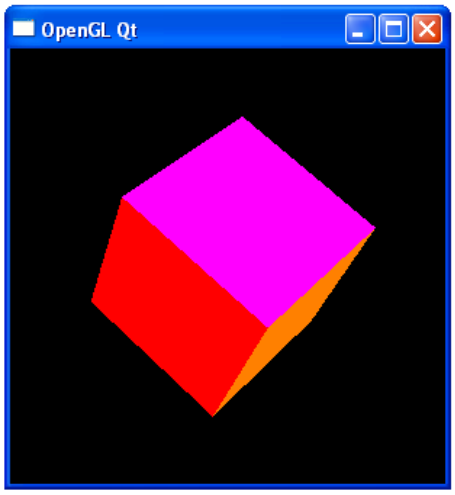
OpenGL with Qt

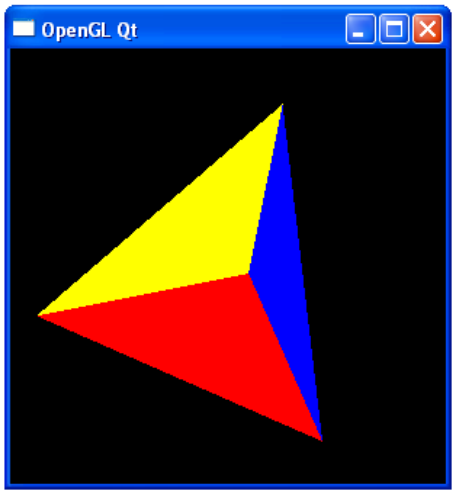
Overview - OpenGL API

This example demonstrates how draw a 3D square using the Qt **QtOpenGL** module.

OpenGL (Open Graphics Library) is a standard specification defining a cross-platform/language API for writing applications that produce 2D and 3D computer graphics. Qt provides access to OpenGL through the **OpenGL** Module.

To show how it works we will create a 3D square using `QtOpenGL`, we will set different colors in each face of the square and add a timer to spin the square. We need to subclass `QGLWidget` (inherits from `QWidget`), and link the application against **QtOpenGL** and **OpenGL** library.





OpenGLQt.pro

To link the application against the **QtOpenGL** module and the system's **OpenGL** library, add **QT += opengl** in **.pro** file

```
QT       += core gui\
            opengl
TARGET = OpenGLQt
TEMPLATE = app
SOURCES += main.cpp\
         openglqt.cpp
HEADERS  += openglqt.h
FORMS    += openglqt.ui
```

openglqt.h

```
#ifndef OPENGLQT_H
#define OPENGLQT_H

#include <QMainWindow>
#include <QtOpenGL>
namespace Ui {
    class OpenGLQt;
}

class OpenGLQt : public QGLWidget
{
    Q_OBJECT

public:
    explicit OpenGLQt(QWidget *parent = 0);
    ~OpenGLQt();

private:
    Ui::OpenGLQt *ui;


protected:
void initializeGL();
void resizeGL(int width, int height);
void paintGL();
void mousePressEvent(QMouseEvent *event);
void mouseMoveEvent(QMouseEvent *event);
void mouseDoubleClickEvent(QMouseEvent *event);
private:
```


(C) Copyright Nokia 2013. All rights reserved.

```
void draw();
int faceAtPosition(const QPoint &pos);
GLfloat rotationX;
GLfloat rotationY;
GLfloat rotationZ;
QColor faceColors[4];
QPoint lastPos;
void Spin(int xAngle, int yAngle, int zAngle);



private slots:
 void Rotate();
};

#endif // OPENGLQT_H
```

The **OpenGLQt** class inherits from **QGLWidget**. The `initializeGL()`, `resizeGL()`, and `paintGL()` functions are reimplemented from `QGLWidget`.

