

Ausarbeitung für das GDV-Praktikum (OpenGL-Teil) SoSe 2017**1. Teilnehmer/in**


Name:
Vorname:
Matr. Nr.:

Termin:

(z.B. Mo2x)

2. Teilnehmer/in

Name:
Vorname:
Matr. Nr.:

Ausarbeitung:: spätestens eine Woche nach Ihrem dritten OpenGL-Termin!!!: Ausarbeitung **nicht** per Email **!!!**: Ausarbeitung ins Fach Ihrer Dozentin bzw. Ihres Dozenten im Sekretariat
(1 Ausarbeitung pro Gruppe)

Checkliste: (Bitte haken Sie ab)**1:** Diese Ausarbeitung enthält in **Papierform**:

- ☐ als 1. Seite dieses Blatt, **nicht** jedoch den restlichen Text der Aufgabenstellung;
- ☐ die Beantwortung der gestellten Fragen (Einführungsteil zu OpenGL);
(Unvollständige, grob falsche oder von anderen Gruppen abgeschriebene Antworten können zu Punktabzug bei den Zusatzpunkten oder sogar zum Nicht-Bestehen des Praktikums führen!)

2: Diese Ausarbeitung enthält in **Papierform**:

- ☐ die Kurzbeschreibung inkl. (Hand-)Skizzen der eigenen Lösung; (**Weg von der Idee bis zur Lösung**);
- ☐ den beschrifteten Szenengraph der eigenen Lösung.

3: Diese Ausarbeitung enthält eine (mit Namen, Gruppennummer und Jahr) **beschriftete CD** mit:

- ☐ dem gut kommentierten Quellcode (*.cpp, *.h, ...) der eigenen Lösung(en); sowie allen weiteren zugehörigen Dateien: z.B. Solution; Texturen, ...
- ☐ falls Windows: dem ausführbaren Programm (*.exe) der eigenen Lösung, sowie den verwendeten DLLs (z.B. freeglut*.dll), damit Demos überall direkt von dieser CD möglich sind.

4: Bitte stecken Sie die Ausdrucke (1 und 2) und die CD in eine Klarsichthülle.**Danke!**

Vorbemerkungen:

Die folgende Anleitung ist auf Windows als Betriebssystem und MS-Studio als Entwicklungs-Umgebung zugeschnitten. Natürlich ist es Ihnen freigestellt, auch andere Betriebssysteme oder Entwicklungs-Umgebungen einzusetzen; (bei einem anderen OS kann es sein, dass Sie statt der FreeGLUT (siehe weiter unten) nur die GLUT einsetzen können: <http://www.opengl.org/resources/libraries/glut/>)

Zur Bearbeitung des geführten Teils sollten Sie nur einen Praktikumsblock benötigen! Die beiden weiteren Blöcke stehen Ihnen dann für die „eigene Lösung“ zur Verfügung.

Die OpenGL Einführung unterteilt sich in vier Lerneinheiten:

1. OpenGL-Grundfunktionen;
2. OpenGL-Kamera;
3. OpenGL-Transformationen - anhand eines Szenengraphs;
4. OpenGL-Animationen.

1. Erstellen einer Konsol-Applikation

1. (a) Vorarbeiten, falls Sie daheim entwickeln wollen:

Holen Sie sich die **FreeGLUT** (zum "Selber-Erstellen" oder als "Prepackaged Release" von

<http://freeglut.sourceforge.net/index.php#download/>

und die OpenGL-Ergänzungen (OpenGL_Ergaenzungen.zip) von der Homepage Groch: Graph. DV > Praktikum.

Wenn Sie MS Visual Studio einsetzen, so kopieren Sie den FreeGLUT-Ordner GL mit allen h-Dateien in den include-Ordner Ihrer Studio-Installation:

...\\Microsoft Visual Studio ...\\VC\\include.

