Technische Hochschule Mittelhessen

Prof. Dr.-Ing. A. Christidis

**Fachbereich Mathematik,
Naturwissenschaften und Informatik**

8. August 2014

Klausur
Computergrafik
für Bachelor-Studierende
SS 2014

– Lösungshilfe –

Personalien:

Name, Vorname: .....

Matrikelnummer: .....

Hinweise:

- **Die Bearbeitungszeit beträgt 90 Minuten.**
- **Alle schriftlichen Hilfsmittel sind zugelassen; andere Hilfsmittel, insb. elektr. Rechen- und Kommunikationsapparate, dürfen nicht verwendet werden.**
- **Ausgesprochene Folgefehler (durch Übertragung falscher Zwischenergebnisse) werden in Folgerechnungen als richtig gewertet.**
- **Die Aufgaben sollen nur auf diesen Blättern (inkl. Rückseite) bearbeitet werden. Bei Bedarf wird zusätzliches Papier zur Verfügung gestellt.**
- **Zur sicheren Zuordnung aller Lösungen wird um eine persönliche Kennung (Name u./o. Matrikelnr.) auf allen Blättern gebeten.**
- **Auf Wunsch darf auch Bleistift verwendet werden.**

Zur leichteren Lesbarkeit werden Substantive nur in einem Geschlecht („Nutzerin“) verwendet.

1. Aufgabe (10 Punkte)

a) Sie lernen jemanden kennen, der als Informatiker Software für das Fernsehen entwickelt. Seine jüngste Arbeit diente zum Wechsel zwischen zwei Sendungen: Das letzte Bild der vorausgehenden Sendung wird eingefroren (Abb. 1.1). und dann „weggekippt“, als wäre es eine Postkarte (Abb. 1.2), bevor das Anfangsbild der nächsten Sendung kommt.

Abb. 1.1

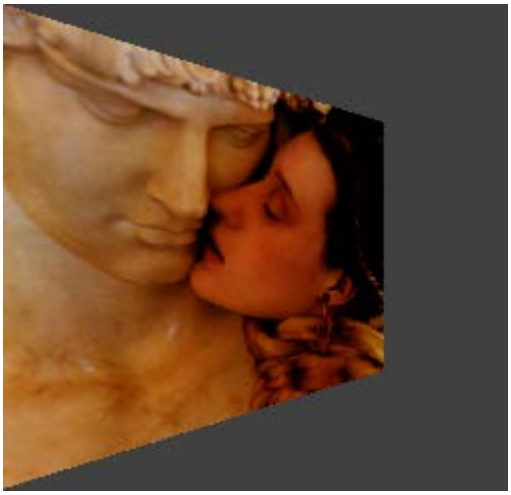


Abb. 1.2

Fällt diese Software-Entwicklung in den Bereich der Bildverarbeitung, der Computergrafik oder in keinen der beiden?

Bitte begründen kurz Sie Ihre Antwort!

Diese Arbeit fällt in den Bereich der Computergrafik: Sie erzeugt visuelle Darstellungen unter Verwendung logisch-mathematischer Methoden.

b) Sie arbeiten an Grafiken, die auf dem Display eines Kopiergeräts erscheinen sollen. Der Bildschirm soll ein Seitenverhältnis von 4:3 (Breite zu Höhe) haben und über 360 (Pixel-)Zeilen verfügen.

Wie viele Pixel wird die Bildschirmdiagonale enthalten, wenn sie mit dem Bresenham-Algorithmus gezogen wird?

Bitte erläutern Sie, mit welchen Berechnungen und Überlegungen Sie zu Ihrem Ergebnis gelangten!

Breite : Höhe = Breite : 360 = 4 : 3

⇒ Breite = 4 x 360 / 3 = 480

Bresenham zieht immer die Diagonale eines (gedachten) Rechtecks, die Anzahl der Pixel ist gleich der längeren seiner Seiten; d.h. hier:

Die Diagonale ist 480 Pixel lang.

2. Aufgabe (45 Punkte)

Sie bereiten einen Animationsfilm vor, mit dem die Funktion einer Abrißbirne (Abb. 2.1) erklärt wird. Dazu erzeugen Sie ein einfaches Modell, dessen Kran am Koordinatenursprung steht.

Abb. 2.1

Die Kranspitze liegt in der Höhe *H*, die Stahlkugel in der Höhe *h* (und hängt an einer Kette der Länge *H* – *h*) über der Erde (x-z-Ebene). Der Kran ist so geneigt, daß der Abstand der Kugel von der y-z-Ebene die Länge *l* hat (s. Seitenansicht Abb. 2.2a, Draufsicht Abb. 2.2b).

