



HMIN31:Moteurs de jeux



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

Préparé par:

Irudukunda Valentin

Prof:

NOURA faraj

September 2020

Question 1

a) Fonctionnement des méthodes de dessin

La classe qui s'occupe de dessiner c'est la classe **geometryengine**
Qui contient deux méthodes principales et deux variables Vertex buffer Objects;

Variable:

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

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

Méthode:

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

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

a) Fonctionnement du mécanisme et fonction permettant la transmission des mises à jour à partir des entrées utilisateur

Les trois méthodes principales gérant les entrées utilisateur se trouvent dans la classe **mainwindow** :

mousePressEvent : permettant de sauvegarder la position actuelle de la souris une fois cliquée

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

mouseReleaseEvent :permettant de savoir quand l'utilisateur à relâcher la souris pour appliquer les éventuels transformation nécessaire dans l'exemple du cube on déterminer 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->enableVertexAttribArray(vertexLocation);
    program->setAttributeBuffer(vertexLocation, GL_FLOAT, offset, 3, sizeof(VertexData));

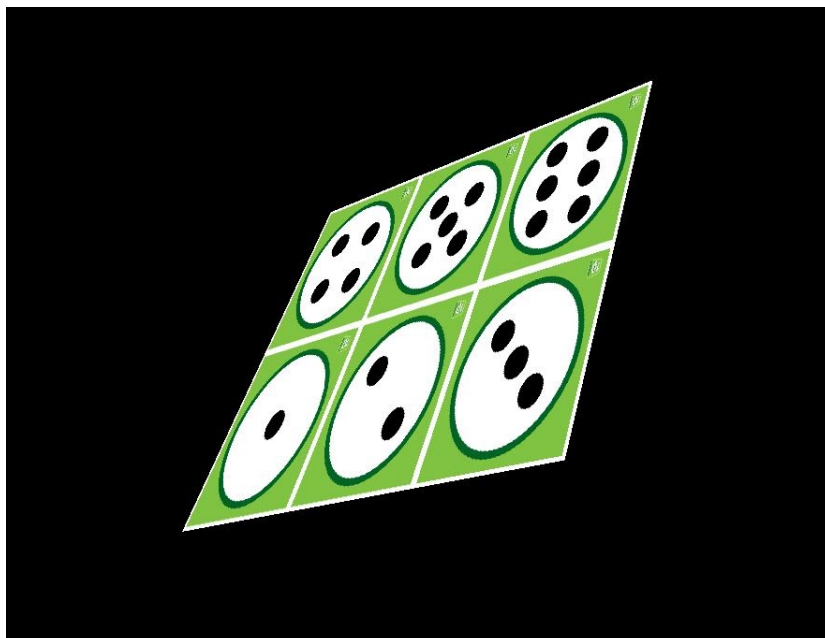
    // Offset for texture coordinate
    offset += sizeof(QVector3D);

