



# Programmieren mit *OpenGL* und *GLUT*

Manfred Brill

Oktober 2005

# Inhaltsverzeichnis

<b>1</b>	<b>OpenGL</b>	<b>1</b>
1.1	Bibliotheken und Header-Dateien . . . . .	1
1.2	Namenskonventionen . . . . .	2
1.3	<i>OpenGL</i> Datentypen . . . . .	2
<b>2</b>	<b>GLUT</b>	<b>4</b>
<b>3</b>	<b>GLUT initialisieren und ein <i>OpenGL</i>-Fenster erzeugen</b>	<b>6</b>
3.1	Callback-Funktionen für die grafische Ausgabe . . . . .	7
3.2	Callback-Funktionen für Peripherie-Geräte . . . . .	7
3.3	Event-Loop . . . . .	9
<b>4</b>	<b>Eine <i>OpenGL</i>-Schablone</b>	<b>11</b>
4.1	Hauptprogramm . . . . .	11
4.2	Die Implementierung der Callbacks . . . . .	12
4.3	Ein Makefile für <i>Cygwin</i> und <i>Linux</i> . . . . .	17
	<b>Literatur</b>	<b>19</b>

# Kapitel 1

## OpenGL

OpenGL als 3D-API wird von einem *Architecture Review Board* entwickelt, dem unter anderem *Silicon Graphics*, *IBM*, *Hewlett-Packard*, *SUN*, *ATI* und *nvidia* angehören. Die Ursprünge von OpenGL stammen von *Silicon Graphics*. Dort hieß das API *GL*, eine Abkürzung für *Graphics Library*. Ein wesentlicher Unterschied zwischen *GL* und *OpenGL* ist, dass *OpenGL* keinerlei Funktionen enthält, die in irgendeiner Weise mit dem Fenstersystem, sei es *X11* oder *Microsoft Windows*, kommuniziert.

*OpenGL* enthält über 200 Funktionen. Wenn Sie vorhaben, häufig mit dem API zu arbeiten sind die bei Addison-Wesley erschienenen Bücher *OpenGL Reference Manual* („blue book“, [WND00]), *OpenGL Programming Guide* („red book“, [KF00]) und *OpenGL Shading Language* („orange book“, [Ros04]) ein absolutes Muss! Die Integration in *Microsoft Windows* ist sehr gut in [Fos96] (das „white book“) und [WS04] beschrieben. *OpenGL* und *X11* wird in [Kil96b] („green book“) dargestellt.

Dieser Text stellt den prinzipiellen Aufbau eines OpenGL-Programms dar, das für die vom Windowsystem abhängigen Bestandteile *GLUT* verwendet. Zum Teil sind die Vorschläge subjektiv und aus der persönlichen Erfahrung des Autors gewonnen. Sehen Sie also den Inhalt als Vorschlag – und entwickeln Sie in Ihrer Arbeit mit OpenGL den für Sie und Ihre Projekte passenden Aufbau.

Mehr zu *OpenGL* und den in diesem Text verwendeten Begriffe finden Sie in [BB05], [Hil01], [MB05] oder [Ang05].

### 1.1 Bibliotheken und Header-Dateien

*OpenGL* besteht aus zwei Bibliotheken. Einmal dem Kern des API, der unter *UNIX* als `libGL.so` oder `libGL.a` in `/usr/lib` zu finden ist. Für *Microsoft Windows* suchen Sie entsprechend nach `opengl32.lib` oder `opengl32.dll`. Daneben gibt es die *OpenGL Utility Library* oder kurz *GLU* (für *UNIX* ist dies `libGLU.so` bzw. `libGLU.a`, für *Microsoft Windows* `glu32.lib` bzw. `glu32.dll`). Die Deklarationen der in diesen beiden Bibliotheken enthaltenen Funktionen sind in den Dateien `Gl.h` und `Glu.h` enthalten. Im *UNIX*-Betriebssystem finden Sie diese beiden Header-Dateien im Verzeichnis `/usr/include/GL`; für *Microsoft Windows* und *Microsoft Visual* in einem entsprechenden Verzeichnis. Die für *Microsoft Windows* zur Verfügung gestellten Bibliotheken und Header-Dateien entsprechen *nicht* dem aktuellen Stand der *OpenGL*-Spezifikation 2.0. Für *ATI* oder *nVidia*-Grafikkarten erhalten Sie auf den Web-Seiten der Hersteller entsprechend an die *OpenGL*-Treiber der Karten angepasste Versionen. Alle führenden Hersteller haben sehr ausführliche *Developer*-Sites; Links dazu finden Sie auf [www.vislab.de](http://www.vislab.de). Für *Linux* gibt es entweder eine Software-Implementierung von *OpenGL*, die *MESA*-Bibliothek. Dies ist eine langsame, aber komplette Implementierung von *OpenGL*. Die großen Grafikkarten-Hersteller bieten auch für *Linux* entsprechende Treiber an; bei der Installation dieser Treiber

werden im Normalfall auch die Headerdateien und Bibliotheken installiert.

## 1.2 Namenskonventionen

Ein Programm, das *OpenGL* verwendet sind also die beiden `include`-Anweisungen

```
#include <GL/gl.h>
#include <GL/glu.h>
```

enthalten. Es soll hier nicht der Versuch unternommen werden, die im *API* enthaltenen Funktionen aufzuzählen. Prinzipiell gilt in *OpenGL* die Namenskonvention, dass die Funktionen in der Kernbibliothek mit dem Präfix `gl` versehen sind, beispielsweise

```
glClear(GL_COLOR_BUFFER_BIT);
```

für das Löschen des aktuellen Fensterinhalts. Funktionen aus der *Utility Library* haben den Präfix `glu`, beispielsweise

```
gluNewQuadric();
```

für das Anlegen einer neuen Quadrik.

