**M1 Informatique - IG3D - Introduction à openGL Documentation**

**Auteur**

Mathias Paulin [Mathi.nosp@m.as.P.nosp@m.aulin.nosp@m.@iri.nosp@m.t.fr](http://www.irit.fr/~Mathias.Paulin/M1Info/IG3D/TP/doc/html/index.html)

**Introduction**

Cette documentation correspond à l'environnement de développement d'un logiciel d'informatique graphique 3D utilisé pour les travaux pratiques de l'UE "Introduction à l'Informatique Graphique 3D" en Master 1 informatique de l'université Paul Sabatier, Toulouse.

**Installation**

Télécharger l'archive [m1-iig3d.tgz](http://www.irit.fr/~Mathias.Paulin/M1Info/IG3D/TP/m1-iig3d.tgz) et la décompresser dans un répertoire spécifique. Après décompression : les répertoires suivants seront générés :

* src : Répertoire contenant les source C++ du logiciel. Certains de ces sources devront être complétés et modifiés pour la réalisation des travaux pratiques
* shaders : Répertoire contenant les sources GLSL des shader qui seront utilisés et modifiés pendant les TPs
* data : Répertoire contenant des fichier 3D exemples
* doc : Répertoire contenant cette documentation

**Dépendances**

Le code fourni, écrit en C++ n'a été testé que sous Linux et mac OS X et nécessite les environnement de développement suivant pour être exploité :

* Un compilateur C++
* L'environement de compilation cmake (version 2.6 au moins)
* L'environement de développement et d'éxécution OpenGL supportant OpenGL 3.2 - core. L'installation de l'environnement de développement et d'éxécution OpenGL 3.2 - core n'est pas décrite ici et nécessite la disponibilité d'une carte graphique récente (dite DirectX 10), l'instalation d'un driver adapté et des bibliothèques de développement associée. Se référer au site [opengl.org](http://www.opengl.org/) pour toute documentation concernant openGL.
* L'environnement de développement QT 4.7 au moins avec, au moins, les modules QTCore, QTGui et QTOpenGL.
* Bien que non nécessaire pour la réalisation des travaux pratiques, nous invitons les étudiants à installer le logiciel de création 3D Blender pour créer et éditer les objets et scènes qui seront visualisées par le logiciel développé en TP.

**Compilation**

Le système de compilation utilisé par ce code est le système [cmake](http://www.cmake.org/). L'ensemble des sources de ce logiciel d'infographie 3D est composé de fichiers C++ définissant l'architecture de base et placés dans le répertoire src . Le répertoire glm contient une bibliothèque de gestion de vecteurs et de matrices en C++, développée pour être utilisée avec OpenGL puisque suivant les mêmes spécification que le langage GLSL. Voir le site [glm.g-truc.net](http://glm.g-truc.net/) pour tout renseignements sur cette bibliothèque.

Le répertoire fileloaders contient le code source C++ nécessaire au chargement de fichiers de données pour le logiciel. Le code de ce répertoire ne sera pas édité ou modifié pour les TPs mais uniquement utilisé et étendu.

Le répertoire rendersystem contient le code C++ principal de l'aplication. Ce code C++ est composé d'une partie IHM dévelopée avec QT (qui ne fera pas l'objet de modification pour les TPs) et d'une partie renderer correspondant au code source C++ de l'application qui sera développée pendant les TPs. Seuls les fichiers [**renderer.h**](http://www.irit.fr/~Mathias.Paulin/M1Info/IG3D/TP/doc/html/a00070.html) et [**renderer.cpp**](http://www.irit.fr/~Mathias.Paulin/M1Info/IG3D/TP/doc/html/a00069.html) devront être modifiés pendant les TPs et remis à l'enseignant pour évaluation des travaux.

Pour compiler le programme, nous utiliserons la possibilité offerte par cmake de compiler en dehors de l'arborescence des sources. Les étudiants sont libres d'utiliser l'EDI qu'ils souhaitent (QTCreator, kdevelop, ...) mais aucune assistance ne sera apportée sur l'utilisation d'un EDI quelconque. Voici la marche à suivre pour compiler votre programme :

**Première compilation**

* Créer (la première fois) un répertoire build dans le répertoire d'extraction de l'archive [minimalrenderer.tgz](http://www.irit.fr/~Mathias.Paulin/M1Info/IG3D/TP/minimalrenderer.tgz) (au même niveau que les répertoires src, shaders, data et doc)
* Dans un terminal, se positionner dans ce répertoire et taper la commande de génération des makefiles : **cmake ../main**
* Compiler le programme par la commande **make**

**Toutes les autres compilations**

* Compiler le programme par la commande **make** lancée dans le répertoire build

**Exécution**

* Executer le programme en tapant la commande **./minimalRenderer** dans le répertoire bin (ou ../bin/minimalRenderer depuis le répertoire build)

**Scéance 1 : mise en place d'un environnement logiciel.**

L'objectif de cette séance et de comprendre et de mettre en place l'architecture générale d'une application de synthèse d'images 3D.

