ruff-render-gl

Claude Heiland-Allen

2011–2015

Contents

1 .gitignore

```
-
GL
glf
gli
glm
src/glm
src/ruff-render-gl-fp32
src/ruff-render-gl-fp64
```

2 README

```
git clone git://ogl-samples.git.sourceforge.net/gitroot/ogl-samples/ogl-samples
git clone http://code.mathr.co.uk/ruff-render-gl.git
cd ruff-render-gl
ln -s ../ogl-samples/external/glew-1.7.0/include/GL/
ln -s ../ogl-samples/common/glf/
ln -s ../ogl-samples/external/gli-0.3.1.0/gli/
ln -s ../ogl-samples/external/glm-0.9.2.4/glm/
cd src
mkdir -p glm/gtc
touch glm/gtc/random.hpp
make
# 32bit float (OpenGL 3.3)
```

```
./ruff-gl-render-fp32
# 64bit float (OpenGL 4.0)
./ruff-gl-render-fp64
```

3 src/fp32/colour.frag

```
#version 330 core
precision highp float;

#define POSITION        0
#define COLOR           3
#define FRAG_COLOR      0

in block {
  vec4 Color;
} In;

layout(location = FRAG_COLOR, index = 0) out vec4 Color;

void main() {
 Color = In.Color;
}
```

4 src/fp32/colour.vert

```
#version 330 core
precision highp float;

#define POSITION        0
#define COLOR           3
#define FRAG_COLOR      0

uniform mat4 MVP;

uniform vec3 interior;
uniform vec3 border;
uniform vec3 exterior;

layout(location = POSITION) in vec4 Position;
layout(location = COLOR) in vec4 Color;

out block {
  vec4 Color;
} Out;

void main() {
  float n = Color.x;
  float d = Color.y;
//float a = Color.z;
  vec4 c = vec4(interior, 1.0);
  if (n > 0.0) {
    float k = clamp(0.5 + 0.5 * log2(0.5 * d), 0.0, 1.0);
    c.rgb = mix(border, exterior, k);
  }
  gl_Position = MVP * Position;
  Out.Color = c;
```

```
}
```

5 src/fp32/cook.vert

```
#version 330 core
precision highp float;

#define POSITION        0
#define COLOR           3
#define FRAG_COLOR      0

layout(location = POSITION) in vec4 Position;
layout(location = COLOR) in vec4 Color;

out block {
        vec4 Color;
        vec4 Position;
} Out;

const float er2 = 65536.0;

uniform float iters;
uniform float scale;

void main() {
        vec2 c = Position.xy;
        vec2 z = Color.xy;
        vec2 dz = Color.zw;
        float zx2 = z.x * z.x;
        float zy2 = z.y * z.y;
        float z2 = zx2 + zy2;
        float z2xy = 2.0 * z.x * z.y;
        float dz2 = dz.x * dz.x + dz.y * dz.y;
  float n = 1.0 + iters - log2(log(z2) / log(er2));
  float d = log(z2) * sqrt(z2 / dz2) / scale;
  float a = atan(z.y, z.x);
        Out.Position = Position;
        Out.Color = vec4(n, d, a, 1.0);
}
```

6 src/fp32/escape.geom

```
#version 330 core
#extension GL_ARB_separate_shader_objects : enable
precision highp float;

#define POSITION        0
#define COLOR           3
#define FRAG_COLOR      0

layout(points) in;
layout(points, max_vertices = 1) out;

layout(location = POSITION) in vec4 OutPosition[1];
layout(location = COLOR) in vec4 OutColor[1];
```

```
uniform bool escape;

out block {
        vec4 Position;
        vec4 Color;
} Out;

const float er2 = 4.0;

void main() {
        vec2 z = OutColor[0].xy;
        float zx2 = z.x * z.x;
        float zy2 = z.y * z.y;
        float z2 = zx2 + zy2;
  if (escape == !(z2 < er2)) {
                Out.Position = OutPosition[0];
                Out.Color = OutColor[0];
                EmitVertex();
                EndPrimitive();
        }
}
```

7 src/fp32/escape.vert

```
#version 330 core
precision highp float;

#define POSITION        0
#define COLOR           3
#define FRAG_COLOR      0

layout(location = POSITION) in vec4 Position;
layout(location = COLOR) in vec4 Color;

out vec4 OutPosition;
out vec4 OutColor;

void main() {
  OutPosition = Position;
  OutColor = Color;
}
```

