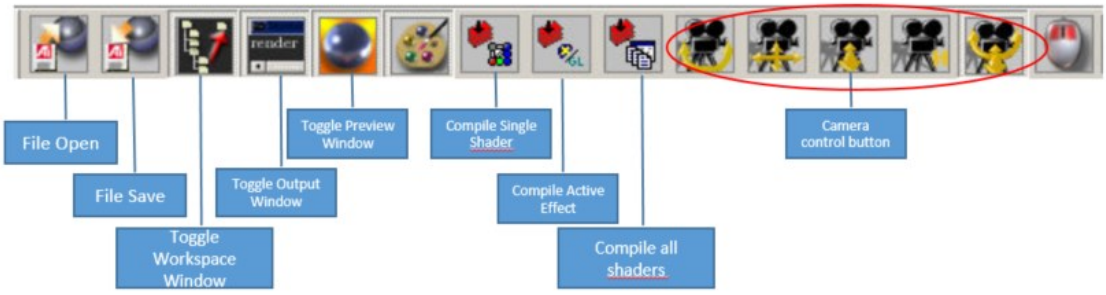
The aim of this lab is to develop practical experience in using the RenderMonkey shader programming IDE. You can find the software by searching RenderMonkey from the Windows start menu. You can also directly go to the folder C:\Program Files (x86)\AMD\RenderMonkey 1.82 to find the program. The software is free and can be downloaded from GPUOpen.com at https://gpuopen.com/archive/gamescgi/rendermonkey-toolsuite/.

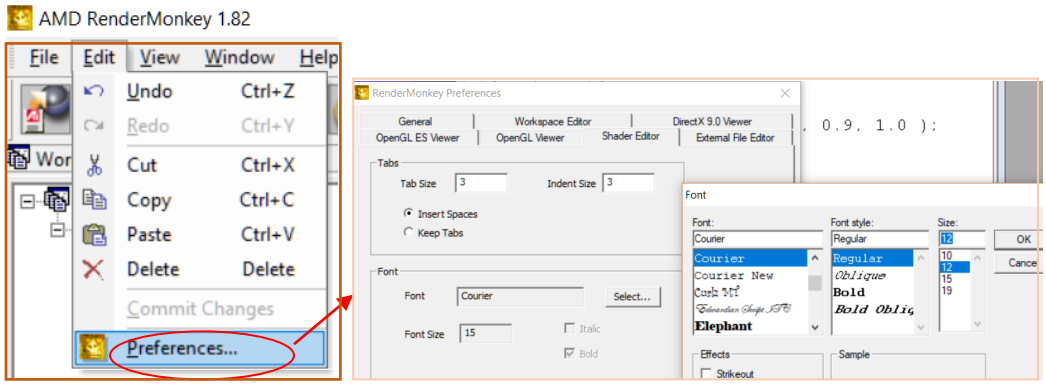
Exercise 1. Understand the basic RenderMonkey features Whi

One essential task in graphics programming is to create and organize graphics asset such as geometric models and textures. One advantage of RenderMonkey is that it allows shader developers directly load geometric models and textures represented in some commonly used formats. There are many other useful features for the software, please read its documentation for details.

1. Run RenderMonkey
2. Familiar yourself with the software by reading the RenderMonkey documentation, which can be found from online at GPUOpen.com or locally in the folder C:\Program Files (x86)\AMD\RenderMonkey 1.82.
3. Browsing RenderMonkey samples
   a) From the main menu, click File→open or directly use shot-cut key "Ctrl+O"
   b) Go to C:\Program Files (x86)\AMD\RenderMonkey 1.82\Examples
   c) Then view the graphics effects in folder GL2 and Advanced. If you want to see some effects created in DirectX HLSL, you can have a look at the examples shown in the DX9 folder.
4. Familiar yourself with the Application Toolbar

