

Einführung in die Computergrafik für Augmented Reality

Prof. Dr. Philipp Jenke

15. Oktober 2018

Inhaltsverzeichnis

1 Einführung	5
1.1 Mathematische Grundlagen: Vektoren	5
1.1.1 Präambel	5
1.1.2 Vektoren	5
1.1.3 Skalierung	6
1.1.4 Skalarprodukt	6
1.1.5 Orthogonalität	9
1.1.6 Kreuzprodukt	9
2 Polygonale Netze	11
2.1 Mathematische Grundlagen: Matrizen	11
2.1.1 Transponierte	11
2.1.2 Multiplikation	12
2.1.3 Inverse	12
2.1.4 Skalierungsmatrix	13
2.1.5 Rotationsmatrix	14
3 Transformationen	15
3.1 Mathematische Grundlagen: Koordinatensysteme	15
3.1.1 Homogene Koordinaten	15
3.1.2 Basisvektoren	16
3.1.3 Transformationsmatrizen	17
3.1.4 Rotationsmatrix	17
3.1.5 Transformation zwischen beliebigen Koordinatensystemen	18
3.2 Kamera-Transformation	21
3.2.1 Model-Transformation	21
3.2.2 View-Transformation	22
3.2.3 Perspektivische Transformation	25
3.2.4 Pixel	26
3.3 Tracking	28
4 Kurven	30
4.1 Mathematische Grundlagen: Basisfunktionen	30
5 Beleuchtung und Texturen	31
6 Animation	32
6.1 Einführung	32
6.2 Skelett und Knochen	32
6.2.1 Nächster Knochen	34
6.2.2 Einschub: Abstand Punkt-Gerade	35
6.2.3 Gewichtet über alle Knochen	35
6.3 Inverse Kinematik	36
6.3.1 Optimierung	36

7 Datenstrukturen	38
7.1 Mathematische Grundlagen: Geometrische Elemente	38
7.1.1 Gerade/Strahl	38
7.1.2 Ebene	38
8 Simulation	40
8.1 Partikel	40
8.1.1 Beispiel: Feuer	40
8.1.2 Integration	41
8.1.3 Simulation eines Partikelsystems	42
8.1.4 Simulation von Textilien	43
8.2 Simulation von Starrkörpern	45
8.3 Simulation Deformierbarer Körper	47

Vorwort

Dieses Skript stellt einen Teil der Vorlesungsmaterialien zur Veranstaltung *Computergrafik für Augmented Reality* dar. Ein Teil deswegen, weil nicht alle Inhalte in diesem Skript auftauchen. Der Gesamthalt der Veranstaltung setzt sich nämlich zum einen aus diesem Skript und zum anderen aus den Vorlesungsfolien zusammen.

Diesem Skript liegen die folgenden Ansätze zugrunde:

Es gibt **keine Trennung** zwischen den mathematischen Grundlagen und den Computergrafik-Inhalten. Das bedeutet insbesondere, dass es kein eigenständiges Kapitel (oder einen Anhang) für die mathematischen Grundlagen gibt. Stattdessen gibt es immer wieder einen Ausflug zu den Grundlagen an der Stelle, an der diese auch benötigt werden.

Die Inhalte sind **auf Lücke geschrieben**. Das bedeutet, dass keines der Themenfelder umschliessend behandelt wurde. Anstelle dessen habe ich jeweils aus meiner Sicht spannende und repräsentative Ausschnitte und Anwendungen herausgenommen, die ich dann im Detail besprochen haben. Meist gibt es dabei ein sehr weites Spektrum von Alternativen und Erweiterungen, die dann aber nicht in den Veranstaltungen und nicht in den Veranstaltungsmaterialien auftauchen. Ich versuche, jeweils einen Ausblick auf das Spektrum und ggf. Quellen zur Vertiefung aufzulisten.

Die Inhalte, die besprochen werden, werden **soweit vertieft**, dass sie sich mit dem Material direkt umsetzen – meist also implementieren – lassen. Das hast oft zu Folge, dass Grundlagen aus vorangegangenen Lehrveranstaltungen wiederholt werden (insbesondere bei den mathematischen Grundlagen und bei Algorithmen und Datenstrukturen). Meine Erfahrung aus vorangegangenen Durchläufen der Lehrveranstaltung zeigte aber, dass dieses Vorgehen sinnvoll ist, um auch die Studierenden mitzunehmen, die die Grundlagen vielleicht noch nicht vollständig verinnerlicht hatten.

1 Einführung

1.1 Mathematische Grundlagen: Vektoren

Für den vollständigen mathematischen Kontext richten Sie sich bitte nach der entsprechenden Mathematik-Fachliteratur, z.B. in [Fis05].

1.1.1 Präambel

In diesem Skript verwende ich die folgende Notation:

Notation	Beschreibung
x	Skalar
\vec{v} oder \mathbf{x}	Vektor
\mathbf{p}	Punkt
\mathbf{M}	Matrix
.	verschiedene Formen der Multiplikation
\times	Kreuzprodukt

1.1.2 Vektoren

Ein Vektor ist ein Tupel im \mathbb{R}^n mit der Dimension n und besteht aus einer Richtung und einer Länge.

Beispielvektor im 2D: $\vec{v} = \begin{pmatrix} v_x \\ v_y \end{pmatrix}$ mit der x -Koordinate v_x und der y -Koordinate v_y .

Die Länge eines Vektors berechnen wir über die euklidische Norm. Die wird mit $\|\vec{v}\|$ bezeichnet:

$$\|\vec{v}\| = \sqrt{\sum_{i=0}^{n-1} v_i^2}$$

für einen Vektor \vec{v} der Dimension n . Im 2D berechnet sich die Länge also als

$$\|\vec{v}\| = \sqrt{v_x^2 + v_y^2}.$$

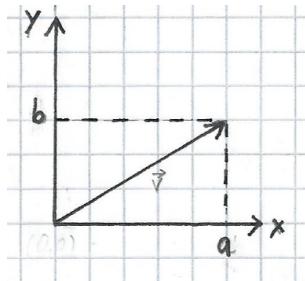


Abbildung 1.1: Ein Vektor im 2D.