8 src/fp32/iterate.vert

```
#version 330 core

#define POSITION        0
#define COLOR           3
#define FRAG_COLOR      0

layout(location = POSITION) in vec4 Position;
layout(location = COLOR) in vec4 Color;

out block {
        vec4 Color;
        vec4 Position;
```

```
} Out;

const float er2 = 65536.0;

void main() {
        vec2 c = Position.xy;
        vec2 z = Color.xy;
        vec2 d = Color.zw;
        float zx2 = z.x * z.x;
        float zy2 = z.y * z.y;
        float z2 = zx2 + zy2;
        float z2xy = 2.0 * z.x * z.y;
//      if (z2 < er2) {
        float zdzx = z.x * d.x - z.y * d.y;
        float zdzy = z.x * d.y + z.y * d.x;
        float dx = 2.0 * zdzx + 1.0;
        float dy = 2.0 * zdzy;
  float zx = zx2 - zy2 + c.x;
  float zy = z2xy + c.y;
        Out.Position = Position;
        Out.Color = vec4(zx, zy, dx, dy);
}
```

9 src/fp64/colour.frag

```
#version 400 core
#extension GL_ARB_vertex_attrib_64bit : enable

#define POSITION        0
#define COLOR           3
#define FRAG_COLOR      0

in block {
  vec4 Color;
} In;

layout(location = FRAG_COLOR, index = 0) out vec4 Color;

void main() {
 Color = vec4(In.Color);
}
```

10 src/fp64/colour.vert

```
#version 400 core
#extension GL_ARB_vertex_attrib_64bit : enable

#define POSITION        0
#define COLOR           3
#define FRAG_COLOR      0

uniform dmat4 MVP;

uniform vec3 interior;
uniform vec3 border;
uniform vec3 exterior;
```

```
layout(location = POSITION) in dvec4 Position;
layout(location = COLOR) in dvec4 Color;

out block {
  vec4 Color;
} Out;

void main() {
  float n = float(Color.x);
  float d = float(Color.y);
//double a = Color.z;
  vec4 c = vec4(interior, 1.0);
  if (n > 0.0) {
    float k = clamp(0.5 + 0.5 * log2(0.5 * d), 0.0, 1.0);
    c.rgb = mix(border, exterior, k);
  }
  gl_Position = vec4(MVP * Position);
  Out.Color = c;
}
```

11 src/fp64/cook.vert

```
#version 400 core
#extension GL_ARB_vertex_attrib_64bit : enable

#define POSITION        0
#define COLOR           3
#define FRAG_COLOR      0

layout(location = POSITION) in dvec4 Position;
layout(location = COLOR) in dvec4 Color;

out block {
        dvec4 Color;
        dvec4 Position;
} Out;

const float er2 = 65536.0;

uniform double iters;
uniform double scale;

void main() {
        dvec2 c = Position.xy;
        dvec2 z = Color.xy;
        dvec2 dz = Color.zw;
        double zx2 = z.x * z.x;
        double zy2 = z.y * z.y;
        double z2 = zx2 + zy2;
        double z2xy = 2.0 * z.x * z.y;
        double dz2 = dz.x * dz.x + dz.y * dz.y;
  double n = 1.0 + iters - double(log2(log(float(z2)) / log(er2)));
  double d = double(log(float(z2))) * sqrt(z2 / dz2) / scale;
  double a = double(atan(float(z.y), float(z.x)));
        Out.Position = Position;
        Out.Color = dvec4(n, d, a, 1.0);
```

```
}
```

12 src/fp64/escape.geom

```
#version 400 core
#extension GL_ARB_separate_shader_objects : enable
#extension GL_ARB_vertex_attrib_64bit : enable

#define POSITION        0
#define COLOR           3
#define FRAG_COLOR      0

layout(points) in;
layout(points, max_vertices = 1) out;

layout(location = POSITION) in dvec4 OutPosition[1];
layout(location = COLOR) in dvec4 OutColor[1];

uniform bool escape;

out block {
        dvec4 Position;
        dvec4 Color;
} Out;

const double er2 = 4.0;

void main() {
        dvec2 z = OutColor[0].xy;
        double zx2 = z.x * z.x;
        double zy2 = z.y * z.y;
        double z2 = zx2 + zy2;
  if (escape == !(z2 < er2)) {
                Out.Position = OutPosition[0];
                Out.Color = OutColor[0];
                EmitVertex();
                EndPrimitive();
        }
}
```

