TP2 : Graphe de scène / Transformation / Matériau

2.2. Graphe de scène : Organiser et structurer

Vous allez écrire une fonction `ref_ptr<Group> creerDemineur()` qui retournera un graphe de scène où regroupés les différentes composantes d'un objet.

2.2.1. Créer la méthode ref_ptr<Group> creerScene() { }

```
ref_ptr<Group> creerDemineur() {
      ref_ptr<Group> grpDemineur= new Group;
      // à faire …
      return grpDemineur.get();
}
```

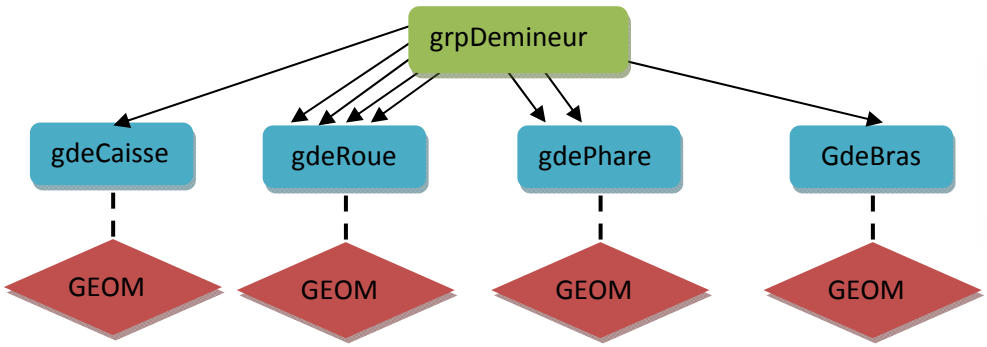
Elle sera appelée dans le main :

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

2.2.2. Graphe de scène : Ajouter des objets prédéfinis

La fonction **CreerDemineur**() va construire un véhicule à partir d'objets prédéfinis d'OSG:
Créer et ajouter au graphe ***grpDemineur*** un geode **gdeCaisse** à partir d'une boîte (*Box*) pour la caisse.
Créer un geode **gdeRoue** à partir d'un cylindre(*Cylinder*). L'ajouter 4 fois à la scène.
Créer et ajouter au graphe un geode **gdeBras** à partir d'un cylindre (*Cylinder*) pour le bras.
Créer un geode **gdePhare** (classe *GeodePyramide*, *voir fin du TP1*). L'ajouter 2 fois à la scène.
NB : voir code en 1.6 pour récupérer la géométrie de ces objets prédéfinis et les associer aux géodes.

➔ Pour l'instant, les objets sont « mélangés », car tous positionnées en (0,0,0) par défaut !



Lisibilité et maintenance du code :

Il faut **absolument** privilégier l'utilisation de constantes littérales centralisées en début de code plutôt que des constantes numériques insignifiantes éparpillées dans tout le code ! Exemple :

```
#define LONGUEUR        4.0
#define LARGEUR         2.0
#define HAUTEUR         1.0
#define ROUE_LARGEUR    0.2
#define ROUE_DIAM       0.5
#define BRAS_LONGUEUR   4.0
#define BRAS_DIAM       0.1
#define T_PHARE         0.2
etc …
```

Il faut ensuite utiliser systématiquement ces constantes littérales dans votre code, par exemple :

```
ref_ptr<Geode>    gdeCaisse(new Geode);
ref_ptr<Box>      gdeCaisseShape(new Box( Vec3f(),LONGUEUR,LARGEUR,HAUTEUR));
ref_ptr<ShapeDrawable>  gdeCaisseDrawable(new ShapeDrawable(gdeCaisseShape.get()));
gdeCaisse->addDrawable(gdeCaisseDrawable.get());
grpScene->addChild(gdeCaisse);
```

Attention :

