# Moteurs de Jeux - Compte rendu TP1

## Question 1

La classe MainWidget gère tout ce qui est en rapport avec l'initialisation d'OpenGL et les évènements souris.
C'est-à-dire :

- gestion de l'évènement de clic et relâchement de la souris
- gestion d'un timer
- initialisation de GL, des shaders et des textures
- gestion de redimensionnement de la fenêtre

La classe GeometryEngine gère tout ce qui est en rapport avec un cube.
C'est-à-dire :

- créer le cube avec tous ses sommets
- stocker les indices des sommets pour chaque triangle composant le cube
- allouer la mémoire nécessaire au stockage de toutes ses informations
- afficher le cube avec des triangles grâce aux indices stockés précedemment

Le fichier fshader.glsl sert à gérer la texture des fragments/pixels correspondant à l'objet.

Le fichier vshader.glsl sert à gérer la position et la texture des sommets.

## Question 2

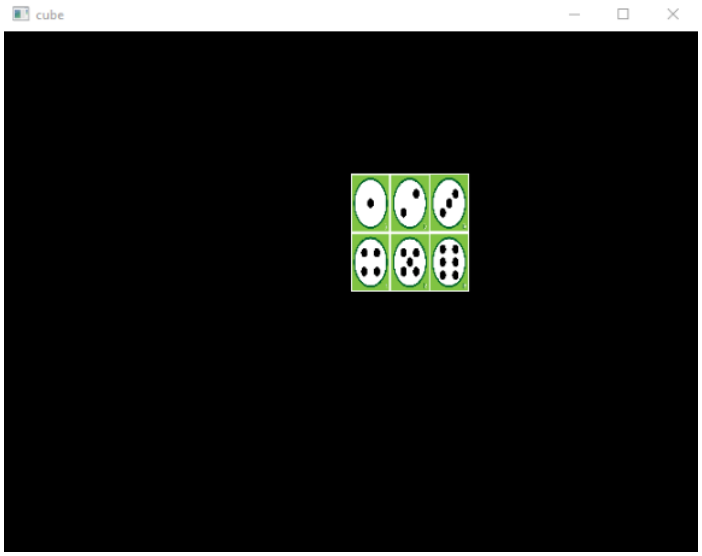
La méthode *void GeometryEngine::initCubeGeometry()* initialise les coordonnées du cube.
Dans un premier temps les différentes faces du cube, 6 au total, sont créer grâce aux sommets, 8 au total. Au total il y aura 24 sommets de créer, mais seulement 8 différents, car chaque sommet est partagé avec 3 faces du cube. Les Vector3D servent donc à créer les 24 sommets alors que les Vector2D servent à positionner les 6 textures 2D, correspondant aux 6 faces, du cube. Tout cela est stocké dans un tableau.
Dans un second temps on stocke les indices des sommets correspondant à chaque face carrée du cube, en séparant chaque carré en deux triangles, dans un autre tableau.
Pour finir, on stocke les deux tableaux dans deux buffers VBO différents pour lesquels on alloue la mémoire nécessaire.

La méthode *void GeometryEngine::drawCubeGeometry(QOpenGLShaderProgram *program)* dessine le cube.
Dans un premier temps les deux buffers VBO utilisés précedemment dans la méthode *initCubeGeometry()* sont chargés en mémoire.
Ensuite, OpenGL est configuré de sorte à ce qu'il trouve les informations nécessaires aux positionnement des sommets et des textures.
Enfin, la méthode *dlDrawElements* est configurée afin qu'OpenGL sait qu'il faut dessiner des triangles (GL_TRIANGLE_STRIP).

# Question 3

```
void GeometryEngine::initPlaneGeometry()
{
    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++){
            vertices[index] = {QVector3D((float)x/nbVertecesInRow,(float)y/nbVertecesInRow,0), 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));
}
```

