Aufgaben und Lösungen

Oktober 2005

Sie finden in diesem Dokument alle Aufgaben und die zugehörigen Lösungen aus

Michael Bender, Manfred Brill: *Computergrafik*

2. Auflage, Hanser Verlag, München, 2006

http://www.vislab.de/cgbuch

Diese Unterlagen wurden mit LaTeX erstellt.

Kapitel 7

Computer-Animation

7.2 Basistechnologien und Interpolation

1. Die Polygonzüge $P_1 = \{(0,1),(1,1),(1,0)\}$ und $P_2 = \{(1,1),(1,0),(0,0)\}$ sollen zwei Keyframes einer Animation darstellen.

   a) Berechnen Sie mit Hilfe von punktweiser linearer Interpolation $(1-t)P_1 + tP_2$ die Frames für $t_i = \frac{i}{10}$, $0 \leq i \leq 10$ und stellen Sie diese Frames in einem kartesischen Koordinatensytem dar!

   b) Berechnen Sie Frames für die beiden Keyframes, falls die Animation aus einer Drehung von P_1 um den Punkt $Z = (\frac{1}{2}, \frac{1}{2})$ besteht!

   Lösung:

   a) In Abbildung 7.1 sehen Sie nochmals die beiden Ausgangsposition. Allgemein gilt für ein $t \in \mathbb{R}$

   $$X_1(t) = (1-t)\begin{pmatrix}0\\1\end{pmatrix} + t\begin{pmatrix}1\\1\end{pmatrix} = \begin{pmatrix}t\\1\end{pmatrix},$$
   $$X_2(t) = (1-t)\begin{pmatrix}1\\1\end{pmatrix} + t\begin{pmatrix}1\\0\end{pmatrix} = \begin{pmatrix}1\\1-t\end{pmatrix}$$
   $$X_3(t) = (1-t)\begin{pmatrix}1\\0\end{pmatrix} + t\begin{pmatrix}0\\0\end{pmatrix} = \begin{pmatrix}1-t\\0\end{pmatrix}.$$

   Es ist $P_1 = \{X_1(0), X_2(0), X_3(0)\}$ und $P_2 = \{X_1(1), X_2(1), X_3(1)\}$. Der Übergang zwischen den Punkten erfolgt immer auf einer Linie; beispielsweise verlässt $X_1(t)$ nie die Linie zwischen $(0,1)$ und $(1,1)$. Das bedeutet insbesondere, dass der Abstand zwischen den einzelnen Punkten nicht konstant bleibt. Es ist

   $$\|X_2(t) - X_1(t)\|^2 = 1 - 2t + 2t^2 = \|X_3(t) - X_2(t)\|^2.$$

   Die Abstände zwischen den einzelnen Punkte für festes t sind allerdings gleich. Der maximale Abstand wird für $t = 0$ und $t = 1$ angenommen; minimal ist der Abstand bei $t = \frac{1}{2}$, dann ist $\|X_2(t) - X_1(t)\| = \|X_3(t) - X_2(t)\| = \frac{1}{4}\sqrt{10} \approx 0.79$.

   b) Die Ergebnisse der Teilaufgabe a) deuten darauf hin, dass lineare Interpolation der Polygonecken nicht zum gewünschten Ergebnis führt. Betrachtet man sich die beiden Keyframes, dann deutet vieles darauf hin, dass der Übergang von P_1 zu P_2 durch eine Drehung im Uhrzeigersinn um den Punkt $(\frac{1}{2}, \frac{1}{2})$ erfolgt.

Abbildung 7.1: Die beiden Keyframes für Aufgabe 1

Die Transformationsmatrix für diese Drehung für einen Winkel $\varphi \in [0, -\pi]$ ist gegeben durch

$$\begin{aligned} T &= T(\frac{1}{2}, \frac{1}{2}, 0) R_z(\varphi) T(-\frac{1}{2}, -\tfrac{1}{2}, 0) \\ &= \begin{pmatrix} \cos(\varphi) & -\sin(\varphi) & 0 & -\frac{1}{2}\cos(\varphi) + \frac{1}{2}\sin(\varphi) + \frac{1}{2} \\ \sin(\varphi) & \cos(\varphi) & 0 & -\frac{1}{2}\sin(\varphi) - \frac{1}{2}\cos(\varphi) + \frac{1}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \end{aligned}$$

c) Setzt man jetzt 10 äquidistant gewählte Winkel aus dem Intervall $[0, \pi]$ ein, dann erhält man in b) immer Polygonzüge, bei denen die Abstände zwischen den drei beteiligten Punkten immer konstant gleich 1 sind.

