Tutorial 7: Texturing, Multitexturing, loading third-party files and positioning objects

Franclin Foping

franclin@netcourrier.com

May 10, 2008

Abstract

In the previous tutorial, we have learnt how to create a simple geometry in OSG. The objectives of this tutorial are four-fold: adding texture to an object as well as achieving a multitexturing object, loading a 3DS model and add it to your scene and finally positioning an object to the scene. I will assume that you already the rationale of all these tenets.

1 Structure of the tutorial

At the end of this tutorial, the reader should be able to understand how to add textures to objects, load 3Ds models and positioning objects in the world. The tutorial will commence with loading 3Ds models in section 2. Our scene graph will be presented in section 5. Texturing will be explained in section 3.

2 Loading third-party files

We learnt from the very first tutorial that OSG is shipped with several plugins to read third-party files. These include 3Ds models, osg files and so forth.

We have also noticed in the first tutorial that these plug-ins were part of the *osgDB* library. The good news is that many commonplace file formats are included in the *osgDB* library for instance Alias Wavefront OBJ, COLLADA DAE, OpenFlight FLT and many others.
Before describing the process of loading them, I will briefly mention that the osgDB library also provides mechanisms to write your scene or a part of it to an OpenSceneGraph file format (the **osg** file format). Here is how you should proceed to load and attach a third-party file format to your scene. Remember to include the **osgDB::ReadFile** header file.

1. You call the following method from the osgDB library:
   ObjectNode = osgDB::readNodeFile("YourModel.3ds")
2. This call will return a pointer to the node containing your model
3. Check that this pointer is valid. You already know from your C++ experience that using a dangling pointer is something to avoid at all costs.
4. Attach this pointer to your scene (**rootNode→addChild(ObjectNode)**)

Similarly, to read a texture image, you should proceed as follow:

1. You call the following method from the osgDB library:
   ImageObject = osgDB::readImageFile("YourModel.3ds")
2. This call will return a pointer to the node containing your model
3. Check that this pointer is valid. You already know from your C++ experience that using a dangling pointer is something to avoid at all costs.
4. Attach this pointer to the texture object (**texture→ setImage(ImageObject)**)

As we can see, there is also a bunch of other plug-ins for common image file formats: JPG, TGA, DDS, PNG and so forth.

3 Texturing

OSG provides you with a Texture class which wraps OpenGL functions to create and bind textures. In order to add a texture to your geometry, here are the basic steps:

- Having created the texture object, be it 1D, 2D or 3D
- You should now use the following calls:

```
osg::ref_ptr<osg::StateSet> stateSet (node->getOrCreateStateSet());
stateSet->setTextureAttributeAndModes (0,   // Texture unit
                                      texture1.get(), //Texture Object
                                      osg::StateAttribute::ON);
```

These are well described in the accompanied source code. I advise you to look at it.

4 Positioning objects in the scene

There are mainly two ways to specify the position of your objects in OSG:

1. Using the **PositionAttitudeTransform** class. It makes use of a 3D vector. You can apply transformations (translation, scaling, shearing and rotation) to your objects.
2. using a **MatrixTransform** object. Unlike the previous class, it uses a Matrix to apply transformation to its children

This is the algorithm is as follows:

- Create a PositionAttitudeTransform (resp. MatrixTransform) object
- Attach it to the root node of the scene
- Attach your object to the PositionAttitudeTransform (resp. MatrixTransform) object

- Any transformation you specified to this node will be applied to your object.

Again, I have detailed these two processes in the enclosed source code so I guess you will have a go at it as practice makes perfect!

5 Scene graph of the scene

Surprisingly enough, the scene graph is very similar as to the one of the previous tutorial. This is because we have only added a state set object to the geode, that is it!

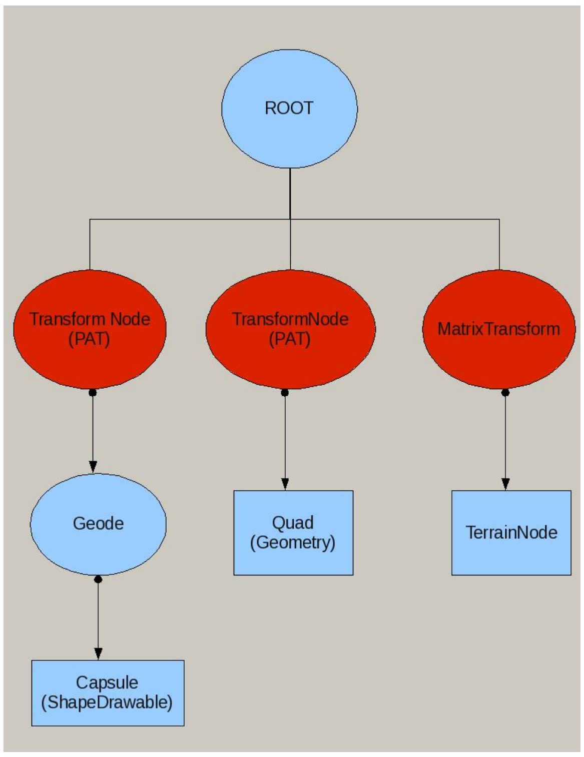

Figure 1: A scene graph

The final screenshot of the scene is as follow:

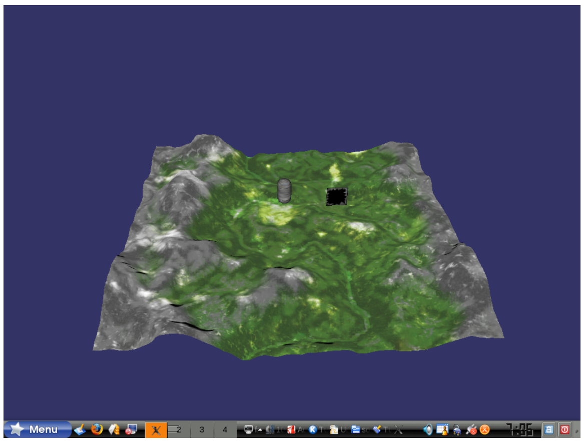

Figure 2: Our final scene

6 Do-it-yourself

Don't worry if at this juncture you can move your object, in the second part of these tutorials you will be impressed. For now here is your task list:

1. Get yourself some 3Ds models on the internet and add them to the scene
2. Try to change the position of the right quad.
3. An astute student has immediately noticed that the quad is **multi-textured**, yes that is right, your task is to understand how this was achieved and try to tweak it. Again don't worry if things are static for now later on we will achieve advanced effects and manipulate the texture matrix.
4. Your final job will be to optimize the scene graph. Indeed OSG provides you with an optimizer to help you optimize your scene graph and improve your frame rate. By uncommenting the lines 165 and 166, you will be able to enjoy it! However, for small scenes like this one, you won't notice an improvement but believe for bigger scenes there is a huge outcome!