Université de la Rochelle
Licence d'Informatique - IHM (INFO132601)

TP 4 - *Appel de fonctions 3D OpenGL, utilisation de QtDesigner*

© B. Besserer, R. Péteri

Année universitaire 2016-2017

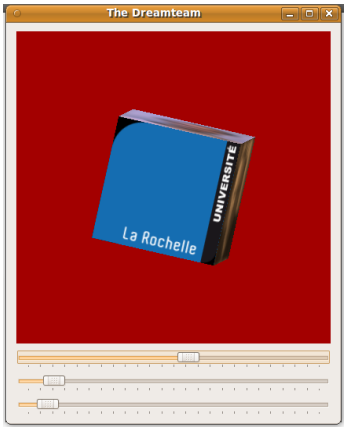
VERSION AVEC ELEMENTS DE CORRECTION

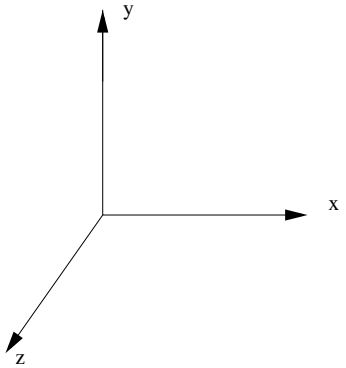
1 Cahier des charges

Le but de ce TP est d'écrire une application complète sous Q4t utilisant des fonctions OpenGL pour l'affichage graphique 3D :
"OpenGL is a standard API for rendering 3D graphics.
OpenGL only deals with 3D rendering and provides little or no support for GUI programming issues. The user interface for an OpenGL application must be created with another toolkit, such as Motif on the X platform, Microsoft Foundation Classes (MFC) under Windows, or Qt on both platforms.*
The Qt OpenGL module makes it easy to use OpenGL in Qt applications. It provides an OpenGL widget class that can be used just like any other Qt widget, except that it opens an OpenGL display buffer where you can use the OpenGL API to render the contents." extrait de l'aide de Qt4.
Les fonctions OpenGL seront données car nous nous focaliserons sur leur utilisation sous Qt au travers de la classe QGLWidget.
Il s'agit de pouvoir réaliser de plusieurs façons la rotation d'un cube 3D autour de ses trois axes x, y et z. Ce cube est un objet 3D OpenGL sur lesquel sont plaquées des textures (ce sont des images et elles constituent les *ressources* du programme).
Ce cube devra tourner sur ses 3 axes :

- grâce à 3 curseurs de la classe QScale
- de manière continue grâce à un timer
- grâce à l'interaction avec les boutons de la souris

Ce TP est prévu pour être réalisé en Qt sous LINUX avec une carte graphique supportant l'accélération OpenGL.
La copie d'écran suivante présente l'interface à obtenir (à gauche) et le système de coordonnée pour l'affichage 3D (à droite) :





Un code source de départ est disponible sous Moodle au téléchargement (`tp4_base.tgz`), ainsi les images ressources servant de textures pour les faces du cube.

Décompresser cette archive dans votre répertoire de travail (`tar xzf tp4_base.tgz`). Sous Windows, il se peut que vous ayez à rajouter dans votre `.pro` :`LIBS+=lopengl32`

2 Fonctions OpenGL sous Qt

2.1 La classe glwidget

Commencez tout d'abord par compiler le programme de base. Vous devriez voir le logo de l'université qui s'affiche au milieu de la fenêtre. Il s'agit en fait de la vue de face du cube qui tournera autour de ses axes.
Ouvrez ensuite les fichiers glwidget.h et glwidget.cpp qui constituent la définition et l'implémentation de la classe glwidget gérant la fenêtre d'affichage 3D.
glwidget est une classe dérivée de QGLWidget qui sert à l'affichage 3D en utilisant l'acceleration OpenGL de la carte graphique.

Fichier glwidget.h :

```
class GLWidget : public QGLWidget
{
    Q_OBJECT

public:
    GLWidget(QWidget *parent = 0);
    ~GLWidget();

    QSize minimumSizeHint() const;
    QSize sizeHint() const;
    void setClearColor(const QColor &color);


protected:
    void initializeGL();
    void paintGL();
    void resizeGL(int width, int height);


private:
    GLuint makeObject();
    QColor clearColor;
    static GLuint sharedObject;

};
```

