Vorlesungsmodul Computergrafik
- VorlMod CompGra -

Matthias Ansorg

17. März 2003 bis 8. Oktober 2003

Zusammenfassung

Studentische Mitschrift zur Vorlesung Computergrafik bei Prof. Dr. Walter Bachmann (Sommersemester 2003) im Studiengang Informatik an der Fachhochschule Gießen-Friedberg. Nach Prof. Bachmann reicht das von ihm auf seiner Homepage zur Verfügung gestellte Material aus, um sich auf die Klausur vorzubereiten. Diese Mitschrift ist identisch zu [1] gegliedert.

- **Bezugsquelle:** Die vorliegende studentische Mitschrift steht im Internet zum Download bereitPersönliche Homepage Matthias Ansorg http://matthias.ansorgs.de/InformatikDiplom/Modul.CompGra.Bachmann/CompGra.pdf.
- **Lizenz:** Diese studentische Mitschrift ist public domain, darf also ohne Einschränkungen oder Quellenangabe für jeden beliebigen Zweck benutzt werden, kommerziell und nichtkommerziell; jedoch enthält sie keinerlei Garantien für Richtigkeit oder Eignung oder sonst irgendetwas, weder explizit noch implizit. Das Risiko der Nutzung dieser studentischen Mitschrift liegt allein beim Nutzer selbst. Einschränkend sind außerdem die Urheberrechte der angegebenen Quellen zu beachten.
- **Korrekturen und Feedback:** Fehler zur Verbesserung in zukünftigen Versionen, sonstige Verbesserungsvorschläge und Wünsche bitte dem Autor per e-mail mitteilen: Matthias Ansorg <mailto:matthias@ansorgs.de>.
- **Format:** Die vorliegende studentische Mitschrift wurde mit dem Programm LyX(graphisches Frontend zu LaTeX) unter Linux geschrieben und mit pdfLaTeXals pdf-Datei erstellt. Grafiken wurden mit dem Programm xfig unter Linux erstellt und als pdf-Dateien exportiert.
- **Dozent:** Prof. Dr. Walter Bachmann. Seit über 64 Semester an der FH Gießen beschäftigt: seit Oktober 1971. Studierte Elektrotechnik und Physik, teilweise Mathematik. War Mitgründer des Studiengangs Informatik an der FH Gießen-Friedberg in 1989.
- **Verwendete Quellen:** <quelle> {<quelle>}.
- **Klausur:**
  - In der ersten Vorlesung wird festgelegt, welche Formalitäten für die Veranstaltung gelten.
  - Das Praktikum besteht aus Übungen mit dem Computer, die am Ende des Semesters in ein Projekt münden.
  - Man kann sich aussuchen, an welcher Übung man teilnehmen will, und auch während des Semesters die Übungsgruppe wechseln. Es gibt 7-8 Praktikumsaufgaben. Wenn man einmal fehlt, ist das nicht besonders schlimm, denn »es könnte ja sein, dass diese Dinge nicht in der Klausur vorkommen«. In den Übungen wird programmiert mit dem Microsoft Visual Studio.
  - Es gibt Kontrollfragen zur Selbstkontrolle, die man jedoch nicht einreichen muss und die auch nicht korrigiert werden.
  - Es gibt zwei Hausübungen. Man kann damit 10% der Klausurpunkte als Pluspunkte erreichen und damit die Klausur verbessern. Ihre Abgabe ist jedoch keine Pflicht, d.h. es gibt keine formalen Klausurvoraussetzungen. Die Übungen sind inert 10 Tagen nach Aufgabenstellung per e-mail einzureichen, Datum der e-mail gilt. Eine Übung wird einem der Blätter mit Kontrollfragen zur Selbstkontrolle entsprechen, die andere wird eine Programmieraufgabe sein.
  - In der Klausur dürfen alle Hilfsmittel verwendet werden, die einem persönlich zugeordnet werden können: Ausdrucke von Skripten und Büchern, geliehene Bücher, handschriftliche Notizen usw. Notebooks jedoch dürfen nur verwendet werden, wenn vorher organisiert werden kann, wie genügend Steckdosen zur Verfügung stehen. Natürlich bestehen bei einer Fülle von Material die Probleme des information retrieval.

– Nach [6] zu urteilen enthalten die Klausuren hpts. Lückentext-Aufgaben; verlangt wird hauptsächlich OpenGL-Programmierung, d.h. Syntax und Semantik der OpenGL-Funktionsaufrufe muss man kennen oder nachschauen können. Daneben werden rel. einfache Rechnungen verlangt, etwa Matrizenmultiplikation, jedoch keine symbolischen Herleitungen und Beweise. Ein Grundverständnis des Stoffes ist also notwendig, jedoch muss man lange nicht alle Details des Skriptes [1] verstanden haben. Man braucht eine OpenGL-Referenz oder das Skript in brauchbarer Form, um mit den Programmieraufgaben zurechtzukommen. Die Wiederholungsfragen [3] dürften vom Typ der Klausuraufgaben sein, sind also gut zur Klausurvorbereitung geeignet. Die Praktikumsaufgaben haben wesentlich mehr Umfang als die Klausuraufgaben; die Programmieraufgaben davon sind auch schwerer, durch die gute Anleitung aber nicht wesentlich schwerer.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einführung

In [1] heißt dieses Kapitel »GDV-Script«.
Computergrafik nutzt Erkenntnisse und Methoden aus anderen Fachgebieten:

- mathematische Ableitungen und Alorithmen

- analytische Geometrie, Vektoren, Matrizen. In der Mathematik bestehen alle Koordinaten aus reellen Zahlen (Analogwerte), diese müssen auf Digitalwerte abgebildet werden. Matrizen dienen der Veränderung von Bildern. Um das natürliche Sehen nachzubilden, verwendet man Zentralprojektion, wozu 4×4-Matrizen nötig sind. Die Rechnung mit diesen Matrizen wird durch Karten hardwarebeschleunigt.
- physikalische Prinzipien: Optik.
- realitätsnahe Visualisierungsverfahren optischer Vorgänge: Raytracing, Radiosity, Rendering.
- geometrische Modellierung von Objekten. Das Objekt ist mehr als sein Bild, es besitzt Material, Form, Fähigkeiten, Gewicht und andere Eigenschaften. Das alles muss man in einer Datenstruktur darstellen, um Bewegungen des Objektes berechnen zu können. Das Bild ist dann nur die Projektion des Objektes auf eine Fläche unter Einfluss externer Faktoren wie Licht. Sehen ist die Wahrnehmung solcher Projektionen.

2 Einteilung der GDV

Computergrafik Es liegt noch kein fertiges Bild vor, sondern künstlich mit Algorithmen erzeugt.

Bildverarbeitung Es liegt ein Bild in Pixeln vor, das irgendwie aufbereitet werden muss: Kanten finden, Lungenvolumen aus CT berechnen, Kontrast verstärken usw. Diese Verfahren wirken mit Laplace-Operatoren auf die Bilder ein, wobei sie die Pixel modifizierend verändern und neue Bilder entstehen. Dies ist Stoff der Veranstaltung »Bildverarbeitung«.

Bildanalyse Bestimmte Informationen aus Bildern herausfiltern: Mustererkennung. Beispiele: Verkehrszählungen, Werbung erkennen. Hierzu gehört die Ermittlung topologischer Daten. Dies ist nicht Stoff der Veranstaltung Computergrafik.

Bilddatenreduktion, Kompressionsverfahren Reduktion des Datenumfangs von Bildern und Filmen. Dies ist nicht Stoff der Veranstaltung Computergrafik. Es gibt qualitätserhaltende und qualitätsmindernde Verfahren. Die besten qualitätsmindernden Verfahren erreichen in günstigen Fällen einen Kompressionsfaktor von 1000.

3 Mathematische Verfahren

Das Kapitel »Mathematische Verfahren« ist in [1] lediglich eine Art Formelsammlung, hilft aber nicht zum Verständnis.

