HMIN317:Moteurs de jeux

Rapport TP1&2 :Prise en main de Qt Creator,Git et Opengl ES

Preparé par:

Iradukunda Valentin

Prof:

NOURA faraj

September 2020

Question 1

a)Fonctionnement des méthodes de dessin

La classe qui s'occupe qui s'occupe de dessiner c'est la classe **geometryengine**
Qui contient deux méthode principale et deux variable Vertex buffer Objects;

Variable:

ArrayBuffer:un vertex buffer object qui va contenir les données de vertex dans la memoire graphique

IndexBuffer : un vertex buffer object qui nous permet de ne pas repeter les sommets Lors du dessin ,ce qui nous permet d'économiser la memoire et d'avoir un rendu plus rapide.
Methode:

initCubeGeometry : initialise les données de sommets (position et texture)et les transmet aux deux variable **VBO** situé sur la mémoire de la carte graphique

initCubeGeometry : permet de lier les VBOs aux shader et de dessiner le cube en utilisant le initialise les données de sommets (position et texture)et les transmet aux deux variable **VBO** situé sur la mémoire de la carte graphique

a)Fonctionnement du mechanisme et fonction permettant la transmission des mise à jours à partir des entrés utilisateur

Les trois methode principale gerant les entres utilisateur se trouve dans la classe **mainwidget :**

mousePressEvent :permettant de sauvegarder la position actuel de la souris une fois cliqué

keyPressEvent :permettant de savoir sur quelle touche du clavier l'utiisateur à appuyer et de lui associer une action

mouseReleaseEvent :permettant de savoir quand l'utilisateur à relacher la souris pour appliquer les eventuels transformtion necessaire dans l'exemple du cube on determiiner l'accélération et la vitesse de rotation du cube .

Question2

La methode init plane geometry

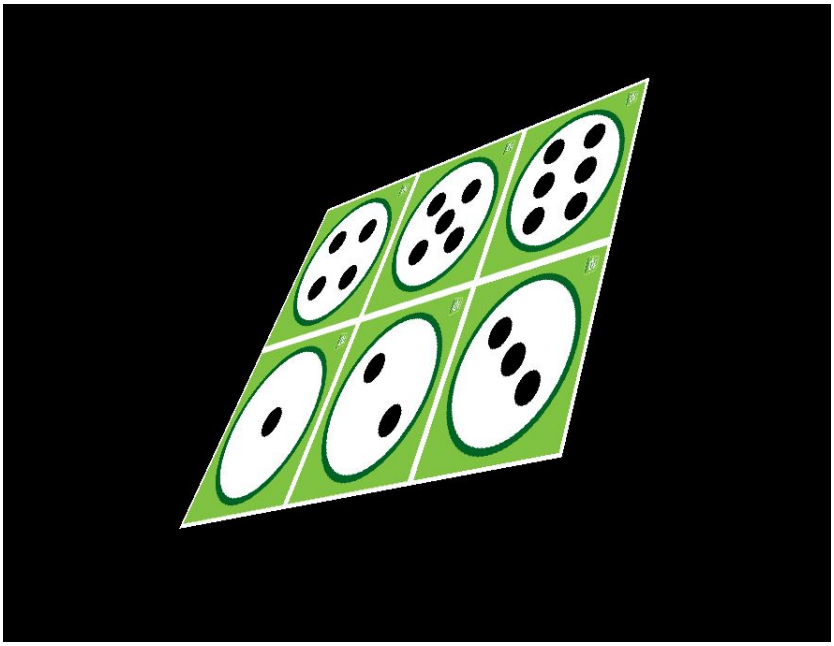
```
void GeometryEngine::initPlaneGeometry(int columns,int rows)
{
    int n = columns;
    int m=rows;
    // For plane, we need 8 vertices on the same plane z=0
    VertexData vertices[n*n] ;

      for(int j = 0; j < m; j++) {
        for(int i = 0; i < n; i++) {
            vertices[i+j*n] = {QVector3D((float)(i-n/2)/n, (float)(j-n/2)/n, (i%2)/(float)10),
              QVector2D((float)i/(n-1),(float) j/(n-1))};

        }
        //std::cout << std::endl;
    }
    int temp;
     //int taille=3*2*(n-1)*(m-1)+((m-2)*2);
      int taille=3*2*(n-1)*(m-1);
     GLushort  indexes[taille];

   int offset=0;

   bool retrieved=false;
       for(int j=0;j<m-1;j++){


           retrieved=false;
           for(int i=0;i<n-1;i++){

                indexes[i+offset+j*n]=i+j*n;
                offset++;

               indexes[i+offset+j*n]=n*(j+1)+i;
                offset++;

               indexes[i+offset+j*n]=(i+1)+j*n;
                 offset++;

               indexes[i+offset+j*n]=(i+1)+j*n;
               offset++;


               indexes[i+offset+j*n]=n*(j+1)+i;
               offset++;


               indexes[i+offset+j*n]=(i+1)+n+(j*n);
               temp=(i+1)+n+(j*n);


           }
           offset--;


}
```

