OSG : initiation graphe de scène
Démineur : modélisation - animation

TP1 : initiation / Configuration / Géométrie / Etat

1. Environnement TP / application minimale OSG

ATTENTION : La précédure proposée ci-dessous vous permettra de configurer un environnement de développement. Il s'agit surtout d'informations qui vous serviront à installer cette bibliothèque ***sur votre poste personnel*** *et pouvoir continuer votre travail et la découverte de la bibliothèque chez vous.*

1.1. Installation sur votre PC

Pour un travail dans les salles de TP de l'IUT, vous pouvez directement passer au paragraphe 1.3.

1.1.1. Installation simplifiée à l'aide de binaires (sources déjà compilées)

Dans l'environnement de développement de l'IUT : Windows 7, C++ (Visual Studio 2010) :
Télécharger les librairies Windows déjà compilées sur cette page : (version utilisée 3.2.1 & MS VS2010)
Télécharger les archives debug et release.

⇨ Décompresser l'archive Debug dans un dossier OSG321.

Votre dossier OSG321 doit avoir la structure suivante :

```
OSG321
        |__ include
        |__ data
        |__ lib
        |__ doc
        |__ share
        |__ bin
```

⇨ Décompresser l'archives Release dans ce même dossier OSG321.
S'Il vous est demandé de fusionner les dossiers : répondez OUI pour tous
S'Il vous est demandé de remplacer les fichiers, répondez NON pour tous
S'il ne vous est rien demandé, vous vous êtes trompé ! ☹

Votre dossier `OSG321` a gardé la même structure.
Les librairies statiques (extension `.lib`) se trouvent dans le dossier `lib`.
Les librairies dynamiques (extension `.dll`) se trouvent dans le dossier `bin`.
Les fichiers d'entête (extension `.h`) se trouvent dans le dossier `include`.
Dans le dossier `lib` et `bin` se trouvent à la fois les versions debug ET release des librairies (les versions debug des bibliothèques ont la lettre **d** juste avant l'extension).

Ajouter les exemples :

Le dossier data contient les données utiles pour l'exécution des exemples.

Pour le code source des exemples, télécharger et décompresser l'archive sur cette page.

Copier le dossier `examples` (uniquement) dans votre dossier `OSG321` qui a donc la structure suivante :

```
OSG321
        |__ include
        |__ data
        |__ examples
        |__ lib
        |__ doc
        |__ share
        |__ bin
```

1.1.2. Compilation d'OpenSceneGraph

! ne pas le faire sur les postes de l'IUT mais éventuellement sur votre PC !

➔ *C'est une alternative. Plutôt que de copier les binaires déjà compilés, une solution est de compiler la bibliothèque à partir des sources sur votre poste. Un intérêt est de toujours travailler avec la dernière version de la bibliothèque et d'obtenir des binaires compatibles avec votre environnement. (Infos utile pour installation MsVS disponibles sur cette page)*

Remarque : La procédure ci-dessous est détaillée (avec capture d'écrans) dans le document « `compiler OSG 3.2.1 janvier 2014.pdf` ».

Il faut installer au préalable CMake (disponible sur cette page, version actuelle 2.8.10).

Pour le code source de la bibliothèque OpenSceneGraph, il faut télécharger puis décompresser l'archive disponible sur cette page (version actuelle 3.2.1).

Lancer CMake : a) indiquer l'emplacement des fichiers sources (« where is the source code «) b) indiquer le dossier où placer les projets visual et les binaires résultats de la compilation. («where to build the binaries ») b) cliquer sur configure pour indiquer quel est le compilateur cible (ici visual 2010) d) cliquer sur generate

Les fichiers projets sont générés et placés dans le dossier cible indiqué. Ouvrir visual 2010. Compiler le projet `ALL_BUILD` en mode *release*. Compiler le projet `ALL_BUILD` en mode *debug*. Les sous dossier bin et lib contiennent respectivement les bibliothèques dynamique (dll) et statiques (lib).

