Affichage d’une scène dans la plateforme

Fabrice Lamarche
ESIR / Université de Rennes 1

Table des matières

Ce document a pour objectif de vous décrire les différentes fonctionnalités mise à votre disposition pour effectuer le rendu de vos scènes 3D avec la plateforme d'animation / rendu qui vous est fournie dans le cadre de vos travaux pratiques.

1. Le graphe de scène

Les nœuds du graphe de scène sont décrits dans l'espace de nommage *SceneGraph*. La classe mère des nœuds du graphe de scène est *NodeInterface*. Cette classe abstraite définit une unique méthode *void draw()* qui doit être appelée pour effectuer le rendu du nœud du graphe de scène.

1.1. Gestion des transformations

Les nœuds de transformations du graphe de scène appliquent leur transformation sur tous leurs nœuds fils. Ces nœuds héritent de la classe **Transform**, elle-même héritant de la classe **Group** (regroupement de plusieurs nœuds). Au sein du graphe de scène, les transformations sont gérées par les nœuds suivants :

- **SceneGraph::EulerRotation** : une rotation définie par les angles d'Euler (en radians).
- **SceneGraph::Rotate** : un rotation définie par un axe et un angle (en radians).
- **SceneGraph::Scale** : application de facteurs d'echelle suivant les axes X, Y et Z.
- **SceneGraph::Translate** : application d'une translation.

Via l'héritage du nœud *Transform*, chaque nœud de transformation stocke la matrice courante lors de l'affichage. Il n'y a donc pas de limite au nombre de transformations pouvant s'enchainer.

1.2. Les géométries et autres objets de rendu

La plateforme met à votre disposition plusieurs nœuds vous permettant d'effectuer le rendu de géométries.

- **SceneGraph::CoordinateSystem** : utile pour le débogage, cette classe affiche les axes du repère local (X en rouge, Y en vert, Z en bleu).
- **SceneGraph::Cylinder** : cette classe permet d'afficher un cylindre aligné sur l'axe Z.
- **SceneGraph::Sphere** : rendu d'une sphère.
- **SceneGraph::Mesh** : cette classe permet d'afficher une instance de HelperGl::Mesh représentant un mesh graphique avec un matériau associé.
- **SceneGraph::MeshVBO_v2** : cette classe permet d'afficher une instance de HelperGl::Mesh représentant un mesh graphique avec un matériau associé. Sa particularité, par rapport à la classe Mesh, est d'optimiser le rendu en utilisant des Vertex Buffer Objects ; les performances sont très nettement meilleures.
- **SceneGraph::PointRenderer** : rendu de points. Les points sont caractérisés par une position et une couleur qui peuvent éventuellement changer dynamiquement.
  SceneGraph::ParticleSystemNode : une classe permettant d'effectuer le rendu d'un système à base de particules (Animation::ParticleSystem).
- **SceneGraph::Patch** : un patch dynamique permettant, par exemple, d'effectuer un rendu de tissu. Dans ce patch, les positions des sommets peuvent être changées dynamiquement. Le système recalculera les normales aux faces de manière à ce que le rendu soit correct.

La plupart de ces nœuds de géométrie acceptent un paramètre décrivant le matériau (classe **HelperGl::Material**) utilisé lors du rendu. La configuration du matériau est décrite en section 2.2.

2. Les autres composants du rendu

2.1. Gestion des lumières

La plateforme met à votre disposition la classe **HelperGl::LightServer** pour gérer les lumières. Il est nécessaire d'utiliser cette classe pour la gestion des lumières si vous souhaitez que tout se passe bien du point de vue du rendu. Ce serveur est capable de gérer 8 lumières allumées en simultanée (la contrainte minimale imposée par les anciennes version d'OpenGL). La configuration que ce serveur effectue sur les lumières est compatible avec les affichages via les shaders standards OpenGL (flat et Gouraud) et le shader de Phong qui vous est fourni par la plateforme.

Vous ne pouvez créer une instance de la classe **HelperGl::LightServer** car une seule et unique instance existe dans vote application. Cette instance peut être récupérée en appelant la méthode de classe *HelperGl::LightServer * getSingleton().* KLes lumières sont représentées par une classe interne nommée *HelperGl::LightServer::Light*. Les instances de cette classe peuvent être créées via la méthode *createLight* et détruites via la méthode *destroyLight* de la classe *LightServer*. Chaque lumière créée possède une position, une couleur ambiante, spéculaire et diffuse. D'autre part, chaque lumière peut être activée ou désactivée à volonté. Référez-vous à la documentation de cette classe pour plus d'information.

Les lumières ne sont pas représentées dans le graphe de scène et ne subissent donc aucune des transformations appliquées aux objets ou par la caméra.

