Part 1

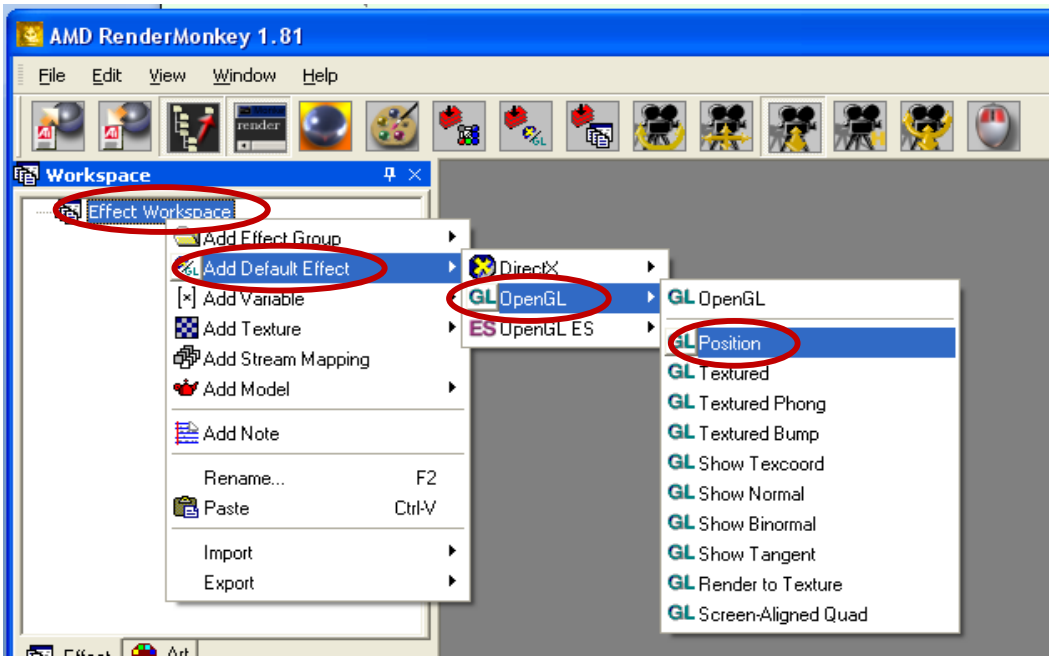
The aim of this workshop is to experience the use of RenderMonkey shader programming IDE.

1. Run RenderMonkey tutorials, which can be downloaded from http://www.createdbyx.com/vid-Programming__ATI+Render+Monkey+basics-Media.aspx , to learn how to use the RenderMonkey IDE.
2. Refer to the "RenderMonkey Documentation.pdf" in the directory of RenderMonkey for more detailed description about the software.
3. The above tutorial is about HLSL, but the shader programming language we are going to learn is GLSL. Following the instruction presented in the tutorial, try similar things about GLSL.
4. Read GLSL tutorial presented at http://www.lighthouse3d.com/opengl/glsl/index.php?intro for how to integrate GLSL in your application. You can download and try a sample from http://www.lighthouse3d.com/opengl/glsl/examples/glutglsl_2.0.zip.

