Exercise 1. Implement basic lighting model in GLSL shaders

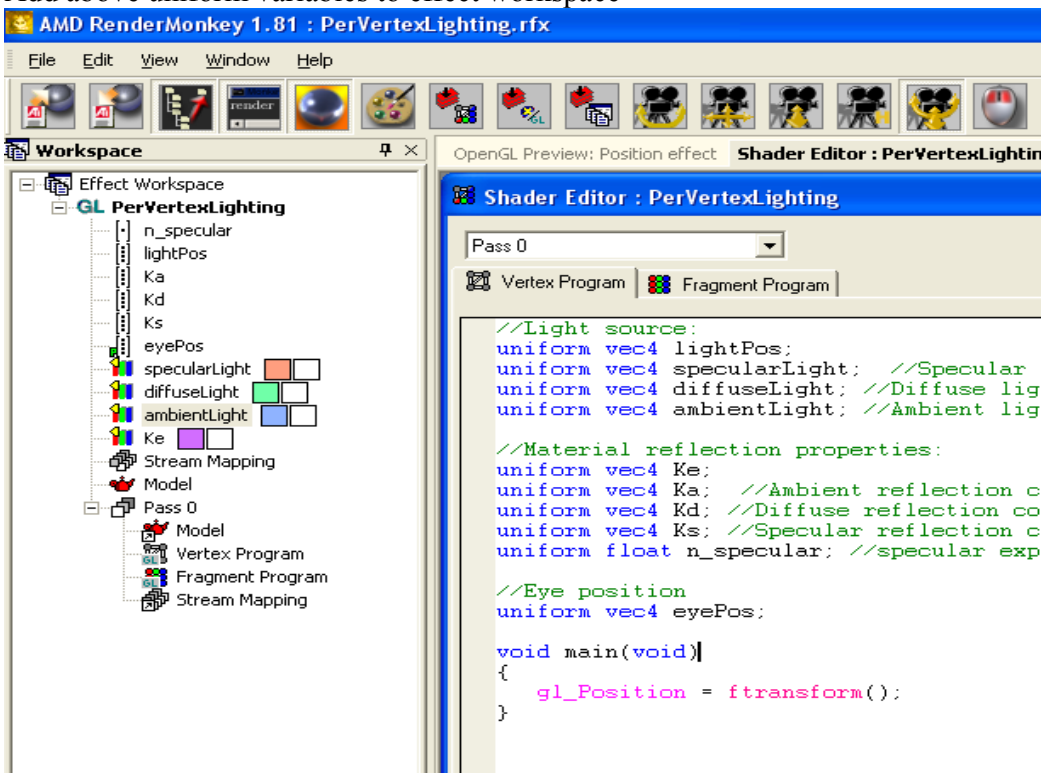
1. Per-vertex lighting
   1) Open the default OpenGL **Position effect**. Rename the effect as PerVertexLighting.
   2) Open Vertex program and insert the following uniform variables:

```
//Light source:
uniform vec4 lightPos;
uniform vec4 specularLight;
uniform vec4 diffuseLight;
uniform vec4 ambientLight;

//Material reflection properties:
uniform vec4 Ke;
uniform vec4 Ka;
uniform vec4 Kd;
uniform vec4 Ks;
uniform float n_specular;

//Eye position
uniform vec4 eyePos;
```

   3) Add above uniform variables to effect workspace



.

4) Double click each variables and set proper values for these variables. The values for material coefficients Ka, Kd, Ks should be between 0 and 1. The value for n_specular should be between 1 and 200. You can also set the semantic for eyePos as ViewPosition by left clicking the name of the variable.
5) Add a varying variable to both the vertex shader and the fragment shader:

```
varying vec4 ColorAtVertex;
```

6) Insert the necessary lines of code into the vertex shader shown in my lecture notes to calculate the colour for each vertex.
7) Set the gl_FragColor to be **ColorAtVertex** in the fragment shader.
8) Save your effect workspace and recompile your program. You should now see an illuminated sphere. If the image does not look right. Tune the image by changing the variable values.
9) Change the model and observe the quality of the rendered image.