4 Computergrafik

5 Gerätetechnik

5.1 Der CRT-Monitor

CRT bedeutet: Cathode Ray Tube, Kathodenstrahlröhre. Die Bildschirmoberfläche enthält 3 unterschiedliche Schichten Phospor für die drei Grundfarben - aufgrund von unterschiedlichen Sichtwinkeln von den Elektronenkanonen zu den Pixeln trifft jede Elektronenkanone nur Punkte ihrer eigenen Phopsporschicht. Es gibt ebenfalls drei Elektronenkanonen für die drei Grundfarben. Jedes Pixel wird im RGB-Format durch 3 Byte dargestellt, je eines für jede Grundfarbe. So sind insgesamt 3^{24} verschiedene Farben darstellbar. Unter OpenGL werden Farben durch drei Werte zwischen $0,0$ und $1,0$ dargestellt: `glColor3d` wandelt in die Werte der jeweiligen Grafikkarte um, die ja auch geringere Bereiche als $0 \ldots 255$ enthalten könnten.

Der Speicher der Grafikkarte enthält alle darzustellenden Pixel im RGB-Format und wird mit bestimmter Wiederholfrequenz vom Monitor ausgelesen. Grafikkarten enthalten Spezialbefehle wie Umsetzen von Linien oder 3D-Objekte in Pixel, die Prozessoren werden so entlastet. Grafikkarten können auch selbst programmierbar sein, womit der Anwender jedoch keinen Kontakt hat.

6 Grafische Algorithmen

6.1 Raster-Linien

6.2 Ganzzahlige Raster-Punkte

6.3 Bresenham-Algorithmus

Definition: Gaußklammer $[a]$ ist die größte ganze Zahl $z \leq a$: $a - 1 < z \leq a$ mit $z \in \mathbb{Z}$.

Rundungsfunktion Hier eine mathematische Formulierung:

$$r(x) = \left[a + \frac{1}{2}\right] \tag{1}$$

Prinzipielle Berechnungsmöglichkeit aller Pixel Nachdem n als maximaler Abstand in einer Koordinatenrichtung bekannt ist, können die Koordinaten aller Pixel g_k, h_k für $k \in [0, 1, \ldots, n]$ berechnet werden gemäß:

$$g_k = r\left(i_1 + k\frac{i_2 - i_1}{n}\right) \tag{2}$$

$$h_k = r\left(j_1 + k\frac{j_2 - j_1}{n}\right) \tag{3}$$

Dann sind die Punkte $Q_k(g_k, h_k)$ alle zu zeichnenden Pixel in ganzzahligen Koordinaten. Inklusive des Anfangspunktes $P_1 = Q_0$ und des Endpunktes $P_2 = Q_n$.

Kern des Bresenham-Algorithmus

```
1 wk = n >> 1;
2 //INV: wk < di
3 //INV: i = i1 + k
4 for ( k = 0 ; k <= n ; k ++ ) {
5   putPixel(i,j);
6   wk += dj; i += 1;
7   if ( wk >= di ) { wk -= di; j += 1; }
8 }
```

Bedeutung der Konstanten und Variablen:

i Momentaner Ganzzahliger Wert der x-Achsen-Koordinate in Pixeln.

j Momentaner Ganzzahliger Wert der y-Achsen-Koordinate in Pixeln.

n Maximum von d_i, d_j, hier d_i. Damit ist $n + 1$ die Anzahl der zu zeichnenden Pixel.

k Bloße Laufvariable. Begrenzt die Anzahl der Durchläufe und damit die Anzahl der gezeichneten Pixel auf $n + 1$.

d_i $d_i = |i_2 - i_1|$ bei Anfangspunkt $P_1(i_1, j_1)$ und Endpunkt $P_2(i_2, j_2)$. $\frac{d_i}{n} = \frac{d_i}{d_i} = 1$ ist die Schrittweite für i bei jedem Schritt der for-Schleife[1], d.h. i wächst mit jedem Schritt um 1 Pixel.

[1] Abzuleiten aus Gleichung 2.

d_j $d_j = |j_2 - j_1|$ bei Anfangspunkt P_1 und Endpunkt P_2. $\frac{d_j}{n} = \frac{d_j}{d_i}$ ist die Schrittweite für j bei jedem Schritt der for-Schleife[2], d.h. j wächst stets um weniger als 1 Pixel, denn $d_j < d_i$. Wenn $\frac{d_j}{n}$ ein Maß für die Schrittweite ist, dann auch d_j selbst.

w_k Sei j_r der größere, reelle Wert von j. Dann ist $w_k = (j_r - j) \cdot d_i$ eine ganze Zahl als Maß der Differenz $j_r - j$. Die Initialisierung mit $j_r - j = 0,5 \Rightarrow w_k = [0,5 \cdot d_i]$ gibt j einen halben Pixel Vorsprung; dies ersetzt die in Gleichung 3 ggf. durchgeführte Aufrundung um $0,5$.

Schritt für Schritt kommentiert:

Zeile 1 Division durch 2 mit bitshift-Operator. Warum $w_k = [0,5 \cdot n] = [0,5 \cdot d_i]$ initialisiert wird, wurde oben erklärt. Hier wird eine »flache Gerade« vorausgesetzt, also $d_j < d_i$: dann ist $\frac{d_i}{n} = 1$ und wird stets direkt zu i addiert, $\frac{d_j}{n} < 1$ und wird stets in w_k in angepasster Form (s.o.) aufsummiert.

Zeile 4 Die for-Schleife wird $n + 1$-mal ausgeführt, jedesmal wird ein Pixel gezeichnet.

Zeile 5 Einen Pixel $P(i, j)$ zeichnen. Der erste Pixel wird also für die Initialisierungswerte $i = i_1$, $j = j_1$ des Anfangspunktes $P_1(i_1, j_1)$ der Strecke gezeichnet.

Zeile 6 Ursprünglich war diese Zeile

```
if ( wk < di ) { wk += dj; i += 1; }
```

Durch die Schleifeninvariante gilt jedoch stets $w_k < d_i$, also kann die Bedingung entfallen. In dieser Zeile werden i und j entsprechend ihren Schrittweiten $\frac{d_i}{d_i} = 1$ bzw. $\frac{d_j}{d_i}$ erhöht. Weil $\frac{d_j}{d_i} < 1$, wird dieser Wert in w_k in angepasster Form zwischengespeichert.

Zeile 7 d_i entspricht einer Schrittweite $\frac{d_i}{d_i} = 1$ in Pixeln. Wenn also durch mehrfaches Aufsummieren von d_j endlich $w_k \geq d_i$ geworden ist, darf auch j um 1 Pixel erhöht werden. Damit wird die Differenz $j_r - j$ um 1Pixel kleiner, deshalb die Subtraktion $w_k - d_i$.

6.4 Bitmap-Code

6.5 GL-Linien

6.6 Zeichnen von Polygonen

6.7 Vertex-Arrays

6.8 Listen

6.9 LineStipple

6.10 PolygonStipple

[2] Abzuleiten aus Gleichung 3.