*Remarque : La compilation fournit une **bibliothèque dynamique**. La bibliothèque statique `.lib` est un fichier d'importation des fonctions et indique dans quelles dll elles sont disponibles. La bibliothèque n'est donc pas statique : le .lib ne contient aucun code qui serait directement intégré à votre exécutable. Au moment, de l'exécution, votre application avoir accès aux DLL dont elle a besoin.*

1.2. Configuration / Variables d'environnement

(pour la configuration du poste de travail à l'IUT, voir aussi 1.5)

Panneau de configuration → Système → Modifier les variables d'environnement …

Propriétés Système → Variablement d'environnement …

Variablement d'environnement → Variablement système → Nouvelle :

OSG_ROOT	=le_chemin_vers_le_dossier\OSG321 (par exemple C:\OSG321)
OSG_LIB_PATH	%OSG_ROOT%\lib
OSG_FILE_PATH	%OSG_ROOT%\data
OSG_INCLUDE_PATH	%OSG_ROOT%\include
OSG_BIN_PATH	%OSG_ROOT%\bin
OSG_NOTIFY_LEVEL	NOTICE
OSG_SAMPLES_PATH	%OSG_ROOT%\bin
La variable PATH doit contenir (entre autres) :	… ;%OSG_BIN_PATH%;%OSG_FILE_PATH%; %OSG_SAMPLES_PATH% ; ….

*Comment connaître les variables d'environnement de notre poste : ligne de commande (cmd) puis **SET***

1.3. Installation sur les PC de l'IUT

Chaque année, je recompile les codes sources de la dernière version pour la configuration des postes de l'IUT : en janvier 2014 : Windows 7, MsVS10 , 64 bits , OSG 3.2.1

- ⇨ Récupérer les binaires sur le réseau
- ⇨ Le copier sur la racine du disque C.
- ⇨

1.4. Test de l'installation d'OSG :

dans la console (cmd.exe), tapez :

```
osgversion.exe
```

Pour un premier test , tapez :

```
osgviewer cow.osg
```

cow.osg est une ressource qui se trouve dans OSG_ROOT \ DATA

pour l'exécuter dans une fenêtre :

```
osgviewer –-window 100 100 500 500 cow.osg
```

pour obtenir des statistiques à l'exécution :

```
appuyer plusieurs fois sur la touche 'S' pendant l'exécution
```

Obtenir tous les paramètres d'exécution : tapez sur la ligne de commande

```
osgviewer –h
```

Obtenir toutes les options pendant l'exécution :

```
touche « H »  (dans la fenêtre graphique d'exécution)
```

e.g. changer de caméra, de texture, de représentation (filaire , solide, points) , etc ….

1.5. Configurer l'IDE MS VS sur votre PC

(pour la configuration du poste de travail à l'IUT, voir aussi 1.5)

Créer un projet (type : projet vide)
Ajouter un fichier Source (type : fichier C++)

Définir les propriétés de votre projet (bouton droit sur nom du projet) en mode debug:

- Choisir Configuration DEBUG
- Fixer les paramètres suivants

C/C++ → Général → Autres répertoires Include : **$(OSG_INCLUDE_PATH)**
Editeur de liens → Général → répertoire de bibliothèques supplémentaires : **$(OSG_LIB_PATH)**
Editeur de liens → entrée → dépendances supplémentaires :
osgd.lib ; osgGAd.lib ; osgDBd.lib ; osgViewerd.lib ; osgTextd.lib ; osgUtild.lib ; OpenThreadsd.lib ;

Enfin, Définir les propriétés de votre projet en mode release:

- Choisir Configuration **RELEASE**
- Fixer les paramètres suivants

C/C++ → Général → Autres répertoires Include : **$(OSG_INCLUDE_PATH)**
Editeur de liens → Général → répertoire de bibliothèques supplémentaires : **$(OSG_LIB_PATH)**
Editeur de liens → entrée → dépendances supplémentaires :
osg.lib osgGA.lib osgDB.lib osgViewer.lib osgText.lib osgUtil.lib OpenThreads.lib

(pour un environnement de travail à emporter, voir aussi 1.8)

1.6. Configuration des salles de TP de l'IUT

Vous ne pouvez pas fixer les variables d'environnement (doits administrateurs requis pour 1.2). **Ces variables n'ont probablement pas été définies DONC :**

⇨ Il vous faudra indiquer les chemins « en dur », e.g. c:\OSG321\..., pour configurer votre IDE(voir 1.4).
⇨ Pour les tests de commandes en ligne(voir 1.3), il faudra vous placer dans le dossier bin de la distribution Release.
⇨ Pour qu'à l'exécution, Visual puisse trouver les DLL, il faudra configurer votre projet de la façon suivante :

- Bouton Droit sur le projet -> « propriétés »
- Dans configuration -> Choisir « Toutes les Configurations »
- Dans Débogage -> Environnement : ajouter le chemin d'accès aux DLL au PATH de l'OS de la façon suivante : « PATH=%PATH% ;c:\.....\bin »

A ce stade , vous pouvez commencer votre projet !

