Ausarbeitung für das GDV-Praktikum (OpenGL-Teil) WiSe 2014/15

1. Teilnehmer/in

Name:

Vorname:

Matr. Nr.:

Termin: (z.B. Di2x)

Gruppe: (1 ... 8)

2. Teilnehmer/in

Name:

Vorname:

Matr. Nr.:

****

Ausarbeitung:

☺: spätestens eine Woche nach Ihrem dritten OpenGL-Termin!!!

☹: Ausarbeitung **nicht** per Email **!!!**

☺: Ausarbeitung ins Fach Ihres Praktikums-Betreuers im Sekretariat
(**1 Ausarbeitung pro Gruppe**)

****

Checkliste: (Bitte haken Sie ab)

1: Diese Ausarbeitung enthält in **gedruckter** Form:

- ☐ 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!)**
- ☐ geforderte Skizzen inkl. Beschriftungen (Einführungsteil zu OpenGL);
- ☐ Szenengraph (Einführungsteil zu OpenGL); kein Programmlisting.

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. Texturen, Solution; …
- ☐ 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! ☺

1. Inhalt

STRG+KLICKEN SIE (IN DER DOC-DATEI) AUF DIE SEITENNUMMER IM INHALTSVERZEICHNIS UM ZU DEM ENTSPRECHENDEN ABSCHNITT ZU GELANGEN. ENTSPRECHEND FÜR DIE FOLGENDEN FRAGEN UND BEGRIFFE.

LINKS ZU:

FRAGEN ZUR AUSARBEITUNG:

Frage_1
Frage_2
Frage_3
Frage_4
Frage_5
Frage_6
Frage_7

BEGRIFFE:

vertices
Tiefenpuffer
modelview_matrix
projection_matrix
Frustum
Viewport
Graphikfenster
Einheitsmatrix
ModelView_Modus
Projection_Modus
Reihenfolge_von_Transformationen
Animation

2. Erstellen einer Konsol-Applikation

Vorbemerkung: 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/)

1. 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.

   Das Archiv **OpenGL_Ergaenzungen.zip** enthält einfache Demo-Programme, die zeigen, wie man Texturen, Licht, Uhrzeit, Funktionstasten, Maus-Tasten und Menus verwendet.

   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. Arbeiten Sie im Labor bitte auf Laufwerk d: (und nach jeder Sitzung bitte alle Dateien löschen), aber nicht auf Laufwerk c:.
3. Starten Sie Microsoft Visual Studio.
4. Legen Sie ein neues Projekt an: Datei>Neu>Projekt>Visual C++: Win32-Konsolenprojekt > Projektname eingeben.
5. Löschen Sie im Projektmappen-Explorer die mit dem von Ihnen eingegebenen Projektnamen erzeugte cpp-Datei und fügen Sie statt dessen die vorgegebene Datei Nix.cpp ein:

   Im Projektmappen-Explorer rechter Maus-Click auf „Sources“: Hinzufügen>Vorhandenes Element hinzufügen …
6. 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.
7. Sehen Sie sich den Quellcode von Nix.cpp an.
8. Fügen Sie in „glutCreateWindow (" *** Nix zu sehen");“ für „***“ ihre Gruppennummer ein. Führen Sie diesen Schritt (Fensterbezeichnung vergeben) bei jeder der folgenden Programmänderungen aus.
9. Erstellen Sie das Projekt und führen Sie das Programm aus: „▶“

Wie Sie sehen, sehen Sie nix!

Das Programm macht trotzdem etwas: es initialisiert den frame buffer mit dem Default-Farbwert SCHWARZ und den depth buffer (=Tiefenpuffer = z-buffer) mit der maximalen Entfernung.

Die Initialisierung erfolgt durch den Funktionsaufruf
glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
Frage 1: Weshalb sollte dies in jedem Frame erfolgen? (Was kann insbesondere bei Animationen passieren, wenn man die Initialisierung weglässt?) → Frage_2

3. Änderung der Hintergrundfarbe

Ehe Sie mit den Änderungen beginnen, gehen Sie ab jetzt aus Sicherheitsgründen bei der Erstellung jeder neuen Version (Programmerweiterung) nach folgendem Schema vor; (im folgenden Beispiel heißt die alte Version „Nix“ und die neue Version „Gold“). Hierdurch wird gewährleistet, dass Sie ab jetzt die jeweils alten Programmversionen im Projekt weiter zur Verfügung haben und bei Bedarf wieder aktivieren können:

Schema:

1. Speichern Sie Nix.cpp unter dem Namen Gold.cpp:
   Datei> Speichern von Nix.cpp unter ...
2. Fügen Sie „Gold.cpp“ Ihrem Projekt hinzu.
3. Nehmen Sie Nix.cpp von der Projekterstellung aus: Im Projektmappen-Explorer Nix.cpp markieren, dann nach Rechts-Click: Eigenschaften>Allgemein>Vom Build ausschliessen>Ja.
4. Editieren Sie Gold.cpp.
5. Erstellen Sie das Projekt und führen Sie das Programm aus: „►“

In den nächsten Abschnitten wird eine derartige Aktion (1 bis 3) in folgender Notation angegeben:

„Vorgänger.cpp → Nachfolger.cpp“ (= Neue Datei im Projekt ersetzt alte Datei und wird getestet).

Hier also:

„Nix.cpp→Gold.cpp“

Ändern Sie nun also Ihr Gold.cpp:

Kopieren Sie die folgende Zeile in die Init-Funktion

```
glClearColor ( 0.33f, 0.225f, 0.0f, 1.0f );      // Was wird hier definiert?
```

Fügen Sie hinter die Funktion „glClear(...)“ in „RenderScene“ schon mal den folgenden Aufruf ein.

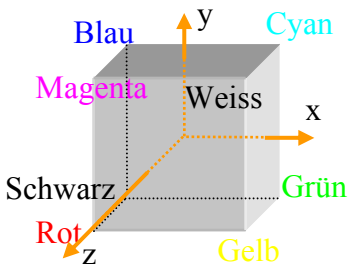
```
glLoadIdentity ();
```

Passen Sie im Folgenden auch die Bezeichnung Ihres Fensters immer dem aktuellen Zustand an.

→ Erstellen Sie das Projekt und führen Sie das Programm aus: „►“

4. Würfel darstellen

"Gold.cpp→ WuerfelKGCB.cpp" (→Ersetzen)



1. Fügen Sie die Dateien Wuerfel.cpp und Wuerfel.h Ihrem Projekt hinzu. Der Würfel ist wie rechts dargestellt aufgebaut. Der Ursprung seines körpereigenen Koordinatensystems liegt in der Würfelmitte.

Die als "Vorderseite" bezeichnete Seite liegt bei z = +fSeitenL/2. Dabei ist fSeitenL die Seitenlänge des Würfels (Parameter).

Die Darstellungsreihenfolge der 6 Seiten ist:
Vorderseite - rechte Seite - Rückseite - linke Seite - Deckfläche - Boden.

Die Farben der 8 Ecken (vertices) sind in der Skizze angegeben.

2. Fügen Sie folgende Zeile ein: `#include "Wuerfel.h"`

3. Fügen Sie in der Funktion „RenderScene“ vor dem Funktionsaufruf „glutSwapBuffers“ den Aufruf

```
Wuerfel (extent);
```

ein.

4. Deklarieren Sie „extent“ als globale Variable:

```
GLfloat extent = 1.0; // Mass fuer die Ausdehnung des Modells
```

5. Kopieren Sie die folgenden 5 Funktionsaufrufe in die „Reshape“- Funktion:

```
glMatrixMode(GL_PROJECTION); // Matrix für Transf.: Frustum->Viewport
glLoadIdentity();
glViewport(0,0,width,height);
glOrtho(-extent,+extent,-extent,+extent,-extent,+extent); // Frustum
glMatrixMode(GL_MODELVIEW); // Modellierungs-/Viewing-Matrix
```

Der Sinn ist folgender:

Sie haben jetzt ein Modell (nämlich den Würfel), das gerendert werden soll. Um die Abbildung auf dem Bildschirm durchzuführen, muss das 3D-Frustum auf einen 2D-Bereich des Bildschirms abgebildet werden. Die Maße des (quaderförmigen) Frustums werden in der Funktion glOrtho definiert; (Reihenfolge: linke, rechte, untere, obere vordere und hintere Begrenzung (=Clipping plane)). Im obigen Fall ist das Frustum ein Würfel mit der Seitenlänge 2*extent. Mit anderen Worten, der mit der Seitenlänge extent definierte Würfel "sitzt" in der Mitte des Frustums und füllt genau 1/8 dessen Volumens aus.

Das Frustum, samt seinem Inhalt, wird nun auf den 2D-Viewport projiziert. Letzterer nimmt wegen der Festlegungen in der Funktion

```
glViewport ( 0, 0, width, height);
```

genau die Abmessungen des Graphikfensters, das in „glutInitWindowSize (600,600);“ definiert wurde, ein. Der Viewport ist damit 600 x 600 Pixel groß.

