Technische Hochschule Mittelhessen

Prof. Dr.-Ing. A. Christidis

**Fachbereich Mathematik,
Naturwissenschaften und Informatik**

12. Juli 2011

Klausur
Computergrafik
für Bachelor-Studierende
SS 2011

– 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** (25 Punkte)

a) Sie lernen jemanden kennen, der seit Jahrzehnten als Entwickler auf dem Gebiet der Informationstechnologie tätig ist. Sie erfahren, daß er dabei sei, durch die Verwendung neuartiger Materialien Digitalkameras so stark zu verkleinern, daß man in die Hautporen von Lebewesen eindringen und Haarwurzeln fotografieren könnte. Erfolgreiche Anwendungen seiner Arbeit hätten schon dazu beigetragen, daß die Wurzeln von Elefantenhaaren fotografiert und als dreidimensionale Grafiken modelliert worden seien.

Handelt es sich bei diesem großen Fachmann um einen Experten für Bildbearbeitung, für Bildverarbeitung, für Computergrafik, für alle drei oder für keine von ihnen? Bitte kreuzen Sie die richtige(n) Antwort(en) an!

	Experte für Bildbearbeitung
	Experte für Bildverarbeitung
	Experte für Computergrafik
X	Experte für keines der vorgenannten Gebiete

b) Welches der folgenden Kriterien war für die korrekte Beantwortung der Frage a) maßgeblich? Bitte kreuzen Sie die richtige Antwort an!

	Es sollen Untersuchungen an Lebewesen vorgenommen werden.
X	Es geht um den Einsatz neuer Materialien.
	Es wird Software entwickelt, die Dateien (z.B. Bilddateien) bearbeitet.
	Die neu entwickelte, lichtempfindliche Hardware wird speziell für Bildaufnahmen eingesetzt.
	Ziel ist die synthetische Erzeugung eines photorealistischen Grafik-Modells.
	Es handelt sich um ein innovatives Projekt.
	Es geht um die Behandlung natürlicher / reeller Bilder.
	Es entstehen grundlegende Funktionalitäten, die auch in biometrischen Verfahren verwendet werden können.

c) Ein Hersteller von Haushaltsgeräten beauftragt Ihre Arbeitsgruppe mit der Software-Entwicklung für Liniengrafiken auf den Displays seiner Apparate. Das Projekt muß mit möglichst wenig, möglichst schnellem Code auskommen. Aufgrund Ihrer hervorragenden Leistung in der Klausur Computergrafik bekommen Sie die Verantwortung für den sensiblen Bereich der Linien-Implementierung nach Bresenham.

Den ersten Oktanten wollen Sie direkt aus den Unterlagen dieser Vorlesung übernehmen; dann bekommen Sie vom Auftraggeber die Einschränkung, daß Ihr Programm, zumindest für die Versionen der ersten Jahre, ausschließlich solche Linien zeichnen soll, deren Steigung α immer ein Vielfaches von 45° beträgt ($\alpha = n \times \pi/4,\ n \in N,\ 0 \leq n < 8$).

(b.w.) ⇒

Welche Oktanten werden nun wirklich benötigt? Bitte kreuzen Sie die richtige(n) Antwort(en) an!

X	1. Oktant	X	5. Oktant
X	2. Oktant	X	6. Oktant
X	3. Oktant	X	7. Oktant
X	4. Oktant	X	8. Oktant

d) Im obigen Projekt soll ein Display mit 400 × 200 Pixeln (Breite × Höhe) verwendet werden. Die Pixel sind nicht quadratisch; sie haben die Maße 0,2 mm × 0,3 mm (Breite × Höhe).

Wie viele Pixel enthält die Display-Diagonale, wenn sie mit dem Linien-Algorithmus nach Bresenham gezogen wird? (Zahl genügt)

400

Wie viele cm beträgt die Display-Diagonale?

(Ergebnis nur mit Berechnung und stichwortartiger Begründung gültig!)

Display- Breite : 400 × 0,2 mm = 80 mm = 8 cm

Display- Höhe: 200 × 0,3 mm = 60 mm = 6 cm

Display-Diagonale: $(8^2 + 6^2)^{½}$ cm = $(64 + 36)^{½}$ cm = 10 cm

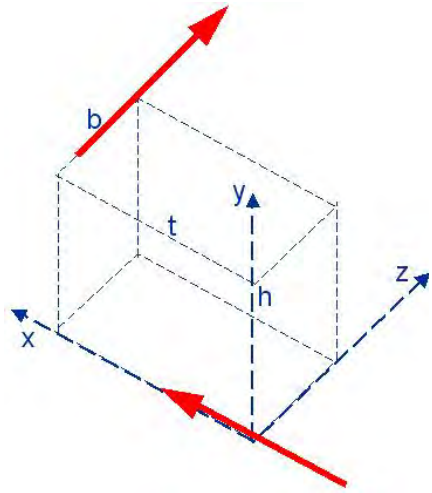
2. **Aufgabe** (30 Punkte)

Sie planen einen 3D-Zeichentrickfilm über Tarzan (Abb. 2.1), der seinen Speer „um die Ecke“ werfen kann, um sich gegen böse Elfenbeinwilderer zu wehren.

