Ludwig-Maximilians-Universität München
LFE Medieninformatik
Prof. Dr. Andreas Butz
Dominikus Baur, Sebastian Boring

Sommersemester 2009
Übungsblatt 2
01.Mai 2009

Computergrafik 1

Abgabetermin:
Die Lösung zu diesem Übungsblatt ist bis zum Freitag den **8. Mai 2009, 12:00 Uhr s.t.** über UniWorx abzugeben.

Inhalt:
Dieses Blatt dient zum Bekanntmachen mit dem Windowing Toolkit `Qt`. Hierzu erzeugen Sie ein Fenster, das bereits die vier Ansichten eines gewöhnlichen 3D Modellierungswerkzeugs (z.B. *3D Studio Max* oder *Cinema 4D*) enthält. Zusätzlich erlenen Sie den Umgang mit der Entwicklungsumgebung *Qt Creator*. Pro Aufgabe können maximal zehn (in Aufgabe 7 maximal 20) Punkte erreicht werden. Zum Bestehen des Übungsblatts müssen in **JEDER** Aufgabe mindestens fünf (in Aufgabe 7 mindestens 10) Punkte erreicht werden.

Geben Sie zu jeder Aufgabe alle benötigten Header-, Source-, und Projektdateien (von *Qt Creator* erzeugt) mit ab, d.h. `*.h`, `*.cpp` und `*.pro` Dateien. Abgaben die nicht kompilieren werden mit 0 Punkten bewertet. Fassen Sie alle Aufgaben zu einer zip-Datei zusammen und laden Sie sie bei UniWorx hoch. Für jede Aufgabe soll ein Ordner mit dem Namen „auf1-X“ angelegt werden.

Aufgabe 4 Qt Fenster (10 Punkte)

In dieser Aufgabe erlernen Sie den Umgang mit der Entwicklungsumgebung *Qt Creator* und erzeugen ein neues Fenster. Dieses Fenster wird in den folgenden Aufgaben (auch über dieses Übungsblatt hinaus) weiter verwendet.

Laden Sie sich dafür zunächst die Entwicklungsumgebung (hier *Qt SDK: Complete Development Environment*) von der Seite `http://www.qtsoftware.com/downloads` für Ihr Betriebssystem herunter und installieren Sie diese. Starten Sie anschließend *Qt Creator*.

a) Erzeugen Sie ein neues Projekt vom Typ *Qt4 Gui Application* und nennen Sie es `CG1_3D`. Fügen Sie auf der nächsten Seite des Dialogs das Modul *QtOpenGL* hinzu, da dies für die weiteren Aufgaben benötigt wird. Auf der folgenden Seite könnten Sie nun die Namen der Hauptklassen ändern. Lassen Sie jedoch alle Einstellungen wie sie sind, entfernen aber das Häkchen bei *Generate from*, damit keine Datei vom Typ `*.ui` zusätzlich erstellt wird (**2 Punkte**).

b) Geben Sie dem Hauptfenster den Titel `Computer Graphics 1 :: 3D` mit der Funktion `setWindowTitle` im Konstruktor des Hauptfensters (**2 Punkte**).

c) Erzeugen Sie ein Layout für das Fenster. Fügen sie dieses Layout als Klassenvariable hinzu, um es später in anderen Methoden verwenden zu können. Verwenden Sie ein geeignetes Layout, um später die vier Ansichten in einer 2×2 Matrix dargestellen zu können. Initialisieren Sie dieses ebenfalls im Konstruktor des Hauptfensters (**3 Punkte**).

d) Erstellen sie zusätzlich eine Klassenvariable vom Typ `QWidget`. Initialisieren Sie dies im Konstruktor des Hauptfensters und geben Sie diesem das zuvor erstellte Layout mittels der

Funktion `setLayout`. Dieses `QWidget` soll nun dem Hauptfenster als zentrales Widget dienen (Funktion `setCentralWidget`) (**3 Punkte**).

Aufgabe 5 Die vier *OpenGL* Ansichten (10 Punkte)

Nachdem Sie Fenster und Layout erstellt haben, fügen Sie in dieser Aufgabe die vier *OpenGL* Ansichten hinzu, in denen später die 3D-Szene (aus verschiedenen Kamera-Perspektiven) dargestellt werden soll. Laden Sie sich die Szene-Dateien von der Übungs-Homepage herunter. Das Archiv enthält zwei Dateien `scene.h` und `scene.cpp`.

a) Binden Sie beide Dateien in Ihr Projekt ein. Die Szene enthält momentan nur eine Funktion `render`, die später aus den Ansichten heraus aufgerufen werden soll. Erstellen sie nun eine Klassenvariable vom Typ `Scene` im Hauptfenster (**2 Punkte**).

