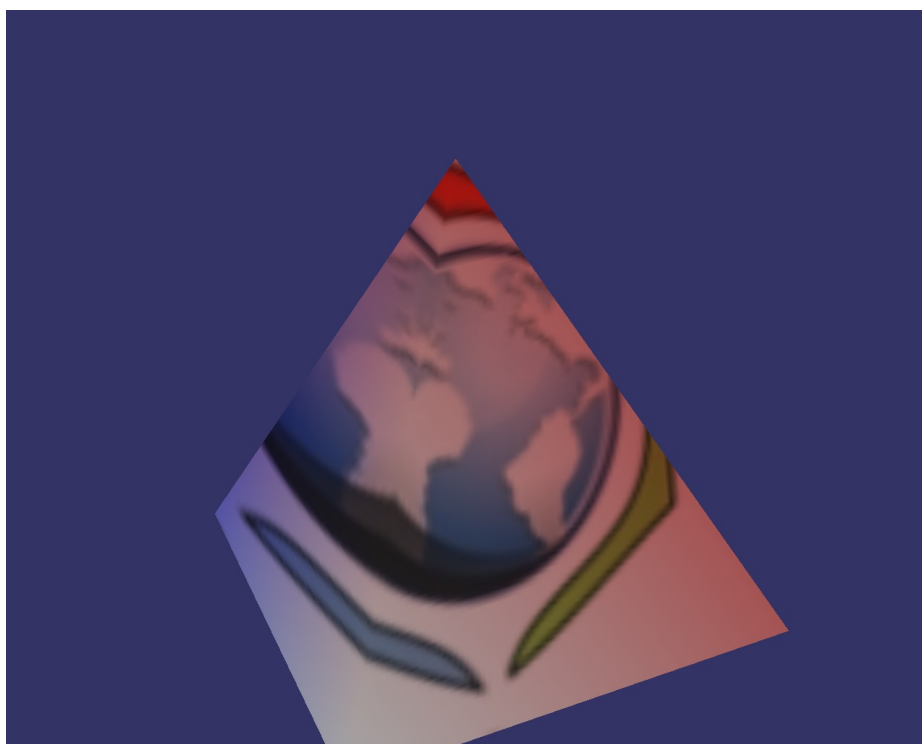


Travailler avec Open Scene Graph

2. Les Textures

Traduction par Thomas Baquet



Source: <http://www.openscenegraph.org/projects/osg/wiki/Support/Tutorials/> -by **Joseph Sullivan**

Traduction: Mars 2008

Sources: 2003-2006

NdT: Avant de commencer

Il est possible qu'il y ait encore des erreurs de traduction, si c'est le cas, prévenez-moi. Aucune responsabilité ne peut cependant être engagée contre le traducteur ou l'auteur suite à ces erreurs, ou à quelque erreur à la lecture de la traduction.

Il y a eu également eu des mots impossibles à retraduire en les mettant en contexte dans la traduction (je pense à des mots comme « geometry » par exemple qui est employé comme nom simple). Dans ces cas, j'ai soit employé le mot en anglais, soit tourné la phrase de façon à garder le même sens.

En lisant ce document, le lecteur prend pleinement conscience de ce qui vient d'être écrit.

But du tutoriel

Ajouter une texture à la forme géométrique dessinée par les primitives d'OpenGL que nous avons vu dans le tutoriel précédent.

Avant de commencer

Le tutoriel précédent nous a appris à créer des scènes incluant des formes de base créées à partir de primitives d'OpenGL. Cette section va expliquer comment leur ajouter des textures. Pour rendre le code plus facile à utiliser, nous allons mettre le code de la pyramide dans une fonction qui crée une nouvelle instance de Geode et retourne un pointeur dessus. Le code suivant vient donc du tutoriel précédent.

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

    // Spécification des vertices:
    osg::Vec3Array* pyramidVertices = new osg::Vec3Array;
    pyramidVertices->push_back( osg::Vec3(0, 0, 0) ); // Devant gauche
    pyramidVertices->push_back( osg::Vec3(2, 0, 0) ); // Devant droite
    pyramidVertices->push_back( osg::Vec3(2, 2, 0) ); // Derrière droite
    pyramidVertices->push_back( osg::Vec3( 0,2, 0) ); // Derrière gauche
    pyramidVertices->push_back( osg::Vec3( 1, 1,2) ); // Sommet

    // Associe cet ensemble de vertices avec pyramidGeometry lui-même
    // associé avec pyramidGeode
    pyramidGeometry->setVertexArray( pyramidVertices );

    // Crée une primitive QUAD pour la base
    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);

