

Übungen zu Grafikprogrammierung in C++

Was man wissen muss:

- Konzepte der Objektorientierung
- Objektorientierung in C++

Aufgabe 15 (H) OpenGL Benutzeroberfläche

Lesen Sie sich im Red Book (Link auf der Homepage des Grafikprogrammierpraktikums) das Kapitel 1 durch.

In dieser Aufgabe soll der Rendering Kontext für eine beliebige OpenGL Anwendung erzeugt werden. Der erste Schritt ist das Erstellen eines Visual Studio Projektes.

- Neue VS Solution erzeugen (Win32 Konsolen Projekt).
- Leeres Projekt erstellen. (->Anwendungseinstellungen->Leeres Projekt)
- Dem Projekt eine `main.cpp` hinzufügen.

Nachdem das Projekt erzeugt wurde, soll nun die Benutzeroberfläche mit Hilfe von GLUT (OpenGL Utility Toolkit) erstellt werden. Eine einfache und plattformunabhängige Implementierung davon ist GLUT (zu finden im Verzeichnis `aufgaben/GLUT`).

Das Programm: Als erstes müssen die entsprechenden Headerdateien eingefügt werden.

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

In der main Funktion wird die Benutzeroberfläche initialisiert und gestartet.

```
int main (int argc, char** argv) {  
    glutInit(&argc, argv);  
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);  
    glutInitWindowPosition(0, 0);  
    glutInitWindowSize(640, 480);  
    glutCreateWindow("OpenGL-Praktikum");  
    // reserviert fuer die callbacks ... coming soon  
    glutMainLoop();  
    return 0;  
}
```

Die Verzeichnisse für die Headerdateien (includes) und Bibliotheken (libraries) müssen noch richtig gesetzt werden. Die aktuelle Version von GLUT befindet sich in dem Verzeichnis `aufgaben/GLUT`. Um diese in der lokalen Arbeitskopie verfügbar zu haben, muss ein update auf den SVN Aufgabenpfad gemacht werden. **Bitte Dateien nicht kopieren und alle Pfade relativ (nicht absolut) auf die entsprechenden Verzeichnisse setzen.**

Man kann jetzt eine Oberfläche sehen, jedoch wird noch nichts gezeichnet.

Durch das Registrieren von sogenannten "Callback"-Funktionen kann man erreichen, dass nach dem erstmaligen Aufruf von `glutMainLoop()` bestimmte Funktionen bei bestimmten Aktionen aufgerufen werden. Im Folgenden werden die wichtigsten Callback-Funktionen erklärt und sollen implementiert werden. In allen Callback-Funktionen soll auf der Standardausgabe eine Zeile, die den Aufruf dieser Funktion signalisiert, ausgegeben werden.

glutDisplayFunc(display) Die Funktion `display` wird jedesmal aufgerufen, wenn sich das Fenster verändert (verschoben oder die Größe manipuliert wird). In der Funktion sollen folgende Zeilen implementiert werden:

```
glClearColor(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); // löscht bestimmte buffer
glLoadIdentity(); // lädt die Einheitsmatrix
glutSwapBuffers(); // tauscht den buffer aus, falls double buffer verwendet wird
```

glutReshapeFunc(reshape) Die Funktion `reshape(int width, int height)` wird aufgerufen, wenn sich die Größe ändert. In der Funktion sollen folgende Zeilen implementiert werden.

```
glViewport(0, 0, width, height); // setzt das Ausgabefenster
```

glutKeyboardFunc(keyboard) Die Funktion `keyboard(unsigned char key, int x, int y)` wird bei jeder Tastatureingabe aufgerufen. Hier soll die gedrückte Taste und die aktuelle Position der Maus im Ausgabefenster auf der Standardausgabe ausgegeben werden.

glutMouseFunc(mouse) Die Funktion `mouse(int button, int state, int x, int y)` wird bei jedem Drücken eines Mausbuttons ausgeführt. Es soll auf der Standardausgabe ausgegeben werden, welcher Button gedrückt bzw. losgelassen wurde, sowie die aktuelle Position der Maus im Ausgabefenster.

glutMotionFunc(mouseMotion) Die Funktion `mouseMotion(int x, int y)` wird jedesmal ausgeführt, wenn sich die Maus bei gedrücktem Button bewegt. Es soll auf der Standardausgabe die aktuelle Änderung der Mausposition in `x` und `y` Richtung ausgegeben werden.

Eine komplette Referenz mit allen Callback-Funktionen kann unter <http://www.opengl.org/resources/libraries/glut/spec3/spec3.html> gefunden werden.

Optional können alle callbacks implementiert werden und ihr Verhalten getestet werden.

Aufgabe 16 (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 entweder die von Ihnen bereits programmierte Transform-Klasse verwenden oder die von OpenGL angebotenen Funktionen `glRotatef`,

`glTranslatef` und `glScalef`. Auf der Basis der vorhergehenden Aufgabe und der Aufgabe zur Objektorientierung sollen nun erste zweidimensionale Objekte erzeugt werden.

- a) Erweitern Sie nun die Render-Methoden der beiden Klassen `Triangle` und `Rectangle` so, dass ein Dreieck und ein andersfarbiges Rechteck gezeichnet werden. Diese sollen in der OpenGL Benutzeroberfläche der letzten Aufgabe aufgerufen werden.
- b) Fügen Sie die Methode `rotate(int reverse)` der Datei hinzu, welche das in dieser Datei definierte `Triangle` im Uhrzeigersinn und das `Rectangle` 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 die Methode `scale(int reverse)` der Datei hinzu, welche die in dieser Datei definierten Polygone 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 die Methode `reset()` hinzu, die die Ausgangsposition sowie -größe des Original-Polygons 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) Alle Solutiondateien (*.sln) und Projektdateien (*.vcproj) müssen eingecheckt sein
- b) Alle Pfade (includes und libs) müssen **relativ** gesetzt sein
- c) 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.