Moteurs de Jeux - Compte rendu TP3

Synchroniser les fenêtres et changement de saisons

Pour effectuer cela j'ai créé une fenêtre par saison, en lui passant sa saison actuelle en paramètre, ce qui fait donc 4 fenêtres au total. J'ai ensuite connecté ces 4 fenêtres à un timer pour que les fenêtres soient averties toutes les *x* secondes par le timer.

Voici le code dans le fichier *main.cpp* :

```
SeasonTimer seasonTimer(1000);
seasonTimer.startTimer();

// Paramétrage et affichage des 4 instances avec saisons et fps
MainWidget widget1(0, 1);
widget1.setWindowTitle("Winter");
MainWidget widget2(1, 10);
widget2.setWindowTitle("Spring");
MainWidget widget3(2, 100);
widget3.setWindowTitle("Summer");
MainWidget widget4(3, 1000);
widget4.setWindowTitle("Fall");
widget1.show();
widget2.show();
widget3.show();
widget4.show();

QObject::connect(&seasonTimer, &SeasonTimer::seasonChange, &widget1, &MainWidget::changeSeason);
QObject::connect(&seasonTimer, &SeasonTimer::seasonChange, &widget2, &MainWidget::changeSeason);
QObject::connect(&seasonTimer, &SeasonTimer::seasonChange, &widget3, &MainWidget::changeSeason);
QObject::connect(&seasonTimer, &SeasonTimer::seasonChange, &widget4, &MainWidget::changeSeason);
```

Et voici le code du timer de saison :

```
#include <QBasicTimer>
#include <QTimerEvent>
#include "mainwidget.h"

class SeasonTimer: public QOpenGLWidget
{
     Q_OBJECT
public:
    explicit SeasonTimer(float seasonDuration);
    ~SeasonTimer();
    void startTimer();

signals:
    void seasonChange();

protected:
    void timerEvent(QTimerEvent *e) override;

private:
    QBasicTimer timer;
    float seasonDuration;
};
```

```
SeasonTimer::SeasonTimer(float seasonDuration):
    seasonDuration(seasonDuration)
{
}

SeasonTimer::~SeasonTimer()
{
}

void SeasonTimer::startTimer()
{
    std::cout << "Timer started..." << std::endl;
    timer.start(seasonDuration, this);
}

void SeasonTimer::timerEvent(QTimerEvent *)
{
    std::cout << "The season changed!" << std::endl;
    emit seasonChange();
}
```

On peut voir qu'à chaque fois que le timer entre dans son évènement il notifie tous ceux qui « l'écoute » en émettant le signal *seasonChanged*.
Ce signal est donc capté par les 4 fenêtre de saison, la méthode *changeSeason* est alors appelée. Cette méthode incrémente l'ID de la saison et met à jour la couleur du terrain en fonction de la nouvelle saison.

```
void MainWidget::updateSeasonColor(){
    switch (season) {
        //Winter
        case 0:
            terrainColor = QVector3D(0.75, 0.75, 0.75);
            break;
        //Spring
        case 1:
            terrainColor = QVector3D(0.25, 1, 0.0);
            break;
        //Summer
        case 2:
            terrainColor = QVector3D(1.0, 0.9, 0.0);
            break;
        //Fall
        default:
            terrainColor = QVector3D(1.0, 0.5, 0.0);
            break;
    }
    program.setUniformValue("texture", 0);
}

void MainWidget::changeSeason(){
    season = (season+1)%4;
    std::cout << "Season " << season << std::endl;
    updateSeasonColor();
    geometries->updateTerrainColor(terrainColor);
}
```

A chaque fois que la fonction *updateTerrainColor* est appelée le terrain est mis à jour avec la nouvelle couleur.

```
void GeometryEngine::updateTerrainColor(QVector3D color)
{
    terrainColor = color;
    initHeightMapGeometry(terrainColor);
}
```