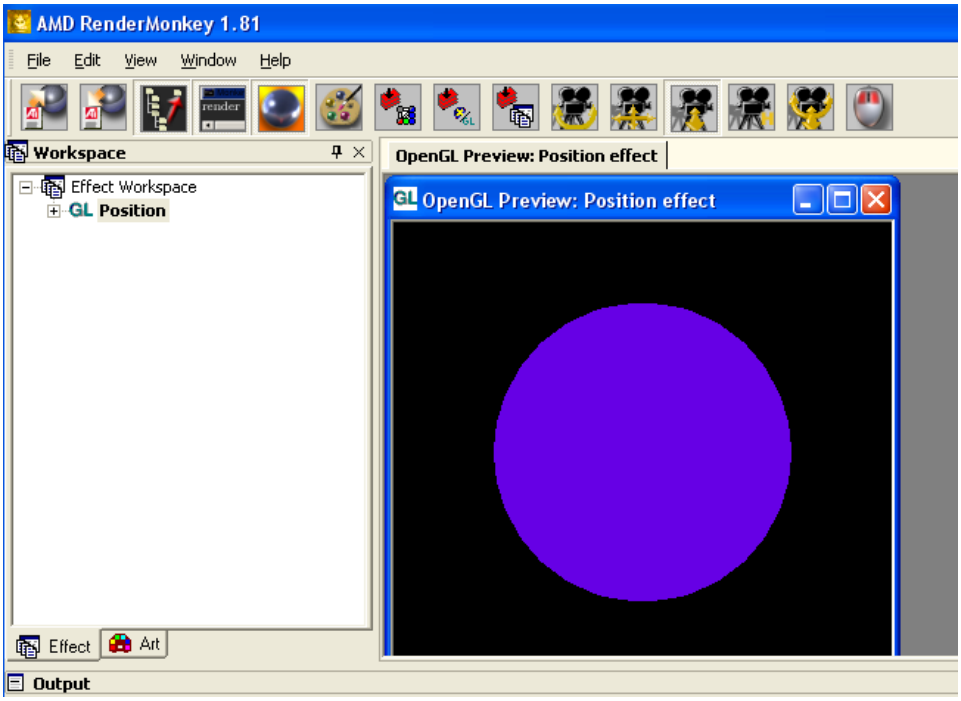
Part 2

Exercise 1. Create your first Rendermonkey effect

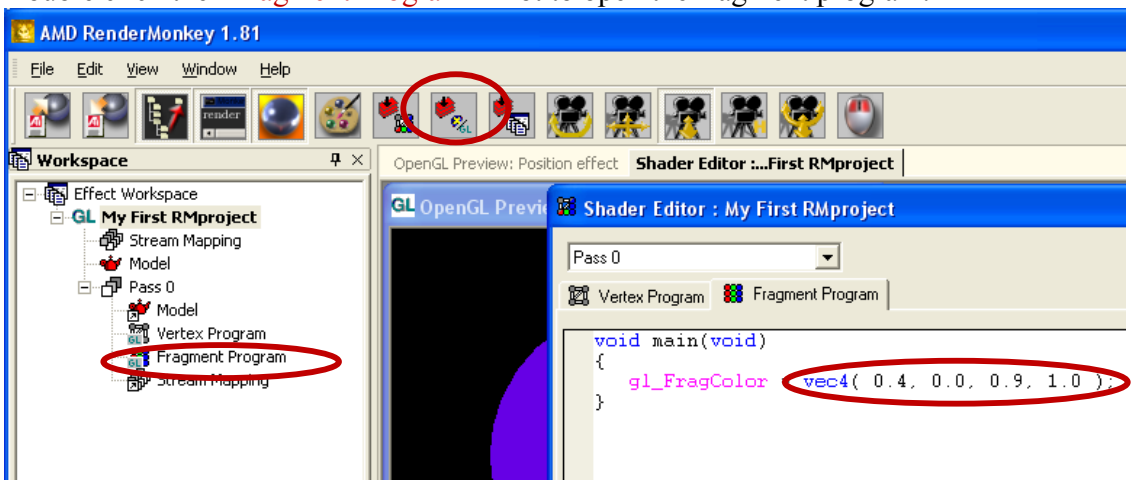
1. Run RenderMonkey
2. Right click on Effect Workspace and select Add Default Effect.