2. Schreiben Sie ein Programm, mit dessen Hilfe das simple Auto-Modell aus Abbildung 7.6 darstellt und über 900 Frames animiert. Verwenden Sie dabei lineare Interpolation für die Berechnung der Zwischenpositionen! Verwenden Sie für das „Chassis“ den Polygonzug $\{(-0.2, 0.25)$, $(1.85, 0.25)$, $(1.85, 0.75)$, $(1.4, 0.75)$, $(1.4, 1.25)$, $(0.25, 1.25)$, $(0.25, 0.75)$, $(-0.2, 0.75)\}$ und für die „Reifen Kreise mit Radius $r = 0.25$.

Lösung:
Diese Aufgabe kann mit Hilfe von `OpenGL` implementiert werden. Hier der `display`-Callback für die GLUT:

```
void display(void)
{
    float xPos;
    // Lineare Interpolation
    xPos = linear(frame, numberOfFrames, minX, maxX);

    glMatrixMode(GL_MODELVIEW);
    glClear(GL_COLOR_BUFFER_BIT);

    glColor3f(0.0, 0.0, 0.0);
    // Boden ausgeben
    glBegin(GL_LINE_STRIP);
       glVertex2f(-0.5, -0.25);
       glVertex2f(8.5, -0.25);
    glEnd();

    // Auto ausgeben mit Hilfe der Funktion drawCar
    glPushMatrix();
       glTranslatef(xPos, 0.0, 0.0);
       drawCar(xPos);
    glPopMatrix();
}
```

Hier die Implementierung der linearen Interpolation und die Ausgabefunktion:

```
void drawCircle(float r, int n)
```

```
{
    int i;
    float x;
    glBegin(GL_LINE_LOOP);
        for (i=0; i<n-1; i++) {
            x = (float)i/(float)(n-1);
            glVertex2f(r*cos(2.0*M_PI*x),
                       r*sin(2.0*M_PI*x));
        }
    glEnd();
}

void drawCar(float xPos)
{
    glColor3f(0.0, 0.0, 0.0);
    glBegin(GL_LINE_LOOP);
        glVertex2f(-0.2, 0.25);
        glVertex2f(1.85, 0.25);
        glVertex2f(1.85, 0.75);
        glVertex2f(1.4, 0.75);
        glVertex2f(1.4, 1.25);
        glVertex2f(0.25, 1.25);
        glVertex2f(0.25, 0.75);
        glVertex2f(-0.2, 0.75);
    glEnd();

    // Hinter-Reifen ausgeben
    glPushMatrix();
        glTranslatef(0.25, 0.0, 0.0);
            drawCircle(0.25, 19);
        glPopMatrix();

    // Vorder-Reifen ausgeben
    glPushMatrix();
        glTranslatef(1.4, 0.0, 0.0);
            drawCircle(0.25, 19);
        glPopMatrix();
}

float linear(int frame, int numberOfFrames, float key0, float key1)
{
    float t;
    t = (float)frame/(float)numberOfFrames;
    return (1.0-t)*key0 + t*key1;
}
```

Den kompletten Quelltext finden Sie unter den Downloads zu diesem Kapitel!

3. Implementieren sie eine Funktion, die es Ihnen mit Hilfe von Hermite-Polynomen erlaubt, Geschwindigkeitskurven zu definieren, und steuern Sie damit die Animation aus Aufgabe 2 so, dass das Modell langsam beschleunigt, schneller wird und am Ende langsam abbremst und zum Halten kommt!

   Lösung:
   Im Quelltext finden Sie drei verschiedene Inbetweenings. In Abbildung 7.2 sehen Sie ein Screen-Capture des Programms. Einmal ganz oben linear, dann ease-in-ease-out, also zwei Keyframes und zu Beginn und am Ende die Ableitung Null.

Ganz unten sehen Sie ein Auto, das von einem Hermite-Spline gesteuert wird. Dabei wurden neben den beiden Anfangs- und Endframe noch nach einem Drittel und nach zwei Drittel der Wegstrecke Schlüsselszenen eingefügt; und dort die Ableitung, also die Geschwindigkeit des Autos, auf Null gesetzt.

Wollen Sie dieses Verhalten ändern, können Sie dies durch eine Änderung des Quelltexts in der Funktion `display` tun. Hier der Ausschnitt, in dem die Änderungen vorgenommen werden müssen.

```
// Hier die Schlüsselszenen.
keys[0] = minX;
keys[1] = (maxX - minX)/3.0;
keys[2] =  2.0 * keys[1];
keys[3] = maxX;

// Hier die Geschwindigkeitswerte
derivs[0] = 0.0;
derivs[1] = 0.0;
derivs[2] = 0.0;
derivs[3] = 0.0;

xPos0 = nonlinear(frame, numberOfFrames, keys, derivs);
xPos1 = easeineaseout(frame, numberOfFrames, minX, maxX);
xPos2 = linear(frame, numberOfFrames, minX, maxX);
```

Den kompletten Quellcode finden Sie unter den Downloads zu diesem Kapitel!