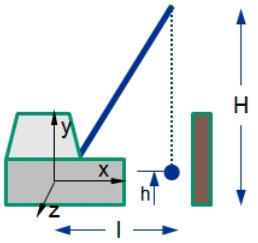


Abb. 2.2a

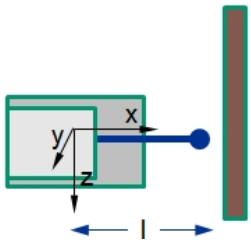


Abb. 2.2b

Sie wollen im Zeichentrick den Abbruch einer niedrigen, breiten Mauer darstellen. Dazu dreht sich der Kran vorsichtig um 90° um die y-Achse, bis der Kranführer mit dem Rücken zum Betrachter sitzt, dann setzt er leicht zurück (wobei er seine Entfernung von der Mauer erhöht), um dann ruckartig an die alte Position zurückzukehren, so daß die Stahlkugel durch ihre Trägheit gegen die Mauer schwingt. (Abb. 2.3a, Abb. 2.3b). Der Kranführer versteht es, der Stahlkugel einen solchen Schwung zu geben, daß sie relativ genau um den Winkel θ=45° pendelt (sin45° = cos45° = $\sqrt{2}$ / 2).

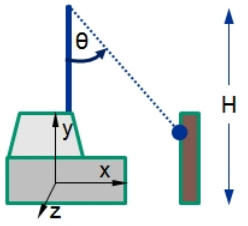


Abb. 2.3a

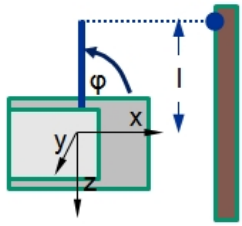


Abb. 2.3b

Vor Codierung dieser Darstellung berechnen Sie die Koordinaten des Kugel-Schwerpunkts beim weitesten Ausschlag in vier Schritten.

Sie drehen den Kran um den rechten Winkel φ um die y-Achse, bis er über der negativen z-Achse steht (Transformationsmatrix $\underline{\mathbf{T}}_{(i)}$). Anschließend verschieben Sie seine Spitze in die x-z-Ebene (Transformationsmatrix $\underline{\mathbf{T}}_{(ii)}$), um dann die Stahlkugel um θ um die z-Achse zu drehen (Transformationsmatrix $\underline{\mathbf{T}}_{(iii)}$). Abschließend verschieben Sie den Kran so, daß seine Spitze wieder die Höhe *H* erlangt (Transformationsmatrix $\underline{\mathbf{T}}_{(iv)}$).

Beantworten Sie bitte folgende Fragen:

a) Geben Sie bitte (in Grad, unter Berücksichtigung des Drehsinns) den rechten Winkel **φ** an, um welchen der Kran anfänglich gedreht wird, damit er über der negativen z-Achse steht. Wie groß sind dann **sin φ** und **cos φ**?

φ = ***+ 90°*** **sin φ =** ***1*** **cos φ =** ***0***

b) Wie lautet die Transformationsmatrix $\underline{\mathbf{T}}_{(i)}$, die den Kran um **φ** bis zur z-Achse dreht? Geben Sie sie bitte sowohl in symbolischer (sin φ, ...) als auch in arithmetischer (zahlenmäßiger) Form an!

$$\underline{\mathbf{T}}_{(i)} = \begin{pmatrix} \cos\varphi & 0 & \sin\varphi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\varphi & 0 & \cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

c) Wie lautet die Transformationsmatrix $\underline{\mathbf{T}}_{(ii)}$, mit der die Kranspitze senkrecht auf den Boden verschoben wird? Geben Sie sie bitte direkt mit den verfügbaren Größen an!

$$\underline{\mathbf{T}}_{(ii)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -H \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

d) Geben Sie nun bitte (in Grad, unter Berücksichtigung des Drehsinns) den Winkel **θ** an (|θ|=45°), um welchen die Kette mit der Abrißbirne um die z-Achse gedreht wird. Wie groß sind dann **sin θ** und **cos θ**?

θ = + 45° **sin θ** = $\sqrt{2}/2 \approx 0{,}7$ **cos θ** = $\sqrt{2}/2 \approx 0{,}7$

e) Welche Transformationsmatrix $\underline{\mathbf{T}}_{(iii)}$ dreht, wie oben beschrieben, die Kette mit der Abrißbirne um die z-Achse um den Winkel **θ**? Geben Sie sie bitte sowohl in symbolischer als auch in arithmetischer Form an!

$$\underline{\mathbf{T}}_{(iii)} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1/2^{½} & -1/2^{½} & 0 & 0 \\ 1/2^{½} & 1/2^{½} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

f) Mit welcher Transformationsmatrix $\underline{\mathbf{T}}_{(iv)}$ wird schließlich die Kranspitze zurück auf die Höhe *H* verschoben? Geben Sie sie bitte direkt mit den verfügbaren Größen an!

$$\underline{\mathbf{T}}_{(iv)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & H \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

g) Wie berechnet sich nun die Transformationsmatrix $\underline{\mathbf{T}}$, mit welcher alle vorausgegangenen Transformationen zu einer Operation vereinigt werden?

$$\underline{\mathbf{T}} = \underline{T}_{(iv)} \bullet \underline{T}_{(iii)} \bullet \underline{T}_{(ii)} \bullet \underline{T}_{(i)}$$

h) Berechnen Sie nun bitte die Transformationsmatrix $\underline{\mathbf{T}}$ anhand der bisher gemachten Angaben. (Symbole, auch Wurzelzeichen, können unverändert in die Rechnung eingehen.)