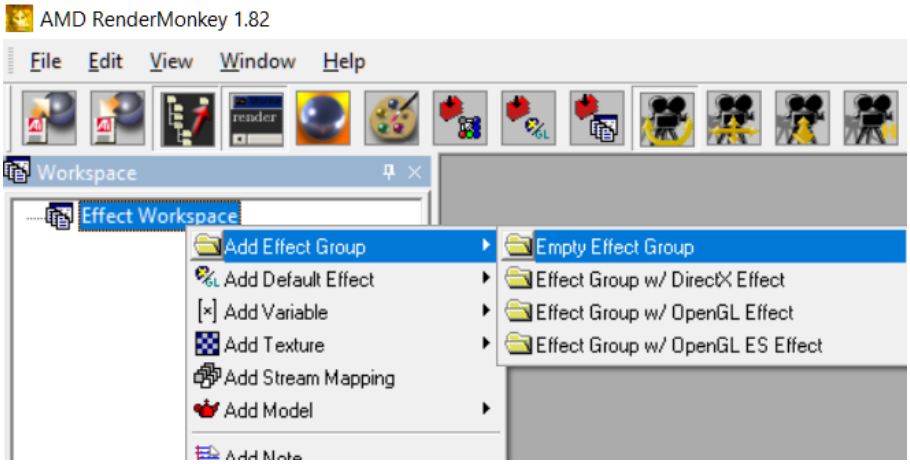


5. You can also edit the RenderMokey preference to modify the application settings (Please refer RenderMonkey documentation for details).

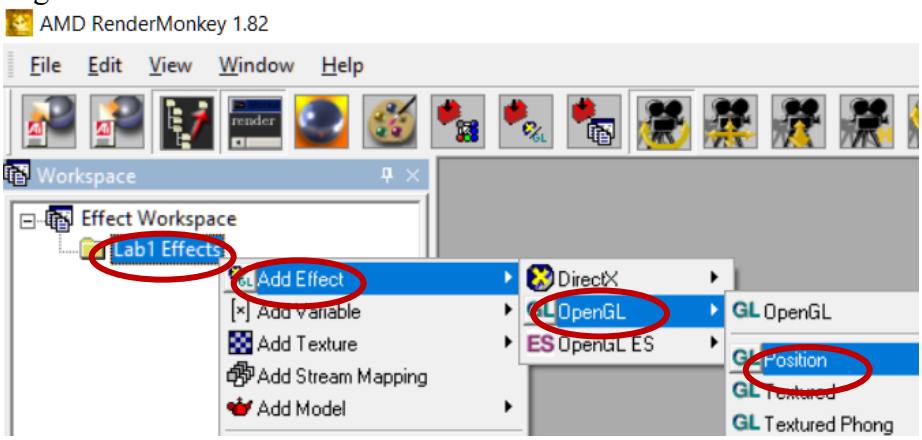


Exercise 1. Create your first GLSL effect

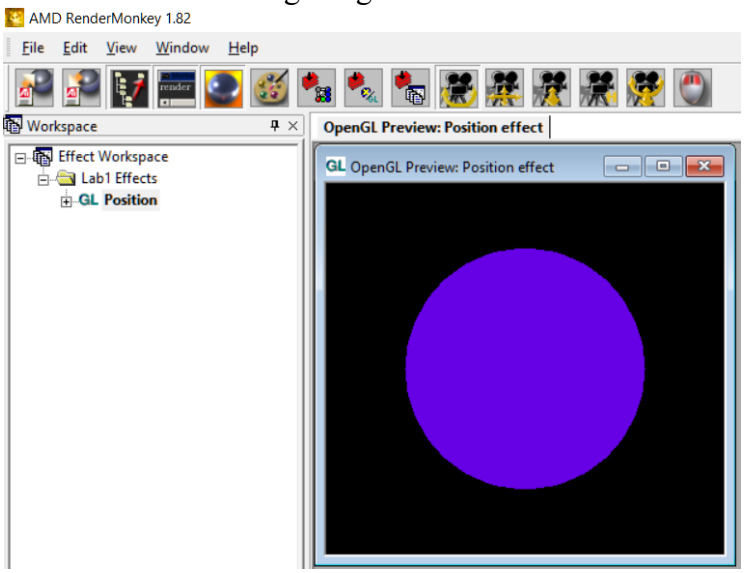
1. Run RenderMonkey
2. Create an OpenGL effect group by right clicking on Effect Workspace and select Add Effect Group, and then Empty Effect Group.