Die Konstanten, die in den Headerdateien definiert sind werden immer groß geschrieben; und beginnen entsprechend mit `GL` und `GLU`. Einzelne Worte werden mit einem Unterstrich getrennt. Beispiele dafür sind `GL_MODELVIEW` oder `GL_COLOR_BUFFER_BIT`.

## 1.3 OpenGL Datentypen

*OpenGL* arbeitet intern mit spezifischen Datentypen für ganze Zahlen und Gleitkommazahlen. Es gibt zwar inzwischen eine IEEE-Norm für Gleitkommaarithmetik, aber eine Variable vom Typ `int` ist manchmal eine 16-Bit Größe, manchmal eine 32-Bit Größe, je nach Betriebssystem und Compiler. Aus diesem Grund sind in *OpenGL* eigene Datentypen vorgesehen, die Sie auch verwenden sollten, um sicher zustellen, dass Ihr Programmcode portabel bleibt. Tabelle 1.1 enthält die einzelnen Datentypen.

**Tabelle 1.1:** OpenGL Datentypen und Suffixes

OpenGL Datentyp	Datentyp	Entsprechender C/C++-Typ	Suffix
<code>GLbyte</code>	8-Bit Integer	<code>signed char</code>	<code>b</code>
<code>GLshort</code>	16-Bit Integer	<code>short</code>	<code>s</code>
<code>GLint</code> , <code>GLsizei</code>	32-Bit Integer	<code>long</code>	<code>i</code>
<code>GLfloat</code> , <code>GLclampf</code>	32-Bit Gleitkommazahl	<code>float</code>	<code>f</code>
<code>GLdouble</code> , <code>GLclampd</code>	64-Bit Gleitkommazahl	<code>double</code>	<code>d</code>
<code>GLboolean</code> , <code>GLubyte</code>	8-Bit Integer ohne Vorzeichen	<code>unsigned char</code>	<code>ub</code>
<code>GLushort</code>	16-Bit Integer ohne Vorzeichen	<code>unsigned short</code>	<code>us</code>
<code>GLenum</code> , <code>GLbitfield</code> , <code>GLuint</code>	32-Bit Integer ohne Vorzeichen	<code>unsigned int</code>	<code>ui</code>

Eine ganze Reihe von *OpenGL*-Funktionen, allen voran `glVertex` zur Übergabe von Eckpunkt-Koordinaten, sehen vor, dass neben dem Stammnamen die Anzahl der übergebenen Argumente, gefolgt vom verwendeten Datentyp der Argumente übergeben werden. Sie werden häufig die Notation `glVertex*` in der Literatur finden; damit soll angegeben werden, dass je nach übergebenen Argumenten der richtige Aufruf gebildet werden muss. Tabelle 1.1 enthält die Suffixe für die in *OpenGL* vorgesehenen Datentypen. Ist das Argument ein Feld, dann wird dem Suffix noch `v` beigefügt.

Im folgenden Quelltext sind an Hand der Funktionen `glVertex*` und `glColor*` einige mögliche Aufrufe dargestellt:

```
glVertex2i(100, 50);           // Zwei Koordinaten vom Typ GLint.
glVertex3f(100.0, 50.0, 10.0); // Drei Koordinaten vom Typ GLfloat.
glVertex4s(100, 50, 10, 1);    // Punkt in homogenen Koordinaten,
                               // vom Typ GLshort.

GLdouble point[3] = {100.0, 50.0, 1.0};
glVertex3dv(point);           // Drei Koordinaten vom Typ GLfloat,
                               // als Feld übergeben.

glColor3f(1.0, 0.0, 0.0);     // Die Farbe auf rot setzen.
glColor4d(1.0, 0.0, 0.0, 1.0); // Die Farbe auf rot setzen,
                               // mit Alpha-Wert 1.0.

GLfloat green[3] = {0.0, 1.0, 0.0};
glColor3fv(green);           // Die Farbe grün, als Feld übergeben.
```

# Kapitel 2

## GLUT

Wie bereits im letzten Abschnitt bemerkt enthält *OpenGL* keine Funktionen, die in irgendeiner Weise mit dem verwendeten Fenstersystem kommunizieren. Es gibt natürlich die Möglichkeit, *OpenGL* mit dem jeweils verwendeten Oberflächen-System zu verheiraten. Auf der Basis der Arbeit für dieses Buch hat Mark Kilgard das *GL Utility Toolkit* oder kurz *GLUT* entwickelt. Diese Bibliothek hat das Ziel, eine einheitliche Schnittstelle zum verwendeten Fenstersystem anzubieten. Der Schwerpunkt liegt hier nicht auf der Performanz, sondern auf Portabilität. *GLUT* steht auf alle relevanten *UNIX*-Plattformen, insbesondere *LINUX*, *MacOS* und *Microsoft Windows* zur Verfügung.

Wie für *OpenGL* gibt es eine Bibliothek und eine Header-Datei. Diese installieren Sie zweckmässiger Weise in das Verzeichnis, in dem auch die beiden *OpenGL*-Header abgelegt sind. Verwenden Sie den *GLUT*, dann reicht eine Anweisung

```
#include <GL/glut.h>
```

Diese Datei stellt sicher, dass die *OpenGL*-Header-Dateien ebenfalls hinzugefügt werden. Eine Einführung in *GLUT* finden Sie im Anhang des „red book“ [WND00] oder in [Kil96b].

**Tip:**

Im WWW finden Sie neben der Bibliothek und den Header-Dateien für *GLUT* auch ein PDF-Dokument mit der aktuellen Dokumentation ([Kil96a]); zur Zeit zur Version 3.7.

**Tip:**

Links zum Download finden Sie auf der Website [www.vislabs.de](http://www.vislabs.de).

