightness)
{
  V.origin = S.position;
  V.direction = vec3(0.0);
  Hit h;
  h.surface = S;
  h.ray = V;
  h.factor = 0.0;
  h.emit = brightness;//S.de < 1.0e-4 ? brightness : 0.0;
  return h;
}

// perfectly reflective material

float Mirror(float S, Ray V, float albedo)
{
  return S;
}

Hit Mirror(Surface S, Ray V, float albedo)
{
  V.origin = S.position;
  V.direction = reflect(V.direction, S.normal);
  return Hit(S, V, albedo, 0.0);
}

// opaque diffuse material

float Diffuse(Random PRNG, float S, Ray V, float albedo)
{
  return S;
}

Hit Diffuse(Random PRNG, Surface S, Ray V, float albedo)
{
  V.origin = S.position;
  V.direction = cosHemisphere(haltonDisc(PRNG), S.normal);
  Hit h;
  h.surface = S;
  h.ray = V;
  h.emit = 0.0;
```

```
  h.factor = albedo / pi;
  return h;
}

// opaque glossy material

float CookTorrance(vec3 I, vec3 O, vec3 N, float albedo, float shine, float roughness, float eta)
{
  float diffuse = albedo / pi;
  vec3 H = normalize(O + I); // half vector
  float a2 = roughness * roughness;
  // distribution term (microfacets)
  float D = 0.0;
  float HN = dot(H, N);
  if (HN > 0.0)
  {
    float HN2 = HN * HN;
    float den = HN2 * a2 + (1.0 - HN2);
    D = a2 / (pi * den * den);
  }
  // fresnel term (approximation)
  float e = (eta - 1.0) / (eta + 1.0);
  float F = mix(pow(1.0 - dot(H, O), 5.0), 1.0, e * e);
  // geometry term (self-occulusion)
  float G = 1.0;
  float IH = dot(I, H);
  float OH = dot(O, H);
  if ((dot(I, N) > 0.0) == (IH > 0.0)) G *= 2.0 / (1.0 + sqrt(1.0 + a2 * (1.0 - IH * IH) / (IH * IH)));
  if ((dot(O, N) > 0.0) == (OH > 0.0)) G *= 2.0 / (1.0 + sqrt(1.0 + a2 * (1.0 - OH * OH) / (OH * OH)));
  // combined
  float specular = D * F * G / (4.0 * dot(I, N) * dot(O, N));
  // surface
  return mix(diffuse, specular, shine);
}

float CookTorrance(Random PRNG, float S, Ray V, float albedo, float shine, float roughness, vec2 index)
{
  return S;
}

Hit CookTorrance(Random PRNG, Surface S, Ray I, float albedo, float shine, float roughness, vec2 index)
{
  Ray O = I;
  O.origin = S.position;
  O.direction = cosHemisphere(haltonDisc(PRNG), S.normal);
  float eta = I.index.x / index.x;
  Hit h;
  h.surface = S;
  h.ray = O;
  h.emit = 0.0;
  h.factor = CookTorrance(I.direction, O.direction, S.normal, albedo, shine, roughness, eta);
```

```
190     return h;
      }

      // Fresnel equations

195   struct Fresnel { float Rs, Rp, Ts, Tp; };

      Fresnel fresnel(vec3 incident, vec3 normal, float n1, float n2)
      {
        float dotNI = dot(incident, normal);
200     float eta = n1 / n2;
        float c1 = -dotNI;
        float c2 = 1.0 - (1.0 - c1 * c1) * (eta * eta);
        Fresnel f;
        if (c2 < 0.0)
205     {
          // total internal reflection
          f.Rs = 1.0;
          f.Rp = 1.0;
          f.Ts = 0.0;
210       f.Tp = 0.0;
        }
        else
        {
          c2 = sqrt(c2);
215       float rs = (eta * c1 - c2) / (eta * c1 + c2);
          float rp = (eta * c2 - c1) / (eta * c2 + c1);
          float ts = 2.0 * eta * c1 / (eta * c1 + c2);
          float tp = 2.0 * eta * c1 / (eta * c2 + c1);
          f.Rs = rs * rs;
220       f.Rp = rp * rp;
          f.Ts = ts * ts;
          f.Tp = tp * tp;
        }
        return f;
225   }

      // refractive materials

      float Transparent(Random PRNG, float S, Ray V, vec2 index)
230   {
        return S;
      }

      Hit Transparent(Random PRNG, Surface S, Ray V, vec2 index)
235   {
        V.origin = S.position;
        vec2 nk1 = V.index;
        vec2 nk2 = index;
        bool inside = false;
240     if (dot(-V.direction, S.normal) < 0.0)
        {
          nk1 = nk2;
          nk2 = vec2(1.0, 0.0); // vacuum
          S = Invert(S);
245       inside = true;
        }
```

```
  Fresnel F = fresnel(V.direction, S.normal, nk1.x, nk2.x);
  // FIXME polarization
  float R = 0.5 * (F.Rs + F.Rp);
  float T = 0.5 * (F.Ts + F.Tp);
  float RT = R + T;
  float hRT = halton1(PRNG);
  if (mix(0.0, RT, hRT) <= R)
  {
    V.direction = normalize(reflect(V.direction, S.normal));
    V.index = nk1;
  }
  else
  {
    V.direction = normalize(refract(V.direction, S.normal, nk1.x / nk2.x));
    V.index = nk2;
  }
  if (inside)
  {
    S = Invert(S);
  }
  Hit h;
  h.surface = S;
  h.ray = V;
  h.factor = 1.0;
  h.emit = 0.0;
  return h;
}

// thin film soap bubble

float SoapBubble(Random PRNG, float S, Ray V, float thickness, vec2 index)
{
  return S;
}

Hit SoapBubble(Random PRNG, Surface S, Ray V, float thickness, vec2 index)
{
  V.origin = S.position;
  vec2 nk_air = V.index;
  vec2 nk_film = index;
  bool inside = false;
  if (dot(-V.direction, S.normal) < 0.0)
  {
    S = Invert(S);
    inside = true;
  }
  vec3 refracted = normalize(refract(V.direction, S.normal, nk_air.x / nk_film.x
    ↪ ));
  Fresnel F_in  = fresnel(V.direction, S.normal, nk_air.x, nk_film.x);
  Fresnel F_out = fresnel(refracted, S.normal, nk_film.x, nk_air.x);
  // FIXME polarization, absorption
  float T_in  = 0.5 * (F_in .Ts + F_in .Tp);
  float T_out = 0.5 * (F_out.Ts + F_out.Tp);
  float R_in  = 0.5 * (F_in .Rs + F_in .Rp);
  float R_film = 0.5 * (F_out.Rs + F_out.Rp);
  float d = 2.0 * pi * thickness / V.wavelength / abs(dot(refracted, S.normal));
  vec2 D = vec2(cos(d), sin(d));
```

```
    vec2 D2 = cSqr(D);
    vec2 one = vec2(1.0, 0.0);
    float T = length(cDiv(T_in * T_out * D, one - R_film * R_film * D2));
    float R = length(cDiv(T_in * T_out * R_film * D2, one - R_film * R_film * D2) ↙
        ↳ - vec2(R_in, 0.0));
    float hRT = halton1(PRNG) * (R + T);
    if (hRT < R)
    {
      V.direction = normalize(reflect(V.direction, S.normal));
    }
    // otherwise ray continues in same direction
    if (inside)
    {
      S = Invert(S);
    }
    Hit h;
    h.surface = S;
    h.ray = V;
    h.factor = 1.0;
    h.emit = 0.0;
    return h;
  }

  // reflective thin film

  float ThinFilm(float S, Ray V, float thickness, float factor)
  {
    return S;
  }

  Hit ThinFilm(Surface S, Ray V, float thickness, float factor)
  {
    float t = 4.0 * pi * (-dot(V.direction, S.normal)) * thickness / V.wavelength;
    float f = 0.5 * length(vec2(cos(t) + 1.0, sin(t)));
    V.direction = reflect(V.direction, S.normal);
    Hit h;
    h.surface = S;
    h.ray = V;
    h.factor = factor * f;
    h.emit = 0.0;
    return h;
  }

  float Checkerboard(vec3 x, float a, float b)
  {
    x -= floor(x);
    bool c = (x.x > 0.5) != (x.y > 0.5) != (x.z > 0.5);
    return c ? a : b;
  }

  Hit Checkerboard(vec3 x, Hit a, Hit b)
  {
    x -= floor(x);
    bool c = (x.x > 0.5) != (x.y > 0.5) != (x.z > 0.5);
    return c ? a : b;
  }
```

```
float absorption(float k, float nm, float dist)
{
  return exp(-4.0 * pi * k * dist / (1.0e-9 * nm));
}
```

25 include/Raymond-Observer.frag

```
#donotrun

/*
Raymond - a physics-inspired ray tracer for Fragmentarium
Copyright (C) 2018  Claude Heiland-Allen
License GPL3+ <http://www.gnu.org/licenses/>
*/

// linear XYZ to linear RGB
// https://en.wikipedia.org/wiki/SRGB#The_forward_transformation_(↙
    ↳ CIE_XYZ_to_sRGB)
vec3 xyz2rgb(in vec3 l)
{
  const vec3 whitePoint = vec3(0.9505, 1.0000, 1.0890);
  const mat3 matrix = mat3
    (  3.2406, -1.5372, -0.4986
    , -0.9689,  1.8758,  0.0415
    ,  0.0557, -0.2040,  1.0570
    );
  return (whitePoint * l) * matrix;
}

// linear RGB to linear XYZ
// https://en.wikipedia.org/wiki/SRGB#The_reverse_transformation
vec3 rgb2xyz(in vec3 l)
{
  const vec3 whitePoint = vec3(0.9505, 1.0000, 1.0890);
  const mat3 matrix = mat3
    ( 0.4124, 0.3576, 0.1805
    , 0.2126, 0.7152, 0.0722
    , 0.0193, 0.1192, 0.9505
    );
  return (l * matrix) / whitePoint;
}

// CIE standard observer
// https://en.wikipedia.org/wiki/CIE_1931_color_space#CIE_standard_observer
vec3 observer(in float wavelength)
{
const int wavelength_min = 390;
const int wavelength_max = 830;
#define wavelength_count 441
const vec3 tristimulus_curve[wavelength_count] = vec3[wavelength_count]
  ( vec3(3.769647e-3, 4.146161e-4, 1.847260e-2)
  , vec3(4.532416e-3, 5.028333e-4, 2.221101e-2)
  , vec3(5.446553e-3, 6.084991e-4, 2.669819e-2)
  , vec3(6.538868e-3, 7.344436e-4, 3.206937e-2)
  , vec3(7.839699e-3, 8.837389e-4, 3.847832e-2)
  , vec3(9.382967e-3, 1.059646e-3, 4.609784e-2)
  , vec3(1.120608e-2, 1.265532e-3, 5.511953e-2)
```