6.11 Clipping

```
FUNCTION clipLine(iMin,jMin,iMax,jMax:INTEGER; VAR i1,j1,i2,j2:INTEGER):BOOLEAN;
  VAR
    c,c1,c2:BYTE;
    i,j:REAL;
    flag:BOOLEAN;
  BEGIN
  flag:=FALSE; (* Arbeit noch nicht beendet *)
  c1:=clip( iMin,jMin,iMax,jMax, i1,j1);
  c2:=clip( iMin,jMin,iMax,jMax, i2,j2);
  REPEAT
    IF ((c1 OR c2)=0) THEN
      BEGIN (* beide Punkte innen *)
      flag:=TRUE; (* Arbeit beendet *)
      clipLine:=TRUE
      END
    ELSE
      BEGIN (* ein oder beide Punkte außen *)
        IF ((c1 AND c2)<>0) THEN
          BEGIN (* beide Punkte außen, keine Linienstück innen *)
          flag:=TRUE;
          clipLine:=FALSE
          END
        ELSE
          BEGIN (* ein oder beide Punkte außen, ein Linienstück innen *)
          IF NOT flag THEN
            BEGIN
            IF c1=0 THEN (* c1 innen? Dann Lage von c2 behandeln. *)
              c:=c2
            ELSE
              c:=c1; (* c1 außen? Dann Lage von c1 behandeln. *)
                (* c2 liegt evtl. auch außen und wird in einem nächsten *)
                (* Schleifendurchlauf behandelt *)
            IF ((c AND 1) <> 0) THEN (* behandelter Punkt hat i<iMin *)
              BEGIN
              i:=iMin;
              j:=j1+(j2-j1)/(i2-i1)*(i-i1)
              END;
            IF ((c AND 2) <> 0) THEN (* behandelter Punkt hat i>iMax *)
              BEGIN
              i:=iMax;
              j:=j1+(j2-j1)/(i2-i1)*(i-i1)
              END;
            IF ((c AND 4) <> 0) THEN (* behandelter Punkt hat j<jMin *)
              BEGIN
              j:=jMin;
              i:=i1+(i2-i1)/(j2-j1)*(j-j1)
              END;
            IF ((c AND 8) <> 0) THEN (* behandelter Punkt hat j>jMax *)
              BEGIN
              j:=jMax;
              i:=i1+(i2-i1)/(j2-j1)*(j-j1)
              END;
            END;
          IF c=c1 THEN (* behandelter Punkt war c1 *)
            BEGIN
            i1:=ROUND(i);
            j1:=ROUND(j);
            (* durchs Runden kann c1 wieder außerhalb des ClipArea liegen *)
            (* deshalb erneutes Clipping und Lagebehandlung im nächsten *)
            (* Schleifendurchlauf *)
            c1:=clip(iMin,jMin,iMax,jMax, i1,j1);
```

```
            END
          ELSE (* behandelter Punkt war c2 *)
            BEGIN
            i2:=ROUND(i);
            j2:=ROUND(j);
            (* Warum erneutes Clipping? Siehe letzten Kommentar. *)
            c2:=clip(iMin,jMin,iMax,jMax, i2,j2);
            END;
          END;
      END;
  UNTIL flag; (* Arbeit zu Ende *)
END;
```

7 2D-Welt-Koordinaten

7.1 Rxy-Welt nach Ixy-Screen

»Rxy« sind x/y-Koordinaten als Real-Zahlen; »Ixy« sind x/y-Koordinaten als Integer-Zahlen.
Warum wird auf einmal mit zwei Geraden in Zweipunkteform herumhantiert? Weil:

- Die Abbildung der x-Koordinaten auf die i-Koordinaten ist eine lineare Abbildung, also eine Gerade. Vergleiche ein R/I-Koordinatensystem mit den Punkten (x_{min}, i_{min}) und (x_{max}, i_{max}), durch eine Gerade verbunden.
- Die Abbildung der y-Koordinaten auf die j-Koordinaten ist ebenfalls eine lineare Abbildung, also eine Gerade. Vergleiche ein R/I-Koordinatensystem mit den Punkten (y_{min}, j_{min}) und (y_{max}, j_{max}), durch eine Gerade verbunden.

Die Funktion für die horizontalen Koordinaten, als Geradengleichung in Zweipunkteform, zum Funktionswert aufgelöst[3]:

$$i = r\left(i_{min} + \frac{i_{max} - i_{min}}{x_{max} - x_{min}} \cdot (x - x_{min})\right) \tag{4}$$

Um Konstante zusammenzufassen, führen wir $x_{Ti} = \frac{i_{max} - i_{min}}{x_{max} - x_{min}}$ ein. Damit ergibt sich aus Gleichung 4:

$$\begin{aligned} i &= r\left(i_{min} + x_{Ti} \cdot (x - x_{min})\right) \\ &= r\left(i_{min} + x \cdot x_{Ti} - x_{min} \cdot x_{Ti}\right) \end{aligned} \tag{5}$$

Um noch mehr Konstante zusammenzufassen, führen wir $x_{Si} = i_{min} - x_{min} \cdot x_{Ti} + \frac{1}{2}$ ein. Dabei integrieren wir auch noch die Rundungsfunktion entsprechend Gleichung 1: Addition von $\frac{1}{2}$ in x_{Si} und anschließende Anwendung der Gauß-Klammer durch die Umwandlung von Real- in Integer-Zahl im Computer.

$$i = [x \cdot x_{Ti} + x_{Si}]$$

Dies ist nun eine einfache Geradengleichung in Hauptform. Ihr i-Achsenabschnitt ist x_{Si}, d.i. i für $x = 0$[4]. Ihre Steigung ist x_{Ti}, d.i. der Skalenumwandlungsfaktor zwischen x- und i-Achse[5].

Analog berechnen wir die Funktion für die vertikalen Koordinaten:

$$j = [y \cdot y_{Tj} + y_{Sj}]$$

mit $y_{Tj} = \frac{j_{min} - j_{max}}{y_{max} - y_{min}}$ und $y_{Sj} = j_{min} - y_{Tj} \cdot y_{max}$. Die Unsymmetrie in den Formeln für x_{Ti} und y_{Tj} kommt daher, dass die j-Achse entgegen der reellen y-Achse orientiert ist, nämlich von oben nach unten.

7.2 Isotrope 2D-Abbildung

Formeln für die Breite und Höhe des auf dem Bildschirm sichtbaren Rechtecks[6]:

$$\begin{aligned} d_k &= a \cdot d_i \\ d_l &= b \cdot d_j \end{aligned}$$

[3] Die Rundungsfunktion $r(x)$ wurde definiert in Gleichung 1.
[4] Vielleicht steht S in x_{Si} ja für »space« wie »Raum«, hier i.S.v. Mindestwert?
[5] Vielleicht steht T in x_{Ti} ja für »tranformation«, d.i. Umwandlung?
[6] Korrigiert gegenüber [1].

Erste Formel für die Korrektur, so dass eine isotrope Abbildung entsteht:

$$1 = \frac{d_y \cdot a \cdot d_i}{(d_x + x_x) \cdot b \cdot d_j} \Leftrightarrow 1 + \frac{x_x}{d_x} = \frac{d_y \cdot a \cdot d_i}{d_x \cdot b \cdot d_j} \Leftrightarrow x_x = (h-1) \cdot d_x,\ y_y = 0$$

Wenn im Viewport ein Viereck mit falschen Proportionen dargestellt wird, warum wird denn dann dieses Viereck in der 2D-Welt geändert statt seine Darstellun im Viewport? Seit wann wird das Objekt dem Betrachter anepasst? Achtung: es geht nicht um beliebige Vierecke, die als Formen dargestellt werden, sondern um das Viereck, das die Grenzen des Viewport bildet. Dieses ist meist durch das Gerät vorgegeben. Um nun isotrope Darstellung zu erhalten, könnte man die Proportionen des Viewport ändern, indem man nur mehr einen Teil des Viewports nutzt, um den aktuellen Ausschnitt de 2D-Welt darzustellen. Das wendet man auch an. Zusätzlich nutzt man den freigewordenen Teil des Viewport, um noch mehr von der 2D-Welt darzustellen. Das heißt, man ändert das beobachtete Viereck in der 2D-Welt!

7.3 Entwurf einer Schnittstelle

7.4 GL-Viewport

7.5 GL-normalisierte Koordinaten

Wie wird verhindert, dass durch Umrechnung in normalisierte Koordinaten jede Szene der 2D-Welt ohneBerücksichtigung der Proportionen in ein Quadrat bzw. einen Würfel von $2LE$ Seitenlänge gepresst wird? Etwa indem $x_{max} - x_{min} = y_{max} - y_{min} = z_{max} - z_{min}$ gefordert wird?