Wie bei *OpenGL* wird auch hier eine Namenskonvention eingehalten; Funktionen haben *immer* den Präfix `glu`; die Konstanten aus `glut.h` werden groß geschrieben und beginnen immer mit `GLUT`. `chapter`Ein typisches *GLUT*-Hauptprogramm Ein typisches *GLUT*-Hauptprogramm in C oder C++ enthält Anweisungen, mit deren Hilfe ein Fenster für die *OpenGL*-Ausgaben erzeugt und initialisiert werden kann. Bevor diese Funktionen im Einzelnen betrachtet werden hier ein Hauptprogramm, das in den folgenden Abschnitten im Einzelnen erläutert wird.

```
/* -----  
 *   Muster eines Hauptprogramms mit OpenGL und GLUT  
 * ----- */  
int main(int argc, char **argv)  
{  
    /* GLUT initialisieren */
```

```
glutInit(&argc, argv);
/* Ausgabezustand festlegen
glutInitDisplayMode( GLUT_DOUBLE | GLUT_RGBA);
/* Fenstergröße und -position festlegen */
const int WSizeX = 1024, WSizeY = 768;
glutInitWindowSize(WSizeX, WSizeY);
glutInitWindowPosition(100,100);
/* Das Fenster erzeugen und Titeltext definieren */
glutCreateWindow("Mein erstes OpenGL-Programm");

/* GLUT den display-Callback übergeben */
glutDisplayFunc(display);
/* GLUT den Reshape-Callback übergeben */
glutReshapeFunc(reshape);
/* GLUT den Tastatur-Callback übergeben */
glutKeyboardFunc(keyboard);
/* GLUT den Maustasten-Callback übergeben */
glutMouseFunc(mouse);
/* GLUT den Maus-Callback übergeben */
glutMouseMotionFunc(mouseMotion);
/* GLUT den Idle-Callback übergeben */
glutIdleFunc(idle);
/* Weitere Initialisierungen durchführen,
   Hintergrundfarbe des Fensters, Kamera, ... */
init();
/* Die GLUT-Eventloop starten */
glutMainLoop();
return 0;
}
```

## Kapitel 3

# GLUT initialisieren und ein OpenGL-Fenster erzeugen

Die erste GLUT-Funktion, die Sie aufrufen ist

```
void glutInit(int argc, char **argv);
```

Welche Kommandozeilen-Parameter Sie übergeben hängt vom verwendeten Fenstersystem ab. Die `glutInit` übergebenen Parameter sollten mit denen übereinstimmen, die auch `main` erhalten hat.

Anschließend legen wir mit

```
void glutInitDisplayMode(unsigned int mode);
```

fest, wie die Darstellung im später erzeugten Fensters erfolgen soll. Hier gibt es eine ganze Menge von Optionen, beispielsweise ob mit *double buffering* gearbeitet werden soll, oder ob eine Farbtabelle (`GLUT_INDEX`) oder mit frei mischbaren RGBA-Farben gearbeitet wird. Wie Sie diese beiden Optionen einstellen sehen Sie im oben dargestellten Muster-Hauptprogramm. Der Default-Zustand ist durch die Maske `GLUT_RGBA | GLUT_SINGLE` definiert. Auf einigen Grafikkarten, beispielsweise *ATI FireGL* arbeitet das *single buffering* nicht korrekt; stellen Sie dort einfach *double buffering* ein durch Verwendung von `GLUT_DOUBLE`.

Jetzt können wir festlegen, wie groß das zu erzeugende Fenster werden soll, und wo seine Position ist:

```
void glutInitWindowSize(int width, int height);  
void glutInitWindowPosition(int x, int y);
```

Die Argumente `int x`, `int y` legen die *linke, obere* Ecke des Fensters fest, relativ zum Display. `width` und `height` legen die Größe in Pixeln fest. Window-Koordinaten sind meist Koordinatensystem, deren *x*-Achse von links nach rechts und deren *y*-Achse von oben nach unten verlaufen.

Ein Fenster wird schließlich mit

```
int glutCreateWindow(char *name);
```

erzeugt. Dabei gelten die vorher festgelegten Eigenschaften. Die übergebene Zeichenkette wird als Fenstername verwendet. Allerdings wird das Fenster noch nicht dargestellt, dies geschieht erst, wenn mit Hilfe der Funktion `glutMainLoop()` gang am Ende des Quelltexts die Event-Loop geöffnet wird. Der zurückgegebene Integer-Wert ist ein eindeutiger Schlüssel für dieses Fenster und kann im Programm beispielsweise dazu verwendet werden, mehrere Fenster in einer Anwendung zu öffnen.

### 3.1 Callback-Funktionen für die grafische Ausgabe

*GLUT* bindet Ihr *OpenGL*-Programm an ein Fenstersystem an. Der gemeinsame Nenner aller Systeme ist, dass ein Event-Handling durchgeführt wird. Das bedeutet, in einer Endlosschleife wartet ein Event-Handler auf ein Ereignis, beispielsweise, ob Sie die Maus bewegen, eine Maus-Taste oder die Tastatur auslösen. Ist für ein aufgetretenes Event ein *Callback* registriert, dann wird dieser aufgerufen. Dieses Vorgehen entspricht dem Konzept in *X11* und dem *Xt-Toolkit*.