Une telle application est généralement développée selon le motif de conception Modèle-Vue-Contrôleur

* Le modèle est la scène 3D et est représenté par des classes gérant les maillages, les textures, les caméra, ...
* La vue est implantée par un system de rendu construit au dessus des interfaces de programmation d'application de bas niveau (OpenGL ou DirectX).
* Le contrôleur est l'interface entre le système et la vue et gère les interactions avec l'utilisateur.

Dans notre application, le modèle est écrit en C++, la vue en C++/OpenGL et le contrôleur utilise l'API Qt.

Nous nous fondons sur la spécification [OpenGL 3.2 core](http://www.opengl.org/registry/doc/glspec32.core.20091207.pdf) comme API de bas niveau. Il est recommandé de consulter cette spécification ainsi que le [manuel de référence](http://www.opengl.org/sdk/docs/man3/) de cette API pour bien comprendre les concepts manipulés.

**1 - Représentation de la géométrie pour un affichage par OpenGL.**

Fichier [**glmesh.cpp**](http://www.irit.fr/~Mathias.Paulin/M1Info/IG3D/TP/doc/html/a00049.html), classe [**rendersystem::GlMesh**](http://www.irit.fr/~Mathias.Paulin/M1Info/IG3D/TP/doc/html/a00027.html).

Les données géométriques affichées par notre application sont des maillages, définis dans le code source ou chargés depuis des fichiers externes, qui doivent être préparés pour un affichage par OpenGL.

La première étape de cette séance de travaux pratiques concerne la programmation de la classe [**rendersystem::GlMesh**](http://www.irit.fr/~Mathias.Paulin/M1Info/IG3D/TP/doc/html/a00027.html), qui est chargée de faire ces opérations. Cette classe est déclarée (et partielement vide) dans le fichier [**glmesh.cpp**](http://www.irit.fr/~Mathias.Paulin/M1Info/IG3D/TP/doc/html/a00049.html). [**rendersystem::GlMesh**](http://www.irit.fr/~Mathias.Paulin/M1Info/IG3D/TP/doc/html/a00027.html) hérite de la classe [**loaders::Mesh**](http://www.irit.fr/~Mathias.Paulin/M1Info/IG3D/TP/doc/html/a00011.html) et rajoute à la représentation des maillages les fonctionalités nécessaires pour un affichage par OpenGL.

Pour pouvoir être affiché en OpenGL, un maillage doit être représenté sous la forme d'un **Vertex Array Object**, objet abstrait OpenGL permettant de décrire l'organisation des données du maillage en mémoire et comment y accéder. Lorsque notre maillage est ainsi représenté, son tracé peut alors être réalisé simplement en demandat à OpenGL d'utiliser l'objet et d'en dessiner les éléments.

Deux méthodes doivent être programmées pour implanter la classe [**rendersystem::GlMesh**](http://www.irit.fr/~Mathias.Paulin/M1Info/IG3D/TP/doc/html/a00027.html) :

* [**rendersystem::GlMesh::compileGL()**](http://www.irit.fr/~Mathias.Paulin/M1Info/IG3D/TP/doc/html/a00027.html#a45fa26bf6414428cf68afa8d5a35d1fe)

Le rôle de cette fonction est de créer les objets OpenGL et de décrire l'organisation des données dans ces objets pour en permettre le tracé. Nous utilisons dans notre application les notions de VertexArrayObject et de VertexBufferObject pour représenter nos maillages. Les explications concernant l'utilisation de ces objets vous seront données en scéance. La spécification de ces concepts est sur le site [opengl.org](http://www.opengl.org/). Se reporter aux commentaires de la fonction dans le fichier [**glmesh.cpp**](http://www.irit.fr/~Mathias.Paulin/M1Info/IG3D/TP/doc/html/a00049.html) ou dans la documentation de la fonction pour avoir des indications sur la marche à suivre.

* [**rendersystem::GlMesh::drawGL()**](http://www.irit.fr/~Mathias.Paulin/M1Info/IG3D/TP/doc/html/a00027.html#a81552b64abf3926a0e40751c85d06b3e)

Le rôle de cette fonction est de donner les ordres OpenGL pour le dessin d'un maillage.

