Introduction to GLSL

EECS 487

October 2, 2006

before we start…

- special assignment sign-up
- class contribution:
  - phorum, class
  - code, models, textures
  - attend discussions

project 2

- start now
- read phorum
- for GLSL:
  - use autolab
  - or your own computer if gfx card is good
  - other labs?

homework 1

OpenGL Pipeline

1. vertex processing
   - transformations: 3D → 2D
   - lighting
2. clipping, primitive assembly
3. fragment processing
   - rasterize primitives
   - interpolate colors, texture coordinates, etc.
4. fragment test, etc.
   - depth, alpha
   - alpha blending

Programmable parts

- vertex processing
  - transformations: 3D → 2D
  - lighting
- clipping, primitive assembly
- fragment processing
  - rasterize primitives
  - interpolate colors, texture coordinates, etc.
- fragment test, etc.
  - depth, alpha
  - alpha blending

Basic idea

- Replace vertex or fragment computations with application-provided *programs*
  - also called *shaders*
- Written in high-level language: GLSL
- Graphics driver compiles and links program at run-time
- Application activates the program to replace fixed-functionality OpenGL pipeline

2 issues

1. How to write shaders
2. How to activate shaders in OpenGL

our focus: #1

jot handles #2

- nothing deep; read the manual

GLSL: C Basis

- Based on C, with some C++ features
- Graphics-friendly data types: `vec2, vec3, vec4, mat2, mat3, mat4, void, bool, float, int,` …
- structs, 1D arrays, functions, iteration, if/else

Code snippet

```
void main() {
    const float f = 3.0;
    vec3 u(1.0), v(0.0, 1.0, 0.0);
    for (int i=0; i<10; i++)
        v = f * u + v;
    …
}
```

General purpose?

- Seems like general purpose computing.
  - Anything missing?

Missing features

- No pointers or dynamically allocated memory
- No strings, characters
- No double, byte, short, long, unsigned…
- No file I/O
- No printf()
- Focus is numerical computation

Other differences

- No automatic type conversion

```
float f = 1; // WRONG
float f = 1.0; // much better
```

- Simplifies things
- Instead of casting, use constructors:

```
vec3 v3 = vec3(0.5, 1.0, 0.5);
vec4 v4 = vec4(v3, 1.0);
vec2 v2 = vec2(v4);
```

Other differences

- 3 kinds of function parameters:
  - **`in`** (assumed)
  - **`out`**
  - **`inout`**
- no pointers or references

Graphics-friendly functions

- `sin, cos, tan, asin, acos, atan, ...`
- `pow, exp2, log2, sqrt, …`
- `abs, floor, ceil, mod, min, max, clamp…`
- `mix, step, smoothstep`
- `length, distance, dot, cross, normalize`
- `reflect (!)`
- more…

Type qualifiers

Variables passed to shaders from the application:

`uniform:`

- value is constant over primitive (e.g. light direction)

`attribute:`

- value varies per-vertex (e.g. vertex normal)
- built-in (e.g. `gl_Vertex`) or application-specific

`varying:`

- output from a vertex shader
- input to a fragment shader
- (interpolated per-fragment)

Example: vertex shader

```
varying vec3 N;  // per-fragment normal

void main()
{
   // compute the vertex normal in eye coordinates:
   N = gl_NormalMatrix * gl_Normal;

   // output vertex position in clip coordinates
   gl_Position = ftransform();
}
```

Example: fragment shader

```
varying vec3 N;  // per-fragment normal

// Takes the normal (interpolated per pixel) and
// simply turns it into a color.

void main() {
   // N is interpolated across the triangle,
   // so normalize it:
   vec3 n = normalize(N);
   gl_FragColor = vec4(abs(n), 1.0);
 }
```

Example: standard lighting vertex program

```
// vertex program for standard OpenGL lighting computed per-pixel.

varying vec3 N;  // per-fragment normal in eye coords
varying vec4 P;  // per-fragment position in eye coords

void main()
{
   // compute the vertex normal and position in eye coordinates:
   N = gl_NormalMatrix * gl_Normal;
   P = gl_ModelViewMatrix * gl_Vertex;

   // output vertex position in clip coordinates
   gl_Position = ftransform();
}

// lighting.vp
```

Example: standard lighting fragment program

```
// fragment program to compute OpenGL lighting model (per-pixel)
// for case of directional lights 0 thru 4. Uses reflection
// vector instead of halfway vector.
//
// Note: We assume lights 0 - 3 are enabled (and others disabled).

varying vec3 N;  // per-fragment normal, eye space
varying vec4 P;  // per-fragment location, eye space

void main()
{
   vec4 a = gl_FrontMaterial.ambient;
   vec4 d = gl_FrontMaterial.diffuse;
   vec4 s = gl_FrontMaterial.specular;

   // N is interpolated across the triangle, so normalize it:
   vec3 n = normalize(N);

   // continued…
```

cont'd

```
// unit view vector in eye space:
vec3 v = normalize(-P.xyz);

// base color: global ambient light:
vec4 color = gl_LightModel.ambient * a;

// add contributions from each light:
for (int i=0; i<4; i++) {

   // store unit vector to light
   vec3 l = normalize(
      (gl_LightSource[i].position.w == 0.0 ?
       gl_LightSource[i].position :          // directional
       gl_LightSource[i].position - P).xyz   // positional
      );

   // find n dot l:
   float nl = max(dot(n,l), 0.0);

   // continued…
```

cont'd

```
 // attenuation:
 float t = 1.0;

 if (gl_LightSource[i].position.w != 0.0) { // if positional
    float d = distance(gl_LightSource[i].position, P);
    float k0 = gl_LightSource[i].constantAttenuation;
    float k1 = gl_LightSource[i].linearAttenuation;
    float k2 = gl_LightSource[i].quadraticAttenuation;
    t = 1.0 / (k0 + d*(k1 + d*k2));
}

// spotlight:
if (gl_LightSource[i].spotCutoff < 180.0) { // if spotlight in effect
    float sl = dot(normalize(gl_LightSource[i].spotDirection), -l);
    t *= (sl < gl_LightSource[i].spotCosCutoff) ?
         0.0 :                                              // out of cone
         pow(max(sl,0.0), gl_LightSource[i].spotExponent);  // inside cone
}

// continued…
```

cont'd

```
    // unit reflection vector in eye space:
    vec3  r = reflect(-l,n);
    float rv = pow(max(0.0, dot(r, v)), gl_FrontMaterial.shininess);
    color += t * ((gl_LightSource[i].ambient   * a)      +
                  ((gl_LightSource[i].diffuse  * d) * nl) +
                  ((gl_LightSource[i].specular * s) * rv));
    }
    gl_FragColor = color;
}

// lighting.fp
```

Online resources

http://developer.3dlabs.com/openGL2/
http://www.lighthouse3d.com/opengl/glsl/
http://www.opengl.org/documentation/glsl/