openglqt.cpp

```
#include "openglqt.h"
#include "ui_openglqt.h"

OpenGLQt::OpenGLQt(QWidget *parent) :
      QGLWidget(parent),
    ui(new Ui::OpenGLQt)
{


    //QGLWidget::setFormat() to specify the OpenGL display context
    setFormat(QGLFormat(QGL::DoubleBuffer | QGL::DepthBuffer));
    rotationX = -21.0;
    rotationY = -57.0;
    rotationZ = 0.0;

    // Note: Square faces are set with static color and are cannot be changed during runtime
     // set the color of the faces of Tetrahedron
    faceColors[0] = Qt::red;
    faceColors[1] = Qt::green;
    faceColors[2] = Qt::blue;
    faceColors[3] = Qt::yellow;

    initializeGL();
    resizeGL(51,51);
    paintGL();


    //timer to spin the object
    QTimer *timer = new QTimer(this);
    connect(timer, SIGNAL(timeout()), this, SLOT(Rotate()));
    timer->start(20);


}

OpenGLQt::~OpenGLQt()
{
    delete ui;
}


void OpenGLQt::initializeGL()
{
qglClearColor(Qt::black);
glShadeModel(GL_FLAT);
glEnable(GL_DEPTH_TEST);
glEnable(GL_CULL_FACE);
}

void OpenGLQt::resizeGL(int width, int height)
{
glViewport(0, 0, width, height);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
GLfloat x = GLfloat(width) / height;
glFrustum(-x, x, -1.0, 1.0, 4.0, 15.0);
glMatrixMode(GL_MODELVIEW);
}


void OpenGLQt::paintGL()
{
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

draw();
}

void OpenGLQt::draw()
{

    //change the value if don't want to spin automatically and want to check the mouse event
   // glRotatef(rotationX/16.0f, 1.0, 0.0, 0.0); to glRotatef(rotationX, 1.0, 0.0, 0.0);
   // glRotatef(rotationY/16.0f, 0.0, 1.0, 0.0); to glRotatef(rotationY, 0.0, 1.0, 0.0);
    //glRotatef(rotationZ/16.0f, 0.0, 0.0, 1.0); to glRotatef(rotationZ, 0.0, 0.0, 1.0);


    // Draw Tetrahedron
  static const GLfloat P1[3] = { 0.0, -1.0, +2.0 };
    static const GLfloat P2[3] = { +1.73205081, -1.0, -1.0 };
    static const GLfloat P3[3] = { -1.73205081, -1.0, -1.0 };
    static const GLfloat P4[3] = { 0.0, +2.0, 0.0 };
    static const GLfloat * const coords[4][3] = {
    { P1, P2, P3 }, { P1, P3, P4 }, { P1, P4, P2 }, { P2, P4, P3 }
    };
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glTranslatef(0.0, 0.0, -10.0);
    glRotatef(rotationX/16.0f, 1.0, 0.0, 0.0);
    glRotatef(rotationY/16.0f, 0.0, 1.0, 0.0);
    glRotatef(rotationZ/16.0f, 0.0, 0.0, 1.0);
    for (int i = 0; i < 4; ++i) {
    glLoadName(i);
    glBegin(GL_TRIANGLES);
```


