Uniform variables

Uniform Variable

- A GLSL variable the user can specify value from the C/Java side.
- Its value is constant while drawing each vertex and pixel.
- Suitable for specifying
  - Material properties
  - Transformation matrices
  - Light sources
  - Texture

Declaring a Uniform Variable in GLSL

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

```
// The values for these are initialized in the Java code!
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.
- Uniforms that are declared but not used are “optimized” out
  - OpenGL throws an error if you try to set a nonexistent uniform

Using Uniform Variables in the CS4620 Framework

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

```
// In GLSL: uniform vec3 color;
program.use();
Vector3 c = new Vector3(1.0f, 0.5f, 1.0f);
GLUniform.set(program.getUniform("color"), c);

// In GLSL: uniform mat4 MVP;
// For matrices, use setST, not set! A boolean is provided
// for transposing.
GLUniform.setST(program.getUniform("MVP"),
    camera.mViewProjection, false);
```

Attribute variables

Attribute Variables

- A variable containing an attribute for a single vertex.
- Position, normal, texture coordinate, etc.
- Each time the shader is run, the attribute variables receive the values for the current vertex.
- These only appear in vertex shaders. (Why?)

Attribute Mapping

- Attribute variables map to OpenGL buffers.
- OpenGL buffers have an index, GLSL attribute variables have a name.
- Must ensure the mapping from buffer indices to variable names is correct.
- In the provided framework:

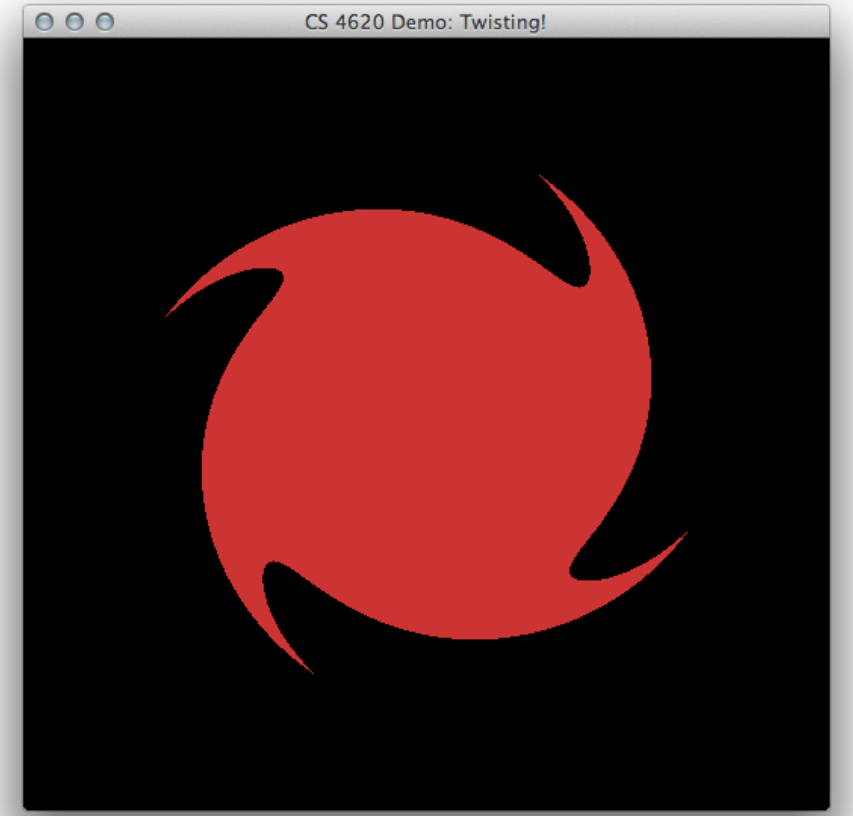
```
// Create a data buffer to fill in the attribute data
GLBuffer vertexPositions = new GLBuffer(BufferTarget.ArrayBuffer,
   BufferUsageHint.StaticDraw, true);

vertexPositions.setAsVertexVec3();