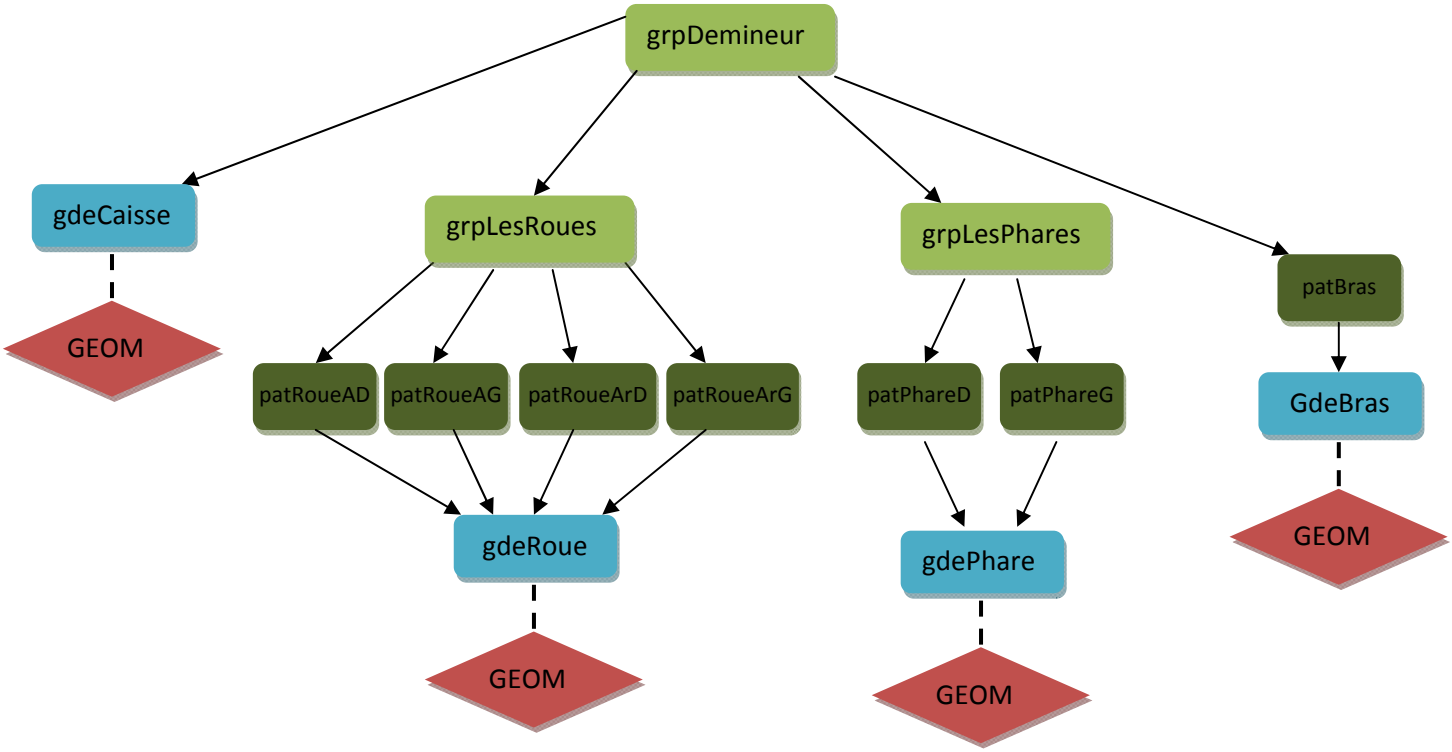
*- Pour les roues, seule **une** instance de Geode (et du Drawable associé) sera créée : gdeRoue. Ce même geode sera utilisé (et partagé) plusieurs fois pour toutes les roues.*

- Les shape Box et Cylindre ne peuvent être ajoutés directement au graphe de scène ! Leur géométrie est récupérée pour être associé à un geode (voir exemple ci-dessus)

2.2.3. Graphe de scène : Structurer la scène avec les transformatio

L'Objet Démineur est composé d'une caisse, des 4 roues, du bras et de deux phares, pour l'instant tous positionnées par défaut en (0,0,0).
L'objectif est maintenant de les positionner au bon endroit en utilisant les groupes de transformation (classes dérivées de `osg::Transform`) : un nœud qui va appliquer une transformation à tous ses fils. *(voir extrait du code plus bas)*

➔ Modifier la fonction `creerDemineur()` qui retournera un graphe où seront correctement positionnées et orientées les différentes composantes (utilisation de nœuds `PositionAttitudeTransform`).



Conseils de nomenclature: préfixe des noms d' objets composant le graphe de scène.