Die folgenden Callbacks müssen natürlich nicht alle vorhanden sein; wenn Sie keine Tastatur-Kommandos vorsehen, können Sie die entsprechende Registrierung einfach weglassen. Den folgenden Callback aber werden Sie immer aufnehmen, nämlich die Registrierung der Funktion, die für die grafische Ausgabe Ihres Programms zuständig ist:

```
void glutDisplayFunc(void (*func)(void));
```

Die übergebene Funktion wird immer dann aufgerufen, wenn der Fensterinhalt neu gezeichnet werden muss. Dies ist der Fall beim ersten Öffnen des Fensters, oder wenn eine Verdeckung durch ein anderes Fenster wegfällt. Sie können auch durch die Funktion

```
void glutPostRedisplay(void);
```

eine Neuaufbau des Fensterinhalts durch die mit `glutDisplayFunc` registrierte Funktion jederzeit erzwingen. Achten Sie darauf, dass die Aufrufliste für die Callback-Funktion zwingend festgelegt sind; Sie können keine Funktion an `glutDisplayFunc` übergeben, die beispielsweise einen anderen Rückgabotyp oder eine nicht-leere Parameterliste besitzt. Das schränkt natürlich die Kommunikation zwischen der grafischen Ausgabe und Ihrer Anwendung ein – das ist der Preis dafür, dass Sie die Programmierung in *X11* oder *Microsoft Windows* nicht selbst durchführen müssen. Programme mit *GLUT* sind *quell-kompatibel*; Sie können eine Quelle, die Sie auf *Microsoft Windows* übersetzt haben durch übersetzen und binden auf *Linux* portieren.

Wird das von *GLUT* erzeugte Fenster verschoben oder in seiner Größe verändert, dann sollten Sie eine Funktion registrieren, die für diesen Fall beispielsweise mit `glViewport()` das Verhältnis zwischen Höhe und Breite des Fensters berücksichtigt:

```
void glutReshapeFunc(void (*func)(int width, int height));
```

Die Argumente der übergebenen Funktion werden beim Aufruf mit der neuen Breite und Höhe des Fensters in Pixeln belegt. Neben dem Viewport sollten Sie hier auch die Sicht entsprechend anpassen. Wird keine Funktion registriert verwendet *GLUT* als Default eine Funktion, die einfach `glViewport(0, 0, width, height)` aufruft.

### 3.2 Callback-Funktionen für Peripherie-Geräte

*GLUT* kennt natürlich die Tastatur oder eine Maus. Die Maus wird dabei immer als eine Dreitasten-Maus interpretiert. Neben diesen beiden Peripherie-Geräten gibt es inzwischen auch eine ganze Menge anderer Geräte die unterstützt werden, beispielsweise die *Space-Mouse*. Weitere Einzelheiten können Sie der online verfügbaren *GLUT*-Dokumentation entnehmen.

Eine Funktion, die Tastatur-Eingaben verarbeitet wird mit

```
void glutKeyboardFunc(void (*func)(unsigned int key, int x, int y));
```

registriert. Der `key`-Parameter enthält das eingegebene *ASCII*-Zeichen. Die Parameter `x, y` enthalten die Position des Maus-Zeigers (in Fensterkoordinaten) zum Zeitpunkt der Tastatureingabe. Für Tastatur-Eingaben wie die Cursor-Tasten oder die Funktionstasten gibt es daneben die Funktion

```
void glutSpecialFunc(void (*func)(unsigned int key, int x, int y));
```

Eine minimale Callback-Funktion für die Tastatur könnte darin bestehen, mit `q, Q` oder mit `ESC` das Programm zu beenden:

```
/* Der Tastatur-Callback in Minimalausstattung */
void keyboard(unsigned char key, int x, int y)
{
    switch (key) {
        case 27:
        case 'q':
        case 'Q':
            exit(0);
            break;
    }
}
```

Im folgenden Quelltext sehen sie, wie eine `special`-Funktion aussehen könnte. Für die F1-Taste wird auf der Konsole eine Information über das Programm ausgegeben; mit F12 wird zwischen der Ausgabe ursprünglich geöffneten Fenster und einer Full-Screen Ausgabe umgeschaltet. alle weiteren Tasten sind in diesem Beispiel nicht belegt.

```
void special(int key, int x, int y)
{
    switch (key) {
        case GLUT_KEY_F1:
            about();
            break;
        case GLUT_KEY_F12:
            FULL = !FULL;
            if (FULL)
                glutFullScreen();
            else
                glutReshapeWindow(WSizeX, WSizeY);
                glutPostRedisplay();
            break;

        /* Noch frei! */
        case GLUT_KEY_LEFT:
        case GLUT_KEY_RIGHT:
        case GLUT_KEY_UP:
        case GLUT_KEY_DOWN:
        case GLUT_KEY_PAGE_UP:
        case GLUT_KEY_PAGE_DOWN:
        case GLUT_KEY_END:
        case GLUT_KEY_HOME:
        case GLUT_KEY_F2:
        case GLUT_KEY_F3:
        case GLUT_KEY_F4:
        case GLUT_KEY_F5:
        case GLUT_KEY_F6:
        case GLUT_KEY_F7:
        case GLUT_KEY_F8:
        case GLUT_KEY_F9:
        case GLUT_KEY_F10:
        case GLUT_KEY_F11:
        case GLUT_KEY_INSERT:
        default:
            break;
    }
}
```

*GLUT* bietet die Möglichkeit, den drei Maustasten Menus beizugeben. Dazu gibt es die Funktionen

```
// Menu erzeugen und Callback registrieren
```

```
int glutCreateMenu(void (*func)(int));
// Menu-Eintrag hinzufügen:
void glutAddMenuEntry(char *text,int case);
// Maus-Taste für das Menu festlegen:
glutAttachMenu(int button);
```

Mit `glutCreateMenu` wird ein neues Menu erzeugt und eine Funktion registriert, die die Auswahl behandelt. Mit `glutAddMenuEntry` werden dann die Menu-Einträge definiert; der zweite übergebene Parameter legt den Integer-Wert fest, der bei der Auswahl übergeben wird.

