



## Creating Textured Geometry using StateSets

### Goals:

Add a texture to geometry defined by OpenGL drawing primitives introduced in [tutorial two](#)

### Background:

The previous tutorial introduced viewing scenes that include basic shapes created from OpenGL primitives. This section explains how to add textures to these shapes. To make the code easier to use, we'll put the pyramid code into a function that creates a new geode and returns a pointer to it. The following code is from tutorial two. Changes are highlighted in red. These changes involve putting the code into a function and creating texture coordinates for our list of vertices.

```
osg::Geode* createPyramid()
{
    osg::Geode* pyramidGeode = new osg::Geode();
    osg::Geometry* pyramidGeometry = new osg::Geometry();
    pyramidGeode->addDrawable(pyramidGeometry);

    // Specify the vertices:
    osg::Vec3Array* pyramidVertices = new osg::Vec3Array;
    pyramidVertices->push_back( osg::Vec3(0, 0, 0) ); // front left
    pyramidVertices->push_back( osg::Vec3(2, 0, 0) ); // front right
    pyramidVertices->push_back( osg::Vec3(2, 2, 0) ); // back right
    pyramidVertices->push_back( osg::Vec3( 0,2, 0) ); // back left
    pyramidVertices->push_back( osg::Vec3( 1, 1,2) ); // peak

    //Associate this set of vertices with the geometry associated with the
    //geode we added to the scene.
    pyramidGeometry->setVertexArray( pyramidVertices );

    // Create a QUAD primitive for the base by specifying the
    // vertices from our vertex list that make up this QUAD:
    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);

    // Add this primitive to the geometry: pyramidGeometry->addPrimitiveSet(pyramidBase

    // code to create other faces goes here!
    // (removed to save space, see tutorial two)

    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

    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

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

// Since the mapping from vertices to texture coordinates is 1:1,
// we don't need to use an index array to map vertices to texture
// coordinates. We can do it directly with the 'setTexCoordArray'
// method of the Geometry class.
// This method takes a variable that is an array of two dimensional
// vectors (osg::Vec2). This variable needs to have the same
// number of elements as our Geometry has vertices. Each array element
// defines the texture coordinate for the corresponding vertex in the
// vertex array.
osg::Vec2Array* texcoords = new osg::Vec2Array(5);
(*texcoords)[0].set(0.00f,0.0f); // tex coord for vertex 0
(*texcoords)[1].set(0.25f,0.0f); // tex coord for vertex 1
(*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);

return pyramidGeode;
}

```

### Loading a Texture, Creating a State Set, assigning it to a Node

The method for rendering primitives is controlled using StateSets. This section of code demonstrates how to load a texture from file, create a StateSet in which this texture is enabled, and assign this StateSet to a node in the scene:

```

int main()
{
    osgProducer::Viewer viewer;

    // Declare a group to act as root node of a scene:
    osg::Group* root = new osg::Group();

    osg::Geode* pyramidGeode = createPyramid();

    root->addChild(pyramidGeode);

    osg::Texture2D* KLN89FaceTexture = new osg::Texture2D;
    // protect from being optimized away as static state:
    KLN89FaceTexture->setDataVariance(osg::Object::DYNAMIC);

    // load an image by reading a file:
    osg::Image* klnFace = osgDB::readImageFile("KLN89FaceB.tga");
    if (!klnFace)
    {
        std::cout << " couldn't find texture, quitting." << std::endl;
        return -1;
    }

    // Assign the texture to the image we read from file:
    KLN89FaceTexture->setImage(klnFace);

    // Create a new StateSet with default settings:
    osg::StateSet* stateOne = new osg::StateSet();

    // Assign texture unit 0 of our new StateSet to the texture
    // we just created and enable the texture.
    stateOne->setTextureAttributeAndModes
        (0,KLN89FaceTexture,osg::StateAttribute::ON);
}

```

```
// Associate this state set with the Geode that contains
// the pyramid:
pyramidGeode->setStateSet(stateOne);

//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();
}
return 0;
}
```

Source code [here](#).

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

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