Le methode Draw Plane Geometry

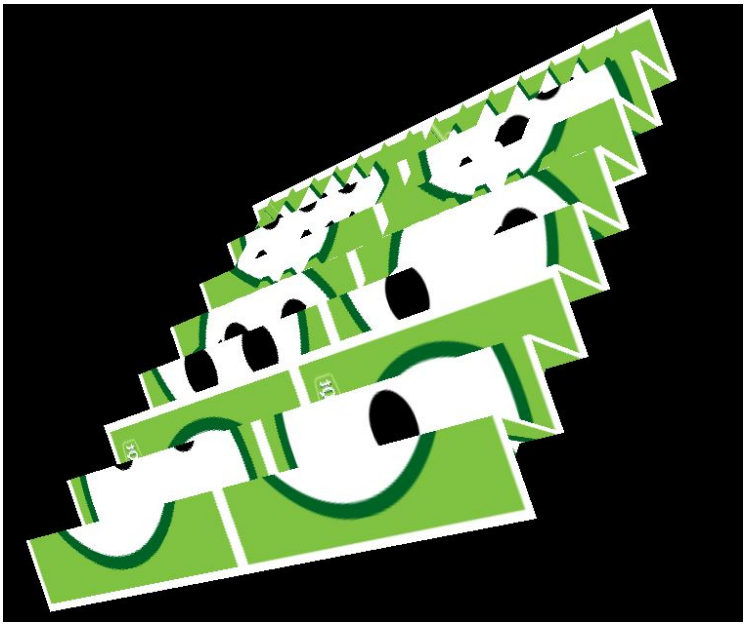
```
void GeometryEngine::drawPlaneGeometry(QOpenGLShaderProgram *program,int columns,int rows)
{


    // 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));


        glDrawElements(GL_TRIANGLE_STRIP, 1350, GL_UNSIGNED_SHORT, 0);


}
```

Resultat

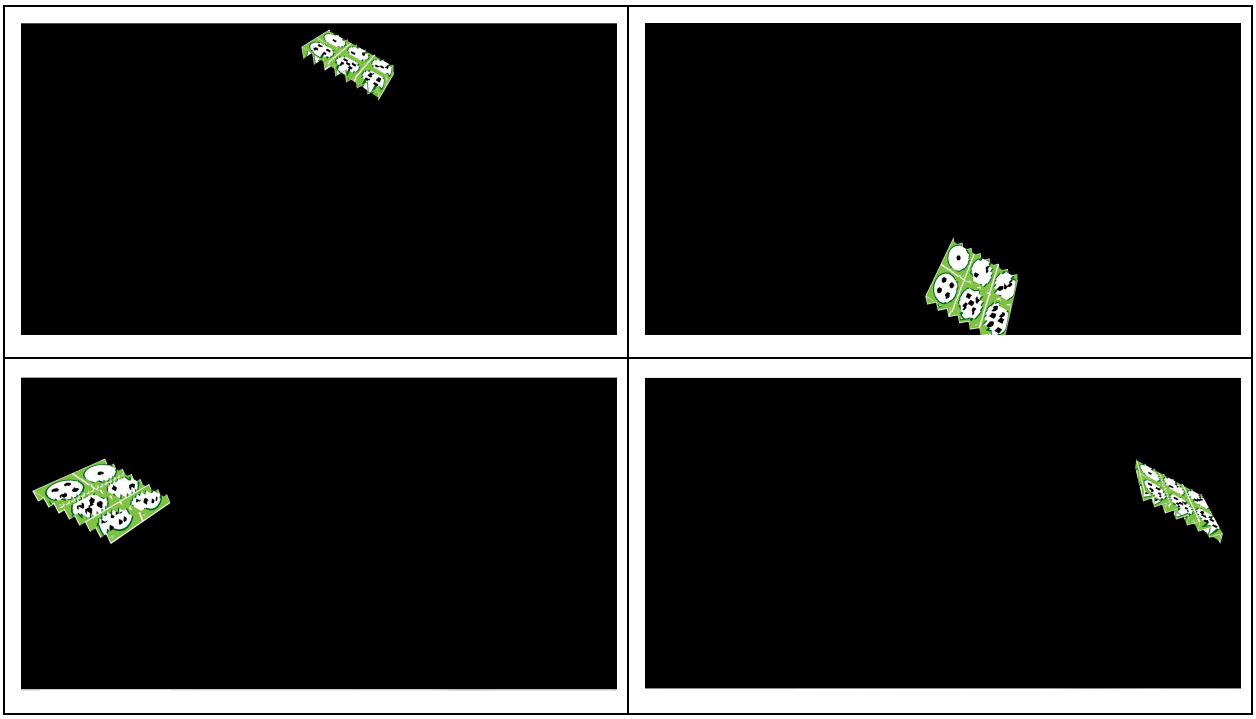
Question 2 création de la surface et plaquage de la texture