Die Datei freeglut.lib kopieren Sie nach

...\\Microsoft Visual Studio ...\\VC\\lib

und die freeglut.dll nach

...\\Microsoft Visual Studio ...\\VC\\bin

oder in das Verzeichnis, in dem Ihre exe-Dateien abgelegt werden oder (zentral) z.B. nach C:\\Windows\\system32 oder nach C:\\Windows\\SysWOW64.

(b) Wenn Sie auf den Labor-Rechnern arbeiten, so müssen Sie zur Verwendung der FreeGLUT in den Projekt-Eigenschaften folgende Verzeichnisse hinzufügen:

- 1) C/C++/Additional Include Directories: C:\\SDK\\freeglut\\include
- 2) Linker/Additional Library Directory: C:\\SDK\\freeglut\\lib

(c) Das Archiv **OpenGL_Ergaenzungen.zip** enthält einfache Demo-Programme, zu Texturen, Licht, Uhrzeit, Funktionstasten, Maus-Tasten und Menus.

Tipps: 1.) Im Internet oder in der Bibliothek gibt es zahlreiche Infos und Tutorials bzw. Bücher zu OpenGL.

2.) OpenGL-Referenz (ohne GLUT): <http://www.opengl.org/sdk/docs/man2/>

3.) GLUT-Ref.: <https://www.opengl.org/resources/libraries/glut/spec3/spec3.html>

2. Starten Sie Microsoft Visual Studio.
3. Legen Sie ein neues Projekt an: **Datei>Neu>Projekt>Visual C++: Win32-Konsolenprojekt > Projektname** eingeben.
4. Löschen Sie im Projektmappen-Explorer die mit dem von Ihnen eingegebenen Projektnamen erzeugte cpp-Datei und fügen Sie statt dessen die vorgegebene Datei [teil_1.cpp](#) ein:

Im Projektmappen-Explorer rechter Maus-Click auf „Sources“: **Hinzufügen>Vorhandenes Element hinzufügen ...**

5. Verhindern Sie, dass vorkompilierte Header verwendet werden: Im Projektmappen-Explorer ganz oben den von Ihnen eingegebenen Projektnamen markieren, dann nach Rechts-Click: **Eigenschaften>C/C++:Vorkompilierte Header>Erstellen/Verwenden> Vorkompilierte Header nicht verwenden.**
6. Sehen Sie sich den Quellcode von `teil_1.cpp` an und fügen Sie in „`glutCreateWindow (\"Name_1; Name_2\")`“ ihre beiden Namen ein.
7. Erstellen Sie das Projekt und führen Sie das Programm aus.

Fragen:

1. Was sehen Sie nach der Ausführung des Programmes im Graphik-Fenster?
2. Sind die Anweisungen `glBegin` und `glEnd` unbedingt notwendig? Was passiert, wenn man sie weglässt?

Anmerkung: Für alle nachfolgenden Bearbeitungsschritte ist es wichtig zu wissen, dass die OpenGL Kamera im Ursprung des Koordinatensystems sitzt und entlang der negativen Z-Achse blickt.

OpenGL arbeitet mit einem rechtshändigen Koordinatensystem.

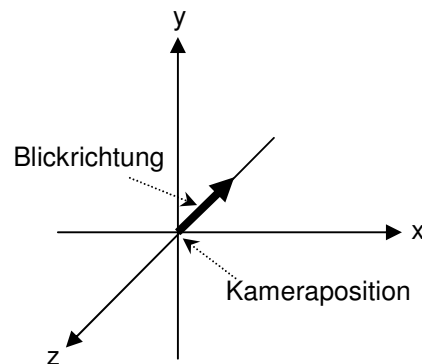


Abbildung 1: Rechtshändiges OpenGL-Koordinatensystem

Das Quadrat soll nun eingefärbt werden. Unten links soll das Quadrat rot und an allen anderen Eckpunkten blau eingefärbt werden. Farben können mit der nachfolgenden Anweisung gesetzt werden:

```
glColor4f( R, G, B, Alpha);
```