(C) Copyright Nokia 2013. All rights reserved.

```
    glColor(faceColors[i]);
    for (int j = 0; j < 3; ++j) {
    glVertex3f(coords[i][j][0], coords[i][j][1],
    coords[i][j][2]);
    }
    glEnd();
    }

/*

    // Draw Square
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glTranslatef(0.0, 0.0, -10.0);
    glRotatef(rotationX/16.0f, 1.0, 0.0, 0.0);
    glRotatef(rotationY/16.0f, 0.0, 1.0, 0.0);
    glRotatef(rotationZ/16.0f, 0.0, 0.0, 1.0);
    glBegin(GL_QUADS);

    glColor3f(0.5f,0.5f,1.0f);
    glColor3f(0.0f,1.0f,0.0f);
    glVertex3f( 1.0f, 1.0f,-1.0f);
    glVertex3f(-1.0f, 1.0f,-1.0f);
    glVertex3f(-1.0f, 1.0f, 1.0f);
    glVertex3f( 1.0f, 1.0f, 1.0f);
    glColor3f(1.0f,0.5f,0.0f);
    glVertex3f( 1.0f,-1.0f, 1.0f);
    glVertex3f(-1.0f,-1.0f, 1.0f);
    glVertex3f(-1.0f,-1.0f,-1.0f);
    glVertex3f( 1.0f,-1.0f,-1.0f);
    glColor3f(1.0f,0.0f,0.0f);
    glVertex3f( 1.0f, 1.0f, 1.0f);
    glVertex3f(-1.0f, 1.0f, 1.0f);
    glVertex3f(-1.0f,-1.0f, 1.0f);
    glVertex3f( 1.0f,-1.0f, 1.0f);
    glColor3f(1.0f,1.0f,0.0f);
    glVertex3f( 1.0f,-1.0f,-1.0f);
    glVertex3f(-1.0f,-1.0f,-1.0f);
    glVertex3f(-1.0f, 1.0f,-1.0f);
    glVertex3f( 1.0f, 1.0f,-1.0f);
    glColor3f(0.0f,0.0f,1.0f);
    glVertex3f(-1.0f, 1.0f, 1.0f);
    glVertex3f(-1.0f, 1.0f,-1.0f);
    glVertex3f(-1.0f,-1.0f,-1.0f);
    glVertex3f(-1.0f,-1.0f, 1.0f);
    glColor3f(1.0f,0.0f,1.0f);
    glVertex3f( 1.0f, 1.0f,-1.0f);
    glVertex3f( 1.0f, 1.0f, 1.0f);
    glVertex3f( 1.0f,-1.0f, 1.0f);
    glVertex3f( 1.0f,-1.0f,-1.0f);
    glEnd();


*/
}

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

void OpenGLQt::mouseMoveEvent(QMouseEvent *event)
{
GLfloat dx = GLfloat(event->x() - lastPos.x()) / width();
GLfloat dy = GLfloat(event->y() - lastPos.y()) / height();
if (event->buttons() & Qt::LeftButton) {
rotationX += 180 * dy;
rotationY += 180 * dx;
updateGL();
}
else if (event->buttons() & Qt::RightButton)
{
rotationX += 180 * dy;
rotationZ += 180 * dx;
updateGL();


}
lastPos = event->pos();
}

void OpenGLQt::mouseDoubleClickEvent(QMouseEvent *event)
{
int face = faceAtPosition(event->pos());
if (face != -1) {
QColor color = QColorDialog::getColor(faceColors[face], this);
if (color.isValid()) {
faceColors[face] = color;
updateGL();
}
}
}

int OpenGLQt::faceAtPosition(const QPoint &pos)
{
const int MaxSize = 512;
GLuint buffer[MaxSize];
GLint viewport[4];
glGetIntegerv(GL_VIEWPORT, viewport);
glSelectBuffer(MaxSize, buffer);
glRenderMode(GL_SELECT);
glInitNames();
glPushName(0);
glMatrixMode(GL_PROJECTION);
glPushMatrix();
glLoadIdentity();
gluPickMatrix(GLdouble(pos.x()), GLdouble(viewport[3] - pos.y()),
5.0, 5.0, viewport);
GLfloat x = GLfloat(width()) / height();
glFrustum(-x, x, -1.0, 1.0, 4.0, 15.0);
draw();
glMatrixMode(GL_PROJECTION);
glPopMatrix();
if (!glRenderMode(GL_RENDER))
return -1;
return buffer[3];
}

void OpenGLQt::Rotate()
{
```


(C) Copyright Nokia 2013. All rights reserved.

```
    //QMessageBox::about(this,"Hello","world");
      this->Spin(+2 * 16, +2 * 16, -1 * 16);
}
void OpenGLQt::Spin(int xAngle, int yAngle, int zAngle)
{
    rotationX += xAngle;
    rotationY += yAngle;
    rotationZ += zAngle;
    updateGL();
}
```

1. In the constructor we initialize the class private variables and call `QGLWidget::setFormat()`. `QGLWidget::setFormat()` is called to specify the **OpenGL** display context.
2. The `initializeGL()` function is called just once and before `paintGL()` is called. This is the place where we can set up the **OpenGL** rendering context, define display lists, and perform other initializations.
3. The `resizeGL()` function is called before `paintGL()` is called the first time, but after `initializeGL()` is called. It is also called whenever the widget is re-sized. This is the place where we can set up the **OpenGL** viewport, projection, and any other settings that depend on the widget's size.
4. The `paintGL()` function is called whenever the widget needs to be repainted. The actual drawing is performed by the private function `draw()`.
5. In `draw()`, we draw the Square, taking into account the x, y, and z rotations and the colors of the faces.
6. The `Rotate()` function is called to Spin the Square after a particular time, depends on `QTimer`
7. The `Spin()` function is called to rotate the square at a particular angle.
8. After modifying the `rotationX`,`rotationY` and `rotationZ` variable, we call `updateGL()` to redraw the scene.
9. If the system doesn't have `OpenGL` support then we also display a message in the main.cpp file.

main.cpp

```
#include <QtGui/QApplication>
#include "openglqt.h"
#include <QApplication>
#include <iostream>
#include <QtOpenGL>
using namespace std;
int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    if (!QGLFormat::hasOpenGL()) {
    cerr << "This system has no OpenGL support" << endl;
    return 1;
    }
    OpenGLQt openglqt;
    openglqt.setWindowTitle(QObject::tr("OpenGL Qt"));
    openglqt.resize(300, 300);
    openglqt.show();
    return app.exec();
}
```

More details on OpenGL with Qt are available here: How to utilise OpenGL ES 2.0 on Symbian^3 and Maemo

Source Code

The full source code presented in this article is available here File:OpenGLQt.zip

--somnathbanik

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.