Question 4.Modification de l’altitude Z pour réaliser un relief



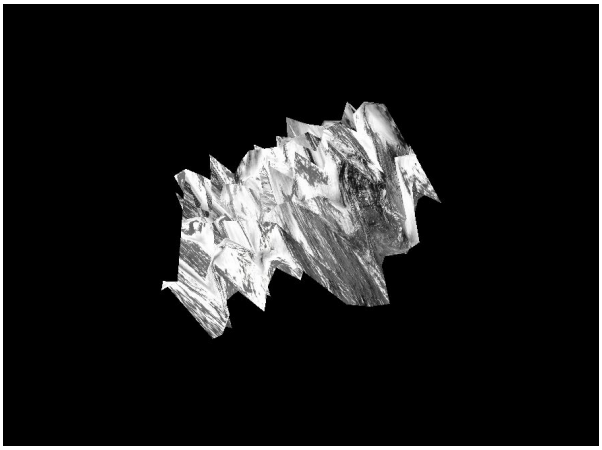
Movement de la camera (haut ,bas,gauche et droite)

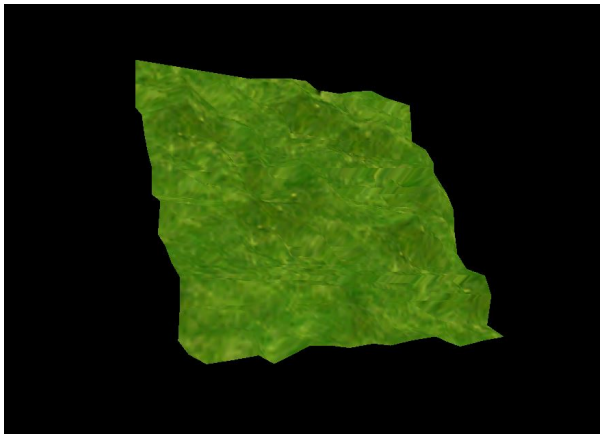


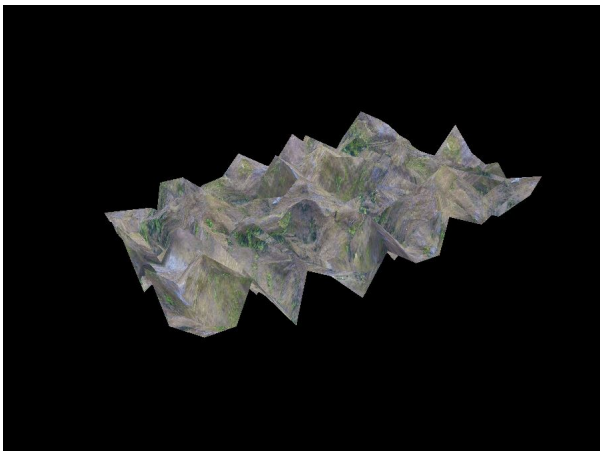
Question 5

Lecture d'un un height pour que l'altitude en chaque point soit fourni par ce dernier

```
int n = columns;
int m=rows;
// For plane, we need 8 vertices on the same plane z=0
VertexData vertices[n*n] ;
 QImage img = QImage("../cube/snowrocks.png");

for(int j = 0; j < m; j++) {
    for(int i = 0; i < n; i++) {

        float z = img.pixelColor(i*img.width()/(float)n,
                                 j*img.height()/(float)n).black()/512.f;

        vertices[i+j*n] = {QVector3D((float)i/n, (float)j/n, z), QVector2D((float)i/(n-1),(float) j/(n-1))};

    }
    //std::cout << std::endl;
}
```







Question 6

Pour faire tourner la surface autour de son origine on modifie la position des sommets Pour que l'origine de la surface soit au centre

```
for(int j = 0; j < m; j++) {
    for(int i = 0; i < n; i++) {
        vertices[i+j*n] = {QVector3D((float)(i-n/2)/n, (float)(j-n/2)/n, (i%2)/10.0), QVector2D((float)i/(n-1),(float) j/(n-1))};

    }
    //std::cout << std::endl;
}
```

Et pour faire tourner la surface de manière constante on décrémente plus angular speed dans la méthode **timerEvent** de la classe **mainWidget**

```
void MainWidget::timerEvent(QTimerEvent *)
{

        rotation = QQuaternion::fromAxisAndAngle(rotationAxis, angularSpeed) * rotation;

        // Request an update
        update();

}
```