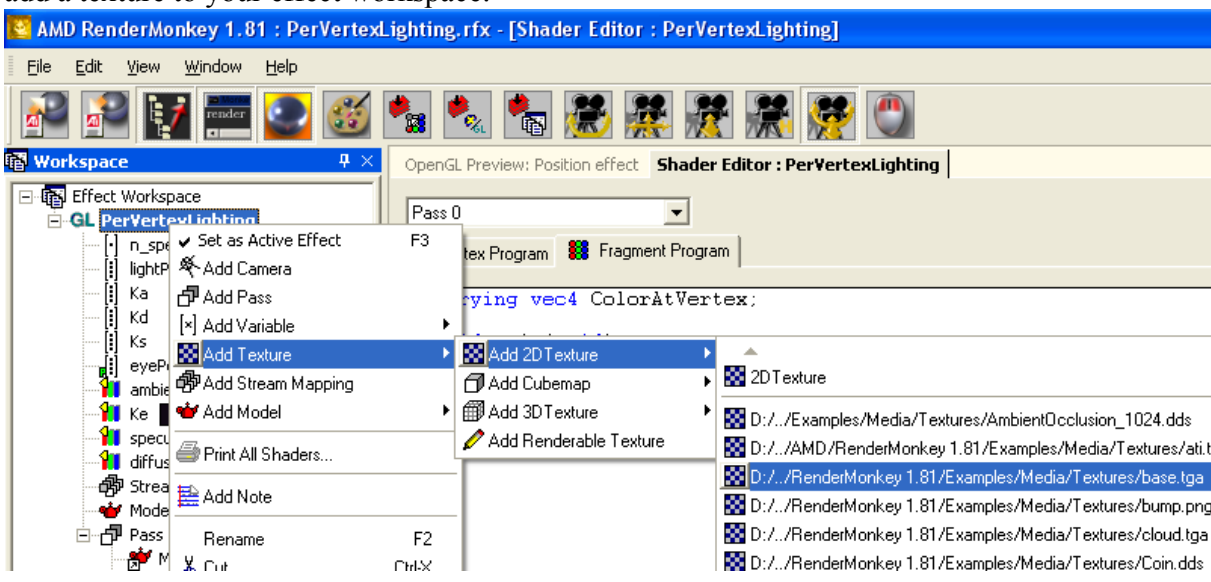
2. Per-pixel lighting

Once you have successfully get per-vertex lighting implemented, create a new effect and name it as perPixelLighting, then implement your light model in the pixel shader