13 src/fp64/escape.vert

```
#version 400 core
#extension GL_ARB_vertex_attrib_64bit : enable
#extension GL_NV_gpu_shader5 : enable

#define POSITION        0
#define COLOR           3
#define FRAG_COLOR      0

layout(location = POSITION) in dvec4 Position;
layout(location = COLOR) in dvec4 Color;

out dvec4 OutPosition;
out dvec4 OutColor;
```

```
15  void main() {
      OutPosition = Position;
      OutColor = Color;
    }
```

14 src/fp64/iterate.vert

```
    #version 400 core
    #extension GL_ARB_vertex_attrib_64bit : enable

    #define POSITION        0
 5  #define COLOR           3
    #define FRAG_COLOR      0

    layout(location = POSITION) in dvec4 Position;
    layout(location = COLOR) in dvec4 Color;
10
    out block {
            dvec4 Color;
            dvec4 Position;
    } Out;
15
    const double er2 = 65536.0;

    void main() {
            dvec2 c = Position.xy;
20          dvec2 z = Color.xy;
            dvec2 d = Color.zw;
            double zx2 = z.x * z.x;
            double zy2 = z.y * z.y;
            double z2 = zx2 + zy2;
25          double z2xy = 2.0 * z.x * z.y;
            double zdzx = z.x * d.x - z.y * d.y;
            double zdzy = z.x * d.y + z.y * d.x;
            double dx = 2.0 * zdzx + 1.0;
            double dy = 2.0 * zdzy;
30    double zx = zx2 - zy2 + c.x;
      double zy = z2xy + c.y;
            Out.Position = Position;
            Out.Color = dvec4(zx, zy, dx, dy);
    }
```

15 src/Makefile

```
    LIBS = -lGLEW -lglut
    CCFLAGS = -I. -I.. -Wall -pedantic -Wextra -O3 -march=native -ggdb

    all: ruff-render-gl-fp32 ruff-render-gl-fp64
 5
    ruff-render-gl-fp32: ruff-render-gl.cc
            g++ $(CCFLAGS) -DFP=32 -o ruff-render-gl-fp32 ruff-render-gl.cc $(LIBS)

    ruff-render-gl-fp64: ruff-render-gl.cc
10          g++ $(CCFLAGS) -DFP=64 -o ruff-render-gl-fp64 ruff-render-gl.cc $(LIBS)
```