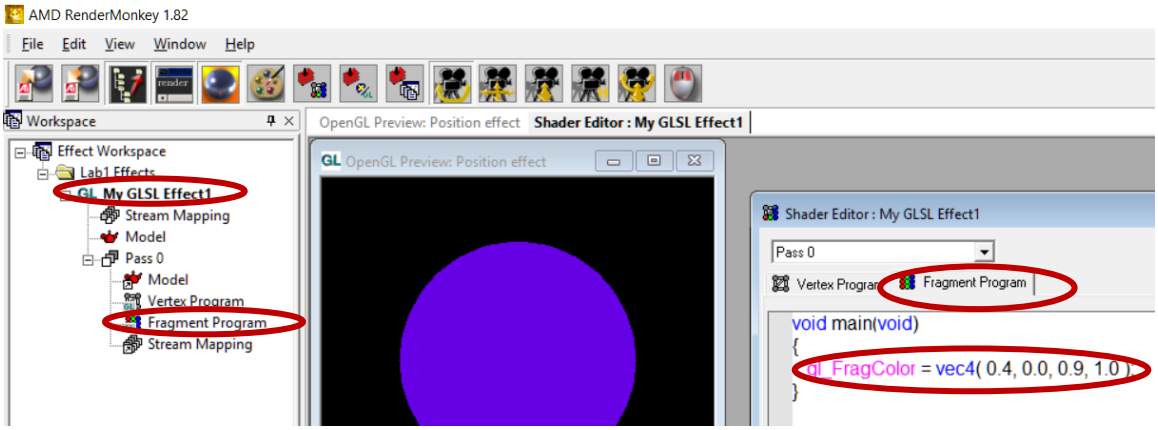
3. Right click on the default group name and rename it as, say, Lab1 effects.
4. Right click on "Lab1 Effects" and select Add Default Effect.



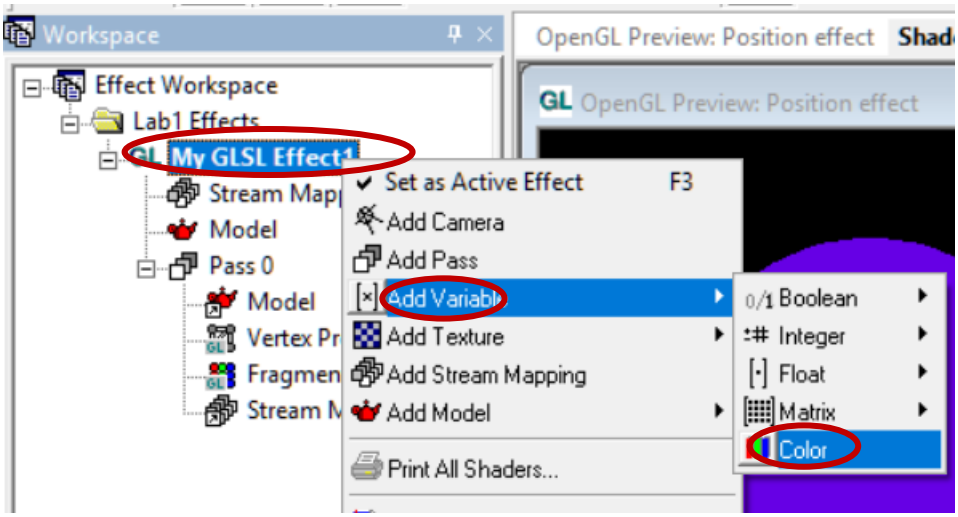
5. Select an OpenGL effect, say, the Position effect.
6. You should now see the following image



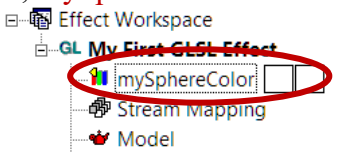
7. Right click on "Position" in the Effect Workspace (or press F2) to rename it as "My GLSL Effect1".
8. Click the '+' to expand the workspace.
9. Double click the "Fragment Program" node in the Effect Workspace to open the fragment program.