```
50      , vec3(1.334965e-2, 1.504753e-3, 6.575257e-2)
        , vec3(1.585690e-2, 1.780493e-3, 7.822113e-2)
        , vec3(1.877286e-2, 2.095572e-3, 9.276013e-2)
        , vec3(2.214302e-2, 2.452194e-3, 1.096090e-1)
        , vec3(2.601285e-2, 2.852216e-3, 1.290077e-1)
55      , vec3(3.043036e-2, 3.299115e-3, 1.512047e-1)
        , vec3(3.544325e-2, 3.797466e-3, 1.764441e-1)
        , vec3(4.109640e-2, 4.352768e-3, 2.049517e-1)
        , vec3(4.742986e-2, 4.971717e-3, 2.369246e-1)
        , vec3(5.447394e-2, 5.661014e-3, 2.725123e-1)
60      , vec3(6.223612e-2, 6.421615e-3, 3.117820e-1)
        , vec3(7.070048e-2, 7.250312e-3, 3.547064e-1)
        , vec3(7.982513e-2, 8.140173e-3, 4.011473e-1)
        , vec3(8.953803e-2, 9.079860e-3, 4.508369e-1)
        , vec3(9.974848e-2, 1.005608e-2, 5.034164e-1)
65      , vec3(1.104019e-1, 1.106456e-2, 5.586361e-1)
        , vec3(1.214566e-1, 1.210522e-2, 6.162734e-1)
        , vec3(1.328741e-1, 1.318014e-2, 6.760982e-1)
        , vec3(1.446214e-1, 1.429377e-2, 7.378822e-1)
        , vec3(1.566468e-1, 1.545004e-2, 8.013019e-1)
70      , vec3(1.687901e-1, 1.664093e-2, 8.655573e-1)
        , vec3(1.808328e-1, 1.785302e-2, 9.295791e-1)
        , vec3(1.925216e-1, 1.907018e-2, 9.921293e-1)
        , vec3(2.035729e-1, 2.027369e-2, 1.051821)
        , vec3(2.137531e-1, 2.144805e-2, 1.107509)
75      , vec3(2.231348e-1, 2.260041e-2, 1.159527)
        , vec3(2.319245e-1, 2.374789e-2, 1.208869)
        , vec3(2.403892e-1, 2.491247e-2, 1.256834)
        , vec3(2.488523e-1, 2.612106e-2, 1.305008)
        , vec3(2.575896e-1, 2.739923e-2, 1.354758)
80      , vec3(2.664991e-1, 2.874993e-2, 1.405594)
        , vec3(2.753532e-1, 3.016909e-2, 1.456414)
        , vec3(2.838921e-1, 3.165145e-2, 1.505960)
        , vec3(2.918246e-1, 3.319038e-2, 1.552826)
        , vec3(2.989200e-1, 3.477912e-2, 1.595902)
85      , vec3(3.052993e-1, 3.641495e-2, 1.635768)
        , vec3(3.112031e-1, 3.809569e-2, 1.673573)
        , vec3(3.169047e-1, 3.981843e-2, 1.710604)
        , vec3(3.227087e-1, 4.157940e-2, 1.748280)
        , vec3(3.288194e-1, 4.337098e-2, 1.787504)
90      , vec3(3.349242e-1, 4.517180e-2, 1.826609)
        , vec3(3.405452e-1, 4.695420e-2, 1.863108)
        , vec3(3.451688e-1, 4.868718e-2, 1.894332)
        , vec3(3.482554e-1, 5.033657e-2, 1.917479)
        , vec3(3.494153e-1, 5.187611e-2, 1.930529)
95      , vec3(3.489075e-1, 5.332218e-2, 1.934819)
        , vec3(3.471746e-1, 5.470603e-2, 1.932650)
        , vec3(3.446705e-1, 5.606335e-2, 1.926395)
        , vec3(3.418483e-1, 5.743393e-2, 1.918437)
        , vec3(3.390240e-1, 5.885107e-2, 1.910430)
100     , vec3(3.359926e-1, 6.030809e-2, 1.901224)
        , vec3(3.324276e-1, 6.178644e-2, 1.889000)
        , vec3(3.280157e-1, 6.326570e-2, 1.871996)
        , vec3(3.224637e-1, 6.472352e-2, 1.848545)
        , vec3(3.156225e-1, 6.614749e-2, 1.817792)
105     , vec3(3.078201e-1, 6.757256e-2, 1.781627)
        , vec3(2.994771e-1, 6.904928e-2, 1.742514)
```

```
    , vec3(2.909776e-1, 7.063280e-2, 1.702749)
    , vec3(2.826646e-1, 7.238339e-2, 1.664439)
    , vec3(2.747962e-1, 7.435960e-2, 1.629207)
    , vec3(2.674312e-1, 7.659383e-2, 1.597360)
    , vec3(2.605847e-1, 7.911436e-2, 1.568896)
    , vec3(2.542749e-1, 8.195345e-2, 1.543823)
    , vec3(2.485254e-1, 8.514816e-2, 1.522157)
    , vec3(2.433039e-1, 8.872657e-2, 1.503611)
    , vec3(2.383414e-1, 9.266008e-2, 1.486673)
    , vec3(2.333253e-1, 9.689723e-2, 1.469595)
    , vec3(2.279619e-1, 1.013746e-1, 1.450709)
    , vec3(2.219781e-1, 1.060145e-1, 1.428440)
    , vec3(2.151735e-1, 1.107377e-1, 1.401587)
    , vec3(2.075619e-1, 1.155111e-1, 1.370094)
    , vec3(1.992183e-1, 1.203122e-1, 1.334220)
    , vec3(1.902290e-1, 1.251161e-1, 1.294275)
    , vec3(1.806905e-1, 1.298957e-1, 1.250610)
    , vec3(1.707154e-1, 1.346299e-1, 1.203696)
    , vec3(1.604471e-1, 1.393309e-1, 1.154316)
    , vec3(1.500244e-1, 1.440235e-1, 1.103284)
    , vec3(1.395705e-1, 1.487372e-1, 1.051347)
    , vec3(1.291920e-1, 1.535066e-1, 9.991789e-1)
    , vec3(1.189859e-1, 1.583644e-1, 9.473958e-1)
    , vec3(1.090615e-1, 1.633199e-1, 8.966222e-1)
    , vec3(9.951424e-2, 1.683761e-1, 8.473981e-1)
    , vec3(9.041850e-2, 1.735365e-1, 8.001576e-1)
    , vec3(8.182895e-2, 1.788048e-1, 7.552379e-1)
    , vec3(7.376817e-2, 1.841819e-1, 7.127879e-1)
    , vec3(6.619477e-2, 1.896559e-1, 6.725198e-1)
    , vec3(5.906380e-2, 1.952101e-1, 6.340976e-1)
    , vec3(5.234242e-2, 2.008259e-1, 5.972433e-1)
    , vec3(4.600865e-2, 2.064828e-1, 5.617313e-1)
    , vec3(4.006154e-2, 2.121826e-1, 5.274921e-1)
    , vec3(3.454373e-2, 2.180279e-1, 4.948809e-1)
    , vec3(2.949091e-2, 2.241586e-1, 4.642586e-1)
    , vec3(2.492140e-2, 2.307302e-1, 4.358841e-1)
    , vec3(2.083981e-2, 2.379160e-1, 4.099313e-1)
    , vec3(1.723591e-2, 2.458706e-1, 3.864261e-1)
    , vec3(1.407924e-2, 2.546023e-1, 3.650566e-1)
    , vec3(1.134516e-2, 2.640760e-1, 3.454812e-1)
    , vec3(9.019658e-3, 2.742490e-1, 3.274095e-1)
    , vec3(7.097731e-3, 2.850680e-1, 3.105939e-1)
    , vec3(5.571145e-3, 2.964837e-1, 2.948102e-1)
    , vec3(4.394566e-3, 3.085010e-1, 2.798194e-1)
    , vec3(3.516303e-3, 3.211393e-1, 2.654100e-1)
    , vec3(2.887638e-3, 3.344175e-1, 2.514084e-1)
    , vec3(2.461588e-3, 3.483536e-1, 2.376753e-1)
    , vec3(2.206348e-3, 3.629601e-1, 2.241211e-1)
    , vec3(2.149559e-3, 3.782275e-1, 2.107484e-1)
    , vec3(2.337091e-3, 3.941359e-1, 1.975839e-1)
    , vec3(2.818931e-3, 4.106582e-1, 1.846574e-1)
    , vec3(3.649178e-3, 4.277595e-1, 1.720018e-1)
    , vec3(4.891359e-3, 4.453993e-1, 1.596918e-1)
    , vec3(6.629364e-3, 4.635396e-1, 1.479415e-1)
    , vec3(8.942902e-3, 4.821376e-1, 1.369428e-1)
    , vec3(1.190224e-2, 5.011430e-1, 1.268279e-1)
    , vec3(1.556989e-2, 5.204972e-1, 1.176796e-1)
```

```
        , vec3(1.997668e-2, 5.401387e-1, 1.094970e-1)
165     , vec3(2.504698e-2, 5.600208e-1, 1.020943e-1)
        , vec3(3.067530e-2, 5.800972e-1, 9.527993e-2)
        , vec3(3.674999e-2, 6.003172e-1, 8.890075e-2)
        , vec3(4.315171e-2, 6.206256e-1, 8.283548e-2)
        , vec3(4.978584e-2, 6.409398e-1, 7.700982e-2)
170     , vec3(5.668554e-2, 6.610772e-1, 7.144001e-2)
        , vec3(6.391651e-2, 6.808134e-1, 6.615436e-2)
        , vec3(7.154352e-2, 6.999044e-1, 6.117199e-2)
        , vec3(7.962917e-2, 7.180890e-1, 5.650407e-2)
        , vec3(8.821473e-2, 7.351593e-1, 5.215121e-2)
175     , vec3(9.726978e-2, 7.511821e-1, 4.809566e-2)
        , vec3(1.067504e-1, 7.663143e-1, 4.431720e-2)
        , vec3(1.166192e-1, 7.807352e-1, 4.079734e-2)
        , vec3(1.268468e-1, 7.946448e-1, 3.751912e-2)
        , vec3(1.374060e-1, 8.082074e-1, 3.446846e-2)
180     , vec3(1.482471e-1, 8.213817e-1, 3.163764e-2)
        , vec3(1.593076e-1, 8.340701e-1, 2.901901e-2)
        , vec3(1.705181e-1, 8.461711e-1, 2.660364e-2)
        , vec3(1.818026e-1, 8.575799e-1, 2.438164e-2)
        , vec3(1.931090e-1, 8.682408e-1, 2.234097e-2)
185     , vec3(2.045085e-1, 8.783061e-1, 2.046415e-2)
        , vec3(2.161166e-1, 8.879907e-1, 1.873456e-2)
        , vec3(2.280650e-1, 8.975211e-1, 1.713788e-2)
        , vec3(2.405015e-1, 9.071347e-1, 1.566174e-2)
        , vec3(2.535441e-1, 9.169947e-1, 1.429644e-2)
190     , vec3(2.671300e-1, 9.269295e-1, 1.303702e-2)
        , vec3(2.811351e-1, 9.366731e-1, 1.187897e-2)
        , vec3(2.954164e-1, 9.459482e-1, 1.081725e-2)
        , vec3(3.098117e-1, 9.544675e-1, 9.846470e-3)
        , vec3(3.241678e-1, 9.619834e-1, 8.960687e-3)
195     , vec3(3.384319e-1, 9.684390e-1, 8.152811e-3)
        , vec3(3.525786e-1, 9.738289e-1, 7.416025e-3)
        , vec3(3.665839e-1, 9.781519e-1, 6.744115e-3)
        , vec3(3.804244e-1, 9.814106e-1, 6.131421e-3)
        , vec3(3.940988e-1, 9.836669e-1, 5.572778e-3)
200     , vec3(4.076972e-1, 9.852081e-1, 5.063463e-3)
        , vec3(4.213484e-1, 9.863813e-1, 4.599169e-3)
        , vec3(4.352003e-1, 9.875357e-1, 4.175971e-3)
        , vec3(4.494206e-1, 9.890228e-1, 3.790291e-3)
        , vec3(4.641616e-1, 9.910811e-1, 3.438952e-3)
205     , vec3(4.794395e-1, 9.934913e-1, 3.119341e-3)
        , vec3(4.952180e-1, 9.959172e-1, 2.829038e-3)
        , vec3(5.114395e-1, 9.980205e-1, 2.565722e-3)
        , vec3(5.280233e-1, 9.994608e-1, 2.327186e-3)
        , vec3(5.448696e-1, 9.999930e-1, 2.111280e-3)
210     , vec3(5.618898e-1, 9.997557e-1, 1.915766e-3)
        , vec3(5.790137e-1, 9.989839e-1, 1.738589e-3)
        , vec3(5.961882e-1, 9.979123e-1, 1.577920e-3)
        , vec3(6.133784e-1, 9.967737e-1, 1.432128e-3)
        , vec3(6.305897e-1, 9.957356e-1, 1.299781e-3)
215     , vec3(6.479223e-1, 9.947115e-1, 1.179667e-3)
        , vec3(6.654866e-1, 9.935534e-1, 1.070694e-3)
        , vec3(6.833782e-1, 9.921156e-1, 9.718623e-4)
        , vec3(7.016774e-1, 9.902549e-1, 8.822531e-4)
        , vec3(7.204110e-1, 9.878596e-1, 8.010231e-4)
220     , vec3(7.394495e-1, 9.849324e-1, 7.273884e-4)
```