16 src/ruff-render-gl.cc

```
/*
ruff-render-gl -- GPU accelerated tile renderer for ruff
(C) 2011 Claude Heiland-Allen <claude@mathr.co.uk>

based on:

// **********************************
// OpenGL Transform Feedback Separated
// 06/04/2010 - 22/06/2011
// **********************************
// Christophe Riccio
// ogl-samples@g-truc.net
// **********************************
// G-Truc Creation
// www.g-truc.net
// **********************************

*/

#include <glf/glf.hpp>

#ifndef FP
#define FP 64
#endif

#if (FP == 32)
#define SHADER_DIR "fp32/"
const int major = 3;
const int minor = 3;
typedef glm::vec4 vec4;
typedef glm::mat4 mat4;
const mat4 IDENTITY = mat4(1.0f);
typedef float flt;
#define glVertexAttribPointer_(a,b,c,d,e) glVertexAttribPointer(a,b,GL_FLOAT,c,d,e)
#define glUniformMatrix4fv_ glUniformMatrix4fv
#define glUniform1f_ glUniform1f
#else
#if (FP == 64)
#define SHADER_DIR "fp64/"
const int major = 4;
const int minor = 0;
typedef glm::dvec4 vec4;
typedef glm::dmat4 mat4;
const mat4 IDENTITY = mat4(1.0);
typedef double flt;
#define glVertexAttribPointer_(a,b,c,d,e) glVertexAttribLPointer(a,b,GL_DOUBLE,d,e)
#define glUniformMatrix4fv_ glUniformMatrix4dv
#define glUniform1f_ glUniform1d
#else
#error unknown FP
#endif
#endif
```

```
typedef vec4 point[2];

namespace
{
  std::string const SAMPLE_NAME = "OpenGL Mandelbrot Set ";
  std::string const VERT_SHADER_SOURCE_ITERATE(SHADER_DIR "iterate.vert");
  std::string const VERT_SHADER_SOURCE_ESCAPE (SHADER_DIR "escape.vert");
  std::string const GEOM_SHADER_SOURCE_ESCAPE (SHADER_DIR "escape.geom");
  std::string const VERT_SHADER_SOURCE_COOK   (SHADER_DIR "cook.vert");
  std::string const VERT_SHADER_SOURCE_COLOUR (SHADER_DIR "colour.vert");
  std::string const FRAG_SHADER_SOURCE_COLOUR (SHADER_DIR "colour.frag");

        int const SAMPLE_SIZE_WIDTH(256);
        int const SAMPLE_SIZE_HEIGHT(256);
        int const SAMPLE_MAJOR_VERSION(major);
        int const SAMPLE_MINOR_VERSION(minor);

        glf::window Window(glm::ivec2(SAMPLE_SIZE_WIDTH, SAMPLE_SIZE_HEIGHT));

        GLsizei const VertexCount(SAMPLE_SIZE_WIDTH * SAMPLE_SIZE_HEIGHT);
  GLuint iteratees = VertexCount;
  bool firstDisplay = true;
  int iterationcount = 0;
  flt cx = 0;
  flt cy = 0;
  flt r = 2;
  flt pixelspacing = flt(4) / SAMPLE_SIZE_WIDTH;

  GLuint IterateProgramName(0);
  GLuint IterateArrayBufferName(0);
  GLuint IterateVertexArrayName(0);

  GLuint EscapeProgramName(0);
  GLuint EscapeArrayBufferName(0);
  GLuint EscapeVertexArrayName(0);
  GLint  EscapeUniformEscape(0);

  GLuint CookProgramName(0);
  GLuint CookArrayBufferName(0);
  GLuint CookVertexArrayName(0);
  GLint  CookUniformIters(0);
  GLint  CookUniformScale(0);

  GLuint ColourProgramName(0);
  GLuint ColourArrayBufferName(0);
  GLuint ColourVertexArrayName(0);
  GLint  ColourUniformMVP(0);
  GLint  ColourUniformInterior(0);
  GLint  ColourUniformBorder(0);
  GLint  ColourUniformExterior(0);

  GLuint Query(0);

}//namespace

bool initProgram()
{
```

```
	bool Validated = true;

	// Create program 'iterate'
	//if(Validated)
	{
		GLuint VertexShaderName = glf::createShader(GL_VERTEX_SHADER, ↙
		    ↳ VERT_SHADER_SOURCE_ITERATE);

		IterateProgramName = glCreateProgram();
		glAttachShader(IterateProgramName, VertexShaderName);
		glDeleteShader(VertexShaderName);

		GLchar const * Strings[] = {"block.Position", "block.Color"};
		glTransformFeedbackVaryings(IterateProgramName, 2, Strings, ↙
		    ↳ GL_INTERLEAVED_ATTRIBS);
		glLinkProgram(IterateProgramName);

		Validated = Validated && glf::checkProgram(IterateProgramName);

/*
		// BUG AMD 10.12
		char Name[64];
		memset(Name, 0, 64);
		GLsizei Length(0);
		GLsizei Size(0);
		GLenum Type(0);

		glGetTransformFeedbackVarying(
			IterateProgramName,
			0,
			64,
			&Length,
			&Size,
			&Type,
			Name);

		Validated = Validated && (Size == 1) && (Type == GL_flt_VEC4);
*/
	}


	// Create program 'escape'