10. Change the fragment colour value gl_FragColor from vec4(0.4, 0.0, 0.9, 1.0) to vec4(1.0, 0.0, 0.0, 1.0) and then recompile the shader. You should now see a red sphere.
11. Now add a new colour variable in the workspace by right clicking on the effect name "My First GLSL Effect" and subsequently select Add Variable and Color:

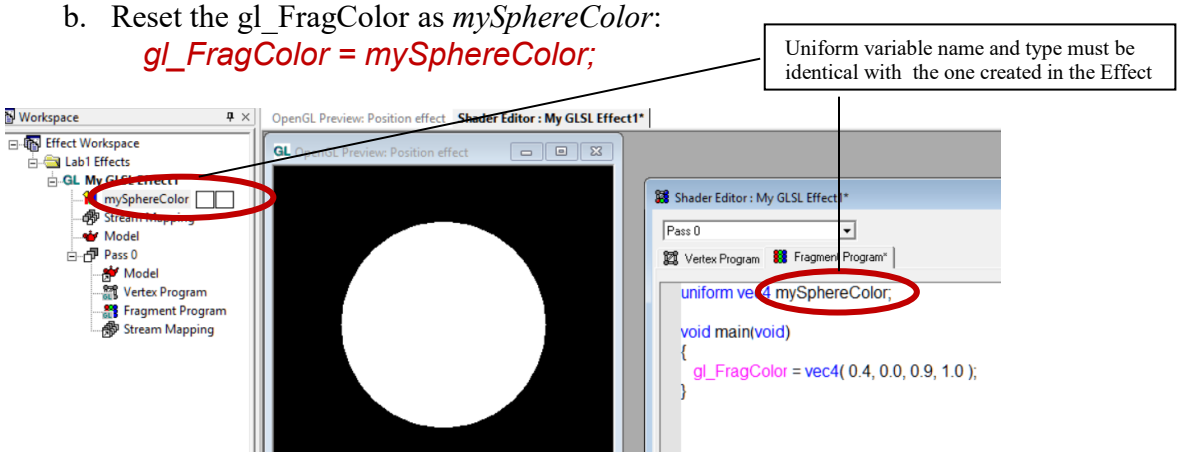


12. You should now have a new variable with default name "myColor". Rename it with a name you like, for instance, mySphereColor.



13. Open and edit the fragment shader
    a. Add a uniform variable to the fragment shader. The type and the name of the variable must match the one you have created:
       uniform vec4 mySphereColor;