```
    , vec3(7.586285e-1, 9.815036e-1, 6.606347e-4)
    , vec3(7.777885e-1, 9.776035e-1, 6.001146e-4)
    , vec3(7.967750e-1, 9.732611e-1, 5.452416e-4)
    , vec3(8.154530e-1, 9.684764e-1, 4.954847e-4)
    , vec3(8.337389e-1, 9.631369e-1, 4.503642e-4)
    , vec3(8.515493e-1, 9.571062e-1, 4.094455e-4)
    , vec3(8.687862e-1, 9.502540e-1, 3.723345e-4)
    , vec3(8.853376e-1, 9.424569e-1, 3.386739e-4)
    , vec3(9.011588e-1, 9.336897e-1, 3.081396e-4)
    , vec3(9.165278e-1, 9.242893e-1, 2.804370e-4)
    , vec3(9.318245e-1, 9.146707e-1, 2.552996e-4)
    , vec3(9.474524e-1, 9.052333e-1, 2.324859e-4)
    , vec3(9.638388e-1, 8.963613e-1, 2.117772e-4)
    , vec3(9.812596e-1, 8.883069e-1, 1.929758e-4)
    , vec3(9.992953e-1, 8.808462e-1, 1.759024e-4)
    , vec3(1.017343, 8.736445e-1, 1.603947e-4)
    , vec3(1.034790, 8.663755e-1, 1.463059e-4)
    , vec3(1.051011, 8.587203e-1, 1.335031e-4)
    , vec3(1.065522, 8.504295e-1, 1.218660e-4)
    , vec3(1.078421, 8.415047e-1, 1.112857e-4)
    , vec3(1.089944, 8.320109e-1, 1.016634e-4)
    , vec3(1.100320, 8.220154e-1, 9.291003e-5)
    , vec3(1.109767, 8.115868e-1, 8.494468e-5)
    , vec3(1.118438, 8.007874e-1, 7.769425e-5)
    , vec3(1.126266, 7.896515e-1, 7.109247e-5)
    , vec3(1.133138, 7.782053e-1, 6.507936e-5)
    , vec3(1.138952, 7.664733e-1, 5.960061e-5)
    , vec3(1.143620, 7.544785e-1, 5.460706e-5)
    , vec3(1.147095, 7.422473e-1, 5.005417e-5)
    , vec3(1.149464, 7.298229e-1, 4.590157e-5)
    , vec3(1.150838, 7.172525e-1, 4.211268e-5)
    , vec3(1.151326, 7.045818e-1, 3.865437e-5)
    , vec3(1.151033, 6.918553e-1, 3.549661e-5)
    , vec3(1.150002, 6.791009e-1, 3.261220e-5)
    , vec3(1.148061, 6.662846e-1, 2.997643e-5)
    , vec3(1.144998, 6.533595e-1, 2.756693e-5)
    , vec3(1.140622, 6.402807e-1, 2.536339e-5)
    , vec3(1.134757, 6.270066e-1, 2.334738e-5)
    , vec3(1.127298, 6.135148e-1, 2.150221e-5)
    , vec3(1.118342, 5.998494e-1, 1.981268e-5)
    , vec3(1.108033, 5.860682e-1, 1.826500e-5)
    , vec3(1.096515, 5.722261e-1, 1.684667e-5)
    , vec3(1.083928, 5.583746e-1, 1.554631e-5)
    , vec3(1.070387, 5.445535e-1, 1.435360e-5)
    , vec3(1.055934, 5.307673e-1, 1.325915e-5)
    , vec3(1.040592, 5.170130e-1, 1.225443e-5)
    , vec3(1.024385, 5.032889e-1, 1.133169e-5)
    , vec3(1.007344, 4.895950e-1, 1.048387e-5)
    , vec3(9.895268e-1, 4.759442e-1, 0.000000)
    , vec3(9.711213e-1, 4.623958e-1, 0.000000)
    , vec3(9.523257e-1, 4.490154e-1, 0.000000)
    , vec3(9.333248e-1, 4.358622e-1, 0.000000)
    , vec3(9.142877e-1, 4.229897e-1, 0.000000)
    , vec3(8.952798e-1, 4.104152e-1, 0.000000)
    , vec3(8.760157e-1, 3.980356e-1, 0.000000)
    , vec3(8.561607e-1, 3.857300e-1, 0.000000)
    , vec3(8.354235e-1, 3.733907e-1, 0.000000)
```

```
    , vec3(8.135565e-1, 3.609245e-1, 0.000000)
    , vec3(7.904565e-1, 3.482860e-1, 0.000000)
    , vec3(7.664364e-1, 3.355702e-1, 0.000000)
    , vec3(7.418777e-1, 3.228963e-1, 0.000000)
    , vec3(7.171219e-1, 3.103704e-1, 0.000000)
    , vec3(6.924717e-1, 2.980865e-1, 0.000000)
    , vec3(6.681600e-1, 2.861160e-1, 0.000000)
    , vec3(6.442697e-1, 2.744822e-1, 0.000000)
    , vec3(6.208450e-1, 2.631953e-1, 0.000000)
    , vec3(5.979243e-1, 2.522628e-1, 0.000000)
    , vec3(5.755410e-1, 2.416902e-1, 0.000000)
    , vec3(5.537296e-1, 2.314809e-1, 0.000000)
    , vec3(5.325412e-1, 2.216378e-1, 0.000000)
    , vec3(5.120218e-1, 2.121622e-1, 0.000000)
    , vec3(4.922070e-1, 2.030542e-1, 0.000000)
    , vec3(4.731224e-1, 1.943124e-1, 0.000000)
    , vec3(4.547417e-1, 1.859227e-1, 0.000000)
    , vec3(4.368719e-1, 1.778274e-1, 0.000000)
    , vec3(4.193121e-1, 1.699654e-1, 0.000000)
    , vec3(4.018980e-1, 1.622841e-1, 0.000000)
    , vec3(3.844986e-1, 1.547397e-1, 0.000000)
    , vec3(3.670592e-1, 1.473081e-1, 0.000000)
    , vec3(3.497167e-1, 1.400169e-1, 0.000000)
    , vec3(3.326305e-1, 1.329013e-1, 0.000000)
    , vec3(3.159341e-1, 1.259913e-1, 0.000000)
    , vec3(2.997374e-1, 1.193120e-1, 0.000000)
    , vec3(2.841189e-1, 1.128820e-1, 0.000000)
    , vec3(2.691053e-1, 1.067113e-1, 0.000000)
    , vec3(2.547077e-1, 1.008052e-1, 0.000000)
    , vec3(2.409319e-1, 9.516653e-2, 0.000000)
    , vec3(2.277792e-1, 8.979594e-2, 0.000000)
    , vec3(2.152431e-1, 8.469044e-2, 0.000000)
    , vec3(2.033010e-1, 7.984009e-2, 0.000000)
    , vec3(1.919276e-1, 7.523372e-2, 0.000000)
    , vec3(1.810987e-1, 7.086061e-2, 0.000000)
    , vec3(1.707914e-1, 6.671045e-2, 0.000000)
    , vec3(1.609842e-1, 6.277360e-2, 0.000000)
    , vec3(1.516577e-1, 5.904179e-2, 0.000000)
    , vec3(1.427936e-1, 5.550703e-2, 0.000000)
    , vec3(1.343737e-1, 5.216139e-2, 0.000000)
    , vec3(1.263808e-1, 4.899699e-2, 0.000000)
    , vec3(1.187979e-1, 4.600578e-2, 0.000000)
    , vec3(1.116088e-1, 4.317885e-2, 0.000000)
    , vec3(1.047975e-1, 4.050755e-2, 0.000000)
    , vec3(9.834835e-2, 3.798376e-2, 0.000000)
    , vec3(9.224597e-2, 3.559982e-2, 0.000000)
    , vec3(8.647506e-2, 3.334856e-2, 0.000000)
    , vec3(8.101986e-2, 3.122332e-2, 0.000000)
    , vec3(7.586514e-2, 2.921780e-2, 0.000000)
    , vec3(7.099633e-2, 2.732601e-2, 0.000000)
    , vec3(6.639960e-2, 2.554223e-2, 0.000000)
    , vec3(6.206225e-2, 2.386121e-2, 0.000000)
    , vec3(5.797409e-2, 2.227859e-2, 0.000000)
    , vec3(5.412533e-2, 2.079020e-2, 0.000000)
    , vec3(5.050600e-2, 1.939185e-2, 0.000000)
    , vec3(4.710606e-2, 1.807939e-2, 0.000000)
    , vec3(4.391411e-2, 1.684817e-2, 0.000000)
```