Eine Minimalausstattung für ein solches Menu könnte wieder darin bestehen, mit der rechten Maustaste das Programm beenden zu können:

```
/* Maus-Menu für die rechte Maustaste; Minimalausstattung */
void standard(int id)
{
    switch (id) {
        case 100:
            exit(0);
            break;
    }
}
```

Mit `glutAttachMenu` schließlich wird die Maus-Taste ausgewählt, mit der das Menu aktiviert wird.

```
/* Menu registrieren */
void mouseMenu(void)
{
    glutCreateMenu(standard);
    glutAddMenuEntry("Quit",100);
    glutAttachMenu(GLUT_RIGHT_BUTTON);
}
```

Neben Menüs wollen Sie sicher auch Events für die Maustasten definieren. Dazu gibt es die Funktion

```
void glutMouseFunc(void (*func)(int button, int state, int x, int y));
```

Die damit registrierte Funktion wird aufgerufen, wenn eine Maustaste betätigt oder losgelassen wird. Der `button`-Parameter hat entweder `GLUT_LEFT_BUTTON`, `GLUT_MIDDLE_BUTTON` oder `GLUT_RIGHT_BUTTON` als Wert. Der `state`-Parameter hat entweder `GLUT_UP` oder `GLUT_DOWN` als Wert. Die beiden `x`, `y`-Parameter enthalten die Mausposition in *Fensterkoordinaten* zum Zeitpunkt des Events.

Mit der Funktion

```
void glutMotionFunc(void (*func)(int x, int y));
```

können Sie einen Callback registrieren, der aufgerufen wird, wenn Sie die Maus im `GLUT`-Fenster bewegen und gleichzeitig eine oder mehrere der Maustasten gedrückt halten. Die beiden `x`, `y`-Parameter enthalten die Mausposition in Fensterkoordinaten zum Zeitpunkt des Events.

### 3.3 Event-Loop

Sind alle Initialisierungen durchgeführt und alle Callbacks registriert, dann wird mit der Funktion

```
void glutMainLoop(void);
```

die Event-Loop von `GLUT` gestartet. Die registrierten Callbacks werden getriggert, wenn ein Event auftritt. Mit der Funktion

```
void glutIdleFunc(void (*func)(void));
```

können Sie eine Funktion registrieren, die ausgeführt wird, falls gerade keine Events vorliegen. Wird `NULL` übergeben, dann wird keine Funktion aufgerufen – es passiert nichts. Dies stellt natürlich auch den Default-Zustand dar. Damit können Sie einen „Bildschirm-Schoner“ implementieren.

## Kapitel 4

# Eine *OpenGL*-Schablone

In den Programmen, die Sie als Lösung für die Fallstudien in [BB05] oder im Praktikum ([BJL05]) im Visualisierungslabor der Fachhochschule Kaiserslautern erhalten wird immer ein Template verwendet. Dies soll es Ihnen erleichtern, sich in den Quellen zu orientieren; gleichzeitig sind *alle* Prototypen der Callbacks in `glutcallbacks.h` bereits vereinbart, so dass sich langwieriges Nachschlagen in der *GLUT*-Referenz erübrigen. Das soll Sie nicht davon abhalten, mit der Zeit ein eigenes Template zu erarbeiten. Aber die Erfahrung in den Praktikas seit 1998 hat gezeigt, dass die Schablonen eine große Hilfe darstellen.

### 4.1 Hauptprogramm

Für die *GLUT*-Callbacks werden in der Header-Datei `glutcallbacks.h` alle relevanten Prototypen vereinbart. In dieser Datei sind auch die *int*-Konstanten `WSizeX` und `WSizeY` definiert, mit der Sie die Fenstergröße festlegen. Auch die Fenster-Überschrift ist in dieser Datei als Konstante `windowTitle` definiert.

Darüberhinaus gibt es dort einen Prototyp für die Funktion `void callbacks(void)`, in der die Registrierung vorgenommen wird. Für Initialisierung und *OpenGL*-Funktionen, die vor der ersten grafischen Ausgabe ausgeführt werden müssen gibt es die Funktion `void init(void)`.

```
#include "glutcallbacks.h"

int main(int argc, char** argv)
{
    // Die Positionsparameter
    glutInit(&argc, argv);
    glutInitDisplayMode( GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(WSizeX, WSizeY);
    glutInitWindowPosition(100, 100);
    glutCreateWindow(windowTitle);

    /* Die Callbacks registrieren */
    callbacks();
    /* Die init-Funktion aufrufen */
    init();

    /* Event-Loop starten */
    glutMainLoop();
    return 0;
}
```

In Abbildung 4.1 sehen Sie das Ergebnis dieses Hauptprogramms in *Microsoft Windows XP*.

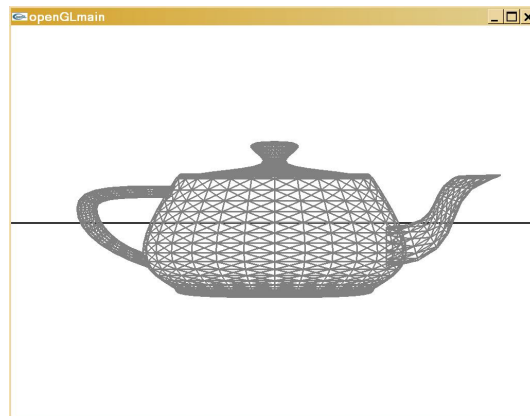


Abbildung 4.1: Das OpenGL-Fenster des Hauptprogramms in `openGLmain.cpp`

## 4.2 Die Implementierung der Callbacks

Die Implementierung der Callbacks und der Funktionen wie `init` oder `callbacks` finden Sie in der Datei `glutcallbacks.cpp`. Hier wird nur ein kleiner Teil dieser Datei beschrieben. Wie Sie beispielsweise mit Hilfe einer `glutMotionFunc` einen *Trackball*, also eine Steuerung der Sicht mit Hilfe der Maus, implementieren ist in einem eigenen Text ([Bri05] beschrieben).