Fichier glwidget.cpp :

```
}
#include <QtGui>
#include <QtOpenGL>
#include <math.h>
#include "glwidget.h"

GLuint GLWidget::sharedObject = 0;

GLWidget::GLWidget(QWidget *parent)
    : QGLWidget(parent)
{ clearColor = Qt::black;}

GLWidget::~GLWidget()
{    makeCurrent();
     glDeleteLists(sharedObject,1); }

QSize GLWidget::minimumSizeHint() const
{  return QSize(150, 150);}

QSize GLWidget::sizeHint() const
{    return QSize(400, 400);}

void GLWidget::setClearColor(const QColor &color)
{   clearColor = color;
    updateGL();}
```

```
// Initialisation de l'affichage OpenGL
void GLWidget::initializeGL()
{
    if (!sharedObject)    sharedObject = makeObject();
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_CULL_FACE);
    glEnable(GL_TEXTURE_2D);
}

// Fonction de dessin de la fenêtre OpenGL
void GLWidget::paintGL()
{
    qglClearColor(clearColor);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glTranslated(0.0, 0.0, -10.0);
    glCallList(sharedObject);
}

// Pour le redimensionnement de la widget d'affichage opengl
void GLWidget::resizeGL(int width, int height)
{
    int side = qMin(width, height);
    glViewport((width - side) / 2, (height - side) / 2, side, side);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-0.5, +0.5, +0.5, -0.5, 4.0, 15.0);
    glMatrixMode(GL_MODELVIEW);
}
// Création du cube avec les textures
GLuint GLWidget::makeObject()
{    static const int coords[6][4][3] = {
        { { +1, -1, -1 }, { -1, -1, -1 }, { -1, +1, -1 }, { +1, +1, -1 } },
        { { +1, +1, -1 }, { -1, +1, -1 }, { -1, +1, +1 }, { +1, +1, +1 } },
        { { +1, -1, +1 }, { +1, -1, -1 }, { +1, +1, -1 }, { +1, +1, +1 } },
        { { -1, -1, -1 }, { -1, -1, +1 }, { -1, +1, +1 }, { -1, +1, -1 } },
        { { +1, -1, +1 }, { -1, -1, +1 }, { -1, -1, -1 }, { +1, -1, -1 } },
        { { -1, -1, +1 }, { +1, -1, +1 }, { +1, +1, +1 }, { -1, +1, +1 } }
    };
    GLuint textures[6];
for (int j=0; j < 6; ++j)
 textures[j] = bindTexture(QPixmap(QString(":/images/side%1.png").arg(j + 1)),
         GL_TEXTURE_2D);

    GLuint list = glGenLists(1);
    glNewList(list, GL_COMPILE);
    for (int i = 0; i < 6; ++i) {
        glBindTexture(GL_TEXTURE_2D, textures[i]);
        glBegin(GL_QUADS);
        for (int j = 0; j < 4; ++j) {
            glTexCoord2d(j == 0 || j == 3, j == 0 || j == 1);
            glVertex3d(0.2 * coords[i][j][0], 0.2 * coords[i][j][1],
                       0.2 * coords[i][j][2]);
        }
        glEnd();
    }

    glEndList();
    return list;
}
```

2.2 Mise en place des fonctions de rotation dans les classes

On va tout d'abord implémenter les fonctions de rotation des classes GLWidget et window.
Ces fonctions seront appelées lorsque l'on souhaiterait faire tourner le cube suivant les 3 modes d'interaction (curseurs, timer et boutons de la souris).

Dans la classe GLWidget

- Déclarer en variables membres les paramètres de rotation suivant les 3 axes : xRot, yRot et zRot. Ces données membres seront initialisées à 0 dans le constructeur de la classe.
- Implémenter la fonction membre rotateBy(int xAngle, int yAngle, int zAngle) chargée d'effectuer une rotation d'angle (xRot,yRot,zRot).
  Aide OpenGL : La commande OpenGL pour dessiner est paintGL() et rafraîchir l'affichage : updateGL(). La fonction pour effectuer une rotation d'angle xRot suivant le 1er axe x est `glRotated(xRot / 16.0, 1.0, 0.0, 0.0)` et se fait pendant le dessin (la division de `xRot` par 16 permettra par la suite d'avoir une animation plus fluide....).

Dans la classe Window

Dans la classe Window, créer le slot privé rotateOneStep() qui effectue une rotation élémentaire de 32° suivant x, 32° suivant y et -16° suivant z. Ce slot sera appelé plus tard lors de la mise en place du timer.

2.3 Rotation à partir des 3 curseurs QScale

On souhaite créer les 3 curseurs associés aux rotations selon les 3 axes (voir capture d'écran).

- Créer dans la classe de l'application principale une fonction générique createSlider() qui crée et renvoie l'adresse d'un QSLider. Les QSlider auront comme options un intervalle de (0, 360), un pas de 16, ainsi que les options pour les graduations (fonctions membres de QScale) : setPageStep(15), setTickInterval(15) et setTickPosition(QSlider : :TicksRight).
- Creer dans la classe adéquate les slots associés dérivant de rotateBy(int xAngle, int yAngle, int zAngle).
- Effectuer les connections SIGNAL/SLOTS appropriées.