$\underline{\mathbf{T}} =$

$\underline{T}_{(i)}$

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

$\underline{T}_{(ii)}$

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -H \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \Rightarrow \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & -H \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$\underline{T}_{(iii)}$

$$\begin{pmatrix} 1/2^{½} & -1/2^{½} & 0 & 0 \\ 1/2^{½} & 1/2^{½} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \Rightarrow \begin{pmatrix} 0 & -1/2^{½} & 1/2^{½} & H/2^{½} \\ 0 & 1/2^{½} & 1/2^{½} & -H/2^{½} \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$\underline{T}_{(iv)}$

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & H \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \Rightarrow \begin{pmatrix} 0 & -1/2^{½} & 1/2^{½} & H/2^{½} \\ 0 & 1/2^{½} & 1/2^{½} & H-H/2^{½} \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$\underline{T}$

i) Berechnen Sie bitte nun die Koordinaten der Kugel $[X, Y, Z]^T$ bei ihrem Endausschlag nach der obigen Beschreibung mit Hilfe der Transformationsmatrix **T**.

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = \begin{pmatrix} 0 & -1/2^{½} & 1/2^{½} & H/2^{½} \\ 0 & 1/2^{½} & 1/2^{½} & H-H/2^{½} \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} H/2^{½} - h/2^{½} \\ h/2^{½} + H - H/2^{½} \\ -l \\ 1 \end{pmatrix} \qquad \begin{pmatrix} l \\ h \\ 0 \\ 1 \end{pmatrix}$$

j) Die bisher erfolgte Vorausberechnung der o.a. Animation soll ermitteln helfen, welche Dimensionierung der beschriebenen Szene geeignet erscheint, die Funktion einer Abrißbirne mit der Animation zu erklären. Dazu soll die Beantwortung der folgenden Fragen dienen.

Bitte ergänzen Sie die u.a. Aussagen unter Verwendung der Größen *h*, *H*, *l* und X, Y, Z; die Vorzeichen brauchen nicht beachtet zu werden (absolute Beträge reichen).

l	ist der minimale Abstand zwischen Kran und Mauer vor dem Abriß, der noch Kollisionsfreiheit bis zum Beginn des Abbruchs garantiert.
X	ist der maximale Abstand zwischen Kran und Mauer, der noch einen Abriß der Mauer ermöglicht (unter Annahme des o.a. Endausschlags).
Y	ist die maximale Höhe, die eine Mauer haben darf, wenn sie bei der gegebenen Aufstellung (fast) unbeschadet bleiben soll.
Z (l)	ist ein Maß für die minimale Breite der Mauer, wenn die Stahlkugel nicht an ihr vorbeipendeln soll.

3. Aufgabe (45 Punkte + 5 Sonderpunkte)

Ein guter Bekannter bittet Sie verzweifelt um Hilfe: Im Zuge neuer, mutiger Schulreformen muß er als Geographie-Lehrer nach den Sommerferien Computergrafik unterrichten.

Um ihm zu helfen, schreiben Sie das OpenGL-Programm `tripod.c` (s. Ende dieser Aufgabe), das u.a. einen Globus darstellt. Unten links im Fenster fügen Sie auch ein Dreibein hinzu, um die Richtung der Hauptachsen anzuzeigen (Abb. 3.1). Dann wollen Sie mit ihm den Source-Code durchgehen.

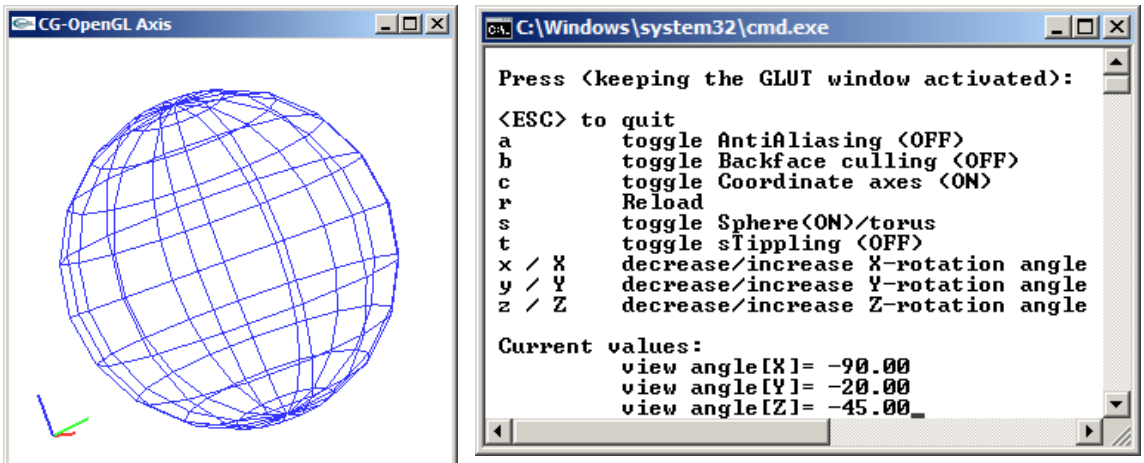


Abb. 3.1 Programm `tripod`

Beantworten Sie bitte folgende Fragen zum Code von `tripod.c`:

a) Noch bevor Sie mit dem Programm fertig sind, läßt sich Ihr Bekannter das `main()` erklären. Bei der Feststellung, daß dort keine explizite Fenstergröße angegeben wird, fragt er Sie ängstlich, ob sein alter Rechner mit einem VGA-Standard von 640×480 Pixeln das eingerichtete Fenster darstellen kann.

