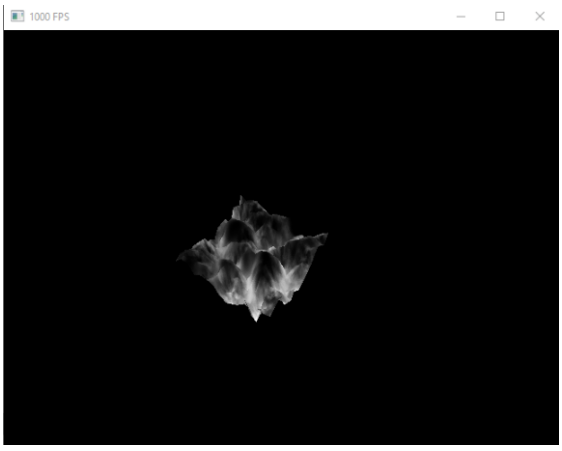
Moteurs de Jeux - Compte rendu TP2

Question 1

```
void GeometryEngine::initHeightMapGeometry(){
    QImage img = QImage("../TP1/heightmap-1.png");
    unsigned short nbVertecesInRow = 16;
    VertexData* vertices = new VertexData[nbVertecesInRow*nbVertecesInRow];

    int index = 0;
    for(int x=0; x<nbVertecesInRow; x++){
        for(int y=0; y<nbVertecesInRow; y++){
            float z = img.pixelColor(x*img.width()/(float)nbVertecesInRow, y*img.height()/(float)nbVertecesInRow).black()/512.f;
            vertices[index] = {QVector3D((float)x/nbVertecesInRow,(float)y/nbVertecesInRow,z),
                               QVector2D((float)x/(nbVertecesInRow-1),(float)y/(nbVertecesInRow-1))};
            index++;
        }
    }

    unsigned short nbIndexes = (nbVertecesInRow*2+1)*2+(nbVertecesInRow*2+2)*(nbVertecesInRow-3);
    index = 0;
    GLushort* indices = new GLushort[nbIndexes];
    unsigned short j=0;
    for(unsigned short i=0; i<nbVertecesInRow-1; i++){
        // Duplique l'indice en début de ligne, sauf pour la première
        if(i!=0){
            indices[index] = i*nbVertecesInRow+j;
            index++;
        }
        for(j=0; j<nbVertecesInRow; j++){
            indices[index] = i*nbVertecesInRow+j;
            index++;
            indices[index] = (i+1)*nbVertecesInRow+j;
            index++;
        }
        // Duplique l'indice en fin de ligne, sauf pour la dernière
        if(i!=nbVertecesInRow-2){
            indices[index] = (i+1)*nbVertecesInRow+(j-1);
            index++;
            // Réinitialisation nécessaire pour le premier if
            j=0;
        }
    }

    // Transfer vertex data to VBO 0
    arrayBuf.bind();
    arrayBuf.allocate(vertices, nbVertecesInRow*nbVertecesInRow * sizeof(VertexData));

    // Transfer index data to VBO 1
    indexBuf.bind();
    indexBuf.allocate(indices, nbIndexes * sizeof(GLushort));
}
```

Voici le résultat observé avec la première carte d'altitude :



Question 2

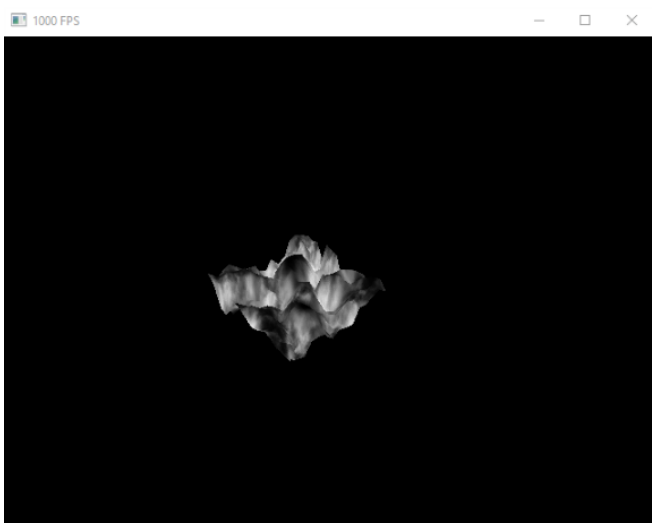
Dans un premier temps j'ai changé la boucle de création des positions des sommets de façon à ce que l'origine soit au centre du terrain et non en bas à gauche comme précédemment.

