HAI911I Développement d'applications interactives

Adrien Houle

Septembre 2023

Dépot de code : `https://github.com/HouleAdrien/prograInteractiveM2imagine/tree/main/TP1`

1 Question 1

Analyser le code et la structure du programme. Familiarisez avec l'application Qt. Vous constatez qu'il y a des sliders à droite. Faites en sorte que les changements de sliders fassent tourner l'objet en ajoutant des signaux et des slots et en les connectant. Faites en sorte que les sliders soient mis à jour lorsque l'utilisateur fait tourner l'objet.

Pour connecter les sliders à l'objet j'ai tout d'abord rajouter les slots et signals dans le glwidget.h.

```
public slots:
    void setXRotation(int angle);
    void setYRotation(int angle);
    void setZRotation(int angle);
    void cleanup();

signals:
    void XRotationChanged(int angle);
    void YRotationChanged(int angle);
    void ZRotationChanged(int angle);
```

Figure 1: Slots et signals

Après j'ai rajouté les emits dans les différentes fonctions de rotation.

```
void GLWidget::setXRotation(int angle)
{
    qNormalizeAngle(angle);
    if (angle != m_xRot) {
        m_xRot = angle;
        emit XRotationChanged(angle);

        update();
    }
}
```

Figure 2: Exemple d'ajout de l'emit sur la rotation x

Puis pour finir j'ai connecter dans les deux sens mon slider et mes fonctions.

```
connect(xSlider, &QSlider::valueChanged, glWidget, &GLWidget::setXRotation);
connect(glWidget, &GLWidget::XRotationChanged, xSlider, &QSlider::setValue);

connect(ySlider, &QSlider::valueChanged, glWidget, &GLWidget::setYRotation);
connect(glWidget, &GLWidget::YRotationChanged, ySlider, &QSlider::setValue);

connect(zSlider, &QSlider::valueChanged, glWidget, &GLWidget::setZRotation);
connect(glWidget, &GLWidget::ZRotationChanged, zSlider, &QSlider::setValue);
```

Figure 3: Mise en place des connexions

2 Question 2

Créer une classe maillage. Y ajouter un vertex buffer et un index buffer. Remplir les buffers pour créer un cube (maillage indexé). Affichez le.

Pour faire mon cube j'ai simplement suivi ce tuto et je l'ai appliqué a une classe.

`https://doc.qt.io/qt-6/qtopengl-cube-example.html`
Tout d'abord la déclaration de ma classe dans laquel on peut voir mes deux buffers membres.

```
#ifndef MAILLAGE_H
#define MAILLAGE_H
#include <QOpenGLFunctions>
#include <QOpenGLShaderProgram>
#include <QOpenGLBuffer>

class Maillage : protected QOpenGLFunctions
{
public:
    Maillage();
    virtual ~Maillage();
    void drawCubeGeometry(QOpenGLShaderProgram *program);

private:
    void initCubeGeometry();

    QOpenGLBuffer arrayBuf;
    QOpenGLBuffer indexBuf;
};

#endif // MAILLAGE_H
```

Figure 4: Déclaration classe maillage

Lors de la construction de ma classe je crée mes buffers et les initialise avec ma géométrie de cube.

```
struct VertexData
{
    QVector3D position;
    QVector2D texCoord;
};

Maillage::Maillage() : indexBuf(QOpenGLBuffer::IndexBuffer)
{
    initializeOpenGLFunctions();

    // Generate 2 VBOs
    arrayBuf.create();
    indexBuf.create();

    // Initializes cube geometry and transfers it to VBOs
    initCubeGeometry();
}
```