    // L'ajoute à PyramidGeometry:
    // pyramidGeometry->addPrimitiveSet(pyramidBase);
    // Le code pour créer d'autres faces ici!
    // (retiré pour raison d'espace, voir tutoriel précédent)

    // Création des couleurs...
    osg::Vec4Array* colors = new osg::Vec4Array;
    colors->push_back(osg::Vec4(1.0f, 0.0f, 0.0f, 1.0f) ); //index 0 rouge
    colors->push_back(osg::Vec4(0.0f, 1.0f, 0.0f, 1.0f) ); //index 1 vert
    colors->push_back(osg::Vec4(0.0f, 0.0f, 1.0f, 1.0f) ); //index 2 bleu
    colors->push_back(osg::Vec4(1.0f, 1.0f, 1.0f, 1.0f) ); //index 3 blanc

    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 assigné à la couleur 0
colorIndexArray->push_back(1); // vertex 1 assigné à la couleur 1
colorIndexArray->push_back(2); // vertex 2 assigné à la couleur 2
colorIndexArray->push_back(3); // vertex 3 assigné à la couleur 3
colorIndexArray->push_back(0); // vertex 4 assigné à la couleur 0

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

// Comme le mapping des vertices aux coordonnées de texture est 1:1,
// nous n'avons pas besoin d'utiliser un index de tableau pour mapper
// les vertices aux coordonnées de textures. On peut le faire direc-
// tement avec la méthode 'setTexCoordArray' de la classe Geometry.
// Cette méthode prend une variable qui est un tableau de vecteur de
// deux dimensions (osg::Vec2). Cette variable a besoin d'avoir le
// même nombre de vertices que dans notre Geometry. Chaque élément du
// tableau définit les coordonnées de textures pour correspondre aux
// vertices dans le tableau de vertices.
osg::Vec2Array* texcoords = new osg::Vec2Array(5);
(*texcoords)[0].set(0.00f,0.0f); // coord. de texture pr. vertex 0
(*texcoords)[1].set(0.25f,0.0f); // coord. de texture pr. 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;
}

```

Charger une texture, créer un état, et assignation à un noeud

La méthode pour rendre des primitives est contrôlée par StateSets. Cette section de code montre comment charger une texture d'un fichier, créer un état StateSet dans lequel cette texture sera activée, et assigner cet état à un noeud de la scène. La première section commence comme dans le tutoriel précédent. On initialise une vue et construit une scène avec une simple pyramide.

```
int main()
{
    osgProducer::Viewer viewer;

    // Déclare une groupe comme noeud principal de la scène:
    osg::Group* root = new osg::Group();
    osg::Geode* pyramidGeode = createPyramid();
    root->addChild(pyramidGeode);
```

Maintenant, il faut ajouter une texture. Ici nous allons déclarer une instance de texture et lui mettre ses données comme « DYNAMIC ». (Si nous ne le faisons pas, certaines routines d'optimisation d'Osg pourraient les retirer). La classe de texture encapsule les modes de texture d'OpenGL (wrap, filter, etc) comme beaucoup d'autre, comme osg::Image. Le code suivant montre comment lire une instance osg::Image depuis un fichier et l'associer à une texture.

```
osg::Texture2D* KLN89FaceTexture = new osg::Texture2D;

// protection pour les optimisations:
KLN89FaceTexture->setDataVariance(osg::Object::DYNAMIC);

// charge une image par la lecture d'un fichier
osg::Image* klnFace = osgDB::readImageFile("KLN89FaceB.tga");
if (!klnFace)
{
    std::cout << " couldn't find texture, quitting." << std::endl;
    return -1;
}

// Assigne la texture que nous avons lue depuis le fichier:
KLN89FaceTexture->setImage(klnFace);
```

Les textures peuvent être associées avec un renderer StateSets. La prochaine étape est de créer un StateSet, associer et activer notre texture à cet état, assigner et mettre notre StateSet à pyramidGeode.

```
// Crée un nouveau StateSet avec les options par défaut:
osg::StateSet* stateOne = new osg::StateSet();

// Assigne notre texture à la texture 0 de notre StateSet
stateOne->setTextureAttributeAndModes
```

```
(0,KLN89FaceTexture,osg::StateAttribute::ON);  
// Associe cet état avec le Geode qui contient la pyramide  
pyramidGeode->setStateSet(stateOne);
```

La dernière étape est la boucle de simulation:

```
//L'étape finale est de préparer la boucle et d'entrer dedans  
viewer.setUpViewer(osgProducer::Viewer::STANDARD_SETTINGS);  
viewer.setSceneData( root );  
viewer.realize();  
  
while( !viewer.done() )  
{  
    viewer.sync();  
    viewer.update();  
    viewer.frame();  
}  
return 0;  
}
```

Joyeux coding!