```
335     , vec3(4.091411e-2, 1.569188e-2, 0.000000)
        , vec3(3.809067e-2, 1.460446e-2, 0.000000)
        , vec3(3.543034e-2, 1.358062e-2, 0.000000)
        , vec3(3.292138e-2, 1.261573e-2, 0.000000)
        , vec3(3.055672e-2, 1.170696e-2, 0.000000)
340     , vec3(2.834146e-2, 1.085608e-2, 0.000000)
        , vec3(2.628033e-2, 1.006476e-2, 0.000000)
        , vec3(2.437465e-2, 9.333376e-3, 0.000000)
        , vec3(2.262306e-2, 8.661284e-3, 0.000000)
        , vec3(2.101935e-2, 8.046048e-3, 0.000000)
345     , vec3(1.954647e-2, 7.481130e-3, 0.000000)
        , vec3(1.818727e-2, 6.959987e-3, 0.000000)
        , vec3(1.692727e-2, 6.477070e-3, 0.000000)
        , vec3(1.575417e-2, 6.027677e-3, 0.000000)
        , vec3(1.465854e-2, 5.608169e-3, 0.000000)
350     , vec3(1.363571e-2, 5.216691e-3, 0.000000)
        , vec3(1.268205e-2, 4.851785e-3, 0.000000)
        , vec3(1.179394e-2, 4.512008e-3, 0.000000)
        , vec3(1.096778e-2, 4.195941e-3, 0.000000)
        , vec3(1.019964e-2, 3.902057e-3, 0.000000)
355     , vec3(9.484317e-3, 3.628371e-3, 0.000000)
        , vec3(8.816851e-3, 3.373005e-3, 0.000000)
        , vec3(8.192921e-3, 3.134315e-3, 0.000000)
        , vec3(7.608750e-3, 2.910864e-3, 0.000000)
        , vec3(7.061391e-3, 2.701528e-3, 0.000000)
360     , vec3(6.549509e-3, 2.505796e-3, 0.000000)
        , vec3(6.071970e-3, 2.323231e-3, 0.000000)
        , vec3(5.627476e-3, 2.153333e-3, 0.000000)
        , vec3(5.214608e-3, 1.995557e-3, 0.000000)
        , vec3(4.831848e-3, 1.849316e-3, 0.000000)
365     , vec3(4.477579e-3, 1.713976e-3, 0.000000)
        , vec3(4.150166e-3, 1.588899e-3, 0.000000)
        , vec3(3.847988e-3, 1.473453e-3, 0.000000)
        , vec3(3.569452e-3, 1.367022e-3, 0.000000)
        , vec3(3.312857e-3, 1.268954e-3, 0.000000)
370     , vec3(3.076022e-3, 1.178421e-3, 0.000000)
        , vec3(2.856894e-3, 1.094644e-3, 0.000000)
        , vec3(2.653681e-3, 1.016943e-3, 0.000000)
        , vec3(2.464821e-3, 9.447269e-4, 0.000000)
        , vec3(2.289060e-3, 8.775171e-4, 0.000000)
375     , vec3(2.125694e-3, 8.150438e-4, 0.000000)
        , vec3(1.974121e-3, 7.570755e-4, 0.000000)
        , vec3(1.833723e-3, 7.033755e-4, 0.000000)
        , vec3(1.703876e-3, 6.537050e-4, 0.000000)
        , vec3(1.583904e-3, 6.078048e-4, 0.000000)
380     , vec3(1.472939e-3, 5.653435e-4, 0.000000)
        , vec3(1.370151e-3, 5.260046e-4, 0.000000)
        , vec3(1.274803e-3, 4.895061e-4, 0.000000)
        , vec3(1.186238e-3, 4.555970e-4, 0.000000)
        , vec3(1.103871e-3, 4.240548e-4, 0.000000)
385     , vec3(1.027194e-3, 3.946860e-4, 0.000000)
        , vec3(9.557493e-4, 3.673178e-4, 0.000000)
        , vec3(8.891262e-4, 3.417941e-4, 0.000000)
        , vec3(8.269535e-4, 3.179738e-4, 0.000000)
        , vec3(7.689351e-4, 2.957441e-4, 0.000000)
390     , vec3(7.149425e-4, 2.750558e-4, 0.000000)
        , vec3(6.648590e-4, 2.558640e-4, 0.000000)
```

```
    , vec3(6.185421e-4, 2.381142e-4, 0.000000)
    , vec3(5.758303e-4, 2.217445e-4, 0.000000)
    , vec3(5.365046e-4, 2.066711e-4, 0.000000)
395 , vec3(5.001842e-4, 1.927474e-4, 0.000000)
    , vec3(4.665005e-4, 1.798315e-4, 0.000000)
    , vec3(4.351386e-4, 1.678023e-4, 0.000000)
    , vec3(4.058303e-4, 1.565566e-4, 0.000000)
    , vec3(3.783733e-4, 1.460168e-4, 0.000000)
400 , vec3(3.526892e-4, 1.361535e-4, 0.000000)
    , vec3(3.287199e-4, 1.269451e-4, 0.000000)
    , vec3(3.063998e-4, 1.183671e-4, 0.000000)
    , vec3(2.856577e-4, 1.103928e-4, 0.000000)
    , vec3(2.664108e-4, 1.029908e-4, 0.000000)
405 , vec3(2.485462e-4, 9.611836e-5, 0.000000)
    , vec3(2.319529e-4, 8.973323e-5, 0.000000)
    , vec3(2.165300e-4, 8.379694e-5, 0.000000)
    , vec3(2.021853e-4, 7.827442e-5, 0.000000)
    , vec3(1.888338e-4, 7.313312e-5, 0.000000)
410 , vec3(1.763935e-4, 6.834142e-5, 0.000000)
    , vec3(1.647895e-4, 6.387035e-5, 0.000000)
    , vec3(1.539542e-4, 5.969389e-5, 0.000000)
    , vec3(1.438270e-4, 5.578862e-5, 0.000000)
    , vec3(1.343572e-4, 5.213509e-5, 0.000000)
415 , vec3(1.255141e-4, 4.872179e-5, 0.000000)
    , vec3(1.172706e-4, 4.553845e-5, 0.000000)
    , vec3(1.095983e-4, 4.257443e-5, 0.000000)
    , vec3(1.024685e-4, 3.981884e-5, 0.000000)
    , vec3(9.584715e-5, 3.725877e-5, 0.000000)
420 , vec3(8.968316e-5, 3.487467e-5, 0.000000)
    , vec3(8.392734e-5, 3.264765e-5, 0.000000)
    , vec3(7.853708e-5, 3.056140e-5, 0.000000)
    , vec3(7.347551e-5, 2.860175e-5, 0.000000)
    , vec3(6.871576e-5, 2.675841e-5, 0.000000)
425 , vec3(6.425257e-5, 2.502943e-5, 0.000000)
    , vec3(6.008292e-5, 2.341373e-5, 0.000000)
    , vec3(5.620098e-5, 2.190914e-5, 0.000000)
    , vec3(5.259870e-5, 2.051259e-5, 0.000000)
    , vec3(4.926279e-5, 1.921902e-5, 0.000000)
430 , vec3(4.616623e-5, 1.801796e-5, 0.000000)
    , vec3(4.328212e-5, 1.689899e-5, 0.000000)
    , vec3(4.058715e-5, 1.585309e-5, 0.000000)
    , vec3(3.806114e-5, 1.487243e-5, 0.000000)
    , vec3(3.568818e-5, 1.395085e-5, 0.000000)
435 , vec3(3.346023e-5, 1.308528e-5, 0.000000)
    , vec3(3.137090e-5, 1.227327e-5, 0.000000)
    , vec3(2.941371e-5, 1.151233e-5, 0.000000)
    , vec3(2.758222e-5, 1.080001e-5, 0.000000)
    , vec3(2.586951e-5, 1.013364e-5, 0.000000)
440 , vec3(2.426701e-5, 9.509919e-6, 0.000000)
    , vec3(2.276639e-5, 8.925630e-6, 0.000000)
    , vec3(2.136009e-5, 8.377852e-6, 0.000000)
    , vec3(2.004122e-5, 7.863920e-6, 0.000000)
    , vec3(1.880380e-5, 7.381539e-6, 0.000000)
445 , vec3(1.764358e-5, 6.929096e-6, 0.000000)
    , vec3(1.655671e-5, 6.505136e-6, 0.000000)
    , vec3(1.553939e-5, 6.108221e-6, 0.000000)
    , vec3(1.458792e-5, 5.736935e-6, 0.000000)
```

```
    , vec3(1.369853e-5, 5.389831e-6, 0.000000)
    , vec3(1.286705e-5, 5.065269e-6, 0.000000)
    , vec3(1.208947e-5, 4.761667e-6, 0.000000)
    , vec3(1.136207e-5, 4.477561e-6, 0.000000)
    , vec3(1.068141e-5, 4.211597e-6, 0.000000)
    , vec3(1.004411e-5, 3.962457e-6, 0.000000)
    , vec3(9.446399e-6, 3.728674e-6, 0.000000)
    , vec3(8.884754e-6, 3.508881e-6, 0.000000)
    , vec3(8.356050e-6, 3.301868e-6, 0.000000)
    , vec3(7.857521e-6, 3.106561e-6, 0.000000)
    , vec3(7.386996e-6, 2.922119e-6, 0.000000)
    , vec3(6.943576e-6, 2.748208e-6, 0.000000)
    , vec3(6.526548e-6, 2.584560e-6, 0.000000)
    , vec3(6.135087e-6, 2.430867e-6, 0.000000)
    , vec3(5.768284e-6, 2.286786e-6, 0.000000)
    , vec3(5.425069e-6, 2.151905e-6, 0.000000)
    , vec3(5.103974e-6, 2.025656e-6, 0.000000)
    , vec3(4.803525e-6, 1.907464e-6, 0.000000)
    , vec3(4.522350e-6, 1.796794e-6, 0.000000)
    , vec3(4.259166e-6, 1.693147e-6, 0.000000)
    , vec3(4.012715e-6, 1.596032e-6, 0.000000)
    , vec3(3.781597e-6, 1.504903e-6, 0.000000)
    , vec3(3.564496e-6, 1.419245e-6, 0.000000)
    , vec3(3.360236e-6, 1.338600e-6, 0.000000)
    , vec3(3.167765e-6, 1.262556e-6, 0.000000)
    , vec3(2.986206e-6, 1.190771e-6, 0.000000)
    , vec3(2.814999e-6, 1.123031e-6, 0.000000)
    , vec3(2.653663e-6, 1.059151e-6, 0.000000)
    , vec3(2.501725e-6, 9.989507e-7, 0.000000)
    , vec3(2.358723e-6, 9.422514e-7, 0.000000)
    , vec3(2.224206e-6, 8.888804e-7, 0.000000)
    , vec3(2.097737e-6, 8.386690e-7, 0.000000)
    , vec3(1.978894e-6, 7.914539e-7, 0.000000)
    , vec3(1.867268e-6, 7.470770e-7, 0.000000)
    , vec3(1.762465e-6, 7.053860e-7, 0.000000)
    );
    float l = wavelength - wavelength_min;
    int i0 = int(floor(l));
    int i1 = i0 + 1;
    float f1 = l - float(i0);
    if (i0 < 0) return vec3(0.0);
    if (i1 > wavelength_count - 1) return vec3(0.0);
    return mix(tristimulus_curve[i0], tristimulus_curve[i1], f1);
#undef wavelength_count
}
```

26 include/Raymond-Pinhole.frag

```
#donotrun

/*
Raymond - a physics-inspired ray tracer for Fragmentarium
Copyright (C) 2018  Claude Heiland-Allen
License GPL3+ <http://www.gnu.org/licenses/>
*/

#group Pinhole
```

```
10
    // Pinhole aperture radius and depth (mm)
    uniform vec2 Aperture; slider[(0,0),(0.1,0.01),(10.0,10.0)]

    // Size of film plane of camera (mm)
15  uniform float Size; slider[0.0,35.0,1000.0]

    // Range of light wavelengths to trace (nm)
    uniform vec2 Wavelengths; slider[(0.0,0.0),(300.0,780.0),(1000.0,1000.0)]

20  struct Pinhole
    {
      vec2 aperture; // m
      float size; // m
      vec2 wavelengths; // nm
25  };

