

QT E OPENGL

Le librerie QT forniscono al loro interno una classe per poter creare una widget correttamente inizializzata per il rendering tramite opengl. Questo ci facilita di molto le cose, visto che non dobbiamo aver a che fare con glx o simili, se vogliamo avere una finestra di rendering accompagnata a classiche widget come bottoni, lineedit, ecc (come siamo costretti a fare se siamo sotto window o utilizziamo gtk).

La classe principale è QGLWidget, che fornisce come dicevo una widget per il rendering diretto tramite le api opengl, e, in pratica, è realizzata in modo molto simile alla classica libreria glut.

Noi non dobbiamo fare altro creare una classe che estende QGLWidget e in particolare implementa i metodi virtuali initializeGL paintGL e resizeGL. Dopodichè tramite il tipico sistema qt di slot e signals colleghiamo ad alcune funzioni create da noi in questa classe alcuni eventi che ci interessano come la pressione di un tasto da tastiera o un movimento del mouse o ancora un evento su qualche altro componente come per esempio un pulsante.

Riporto di seguito un esempio di classe di questo tipo.

Dal file glbox.h:

```
#include <qgl.h>

class GLBox : public QGLWidget
{
    Q_OBJECT

public:
    GLBox( QWidget* parent, const char* name );
    ~GLBox();

public slots:
    /*slots collegati ad alcune widget esterne, in questo esempio*/
    void      setXRotation( int degrees );
    void      setYRotation( int degrees );
    void      setZRotation( int degrees );

protected:
    void      initializeGL();
    void      paintGL();
    void      resizeGL( int w, int h );

private:
    /*alcune variabili interne private di esempio*/
    GLuint object;
    GLfloat xRot, yRot, zRot, scale;
};
```

Dal file glbox.cpp:

```
/*Nel costruttore si inizializzano le tipiche variabili interne di sistema*/
GLBox::GLBox( QWidget* parent, const char* name )
    : QGLWidget( parent, name )
{
    xRot = yRot = zRot = 0.0;        // default object
rotation
    scale = 1.25;                    // default object scale
    object = 0;
}

/*Nel distruttore si deallocano eventuali risorse occupate*/
GLBox::~GLBox()
{
}

/*La funzione paintGL equivale alla funzione registrata tramite glutDisplayFunc e
richiamata tramite glutPostRedisplay() e svolge le stesse identiche funzioni utilizzando
le stesse api opengl*/
void GLBox::paintGL()
{
    glClear( GL_COLOR_BUFFER_BIT );

    glTranslatef( 0.0, 0.0, -10.0 );
    glScalef( scale, scale, scale );
    glRotatef( xRot, 1.0, 0.0, 0.0 );
    glRotatef( yRot, 0.0, 1.0, 0.0 );
    glRotatef( zRot, 0.0, 0.0, 1.0 );

    glBegin( GL_LINES );
    glVertex3f( 1.0, 0.5, -0.4 );    glVertex3f( 1.0,
0.5, 0.4 );
    glVertex3f( 1.0, -0.5, -0.4 );  glVertex3f( 1.0,
-0.5, 0.4 );
    glVertex3f( -1.0, -0.5, -0.4 ); glVertex3f( -1.0,
-0.5, 0.4 );
    glVertex3f( -1.0, 0.5, -0.4 );  glVertex3f( -1.0,
0.5, 0.4 );
    glEnd();
}

/*la funzione initializeGL viene chiamata solo una volta alla prima inizializzazione
dell'oggetto. Notare qglClearColor*/
void GLBox::initializeGL()
{
    qglClearColor( black );
    glShadeModel( GL_FLAT );
}
```

```
/* resizeGL viene è la funzione che viene chiamata in corrispondenza dell'evento di
resize e viene gestita tramite le stesse api opengl che si utilizzano con glut o simili*/
void GLBox::resizeGL( int w, int h )
```

```
{
    glViewport( 0, 0, (GLint)w, (GLint)h );
    glMatrixMode( GL_PROJECTION );
    glLoadIdentity();
    glFrustum( -1.0, 1.0, -1.0, 1.0, 5.0, 15.0 );
    glMatrixMode( GL_MODELVIEW );
    glLoadIdentity();
}
```

```
/*Riporto un esempio di uno degli slot, per mettere in luce che in questo caso per
forzare un evento di display anziché la glutPostRedisplay viene chiamata la funzione
updateGL*/
```

```
void GLBox::setXRotation( int degrees )
{
    xRot = (GLfloat)(degrees % 360);
    updateGL();
}
```

A questo punto basterà aggiungere nella funzione di inizializzazione della classe che implementa la finestra di visualizzazione il seguente codice:

```
/*Prima si crea una widget del tipo glbox, in questo caso f è il frame all'interno del
quale la si vuole aggiungere*/
GLBOX* c = new GLBox( f, "glbox");
```

```
//Poi si connettono gli slot di glbox con i segnali che ci interessano, per esempio:
QSlider* x = new QSlider ( 0, 360, 60, 0,
QSlider::Vertical, this, "xsl" );
x->setTickmarks( QSlider::Left );
QObject::connect( x, SIGNAL(valueChanged(int)),c,SLOT
(setXRotation(int)) );
```

```
/*Infine aggiungiamo al layout desiderato nella posizione desiderata il nostro oggetto
glbox:*/
layout1->addWidget( c, 1 );
```

Naturalmente la finestra che contiene questo codice può essere una finestra qualsiasi, e il codice per realizzarla è quello di una normalissima QWidget che può benissimo essere creata tramite QT Designer. Anche il file di main del programma, che istanzia tale finestra e la setta visibile è lo stesso che per qualsiasi altra finestra di un programma QT.