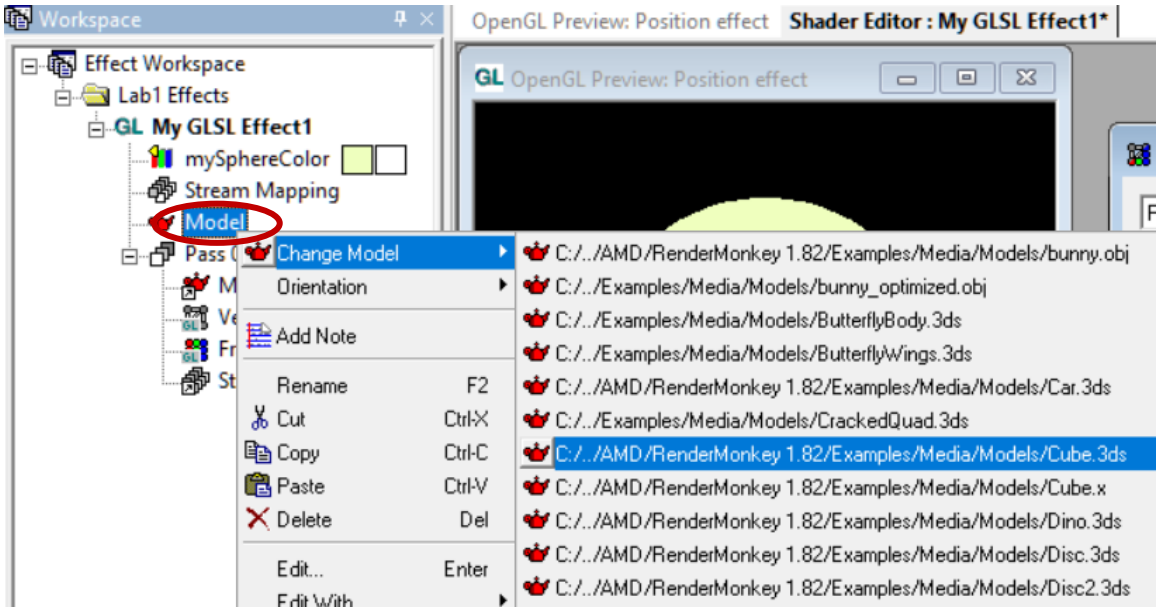
    b. Reset the gl_FragColor as *mySphereColor*:
       gl_FragColor = mySphereColor;



14. Recompile your shader, then double click on the colour variable name, a colour pallet will popup. You can now change the colour of the sphere by moving your mouse (with left button down) over the opened colour pallet.

15. Change geometric model. You can replace the geometric model and replace it with other models provided by the RenderMonkey tool by right-clicking "Model" node (see the following figure). If you would like to use your own model, simply double

click on the model name, then a file search window will pop up for you to load your model.



Exercise 2. Implement scaling transformation in vertex shader

1. Open Vertex Program (you should now know how to do this!)
2. The first thing you need to do is to replace the following line of code

