Compte rendu – Moteur de jeux

Question 1

- *A quoi servent les classes MainWidget et GeometryEngine?*

MainWidget sert de fenêtre principale, charge les ressources et crée une instance de GeometryEngine qui elle sert à créer et afficher les points du cube. MainWidget se charge ensuite de gérer les événements et la boucle principale.

- *A quoi servent les fichiers fshader.glsl et vshader.glsl ?*

Se sont des shaders respectivement FragmentShader et VertexShader. Les shaders sont des programmes exécutés directement par la carte graphique.
Le VertexShader se charge généralement de projeter les points de l'espace 3D dans l'espace du viewport. Quant au FramentShader, il s'occupe de calculer la couleur de chaque pixel.

Question 2

- *Expliquer le fonctionnement des deux méthodes* `void GeometryEngine::initCubeGeometry()` *et* `void GeometryEngine::drawCubeGeometry(QOpenGLShaderProgram *program)`

InitCubeGeometry alloue et assigne les tableaux de points et les indices de points pour les triangles dans les tableaux OpenGL pour pouvoir les utiliser directement dans le rendu.
DrawCubeGeometry se charge d'envoyer les indices des tableaux dans les shaders puis d'afficher le cube.

Question 3

> *En vous inspirant des deux méthodes précédente, écrivez les méthodes permettant d'initialiser et d'afficher une surface plane (16*16 sommets) composée de triangles. Pour cela, vous devrez écrire deux nouvelles méthodes* `void GeometryEngine::initPlaneGeometry()` *et* `void GeometryEngine::drawPlaneGeometry(QOpenGLShaderProgram *program)`

Il suffit de créer un tableau de 256 points dont les coordonnées spatiales et coordonnées de textures sont déduite en fonction de l'indice du point.
`position[i * 16 + j] = (-1.0 + i / 8, -1.0 + j / 8, 0)` avec $\forall i,j \in [0,15]$
`texture[i * 16 + j] = (i / 15, j / 15)` avec $\forall i,j \in [0,15]$
Ensuite il suffit de créer un tableau d'indice pour chaque triangle et pour chaque face.

Question 4

> *Modifier l’altitude (z) des sommets pour réaliser un relief.*
> *Déplacer la caméra à hauteur fixe au-dessus du terrain.*
> *Utiliser le clavier pour avancer, reculer, et déplacer la camera de gauche à droite.*

Pour l'altitude, il suffit simplement de modifier la coordonnée z de chaque point. Dans mon cas, j'ai appliqué une fonction aléatoire.
La rotation initiale du plan fait que je n'ai pas eu besoin de modifier quoi que ce soit pour placer la caméra au-dessus du plan.
Enfin, pour lire les entrées clavier, j'ai surchargé les méthodes `keyPressEvent` et `keyReleaseEvent` dans lequelles j'enregistre les touches pressées puis j'utilise un `ElapsedTimer` pour déplacer la caméra à vitesse constante lorsque les touches `UP`, `DOWN`, `LEFT` et `RIGHT` sont pressées.

Question bonus

- *Expliquer comment vous vous y prendriez pour les parties bonus*

Dans le cas de la lumière, je pense que j'aurai envoyé les coordonnées de celles-ci et celles de la caméra dans le VertexShader pour calculer les composants ambiants, spéculaires et diffus et calculer la couleur du point avec ceux-ci.

Enfin, pour la couleur en fonction de l'altitude, j'aurais calculé celle-ci dans le VertexShader pour faire varier la couleur du point.