8 OpenGL-Einführung

NURBS Non-Uniform Rational B-Spline

8.1 GL-Syntax

8.2 GL-Fehler

8.3 Viewport

8.4 Display-Listen

9 Matrizen-Transformationen

Die Mathematik bei Matrizen-Transformationen und der hintergrund homogener Koordinaten wird erklärt in [10, Kap. 15]. Parameter im Aufruf `gl_Rotated(double a, double u,double v,double w)`:

a Drehwinkel

u, v, w Ursprungsgerade $\vec{X} = \lambda \begin{pmatrix} u \\ v \\ w \end{pmatrix}$ als Drehachse.

Die OpenGL-Funktion `glOrtho()` und damit auch die eigene Reimplementierung `gl_Ortho()` in diesem Kapitel leisten eine Parallelprojektion. Zu Projektionen siehe Kapitel 10.

10 Projektionen

Einteilung der Projektionen

- Projektionen mit konvergierenden Projektionsstrahlen. Beispiele:
  - Zentralprojektion (auch genannt »Perspektive«). Eingesetzt im Auge und in der Kamera.
- Projektionen mit parallelen Projektionsstrahlen. Weitere Unterteilung:
  - schiefe Projektionen (Strahlen nicht senkrecht zur Projektionsebene). Beispiele:
    - Isometrie
    - Dimetrie
    - Kavalliersperspektive
    - Militärperspektive
  - orthogonale Projektionen (Strahlen sind senkrecht zur Projektionsebene)

10.1 Isometrie, Dimetrie, Kavalliersperspektive, Militärperspektive

Die Prinzipien, um die Abbildung [1, S. 115] verstehen zu können:

- Die Parallelprojektion wird hier in einer perspektivischen Zeichnung dargestellt. Deshalb können die Prinzipien der Parallelprojektion hier schwer erkannt werden. Parallele Projektionsstrahlen sehen in der perspektivischen Darstellung ja nicht mehr parallel aus!
- Man stelle sich das Koordinatensystem mit parallelem Licht (etwa Sonnenlicht) auf die Zeichenebene projeziert vor. Die Zeichenebene denke man sich als beliebig große Fläche mit fester Lage im räumlichen Koordinatensystem.
- Hier liegt der Projektionsstrahl in der x/y-Ebene des Koordinatensystems, d.h. er steht senkrecht auf der z-Achse.
- Der Schnittwinkel des Projektionsstrahls mit der y-Achse ist $90° - w_1$, der Schnittwinkel mit der x-Achse ist $90° - w_2$. w_1 und w_2 sind also Winkel in der x/y-Ebene und nicht in der Zeichenebene! Auch die beiden eingezeichneten Dreiecke, in denen diese Winkel vorkommen, liegen also in der x/y-Ebene. Sie wurden ausgewählt, weil sie besondere Projektionen liefern: ihre Strecken liegen teilweise in Richtung der Achsen u bzw. v der Zeichenebene, sind also unmittelbar als Summanden verwendbar, um die Koordinaten (u, v) zu berechnen. Diese beiden Dreiecke haben sowohl in der perspektivischen Darstellung als auch in der Realität einen rechten Winkel, weil eine Strecke in jedem Dreieck auf den Fluchtpunkt der perspektivischen Darstellung ausgerichtet ist. Diese beiden Dreiecke wären jedoch in der Parallelprojektion nicht sichtbar, denn eine Strecke in jedem Dreieck ist eine projezierende Gerade, d.h. parallel zum Projektionsvektor.

Es bleiben einige Fragen die es möglich machen dass o.a. Abbildung doch falsch verstanden ist :) :

- Aber wenn der Projetionsstrahl senkrecht zur z-Achse wäre, so würde die z-Koordinate doch stets längentreu abgebildet. Hier aber ist die Abbildung $z - y \cdot \sin w_1 - x \cdot \sin w_2$.
- Eine Parallelprojektion ist also eine Projektion in den angegebenen Raumausschnitt bzw. die angegebene Fläche?
- Die in [1, S. 115] genannten Winkelpaare ergeben nicht stets $w_1 + w_2 = 90°$. Das heißt, die Abbildung beinhaltet überhaupt keine perspektivische Darstellung, die Winkel w_1, w_2 sind keine Winkel im Raum, sondern in der Fläche, und bestimmen die Art der Parallelprojektion?

Grundlagen zur Parallelprojektion:

- http://www.geometrie.tuwien.ac.at/asperl/gzbuch2k/grundlagen/parallelprojektion.htm
- http://graphics.cs.uni-sb.de/Courses/ws0001/cg/folien/Projektion.pdf

- Es gibt zwei Wichtige Matrizen »Mat« M und »Ortho« O, die zueinander invers sind, d.h. es gilt:

$$M \cdot O = O \cdot M = E_4 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

  Offensichtlich dient »Ortho« der Normalisierung von Koordinaten vor der Parallelprojektion auf die Zeichenebene, d.h. der minimale Kordinatenwert wird stets auf -1, der maximale Koordinatenwert stets auf 1 abgebildet. Eine Zentralprojektion wird also ersetzt durch eine Normalisierung der Koordinaten und eine Parallelprojektion. Bedeutet das, dass die Koordinaten bei der Normalisierung je nach Abstand vom Betrachter mehr oder weniger gestaucht werden?

10.2 Orthogonale Projektion

10.3 Allgemeine Orthogonale Projektion

10.4 Orthogonale Projektion

10.5 GL-Projektion

10.6 Zentralprojektion

10.7 Wahl der Bildebene

10.8 Wahl des Bildmittelpunktes

10.9 Zentralprojektion in homogenen Koordinaten

10.9.1 1. Sonderfall

10.9.2 2. Sonderfall

10.10 Zentralprojektion in Kugel

10.11 Sichtbarkeiten

10.11.1 Normalen- und Flächenvektor

10.11.2 Analytische Berechnung des Normalen-Vektors

11 Farbe

12 HTML-Farben

13 Objekte

14 Texturen

15 Praktikum

OpenGL unter Linux:
Zuerst installiere man folgende Pakete:

- mesa
- mesademos
- mesa-devel
- mesaglu
- mesaglu-devel
- mesaglut
- mesaglut-devel
- 3Ddiag

Wichtige Verzeichnisse einer Mesa-Installation:

- /usr/share/doc/packages/mesa/
- /usr/lib/mesa/bin/
- /usr/share/man3/gl*
- /usr/include/GL/

SuSE enthält keine Source-Codes lauffähiger Mesa-Programme. Also aus dem Internet besorgen.

15.1 1. Praktikum

15.2 2. Praktikum

15.3 3. Praktikum

15.4 4. Praktikum

15.5 5. Praktikum

15.6 6. Praktikum

15.7 7. Praktikum

16 Testklausur

Quelle: [6]. Alles, was zur Aufgabenstellung gehört, steht in »Anführungszeichen«. Die Lösungen wurden selbst entwickelt und nicht überprüft.

```
GDV-Klausur
Tag:26.9.2002
Name,Vorname: . . . . . .
Matrikel-Nr : . . . . . .
hg-Nummer-Nr: . . . . . .
        Aufg: | 1 | 2 | 3 | 4 | 5 |
        Soll: | 3 | 3 | 3 | 3 | 3 | Pkte
```

16.1 Aufgabe 1

»Für eine Drehung um die z-Achse kann die folgende 4×4-Matrix $A(w)$ verwendet werden:«

$$A(w) = \begin{pmatrix} \cos w & -\sin w & 0 & 0 \\ \sin w & \cos w & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