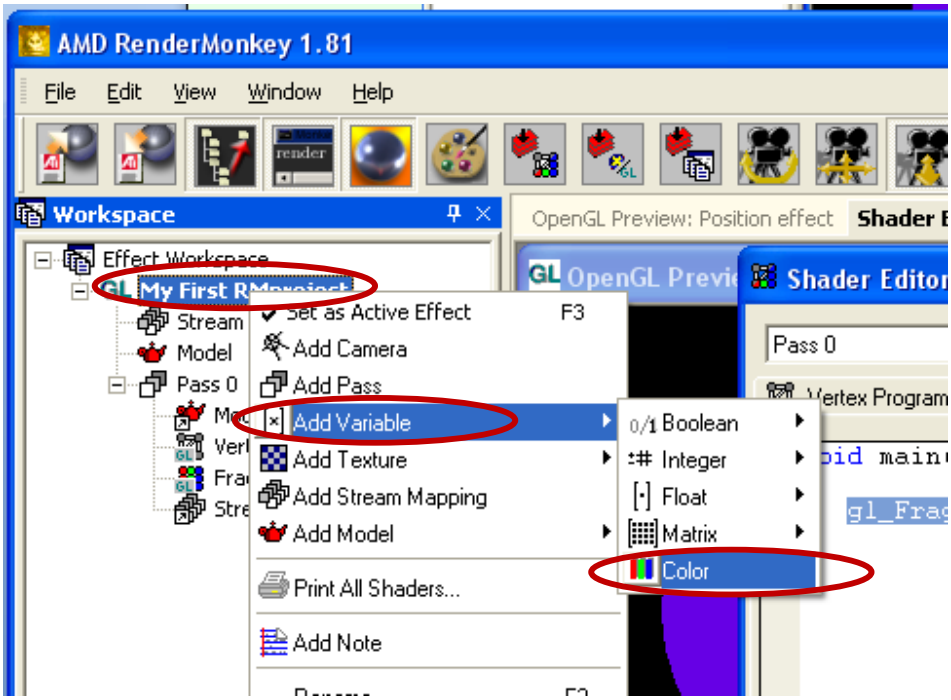
3. Select an OpenGL default effect
4. Select Position effect
5. You should now see the following image



6. Right click on "Position" (or press F2) to rename it as "My First RMproject".
7. Click the '+' to expand the workspace.
8. Double click the "Fragment Program" knot to open the fragment program.



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



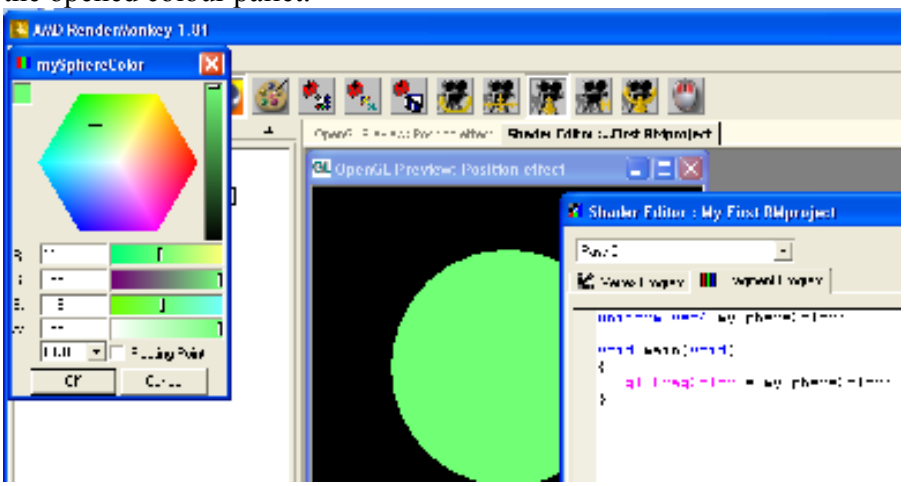
11. You should now have a new variable with default name "myColor". Rename it by pressing F2 to whatever name you like, for instance, mySphereColor.
12. Add the following new line of GLSL code at the top of the fragment shader

```
uniform vec4 mySphereColor;
```

