trudy

Claude Heiland-Allen

2019

Contents

1 examples/MuET.frag

```
#version 400 compatibility

#info MuET (c) Claude Heiland-Allen 2019, license AGPL3+

#include "Trudy-sRGB.frag"
#include "Trudy.frag"

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

// http://lolengine.net/blog/2013/07/27/rgb-to-hsv-in-glsl
vec3 hsv2rgb(vec3 c)
{
    vec4 K = vec4(1.0, 2.0 / 3.0, 1.0 / 3.0, 3.0);
    vec3 p = abs(fract(c.xxx + K.xyz) * 6.0 - K.www);
    return c.z * mix(K.xxx, clamp(p - K.xxx, 0.0, 1.0), c.y);
}

#group MuET

uniform int Iterations; slider[0,100,100000]
uniform float EscapeRadius; slider[0.0,16,100.0]
uniform int Power; slider[2,2,16]
#if __VERSION__ >= 400
uniform dvec2 Seed; slider[(-2,-2),(0,0),(2,2)]
#else
uniform vec2 Seed; slider[(-2,-2),(0,0),(2,2)]
#endif

uniform float time;

#group Colour

uniform float LightDirection; slider[-360,45,360]

#group Formula1
```

```
uniform bool Active1; checkbox[true]
uniform bool AbsX1; checkbox[false]
uniform bool AbsY1; checkbox[false]
uniform bool NegX1; checkbox[false]
uniform bool NegY1; checkbox[false]
uniform bool Swap1; checkbox[false]

#group Formula2

uniform bool Active2; checkbox[false]
uniform bool AbsX2; checkbox[false]
uniform bool AbsY2; checkbox[false]
uniform bool NegX2; checkbox[false]
uniform bool NegY2; checkbox[false]
uniform bool Swap2; checkbox[false]

#group Formula3

uniform bool Active3; checkbox[false]
uniform bool AbsX3; checkbox[false]
uniform bool AbsY3; checkbox[false]
uniform bool NegX3; checkbox[false]
uniform bool NegY3; checkbox[false]
uniform bool Swap3; checkbox[false]

#group Formula4

uniform bool Active4; checkbox[false]
uniform bool AbsX4; checkbox[false]
uniform bool AbsY4; checkbox[false]
uniform bool NegX4; checkbox[false]
uniform bool NegY4; checkbox[false]
uniform bool Swap4; checkbox[false]

float EscapeRadius2 = pow(2.0, float(EscapeRadius));
int NActive = int(Active1) + int(Active2) + int(Active3) + int(Active4);
int IterationsN = NActive == 0 ? 0 : Iterations / NActive;
int IterationsM = NActive * IterationsN;

vec2 cMul(vec2 a, vec2 b)
{
  return vec2(a.x * b.x - a.y * b.y, a.x * b.y + a.y * b.x);
}

vec2 cPow(vec2 a, int n)
{
  // assert(n >= 2);
  vec2 p = a;
  for (int m = 2; m <= n; ++m)
  {
    p = cMul(p, a);
  }
  return p;
}

float dwell(vec2 c)
```

```
{
  int i = 0;
  vec2 z = vec2(Seed);
  for (; i < IterationsM;)
  {
    if (Active1)
    {
      if (AbsX1) z.x = abs(z.x);
      if (AbsY1) z.y = abs(z.y);
      if (NegX1) z.x = -z.x;
      if (NegY1) z.y = -z.y;
      if (Swap1) z.xy = z.yx;
      z = cPow(z, Power) + c;
      ++i;
      if (! (dot(z, z) < EscapeRadius2)) break;
    }
    if (Active2)
    {
      if (AbsX2) z.x = abs(z.x);
      if (AbsY2) z.y = abs(z.y);
      if (NegX2) z.x = -z.x;
      if (NegY2) z.y = -z.y;
      if (Swap2) z.xy = z.yx;
      z = cPow(z, Power) + c;
      ++i;
      if (! (dot(z, z) < EscapeRadius2)) break;
    }
    if (Active3)
    {
      if (AbsX3) z.x = abs(z.x);
      if (AbsY3) z.y = abs(z.y);
      if (NegX3) z.x = -z.x;
      if (NegY3) z.y = -z.y;
      if (Swap3) z.xy = z.yx;
      z = cPow(z, Power) + c;
      ++i;
      if (! (dot(z, z) < EscapeRadius2)) break;
    }
    if (Active4)
    {
      if (AbsX4) z.x = abs(z.x);
      if (AbsY4) z.y = abs(z.y);
      if (NegX4) z.x = -z.x;
      if (NegY4) z.y = -z.y;
      if (Swap4) z.xy = z.yx;
      z = cPow(z, Power) + c;
      ++i;
      if (! (dot(z, z) < EscapeRadius2)) break;
    }
  }
  if (! (dot(z, z) < EscapeRadius2))
  {
    return float(i) + 1.0 - log(log(length(z))) / log(float(Power));
  }
  else
  {
    return 0.0;
```

```
150   }
    }

    vec3 hue(float d)
    {
155   float h = floor(fract((d - time) / 3.0) * 3.0);
      if (h == 0.0) return vec3(0.0, 1.0, 1.0);