Hat ein Vektor die Länge 1, so spricht man von einem Einheitsvektor oder einem normierten Vektor. Jeder Vektor lässt sich in einen Einheitsvektor umwandeln, indem man ihn durch seine Länge teilt:

$\frac{\vec{v}}{\|\vec{v}\|}$. Verändert man einen Vektor, sodass er seine Richtung beibehält aber die Länge 1 bekommt, so nennt man das *Normierung*.

Übung 1.1 Normieren Sie den Vektor $\begin{pmatrix} 1 \\ 1 \end{pmatrix}$.

Vektoren können unterschiedliche Bedeutungen haben. Insbesondere unterscheidet man Ortsvektoren und Richtungsvektoren. Ortsvektoren haben einen festen Startpunkt, nämlich den Ursprung. Sie beschreiben also eine Position im Raum. In Abbildung 1.2 ist der Vektor \vec{v} ein Ortsvektor für den Punkt P .

Richtungsvektoren hingegen beschreiben nur eine Richtung. Sie sind positionsunabhängig und haben daher keinen festen Startpunkt. In Abbildung 1.2 ist der Vektor \vec{w} ein Richtungsvektor. Die Summe aus einem Ortsvektor und einem Richtungsvektor ergibt wieder einen Ortsvektor: $\vec{u} = \vec{v} + \vec{w}$ (Beispiel aus Abbildung 1.2).

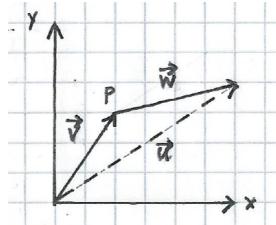


Abbildung 1.2: Es gibt zwei verschiedene Interpretationen für Vektoren: als Ortsvektor oder als Richtungsvektor.

Übung 1.2 Was für einen Typ Vektor ergibt die Differenz zweier Ortsvektoren?

1.1.3 Skalierung

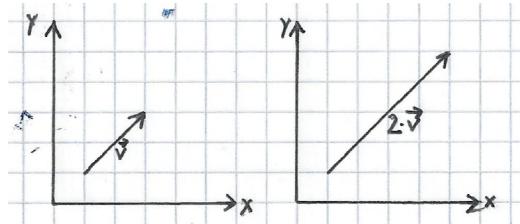


Abbildung 1.3: Die Skalierung eines Vektors verändert die Länge, nicht aber die Richtung des Vektors.

Vektoren lassen sich skalieren. Dazu werden sie einfach mit einem Skalar multipliziert. Für einen Vektor $\vec{v} = \begin{pmatrix} v_x \\ v_y \end{pmatrix}$ gilt: $\lambda \vec{v} = \begin{pmatrix} \lambda v_x \\ \lambda v_y \end{pmatrix}$ für einen Skalierungsfaktor λ . Abbildung 1.3 zeigt die Skalierung eines Vektors mit dem Faktor 2.

Übung 1.3 Skalieren Sie den Vektor $\begin{pmatrix} -1 \\ 2 \end{pmatrix}$ mit den Faktor 1.5.

1.1.4 Skalarprodukt

Das Skalarprodukt (auch inneres Produkt genannt) von zwei Vektoren (gleicher Dimension) liefert eine Zahl (ein Skalar). Dazu werden die beiden Vektoren komponentenweise miteinander multipliziert,

also:

$$\vec{v} \cdot \vec{w} = \sum_{i=0}^{n-1} v_i w_i$$

Im 2D lautet die Berechnung also:

$$\vec{v} \cdot \vec{w} = \begin{pmatrix} v_x \\ v_y \end{pmatrix} \cdot \begin{pmatrix} w_x \\ w_y \end{pmatrix} = v_x w_x + v_y w_y$$

Hinweis: Häufig findet man in der Literatur die Notation $\vec{v}^t \cdot \vec{w}$ für das Skalarprodukt, der erste Vektor wurde also als transponierter Zeilen- (und nicht als Spalten-) Vektor verrechnet. Diese Notation passt auch besser zur Multiplikation von Matrizen. Wir nehmen uns hier die Freiheit, beide Notationen zu akzeptieren (und häufig der Einfachheit halber auf das t zu verzichten). Das Ergebnis ist unabhängig von der Dimension der Vektoren ein Skalar. Das Skalarprodukt ist für Vektoren beliebiger Dimension definiert.

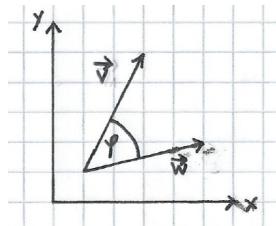


Abbildung 1.4: Das Skalarprodukt zweier Vektoren korrespondiert zum Kosinus des Winkels zwischen den beiden Vektoren.

Übung 1.4 Berechnen Sie das Skalarprodukt zwischen den Vektoren $\begin{pmatrix} -1 \\ 2 \end{pmatrix}$ und $\begin{pmatrix} 2 \\ 1 \end{pmatrix}$.

Das Skalarprodukt hat einige interessante Eigenschaften. So lässt sich das Skalarprodukt zweier Vektoren zur Berechnung des Winkels zwischen den beiden Vektoren verwenden:

$$\vec{v} \cdot \vec{w} = \cos(\phi) \|\vec{v}\| \|\vec{w}\|$$

oder

$$\phi = \arccos \frac{\vec{v} \cdot \vec{w}}{\|\vec{v}\| \|\vec{w}\|}$$

Übung 1.5 Wie groß ist der Winkel ϕ im Gradmaß zwischen den beiden Vektoren $\begin{pmatrix} 2 \\ 0 \end{pmatrix}$ und $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$?

