Introduction to the OpenGL Shading Language

David Wolff

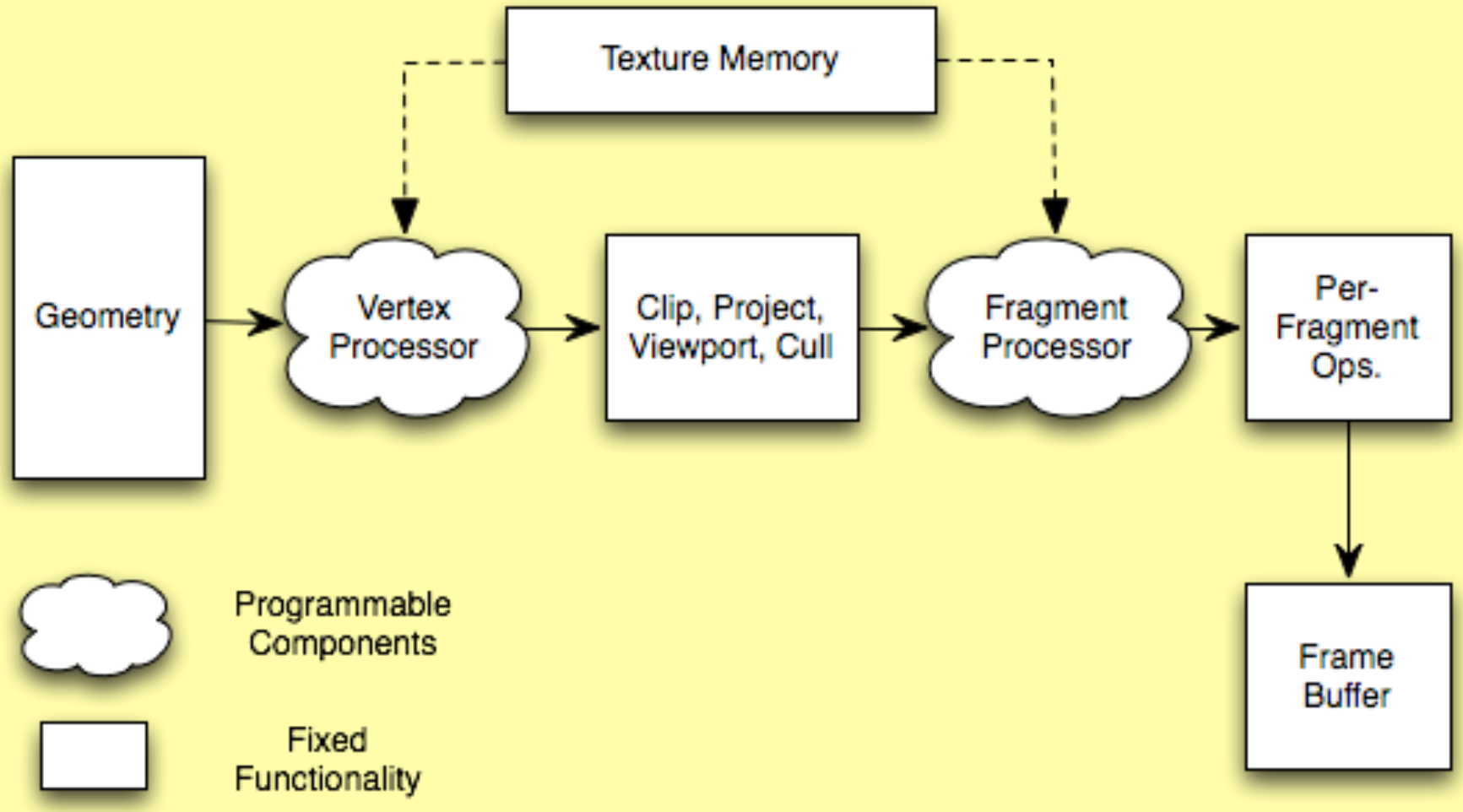
Pacific Lutheran University

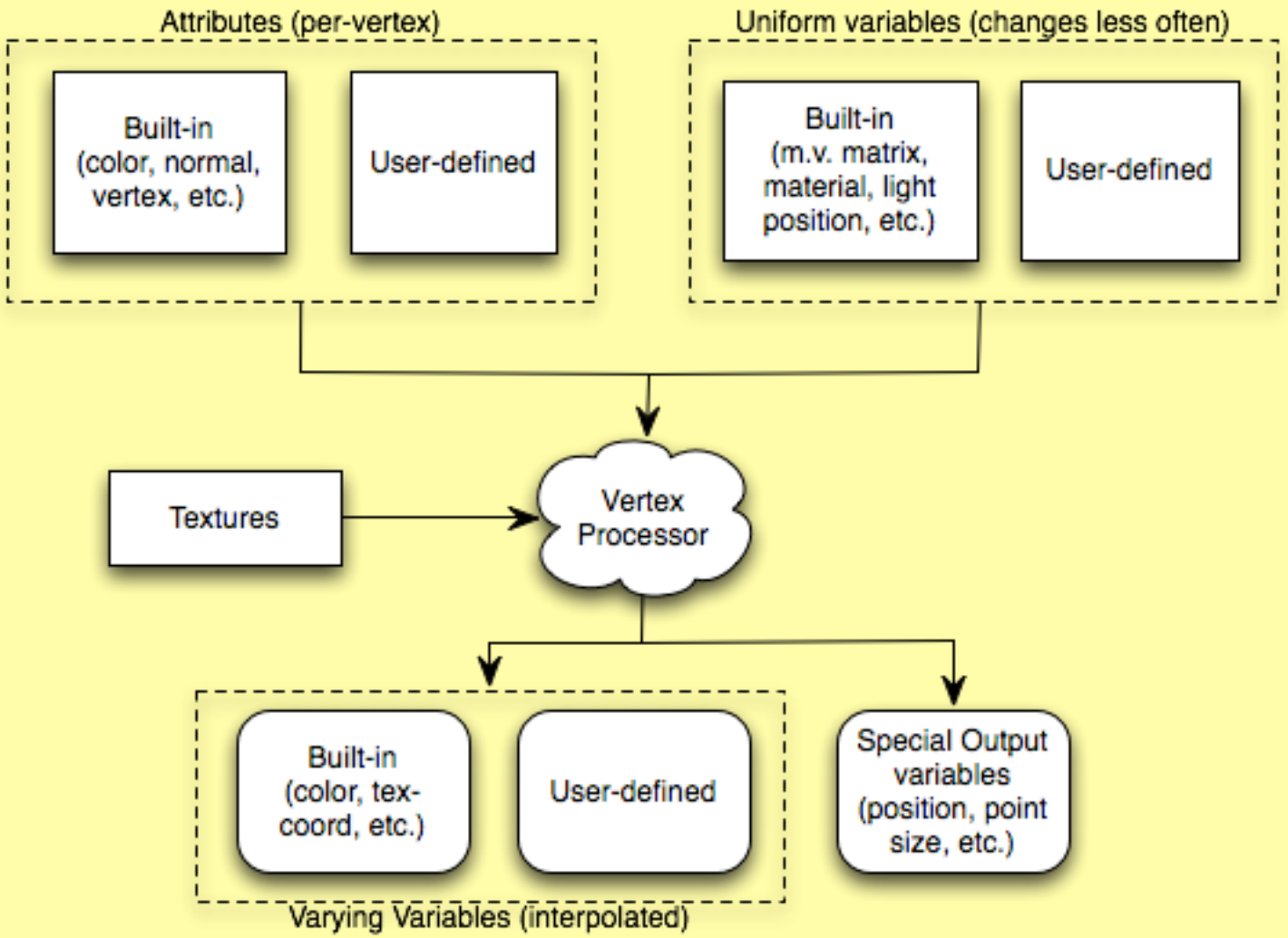
CCSC-NW 2008 Ashland, OR

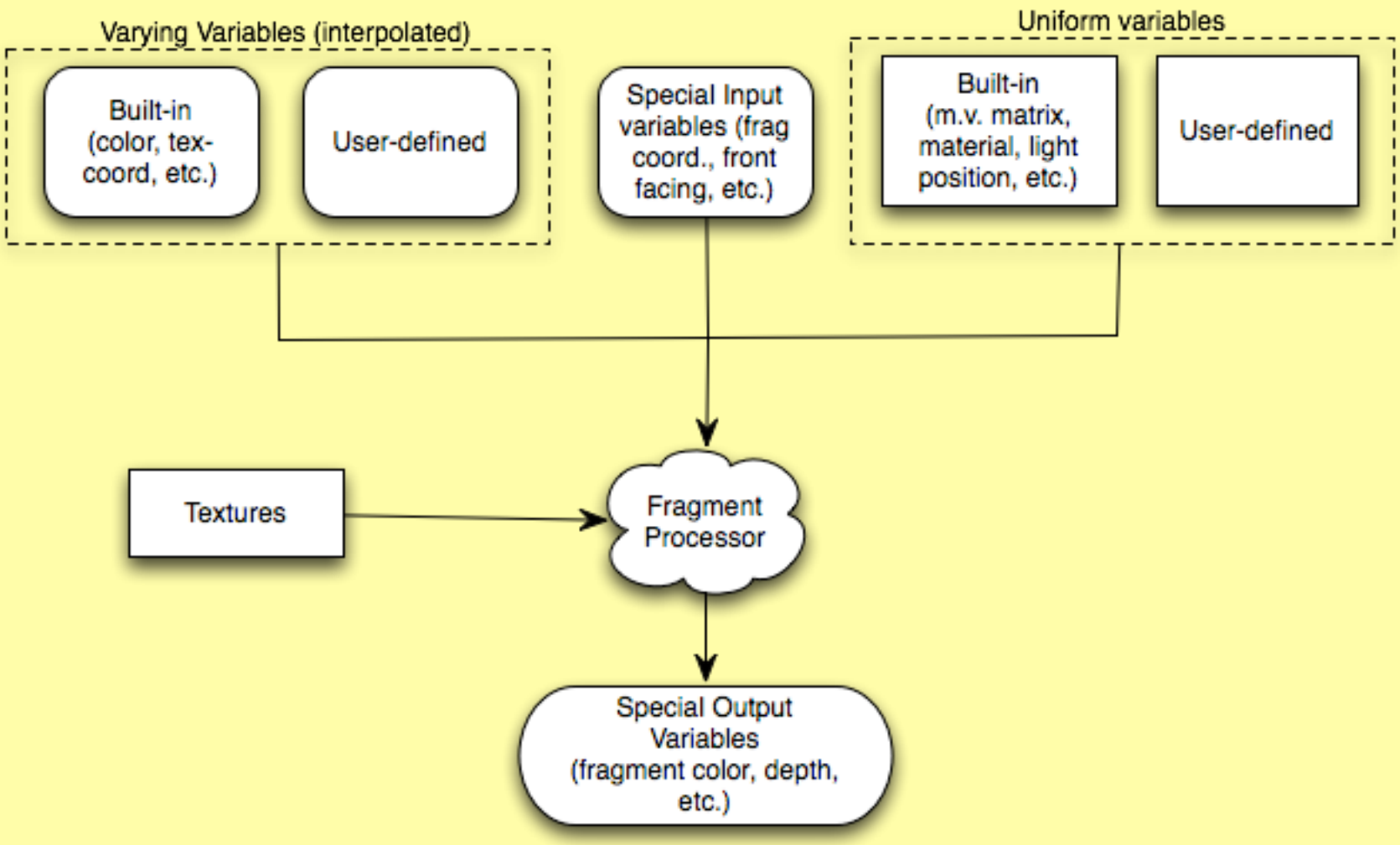
- Schedule

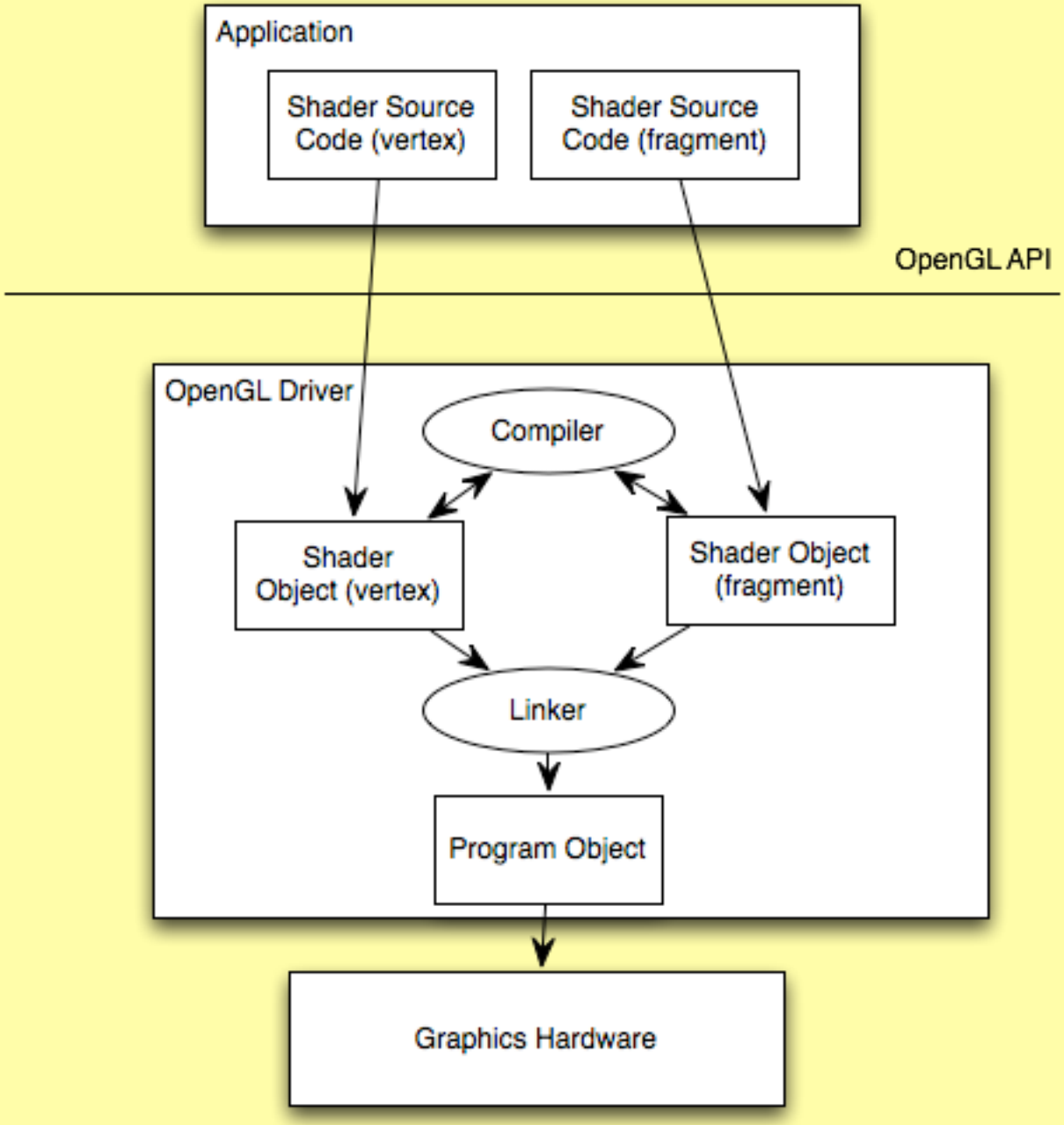
1. OpenGL pipeline, setup Eclipse (10 min)
2. Hello World shaders (15 min)
3. GLSL Overview (10 min)
4. Cartoon Shader (10 min)
5. Bump (Normal) Map Shader (10 min)
6. Refraction/Reflection Shader (10 min)
7. Mandelbrot Shader (10 min)
8. Demo of Eclipse Plugin (10 min)

The OpenGL Programmable Pipeline


Texture Memory
Geometry
Vertex Processor
Clip, Project, Viewport, Cull
Fragment Processor
Per-Fragment Ops.
Frame Buffer
Programmable Components
Fixed Functionality


Attributes (per-vertex)
Built-in (color, normal, vertex, etc.)
User-defined
Uniform variables (changes less often)
Built-in (m.v. matrix, material, light position, etc.)
User-defined
Textures
Vertex Processor
Built-in (color, tex-coord, etc.)
User-defined
