Effects Framework for OpenGL Testing

Ian Romanick <ian.d.romanick@intel.com>
23-September-2013
Agenda

- Why?
- What is an “effect” anyway?
- nvFX
- Is nvFS useful?
Why?
Why?

uniform mat4x2 arg0;
uniform mat4x2 arg1;
uniform float tolerance;
uniform mat4x2 expected;

void main()
{
    mat4x2 result = matrixCompMult(arg0, arg1);
    mat4x2 residual = result - expected;
    gl_FragColor = error_sq <= tolerance * tolerance
        ? vec4(0.0, 1.0, 0.0, 1.0) : vec4(1.0, 0.0, 0.0, 1.0);
}
#version 130
in vec3 normal_es, position_es;
out vec4 color;
uniform vec3 light_es = vec3(0.0, 15.0, 4.0);
uniform float m = 0.2;
uniform float ri = 1.5;
uniform vec3 color_s = vec3(1.0);
uniform vec3 color_d = vec3(1.0, 0.0, 0.0);

float schlick(float ni, float cos_theta)
{
    float c = 1.0 - cos_theta;
    float r0 = (ni - 1.0) / (ni + 1.0);
    r0 = r0 * r0;
    return r0 + (1.0 - r0) * pow(c, 5.0);
}

float G(float n_dot_l, float n_dot_h,
        float n_dot_v, float v_dot_h)
{
    float c = 2.0 * n_dot_h / v_dot_h;
    return min(1.0, c * min(n_dot_v,
                            n_dot_l));
}

float beckmann(float m, float cos_theta)
{
    float c2 = cos_theta * cos_theta;
    m = max(m, 1e-6);
    float m_c2 = m * c2;
    return exp((c2 - 1.0) / (m * m_c2))
          / (4.0 * m_c2 * m_c2);
}

void main(void)
{
    vec3 l = normalize(light_es - position_es);
    vec3 v = -normalize(position_es);
    vec3 n = normalize(normal_es);
    vec3 h = normalize(l + v);
    float ndl = dot(n, l);
    float ndh = dot(n, h);
    float ndv = dot(n, v);
    
    float f = schlick(ri, ndv) * beckmann(m, ndh)
                * G(ndl, ndh, ndv, dot(v, h)) / ndv;
    
    vec3 spec = f * color_s;
    vec3 diff = color_d * max(ndl, 0.);
    
    color = vec4((spec + diff), 1.0);
}
Why?

- shader_runner is piglit's mechanism for testing shaders
  - Really hard to draw anything other than a rectangle
  - Really hard to get additional per-vertex data to the shader
  - Really hard to use a non-trival texture
    - You can use any texture you want, as long as you only want checkerboard or RGBW
  - Really hard to set other GL state
  - Difficult to extend
    - The parser... gives me nightmares.
  - etc.
Why?

• Holy grail: Easily import shaders from real apps
  • shader_runner doesn't really help here
    • There are a couple shader tests like this
  • apitrace could help
    • Trace files tend to be quite large
    • Trace files are difficult to tweak
      • Want to modify a GL 3.3 test to run on GL ES 3.0
    • Trace files are difficult to create from scratch
      • Write an application, then trace it
What is an effect?

• High-level encapsulation of a drawing method
  • Shader code
  • Uniform values
  • GL state settings
    • Samplers, textures, etc.
    • Rasterization settings

• An effect may contain multiple passes
  • Set one shader & parameters, draw, repeat...
What is nvFX?

• An effects file format created by Tristan Lorch (NVIDIA)
  • Inspired by cgFX, but not specific to cg
  • Open-source library
    • [https://github.com/tlorach/nvFX](https://github.com/tlorach/nvFX)

• See also:
  • Search “nvfx site:developer.nvidia.com”
  • Search “nvfx site:khronos.org”
GLSLShader {// Prepend to all shaders
    #version 130
    uniform mat4 mvp;
}
GLSShader ObjectVS {
    in vec4 position;
    in vec3 normal;
    out vec3 normal_eye_space;
    void main() {
        ...
    }
}
GLSLShader ObjectFS {
    ...
}
GLSLShader DiffuseFromTexture {
    uniform sampler2D tex;
    vec4 getDiffuse(vec3 tc) {
        return texture(tex, tc);
    }
}
namespace floor {
    GLShader VS {
        ...
    }
    ...
}

Technique TECH_Floor {
    Pass p0 {
        VertexProgram = floor::VS;
        FragmentProgram = floor::FS;
    }
}
Mixed Versions

GLSLShader common_gl {
    #version 130
}
GLSLShader common_gles {
    #version 300 es
}

GLSLShader foo {
    ...
}

// C++ code has to read the
// annotation and do something smart
// with it.
Technique TECH_gl <
    GLSL_min_version = 1.30
> {
    VertexProgram = { common_gl,
                     foo };
    ...
}

Technique TECH_gles <
    GLSL_min_version = 3.00
> {
    VertexProgram = { common_gles,
                     foo };
    ...
}
nvFX Advantages

• More robust language for combining shaders into programs
• More robust language for changing GL state
• Much better mechanism for associating data with vertex attributes
• Multiple passes
• Non-screen render targets
  • So that effects can render shadow maps, etc.
• Shaders targeting multiple shading languages can live in one place
  • Sharing shader text across versions is clunky
• Documentation :)
nvFX Disadvantages

• Still requires a lot of C++ code to use
• No direct integration with models
  • Model files would generally reference effects (by name) that are defined in the fx files
  • Sort of the opposite binding order from what we want
• No transform feedback support
• No direct way to verify results of rendered image
• No way to specify effect requirements
  • Like “GLSL >= 1.30” in shader_runner
  • Annotations may fill this gap
• No Linux or Mac build targets yet
  • It uses cmake, so it shouldn't be too hard to add...
Can piglit use nvFX?

- Probably not as-is
  - Not straightforward to replace tests that draw many quads & probe results
  - No obvious way to supply additional vertex data
    - Standard set of model files?
    - We'd probably have to extend their parser
- If apitrace could generate nvFX files...
Can shader_runner borrow ideas from nvFX?

• Nice file format
  • Decent parser, too
  • Clean syntax for textures and state information

  [require]
  GL >= 3.0

  [fx]
  ...

  [test]
  technique foo
  draw rect -1 -1 1 1
  probe rgb 10 10 0 1 0
  ...

• May provide an eventual migration path to nvFX