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

    glDrawElements(GL_TRIANGLE_STRIP, 1350, GL_UNSIGNED_SHORT, 0);
}
```

Resultat

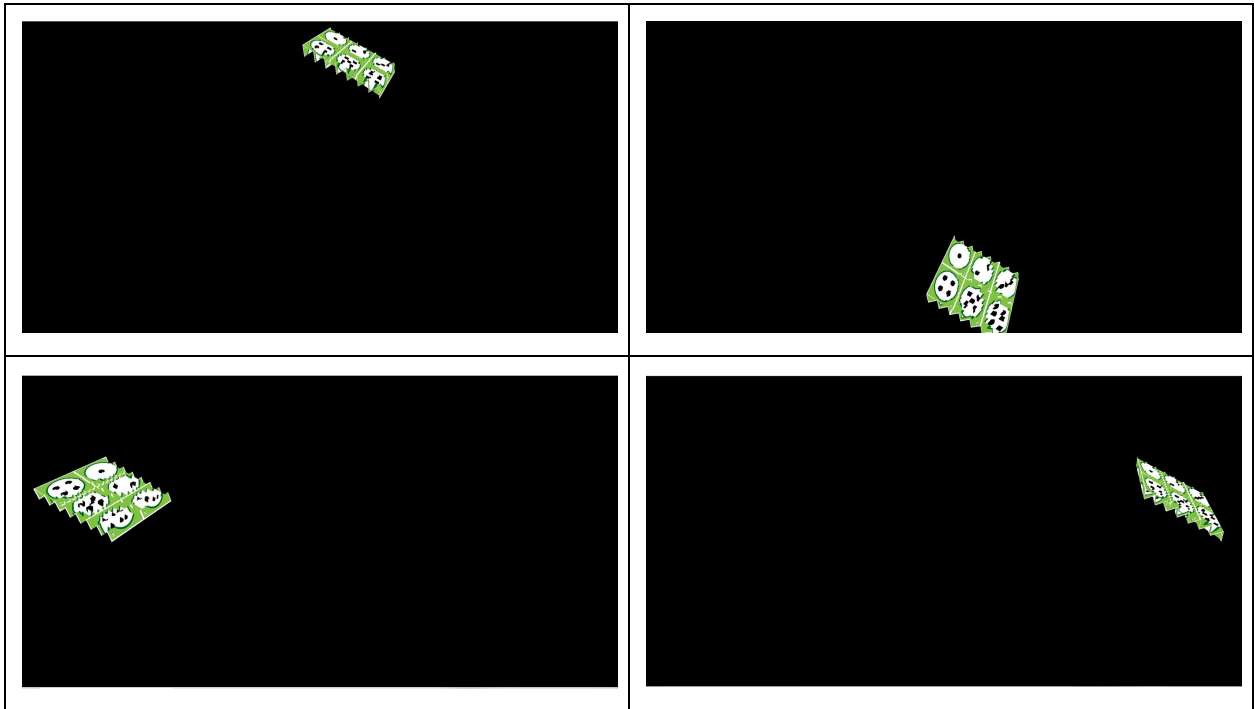
Question 2 création de la surface et plaquage de la texture



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;
}
```

Pour pouvoir utiliser trois texture sur un même terrain on les initialise et on les active

INITIALISATION DES TEXTURES

```
void MainWindow::initTextures()
{
    // Load cube.png image
    texture = new QOpenGLTexture(QImage("../cube/grass.png"));

    // Set nearest filtering mode for texture minification
    texture->setMinificationFilter(QOpenGLTexture::Nearest);

    // Set bilinear filtering mode for texture magnification
    texture->setMagnificationFilter(QOpenGLTexture::Linear);

    // Wrap texture coordinates by repeating
    // f.ex. texture coordinate (1.1, 1.2) is same as (0.1, 0.2)
    texture->setWrapMode(QOpenGLTexture::Repeat);

    // Load cube.png image
    texture1 = new QOpenGLTexture(QImage("../cube/rock.png"));

    // Set nearest filtering mode for texture minification
    texture1->setMinificationFilter(QOpenGLTexture::Nearest);

    // Set bilinear filtering mode for texture magnification
    texture1->setMagnificationFilter(QOpenGLTexture::Linear);

    // Wrap texture coordinates by repeating
    // f.ex. texture coordinate (1.1, 1.2) is same as (0.1, 0.2)
    texture1->setWrapMode(QOpenGLTexture::Repeat);

    // Load cube.png image
    texture2 = new QOpenGLTexture(QImage("../cube/snowrocks.png"));

    // Set nearest filtering mode for texture minification
    texture2->setMinificationFilter(QOpenGLTexture::Nearest);

    // Set bilinear filtering mode for texture magnification
    texture2->setMagnificationFilter(QOpenGLTexture::Linear);

    // Wrap texture coordinates by repeating
    // f.ex. texture coordinate (1.1, 1.2) is same as (0.1, 0.2)
    texture2->setWrapMode(QOpenGLTexture::Repeat);
}
```

