Moteur de jeux – TP1

Exercice 1 :

- La classe **MainWidget** sert à afficher la fenêtre, créer un context OpenGL, afficher le rendu, gère les différents éventements (Souris – Temps). Cette classe va aussi initialiser les shaders et charger/appliquer les textures.
- La classe **GeometryEngine** va permettre de créer et dessiner un mesh 3D (ici un cube).
- Le fichier **vshader.glsl** représente le *vertex shader* qui sera chargé et appliqué sur les sommets du mesh.
- Le fichier **fshader.glsl** représente le *fragment/pixel shader* qui sera chargé et appliqué sur les pixels du rendu.

Exercice 2 :

- `void GeometryEngine::initCubeGeometry()`
  - Tout d'abords, cette méthode va créer un tableau de sommets pour chaque face d'un cube.
  - On va maintenant créer un tableau d'indice où on va spécifier dans quel ordre on doit placer les différents sommets pour former les faces du cubes (sachant que l'on travaille avec des triangles, donc 2 triangles par faces).
  - Enfin, on indique à OpenGL que nous allons utiliser les buffers/VBO (avec `arrayBuf.bind()`), et on alloue à chacun de ces buffers, le tableau auquel ils sont destinés.

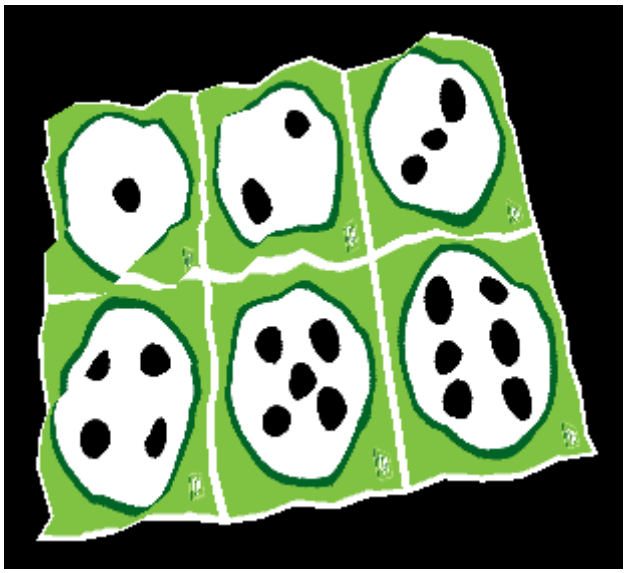
- `void GeometryEngine::drawCubeGeometry(...)`
  - On initialise les buffers/VBO.
  - On indique à OpenGL comment trouver la position des sommets, on pourra accèder à ces informations dans le shader avec le nom indiqué : `"a_texcoord"`.
  - On réalise la même chose mais avec les coordonnés de la texture.
  - Enfin, on demande à OpenGL de dessiner/traiter les éléments.

Exercice 3 :

Pour créer un plan avec 16 x 16 sommets, j'ai utiliser l'algorithme suivant (page suivante) :*// On dessine les 15 lignes de rectangles*

```
for(uint y = 0; y < 16; ++y)
{
    // On définit nos sommets
    for(uint x = 0; x < 16; ++x)
    {
        float posX = 0.125f * (float)x - 1.f;
        float posY = 0.125f * (float)y - 1.f;
        float posZ = (-1.f + (float)(rand()%200) / 200.f) * 0.2f;
        vertices[x + y * 16] = {QVector3D(posX, posY, posZ),
QVector2D((float)x / 15.0f, (float)y / 15.0f)};
    }
}
// On trouve les indices
for(uint y = 0; y < 15; ++y) // 15 lignes de rectangle
{
    // On définit nos sommets
    for(uint x = 0; x < 15; ++x) // 15 rectangle par lignes
    {
        indices[x * 6 + y * 15 * 6]     = (x + 1) + (y + 1) * 16;
        indices[x * 6 + y * 15 * 6 + 1] = x + (y + 1) * 16;
        indices[x * 6 + y * 15 * 6 + 2] = x + y * 16;
        indices[x * 6 + y * 15 * 6 + 3] = x + y * 16;
        indices[x * 6 + y * 15 * 6 + 4] = (x + 1) + y * 16;
        indices[x * 6 + y * 15 * 6 + 5] = (x + 1) + (y + 1) * 16;
    }
}
```

Voici le résultat (la déformation est du haut relief demandé à la question 4) :



Pour dessiner les triangles, je n'utilise pas la méthode GL_TRIANGLE_STRIP mais :

```
glDrawElements(GL_TRIANGLES, 15 * 15 * 6, GL_UNSIGNED_SHORT, 0);
```

Exercice 4 :

Pour ajouter du relief, j'utilise une fonction C pour créer un aléatoire, que je contrôle pour avoir un résultat décent :

```
float posZ = (-1.f + (float)(rand()%200) / 200.f) * 0.2f;
```

Pour zoomer et dézoomé sur mon objet, je surcharge la méthode keyPressEvent avec le code suivant (mainwidget) :

```
void MainWidget::keyPressEvent(QKeyEvent *event)
{
    float speed = 0.1f;
    // On avance
    switch(event->key())
    {
        case Qt::Key_Up:
            camEye.setZ(camEye.z() + speed);
            break;
        case Qt::Key_Down:
            camEye.setZ(camEye.z() - speed);
            break;
    }
    projection.lookAt(camEye, camCenter, camUp);
    update();
}
```

camEye, camCenter et camUp sont des QVector3D définit comme suit :

- camEye : 0, 0, -3.f pour avoir assez de recule pour voir l'objet.
- camCenter : à 0 pour dire que l'on pointe au centre de la scène.
- camUp : 0, 1, 0 qui permet de placer notre caméra droite.

Il suffit de donner ces vecteurs 3D à la méthode lookAt de projection pour appliquer l'effet, sans oublier d'update() notre scène.

Un problème visuel survient cependant, lorsque l'on clique sur l'une des touches pour déplacer la caméra, 1 image sur 2 est vide (noire). Si on laisse appuyé sur le bouton, alors on voit un clignotement, mais on voit bien la caméra qui s'avance vers l'objet.