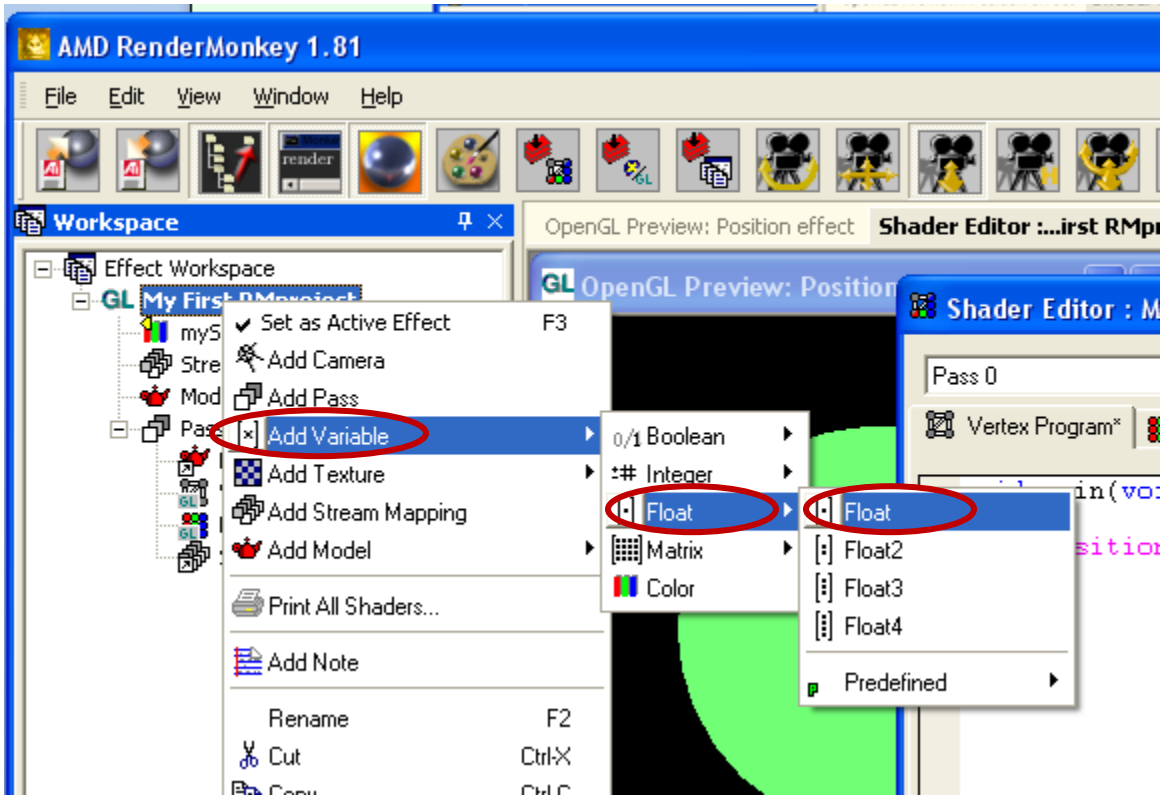
```
gl_Position = ftransform();
```

with the following two lines of code

```
vec4 inPos=gl_Vertex;     //create a vec4 variable to store vertex position
gl_Position = gl_ModelViewProjectionMatrix * inPos;
```

The two lines of GLSL code do the same thing as *ftransform()*, which transforms a vertex from world space into the clipping space. To be able to change the shape of a geometric object, we must declare a variable to capture the coordinates of a vertex's position, which is initially stored in gl_Vertex. Recompile the GLSL code, you should see exactly the same effect.

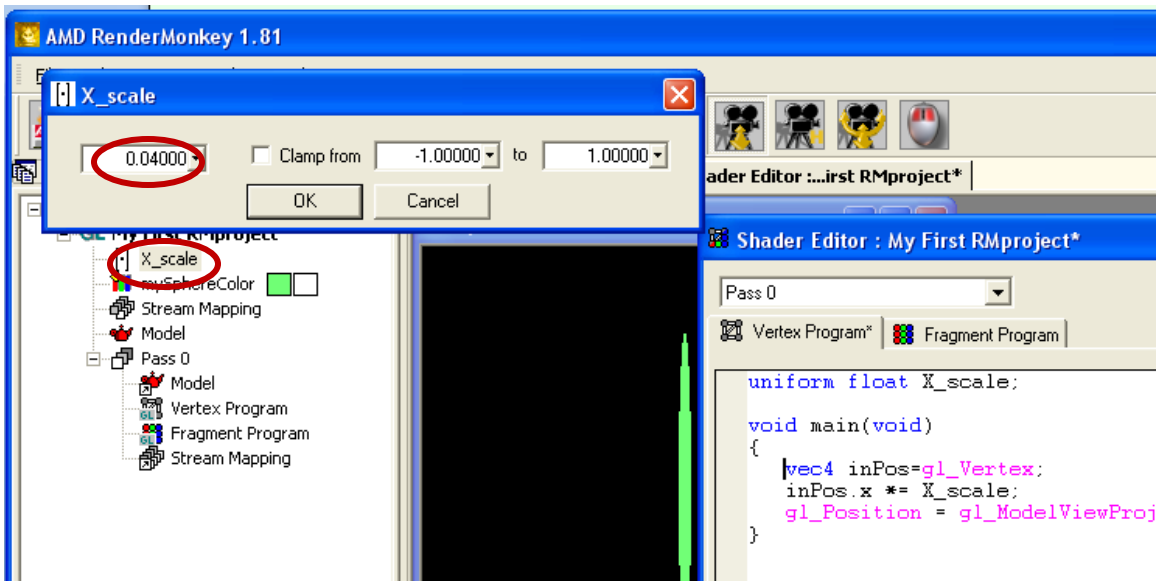
3. Add a new variable in the workspace for specifying the amount of value for scaling the given geometric shape along the x-axis and name it as "X_scale".



