LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Prof. Dr. Andreas Butz | Prof. Dr. Ing. Axel Hoppe

Dipl.-Medieninf. Dominikus Baur
Dipl.-Medieninf. Sebastian Boring

Übung: Computergrafik 1

Einführung ins Qt-Framework: QtCreator, einfache GUIs
Einführung in OpenGL: Grundlagen

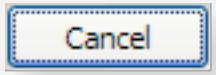
Qt

Qt – Allgemeines

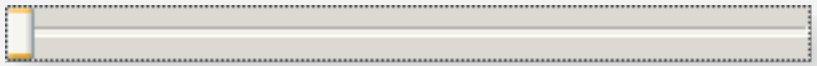
- Plattformunabhängiger GUI (u.a.) Toolkit
- Entwickelt von Qt Software (Nokia), ehemals Trolltech
- Prominente Anwendungen: KDE, Opera, Google Earth, Skype
- Eigentlich ein zusätzlicher Präprozessor, der C++-Code erzeugt
- Kostenlos und Open Source
- Verfügbar für X11, OSX, Windows, Embedded (PDA, Smartphone)
- SDK stellt Entwicklungsumgebung (QtCreator) und diverse Tools zur Verfügung

(Quelle: en.wikipedia.org/wiki/Qt_(toolkit)