    Pinhole pinhole_uniforms()
    {
      Pinhole P;
30    P.aperture = Aperture / 1000.0;
      P.size = Size / 1000.0;
      P.wavelengths = Wavelengths;
      return P;
    }
35
    Ray pinhole(Random PRNG, Camera C, vec2 pixel, Pinhole P, out float intensity)
    {
      vec2 rect = halton2(srand(PRNG, 1));
      vec2 disc = haltonDisc(srand(PRNG, 2));
40    float wave = halton1(srand(PRNG, 3));
      float dist = P.size / 2.0 / tan(C.fieldOfView / 2.0);
      vec2 p = pixel + dFdx(pixel) * rect.x + dFdy(pixel) * rect.y;
      vec3 from = C.origin - dist * C.Z + P.size / 2.0 * (p.x * C.X + p.y * C.Y);
      vec3 to = C.origin + P.aperture.x * (disc.x * C.X + disc.y * C.Y);
45    Ray V;
      V.origin = to;
      V.direction = normalize(to - from);
      V.wavelength = mix(P.wavelengths.x, P.wavelengths.y, wave);
      V.index = vec2(1.0, 0.0); // vacuum
50    // vignette
      // <http://www.galerie-photo.com/stenope-cercle-image-theorie.html>
      float a = acos(clamp(dot(C.Z, V.direction), 0.0, 1.0));
      float t = tan(a);
      float d = P.aperture.x * 2.0;
55    float e = P.aperture.y;
      float ted = clamp(t * e / d, 0.0, 1.0);
      intensity = acos(ted) - t * e * d * sqrt(1.0 - ted * ted);
      return V;
    }
```

27 include/Raymond-Quartz.frag

```
#donotrun

/*
Raymond - a physics-inspired ray tracer for Fragmentarium
```

```
/*
  Copyright (C) 2018  Claude Heiland-Allen
  License GPL3+ <http://www.gnu.org/licenses/>
*/

vec2 Quartz_nk(float wavelength) {
  const float quartz_min   = 390;
  const float quartz_step  =   2;
#define quartz_count 221
  const vec2 quartz_nk[quartz_count] = vec2[quartz_count]
    ( vec2(1.490867, 0.000009)
    , vec2(1.490629, 0.000007)
    , vec2(1.490396, 0.000005)
    , vec2(1.490165, 0.000003)
    , vec2(1.489938, 0.000002)
    , vec2(1.489714, 0.000001)
    , vec2(1.489494, 0.000001)
    , vec2(1.489276, 0.000000)
    , vec2(1.489062, 0.000000)
    , vec2(1.488850, 0.000000)
    , vec2(1.488641, 0.000000)
    , vec2(1.488435, 0.000000)
    , vec2(1.488231, 0.000000)
    , vec2(1.488031, 0.000000)
    , vec2(1.487833, 0.000000)
    , vec2(1.487638, 0.000000)
    , vec2(1.487446, 0.000000)
    , vec2(1.487256, 0.000000)
    , vec2(1.487069, 0.000000)
    , vec2(1.486884, 0.000000)
    , vec2(1.486702, 0.000000)
    , vec2(1.486522, 0.000000)
    , vec2(1.486345, 0.000000)
    , vec2(1.486170, 0.000000)
    , vec2(1.485997, 0.000000)
    , vec2(1.485827, 0.000000)
    , vec2(1.485659, 0.000000)
    , vec2(1.485493, 0.000000)
    , vec2(1.485329, 0.000000)
    , vec2(1.485167, 0.000000)
    , vec2(1.485008, 0.000000)
    , vec2(1.484851, 0.000000)
    , vec2(1.484695, 0.000000)
    , vec2(1.484542, 0.000000)
    , vec2(1.484390, 0.000000)
    , vec2(1.484241, 0.000000)
    , vec2(1.484094, 0.000000)
    , vec2(1.483948, 0.000000)
    , vec2(1.483804, 0.000000)
    , vec2(1.483662, 0.000000)
    , vec2(1.483522, 0.000000)
    , vec2(1.483383, 0.000000)
    , vec2(1.483247, 0.000000)
    , vec2(1.483111, 0.000000)
    , vec2(1.482978, 0.000000)
    , vec2(1.482846, 0.000000)
    , vec2(1.482716, 0.000000)
    , vec2(1.482588, 0.000000)
```

```
        , vec2(1.482461, 0.000000)
        , vec2(1.482335, 0.000000)
        , vec2(1.482211, 0.000000)
65      , vec2(1.482089, 0.000000)
        , vec2(1.481968, 0.000000)
        , vec2(1.481848, 0.000000)
        , vec2(1.481730, 0.000000)
        , vec2(1.481613, 0.000000)
70      , vec2(1.481498, 0.000000)
        , vec2(1.481384, 0.000000)
        , vec2(1.481271, 0.000000)
        , vec2(1.481160, 0.000000)
        , vec2(1.481050, 0.000000)
75      , vec2(1.480941, 0.000000)
        , vec2(1.480834, 0.000000)
        , vec2(1.480727, 0.000000)
        , vec2(1.480622, 0.000000)
        , vec2(1.480518, 0.000000)
80      , vec2(1.480416, 0.000000)
        , vec2(1.480314, 0.000000)
        , vec2(1.480214, 0.000000)
        , vec2(1.480115, 0.000000)
        , vec2(1.480016, 0.000000)
85      , vec2(1.479919, 0.000000)
        , vec2(1.479823, 0.000000)
        , vec2(1.479729, 0.000000)
        , vec2(1.479635, 0.000000)
        , vec2(1.479542, 0.000000)
90      , vec2(1.479450, 0.000000)
        , vec2(1.479359, 0.000000)
        , vec2(1.479269, 0.000000)
        , vec2(1.479181, 0.000000)
        , vec2(1.479093, 0.000000)
95      , vec2(1.479006, 0.000000)
        , vec2(1.478920, 0.000000)
        , vec2(1.478835, 0.000000)
        , vec2(1.478751, 0.000000)
        , vec2(1.478667, 0.000000)
100     , vec2(1.478585, 0.000000)
        , vec2(1.478503, 0.000000)
        , vec2(1.478423, 0.000000)
        , vec2(1.478343, 0.000000)
        , vec2(1.478264, 0.000000)
105     , vec2(1.478186, 0.000000)
        , vec2(1.478108, 0.000000)
        , vec2(1.478032, 0.000000)
        , vec2(1.477956, 0.000000)
        , vec2(1.477881, 0.000000)
110     , vec2(1.477807, 0.000000)
        , vec2(1.477733, 0.000000)
        , vec2(1.477660, 0.000000)
        , vec2(1.477588, 0.000000)
        , vec2(1.477517, 0.000000)
115     , vec2(1.477447, 0.000000)
        , vec2(1.477377, 0.000000)
        , vec2(1.477308, 0.000000)
        , vec2(1.477239, 0.000000)
```

```
        , vec2(1.477171, 0.000000)
120     , vec2(1.477104, 0.000000)
        , vec2(1.477038, 0.000000)
        , vec2(1.476972, 0.000000)
        , vec2(1.476907, 0.000000)
        , vec2(1.476842, 0.000000)
125     , vec2(1.476778, 0.000000)
        , vec2(1.476715, 0.000000)
        , vec2(1.476652, 0.000000)
        , vec2(1.476590, 0.000000)
        , vec2(1.476529, 0.000000)
130     , vec2(1.476468, 0.000000)
        , vec2(1.476408, 0.000000)
        , vec2(1.476348, 0.000000)
        , vec2(1.476289, 0.000000)
        , vec2(1.476230, 0.000000)
135     , vec2(1.476172, 0.000000)
        , vec2(1.476115, 0.000000)
        , vec2(1.476058, 0.000000)
        , vec2(1.476001, 0.000000)
        , vec2(1.475946, 0.000000)
140     , vec2(1.475890, 0.000000)
        , vec2(1.475835, 0.000000)
        , vec2(1.475781, 0.000000)
        , vec2(1.475727, 0.000000)
        , vec2(1.475674, 0.000000)
145     , vec2(1.475621, 0.000000)
        , vec2(1.475568, 0.000000)
        , vec2(1.475517, 0.000000)
        , vec2(1.475465, 0.000000)
        , vec2(1.475414, 0.000000)
150     , vec2(1.475364, 0.000000)
        , vec2(1.475314, 0.000000)
        , vec2(1.475264, 0.000000)
        , vec2(1.475215, 0.000000)
        , vec2(1.475166, 0.000000)
155     , vec2(1.475118, 0.000000)
        , vec2(1.475070, 0.000000)
        , vec2(1.475022, 0.000000)
        , vec2(1.474975, 0.000000)
        , vec2(1.474929, 0.000000)
160     , vec2(1.474883, 0.000000)
        , vec2(1.474837, 0.000000)
        , vec2(1.474791, 0.000000)
        , vec2(1.474746, 0.000000)
        , vec2(1.474702, 0.000000)
165     , vec2(1.474657, 0.000000)
        , vec2(1.474614, 0.000000)
        , vec2(1.474570, 0.000000)
        , vec2(1.474527, 0.000000)
        , vec2(1.474484, 0.000000)
170     , vec2(1.474442, 0.000000)
        , vec2(1.474400, 0.000000)
        , vec2(1.474358, 0.000000)
        , vec2(1.474317, 0.000000)
        , vec2(1.474276, 0.000000)
175     , vec2(1.474235, 0.000000)
```

```
        , vec2(1.474195, 0.000000)
        , vec2(1.474155, 0.000000)
        , vec2(1.474115, 0.000000)
        , vec2(1.474076, 0.000000)
180     , vec2(1.474037, 0.000000)
        , vec2(1.473998, 0.000000)
        , vec2(1.473960, 0.000000)
        , vec2(1.473922, 0.000000)
        , vec2(1.473884, 0.000000)
185     , vec2(1.473847, 0.000000)
        , vec2(1.473810, 0.000000)
        , vec2(1.473773, 0.000000)
        , vec2(1.473737, 0.000000)
        , vec2(1.473700, 0.000000)
190     , vec2(1.473665, 0.000000)
        , vec2(1.473629, 0.000000)
        , vec2(1.473594, 0.000000)
        , vec2(1.473559, 0.000000)
        , vec2(1.473524, 0.000000)
195     , vec2(1.473489, 0.000000)
        , vec2(1.473455, 0.000000)
        , vec2(1.473421, 0.000000)
        , vec2(1.473387, 0.000000)
        , vec2(1.473354, 0.000000)
200     , vec2(1.473321, 0.000000)
        , vec2(1.473288, 0.000000)
        , vec2(1.473255, 0.000000)
        , vec2(1.473223, 0.000000)
        , vec2(1.473191, 0.000000)
205     , vec2(1.473159, 0.000000)
        , vec2(1.473127, 0.000000)
        , vec2(1.473096, 0.000000)
        , vec2(1.473065, 0.000000)
        , vec2(1.473034, 0.000000)
210     , vec2(1.473003, 0.000000)
        , vec2(1.472973, 0.000000)
        , vec2(1.472942, 0.000000)
        , vec2(1.472912, 0.000000)
        , vec2(1.472883, 0.000000)
215     , vec2(1.472853, 0.000000)
        , vec2(1.472824, 0.000000)
        , vec2(1.472795, 0.000000)
        , vec2(1.472766, 0.000000)
        , vec2(1.472737, 0.000000)
220     , vec2(1.472709, 0.000000)
        , vec2(1.472680, 0.000000)
        , vec2(1.472652, 0.000000)
        , vec2(1.472625, 0.000000)
        , vec2(1.472597, 0.000000)
225     , vec2(1.472570, 0.000000)
        , vec2(1.472542, 0.000000)
        , vec2(1.472515, 0.000000)
        , vec2(1.472489, 0.000000)
        , vec2(1.472462, 0.000000)
230     , vec2(1.472436, 0.000000)
        , vec2(1.472409, 0.000000)
        , vec2(1.472383, 0.000000)
```