1.7. Premier test de compilation

Dans un fichier source, ajouter le code suivant *(code minimal pour compiler mais pas à exécuter !!)*

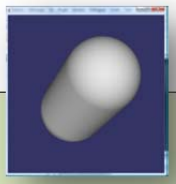
```
#include <windows.h >
#include <osgViewer/Viewer>
int main() {
return  (new osgViewer::Viewer)->run();
}
```

Compilez Touche «F7»

Ne pas exécuter ! Pas de graphe de scène défini, donc erreur à l'exécution !

S'il y a des erreurs de compilation, vérifier la configuration (compilez en x64 et non Win32)

1.8. Premier test d'exécution



Modifier avec le code suivant et exécuter :

```
#include <osgViewer/Viewer>
#include <osg/ShapeDrawable >
using namespace osg;
int main() {
ref_ptr<Group>           root (new Group);
ref_ptr<Geode>           myshapegeode (new Geode);
ref_ptr<Capsule>         myCapsule (new Capsule(Vec3f(),1,2));
ref_ptr<ShapeDrawable> capsuledrawable
                                  (new ShapeDrawable(myCapsule.get()));
myshapegeode->addDrawable(capsuledrawable.get());
root->addChild(myshapegeode.get());
osgViewer::Viewer * viewer = new osgViewer::Viewer;
viewer->setSceneData(root.get());
viewer->setUpViewInWindow(50, 50, 512, 512);
return viewer->run();
}
```

1.9. Environnement de développement nomade

Si vous voulez pouvoir emporter votre projet et le poursuivre sur n'importe quel PC :

- placer toute l'installation OpenSceneGraph dans un dossier, e.g. OSG321.
- Y ajouter un sous dossier MesProjetsVC++ (vous y placerez vos dossiers projet Visual)
- Dans la configuration de votre projet dans visual, remplacer les variables d'environnement par des chemins relatifs : **$(OSG_INCLUDE_PATH)** **par** **../../include**
  $(OSG_LIB_PATH) **par** **../../lib**

Votre dossier OSG321 ressemble à :

```
OSG321
       |__ include
       |__ datasets
       |__ lib
       |__ doc
       |__ share
       |__ bin
       |__ examples
       |__ MesProjetsVC++
             |__ MonProjet
                   |__ debug
```

2. Graphe de scène

2.1. Graphe de scène : Créer un objet/une géométrie

2.1.1. Géométrie : Créer une face

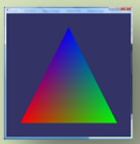
La méthode qui permet de créer le nœud Triangle :

```
ref_ptr<Node> creerTriangle() {

ref_ptr<Geometry> gdeTriangle = new Geometry;
// un tableau de trois sommets.
Vec3Array* tabSommet = new Vec3Array;
tabSommet->push_back(Vec3(-1, 0, -1));
tabSommet->push_back(Vec3(1, 0, -1));
tabSommet->push_back(Vec3(0, 0, 1));
// Ajouter le tableau de sommets a l'objet Geometry.
gdeTriangle ->setVertexArray(tabSommet);
// Créer une primitive Triangle, y ajouter les sommets
DrawElementsUInt* pPrimitiveSet =
new DrawElementsUInt( PrimitiveSet::TRIANGLES, 0 );
pPrimitiveSet->push_back(0);  // sommets = points n° 0,1 et 2
pPrimitiveSet->push_back(1);
pPrimitiveSet->push_back(2);
// Ajouter la primitive à l'objet Geometry.
gdeTriangle->addPrimitiveSet(pPrimitiveSet);
// Ajouter un tableau de couleurs.
// Chaque sommet du triangle aura une couleur différente.
Vec4Array* tabCouleur = new Vec4Array;
tabCouleur->push_back(Vec4(1.0f, 0.0f, 0.0f, 1.0f));
tabCouleur->push_back(Vec4(0.0f, 1.0f, 0.0f, 1.0f));
tabCouleur->push_back(Vec4(0.0f, 0.0f, 1.0f, 1.0f));
gdeTriangle->setColorArray(tabCouleur);
// s'assurer que le triangle utilisera une couleur par sommet.
gdeTriangle->setColorBinding(Geometry::BIND_PER_VERTEX);
// créer la Geode, y associer la géométrie
ref_ptr<Geode> nodeGeo = new Geode;
nodeGeo->addDrawable(gdeTriangle);
return nodeGeo.get();   }
```

