Compte-rendu

TP1 – Moteur de jeux

GALLEAN Benjamin

Lien GIT : https://github.com/Fireez30/MoteurDeJeuxM2

Question 1 : *A quoi servent les classes MainWidget et GeometryEngine?*

La classe MainWidget sert à mettre en place la fenêtre et OpenGL pour pouvoir afficher le cube, et récupère aussi les input de l'utilisateur pour pouvoir le faire tourner avec la souris.

Elle dessine ensuite le cube et lui applique la texture.

Geometry Engine sert à initialiser les points qui vont composer le cube ainsi que les coordonées sur la texture que l'on voudra appliquer. Ensuite on regroupe les points en triangles pour pouvoir les afficher avec drawElements(), on envoie les points et les indices dans les buffers OpenGL avant de les afficher

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

Ces fichiers servent a pouvoir passer la texture à appliquer au cube en coordonées écran, puis a appliquer la couleur de la texture aux points

Question 2 : *Expliquer le fonctionnement des deux méthodes*

void GeometryEngine::initCubeGeometry() :

Cette fonction crée un tableau de points avec leurs coordonnées dans l'espace 3D, ainsi que pour chaque point une coordonnées dans l'espace 2D de la texture du dé qui lui correspond.

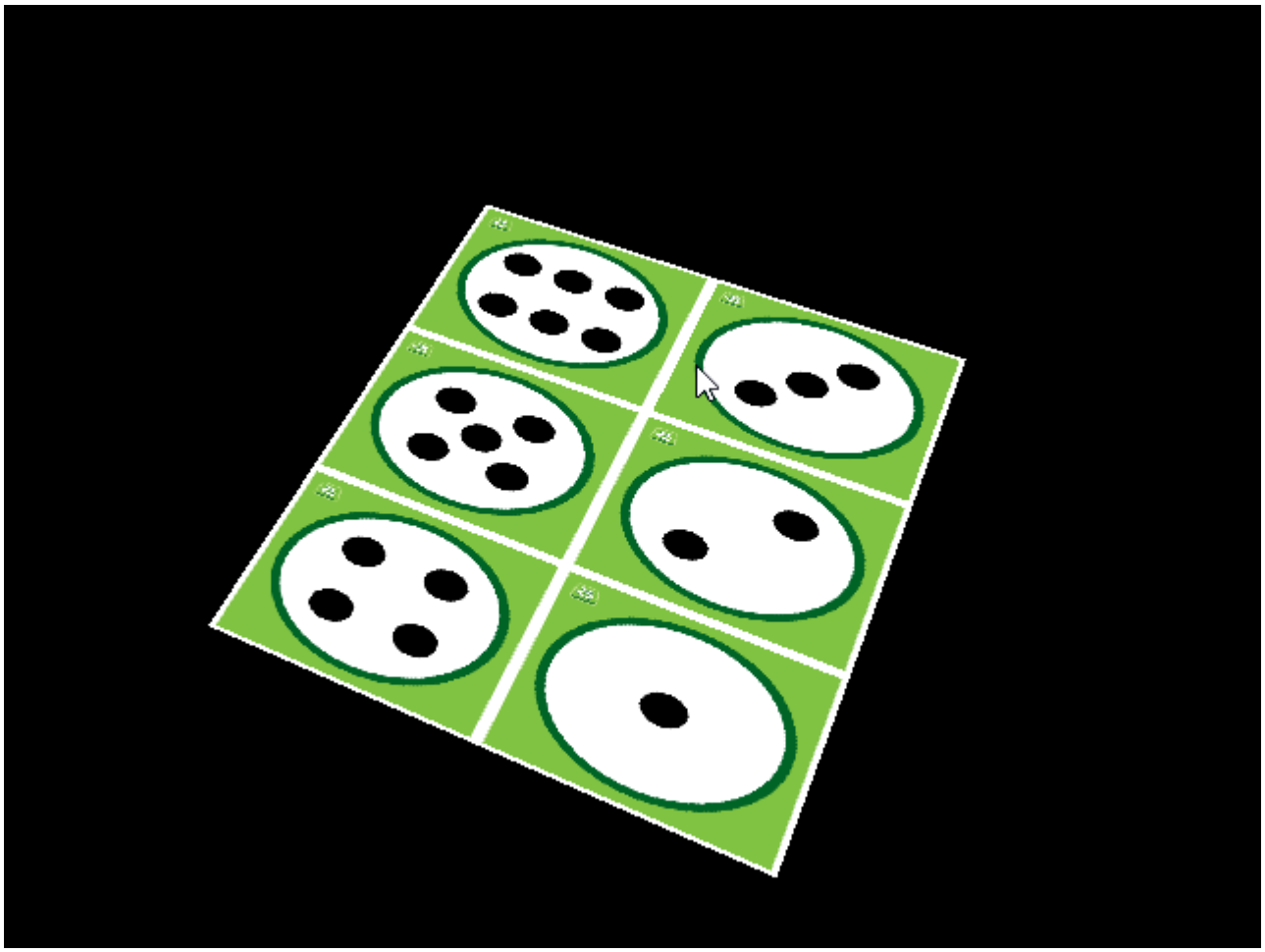
Elle crée ensuite un tableau de short qui correspond aux indices des points crées précédemment. Enfin, elle envoie le tableau de point et le tableau d'indices des triangles dans les buffers OpenGL,