Special Output variables (position, point size, etc.)
Varying Variables (interpolated)


Varying Variables (interpolated)
Built-in (color, tex-coord, etc.)
User-defined
Special Input variables (frag coord., front facing, etc.)
Uniform variables
Built-in (m.v. matrix, material, light position, etc.)
User-defined
Textures
Fragment Processor
Special Output Variables (fragment color, depth, etc.)

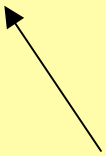

Application
Shader Source Code (vertex)
Shader Source Code (fragment)
OpenGL API
OpenGL Driver
Compiler
Shader Object (vertex)
Shader Object (fragment)
Linker
Program Object
Graphics Hardware

```
// Compile vertex shader object
int vertShader = glCreateShader(GL_VERTEX_SHADER);
glShaderSource(vertShader, 1, vertSource, null);
glCompileShader(vertShader);

// Compile fragment shader object
int fragShader = glCreateShader(GL_FRAGMENT_SHADER);
glShaderSource(fragShader, 1, fragSource, null);
glCompileShader(fragShader);

// Create and link program object
int program = glCreateProgram();
glAttachShader(program,vertShader);
glAttachShader(program,fragShader);
glLinkProgram(program);
… …

glUseProgram(program);
```

```
public class GLSLProgram {

      private int id;

      public void compileVertexShader(String src)
            throws ShaderException

      public void compileFragmentShader(String src)
            throws ShaderException

      public void link() throws ShaderException

      public void enable() throws ShaderException
      public void disable()
      ...
}
```

In `Main.java` (package edu.plu.daw.shaderdemo)

```
ShaderDemo demo = new HelloShaderDemo(canvas);
```



Change for each demo

Shader source code is in:
`resources.shaders`

Hello World Shader(s)

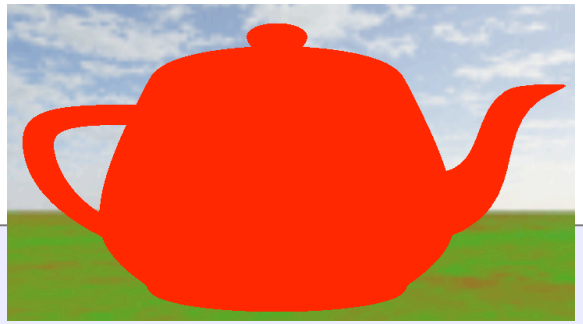
Main.java

```
ShaderDemo demo = new HelloShaderDemo(canvas);
```

Shader Source: `hello.vert`
`hello.frag`

(in `resources.shaders`)

Hello World!



```
// Vertex Shader
void main( )
{
     gl_Position = gl_ModelViewProjectionMatrix *
                          gl_Vertex;
}
```