Je n'ai presque rien changé dans la méthode *initHeightMapGeometry* par rapport au dernier TP si ce n'est d'avoir rajouté un paramètre de couleur à chaque sommet.
Un sommet a donc maintenant :

- une position
- une texture
- une couleur

```
void GeometryEngine::initHeightMapGeometry(QVector3D terrainColor){
    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++){
            // 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-sty
                                     y*img.height()/(float)nbVertecesInRow).black()/512.f;                    use of old-sty
            vertices[index] = {QVector3D((float)(x-nbVertecesInRow/2)/nbVertecesInRow,(float)(y-nbVertecesInRow/2)/nbVertecesInRow,z),
                               QVector2D((float)x/(nbVertecesInRow-1),(float)y/(nbVertecesInRow-1)),
                               terrainColor};
            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));
}
```

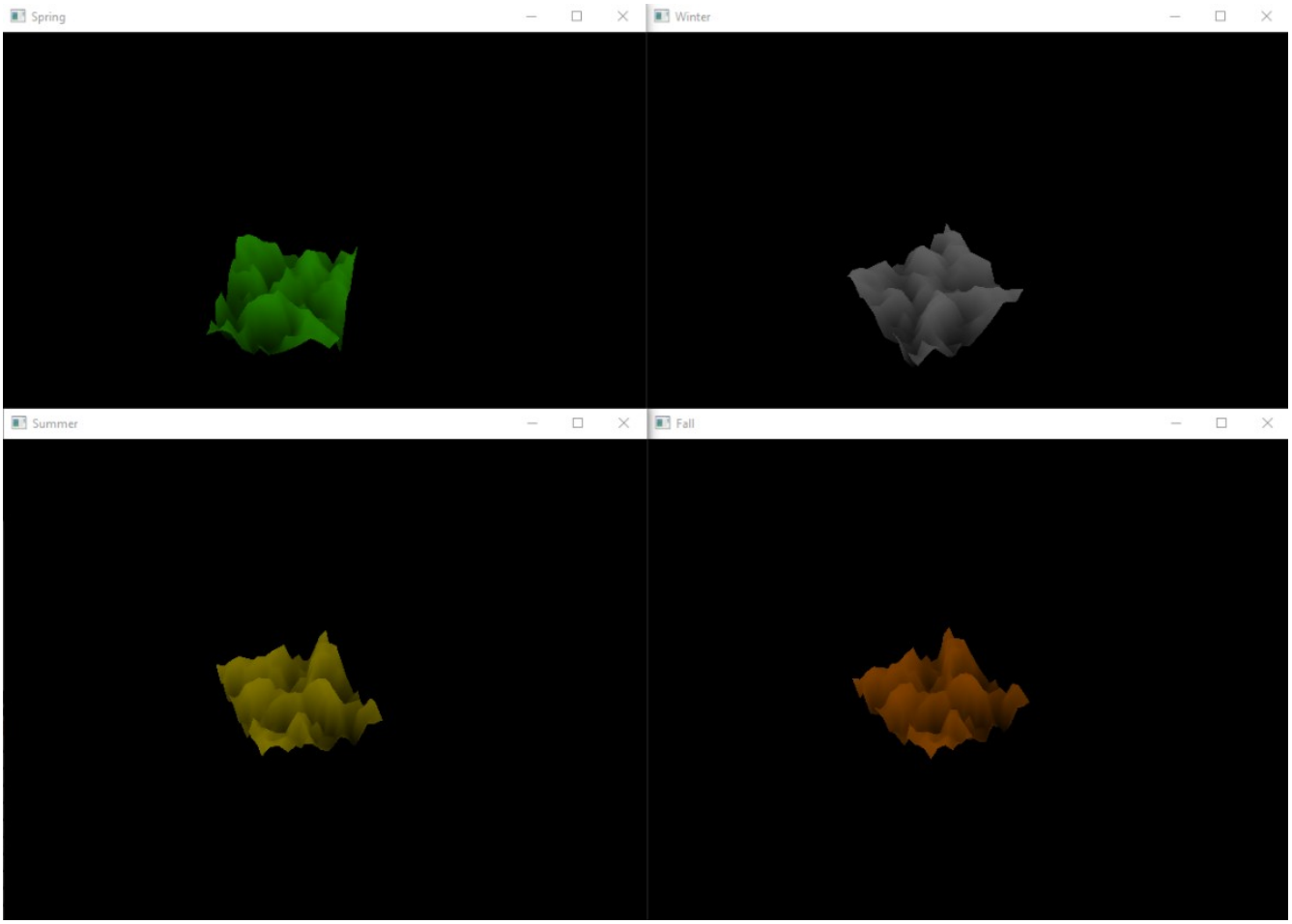
Il m’a bien sûr également fallu changer la pipeline OpenGL et les shaders pour qu’ils prennent en compte la couleur de chaque sommet.

```
void GeometryEngine::drawPlaneGeometry(QOpenGLShaderProgram *program)
{
    unsigned short nbVertecesInRow = 16;
    // Tell OpenGL which VBOs to use
    arrayBuf.bind();
    indexBuf.bind();

    // Offset for position
    quintptr offset = 0;

    // Tell OpenGL programmable pipeline how to locate vertex position data
    int vertexLocation = program->attributeLocation("a_position");
    program->enableAttributeArray(vertexLocation);
    program->setAttributeBuffer(vertexLocation, GL_FLOAT, offset, 3, sizeof(VertexData));

    // Offset for texture coordinate
    offset += sizeof(QVector3D);

    // Tell OpenGL programmable pipeline how to locate vertex texture coordinate data
    int texcoordLocation = program->attributeLocation("a_texcoord");