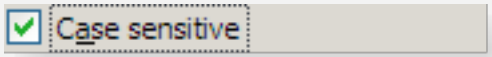
Qt Widgets



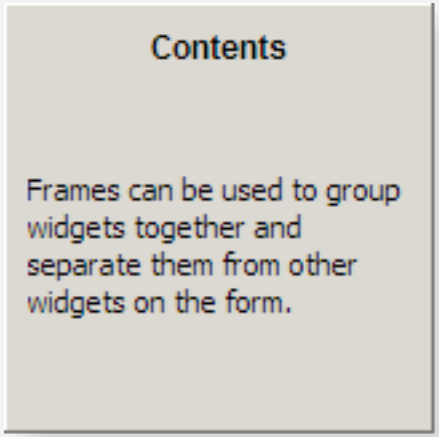
QPushButton



QSlider

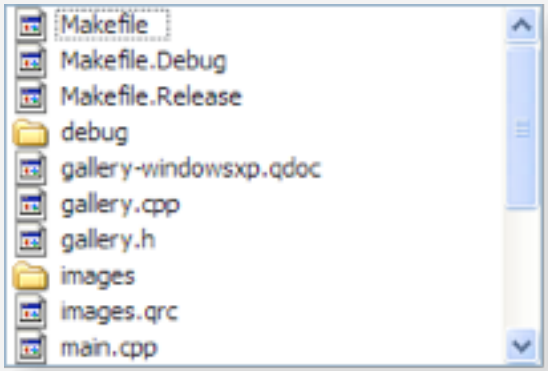


QCheckBox

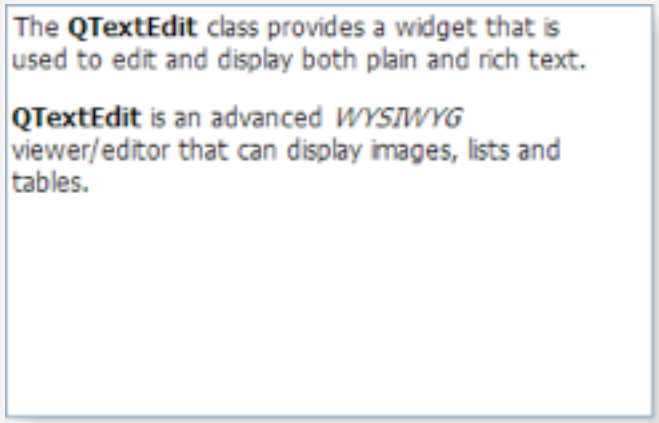


QFrame

QLineEdit



QListView

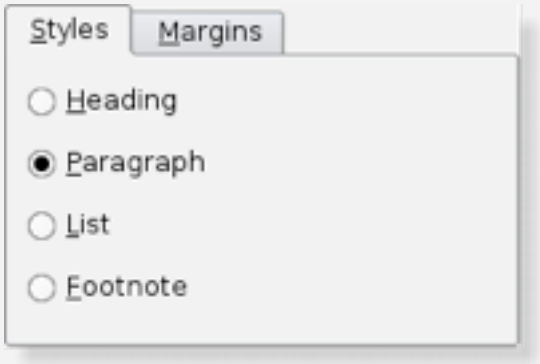


QTextEdit

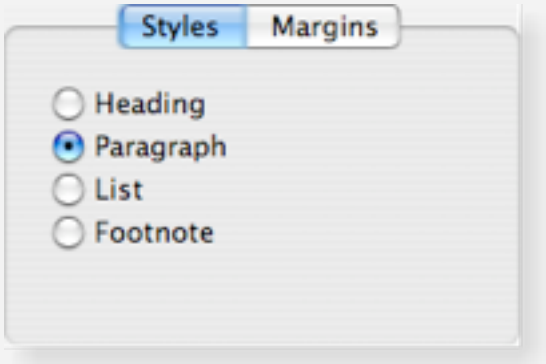
(Quelle: http://doc.trolltech.com/4.5/gallery-windowsxp.html)

Qt Styles

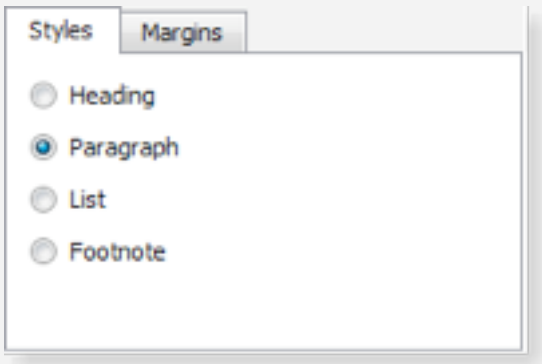
- Qt bietet für die verfügbaren Widgets verschiedene Styles, die ein globales Look & Feel definieren
- Eigene Styles lassen sich durch Erben von QStyle erzeugen (Beispiel siehe http://doc.trolltech.com/4.3/widgets-styles.html)
- Styles werden durch Aufruf von QApplication::setStyle(Style s) gesetzt



QPlastiqueStyle



QMacStyle

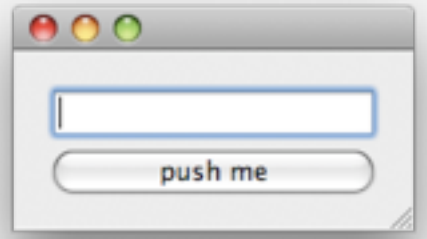


QWindowsVistaStyle

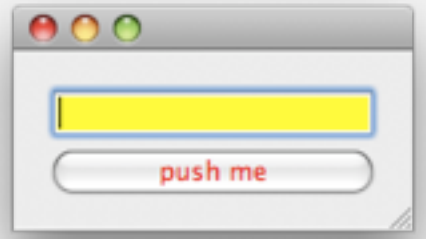
(Quelle: http://doc.trolltech.com/4.5/gallery.html)

Qt Style Sheets

- Style Sheets sind eine an CSS angelehnte Möglichkeit seine Applikation zu skinnen und damit weniger aufwändig als Styles
- Style Sheets werden durch einen Aufruf von QApplication::setStyleSheet(QString s) gesetzt