3. An welchen Positionen im Programm müssen diese `glColor4f`-Anweisungen stehen?

Der Hintergrund ist bisher noch nicht gelöscht worden. Holen Sie das nach!

```
glClear( GL_COLOR_BUFFER_BIT );
```

4. Welche Position im Code ist am besten zum Aufruf von `glClear` geeignet und warum?

Legen Sie nun einen orangefarbenen Hintergrund an. Hierfür können Sie den folgenden Befehl nutzen:

```
glClearColor( R, G, B, Alpha );
```

5. In welcher Reihenfolge müssen `glClear` und `glClearColor` aufgerufen werden?
6. Was passiert, wenn Sie in einer Animation `glClear(GL_COLOR_BUFFER_BIT);` weglassen? (Ihre Vermutung können Sie aber erst in Teil 4 überprüfen.)

Erzeugen Sie eine zweite Fläche mit den hier angegebenen Koordinaten:

```
glBegin( GL_POLYGON );
    glColor4f( 0., 1., 0., 1.);
    glVertex3f( -0.5, -0.5, -1. );
    glVertex3f( 0.5, -0.5, -1. );
    glVertex3f( 0.5, 0.5, -1. );
    glVertex3f( -0.5, 0.5, -1. );
glEnd();
```

Fügen Sie diese Fläche direkt unterhalb der rot/blau eingefärbten Fläche in den Code ein.

7. *Welche der beiden Flächen sehen Sie?*

8. *Erzeugen Sie die Flächen mal in einer anderen Reihenfolge: Was fällt Ihnen auf und warum ist das so?*

Im nächsten Schritt soll die Z-Buffer Funktionalität in Ihr Programm integriert werden. Dazu müssen Sie Ihr Programm wie folgt erweitern:

- In `main(...)` muss `glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB)` um `GLUT_DEPTH` erweitert werden:
`glutInitDisplayMode(GLUT_ DOUBLE | GLUT_RGB |GLUT_DEPTH);`
- In `RenderScene()` muss ganz am Anfang neben dem Color-Buffer auch der Z-Buffer initialisiert werden:
`glClearColor(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);`
- In `Init()`..müssen die beiden Zeilen eingefügt werden:
`glEnable(GL_DEPTH_TEST);`
`glClearDepth(1.0);`
- Damit der Z-Buffer wirklich arbeitet, muss in `Reshape()` u.a. noch das Frustum definiert werden. Eine Möglichkeit dazu ist:

```
// Matrix für Transformation: Frustum->viewport
glMatrixMode(GL_PROJECTION);
// Aktuelle Transformations-Matrix zuruecksetzen
glLoadIdentity ();
// Viewport definieren
glViewport(0,0,width,height);
// Frustum definieren (siehe unten)
glOrtho( -1., 1., -1., 1., 0.3, 1.3);
// Matrix für Modellierung/Viewing
glMatrixMode(GL_MODELVIEW);
```

Erläuterung:

```
glOrtho( GLdouble left, GLdouble right,
         GLdouble bottom, GLdouble top,
         GLdouble near, GLdouble far );
```

Die `near` und `far` Werte beschreiben den Abstand der Near- und Far-Clippingplane vom aktuellen Standpunkt der Kamera in Blickrichtung der Kamera. Werden für `near` und `far` negative Werte gesetzt, dann liegt das aufgespannte Frustum hinter der Kamera!

- Entspricht die Ansicht der Flächen nun Ihren Erwartungen?*
 - Beschreiben Sie kurz, wie der Z-Buffer funktioniert.*

2. OpenGL-Kamera

Erstellen Sie eine Kopie von **teil_1.cpp** unter dem Namen **teil_2.cpp**, schließen Sie die Datei **teil_1.cpp** vom Build aus (**teil_1.cpp** -> **Eigenschaften > Vom Build ausschließen: ja**) und fügen Sie die Dateien **teil_2.cpp**, **wuerfel.cpp** und **wuerfel.h** zu Ihrem Projekt hinzu. (**Projekt > Dem Projekt hinzufügen > Dateien**). Nehmen Sie alle weiteren Ergänzungen bitte in **teil_2.cpp** vor.

