GLSL Tutorial

CIS 565 Fall 2011
Qing Sun
sunqing@seas.upenn.edu

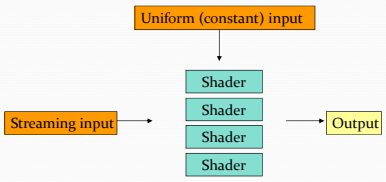
GLSL Syntax Overview

- GLSL is like C without
  - Pointers
  - Recursion
  - Dynamic memory allocation
- GLSL is like C with
  - Built-in vector, matrix and sampler types
  - Constructors
  - A great math library
  - Input and output qualifiers

Allow us to write concise, efficient shaders.

GLSL Syntax: `in`/ `out`/ `uniform`

- Recall



GLSL Syntax: `in`/ `out`/ `uniform`

- Example

```
#version 330

uniform mat4 u_ModelView;
in vec3 Position;
in vec3 Color;
out vec3 fs_Color;

void main(void)
{
  fs_Color = Color;
  gl_Position = u_ModelView * vec4(Position, 1.0);
}
```

`uniform`: shader input constant across `glDraw`

`in`: shader input varies per vertex attribute

`out`: shader output

GLSL Syntax

- GLSL has a preprocesssor

```
#version 330

#ifdef FAST_EXACT_METHOD
  FastExact();
#else
  SlowApproximate();
#endif

// ... many others
```

- All Shaders have main()

```
void main(void)
{
}
```

GLSL Syntax: Types

- Scalar types: `float`, `int`, `uint`, `bool`
- Vectors are also built-in types:
  - `vec2`, `vec3`, `vec4`
  - `ivec*`, `uvec*`, `bvec*`
- Access components three ways:
  - .x, .y, .z, .w position or direction
  - .r, .g, .b, .a color
  - .s, .t, .p, .q texture coordinate

`myColor.xyz` `myColor.xgb`

GLSL Syntax: Vectors

- Constructors

```
vec3 xyz = vec3(1.0, 2.0, 3.0);

vec3 xyz = vec3(1.0); // [1.0, 1.0, 1.0]

vec3 xyz = vec3(vec2(1.0, 2.0), 3.0);
```

GLSL Syntax: Vectors

- Swizzle: select or rearrange components

```
vec4 c = vec4(0.5, 1.0, 0.8, 1.0);

vec3 rgb = c.rgb;  // [0.5, 1.0, 0.8]

vec3 bgr = c.bgr;  // [0.8, 1.0, 0.5]

vec3 rrr = c.rrr;  // [0.5, 0.5, 0.5]

c.a = 0.5;         // [0.5, 1.0, 0.8, 0.5]
c.rb = 0.0;        // [0.0, 1.0, 0.0, 0.5]

float g = rgb[1];  // 0.5, indexing, not swizzling
```

GLSL Syntax: Matrices

- Matrices are built-in types:
  - Square: `mat2`, `mat3`, `mat4`
  - Rectangular: `matmxn` `m` columns, `n` rows
- Stored column major

GLSL Syntax: Matrices

- Constructors

```
mat3 i = mat3(1.0); // 3x3 identity matrix

mat2 m = mat2(1.0, 2.0,  // [1.0 3.0]
              3.0, 4.0); // [2.0 4.0]
```

- Accessing elements

```
float f = m[column][row];

float x = m[0].x; // x component of first column

vec2 yz = m[1].yz; // yz components of second column
```

Treat matrix as array of column vectors

Can swizzle too!

GLSL Syntax: Vectors and Matrices

- Matrix and vector operations are easy and fast:

```
vec3 xyz = // ...

vec3 v0 = 2.0 * xyz; // scale
vec3 v1 = v0 + xyz;  // component-wise
vec3 v2 = v0 * xyz;  // component-wise

mat3 m = // ...
mat3 v = // ...

mat3 mv = v * m;      // matrix * matrix
mat3 xyz2 = mv * xyz; // matrix * vector
mat3 xyz3 = xyz * mv; // vector * matrix
```

GLSL Syntax: Samplers

- Opaque types for accessing textures

```
uniform sampler2D colorMap; // 2D texture

vec3 color = texture(colorMap, vec2(0.5, 0.5)).rgb;

vec3 colorAbove = textureOffset(colorMap,
  vec2(0.5, 0.5), ivec2(0, 1)).rgb;

vec2 size = textureSize(colorMap, 0);

// Lots of sampler types:  sampler1D,
// sampler3D, sampler2DRect, samplerCube,
// isampler*, usampler*, ...

// Lots of sampler functions: texelFetch, textureLod
```