Figure 5: Constructeur

```
void Maillage::initCubeGeometry(){
    VertexData vertices[] = {
        // Vertex data for face 0
        {QVector3D(-0.1f, -0.1f,  0.1f), QVector2D(0.0f, 0.0f)},  // v0
        {QVector3D( 0.1f, -0.1f,  0.1f), QVector2D(0.33f, 0.0f)}, // v1
        {QVector3D(-0.1f,  0.1f,  0.1f), QVector2D(0.0f, 0.5f)},  // v2
        {QVector3D( 0.1f,  0.1f,  0.1f), QVector2D(0.33f, 0.5f)}, // v3

        // Vertex data for face 1
        {QVector3D( 0.1f, -0.1f,  0.1f), QVector2D( 0.0f, 0.5f)}, // v4
        {QVector3D( 0.1f, -0.1f, -0.1f), QVector2D(0.33f, 0.5f)}, // v5
        {QVector3D( 0.1f,  0.1f,  0.1f), QVector2D(0.0f, 1.0f)},  // v6
        {QVector3D( 0.1f,  0.1f, -0.1f), QVector2D(0.33f, 1.0f)}, // v7

        // Vertex data for face 2
        {QVector3D( 0.1f, -0.1f, -0.1f), QVector2D(0.66f, 0.5f)}, // v8
        {QVector3D(-0.1f, -0.1f, -0.1f), QVector2D( 1.0f, 0.5f)},  // v9
        {QVector3D( 0.1f,  0.1f, -0.1f), QVector2D(0.66f,  1.0f)}, // v10
        {QVector3D(-0.1f,  0.1f, -0.1f), QVector2D( 1.0f,  1.0f)},  // v11

        // Vertex data for face 3
        {QVector3D(-0.1f, -0.1f, -0.1f), QVector2D(0.66f, 0.0f)}, // v12
        {QVector3D(-0.1f, -0.1f,  0.1f), QVector2D( 1.0f, 0.0f)},  // v13
        {QVector3D(-0.1f,  0.1f, -0.1f), QVector2D(0.66f, 0.5f)}, // v14
        {QVector3D(-0.1f,  0.1f,  0.1f), QVector2D( 1.0f, 0.5f)},  // v15

        // Vertex data for face 4
        {QVector3D(-0.1f, -0.1f, -0.1f), QVector2D(0.33f, 0.0f)}, // v16
        {QVector3D( 0.1f, -0.1f, -0.1f), QVector2D(0.66f, 0.0f)}, // v17
        {QVector3D(-0.1f, -0.1f,  0.1f), QVector2D(0.33f, 0.5f)}, // v18
        {QVector3D( 0.1f, -0.1f,  0.1f), QVector2D(0.66f, 0.5f)}, // v19

        // Vertex data for face 5
        {QVector3D(-0.1f,  0.1f,  0.1f), QVector2D(0.33f, 0.5f)}, // v20
        {QVector3D( 0.1f,  0.1f,  0.1f), QVector2D(0.66f, 0.5f)}, // v21
        {QVector3D(-0.1f,  0.1f, -0.1f), QVector2D(0.33f,  1.0f)}, // v22
        {QVector3D( 0.1f,  0.1f, -0.1f), QVector2D(0.66f,  1.0f)}  // v23
    };

    GLushort indices[] = {
         0,  1,  2,  3,  3,     // Face 0 - triangle strip ( v0,  v1,  v2,  v3)
         4,  4,  5,  6,  7,  7, // Face 1 - triangle strip ( v4,  v5,  v6,  v7)
         8,  8,  9, 10, 11, 11, // Face 2 - triangle strip ( v8,  v9, v10, v11)
        12, 12, 13, 14, 15, 15, // Face 3 - triangle strip (v12, v13, v14, v15)
        16, 16, 17, 18, 19, 19, // Face 4 - triangle strip (v16, v17, v18, v19)
        20, 20, 21, 22, 23      // Face 5 - triangle strip (v20, v21, v22, v23)
    };

    // Transfer vertex data to VBO 0
    arrayBuf.bind();
    arrayBuf.allocate(vertices, 24 * sizeof(VertexData));

    // Transfer index data to VBO 1
    indexBuf.bind();
    indexBuf.allocate(indices, 34 * sizeof(GLushort));

}
```