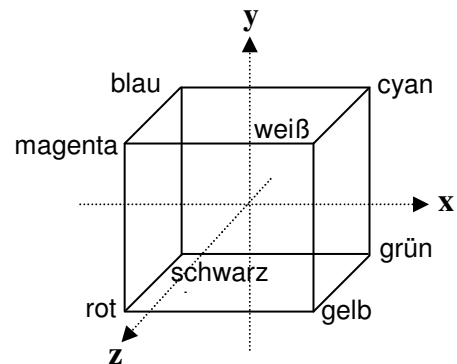


Abbildung 2: Der Aufbau des Würfels

Includieren Sie die Datei **wuerfel.h**:

```
#include "wuerfel.h"
```

Ersetzen Sie das Code-Stück, in dem die beiden Flächen erzeugt werden, durch:

```
Wuerfel(0.4);
```

Weiterhin

```
glOrtho( -1., 1., -1., 1., 0.3, 1.3);
```

ändern in:

```
glOrtho( -1., 1., -1., 1., 0.0, 1.0);
```

10. Welche Fläche sehen Sie und warum sehen Sie gerade diese Fläche?

Um den Würfel von vorne zu sehen, können Sie entweder den Würfel um Eins auf der negativen Z-Achse verschieben

```
...glTranslatef( 0., 0., -1.);
```

oder Sie verschieben in `RenderScene()` die Kamera durch den Befehl:

```
gluLookAt ( 0., 0., 1., 0., 0., 0., 0., 1., 0.);
```

Sie müssten in beiden Fällen denselben Bildausschnitt sehen.

Die Parameter der Funktion haben diese Bedeutung:

```
gluLookAt ( eyex, eyey, eyez,           // Kamera-Position
            centerx, centery, centerz,   // Betrachtete Position
            upx, upy, upz);              // Kamera-View-Up-Vektor
```

Folgende Skizze visualisiert die Ausrichtung der Kamera nach der Ausführung der Funktion:

```
gluLookAt ( 0., 0., 1., 0., 0., 0., 0., 1., 0.);
```

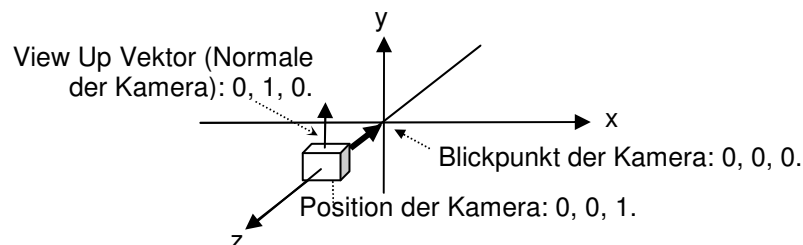


Abbildung 3: Ausrichtung der Kamera

11. Probieren Sie (ohne Translation des Würfels) die folgenden Kamerapositionen aus und dokumentieren Sie die dazu verwendeten `gluLookAt`-Aufrufe:

a) Betrachtung der Szene von vorne oben; (Kameraposition: (0., 1., 1.).

Sind die Parameterwerte in `glOrtho` richtig gesetzt? Falls **NEIN**, was stimmt nicht?

b) Betrachtung der Szene direkt von rechts; Kameraposition: (1., 0., 0.). Wie lauten die kompletten Aufrufe von `glOrtho` und `gluLookAt`?

c) Betrachtung der Szene von rechts oben: wie lautet die Kameraposition: (?, ?, ?).