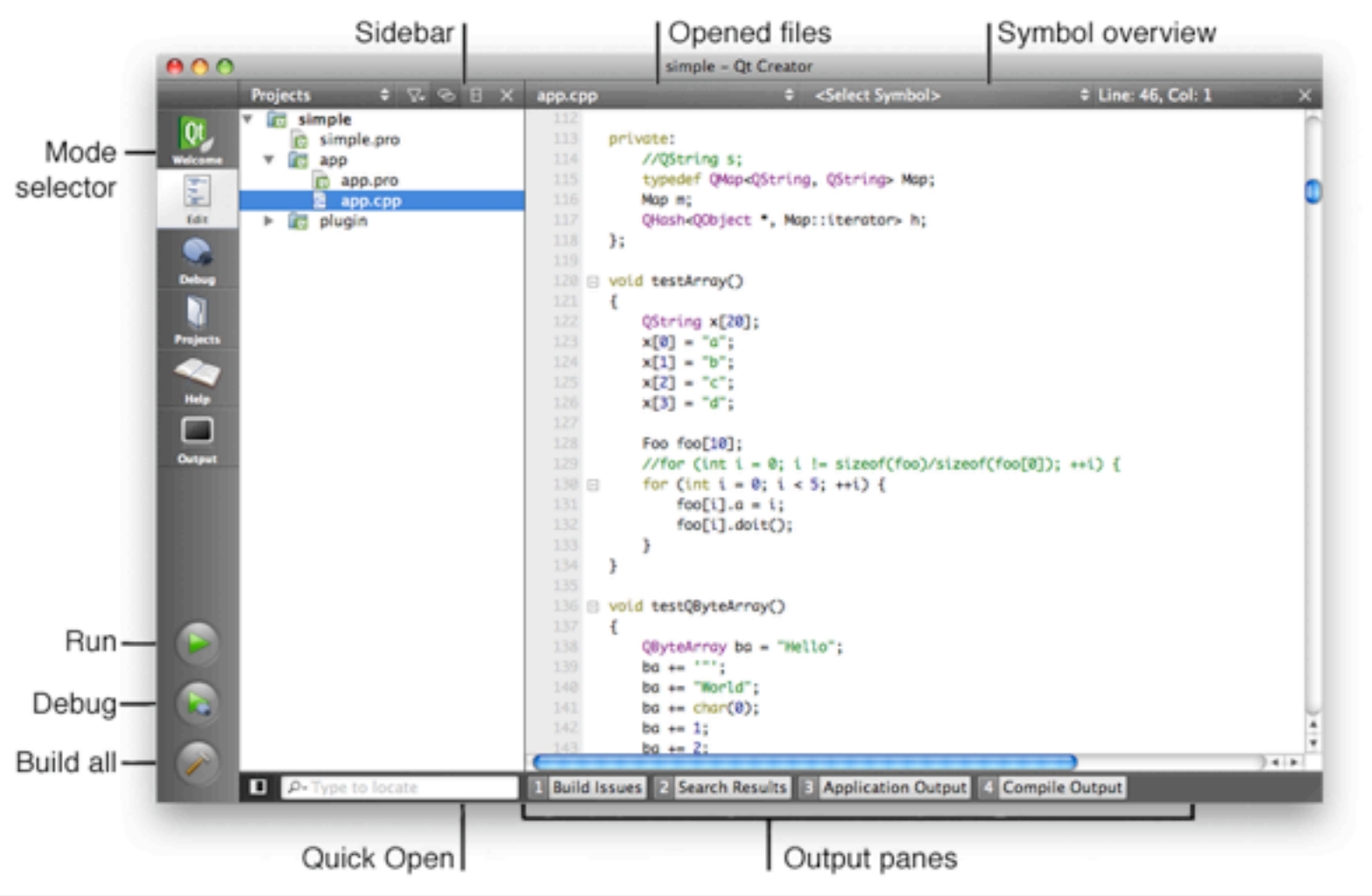



```
app.setStyleSheet(
        "QLineEdit { background: yellow }"
        "QPushButton { color: red }"
);
```



(Quelle: http://doc.trolltech.com/4.5/stylesheet.html)

Qt Creator

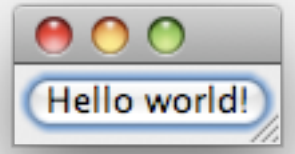


(Quelle: http://doc.trolltech.com/qtcreator-1.1/creator-quick-tour.html)

Hello Qt

helloQt.cpp

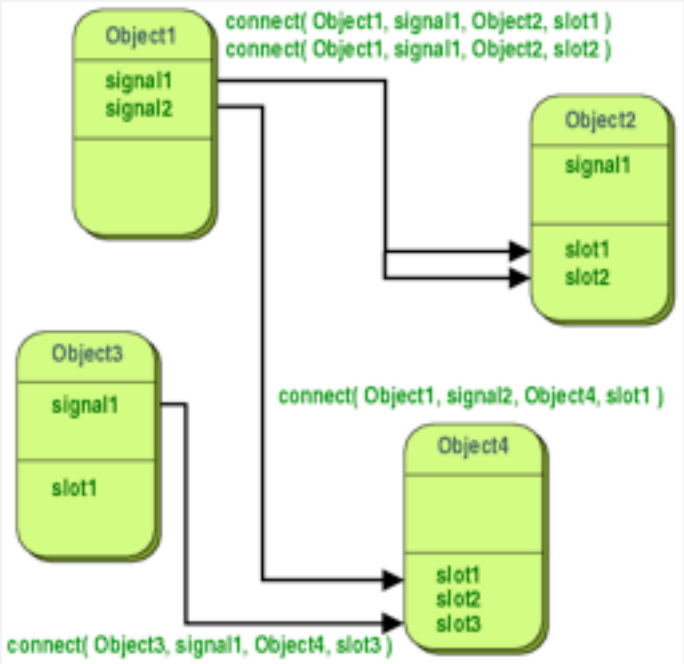
```
#include <QApplication>
#include <QPushButton>

int main(int argc, char* argv[]){
    QApplication app(argc, argv);

    QPushButton hello("Hello world!");
    hello.resize(100, 30);

    hello.show();
    return app.exec();
}
```