2.2. Les matériaux

La plateforme met à votre disposition trois classes principales liées à la gestion des matériaux : la classe **HelperGl::TextureServer**, la classe **HelperGl::PhongShader** et la classe **HelperGl::Material**.

La classe **HelperGl::TextureServer** vous permet de charger des textures à partir de fichiers *jpg* ou *png*. Seule une instance de cette classe existe pour votre application et cette instance peut être récupérée via la méthode de classe *HelperGl::TextureServer * getSingleton()*. Cette classe possède une unique méthode GLuint loadTexture(std::string const & filename) permettant de charger une texture en la référençant dans OpenGL. Cette méthode retourne l'identifiant OpenGL de la texture.

La classe **HelperGl::PhongShader** représente un shader de Phong qui vous est fourni. Seule une instance de cette classe existe pour votre application et cette instance peut être récupérée via la méthode de classe *HelperGl::PhongShader * getSingleton()*. Attention, ce shader suppose que votre géométrie possède des normales aux sommets lors de l'affichage. Pour activer ce shader, il faut appeler la méthode suivante : *void enable (unsigned char lightMask, bool usetexture)*. Le paramètre *lightMask* est un octet pour lequel un bit à 1 signale que la lumière d'indice correspondant est allumée et doit être prise en compte. Le paramètre *useTexture* signale que le shader doit plaquer une texture (si vrai) ou n'utilise pas de texture (si faux). Pour désactiver ce shader, la méthode *void disable()* doit être appelée.

La classe **HelperGl::Material** vous permet de décrire un matériau. Ce matériau possède plusieurs propriétés telles que les couleurs spéculaire, diffuse et ambiante ainsi qu'une éventuelle texture.

Référez-vous à la documentation pour plus de détails. Pour activer un matériau, vous disposer de la fonction *HelperGl::Draw::draw(const Material &)*.

2.3. La caméra

Une classe **HelperGL::Camera** vous est fournie. Il s'agit d'une caméra à la première personne se déplaçant suivant les axes de visée, gauche et haut. Cette classe n'est pas un nœud du graphe de scène. Afin d'appliquer les transformations de la caméra à la scène, il faut effectuer, préalablement au rendu de cette dernière, l'appel suivant :

GL::loadMatrix(camera.getInverseTransform()) ;

La fonction GL::loadMatrix est définie dans le fichier « GL/compatibility.h ». L'appel ci-dessus remplace la matrice courante d'OpenGL par l'inverse de la matrice de transformation de la caméra. Cela permet de repasser les coordonnées fournies dans le repère global dans le repère local de la caméra.

3. Les géométries

3.1. La classe Mesh

La classe **HelperGl::Mesh** représente une géométrie à facettes triangulaires. Cette classe met à disposition deux méthodes permettant de calculer les normales aux faces, ainsi qu'aux sommets. Si vous souhaitez utiliser une géométrie avec un shader de type Phong ou Gouraud, il faut, une fois la géométrie définie, effectuer les appels suivants :

mesh->computeFacesNormals() ; // Calcule les normales aux faces
mesh->computeVertexNormal() ; // Calcule les normales aux sommets

Pour afficher un objet de ce type, vous diposez de trois méthodes.

- Utiliser la fonction *HelperGl::Draw::draw(const Mesh &)*. Cette fonction affichera votre géométrie dans le contexte de rendu OpenGl. Cet affichage est lent, dans le sens où il fait appel au primitives OpenGl de type glNormal, glVertex etc…
- Utiliser le nœud de graphe de scène **SceneGraph::Mesh**. Même remarque que pour la fonction précédente, l'affichage est lent.
- Utiliser le nœud de graphe de scène **SceneGraph::MeshVBO_v2**. Ce nœud utilise les vertex buffer objects pour réaliser l'affichage. Cela représente la solution la plus performante.

Dans tous les cas, l'instance de **HelperGl::Mesh** est porteuse d'un matériau qui sera utilisé pour l'affichage.

3.2. Chargement d'un fichier 3DS

La plateforme vous permet de charger des fichiers au format 3DS. Pour ce faire, la classe **HelperGl::Loader3ds** est mise à disposition. Le constructeur de cette classe prend en paramètre le chemin du fichier 3ds à charger ainsi que le chemin du répertoire contenant les textures. Dès lors qu'une instance est créée, elle charge le fichier fourni.

Cette classe met à disposition la méthode *const std::vector<HelperGl::Mesh *> & getMeshes() const* vous permettant de récupérer l'ensemble des géométries chargées. Il est à noter que les matériaux sont aussi gérés et attribués directement aux objets les utilisant. La destruction d'une instance de Loader n'implique pas la destruction des géométries et matériaux associés. Cela est de la responsabilité de l'utilisateur.