Rapport Moteur3D

Faculté des Sciences et Techniques

Master M1 ISICG

Moteur 3D Temps réel

Turikumwe Fabrice

Contents

1 Introduction

1.1 Moteur3D et OpenGL

Moteur3D est un composant logiciel permettant de générer des images à partir de données diverses. Dans ce cours on a étudier des principes et fonctionnement d'un moteur3D, le fonctionnement du pipeline graphique et programmation de shaders tout en utilisant c++ et opengl version 4.4.
Le pipeline graphique est la séquence d'étapes suivies par OpenGL lors du rendu d'objets. Le pipeline lancé lorsque on effectue une opération de render. Les opérations de rendre nécessitent la présence d'un objet de tableau de sommets correctement défini et d'un objet de programme lié ou d'un objet de pipeline de programme qui fournit les shaders pour les étapes de pipeline programmables. Le pipeline graphique d'opengl contient plusieurs grandes étapes programmables et non programmables. Les étapes obligatoire sont celle du vertex shader et le fragment shader, les étapes optionnelle sont celle du tesselation et geometry shader.
Le Vertex Shader est l'étape Shader dans le pipeline de rendu qui gère le traitement des sommets. Les vertex shaders sont alimentés en données Vertex Attribute, comme spécifié à partir d'un objet de tableau de sommets.
Un Fragment Shader est l'étape Shader qui traitera un fragment généré par la rastérisation en un ensemble de couleurs et une valeur de profondeur. Donc pour chaque échantillon de pixels couverts par une primitive, un "fragment" est généré.
La tessellation est l'étape de traitement des sommets dans le pipeline de rendu OpenGL où les patchs de données de sommets sont subdivisés en plus petits primitives.

Tout au long du cours, on a étudier et utiliser tout les étapes programmables et comprendre leur fonctionnement et l'avantage qu'ils ont sur le rendu.

1.2 Les TPs

1.2.1 TP1

Le premier TP du cours était simplement une introduction à OpenGl et la compréhension de shaders. L'objectif du tp était l'affichage d'un triangle 2D. Pour l'exécuter, on a besoin de programmer un vertex shader et un fragment shader, et les fournir les données de sommets et de couleur.
Les données de sommets doit être stocker dans un structure de données. Un VBO doit contenir les sommets du triangle qui sont obtenu du structure de données en fonction de sa taille. Un VAO encapsule le VBO et précise l'organisation des attributs. Une programme est créer et on attache les deux shaders. Dans le rendu après avoir nettoyer le frame buffer, on peut projeter le triangle.

1.2.2 TP2

Le deuxieme TP était toujours en 2D avec une introduction d'un EBO et différents couleurs sur l'objet de rendu. L'EBO empêchez le surcharge de données de sommets, qui optimise les performances pour des modelés plus complexes qui contient plusieurs triangles. L'EBO stocker les indices et on décide quels sommets à dessiner. Avec cette technique on a créer un quad.
En plus de ça, on a était introduit au variable uniforme, Ceux-ci agissent comme des paramètres que l'utilisateur d'un programme shader peut transmettre à ce programme. Leurs valeurs sont stockées dans un objet programme. C'est une paramètre trop important qui nous à servi pour les animations et la communication avec les shaders.
Dans ce TP on a présenter le premier interface utilisateur pour modifié les valeurs uniforme et changer la couleur du fond et la luminosité.

1.2.3 TP3

Dans ce TP3, on a progresser dans un univers 3D avec un rendu de cubes. En utilisant les techniques vu dans les derniers tps, surtout avec les EBOs, VAO et VBO. En plus de ça, à l'aide de variable uniforme, on à ajouté des transformation géométriques.
On a été fourni une classe de Caméra pour parcourir dans notre univers. En plus de ça, On a vu les matrices de transformations, View Matrix, Projection Matrix et le Model Matrix qui séparer les transformations proprement.

1.2.4 TP4

On nous a présenté des classes pour représenter des objets 3D décrit par des maillages de triangles. Les classes contenaient des informations de sommets, un tableau d'indices, les données de matériau comme les textures et d'autres données considerable.
De plus on a vu les différents types d'éclairage et leur applications avec le modèle de Phong et le modèle de Blinn-Phong. C'était une aspect trop intéressant pour voir le changement du model Bunny en fonction d'éclairage.

1.2.5 TP5

Dans ce TP, on est allés beaucoup plus loin avec les textures, en utilisation des variables uniformes en fonction sur un type de texture. Après le chargement du palais de Sponza, on a appliqué plusieurs techniques et filtres. Beaucoup de filtres servaient à rendre le rendu plus meilleurs en terme de qualité, comme avec les normals maps, gérer le transparence de quelques matériaux et l'anti-aliasing.

1.2.6 TP6

Le TP6 est plus avancé et moins détaillée que les autres. Les Tps précédents étaient basé sur la technique de Forward render, mais dans celle la on dévait utilisé le deferred rendering qui est une autre type de rendu avancée avec plusieurs cibles de rendu.

La première partie concerne le geometry pass, toute les données de textures sont séparer et stocker dans un G-Buffer puis ils peut être appeler différemment dans le rendu.
La deuxième partie concerne le shading pass, les données de textures du G-Buffer sont envoyés dans une fragment shader qui les utilisera pour calculer l'éclairage. Il faut créer un quad pour que le shader puisse dessiner dessus. Les données de texture sont calculés sont que pour les pixel courant, celle la éviter de calculer toute la scène et faire des optimisations.

1.2.7 Projet

J'aurais du aimer travailler sur le projet parce que il y avais plusieurs méthodes qui m'interessais comme le shadow mapping et l'anti aliasing FXAA mais j'ai pas eu le temps de finir le TP en temps pour y travailler.