**Avant de continuer**, vous devez ecrire le code de ces deux fonctions et compiler votre programme. Se reporter aux commentaires de la fonction dans le fichier [**glmesh.cpp**](http://www.irit.fr/~Mathias.Paulin/M1Info/IG3D/TP/doc/html/a00049.html) ou dans la documentation de la fonction pour avoir des indications sur la marche à suivre.

**2 - Initialisation générale de l'application.**

Fichier [**renderer.cpp**](http://www.irit.fr/~Mathias.Paulin/M1Info/IG3D/TP/doc/html/a00069.html), fonction [**rendersystem::Renderer::initRessources()**](http://www.irit.fr/~Mathias.Paulin/M1Info/IG3D/TP/doc/html/a00034.html#ae8316182754a20571b5294b42994eb00)

Au lancement de l'application, les différents paramètres et états du pipeline de discrétisation OpenGL doivent être définis et initialisés. Ces états vont paramétrer le pipeline ou le contrôleur pour permettre à l'application de réaliser ses tâches. Dans notre cas, les paramètres et états à initialiser concernent

* le mode de rendu OPENGL\_H
* les paramètres de caméra pour la visualisation de la scène

Toute application de synthèse d'images nécessite l'utilisation de resources externes pour fonctionner. Lors de l'initialisation de l'application, les resources dont la durée de vie est celle de l'application doivent être initialisées. Dans le cas de notre applicaiton exemple, développée dans le cadre de ces travaux pratiques, ces resources seront :

* le maillage à visualiser
* les shaders pour les étages Vertex et Fragment
* les paramètres d'apparence de l'objet à visualiser (textures, materiau, ...)

**2.1 - Initialisation des paramètres OpenGL.**

Dans notre application, nous souhaitons visualiser nos modèles sous forme de faces pleines avec élimination des parties cachées. Cette initialisation est réalisée dans la fonction [**rendersystem::Renderer::initRessources()**](http://www.irit.fr/~Mathias.Paulin/M1Info/IG3D/TP/doc/html/a00034.html#ae8316182754a20571b5294b42994eb00) les états OpenGL nécessaires.

**2.2 - Initialisation des paramètres de vue.**

La fonction [**rendersystem::Renderer::initView()**](http://www.irit.fr/~Mathias.Paulin/M1Info/IG3D/TP/doc/html/a00034.html#aafcf982312922a03e9bb1f072c10b3ca) permet de définir les paramètres par défaut pour la visualisation d'un maillage. La matrice de vue rendersystem::Renderer::g\_viewMatrix est initialisée dans la fonction [**rendersystem::Renderer::initView()**](http://www.irit.fr/~Mathias.Paulin/M1Info/IG3D/TP/doc/html/a00034.html#aafcf982312922a03e9bb1f072c10b3ca) et peut être changée si vous souhaitez un autre point de vue par défaut.

**2.3 - Chargement et compilation des données géométriques de l'application.**

Cet exercice n'est pas à faire pour ce TP

En utilisant la classe [**loaders::obj\_mtl::ObjLoader**](http://www.irit.fr/~Mathias.Paulin/M1Info/IG3D/TP/doc/html/a00016.html) de chargement de fichier Alias/Wavefront OBJ, charger l'objet "../data/camel.obj" et stockez ses différentes composantes dans un vecteur (std::vector<loaders::Mesh\*>). Transformez ces composantes en vecteur de primitives affichables en utilisant votre classe rendersystem::MyGLMesh.

**2.4 - Chargement et compilation des shaders.**

Cet exercice n'est pas à faire pour ce TP

La spécification OpenGL 3.2 core nécessite la définition de shaders pour la gestion des différentes données transitant dans le pipeline de discrétisation. Dans notre application, les shaders sont définis dans des fichiers sources glsl qui doivent être chargés, compilés et liés avant de pouvoir être utilisés pour le rendu. L'objectif de la fonction [**rendersystem::Renderer::initShaders()**](http://www.irit.fr/~Mathias.Paulin/M1Info/IG3D/TP/doc/html/a00034.html#a9f0fe0c105c553fe28c933db8169ef60) est de préparer l'application afin qu'elle utilise par défaut les shaders "../shaders/vertexdefault.glsl" et "../shaders/fragmentdefault.glsl". Il est à noter qu'un shader doit être vue comme une unité de compilation qui est ensuite intégrée dans un programme.