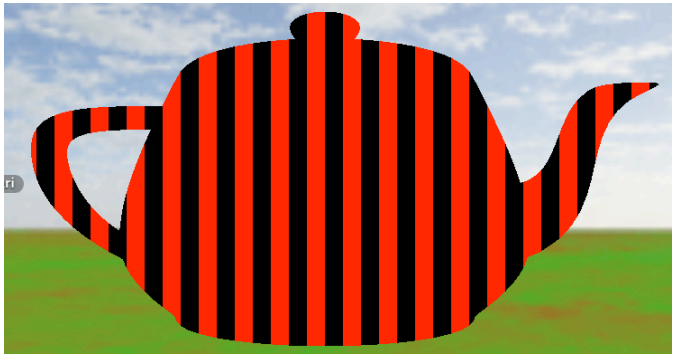
```
// Fragment Shader
void main( )
{
     gl_FragColor = vec4(1.0,0.0,0.0,1.0);
}
```

```
// Vertex Shader
void main( )
{
     gl_Position = gl_ModelViewProjectionMatrix *
                        gl_Vertex;
}
```

Nearly equivalent to:

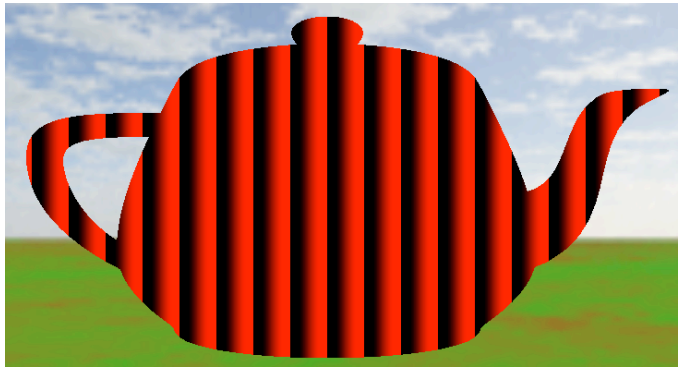
```
// Vertex Shader
void main( )
{
     gl_Position = ftransform();
}
```

A striped fragment shader



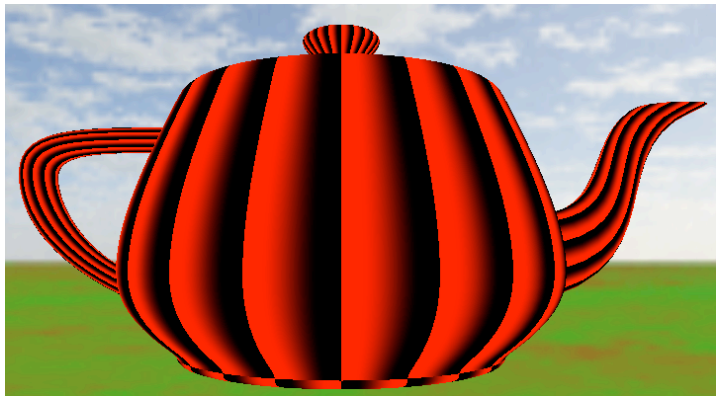
```
// Fragment Shader
void main( )
{
      float scale = 20.0 / 800.0;
      float fr = fract(gl_FragCoord.x * scale);
      gl_FragColor = vec4(
        step( 0.5, fr ),0.0,0.0,1.0
      );
}
```

Smoother transition



```
// Fragment Shader
void main( )
{
      float scale = 20.0 / 800.0;
      float fr = fract(gl_FragCoord.x * scale);
      gl_FragColor = vec4(
        smoothstep( 0.2,0.8, fr ),0.0,0.0,1.0
      );
}
```

Let's use the texture coordinates instead:



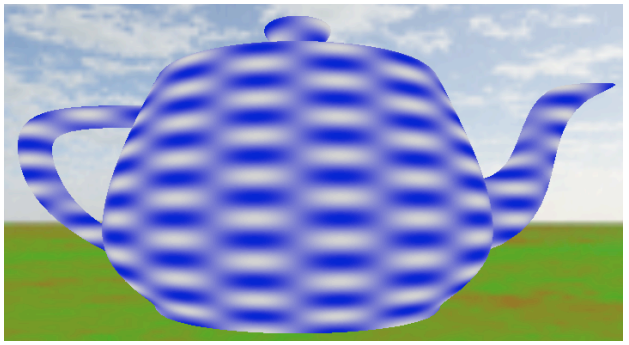
Add to vertex shader:

```
// Pass along the texture coordinate
gl_TexCoord[0] = glMultiTexCoord0;
```

Modify fragment shader:

```
float scale = 5.0;
float fr = fract(gl_TexCoord[0].s * scale);
gl_FragColor = vec4(
  smoothstep(0.2,0.8,fr),0.0,0.0,1.0
);
```

Using the model coordinates
(and a different pattern):



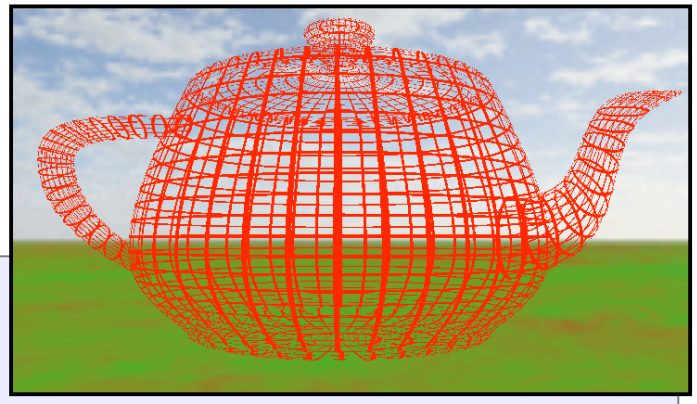
Add to vertex shader:

```
varying vec3 MCposition;          // Above main def
…
MCposition = vec3(gl_Vertex);     // inside main
```

Replace fragment shader:

```
varying vec3 Mcposition;          // Above main def.
…
// Entire contents of main:
vec3 col1 = vec3(0.8,0.8,0.8);
vec3 col2 = vec3(0.0,0.0,0.8);
float value = 0.5 * (1.0+(sin(MCposition.x*5.0)
      * sin(MCposition.z*20.0)) );
vec3 color = mix( col1, col2, value );
gl_FragColor = vec4(color,1.0);
```