(Quelle: http://doc.trolltech.com/4.3/tutorial-t1.html)

Qt Signals and Slots

- In anderen grafischen Frameworks werden Callbacks benutzt um Kommunikation zwischen Komponenten zu ermöglichen.
- Qt bietet dazu den Signals and Slots Mechanismus an:



(Quelle: http://doc.trolltech.com/4.5/signalsandslots.html)

Qt Signals and Slots

```cpp
#ifndef COUNTER_H
#define COUNTER_H

#include <QObject>

class Counter : public QObject
{
    Q_OBJECT

public:
    Counter() { m_value = 0; }

    int value() const { return m_value; }

public slots:
    void setValue(int value);

signals:
    void valueChanged(int newValue);

private:
    int m_value;
};

#endif // COUNTER_H
```

counter.h

```cpp
#include "counter.h"

void Counter::setValue(int value)
{
    if (value != m_value) {
        m_value = value;
        emit valueChanged(value);
    }
}
```

counter.cpp

(Quelle: http://doc.trolltech.com/4.5/signalsandslots.html)

Qt Signals and Slots

- Signals und Slots werden über QObject::connect verbunden
- Nicht aufgefangene Signale werden ignoriert
- Verbindungen können über QObject::disconnect wieder gelöst werden

```
Counter a, b;
QObject::connect(&a, SIGNAL(valueChanged(int)),
                 &b, SLOT(setValue(int)));

a.setValue(12);     // a.value() == 12, b.value() == 12
b.setValue(48);
```

main.cpp

(Quelle: http://doc.trolltech.com/4.5/signalsandslots.html)

Qt Meta-Object Compiler

- Qt bietet diverse Erweiterungen zu C++ an, aber arbeitet mit regulären C++-Compilern wie gcc
- Dazu wird der Programmcode zuerst durch den Qt's *moc* (Meta-Object Compiler) in regulären C++-Code umgewandelt
- Schlüsselwörter für moc sind:
  - `Q_OBJECT:`
    - Beginn eines Qt-Objekts, um z.B. `signal, slot, emit` zu verarbeiten
  - `Q_PROPERTY:`
    - Zur Definition von properties (siehe http://doc.trolltech.com/4.5/properties.html)
  - `Q_CLASSINFO:`
    - Für Metadaten

(Quelle: http://doc.trolltech.com/4.5/moc.html)

Qt Beispiel #2

main.cpp

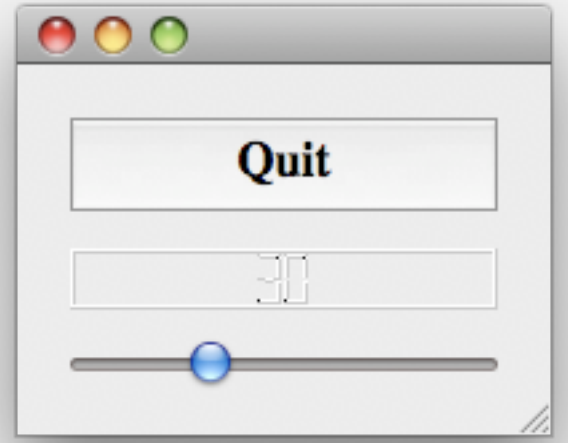
```
#include <QtGUI>

class MyWidget : public QWidget
{
public:
    MyWidget(QWidget *parent = 0);
};

MyWidget::MyWidget(QWidget *parent)
    : QWidget(parent)
{
    QPushButton *quit = new QPushButton(tr("Quit"));
    quit->setFont(QFont("Times", 18, QFont::Bold));

    QLCDNumber *lcd = new QLCDNumber(2);
    lcd->setSegmentStyle(QLCDNumber::Filled);

    QSlider *slider = new QSlider(Qt::Horizontal);
    slider->setRange(0, 99);
    slider->setValue(0);

    connect(quit, SIGNAL(clicked()), qApp, SLOT(quit()));
    connect(slider, SIGNAL(valueChanged(int)),
            lcd, SLOT(display(int)));
```

(Quelle: http://doc.trolltech.com/4.3/tutorial-t5.html)

Qt Beispiel #2

main.cpp (*ctd.*)

```
    QVBoxLayout *layout = new QVBoxLayout;
    layout->addWidget(quit);
    layout->addWidget(lcd);
    layout->addWidget(slider);
    setLayout(layout);
}

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    MyWidget widget;
    widget.show();
    return app.exec();
}
```