4. Modify the vertex position stored in variable inPos by scaling the x-coordinate with a value stored in X_scale:

```
uniform float X_scale;

void main(void)
{
   vec4 inPos=gl_Vertex;     //store the vertex position to inPos
   inPos.x *= X_scale;       //scaling the x-coordinate of the vertex

   //Transform the changed vertex position into clipping space
   gl_Position = gl_ModelViewProjectionMatrix * inPos;
}
```

5. Recompile your shader program.
6. Double click on the variable name "X_scale". You can now scale the shape of the geometric shape along the x-axis by change the X_scale value.



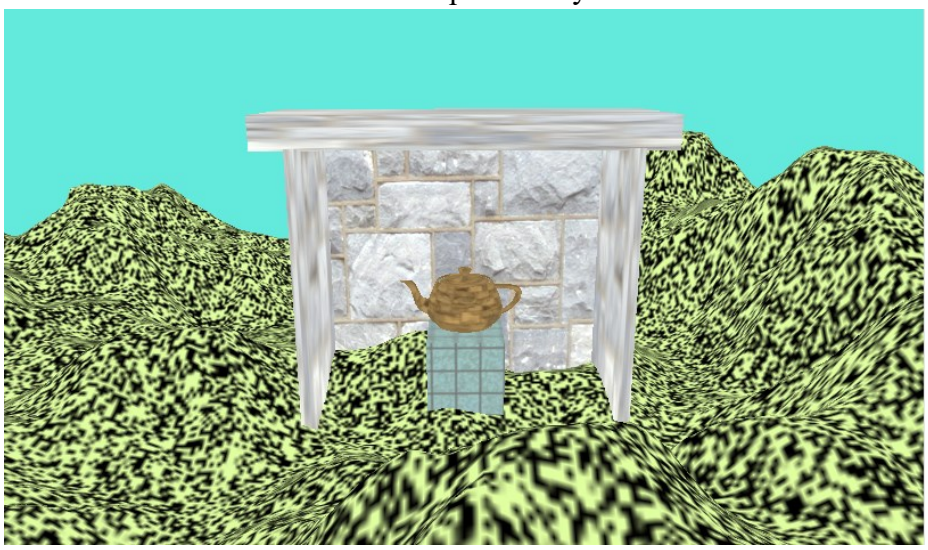
Exercise 3. Implement scaling transformation along y-axis and z-axis in vertex shader. Instead of adding three float type variables, you can add one single 3D vector type variable , Scale, by selecting variable type "float3". You can scale x, y, z components in the following way:

…
*inPos.xyz *= Scale;*
…

Exercise 4. Implement translation and rotation transformations in vertex shader. This can be done in a similar way as you do scaling transformation. To implement rotation, you need to know how to represent a rotation in matrix form, which can be found directly online or from my lecture notes shown in 08214. For example, the rotation by z-axis can be represented by a 4x4 matrix, which can be written as a GLSL function.