A “lattice” fragment shader



```
// Fragment Shader
void main( )
{
      vec2 threshold = vec2(0.13,0.13);
      vec2 scale = vec2(10.0,10.0);
      float ss = fract(gl_TexCoord[0].s * scale.s);
      float tt= fract(gl_TexCoord[0].t * scale.t);

      if((ss > threshold.s) && (tt > threshold.t))
            discard;

      gl_FragColor = vec4(1.0,0.0,0.0,0.0);
}
```

Overview of the GLSL

Data Types

Scalar	Vector	Matrix	Samplers
float	vec2	mat2	sampler1D
int	vec3	mat3	sampler2D
bool	vec4	mat4	sampler3D
			samplerCube
	ivec2		sampler1DShadow
	ivec3		sampler2DShadow
	ivec4		
	bvec2		
	bvec3		
	bvec4		

Constructors

Vectors

```
vec4 v;
v = vec4(1.0, 2.0, 3.0, 4.0);
ivec2 c = ivec2(3, 4);
vec3 color = vec3(0.2, 0.5, 0.8);
vec4 color4 = vec4( color, 1.0 );    // vec4(0.2,0.5,0.8,1.0)
vec3 v = vec3(0.6);                  // equiv to vec3(0.6,0.6,0.6)
```

Matricies

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

$$\begin{bmatrix} 1.0 & 3.0 \\ 2.0 & 4.0 \end{bmatrix}$$

```
mat2 m2 = mat2(2.0);
```

$$\begin{bmatrix} 2.0 & 0.0 \\ 0.0 & 2.0 \end{bmatrix}$$

Selecting and Swizzling

```
vec4 v4;
v4.rgba;        // vec4
v4.rgb;         // vec3 of first three comp.
v4.b;           // float
v4.xy;          // vec2
v4.xgba;        // illegal (not from same set)
```

```
vec4 pos = vec4(1.0, 2.0, 3.0, 4.0);
vec4 swiz = pos.wzyx;       // swiz = (4.0, 3.0, 2.0, 1.0)
vec4 dup = pos.xxyy;        // dup = (1.0, 1.0, 2.0, 2.0)
```

Operators

[]	Index
.	Selection/swizzle
++ --	Postfix increment/decrement
++ --	Prefix increment/decrement
- !	Unary negation, not
* /	Multiply and divide
+ -	Add and subtract
> < >= <=	Relational
== !=	Equality
&&	Logical and
^^	Exclusive or
||	Inclusive or
?:	Selection
= += -= *= /=	Assignment

Component-wise Operation

```
vec3 v,u;
float f;
mat4 m;

v = u + f;      // component-wise addition
w = v + u;      // component-wise addition

v * u;          // component wise multiply
v * m;          // row-vector matrix mult.
m * v;          // matrix column-vector mult.
m * m;          // standard matrix mult.
```

Variable Names

Similar rules to C

Notable exception: gl_ __ (reserved)

Control Structures

Similar to C: **for**, **while**, **do-while**, **if**, **if-else**, **break**, **continue**

Not available: **goto**, **switch**

Functions

- Syntax similar to C/C++
- **return** supported
- No promotion of arguments, so matches must be exact.
- No recursion (direct or indirect)
- Call by value-return

Parameter Qualifiers

in	Copy in but don't copy out
out	Only copy out, undefined at entry to function
inout	Copy in and copy out

Built-in Functions

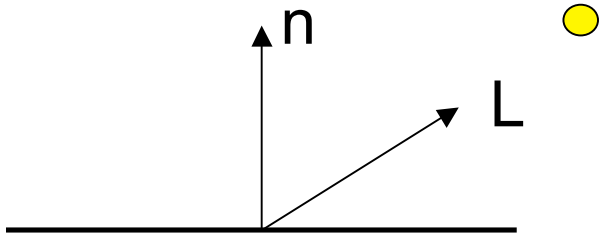
```
float length(float x)
float distance(vec3 p0, vec3 p1)
float dot( vec3 a, vec3 b)
vec3 normalize(vec3 x)
vec3 reflect( vec3 I, vec3 N)
vec3 mix(vec3 x, vec3 y, float a)
...
```

Cartoon Shader

In `Main.java`

```
ShaderDemo demo = new ToonShaderDemo(canvas);
```

Eye coordinates



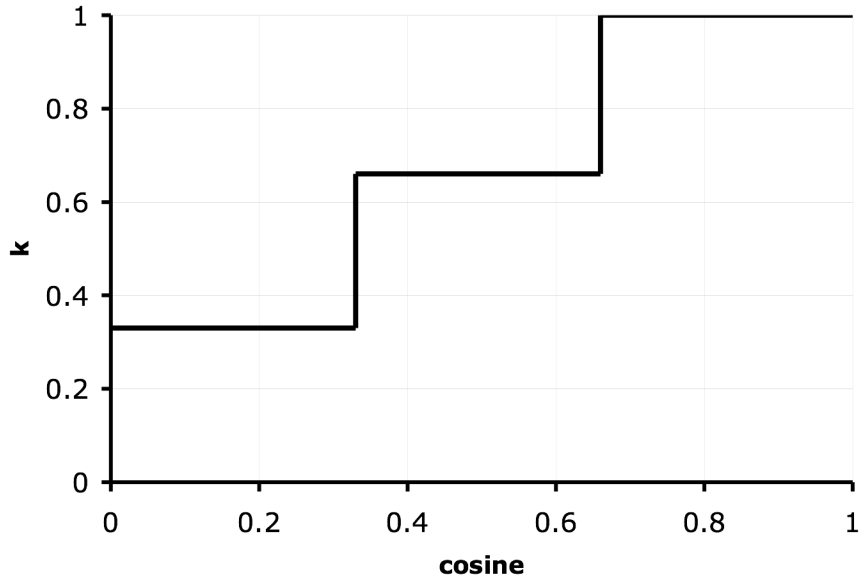
Diffuse Shading

$$I = \rho_d \max\left(\frac{\mathbf{n}\cdot\mathbf{L}}{|\mathbf{n}|\,|\mathbf{L}|}, 0\right)$$