Frage 2: Setzen Sie versuchsweise statt width width/2 und statt height height/2 ein. Was geschieht? Welche Funktion haben die beiden ersten Parameter, die im Aufruf die Werte 0 besitzen? (Sie finden das z.B. durch Ausprobieren heraus. → Frage_3

OpenGL verwendet neben der Transformationsmatrix, die für die obige Projektion erforderlich ist, und die deshalb "**projection matrix**" genannt wird, noch eine weitere. Diese dient dazu, das hierarchisch aufgebaute Modell zu erstellen (modelling) und zu betrachten (viewing). Letztere wird als "**modelview matrix**" bezeichnet.

Beide Matrizen werden als Einheitsmatrix initialisiert. Dies geschieht durch die Funktion „glLoadIdentity".

Da 2 Transformationsmatrizen existieren, muss zwischen 2 Modi umgeschaltet werden:

Nach der Ausführung von

„glMatrixMode(GL_PROJECTION);"

beziehen sich jetzt alle Aktivitäten auf den Projection-Modus. Folglich wird mit dem hierauf folgenden Aufruf von „glLoadIdentity" die Projection-Matrix initialisiert.
Nach der Ausführung von

„glMatrixMode(GL_MODELVIEW);"

beziehen sich alle Aktivitäten jetzt auf den ModelView-Modus. Folglich wird mit dem hierauf folgenden Aufruf von „glLoadIdentity" (der in „RenderScene" angesiedelt ist) die ModelView-Matrix initialisiert.

→ Erstellen Sie das Projekt und führen Sie das Programm aus: „▶"

Frage 3: Haben Sie zu Beginn dieses Abschnitts 4 den Würfel mit den von Ihnen erwarteten Würfel-Seiten angezeigt bekommen? Da Ihre Grafikkarte zu schnell ist, um den Bildaufbau zu beobachten, sollen Sie die Anzeige „künstlich" sehr stark verlangsamen:

```
#include <time.h>
…
```

und nach dem Aufruf des ersten Primitivs (Achtung: der Würfel ist kein Primitiv!!!):

```
glutSwapBuffers();
Sleep(1000); //nach diesem Experiment beides wieder entfernen!!!
```

… und dasselbe noch einmal hinter dem zweiten und dritten Primitiv!

→ Erstellen Sie das Projekt und führen Sie das Programm aus: „▶"

→ Frage_4

5. Würfel richtig darstellen

" WuerfelKGCB.cpp→ WuerfelRYWM.cpp" (→Ersetzen)

Frischen Sie Ihre Kenntnisse über den Tiefenpuffer (z-buffer) auf (gehört zur Vorbereitung!!).

Fügen Sie in der Funktion „Init" hinter „glClearColor" die beiden Funktionsaufrufe

```
glEnable(GL_DEPTH_TEST);
glClearDepth(1.0);
```

ein.

→ Erstellen Sie das Projekt und führen Sie das Programm aus: „►"

Frage 4: Was hat sich am Ergebnis verändert. Erläutern Sie die Veränderung.

→ Frage_5

Frage 5: Setzen Sie versuchsweise in glClearDepth den Parameter auf den Wert 0.0. Weshalb verschwindet der Würfel? Können Sie den Funktionsaufruf auch weglassen? Was schließen Sie aus Ihrer Antwort? → Frage_6

6. Würfel statisch drehen

" WuerfelRYWM.cpp → Wuerfelrotyx.cpp" (→Ersetzen)

Da Sie jetzt den Würfel mal wirklich in 3D sehen wollen, müssen Sie ihn als nächstes drehen. Um ihm zunächst eine Drehung um die y-Achse zu "verpassen", benötigen Sie einen Funktionsaufruf der Form:

```
glRotatef ( Winkel, x, y, z );
```

Hierbei wird durch Winkel (in Grad) das Maß der Drehung definiert und durch x,y,z eine Achse, um die die Drehung erfolgen soll. Beachten Sie, dass die „glRotatef"-Funktion die Drehung selbst noch nicht hervorruft, sondern lediglich die **modelview matrix** modifiziert. Am Bildschirm macht sich dies erst bemerkbar, wenn die entsprechenden Primitive in der Funktion Wuerfel gerendert werden.

Fügen Sie also jetzt unmittelbar vor dem Aufruf Wuerfel (extent) den Aufruf

```
glRotatef ( 30.0, 0.0, 1.0, 0.0 );  //Rotation +30 Grad um welche Achse?
```

ein.