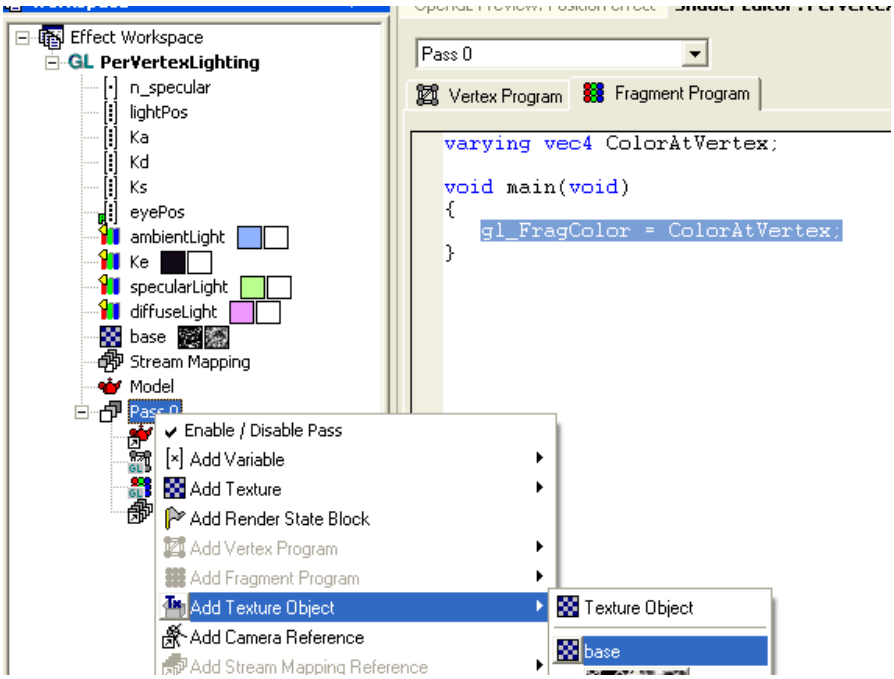
Exercise 2. Texture mapping

Start with the light effects you have created in Exercise 1, either per-vertex lighting effect or per-fragment lighting effect. :

1) add a texture to your effect workspace:



2) Click Pass 0 and add a texture object:



3) Rename the texture object as MyTextureObj and associate it with the added texture in step 1;
4) in your pixel shader, declare a sampler2D variable corresponding to the texture object:
   uniform sampler2D MyTextureObj;

5) Insert in both the **vertex shader** and the **fragment shader** a varying variable to pass on the texture coordinate to fragment shader:
   varying vec2 texCoord;

6) In the vertex shader, insert the following line of code:
   texCoord = gl_MultiTexCoord0.xy;

7) In the body of fragment shader, insert the following code to modify the gl_FragColor:
   vec4 texColor = texture2D(MyTextureObj, texCoord);
   gl_FragColor = ColorAtVertex * texColor;
8) Save your work and recompile the effect, you should now see a textured graphics effect.
9) Try some other ways to using the texture colour, such as replacing the light colour with texture colour, modifying or blending texture colour with the light colour.
10) Try difference textures.

Exercise 3. Bump mapping using a normal map.

1) Starting from your **per-fragment lighting**

a) In Vertex shader declare the following variables:

```
uniform vec3 LightPos;
uniform vec3 EyePos;

varying vec2 Texcoord;
varying vec3 ViewDirection;
varying vec3 LightDirection;
varying vec3 vNormal;

attribute vec3 rm_Binormal;
attribute vec3 rm_Tangent;
```

b) In vertex shader, find normal, view direction and light direction in the view space:

```
gl_Position = ftransform();
Texcoord    = gl_MultiTexCoord0.xy;

vec4 Pos = gl_ModelViewMatrix * gl_Vertex;

vec3 vViewDirection  = EyePos - Pos.xyz;
vec3 vLightDirection = LightPos - Pos.xyz;

vNormal        = gl_NormalMatrix * gl_Normal;
vec3 vBinormal      = gl_NormalMatrix * rm_Binormal;
vec3 vTangent       = gl_NormalMatrix * rm_Tangent;
```

c) Transform these vectors into tangent space using the following matrix :

```
mat3 View2Tangent = mat3(vTangent.x, vBinormal.x, vNormal.x,
                         vTangent.y, vBinormal.y, vNormal.y,
                         vTangent.z, vBinormal.z, vNormal.z);

ViewDirection =  View2Tangent *  vViewDirection;
LightDirection =  View2Tangent *  vLightDirection;
```

2) In fragment shader, load and define bump texture:

```
uniform sampler2D bumpMap;
```

3) Use the texture as the normal map by converting it in the following way:

```
vec3 N = normalize(   2.0*texture2D( bumpMap, Texcoord ).xyz - 1.0  );
```

4) Apply the basic lighting model using normal vector **N**.
5) Recompile your effect, you should see a bumpy graphics object.
6) You can now introduce a uniform variable to specify the bump density.

==========

Cantact: q.li@hull.ac.uk Dr Qingde Li
Department of computer science University of Hull