1. »Eine Drehung um den Winkeln $w = \frac{\pi}{2}$ ergibt die Zahlen-Matrix:«

$$A = \begin{pmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

2. »Die Matrix A soll für eine Spiegelung an der z-Achse verwendet werden. Für den Winkel« $w = \pi$ »ergibt sich die Spiegelungsmatrix $S = A(w)$«

$$S = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

3. »Ein Punkt P in homogenen Koordinaten sei: $P = (-3; 2; 1; 1)$. Welche Koordinaten ergeben sich für $Q = S \cdot P$?«

$$Q = \begin{pmatrix} 3 \\ -2 \\ 1 \\ 1 \end{pmatrix}$$

16.2 Aufgabe 2

16.2.1 Teil 1

»Geben Sie mindestens 2 Stichworte an zu:«

1. »Computer Grafik«
   - behandelt künstlich hergestellte Bilder
   - mit Programmen aus Datentypen
   - Grundelemente Pixel, Linien, Flächen, Texte, Objekte
2. »image processing«
   - verarbeitet unstrukturierte Pixelmengen
   - Verfahren zur Bildverbesserung
   - Beispiele: Kontrast, Rauschunterdrückung, Konturerfassung
3. »picture analysis«
   - erhält unstrukturierte Pixelmengen
   - will Strukturen in Pixelmengen erkennen
   - Objekte finden durch Mustererkennung
   - geometrische und topologische Daten erhalten
4. »Kompressionsverfahren«
   - verlustfreie Kompression
   - verlustbehaftete Kompression
   - Geschwindigkeit wichtig
5. »Bresenham-Algorithmus«
   - Raster-Linien zeichnen
   - keine Multiplikationen nötig
   - keine floating-point-Arithmetik nötig
6. »homogene Koordinaten«
   - notwendig, um Translationen durch Matrizenmultiplikationen ausdrücken zu können
   - beliebig viele beliebige Transformationen durch Matrizenmultiplikation in einer 4×4-Matrix darstellbar
7. »Viewport«
   - Sichtfenster mit beschränkten ganzzahligen Gerätekoordinaten auf die virtuelle Welt mit unbeschränkten reellen Koordinaten
   - Transformation der Koordinaten nötig
   - isotrope (winkeltreue) oder anisotrope (nicht winkeltreue) Transformation
8. »Die Erscheinungsform des Maus-Cursors entsteht durch Verknüpfung mit einer« 16-Word-AND»-Maske und« 16-Word-XOR»-Maske.«

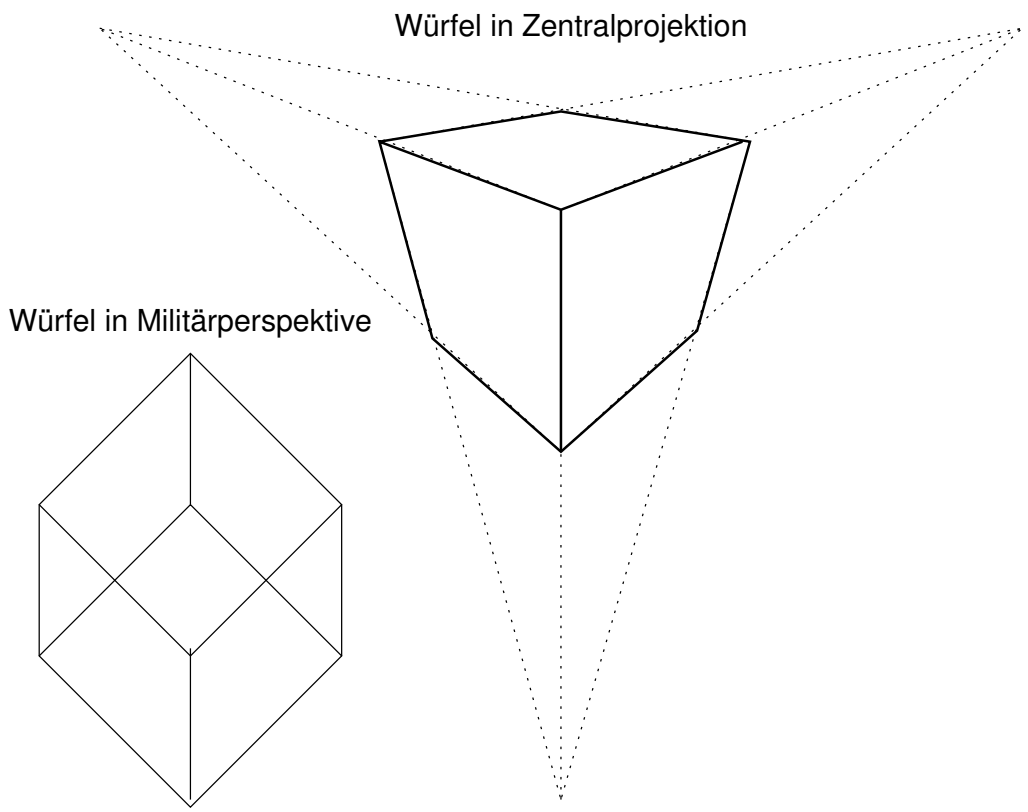

Abbildung 1: Würfel in Militärpersektive und Zentralperspektive

16.2.2 Teil 2

»Skizzieren Sie frei Hand einen Würfel:«

1. »in Militärperspektive«
2. »in Zentralprojektion«

Lösung siehe Abbildung 1.

16.3 Aufgabe 3

»Die folgende Aufgabe besteht aus den Teilaufgaben [1],[2],[3]. Es soll eine 2D-Figur gezeichnet werden.«

16.3.1 Teil 1

»Ergänzen Sie zunächst eine Funktion `draw_nEck(int n)` so, dass EIN gefülltes n-Eck mit einer Spitze nach oben gezeichnet wird. Hierzu nutzen Sie bitte die Kreisgleichung in Parameter-Form: $\mathrm{Radius} = 1.0$, Kreis-Mittelpunkt bei $(0,0;0,0)$.

```
void draw_nEck(int n) {
    double x, y, w, PI= 3.14159265;
    glBegin( ______________________ );
    for( int i=0; i < n; i++ ) {
      w = PI * ___________________;
      y = ________________________; //trig Fkt
      x = ________________________; //trig Fkt
      glVertex2d( x, y );
    }
    glEnd();
}//«
```

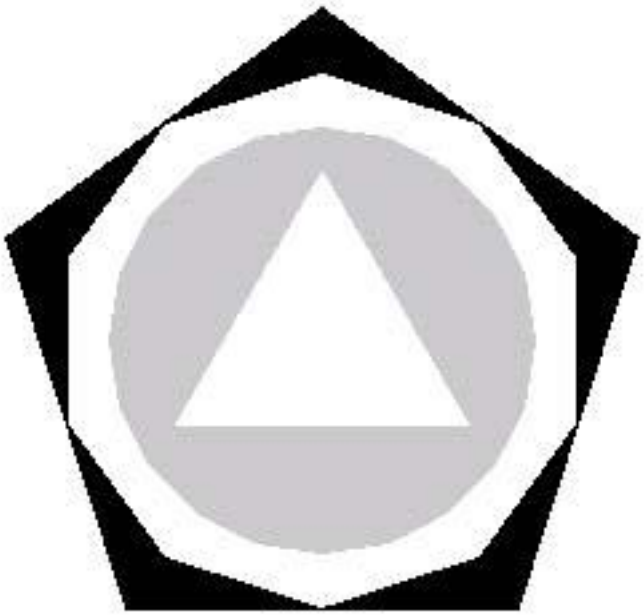

Abbildung 2: Zu Aufgabe 3 der Testklausur

Lösung (Quelle: [9, S. 1]):

```
void draw_nEck(int n) {
    double x, y, w, PI= 3.14159265;
    glBegin( GL_POLYGON );
    for( int i=0; i < n; i++ ) {
      w = PI * 2 * i / n;
      y = cos(w); //trig Fkt
      x = sin(w); //trig Fkt
      glVertex2d( x, y );
    }
    glEnd();
}
```

16.3.2 Teil 2

»Die Funktion `draw_nEck(int n)` soll nun zum Zeichnen einer zusammen gesetzten 2D-Figur verwendet werden. Hierzu soll die Funktion `draw_bild()` ergänzt werden. Die Figur besteht aus 4 regelmässigen n-Ecken unterschiedlicher Grösse, d.h. `draw_nEck()` wird 4 mal aufgerufen. Die n-Ecke sind in der $x - y$-Ebene.