      if (h == 1.0) return vec3(1.0, 0.0, 1.0);
      return vec3(1.0,1.0,0.0);
    }
160
    vec3 shade(float d00, float d01, float d10, float d11)
    {
      vec2 e = vec2(d00 - d11, d01 - d10);
      float de = 1.0 / (log(2.0) * length(e));
165   float slope = dot(normalize(e), normalize(vec2(cos(radians(LightDirection)), ↙
        ↳ sin(radians(LightDirection)))));
      if (0.0 == d00 * d01 * d10 * d11 || isinf(de) || isnan(de)) return vec3(0.0);
      vec3 h = hsv2rgb(vec3(degrees(atan(e.y, e.x)) / 360.0, clamp(slope, 0.0, 1.0),↙
        ↳ 1.0));
      return h * tanh(clamp(4.0 * de, 0.0, 4.0));
    }
170
    vec3 color(vec2 p, vec2 dx, vec2 dy)
    {
      dx *= 0.25;
      dy *= 0.25;
175   float d00 = dwell(p - dx - dy);
      float d01 = dwell(p - dx + dy);
      float d10 = dwell(p + dx - dy);
      float d11 = dwell(p + dx + dy);
      return shade(d00, d01, d10, d11);
180 }

    #if __VERSION__ >= 400

    dvec2 cMul(dvec2 a, dvec2 b)
185 {
      return dvec2(a.x * b.x - a.y * b.y, a.x * b.y + a.y * b.x);
    }

    dvec2 cPow(dvec2 a, int n)
190 {
      // assert(n >= 2);
      dvec2 p = a;
      for (int m = 2; m <= n; ++m)
      {
195     p = cMul(p, a);
      }
      return p;
    }

200 float dwell(dvec2 c)
    {
      int i = 0;
      dvec2 z = dvec2(Seed);
      for (; i < IterationsM;)
```

```
205   {
        if (Active1)
        {
          if (AbsX1) z.x = abs(z.x);
          if (AbsY1) z.y = abs(z.y);
210       if (NegX1) z.x = -z.x;
          if (NegY1) z.y = -z.y;
          if (Swap1) z.xy = z.yx;
          z = cPow(z, Power) + c;
          ++i;
215       if (! (dot(z, z) < EscapeRadius2)) break;
        }
        if (Active2)
        {
          if (AbsX2) z.x = abs(z.x);
220       if (AbsY2) z.y = abs(z.y);
          if (NegX2) z.x = -z.x;
          if (NegY2) z.y = -z.y;
          if (Swap2) z.xy = z.yx;
          z = cPow(z, Power) + c;
225       ++i;
          if (! (dot(z, z) < EscapeRadius2)) break;
        }
        if (Active3)
        {
230       if (AbsX3) z.x = abs(z.x);
          if (AbsY3) z.y = abs(z.y);
          if (NegX3) z.x = -z.x;
          if (NegY3) z.y = -z.y;
          if (Swap3) z.xy = z.yx;
235       z = cPow(z, Power) + c;
          ++i;
          if (! (dot(z, z) < EscapeRadius2)) break;
        }
        if (Active4)
240     {
          if (AbsX4) z.x = abs(z.x);
          if (AbsY4) z.y = abs(z.y);
          if (NegX4) z.x = -z.x;
          if (NegY4) z.y = -z.y;
245       if (Swap4) z.xy = z.yx;
          z = cPow(z, Power) + c;
          ++i;
          if (! (dot(z, z) < EscapeRadius2)) break;
        }
250   }
      if (! (dot(z, z) < EscapeRadius2))
      {
        return float(i) + 1.0 - log(log(float(length(z)))) / log(float(Power));
      }
255   else
      {
        return 0.0;
      }
    }
260
    vec3 color(dvec2 p, vec2 dx, vec2 dy)
```

```
{
  dx *= 0.25;
  dy *= 0.25;
  float d00 = dwell(p - dx - dy);
  float d01 = dwell(p - dx + dy);
  float d10 = dwell(p + dx - dy);
  float d11 = dwell(p + dx + dy);
  return shade(d00, d01, d10, d11);
}

#endif

#preset Mystery
Zoom = 123456789 Logarithmic
ZoomFactor = 0
EnableTransform = true
RotateAngle = 0
StretchAngle = 0
StretchAmount = 0
Center = -0.057941214880704258082004 19,0.81326829943642640918963 78
Jitter = 1
Iterations = 16384
EscapeRadius = 16
Power = 2
Seed = 0,0
LightDirection = 45
Active1 = true
AbsX1 = false
AbsY1 = false
NegX1 = false
NegY1 = false
Swap1 = false
Active2 = false
AbsX2 = false
AbsY2 = false
NegX2 = false
NegY2 = false
Swap2 = false
Active3 = false
AbsX3 = false
AbsY3 = false
NegX3 = false
NegY3 = false
Swap3 = false
Active4 = false
AbsX4 = false
AbsY4 = false
NegX4 = false
NegY4 = false
Swap4 = false
Exposure = 2.5
#endpreset
```

