Tutorial 1: An overview of OpenSceneGraph

Franclin Foping
franclin@netcourrier.com

May 7, 2008

Abstract

At the end of this tutorial, the reader should be able to understand what is OpenSceneGraph and how many companies are currently using it. Because OpenSceneGraph is built on top of OpenGL, I will also explain its similarities and dissimilarities with OpenGL.

1 Structure of the tutorial

Before talking about OpenSceneGraph (OSG), I will start the tutorial by describing scene graphs in section 2. In section 3, I will describe OSG in depth and a comparison with OpenGL will be presented in section 4. The tutorial will be closed by giving a set of references for further reading.

2 Scene graph definition

2.1 Graphs' highlights

A **graph** is a data structure, an abstract data type (ADT) consisting of a set of nodes and a set of edges describing the connections between nodes. Foping (2006) studied in depth planar graphs and also described an optimal algorithm to traverse them.
A key design element in many graphics programs is a **scene graph**. Indeed,

they help manage objects as well as their associated transformations. A well constructed and designed scene graph accounts for an easier management of the entire program.
Scene graphs consist of a graph of nodes depicting the spatial sketch of a 3D scene while hiding graphic characteristics in objects. Hence the strengths of scene graphs; spatial organization for culling and embedding the entire scene in a scene graph. In addition to that, there is a need to have a powerful, easy-to-use and scalable application.

2.2 Scene graphs

A **scene graph** is a data structure aiming at organising the logical and often spatial representation of a graphical scene. It is typically drawn with the root at the top, and leaves at the bottom. It starts with a top-most root node which encompasses the whole virtual world. The world is then broken down into a hierarchy of nodes representing either spatial groups of objects, settings of the position of objects, animations of objects, or definitions of logical relationships between objects such as those to manage the various states of a traffic light. The leaves of the graph represent the physical objects themselves, their drawable geometry and their material properties. Formally speaking, a scene graph can be defined as a tree or as a directed acyclic graph with a random number of children. The following diagram shows an example of a scene graph representing the solar system.

2.3 Benefits of scene graphs

- **Performance**: scene graphs provide an excellent framework for optimizing graphics performance. A good graphics engine should encompass the culling of objects that will not be sent to the graphics pipeline for rendering on screen, and state sorting of properties such as textures and materials, so that all objects that are alike are rendered together. By not supplying this technique, both the central processing unit (CPU) and the graphics processing unit (GPU) will be swamped

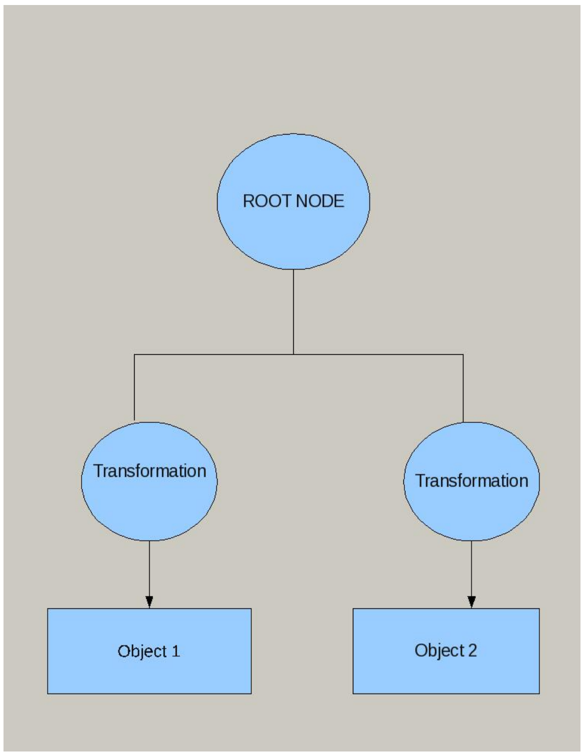

Figure 1: A scene graph

by a tremendous amount of data. Scene graphs make this process less painful by supplying a mechanism to sort the state of objects.

- **Productivity**: scene graphs take away much of the hard work required to develop high performance graphics programs. The engine manages all the low-level graphics, reducing what would be thousands of lines of OpenGL down to a small number of simple calls. Furthermore, one of the most powerful concepts in object-oriented programming is object aggregation, enshrined in the composite design pattern, which perfectly fits in the scene graph tree structure, hence making a highly flexible and reusable design. Scene graphs also often come with additional utility libraries which range from helping users set up and manage graphics windows to importing 3D models and images. A dozen lines of code can be enough to load mesh data and create a real-time simulation.

- **Portability**: scene graphs encapsulate much of the lower level tasks of rendering graphics and reading and writing data, reducing or even removing the platform specific codes used in an application. If the underlying scene graph is portable then moving from platform to platform can be as simple as recompiling the source code.