// Set vertexPositions, e.g. by reading in a Mesh

vertexPositions.useAsAttrib(program.getAttribute("in_Vertex"));
```

Demo: Twisting



2D Twisting

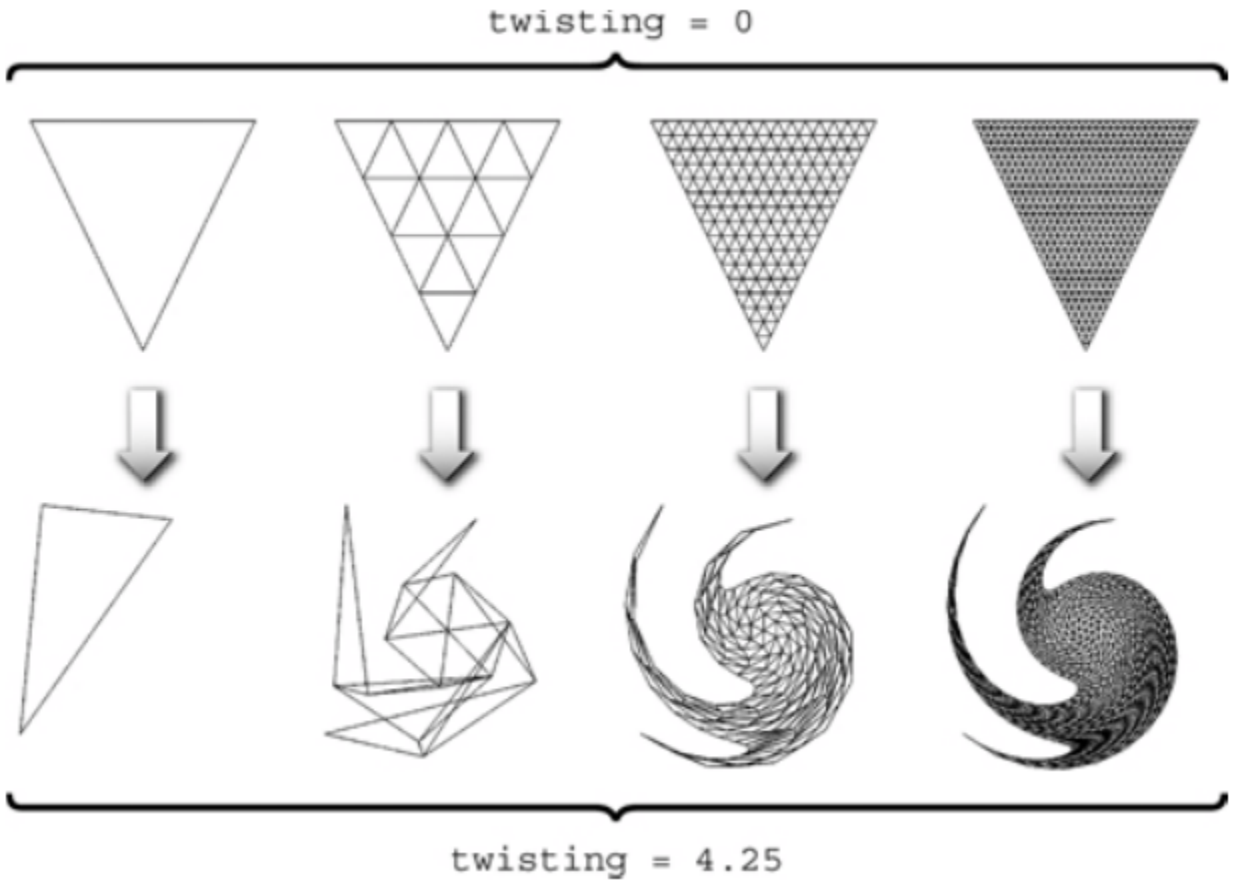
- We transform vertices according to the following equation:

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

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 "un_Twist"

2D Twisting



Vertex Shader Code

```
#version 120

uniform mat4 un_Projection;
uniform mat4 un_ModelView;
uniform vec3 un_DiffuseColor;
uniform float un_Twist;

attribute vec2 in_Vertex;

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

Fragment Shader Code

```
#version 120

uniform mat4 un_Projection;
uniform mat4 un_ModelView;
uniform vec3 un_DiffuseColor;
uniform float un_Twist;

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

Using the GLSL Program

