Université de Montpellier

Compte rendu - TP3 et TP4

HMIN317 - Moteur de Jeux
Année universitaire 2018/2019

Niveaux de détails et Gestionnaire de Scène

Auteur :
Roï Shviro

Lien GitHub :
https://github.com/Celthim/Moteur_jeux

TP3 - Synchronisation et LOD

Synchroniser les fenêtres entre elles

On reprend les fenêtres du TP précédent dans le but d'instaurer une communication entre elles. Avant tout j'ai créé une classe **MainWindow** dans le but d'implémenter un Layout pour rassembler nos fenêtres de manière plus élégante. J'y ai ajouté un signal transmettant l'information de quelle fenêtre a reçu un clique, permettant ainsi de changer de focus et donc de garder les fonctionnalités précédentes (déplacement ZQSD, zoom molette, accélération/décélération UP/DOWN).

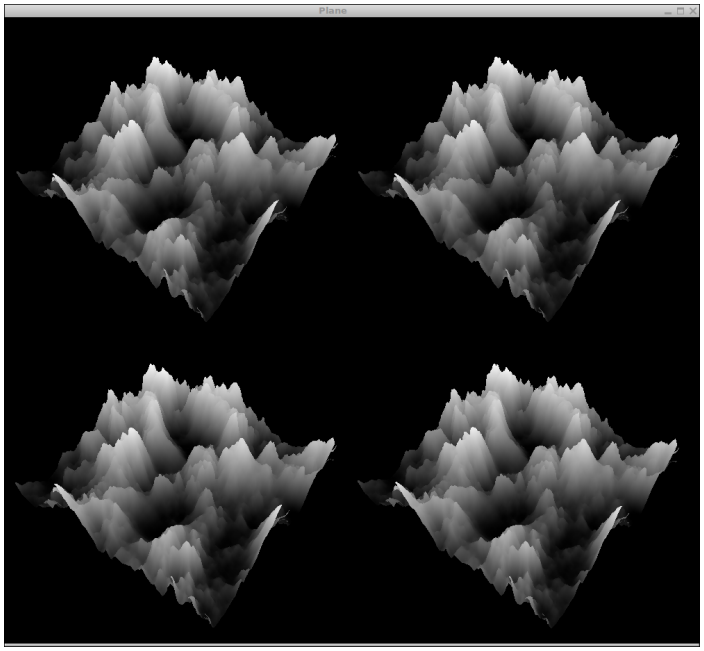

FIGURE 1 : 4 Fenêtres regroupées à l'aide d'un gridLayout

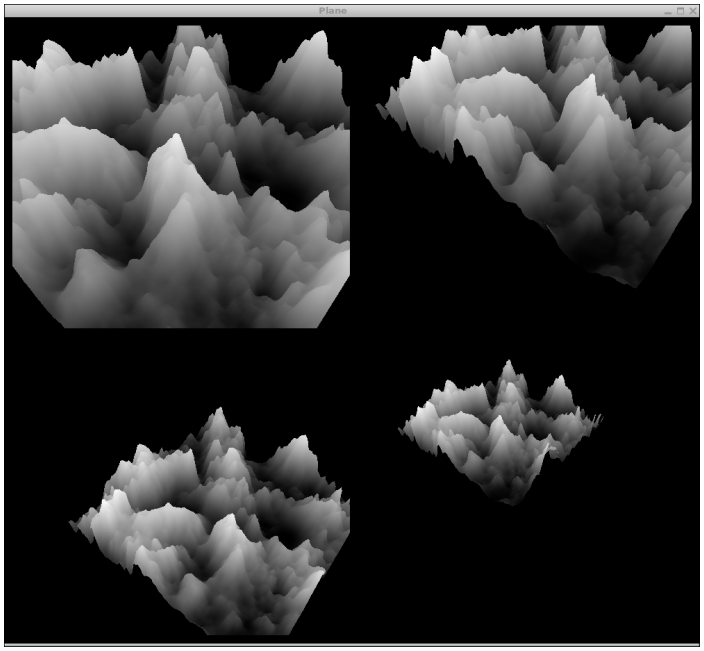

FIGURE 2 : Conservation des anciennes fonctionnalités

Dans le but de faire communiquer nos fenêtres, on veut instaurer un système de saison appliquant un changement de couleur de la scène à l'aide du fragment shader. Pour celà on commence déjà par introduire une variable au sein de fshader.glsl permettant de changer la couleur de notre rendu afin de représenter une saison.

```
//Dans fshader.glsl
uniform vec4 ColorShader;

void main()
  {
      // Set fragment color from texture
      gl_FragColor = texture2D(texture, v_texcoord)*ColorShader;
  }
```

```
//Dans mainwidget.cpp, paintGL
program.setUniformValue("ColorShader", QVector4D(1.0,0.5,0.0,1.0));
```