Samplers must be uniforms

`texture ()` returns a `vec4`; extract the components you need

2D texture uses 2D texture coordinates for lookup

2D texture coordinate

2D integer offset

GLSL Syntax: Samplers

- Textures
  - Are like 2D arrays
  - Were the backbone of GPGPU

GLSL Built-in Functions

- Selected trigonometry functions

```
float s = sin(theta);
float c = cos(theta);
float t = tan(theta);

float theta = asin(s);
// ...

vec3 angles = vec3(/* ... */);
vec3 vs = sin(angles);
```

Angles are measured in radius

Works on vectors component-wise

GLSL Built-in Functions

- Exponential Functions

```
float xToTheY = pow(x, y);
float eToTheX = exp(x);
float twoToTheX = exp2(x);

float l = log(x);   // ln
float l2 = log2(x); // log2

float s = sqrt(x);
float is = inversesqrt(x);
```

GLSL Built-in Functions

- Selected common functions

```
float ax = abs(x);  // absolute value
float sx = sign(x); // -1.0, 0.0, 1.0

float m0 = min(x, y); // minimum value
float m1 = max(x, y); // maximum value
float c = clamp(x, 0.0, 1.0);

// many others:  floor(), ceil(),
// step(), smoothstep(), ...
```

GLSL Built-in Functions

- Rewrite with one function call

```
float minimum = // ...
float maximum = // ...
float x = // ...

float f = min(max(x, minimum), maximum);
```

`clamp()`

GLSL Built-in Functions

- Rewrite without the `if` statement

```
float x = // ...
float f;

if (x > 0.0)
{
  f = 2.0;
}
else
{
  f = -2.0;
}
```

`sign()`

GLSL Built-in Functions

- Rewrite without the `if` statement

```
bool b = // ...
vec3 color;

if (b)
{
  color = vec3(1.0, 0.0, 0.0);
}
else
{
  color = vec3(0.0, 1.0, 0.0);
}
```

No built-in functions required

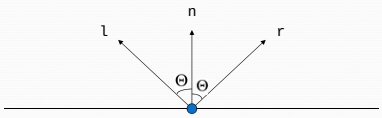
GLSL Built-in Functions

- Selected geometric functions

```
vec3 l = // ...
vec3 n = // ...
vec3 p = // ...
vec3 q = // ...

float f = length(l);      // vector length
float d = distance(p, q); // distance between points

float d2 = dot(l, n);     // dot product
vec3 v2 = cross(l, n);    // cross product
vec3 v3 = normalize(l);   // normalize

vec3 v3 = reflect(l, n);  // reflect

// also:  faceforward() and refract()
```

GLSL Built-in Functions

- `reflect(-l, n)`
  - Given l and n, find r. Angle in equals angle out



GLSL Built-in Functions

- What is wrong with this code?

```
vec3 n = // ...

normalize(n);
```

`normalize()` returns a new vector

GLSL Built-in Functions

- Selected matrix functions

```
mat4 m = // ...

mat4 t = transpose(m);
float d = determinant(m);
mat4 d = inverse(m);
```

Think performance before using these!

GLSL Built-in Functions

- Selected vector rational functions

```
vec3 p = vec3(1.0, 2.0, 3.0);
vec3 q = vec3(3.0, 2.0, 1.0);

bvec3 b = equal(p, q);        // (false, true, false)
bvec3 b2 = lessThan(p, q);    // (true, false, false)
bvec3 b3 = greaterThan(p, q); // (false, false, true)

bvec3 b4 = any(b);            // true
bvec3 b5 = all(b);            // false
```

GLSL Built-in Functions

- Rewrite in one line of code

```
bool foo(vec3 p, vec3 q)
{
  if (p.x < q.x)
  {
    return true;
  }
  else if (p.y < q.y)
  {
    return true;
  }
  else if (p.z < q.z)
  {
    return true;
  }
  return false;
}
```

```
return any(lessThan(p, q));
```

GLSL Resources (3.30)

- OpenGL/GLSL Quick Reference Card
  - http://www.khronos.org/files/opengl-quick-reference-card.pdf
- GLSL Spec
  - http://www.opengl.org/registry/doc/GLSLangSpec.3.30.6.clean.pdf
- NShader: Visual Studio GLSL syntax highlighting
  - http://nshader.codeplex.com/