```
    , vec2(1.472357, 0.000000)
    , vec2(1.472332, 0.000000)
    );
  float x  = (wavelength - quartz_min) / quartz_step;
  int   i0 = int(floor(x));
  int   i1 = i0 + 1;
  float f1 = x - float(i0);
  if (i0 < 0) return quartz_nk[0];
  if (i1 > quartz_count - 1) return quartz_nk[quartz_count - 1];
  vec2 y0 = quartz_nk[i0];
  vec2 y1 = quartz_nk[i1];
  return mix(y0, y1, f1);
#undef quartz_count
}

float Quartz(Random PRNG, float S, Ray V)
{
  return S;
}

Hit Quartz(Random PRNG, Surface S, Ray V)
{
  return Transparent(PRNG, S, V, Quartz_nk(V.wavelength));
}
```

28 include/Raymond-Quaternion.frag

```
#donotrun

/*
Raymond - a physics-inspired ray tracer for Fragmentarium
Copyright (C) 2018  Claude Heiland-Allen
License GPL3+ <http://www.gnu.org/licenses/>
*/

vec4 slerp(vec4 v0, vec4 v1, float t)
{
  float d = dot(v0, v1);
  if (d < 0.0)
  {
    v1 = -v1;
    d = -d;
  }
  if (d > 0.9995)
  {
    return normalize(mix(v0, v1, t));
  }
  float theta_0 = acos(d);
  float theta = theta_0 * t;
  float sin_theta = sin(theta);
  float sin_theta_0 = sin(theta_0);
  float s0 = cos(theta) - d * sin_theta / sin_theta_0;
  float s1 = sin_theta / sin_theta_0;
  return s0 * v0 + s1 * v1;
}

vec3 qMul(vec4 q, vec3 v)
```

```
{
  vec3 t = 2.0 * cross(q.xyz, v);
  return v + q.w * t + cross(q.xyz, t);
}

vec4 qInv(vec4 q)
{
  return vec4(-q.xyz, q.w) / dot(q, q);
}
```

29 include/Raymond-Random.frag

```
#donotrun

/*
Raymond - a physics-inspired ray tracer for Fragmentarium
Copyright (C) 2018  Claude Heiland-Allen
License GPL3+ <http://www.gnu.org/licenses/>
*/

// uniforms in [0,1)^n

float uniform01(Random PRNG)
{
  return uniform01(PRNG.seed);
}

float uniform01(Random PRNG, int i)
{
  return uniform01(srand(PRNG, i));
}

float uniform1(Random PRNG)
{
  return uniform01(PRNG, 1);
}

vec2 uniform2(Random PRNG)
{
  return vec2(uniform01(PRNG, 1), uniform01(PRNG, 2));
}

vec3 uniform3(Random PRNG)
{
  return vec3(uniform01(PRNG, 1), uniform01(PRNG, 2), uniform01(PRNG, 3));
}

vec4 uniform4(Random PRNG)
{
  return vec4(uniform01(PRNG, 1), uniform01(PRNG, 2), uniform01(PRNG, 3), ↙
      ↳ uniform01(PRNG, 4));
}

// uniform disc

vec2 uniformDiscNaive(vec2 u01)
{
```

```
  float r = sqrt(u01.x);
  float t = 2.0 * pi * u01.y;
  return r * vec2(cos(t), sin(t));
}

// http://www.pbr-book.org/3ed-2018/Monte_Carlo_Integration/2D_Sampling_with_Multidimensional_Transformations.html#SamplingaUnitDisk
vec2 uniformDisc(vec2 u01)
{
  vec2 u11 = 2.0 * u01 - vec2(1.0);
  if (u11 == vec2(0.0)) return u11;
  float r, t;
  if (abs(u11.x) > abs(u11.y))
  {
    r = u11.x;
    t = pi / 4.0 * (u11.y / u11.x);
  }
  else
  {
    r = u11.y;
    t = pi / 2.0 - pi / 4.0 * (u11.x / u11.y);
  }
  return r * vec2(cos(t), sin(t));
}

vec2 uniformDisc(Random PRNG)
{
  return uniformDisc(uniform2(PRNG));
}

// Box-Muller transform

vec2 gaussian(vec2 u01)
{
  float s = u01.x;
  float t = 2.0 * pi * u01.y;
  if (s > 0.0)
    return vec2(cos(t), sin(t)) * sqrt(-2.0 * log(s) / s);
  else
    return vec2(0.0);
}

vec2 gaussian2(Random PRNG)
{
  return gaussian(uniform2(PRNG));
}

float gaussian1(Random PRNG)
{
  return gaussian2(PRNG).x;
}

vec4 gaussian4(Random PRNG)
{
  vec4 u = uniform4(PRNG);
  return vec4(gaussian(u.xy), gaussian(u.zw));
}
```

```
vec3 gaussian3(Random PRNG)
{
  return gaussian4(PRNG).xyz;
}

// a rotation matrix that makes Z point along the normal

mat3 localCoordinates(vec3 normal)
{
  vec3 w = X;
  if (abs(dot(Y, normal)) < abs(dot(w, normal))) w = Y;
  if (abs(dot(Z, normal)) < abs(dot(w, normal))) w = Z;
  vec3 a = normalize(cross(w, normal));
  vec3 b = normalize(cross(a, normal));
  if (dot(normal, cross(a, b)) < 0.0) { vec3 t = a; a = b; b = t; }
  return mat3(a, b, normal);
}

// cosine weighted hemisphere from uniform disc and normal

vec3 cosHemisphere(vec2 uDisc, vec3 normal)
{
  mat3 m = localCoordinates(normal);
  vec3 w = vec3(uDisc, sqrt(max(0.0, 1.0 - dot(uDisc, uDisc))));
  return m * w;
}
```

30 include/Raymond-Screen.frag

```
#donotrun

/*
Raymond - a physics-inspired ray tracer for Fragmentarium
Copyright (C) 2018  Claude Heiland-Allen
License GPL3+ <http://www.gnu.org/licenses/>
*/

float Screen_hump(vec3 spec, float wavelength)
{
  float delta = (wavelength - spec[0]) / spec[1];
  return spec[2] / (1.0 + 4.0 * delta * delta);
}

// <http://www.marcelpatek.com/LCD.html> (Figure 2)
float CRT(vec3 linearRGB, float wavelength)
{
  const vec3 r[2] = vec3[2]
    ( vec3(710.0,  25.0, 0.5)
    , vec3(625.0,  25.0, 1.0)
    );
  const vec3 g =
      vec3(540.0, 100.0, 0.5);
  const vec3 b =
      vec3(450.0, 100.0, 0.5);
  float v = 0.0;
  v += linearRGB.r * Screen_hump(r[0], wavelength);
```

```
  v += linearRGB.r * Screen_hump(r[1], wavelength);
  v += linearRGB.g * Screen_hump(g   , wavelength);
  v += linearRGB.b * Screen_hump(b   , wavelength);
  return v;
}

// <http://www.marcelpatek.com/LCD.html> (Figure 3)
float LCD(vec3 linearRGB, float wavelength)
{
  const vec3 r[2] = vec3[2]
    ( vec3(710.0, 25.0, 0.1 * 0.8)
    , vec3(610.0, 15.0, 1.0 * 0.8)
    );
  const vec3 g[3] = vec3[3]
    ( vec3(585.0, 25.0, 0.2)
    , vec3(545.0, 15.0, 1.0)
    , vec3(490.0, 20.0, 0.3)
    );
  const vec3 b[3] = vec3[3]
    ( vec3(550.0, 25.0, 0.1 * 0.5)
    , vec3(590.0, 40.0, 0.9 * 0.5)
    , vec3(450.0, 50.0, 1.0 * 0.5)
    );
  float v = 0.0;
  v += linearRGB.r * Screen_hump(r[0], wavelength);
  v += linearRGB.r * Screen_hump(r[1], wavelength);
  v += linearRGB.g * Screen_hump(g[0], wavelength);
  v += linearRGB.g * Screen_hump(g[1], wavelength);
  v += linearRGB.g * Screen_hump(g[2], wavelength);
  v += linearRGB.b * Screen_hump(b[0], wavelength);
  v += linearRGB.b * Screen_hump(b[1], wavelength);
  v += linearRGB.b * Screen_hump(b[2], wavelength);
  return v;
}
```

31 include/Raymond-SkyBox.frag

```
#donotrun

/*
Raymond - a physics-inspired ray tracer for Fragmentarium
Copyright (C) 2018  Claude Heiland-Allen
License GPL3+ <http://www.gnu.org/licenses/>
*/

// implement this
float screen(vec4 tex, float wavelength);

#group SkyBox

// equirectangular (360x180 degrees) texture for background
uniform sampler2D Background; file[]

// distance from origin at which the cube map is drawn
uniform float Distance; slider[0,100,1000]

// no struct SkyBox as samplers can only be uniforms or function arguments
```

```
vec4 textureEqui(sampler2D sampler, vec3 dir)
{
  dir = normalize(dir);
  vec2 coord = vec2(atan(dir.y, -dir.x) / (2.0 * pi), acos(-dir.z) / pi);
  return texture(sampler, coord);
}

vec4 SkyBox_texture(Transform T, Ray V)
{
  return textureEqui(Background, backwardN(T, V.direction));
}

float SkyBox_intensity(Transform T, Ray V)
{
  return screen(SkyBox_texture(T, V), V.wavelength);
}

float SkyBox(Scene_DE tag, Transform T, Ray V)
{
  return Distance - length(backwardP(T, V.origin));
}

Hit SkyBox(Scene_HIT tag, Transform T, Ray V)
{
  Scene_DE DE;
  Surface S;
  S.position = V.origin;
  S.normal = -normalize(V.origin);
  S.de = SkyBox(DE, T, V);
  Hit h;
  h.surface = S;
  h.ray = V;
  h.factor = 0.0;
  h.emit = SkyBox_intensity(T, V);
  return h;
}
```

32 include/Raymond-sRGB-Buffer.frag

```
#donotrun

/*
Raymond - a physics-inspired ray tracer for Fragmentarium
Copyright (C) 2018  Claude Heiland-Allen
License GPL3+ <http://www.gnu.org/licenses/>
*/

#vertex

varying vec2 coord;

void main(void)
{
  gl_Position =  gl_Vertex;
  coord = ((gl_ProjectionMatrix * gl_Vertex).xy + vec2(1.0)) * 0.5;
}
```

```
#endvertex

varying vec2 coord;

uniform sampler2D frontbuffer;

#group Post

uniform float Exposure; slider[0.0,1.0,100.0]

float sRGB(float c)
{
  c = clamp(c, 0.0, 1.0);
  const float a = 0.055;
  if (c <= 0.0031308)
    return 12.92 * c;
  else
    return (1.0 + a) * pow(c, 1.0 / 2.4) - a;
}

vec3 sRGB(vec3 c)
{
  return vec3(sRGB(c.x), sRGB(c.y), sRGB(c.z));
}

void main(void)
{
  vec4 tex = texture2D(frontbuffer, coord);
  vec3 c = sRGB(Exposure * tex.xyz / tex.a);
  gl_FragColor = vec4(c, 1.0);
}
```

33 include/Raymond-sRGB.frag