Eine weitere Eigenschaft des Skalarproduktes ist die Projektion des einen Vektors auf den anderen. Wir wollen also hier den Vektor \vec{s} mit der Länge s bestimmen, der sich durch die Projektion von \vec{v} auf \vec{w} ergibt. Betrachten wir Abbildung 1.5, so ist zu erkennen, dass sich aus dem rechten Winkel zwischen \vec{w} und \vec{s} folgender Zusammenhang ergibt:

$$\cos(\phi) = \frac{\|\vec{s}\|}{\|\vec{v}\|}$$

also

$$\cos(\phi) \|\vec{v}\| = \|\vec{s}\|$$

durch Erweiterung mit $\|\vec{w}\|$ auf beiden Seiten erhalten wir

$$\cos(\phi) \|\vec{v}\| \|\vec{w}\| = \|\vec{s}\| \|\vec{w}\|$$

und damit

$$\vec{v} \cdot \vec{w} = \|\vec{s}\| \|\vec{w}\|$$

also

$$\|\vec{s}\| = \frac{\vec{v} \cdot \vec{w}}{\|\vec{w}\|}$$

Die Länge des projizierten Vektors entspricht also dem Skalarprodukt der beiden Vektoren geteilt durch die Länge des Vektors, auf den projiziert wird.

Die (normierte) Richtung des projizierten Vektors ist einfach $\frac{\vec{w}}{\|\vec{w}\|}$. Somit ergibt sich zusammengefasst für den projizierten Vektor \vec{s} :

$$\vec{s} = \frac{\vec{v} \cdot \vec{w}}{\|\vec{w}\|} \frac{\vec{w}}{\|\vec{w}\|}$$

Dieser Term vereinfacht sich, falls der Vektor \vec{w} normiert ist, zu

$$\vec{s} = (\vec{v} \cdot \vec{w}) \vec{w}.$$

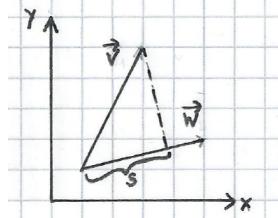


Abbildung 1.5: Mit dem Skalarprodukt lässt sich der Vektor s berechnen, der sich durch die Projektion von v auf w ergibt

Übung 1.6 Berechnen Sie den Abstand zwischen dem Punkt \mathbf{Q} und der Geraden \mathbf{g} (siehe Abschnitt 7.1.1), die durch den Punkt \mathbf{P} und den Richtungsvektor \vec{v} gegeben ist (Abbildung 1.6): $\mathbf{P} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$, $\mathbf{Q} = \begin{pmatrix} 2 \\ 4 \end{pmatrix}$, $\vec{v} = \begin{pmatrix} 4 \\ 1 \end{pmatrix}$.

Hinweis: Verwenden Sie das Skalarprodukt. Der Punkt \mathbf{Q}' ist die Projektion von \mathbf{Q} auf die Gerade \mathbf{g} .

Lösung 1 Normieren von \vec{v} : $\vec{v}' = \frac{1}{\|\vec{v}\|} \vec{v} = \frac{1}{\sqrt{17}} \begin{pmatrix} 4 \\ 1 \end{pmatrix}$

Projektion von \overrightarrow{PQ} auf \vec{v}' : $s = (\overrightarrow{PQ} \cdot \vec{v}') = (\begin{pmatrix} 1 \\ 3 \end{pmatrix} \cdot \frac{1}{\sqrt{17}} \begin{pmatrix} 4 \\ 1 \end{pmatrix}) = \frac{7}{\sqrt{17}}$

Fußpunkt Q' berechnen: $Q' = P + s \cdot \vec{v}' = \begin{pmatrix} 1 \\ 1 \end{pmatrix} + \frac{7}{\sqrt{17}} \cdot \frac{1}{\sqrt{17}} \begin{pmatrix} 4 \\ 1 \end{pmatrix} \approx \begin{pmatrix} 2.65 \\ 1.41 \end{pmatrix}$

Abstand $Q - \text{Strahl} = \|\overrightarrow{QQ'}\| = \left\| \begin{pmatrix} 2 \\ 4 \end{pmatrix} - \begin{pmatrix} 2.65 \\ 1.41 \end{pmatrix} \right\| = \left\| \begin{pmatrix} -0.65 \\ 2.59 \end{pmatrix} \right\| = 2.67$

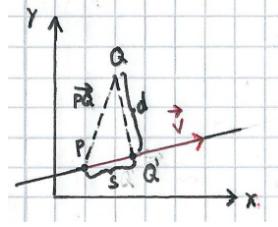


Abbildung 1.6: Das Skalarprodukt gibt auch die Länge s an, die man sich auf dem Vektor w bewegen muss, damit man zu der Position kommt, die der Projektion der Spitze des Vektors w entspricht.

Das Skalarprodukt ist symmetrisch, d. h.

$$\vec{v} \cdot \vec{w} = \vec{w} \cdot \vec{v}.$$

Das Skalarprodukt hat außerdem die Orthogonalitätseigenschaft: Die zwei Vektoren \vec{v} und \vec{w} stehen senkrecht aufeinander genau dann, wenn $\vec{v} \cdot \vec{w} = 0$

Übung 1.7 Geben Sie einen Vektor an, der auf $\begin{pmatrix} -1 \\ 2 \end{pmatrix}$ senkrecht steht, aber nicht $\mathbf{0}$ ist.

1.1.5 Orthogonalität

Häufig wird eine Möglichkeit benötigt, zu einem gegebenen Vektor \vec{v} einen orthogonalen Vektor \vec{w} zu finden, also einen Vektor, der senkrecht auf \vec{v} steht:

$$\vec{v} \perp \vec{w}.$$

Wir unterscheiden bei der Konstruktion den 2D und den 3D. Im zweidimensionalen Fall berechnet sich für einen Vektor

$$\vec{v} = \begin{pmatrix} v_x \\ v_y \end{pmatrix}$$

ein orthogonaler Vektor \vec{w} mit

$$\vec{w} = \begin{pmatrix} -v_y \\ v_x \end{pmatrix}.$$

Überprüfen lässt sich diese Konstruktion durch die Berechnung des Skalarproduktes aus \vec{v} und \vec{w} :

$$\vec{v} \cdot \vec{w} = v_x * (-v_y) + v_y * v_x = -v_x * v_y + v_x * v_y = 0.$$

Im dreidimensionalen Fall verwenden wir das Kreuzprodukt:

1.1.6 Kreuzprodukt

Für Vektoren der Dimension 3 ist ein weiteres Produkt definiert, das Kreuzprodukt. Es berechnet sich so:

$$\vec{v} \times \vec{w} = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} \times \begin{pmatrix} w_x \\ w_y \\ w_z \end{pmatrix} = \begin{pmatrix} v_y w_z - v_z w_y \\ v_z w_x - v_x w_z \\ v_x w_y - v_y w_x \end{pmatrix}.$$

Das Ergebnis der Berechnung eines Kreuzproduktes ist also wieder ein Vektor der Dimension 3. Dieser Vektor hat verschiedenen Bedeutungen:

Das Kreuzprodukt kann, ähnlich wie das Skalarprodukt, dazu verwendet werden, den Winkel zwischen den beiden aufspannenden Vektoren zu berechnen. Im Gegensatz zum Skalarprodukt ergibt sich nicht der Kosinus, sondern der Sinus des Winkels:

$$\|\vec{v} \times \vec{w}\| = \|\vec{v}\| \|\vec{w}\| \sin \phi.$$

Die Länge des Ergebnisvektors entspricht der Fläche des von den beiden Vektoren \vec{v} und \vec{w} aufgespannten Parallelogramms (Abbildung 1.7):

$$A = \|\vec{v} \times \vec{w}\|.$$

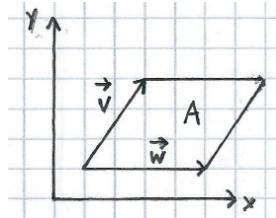


Abbildung 1.7: Das Kreuzprodukt kann verwendet werden, um die Fläche eines Parallelogramms auszurechnen.

Übung 1.8 Wie groß ist der Betrag des Kreuzproduktes zweier Vektoren, die parallel zueinander verlaufen?

Übung 1.9 Wie kann man von der Fläche des Parallelogramms auf die Fläche des Dreiecks zwischen den beiden Vektoren schliessen?

Der Ergebnisvektor des Kreuzproduktes steht immer senkrecht auf den beiden Eingabevektoren. Die Richtung des Vektors lässt sich mit der Rechten-Hand-Regel bestimmen. Zeigt der Vektor \vec{v} in Richtung des Daumens und der Vektor \vec{w} in Richtung des Zeigefingers, dann zeigt der Vektor $\vec{v} \times \vec{w}$ in die Richtung des Mittelfingers (siehe Abbildung 1.8).

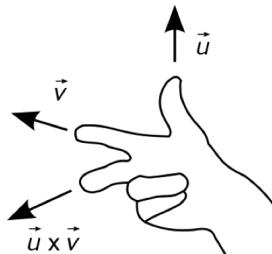


Abbildung 1.8: Mit Hilfe der Rechten-Hand-Regel lässt sich die Richtung des Ergebnisvektors eines Kreuzproduktes bestimmen.

Diese Eigenschaft lässt sich auch an folgender Umformulierung des Kreuzproduktes ablesen:

$$\vec{v} \times \vec{w} = (\|\vec{v}\| \|\vec{w}\| \sin \phi) \vec{n},$$

wobei \vec{n} senkrecht auf \vec{v} und \vec{w} steht.

2 Polygonale Netze

2.1 Mathematische Grundlagen: Matrizen

Wir betrachten folgendes Beispiel: Lotta isst meist Fisch () und Äpfel (). Zwei Informationen sind zu ihrem Konsum bekannt: Die Anzahl der gegessenen Fische und die Anzahl der gegessenen Äpfel zusammen ergibt 18. Außerdem weiss man, dass Lotta für jeden gegessenen Fisch zwei Äpfel isst. Bleibt die Frage, wie viele Fische und wie viele Äpfel sie denn nun genau isst?

Die Lösung zu der Frage ergibt sich aus den folgenden beiden Gleichungen:

$$\begin{array}{lcl} \text{fish} + \text{apple} & = & 18 \\ \text{fish} - 2 \text{apple} & = & 0 \end{array}$$

Dieses lineare Gleichungssystem lässt sich alternativ in der Form

$$\begin{pmatrix} 1 & 1 \\ 1 & -2 \end{pmatrix} \begin{pmatrix} \text{fish} \\ \text{apple} \end{pmatrix} = \begin{pmatrix} 18 \\ 0 \end{pmatrix}$$

oder allgemein als

$$\mathbf{Ax} = \mathbf{b}$$

beschreiben. \mathbf{A} bezeichnen wir als Matrix. Eine Matrix, die aus n Zeilen und m Spalten besteht, bezeichnet man als $n \times m$ Matrix. Entsprechend werden die Einträge einer Matrix (am Beispiel einer 2×3 Matrix) so bezeichnet:

$$\mathbf{A} = \begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} \\ a_{1,0} & a_{1,1} & a_{1,2} \end{pmatrix}$$

2.1.1 Transponierte

Durch Vertauschen aller Einträge entlang der Diagonalen einer Matrix \mathbf{A} erhält man die transponierte Matrix \mathbf{A}^T (oder auch \mathbf{A}^t). Es gilt also:

$$A^T_{i,j} = A_{j,i}$$

oder am Beispiel:

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 5 \end{pmatrix}^T = \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}.$$

Die Transponierte einer $n \times m$ Matrix ist also eine $m \times n$ Matrix.

2.1.2 Multiplikation

Matrizen lassen sich komponentenweise miteinander multiplizieren. Die Dimensionen müssen allerdings zueinander passen: $\mathbb{R}^{m \times n} \cdot \mathbb{R}^{n \times l} \rightarrow \mathbb{R}^{m \times l}$. Die Anzahl der Spalten der ersten Matrix muss also identisch sein zur Anzahl der Zeilen der zweiten Matrix. Die Ergebnismatrix hat dann so viele Zeilen wie die erste Matrix und so viele Spalten wie die zweite Matrix. Der Matrixeintrag an der Stelle i, j berechnet sich als das Skalarprodukt der i -ten Zeile der ersten Matrix $A_{i,*}$ mit der j -ten Spalte der zweiten Matrix $B_{*,j}$: $\mathbf{C} = \mathbf{A} \cdot \mathbf{B}$ mit

$$C_{i,j} = A_{i,*} \cdot B_{*,j}$$

für $i \in m$ und $j \in l$.

Die Multiplikation zwischen einer Matrix \mathbf{M} und einem Vektor \mathbf{v} ist ein Sonderfall dieser Berechnungsvorschrift mit $l = 1$.

Übung 2.1 Berechnen Sie das Produkt der Matrizen $\mathbf{A} = \begin{pmatrix} 2 & 1 \\ 5 & 3 \end{pmatrix}$ und $\mathbf{B} = \begin{pmatrix} -1 & 1 \\ -2 & 2 \end{pmatrix}$, sowie das Produkt der Matrix \mathbf{A} mit dem Vektor $\mathbf{v} = \begin{pmatrix} 2 \\ 3 \end{pmatrix}$

2.1.3 Inverse

Kommen wir zurück zu dem Problem mit den und . Die Beschreibung des Problems mit Hilfe einer Matrix ist natürlich nur der erste Schritt. Jetzt soll noch bestimmt werden, wie viele Fische und Äpfel es denn wirklich sind, die Lotte isst. Dazu lösen wir das Gleichungssystem:

$$\mathbf{Ax} = \mathbf{b}$$

nach \mathbf{x} auf:

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}.$$

\mathbf{A}^{-1} ist die inverse Matrix oder einfach die Inverse von \mathbf{A} .

Für quadratische Matrizen $\mathbf{M} \in \mathbb{R}^{n \times n}$ gibt es (unter gewissen Voraussetzungen) eine inverse Matrix \mathbf{M}^{-1} . Diese hat folgende Eigenschaft: $\mathbf{M} \cdot \mathbf{M}^{-1} = I = \mathbf{M}^{-1} \cdot \mathbf{M}$. Die Matrix \mathbf{I} ist die Identitätsmatrix. Diese hat den Wert 1 überall auf der Hauptdiagonalen und ansonsten überall den Wert 0. Allgemein ist es nicht trivial, die Inverse einer Matrix zu berechnen. Im 2D gilt folgende Berechnungsvorschrift:

$$\mathbf{M}^{-1} = \frac{1}{\det(\mathbf{M})} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$$

und die Determinante

$$\det(\mathbf{M}) = ad - bc$$

für jede Matrix

$$\mathbf{M} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}.$$

bei der die Determinante nicht den Wert 0 hat.

Die Matrix

$$\begin{pmatrix} 1 & 1 \\ 1 & -2 \end{pmatrix}$$

hat die Determinante $-2 - 1 = -3$ und damit die Inverse

$$\frac{1}{3} \begin{pmatrix} 2 & 1 \\ 1 & -1 \end{pmatrix}.$$

Folglich ist

$$\begin{pmatrix} \text{fish} \\ \text{apple} \end{pmatrix} = \frac{1}{3} \begin{pmatrix} 2 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 18 \\ 0 \end{pmatrix} = \begin{pmatrix} 12 \\ 6 \end{pmatrix}.$$

Übung 2.2 Berechnen Sie die inverse Matrix für die Matrix $\mathbf{M} = \begin{pmatrix} 2 & 1 \\ 5 & 3 \end{pmatrix}$.

Matrizen repräsentieren affine Transformationen. Wir betrachten hier Skalierung und Rotation.

2.1.4 Skalierungsmatrix

Bei einer Skalierungsmatrix stehen die Skalierungsfaktoren für die jeweiligen Dimensionsrichtungen auf der Hauptdiagonalen. Die restlichen Einträge der Skalierungsmatrix sind 0.

$$\mathbf{T} = \begin{pmatrix} a & 0 \\ 0 & b \end{pmatrix}.$$

Eine Skalierungsmatrix im 2D, die keine Skalierung in x -Richtung vornimmt und in y -Richtung um den Faktor 2 skaliert, sieht entsprechend so aus (Abbildung 2.1):

$$\mathbf{T}_{1,2} = \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix}.$$

Einen transformieren Vektor \vec{v}' erhält man als Ergebnis der Berechnung

$$\vec{v}' = \mathbf{T}_{1,2} \cdot \vec{v}.$$

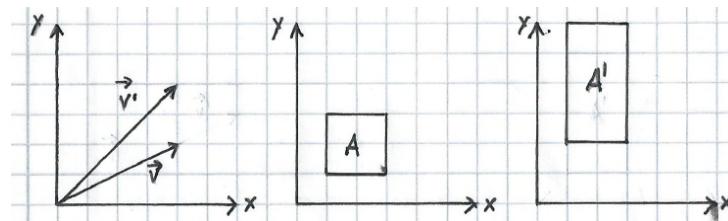


Abbildung 2.1: Skalierung.

Übung 2.3 Geben Sie eine Skalierungsmatrix im 2D an, die in x -Richtung mit dem Faktor 0.5 staucht und in y -Richtung mit dem Faktor 3 streckt. Skalieren Sie testweise mit der Matrix den Vektor $\begin{pmatrix} 2 \\ 2 \end{pmatrix}$.