Dans le main, créer le graphe de scène : définir une racine (Group) et y associer le nœud triangle :

```
int main()
 {
      ref_ptr<Group>           root (new Group);
      root->addChild(creerTriangle());
      osgViewer::Viewer * viewer = new osgViewer::Viewer;
      viewer->setSceneData(root.get());
      return viewer->run();
}
```



Pour info :

Pour obtenir les mêmes statistiques que celle de osgviewer.exe, il faut ajouter les lignes suivantes :

```
#include <osgViewer/ViewerEventHandlers>
viewer->addEventHandler(new osgViewer::StatsHandler);
```

Pour avoir une exécution dans une fenêtre :

```
viewer->setUpViewInWindow(50, 50, 512, 512);
```

2.1.2. Optimisation : suppression de faces (phase de culling)

Si vous faîtes tourner votre objet, vous voyez les deux faces !!

Dans le main, ajouter la ligne suivante.

```
root->getOrCreateStateSet()->setAttributeAndModes(
                                    new CullFace(), StateAttribute::ON);
```

Exécuter et faire tourner votre objet…. Qu'observez vous ?

Ouvrez la documentation de la classe CullFace :

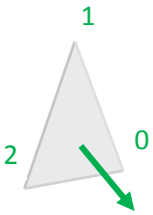
- que dire du constructeur utilisé ici ?
- quels sont ses arguments ?
- quels sont les alternatives ?

Inverser les lignes suivantes :

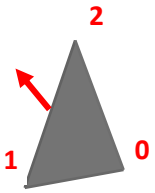
```
pPrimitiveSet->push_back(2);     // pPrimitiveSet->push_back(1);
pPrimitiveSet->push_back(1);    // pPrimitiveSet->push_back(2);
```

Exécuter et faire tourner votre objet….

Qu'observez vous ? Expliquez…..



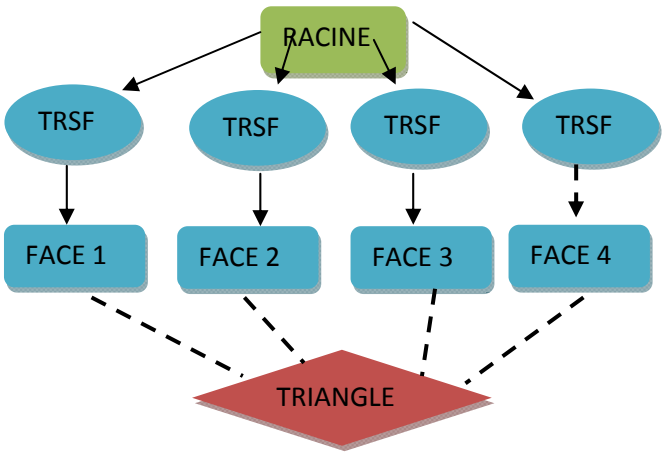
Cela montre que l'ordre des références de points ajoutées au tableau des primitives influe sur la normale de ces primitives : à ajouter dans le sens trigonométrique !!

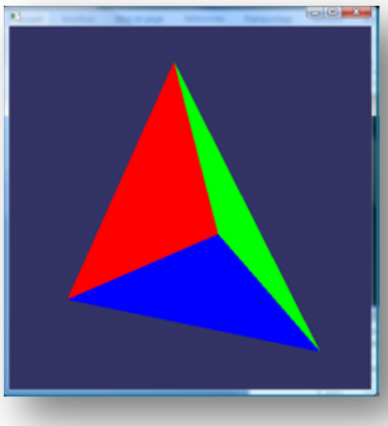


2.1.3.

2.1.4. Géométrie : Créer une pyramide

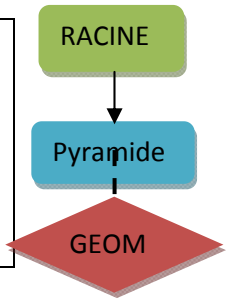
Première Solution : Regrouper sous un même nœud 4 nœuds fils faisant référence à une même géométrie partagée (triangle) avec à chaque fois une transformation pour positionner les triangles les uns par rapports aux autres pour obtenir un tétraèdre régulier … possible mais trop lourd pour une simple pyramide !