void GeometryEngine::drawCubeGeometry(QOpenGLShaderProgram *program) :

Cette fonction va afficher le cube dans la scène en utilisant les tableaux crées précédemment. Elle va dire à OpenGL quels buffers utiliser (ceux bind dans la fonction précédente), préciser ou trouver la texture associée à chaque triangle avant de lancer le rendu du cube avec « glDrawElements »

Question 3 : *Ecrire 2 méthodes pour calculer un plan de 16x16 sommets*

Voici le résultat de mes fonctions :



Question 4 : *Modifier l'altitude (z) des sommets pour réaliser un relief.*

Ici la hauteur des sommets est générée aléatoirement entre 0 et 3 lors de la création des points du plan.

Utiliser le clavier pour avancer, reculer, et déplacer la camera de gauche à droite.

Voici la fonction s'occupant des inputs pour cette question :

```
void MainWidget::keyPressEvent (QKeyEvent * event)
{
    if(event->key() == Qt::Key_Q)
            x++;

    if(event->key() == Qt::Key_D)
            x--;

    if(event->key() == Qt::Key_Z)
           y--;

    if(event->key() == Qt::Key_S)
           y++;

    if(event->key() == Qt::Key_W)
           z++;

    if(event->key() == Qt::Key_X)
           z--;

        update();
}
```

Ici, je gère le déplacement de la caméra avec Z Q S et D et le zoom avec X et W.

Je modifie des variables stockées dans la classe MainWidget, qui seront simplement utilisés dans la fonction PaintGL() pour positionner la caméra :

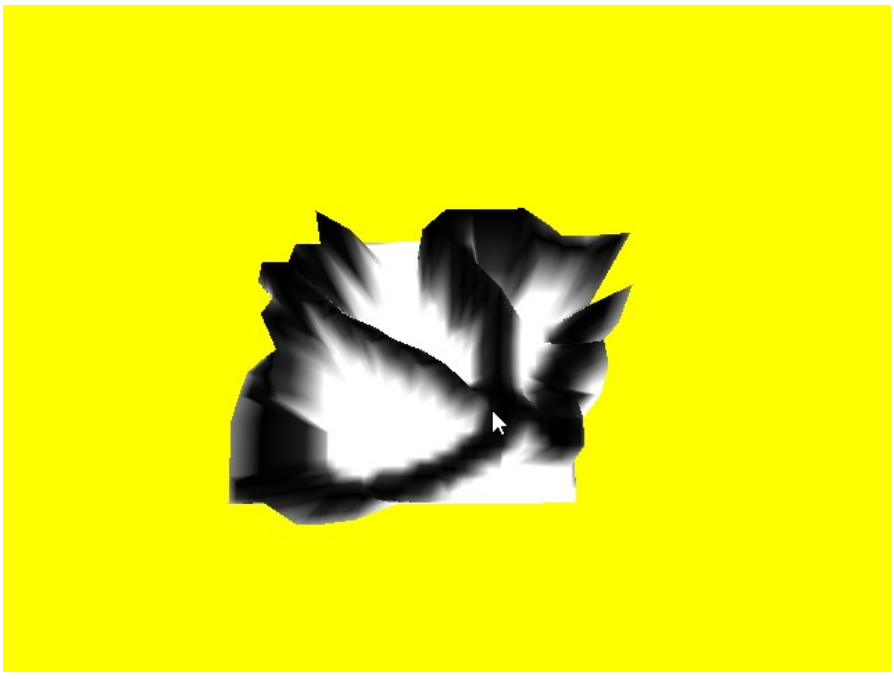
```
// Calculate model view transformation
QMatrix4x4 matrix;
matrix.translate(x, y, z);
matrix.rotate(rotation);
```

