UNIVERSITÉ
DE MONTPELLIER

Compte rendu de TP1

Moteur de Jeux

Rémi SOULIER
Master 2 IMAGINA
27/09/2017

UNIVERSITÉ DE MONTPELLIER

Je précise qu'il s'agit là de ma première expérience avec OpenGl. N'ayant pas suivi la formation de Master 1 IMAGINA j'ai découverts à travers ce TP cette bibliothèque graphique. J'ai donc rencontré de nombreux problèmes de compréhension qui ont ralenti ma progression du TP. Pour palier à ces derniers j'ai effectué de nombreuses recherches internet pour tenter de trouver un maximum de ressources utiles. Cependant la plupart d'entre m'aurait demandé de m'y pencher un long moment afin de les maîtriser totalement et de pouvoir les utiliser dans mon code.

Question 1

La classe MainWidget représente la fenêtre principale. Elle permet d'initialiser et de gérer les paramètres OpenGL ainsi que les événements.
La classe GeometryEngine quant à elle gère l'objet à afficher en lui même. On y retrouve notamment les coordonnées du cube.

Question 2

La fonction initCubeGeometry() permet d'initialiser les coordonnées du cube et également ceux de la texture. Chaque vertex est associé à une coordonnée sur le fichier texture.
Les indices des vertex sont stockés dans un tableau de telle sorte qu'il puisse être représentés sous forme de triangle strip. Ces deux tableaux sont enfin stockés dans les VBO.

La fonction drawCubeGeometry indique au programme ou trouver les vertex, ou trouver les coordonnées de textures et indique également la manière de rendre notre objet, ici il sera rendu grace a des TRIANGLE_STRIP.

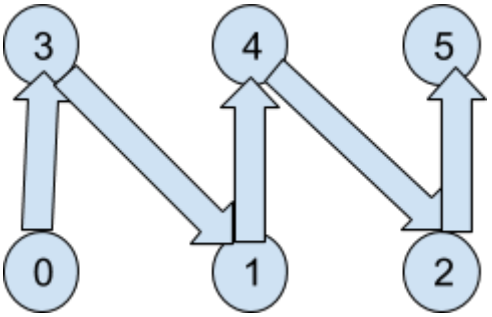
Question 3

Pour mettre le cube à plat, j'ai tout d'abord commencé par la méthode la plus intuitive, c'est à dire modifier directement les coordonées des vertex en s'assurant que z soit à 0 pour chacun.
Le problème de cette méthode c'est qu'elle ne pourrait être mise en oeuvre pour un objet avec un grand nombre de vertex, comme un plan 16*16.

Rémi SOULIER
Master 2 IMAGINA
27/09/2017

Il a donc fallu ensuite mettre en place une méthode générique de construire un plan. Pour les vertex, le principe est simple, ils sont placés dans le tableau de vertex de la façon suivante : [0,0,0][0,1,0][0,2,0]...[0,15,0][1,0,0]...[15,15,0]

Pour le tableau d'indique, qui quant à lui indique l'ordre dans lequel prendre les sommets pour créer des TRIANGLE_STRIP corrects, est plus complexe à construire.
L'ordre à respecter est le suivant :



Cependant, une fois arrivé en fin de ligne, pour indiquer au VBO qu'il faut reprendre au début, il est nécessaire de doubler le dernier indice et de le lier avec le premier de la seconde ligne (ci dessus : 5 → 3).
Malheureusement, je ne suis pas parvenu à une implémentation correcte de ce procédé.

Afin de toujours garder l'objet visible, j'ai modifié les coordonnées de la matrice modèle-vue de telle sorte que l'objet soit centré par rapport à la caméra.

Question 4

Même si mon rendu du plan n'est pas correct j'ai quand même pu modifier la hauteur de certain point du plan et tester que cela fonctionne bien.
Pour cela j'utilise un fonction de randomisation sur un flottant entre 0.0 et 2.0 et j'assigne ensuite cette valeur à la coordonée z de plan. Ainsi le plan obtenu comporte du relief.
Je n'ai pas implémenté la fonctionnalité qui permet de déplacer la caméra.
Cependant si je l'avais fais, j'aurais overrides la fonction keyPressedEvent de Qt (au même titre que mousePressedEvent) et en fonction du bouton pressé, j'aurais translaté le plan pour donner l'illusion que la caméra se déplace au dessus de ce dernier.