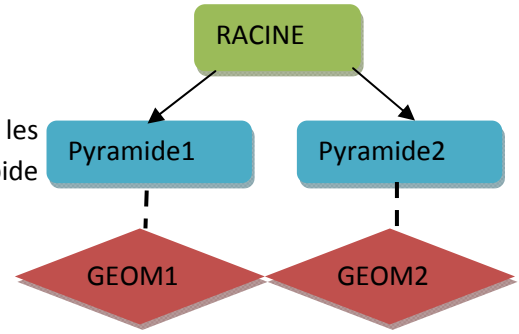


Comme souvent en développement 3D Tps Réel, c'est une histoire de compromis (place mémoire, tps d'exécution , réalisme, …). Le principe ici est intéressant : partager un drawable (la géométrie du triangle) sur plusieurs objets. Mais le gain ici n'est pas suffisant pour compenser la contrepartie : mettre en place 3 nœuds de transformations (stockage et exécution)

Vous allez développer la ***deuxième solution*** : Ajouter un point `(0, 2, 0)` et définir les 4 faces de la pyramide. Modifier la méthode CréerTriangle qui deviendra CreerPyramide.
Pour associer une couleur par face, vous changerez l'argument de `setColorBinding` (voir la documentation)
Dans vos `pPrimitiveSet->push_back`, attention à placer les points dans le sens trigonométrique si vous voulez que la normale soit bien orientée.



2.1.5. Transformation : Ajouter une deuxième pyramide

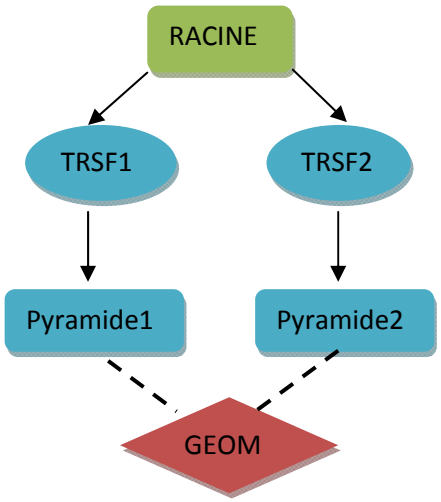


1ère Solution : Copier-coller le code précédent en modifiant les coordonnées des sommets (e.g. + 20 pour chaque coordonnées x) : Rapide à coder mais inefficace (géométrie stockée deux fois)

Nous ne retiendrons évidemment pas cette méthode.

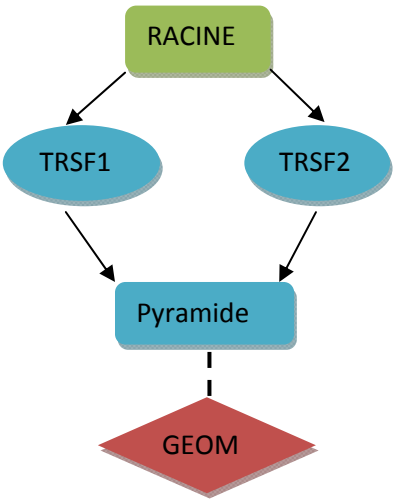
2ème Solution : Placer deux groupes de transformation qui permettront de positionner les pyramides dans la scène en (-6,0,0) et (6,0,0). Vous pourrez essayer ces deux alternatives de graphe de scène et comparer les résultats obtenus !

La géométrie est partagée.
par 2 géodes



OU

Une seule géode est associée à
2 nœuds de transformations



A Vérifier :

si vous ne placez qu'un objet « translaté », on ne voit aucune différence à l'exécution : c'est normal !
La classe viewer utilisée positionne la caméra de sorte à voir la scène centrée et en entier. Quand vous placez les deux objets , la caméra pointera vers un point à mi distance entre les deux objets et aura suffisamment de recul pour voir les deux objets.

2.1.6. POO : Proposer une CLASSE pyramide GeodePyramide

On souhaite réutiliser le code précédent simplement. Faire une classe géométrie dérivée de Geode.
Proposer en paramètre une dimension (ou facteur d'échelle) ainsi qu'un tableau de 4 couleurs.
Utiliser cette classe pour ajouter une troisième pyramide dans votre scène (par défaut en (0,0,0)).

```
root->addChild(new GeodePyramide(1,BLANC,ROUGE,BLEU,BLANC));
```