Nom des objets de classe Transform:	**trsf**NomDuNoeud
Nom des objets de classe PositionAttitudeTransform (ou classes dérivées) :	**pat**NomDuNoeud
Nom des objets de classe Group :	**grp**NomDuNoeud
Nom des objets de classe Geode(ou classes dérivées) :	**gde**NomDuNoeud
Nom des objets de classe Drawable (ou classes dérivées) :	**drw**NomObjet
Nom des objets de classe Node :	**node**NomDuNoeud

(voir également les conseils de nomenclature précisés dans le cours)

➔ Exemple pour placer une roue :

```
ref_ptr<PositionAttitudeTransform> patRoueAvD=new PositionAttitudeTransform());
// position du cylindre / Roue  (utiliser les constantes litterales ! )
patRoueAvD->setPosition(Vec3d(      - (LONGUEUR/2-ROUE_DIAM*1.5),
                                    + (LARGEUR/2+ ROUE_LARGEUR),
                                    - HAUTEUR/2));
// orientation du cylindre / Roue => rotation autour d'un axe
Quat  roueAttitude(DegreesToRadians(90.0f) , osg::Vec3f (1,0,0));
patRoueAvD->setAttitude(roueAttitude);
// associer le géode au nœud
// le même gdeRoue (et même drawable)est associé aux 4 roues
patRoueAvD->addChild(gdeRoue); ...
patRoueAvG->addChild(gdeRoue); ...  // le noeud gdeRoue est instancié une fois
patRoueArD->addChild(gdeRoue); ...   // mais affilié à 4 noeuds de transformation
patRoueArG->addChild(gdeRoue); ...
```

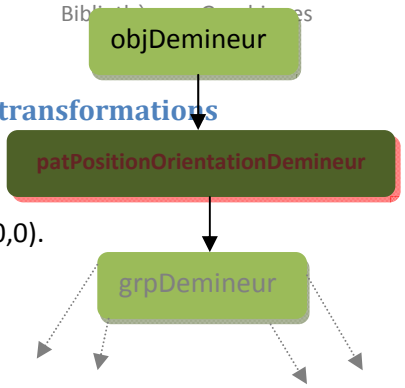
➔ Pour positionner le bras plus facilement, il faut utiliser la méthode `SetPivotPoint` (doc) pour repositionner le pivot du cylindre à son extrémité (par défaut au centre de l'objet).

```
...
patBras->setPivotPoint(Vec3d(0,0,- BRAS_LONGUEUR /2));
patBras->setPosition(Vec3d(0,0,HAUTEUR / 2));
Quat  brasAttitude(BRAS_ANGLE_INITIAL , osg::Vec3f (0,1,0));
...
```

➔ idem pour positionner les phares :

```
ref_ptr<GeodePyramide>        gdePhare;
gdePhare=new GeodePyramide(PHARE_DIM,BLANC,ROUGE,BLEU,BLANC);
      // Phare Droit
ref_ptr<PositionAttitudeTransform> patPhareD(new PositionAttitudeTransform());
patPhareD->setPivotPoint(Vec3d(0,0,- PHARE_DIM /2));
patPhareD->setPosition(Vec3d(-  (LONGUEUR/2-R_DIAM),  (LARGEUR/2- PHARE_DIM),  HAUTEUR/2  ));
Quat  phareAttitude(DegreesToRadians(-90.0f) , osg::Vec3f (0,0,1));
patPhareD->setAttitude(phareAttitude);
patPhareD->addChild(gdePhare);//➔le même drawable esty associé aux 2 phares
      // Phare Gauche
ref_ptr<PositionAttitudeTransform> patPhareG(new PositionAttitudeTransform());
patPhareG->setPivotPoint(Vec3d(0,0,- PHARE_DIM /2)) ;
patPhareG->setPosition(Vec3d(-  (LONGUEUR/2-R_DIAM), - (LARGEUR/2- PHARE_DIM),  HAUTEUR/2 ));
patPhareG->setAttitude(phareAttitude);
patPhareG->addChild(gdePhare);//➔ le même drawable esty associé aux 2 phares
      // les 2 phares
ref_ptr<Transform>           lesPhares(new Transform());
lesPhares->addChild(patPhareD.get());
lesPhares->addChild(patPhareG.get());
```

2.2.4. Graphe de scène : Exploiter la hiérarchie et combiner les transformations



Par la suite il sera nécessaire de déplacer notre objet composé « démineur ».
Pour l'instant, on souhaite le positionner ailleurs qu'à sa position par défaut (0,0,0).
➔ Ajouter un nœud `PositionAttitudeTransform` en amont de la définition des différents composants.

```
ref_ptr<Group> creerDemineur() {

   ref_ptr<Group> grpObjetDemineur= new Group;

   ref_ptr<PositionAttitudeTransform>          patPositionOrientationDemineur;
   patPositionOrientationDemineur = new PositionAttitudeTransform());

   ref_ptr<Group> grpDemineur= new Group;
   ...
   grpObjetDemineur->addChild(patPositionOrientationDemineur);
   patPositionOrientationDemineur ->addChild(grpDemineur);
   return grpObjetDemineur.get();
}
```