ACTIVATION DE TEXTURE

```
// Activation de Trois Texture
glActiveTexture(GL_TEXTURE0);
texture->bind();
glActiveTexture(GL_TEXTURE1);
texture1->bind();

glActiveTexture(GL_TEXTURE2);
texture2->bind();

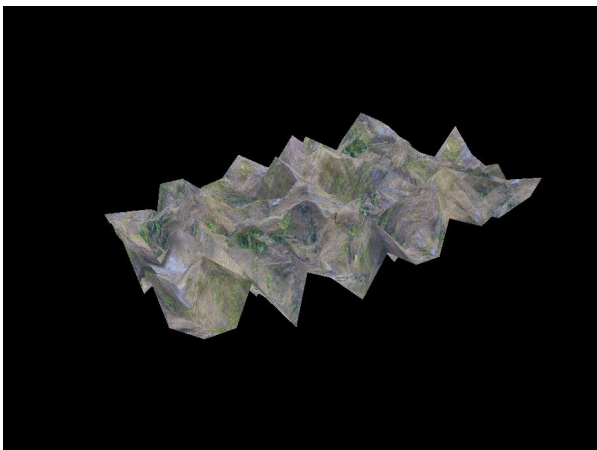
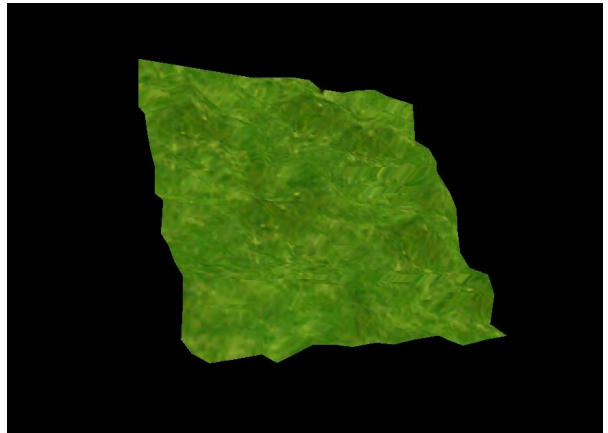
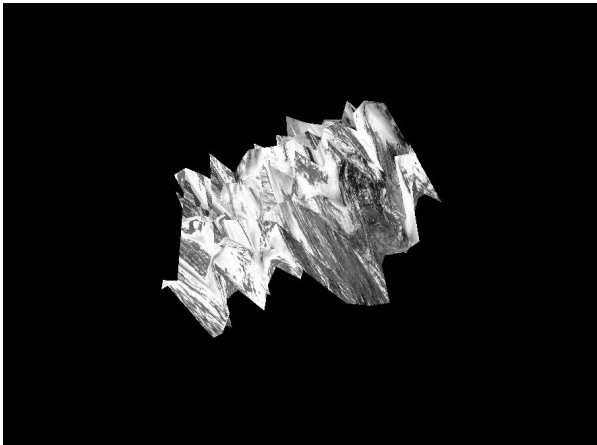
// Lier les texture au fragment shader
program.setUniformValue("texture", 0);
program.setUniformValue("texture1", 1);
program.setUniformValue("texture2", 2);
```

Et après on modifie le fragment shader pour appliquer la texture selon la coordonnée Z du heightMap

```
#version 330 core
#ifdef GL_ES
// Set default precision to medium
precision mediump int;
precision mediump float;
#endif

uniform sampler2D texture;
uniform sampler2D texture1;
uniform sampler2D texture2;
varying vec2 v_texcoord;
in vec4 poz;
out vec4 color;

//! [0]
void main()
{
    // Set fragment color from texture
    if(poz.z>0.4){
        color = texture2D(texture, v_texcoord);
    }
    else{
        if(poz.z<=0.4&&poz.z>0.32)
        {color = texture2D(texture1, v_texcoord);}else{
            color = texture2D(texture2, v_texcoord);
        }
    }
}
```

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] = {Quaternion3D((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 MainWindow::timerEvent(QTimerEvent *)  
{  
    rotation = Quaternion::fromAxisAndAngle(rotationAxis, angularSpeed) * rotation;  
    // Request an update  
    update();  
}
```