MORE GLSL

Pramook Khungurn

CS 4621, Fall 2011

GLSL DATA TYPES

GLSL Data Types

- Both GLSL and Java
  - float, int
- GLSL has, but Java has not
  - vec3, vec4, vec4: vectors
  - mat2, mat3, mat4: matrices
  - sampler1D, sampler2D, sample3D, samplerCube, etc: textures
- Java has, but GLSL has not
  - Object
  - String
  - etc

vec2

- Represents a vector in 2D. Each component is a float.

```
vec2 a;
a.x = 0.0;
a.y = 1.0; // a = (0,1)

vec2 b;
b.s = 10.0;
b.t = 12.5; // b = (10,12.5)

vec2 c;
c[0] = 9.0;
c[1] = 8.0; // c= (9,8)
```

vec2

```
float p = a.t;        // p = 1
float q = b[1] + c.x // q = 21.5

vec2 d = vec2(3,c.y * 2); // d = (3,18)

vec3 d = a + b; // d = (10,13.5)

vec3 e = b - c; // e = (1,4.5)

vec3 f = b * c; // f = (90,100)

vec3 g = 3 * a; // g = (0,3)

float h = length(c); // h = 12.042
```

vec3

```
vec3 a;
a.x = 10.0; a.y = 20.0; a.z = 30.0;  // a = (10,  20,  30)
a.r = 0.1;  a.g = 0.2;  a.b = 0.3;   // a = (0.1, 0.2, 0.3)
a.s = 1.0,  a.t = 2.0;  a.p = 3.0;   // a = (1, 2, 3)

vec3 b = vec3(4.0, 5.0, 6.0);

vec3 c = a + b; // c = (5, 7, 9)
vec3 d = a - b; // d = (-3, -3, -3)
vec3 e = a * b; // e = (4, 10, 18)
vec3 f = a * 3; // e = (3, 6, 9)
float g = dot(a,b); // g = 32
vec3 h = cross(a,b); // h = (-5,6,-3)
float i = length(a); // i = 3.742
```

vec4

```
vec4 a;
a.x = 10.0; a.y = 20.0; a.z = 30.0; a.w = 40.0; // a = (10, 20, 30, 40)
a.r = 0.1;  a.g = 0.2;  a.b = 0.3;  a.a = 0.4;  // a = (0.1, 0.2, 0.3, 0.4)
a.s = 1.0;  a.t = 2.0;  a.p = 3.0;  a.q = 4.0;  // a = (1,2,3,4)

vec4 b = vec4(5,6,7,8);

vec4 c = a + b; // c = (6, 8, 10, 12)
vec4 d = a - b; // d = (-4, -4, -4, -4)
vec4 e = a * b; // e = (5, 12, 21, 32)
vec4 f = a * 3; // f = (3, 6, 9, 12)
float g = length(a); // g = 5.477
```

mat2

- Represents a 2 by 2 matrix. Each component is a float.

```
mat2 A = mat2(1.0, 2.0, 3.0, 4.0); // in column-major order

vec2 x = vec2(1.0, 0.0);
vec2 y = vec2(0.0, 1.0);

vec2 a = A * x; // a = (1,2)
vec2 b = A * y; // b = (3,4)
```

mat3

```
mat3 A = mat3(1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0);
        // in column-major order

vec3 x = vec3(1.0, 0.0, 0.0);
vec3 y = vec3(0.0, 1.0, 0.0);
vec3 z = vec3(0.0, 0.0, 1.0);

vec3 a = A * x; // a = (1,2,3)
vec3 b = A * y; // b = (4,5,6)
vec3 c = A * z; // c = (6,7,8)
```

mat4

- 4x4 matrices. Can store affine transformations.

```
mat4 A = mat2(1.0,  2.0,  3.0,  4.0,
              5.0,  6.0,  7.0,  8.0,
              9.0,  10.0, 11.0, 12.0
              13.0, 14.0, 15.0, 16.0); // in column-major order

vec4 x = vec4(1.0, 0.0, 0.0, 0.0);
vec4 y = vec4(0.0, 1.0, 0.0, 0.0);
vec4 z = vec4(0.0, 0.0, 1.0, 0.0);
vec4 w = vec4(0.0, 0.0, 0.0, 1.0);

vec4 a = A * x; // a = (1,2,3,4)
vec4 b = A * y; // b = (5,6,7,8)
vec4 c = A * z; // c = (9,10,11,12)
vec4 d = A * w; // d = (13,14,15,16)
```