	//if(Validated)
	{
		GLuint VertexShaderName = glf::createShader(GL_VERTEX_SHADER, ↙
		    ↳ VERT_SHADER_SOURCE_ESCAPE);
		GLuint GeometryShaderName = glf::createShader(GL_GEOMETRY_SHADER↙
		    ↳ , GEOM_SHADER_SOURCE_ESCAPE);

		EscapeProgramName = glCreateProgram();
		glAttachShader(EscapeProgramName, VertexShaderName);
		glAttachShader(EscapeProgramName, GeometryShaderName);
		glDeleteShader(VertexShaderName);
		glDeleteShader(GeometryShaderName);

		GLchar const * Strings[] = {"block.Position", "block.Color"};
		glTransformFeedbackVaryings(EscapeProgramName, 2, Strings, ↙
```

```
                        ↪ GL_INTERLEAVED_ATTRIBS);
                glLinkProgram(EscapeProgramName);

                Validated = Validated && glf::checkProgram(EscapeProgramName);
/*
                // BUG AMD 10.12
                char Name[64];
                memset(Name, 0, 64);
                GLsizei Length(0);
                GLsizei Size(0);
                GLenum Type(0);

                glGetTransformFeedbackVarying(
                        EscapeProgramName,
                        0,
                        64,
                        &Length,
                        &Size,
                        &Type,
                        Name);

                Validated = Validated && (Size == 1) && (Type == GL_flt_VEC4);
*/
        }
        // Get variables locations
        //if(Validated)
        {
                EscapeUniformEscape = glGetUniformLocation(EscapeProgramName, "↙
                    ↪ escape");
                Validated = Validated && (EscapeUniformEscape >= 0);
        }


        // Create program 'cook'
        //if(Validated)
        {
                GLuint VertexShaderName = glf::createShader(GL_VERTEX_SHADER, ↙
                    ↪ VERT_SHADER_SOURCE_COOK);

                CookProgramName = glCreateProgram();
                glAttachShader(CookProgramName, VertexShaderName);
                glDeleteShader(VertexShaderName);

                GLchar const * Strings[] = {"block.Position", "block.Color"};
                glTransformFeedbackVaryings(CookProgramName, 2, Strings, ↙
                    ↪ GL_INTERLEAVED_ATTRIBS);
                glLinkProgram(CookProgramName);

                Validated = Validated && glf::checkProgram(CookProgramName);
/*
                // BUG AMD 10.12
                char Name[64];
                memset(Name, 0, 64);
                GLsizei Length(0);
                GLsizei Size(0);
                GLenum Type(0);
```

```
                glGetTransformFeedbackVarying(
                        CookProgramName,
                        0,
220                     64,
                        &Length,
                        &Size,
                        &Type,
                        Name);
225
                Validated = Validated && (Size == 1) && (Type == GL_flt_VEC4);
*/
        }
        // Get variables locations
230     //if(Validated)
        {
                CookUniformIters = glGetUniformLocation(CookProgramName, "iters↙
                    ↳ ");
                CookUniformScale = glGetUniformLocation(CookProgramName, "scale↙
                    ↳ ");
                Validated = Validated && (CookUniformIters >= 0) && (↙
                    ↳ CookUniformScale >= 0);
235     }


  // Create program 'colour'
        //if(Validated)
240     {
                GLuint VertexShaderName = glf::createShader(GL_VERTEX_SHADER, ↙
                    ↳ VERT_SHADER_SOURCE_COLOUR);
                GLuint FragmentShaderName = glf::createShader(GL_FRAGMENT_SHADER↙
                    ↳ , FRAG_SHADER_SOURCE_COLOUR);

                ColourProgramName = glCreateProgram();
245             glAttachShader(ColourProgramName, VertexShaderName);
                glAttachShader(ColourProgramName, FragmentShaderName);
                glDeleteShader(VertexShaderName);
                glDeleteShader(FragmentShaderName);