Abb. 2.1

Für eine entsprechende Szene wollen Sie die Koordinaten des in die Kurve fliegenden Speeres vorausberechnen. Hierbei wird angenommen, daß Tarzans Hand, die den Speer abwirft, in der Höhe **h** (gemessen entlang der y-Achse) über dem Koordinaten-Ursprung steht. Der Abwurf erfolgt in positiver x-Richtung, und der Speer verfolgt eine rechtwinklige Flugbahn, die ihn zuerst in die Tiefe **t** des Dschungels (parallel zur x-Achse) und dann in die Breite **b** (parallel zur z-Achse) führt.

Zur Vereinfachung Ihrer Berechnung und Ihres Programms modellieren Sie den Speer um den Koordinaten-Ursprung, verschieben ihn wie vorgesehen auf die Höhe von Tarzans Hand und drehen ihn schon dort um die y-Achse, um ihn erst danach an die aktuelle Position zu setzen (Abb. 2.2).



Berechnen Sie bitte die Koordinaten-Transformation für den Speer, indem Sie folgende Fragen behandeln:

a) Wie lautet die Transformationsmatrix $\underline{\mathbf{T}}_{(i)}$, mit welcher der Speer auf die Höhe **h** von Tarzans Hand über den Koordinaten-Ursprung verschoben wird?

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

b) Geben Sie (in Grad, unter Berücksichtigung des Drehsinns) den Winkel **α** an, um welchen der Speer gedreht werden muß, damit er nicht entlang der x-, sondern entlang der z-Achse steht. Wie groß sind dann **sin α** und **cos α**?

α = ***– 90°*** **sin α =** ***–1*** **cos α =** ***0***

c) Wie lautet die Transformationsmatrix $\underline{\mathbf{T}}_{(ii)}$, die den Speer parallel zur z-Achse dreht? Geben Sie sie bitte sowohl in symbolischer (sin α, ...) als auch in arithmetischer (zahlenmäßiger) Form an!

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

d) Wie lautet die Transformationsmatrix $\underline{\mathbf{T}}_{(iii)}$, die den Speer schließlich an die geforderte Position (**t**, **h**, **b**) verschiebt?

$$\underline{\mathbf{T}}_{(iii)} = \begin{pmatrix} 1 & 0 & 0 & t \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & b \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

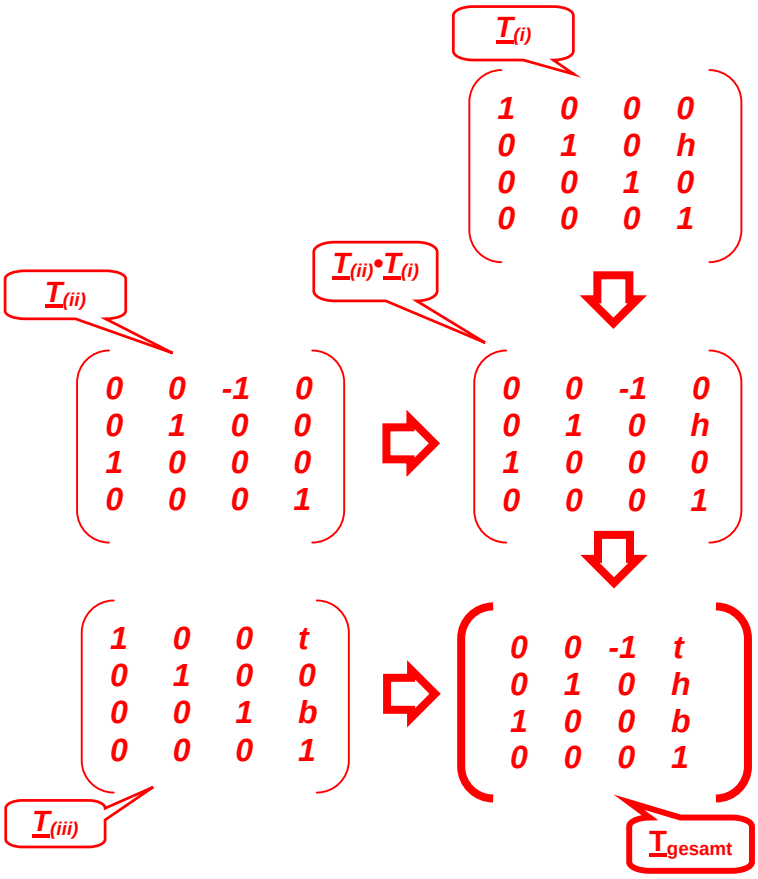
e) Wie berechnet sich die gesuchte Transformationsmatrix für die Koordinaten der Speer-Enden $\underline{\mathbf{T}}_{gesamt}$ aus den bisher besprochenen Transformationen $\underline{\mathbf{T}}_{(i)}$ bis $\underline{\mathbf{T}}_{(iii)}$?