Array

- We can declare fixed-size arrays.
- Use C syntax.

```
float A[4];
A[0] = 5; A[3] = 10;

vec4 B[10];
B[3] = vec4(1,2,3,4); B[8].y = 10.0;
```

Twizzling

- Used to construct a vector from another vector by referring to multiple components at one time.

```
vec4 a = vec4(1,2,3,4);

vec3 b = a.xyz;  // b = (1,2,3)
vec2 c = a.qp;   // c = (4,3)
vec4 d = a.xxyy; // d = (1,1,2,2)
```

Type Conversion

- Syntax: <<variable>> = <<type>>(<<value>>);
- Expression on RHS = “constructor expression.”
- Example:

```
float a = 1.0;
int b = int(a);
```

Type Conversion

- We can create larger vectors from smaller ones.

```
vec2 a = vec2(1,2);
vec2 b = vec2(3,4);

vec4 c = vec4(a,b);     // c = (1,2,3,4)

vec3 d = vec3(0,0,1);

vec4 e = vec4(d,0);    // d = (0,0,1,0)

vec4 f = vec2(0,a,3); // f = (0,1,2,3)
```

UNIFORM VARIABLES

Uniform Variable

- A GLSL variable the user can specify value from the C/Java side.
- Its value shouldn’t change very often.
  - Hundreds or thousands polygon at a time.
- Suitable for specifying
  - Material properties
  - Transformation matrices
  - Light sources
  - Texture
- Cannot change value between glBegin(…) and glEnd(…)

Declaring a Uniform Variable in GLSL

- Declare as a global variable (outside functions).
- Prefix the variable type with keyword “uniform.”
- Examples:

```
uniform float shininess;
uniform vec3 color;
uniform mat4 model_transform;

void main()
{
    // Code here…
}
```

Caveats

- Uniform variables are shared between vertex and fragment shaders.
  - Declare once in vertex shader and once more in fragment shader.
- As a result, types of uniform variables in vertex and fragment shaders must be consistent.
  - Cannot have uniform int x; in vertex shader,
    but uniform float x; in fragment shader.

Using Uniform Variables in the CS4621 Framework

- Uniform variables are encapsulated by `Uniform` class.
- Use `Program.getUniform(<name>)` to get the instance corresponding to the name.
- Set values by `Uniform.set**(...)` methods.

```
// In GLSL: uniform float3 p;
program.use();
Uniform p = program.getUniform("p");
p.set3Float(1.0f, 2.0f, 3.0f);

// In GLSL: uniform int count;
program.use();
Uniform countUniform = program.getUniform("count");
countUniform.set1Int(10);
```