250             glLinkProgram(ColourProgramName);
                Validated = Validated && glf::checkProgram(ColourProgramName);
        }
        // Get variables locations
        if(Validated)
255     {
                ColourUniformMVP = glGetUniformLocation(ColourProgramName, "MVP↙
                    ↳ ");
                ColourUniformInterior = glGetUniformLocation(ColourProgramName, ↙
                    ↳ "interior");
                ColourUniformBorder = glGetUniformLocation(ColourProgramName, "↙
                    ↳ border");
                ColourUniformExterior = glGetUniformLocation(ColourProgramName, ↙
                    ↳ "exterior");
260
                Validated = Validated && (ColourUniformMVP >= 0) && (↙
                    ↳ ColourUniformInterior >= 0) && (ColourUniformBorder >= 0) ↙
                    ↳ && (ColourUniformExterior >= 0);
        }
```

```
        return Validated && glf::checkError("initProgram");
}

void initVertexArray1(GLuint *name, GLuint arr, const char *err) {
glf::checkError("iva 1");
        glGenVertexArrays(1, name);
glf::checkError("iva 2");
    glBindVertexArray(*name);
glf::checkError("iva 3");
                glBindBuffer(GL_ARRAY_BUFFER, arr);
glf::checkError("iva 4");
                glVertexAttribPointer_(glf::semantic::attr::POSITION, 4, ↙
                    ↳ GL_FALSE, sizeof(point), 0);
glf::checkError("iva 5");
                glBindBuffer(GL_ARRAY_BUFFER, arr);
glf::checkError("iva 6");
                glVertexAttribPointer_(glf::semantic::attr::COLOR, 4, GL_FALSE, ↙
                    ↳ sizeof(point), GLF_BUFFER_OFFSET(sizeof(vec4)));
glf::checkError("iva 6");
                glBindBuffer(GL_ARRAY_BUFFER, 0);
glf::checkError("iva 8");
                glEnableVertexAttribArray(glf::semantic::attr::POSITION);
glf::checkError("iva 9");
                glEnableVertexAttribArray(glf::semantic::attr::COLOR);
glf::checkError("iva A");
        glBindVertexArray(0);
glf::checkError("iva B");
        glf::checkError(err);
}

bool initVertexArray() {
  glf::checkError("initVertexArray 0");
  initVertexArray1(&IterateVertexArrayName, IterateArrayBufferName, "iva iterate↙
      ↳ ");
  initVertexArray1(&EscapeVertexArrayName , EscapeArrayBufferName , "iva escape↙
      ↳ ");
  initVertexArray1(&CookVertexArrayName   , CookArrayBufferName   , "iva cook");
  initVertexArray1(&ColourVertexArrayName , ColourArrayBufferName , "iva colour↙
      ↳ ");
  return glf::checkError("initVertexArray");
}

void initArrayBuffer1(GLuint *name, const char *err, void *data) {
        glGenBuffers(1, name);
  glBindBuffer(GL_ARRAY_BUFFER, *name);
    glBufferData(GL_ARRAY_BUFFER, sizeof(point) * VertexCount, data, ↙
        ↳ GL_DYNAMIC_COPY);
        glBindBuffer(GL_ARRAY_BUFFER, 0);
  glf::checkError(err);
}

void resetView() {
  point *positions = (point *) calloc(1, sizeof(point) * VertexCount);
  point *p = positions;
  for (int i = 0; i < SAMPLE_SIZE_WIDTH; ++i) {
    flt x = cx + (i - SAMPLE_SIZE_WIDTH/2) * r / (SAMPLE_SIZE_WIDTH/2);
```

```
      for (int j = 0; j < SAMPLE_SIZE_HEIGHT; ++j) {
315     flt y = cy + (j - SAMPLE_SIZE_HEIGHT/2) * r / (SAMPLE_SIZE_HEIGHT/2);
        *p[0] = vec4(x, y, flt(0), flt(1));
        p++;
      }
    }
320 glf::checkError("view 1");
    glBindBuffer(GL_ARRAY_BUFFER, IterateArrayBufferName);
glf::checkError("view 2");
      glBufferSubData(GL_ARRAY_BUFFER, 0, sizeof(point) * VertexCount, positions);
glf::checkError("view 3");
325   glBindBuffer(GL_ARRAY_BUFFER, 0);
glf::checkError("view 4");
    free(positions);
    iterationcount = 0;
    iteratees = VertexCount;
330 }

bool initArrayBuffer()
{
    glf::checkError("initArrayBuffer 0");
335 void *zero = calloc(1, sizeof(point) * VertexCount);
        initArrayBuffer1(&IterateArrayBufferName, "iab iterate", zero);
    free(zero);
        initArrayBuffer1(&EscapeArrayBufferName, "iab escape", NULL);
        initArrayBuffer1(&CookArrayBufferName, "iab cook", NULL);