- **Scalability**: along with being able to dynamically manage the complexity of scenes to account for differences in graphics performance accross a range of machines, scene graphs also make it much easier to manage complex hardware configurations, such as clusters of graphics machines, or multiprocessor/multipipe systems. A good scene graph will allow the developer to focus on the content of the application not the lower level.
- Hierarchical modelling and inheritance: one of the most important concepts in object-oriented programming is the ability to define a behaviour or a state in a parent class so that its children can easily inherit from those properties and attributes. Scene graphs provide this feature.

3 OSG description

OSG is an open source high performance 3D graphics toolkit made by **Robert Osfield** and **Don Burns**, used to develop applications in fields such as visual simulation, games, virtual reality, scientific visualization and modelling. Built on top of OpenGL and entirely written in C++, it runs on all Windows® platforms, OSX, GNU/Linux, IRIX, Solaris^{TM}, HP-Ux, AIX and FreeBSD operating systems. It also provides a viewer to display the 3D virtual scene. OSG is currently used by many companies including Boeing for its flight simulation, NASA for its Earth visualization, Indra and Tesco. The whole graphic pipeline to render the scene is directly embedded in the core library. Every frame, the scene graph is traversed and objects in the view volume are then drawn. Figure 2 shows its functional components.

- The core OSG libraries provide essential scene graph and rendering

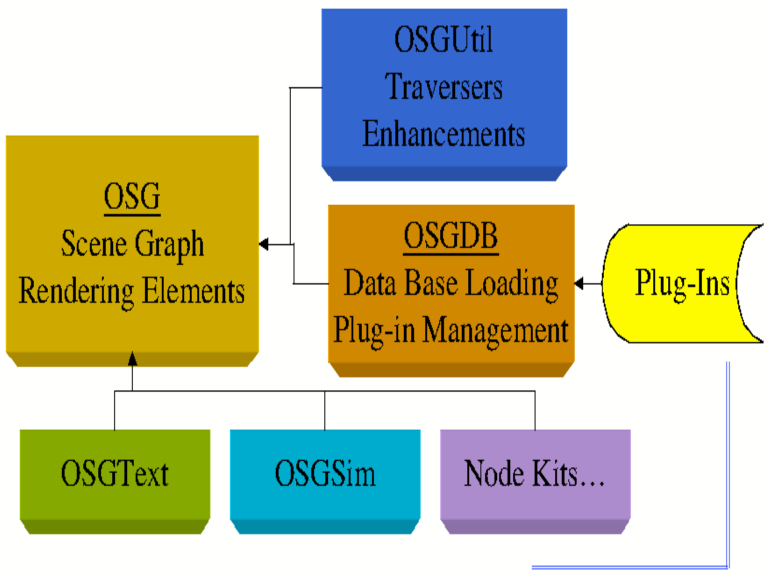

Figure 2: Core OSG (Martz (2007))

features, as well as additional features required by 3D graphics applications.

- NodeKits extend functionalities of core OSG scene graph node classes to provide higher-level types and special effects such as particle systems or precipitation effects (rain, snow and fire).
- There are forty-five plug-ins in the core OpenSceneGraph distribution. They offer support for reading and writing both native and third-party file formats (3ds, jpg, bmp, dds, etc.).

3.1 The core OSG library

This part of OpenSceneGraph provides core features, classes and methods for operating on the scene graph. It is made up of four libraries:

- **The osg library** contains all classes related to the scene graph node. It also contains class for mathematical operations and rendering state specification and management. I will describe some key classes of this library in another tutorial, just bear with me for now.

- **The osgUtil library** contains classes and methods to operate on a scene graph and its contents. This library also provides methods to gather statistics of a program (**StatHandler**), optimizing the scene graph (**optimizer**). This library also provides interfaces for geometric operations. Because of its importance to OSG applications, I will present the *optimizer* in a separate tutorial.

- **The osgDB library** contains classes and methods to create and render 3D databases. It provides plug-ins to read and write file formats supported by OSG.

3.2 The processing pipeline

Rendering a scene graph is achieved by traversing the graph and sending the resulting state and geometry data to the graphics card as OpenGL commands. All these happen on a frame-by-frame basis. Figure 3 shows the pipeline processing.

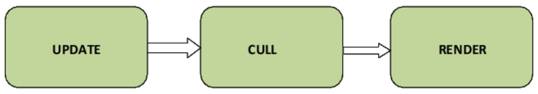

Figure 3: OSG pipeline processing

3.2.1 The event traversal

This is where every GUI event are handled such as keyboard and mouse interaction.

3.2.2 The update traversal

This traversal modifies the state and geometry. These updates are performed either by the application or through callbacks functions attached to nodes that they operate on. By updating a node, the developer changes its attributes such position and colour. Callbacks can be used to achieve animations. This mechanism will be described in another tutorial.