13. Reset the gl_FragColor as *mySphereColor*:

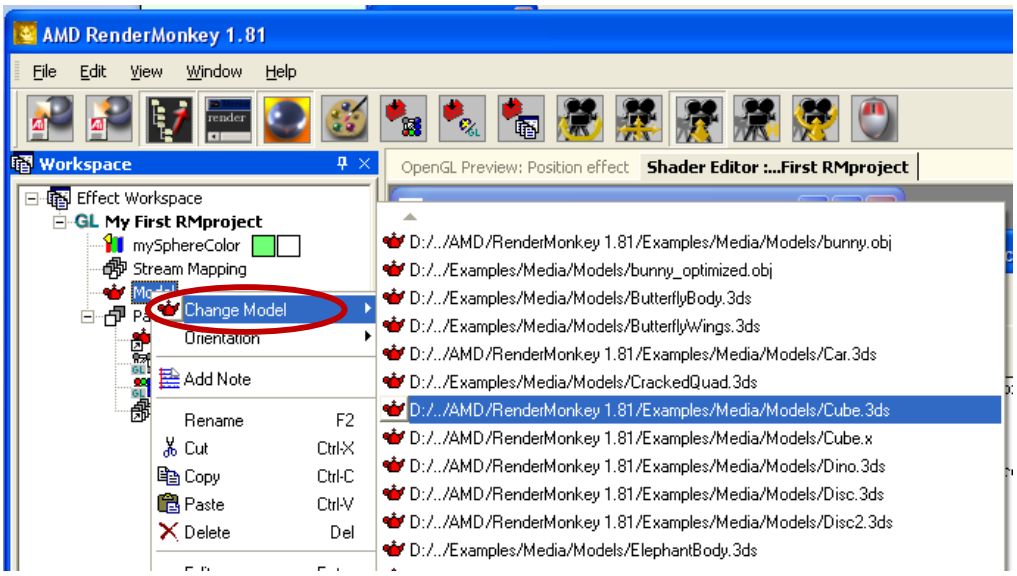
```
uniform vec4 mySphereColor;

void main(void)
{
  gl_FragColor = mySphereColor;
}
```

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



Exercise 2. Change model

You can change the geometric model by right clicking the “Model” knot (see the following figure).



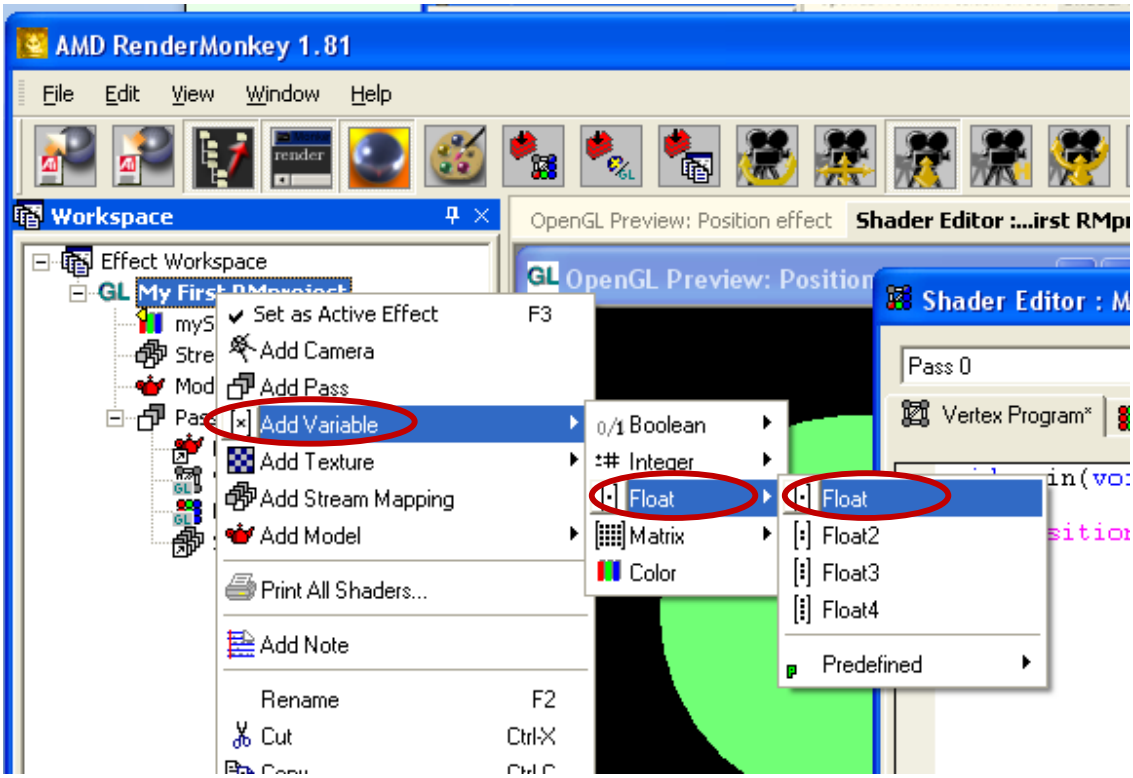
Exercise 3. Scaling the geometric object