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

f) Berechnen Sie jetzt bitte die gesuchte Transformationsmatrix $\underline{\mathbf{T}}_{gesamt}$ nach den obigen Angaben.

(b.w.) ⇒

$$\underline{T}_{gesamt} = \underline{T}_{(iii)} \bullet [\ \underline{T}_{(ii)} \bullet \underline{T}_{(i)}\ ]$$



g) Berechnen Sie nun bitte die Koordinaten der Speer-Enden, wenn der Speer auf der x-Achse zwischen den Punkten x = -1 und x = 1 modelliert worden war:

(Eventuelle Folgefehler werden als richtig angerechnet.)

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

$$\begin{pmatrix} 0 & 0 & -1 & t \\ 0 & 1 & 0 & h \\ 1 & 0 & 0 & b \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad \begin{pmatrix} t & t \\ h & h \\ b-1 & b+1 \\ 1 & 1 \end{pmatrix}$$

3. **Aufgabe** (45 Punkte)

Sie haben mit Freunden eine erste Version von einem virtuellen Autorennen implementiert (Abb.3.1) und die beiden Quellcode-Dateien (`Carrera.c` und `Carrera.h`) der Öffentlichkeit zur Verfügung gestellt. Sie sind am Ende dieser Aufgabe ausgedruckt.

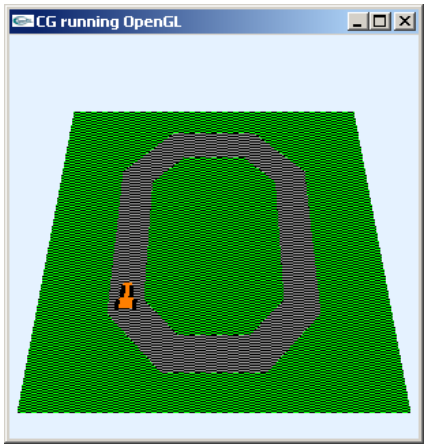


Abb. 3.1

Neugierige, die Sie schon immer wegen Ihrer Teilnahme an einer interessanten Grafik-Vorlesung beneideten, wollen nun von Ihnen auch Einzelheiten zu dieser Software wissen.

Beantworten Sie ihnen bitte folgende Fragen:

a) Das Programm eröffnet offenbar ein GLUT-Fenster. Wie groß ist dieses bei seiner Entstehung, und woran ist dies zu im Programm-Code zu erkennen?

Da kein Aufruf von `glutInitWindowSize()` verwendet wird, hat das erzeugte Fenster die voreingestellte (Default-)Größe von 300 x 300 Pixeln. (Der Aufruf `glutInitWindowPosition(300,300);` betrifft nur die Fenster-Positionierung und hat nichts mit der Fenster-Größe zu tun.)

b) Die Fenster-Software GLUT übernimmt die Verwaltung des Fensters. Welche Funktion Ihres Programms sorgt für die Auffrischung des Fenster-Inhalts, und wie teilt Ihr Programm der GLUT-API mit, daß just diese Funktion aufzurufen ist, wenn die Grafik aktualisiert werden soll?

Die Callback-Funktion `draw()` besorgt die Fenster-Aktualisierung. Die Funktion wird bei GLUT durch den Aufruf `glutDisplayFunc(draw);` angemeldet.

c) Sie erklären Ihren interessierten Bekannten, wie die Rennbahn in der Funktion `scene()` entsteht, und jemand stellt fest, daß die Oberflächen von Rasen und Fahrbahn „unruhig“ und relativ natürlich aussehen. Dabei wirken sie sowohl in Kontrast als auch in „Körnigkeit“ unterschiedlich, was ohne entsprechende Erfahrung in Computergrafik schwer zu beschreiben ist.

Bitte beantworten Sie stichwortartig folgende Fragen:

Wie heißt (auf englisch oder auf deutsch) die Technik, die diese Flächen unruhig erscheinen läßt? Durch welche OpenGL-Anweisung wird sie aktiviert?

Sie heißt Stipple / Stippling (Musterung) und wird durch die Anweisung `glEnable(GL_POLYGON_STIPPLE);` aktiviert.