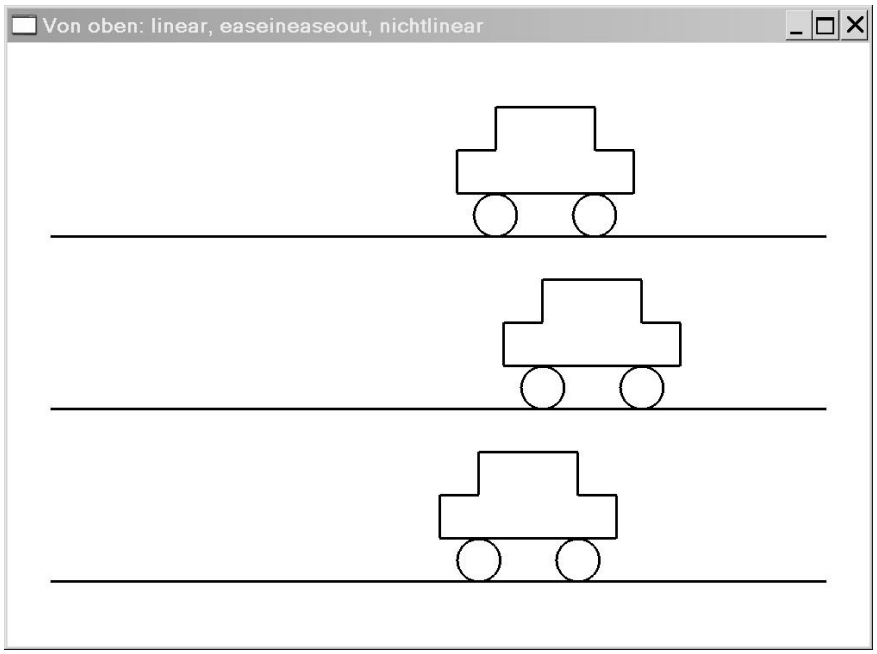

Abbildung 7.2: Screen-Capture zu Aufgabe 3

4. Drehen Sie ein Quadrat der Seitenlänge 1 in der xy-Ebene, dessen linker unterer Eckpunkt im Ursprung liegt mit Hilfe von Quaternionen!

   Lösung:

Die Rotationsachse ist $\mathbf{n} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$, die z-Achse. Dann entspricht eine Drehung um $\varphi = 45°$ dem Quaternion

$$q = (\cos(45°), 0, 0, \sin({}^4 5°)) = (\frac{1}{2}\sqrt{2}, 0, 0, \frac{1}{2}\sqrt{2}).$$

Das konjugierte Quaternion ist dann

$$q' = (\frac{1}{2}\sqrt{2}, 0, 0, -\frac{1}{2}\sqrt{2}).$$

Exemplarisch soll der Punkt $P = (1, 1, 0)$ gedreht werden. Dann bilden wir diesen Punkt auf das Quaternion $p = (0, 1, 1, 0)$ ab; die Rotation ist dann in Quaternionen gegeben durch

$$\begin{aligned} R_q(p) &= q \cdot p \cdot q' \\ &= (\frac{1}{2}\sqrt{2}, 0, 0, -\frac{1}{2}\sqrt{2})(0, 1, 1, 0)(\frac{1}{2}\sqrt{2}, 0, 0, -\frac{1}{2}\sqrt{2}) \\ &= (0, 0, \sqrt{2}, 0)(\frac{1}{2}\sqrt{2}, 0, 0, -\frac{1}{2}\sqrt{2}) \\ &= (0, -1, 1, 0). \end{aligned}$$

Der rotierte Punkt kann jetzt im Imaginärteil abgelesen werden als $P' = (-1, 1, 0)$.

Bei den Downloads zu diesem Kapitel finden Sie eine Implementierung einer `C++`-Klasse `vlQuaternion`, mit deren Hilfe Sie Ihre Ergebnisse überprüfen können!

5. Jedem Quaternion mit Länge 1 entspricht eine Rotationsmatrix. Beschreiben Sie eine Funktion, die diese Umwandlung durchführt!

   Lösung:
   Gegeben sei ein Quaternion $q = s + ia + jb + kc$ mit $|q| = 1$ mit $s = \cos(\frac{\theta}{2})$ und $a = \sin(\frac{\theta}{2})n_1$, $b = \sin(\frac{\theta}{2})n_2$, $c = \sin(\frac{\theta}{2})n_3$, wenn die Drehachse gegeben ist durch $(n_1, n_2, n_3)^T$.
   Eine solche Drehung kann dargestellt werden durch

$$R = \begin{pmatrix} tn_1^2 + c & tn_1n_2 - sn_3 & tn_1n_3 + sn_2 & 0 \\ tn_1n_2 + sn_z & tn_2^2 + c & tn_2n_3 - sn_1 & 0 \\ tn_1n_3 - sn_2 & tn_2n_3 + sn_1 & tn_3^2 + c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

   mit $s = \sin\theta, c = \cos\theta$ und $t = 1 - \cos\theta$; diese Darstellung finden sie im Kapitel 2 auf Seite 20. Die euklidische Länge der Achse ist 1, damit kann die Matrix geschrieben werden als

$$R = \begin{pmatrix} 1 - t(n_2^2 + n_3^2) & tn_1n_2 - sn_3 & tn_1n_3 + sn_2 & 0 \\ tn_1n_2 + sn_3 & 1 - t(n_1^2 + n_3^2) & tn_2n_3 - sn_1 & 0 \\ tn_1n_3 - sn_2 & tn_2n_3 + sn_1 & 1 - t(n_1^2 + n_2^2) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

   Für die trigonometrischen Funktionen gilt:

$$2\sin^2(\frac{\theta}{2}) = 1 - \cos(\theta), \sin(\theta) = 2\sin(\frac{\theta}{2})\cos(\frac{\theta}{2}).$$

   Dann können wir die folgenden Gleichungen herleiten:

$$\begin{aligned} &2sa = n_1\sin(\theta), 2sb = n_2\sin(\theta), 2sc = n_3\sin(\theta), \\ &2a^2 = n_1^2(1 - \cos(\theta)), 2b^2 = n_2^2(1 - \cos(\theta)), 2c^2 = n_3^2(1 - \cos(\theta)), \\ &2ab = n_1n_2(1 - \cos(\theta)), 2ac = n_1n_3(1 - \cos(\theta)), 2bc = n_2n_3(1 - \cos(\theta)). \end{aligned}$$

Dasas Element r_{11} ist gegeben durch

$$r_{11} = 1 - (1 - \cos(\theta))(n_2^2 + n_3^2) = 1 - 2b^2 - 2c^2;$$

r_{12} durch

$$r_{12} = -\sin(\theta)n_3 + (1 - \cos(\theta))n_1n_2 = -2sc + 2ab;$$

r_{13} durch

$$r_{13} = \sin(\theta)n_2 + (1 - \cos(\theta))n_1n_3 = 2sb + 2ac.$$

Insgesamt ist die Rotationsmatrix R zum Quaternion $q = s + ia + jb + kc$ mit $|q| = 1$ gegeben durch

$$R = \begin{pmatrix} 1 - 2b^2 - 2c^2 & -2sc + 2ab & 2sb + 2ac \\ 2sc + 2ab & 1 - 2a^2 - 2b^2 & -2sa + 2bc \\ -2sb + 2ac & 2sa + 2bc & 1 - 2a^2 - 2b^2 \end{pmatrix}$$

7.3 Animation hierarchischer Objekte

1. Berechnen Sie die Koordinaten des Greifers G_3 des Roboters aus Abbildung 7.39 für $a_1 = 2, \theta_2 = 10°, a_3 = 2$ und den Steuerparametern $\theta_1 = -45°, a_2 = 1, \theta_3 = 30°$ beziehungsweise $\theta_1 = 30°, a_2 = 2, \theta_3 = -45°$. Gehen Sie dabei davon aus, dass G_0 im Ursprung des Welt-Koordinatensystems liegt. Skizzieren Sie zur Probe die Position des Roboters!

   Lösung:
   Die Greiferposition für den ersten Satz von Parametern ist gegeben durch $(4.22576, -2.1621)^T$. Für den zweiten Satz von Parametern erhält man die Greiferposition $(5.25653, 2.11126)^T$.

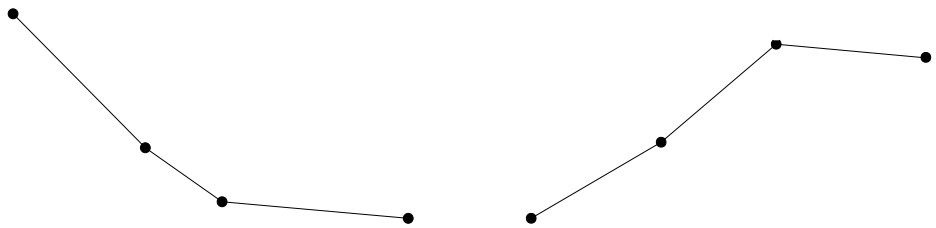

Abbildung 7.3: Die Positionen in Aufgabe 1; links der erste, rechts der zweite Parametersatz

2. Berechnen Sie die Bahnfläche des Greifers G_3 des Roboters aus Abbildung 7.39 für $a_1 = 2, \theta_2 = 10°, a_3 = 2$, falls $a_2 = 2$ ist und die Achswinkel θ_1 und θ_3 jeweils zwischen $-45°$ und $45°$ variieren!

   Lösung:
   Man erhält eine ebene Fläche, da alle Gelenke des Roboters in einer Ebene liegen. In Abbildung 7.4 zeigt den Erreichbarkeitsbereich für die Winkel $\theta_1, \theta_3 \in [-45°, 45°]$. Eine Näherung erhält man durch Variieriung beiden Winkel, in der Abbildung mit $1°$ Schritten.

3. Beschreiben Sie den Roboter auf Seite 77 mit Denavit-Hartenberg-Parametern!

   Lösung:
   Wenn man diesmal die Kette von der Bodenplatte des Beins aufbaut, erhält man für das Bein das erste Gelenk mit einem Längenparameter a_1 und dem Rotationswinkel θ_1.

   Als zweites Gelenk folgt jetzt der Kopf. Dabei ist in a_2, dem Achsabstand auch die Torsogroße zu berücksichtigen. Sie ist auf jeden Fall fest. Variabel ist der Kopfwinkel θ_2. Jetzt folgen noch zwei Drehgelenke für die beiden Schultern, und noch zwei Schiebegelenke für die beiden Daumen.

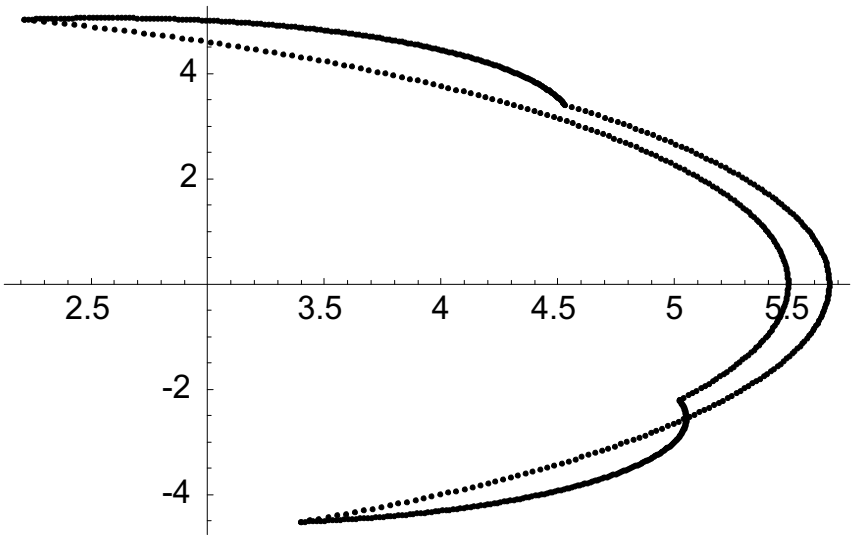

Abbildung 7.4: Der Erreichbarkeitsraum zu Aufgabe 2

4. Die Pseudo-Inverse der Jacobi-Matrix für das Beispiel der inversen Kinematik in Abbildung 7.48 ist gegeben als

$$J^+ = \begin{pmatrix} 0.320066 & 0.382848 \\ -0.374437 & -0.179934 \\ -0.491589 & -0.320066 \end{pmatrix}.$$

Bestimmen Sie die Korrektur für verschiedene selbstgewählte Zielpunkte; bestimmen Sie für die Analyse der Ergebnisse den Tracking-Fehler und den euklidischen Abstand zwischen Zielpunkt und neuer Endeffektor-Position!

Lösung:
Die Lage des Endeffektors in der Ausgangslage war (vgl. Seite 463) gegeben durch $3+\sqrt{3}, 1+\sqrt{3}) = (4.73205, 2.73205)$.

Für den Zielpunkt $(4.7, 2.6)$ erhält man den Korrektorvektor

$$(\Delta\theta_1, \Delta\theta_2, \Delta\theta_3) = (-0.0010614, 0.000624155, 0.00101265)$$

und die Endeffektor-Position $(4.69522, 2.59882)$. Daraus ergibt sich ein Tracking-Fehler von $5.55 \cdot 10^{-17}$), das kann als Null angesehen werden, und als euklidischen Abstand zwischen Ziel und Endeffektor- Position von 0.153959.

Für den Zielpunkt $(4.75, 2.7)$ erhält man den Korrektorvektor

$$(\Delta\theta_1, \Delta\theta_2, \Delta\theta_3) = (-0.000113894, -0.000016647, 0.0000250411)$$

und die Endeffektor-Position $(4.74989, 2.69994)$. Daraus ergibt sich ein Tracking-Fehler von $3.47 \cdot 10^{-18}$), das kann als Null angesehen werden, und als euklidischen Abstand zwischen Ziel und Endeffektor- Position von $3.8 \cdot 10^{-3}$.

7.4 Prozedurale Animationstechniken

1. Berechnen Sie den Pfad eines Partikels, das sich zum Zeitpunkt $t = 0$ im Ursprung befindet und den Geschwindigkeitsvektor $\mathbf{v}(0) = (1, 1, 1)^T$ hat. In der Szene liegen drei stationäre Ebenen, die durch $x + z = 0$, $x = 1$ und $x = -1$ gegeben sind. Bei der Bewegung soll die

Erdbeschleunigung $\mathbf{g} = (0, 0, -9.8)^T$ berücksichtigt werden!

Lösung:
In Abbildung 7.5 sehen Sie das Ergebnis der Berechnung mit einer Schrittweite von $\Delta t = 0.4$ für $t \in [0; 1]$. Es treten nur Kollissionen mit den Ebenen $x + z = 1$ und $x = 1$ auf. Für alle Ebenen wurde $\alpha = 0.5$ verwendet. Die Darstellung zeigt nur die xz-Ebene, denn die Kollissionen erzeugen nur Kräfte in dieser Ebene.

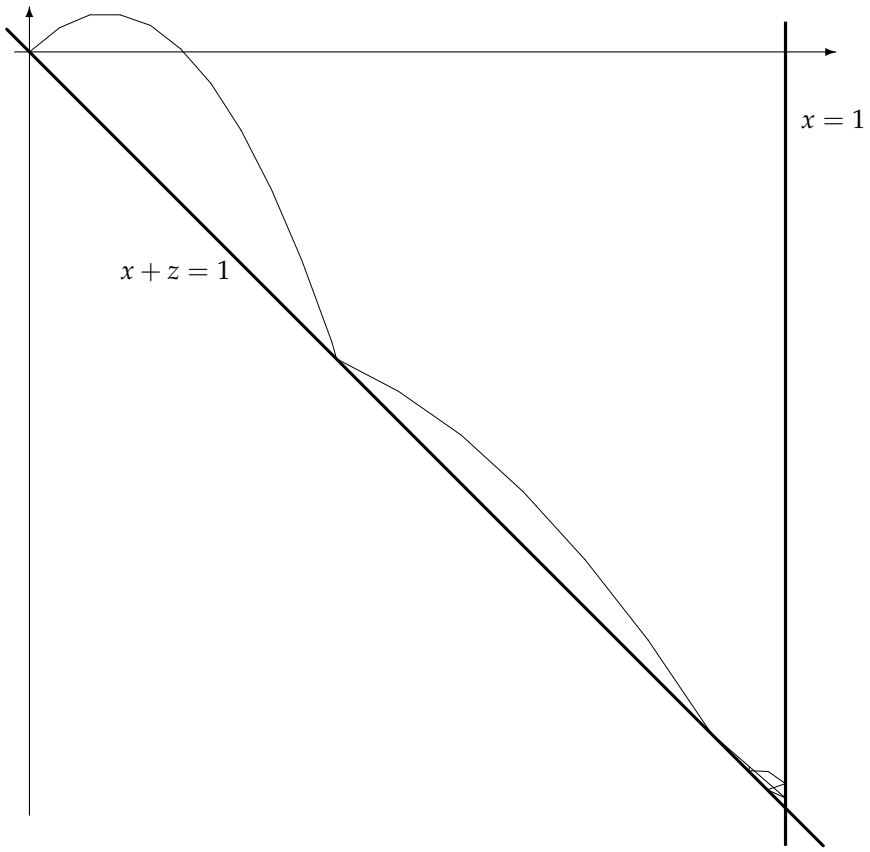

Abbildung 7.5: Der Pfad des Partikels für Aufgabe 1

2. Schreiben Sie ein Programm, das für das Partikel aus Abbildung 7.53 mit Ausgangsposition im Ursprung, Ausgangsgeschwindigkeit $(1, 1)^T$ die Bahn unter dem Einfluss der Gravitation und einer Reibung berechnet und mit OpenGL ausgibt! Variieren Sie die Ausgangsgeschwindigkeit, die Gravitation und die Reibungskonstante, und beobachten Sie die Resultate!

   Lösung:
   In Abbildung 7.6 sehen Sie die Ausgabe des Programms. Von links nach rechts und von oben nach unten wurden die Werte $k_r = 0, 1, 5$ und $k_r = 10$ verwendet. Die Quellen finden Sie unten den Quellcodes zu diesem Kapitel.

   Hier ein Auszug aus den Quellen; die Ausgabe wird auf dem Feld `points` abgelegt, das in `init` entsprechend der gewünschten Anzahl der Integrationsschritte dimensioniert wird.

```
void euler(double x0[2], double v0[2], double deltat, int steps, double kr,
           double **points)
{
    int i;
    double g = -9.8;
```

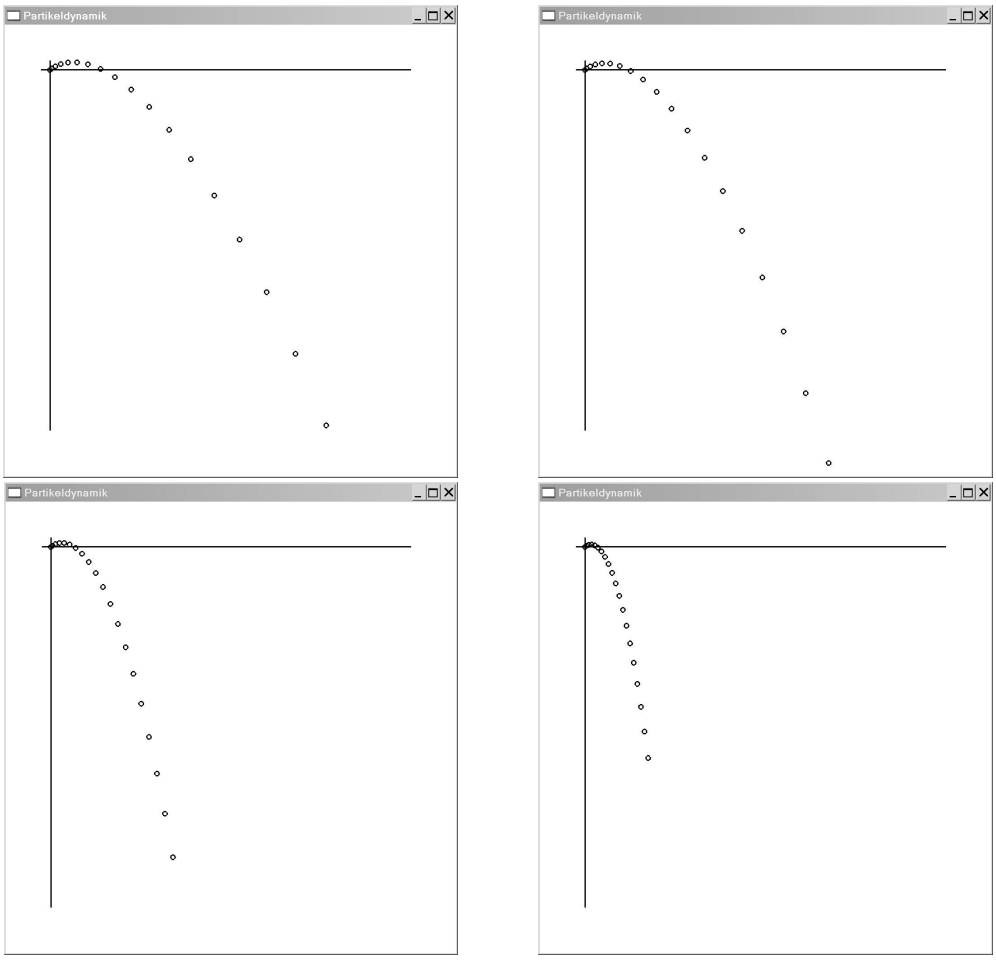

Abbildung 7.6: Die Abbildungen zu Aufgabe 2

```
    double v[2], a[2]; // Die Geschwindigkeit und Beschleunigung

    points[0][0] = x0[0]; points[0][1] = x0[1];

    v[0] = v0[0];
    v[1] = v0[1];

    for (i=1; i<= steps; i++) {
      a[0] = -kr*v[0];
      a[1] = g - kr*v[1];
      points[i][0] = points[i-1][0] + v[0]*deltat + 0.5*a[0]*deltat*deltat;
      points[i][1] = points[i-1][1] + v[1]*deltat + 0.5*a[1]*deltat*deltat;
      v[0] +=  deltat*a[0];
      v[1] +=  deltat*a[1];
    }
}
```

7.6 Fallstudien

7.6.1 Key-Framing und Pfad-Animation in Alias MAYA

1. Modellieren und animieren Sie in der Alias MAYA Personal Learning Edition einen Ball, der wie im Text beschrieben von einer Stufe fällt.
   a) Stellen Sie dabei das Verhalten des Balls so ein, dass er scheinbar aus einem weichen, elastischen Material besteht. Verwenden Sie dafür den Graph View von Alias MAYA.
   b) Verändern Sie das Material des Balls, so dass er sich wie eine Kegelkugel verhält!

   Lösung:
   Die Maya-Datei finden Sie bei den Downloads zu diesem Kapitel.

2. Modellieren Sie einen Quader, der sich in 60 Frames von $x = -10$ nach $x = 10$ bewegt. Erzeugen Sie Key-Frames für die Frames $0, 30$ und 60. Verändern Sie den Funktionsverlauf im Graph View so, dass ein Slow-In-Slow-Out-Verhalten erzeugt wird!

   Lösung:
   Die Maya-Datei finden Sie bei den Downloads zu diesem Kapitel.

3. Modellieren Sie ein kleines Kinderspielzeug oder verwenden Sie eines der Modelle, die in der Personal Learning Edition zur Verfügung stehen. Erstellen Sie eine NURBS-Kurve, die die Form einer Acht annimmt, und verbinden Sie das Objekt mit dem Pfad.
   a) Die Geschwindigkeit des Objekts soll vor den Kurven abnehmen und zum Ausgang der Kurven wieder zunehmen. Stellen Sie dieses Verhalten mit Hilfe des *Graph View* ein!
   b) Verändern Sie das Verhalten des Objekts aus Aufgabe a) so, dass in der Animation einige der Grundprinzipien der Animation enthalten sind!

   Lösung:

   a) Die Maya-Datei finden Sie bei den Downloads zu diesem Kapitel.
   b) Die Maya-Datei finden Sie bei den Downloads zu diesem Kapitel.
   Anticipation ist möglich durch entsprechende Verformungen des Objekts „vor dem Start“; Übertreibung durch ein entsprechendes Verformen oder übertriebenes Kurvenverhalten (beispielsweise auf zwei Rädern zu fahren). Wichtig ist auch, dass die einzelnen Aktionen überlappen!

7.6.2 Expressions in Alias MAYA

1. Bewegen Sie ein einfaches Objekt mit Hilfe der Funktionen `linstep`, `smoothstep` und `hermite`!

   Lösung:
   Bei den Downloads zu diesem Kapitel finden Sie eine entsprechende *MAYA PLE*-Datei!

2. Erstellen Sie zehn Kugeln in Alias MAYA, die in einer Reihe entlang der x-Achse angeordnet sind. Animieren Sie die Position der Kugeln durch eine Expression. Dabei sollen die Kugeln sukzessive nach oben wegfliegen. Die Animation soll im Frame 10 starten. Die nächste Kugel soll sich dann anfangen zu bewegen, wenn der Vorgänger eine Höhe erreicht hat, die dem Fünffachen ihres Radius entspricht. Die Kugeln sollen in einer Höhe, die dem Zehnfachen ihres Radius entspricht, „verschwinden“.

   Lösung:
   Bei den Downloads zu diesem Kapitel finden Sie eine entsprechende *MAYA PLE*-Datei!

3. Erstellen Sie eine Szene mit einem autoähnlichen Objekt, und animieren Sie die Translation durch eine Expression. Koppeln Sie die Rotation der Räder an diese Rotation mit Hilfe einer Expression!

   Lösung:
   Bei den Downloads zu diesem Kapitel finden Sie eine entsprechende *MAYA PLE*-Datei!

4. Erstellen Sie eine Szene, die einen Arm modelliert, wie auf Seite 450 beschrieben. Das Skelett können Sie entweder mit Hilfe eines Polygonzugs erstellen; möglich ist auch die Verwendung von *Sceleton|Joint Tool*. Erstellen Sie Expressions, die die Kontrollpunkte des Lattice mit den Koordinaten des Skeletts koppelt. Beobachten Sie für das Modellieren Ihren eigenen Oberarm!

   Lösung:
   Bei den Downloads zu diesem Kapitel finden Sie eine entsprechende *MAYA PLE*-Datei!

7.6.3 Partikelsysteme in Alias MAYA

1. Erstellen Sie ein einfaches Partikelsystem und experimentieren Sie mit den verschiedenen Vektorfeldern in der Alias MAYA PLE!

   Lösung:
   Bei den Downloads zu diesem Kapitel finden Sie eine entsprechende *MAYA PLE*-Datei!

2. Mit Hilfe des Attributs *position* und *positionPP* können Sie die Position der Partikel in einer Expression kontrollieren. Erstellen Sie Expressions, die die Abbildungen 7.54 und 7.55 nachempfinden!

   Lösung:
   Bei den Downloads zu diesem Kapitel finden Sie eine entsprechende *MAYA PLE*-Datei!

3. Modellieren Sie mit Hilfe von Kollisionsobjekten und Vektorfeldern in der Alias MAYA PLE eine rotationssymmetrische Staupunktströmung wie in Abbildung 7.56!

   Lösung:
   Bei den Downloads zu diesem Kapitel finden Sie eine entsprechende *MAYA PLE*-Datei!

Literaturverzeichnis

[BB06] BENDER, MICHAEL und BRILL, MANFRED: *Computergrafik – Ein anwendungsorientiertes Lehrbuch*. 2. Auflage, Hanser, 2006.

[WND00] WOO, MASON; NEIDER, JACKIE und DAVIS, TOM: *OpenGL Programming Guide (2nd Ed.) – The Official Guide to Learning OpenGL*. Addison-Wesley, 2000.