Technische Universität München
Fakultät für Informatik
Bildverarbeitung und Graphikprogrammierung in C++
C. Bichlmeier, S. Kettner, O. Kutter und M. Blume

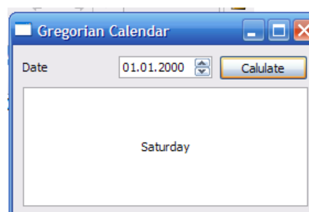
SS 2007
Übungsblatt 3
8. Mai 2007

Übungen zu Bildverarbeitung und Graphikprogrammierung in C++

Aufgabe 7 (H) Gregorianischer Kalender mit QT

Der Gregorianische Kalender aus Aufgabe 2 soll mittels QT implementiert werden. Es soll ein QWidget gezeichnet werden, das als Eingabe ein Datum erlaubt und daraus den jeweiligen Wochentag berechnet und auch anzeigt. Der Code für die Berechnung kann vollständig aus Aufgabe 2 übernommen werden.

Beispielsweise könnte die Applikation folgendermaßen aussehen:



Dafür werden zwei `QLabel`, ein `QDateEdit`, ein `QPushButton`, `QVBoxLayout` sowie ein `QHBoxLayout` verwendet. Nähere Informationen zu den jeweiligen Komponenten finden Sie auf der Qt-Referenzseite `http://doc.trolltech.com/4.2/index.html`.

Um eigene `Widgets`, `SLOTS` und `SINGALS` verwenden zu können, muss der QT `moc`-Kompiler verwendet werden. Im VS kann dies durch einen *Custom Build Step* realisiert werden. Dieser kann über die *Properties* des Headers der jeweiligen Klasse definiert werden. Die Parameter hierfür sind:

- *Command Line*, z.B.:
  `...\Qt\bin\moc.exe "\$(InputPath)" -o "\$(InputDir)\moc_\$(InputName).cpp"`
- *Description*:
  Was auch immer in dem VS Output-Fenster dargestellt werden soll, wenn dieser Schritt ausgeführt wird, z.B.:
  `MOCen von $(InputName)`
- *Outputs*, z.B.:
  `"$(InputDir)\moc_$(InputName).cpp"`

Zudem muss das Macro `Q_OBJECT` eingebunden werden. Zuletzt muss die erzeugte („geMOCte“) Datei noch zu den Quelldateien im VS hinzugefügt werden.
Im Folgenden ist ein kurzes Codefragment für eine von `QWidget` abgeleitete Klasse. Diese erlaubt durch die definition von slots, benutzerdefinierte Aktionen bei bestimmten Signalen auszuführen (siehe `startCalculation`).

```
#include <QWidget>

class MyWidget : public QWidget {
  Q_OBJECT
public:
  MyWidget();
  ~MyWidget();

public slots:
  void startCalculation();
};
```

Aufgabe 8 (H) 2D Objekte anzeigen und manipulieren

Lesen Sie sich im Red Book (Link auf der Homepage des Graphikprogrammierpraktikums) das Kapitel 2, *Drawing Geometric Objects* durch. Ziel dieser Aufgabe ist, ein Dreieck und ein Rechteck in einem Fenster nebeneinander darzustellen und Transformationen auf ihnen auszuführen. Um die Transformationen auszuführen, können Sie die von OpenGL angebotenen Funktionen `glRotatef`, `glTranslatef` und `glScalef` benutzen.

In Qt bieten die sog. `QGLWidgets` die Möglichkeit, innerhalb eines `QWidget` mittels OpenGL Objekte zu zeichnen. Die wichtigsten Funktionen, die dazu implementiert werden müssen, werden in dem folgenden Beispiel für ein Klassengerüst dargestellt.

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

class MyGLWidget : public QGLWidget
{
Q_OBJECT

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

protected:
virtual void initializeGL(); //Initialisiert OpenGL
virtual void paintGL(); //zeichnet OpenGL Objekte

virtual void mousePressEvent(QMouseEvent* event); //mouse handling
virtual void mouseMoveEvent(QMouseEvent* event); //mouse handling

virtual void keyPressEvent(QKeyEvent* event); //keyboard handling
};
```

Auf der Basis der vorhergehenden Aufgabe zur Einführung in Qt und der eben beschrieben Struktur sollen nun erste zweidimensionale Objekte erzeugt werden.

Im Folgenden ist eine Implementierung der Funktion textttpaintGL, die ein Quadrat in die Mitte des Viewports zeichnet:

```
void MyGLWidget::initializeGL() {
glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
```

```
glViewport(0, 0, this->size().width(), this->size().height());
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(-1.0,1.0,-1.0,1.0, -1.0, 1.0);
glMatrixMode(GL_MODELVIEW);
}

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

glLoadIdentity();

glColor3f(1.0f,0.0f,0.0f);

//Quad
glBegin(GL_QUADS);
glVertex3f(-0.5f, 0.5f, 0.0f);
glVertex3f(0.5f, 0.5f, 0.0f);
glVertex3f(0.5f, -0.5f, 0.0f);
glVertex3f(-0.5f, -0.5f, 0.0f);
glEnd();
}
```

a) Erstellen Sie ein QGLWidget, das ein Dreieck und ein andersfarbiges Rechteck nebeneinander zeichnet.

b) Fügen Sie eine Methode `rotate(int reverse)` hinzu, welche das Dreieck im Uhrzeigersinn und das Rechteck gegen den Uhrzeigersinn um 5 Grad rotiert, falls `reverse = 1`, bei `reverse = -1` soll sich die Drehrichtung umkehren. Die Funktion soll so eingebunden werden, dass Sie diese Funktion testen können, indem Sie in dem Fenster die Maus mit dem gedrückten linken Mausknopf nach oben oder unten bewegen.

c) Fügen Sie eine Methode `scale(int reverse)` hinzu, welche die definierten Objekte vergrößert, falls `reverse = 1`, bzw. verkleinert, falls `reverse = -1`. Die Funktion soll so integriert werden, dass Sie diese Funktion testen können, indem Sie in dem Fenster die Maus mit dem gedrückten rechten Mausknopf nach oben oder unten bewegen.

d) Fügen Sie eine Methode `reset()` hinzu, die die Ausgangsposition sowie -größe der Original-Objekte wiederherstellt. Testen Sie die Methode, indem Sie den ”r”-Knopf auf der Tastatur drücken.

e) Sie können nun Objekte auf einer 2D Fläche transformieren. Programmieren Sie eine Funktion `doWeirdThingsWithPolygons()`, die verschiedene Polygone in einem Fenster erzeugt und diese zufällig transformiert. Man könnte beispielsweise hüpfende Polygone programmieren. Sie können diese Funktion ausführen, indem Sie den ”w”-Knopf der Tastatur drücken.

Voraussetzungen, dass eine Aufgabe korrigiert wird:

a) Der gesamte Code sollte kommentiert sein

b) Alle Solutiondateien (*.sln) und Projektdateien (*.vcproj) müssen eingecheckt sein

c) Alle Pfade (includes und libs) müssen **relativ** gesetzt sein

d) Zum Testen, ob alles stimmt, nach dem finalen `commit` (Montag vor 12Uhr) einen neuen `checkout` machen.

Wenn das Kompilieren und Starten der Solution bzw. aller Projektdateien fehlerfrei funktioniert, dann **und nur dann** wird die Aufgabe korrigiert.