b) Erstellen Sie ein neues Objekt `GLWidget`, d.h. die Dateien `glwidget.h` und `glwidget.cpp`. Dieses Objekt soll von der Klasse `QGLWidget` erben. Binden Sie dafür den notwendigen Header ein. Achten Sie zudem darauf, dass dieses Objekt das Attribut `Q_OBJECT` erhält. Als Argument für den Konstruktor soll die im Hauptfenster erstellte Szene übergeben werden. Die Szene soll dann auch in jedem `GLWidget` als Klassenvariable gespeichert (aber nicht verändert) werden (**2 Punkte**).

c) Überschreiben Sie nun im Header folgende Funktionen und implementieren Sie diese:

- **`void`** `initializeGL()`:
  In dieser Funktion müssen Sie die grundlegenden *OpenGL* Einstellungen vornehmen, d.h. Einstellungen, die nach dem Erstellen nicht mehr geändert werden. In diesem Übungsblatt benötigen Sie lediglich die Tiefenüberprüfung (`GL_DPETH_TEST`) und das Shader Modell (`GL_SMOOTH`).
- **`void`** `paintGL()`:
  Diese Funktion ist verantwortlich für das Zeichnen der Szene und wird bei jedem Aufruf von `updateGL()` ausgeführt. Anders als bei Computerspielen, verwenden wir hier keine Schleife, die permanent die Szene neu zeichnet, sondern veranlassen das Zeichnen nur, wenn sich etwas ändert. Da Sie mehrere *OpenGL* Ansichten erstellen werden, müssen bei jedem Aufruf die Funktionen `glViewport` und `gluPerspective` ausgeführt werden. Denken Sie hierbei daran, den Matrix-Modus mittels der Funktion `glMatrixMode` zuvor auf `GL_PROJECTION` und anschließend wieder auf `GL_MODELVIEW` zu setzen. Laden Sie zusätzlich nach jedem Wechsel des Matrix-Modus die Einheitsmatrix mit der Funktion `glLoadIdentity`. Nachdem Sie die wieder die Modelview-Matrix geladen haben, verwenden Sie `qglClearColor` (setzen der Hintergrundfarbe) sowie `glClear` (Löschen des Farb- und Tiefenpuffers) und lassen dann die Szene mit der Funktion `render` der Klasse `Scene` rendern.
- **`void`** `resizeGL(`**`int`** `width, `**`int`** `height)`:
  Diese Funktion wird aufgerufen, wenn sich Breite oder Höhe der Ansicht geändert haben. Die Parameter sind dann die **neue** Breite sowie die **neue** Höhe. In dieser Funktion werden (sofern nur eine Ansicht vorhanden ist) die Projektions-Parameter (wie bei `paintGL` beschrieben) eingestellt. Führen Sie also die gleichen Schritte aus, die Sie zu Beginn der Funktion `paintGL` durchführen. Sie können für das Setzen der Parameter auch eine eigene Klassen-Funktion erstellen, die dann von den Funktionen `resizeGL` und `paintGL` aufgerufen wird.

(**4 Punkte**)

d) Nun müssen diese Ansichten noch dem Hauptfenster zugeordnet werden. Erstellen sie hierzu ein zweidimensionales Array (2 Reihen und zwei Spalten) vom Typ `GLWidget` als Klassenvariable im Hauptfenster. Initialisieren Sie diese im Konstruktor des Hauptfensters und fügen SIe diese Ihrem zuvor erstellten Layout hinzu. Achten Sie dabei auf eine korrekte Platzierung in der 2×2 Matrix (**2 Punkte**).

Aufgabe 6 Verschiedene Kamera-Ansichten (10 Punkte)

Da jede Ansicht bisher die selbe Perspektive verwendet wird, werden Sie in dieser Aufgabe verschiedene Ansichten der Szene erstellen. Dazu werden Sie den Umgang mit `gluPerspective` sowie `glOrtho` erlernen. In der Computergrafik ist es üblich nicht die Kamera, sondern die Szene relativ zur Kamera zu verschieben, was aber zu dem gleichen Ergebnis führt.

a) Erzeugen Sie sich zunächst ein Objekt namens `Camera`, also erneut zwei Dateien `camera.h` und `camera.cpp`. Eine Kamera hat eine dreidimensionale Position, sowie eine bestimmte Drehung um jede der drei Achsen. Berücksichtigen Sie dies auch im Konstruktor, erlauben Sie jedoch auch die eine Standard-Initialisierung der Kamera (**2 Punkte**).

