CS4815 Week12 Lab Exercise

Lab Objective: In our final lab of the semester we will take an introductory look at the very important 3D graphics toolkit *OpenSceneGraph*. We will look at how a texture may be applied to an object within this framework.

Here's a **quick summary** of the tasks:

❶ Copy the main source files for this week's lab

❷ Compile the `GeometryTest.cpp` program; run it and examine it

❸ Do likewise for the second program `texturedGeometry`, fixing the compiler error along the way

❹ Modify the texture map as described below

❺ Submit your completed program using the handin command

```
~cs4815/progs/handin -m cs4815 -p w12
```

There's heaps of documentation for OpenSceneGraph; here's the main site.

In Detail

❶ There are two source files for you to consider this week. Copy them, `GeometryTest.cpp` and `texturedGeometry.cpp`, and the associated `Makefile` from this week's sub-directory of the class account. In addition copy the "texture" file `KLN89FaceB.tga`

❷ Compile the two programs using the makefile. Just doing `make all` will do the trick here.

First run the program `GeometryTest`; what affect do the three mouse buttons have? You can quit the program by hitting the `Esc` key.

Now that you have an idea of what the program does take as look at the code for this program in `GeometryTest.cpp`. It will not be possible to explain all of `OpenSceneGraph` here but as you can see it is possible to hide a lot of the details of `OpenGL` and program at a very high level – *once you know what you are doing.*

OpenSceneGraph allows us to bring a bit of organisation to our graphics scenes by allowing us to have graphical 'objects' that can be transformed, rotated, etc. independent of

the remainder of the scene. These are called 'geometry nodes' or `geode`s. They need to be defined in terms of vertices, of course, and they need to be coloured, textured, etc. but being able to do this independently for each object gives great power and flexibility to the overall framework.

One very frequently occurring function used is `push_back()`. This is a member function of a 'container thingy' class that allows us to successively add elements at the end of the container. (We could easily think of this as a linked list here.)

The program will have two 'objects', a pyramid and a cross. Starting at line 38

```
osg::Vec3Array* pyramidVertices = new osg::Vec3Array;
```

we create a set of vertices called `pyramidVertices`. These vertices are associated with the "pyramid geode"[1] in line 56

```
pyramidGeometry->setVertexArray( pyramidVertices);
```

Note very carefully how the base and faces of the pyramid are constructed. The order of vertices of the faces is crucial as this determines what direction the normals point in.

The only two remaining points to remark on are

- how an array of colours is created (line 117), and these are (eventually) associated to vertices of the pyramid and the cross
- to note how easy it is to introduce a *second* pyramid to the scene; we can set the position of this pyramid independent of the first (line 175):

```
osg::Vec3 pyramidTwoPosition(15,0,0);
pyramidTwoXForm->setPosition( pyramidTwoPosition );
```

❸ Compile the second program now. When you try to run it it should report an error. This is because the path for loading the texture map is not set correctly. You should copy over the texture file from the same place as the source files into your current directory and set the path properly (line 122). It is called `KLN89FaceB.tga`.

Either before or after you fix this run-time error you should have a look at the texture map. You might be able to get away with simply giving the file as a URL to `firefox`; otherwise, any of the graphics programs accessible from the start-up menu should be able to display the picture. From the command line I was able to examine the file using:

```
gwenview KLN89FaceB.tga
```

Run the program to see it in action. Rotate the pyramid around so that you inspect all faces of it of it. Note that the colours of the vertices are not dictated by the texture map, just the pattern.

[1] If you follow the code logic of the variable `pyramidGeometry` you will see that I'm leaving out a step in the process here but it's not important that we know all of the details.

The key section of code of this program begins at line 96 where five points of the texture space are defined and then mapped to the five vertices of the pyramid (line 102). You can think of this as putting 5 pins in the `.tga` file at the specified coordinates and then pushing each of these pins into a vertex of the pyramid. (This analogy is not perfect since it explains what happens at the bottom of the pyramid – not the base! – than at the very peak.)

Two questions arise then:

1. What *is* happening at the apex of the pyramid, so?
2. Why does one face have the labels form the texture map back to front?

❹ The task for this week is to now modify this texture map so that the effect is opposite to that seen currently: across three faces of the pyramid the picture of the car radio should be draped across *in reverse*, while on the fourth face the entire picture should appear *in forward*.

You should also change the colours of the vertices so that they are five *distinct* colours; use any colours you can imagine. The modified file `texturedGeometry.cpp` will be the single item to be collected as part of the `handin` process.

❺ Using the handin command given at the top of the lab sheet please submit your lab exercise by the usual deadline next week.