Am Anfang der Datei finden Sie eine Versions-Angabe und eine Liste der in der Version implementierten Funktionen. Dabei wird zwischen den Callbacks für *GLUT* und einigen Hilfsfunktionen wird `init` oder `about` unterschieden.

```

/* -----
 * $RCSfile: glutcallbacks.cpp $
 * $Revision: 1.5 $
 * $Date: 2005/10/04 06:37:32 $
 * ----- */
/* -----
 * glutCallbacks:
 * init      - Initialisieren von OpenGL
 * display   - Zeichnen
 * idle      - Idle-Funktion
 * reshape   - Fensterreshape
 * keyboard  - Tastatur-Callbacks
 * special   - Funktionstastatur/Cursor-Tasten Callbacks
 * entryfunc - Maus ins Fenster/aus dem Fenster Callback
 * visible   - Sichtbarkeitsstatus des Fensters verändert Callback
 * mouse     - Callback für Maus-Tasten
 * mousemenu - Menu-Callback für die Maus-Tasten
 * mouseMotion - Callback bei Bewegung des Maus-Cursors
 * ----- */
/* -----
 * Hilfsfunktionen:
 * about     - Ausgabe von Text-Informationen über
 *           Tastaturbelegung und Programmbeschreibung
 * grid      - Ausgabe eines Gitters in der xz-Ebene
 * ----- */

```

Wie bereits bemerkt sind die Prototypen der *GLUT*-Callbacks zwingend vorgeschrieben, so dass eine Kommunikation zwischen der Anwendung und den Callbacks *nicht* über Parameterlisten erfolgen kann. Für dieses Problem gibt es mehrere Lösungen. Eine stellt die Verwendung von *globalen* Variablen dar – sicher nicht die schönste Möglichkeit. Allerdings werden Sie in Texten über

Realtime-Grafik-Software häufig globale Variable finden. Aus diesem Grund habe ich mich dafür entschieden, diese auch in der zur Zeit vorliegenden Implementierung von `glutcallbacks.cpp` zu verwenden. Mittelfristig ist geplant, die globalen Variablen durch das *Design Pattern Singleton* ([ERRJ94]) zu ersetzen.

Beachten Sie, dass diese Variablen *Datei*-global sind; außerhalb der Datei sind sie nicht verfügbar. Eine Ausnahme stellen nur die Konstanten für die Fenstergröße und den Fenstertitel dar, die in `glutcallbacks.h` definiert sind.

```
// Frame-Rate berechnen und ausgeben?  
bool REPORTSPEED = false;  
  
// Fullscreen oder nicht?  
bool FULL = false;  
  
// Globale Festlegung, ob ein Gitter in der xy-Ebene dargestellt werden soll  
bool GRID = true;
```

**Tip:**

Falls Sie eigene globale Variable definieren achten Sie unbedingt darauf, dass diese in *Großbuchstaben* benannt werden!

Die Datei `glutcallbacks.cpp` ist zweigeteilt. Im ersten Abschnitt, nach den globalen Variablen, finden Sie die Funktionen, die mit Sicherheit von Ihnen verändert werden müssen, beispielsweise `display`, `init`, `keyboard`.

Der `display`-Callback enthält die Ausgabe einer *Utah Teapots* und ein Gitter in der *xz*-Ebene zur besseren Orientierung. Es wird die Default-Sicht in *OpenGL* verwendet, entlang der negativen *z*-Achse. Die Funktion geht davon aus, dass Sie im Hauptprogramm *double buffering* aktiviert haben. Mit dem Aufruf `glutSwapBuffers()` wird der *back buffer*, in den Sie die Ausgaben machen mit dem momentan angezeigten *front buffer* ausgetauscht.

```
/* display-Callback für GLUT */  
void display(void)  
{  
    int start, end;  
    char titleBuffer[64] = " ";  
  
    if (REPORTSPEED) {  
        start = glutGet(GLUT_ELAPSED_TIME);  
    }  
  
    /* Buffer zurücksetzen */  
    glClear(GL_COLOR_BUFFER_BIT);  
    /* Ein Gitter ausgeben */  
    grid(10);  
    /* Den Utah Teapot ausgeben */  
    glColor3f(0.5, 0.5, 0.5);  
  
    /* Teapot nach "hinten" schieben, damit  
    * er sichtbar wird ... */  
    glMatrixMode(GL_MODELVIEW);  
    glLoadIdentity();  
    glTranslatef(0.0, 0.0, -0.5);  
  
    /* Den Utah Teapot ausgeben */  
    glutSolidTeapot(0.5);  
}
```

```

// Framerate ausgeben?
if (REPORTSPEED) {
    glFinish();
    end = glutGet(GLUT_ELAPSED_TIME);
// Frame-Rate ohne Nachkommastellen
sprintf(titleBuffer, "Framerate %1.i Frames/Sekunde (%2.i ms )",
        1000/(end-start), end-start);
glutSetWindowTitle(titleBuffer);

}

/* Die Buffer austauschen */
glutSwapBuffers();
}

```

Die `init`-Funktion legt die Hintergrundfarbe weiß und die Zeichenfarbe schwarz fest. Danach erfolgt noch die Festlegung der Strichdicke und Festlegung, dass der Teapot aus `display` als Drahtmodell ausgegeben wird.

```

void init(void)
{
    /* Weisser Hintergrund */
    glClearColor(1.0, 1.0, 1.0, 1.0);
    /* Zeichenfarbe */
    glColor3f(0.0, 0.0, 0.0);
    /* Liniendicke */
    glLineWidth(3.0);
    /* Drahtgeometrie ausgeben */
    glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
}