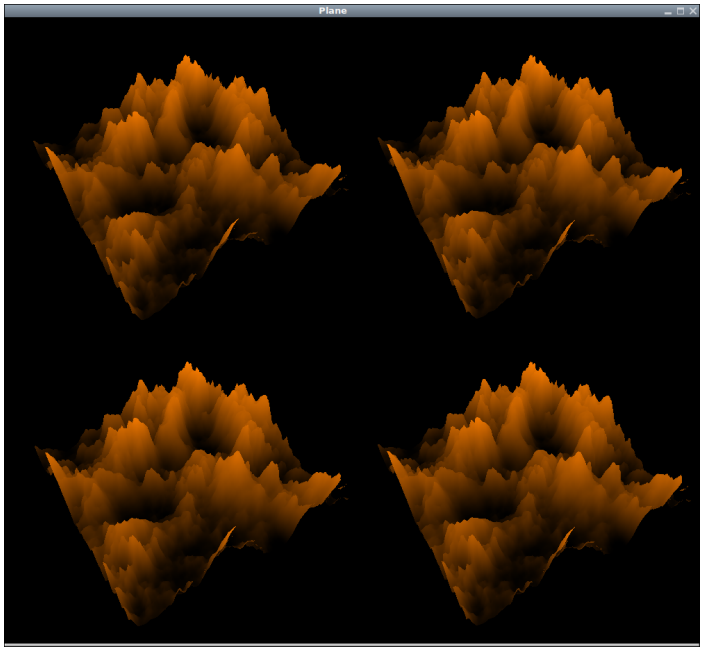

FIGURE 3 : Couleur du rendu modifiée

On crée à présent une classe calendar.cpp en charge d'émettre des signaux toutes les X secondes annonçant aux fenêtres qu'elles doivent changer de saison.

```
//Dans calendar.cpp
void Calendar::timerEvent(QTimerEvent *)
  {
      emit seasonChanged();
  }
```

```
//Dans main.cpp
Calendar c(10);

  QObject::connect(&c, SIGNAL(seasonChanged()), &widget, SLOT(changeSeason()) );
  QObject::connect(&c, SIGNAL(seasonChanged()), &widget1, SLOT(changeSeason()) );
  QObject::connect(&c, SIGNAL(seasonChanged()), &widget2, SLOT(changeSeason()) );
  QObject::connect(&c, SIGNAL(seasonChanged()), &widget3, SLOT(changeSeason()) );
```

```
//Dans mainwindget.cpp

  void MainWidget::changeSeason(){
      CurrentSeason = (CurrentSeason+1)%4;
  }

      //paintGL
  switch(CurrentSeason){

      case 0 :
          program.setUniformValue("ColorShader", QVector4D(0.8,0.8,0.0,1.0));
          BGC= QColor(46,46,0);
          break;
      case 1 :
          program.setUniformValue("ColorShader", QVector4D(0.8,0.8,1.0,1.0));
          BGC= QColor(0,46,46);
          break;
      case 2 :
          program.setUniformValue("ColorShader", QVector4D(0.2,1.0,0.0,1.0));
          BGC= QColor(0,46,0);
          break;
      case 3 :
          program.setUniformValue("ColorShader", QVector4D(1.0,0.5,0.0,1.0));
          BGC= QColor(46,23,0);
          break;
      default : ;

  }

  glClearColor(BGC.redF(), BGC.greenF(), BGC.blueF(), 1);
```