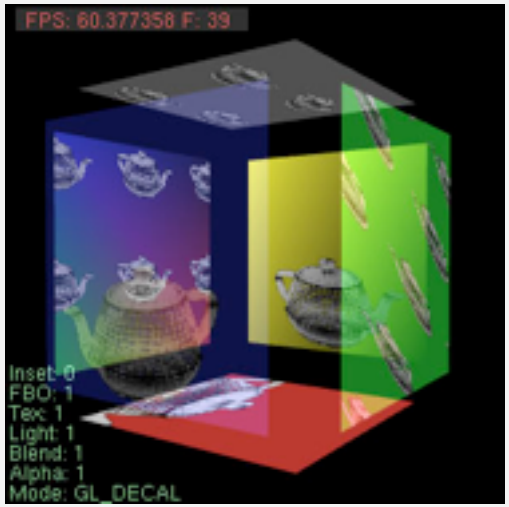
(Quelle: http://doc.trolltech.com/4.3/tutorial-t5.html)

OpenGL

- Computergrafik: Echte Welt -> gerasterte Ausgabe

Grundlegendes

- OpenGL ist eine API zur 3D Programmierung
- Bietet Funktionen zum Zeichnen von 3D-Szenen und Schnittstelle zur Grafikhardware.
- Keinerlei Funktionalität für Zugriff auf/von OS (Fenster, Benutzereingaben usw.)
- Erweiterungen des OpenGL Standards fügen diese Funktionalitäten hinzu.



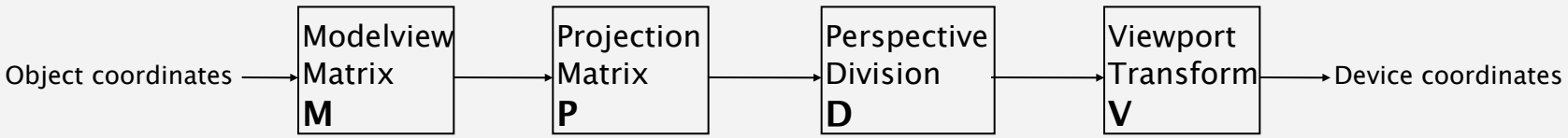
(Quelle: http://commons.wikimedia.org/wiki/Main_Page)

OpenGL Viewing

- OpenGL ist eine State-Machine die das Zeichnen von 3D-Szenen und Animationen erlaubt
- Grafikprimitive werden zusammengesetzt und manipuliert (Statusänderungen)
- Primitive:
  - Punkte
  - Linien
  - Polygone
  - Bitmaps
- Außerdem werden Lichtquellen und Kamera positioniert
- Anschließend wird die Szene ausgegeben
- Das Ganze findet (meistens) in einer Endlosschleife statt

OpenGL Transformationen

- OpenGL nutzt Matrizen um interne 3D-Koordinaten auf den Bildschirm zu transformieren
  1. object space -> eye space via GL_MODELVIEW
  2. eye space -> clip space via GL_PROJECTION
- OpenGL Transformationsmatrizen sind homogen und haben 16 Einträge
- glMatrixMode(*Matrix*) wechselt zwischen verschiedenen Matrizen
- glLoadIdentity() lädt die Identitätsmatrix



OpenGL in Qt

```
#include <QtGui>
#include <QGLWidget>

class GlTest : public QGLWidget
{
    Q_OBJECT

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

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

#endif // GLTEST_H
```

gltest.h

```
#include <QtGui/QApplication>
#include "gltest.h"

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    GlTest w;
    w.show();
    return a.exec();
}
```