```
public void draw(GameTime gameTime) {
   glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
   glClear(GL2.GL_COLOR_BUFFER_BIT);

   program.use();

   // Set the uniforms
   GLUniform.setST(program.getUniform("un_Projection"),
        mProjection);
   GLUniform.setST(program.getUniform("un_ModelView"),
        mModelView);
   GLUniform.set(program.getUniform("un_DiffuseColor"), color);
   GLUniform.set(program.getUniform("un_Twist"), 0.5f);

   // Set the attribute
   vertexPositions.useAsAttrib(program.getAttribute("in_Vertex"));

   glDrawElements(...); // Draw the mesh

   GLProgram.unuse();
}
```

Varying variables

Varying Variables

- Interface between vertex and fragment shaders.
- Vertex shader outputs a value at each vertex, writing it to this variable.
- Fragment shader reads a value from the same variable, automatically interpolated to that fragment.
- No need to declare these in the Java program. (Why?)

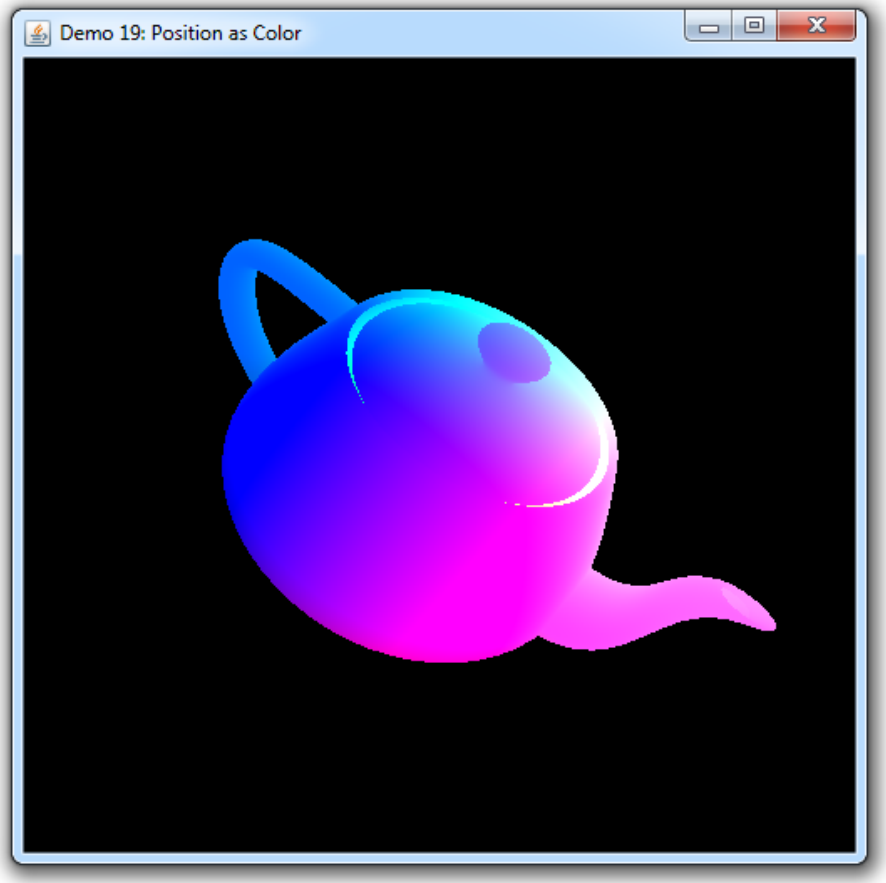
Declaring Varying Variables

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

```
varying vec3 color;

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

Demo: 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 120

uniform mat4 un_Projection;
uniform mat4 un_ModelView;

attribute vec3 in_Vertex;

varying vec3 ex_Color;

void main()
{
   gl_Position = un_Projection * un_ModelView * vec4(in_Vertex, 1);
   ex_Color = (in_Vertex.xyz + vec3(1,1,1)) * 0.5;
}
```

Fragment Shader Code

```
#version 120

uniform mat4 un_Projection;
uniform mat4 un_ModelView;

varying vec3 ex_Color;

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