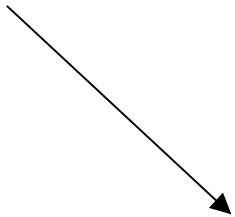
Cartoon

$$I = \rho_d k$$

$$k = \begin{cases} 0 & \frac{\mathbf{n}\cdot\mathbf{L}}{|\mathbf{n}||\mathbf{L}|} < 0 \\ 0.33 & 0 <= \frac{\mathbf{n}\cdot\mathbf{L}}{|\mathbf{n}||\mathbf{L}|} < 0.33 \\ 0.66 & 0.333 <= \frac{\mathbf{n}\cdot\mathbf{L}}{|\mathbf{n}||\mathbf{L}|} < 0.66 \\ 1.0 & \frac{\mathbf{n}\cdot\mathbf{L}}{|\mathbf{n}||\mathbf{L}|} >= 0.66 \end{cases}$$

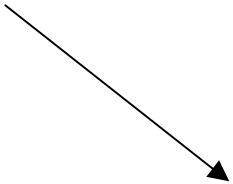


Compute cosine in
vertex shader



Interpolate using varying variable:

```
varying NdotL;
```



Compute shading
per-fragment

Vertex Shader (toon.vert)

```
varying float NdotL;   //  The cosine term

void main()
{
  // Get the position of the vertex in eye coordinates
  vec4 ecPos = gl_ModelViewMatrix * gl_Vertex;
  vec3 ecPos3 = (vec3(ecPos)) / ecPos.w;

  // The light position from OpenGL
  vec3 LightPosition = vec3(gl_LightSource[0].position);

  // Transform and normalize the normal
  vec3 tnorm = normalize(gl_NormalMatrix * gl_Normal);

  // The vector from the vertex to the light source
  vec3 lightVec = normalize( LightPosition - ecPos3 );

  // Compute the cosine term
  NdotL = dot(lightVec, tnorm);

  gl_Position = ftransform();
}
```

Fragment Shader (toon.frag)

```
varying float NdotL;  // Interpolated over the face

void main()
{
  vec3 SurfaceColor = vec3(gl_FrontMaterial.diffuse);

  // Produces the stair step pattern
  float scale = ceil( 3.0 * NdotL ) / 3.0;

  gl_FragColor = vec4( SurfaceColor * scale, 1.0 );
}
```

Black Lines on Edges

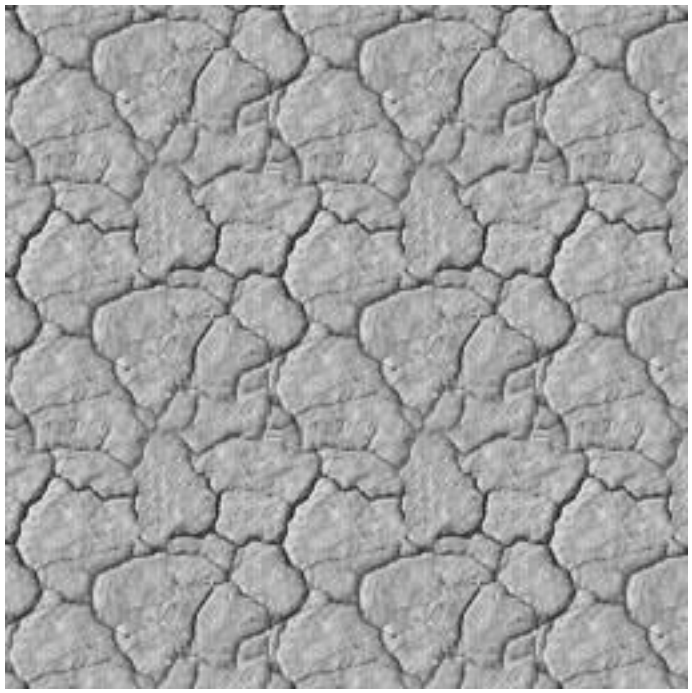
- Pass 1:
  - shader enabled
  - backface culling
- Pass 2:
  - Shader disabled
  - Lines (fill off)
  - Depth function: <=
  - Cull front faces

Normal-Map Shader

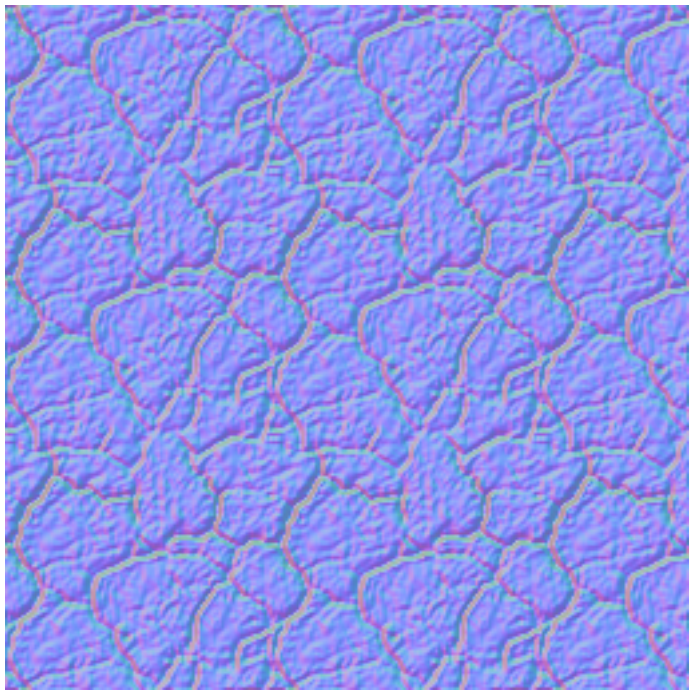
In `Main.java`

```
ShaderDemo demo = new
                  NormalMapShaderDemo(canvas);
```