Um den Würfel zusätzlich um die x-Achse zu drehen wird ein weiterer „glRotatef"-Aufruf benötigt. Damit diese letzte Rotation **nach** der Drehung um die y-Achse erfolgt, müssen Sie den Aufruf unmittelbar **vor** den anderen setzen.

Damit sieht die Folge der Aufrufe jetzt wie folgt aus:

```
glRotatef ( 20.0, 1.0, 0.0, 0.0 );  //Rotation um +20 Grad um welche Achse?
glRotatef ( 30.0, 0.0, 1.0, 0.0 );  //Rotation um +30 Grad um welche Achse?
Wuerfel (extent);
```

→ Erstellen Sie das Projekt und führen Sie das Programm aus: „►"

Voilà, der RGB–Würfel in voller Pracht !!!

7. Tisch

" Wuerfelrotyx.cpp → MeinTisch.cpp " (→Ersetzen)

1. Fügen Sie die Dateien Tisch.cpp, Tisch.h, Tischplatte.cpp und Tischplatte.h Ihrem Projekt hinzu.

Frage 6: In Tischplatte.cpp werden 3 unterschiedliche graphische Primitive verwendet. Welche? Deuten Sie in einer Skizze an, in welcher Reihenfolge dabei die vertices miteinander verbunden werden. → Frage_7

2. Fügen Sie folgende Zeilen ein:
```
#include "Tisch.h"
#include "Tischplatte.h"
```

Damit ist ein kompletter Tisch definiert. Rufen Sie jetzt in „RenderScene“ statt „Wuerfel“ „Tisch“ auf. Damit sieht „RenderScene“ jetzt wie folgt aus:

```
void RenderScene(void)
{
// Hier befindet sich der Code der in jedem frame ausgefuehrt werden muss

        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);// was loeschen?
        glLoadIdentity ();
        glRotatef ( 20.0, 1.0, 0.0, 0.0 );
        glRotatef ( 30.0, 0.0, 1.0, 0.0 );
        Tisch(extent);
        glutSwapBuffers();

}
```

→ Erstellen Sie das Projekt und führen Sie das Programm aus: „▶“

Skalieren Sie den Tisch durch Einfügung von

```
glScalef (0.5, 0.5, 0.5 );
```

direkt vor dem Tisch-Aufruf in allen 3 Achsen x, y und z auf die Hälfte, damit der Tisch nicht die komplette Darstellungsfläche einnimmt.

→ Erstellen Sie das Projekt und führen Sie das Programm aus: „▶“

Frage 7: Fertigen Sie einen Szenengraph für den Tisch an, der alle Objekte und Transformationen unter Benennung ihrer Namen enthält. Achten Sie auf die richtige Reihenfolge. Zeichnen Sie den Tisch im Querschnitt (2D-Seitenansicht) und fügen Sie sein körpereigenes Koordinatensystem hinzu.

8. Kamerasicht

" MeinTisch.cpp → LookAtTisch.cpp " (→Ersetzen)

1. Ersetzen Sie in „RenderScene“ die 3 Aufrufe

```
glRotatef ( 20.0, 1.0, 0.0, 0.0 );
glRotatef ( 30.0, 0.0, 1.0, 0.0 );
glScalef (0.5, 0.5, 0.5 );
```

durch den Aufruf

```
gluLookAt (0.0, 10.0*extent, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -1.0);  // Kamerasicht von wo?
```

und

2. die Definition des Frustums

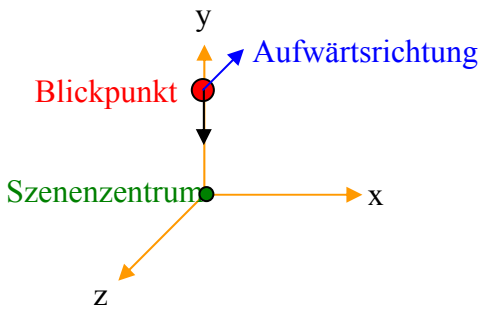
```
glOrtho(-extent,+extent,-extent,+extent,-extent,+extent); //Frustum
```

durch eine neue Definition

```
glOrtho(-extent*2.0,+extent*2.0,-extent*2.0,+extent*2.0,0.0,+20.0*extent); //Frustum
```

