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:

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

```glsl
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 CS4620 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`.

```plaintext
// 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

twisting = 0

twisting = 4.25
#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;
}
#version 110

uniform vec3 color;

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

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:

```cpp
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.

  \[
  \text{color} = \frac{\text{position} + (1,1,1)}{2}
  \]
#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.
#version 110

varying vec3 color;

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