Der Würfel sieht irgendwie „verzerrt“ aus! Beheben Sie dies, indem Sie anstelle des `glOrtho`-Befehls einfügen:

```
// gluPerspective(senkr. Oeffnungsw., Seitenverh., zNear, zFar);
gluPerspective(45., 1., 0.1, 2.0);
```

3. OpenGL-Transformationen - anhand eines Szenengraphs

Erstellen Sie eine Kopie von **teil_2.cpp** unter dem Namen **teil_3.cpp** (siehe oben) und arbeiten Sie in **teil_3.cpp** weiter. (Setzen Sie `gluLookAt(...)` wieder so, dass Sie von vorne auf die Szene schauen).

Ihre Aufgabe ist es nun, einen im 45° Winkel nach unten hängenden Roboterarm zu implementieren. Der Arm besteht aus einem Ober- und einem daran anschließenden Unterarm (siehe Abbildung 4 sowie letzter Schritt in Abbildung 6). Hierfür sollten Sie den bereits bekannten Würfel (siehe Funktion `wuerfel` mit Größe 0.4!) als Grundelement für den Ober- und Unterarm benutzen. Wie der Arm modelliert werden soll, erkennen Sie am besten in der Abbildung 6. Hier sind alle durchzuführenden Teilschritte skizziert. In der Abbildung 5 ist der zugehörige Szenengraph zu sehen.

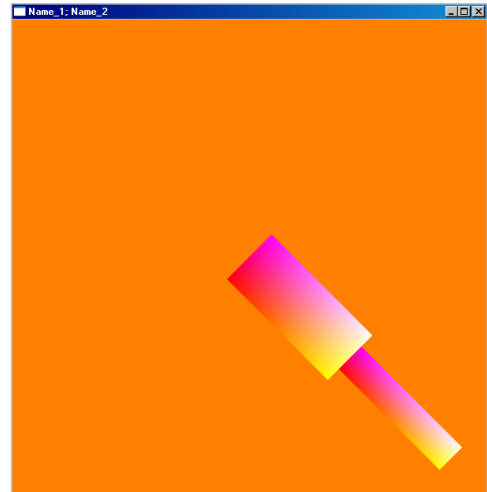


Abbildung 4: Roboterarm

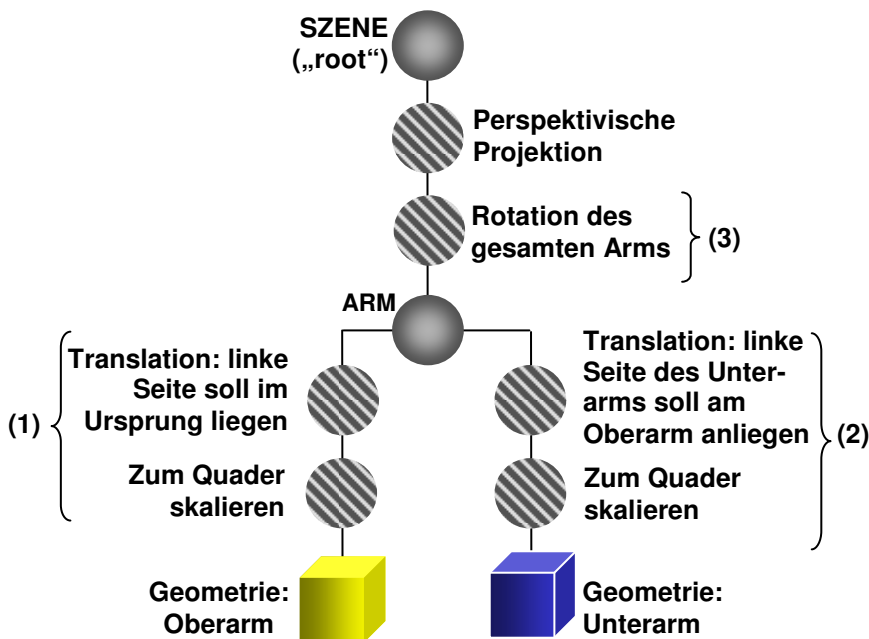


Abbildung 5: Szenengraph zum Armaufbau

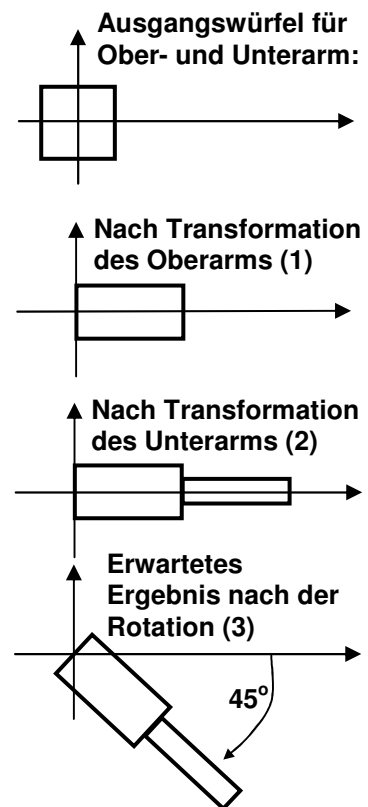


Abbildung 6: Durchzuführende Transformationen

Realisieren Sie den Szenengraph mit Hilfe von Push- und Pop-Befehlen.

12. Ist es möglich, den Rotate-Befehl vom „oberen“ Teil in die beiden Äste des Szenegraphs zu verlagern? (Wenn **JA**, wohin und wie? Wenn **NEIN**, warum nicht?)

Zur Implementierung benötigen Sie folgende OpenGL-Funktionen:

```
glRotatef(Winkel, fX, fY, fZ);  
glScalef(fX, fY, fZ);  
glTranslatef( fX, fY, fZ);  
glPushMatrix(); //Matrix wird auf den Stack gesichert  
glPopMatrix();  //Matrix wird vom Stack geholt und gesetzt
```

4. OpenGL-Animationen

Erstellen Sie eine Kopie von **teil_3.cpp** unter dem Namen **teil_4.cpp** (siehe oben) und arbeiten Sie in **teil_4.cpp** weiter.

Der Roboterarm soll nun so animiert werden, dass er um die z-Achse kreist.

Fügen Sie dazu ganz oben unterhalb von `include` folgende Zeile ein:

```
float fRotation = 315.0;    // globale Variable    :-(
```

Weiterhin müssen Sie in `Animate(...)` vor `glutPostRedisplay()` ergänzen:

```
    fRotation = fRotation - 1.0;    // Rotationswinkel aendern  
    if ( fRotation <= 0.0) {  
        fRotation = fRotation + 360.0;  
    }
```

Jetzt müssen Sie nur noch `fRotation` in `RenderScene()` zur Rotation des Arms verwenden.

Abschließend sollen Sie noch die Rotationsachse des Arms ändern: fügen Sie in `RenderScene()` oberhalb der Animations-Rotation (siehe oben) ein:

```
    glutWireCube(0.2);
```

und drehen Sie den gesamten Arm dann (statt um die z-Achse) um die rechte, obere (parallel zur z-Achse verlaufende) Kante dieses Würfels – diese dient also als „Schultergelenk“. Ändern Sie die Darstellung noch so ab, dass der Arm immer vollständig zu sehen ist – dazu sollen Sie keine Skalierungen verwenden, sondern eine andere Lösung finden!

Schauen Sie sich das Ganze auch mal von schräg, vorn, oben an.

13. Vergessen Sie bitte nicht die in Frage 6 angesprochene Nicht-Ausführung des `glClear`-Befehls für den Bild-Hintergrund. Der Tiefenpuffer muss trotzdem gelöscht werden:

```
glClear ( GL_DEPTH_BUFFER_BIT );
```

Aufgabenstellung für die „Eigene Lösung“ OpenGL-Teil

Sie haben die freie Wahl, was Sie unter dem Thema "Drohnen" bzw. sonstige „Propeller-Maschine“ darstellen wollen. Dabei sollen mehrere Propeller getriebene Geräte in Bewegung dargestellt werden, die sich synchron oder als „Solisten“ bewegen. Sie können ein Abbild real existierender Geräte implementieren oder Ihrer Phantasie freien Lauf lassen.

Ihre Lösung muss mindestens folgende Bedingungen erfüllen:

- 3D-Lösung;
- 3D-Hierarchie (zugehöriger mehrstufiger Szenegraph mit mehreren parallelen Ästen);
- Animation mit sich überlagernden Bewegungen für ein zusätzliches Objekt, das aus mindestens zwei Teilobjekten besteht: Teilobjekt 1 bewegt sich und nimmt dabei Teilobjekt 2 mit - dabei führt Teilobjekt 2 zusätzliche Bewegungen durch.

Beispiele:

Teilobjekt1: vorwärts fliegender Vogel; Teilobjekt 2/3: seine schlagenden Flügel
oder:

Teilobjekt1: vorwärts fliegender Flugzeug; Teilobjekt 2: sein rotierender Propeller.

Vorschläge für Erweiterungen:

- Licht, Texturen (z.B. Skybox), Interaktion, ... [siehe unten bei „Tipp“]
- Möglichkeit zur Umschaltung auf verschiedene Kamerastandorte oder eine animierte Ego-Perspektive. (Falls Sie Lichtquellen verwenden, so achten Sie darauf, dass diese stationär sind und sich nicht mit dem Betrachter mitbewegen!)

Die Lösung darf kein Planeten-System und keine Beinahe-Kopie aus dem Internet sein!

Sprechen Sie Ihre Lösungsidee beim zweiten Termin mit Ihren Betreuern durch; (häufig wird der Fehler gemacht, dass man sich zuviel vornimmt). Wichtiger als eine „Wow-Lösung“ ist, dass Sie Ihre eigene Lösung genau erklären können. Dies gilt für **beide** Teilnehmer jeder Gruppe!!!

Vorsicht: Auch wenn Ihre Nachbarn das u.U. tun und damit Eindruck machen: dies ist eine Einführung in OpenGL für alle! Wir unterstützen Sie nicht (oder höchstens nur minimal), wenn Sie mit Beleuchtung und Texturen oder anderen fortgeschrittenen Techniken arbeiten wollen; die Zeit würde bei der Betreuung der „Anfänger“ fehlen. Auch bei der Verwendung derartiger Techniken gilt: bitte nichts Unverstandenes in Ihr Programm integrieren. Weiterhin gilt auch, dass Ihr Ergebnis durchaus nicht besser wird, wenn Sie Modelle, die Sie mit fortgeschrittenen Tools erzeugen, importieren. Ein sauber implementiertes Modell (aus eigenen oder FreeGLUT-Objekten) ist wesentlich mehr wert, als z.B. ein importierter Laubfrosch!

Tipp: In der OpenGL-Ergänzung (OpenGL_Ergaenzungen.zip) im GDV-Praktikums-Download-Groch finden Sie zahlreiche Beispiele für die Verwendung von Licht, Texturen usw..

Schluss-Bemerkung: Die schönsten Lösungen (... das sind nicht unbedingt die umfangreichsten oder komplexesten) sollen für künftige Semester als Hörsaal-Demos genutzt werden können. Falls sie also z.B. Maus- oder Tastatur-Interaktionen einsetzen, so geben Sie bitte am Anfang Ihres Programms eine kurze Bedienungs-Anleitung im DOS-Fenster aus. Da die im Hörsaal eingesetzten Rechner u.U. nicht die allerschnellsten sind, sollte Ihr Programm zusätzlich die Möglichkeit bieten, die Animation z.B. durch eine interaktive Änderung der Animations-Parameter (z.B. Winkel-Inkrement) schneller bzw. langsamer laufen zu lassen.

Viel Spaß und viel Erfolg

B. Frömmer und E. Hergenröther