➔ Proposer une fonction acceptant un argument position permettant de placer l'objet.

```
void déplacer(ref_ptr<Group> monDemineur, Vec3d vectDeplcmt) {
      PositionAttitudeTransform * patDemineur = monDemineur->getChild(0));
      patDemineur->setPosition(vectDeplcmt);
}
...
int main() {
      ref_ptr<Group>    grpRoot (new Group);
      ref_ptr<Group>    monDemineur = creerDemineur();
      grpRoot ->addChild(monDemineur);
      déplacer(monDemineur, Vec3d(4,1,1));
...}
```

Ici, une erreur de compilation à la première ligne de la fonction déplacer !
Expliquez et corrigez. (en forçant le type)
La correction permet d'exécuter mais ce n'est pas satisfaisant : la fonction ne marchera plus dès que le graphe de scène sera modifié, par exemple un nœud intermédiaire entre la racine et son fils ! Pour éviter ce problème et aussi pour avoir un code plus lisible, plus réutilisable, plus « maintenable », la programmation doit être plus OBJET …

2.2.5. … Les bienfaits de la programmation objet

Nous allons créer une classe Démineur et y regrouper la construction de l'objet et les traitements:

- Un constructeur qui créera le graphe de scène. Au moment de la création, les éléments clés du graphe seront référencées dans des données membre de la classe pour être exploitées plus simplement et de façon fiable plus tard (ici : `_patDemineur`, `_racineGDS`). Ces éléments clé seront les nœuds de transformations. Pour l'instant, stocker le nœud de transformation qui permet de déplacer tout l'objet.
- une méthode `déplacer`
- un accesseur `getGDS`

⇨ Demineur.h

```
class Demineur : public osg::Referenced  {
private :
      ref_ptr<PositionAttitudeTransform> _patDemineur;
      ref_ptr<Group>                     _grpRacineGdsDemineur;

public :
                        Demineur ();      // constructeur qui crée le GDS
      ref_ptr<Group>    getGDS();         // retourne la racine du GDS
      void              déplacer(Vec3d vectDeplcmt) ;
};
```

⇨ Demineur.cpp

```
Demineur::Demineur (){
      // stocker dans une donnée membre privée la racine du graphe
      _racineGDS= new Group;

      // à faire : regrouper les lignes de création du GdS du démineur
}

ref_ptr<Group>    Demineur::getGDS() {
      return _grpRacineGdsDemineur;
}

void Demineur::déplacer(Vec3d vectDeplcmt) {
      _patDemineur->setPosition(vectDeplcmt);
}
```

⇨ main.cpp

```
…
ref_ptr<Group>            grpRoot (new Group);

ref_ptr<Demineur> monOBJ_Demineur = new Demineur();
grpRoot ->addChild(monOBJ_Demineur->getGDS());
…
```

3. Immersion (matériau et texture)

Les images utiles image_roue.jpg, Tex_Pierre.jpg et Tex_Sable.jpg sont sur l'ENT. Ces fichier devraient être placés dans le dossier du projet visual (donc pas dans le dossier debug ni release, ni celui de la solution).

3.1. Ajouter un matériau à la caisse, au bras et aux roues

Définir une couleur « émissive » par élément. Vous pouvez vous reporter au cours (partie `StateSet` – Les Attributs) et consulter dans la doc les paramètres de la méthode `setEmission` (doc): quels sont les valeurs possibles du premiers paramètre ? Quel est le type du deuxième paramètre ? Où est déclarée cette constante ?

Par exemple, pour le bras :

```
// affecter un materiau au BRAS
ref_ptr<Material> materiauBras;
materiauBras = new Material;
materiauBras->setEmission(Material::FRONT , BRAS_COULEUR);
gdeBras->getOrCreateStateSet()->setAttribute(materiauBras);
```

Remarque : Pour les roues, le matériau va être associé à l'unique geode `gdeRoue`. Toutes les instances de `gdeRoue` auront donc la même apparence.

3.2. Ajouter des textures

3.2.1. Placer un plan (le sol)

De la même façon que `CreerTriangle()`, ajouter une nouvelle méthode `CreerSol(Vec2f dim, Vec4 couleur)` qui définira une géométrie et l'associera à un geode résultat (définir 4 points et une primitive quadrilatère (`PrimitiveSet::Quads`) à partir de ces points) .