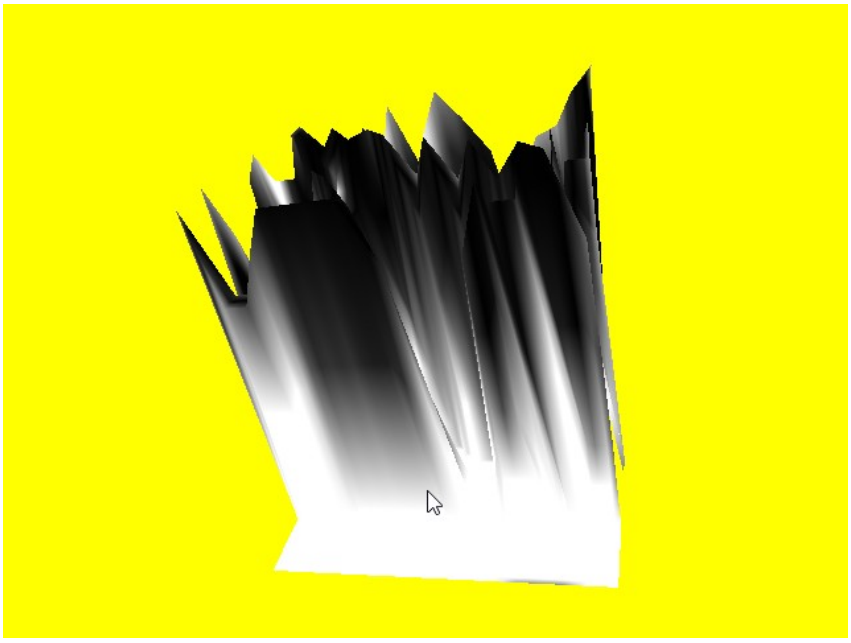
TP2 – Moteur de jeux

Question 1 : *Lire une carte d'altitude et afficher un terrain dont l'altitude en chaque point est donnée par la carte*

Voici la carte que j'ai utilisé pour ce compte rendu :

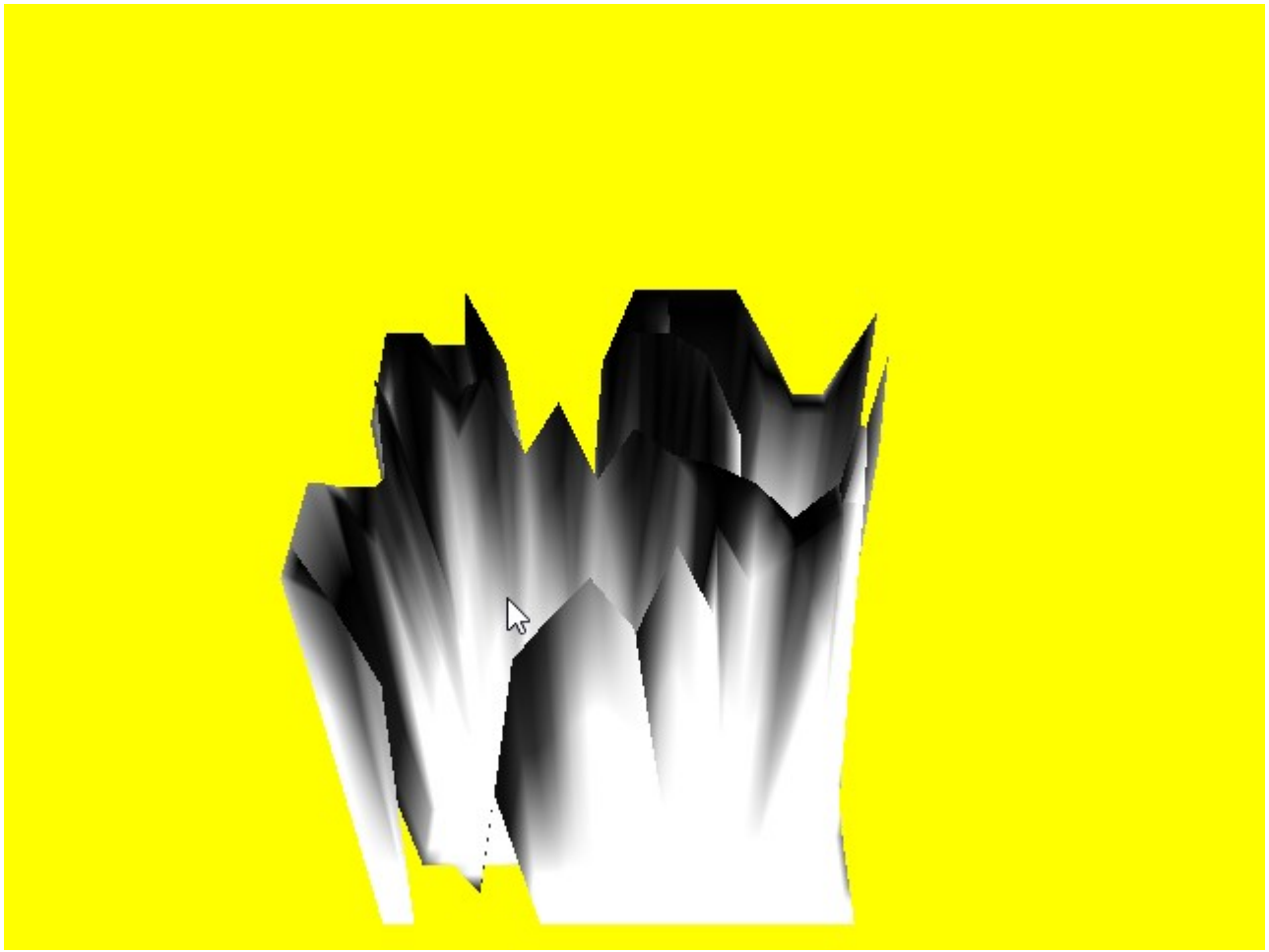
Voici le rendu dans l'application :