Height (Bump) Map



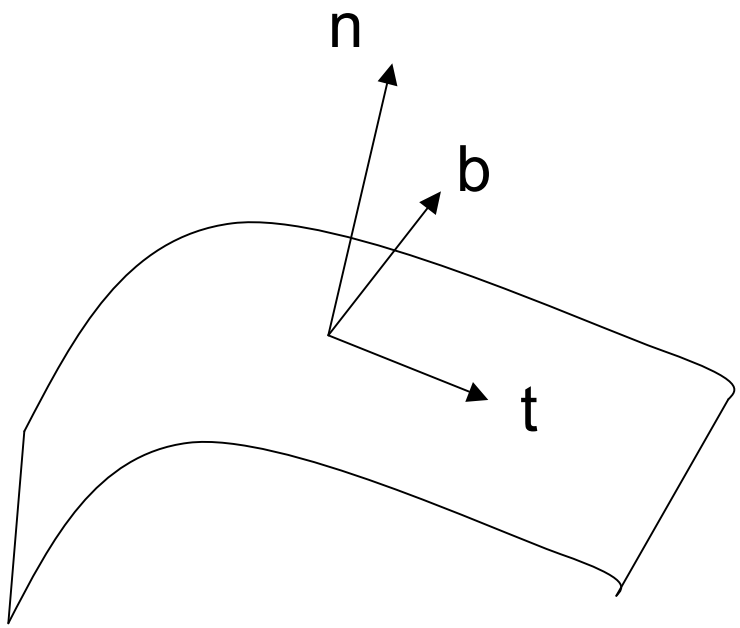
Normal Map



$(n_x, n_y, n_z) \rightarrow (r, g, b)$

Surface Local Frame

Z-axis parallel to unperturbed normal



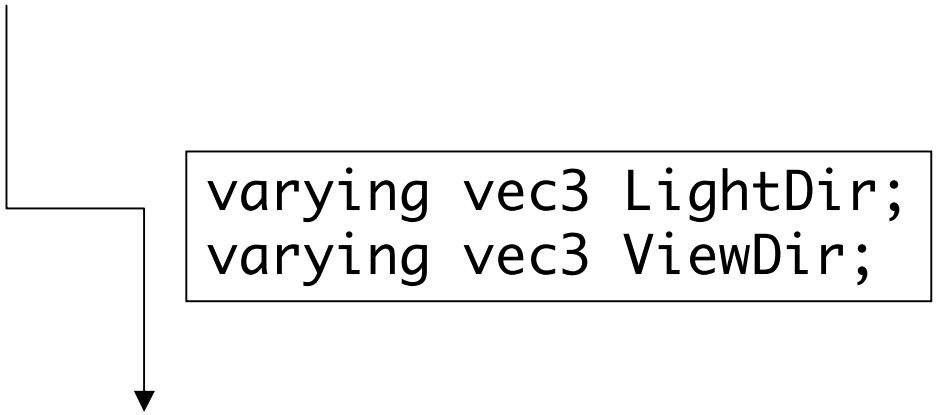
Transform light vector, view vector to surface frame

Vertex Shader

Compute surface local frame in eye coordinates.

```
attribute vec3 objTangent;
```

Convert light and view directions to surface
Local frame.



Fragment Shader

- Lookup normal in texture and perturb normal.
- Compute Phong shading, using texture for diffuse

Shader

```
// Tangent vector in modeling coordinates
attribute vec3 objTangent;
```

Application Code

```
int attrib = glGetAttribLocation(id, “objTangent”);

glTexCoord2d(s,t);
glVertexAttrib3d(attrib, tx, ty, tz);
glNormal3d(nx, ny, nz);
glVertex3d(x,y,z);
```

Shader

```
// sampler2D is a “texture type”
uniform sampler2D normalMap;
… …
vec4 val = texture2D( normalMap, texCoord );
```

Application Code

```
glActiveTexture(GL_TEXTURE0);

// Load texture

… …

int loc = glGetUniformLocation(id, “normalMap”);
glUniform1i(loc, 0);
```

Vertex Shader (`bump.vert`)

```
varying vec3 LightDir;      // Light direction
varying vec3 ViewDir;       // View direction
attribute vec3 objTangent;  // Tangent vector

void main() {
  vec3 eyePosition = vec3( gl_ModelViewMatrix * gl_Vertex );
  vec3 eyeLightDir = vec3(gl_LightSource[0].position) -
      eyePosition;

  vec3 eyeNormal = normalize( gl_NormalMatrix * gl_Normal );
  vec3 eyeTangent = normalize( gl_NormalMatrix * objTangent );
  vec3 eyeBinormal = cross( eyeNormal, eyeTangent );

  // Convert eyeLightDir to tangent space (LightDir)

  // Convert view direction (-eyePosition)
  //   to tangent space (ViewDir)

  gl_TexCoord[0] = gl_MultiTexCoord0;
  gl_Position = ftransform();
}
```

Fragment Shader (`bump.frag`)

```
varying vec3 LightDir;      // Light direction
varying vec3 ViewDir;       // View direction

uniform sampler2D normalMap;  // The normal map texture
uniform sampler2D colorTex;   // The color texture

void main() {
  vec4 normal4 = texture2D(normalMap, gl_TexCoord[0].st );
  vec3 normal = 2.0*normal4.xyz - 1.0;

  // Compute Phong lighting here, store in diffuse and spec

  // This gets the diffuse color from a color texture
  vec4 col = texture2D(colorTex, gl_TexCoord[0].st );

  gl_FragColor = diffuse * col +
                 spec * vec4(0.8,0.8,0.8,1.0);

}
```

Refraction/ Reflection Shader

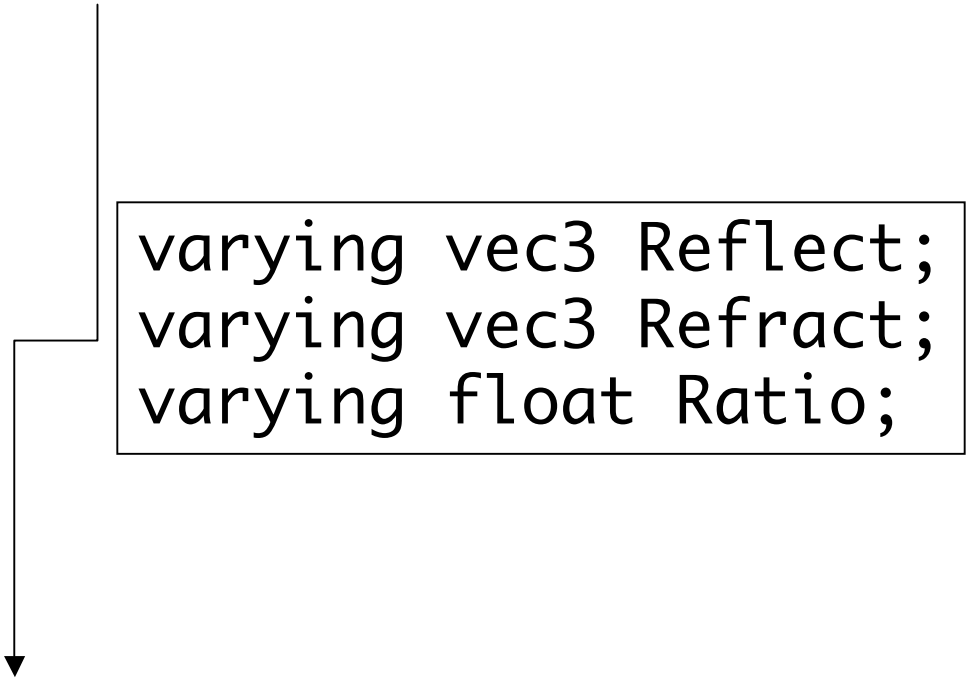
In `Main.java`

```
ShaderDemo demo = new
                  RefractionShaderDemo(canvas);
```

Vertex Shader

Compute Refraction and reflection in eye coordinates.

