Tutorial 14: Shaders in OSG

Franclin Foping

franclin@netcourrier.com

May 12, 2008

Abstract

This is the last tutorial of the first part, which focus on writing static scenes in OSG. We all know that latest GPUs allow programmers to add codes that will be directly run by the GPU. This has several advantages as graphics/game programmers have been using a lot. Can you give a game without a shader? There answer is no. Therefore, shaders play a vital part of graphics program. It is crucial for any developer to know how they work and to be able to implement them in OSG. Fortunately enough, this tutorial will be of great help.

1 Shaders in OSG

OSG allows us to enhance our scene with shaders. However, as it was built upon OpenGL, you can easily guess that the shading language to be used is **GLSL**[1]. However, there is a separate (I am yet to test it unfortunately) project which provides a way to add shaders written in Cg. That program is called **osgNV**[2]. As soon as I have tested it, I will write a tutorial about it. GLSL has got many benefits over HLSL. For instance, it is cross-plateform and easy-to-learn. Although HLSL is widely used in the game industry, I am still fond of GLSL. Remember from the tutorial I mention that these

[1] OpenGL Shading Language

[2] Available at `//osgnv.sourceforge.net`

tutorials were designed for Linux users!
There is a big issue with shaders in Linux: the lack of shader IDE that will let us develop shaders in Linux without having to rely to the most powerful tool from ATI: **RenderMonkey**. As a result, I will ask you to write your shaders with RenderMonkey without porting them to OSG. There is no reason to be fret for GLSL is cross-plateform so you will not need to change any part of your shaders.
In this tutorial, I will not teach you how GLSL works but instead how to add shaders to nodes. I am sure your GPU programming skills are better than mine!
We have got two shape drawable objects in this tutorial: a Cube and a Capsule. The cube will be transformed into an RGB cube and the capsule will be rendered with the Gooch non photorealistic rendering (NPR) algorithm. I strongly advise you to read the GLSL Orange Book. It should be your bible!
Here is the process of adding shaders to a particular node.

1. Create a program object (line 38)
2. Create a vertex shader object and attach the definition of the vertex shader (line 39)
3. Create a fragment shader object and attach the definition of the fragment shader (line 42)
4. Bind the two objects created in steps 2 and 3 to the object create in step 1 (lines 46-47)
5. Finally attach the shader program to the state set of the given node (line 50)

Remember that you can also hard code the shader file in the source code of your program, this is useful when you want to hide your shaders to the users. However any change in your shaders will require a full compilation of your program.
On the other hand, people avid of data-driven design will find useful to separate the shader files from the program so that they can tweak the shaders

without needing to recompile the whole program. I am also keen of this prospect. The final choice is up to you!

2 Forcing the wireframe mode on a node

In order to render an RGB-cube, I had to force a wireframe mode on that node. The process is described at the lines 27, 34 and 35.

3 Scene graph of our scene

The scene graph of our program is shown at the following picture.

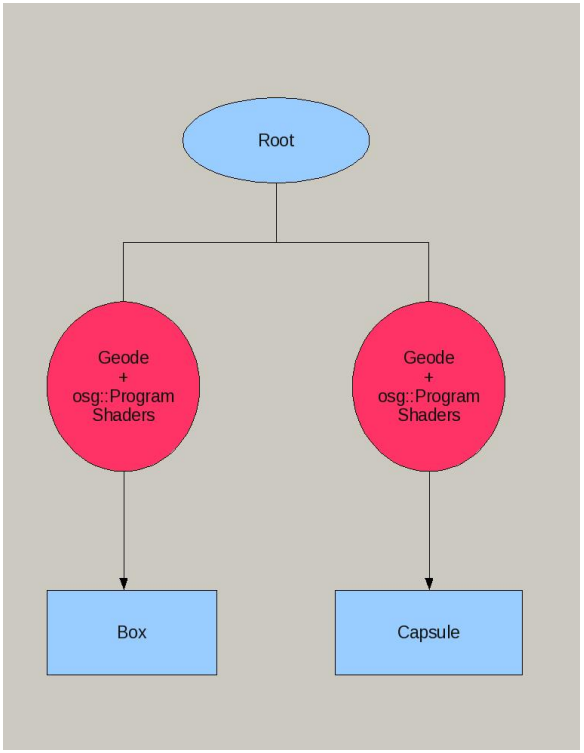

Figure 1: The scene graph

4 Results

The results of our scene is illustrated in the next skecth.

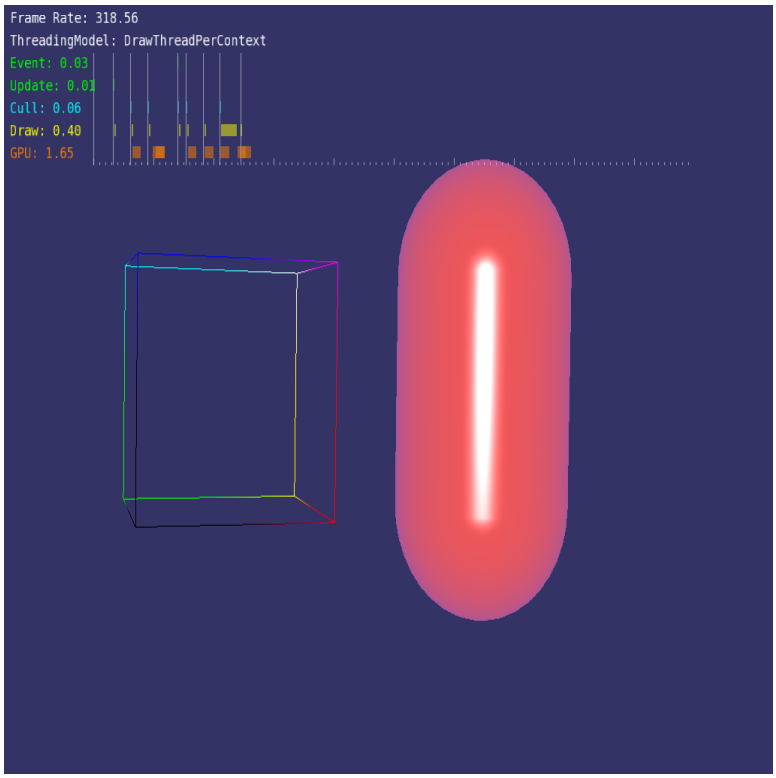

Figure 2: An RGB cube and a NPR rendering

5 Do-it-yourself

Here is your task list:

1. I mention in section 1 that it is possible to define both vertex and fragment shaders in an array of characters in your program. Read the built-in examples **osgshaders** and try to implement them in this tutorial.

2. Remove the wireframe node of the RGB cube.

3. Get yourself a shader on the internet and add to this tutorial. Your object will appear too bright! Could you explain this strange phenomenon? Hint: Think about lighting!