Master Informatique, Synthèse d'Images et Conception Graphique

Projet Moteur 3D

Solal COQUERON

Janvier 2023

Contents

Chapter 1

Présentation des TPs

1.1 TP1

Lors de ce TP nous nous sommes familiarisé avec OpenGL afin de comprendre les principes de base pour afficher un triangle simple. Nous avons dans un premier temps créé et compilé nos premiers shaders, nous avons ensuite créé un programme OpenGL en y attachant nos shaders. Il a fallut ensuite initialiser les données correspondant aux sommets du triangle et les injecter dans un VBO. Il faut ensuite créer un VAO et lier le VBO à ce VAO. Enfin dans la fonction render il nous suffit de clear le frame buffer, d'indiquer le programme à utiliser et de lier le VAO au programme, il ne reste plus qu'à dessiner le triangle grâce à la fonction glDrawArrays et c'est bon

1.2 TP2

Pour le deuxième TP on a vu dans un premier temps l'utilisation des EBO et comment ils pouvaient être utilisé pour limiter le nombre de sommets à envoyer au GPU. On a créé un cube et au lieu d'envoyer six sommets formant nos deux triangles on peut envoyer seulement les quatre sommets de notre cube et dans l'EBO il suffira d'indiquer les indices des sommets des triangles. On a ensuite rajouter un VBO s'occupant des couleurs des différents sommets. Enfin on a vu comment mettre en place des variables de contrôle, on a donc animer le cube de gauche à droite en modifiant une variable dans la fonction animate. L'exercice optionnel consistait en l'élaboration d'un cercle à partir de N triangles. Pour ce faire il fallait calculer tous les points qui composerait le cercle et dont on se servirait pour créér le triangle. Pour ça il fallait calculer l'angle en fonction du nombre de triangles et le multiplier en fonction du point que l'on calculait. Le résultat donne une éllipse car on

ne prendre pas en compte la largeur et la hauteur de l'écran.

1.3 TP3

Dans ce TP nous avons effectué nos premières transformations 3D sur un cube. Nous avons dans un premier temps modifier la matrice de transformation du cube afin qu'il effectue une rotation sur lui même. Ensuite nous avons complété la caméra dans laquelle nous avons calculer la View Matrix et Projection Matrix qui permettent respectivement de passer du World Space au View space et du View space au Clip space.

1.4 TP4

Ce TP était consacré au calcul d'éclairage local. Nous avons d'abord pris connaissance de la classe TriangleMeshModel qui nous permet de charger des modèles 3D. Nous avons ensuite implémenté l'éclairage local selon le modèle de phong, nous avons donc calculer les différents éclairages et avons appliqué l'éclairage ambiant, diffus et spéculaire sur le modèle du lapin. Une fois cela fait avec le modèle du lapin nous avons charger un autre modèle afin de se rendre compte que certains triangle n'était pas éclairés. Cela est dû au fait que certaines normales ne sont pas dans le bon sens, il suffit de les retourner lorsqu'elle ne point pas vers la caméra.

1.5 TP5

Pour ce TP nous avons manipuler les textures pour changer l'apparence de nos objets. Certaines textures peuvent être chargés en fonction du matériau de l'objet 3D. Il fallait dans un premier temps charger ces différentes textures, les transmettre aux shaders et les utiliser si elles existent. Un problème de sur-échantillonage des textures discrètes peut entraîner de l'aliasing, il était donc nécessaire de d'appliquer un filtre de magnification pour réduire cette effet. Un autre problème était le sous-échantillonage qui pouvait générer des effets de moiré, dans ce cas la il fallait appliquer un filtre de minification Nous avons ensuite utiliser une Normal map pour perturber les normales des objets afin de donner une illusion de relief aux surfaces planes pour ce faire il est nécessaire de faire les calculs en tangent space. pour finir nous avons vu différentes façon de gérer la transparence.

1.6 TP6

Dans ce TP nous avons mis en place un pipeline de deffered rendering. Dans un premier temps on effectue une geometry pass qui va remplir le G-Buffer aves les différentes données du modèle. Ensuite on effectue une shading pass où l'on va récupérer les données afin d'effectuer les calculs de lumières.

Chapter 2

Présentation du projet

2.1 SSAO

J'ai choisi d'implémenter cette effet afin de rajouter plus de réalisme à la scène. C'est également un effet que l'on rencontre assez souvent et j'étais curieux de voir comment il fonctionnait en détail et comment l'implémenter. Afin de mettre en place cette effet je me suis servi d'un tutoriel de learnopengl ainsi que de celui sur lequel s'inspire learnopengl qui provient du blog de John Chapman.

Pour le SSAO on va calculer un facteur d'occlusion (lumière bloquée) pour chaque pixel. Pour ce faire nous allons pour chaque pixel générer un échantillon de points aléatoires dans un hémisphère orienté le long de la normale à la surface de ce pixel. Pour que chaque point soit plus pertinent on rapproche tous nos points de l'origine de l'hémisphère. Pour réduire le nombre d'échantillons nécessaire on rajoute un peu d'aléatoire en créant une texture 4*4 de rotations aléatoires que l'on va répéter sur tous l'écran. Cela nous évite de devoir stocker une rotation aléatoire pour chaque pixel.

Dans le fragment shader du SSAO on va calculer pour chaque point de l'échantillon si il sera nécessaire d'augmenter le facteur d'occlusion ou non. L'utilisation de la texture aléatoire apporte du bruit au résultat, afin de remédier à ce problème il va falloir effectuer un blur aux résultats obtenus.

Pour le moment le principal problème que j'ai rencontré est que l'occlusion change en fonction des mouvements de la caméra. Cela est sûrement dû à un des calculs qui sont effectués avec des éléments qui ne sont pas dans le même espace mais je n'ai pas encore trouver la solution.

2.2 Tone Mapping

J'ai implémenté de l'ACES tone mapping afin de ne plus avoir d'effets brillant que je pouvais avoir sur les murs. Cela permet également d'avoir de meilleurs couleurs et donc une scène plus agréable.