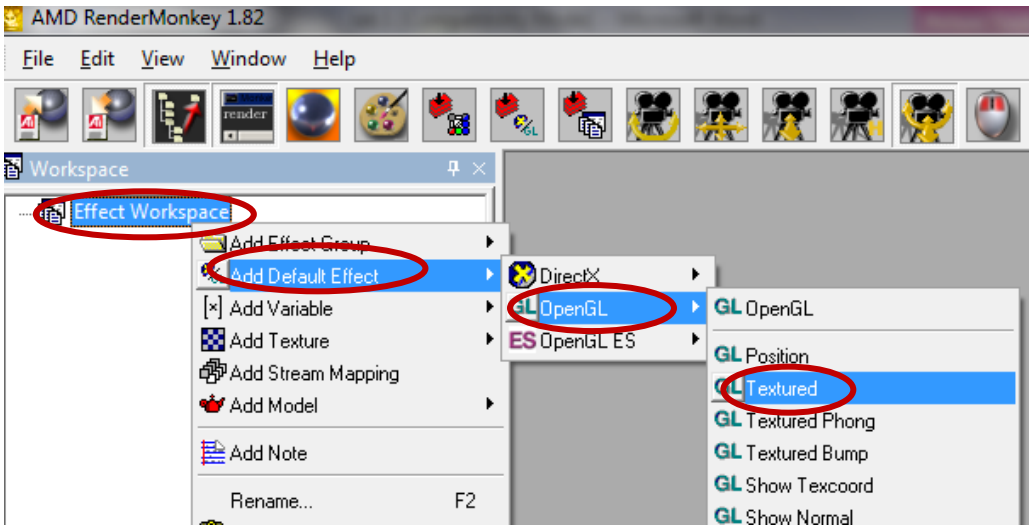
```
mat4 rotZ(float angle)  //angle in radians
{
    float s = sin(angle);
    float c = cos(angle);

    return mat4(
      c,-s, 0, 0,
      s, c, 0, 0,
      0, 0, 1, 0,
      0, 0, 0, 1);
}
```

Exercise 5. Add some more passes to your effect to create the following scene.



Exercise 6. Manage render state.

1. Run RenderMonkey and add a default effect, say, the Textured effect:



2. In Pass 0, add a colour type variable myColor.

3. open the Fragment Program and add myColor as a uniform variable:

```
uniform sampler2D baseMap;
varying vec2 Texcoord;
```

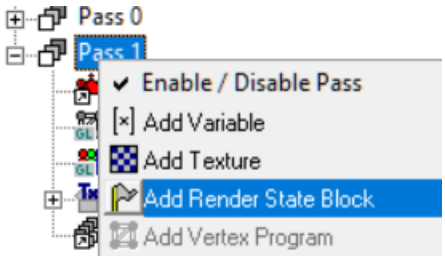
```
uniform vec4 myColor;

void main( void )
… …
```

(1). In Fragment Program, change the texture colour with the colour specified by the variable myColor by modifying the original line of code in the following way:

```
gl_FragColor = myColor * texture2D( baseMap, Texcoord ) ;
```

Now, recompile the shader and double click the variable myColor shown in the Workspace, and change its value, you will see the sphere colour being changed correspondingly.

(2). Add a new geometric model to the effect, say, Cube.x.
(3). Create a copy of Pass 0, the default name for the new pass should be "Pass 1".
(4). Expend Pass 1 and change the model rendered in Pass 1 to be the newly added Cube.
(5). By now you have rendered two objects, one sphere and one cube. We can blend the two effects together by configuring the render state. In Pass 1, add the render state node by right click "Pass 1" and select "Add Render State Block".



(6). There are many ways to blend the two effects. Here we show how to blend them using the value specified in the alpha-channel (alpha-blending) of source colour. Double click "Render State" node and configure the render state according to the diagram shown below:

GL_BlendDestAlpha	...
GL_BlendDestRGB	INV_SRC_ALPHA
GL_BlendEnable	TRUE
GL_BlendEquation	...
GL_BlendSourceAlpha	...
GL_BlendSourceRGB	SRC_ALPHA
GL_ClearColor	...

(7). Add a new float type variable to the effect FadeParam.
(10).Open the fragment shader in Pass 1 and add a new uniform variable FadeParam:

uniform float FadeParam;

(11). Reset the alpha-component for fragment colour gl_FragColor using the value specified in FadeParam:

gl_FragColor.a= FadeParam;

(12). Recompile the shader and double click the variable FadeParam shown in the Workspace, and specify its value to be between 0 and 1, you should be able to see a blending of the sphere effect the cube effect.

Written by Dr Qingde Li(q.li@hull.ac.uk). 04 March 2019.