340     initArrayBuffer1(&ColourArrayBufferName, "iab colour", NULL);
        return glf::checkError("initArrayBuffer");
}

void click(int button, int state, int x, int y) {
345   if (!state) {
      cx += (x - SAMPLE_SIZE_WIDTH /2) * r / (SAMPLE_SIZE_WIDTH /2);
      cy += (SAMPLE_SIZE_HEIGHT/2 - y) * r / (SAMPLE_SIZE_HEIGHT/2);
      if (!button) {
        r  /= 2;
350     pixelspacing /= 2;
      } else {
        r  *= 2;
        pixelspacing *= 2;
      }
355   resetView();
    }
}

bool begin()
360 {
        bool Validated = true;
        Validated = Validated && glf::checkGLVersion(SAMPLE_MAJOR_VERSION, ↙
            ↳ SAMPLE_MINOR_VERSION);
        Validated = Validated && glf::checkExtension("GL_ARB_viewport_array");
        Validated = Validated && glf::checkExtension("↙
            ↳ GL_ARB_separate_shader_objects");
365

/*
  int sep = 0;
  int ilv = 0;
```

```
  glGetIntegerv(GL_MAX_TRANSFORM_FEEDBACK_SEPARATE_COMPONENTS, &sep);
  glGetIntegerv(GL_MAX_TRANSFORM_FEEDBACK_INTERLEAVED_COMPONENTS, &ilv);
  printf("%d %d\n", sep, ilv);
*/

  glClampColor(GL_CLAMP_VERTEX_COLOR, GL_FALSE);
  glClampColor(GL_CLAMP_READ_COLOR, GL_FALSE);
  glClampColor(GL_CLAMP_FRAGMENT_COLOR, GL_FALSE);

glf::checkError("begin 1");
        glGenQueries(1, &Query);
glf::checkError("begin 2");
        if(Validated)
                Validated = initProgram();
glf::checkError("begin 3");
        if(Validated)
                Validated = initArrayBuffer();
glf::checkError("begin 4");
        if(Validated)
                Validated = initVertexArray();
glf::checkError("begin 5");

  resetView();

        return Validated && glf::checkError("begin");
}

bool end()
{
/* // FIXME
        glDeleteVertexArrays(1, &TransformVertexArrayName);
        glDeleteBuffers(1, &TransformArrayBufferName);
        glDeleteProgram(TransformProgramName);

        glDeleteVertexArrays(1, &FeedbackVertexArrayName);
        glDeleteBuffers(1, &FeedbackArrayBufferPositionName);
        glDeleteBuffers(1, &FeedbackArrayBufferColorName);
        glDeleteProgram(FeedbackProgramName);
*/
        glDeleteQueries(1, &Query);

        return glf::checkError("end");
}

void timer1(int v) {
  glutPostRedisplay();
}

void display()
{
for (int repeats = 0; repeats < 10; ++repeats) {

        // Compute the MVP (Model View Projection matrix)
        mat4 Projection = glm::ortho(cx - r, cx + r, cy - r, cy + r);
        mat4 View = IDENTITY;
        mat4 Model = IDENTITY;
        mat4 MVP = Projection * View * Model;
```

```
        // Set the display viewport
        glViewport(0, 0, Window.Size.x, Window.Size.y);

430   if (iterationcount == 0) {
          // Clear color buffer with red
        glClearColor(1.0f, 0.0f, 0.0f, 1.0f);
        glClear(GL_COLOR_BUFFER_BIT);
        glf::checkError("clear");
435   }

      // Disable rasterisation, vertices processing only!
        glEnable(GL_RASTERIZER_DISCARD);

440     // iterate
        {
    glUseProgram(IterateProgramName);
    // dst
/*
445 void glBindBufferRange( GLenum  target,
        GLuint  index,
        GLuint  buffer,
        GLintptr        offset,
        GLsizeiptr      size);
450 */
                glBindBufferBase(GL_TRANSFORM_FEEDBACK_BUFFER, 0, ↙
                    ↳ EscapeArrayBufferName);
//              glBindBufferRange(GL_TRANSFORM_FEEDBACK_BUFFER, 1, ↙
    ↳ EscapeArrayBufferName, sizeof(vec4), sizeof(point) * VertexCount);
//              glBindBufferBase(GL_TRANSFORM_FEEDBACK_BUFFER, 1, ↙
    ↳ EscapeArrayBufferColorName);
    // src
455             glBindVertexArray(IterateVertexArrayName);
    // do it
                glBeginTransformFeedback(GL_POINTS);
                glDrawArrays(GL_POINTS, 0, iteratees);
                glEndTransformFeedback();
460             glBindBufferBase(GL_TRANSFORM_FEEDBACK_BUFFER, 0, 0);
//              glBindBufferBase(GL_TRANSFORM_FEEDBACK_BUFFER, 1, 0);
                glBindVertexArray(0);