```

Mit der Hilfsfunktion `about` wird eine kleine Information über das Programm und seine Bedienung auf der Konsole ausgegeben. Diese Funktion wird mit Hilfe von F1 aktiviert. Sollten Sie eigene Short-Cuts definieren sollten Sie dieses Template entsprechend erweitern. In Abbildung 4.2 finden Sie ein Screen-Capture meiner *Cygwin*-Installation.

```

void about(void)
{
    cout << "-----" << endl;
    cout << " openGLTemplate " << endl;
    cout << " " << endl;
    cout << " Tastaturkommandos: " << endl;
    cout << " F1: Diesen Text ausgeben " << endl;
    cout << " F2: Frame-Rate ausgeben an/aus " << endl;
    cout << " F5: Gitter in der xy-Ebene an- und aus. " << endl;
    cout << " Default ist an. " << endl;
    cout << " F12: Fullscreen an/ausschalten " << endl;
    cout << endl;
    cout << " ESC/ " << endl;
    cout << " q/Q: Programm beenden " << endl;
    cout << "-----" << endl;
    cout << " (C) 2005 Manfred Brill " << endl;
    cout << "-----" << endl;
    cout << endl;
    cout << endl;
}

```

Tastatur-Kommandos, die immer enthalten sind finden Sie auf den Funktions- oder Cursortasten. Anwendungsspezifische Tastatur-Kommandos sollten Sie auf die ASCII-Zeichen legen. Da-

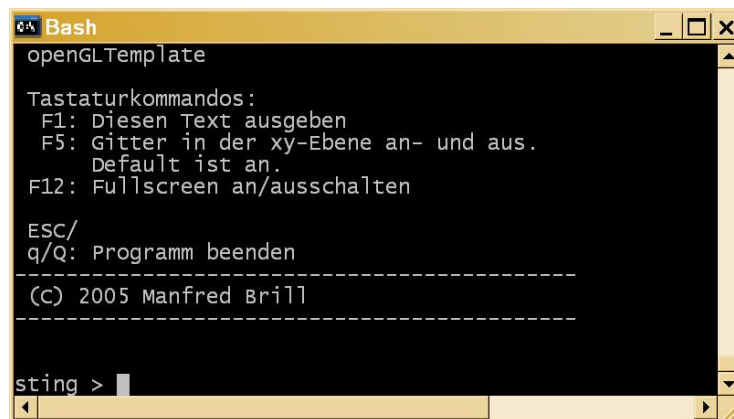


Abbildung 4.2: Ausgabe mit Hilfe der Funktion about

zu dient die Funktion `keyboard`, die im Ausgangszustand nur Short-Cuts für die Beendigung des Programms mit `q`, `Q` und `ESC` vorsieht:

```

/* String-Callback für GLUT
 *
 * Implementiert: Beenden des Programms mit q, Q und ESC.
 */
void keyboard(unsigned char key, int x, int y)
{
    switch( key ) {
        case 27:
        case 'q':
        case 'Q': exit(0);
                break;
        default:
                break;
    }
}

```

Die `idle`, `enter_leave` und `visible`-Callbacks werden zwar registriert, enthalten allerdings *keine* Anweisungen:

Danach, in einem zweiten Teil finden Sie Funktion wie `special`, `grid` oder `reshape`. Diese werden beispielsweise im Praktikum selten verändert.

In der `special`-Funktion sind im Ausgangszustand die folgenden Short-Cuts definiert:

- Aufruf von `about` (F1);
- Ausgabe der Frame-Rate (F2);
- Darstellung eines Gitters (F5);
- Umschalten zwischen Full-Screen oder Fenster-Modus (F12);

Alle weiteren Tasten sind noch frei.

```

void
special(int key, int x, int y)
{
    switch (key) {
        case GLUT_KEY_F1:
            about();
    }
}

```

```

        break;
    case GLUT_KEY_F2:
        cout << "Umschalten der Anzeige der Frame-Rate!" << endl;
        REPORTSPEED = !REPORTSPEED;
        if (REPORTSPEED)
            glutSetWindowTitle("Anzeige der Frame-Rate");
        else
            glutSetWindowTitle(windowTitle);
        break;
    case GLUT_KEY_F12:
        FULL = !FULL;
        if (FULL)
            glutFullScreen();
        else
            glutReshapeWindow(WSizeX, WSizeY);
            glutPostRedisplay();
        break;
    case GLUT_KEY_F5:
        GRID = !GRID;
        glutPostRedisplay();
        break;
    default:
        break;
}
}

```

Die `display`-Funktion ruft eine Funktion `grid` auf, die zur besseren Orientierung ein Liniengitter in der  $xz$ -Ebene darstellt. Die Ausgabe dieses Gitters kann mit `F5` aus- und eingeschaltet werden; dazu dient die globale Variable `GRID`. Mit Hilfe des Parameters `nL` können Sie steuern, wie viele Linien in jeder Richtung ausgegeben werden. Die Farbe der Linien ist zur Zeit *schwarz*; die Strichdicke ist in `init` zur Zeit mit `glLineWidth(3.0)` festgelegt. Mit `F2` schalten Sie die Anzeige der Framerate an und aus. In Abbildung 4.3 sehen Sie beide Zustände des Fensters.

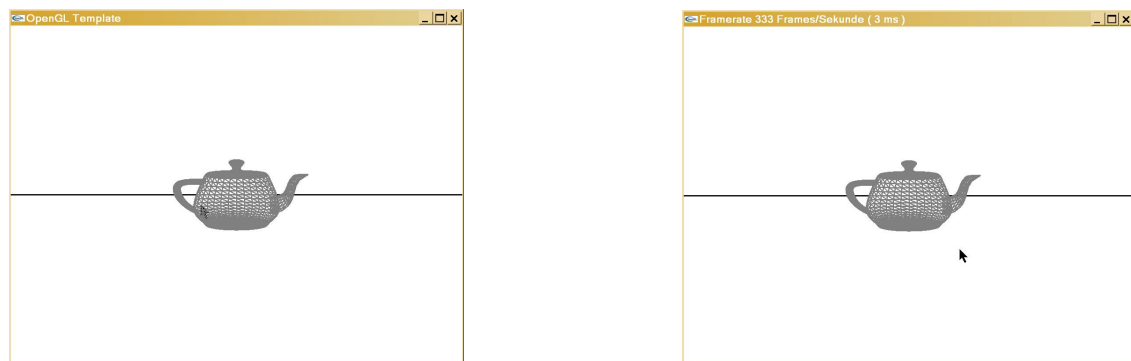


Abbildung 4.3: Das Fenster ohne (links) und mit (rechts) Anzeige der Framerate

```

void grid(int nL)
{
    int i;
    if (GRID) {
        glBegin(GL_LINES);
        glColor3f(0.0, 0.0, 0.0);
        for (i = -nL; i <= nL; i++) {
            glVertex3f((float)i, 0, (float)nL);
            glVertex3f((float)i, 0, (float)(-nL));
        }
    }
}