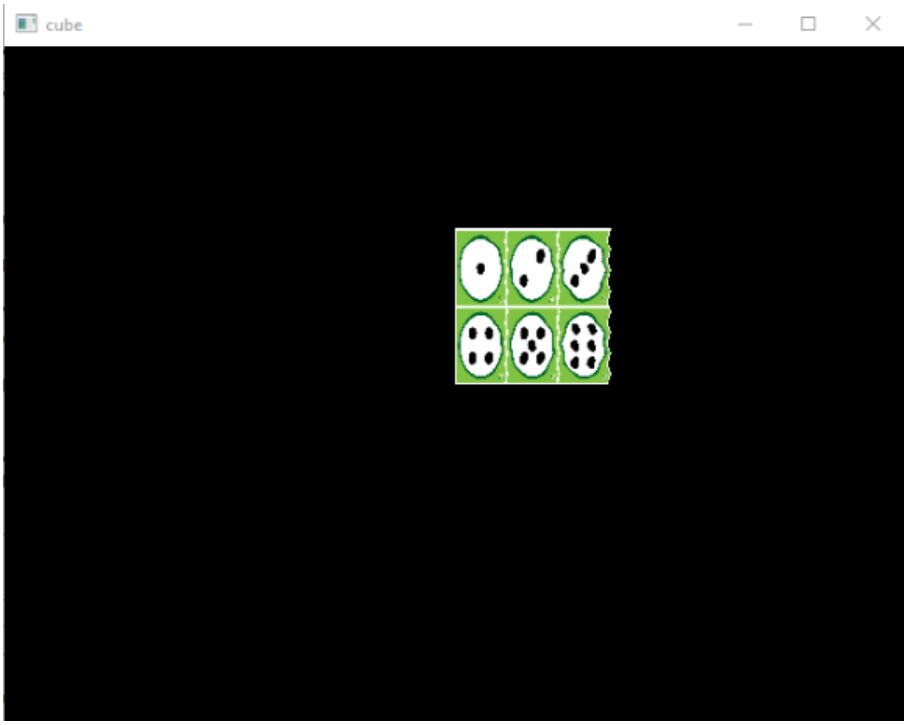
```
void GeometryEngine::drawPlaneGeometry(QOpenGLShaderProgram *program)
{
    unsigned short nbVertecesInRow = 16;
    // Tell OpenGL which VBOs to use
    arrayBuf.bind();
    indexBuf.bind();

    // 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));                    ⚠ implicit conversion

    // 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));                  ⚠ implicit conversion

    // 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);
}
```

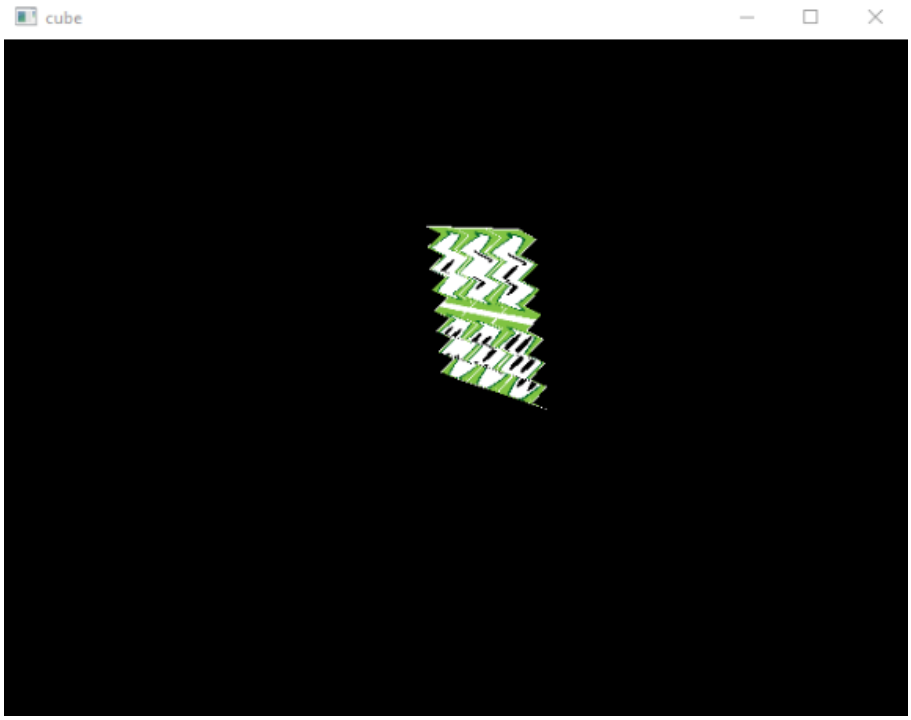



## Question 4

```
void GeometryEngine::initPlaneGeometry()
{
    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++){
            vertices[index] = {QVector3D((float)x/nbVertecesInRow,(float)y/nbVertecesInRow,(y%2)/10.0f), 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));
}
```

J’ai simplement rajouté un calcul de modulo sur la composante z du vecteur de positionnement des sommets de chaque triangle.