3.2.3 The cull traversal

After the previous traversal is completed, a cull traversal is performed in order to find what objects will actually be seen in the screen and pass a reference to the visible objects into the final rendering list. This traversal is also in charge of ordering nodes for blending. The output of this process is the render graph.

3.2.4 The draw traversal

This traversal uses the render graph generated by the previous traversal and sends this to the underlying hardware for rendering. OSG has a multi-processing architecture. In fact, the processing pipeline is achieved in parallel.

4 Comparing OSG and OpenGL

Now that you have got a rough idea of how OSG works, it is time to know more about equivalence between OpenGL and OSG. Although OSG was built on top of OpenGL, their theories are very different. In this section, I will explain some of them. Obviously I will not cover all of them but at least you will be aware of some dissimilarities. Hopefully, this will save you an aweful amount of time!

4.1 Two different coordinate systems

The first thing, every OSG developer (especially those with an OpenGL background) has to be aware of is the different coordinate systems used in both API. Look at the figure 4, as we can see, OSG by default uses a right-handed coordinate system with the Z-axis as the up vector, remember at the beginning of this tutorial I told you that OSG is used for simulation and in physics, the up-vector is usually considered to be the Z-axis. We will see in another tutorial what this means to transformations.

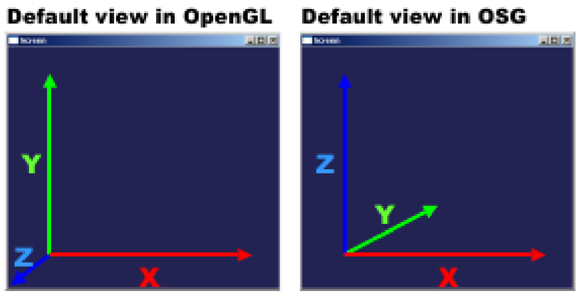

Figure 4: Different coordinate coordinates

4.2 Shaders

Another thing you need to know is the way both APIs handle shaders. In OpenGL, developers must use OpenGL Shader Language (GLSL). Even in OSG it is the same thing, after all OSG can be seen as an OpenGL wrapper. However, in OSG when you write your shaders, you have to always think of a

right-handed coordinate system. This point is important because when you use tools like ATI RenderMonkey or nVIDIA FXComposer to model your shaders, they always use a left-handed coordinate system such as OpenGL, of course you can change their coordinate systems but bear in mind that **you have to use the same coordinate system as OSG**.
Before ending this paragraph, I will mention that it is possible to write your shaders in Cg. There is a project called **OsgNV**[1] specifically designed to help you achieve this task.

4.3 OO Programming

Can you guess why OSG was written in C++ with the standard template library (STL)? Well, OSG uses all advantage of OO programming, inheritance, polymorphism, virtual functions, smart pointers and design patterns. This means that everything is wrapped in nice classes so the users do not have to know the low-level code underneath. On the other hand, OpenGL is not Object-Oriented so beware... A good example of this concept can be illustrated with the fog effect (this will be covered in another tutorial). In OSG, the fog effect is wrapped in a fog class whereas in OpenGL, the developer has to define everything!

4.4 Other aspects

I mentioned previously how OSG wraps OpenGL methods. In fact, attributes related to the geometry are enclosed in a powerful class called **StateSet**. You must understand its rationale to be able write good OSG codes that is why I will cover them in another tutorial.
In OpenGL, developers define the color of vertices using *glColor(...)*, in OSG you have to use a *Material* class. This is also part of the core osg classes.
The final thing I need to mention in the way both APIs handle transformations. In OpenGL, you call *glTranslate, glRotate, glScale...*, sorry to bother you but in OSG you have to change the way to think about transforma-

[1]Available at `http://osgnv.sourceforge.net`

tions. The first you should know is that there are 2 classes to do this job namely, *PositionAttitudeTransform* and *MatrixTransform*. The way these classes work will be explained in another tutorial. You should also know that transformations are applied to all children of the scene graph. This is called **hierarchical modelling**. Think of a car with its components: wheels, engine etc. How are you going to model that in OSG?

5 Conclusion

Well, we have come to the end of this long tutorial. Right now, you should know what is OpenSceneGraph and also how different is it from OpenGL. It is very important to know that beforehand, you should also come back to this tutorial when you are stuck, you can also look at the official website (official website (2007)) if you want to know more about OSG. There is no exercise in this tutorial. However there is something I am yet to mention: The actual installation of OSG on your plateform. That will be the theme of the next tutorial...

References

Foping, S. F. (2006), A polynomial algorithm to find the reversal degree of planar graphs, Master's thesis. University of Yaounde I, Cameroon.

Martz, P. (2007), *OpenSceneGraph Quick Start Guide*, Skew Matrix Software.

official website, O. (2007), 'OSG Official Website', [Available online: `http://www.openscenegraph.org`].