```
#donotrun

/*
Raymond - a physics-inspired ray tracer for Fragmentarium
Copyright (C) 2018  Claude Heiland-Allen
License GPL3+ <http://www.gnu.org/licenses/>
*/

// accumulate linear RGBW and output as sRGB with exposure control
#buffer RGBA32F
#buffershader "Raymond-sRGB-Buffer.frag"
#group Post
uniform float Exposure; slider[0.0,1.0,100.0]

// https://en.wikipedia.org/wiki/SRGB#The_forward_transformation_(↙
    ↳ CIE_XYZ_to_sRGB)
float linear2sRGB(float c)
{
  c = clamp(c, 0.0, 1.0);
  const float a = 0.055;
  if (c <= 0.0031308)
    return 12.92 * c;
```

```
  else
    return (1.0 + a) * pow(c, 1.0 / 2.4) - a;
}

vec3 linear2sRGB(vec3 c)
{
  return vec3(linear2sRGB(c.x), linear2sRGB(c.y), linear2sRGB(c.z));
}

// https://en.wikipedia.org/wiki/SRGB#The_reverse_transformation
float sRGB2linear(float c)
{
  c = clamp(c, 0.0, 1.0);
  const float a = 0.055;
  if (c <= 0.04045)
    return c / 12.92;
  else
    return pow((c + a) / (1.0 + a), 2.4);
}

vec3 sRGB2linear(vec3 c)
{
  return vec3(sRGB2linear(c.x), sRGB2linear(c.y), sRGB2linear(c.z));
}
```

34 include/Raymond-StdDev-Buffer.frag

```
#donotrun

/*
Raymond - a physics-inspired ray tracer for Fragmentarium
Copyright (C) 2018  Claude Heiland-Allen
License GPL3+ <http://www.gnu.org/licenses/>
*/

#vertex

varying vec2 coord;

void main(void)
{
  gl_Position =  gl_Vertex;
  coord = ((gl_ProjectionMatrix * gl_Vertex).xy + vec2(1.0)) * 0.5;
}

#endvertex

varying vec2 coord;

uniform sampler2D frontbuffer;

vec3 stddev(vec3 v)
{
  float m = v[1] / v[0];
  float s = sqrt(v[0] * v[2] - v[1] * v[1]) / v[0];
  return vec3(s);
}
```

```
void main(void)
{
  vec4 tex = texture2D(frontbuffer, coord);
  vec3 c = stddev(tex.xyz / tex.a);
  gl_FragColor = vec4(c, 1.0);
}
```

35 include/Raymond-StdDev.frag

```
#donotrun

/*
Raymond - a physics-inspired ray tracer for Fragmentarium
Copyright (C) 2018  Claude Heiland-Allen
License GPL3+ <http://www.gnu.org/licenses/>
*/

#buffer RGBA32F
#buffershader "Raymond-StdDev-Buffer.frag"

vec3 film_stddev(Random PRNG, float wavelength, float intensity)
{
  return vec3(1.0, intensity, intensity * intensity);
}
```

36 include/Raymond-Surface.frag

```
#donotrun

/*
Raymond - a physics-inspired ray tracer for Fragmentarium
Copyright (C) 2018  Claude Heiland-Allen
License GPL3+ <http://www.gnu.org/licenses/>
*/

// CSG invert

float Invert(float de)
{
  return -de;
}

Surface Invert(Surface s)
{
  s.normal = - s.normal;
  s.de = - s.de;
  return s;
}

// CSG union: minimum by de

float Union(float a, float b)
{
  if (a < b) return a; else return b;
}
```

```
30  Surface Union(Surface a, Surface b)
    {
      if (a.de < b.de) return a; else return b;
    }

35  #define Union3(a,b,c) Union(Union(a, b), c)
    #define Union4(a,b,c,d) Union(Union(Union(a, b), c), d)
    #define Union5(a,b,c,d,e) Union(Union(Union(Union(a, b), c), d), e)
    #define Union6(a,b,c,d,e,f) Union(Union(Union(Union(Union(a, b), c), d), e), f)

40  // CSG intersection: maximum by de

    float Intersection(float a, float b)
    {
      if (a > b) return a; else return b;
45  }

    Surface Intersection(Surface a, Surface b)
    {
      if (a.de > b.de) return a; else return b;
50  }

    #define Intersection3(a,b,c) Intersection(Intersection(a, b), c)
    #define Intersection4(a,b,c,d) Intersection(Intersection(Intersection(a, b), c),↙
        ↪ d)
    #define Intersection5(a,b,c,d,e) Intersection(Intersection(Intersection(↙
        ↪ Intersection(a, b), c), d), e)
55  #define Intersection6(a,b,c,d,e,f) Intersection(Intersection(Intersection(↙
        ↪ Intersection(Intersection(a, b), c), d), e), f)

    // primitives

    float Sphere(Scene_DE tag, Transform T, Ray V)
60  {
      return forwardD(T, length(backwardP(T, V.origin)) - 1.0);
    }

    Surface Sphere(Scene_HIT tag, Transform T, Ray V)
65  {
      vec3 world = V.origin;
      vec3 object = backwardP(T, world);
      vec3 normal = normalize(object);
      float de = length(object) - 1.0;
70    Surface s;
      s.position = forwardP(T, object);
      s.normal = forwardN(T, normal);
      s.de = forwardD(T, de);
      return s;
75  }

    float Plane(Scene_DE tag, Transform T, Ray V, vec3 normal, float dist)
    {
      return forwardD(T, dot(backwardP(T, V.origin), normalize(normal)) - dist);
80  }

    Surface Plane(Scene_HIT tag, Transform T, Ray V, vec3 normal, float dist)
```

```
{
  normal = normalize(normal);
  vec3 world = V.origin;
  vec3 object = backwardP(T, world);
  float d = dot(object, normal) - dist;
  vec3 nearest = object - d * normal;
  Surface s;
  s.position = forwardP(T, object);
  s.normal = forwardN(T, normal);
  nearest = forwardP(T, nearest);
  s.de = distance(s.position, nearest);
  if (d < 0.0)
    s.de = - s.de;
  return s;
}

// https://iquilezles.org/www/articles/distfunctions/distfunctions.htm
float sdBox( vec3 p, vec3 b )
{
  vec3 d = abs(p) - b;
  return length(max(d,0.0))
         + min(max(d.x,max(d.y,d.z)),0.0); // remove this line for an only ↙
             ↳ partially signed sdf
}

float Cuboid(Scene_DE tag, Transform T, Ray V, vec3 size)
{
  return forwardD(T, sdBox(backwardP(T, V.origin), size));
}

Surface Cuboid(Scene_HIT tag, Transform T, Ray V, vec3 size)
{
  vec3 p = backwardP(T, V.origin);
  float d = sdBox(p, size);
  float e = max(d, 1.0e-5 * length(size));
  vec3 n = vec3
    ( sdBox(p + e * X, size) - sdBox(p - e * X, size)
    , sdBox(p + e * Y, size) - sdBox(p - e * Y, size)
    , sdBox(p + e * Z, size) - sdBox(p - e * Z, size)
    );
  Surface s;
  s.position = forwardP(T, p);
  s.normal = forwardN(T, normalize(n));
  s.de = forwardD(T, d);
  return s;
}

float Cube(Scene_DE tag, Transform T, Ray V, float size)
{
  return Cuboid(tag, T, V, vec3(size));
}

Surface Cube(Scene_HIT tag, Transform T, Ray V, float size)
{
  return Cuboid(tag, T, V, vec3(size));
}
```

```
float Cube(Scene_DE tag, Transform T, Ray V)
{
  return Cube(tag, T, V, 1.0);
}

Surface Cube(Scene_HIT tag, Transform T, Ray V)
{
  return Cube(tag, T, V, 1.0);
}
```

37 include/Raymond-Trace.frag

```
#donotrun

/*
Raymond - a physics-inspired ray tracer for Fragmentarium
Copyright (C) 2018  Claude Heiland-Allen
License GPL3+ <http://www.gnu.org/licenses/>
*/

#group Raytrace

// Number of steps to march along each ray.
uniform int Steps; slider[0,100,1000]

// Maximum number of surface intersections per pixel.
uniform int Depth; slider[0,10,100]

// Stop tracing rays at this epsilon (in bits) to stop early.
uniform float MinDist; slider[-24.0,-16.0,8.0]

// Advance rays by this epsilon (in bits) to avoid repeated surface ↙
    ↳ intersections.
uniform float Acne; slider[-24.0,-12.0,8.0]

// Trace towards lights from each surface.
uniform bool SampleLights; checkbox[true]

struct Raytrace
{
  int steps;
  int depth;
  float mindist;
  float acne;
  bool samplelights;
};

Raytrace raytrace_de_uniforms()
{
  Raytrace R;
  R.steps = Steps;
  R.depth = Depth;
  R.mindist = pow(2.0, MinDist);
  R.acne = pow(2.0, Acne);
  R.samplelights = SampleLights;
  return R;
}
```

```
45
Hit raytrace_de_1(Random PRNG, Ray V, Raytrace R, float s)
{
  Scene_DE DE;
  int i = 0;
50  float d = scene(DE, srand(PRNG, i), V);
  for (; i < R.steps; ++i)
  {
    d = scene(DE, srand(PRNG, i), V);
    V.origin += s * d * V.direction;
55  //if (s * d < R.mindist) break;
  }
  Scene_HIT HIT;
  return scene(HIT, srand(PRNG, i), V);
}
60
float raytrace_de(Random PRNG, Ray V, Raytrace R, out Hit D)
{
  float I = 0.0;
  float f = 1.0;
65  float s = 1.0;
  for (int i = 0; i < R.depth; ++i)
  {
    // trace to surface
    Hit h = raytrace_de_1(srand(PRNG, i), V, R, s);
70  if (i == 0) D = h;
    f *= absorption(V.index.y, V.wavelength, distance(V.origin, h.ray.origin));
    if (R.samplelights)
    {
      // ignore lights from organic rays
75    if (h.emit > 0.0)
      {
        if (i == 0)
        {
          I += f * h.emit;
80        f *= h.factor;
        }
        break;
      }
      f *= h.factor;
85    // trace from surface to lights
      Ray U = h.ray;
      vec4 L = light(srand(PRNG, i), U.origin, U.direction);
      if (L.w > 0.0)
      {
90      U.direction = normalize(L.xyz);
        U.origin += R.acne * U.direction;
        Hit l = raytrace_de_1(srand(PRNG, i), U, R, s);
        if (l.emit > 0.0)
        {
95        I += L.w * f * absorption(U.index.y, U.wavelength, distance(U.origin, ↙
              ↪ l.ray.origin)) * l.emit;
        }
      }
      f *= (1.0 - L.w);
    }
100 else
```

```
    {
      I += f * h.emit;
      f *= h.factor;
    }
    // bounce
    V = h.ray;
    V.origin += R.acne * V.direction;
    Scene_HIT HIT;
    Surface S = scene(HIT, srand(PRNG, i + 1), V).surface;
    float s2 = sign(S.de);
    float s3 = sign(dot(V.direction, S.normal));
    s = s2;
    if (! (f > 0)) break;
  }
  return I;
}
```

38 include/Raymond-Transform.frag