➪ Ajouter le sol et l'objet à votre scène :

```
// créer le sol puis le placer à une altitude 0
ref_ptr<Group>    grpRoot (new Group);
ref_ptr<Geode>    sol = creerSol(COTE_PLAN_SOL, BLANC);
ref_ptr<PositionAttitudeTransform> patSol(new PositionAttitudeTransform());
patSol->setPosition(Vec3d(0,0,0));
patSol->addChild(sol);
grpRoot->addChild(patSol);
// créer le véhicule puis le placer au dessus du sol
ref_ptr<Demineur> monOBJ_Demineur = new Demineur();
grpRoot->addChild(monOBJ_Demineur->getGDS());
monOBJ_Demineur->déplacer(Vec3d(0,0, ROUE_RAYON + HAUTEUR/2));
```

3.2.2. Ajouter une texture au plan

Le code ci-dessous est à ajouter à votre fonction `CreerSol`.

➪ Charger l'image

```
// pour readImageFile, ajouter un #include <osgDB/ReadFile>
Image *image = osgDB::readImageFile("Tex_Pierre.jpg");

if (!image) {
      std::cout << "Couldn't load texture." << std::endl;
      return NULL;
}
```

Remarque : Si la lecture du fichier jpg échoue, c'est soit qu'il n'est pas placé au bon endroit (voir plus haut), soit que le fichier osg_jpegd.dll n'est pas accessible. Dans ce cas, ajoutez son chemin (sous dossier osgPlugins-3.0.1 du dossier bin de la distribution openscenegraph) au PATH , soit définitivement dans la variable d'environnement(en dehors de l'IUT) soit au lancement de l'application dans visual (voir TP1). Le dernière solution est aussi de copie la dll dans votre dossier debug ☹

⇨ Affecter des coordonnées de texture à la géométrie :

```
osg::Vec2Array* coordonneeTexture = new osg::Vec2Array(4);
(*coordonneeTexture)[0].set(1.0f, 0.0f);
(*coordonneeTexture)[1].set(1.0f, 1.0f);
(*coordonneeTexture)[2].set(0.0f, 1.0f);
(*coordonneeTexture)[3].set(0.0f, 0.0f);
Votre_géométrie->setTexCoordArray(0, coordonneeTexture);
// pour récupérer une géométrie : geode->getDrawable(0)->asGeometry()
```

⇨ Affecter la texture à la géométrie avec un stateSet :

```
Texture2D *texture = new Texture2D;
texture->setDataVariance(Object::DYNAMIC);
texture->setWrap(Texture::WRAP_S, Texture::CLAMP);
//texture->setWrap(Texture::WRAP_S, Texture::REPEAT);
texture->setImage(image);
votre_geode_sol->getOrCreateStateSet()->
            setTextureAttributeAndModes(0, texture, StateAttribute::ON);
```

Pour info :

A voir l'exemple `osgtexture2D` *(exécution et source).*

A voir l'exemple `osgmultitexture` *(exécution et source)*

A voir également, possibilité de multitexturing (un tableau de coordonnées de texture par texture) : `setTexCoordArray(num_tex,tabCoordoTex)` *et* `setTextureAttributeAndModes (num_tex, Texture2D, StateAttribute::ON);`

A voir : la génération de coordonnées de textures avec osg ::TexGen. (ci-dessous)

3.2.3. Appliquer une texture aux roues

⇨ *Utiliser* `osg ::TexGen` *pour la génération automatique de coordonnées de textures :*

```
// chargement de l'image imageroue (voir code précédent)
Texture2D *textureRoue = new Texture2D;
textureRoue->setImage(imageroue);
// génération des coordonnées de texture
TexGen* texgen = new TexGen;
texgen->setMode(TexGen::OBJECT_LINEAR);
texgen->setPlane(TexGen::R, osg::Plane(0.0f,-1.0f,0.0f,0.0f));
// le StateSet associée à la texture
osg::StateSet* stateset = new osg::StateSet;
stateset->setTextureMode(0,GL_TEXTURE_GEN_R,osg::StateAttribute::ON);
stateset->setTextureAttribute(0,texgen);
stateset->setTextureAttributeAndModes(0,textureRoue,osg::StateAttribute::ON);
//associer la texture à l'objet
gdeRoue->setStateSet(stateset);
```