Decroos Kevin

Moteurs de Jeux
Compte Rendu de TP1

Question 1 :

MainWidget sert à initialiser OpenGL et gérer les actions d'interfaces avec l'utilisateurs (événements). Il définit ainsi les effets des clics des utilisateurs (du fait de la présence des méthodes Event) et charge l'affichage : images, shaders (du fait de la présence des méthodes initShaders() et initTextures(). Y est définit également la procédure pour afficher les rotations grâce aux matrices dans paintGL(). En outre, resizeGL permet de configurer la perspective de l'utilisateur.

GeometryEngine sert à intégrer la forme cubique de l'objet et ses conséquences sur l'affichage (rotation des faces avec l'image en surface). Cela se voit par la présence d'initCubeGeometry, drawCubeGeometry.

Fshader.glsl permet de configurer les shaders, avec la précision (comme elle est définit en medium). Il inclut ensuite une instruction permettant d'adapter la couleur par rapport à une texture et une coordonnée.

Vshader permet aussi de configurer les shaders, avec la précision. Il donne ensuite les instructions aux shaders pour s'adapter à la position des objets/textures en la calculant et en la transférant à Fshader.

Question 2 :

GeometryEngine::initCubeGeometry() permet d'initialiser les positions des arrêtes du cube, ainsi que de chaque face individuellement. Elle définit ainsi les coordonnées de chaque sommet, ainsi qu'à l'image de référence sera découpée.

En commentant les vecteurs 2d, les faces existent toujours mais deviennent blanches, sans texture. En modifiant les coordonnées du vecteur 2d, la partie de l'image affichée sur la face est différente. En effet, une unique image permet d'afficher toutes les parts du cube, et elle doit être découpée pour pouvoir afficher la bonne partie à mettre sur chaque face, d'où[1] les 0,33 et les 0,5, l'image étant en 3x2 images.

En modifiant les coordonnées des vecteurs 3d, les faces se déplacent bien, accompagnées de leur texture associée. Il s'agit donc bien des coordonnées de la face dans le moteur, ainsi que des positions relatives à sa face pour la texture.

Les faces sont ensuite associées en triangles avec des parties communes, et les données sont chargées.

Question 3 :

Modifier initPlaneGeometry() s'est avéré difficile.

J'ai premièrement essayé d'aplanir le système du cube en dupliquant les sommets. J'ai d'abord procédé en faisant créer des carrés à partir de chaque coin supérieur gauche, en bornant ainsi mes doubles boucles entre 0 et la taille - 1.

Créer les indices correspondants a été la partie la plus difficile. J'ai d'abord essayé de reproduire le système de triangles avec les sauts, avant d'apprendre que ce système était généralement peu pratique en pratique, quand les surfaces deviennent plus complexes. Le système aux triangles simples étant donc plus direct et plus pertinent, je suis passé à ce système. A partir de ce moment, j'ai longtemps cherché d'où pouvait provenir les erreurs à partir de mon raisonnement consistant à générer les triangles à partir du coin supérieur gauche de chaque carré. Il y avait ainsi plusieurs problèmes, dus essentiellement à l'allocation mémoire, d'abord aux divers endroits de la classe en général (fin de initPlaneGeometry, fin de la classe), et j'ai cherché à calculer précisément le nombre de triangles effectués pour l'allocation avec des raisonnements plus que douteux, avant de réaliser que la solution était extrêmement simple (juste le faire 6 points par 6 points).

Cette méthode s'est avérée peu efficace au passage aux étapes d'après, et a donc été modifiée afin de placer les points un par un. Dans un premier temps, l'ordre des points (horizontal puis vertical) n'était pas bon, il a donc fallu inverser i et j dans le placement des points.

J'ai ensuite eu beaucoup de difficultés à faire mes indices une fois encore, d'abord à cause du problème de côté qui correspond à la partie visible du carré, puis des problèmes d'allocations qui s'étaient avérés n'être qu'un mauvais appel de fonction dans mainWidget. De fausses pistes d'erreur de raisonnement m'ont fait douter de l'efficacité de mon placement à chaque coin supérieur gauche (en bouclant sur s*(s-1) et en faisant une condition ignorant la dernière ligne à l'aide d'un %), mais il s'est avéré juste, confirmé par l'utilisation de qInfo.

Enfin, par soucis d'optimisation, en apprenant que les indices étaient effectivement alloués à la fin de la méthode, et donc pouvait être attribué à des sommets encore non existants, j'ai intégré la création du tableau d'indices dans la double boucle de création des sommets. Quelques modifications et conditions pour la boucle ont dû être créées.

Enfin, la position du carré a dû être ajusté, et le nombre de sommets laissé libre (en dur, mais cela pourrait être facilement adapté) mais a dû être inséré dans une zone de taille fixe, afin de représenter un niveau de résolution plutôt qu'une taille comptée en sommets (de longueur 1.0).

Question 4 :

Modifier l'altitude a été relativement simple une fois les duplications de côtés supprimés, il suffit juste de modifier des paramètres z.

Déplacer la caméra n'a ensuite été qu'une question de trouver la procédure.

Bonus :

Je ne sais pas comment fonctionne la lumière sur OpenGL, mais je suppose qu'au moins à bas niveau il « suffit » d'augmenter à égal mesure les couleurs (R,G,B) des pixels, avec un gradient selon l'éloignement, et avec des tests de collisions pour les zones d'ombres.

Pour les textures en fonction de l'altitude, cela a été effectué pour le TP suivant, avec une image de noir à blanc et le même QVector2d que pour le cube.