Compte rendu TP2

Arnaud Sanchez et Thomas Rampin

Question 2:

Concernant la modification de l'affichage pour avoir notre caméra incliné à 45 degrés, il nous a suffit de modifier la fonction suivante en mettant un paramètre à 45:

```
QQuaternion framing = QQuaternion::fromAxisAndAngle(QVector3D(1,0,0),45.0);
```

Pour la rotation de la caméra nous avons expérimenté deux méthodes, une facile qui consiste à faire tourner la caméra autour de l'axe des Y. Et une autre plus complexe qui consiste à bouger la caméra et la faire tourner en gardant la cible comme centre de rotation.

Question 1:

Pour la première question nous avons chargé la heightmap comme une texture et nous l'avons envoyé au Vertex Shader. Pour changer la valeur y de chaque vertex, on crée un nouveau point dans le shader contenant la valeur x et z d'origine du point et pour la valeur y, on prend la valeur du canal R du pixel de la heightmap qui correspond au vortex courant de notre plan.

```
vec4 a_height = vec4(a_position.x, (texture2D(height_map,
a_texcoord).r), a_position.z, 1.0f);
```

Question 3:

La mise à jour du widget est contrôlée par l'appel de la fonction update(). A chaque tick le Widget était mis à jour. Nous avons alors mis en place un système qui donne un lapse de temps pendant lequel le timer va être endormi. Cela va nous permettre de contrôler le nombre d'images par secondes par la suite. La classe QTimer contient une fonction `connect()` qui permet après un timeout d'envoyer un signal qui va déclencher la fonction d'update. Ce timeout envoie des signaux à intervalle régulier.

Pour afficher notre fenêtre avec un nombre de FPS différents, on a du ajouter un appel à update() dans la fonction timerEvent() de mainWidget et supprimer tout autre appel ailleurs dans le code. On a ensuite ajouter un QTimer dans mainWidget et on a appeler la fonction start() de ce timer dans la fonction initializeGL() où suivant un certain nombre de millisecondes, le timer déclenche l'appel à la fonction timerEvent(). on obtient ainsi le bon nombre de fps.

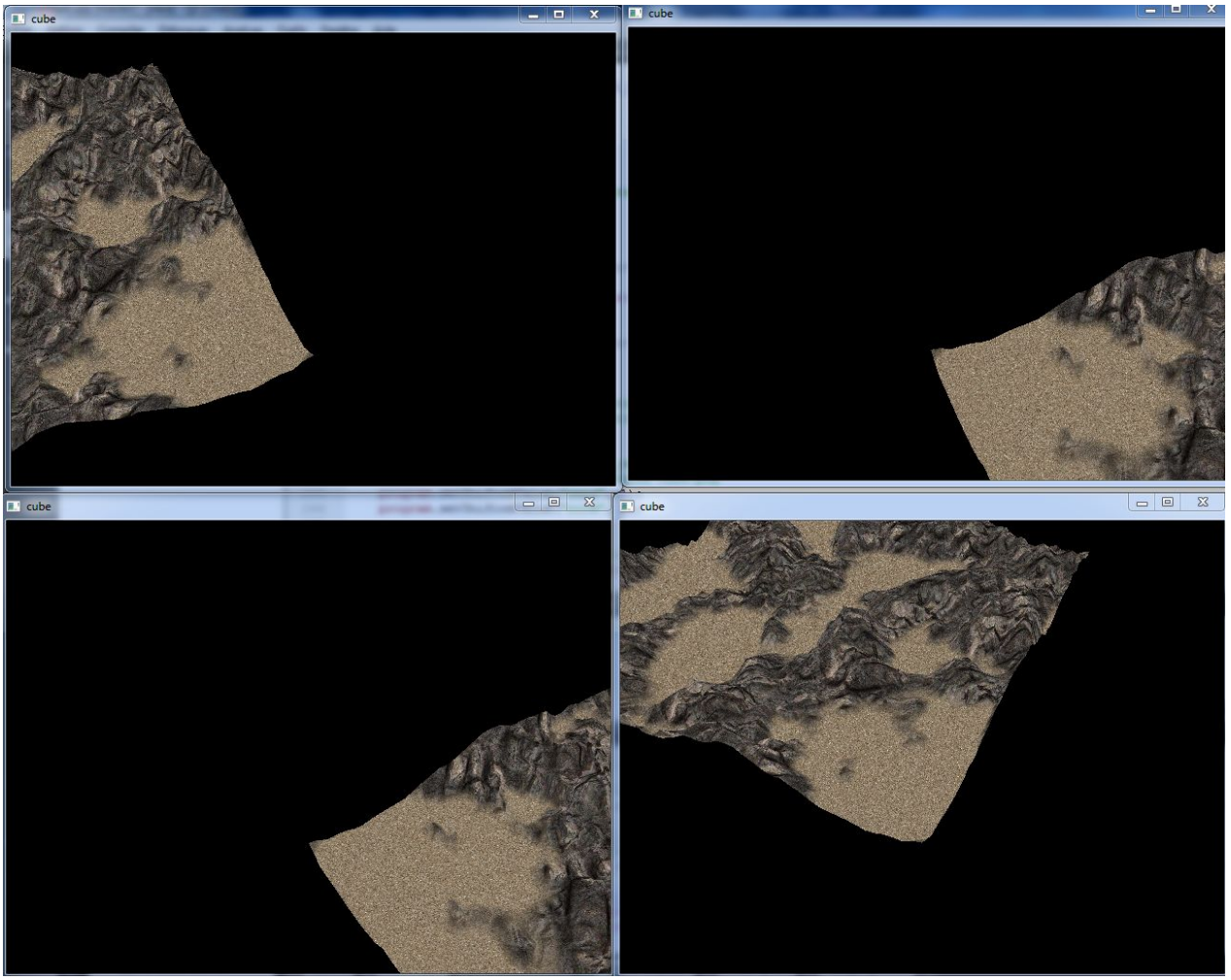
```
timer.start(1000/fps, this);
```

On observe que suivant le nombre de FPS choisi, la vitesse de rotation de la caméra est plus ou moins rapide. Plus les FPS sont élevées, plus la caméra est rapide. Cela est dû au fait que le calcul fait pour bouger la caméra est réalisé à chaque update() donc un plus grand nombre d'update fait que la caméra bouge plus vite.

Bonus :

Textures différentes suivant l'altitude des vertex :

On charge deux textures différentes en plus de la heightmap, une de sable et l'autre de rocher. Dans le vertex shader, après avoir calculé l'altitude de notre vertex, on fait passer au fragment shader la nouvelle valeur y du vertex à l'aide d'une variable "varying" que l'on déclare dans le vertex shader et le fragment shader. Dans le fragment shader, on vérifie simplement si la valeur y est supérieure à une valeur donnée. Si oui, on applique la texture rocher au vertex sinon on applique la texture sable. Aussi on a aussi ajouté un entre deux où la texture de sable et de rocher vont se mélanger afin d'éviter une rupture brusque entre la texture de sable et de rocher.



Différentes vue à 1, 10, 100 et 1000 FPS avec l'application des textures en fonctions de la heightmap.