Compute fresnel ratio



Fragment Shader

Cube-map lookup

Mix refraction and reflection using ratio

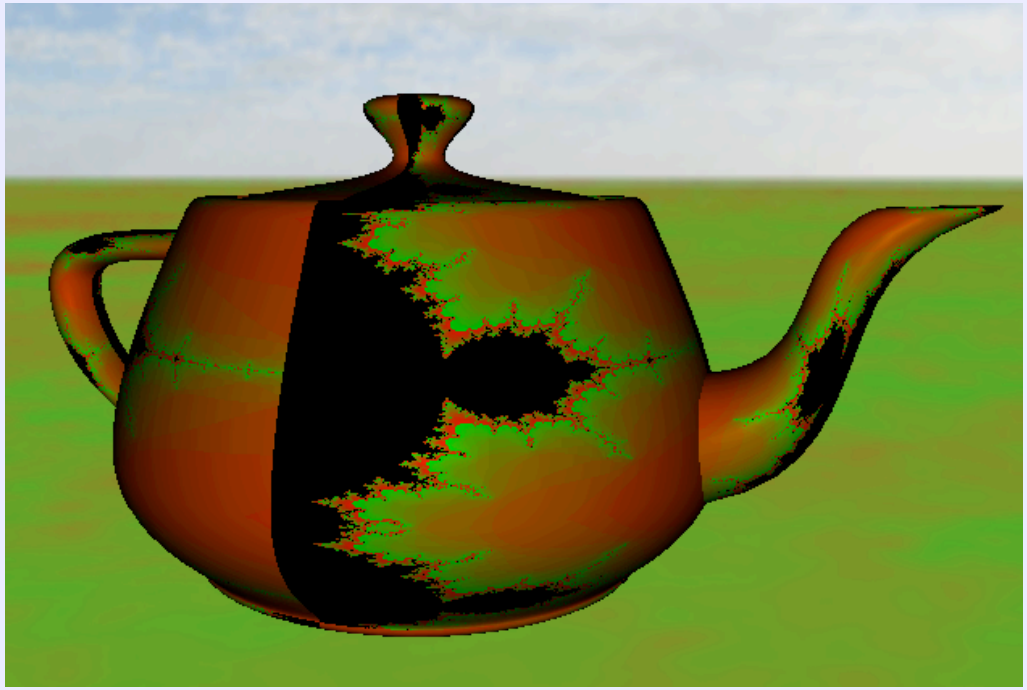
Vertex Shader (`transparent.vert`)

```
// Declaration of constants involving IOR and Fresnel

varying vec3 Reflect;
varying vec3 Refract;
varying float Ratio;

void main() {
  // Convert position (i) and normal (n) to eye coordinates

  Ratio = F + (1.0 - F) * pow((1.0-dot(-i, n)), FresnelPower);

  Refract = refract(i, n, Eta);
  Refract = vec3(gl_TextureMatrix[0] * vec4(Refract, 1.0) );

  Reflect = reflect(i,n);
  Reflect = vec3(gl_TextureMatrix[0] * vec4(Reflect, 1.0) );

  gl_Position = ftransform();
}
```

Fragment Shader (`transparent.frag`)

```
varying vec3 Reflect;
varying vec3 Refract;
varying float Ratio;

uniform samplerCube Cubemap;

void main()
{
  vec3 refractColor = vec3(textureCube(Cubemap, Refract));
  vec3 reflectColor = vec3(textureCube(Cubemap, Reflect));

  vec3 color = mix( refractColor, reflectColor, Ratio );

  gl_FragColor = vec4(color, 1.0);
}
```

Mandelbrot Set Shader



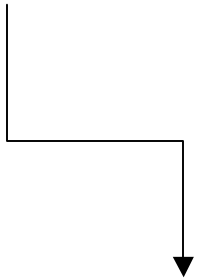
In `Main.java`

```
ShaderDemo demo = new
                  MandelbrotShaderDemo(canvas);
```

Vertex Shader

Compute position on complex plane using texture coordinates, and vertex lighting.

Pass to fragment shader via varying variables.



Fragment Shader

Compute Mandelbrot set membership and color, mix with basic Phong shading.

Vertex Shader (mandelbrot.vert)

```
uniform vec3 LightPosition;
// Other uniforms (material properties)

varying float LightIntensity;
varying vec3 ComplexPosition;

void main()
{
 vec3 ecPosition = vec3(gl_ModelViewMatrix*gl_Vertex );
 vec3 tnorm = normalize(gl_NormalMatrix * gl_Normal );
 vec3 lightVec = normalize(LightPosition - ecPosition );

 // Phong specular calculation here (spec) …

 LightIntensity =
   DiffuseContribution * max(dot(lightVec, tnorm),0.0) +
   SpecularContribution * spec;
 ComplexPosition = vec3(gl_MultiTexCoord0 - 0.5) * 5.0;
 gl_Position = ftransform();
}
```

Fragment Shader
(mandelbrot.frag)

```
varying vec3 ComplexPosition;
varying float LightIntensity;

uniform int MaxIterations;
// Other uniform delcarations omitted…

void main()  {
  // Variable declarations here…

  for(iter = 0; iter < MaxIterations && r2 < 4.0; ++iter) {
    // Compute next iteration of Mandelbrot series…
  }

  vec3 color;
  if( r2 < 4.0 ) { color = InnerColor; }
  else {
    color = mix(OuterColor1, OuterColor2,
      fract( float(iter) * 0.05 ) );
  }

  color *= LightIntensity;
  gl_FragColor = vec4(color, 1.0);
}
```

Shader Debugging Practices

- Use vertex shader output
- Use fragment shader output
- Use simple geometry

Shader Development Tools

- RenderMonkey (ATI/AMD)
- GLMan (Mike Bailey, OSU)
- Apple's OpenGL Shader Builder (Mac OSX)
- ?? Eclipse Plug-in for Education ??

Demo of Eclipse Plug-in