2 include/Trudy-Camera2D.frag

```
#donotrun
```

```
#vertex

out vec2 coord;
out vec2 viewCoord;

#group Camera

uniform float Zoom; slider[1e-4,1,1e16] Logarithmic NotLockable
uniform float ZoomFactor; slider[-100,0,100]
uniform vec2 pixelSize;

uniform bool EnableTransform; checkbox[true]
uniform float RotateAngle; slider[-360,0,360]
uniform float StretchAngle; slider[-360,0,360]
uniform float StretchAmount; slider[-100,0,100]

void main(void)
{
  mat2 transform = mat2(1.0, 0.0, 0.0, 1.0);
  if (EnableTransform)
  {
    float b = radians(RotateAngle);
    float bc = cos(b);
    float bs = sin(b);
    float a = radians(StretchAngle);
    float ac = cos(a);
    float as = sin(a);
    float s = sqrt(pow(2.0, StretchAmount));
    mat2 m1 = mat2(ac, as, -as, ac);
    mat2 m2 = mat2(s, 0.0, 0.0, 1.0 / s);
    mat2 m3 = mat2(ac, -as, as, ac);
    mat2 m4 = mat2(bc, bs, -bs, bc);
    transform = m1 * m2 * m3 * m4;
  }
  float ar = float(pixelSize.y / pixelSize.x);
  coord = transform * ((gl_ProjectionMatrix * gl_Vertex).xy * vec2(ar, 1.0)) / (↙
      ↳ Zoom * pow(2.0, ZoomFactor));
  viewCoord = gl_Vertex.xy;
  gl_Position =  gl_Vertex;
}

#endvertex

in vec2 coord;
in vec2 viewCoord;

out vec4 fragColor;

#if __VERSION__ >= 400
uniform dvec2 Center; slider[(-100,-100),(0,0),(100,100)] NotLockable
#else
uniform vec2 Center; slider[(-100,-100),(0,0),(100,100)] NotLockable
#endif
uniform float Jitter; slider[0,1,10]
uniform vec2 pixelSize;
uniform int subframe;
uniform sampler2D backbuffer;
```

```
60  // implement these
    vec3 color(vec2 p, vec2 dx, vec2 dy);
    #if __VERSION__ >= 400
    vec3 color(dvec2 p, vec2 dx, vec2 dy);
    #endif
65
    void main()
    {
      vec2 dx = dFdx(coord);
      vec2 dy = dFdy(coord);
70    vec2 j = float(Jitter) * vec2
        ( uniform01(hash(vec4(coord, float(subframe), 1.0)))
        , uniform01(hash(vec4(coord, float(subframe), 2.0)))
        );
      vec2 p = coord + j.x * dx + j.y * dy;
75    const float eps = 1.1920928955078125e-07 * 4.0;
      bool double_precision = min(length(dx), length(dy)) < eps && __VERSION__ >= ↙
          ↳ 400;
      vec4 next;
      if (double_precision) next = vec4(color(p + Center, dx, dy), 1.0);
      else                  next = vec4(color(p + vec2(Center), dx, dy), 1.0);
80    vec4 prev = texture(backbuffer, vec2(viewCoord + vec2(1.0)) * 0.5);
      if (! (next == next)) // NaN check
      {
        next = vec4(0.0);
      }
85    fragColor = vec4(prev + vec4(next.xyz * next.w, next.w));
    }
```

3 include/Trudy.frag

```
#donotrun

#include "Trudy-Hash.frag"
#include "Trudy-Camera2D.frag"
```

4 include/Trudy-Hash.frag

```
#donotrun

/*
Trudy - a two-dimensional framework for Fragmentarium
5 Copyright (C) 2019  Claude Heiland-Allen
License GPL3+ <http://www.gnu.org/licenses/>
*/

// implement this
10
uint hash(uint a);

// scaling from uint to [0,1)

15 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;
}
```

5 include/Trudy-sRGB-Buffer.frag

```
#donotrun

/*
Trudy - a two-dimensional framework for Fragmentarium
Copyright (C) 2019  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[-10.0,0.0,10.0]
```

```
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(pow(2.0, Exposure) * tex.xyz / tex.a);
  gl_FragColor = vec4(c, 1.0);
}
```

6 include/Trudy-sRGB.frag

```
#donotrun

/*
Trudy - a two-dimensional framework for Fragmentarium
Copyright (C) 2019  Claude Heiland-Allen
License GPL3+ <http://www.gnu.org/licenses/>
*/

// accumulate linear RGBW and output as sRGB with exposure control
#buffer RGBA32F
#buffershader "Trudy-sRGB-Buffer.frag"

// 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));
}
```