main.cpp

OpenGL in Qt

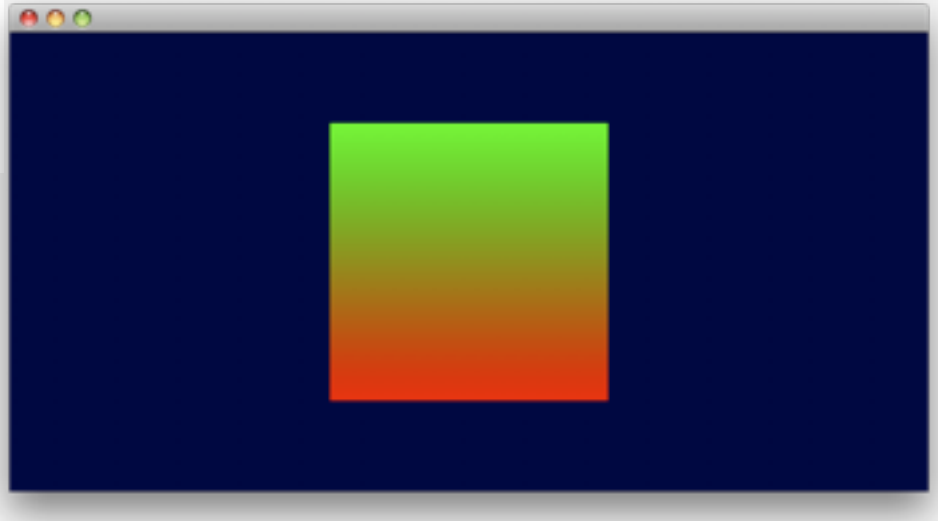
```
#include "gltest.h"

GlTest::GlTest(QWidget *parent)
    : QGLWidget(parent)
{
}

GlTest::~GlTest()
{
}

void GlTest::initializeGL(){
    glClearColor(0.0, 0.0, 0.2, 1.0);
}

void GlTest::resizeGL(int width, int height){
    glViewport(0, 0, (GLint)width, (GLint)height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(45.0f,(GLfloat)width/(GLfloat)height,0.1f,100.0f);
    glMatrixMode(GL_MODELVIEW);
}
```

gltest.cpp

OpenGL in Qt

```
void GlTest::paintGL(){
    glClear(GL_COLOR_BUFFER_BIT);
    glLoadIdentity();
    glTranslatef(0, 0, -4);
    glBegin(GL_POLYGON);
        glColor3f(1.0f, 0.0f, 0.0f); // red
        glVertex3f(-1.0f, -1.0f, 0.0f);
        glVertex3f(1.0f, -1.0f, 0.0f);
        glColor3f(0.0f, 1.0f, 0.0f); // green
        glVertex3f(1.0f, 1.0f, 0.0f);
        glVertex3f(-1.0f, 1.0f, 0.0f);
    glEnd();

}
```

gltest.cpp (*ctd.*)



OpenGL Syntax

- OpenGL definiert eigene numerische Datentypen, die durch Voranstellen eines `GL` gebildet werden (z.B. `GLfloat, GLint, GLvoid`) (können normalerweise durch die regulären Versionen ersetzt werden)
- OpenGL Befehle haben die Struktur
  - `gl`*`Befehlsname`*`{[3/4]{[f/d]}}`
  - Die meisten Befehle werden ohne Zusatz aufgerufen (z.B. `glClearColor, glViewport, glBegin, glEnd`)
  - Bei Befehlen mit mehreren Varianten gibt die Zahl die Anzahl der Parameter und f bzw. d den Datentyp an:
    - `glColor3f(float red, float green, float blue)`
    - `glColor4d(double red, double green, double blue, double alpha)`

(Quelle: http://www.opengl.org)

Projektionen (GL_PROJECTION)

- Zwei grundlegende Projektionsarten: Perspektivisch und orthographisch
  - Bei der perspektivischen Projektion laufen alle Strahlen zu einem Fluchtpunkt zusammen (glFrustrum, gluPerspective)
  - Bei der orthographischen Projektion sind alle Strahlen parallel zueinander (glOrtho)
- `glViewport` definiert die Größe des Zielfensters.

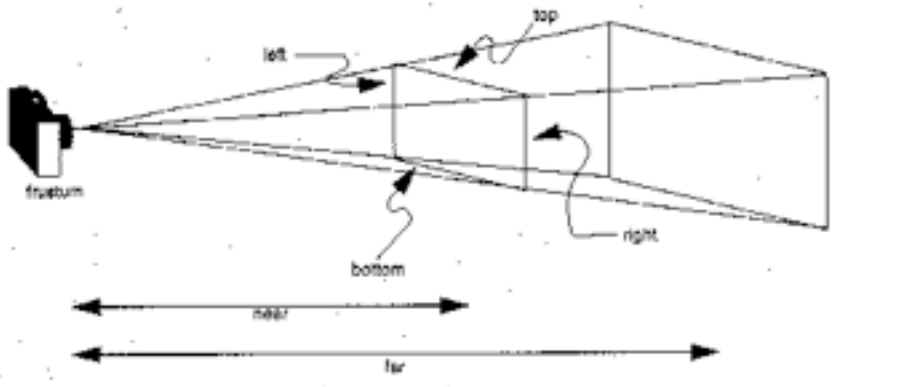


Figure 3-13 Perspective Viewing Volume Specified by glFrustum()