Figure 6: Initialisation

Puis ma fonction de draw que j'apelle dans mon paintGL, servant a passer ma géométrie au programme opengl.

```
void Maillage::drawCubeGeometry(QOpenGLShaderProgram *program){
    // 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("vertex");
       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("normal");
       program->enableAttributeArray(texcoordLocation);
       program->setAttributeBuffer(texcoordLocation, GL_FLOAT, offset, 2, sizeof(VertexData));

       // Draw cube geometry using indices from VBO 1
       glDrawElements(GL_TRIANGLE_STRIP, 34, GL_UNSIGNED_SHORT, nullptr);

       arrayBuf.release();
       indexBuf.release();
}
```

Figure 7: Initialisation

En résultat je peux observer ce cube à l'écran.

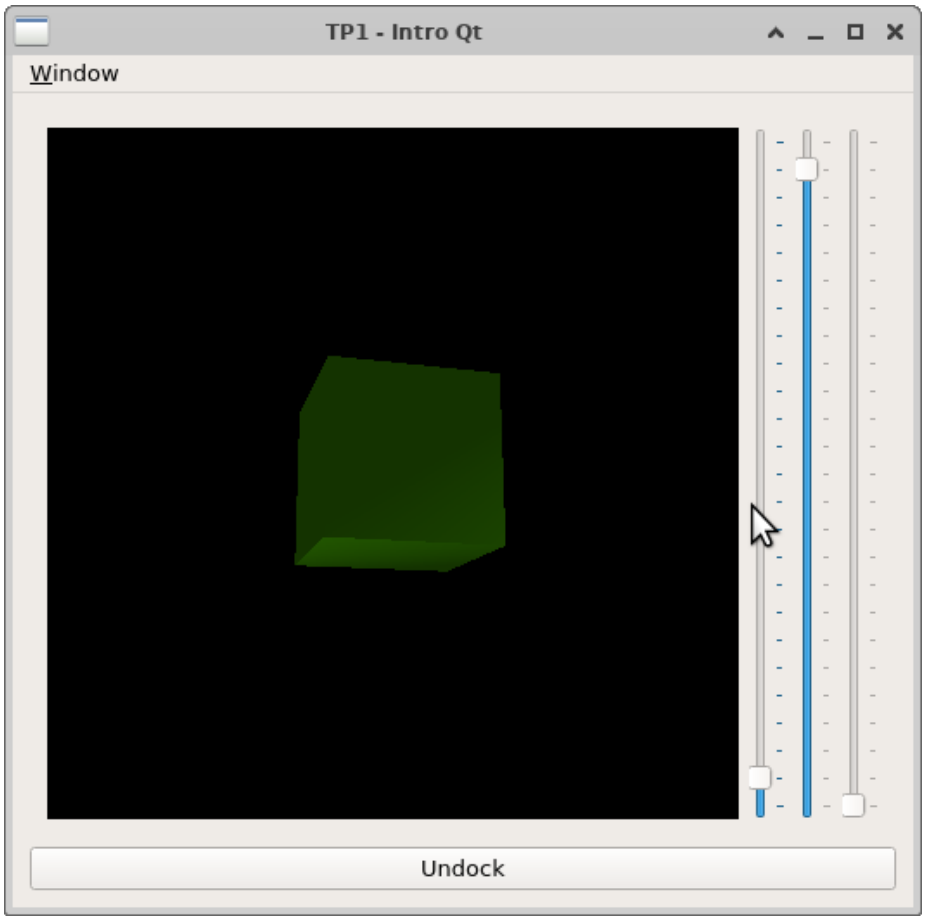

Figure 8: Cube dans la scène opengl

3 Question 3

Ajouter un item au menu pour charger un maillage. Mettre la géométrie chargée à partir d'un ficher dans une instance de votre classe maillage et l'afficher.

Pour cette partie je n'ai pas réussi à implémenter une classe fonctionnelle chargeant depuis un fichiers je me suis donc arrêté au cube. (Le code du github s'arrête au cube je n'ai pas push mon code non fonctionnel)