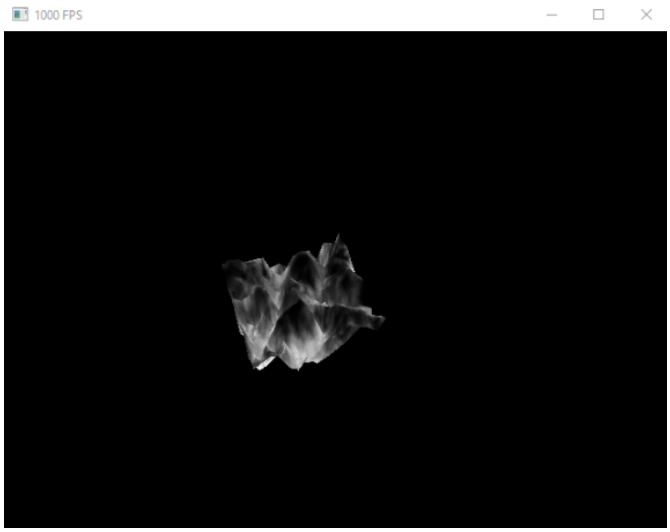
```
for(int x=0; x<nbVertecesInRow; x++){
    for(int y=0; y<nbVertecesInRow; y++){
        // La composante z du vecteur de position est calculée en fonction du niveau de gris/noir de la heightmap
        float z = img.pixelColor(x*img.width()/(float)nbVertecesInRow,          use of old-style cast   implicit conversi
                                 y*img.height()/(float)nbVertecesInRow).black()/512.f;     use of old-style cast   implicit con
        vertices[index] = {QVector3D((float)(x-nbVertecesInRow/2)/nbVertecesInRow,(float)(y-nbVertecesInRow/2)/nbVertecesInRow,z),
                           QVector2D((float)x/(nbVertecesInRow-1),(float)y/(nbVertecesInRow-1))};
        index++;
    }
}
```

Ensuite, j'ai modifié l'événement *timerEvent* pour que la vitesse de rotation reste constante. Pour cela j'ai changé les valeurs initiales des variables *angularSpeed* et *rotationAxis*.

```
//! [1]
void MainWidget::timerEvent(QTimerEvent *)
{
    // On veut une vitesse constante, pas besoin de diminuer la vitesse
    rotation = QQuaternion::fromAxisAndAngle(rotationAxis, angularSpeed) * rotation;
    update();
}

MainWidget::MainWidget(QWidget *parent) :
    QOpenGLWidget(parent),
    geometries(0),
    texture(0),
    angularSpeed(1.0),
    cam(-0.25,-0.5,0),
    rotationAxis(0,0,1)
{
}
```

Maintenant le terrain tourne autour de son origine.





Question 3

La mise à jour du terrain est contrôlée grâce à la fonction/l'événement *timerEvent* de la classe MainWidget.
La classe Qtimer, comme son nom l'indique, sert à gérer un timer.
En faisant *QTimer *timer = new QTimer(this)* on crée un objet timer, ensuite, il suffit d'appeler la fonction *start* associée à ce timer. *timer→start(1000)*
Le paramètre de la fonction est en millisecondes, on vient donc de créer un timer qui appelle l'événement *timerEvent* une fois par seconde.

```
// On surchage le constructeur pour pouvoir prendre en paramètre les fps
MainWidget::MainWidget(int fps, QWidget *parent) :
    QOpenGLWidget(parent),
    geometries(0),
    texture(0),
    angularSpeed(1.0),
    cam(-0.25,-0.5,0),
    rotationAxis(0,0,1),
    fps(fps)
{
}
```