```
#donotrun

/*
Raymond - a physics-inspired ray tracer for Fragmentarium
Copyright (C) 2018  Claude Heiland-Allen
License GPL3+ <http://www.gnu.org/licenses/>
*/

// transformations

struct Transform
{
  mat4 forward; // object to world
  mat4 backward; // world to object
  float scale; // forward
};

// composition of transformations

Transform compose(Transform S, Transform T)
{
  Transform R;
  R.forward = S.forward * T.forward;
  R.backward = T.backward * S.backward;
  R.scale = S.scale * T.scale;
  return R;
}

Transform compose(Transform A, Transform B, Transform C)
{
  return compose(compose(A, B), C);
}

Transform compose(Transform A, Transform B, Transform C, Transform D)
{
  return compose(compose(compose(A, B), C), D);
}
```

```
// transformation of distances

float forwardD(Transform T, float d)
{
  return d * T.scale;
}

float backwardD(Transform T, float d)
{
  return d / T.scale;
}

// transformation of points

vec3 forwardP(Transform T, vec3 p)
{
  vec4 q = vec4(p, 1.0) * T.forward;
  return q.xyz / q.w;
}

vec3 backwardP(Transform T, vec3 p)
{
  vec4 q = vec4(p, 1.0) * T.backward;
  return q.xyz / q.w;
}

// transformation of normals

vec3 forwardN(Transform T, vec3 n)
{
  return normalize(n * inverse(transpose(mat3(T.forward))));
}

vec3 backwardN(Transform T, vec3 n)
{
  return normalize(n * inverse(transpose(mat3(T.backward))));
}

// uniform scaling

Transform Scale(float s)
{
  Transform T;
  T.scale = s;
  T.forward = mat4
    ( s, 0.0, 0.0, 0.0
    , 0.0, s, 0.0, 0.0
    , 0.0, 0.0, s, 0.0
    , 0.0, 0.0, 0.0, 1.0
    );
  s = 1.0 / s;
  T.backward = mat4
    ( s, 0.0, 0.0, 0.0
    , 0.0, s, 0.0, 0.0
    , 0.0, 0.0, s, 0.0
    , 0.0, 0.0, 0.0, 1.0
```

```
    );
  return T;
}

// identity transformation

Transform Identity()
{
  return Scale(1.0);
}

// non-uniform scaling

Transform Scale(vec3 s)
{
  Transform T;
  T.forward = mat4
    ( s.x, 0.0, 0.0, 0.0
    , 0.0, s.y, 0.0, 0.0
    , 0.0, 0.0, s.z, 0.0
    , 0.0, 0.0, 0.0, 1.0
    );
  s = 1.0 / s;
  T.backward = mat4
    ( s.x, 0.0, 0.0, 0.0
    , 0.0, s.y, 0.0, 0.0
    , 0.0, 0.0, s.z, 0.0
    , 0.0, 0.0, 0.0, 1.0
    );
  T.scale = determinant(mat3(T.forward)); // FIXME
  return T;
}

// translation

Transform Translate(vec3 v)
{
  Transform T;
  T.scale = 1.0;
  T.forward = (mat4
    ( 1.0, 0.0, 0.0, v.x
    , 0.0, 1.0, 0.0, v.y
    , 0.0, 0.0, 1.0, v.z
    , 0.0, 0.0, 0.0, 1.0
    ));
  v = -v;
  T.backward = (mat4
    ( 1.0, 0.0, 0.0, v.x
    , 0.0, 1.0, 0.0, v.y
    , 0.0, 0.0, 1.0, v.z
    , 0.0, 0.0, 0.0, 1.0
    ));
  return T;
}

// rotation from quaternion
```

```
Transform Rotate(vec4 q)
{
  q = normalize(q);
  float i = q.x;
  float j = q.y;
  float k = q.z;
  float r = q.w;
  Transform T;
  T.scale = 1.0;
  T.forward = mat4
    ( 1.0 - 2.0 * (j * j + k * k), 2.0 * (i * j - k * r), 2.0 * (i * k + j * r),↙
      ↘ 0.0
    , 2.0 * (i * j + k * r), 1.0 - 2.0 * (i * i + k * k), 2.0 * (j * k - i * r),↙
      ↘ 0.0
    , 2.0 * (i * k - j * r), 2.0 * (j * k + i * r), 1.0 - 2.0 * (i * i + j * j),↙
      ↘ 0.0
    , 0.0, 0.0, 0.0, 1.0
    );
  T.backward = transpose(T.forward);
  return T;
}
```

39 include/Raymond-Water.frag

```
#donotrun

/*
Raymond - a physics-inspired ray tracer for Fragmentarium
Copyright (C) 2018  Claude Heiland-Allen
License GPL3+ <http://www.gnu.org/licenses/>
*/

vec2 Water_nk(float wavelength)
{
  const float water_min   = 375;
  const float water_step  =  25;
#define water_count 20
  const vec2  water_nk[water_count] = vec2[water_count]
    ( vec2(1.341, 3.50E-9)
    , vec2(1.339, 1.86E-9)
    , vec2(1.338, 1.30E-9)
    , vec2(1.337, 1.02E-9)
    , vec2(1.336, 9.35E-10)
    , vec2(1.335, 1.00E-9)
    , vec2(1.334, 1.32E-9)
    , vec2(1.333, 1.96E-9)
    , vec2(1.333, 3.60E-9)
    , vec2(1.332, 1.09E-8)
    , vec2(1.332, 1.39E-8)
    , vec2(1.331, 1.64E-8)
    , vec2(1.331, 2.23E-8)
    , vec2(1.331, 3.35E-8)
    , vec2(1.330, 9.15E-8)
    , vec2(1.330, 1.56E-7)
    , vec2(1.330, 1.48E-7)
    , vec2(1.329, 1.25E-7)
    , vec2(1.329, 1.82E-7)
```

```
      , vec2(1.329, 2.93E-7)
      );
  float x  = (wavelength - water_min) / water_step;
  int   i0 = int(floor(x));
  int   i1 = i0 + 1;
  float f1 = x - float(i0);
  if (i0 < 0) return water_nk[0];
  if (i1 > water_count - 1) return water_nk[water_count - 1];
  vec2 y0 = water_nk[i0];
  vec2 y1 = water_nk[i1];
  return mix(y0, y1, f1);
#undef water_count
}

float Water(Random PRNG, float S, Ray V)
{
  return S;
}

Hit Water(Random PRNG, Surface S, Ray V)
{
  return Transparent(PRNG, S, V, Water_nk(V.wavelength));
}
```

40 LICENSE.md

```
### GNU GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

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

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

### Preamble

The GNU General Public License is a free, copyleft license for
software and other kinds of works.

The licenses for most software and other practical works are designed
to take away your freedom to share and change the works. By contrast,
the GNU General Public License is intended to guarantee your freedom
to share and change all versions of a program--to make sure it remains
free software for all its users. We, the Free Software Foundation, use
the GNU General Public License for most of our software; it applies
also to any other work released this way by its authors. You can apply
it to your programs, too.

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

To protect your rights, we need to prevent others from denying you
these rights or asking you to surrender the rights. Therefore, you
have certain responsibilities if you distribute copies of the
software, or if you modify it: responsibilities to respect the freedom
of others.

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

Developers that use the GNU GPL protect your rights with two steps:
(1) assert copyright on the software, and (2) offer you this License
giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains
that there is no warranty for this free software. For both users' and
authors' sake, the GPL requires that modified versions be marked as
changed, so that their problems will not be attributed erroneously to
authors of previous versions.

Some devices are designed to deny users access to install or run
modified versions of the software inside them, although the
manufacturer can do so. This is fundamentally incompatible with the
aim of protecting users' freedom to change the software. The
systematic pattern of such abuse occurs in the area of products for
individuals to use, which is precisely where it is most unacceptable.
Therefore, we have designed this version of the GPL to prohibit the
practice for those products. If such problems arise substantially in
other domains, we stand ready to extend this provision to those
domains in future versions of the GPL, as needed to protect the
freedom of users.

Finally, every program is threatened constantly by software patents.
States should not allow patents to restrict development and use of
software on general-purpose computers, but in those that do, we wish
to avoid the special danger that patents applied to a free program
could make it effectively proprietary. To prevent this, the GPL
assures that patents cannot be used to render the program non-free.

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

TERMS AND CONDITIONS

0. Definitions.

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

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

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

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

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

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

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

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

1. Source Code.

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

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

The "System Libraries" of an executable work include anything, other
than the work as a whole, that (a) is included in the normal form of
packaging a Major Component, but which is not part of that Major
Component, and (b) serves only to enable use of the work with that
Major Component, or to implement a Standard Interface for which an
implementation is available to the public in source code form. A
"Major Component", in this context, means a major essential component
(kernel, window system, and so on) of the specific operating system
(if any) on which the executable work runs, or a compiler used to
produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all
the source code needed to generate, install, and (for an executable
work) run the object code and to modify the work, including scripts to
control those activities. However, it does not include the work's
System Libraries, or general-purpose tools or generally available free
programs which are used unmodified in performing those activities but

which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

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

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

2. Basic Permissions.

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

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

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

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

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

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

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you

receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

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

5. Conveying Modified Source Versions.

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

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

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

6. Conveying Non-Source Forms.

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

- a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.
- b) Convey the object code in, or embodied in, a physical product

(including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

- c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.
- d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.
- e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

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

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

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

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

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

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

7. Additional Terms.

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

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

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

- a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
- b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
- c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

- d) Limiting the use for publicity purposes of names of licensors or authors of the material; or
- e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or
- f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

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

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

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

8. Termination.

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

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

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

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

9. Acceptance Not Required for Having Copies.

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

10. Automatic Licensing of Downstream Recipients.

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

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

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

11. Patents.

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

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

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

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

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

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

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

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying

from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

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

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

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

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

15. Disclaimer of Warranty.

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

16. Limitation of Liability.

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

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

17. Interpretation of Sections 15 and 16.

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

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

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

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

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

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

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

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

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

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

    <program> Copyright (C) <year> <name of author>
    This program comes with ABSOLUTELY NO WARRANTY; for details type `show w↵
        ↳ '.
    This is free software, and you are welcome to redistribute it

```
        under certain conditions; type ‘show c’ for details.

The hypothetical commands \‘show w’ and \‘show c’ should show the
appropriate parts of the General Public License. Of course, your
program’s commands might be different; for a GUI interface, you would
use an ”about box”.

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

The GNU General Public License does not permit incorporating your
program into proprietary programs. If your program is a subroutine
library, you may consider it more useful to permit linking proprietary
applications with the library. If this is what you want to do, use the
GNU Lesser General Public License instead of this License. But first,
please read <https://www.gnu.org/licenses/why-not-lgpl.html>.
```

41 README.md

```
# Raymond

A physics-inspired ray tracer for Fragmentarium.

## Quick Start

Install git, install Fragmentarium.

Get the source code:

    mkdir -p ~/opt/src
    cd ~/opt/src
    git clone https://code.mathr.co.uk/raymond.git

Configure Fragmentarium: menu Edit Preferences, set Include Path to

    Examples/Include;/home/$USER/opt/src/raymond/include

replacing ‘$USER‘ with your username.

Load ‘~/opt/src/raymond/examples/Balls.frag‘ to see some shiny.

## Legal

Raymond - a physics-inspired ray tracer for Fragmentarium

Copyright (C) 2018  Claude Heiland-Allen <claude@mathr.co.uk>

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

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

```
    GNU General Public License for more details.

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