Was antworten Sie ihm, und wie begründen Sie das?

Das Fenster wird nicht so groß: Die Abwesenheit einer Deklaration führt zur Einrichtung des Fensters in der Voreinstellung von nur 300×300 Pixeln.

b) Die noch im `main()` aufgerufene Funktion `init()` enthält die Anweisung:

```
glClearColor (1., 1., 1., 1.);
```

Was bewirkt sie?

Würde das etwas ändern, wenn sie fehlen würde?

Sie legt die Löschfarbe auf Weiß / opak fest.

Würde sie fehlen, so wäre die Löschfarbe Schwarz / opak.

c) Sie werden gefragt, ob in der Funktion `init()` der Aufruf

```
key(' ', 0, 0);
```

ein Callback (-Aufruf) ist. Was antworten Sie?

Wenn ja: Wie häufig und zu welchen Anlässen erfolgt dieser Callback? (Ungefähre Angaben genügen.)

Wenn nein: Ist `key()` keine Callback-Funktion? (Kurze Erläuterung)

Nein. Die Funktion `key()` ist zwar eine der verwendeten Callback-Funktionen. Der o.a. Aufruf ist aber kein Callback, weil er nicht durch GLUT erfolgt.

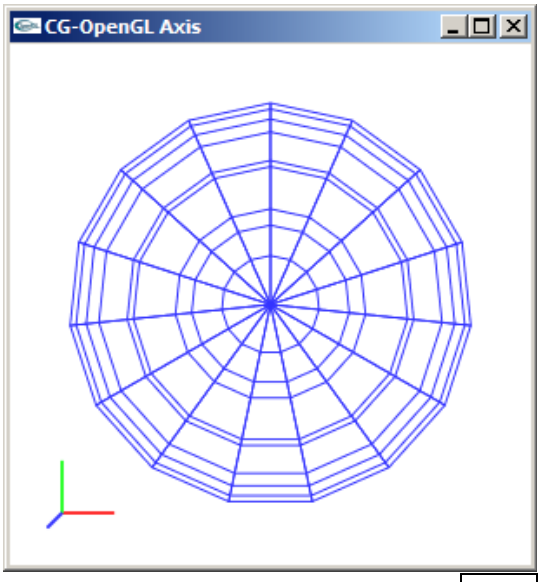
d) Die Funktion `draw()` enthält gleich zweimal im Abstand von wenigen Zeilen einen Aufruf von `glLoadIdentity()`, was eine Art Neubeginn bedeutet.

Erklären Sie bitte stichwortartig, was da jeweils initialisiert wird:

Der erste Aufruf initialisiert das Sichtvolumen, bevor es mit `glFrustum()` gesetzt wird; der zweite initialisiert die Positionierung (Animation) der Modelle, die (u.a.) mit `move()` aktualisiert werden.

e) In `draw()` wird, nach Fensterlöschung und Festlegung von Sicht und Animation, durch Aufruf von `model()` die Zeichnung eines GLUT-Modells veranlaßt. Über den Menüpunkt 'a' (globale Variable `aa`) kann in `model()` Antialiasing ein- oder ausgeschaltet werden.

Kreuzen Sie bitte unten an, in welchem der beiden Bilder Antialiasing angewandt wurde:



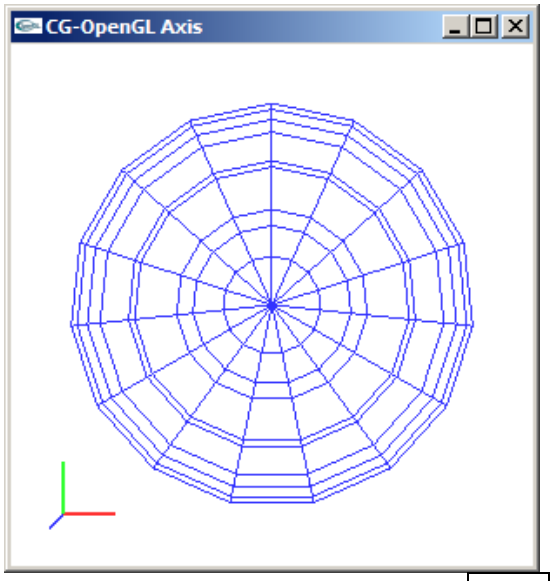


Abb. 3.2 Antialiasing

X	

Woran erkennen Sie, welches Bild mit Antialiasing behandelt wurde? (1-2 Stichworte genügen.)

An den geglätteten Linien: Die Stufen sind nicht so klar sichtbar.

f) In `model()` besteht die Wahl zwischen zwei GLUT-Drahtmodellen: einer Kugel und einem Ring (Menüpunkt 's', globale Variable `sphere`). Weiterhin kann man Backface-Culling wählen (Menüpunkt 'b', globale Variable `backF`).

Das Einschalten von Backface-Culling ändert nichts an der Darstellung der Kugel (Abb. 3.2). Wählt man jedoch den Ring, so ist der Unterschied zwischen den beiden Einstellungen unverkennbar (Abb. 3.3).