2.4 Rotation continue avec un timer

Mettre en place un timer (classe QTimer) pour que la rotation se fasse de manière continue à chaque signal du Timer (réglez le timer à 50 ms)

2.5 Bouton START/STOP - réglage de la vitesse de rotation

Sur le coté droit de la widget principale, rajoutez (en changeant le container) des boutons relatifs au timer : un bouton STOP pour arrêter la rotation continue, un bouton START pour la démarrer, et une spinbox pour régler la vitesse de rotation.

2.6 Rotation par interaction avec la souris

Nous allons ici mettre en place un 3ème type d'interaction pour la rotation qui utilisera la souris.

Ecrire une fonction membre void GLWidget : :mousePressEvent(QMouseEvent *event) qui récuperera dans une variable QPoint de la classe la position du curseur de la souris. Cette position "initiale" sera ensuite utilisée dans une deuxième fonction membre void GLWidget : :mouseMoveEvent(QMouseEvent *event).
La rotation se fera quand le bouton souris est maintenu enfoncé, et sera proportionnelle à (dx,dy) qui est la variation entre point initial et le point en mouvement.
La rotation sera du type : `rotateBy(8 * dy, 8 * dx, 0)` si c'est le bouton gauche qui est pressé lors du déplacement de la souris, sinon `rotateBy(8 * dy, 0, 8 * dx)` si c'est le bouton droit.

3 Intégration d'un composant crée dans QtDesigner

1. En utilisant QtDesigner, réaliser une widget rot_angles comportant 3 triplets spinbox/affichage LCD/label reliés entre eux. Chaque couple spinbox/LCD indiquera une valeur d'angle entre 0 et 360 degrés, respectivement pour l'axe x, y et z. Comme vu en cours, QtDesigner permet de réaliser des interfaces Qt de manière graphique. On prendra soin de lire le cours et l'aide de QtDesigner très bien faite.
2. Intégrer cette widget rot_angles dans votre application pour qu'elle contrôle les 3 axes de rotation de votre cube . . ....


4

Fichier glwidget.cpp:

```
#include <QtGui>
#include <QtOpenGL>

#include <math.h>

#include "glwidget.h"

GLuint GLWidget::sharedObject = 0;
//int GLWidget::refCount = 0;

GLWidget::GLWidget(QWidget *parent)
    : QGLWidget(parent)
{
    clearColor = Qt::black;
    xRot = 0;
    yRot = 0;
    zRot = 0;
}

GLWidget::~GLWidget()
{
        makeCurrent();
        glDeleteLists(sharedObject,1);

}

QSize GLWidget::minimumSizeHint() const
{
    return QSize(150, 150);
}

QSize GLWidget::sizeHint() const
{
    return QSize(400, 400);
}

void GLWidget::rotateBy(int xAngle, int yAngle, int zAngle)
{
xRot += xAngle;
yRot += yAngle;
zRot += zAngle;
updateGL();

}

void GLWidget::rotate_x(int xAngle)
{
rotateBy(xAngle, 0,0);
}

void GLWidget::rotate_y(int yAngle)
{
rotateBy(0,yAngle,0);
}

void GLWidget::rotate_z(int zAngle)
{
rotateBy(0,0,zAngle);
}

void GLWidget::setClearColor(const QColor &color)
{
    clearColor = color;
    updateGL();
}

void GLWidget::initializeGL()
{
    if (!sharedObject)
        sharedObject = makeObject();

    glEnable(GL_DEPTH_TEST);
    glEnable(GL_CULL_FACE);
```

```
    glEnable(GL_TEXTURE_2D);
}

void GLWidget::paintGL()
{
    qglClearColor(clearColor);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
   glTranslated(0.0, 0.0, -10.0);
    glRotated(xRot / 16.0, 1.0, 0.0, 0.0);
    glRotated(yRot / 16.0, 0.0, 1.0, 0.0);
    glRotated(zRot / 16.0, 0.0, 0.0, 1.0);
    glCallList(sharedObject);
}

void GLWidget::resizeGL(int width, int height)
{
    int side = qMin(width, height);
    glViewport((width - side) / 2, (height - side) / 2, side, side);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-0.5, +0.5, +0.5, -0.5, 4.0, 15.0);
    glMatrixMode(GL_MODELVIEW);
}

void GLWidget::mousePressEvent(QMouseEvent *event)
{
    lastPos = event->pos();
}

void GLWidget::mouseMoveEvent(QMouseEvent *event)
{
    int dx = event->x() - lastPos.x();
    int dy = event->y() - lastPos.y();

    if (event->buttons() & Qt::LeftButton) {
        rotateBy(8 * dy, 8 * dx, 0);
    } else if (event->buttons() & Qt::RightButton) {
        rotateBy(8 * dy, 0, 8 * dx);
    }
    lastPos = event->pos();
}

void GLWidget::mouseReleaseEvent(QMouseEvent * /* event */)
{
    emit clicked();
}

GLuint GLWidget::makeObject()
{
    static const int coords[6][4][3] = {
        { { +1, -1, -1 }, { -1, -1, -1 }, { -1, +1, -1 }, { +1, +1, -1 } },
        { { +1, +1, -1 }, { -1, +1, -1 }, { -1, +1, +1 }, { +1, +1, +1 } },
        { { +1, -1, +1 }, { +1, -1, -1 }, { +1, +1, -1 }, { +1, +1, +1 } },
        { { -1, -1, -1 }, { -1, -1, +1 }, { -1, +1, +1 }, { -1, +1, -1 } },
        { { +1, -1, +1 }, { -1, -1, +1 }, { -1, -1, -1 }, { +1, -1, -1 } },
        { { -1, -1, +1 }, { +1, -1, +1 }, { +1, +1, +1 }, { -1, +1, +1 } }
    };


    GLuint textures[6];
    for (int j=0; j < 6; ++j)
        textures[j] = bindTexture(QPixmap(QString(":/images/side%1.png").arg(j + 1)),
                                  GL_TEXTURE_2D);

    GLuint list = glGenLists(1);
    glNewList(list, GL_COMPILE);
    for (int i = 0; i < 6; ++i) {
        glBindTexture(GL_TEXTURE_2D, textures[i]);
        glBegin(GL_QUADS);
        for (int j = 0; j < 4; ++j) {
            glTexCoord2d(j == 0 || j == 3, j == 0 || j == 1);
            glVertex3d(0.2 * coords[i][j][0], 0.2 * coords[i][j][1],
                       0.2 * coords[i][j][2]);
        }
        glEnd();
    }

    glEndList();
    return list;
}
```