Damit haben Sie eine Kamera definiert, die

- sich an der Position **eyex = 0.0, eyey = 10.0, eyez = 0.0** befindet (Blickpunkt), die
- auf das Szenenzentrum **centerx = 0.0, centery = 0.0, centerz = 0.0** gerichtet ist, und die
- so gedreht ist, dass **upx = 0.0, upy = 0.0, upz = -1.0** die "Aufwärtsrichtung" der Kamera darstellt. Diese Größe verändert sich, wenn Sie die Kamera um ihre Hauptachse, also die Achse der Blickrichtung drehen:



gluLookAt (**eyex, eyey, eyez, centerx, centery, centerz, upx, upy, upz**);

→ Erstellen Sie das Projekt und führen Sie das Programm aus: „▶“

Versuchen Sie es auch mal mit

```
gluLookAt(5.0*extent, 5.0*extent, 10.0*extent, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0); // Kamera-
// sicht von wo?
```

(Die Aufwärtsrichtung zeigt jetzt in Richtung Tischbeinachse - so wie's sich ja gehört).

→ Erstellen Sie das Projekt und führen Sie das Programm aus: „▶“

Experimentieren Sie mit der Kamera und ihren inneren und äußeren Parametern.

9. Dynamik

" LookAtTisch.cpp → TischleinDeckDich.cpp " (→Ersetzen)

Zum Schluss soll noch an einem sehr einfachen Beispiel gezeigt werden, wie Animationen realisiert werden können. Hierzu sind folgende Änderungen am Programm erforderlich:

Definieren Sie „hoehe“ und „virthoehe“ global:

```
GLfloat hoehe = 0.0; //steuert die Auf- und Abwärtsbewegung des Tisches
GLfloat virthoehe = 0.0;  //steuert die Auf- und Abwärtsbewegung des Tisches
```

1. In der Animationsfunktion „Animate“ ersetzen Sie die Zeile

```
        std::cout << "value=" << value << std::endl;
```

durch

```
  static GLfloat richtung = 1.0;
  virthoehe = virthoehe + (float) 0.005*richtung;
  if ( virthoehe >= -extent &&  virthoehe <= extent) hoehe = virthoehe;
  if ( virthoehe <= -0.5*extent)        richtung = +1.0;
  if ( virthoehe >=  1.0*extent)  hoehe = extent;
  if ( virthoehe >=  1.5*extent)        richtung = -1.0;
```

2. Fügen Sie die Dateien „Boden.cpp“ und „Boden.h“ Ihrem Projekt hinzu. Ergänzen Sie Ihre Szene damit um einen Fußboden, durch den sich der Tisch hindurchbewegen soll.

3. Einfügen von: `#include "Boden.h"`

4. Rufen Sie Boden an der richtigen(!!!) Stelle auf.

5. Verwenden Sie:

```
        gluLookAt(5.0*extent, 5.0*extent, 10.0*extent, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0); // Kamera-
        // sicht von wo?
```

6. Fügen Sie vor dem Aufruf „Tisch(extent);“die Transformation

```
        glTranslatef( 0.0f, hoehe, 0.0f );
```

ein, die die Bewegung bewerkstelligen soll.

→ Erstellen Sie das Projekt und führen Sie das Programm aus: „▶“

… und jetzt sollte es funktionieren

Vergessen Sie bitte nicht die in Frage 1 angesprochene Nicht-Ausführung des glClear-Befehls für den Bild-Hintergrund. Der Tiefenpuffer muss trotzdem gelöscht werden:

```
  glClear ( GL_DEPTH_BUFFER_BIT );
```

10. Anhang

Achtung

Dinge, die so markiert sind, müssen in der Ausarbeitung behandelt werden.

Zeitplanung

Es gibt keine Vorgabe.
Empfehlung: die "geführte" Aufgabe möglichst schnell lösen.
Optimal: Vorbereitung zu Hause.
Spätestens nach dem 2. OpenGL-Termin sollten Sie mit diesem Teil fertig sein und der Lösungsweg der eigenen Aufgabe sollte klar ersichtlich sein.

Aufgabenstellung für die „Eigene Lösung“ im GDV-Praktikum (OpenGL-Teil)

Sie haben die freie Wahl, was Sie unter dem Thema „myLandingOnMars“ darstellen wollen. "Ihre" Landefähre soll ...:

in Phase 1: von schräg oben ins Bild schweben und langsam und sicher auf dem Mars aufsetzen;

in Phase 2: ...lassen Sie sich 'was einfallen! ;)

in Phase 3: abheben und aus dem Bild fliegen.

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 fliegendes 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- oder 3rd-Person-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 Graphik-Praktikums-Download-Groch finden Sie 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-Inkremente) schneller bzw. langsamer laufen zu lassen.

Viel Spaß und viel Erfolg

Wolf-Dieter Groch, Andreas Behr und Stephan Gimbel