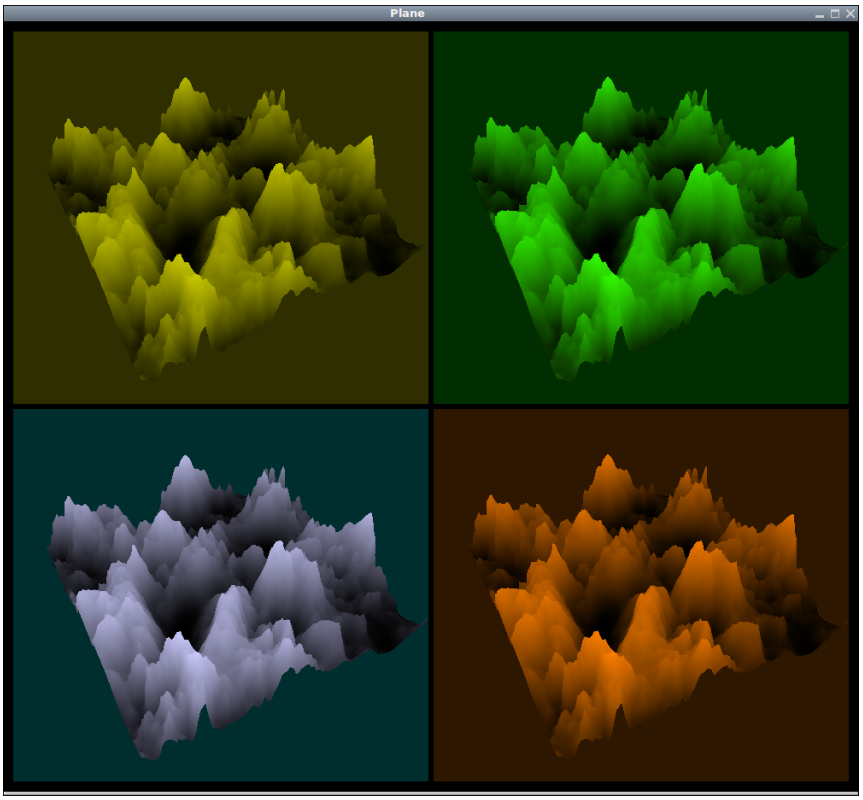

FIGURE 4 : Les 4 fenêtres avec différentes saisons

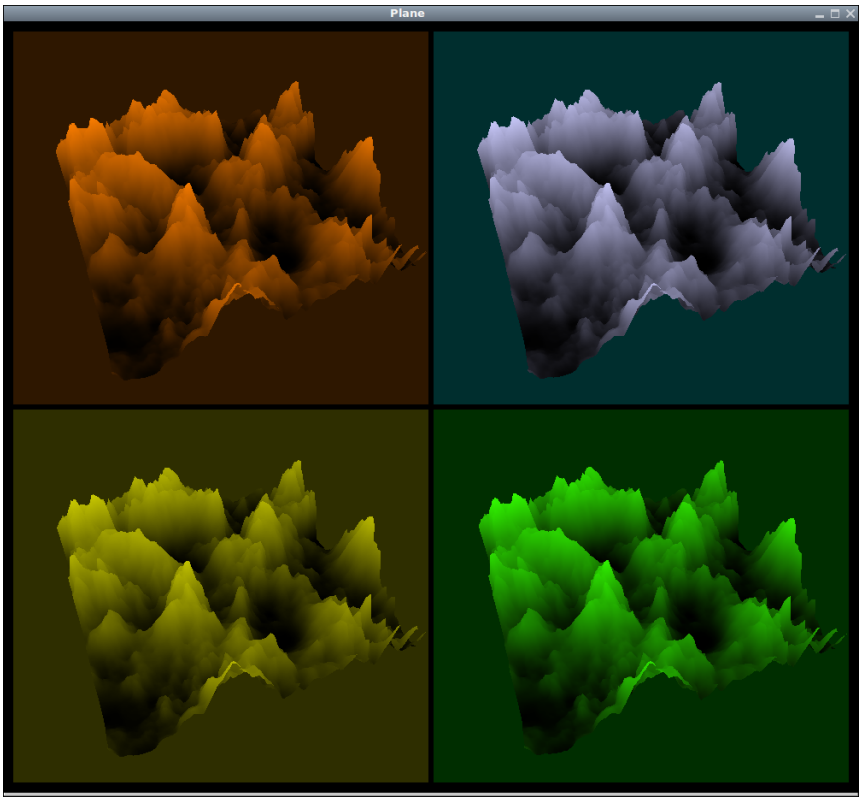

FIGURE 5 : Les 4 fenêtres après avoir reçu les signal de changer de saison

LOD - Le QuadTree

Le but désormais est d'implémenter une structure en capacité de générer en partant de notre tableau de vertices un QuadTree.

```
#include "vertexdata.h"

class QuadTree
{

public:
    QuadTree();
    QuadTree* compute(VertexData* vertices,int sizex, int sizey);
    void init(VertexData* vertices, int sizex, int sizey);

    VertexData value[4];
    QuadTree* f1;
    QuadTree* f2;
    QuadTree* f3;
    QuadTree* f4;
};
```

Malheureusement pour le moment je n'ai pas réussi à faire fonctionner mon quadtree convenablement. J'ai un début de quelque chose, mais non fonctionnel pour le moment.