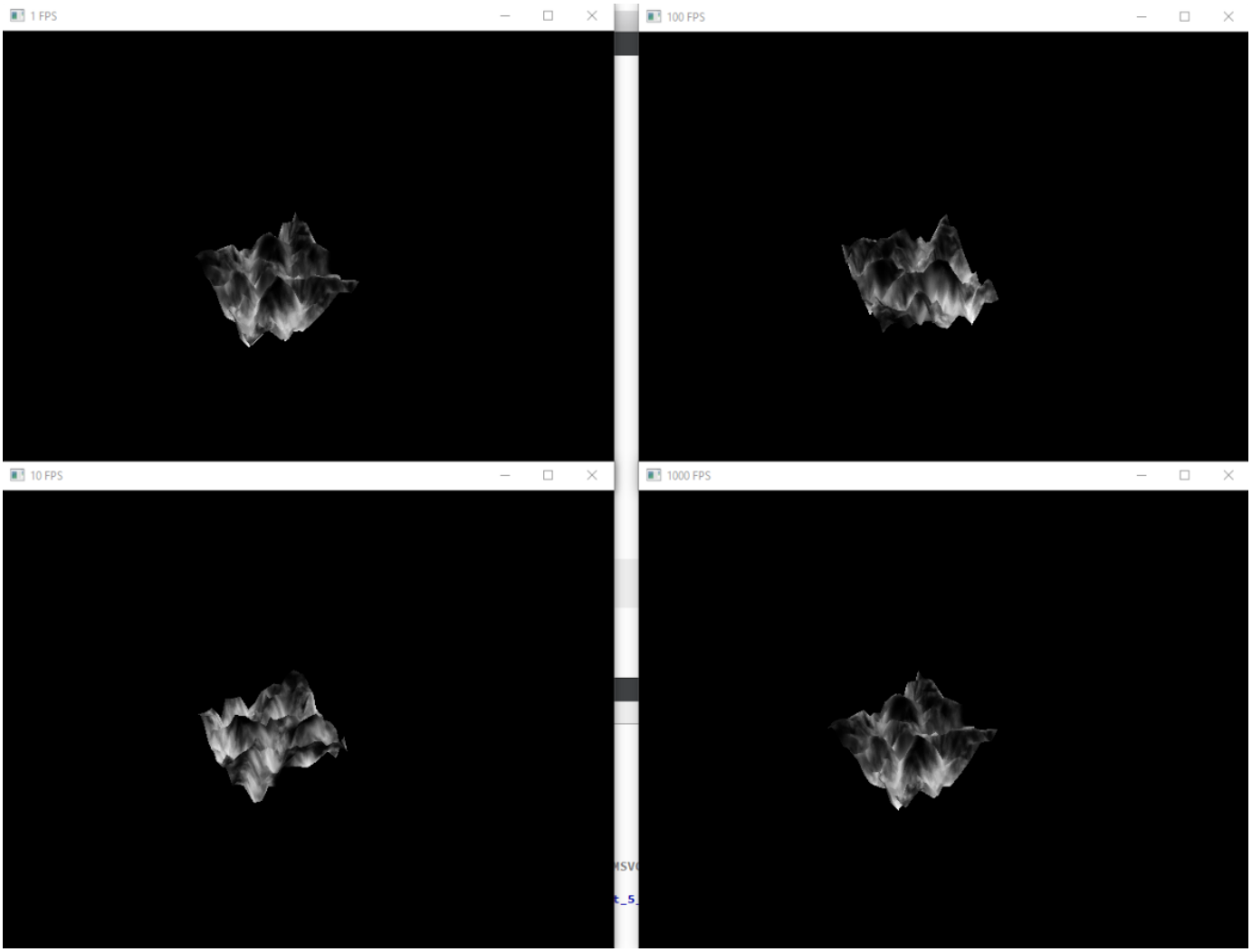
Puis j'ai modifié le lancement du timer pour que l'événement *timerEvent* soit appelé en fonction des fps actuels.

```
void MainWidget::initializeGL()
{
    initializeOpenGLFunctions();

    glClearColor(0, 0, 0, 1);

    initShaders();
    initTextures();

//! [2]
    // Enable depth buffer
    glEnable(GL_DEPTH_TEST);

    // Enable back face culling
    glEnable(GL_CULL_FACE);
//! [2]

    geometries = new GeometryEngine;

    // Fixe le nombre de fois que l'event timerEvent est appelé
    // et donc le taux de rafraichissement, en millisecondes
    timer.start(1000/fps, this);
}
```

Finalement j'ai modifié la fonction *main* pour qu'elle affiche 4 fenêtres avec des taux de rafraîchissement différents.

```
// Paramétrage et affichage des 4 instances
MainWidget widget1(1);
widget1.setWindowTitle("1 FPS");
MainWidget widget2(10);
widget2.setWindowTitle("10 FPS");
MainWidget widget3(100);
widget3.setWindowTitle("100 FPS");
MainWidget widget4(1000);
widget4.setWindowTitle("1000 FPS");
widget1.show();
widget2.show();
widget3.show();
widget4.show();
```

Voici le résultat :



On peut observer que la vitesse de rotation qu'on perçoit augmente plus les fps augmente et que, quand la vitesse de rotation est basse, 1 et 10 fps, le mouvement de rotation est saccadé. Ce qui est totalement normal vu que la fonction *update* n'est pas appelée autant de fois que pour les fenêtres en 100 et 1000 fps.

Pour modifier la vitesse de rotation avec les flèches du haut et du bas j'ai modifié la valeur de *angularSpeed*.

```
void MainWidget::keyPressEvent(QKeyEvent *event){
    switch(event->key()){
        case Qt::Key_Left :
            cam.setX(cam.x()-1.0f/10);
        break;
        case Qt::Key_Right :
            cam.setX(cam.x()+1.0f/10);
        break;
        case Qt::Key_Up :
            //cam.setY(cam.y()+1.0f/10);
            angularSpeed += 0.05;
        break;
        case Qt::Key_Down :
            //.setY(cam.y()-1.0f/10);
            angularSpeed -= 0.05;
        break;
    }
    std::cerr << "angular speed : " << angularSpeed << std::endl;
    update();
}
```

Pour que les 4 fenêtres tourne à la même vitesse il faut que la variable *angularSpeed* est :

- 1.0 comme valeur pour la fenêtre à 1000 fps.
- 1.2 comme valeur pour la fenêtre à 100 fps.
- 7.0 comme valeur pour la fenêtre à 10 fps.
- 60.0 comme valeur pour la fenêtre à 1 fps.

Cependant cela ne change rien au problème du mouvement saccadé pour les fenêtres à 1 et 10 fps vu qu'on ne fait qu'augmenter l'angle de rotation, le taux de rafraîchissement reste le même dans tous les cas.

Je ne suis pas sûr d'avoir compris l'énoncé pour ce dernier point, au final en utilisant les flèches UP et DOWN je modifie la vitesse de rotation du terrain pour arriver à une vitesse identique dans toutes les fenêtres, ce qui semble correspondre à ce qui est demandé.