Auf welcher der beiden Flächen (Wiese, Piste) treten die stärkeren Kontraste (Farbdifferenzen) auf? Woran erkennen Sie das?

Die Wiese verwendet die Farbe `grass[3]={0.,1.,0.}` , die Piste entsprechend `route[3]={.6,.6,.6}` . Beide Flächen erhalten eine schwarze Musterung (`black[3]={0.,0.,0.}`). Die stärkeren Kotraste treten daher bei der Wiese auf, die keine Rot- und Blauanteile hat.

Welche der beiden Flächen erscheint „grobkörniger“, welche feiner strukturiert? Erläutern Sie bitte kurz, woran das eindeutig zu erkennen ist!

Die Wiese ist mit der Musterung `stip1[]={0xF0, 0xF0,...}` feiner: Hier werden abwechselnd 4 Pixel schwarz gesetzt und 4 grün gelassen.

Die Piste ist mit `stip2[]= { 0xFF, 0x00, ...}` gröber: Es werden 8 Pixel schwarz gesetzt und dann 8 grau gelassen.

($F_{16} = 1111_2; 0_{16} = 0000_2$)

d) In der Initialisierungsroutine `init()` wird als Löschfarbe die globale indizierte Variable `about[3]={.9,.95,1.}` vereinbart. Mit ihr soll der nicht belegte Teil des Fensters wahrgenommen werden als (Zutreffendes bitte markieren):

Sand	***Himmel***	Beton	Erde	Wiese	Mauer

e) Jemand weist darauf hin, daß die aktuelle Programm-Version (in der Funktion `model()`) die globale indizierte Variable `color[3]={1.,.5,0.}` verwendet. In einer früheren Version war sie als `color[3]={1.,1.,0.}` definiert. Welche Farbe war dies früher, und welche Farbe bedeuten die jetzigen Farbwerte? Woran erkennen Sie die Auswirkung der veränderten Werte? (Stichwortartige Erklärung)

`color[3]={1.,1.,0.}` steht für Gelb (additive Farbmischung); durch die Einstellung `color[3]={1.,.5,0.}` wird der Rennwagen orange, weil sich der Rot-Anteil erhöht.

f) In der Funktion `init()` werden die Eckpunkt-Koordinaten für das Auto und seine Reifen (globale Variablen `car[][]` und `tire[][]`) gesetzt. Sie sind jeweils realisiert als (Zutreffendes bitte markieren):

Quader	***Rechtecke***	Zylinder	Würfel	Tetraeder

Woran erkennt man innerhalb des Quellcodes die Antwort auf diese Frage?

Sowohl `car[][]` als auch `tire[][]` haben 4 Ecken, deren z-Koordinate auf null gesetzt wird. Das können also keine 3dim. Modelle sein.

g) Ein Bewunderer Ihres Könnens fragt, warum die Funktion `draw()`, die wichtige Aktivitäten auslöst, eine leere Parameterliste hat (`void`). Eine volle Parameterliste könnte (evtl. in einer späteren Programm-Version) die Anzahl der globalen Variablen reduzieren. Was antworten Sie?

Die Parameterliste von `draw()` ist nicht frei wählbar, sondern durch die Callback-Vorgaben von GLUT festgelegt.

h) Die Funktion `model()` „montiert“ die Räder an das Rennwagen-Modell. Welche der folgenden Aussagen geben Aspekte dieses Vorgangs korrekt wieder? Bitte kreuzen Sie die richtige(n) Antwort(en) an!

X	Es wird der Umstand genutzt, daß (in `init()`) sowohl das Auto als auch die Räder um den Koordinaten-Ursprung modelliert sind.
	In `model()` wird erkannt, wo sich das Auto aktuell innerhalb der Landschaft befindet; dann werden die Räder unabhängig vom Auto dorthin gesetzt.
X	`model()` setzt die Räder immer relativ zur Auto-Karosserie. So nutzen Auto und Räder z.T. dieselbe (übergeordnete) Transformation.
	Die Funktion `glPushMatrix()` setzt grafische Modelle aneinander (hier: die Räder an das Auto) so, daß kein Zwischenraum übriggelassen wird; daher bekam sie ihren Namen.
X	Die Funktion `glPushMatrix()` erstellt eine Kopie der noch aktuellen Transformationsmatrix, damit sie später (z.B. nach Erledigung von Neben-rechnungen) wieder zur Verfügung steht.
X	Die Funktionen `glPushMatrix()` und `glPopMatrix()` bilden ein Paar; wiederholter Aufruf der einen ohne entsprechenden Aufruf der anderen kann zu Fehlermeldungen führen.
	`glPopMatrix()` sorgt (anhand eines Fehlerkatalogs) dafür, daß verdeckte Modellteile nicht berechnet werden, damit Rechenzeit eingespart werden kann.

i) Sie wollen die Wirkungsweise der Funktion `move()` auf die mit ihr behandelten Modelle durch eine Reihe von Matrizenprodukten ausdrücken und wählen dafür folgende Beschreibung: (Bitte kreuzen Sie die richtige Antwort an!)