- In ein 5-Eck mit Radius $1,0$ und der Farbe schwarz wird ein 10-Eck mit Radius $0,8$ und der Farbe weiss gezeichnet.
- In das 10-Eck wird ein 20-Eck der Farbe grau und dem Radius $0,64 = 0,8 \cdot 0,8$ gezeichnet.
- In das 20-Eck wird ein 3-Eck mit Radius $0,512 = 0,8 \cdot 0,8 \cdot 0,8$ und der Farbe weiss gezeichnet.

Siehe Abbildung 2. Bitte ergänzen Sie:

```
void draw_bild( void ) {
  glPushMatrix();
  glColor3d( ____, ____, ____ );draw_nEck( ___ );
  glScaled ( ____, ____, ____ );
  glColor3d( ____, ____, ____ );draw_nEck( ___ );
```

```
  glScaled ( ____, ____, ____ );
  glColor3d( ____, ____, ____ );draw_nEck( ___ );
  glScaled ( ____, ____, ____ );
  glColor3d( ____, ____, ____ );draw_nEck( ___ );
  glPopMatrix();
}//«
```

Lösung:

```
void draw_bild( void ) {
  glPushMatrix();
  glColor3d(0.0, 0.0, 0.0); draw_nEck(5);
  glScaled (0.8, 0.8, 0.8);
  glColor3d(1.0, 1.0, 1.0);draw_nEck(10);
  glScaled (0.8, 0.8, 0.8);
  glColor3d(0.5, 0.5, 0.5);draw_nEck(20);
  glScaled (0.8, 0.8, 0.8);
  glColor3d(1.0, 1.0, 1.0);draw_nEck(3);
  glPopMatrix();
}
```

Eine lauffähige Version liegt unter http://matthias.ansorgs.de/InformatikDiplom/Modul.CompGra.Bachmann/Aufg.nEck.cpp.

16.3.3 Teil 3

»Bitte ergänzen Sie an den vorgesehenen Stellen die Funktion `ChangeSize()`, wenn die gezeichneten Figuren (auch bei Viewport-Vergrösserung) winkeltreu bleiben sollen.«

```
#include <windows.h>
#include <gl\glaux.h>
#include <gl\gl.h>
//#include  _________________________________
#include  _________________________________ //Lösung
void draw_nEck(int n);
void draw_bild(void);
double xMin = -1.5, xMax = +1.5;
double yMin = -1.5, yMax = +1.5;
void CALLBACK ChangeSize(int w, int h) {
  glViewport(0,0,w,h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  if (w>h)
    //gluOrtho2D(xMin*____, xMax*_______,yMin,yMax);
    gluOrtho2D(xMin*w/h, xMax*w/h, yMin, yMax); //Lösung
  else
    //gluOrtho2D(xMin,xMax,yMin*_______,yMax*_______);
    gluOrtho2D(xMin, xMax, yMin*h/w, yMax*h/w); //Lösung
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
}
void CALLBACK RenderScene( void ) {
  glClear(GL_COLOR_BUFFER_BIT);
  draw_bild();
  glFlush();
}//«
```

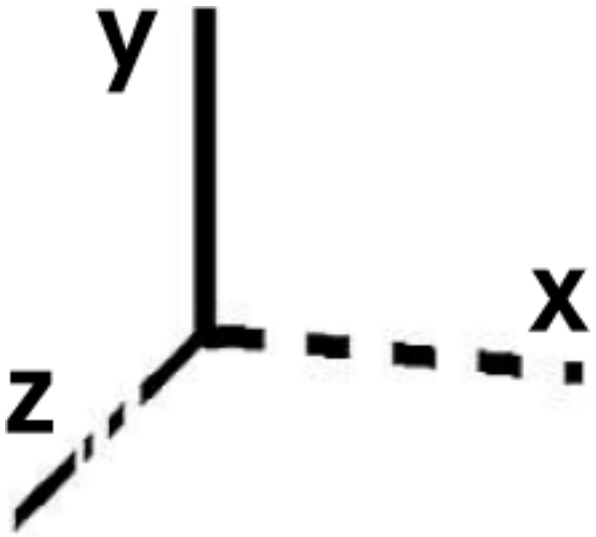

Abbildung 3: Zu Aufgabe 4 der Testklausur

16.4 Aufgabe 4

16.4.1 Teil 1

»Es soll eine Funktion `drawAchsen()` ergänzt werden. `drawAchsen()` soll die 3 Koordinaten-Achsen für ein x, y, z-System zeichen. Schreiben Sie von Hand $'x'$, $'y'$, $'z'$ (wie es bei OpenGl üblich ist) an die folgende Skizze.« (siehe Abbildung 3; dort ist bereits die Lösung eingetragen).

Ergänzen Sie bitte:

- Bei (a): Damit auch gestrichelte Linien möglich sind, soll 'line-stipple' ermöglicht werden.
- Bei (b): Die y-Achse soll als durchgezogene Linie vom Ursprung $(0; 0; 0)$ zum Punkt $(0; 1, 3; 0)$ gezogen werden.
- Bei (c): Die x-Achse soll vom Ursprung $(0; 0; 0)$ zum Punkt $(1, 3; 0; 0)$ gezeichnet werden, wobei beim Zeichnen das Bit-Stipple-Muster '1111 1111 0000 0000' (in dieser Reihenfolge) zum 'Stricheln' verwendet wird.
- Bei (d): Die z-Achse soll vom Ursprung $(0; 0; 0)$ zum Punkt $(0; 0; 1, 3)$ gezeichnet werden, wobei beim Zeichnen das Bit-Stipple-Muster '1111 1111 1100 0100' (in dieser Reihenfolge) zum 'Strichpunktieren' verwendet wird.
- Bei (e): Bitte `drawAchsen()` ergänzen.

```
void drawAchsen( ) {
  glEnable(_______________);//(a)
  glBegin(________);//(b)
   glVertex3d(___, ___, ___);
   glVertex3d(___, ___, ___);
  glEnd();
  glLineStipple(__, _______);//(c)
  glBegin(________);
   glVertex3d(0.0,0.0,0.0);
   glVertex3d(1.3,0.0,0.0);
  glEnd();
  glLineStipple( ____, __________ );//(d)
  glBegin( _____________ );
   glVertex3d(0.0,0.0,0.0);
   glVertex3d(0.0,0.0,1.3);
  glEnd();
  //(e)
  ------------------------------------------
}//«
```

Lösung:

```
void drawAchsen( ) {
  glEnable(GL_LINE_STIPPLE);//(a)
  glBegin(GL_LINES);//(b)
   glVertex3d(0.0, 0.0, 0.0);
   glVertex3d(0.0, 1.3, 0.0);
  glEnd();
  glLineStipple(1, 0xFF00);//(c)
  glBegin(GL_LINES);
   glVertex3d(0.0,0.0,0.0);
   glVertex3d(1.3,0.0,0.0);
  glEnd();
  glLineStipple(1, 0xFFC4);//(d)
  glBegin(GL_LINES);
   glVertex3d(0.0,0.0,0.0);
   glVertex3d(0.0,0.0,1.3);
  glEnd();
  //(e)
  glLineStipple(1, 0xFFFF); glDisable(GL_LINE_STIPPLE);
  drawAchsen(); //???
}
```

16.4.2 Teil 2

»drawAchsen() soll nun in der CALLBACK-Funktion RenderScene() aufgerufen werden. Die Achsen sollen mit der HTML-Farbe '#rrggbb' = '#202064' und der Linien-Dicke 6 gezeichnet werden. Hinweis: rr, gg, bb sind Hex-Ziffern ($0x00 \leq rr \leq 0xFF$, $0x00 \leq gg \leq 0xFF$, $0x00 \leq bb \leq 0xFF$). Bitte ergänzen Sie!

```
// Wie sieht die HTML-Farbe '#202064' etwa aus?
// Mit Worten: _________________________________
void CALLBACK RenderScene(void) {
  glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
  glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
  ////////////////////////////////////
  glEnable( ______________________ );
  ---------------------------------
  ---------------------------------
  drawAchsen();
  glDisable( ______________________ );
  ////////////////////////////////////
  auxSwapBuffers();
}//«
```

Lösung:

```
// Wie sieht die HTML-Farbe '#202064' etwa aus?
// Mit Worten: dunkelgraugetrübtes blau
void CALLBACK RenderScene(void) {
  glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
  glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
  ////////////////////////////////////
  glEnable(GL_LINE_SMOOTH);
  glLineWidth(6.0);
  glColor3b(0x20, 0x20, 0x64);
  drawAchsen();
  glDisable(GL_LINE_SMOOTH);
  ////////////////////////////////////
  auxSwapBuffers();
}
```

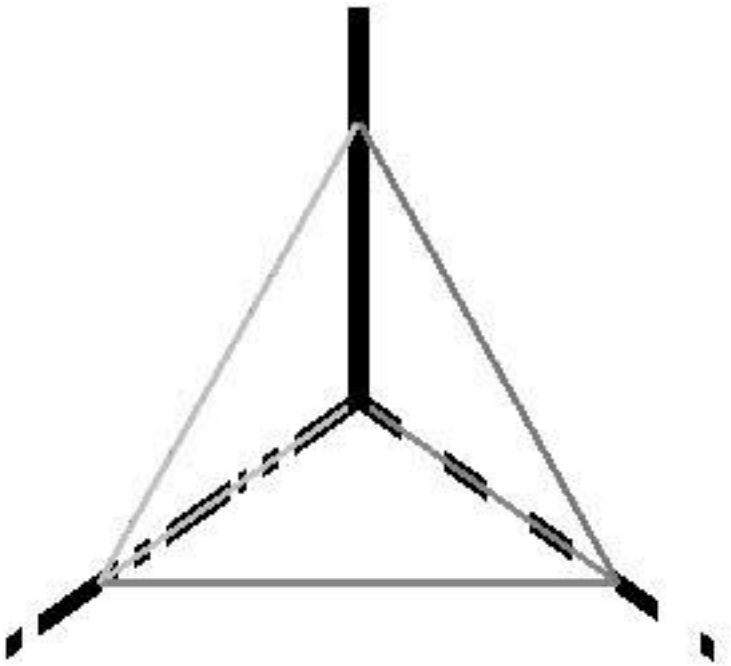

Abbildung 4: Zu Aufgabe 5 der Testklausur

Eine lauffähige Version der Lösung zu dieser ganzen Aufgabe liegt unter http://matthias.ansorgs.de/InformatikDiplom/Modul.CompGra.Bachmann/Aufg.nEck.cpp.

16.5 Aufgabe 5

16.5.1 Teil 1

»Ein pyramiden-förmiger Körper habe 4 Dreiecks-Flächen und die Eckpunkte $P_0\,(0;0;0)$, $P_1\,(1;0;0)$; $P_2\,(0;1;0)$, $P_3\,(0;0;1)$.

- Fläche 0: P_1, P_3, P_0.
- Fläche 1: P_2, P_1, P_0.
- Fläche 2: P_3, P_2, P_0.
- Fläche 3: P_1, P_2, P_3.

Die Flächen-Normalen sind $N_0\,(-1;0;0)$, $N_1\,(0;-1;0)$, $N_2\,(0;0;-1)$, $N_3\,(n_{3x};n_{3y};n_{3z})$. Siehe Abbildung 4.

Berechnen Sie die normierte Flächen-Normale $N_3\,(n_{3x};n_{3y};n_{3z})$, wenn die Eckpunkte (in dieser Reihenfolge) der Fläche 3 durch $P_1\,(1;0;0)$, $P_2\,(0;1;0)$, $P_3\,(0;0;1)$ gegeben sind. Hinweis: Die nicht normierte Normale ist $(P_2 - P_1) \times (P_3 - P_?)$.«

»Die Funktion drawPyramide() soll die Pyramide zeichnen und ist zu ergänzen:

- Bei (a): Tragen Sie die Komponenten der normierte Flächen-Normale $N_3\,(n_{3x}, n_{3y}, n_{3z})$ ein.
- Bei (b): i soll als Index alle Flächen durchlaufen.
- Bei (c): $j = 0, 1, 2$ wird (je Fläche i) als Index zum Durchlaufen der Eckpunkte einer Fläche verwendet. Aus: (i, anzP, j) soll j_0 als Index in die Punktetabelle berechnet werden. j_0 ist dann der x-Index des aktuellen Eck-Punktes P der Punktetabelle.
- Bei (d): Der Funktionsparameter `art` von `drawPyramide(art)` soll nun verwendet werden. Für einen Aufruf mit $\mathrm{art} = 1$ soll ein Pyramiden-Drahtmodell und für $\mathrm{art} = 2$ soll ein Pyramiden-Flächenbegrenzungsmodell gezeichnet werden.

```
//(a) Die normierte Flächen-Normale N3(n3x,n3y,n3z) ist:
//
//   n3x = ________, n3y = ________, n3z = ________,
//
#define anzA  4 // Anzahl von Flaechen
#define anzP  3 // Anzahl Pkt je Flaeche
void drawPyramide(int art) {
  double P[ 3*anzA ] = { // Punktetabelle
     0.0, 0.0, 0.0, // P0(x0,y0,z0)
     1.0, 0.0, 0.0, // P1(x1,y1,z1)
     0.0, 1.0, 0.0, // P2(x2,y2,z2)
     0.0, 0.0, 1.0  // P3(x3,y3,z3)
  };
  double N[ 3*anzA ] = {  // Normalentabelle
   -1.000, 0.000, 0.000,// N0
    0.000,-1.000, 0.000,// N1
    0.000, 0.000,-1.000,// N2//(a)
   _________, _________, _________//N3(n3x,n3y,n3z)
  };
  int K[ anzP*anzA ] = { // Kantentabelle
      0, 3, 2,    // für die 0.Flaeche
      0, 1, 3,    // für die 1.Flaeche
      0, 2, 1,    // für die 2.Flaeche
      1, 2, 3     // für die 3.Flaeche
  };
  for ( int i = 0; i _________________; i ++ ) {//(b)
    glBegin( GL_LINE_STRIP ); //(d)...
    glNormal3d( N[3*i+0], N[3*i+1], N[3*i+2] );
    for ( int j = 0; j < anzP; j ++ ) {
      int j0 = 3*K[ ____________________ ]; //(c)
      glVertex3d( P[j0+0], P[j0+1], P[j0+2] );
    }
    glEnd();
  }
}//«
```

Lösung:

Die normierte Flächennormale ist

$$N_3 = |(P_2 - P_1) \times (P_3 - P_1)| = \left| \begin{pmatrix} -1 \\ 1 \\ 0 \end{pmatrix} \times \begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix} \right| = \left| \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \right| = \frac{\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}}{\sqrt{3}} = \begin{pmatrix} \frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{3}} \end{pmatrix}$$

```
//(a) Die normierte Flächen-Normale N3(n3x,n3y,n3z) ist:
//
//   n3x = 1/sqrt(3), n3y = 1/sqrt(3), n3z = 1/sqrt(3)
//
#define anzA  4 // Anzahl von Flaechen
#define anzP  3 // Anzahl Pkt je Flaeche
void drawPyramide(int art) {
  double P[ 3*anzA ] = { // Punktetabelle
     0.0, 0.0, 0.0, // P0(x0,y0,z0)
     1.0, 0.0, 0.0, // P1(x1,y1,z1)
     0.0, 1.0, 0.0, // P2(x2,y2,z2)
     0.0, 0.0, 1.0  // P3(x3,y3,z3)
  };
  double N[ 3*anzA ] = {  // Normalentabelle
```

```
   -1.000, 0.000, 0.000,// N0
    0.000,-1.000, 0.000,// N1
    0.000, 0.000,-1.000,// N2
    //(a)
    0.577, 0.577, 0.577 //N3(n3x,n3y,n3z)
  };
  int K[ anzP*anzA ] = { // Kantentabelle
      0, 3, 2,    // für die 0.Flaeche
      0, 1, 3,    // für die 1.Flaeche
      0, 2, 1,    // für die 2.Flaeche
      1, 2, 3     // für die 3.Flaeche
  };
  for ( int i = 0; i<anzA; i ++ ) {//(b)
    //glBegin(GL_LINE_STRIP);
    //(d)...
    //glBegin(GL_POLYGON);
      if (art==1)
        glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
      else
        glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
      glNormal3d( N[3*i+0], N[3*i+1], N[3*i+2] );
      for ( int j = 0; j < anzP; j ++ ) {
        int j0 = 3*K[ i*3+j ]; //(c)
        glVertex3d( P[j0+0], P[j0+1], P[j0+2] );
      }
    glEnd();
  }
}
```

16.5.2 Teil 2

»Es darf nun vorausgesetzt werden, dass `main()` existiert, die CALLBACK-Funktion `ChangeSize()` für die Perspektive existiert und das Licht initialisiert ist.

- Bei (a): Es ist ein geeignetes 'push-pop' einzufügen.
- Bei (b): Es existieren die Funktionen `drawAchsen()` und `drawPyramide(2)`. Diese Figuren sollen gedreht gezeichnet werden, etwa so, wie in der obigen Skizze (d.h. etwa Militärperspektive).
- Bei (c): Die Achsen soll schwarz und die Pyramide magenta gezeichnet werden.

```
void CALLBACK RenderScene(void){
  glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  ////////////////////////////////////
  ____________________________________//(a)
  ____________________ (+45,  1,0,0); //(b)
  ____________________________________//(b)
  glColor3d  ( 0.0, 0.0, 0.0);
  drawAchsen();
  glColor3d  ( ____, ____, ____ );    //(c)
  drawPyramide(2);
  ____________________________________//(a)
  ////////////////////////////////////
  auxSwapBuffers();
}//«
```

Lösung:

```
void CALLBACK RenderScene(void){
  glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  ////////////////////////////////////
  glPushMatrix(); //(a)
  glMatrixMode(GL_MODELVIEW);
  //____________________ (+45,  1,0,0); //(b)
  glRotatef(135.0, 0.0, 1.0, 0.0);      //(b)
  glRotatef(45.0, 1.0, 0.0, -1.0);      //(b)
  glColor3d  ( 0.0, 0.0, 0.0);
  drawAchsen();
  glColor3d  ( 1.0, 0.0, 1.0);    //(c)
  drawPyramide(2);
  glPopMatrix(); //(a)
  ////////////////////////////////////
  auxSwapBuffers();
}
```

Literatur

[1] Prof. Dr. Walter Bachmann: »GDV-Script V1.0«. Erhältlich als einzelne HTML-Dateien unter http://homepages.fh-giessen.de/~hg54 oder als PDF-Datei unter http://homepages.fh-giessen.de/~hg54/gdv_script.pdf. Letztere enthält auch die Praktikumsaufgaben 1-4, aber nicht die Wiederholungsfragen. Das Skript enthält den gesamten Vorlesungsstoff und wird in der Vorlesung als Präsentation vorgeführt. Es ist vollständig und wird nur noch leicht verbessert. Die Folien der Vorlesung sind nur eine ggf. veraltete Kurzfassung des Skriptes. Tipp: den OpenGL-Stoff lernt man am besten nicht aus diesem Skript, sondern in der Programmierpraxis; anhand des Skriptes sollte man den theoretischen Hintergrund verstanden haben.

[2] Prof. Dr. Walter Bachmann: GDV-Praktikum http://homepages.fh-giessen.de/~hg54/index_gdv_praktikum.htm. 7 Praktikumsaufgaben in HTML-Format. Die Aufgaben 1-4 sind auch in [1] in PDF-Format enthalten. Die Praktikumsaufgaben werden im Semester fortschreitend freigeschaltet; sie bestehend aus zu erweiternden Quellcode-Grundgerüsten; es gibt jeweils Versionen für Windows und Linux.

[3] Prof. Dr. Walter Bachmann: Fragen zu den Praktikumsaufgaben. Erhältlich als einzelne HTML-Dateien unter http://homepages.fh-giessen.de/~hg54/index_gdv_praktikum.htm oder als PDF-Datei unter http://homepages.fh-giessen.de/~hg54/gdv_fragen.pdf. Letztere enthält jedoch nur die Fragen zu den Praktikumsaufgaben 1-4; die hier schon angekreuzten Antworten haben nichts mit der richtigen Lösung zu tun.

[4] Prof. Dr. Walter Bachmann: OpenGL-Referenz http://homepages.fh-giessen.de/~hg54/gdv_gl_ref.pdf.

[5] Prof. Dr. Walter Bachmann: http://homepages.fh-giessen.de/~hg54/gdv_gl_tutorial.chm.

[6] Prof. Dr. Walter Bachmann: GDV-Testklausur, Version vom SS 2002 http://homepages.fh-giessen.de/~hg54/gdv_2003/gdv_prakt/test_klausur_2002/index.htm. Nur im Internet auszufüllen. Mit Lösungen im vorliegenden Dokument enthalten.

[7] OpenGL API Referenz http://www.cevis.uni-bremen.de/~uwe/opengl/opengl.html

[8] Daniel und Tobias Webelsiep: Übungen in Computergrafik bei Prof. Dr. Bachmann im Sommersemester 2002. Quelle: http://www.webelsiep.de/downloads/quellcode/gdv.zip, referenziert unter http://www.webelsiep.de/default.php?main=studium im Bereich Quellcodes.

[9] Tim Pommerening: »Formelsammlung: GDV v0.0.1 (ohne Gewähr)«. Leider nicht mehr zu finden auf Tims Homepage http://homepages.fh-giessen.de/~hg12061/, daher zeitweise zur Verfügung gestellt unter http://matthias.ansorgs.de/InformatikDiplom/Modul.CompGra/Formeln.Pommerening.pdf.

[10] Matthias Ansorg: »Vorlesungsmodul Mathematik 1«; studentische Mitschrift zur Vorlesung »Mathematik 1« bei Prof. Eichner an der FH Gießen-Friedber, Studiengang Informatik. Version vom 27.5.2003. http://matthias.ansorgs.de/InformatikDiplom/Modul.Mathe1.Eichner/Mathe1.pdf

[11] SGI: »OpenGL on Silicon Graphics Systems« http://www.parallab.uib.no/SGI_bookshelves/SGI_Developer/books/OpenGLonSGI/sgi_html/index.html. Ein nützliches Buch, um OpenGL unter dem X Window System zu verwenden. Referenziert auf SGI-Bookshelves http://www.parallab.uib.no/SGI_bookshelves/SGI_Developer/index.html.

[12] SGI: »OpenGL Programming Guide« http://www.parallab.uib.no/SGI_bookshelves/SGI_Developer/books/OpenGL_PG/sgi_html/index.html. Referenziert auf SGI-Bookshelves http://www.parallab.uib.no/SGI_bookshelves/SGI_Developer/index.html.

[13] SGI: »OpenGL Reference Manual (Addison-Wesley Publishing Company)« http://www.parallab.uib.no/SGI_bookshelves/SGI_Developer/books/OpenGL_RM/sgi_html/index.html. Referenziert auf SGI-Bookshelves http://www.parallab.uib.no/SGI_bookshelves/SGI_Developer/index.html.