Fichier window.cpp:

```
/****************************************************************************
**
** Copyright (C) 2005-2007 Trolltech ASA. All rights reserved.
```

```
****************************************************************************/

#include <QtGui>

#include "qslider.h"
#include "glwidget.h"
#include "window.h"
#include "ui_frame_pour_TP5.h"

Window::Window()
{

    QGridLayout *mainLayout = new QGridLayout;


            QColor clearColor;
            clearColor.setHsv(0, 255, 163);

            glWidget = new GLWidget;
            glWidget->setClearColor(clearColor);
            mainLayout->addWidget(glWidget,1,1);

            QSlider *slider1= createSlider();
            QSlider *slider2= createSlider();
            QSlider *slider3= createSlider();
            mainLayout->addWidget(slider1,2,1);
           mainLayout->addWidget(slider2,3,1);
           mainLayout->addWidget(slider3,4,1);
            QObject::connect( slider1, SIGNAL(valueChanged(int)), glWidget,SLOT(rotate_x(int)) );
            QObject::connect( slider2, SIGNAL(valueChanged(int)), glWidget,SLOT(rotate_y(int)) );
            QObject::connect( slider3, SIGNAL(valueChanged(int)), glWidget,SLOT(rotate_z(int)) );

            Ui::Frame ui;
            QFrame *my_frame = new QFrame;
            ui.setupUi(my_frame);
            mainLayout->addWidget(my_frame,5,1);
            QObject::connect(ui.spinBox_x,SIGNAL(valueChanged(int)),glWidget,SLOT(rotate_x(int)));
            QObject::connect(ui.spinBox_y,SIGNAL(valueChanged(int)),glWidget,SLOT(rotate_y(int)));
            QObject::connect(ui.spinBox_z,SIGNAL(valueChanged(int)),glWidget,SLOT(rotate_z(int)));

            QObject::connect(glWidget,SIGNAL());

            setLayout(mainLayout);


   QTimer *timer = new QTimer(this);
   connect(timer, SIGNAL(timeout()), this, SLOT(rotateOneStep()));
   QObject::connect(ui.pushButton_stop,SIGNAL(clicked()),timer,SLOT(stop()));
   QObject::connect(ui.pushButton_start,SIGNAL(clicked()),timer,SLOT(start()));
   timer->start(50);

    setWindowTitle(tr("The Dreamteam"));
}


void Window::rotateOneStep()
{

       glWidget->rotateBy(+2 * 16, +2 * 16, -1 * 16);
}

QSlider *Window::createSlider()
{
        QSlider *slider = new QSlider(Qt::Horizontal);
        slider->setRange(0, 360 );
        slider->setSingleStep(16);
        slider->setPageStep(15 );
        slider->setTickInterval(15);
        slider->setTickPosition(QSlider::TicksRight);
        return slider;
}
```