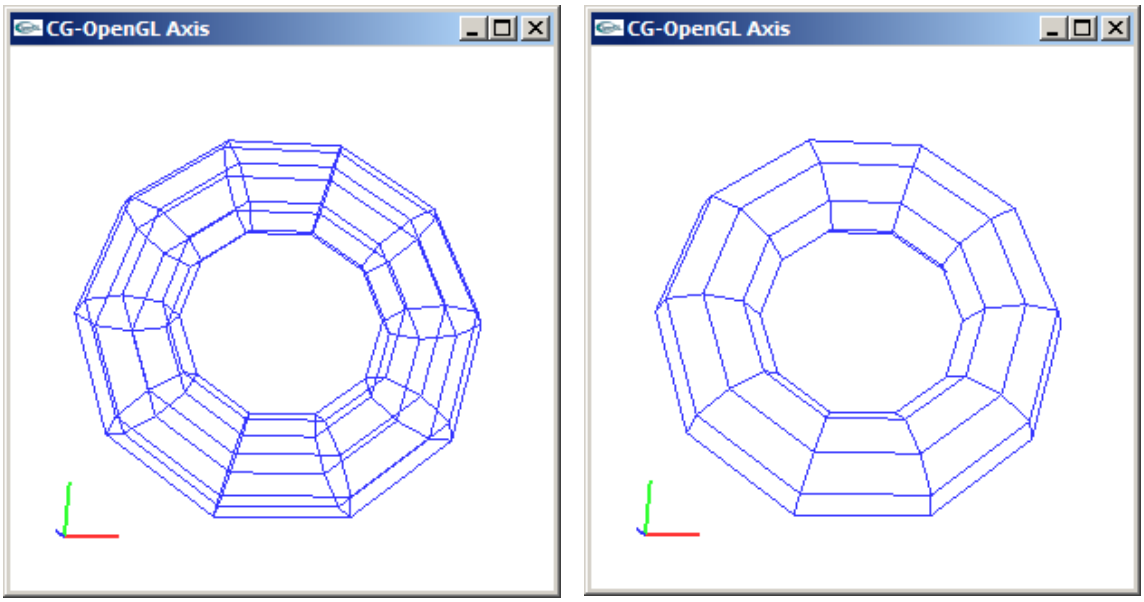


Abb. 3.3 Ring ohne (li.) und mit (re.) Backface-Culling

Da Sie fest damit rechnen, daß Ihr Bekannter Sie danach fragt, wollen Sie ihm die passende(-n) unter den folgenden Erklärungen über den Unterschied zwischen beiden Modellen geben (bitte ankreuzen):

	Ein Teil des Ringes wird in der Hintergrundfarbe gezeichnet.
X	Der Ring ist definiert als Anordnung nicht nur von Linien, sondern auch von Flächen, die zwischen den Linien aufgespannt sind.
	OpenGL kann Linien auch selektiv löschen, ohne Kreuzungspunkte zu beeinträchtigen. Der Ring ist dafür vorbereitet, die Kugel aber nicht.
	Die Kugel-Oberfläche ist in der Voreinstellung (default) als transparent angegeben.
	Vor der Zeichnung wird das Volumen des Rings ermittelt und die Figur virtuell mit einem Stoff in der Hintergrundfarbe ausgefüllt, womit die hinteren Objektkanten verdeckt werden.

g) Noch vor Überlassung Ihres Programms bereiten Sie sich auf weitere Fragen des unkundigen Geographie-Lehrers vor. Dabei fallen Ihnen bei eingeschaltetem Backface-Culling Stellen wie die markierte auf (Abb. 3.4).

Auf entsprechende Frage Ihres Bekannten wollen Sie folgende Erklärung(-en) abgeben (bitte ankreuzen):

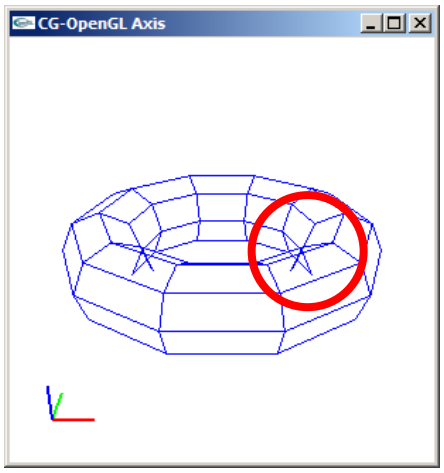


Abb. 3.4

	Das ist die Folge der transparenten Färbung der Modellflächen.
X	Hier verdecken sich keine abgewandten Objektflächen.
	Es handelt sich um einen GLUT-Bug, der nicht mehr behoben wird, weil GLUT nicht mehr aktualisiert wird.
	Es ist normal und gewollt, daß an einem Drahtmodell sämtliche Kanten sichtbar sind.

h) Über den Menüpunkt 'c' kann in `draw()`, abhängig vom Wert der globalen Variablen `tripod`, nach dem Drahtmodell ein Dreibein gezeichnet werden. Die Zeichnung wird in der Funktion `drawAxes()` ausgeführt. Bei einer Vergrößerung der Achsendarstellung (Abb. 3.5) entdecken Sie eine Pixel-Belegung, die Sie nach dem Studium des Bresenham-Algorithmus so nicht erwartet hätten.

