OpenSceneGraph Tutorial

Michael Kriegel, Mei Yii Lim, Matthias Keysermann
Heriot-Watt University, Edinburgh

August 2013

About Open Scene Graph: Open Scene Graph is a modern open source scene graph. Open Scene Graph (or short OSG) and the confusingly similar sounding OpenSG have become the two leading SceneGraph systems. The main single information source on Open Scene Graph is the project's website: http://www.openscenegraph.org/projects/osg. On there you can find various downloads for OSG and plugins along with tutorials, examples and discussion groups.

About This Tutorial: This tutorial is based around tasks and you should work through it chronologically. We encourage you to also use the OSG website as a further information source

Task 1: Setup OSG

There are several ways of installing OSG, depending on the operating system you would like to use and whether you want to work with a binary release or compile OSG yourself. For the purpose of this course we prepared a Windows 7 OSG distribution, containing everything you need to get immediately started. This distribution is installed in the labs G46 and G47 and located the C drive in the folder C:\program files (x86)\openscenegraph. The following environment variables should all be set already as system variables and are necessary to run OSG correctly, but please check and add them as user variables in case they are missing from the system variable list.

OSG environment variables:

- OSG_BIN_PATH = C:\program files (x86)\openscenegraph\bin
- OSG_FILE_PATH = C:\ program files (x86)\openscenegraph\Samples
- OSG_INCLUDE_PATH = C:\ program files (x86)\openscenegraph\include
- OSG_LIB_PATH = C:\ program files (x86)\openscenegraph\lib
- OSG_ROOT = C:\ program files (x86)\openscenegraph
- OSG_SAMPLES_PATH = C:\ program files (x86)\openscenegraph \share\OpenSceneGraph\bin
- The environment variable Path should contain both
  - C:\ program files (x86)\openscenegraph\bin
  - C:\ program files (x86)\openscenegraph\share\OpenSceneGraph\bin

You can set environment variables via Windows Start Menu / Search Box / Type "environment" / Click on "Edit Environment Variables for your Account". As you cannot edit System variables you have to create User variables. For the changes to take effect you will have to log off from windows and log in again. You can also print the value of a variable in the command prompt with "echo %VARNAME%" for example "echo %Path%"

Task 2: Run the example applications

In order run the examples, go to the OSG folder (C:\program files (x86)\openscenegraph) and start the file runexamples.bat. This batch file will run through a couple of OSG demo applications that demonstrate the various features of OSG. To end one demo application and jump to the next one, press 'Esc'. Apart from that, a demo will print additional instructions on the screen if interaction with it is possible. In all the demos the mouse can be used to navigate the camera. The following controls are used:

- hold left mouse button + move mouse → rotate scene
- hold middle mouse button + move mouse → move scene
- hold right mouse button + move mouse → zoom in scene

The same controls are used in every standard Open Scene Graph application, so familiarize yourself with them. Note that you can apply a constant automated rotation, translation or zoom by using the above controls but releasing the mouse button before you stop the movement. Try to make a model spin by itself. Note that the source code for all these examples is available, so they are a good reference if you want to find out how to implement a certain graphics effect later.

Task 3: Compile your first OSG application in Visual Studio 2010

Visual Studio 2010 by Microsoft is the programming environment that you are going to use to write, debug and compile open scene graph applications. Visual Studio supports several programming languages including C#, C++ and Visual Basic, but for our Open Scene Graph purposes we will only use C++.

Solutions and Projects: If you are not familiar with Visual Studio, the distinction of solutions and projects might be a bit confusing at first. A project consists normally of one or several source files and has one output (usually an executable, but may also be a dll etc.). A normal small application of the size you produce during these tutorials normally be a project. A solution is a collection of projects that belong together. For example if you have split a bigger application in several separate modules, each of these modules should be a project and the whole application a solution. In a solution you can set up dependencies, in other words determine the build order. It is unlikely that you will need any of these features for this tutorial. However, even if you only work with a single project you still need a solution for this single project.

Open the first example: For now you will not have to make any modifications, so just open the first really simple example solution/project. It is located in the OSG subfolder *viewer 1*. Copy the whole folder to somewhere on your H: drive or the local D: drive (You won't be able to compile it on the C: drive because you don't have write permission there.). Load the solution file simpleviewer.sln and the project will automatically be loaded as well. Confirm security warnings and info boxes about temporary storage locations that might appear and press "OK".

Compile and run that example (green “play button” in the tool bar). The application should start now and you should see a cow on screen.

Now we will look at the source code for this example (use the solution Explorer on the left hand side to open the single source file 01_SimpleViewer.cpp) . It starts with an #include statement block:

```
#include <iostream>
#include <osg/Group>
#include <osg/Node>
#include <osgDB/ReadFile>
#include <osgViewer/Viewer>
#include <osgGA/TrackballManipulator>
```

If you know C++ this should be familiar. In this block you include the header files of the libraries you are going to use in your program. Open Scene graph header files are all organized by namespaces that refer to the different modules Open Scene Graph consists of. The namespace and the name of the particular header file are separated with a slash. All the osg includes directly refer to a file in the include sub-folder. For example the 5th line allows the program to use all functions, types and classes that are defined in the header file C:\Program Files\openscenegraph\include\osgViewer\Viewer. After, you’ve gone through the whole source code, have a look at those include files and try to figure out, why they have been included.

In the first line of the real program, a viewer is created:

```
int main()
{

//create a viewer
osgViewer::Viewer viewer;
viewer.setCameraManipulator(new osgGA::TrackballManipulator());
```

The viewer is not part of the scene graph itself. It is the module that is responsible for displaying the scene graph. In the next line the viewer gets assigned a camera manipulator. A camera manipulator defines the way the camera is controlled by the mouse. Here we use a trackball manipulator.

```
// Load a model
osg::Node* modelNode = osgDB::readNodeFile("cow.osg");
if (!modelNode)
{
      std::cout << " could not find model " << std::endl;
      return 0;
}
```

In this section a model is loaded. Lets go through this step by step: “cow.osg” is the file name of the model that will be loaded. This is a model file in the native open scene graph model format that comes with the osg installation. osgDB::readNodeFile is the function that handles loading of all 3d models independent of their format. The program will know which loader plug in to use by looking at the file extension of the 3d model file. The function returns a pointer to an osg::Node, which is the top Node of the model’s scene graph.
The last “if-section” checks if there were any errors when loading the model.

Now we create a group object which represents the root of our scene graph. Then we add our loaded model as a child to that group.

```
osg::Group* root = new osg::Group();
root->addChild(modelNode);
```

In order for the viewer to display something we need to associate it to a scene graph.

```
viewer.setSceneData( root );
```

Then we set up the window and associated threads.

```
viewer.realize();
```

The final bit of the program is the rendering loop. For every run through this "while" loop a new frame is drawn on the screen (viewer.frame()):

```
while( !viewer.done() )
{
      viewer.frame();
}
}
```

Smart pointers

OpenSceneGraph provides with the osg::ref_ptr<> template a smart pointer class to objects of all Open Scene Graph classes. If you create a standard pointer (e.g. osg::Node* modelNode), you have to remove the loaded object from memory yourself, if you don't need it anymore or waste memory. Smart pointers automatically count references and delete the associated object if the reference count drops to zero. In Java this concept is called Garbage Collection.
Instead of using standard pointers, you can use a smart pointer in the following way:
`osg::ref_ptr<osg::Node> modelNode = osgDB::readNodeFile("cow.osg");`
For accessing objects stored using a smart pointer, the get() method is used:
`modelNode.get()`
The variable modelNode does not point itself to the memory address of the top node in the model, it rather points to the memory address of the smart pointer construct. To get the memory address of the Node you have to use the get() function.
Try to change all pointers in the example to smart pointers and consider making the use of smart pointers in all your subsequent OSG coding a habit.

Task 4: load your own model

Note: For X3D to VRML conversion the following online converter is suggested:
http://doc.instantreality.org/tools/x3d_encoding_converter/

You have already created a 3d model in the VRML section of this course. Your task is now to load that model into Open Scene Graph by modifying the above example. In fact all you have to change in the source code is the model file name. However, our Open Scene Graph installation does not include a VRML loader so if you try to load a vrml file (e.g. car.wrl), you will get an error message. The easiest way for you to

import your model is via 3d Studio Max. Open 3D Studio Max (should be in the start menu under multimedia / Autodesk). In 3D Studio Use File/Import to import your wrl file. Now you should see your VRML model in 3D Studio Max. to make sure that everything including textures was imported correctly, click on the Perspective view, position the camera so that you get a good view of the model, press F10 to open the Rendering window and click on Render. If your model uses textures you might now get a message that 3ds can't locate those textures. If that happens, just add the path where the textures are located to 3d Studio's path. If everything is fine with your model, you can export it to a 3ds file by Choosing file/export. Now you only need to place this newly created 3ds file and the texture files that were used in a location, where it can be found by OSG. You have several options:

- the easiest option is to just put the files in the working directory. Note: if you start your application directly from visual studio the working directory is set to the solution directory (the folder that contains the solution file). You can change this in the properties of the visual studio project under Configuration Properties/Debugging/Working Directory
- You could also put them in the data folder of osg (C:\Program Files (x86)\openscenegraph \Samples – remember the path variable OSG_FILE_PATH). If you check the initial example, the cow.osg file was loaded from that directory. Unfortunately, you don't have write permission on C:\ so this won't work, but this would be the easiest option if you had OSG installed on your own PC.
- The third option is to add another path to the environment variable OSG_FILE_PATH, where you can place the files. This path obviously has to point to somewhere, where you have permission to put files so ideally on your H:\ drive

Task 5: Do the NPS Tutorials

Now you will have the chance to create some programs yourself. Do that by following the first 3 tutorials (Basic geometry, Textures, Transforms and States) plus any additional ones you want on:
http://www.openscenegraph.org/projects/osg/wiki/Support/Tutorials

If the Open Scene Graph website should be temporarily down (as it happens to be right now while we are writing this) you can still access the tutorials, e.g. through the cache provided by Google search or the Internet archive's wayback machine.

http://web.archive.org/web/20110722000830/http://www.openscenegraph.org/projects/osg/wiki/Support/Tutorials

This second link also has the advantage that we verified the tutorial version at that point in time worked with the osg version installed. By the time you are reading this the first link might already contain once again updated tutorials that need modifications to work with the installed version. Note that the tutorials can also be found and originate from here:

https://www.movesinstitute.org/Sullivan/OSGTutorials/

However try to avoid this last link as it points to outdated versions of the tutorials for older releases of OSG (on the official OSG website they have been updated in line with changes to the library). Changes are however relatively minor and should be easy to figure out.

A few hints that might be useful, while doing the tutorials:

Solution & Project – Creating a new solution and project can be quite troublesome, because there are a number of project settings (compiler and linker options, etc.) that have to be changed in order for an Open Scene Graph project to compile correctly. The much easier and suggested solution is to copy and then modify the example solution. If you want to create a new solution and project from scratch, refer to the instructions on the Open Scene Graph website.

Class reference - When you are programming with a complex API like Open Scene Graph you might need an API reference. The OSG API reference can be found locally in C:\Program Files\openscenegraph\doc\OpenSceneGraphReferenceDocs or online at http://www.openscenegraph.org/projects/osg/wiki/Support/ReferenceGuides

Include files - One thing the tutorials don't tell you, is what to put in your include section. Normally each class has its own header file. So for example, if you use an osg::Transform, the matching include command will be `#include <osg/Transform>`. You don't have to include every single class you use, because if you include one file, you automatically also include all the files that are included by this file, etc. The best way to work is to not worry about the include section until the time of compilation. At this point the compiler error message will tell you, which class is missing, which you then simply add and compile again.

Library files – For every header file that you include you also need to make sure that the corresponding static library file is linked. Windows static library files have the extension lib and the OSG static libraries can be found in the OSG_LIB_PATH (C:\program files (x86)\openscengraph\lib). Note that there is a release and debug version for each library. Debug versions have the same name as Release versions plus the letter d. For example osg.lib is the release library containing the code for the main OSG classes while osgd.lib is the debug version of the same library. Usually you'll know if you need to include additional library files from linker errors during the build process. You add additional libraries in the Visual Studio project properties under Configuration Properties/Linker/Input/Additional Dependencies. Note that whatever you set up there is only valid for the configuration selected in the Configuration combo box. Therefore make sure you add the version of the library (debug / release) that matches the selected configuration.