```

```

    glEnd();

    glBegin(GL_LINES);
        glColor3f(0.0, 0.0, 0.0);
    for (i = -nL; i<= nL; i++) {
        glVertex3f((float)nL,0,(float)i);
        glVertex3f((float)(-nL),0,(float)i);
    }
    glEnd();
}
}

```

Die Funktion `reshape` wird aufgerufen, falls die Fenstergröße verändert wird. Wichtig für diese Funktion ist die korrekte Sicht-Definition und die Angabe des Viewports. Da in diesem Template keine eigene Sichtdefinition erfolgt, ist hier nur der Viewport definiert; dabei wird der Default in *OpenGL* eingestellt:

```

/* reshape-Callback für GLUT          */
void reshape(int w, int h)
{
    // Viewport (Default-Zustand)
    glViewport(0,0, (GLsizei) w, (GLsizei) h);
}

```

Letztendlich wird die Funktion `callbacks` implementiert, in der alle implementierten Callbacks registriert werden:

```

void callbacks(void)
{
    /* display registrieren          */
    glutDisplayFunc(display);
    /* Idle-Funktion registrieren    */
    glutIdleFunc(idle);
    /* keyboard registrieren         */
    glutKeyboardFunc(keyboard);
    /* special registrieren          */
    glutSpecialFunc(special);
    /* Reshape registrieren          */
    glutReshapeFunc(reshape);
    /* Maus ins Fenster/verlässt Fenster */
    glutEntryFunc(enter_leave);
    /* Veränderung Fenstersichtbarkeit */
    glutVisibilityFunc(visible);

    return;
}

```

### 4.3 Ein Makefile für *Cygwin* und *Linux*

Verwenden Sie das Kommando `make` in *Cygwin* oder *Linux*, dann ist ein `Makefile` immer folgendermassen aufgebaut:

```

# -----
# Makefile für ein OpenGL/GLUT-Programm für Cygwin und Linux
# -----
CXX = g++
DEBUG    = -g
CXXFLAGS = -I. ${DEBUG} -Wno-deprecated

```

```
OGLLIBS = -lglut32 -lglu32 -lopengl32

all : openGLmain

openGLmain.o : openGLmain.cpp
    ${CXX} -c ${CXXFLAGS} $<

openGLmain : openGLmain.o glutcallbacks.o
    ${CXX} -o $@ $< glutcallbacks.o ${OGLLIBS} -lm

glutcallbacks.o : glutcallbacks.cpp glutcallbacks.h
    ${CXX} -c ${CXXFLAGS} $<

clean:
    /bin/rm -f *.o *.exe
```

Dieser Makefile kann in *Eclipse* in einem *Standard Make Project* des *CDT*-Plugins verwendet werden. Mehr Informationen zum Aufbau von Makefiles und zum *make*-Kommando finden Sie in [Bri03] und [BJL05].

# Literaturverzeichnis

- [Ang05] ANGEL, EDWARD: *Interactive Computer Graphics – A Top-Down Approach with OpenGL*. Addison-Wesley, 2005.
- [BB05] BENDER, MICHAEL und BRILL, MANFRED: *Computergrafik – ein anwendungsbezogenes Lehrbuch*. 2. Auflage, Hanser, 2005.
- [BJL05] BRILL, MANFRED, JUNG, MATHIAS und LOZANO, THOMAS: *Praktikumsunterlagen Computergrafik*. VisLab, 2005.
- [Bri03] BRILL, MANFRED: *Programmieren mit Cygwin und UNIX*. Visualisierungslabor der FH Kaiserslautern, 2003.
- [Bri05] BRILL, MANFRED: *Trackball und Pilot's View mit OpenGL und GLUT*. Visualisierungslabor der FH Kaiserslautern, 2005.
- [ERRJ94] ERICH GAMMA, RICHARD HELM, RALPH JOHNSON und JOHN VLISSIDES: *Design Patterns*. Addison-Wesley, 1994.
- [Fos96] FOSNER, RON: *OpenGL Programming for Windows 95 and Windows NT*. Addison-Wesley, 1996.
- [Hil01] HILL, FRANCIS: *Computer Graphics using OpenGL*. Prentice Hall, 2001.
- [KF00] KEMPF, RENATE und FRAZIER, CHRIS: *OpenGL Reference Manual (2nd Ed.) – The Official Reference Document to OpenGL*. Addison-Wesley, 2000.
- [Kil96a] KILGARD, MARK: *OpenGL Programming for the X Windows Systems*. Addison-Wesley, 1996.
- [Kil96b] KILGARD, MARK: *The OpenGL Utility Toolkit (GLUT) Programming Interface, API Version 3*. Silicon Graphics, 1996.
- [MB05] MCREYNOLDS, TOM und BLYTHE, DAVID: *Advanced Graphics Programming Using OpenGL*. Morgan Kaufmann, 2005.
- [Ros04] ROST, RANDI: *OpenGL Shading Language*. Addison-Wesley, 2004.
- [WND00] WOO, MASON, NEIDER, JACKIE und DAVIS, TOM: *OpenGL Programming Guide (2nd Ed.) – The Official Guide to Learning OpenGL*. Addison-Wesley, 2000.
- [WS04] WRIGHT, RICHARD und SWEET, MICHAEL: *OpenGL Superbible*. Sams, 2004.