Erklären Sie bitte in Stichworten,

- welche Abweichung vom Bresenham-Algorithmus zu beobachten ist, und
- mit welcher Code-Zeile von `drawAxes()` dies zu begründen ist.

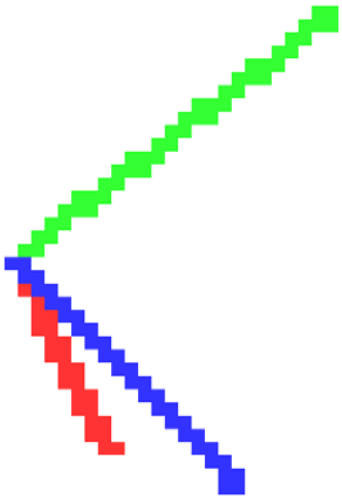


Abb. 3.5

- ***Die Linien sehen nicht danach aus, daß Nord-/Nordost-Entscheidungen getroffen worden wären.***
- ***Der Grund ist, daß die Linie zwei-Pixel-breit ist: `glLineWidth(2.f);`***

i) In der Funktion `drawAxes()` wird die Funktion `move()` aufgerufen; deren Quellencode enthält eine Translation und drei Rotationen um die Hauptachsen x, y, z. Der Aufruf von `move()` in `drawAxes()` erfolgt wiederum nach dem Aufruf einer Translation und vor jenem einer Skalierung.

Sie wollen Ihrem Bekannten erklären, wie sich diese Befehle auswirken, mit einer Zigarettenschachtel, die Sie vor ihm bezüglich gedachter Raumachsen drehen, verschieben und zerdrücken (als Skalierung).

Geben Sie bitte unten mit den Zahlen 1-6 an, in welcher Reihenfolge Sie die Aktionen mit der Zigarettenschachtel vornehmen, um das erwartete Ergebnis der Animation zu erörtern.

5	Translation parallel zur z-Achse in `move()`
4	Rotation um die x-Achse in `move()`
3	Rotation um die y-Achse in `move()`
2	Rotation um die z-Achse in `move()`
6	Translationen parallel zur x- und zur y-Achse in `drawAxes()`
1	Skalierung entlang der x-, der y- und der z-Achse in `drawAxes()`

j) In `drawAxes()` wird auch das Achsen-Dreibein gezeichnet; zur leichteren Identifizierung werden die Achsen x, y und z (Indizes 0, 1 und 2 entsprechend) in unterschiedlichen Farben gezeichnet und (z.B. für monochrome Ausgabe) individuell gemustert (Menüpunkt 't', globale Variable `stipple`).

Beantworten Sie bitte in diesem Kontext folgende Fragen:

Schildern Sie bitte in Stichworten, wie die Muster der drei Achsen entstehen (z.B.: 1x setzen, 2x auslassen o.ä).

	Musterung
x-Achse	***1x setzen, 1x auslassen***
y-Achse	***2x setzen, 2x auslassen***
z-Achse	***3x setzen, 3x auslassen***

k) Die Stippling-Anweisung liegt erwartungsgemäß innerhalb der Schleife für die Dreibein-Zeichnung – aber außerhalb des Anweisungspaares `glBegin()` / `glEnd()`. Ein Versuch, sie zwischen die beiden Befehle zu setzen, scheitert: Die Musterung unterbleibt.

Welchen Hinweis aus der OpenGL-Nutzung sollte man hierzu beachten?

Daß manche Befehle zwischen `glBegin()` und `glEnd()` ignoriert werden; `glLineStipple()` ist einer von ihnen.

l) Eine Detail-Vergrößerung (Abb. 3.6) zeigt:

Bei dieser Musterung wird offenbar der Punkt am Koordinaten-Ursprung gesetzt.

Wie müßte die entsprechende OpenGL-Anweisung lauten, um die exakt umgekehrte Reihenfolge für Pixel-Setzen und -Auslassen zu erzeugen und am Ursprung eine Lücke entstehen zu lassen?

(Nennung ohne Begründung genügt.)

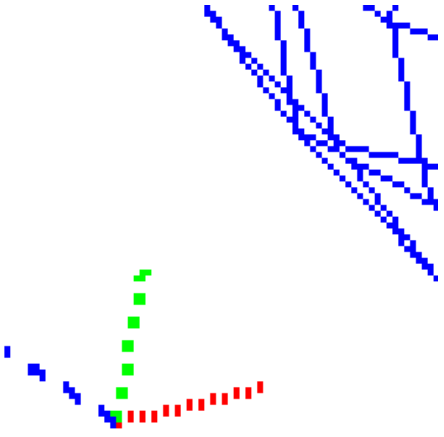


Abb. 3.6

glLineStipple((j1+1), 0xAAAA);

m) Nach ein paar letzten kleinen, unwichtigen Änderungen, die Sie aus Zeitgründen nicht mehr testen, präsentieren Sie `tripod` stolz Ihrem Bekannten (Quellcode s.u.). Dieser startet das Programm – und sieht nichts, außer einem leeren Fenster. Sie verbergen geschickt Ihre Panik und versuchen, seine Features nacheinander anzuwählen – ohne Ergebnis.

Nur die Wahl des Menüpunkts 'c' ändert etwas dahingehend, daß im sonst leeren Fenster das Koordinaten-Dreibein erscheint (s. Abb. 3.7) Darauf erkennen Sie sofort die Ursache für den Fehlschlag.

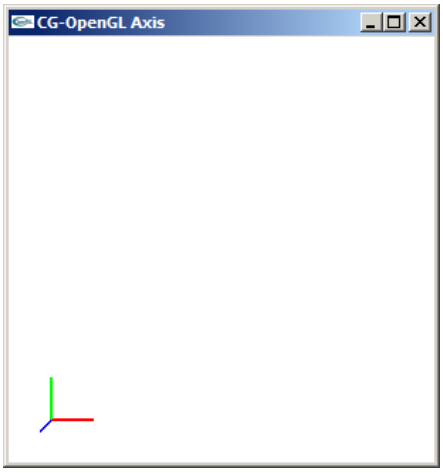


Abb. 3.7

Souverän drücken Sie auf eine beliebige Taste, und ab da läuft das Programm wie gewünscht (Abb. 3.2). Dem staunenden Lehrer liefern Sie dann eine lange Erklärung, die damit beginnt und endet, daß OpenGL ein Zustandsautomat ist.

Schildern Sie bitte die Hauptpunkte Ihrer Erzählung durch Beantwortung folgender Fragen:

Mit welcher Zeichenfarbe werden die drei Hauptachsen des Dreibeins gezeichnet? (Bitte ausfüllen)

	Farbe
x-Achse	***Rot***
y-Achse	***Grün***
z-Achse	***Blau***

Welche Zeichenfarbe war beim Start des Programms eingestellt?

In welcher Farbe erscheint schließlich die GLUT-Figur (i.d.R.: die Kugel)?

Erste Einstellung beim Start des Programms war Weiß, die Figur erscheint in Blau.

(5 Sonderpunkte)

Welche Zeichnungen und/oder Löschungen in welchen Farben führten zu dem Bild nach Abb. 3.7? Welche Rolle spielte dabei die Implementierung von OpenGL als Zustandsautomat?

Was löste der Tastendruck mit der beliebigen Taste aus, und was hat die Implementierung als Zustandsautomat damit zu tun, daß nach dem Tastendruck für die restliche Laufzeit das Programm „repariert“ war?

(Kurze Antwort genügt.)

Als Zustandsautomat hat OpenGL eine voreingestellte Zeichenfarbe (Weiß), solange nichts daran geändert wird.

Abb. 3.7 zeigt ein GLUT-Modell in weißer Farbe vor weißem Hintergrund.

Da das Zeichnen des Dreibeins (in R, G, B) als letzte Farbe Blau verwendet, ist dies die Farbe, die eingestellt und bis zur Beendigung des Programms nicht mehr geändert wird – weil OpenGL ein Zustandsautomat ist.

Das Drücken der beliebigen Taste hatte zur Folge, daß `key()` aufgerufen wurde, an dessen Ende `draw()` ausgeführt wird. Somit löste der Tastendruck das Neuzeichnen in blau aus.

```
/* tripod.c */
/*OpenGL-Darstellung von GLUT-Modellen mit Achsenkreuz*/
#include "tripod.h"

/*Globale Variablen: */
float    angle[3]={0.,0.,0.};
GLdouble nah=3*VIEW_X, fern=5*VIEW_X;
GLfloat  axis[4][3]={{1., 0., 0.}, {0., 1., 0.}, {0., 0., 1.}, {0., 0., 0.}};
float    axcol[3][3]={{1.f,.0f,.0f}, {.0f,1.f,.0f}, {.0f,.0f,1.f}};
int      backF=0, aa=0, tripod=0, sphere=1, stipple=0;

/*************************************************************************/
   void model (void)
/*************************************************************************/
/*Modell zeichnen:*/
{ /*Strichstaerke:*/
  glLineWidth(1.f);

  /*Anti-Aliasing:*/
  if (aa)
  { glEnable (GL_LINE_SMOOTH);   glEnable (GL_BLEND);
    glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
  } else
  { glDisable (GL_LINE_SMOOTH); glDisable (GL_BLEND); }

  /*Backface-Culling:*/
  if (backF) glEnable  (GL_CULL_FACE);
  else       glDisable (GL_CULL_FACE);

  /*Kugel oder Ring:*/
  if (sphere) glutWireSphere(VIEW_X, 15, 10);
  else glutWireTorus(VIEW_X/4., 3*VIEW_X/4., 8, 10);
  return;
}
/*************************************************************************/
   void drawAxes (void)
/*************************************************************************/
/*Achsen zeichnen:*/
{ int j1=0;
  float dummmy=VIEW_X*(fern+nah)/(2.*nah);

  /*Strichstaerke:*/
  glLineWidth(2.f);

  /*Stippling:*/
  if (stipple) glEnable (GL_LINE_STIPPLE);
  else         glDisable(GL_LINE_STIPPLE);

  /*Position Dreibein:*/
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glTranslatef(-.8*dummmy, -.8*dummmy, 0.);
  move();
  glScalef (.2*dummmy, .2*dummmy, .2*dummmy);

  /*Achsen-Dreibein:*/
  for (j1=0; j1<3; j1++)
  { if (stipple) glLineStipple((j1+1), 0x5555);
    glBegin(GL_LINES);
      glColor3fv (axcol[j1]);
      glVertex3fv(axis[3]);
      glVertex3fv(axis[j1]);
    glEnd();
  }
 if (stipple) glDisable (GL_LINE_STIPPLE);
  return;
}
```

```
/*************************************************************************/
  void move (void)
/*************************************************************************/
/*Animation:*/
{ /*Alles ins Sichtvolumen verschieben:*/
  glTranslatef(0., 0., -(fern+nah)/2.);

  /*Positionierung:*/
  glRotatef(angle[X], 1., 0., 0.);
  glRotatef(angle[Y], 0., 1., 0.);
  glRotatef(angle[Z], 0., 0., 1.);
  return;
}

/*************************************************************************/
  void draw (void)
/*************************************************************************/
/*Grafik erstellen:*/
{ glClear(GL_COLOR_BUFFER_BIT);

  /*Sicht:*/
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  glFrustum (-VIEW_X, VIEW_X, -VIEW_Y, VIEW_Y, nah, fern);

  /*Zeichnung:*/
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();

  /*Animation, Positionierung:*/
  move();

  /*Modell / Darstellung:*/
  model();
  if (tripod) drawAxes();
  glFlush();
  return;
}

/*************************************************************************/
  void init (void)
/*************************************************************************/
/*Initialisierung*/
{ /*Loeschfarbe:*/
  glClearColor (1., 1., 1., 1.);
  /*Tastendruck simulieren, um Menue auszugeben:*/
  key(' ', 0, 0);
  return;
}

/*************************************************************************/
  void key (unsigned char key, int x, int y)
/*************************************************************************/
/*Menue und Eingabe-Behandlung:*/
{ switch (key)
  { case ESC: exit(0);
    case 'a': aa = 1 - aa;                                          break;
    case 'b': backF = 1-backF;                                      break;
    case 'c': tripod = 1 - tripod;                                  break;
    case 'r': angle[X]=angle[Y]=angle[Z]=0.;                        break;
    case 's': sphere = 1 - sphere;                                  break;
    case 't': stipple = 1 - stipple;                                break;
    case 'x': angle[X] -= 5.; if (angle[X] <=-360) angle[X]+=360; break;
    case 'X': angle[X] += 5.; if (angle[X] >= 360) angle[X]-=360; break;
    case 'y': angle[Y] -= 5.; if (angle[Y] <=-360) angle[Y]+=360; break;
    case 'Y': angle[Y] += 5.; if (angle[Y] >= 360) angle[Y]-=360; break;
    case 'z': angle[Z] -= 5.; if (angle[Z] <=-360) angle[Z]+=360; break;
    case 'Z': angle[Z] += 5.; if (angle[Z] >= 360) angle[Z]-=360; break;
  }
```

```
  system (CON_CLS);
  printf ("\n\r Press (keeping the GLUT window activated):");
  printf ("\n\n\r <ESC> to quit");
  printf ("\n\r a        toggle AntiAliasing ");
  if (aa) printf ("(ON)"); else printf ("(OFF)");
  printf ("\n\r b        toggle Backface culling ");
  if (backF) printf ("(ON)"); else printf ("(OFF)");
  printf ("\n\r c        toggle Coordinate axes ");
  if (tripod) printf ("(ON)"); else printf ("(OFF)");
  printf ("\n\r r        Reload");
  printf ("\n\r s        toggle Sphere"); if (sphere) printf ("(ON)");
  printf ("/torus"); if (!sphere) printf ("(ON)");
  printf ("\n\r t        toggle sTippling ");
  if (stipple) printf ("(ON)"); else printf ("(OFF)");
  printf ("\n\r x / X    decrease/increase X-rotation angle");
  printf ("\n\r y / Y    decrease/increase Y-rotation angle");
  printf ("\n\r z / Z    decrease/increase Z-rotation angle");
  printf ("\n\n\r Current values:");
  printf ("\n\r          view angle[X]=%7.2f", angle[X]);
  printf ("\n\r          view angle[Y]=%7.2f", angle[Y]);
  printf ("\n\r          view angle[Z]=%7.2f", angle[Z]);

  draw();
  return;
}

/*****************************************************************************/
  int main (int argc, char **argv)
/*****************************************************************************/
{ glutInit(&argc, argv);
  glutCreateWindow("CG-OpenGL Axis");
  glutDisplayFunc(draw);
  glutKeyboardFunc(key);
  init();
  glutMainLoop();
  return 0;
}
```

```
/* tripod.h */

#ifndef TRIPOD_H
  #define TRIPOD_H
  #include <stdio.h> //wg. printf()
  #include <GL/glut.h>

  #define CON_CLS  "cls"
  #define ESC       27
  #define VIEW_X    5
  #define VIEW_Y    5

  enum {X=0, Y=1, Z=2, W=3};

  /*Prototypen:*/
  void draw     (void);
  void drawAxes (void);
  void init     (void);
  void key      (unsigned char key, int x, int y);
  void model    (void);
  void move     (void);

#endif //TRIPOD_H
```

Platz für Notizen: