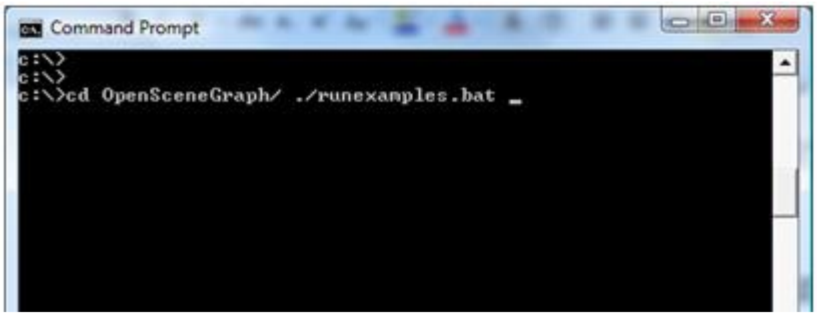
# Introduction to Open Scene Graph

The OpenSceneGraph is an open source high performance 3D graphics toolkit, used by application developers in fields such as visual simulation, games, virtual reality, scientific visualization and modelling.

We will be using Visual Studio C++ as the programming environment for the OSG applications.

- Take a look at some examples of what can be done using OSG. Open a console window and type in

**`cd <<open scene graph directory path>>/runexamples.bat`** . Press Esc to move to the next example.



```
c:\>
c:\>
c:\>cd OpenSceneGraph/ ./runexamples.bat _
```

- Open Visual Studio and look at some sample OSG code from the folder **`C<< open scene graph directory path>>/examples`**

## Your first application - loading and viewing a scene

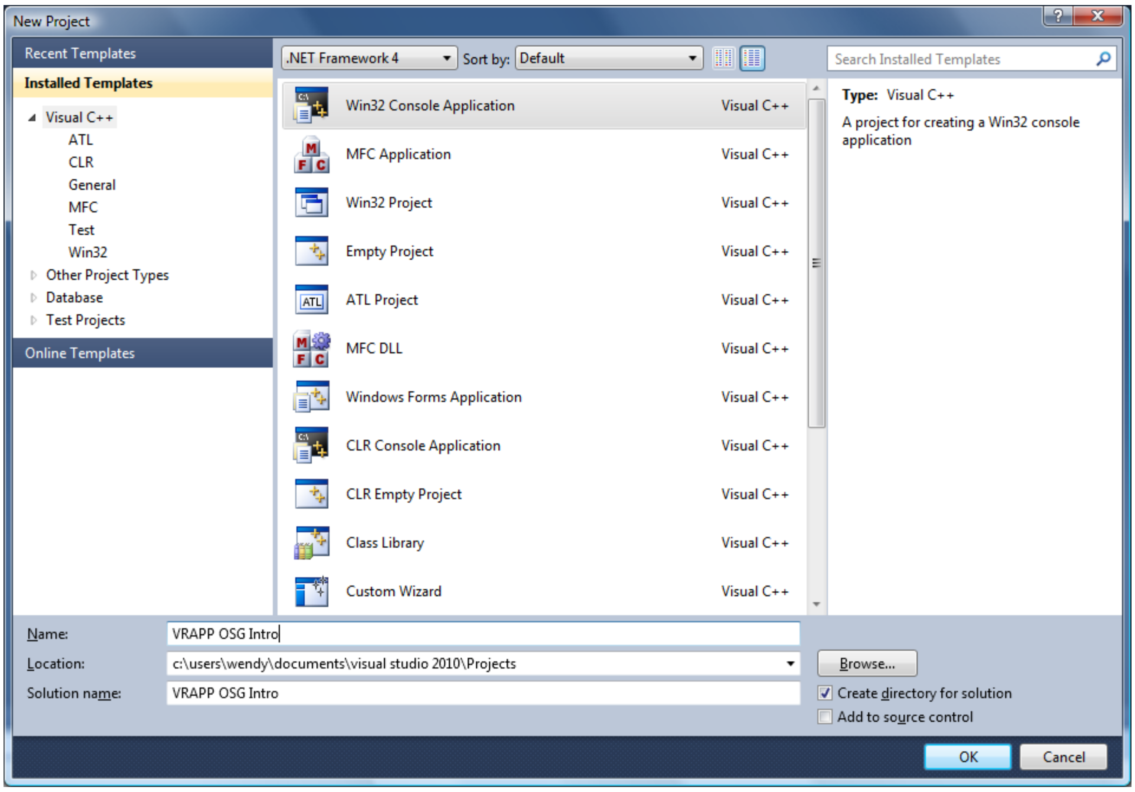
Now you are going to try writing a simple application for yourself. You will load a scene model and display it in an OSG viewer. Then you will add additional objects and mouse control.

Note that the sample code in green (preceded by // or /*) are comments to assist you, and not part of the code.

The greyed out code shows lines you have already added.

Some of the lines of code will run onto a second line of the page in the tutorial. Make sure you keep them on the correct lines in your .cpp file.

- Open a new visual studio Win32 Console Application. Call it *OSG Intro*



Type: Visual C++

A project for creating a Win32 console application

Name: VRAPP OSG Intro

Location: c:\users\wendy\documents\visual studio 2010\Projects

Solution name: VRAPP OSG Intro

[✓] Create directory for solution

[ ] Add to source control

- You will need to set the project file for open scene graph. Remember to do this each time you make a new osg project.
  - under *C++> General* you will need to add the include directory for osg
  - under *Linker>General* you will need to add the library directory for osg
  - under *Linker > Input* you will need to add the osgd.lib, osgviewerd.lib, osgdbd.lib, osgUtild.lib, osgGAd.lib (if you are making a release rather than debug version, take the 'd' off the end of each lib name)

- You will need to add the following to the header declarations:

```
#include "stdafx.h"

#include <osgViewer/Viewer>
#include <osgDB/ReadFile>
#include <osg/Node>
#include <osg/ref_ptr>
```

- Build and run the file to make sure that you have set it up correctly. It should open briefly then return to visual studio.

- Make a new folder called "models" inside your project folder, and copy the model and image files from the resources folder on Moodle into this new folder. It is good to get into the habit of making subfolders to contain your assets for your projects. For this example we are using a .3DS file, but OSG supports a range of different model files.

## Load a model

- Now add the following code into your project file. This will read the model from the file, but won't do anything with it yet.

```
#include "stdafx.h"

#include <osgViewer/Viewer>
#include <osgDB/ReadFile>
#include <osg/Node>
#include <osg/ref_ptr>



int _tmain(int argc, _TCHAR* argv[])
{
       //Declare a node which will serve as the root node
       // the rest of the scene "hangs" on this node

       osg::ref_ptr<osg::Group>  myRoot = new osg::Group();

       //********** load scene model from file ***********************//

       osg::ref_ptr<osg::Node>  groundNode = osgDB::readNodeFile("models/ground.3ds");
// creates a new node for the main object in the scene.  This will be attached to the
root when it is loaded.

       myRoot->addChild(groundNode.get()); // attach the ground to the root node

       //*****************************************************************//


       return 0;
}
```

## Add a viewer

- Build and test the file before continuing. Next you will create a viewer to display your scene, and set the camera to a suitable start point (the default is the centre of your scene object, which is not always what you want!). Add the following code to your file.

```
myRoot->addChild(groundNode); // attach the ground node to the root file

       //*****************************************************************//


//*******set up the viewer to display your scene***************//
```

```
	osgViewer::Viewer viewer; // Declare a 'viewer' which will display the scene

	viewer.setSceneData( myRoot ); //assign the scene graph we created above to 
this viewer
	viewer.setLightingMode(osg::View::LightingMode::SKY_LIGHT); // give the scene 
some lighting

	viewer.getCamera()->setClearColor(osg::Vec4(0,0,1.0,1.0)); // sets background 
colour of scene

	//viewer.setCameraManipulator(new osgGA::TrackballManipulator);

	viewer.getCamera()-
>setViewMatrixAsLookAt(osg::Vec3(0.0,0.0,1.6),osg::Vec3(0.0,5.0,1.0), osg::Vec3(0.0, 
0.0, 1.0));
	/* see sample file for explanation of these settings */

	//****************************************************************//

		return 0;
	}
```

- When you build and test this stage, you still won't see anything, as the viewer is not yet set up to run continuously.  Now we need to start the viewer running, and keep it running until we quit the program.  Add the following code:

```
viewer.getCamera()-
>setViewMatrixAsLookAt(osg::Vec3(0.0,0.0,1.6),osg::Vec3(0.0,5.0,1.0), osg::Vec3(0.0, 
0.0, 1.0));
	/* see sample file for explanation of these settings */

	//****************************************************************//
	viewer.realize(); // start the viewer

	while (!viewer.done()) // until the end of the program
	{
		viewer.frame(); // update to next frame
	}
		return 0;
	}
```

- Now when you run your file, the scene will display in the viewer. You can exit by pressing the **Esc** key.

- By default the viewer is to run in full-screen mode. If you are using two monitors, you will need to set the window to the size that you want

```
	viewer.setUpViewInWindow(20,20, 1000, 1000); // the first two parameters are the x and y position, the last two are the height and width


   viewer.realize();
```

## Adding mouse control and new objects

### Mouse

At the moment we can load and view a model, but that isn't much use on its own. Next we are going to look at adding some mouse control to our camera, and putting some new objects into our scene. You can continue working with your previous file, or create a new cpp file from the previous one. Remember to rebuild and test your file after each section.

- Add the following line of code to your file.

```
	viewer.getCamera()->setClearColor(osg::Vec4(0,0,1.0,1.0)); // sets background colour of scene

	viewer.setCameraManipulator(new osgGA::TrackballManipulator);

	viewer.getCamera()->setViewMatrixAsLookAt(osg::Vec3(0.0,0.0,1.6),osg::Vec3(0.0,5.0,1.0), osg::Vec3(0.0, 0.0, 1.0));
```

- You will also need to add a new header to the file

```
#include <osgViewer/Viewer>
#include <osgDB/ReadFile>
#include <osg/Node>
#include <osg/ref_ptr


#include <osgGA/TrackballManipulator>
```

You will notice when you test the code that you now have mouse control of the scene, but the camera manipulator has overridden the manual settings for your camera. Later on you can investigate more precise camera control.

## Make a new object (shape)

Most VR applications will consist of a base scene, and then additional objects placed in the scene which can provide opportunities for interaction. There are two ways of adding objects in OSG - we can either create them directly, or read them in from a file.

- We are going to need some new headers in the file to handle the shapes and the calculations.

```
#include <osgGA/TrackballManipulator>

#include <osg/Geode>
#include <osg/ShapeDrawable>
#include <osg/Material>
#include <osg/Texture2D>
#include <osg/PositionAttitudeTransform>


int _tmain(int argc, _TCHAR* argv[])
```

We are using the class ShapeDrawable to create a new object in our scene. This class allows us to draw a number of pre-defined shapes, including Sphere, Box, Cone, Cylinder and Capsule. These objects will need to be textured within our code (unlike imported models, which can be textured in the modelling program and the textures imported automatically).

- Firstly we will create a shape. The example uses a box, but you can use whatever shape you prefer.

```
groundNode = osgDB::readNodeFile("models/ground.3ds"); //read in from file the model
to use
      myRoot->addChild(groundNode); // attach the ground node to the root file
```

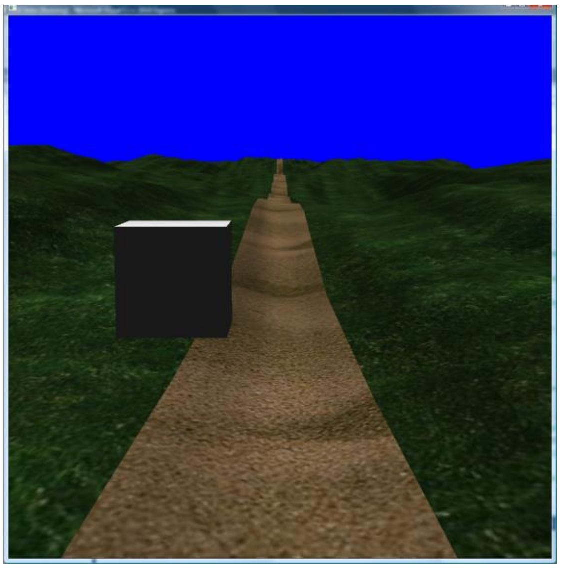
```
	//***************************************************************//

	osg::ref_ptr<osg::Geode> myBox = new osg::Geode(); // this is a node to hold a
drawable object

	myBox->addDrawable(new osg::ShapeDrawable(new osg::Box(osg::Vec3(0.0,0.0,0.0),
1.024))); // the first vector is the centre position, the second parameter is the
width.


	//***************************************************************//

	//*******set up the viewer to display your scene***************//
```

- Now we are going to place our box on a moveable node (Position Attitude Transform or PAT nodes), and attach it to the root node

```
myBox->addDrawable(new osg::ShapeDrawable(new osg::Box(osg::Vec3(0.0,0.0,0.0),
1.024))); // the first vector is the centre position, the second parameter is the
width.

		osg::ref_ptr<osg::PositionAttitudeTransform>  boxXform = new
osg::PositionAttitudeTransform(); // this creates a moveable (transform) node

	boxXform->addChild(myBox.get()); // add the box to the transform node

	myRoot->addChild(boxXform.get()); // adds it all to the root node
```

- Next we need to set the position of the box in the scene.  You might find it easier to comment out your trackball manipulator while you are experimenting with a good position for your object.

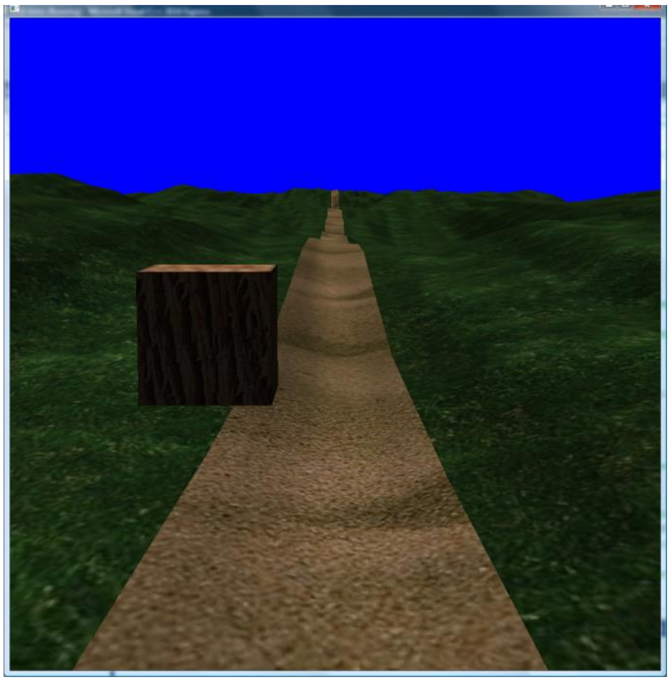
```
myRoot->addChild(boxXform); // adds it all to the root node

	boxXform->setPosition(osg::Vec3(-1,10,1.5 )); // set the desired xyz position
of the box in the scene
```



## Texture the shape

- You will notice that the box doesn't yet have a texture, and so we need to add a texture map. To do this, we first need to create a texture, and then apply it to a stateset. Add the following code to your file, then test the project.

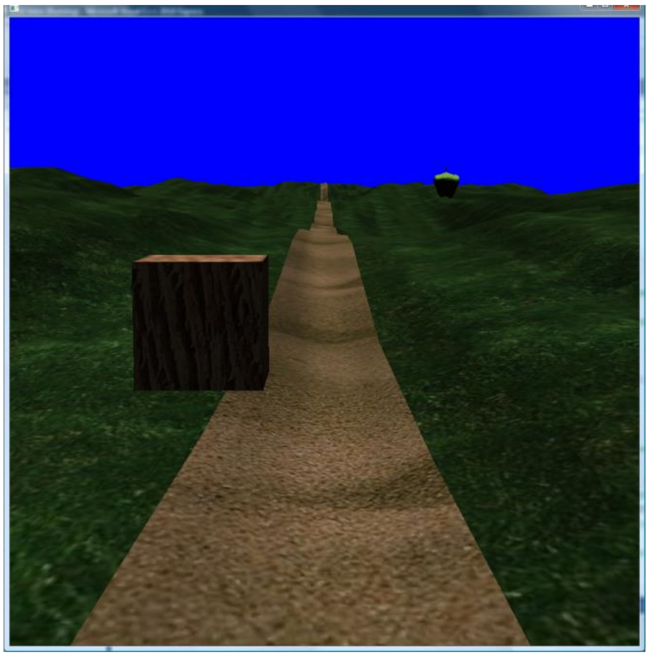
```
boxXform->setPosition(osg::Vec3(-1,10,1.5 )); // set the desired xyz position of the
box in the scene

       //***************************************************************//


       //********** create a texture for the shape  ***********************//


       osg::ref_ptr<osg::Texture2D> woodTexture = new osg::Texture2D; // define a new
texture

       woodTexture->setImage(osgDB::readImageFile("models/bark.jpg"));// load the
texture image from the file

       osg::StateSet* boxStateSet = myBox->getOrCreateStateSet(); // define a state
set which will hold  the texture.

       boxStateSet->setTextureAttributeAndModes(0, woodTexture,
osg::StateAttribute::ON ); // add the texture to the stateset

       myBox->setStateSet(boxStateSet); //apply the stateset to the box node


       //*******set up the viewer to display your scene***************//

       osgViewer::Viewer viewer; // Declare a 'viewer' which will display the scene
```



## Import an object from a file

- You can add models you have created in other software to your OSG projects. We are going to add an apple:

```
myBox->setStateSet(boxStateSet); //apply the stateset to the box node

        //********** create a new node and add the apple object  ******//

    osg::ref_ptr<osg::Node> appleNode = new osg::Node; // create a node for  the
apple

    appleNode = osgDB::readNodeFile("models/apple.3ds"); // read in model for apple

    osg::ref_ptr<osg::PositionAttitudeTransform> appleXform = new
osg::PositionAttitudeTransform(); // set up a transform node for the apple

    appleXform->setPosition(osg::Vec3(1,10,1.5)); // position the apple in the
scene

    myRoot->addChild(appleXform.get()); // add the transform node to the root

    appleXform->addChild(appleNode.get()); // add the apple to the transform node


        //*******set up the viewer to display your scene****************//

    osgViewer::Viewer viewer;
```



Now you have the skills to create a basic OSG scene, experiment with positioning the models in different places, seeing if you can work out how to scale them, or using your own models to load a new scene.