cube

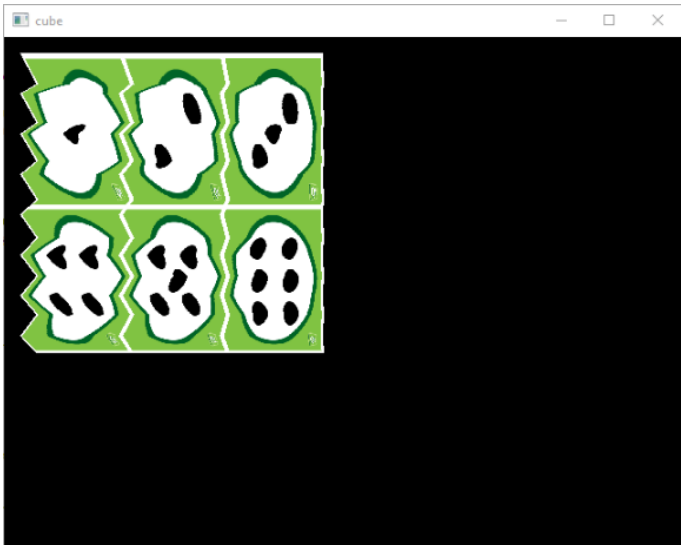

cube

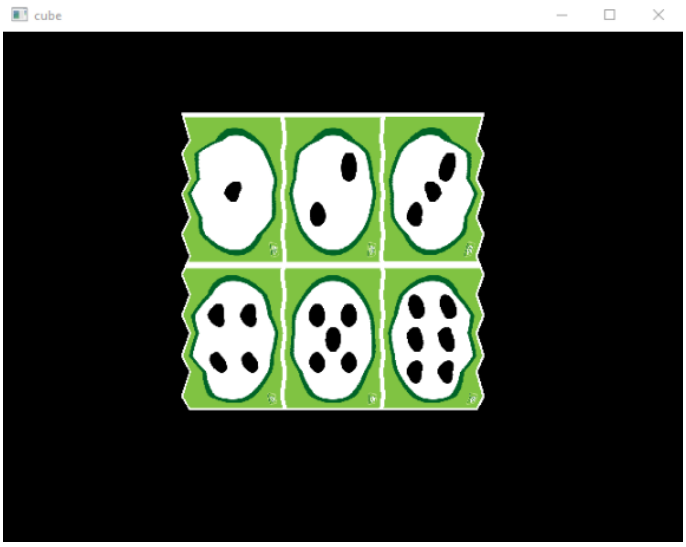
```
void MainWidget::wheelEvent(QWheelEvent *event){
    int numDegrees = event->delta() / 8;
    cam.setZ(cam.z()+numDegrees*0.01f);
    update();
}

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);
        break;
        case Qt::Key_Down :
            cam.setY(cam.y()-1.0f/10);
        break;
    }
    update();
}
```

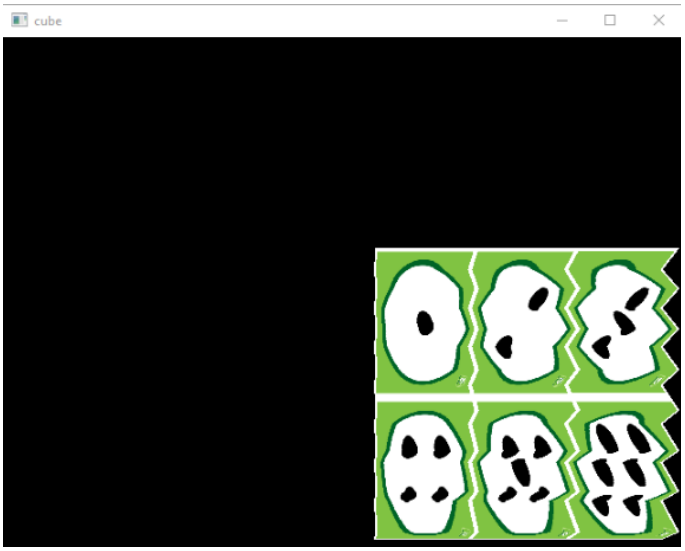
La molette de la souris permet de zoomer et de dézoomer sur l'objet.
Les flèches du clavier permettent de déplacer la caméra sur la scène.

```
void MainWidget::paintGL()
{
    // Clear color and depth buffer
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    texture->bind();

//! [6]
    // Calculate model view transformation
    QMatrix4x4 matrix;
    matrix.translate(0.0, 0.0, 3.0);
    // On applique les modifications apportées par l'utilisateur
    matrix.translate(cam);
    matrix.rotate(rotation);

    // Set modelview-projection matrix
    program.setUniformValue("mvp_matrix", projection * matrix);
//! [6]

    // Use texture unit 0 which contains cube.png
    program.setUniformValue("texture", 0);

    // Draw cube geometry
    //geometries->drawCubeGeometry(&program);

    // Draw plane geometry
    geometries->drawPlaneGeometry(&program);
}
```

Dans la fonction paintGL on applique la matrice de transformation résultante.


cube


cube


cube