    program->enableAttributeArray(texcoordLocation);
    program->setAttributeBuffer(texcoordLocation, GL_FLOAT, offset, 2, sizeof(VertexData));

    // Offset for color coordinate
    offset += sizeof(QVector2D);

    int colorAttribute = program->attributeLocation("a_color");
    program->enableAttributeArray(colorAttribute);
    program->setAttributeBuffer(colorAttribute, GL_FLOAT, offset, 3, sizeof(VertexData));

    // Draw cube geometry using indices from VBO 1
    glDrawElements(GL_TRIANGLE_STRIP, (nbVertecesInRow*2+1)*2 + (nbVertecesInRow*2+2)*(nbVertecesInRow-3), GL_UNSIGNED_SHORT, 0);
}
```

```
#ifdef GL_ES
// Set default precision to medium
precision mediump int;
precision mediump float;
#endif

uniform sampler2D texture;

varying vec2 v_texcoord;
varying float height;
varying vec4 color;

//! [0]
void main()
{
    // Set fragment color from texture
    //gl_FragColor = texture3D(texture, v_texcoord);
    gl_FragColor = height * color;
}
//! [0]
```

```
#ifdef GL_ES
// Set default precision to medium
precision mediump int;
precision mediump float;
#endif

uniform mat4 mvp_matrix;

attribute vec4 a_position;
attribute vec2 a_texcoord;
attribute vec4 a_color;

varying vec2 v_texcoord;
varying float height;
varying vec4 color;

//! [0]
void main()
{
    // Calculate vertex position in screen space
    gl_Position = mvp_matrix * a_position;

    // Pass texture coordinate to fragment shader
    // Value will be automatically interpolated
    // to fragments inside polygon faces
    v_texcoord = a_texcoord;

    height = a_position[2];
    color = a_color;
}
//! [0]
```

On observe donc le résultat suivant, avec un changement de saison à chaque seconde.
Enfin ici on voit pas le changement vu que c'est une image…



Voilà pour ce qui est de mon travail concernant ce TP, j'avais commencé à faire les Quadtree mais sans trop de succès.