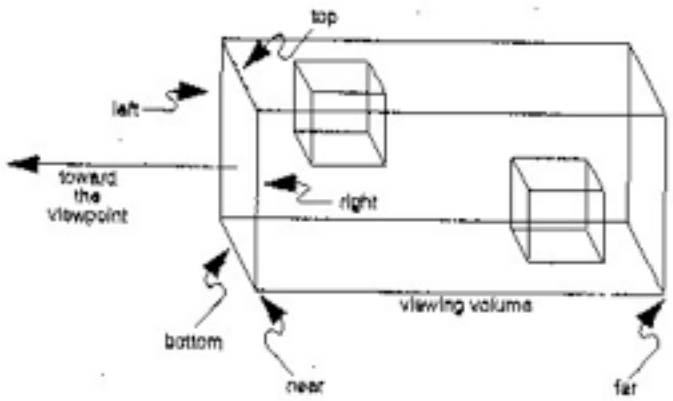


Figure 3-15 Orthographic Viewing Volume

(Quelle: http://www.opengl.org)

Transformationen (GL_MODELVIEW)

- Verschiedene Möglichkeiten Objekte zu transformieren:
  - `glTranslatef(float x, float y, float z)`
    - Verschiebt alle nachfolgenden Objekte entlang der drei Koordinatenachsen (Translation)
  - `glRotatef(float angle, float x, float y, float z)`
    - Rotiert alle nachfolgenden Objekte um Winkel `angle` (in Grad) um eine beliebige Achse (Rotation)
  - `glScalef(float x, float y, float z)`
    - Skaliert alle nachfolgenden Objekte entlang der drei Koordinatenachsen (Skalierung)

(Quelle: http://www.opengl.org)

OpenGL Utilities (GLU) Library

- Definiert in `glu.h`
- Diverse Hilfsfunktionen, besonders praktisch bei Projektionen:
  - `gluPerspective(double fovy, double aspect, double near, double far)`
    - Berechnet die (perspektive) Projektionsmatrix für den Öffnungswinkel fovy mit dem Seitenverhältnis aspect (Breite / Höhe)
  - `gluLookAt(double eyex, double eyey, double eyez, double centerx, double centery, double centerz, double upx, double upy, double upz)`
    - Berechnet die Projektionsmatrix für eine (gedachte) Kamera an der Position (eyex, eyey, eyez) mit Blick auf (centerx, centery, centerz) und oben (upx, upy, upz)

(Quelle: http://www.opengl.org)

2D Objekte

- Im Beispiel:

```
glBegin(GL_POLYGON);
    glColor3f(1.0f, 0.0f, 0.0f); // red
    glVertex3f(-1.0f, -1.0f, 0.0f);
    glVertex3f(1.0f, -1.0f, 0.0f);
    glColor3f(0.0f, 0.0f, 1.0f); // blue
    glVertex3f(1.0f, 1.0f, 0.0f);
    glVertex3f(-1.0f, 1.0f, 0.0f);
glEnd();
```

- Weitere Formen:

```
GL_POINTS, GL_LINES (je 2 Punkte verbunden)
GL_LINE_STRIP, GL_LINE_LOOP
GL_POLYGON, GL_QUADS, GL_TRIANGLES
```

- 3D-Objekte müssen aus 2D-Objekten zusammengesetzt werden (s. Übungsblatt)

(Quelle: http://www.opengl.org)

OpenGL in Qt #2

gltest.h

```
#include <QtGui>
#include <QGLWidget>

class GlTest : public QGLWidget
{
    Q_OBJECT

public:
    GlTest(QWidget *parent = 0);
    ~GlTest();
    QSize sizeHint() const;

public slots:
    void rotateObject(int nuval);

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

private:
    float rotaY;
};

#endif // GLTEST_H
```