Demo 18: 2D Twisting

2D Twisting

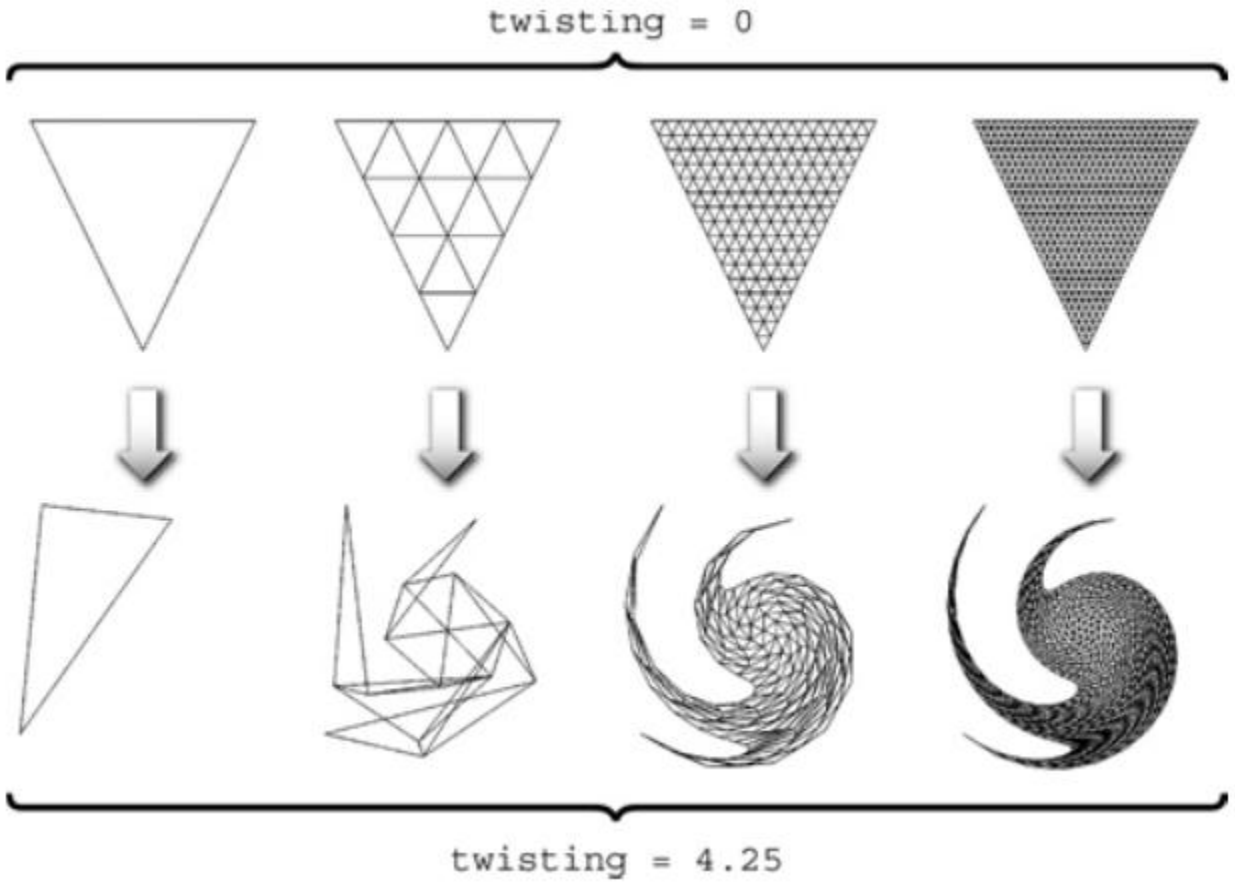
- We transform the vertices according to the following equation:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\left(t\sqrt{x^2+y^2}\right) & -\sin\left(t\sqrt{x^2+y^2}\right) \\ \sin\left(t\sqrt{x^2+y^2}\right) & \cos\left(t\sqrt{x^2+y^2}\right) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

where

- (x,y) is the vertex position in object space.
- (x',y') is the vertex position in clip space.
- t is the twisting factor,
  which is stored in the uniform variable "twisting"

2D Twisting



Vertex Shader Code

```
#version 110

uniform float twisting;

void main()
{
    float angle = twisting * length(gl_Vertex.xy);
    float s = sin(angle);
    float c = cos(angle);
    gl_Position.x = c * gl_Vertex.x - s * gl_Vertex.y;
    gl_Position.y = s * gl_Vertex.x + c * gl_Vertex.y;
    gl_Position.z = 0.0;
    gl_Position.w = 1.0;
}
```

Fragment Shader Code

```
#version 110

uniform vec3 color;

void main()
{
    gl_FragColor = vec4(color, 1);
}
```

GLSL Program Preparation in Java

```
private Program program = null;
private Uniform twisting;
private Uniform color;

public void init(GLAutoDrawable drawable) {
	super.init(drawable);
	final GL2 gl = drawable.getGL().getGL2();

	// Check whether GLSL is supported
	if ( !Shader.checkGlslSupport(gl) ) {
		System.exit(1);
	}

	try {
		// Load, compile and link the shaders
		this.program = new Program(gl,
				"twisting.vert",
				"twisting.frag");

		twisting = program.getUniform("twisting");
		color = program.getUniform("color");
	} catch (GlslException e) {
		System.err.println(e.getMessage());
		System.exit(1);
	}
}
```