        glf::checkError("iterate");
  }
465   // keep unescaped
  GLuint unescaped = 0;
        {
    glUseProgram(EscapeProgramName);
        glf::checkError("unescaped 1");
470   glUniform1i(EscapeUniformEscape, 0);
        glf::checkError("unescaped 2");
    // dst
                glBindBufferBase(GL_TRANSFORM_FEEDBACK_BUFFER, 0, ↙
                    ↳ IterateArrayBufferName);
        glf::checkError("unescaped 3");
475 //              glBindBufferBase(GL_TRANSFORM_FEEDBACK_BUFFER, 1, ↙
    ↳ IterateArrayBufferColorName);
        glf::checkError("unescaped 4");
    // src
```

```
                glBindVertexArray(EscapeVertexArrayName);
        glf::checkError("unescaped 5");
480     // do it
                glBeginQuery(GL_TRANSFORM_FEEDBACK_PRIMITIVES_WRITTEN, Query);
        glf::checkError("unescaped 6");
                glBeginTransformFeedback(GL_POINTS);
        glf::checkError("unescaped 7");
485                     glDrawArrays(GL_POINTS, 0, iteratees);
        glf::checkError("unescaped 8");
                glEndTransformFeedback();
        glf::checkError("unescaped 9");
                glEndQuery(GL_TRANSFORM_FEEDBACK_PRIMITIVES_WRITTEN);
490     glf::checkError("unescaped A");
                glGetQueryObjectuiv(Query, GL_QUERY_RESULT, &unescaped);
        glf::checkError("unescaped");
    }
    // pass through escaped
495 GLuint escaped = 0;
        {
      glUseProgram(EscapeProgramName);
      glUniform1i(EscapeUniformEscape, 1);
      // dst
500               glBindBufferBase(GL_TRANSFORM_FEEDBACK_BUFFER, 0, ↙
                      ↳ CookArrayBufferName);
//                glBindBufferBase(GL_TRANSFORM_FEEDBACK_BUFFER, 1, ↙
    ↳ CookArrayBufferColorName);
      // src
                glBindVertexArray(EscapeVertexArrayName);
      // do it
505               glBeginQuery(GL_TRANSFORM_FEEDBACK_PRIMITIVES_WRITTEN, Query);
                glBeginTransformFeedback(GL_POINTS);
                        glDrawArrays(GL_POINTS, 0, iteratees);
                glEndTransformFeedback();
                glEndQuery(GL_TRANSFORM_FEEDBACK_PRIMITIVES_WRITTEN);
510             glGetQueryObjectuiv(Query, GL_QUERY_RESULT, &escaped);
        glf::checkError("escaped");
    }
    // cook the escapees
        {
515   glUseProgram(CookProgramName);
      glUniform1f_(CookUniformIters, ++iterationcount);
      glUniform1f_(CookUniformScale, pixelspacing);
      // dst
                glBindBufferBase(GL_TRANSFORM_FEEDBACK_BUFFER, 0, ↙
                      ↳ ColourArrayBufferName);
520 //              glBindBufferBase(GL_TRANSFORM_FEEDBACK_BUFFER, 1, ↙
    ↳ ColourArrayBufferColorName);
      // src
                glBindVertexArray(CookVertexArrayName);
      // do it
                glBeginTransformFeedback(GL_POINTS);
525                     glDrawArrays(GL_POINTS, 0, escaped);
                glEndTransformFeedback();
        glf::checkError("cook");
    }

530 // re-enable drawing
```

```
	glDisable(GL_RASTERIZER_DISCARD);

	// colourize the cookeds
		{
535			glUseProgram(ColourProgramName);
			glUniformMatrix4fv_(ColourUniformMVP, 1, GL_FALSE, &MVP[0][0]);
			glUniform3f(ColourUniformInterior, 1.0, 0.0, 0.0);
			glUniform3f(ColourUniformBorder  , 0.0, 0.0, 0.0);
			glUniform3f(ColourUniformExterior, 1.0, 1.0, 1.0);
540			glBindVertexArray(ColourVertexArrayName);
			glDrawArrays(GL_POINTS, 0, escaped);
		glf::checkError("colour");
		}
	if ((iterationcount % 1000) == 0) {
545	  printf("%d\t%d\t%d\t%d\n", iterationcount, iteratees, escaped, unescaped);
	}
	iteratees = unescaped;

		glf::checkError("display");
550	if ((iterationcount % 100) == 0) {
		glf::swapBuffers();
	}
}
	glutTimerFunc(1, timer1, 1);
555	glutIdleFunc(0);
	glutMouseFunc(click);
}

int main(int argc, char* argv[])
560 {
		return glf::run(
			argc, argv,
			glm::ivec2(::SAMPLE_SIZE_WIDTH, ::SAMPLE_SIZE_HEIGHT),
			::SAMPLE_MAJOR_VERSION,
565			::SAMPLE_MINOR_VERSION);
}
```