Les opérations devant être faites pour pouvoir utiliser des shaders OpenGL dans notre application sont les suivantes :

1. Vertex shader :
   1. Charger le source depuis le fichier "../shaders/vertexdefault.glsl"
   2. Créer un objet OpenGL VERTEX\_SHADER et y associer le source
   3. Compiler le shader
   4. Vérifier les erreurs de compilation
2. Fragment shader :
   1. Charger le source depuis le fichier "../shaders/fragmentdefault.glsl"
   2. Créer un objet OpenGL FRAGMENT\_SHADER et y associer le source
   3. Compiler le shader
   4. Vérifier les erreurs de compilation
3. Programme :
   1. Créer un programme OpenGL et y associer les shaders
   2. Fixer les positions des attributs géométriques en accord avec la classe rendersystem::MyGLMesh et les convention d'écriture du source GLSL
      * inPosition –> index 0
      * inNormal –> index 1
      * inTexCoord –> index 2.
   3. Lier le programme
   4. Vérifier les erreurs d'édition de lien

**3 - Fonction générale de rendu.**

La fonction [**rendersystem::Renderer::render()**](http://www.irit.fr/~Mathias.Paulin/M1Info/IG3D/TP/doc/html/a00034.html#abe1317bc098889c2e1accbc2b0f76a3f) est la fonction principale de calcul d'une image de synthèse. Appelée à chaque rafraichissement de l'image son rôle est de séquencer l'ensemble des ordres de dessin de l'API de bas niveau. Dans notre application simple, cette fonction à le rôle suivant.

1. Préparer de l'image pour le rendu :
   * Effacer les buffers de destination
2. Construire les matrices de vue et de projection :
   * Définir une matrice de projection perspective
   * Définir la matrice de vue
   * Calculer la matrice de transformation des normales
   * Calculer la matrice MVP de passage du repère objet au repère image
3. Activer et paramétrer les shader par defaut :
   * Activer le programme par défaut
   * Positionner les paramètres indépendants des objets : les matrices dans notre cas
4. Dessiner les objets de la scène :
   * Dessiner tous les objets de la collection [**rendersystem::Renderer::entities\_**](http://www.irit.fr/~Mathias.Paulin/M1Info/IG3D/TP/doc/html/a00034.html#a80c58ef650c514aa7f2fd950f4107f72)

**Scéance 2 : navigation et exploration dans la scène.**

La navigation et l'exploration d'une sc!ène 3D pass par la gestion d'une métaphore abstraite de Caméra.

Dans notre application, cette notion est simplement représentée par la matrice rendersystem::Renderer::g\_viewMatrix, définissant la transformation permettant de passer du repère scène au repère camera.

La navigation et l'exploration d'une scène correspond à la modification de cette matrice en fonction des interactions utilisateur. Ces interactions sont capturées par l'interface QT et transmises à la fonction [**rendersystem::Renderer::handleMouseEvent()**](http://www.irit.fr/~Mathias.Paulin/M1Info/IG3D/TP/doc/html/a00034.html#a98bf2f211e3840eefca37ca1c16e50cd) que vous allez compléter afin de permettre une exploration simple d'un objet.

Trois type de mouvements sont gérés par notre système initial :

1. Mouvement de type "Observe" (bouton gauche de la souris). Ce mouvement consiste à effectuer une rotation centrée de la scène autour des axes X et Y de la caméra
2. Mouvement de type "Pan" (bouton droit de la souris). Ce mouvement consiste à effectuer une translation de la scène dans le plain (X, Y) de la caméra
3. Mouvement de type "Zoom" (bouton central de la souris). Ce mouvement consiste à réaliser une translation de la scène selon l'axe Z de la caméra

La matrice rendersystem::Renderer::g\_viewMatrix a été initialisée dans l'exercice [**2.2 - Initialisation des paramètres de vue.**](http://www.irit.fr/~Mathias.Paulin/M1Info/IG3D/TP/doc/html/a00001.html#viewSettings) et défini les proriétés de la caméra. Afin de pouvoir réaliser les transformation, il faut récupérer les axes X, Y et Z du repère de cette caméra. Les rotations et les translations seront réalisées en fonction des valeurs des variables dx et dy contenant le déplacement normalisé de la souris dans le repère de l'image.

Sceance 3 : voir fichier copié

**Devoir 1 : construction d'une sphère géodésique.**

L'objectif de ce devoir est de programmer et d'ajouter à la liste des objets que l'on peut visualiser dans notre application un solide Platonique, l'icosahèdre et d'utiliser ce solide comme élément de base pour la construction d'une sphère géodésique.

Dans ce devoir programmerez 2 classes, Icosahèdre et Geodesique permettant de construire un icosahedre selon la définition de ce solide accessible [ici](http://fr.wikipedia.org/wiki/Icosa%C3%A8dre). La sphère géodésique sera obtenue en divisant récursivement jusqu'à un niveau N les faces de l'icosahèdre et en projettant les sommets sur la sphère.