1. Open Vertex Program (you should now know how to do this!)
2. Replace the following line of code

   gl_Position = ftransform();

   with

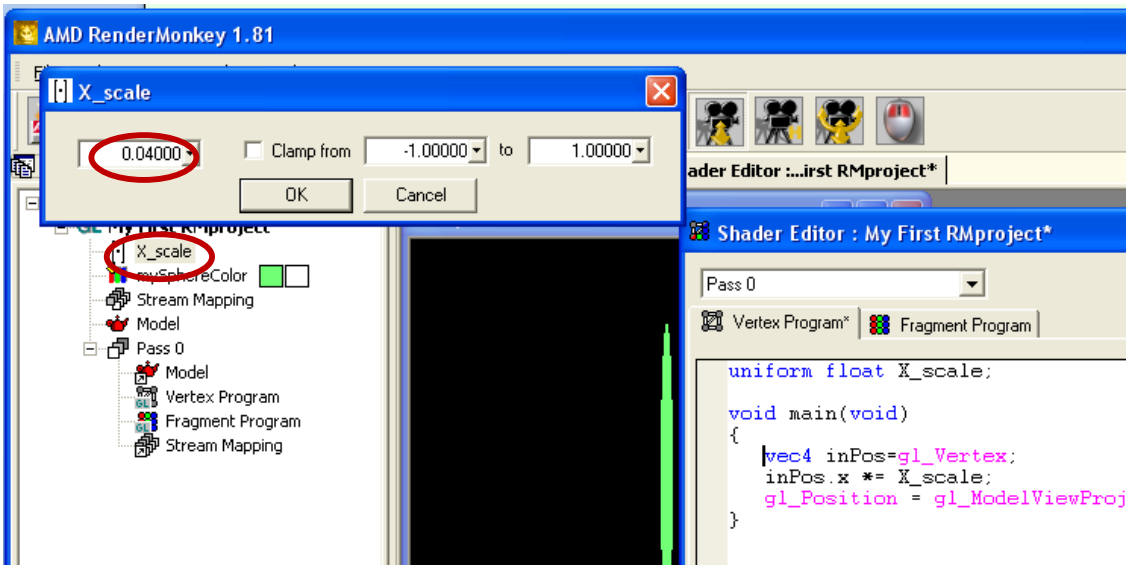
   *gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;*
3. Recompile. You should see exactly the same effect.
4. Add a new variable in the workspace and name it as “X_scale”.



5. Rewrite the vertex shader program as follows

```
uniform float X_scale;

void main(void)
{
   vec4 inPos=gl_Vertex;
   inPos.x *= X_scale;
   gl_Position = gl_ModelViewProjectionMatrix * inPos;
}
```

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



Exercise 4. Implement scaling transformation along y-axis and z-axis in vertex shader.

Written by Dr Qingde Li(q.li@hull.ac.uk).
November 17, 2008.