Question 2 : *Regarder le terrain sous un angle de 45 degés et le faire tourner autour de son origine*

Voici l’angle d’arrivé dans mon application :



Question 3 : *Comment est contrôlée la mise à jour du terrain dans la classe MainWidget ? A quoi sert la classe QTimer ? Comment fonctionne-t-elle ?*

La mise à jour du terrain de la classe MainWidget se fait grâce à un timer lancé à l'initialisation du programme. Il va ensuite être relancé toutes les X ms, X étant son premier paramètre à sa création.

Lors que le timer est relancé, un appel à la fonction timerEvent() est fait, qui va mettre à jour la rotation que l'on va appliquer la matrice de rotation, et enfin appellé la fonction qui refresh l'affichage OpenGL.

Modifier le constructeur de la classe MainWidget pour qu'il prenne en paramètre la fréquence de mise à jour (en frames par seconde).

Rajout d'un paramètre entier "max_fps" qui sera utilisé pour fixer l'intervalle de temps entre les départ de timers, et on va juste appeller le timer toutes les 1000/max_fps ms et ainsi avoir la fréquence de mise à jour voulue. Je ne sais pas ce qu'il se passe avec mon ordinateur mais il semble avoir du mal à aller jusqu'à la fréquence max choisie (suremêment des soucis de performances).

On vérifie bien évidemment que les fps max ne soient pas fixés à 0.

Modifier votre programme principal pour afficher votre terrain dans quatre fenêtres différentes, avec des fréquences de mise à jour différentes (1 FPS, 10 FPS, 100 FPS, 1000 FPS). Qu'observez-vous ?

J'ai rassemblé les fenêtre en utilisant un QgridLayout mais je ne peux plus faire d'input clavier sur mon application, uniquement avec la souris. On voit bien que les 4 fenêtres ne sont pas du tout fluide de la même façon, donc la limitation des fps marche bien.

Il va falloir créer une classe transférant les input clavier aux différentes sous fenêtres.

La classe a été crée et on peut donc faire tourner l'affichage en même temps avec des inputs clavier. Mais on remarque l'apparition d'un soucis : tel quel, le programme est dépendant du nombre de fps de la fenetre. Plus on a de fps, plus la même action se finit tôt.

Utiliser les flèches UP et DOWN pour modifier les vitesses de rotations de votre terrain (vitesse identique dans toutes les fenêtres). Qu'observez-vous ?

Même problème que plus haut

Bonus : Texture selon les cours :

Il faudrait modifier le vshader pour prendre en compte que si un sommet dépasse une position (par ex 0,5) et mettre sa couleur a blanc, sinon il faut texturer dans le fshader avec gl_fragcolor.

Pour jouer avec la lumière, j'implémenterais la méthode phong.