b) Die Kamera soll in einer späteren Aufgabe verschoben werden können. Erzeugen Sie hierfür eine Funktion `moveBy`, der drei Parameter übergeben werden. Diese Parameter beschreiben, **wie weit** die Kamera in der jeweiligen Richtung verschoben werden soll (**2 Punkte**).

c) Erzeugen Sie für jedes `GLWidget` eine eigene Kamera. Berücksichtigen Sie dabei:

- Die Ansicht an Position $(0,0)$ (d.h. links oben) ist die Top-Ansicht, d.h. die Kamera sieht von **oben** auf die Szene. Die Kamera in der Ansicht an Position $(1,0)$ (d.h. rechts oben) sieht von **links** auf die Szene. Die Ansicht an Position $(0,1)$ (d.h. links unten) ist die Front-Ansicht, d.h. die Kamera sieht von **vorne** auf die Szene. Die Ansicht an Position $(1,1)$ (d.h. rechts unten) stellt die Perspektive dar, d.h. die Kamera hat eine **freie Position** in Bezug auf die Szene.
- Die Kameras haben jeweils einen Abstand von 100 bezüglich der Szene. Das heißt, dass die Kameras in den ersten drei Ansichten jeweils um 100 Einheiten auf ihrer jeweiligen Achse verschoben sind (**Tipp:** beachten Sie das Vorzeichen!). Die perspektivische Kamera ist um 100 Einheiten auf allen drei Achsen verschoben (auch hier muss das Vorzeichen bei jeder Achse beachtet werden).
- Geben Sie den Kameras auf ihre Position passende Rotationen (in Grad). Lediglich die Kamera für die perspektivische Ansicht muss um zwei Achsen gedreht werden. Alle anderen Kameras jedoch nur um eine Achse. **Tipp:** Die Kamera der perspektivischen Ansicht ist auf allen Achsen um die gleiche Weite verschoben worden, daher ist der Winkel ein Vielfaches von $45°$.
- Bedenken Sie, dass die ersten drei Ansichten **orthografische** Projektionen sind, d.h. verwenden Sie hier die Methode `glOrtho` anstelle der Funktion `gluPerspective` bevor Sie die Szene rendern.

(**4 Punkte**)

d) In der Computergrafik ist es üblich nicht die Kamera, sondern die Szene relativ zur Kamera zu verschieben, was aber zu dem gleichen Ergebnis führt. In Ihrem Fall soll ebenfalls die Szene relativ zur Kamera bewegt werden. Fügen Sie daher nach dem Laden der Einheitsmatrix (Matrix-Modus beachten) aber **bevor** Sie die Szene rendern noch die nötigen Transformationen ein. Verwenden Sie hierzu die Funktionen `glTranslated` und `glRotated` (**Tipp:**

Stellen Sie sich vor, dass Sie die Kamera um 10 Einheiten nach links bewegen und überlegen Sie sich wie das Ergebnis aussehen würde. Überlegen Sie sich nun, wie sie die Szene anstelle der Kamera bewegen müssten, um das gleiche Ergebnis zu erzielen!). Achten Sie ferner auf die Reihenfolge der Transformationen. Es hat einen großen Einfluss auf das Ergebnis, ob Sie zunächst die Rotationen und dann die Translationen ausführen oder ob Sie die Transformationen umgekehrt anwenden (**Tipp:** Überlegen Sie sich, was es bedeutet erst zu rotieren und dann zu verschieben, bzw. erst zu verschieben und dann zu rotieren!). (**2 Punkte**)

Aufgabe 7 Interaktion in den Ansichten (20 Punkte)

Nachdem die Ansicht erstellt worden ist, müssen nun noch grundlegende Interaktionsmöglichkeiten geschaffen werden. Hierbei verwenden wir Tastatur und Maus, wie sie aus kommerziellen 3D-Modellierungstools (z.B. *3D Studio Max*) bekannt sind.

a) Überschreiben Sie nun die Mausfunktionen im Header `glwidget.h` folgende Funktionen und implementieren Sie diese in der Source-Datei `glwidget.cpp`:

- **`void`** `mousePressEvent(`**`QMouseEvent`** `*event)`:
  Diese Funktion wird aufgerufen, sobald ein Klick innerhalb eines Widgets erfolgt. Aus dem `QMouseEvent` erhalten Sie die benötigten Information, d.h. welche Taste gedrückt wurde, und an welcher Position. In dieser Aufgabe reagieren wir nur auf die **linke** Maustaste! **Tipp:** Speichern Sie sich bei diesem Event gleich die Position an der das Event aufgetreten ist, da Sie es später eventuell noch verwenden müssen.
- **`void`** `mouseMoveEvent(`**`QMouseEvent`** `*event)`:
  Diese Funktion wird immer dann aufgerufen, wenn sich die Maus über dem Widget bewegt (standardmäßig nur dann, wenn auch eine Taste gedrückt ist). Überprüfen Sie hier, ob die linke Taste gedrückt ist und - falls dies der Fall ist - verschieben sie die Kamera entsprechend entlang der $x-$ und $y-$Achse des Widgets. **Achtung:** Bei der Kamera der perspektivischen Ansicht müssen die alle drei Achsen entsprechend angepasst werden.
- **`void`** `mouseReleaseEvent(`**`QMouseEvent`** `*event)`:
  Dieses Event tritt ein, wenn die Maustaste losgelassen wird. Für diese Aufgabe ist dieses Event jedoch noch nicht wichtig, daher müssen Sie nur eine leere Implementierung in die Source-Datei schreiben.

(**8 Punkte**)

b) Überschreiben Sie nun die Funktion **`void`** `wheelEvent(`**`QWheelEvent`** `*event)`. Dieses Ereignis wird ausgelöst, wenn das Mausrad gedreht wurde. Der Wert ist (eventuell abhängig vom Betriebssystem) normalerweise 120 oder -120 (je nachdem in welche Richtung das Mausrad gedreht wurde. Da Sie mit den zwei-dimensionalen Mauskoordinaten die Kamera bisher nur links/rechts bzw. oben/unten steuern konnten, soll das Mausrad dazu verwendet werden, die Kamera nach vorne bzw. hinten zu bewegen. Das heisst, dass die Kamera näher an die Szene hin oder weiter von der Szene wegbewegt werden kann. **Achtung:** Da dieses Event mit 120 einen recht hohen Wert gibt, sollte dieser noch angepasst werden, damit sich die Kamera bei einer Drehung nicht zu weit bewegt (**2 Punkte**).

c) Überschreiben Sie nun die Tastaturfunktionen im Header `glwidget.h` folgende Funktionen und implementieren Sie diese in der Source-Datei `glwidget.cpp` (**Hinweis:** das Widget muss aktiv sein, d.h. sobald ein Mausklick in einem Widget erfolgt, muss die Funktion `setFocus` mit dem Parameter `Qt::MouseFocusReason` aufgerufen werden!):

- **void** `keyPressEvent(`**QKeyEvent** `*event)`:
  Diese Funktion wird aufgerufen, sobald eine Taste gedrückt wird. In dieser Aufgabe ist dies jedoch unerheblich, daher können Sie hier ebenfalls eine leere Implementierung angeben.
- **void** `keyReleaseEvent(`**QKeyEvent** `*event)`:
  Dieses Event tritt ein, wenn eine Taste (und eventuelle Modifizierungs-Tasten, z.b. `Alt`, `Control` oder `Shift`) losgelassen wird. Überprüfen Sie hier, ob die Tastenkombination `Alt + W` gedrückt (bzw. losgelassen) wurde.

Falls dies der Fall war, soll die aktuelle Ansicht im Hauptfenster über allen vier Ansichten gezeichnet werden. Erstellen Sie dazu ein `signal` namens `switchView`, dass beim Erstellen eines Widgets an die Methode `switchGlWidget` im Hauptfenster gebunden wird. Wird nun die geforderte Tastenkombination im widget gedrückt, rufen Sie `emit switchView()` auf, wodurch das Hauptfenster benachrichtigt wird. In diesem werden nun **alle** Ansichten vom Layout entfernt (**Hinweis:** `removeWidget`) und unsichtbar gemacht (**Hinweis:** `hide`). Das Widget, das das Event ausgelöst hat (**Hinweis:** zu erhalten durch `qobject_cast <GLWidget*>(sender())`) wird dann auf sichtbar gesetzt (**Hinweis:** `show`) und dem Layout wieder hinzugefügt (**Tipp:** Sie können bei einem geeigneten Layout auch die Anzahl der Zeilen und Spalten angeben, die das Widget überdecken soll). Schalten Sie dabei auch dieses Widget wieder aktiv (wie oben beschrieben, um weitere Tastatur-Events zu erhalten. Durch erneutes Drücken der Tastenkombination, wird wieder der alte Zustand hergestellt (**8 Punkte**).

d) Rufen Sie bei allen nicht verwendeten Events (d.h. in allen leeren Funktions-Blöcken) die Funktion `ignore` auf dem Event auf. Dadurch kann das Event an andere Objekte weitergegeben werden, was im Verlauf der Übung hilfreich sein kann (**2 Punkte**).