	Ausgehend von einer Einheitsmatrix werden nacheinander eine Translation (in das Sichtvolumen) sowie nacheinander je eine Rotation um die x-, die y- und die z-Achse ausgeführt.
	Ausgehend von einer Einheitsmatrix werden nacheinander eine Translation (in das Sichtvolumen) und je eine Rotation um die z-, die y- und die x-Achse ausgeführt.
X	Ausgehend von einer Einheitsmatrix werden nacheinander je eine Rotation um die z-, die y- und die x-Achse sowie abschließend eine Translation (in das Sichtvolumen) ausgeführt.
	Auf Rotationen um die z-, die y- und die x-Achse und auf eine Translation (in das Sichtvolumen) folgt die Anwendung der Einheitsmatrix.

j) Die Funktion `posit()` enthält zwei Translationen: eine für die Positionierung des Rennwagens (der um den Koordinaten-Ursprung modelliert ist) an die Start-Position und eine für seine Positionierung an die aktuell angefahrene Stelle der Rennbahn. Für die Reihenfolge dieser beiden Transformationen gilt die Aussage: (Bitte kreuzen Sie die richtige(n) Antwort(en) an!)

X	Sie sind vertauschbar, weil zwei aufeinander folgende Translationen unabhängig von ihrer Reihenfolge zur selben Positionierung führen.
	Eine Vertauschung ihrer Reihenfolge wird immer zu einem falschen Ergebnis führen, weil das Matrizenprodukt nicht kommutativ ist.
	Sie sind grundsätzlich nicht vertauschbar, weil sonst die darauf folgende Rotation um den falschen Mittelpunkt erfolgt.
	Eine Vertauschung in der Reihenfolge führt zwar zu einem leicht veränderten Ergebnis; dies ist aber unproblematisch, weil man das bei einem interaktiven Programm durch die Bedienung des Programms ausgleichen kann.

k) Die Funktion `posit()` besteht aus zwei Translationen und einer Rotation. Anschaulich läßt sich ihre Wirkungsweise folgendermaßen darstellen: (Bitte kreuzen Sie die richtige Antwort an!)

	Das Rennwagen-Modell wird an den Start des Rennens gesetzt, von dort aus an die momentan erreichte Position gebracht und dort auf der Stelle so gedreht, daß er tangential zur Fahrtrichtung dargestellt wird.
X	Das Rennwagen-Modell wird zunächst um die eigene (Symmetrie-/ Modellierungs-) Achse gedreht , so daß es passend zur Fahrtrichtung steht; dann wird es um die Strecke verlegt, die es inzwischen zurückgelegt hat. Abschließend wird berücksichtigt, daß Startpunkt nicht der Koordinaten-Ursprung, sondern die Startposition auf der Rennbahn war.
	Das Rennwagen-Modell wird gedreht (konkret: tangential zur aktuellen Fahrtrichtung) an den Start des Rennens gesetzt. Von dort aus wird es mit einer weiteren Translation an die momentan erreichte Position gebracht.

l) In der Funktion `draw()` werden nacheinander `move()` und `posit()` aufgerufen. Diese Tatsache läßt sich aus mathematischer Sicht wie folgt veranschaulichen: (Bitte kreuzen Sie die richtige Antwort an!)

X	Erst wird mit `move()` die Darstellung des Geländes im Fenster berechnet; danach setzt gewissermaßen `posit()` den Rennwagen (durch Fortführung der Transformationen) darauf.
	Aufgrund der in OpenGL implementierten Rechts-Multiplikation von Transformationsmatrizen erkennt man, daß zunächst `posit()` die korrekte Darstellung des Rennwagens berechnet und `move()` dann das Gelände darum „legt“.

m) Im gesamten Code von `Carrera.c` findet sich keine Löschung des Tiefenpuffers. Wodurch wird sichergestellt, daß OpenGL Wiese, Straßenpflaster, Auto und Reifen in richtiger Folge übereinander zeichnet? (Stichwortartige Antwort!)

Die Modelle werden nacheinander gezeichnet; was später gezeichnet wird, überschreibt den Bildspeicher und verdeckt das zuvor Gezeichnete.

```
/* Carrera.c */
/*Darstellung einer Rennbahn mit OpenGL*/
#include "Carrera.h"
/*Globale Variablen: */
float    corner=0.f, radius=0.f, tx=0., ty=0., rz=0., angle[3]={0.,0.,0.},
         about[3]= {.9f, .95f, 1.f}, black[3]={0.f,0.f,0.f},
         grass[3]={0.f,1.f,0.f}, route[3]={.6f,.6f,.6f},
         color[3]={1.f,.5f,0.f};
GLdouble nah=FLD_Y, fern=2*FLD_Y;
GLfloat  field[4][3], car[4][3], tire[4][3];

/*****************************************************************************/
  void move(void)
/*****************************************************************************/
/*Gelaende positionieren:*/
{ glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();

  /*Alles ins Sichtvolumen verschieben:*/
  glTranslatef(0., 0., -(nah+FLD_X/4.));

  /*Positionierung:*/
  glRotatef(angle[X], 1., 0., 0.);
  glRotatef(angle[Y], 0., 1., 0.);
  glRotatef(angle[Z], 0., 0., 1.);
  return;
}
/*****************************************************************************/
  void posit(void)
/*****************************************************************************/
/*Objekt positionieren:*/
{ /*An den Bahn-Start:*/
  glTranslatef(field[1][X]+OFF_X+WIDTH/2.,
               field[1][Y]+OFF_Y+WIDTH+radius, 0.);

  /*Nach dem Start erreichte Position:*/
  glTranslatef(tx, ty, 0.);
  glRotatef(rz, 0., 0., 1.);
}
/*****************************************************************************/
  void model(void)
/*****************************************************************************/
/*Objekt zeichnen:*/
{ int j1=0;

  /*Karosserie:*/
  glDisable (GL_POLYGON_STIPPLE);
  glColor3fv (color);
  glBegin(GL_POLYGON);
    glVertex3fv(car[0]);   glVertex3fv(car[1]);
    glVertex3fv(car[2]);   glVertex3fv(car[3]);
  glEnd();

  /*Raeder:*/
  glColor3fv (black);
  for (j1=0; j1<4; j1++)
  { glPushMatrix();
    if (j1==0) glTranslatef(CAR_X/2.f,  CAR_Y/2.f, 0.);
    if (j1==1) glTranslatef(CAR_X,     -CAR_Y/3.f, 0.);
    if (j1==2) glTranslatef(-CAR_X/2.f, CAR_Y/2.f, 0.);
    if (j1==3) glTranslatef(-CAR_X,    -CAR_Y/3.f, 0.);

    glBegin(GL_POLYGON);
      glVertex3fv(tire[0]);   glVertex3fv(tire[1]);
      glVertex3fv(tire[2]);   glVertex3fv(tire[3]);
    glEnd();
    glPopMatrix();
  }
}
```

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

  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  glFrustum (-FLD_X/2., FLD_X/2., -FLD_Y/2., FLD_Y/2., nah, fern);

  /*Positionierung Gelaende:*/
  move();
  /*Zeichnung Gelaende:*/
  scene();
  /*Positionierung Objekt:*/
  posit();
  /*Zeichnung Objekt:*/
  model();

  glFlush();
}

/*************************************************************************/
  void init(void)
/*************************************************************************/
/*Eckpunkt-Koordinaten Gelaende (gegen den UZS, beginnend o.li.):*/
{ field[0][X] = field[1][X] = -FLD_X/2.; //FLD_X=10
  field[2][X] = field[3][X] =  FLD_X/2.;
  field[0][Y] = field[3][Y] =  FLD_Y/2.; //FLD_Y=10
  field[1][Y] = field[2][Y] = -FLD_Y/2.;
  field[0][Z] = field[1][Z] = field[2][Z] = field[3][Z] = 0.f;

  /*Abmessungen der Kurve:*/
  corner = WIDTH*tan(atan(1.)/2.);
  radius = (FLD_X - 2*OFF_X - 2*WIDTH)/4.;

  /*Abmessungen des Autos (gegen den UZS, beginnend o.li.):*/
  car[0][X] =-CAR_X/2.f;  car[1][X] = -CAR_X;
  car[2][X] = CAR_X;      car[3][X] = CAR_X/2.f;
  car[0][Y] = car[3][Y] =  CAR_Y;
  car[1][Y] = car[2][Y] = -CAR_Y/2.f;
  car[0][Z] = car[1][Z] = car[2][Z] = car[3][Z] = 0.f;

  /*Abmessungen des Reifens (gegen den UZS, beginnend o.li.):*/
  tire[0][X] = tire[1][X] = -CAR_X/4.f;
  tire[2][X] = tire[3][X] =  CAR_X/4.f;
  tire[0][Y] = tire[3][Y] =  CAR_Y/3.f;
  tire[1][Y] = tire[2][Y] = -CAR_Y/3.f;
  tire[0][Z] = tire[1][Z] = tire[2][Z] = tire[3][Z] = 0.f;

  /*Loeschfarbe:*/
  glClearColor (about[0], about[1], about[2], 1.);

  /*Tastendruck simulieren, um Menue auszugeben:*/
  key(' ', 0, 0);
  return;
}
```

```
/*********************************************************************/
  void key(unsigned char key, int x, int y)
/*********************************************************************/
/*Menue und Eingabe-Behandlung:*/
{ const  double  GRAD  = atan(1.) / 45.;
  switch (key)
  { case ESC: exit(0);
    case 'a': rz += 10.;  if (rz >= 360) rz-=360; break;
    case 'd': rz -= 10.;  if (rz <=-360) rz+=360; break;
    case 'w': ty += CAR_Y*cos(rz*GRAD); tx -= CAR_Y*sin(rz*GRAD); 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         turn left  (10 degrees)");
  printf ("\n\r d         turn right (10 degrees)");
  printf ("\n\r w         drive on)");
  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]);
  printf ("\n\r           driving angle=%7.2f", rz);

  draw();
  return;
}

/*********************************************************************/
  int main(int argc, char **argv)
/*********************************************************************/
{ glutInitWindowPosition(300, 300);
  glutInit(&argc, argv);
  glutCreateWindow("CG running OpenGL");
  glutDisplayFunc(draw);
  glutKeyboardFunc(key);
  init();
  glutMainLoop();
  return 0;
}
```

```
/*************************************************************************/
  void scene(void)
/*************************************************************************/
/*Gelaende zeichnen:*/
{ int j1=0;

  /*Wiese:*/
  for (j1=0; j1<2; j1++)
  { if (!j1) { glDisable (GL_POLYGON_STIPPLE); glColor3fv (grass); }
    else     { glEnable (GL_POLYGON_STIPPLE);  glColor3fv (black);
               glPolygonStipple (stip1); }

    glBegin(GL_POLYGON);
      glVertex3fv(field[0]);   glVertex3fv(field[1]);
      glVertex3fv(field[2]);   glVertex3fv(field[3]);
    glEnd();
  }

  /*Piste:*/
  for (j1=0; j1<2; j1++)
  { if (!j1) { glDisable (GL_POLYGON_STIPPLE); glColor3fv (route); }
    else     { glEnable (GL_POLYGON_STIPPLE);  glColor3fv (black);
               glPolygonStipple (stip2); }


    glBegin(GL_QUAD_STRIP);
//SWW ausssen:
      glVertex3f(field[1][X]+OFF_X, field[1][Y]+OFF_Y+WIDTH+radius-corner, 0.f);
//SWW innen:
      glVertex3f(field[1][X]+OFF_X+WIDTH, field[1][Y]+OFF_Y+WIDTH+radius, 0.f);
//NWW aussen:
      glVertex3f(field[0][X]+OFF_X, field[0][Y]-OFF_Y-WIDTH-radius+corner, 0.f);
//NWW innen:
      glVertex3f(field[0][X]+OFF_X+WIDTH, field[0][Y]-OFF_Y-WIDTH-radius, 0.f);
//NNW aussen:
      glVertex3f(field[0][X]+OFF_X+WIDTH+radius-corner, field[0][Y]-OFF_Y, 0.f);
//NNW innen:
      glVertex3f(field[0][X]+OFF_X+WIDTH+radius, field[0][Y]-OFF_Y-WIDTH, 0.f);
//NNO aussen:
      glVertex3f(field[3][X]-OFF_X-WIDTH-radius+corner, field[3][Y]-OFF_Y, 0.f);
//NNO innen:
      glVertex3f(field[3][X]-OFF_X-WIDTH-radius, field[3][Y]-OFF_Y-WIDTH, 0.f);
//NOO aussen:
      glVertex3f(field[3][X]-OFF_X, field[3][Y]-OFF_Y-WIDTH-radius+corner, 0.f);
//NOO innen:
      glVertex3f(field[3][X]-OFF_X-WIDTH, field[3][Y]-OFF_Y-WIDTH-radius, 0.f);
//SOO aussen:
      glVertex3f(field[2][X]-OFF_X, field[2][Y]+OFF_Y+WIDTH+radius-corner, 0.f);
//SOO innen:
      glVertex3f(field[2][X]-OFF_X-WIDTH, field[2][Y]+OFF_Y+WIDTH+radius, 0.f);
//SSO aussen:
      glVertex3f(field[2][X]-OFF_X-WIDTH-radius+corner, field[2][Y]+OFF_Y, 0.f);
//SSO innen:
      glVertex3f(field[2][X]-OFF_X-WIDTH-radius, field[2][Y]+OFF_Y+WIDTH, 0.f);
//SSO aussen:
      glVertex3f(field[1][X]+OFF_X+WIDTH+radius-corner, field[1][Y]+OFF_Y, 0.f);
//SSO innen:
      glVertex3f(field[1][X]+OFF_X+WIDTH+radius, field[1][Y]+OFF_Y+WIDTH, 0.f);
//SWW ausssen:
      glVertex3f(field[1][X]+OFF_X, field[1][Y]+OFF_Y+WIDTH+radius-corner, 0.f);
//SWW innen:
      glVertex3f(field[1][X]+OFF_X+WIDTH, field[1][Y]+OFF_Y+WIDTH+radius, 0.f);
    glEnd();
  }
  return;
}
```

```c
/* Carrera.h */

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

  #define CON_CLS  "cls"
  #define ESC        27
  #define FLD_Y      10
  #define FLD_X      10
  #define WIDTH       1
  #define OFF_X    2*WIDTH
  #define OFF_Y    WIDTH
  #define CAR_X    WIDTH/4.f
  #define CAR_Y    WIDTH/2.f

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

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

/*Globale Variablen: */
  GLubyte stip1[] = {
      0xF0, 0xF0, 0xF0, 0xF0, 0x0F, 0x0F, 0x0F, 0x0F,
      0xF0, 0xF0, 0xF0, 0xF0, 0x0F, 0x0F, 0x0F, 0x0F,
      0xF0, 0xF0, 0xF0, 0xF0, 0x0F, 0x0F, 0x0F, 0x0F,
      0xF0, 0xF0, 0xF0, 0xF0, 0x0F, 0x0F, 0x0F, 0x0F,
      0xF0, 0xF0, 0xF0, 0xF0, 0x0F, 0x0F, 0x0F, 0x0F,
      0xF0, 0xF0, 0xF0, 0xF0, 0x0F, 0x0F, 0x0F, 0x0F,
      0xF0, 0xF0, 0xF0, 0xF0, 0x0F, 0x0F, 0x0F, 0x0F,
      0xF0, 0xF0, 0xF0, 0xF0, 0x0F, 0x0F, 0x0F, 0x0F,
      0xF0, 0xF0, 0xF0, 0xF0, 0x0F, 0x0F, 0x0F, 0x0F,
      0xF0, 0xF0, 0xF0, 0xF0, 0x0F, 0x0F, 0x0F, 0x0F,
      0xF0, 0xF0, 0xF0, 0xF0, 0x0F, 0x0F, 0x0F, 0x0F,
      0xF0, 0xF0, 0xF0, 0xF0, 0x0F, 0x0F, 0x0F, 0x0F,
      0xF0, 0xF0, 0xF0, 0xF0, 0x0F, 0x0F, 0x0F, 0x0F,
      0xF0, 0xF0, 0xF0, 0xF0, 0x0F, 0x0F, 0x0F, 0x0F,
      0xF0, 0xF0, 0xF0, 0xF0, 0x0F, 0x0F, 0x0F, 0x0F,
      0xF0, 0xF0, 0xF0, 0xF0, 0x0F, 0x0F, 0x0F, 0x0F};

  GLubyte stip2[] = {
      0xFF, 0x00, 0xFF, 0x00, 0x00, 0xFF, 0x00, 0xFF,
      0xFF, 0x00, 0xFF, 0x00, 0x00, 0xFF, 0x00, 0xFF,
      0xFF, 0x00, 0xFF, 0x00, 0x00, 0xFF, 0x00, 0xFF,
      0xFF, 0x00, 0xFF, 0x00, 0x00, 0xFF, 0x00, 0xFF,
      0xFF, 0x00, 0xFF, 0x00, 0x00, 0xFF, 0x00, 0xFF,
      0xFF, 0x00, 0xFF, 0x00, 0x00, 0xFF, 0x00, 0xFF,
      0xFF, 0x00, 0xFF, 0x00, 0x00, 0xFF, 0x00, 0xFF,
      0xFF, 0x00, 0xFF, 0x00, 0x00, 0xFF, 0x00, 0xFF,
      0xFF, 0x00, 0xFF, 0x00, 0x00, 0xFF, 0x00, 0xFF,
      0xFF, 0x00, 0xFF, 0x00, 0x00, 0xFF, 0x00, 0xFF,
      0xFF, 0x00, 0xFF, 0x00, 0x00, 0xFF, 0x00, 0xFF,
      0xFF, 0x00, 0xFF, 0x00, 0x00, 0xFF, 0x00, 0xFF,
      0xFF, 0x00, 0xFF, 0x00, 0x00, 0xFF, 0x00, 0xFF,
      0xFF, 0x00, 0xFF, 0x00, 0x00, 0xFF, 0x00, 0xFF,
      0xFF, 0x00, 0xFF, 0x00, 0x00, 0xFF, 0x00, 0xFF,
      0xFF, 0x00, 0xFF, 0x00, 0x00, 0xFF, 0x00, 0xFF };
#endif //CARRERA_H
```

Platz für Notizen: