



Working with Open Scene Graph Geometry

Overview:

This section covers some of the methods that can be used to create geometric primitives. There are several ways to deal with geometry objects: at the lowest level loosely wraps OpenGL primitives; an intermediate level using open scene graph basic shapes and at a higher level, loaded from files. This tutorial covers the lowest level. This method provides the greatest flexibility and requires the most effort. Normally at the scene graph level geometry is loaded from files. Most of the effort of tracking vertices is handled by file loaders.

Background:

A brief explanation of the following classes is helpful:

The 'Geode' class:

The geode class is derived from the 'node' class. Nodes (and thus geodes) can be added to a scene graph as leaf nodes. Geode instances can have any number of 'drawables' associated with them.

The 'Drawable' class hierarchy:

The base class 'drawable' is a pure virtual class with four concrete derived classes. The 'geometry' class can have vertex (and vertex data) associated with it directly, or can have any number of 'primitiveSet' instances associated with it.

Vertex and vertex attribute data (color, normals, texture coordinates) is stored in arrays. Since more than one vertex may share the same color, normal or texture coordinate, and array of indices can be used to map vertex arrays to color, normal or texture coordinate arrays.

The 'PrimitiveSet' class:

This class loosely wraps the OpenGL drawing primitives - POINTS, LINES, LINE_STRIP, LINE_LOOP,... QUADS,... POLYGON.

The code:

The following section of code sets up a viewer to see the scene we create, a 'group' instance to serve as the root of the scene graph, a geometry node (geode) to collect drawables, and a geometry instance to associate vertices and vertex data. (In this case the shape to render is a four-sided pyramid.)

```
...
int main()
{
    ...
    osgProducer::Viewer viewer;
    osg::Group* root = new osg::Group();
    osg::Geode* pyramidGeode = new osg::Geode();
    osg::Geometry* pyramidGeometry = new osg::Geometry();
```

Associate the pyramid geometry with the pyramid geode
Add the pyramid geode to the root node of the scene graph.

```
pyramidGeode->addDrawable(pyramidGeometry);
root->addChild(pyramidGeode);
```

Declare an array of vertices. Each vertex will be represented by a triple -- an instances of the vec3 class. An instance of osg::Vec3Array can be used to store these triples. Since osg::Vec3Array is derived from the STL vector class, we can use the push_back method to add array elements. Push back adds elements to the end of the vector, thus the index of first element entered is zero, the second entries index is 1, etc.

Using a right-handed coordinate system with 'z' up, array elements zero..four below represent the 5 points required to create a simple pyramid.

```
osg::Vec3Array* pyramidVertices = new osg::Vec3Array;
pyramidVertices->push_back( osg::Vec3( 0, 0, 0 ) ); // front left
pyramidVertices->push_back( osg::Vec3(10, 0, 0 ) ); // front right
pyramidVertices->push_back( osg::Vec3(10,10, 0 ) ); // back right
pyramidVertices->push_back( osg::Vec3( 0,10, 0 ) ); // back left
pyramidVertices->push_back( osg::Vec3( 5, 5,10 ) ); // peak
```

Associate this set of vertices with the geometry associated with the geode we added to the scene.

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

Next, create a primitive set and add it to the pyramid geometry. Use the first four points of the pyramid to define the base using an instance of the DrawElementsUInt class. Again this class is derived from the STL vector, so the push_back method will add elements in sequential order. To ensure proper backface culling, vertices should be specified in counterclockwise order. The arguments for the constructor are the enumerated type for the primitive (same as the OpenGL primitive enumerated types), and the index in the vertex array to start from.

```
osg::DrawElementsUInt* pyramidBase =
    new osg::DrawElementsUInt(osg::PrimitiveSet::QUADS, 0);
pyramidBase->push_back(3);
pyramidBase->push_back(2);
pyramidBase->push_back(1);
pyramidBase->push_back(0);
pyramidGeometry->addPrimitiveSet(pyramidBase);
```

Repeat the same for each of the four sides. Again, vertices are specified in counterclockwise order.

```
osg::DrawElementsUInt* pyramidFaceOne =
    new osg::DrawElementsUInt(osg::PrimitiveSet::TRIANGLES, 0);
pyramidFaceOne->push_back(0);
pyramidFaceOne->push_back(1);
pyramidFaceOne->push_back(4);
pyramidGeometry->addPrimitiveSet(pyramidFaceOne);
```

```
osg::DrawElementsUInt* pyramidFaceTwo =
    new osg::DrawElementsUInt(osg::PrimitiveSet::TRIANGLES, 0);
pyramidFaceTwo->push_back(1);
pyramidFaceTwo->push_back(2);
pyramidFaceTwo->push_back(4);
pyramidGeometry->addPrimitiveSet(pyramidFaceTwo);
```

```
osg::DrawElementsUInt* pyramidFaceThree =
    new osg::DrawElementsUInt(osg::PrimitiveSet::TRIANGLES, 0);
pyramidFaceThree->push_back(2);
pyramidFaceThree->push_back(3);
```

```

pyramidFaceThree->push_back(4);
pyramidGeometry->addPrimitiveSet(pyramidFaceThree);

osg::DrawElementsUInt* pyramidFaceFour =
    new osg::DrawElementsUInt(osg::PrimitiveSet::TRIANGLES, 0);
pyramidFaceFour->push_back(3);
pyramidFaceFour->push_back(0);
pyramidFaceFour->push_back(4);
pyramidGeometry->addPrimitiveSet(pyramidFaceFour);

```

Declare and load an array of Vec4 elements to store colors.

```

osg::Vec4Array* colors = new osg::Vec4Array;
colors->push_back(osg::Vec4(1.0f, 0.0f, 0.0f, 1.0f) ); //index 0 red
colors->push_back(osg::Vec4(0.0f, 1.0f, 0.0f, 1.0f) ); //index 1 green
colors->push_back(osg::Vec4(0.0f, 0.0f, 1.0f, 1.0f) ); //index 2 blue
colors->push_back(osg::Vec4(1.0f, 1.0f, 1.0f, 1.0f) ); //index 3 white

```

Declare the variable that will match vertex array elements to color array elements. This vector should have the same number of elements as the number of vertices. This vector serves as a link between vertex arrays and color arrays. Entries in this index array correspond to elements in the vertex array. Their values correspond to the index in the color array. This same scheme would be followed if vertex array elements were matched with normal or texture coordinate arrays.

Note that in this case, we are assigning 5 vertices to four colors. Vertex array element zero (bottom left) and four (peak) are both assigned to color array element zero (red).

```

osg::TemplateIndexArray
<unsigned int, osg::Array::UIntArrayType,4,4> *colorIndexArray;
colorIndexArray =
    new osg::TemplateIndexArray<unsigned int, osg::Array::UIntArrayType,4,4>;
colorIndexArray->push_back(0); // vertex 0 assigned color array element 0
colorIndexArray->push_back(1); // vertex 1 assigned color array element 1
colorIndexArray->push_back(2); // vertex 2 assigned color array element 2
colorIndexArray->push_back(3); // vertex 3 assigned color array element 3
colorIndexArray->push_back(0); // vertex 4 assigned color array element 0

```

The next step is to associate the array of colors with the geometry, assign the color indices created above to the geometry and set the binding mode to `_PER_VERTEX`.

```

pyramidGeometry->setColorArray(colors);
pyramidGeometry->setColorIndices(colorIndexArray);
pyramidGeometry->setColorBinding(osg::Geometry::BIND_PER_VERTEX);

osg::Vec2Array* texcoords = new osg::Vec2Array(5);
(*texcoords)[0].set(0.00f,0.0f);
(*texcoords)[1].set(0.25f,0.0f);
(*texcoords)[2].set(0.50f,0.0f);
(*texcoords)[3].set(0.75f,0.0f);
(*texcoords)[4].set(0.50f,1.0f);
pyramidGeometry->setTexCoordArray(0,texcoords);

```

Now that we have created a geometry node and added it to the scene we can reuse this geometry. For example, if we wanted to put a second pyramid 15 units to the right of the first one, we could add this geode as the child of a transform node in our scene graph.

```

// Declare and initialize a transform node.
osg::PositionAttitudeTransform* pyramidTwoXForm =
    new osg::PositionAttitudeTransform();

// Use the 'addChild' method of the osg::Group class to
// add the transform as a child of the root node and the
// pyramid node as a child of the transform.

```

```
root->addChild(pyramidTwoXForm);
pyramidTwoXForm->addChild(pyramidGeode);

// Declare and initialize a Vec3 instance to change the
// position of the tank model in the scene

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

The final step is to set up and enter a simulation loop.

```
viewer.setUpViewer(osgProducer::Viewer::STANDARD_SETTINGS);
viewer.setSceneData( root );

viewer.realize();

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

Here's the [download](#). Good luck!

This tutorial set's [index](#), [contact page author](#). (jasullivan <at> nps <dot> edu)

Jason McVeigh's OpenSceneGraph [tutorial set](#).