gltest.cpp

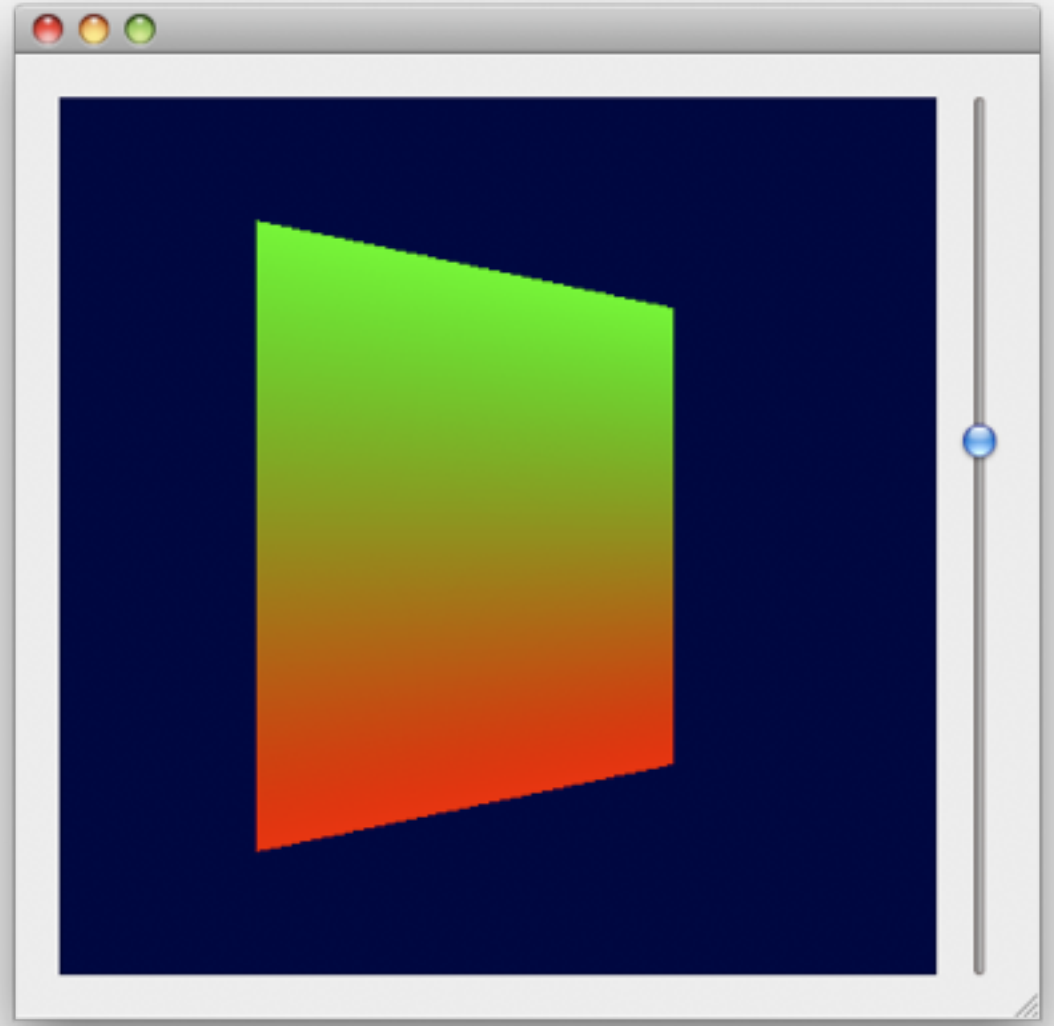
```
void GlTest::paintGL(){
    glClear(GL_COLOR_BUFFER_BIT);
    glLoadIdentity();
    glTranslatef(0, 0, -4);
    glRotatef(rotaY, 0, 1, 0);
    glBegin(GL_POLYGON);
        glColor3f(1.0f, 0.0f, 0.0f); // red
        glVertex3f(-1.0f, -1.0f, 0.0f);
        glVertex3f(1.0f, -1.0f, 0.0f);
        glColor3f(0.0f, 1.0f, 0.0f); // green
        glVertex3f(1.0f, 1.0f, 0.0f);
        glVertex3f(-1.0f, 1.0f, 0.0f);
    glEnd();

}

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

void GlTest::rotateObject(int nuval){
    rotaY = nuval;
    updateGL();
}
```

OpenGL in Qt #2

```
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);

    QWidget* win = new QWidget;
    QSlider rotateSlider;
    rotateSlider.setRange(0, 360);
    GlTest w;

    QHBoxLayout* layout = new QHBoxLayout;
    layout->addWidget(&w);
    layout->addWidget(&rotateSlider);

    win->setLayout(layout);

    QObject::connect(&rotateSlider, SIGNAL(valueChanged(int)),
                     &w, SLOT(rotateObject(int)));

    win->show();
    return a.exec();
}
```

main.cpp

OpenGL in Qt #2



Weiterführende Literatur (Qt)

- Jasmin Blanchette, Mark Summerfield: “C++ GUI Programming with Qt 4”, ISBN-13: 978-0132354165
  Erste Edition kostenlos online: http://www.qtrac.eu/C++-GUI-Programming-with-Qt-4-1st-ed.zip

- http://doc.trolltech.com/4.5/
- http://www.qtsoftware.com/products/

Weiterführende Literatur (OpenGL)

- http://www.opengl.org/sdk/docs/man/
- OpenGL 'Redbook': http://fly.srk.fer.hr/~unreal/theredbook/
- NeHe OpenGL Tutorials: http://nehe.gamedev.net/