Using the GLSL Program

```
program.use();

twisting.set1Float(5.0f);
color.set3Float(0.5f, 0.9f, 0.5f);

int count = 200;
float size = 1.0f / count;

gl.glBegin(GL2.GL_QUADS);

...

gl.glEnd();

program.unuse();
```

VARYING VARIABLES

Varying Variables

- Programmer can specify its values in the vertex shader.
- The values specified in the vertex shader are interpolated.
- When using the same variables in the fragment shader, you get the interpolated value.

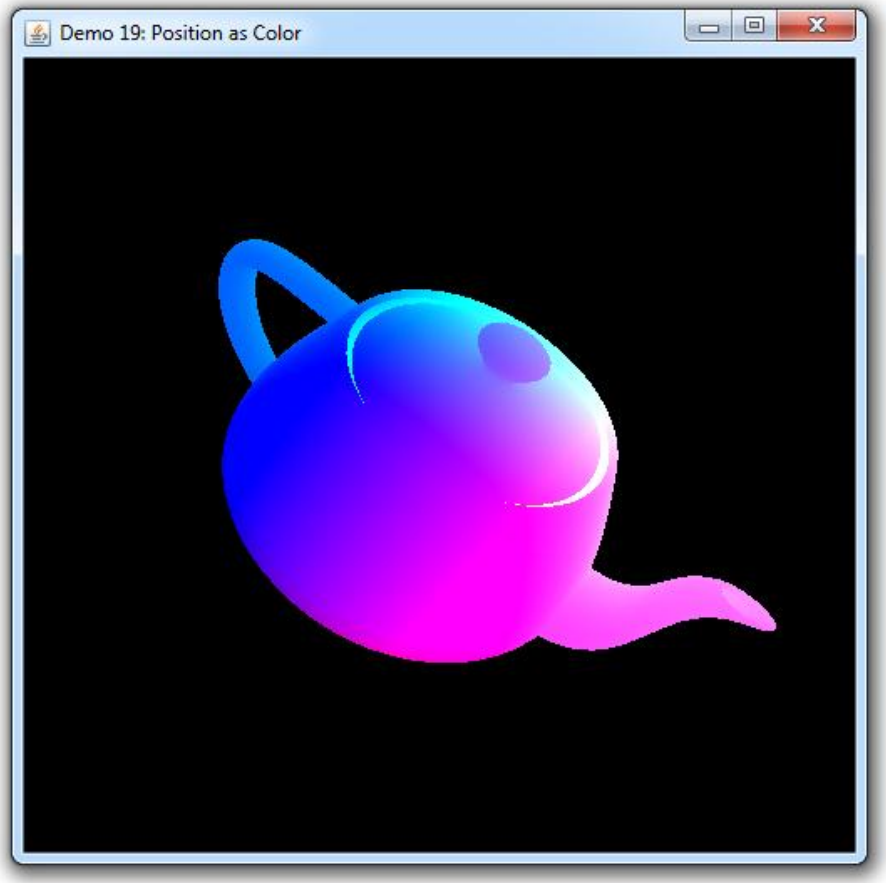
Declaring Varying Variables

- Declare as a global variable (outside functions).
- Syntax: varying <<type>> <<name>>;
- Example:

```
varying vec3 color;

void main()
{
    // Some code here...
}
```

Demo 19: Position as Color



Position as Color

- Compute the color of each fragment from its position in object space.

- color = (position + (1,1,1)) / 2

Vertex Shader Code

```
#version 110

varying vec3 color;

void main()
{
	gl_Position = ftransform();
	color = (vec3(gl_Vertex.xyz) + vec3(1,1,1)) * 0.5;
}
```

ftransform()

- A convenience function.
- Outputs the product of the vertex in object space with the modelview matrix, then the projection matrix.
- Basically, does the normal OpenGL transform for you.

Fragment Shader Code

```
#version 110

varying vec3 color;

void main()
{
	gl_FragColor = vec4(color, 1);
}
```