2.1.5 Rotationsmatrix

Eine Rotation im 2D um den Winkel ϕ wird durch folgende Matrix beschrieben:

$$\mathbf{R} = \begin{pmatrix} \cos \phi & -\sin \phi \\ \sin \phi & \cos \phi \end{pmatrix}$$

Die Rotationsmatrizen im 3D lauten:

$$\begin{aligned}\mathbf{R}_x &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos \phi & -\sin \phi \\ 0 & \sin \phi & \cos \phi \end{pmatrix}, \\ \mathbf{R}_y &= \begin{pmatrix} \cos \phi & 0 & \sin \phi \\ 0 & 1 & 0 \\ -\sin \phi & 0 & \cos \phi \end{pmatrix}, \\ \mathbf{R}_z &= \begin{pmatrix} \cos \phi & -\sin \phi & 0 \\ \sin \phi & \cos \phi & 0 \\ 0 & 0 & 1 \end{pmatrix}\end{aligned}$$

Die Herleitung dieser Matrizen ergibt sich aus der Erzeugung passender Transformationsmatrizen. Dies wird im nächsten Kapitel besprochen.

Übung 2.4 Erstellen Sie eine 2D-Rotationsmatrix, die eine Rotation um 45° gegen den Uhrzeigersinn darstellt. Rotieren Sie damit die Einheitsvektoren der Koordinatenachsen im kartesischen Koordinatensystem.

3 Transformationen

Literatur

- Model-/View- und Projektionsmatrizen: <http://www.opengl-tutorial.org/beginners-tutorials/tutorial-3-matrices/>, abgerufen am 30.10.2017
- Tracking: [DBGJ13]

3.1 Mathematische Grundlagen: Koordinatensysteme

3.1.1 Homogene Koordinaten

Über reguläre Matrizen lassen sich nur affine Transformationen darstellen, also z.B. Skalierung, Scherung oder Rotation. Bei der Definition von Koordinatensystemen in der Computergrafik benötigt man aber häufig die Kombination aus einer Rotation und einer Translation (Verschieben). Translationen sind aber keine affinen Transformationen. Eine vollständige solche Transformation hat daher folgendes Aussehen:

$$T : \mathbf{R} \cdot \vec{x} + \vec{t}$$

mit $\mathbf{R} \in \mathbb{R}^{3 \times 3}$ und $\vec{x}, \vec{t} \in \mathbb{R}^3$. Der Nachteil dieser Darstellung ist, dass für die Repräsentation einer Transformation zwei Komponenten (Rotationsmatrix und Translationsvektor) benötigt werden. Für die Auswertung der Transformation benötigt man außerdem zwei Operationen: Matrix-Vektor-Multiplikation und Vektor-Vektor-Addition.

Es gibt aber eine Möglichkeit, affine Transformationen und Translationen in einer Matrix zusammenzufassen: durch die Verwendung homogener Koordinaten. Praktisch erweitert man alle Vektoren und Matrizen um eine Zeile (Matrizen und Vektoren) und eine Spalte (Matrizen). Aus 2×2 -Matrizen im 2D werden also 3×3 -Matrizen und aus 2-dimensionalen Vektoren 3-dimensionale Vektoren. Die zusätzliche Komponente bei den Vektoren nennt man w -Komponente:

$$\begin{pmatrix} x \\ y \end{pmatrix} \rightarrow \begin{pmatrix} x \\ y \\ w \end{pmatrix}$$

Beim Übergang von regulären Vektoren in ihre entsprechenden homogenen Pendants muss der Wert der w -Koordinate 1 sein. Ist das nicht der Fall, dann teilt man den gesamten Vektor durch w (wir betrachten homogene Vektoren also als invariant unter Skalierungen):

$$\begin{pmatrix} x \\ y \\ w \end{pmatrix} = \begin{pmatrix} \frac{x}{w} \\ \frac{y}{w} \\ 1 \end{pmatrix}$$

Die zusätzlichen Werte der homogenen Matrizen werden alle mit 0 aufgefüllt, außer der Wert auf der Hauptdiagonalen. Dieser wird 1:

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \rightarrow \begin{pmatrix} a & b & 0 \\ c & d & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Homogene Koordinaten erlauben eine Reihe von eleganten Berechnungen und Repräsentationen. Wir betrachten hier nur die Kodierung von Translationen in einer Matrix. Dazu werden die Translationsanteile einfach in die letzte Spalte eingetragen. Im 2D lautet eine Translationsmatrix um den Vektor $\begin{pmatrix} t_x \\ t_y \end{pmatrix}$ also

$$\begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix}$$

Zur Überprüfung verschieben wir den Vektor $\begin{pmatrix} x \\ y \end{pmatrix}$ auf zwei Arten:

$$\begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix} = \begin{pmatrix} x + t_x \\ y + t_y \end{pmatrix}$$

und mit homogenen Koordinaten:

$$\begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x + t_x \\ y + t_y \\ 1 \end{pmatrix}.$$

Übung 3.1 Geben Sie eine homogene Transformationsmatrix im 2D an, die Vektoren um den Winkel $\phi = 45^\circ$ gegen den Uhrzeigersinn dreht und dann um den Vektor $\begin{pmatrix} 1 \\ 2 \end{pmatrix}$ verschiebt.

3.1.2 Basisvektoren

Vektoren 'leben' jeweils in einem Koordinatensystem. Jedes Koordinatensystem hat Koordinatenachsen. Diese sind durch Basisvektoren festgelegt. Das grundlegende Koordinatensystem ist das Kartesische Koordinatensystem. Es hat kanonische Basisvektoren. Im 2D sind das

$$\vec{e}_0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \vec{e}_1 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

Abbildung 3.1 zeigt das Kartesische Koordinatensystem im 2D und 3D.

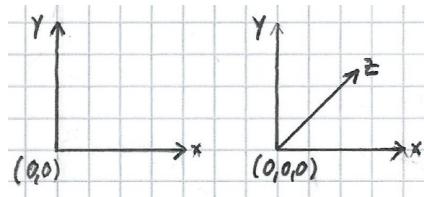


Abbildung 3.1: Kartesisches Koordinatensystem im 2D (links) und 3D (rechts). Hinweis: In Skizzen verwendet ich oft ein linkshändiges Koordinatensystem (x rechts, y oben und z hinten). In OpenGL arbeiten wir aber mit einem rechtshändigen Koordinatensystem.

Ein Koordinatensystem spannt einen Vektorraum auf. Dies geschieht über die Vektoren, die die Achsen repräsentieren. Beim Kartesischen Koordinatensystem zeigen diese Vektoren in die Hauptrichtungen (x, y, \dots) und haben die Einheitslänge 1. Solche aufspannenden Vektoren nennt man Basisvektoren. Mehrere Basisvektoren zusammen bilden eine Basis für den Vektorraum. Das Konzept der Basisvektoren wird später noch einmal bei der Transformation zwischen verschiedenen Koordinatensystem aufgegriffen.

Basisvektoren lassen sich miteinander kombinieren, um Punkte im Vektorraum zu erreichen. Dies nennt man Linearkombinationen. Ein 2D-Vektor $\vec{v} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$ bedeutet also eigentlich:

$$1 \cdot \vec{e}_0 + 2 \cdot \vec{e}_1.$$

Damit ergibt sich wieder der Punkt $\begin{pmatrix} 1 \\ 2 \end{pmatrix}$.

Es sind aber auch andere Koordinatensysteme und damit Basisvektoren denkbar. Braucht man etwa einen 2D-Raum, in dem alle y -Koordinaten mit den Faktor 0.5 gestaucht sind, dann lauten die Basisvektoren

$$\vec{b}_0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \vec{b}_1 = \begin{pmatrix} 0 \\ 0.5 \end{pmatrix}.$$

Der Vektor $\vec{v} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$ gibt dann einen anderen Punkt an, nämlich

$$\begin{aligned} & 1 \cdot \vec{b}_0 + 2 \cdot \vec{b}_1 \\ &= 1 \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} + 2 \cdot \begin{pmatrix} 0 \\ 0.5 \end{pmatrix} \\ &= \begin{pmatrix} 1 \\ 1 \end{pmatrix}. \end{aligned}$$

Übung 3.2 Geben Sie eine Basis im 2D mit den folgenden Eigenschaften an:

- die Länge der Basisvektoren ist jeweils $\sqrt{2}$.
- ein Basisvektor zeigt nach 'Nord-Osten'
- die Basisvektoren stehen senkrecht aufeinander

Allgemein gilt also

$$\vec{p} = \sum_{i=0}^{n-1} v_i \cdot \vec{b}_i$$

mit $v_i \in \mathbb{R}$ und $\vec{b}_i \in \mathbb{R}^3$ für die Dimension n .

Es spielt also eine große Rolle, in welchem Koordinatensystem man sich befindet und welche Basisvektoren dort gelten. Dieser Gedanke wird bei der Transformation zwischen verschiedenen Koordinatensystemen später noch einmal aufgegriffen.

3.1.3 Transformationsmatrizen

Nun machen wir uns folgenden Zusammenhang zunutze, der verwendet werden kann, um beliebige Transformationsmatrizen zu erzeugen: *Die Bilder der Basisvektoren sind die Spalten der Transformationsmatrix*. Die Basisvektoren (siehe Abschnitt 3.1) sind hier die Einheitsvektoren der Koordinatenachsen und deren Bilder haben wir eben hergeleitet.

3.1.4 Rotationsmatrix

Um Rotationsmatrizen herzuleiten, betrachten wir zunächst wieder den 2D-Fall. Insbesondere betrachtet man, wie sich die Einheitsvektoren der Koordinatenachsen im kartesischen Koordinatensystem verändern, wenn man eine Rotation um einen Winkel ϕ gegen den Uhrzeigersinn anwendet (siehe Abbildung 3.2):

$$\begin{pmatrix} 1 \\ 0 \end{pmatrix} \rightarrow \begin{pmatrix} \cos \phi \\ \sin \phi \end{pmatrix}$$

und

$$\begin{pmatrix} 0 \\ 1 \end{pmatrix} \rightarrow \begin{pmatrix} -\sin \phi \\ \cos \phi \end{pmatrix}$$

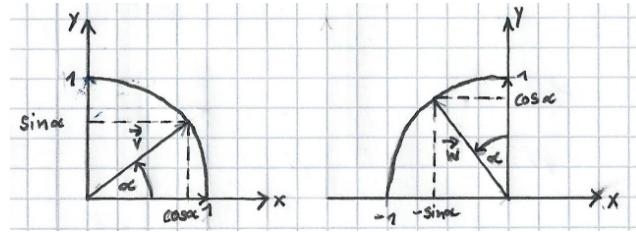


Abbildung 3.2: Rotation der Einheits-Koordinatenachsenvektoren im 2D gegen den Uhrzeigersinn um den Winkel ϕ .

Damit ergibt sich folgende Transformationsmatrix:

$$\mathbf{R} = \begin{pmatrix} \cos \phi & -\sin \phi \\ \sin \phi & \cos \phi \end{pmatrix}$$

Die Herleitung im 3D ergibt sich analog. Es ergeben sich dann drei Rotationsmatrizen für die Rotationen um die x -, y - und z -Achse.

3.1.5 Transformation zwischen beliebigen Koordinatensystemen

Um von einem beliebigen Koordinatensystem K_1 mit Basisvektoren $(\vec{x}_1, \vec{y}_1, \dots)$ in ein anderes Koordinatensystem K_2 mit Basisvektoren $(\vec{x}_2, \vec{y}_2, \dots)$ zu transformieren, geht man einfach den Umweg über das Kartesische Koordinatensystem. Schreibt man die Basisvektoren des ersten Koordinatensystems in die Spalten einer Matrix T_1 , so erhält man eine Transformationsmatrix vom Koordinatensystem K_1 in das Kartesischen Koordinatensystem. Das gleiche gilt für das Koordinatensystem K_2 und dessen Transformationsmatrix T_2 . Um nun von K_1 nach K_2 zu transformieren, geht man folgenden Weg:

$$K_1 \rightarrow K_{\text{Kartesisch}} \rightarrow K_2$$

Die Transformationen von K_1 in $K_{\text{Kartesisch}}$ ist einfach T_1 . Um also einen Vektor aus dem Koordinatensystem K_1 in das Koordinatensystem K_2 zu transformieren, berechnet man

$$\vec{v}^{K_2} = T_2^{-1} \cdot T_1 \cdot \vec{v}^{K_1}.$$

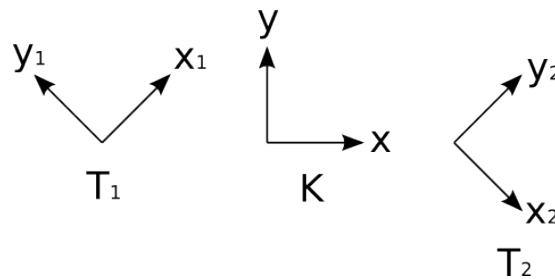


Abbildung 3.3: Transformation zwischen Koordinatensystemen.

Beispiel 1 Abbildung 3.3 zeigt zwei Koordinatensysteme T_1 und T_2 und das kartesische Koordinatensystem K im 2D. Die Basisvektoren lauten:

- $K : x = (1, 0), y = (0, 1)$
- $T_1 : x_1 = (1, 1), y_1 = (-1, 1)$
- $T_2 : x_2 = (1, -1), y_2 = (1, 1)$

Demnach lauten die Transformationsmatrizen:

$$\bullet K = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

$$\bullet T_1 : \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}$$

$$\bullet T_2 : \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}$$

Betrachten wir jetzt den Vektor $v_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ im Koordinatensystem T_1 . Die Koordinaten bedeuten, dass der erste Basisvektor x_1 mit 1 multipliziert wird und der zweite Basisvektor y_1 ebenfalls mit 1 multipliziert wird. Der vollständige Vektor ergibt sich als Summe der beiden Ergebnisse. Um den Vektor in das kartesische Koordinatensystem zu transformieren multipliziert man einfach T_1 gegen den Vektor:

$$v_K = T_1 v_1 = \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 2 \end{pmatrix}$$

Für die Transformation des Vektors v_1 in das Koordinatensystem T_2 benötigt man die Inverse von T_2 :

$$T_2^{-1} = \frac{1}{2} \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}$$

Der transformierte Vektor v_2 im Koordinatensystem T_2 ist also:

$$T_2^{-1} T_1 v_1 = \frac{1}{2} \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$$

zur Kontrolle können wir v_2 wieder in das kartesische Koordinatensystem transformieren, also

$$v_k = T_2 v_2 = \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix} \begin{pmatrix} -1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 2 \end{pmatrix}$$

Es gibt viele Anwendungen für die Transformation von einem Koordinatensystem in ein anderes. Exemplarisch sei hier die Transformation eines Assets in einem Videospiel von seinem lokalen (Modellierungs-)Koordinatensystem $T_{Krieger}$ in das Spiel- oder Weltkoordinatensystem T_{Welt} genannt (siehe Abbildung 3.4).

Übung 3.3 Gegeben ist die Matrix A mit der man Vektoren vom kartesischen Koordinatensystem in ein Koordinatensystem T_1 transformieren kann. Bestimmen Sie die Basisvektoren x_1 und x_2 von T .

$$A = \frac{1}{14} \begin{pmatrix} 3 & -2 \\ 1 & 4 \end{pmatrix}$$

Lösung 2 $A = T_1^{-1}$, also $T_1 = A^{-1} = \frac{1}{14} \begin{pmatrix} 4 & 2 \\ 1 & 3 \end{pmatrix}$. Daher: $x_1 = \begin{pmatrix} 4 \\ -1 \end{pmatrix}$ und $y_1 = \begin{pmatrix} 2 \\ 3 \end{pmatrix}$

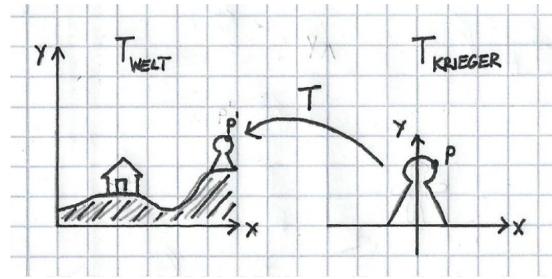


Abbildung 3.4: Transformation zwischen verschiedenen Koordinatensystemen.

Übung 3.4 Gegeben Sie zwei Koordinatensysteme K_1 (Basisvektoren $\begin{pmatrix} 1 \\ 1 \end{pmatrix}$ und $\begin{pmatrix} 2 \\ 3 \end{pmatrix}$) und K_2 (Basisvektoren $\begin{pmatrix} 1 \\ 0.5 \end{pmatrix}$ und $\begin{pmatrix} 0.5 \\ 1 \end{pmatrix}$). Geben Sie eine Matrix M an, die einen beliebigen Vektor \vec{v} aus dem Koordinatensystem K_1 nach K_2 transformiert.

Übung 3.5 A camera (in 2D) is located at $(3, 3)$ and looks at a reference point $(4, 4)$. In its (normalized) coordinate system, it sees a point p_c at $(0.25, 1)$. Compute the coordinates of the point p in the global (world) coordinate system (siehe Abbildung 3.5).

Hints: Determine the (normalized) basis vectors x_c and y_c of the camera determine the transformation matrix from the camera coordinate system into the Cartesian coordinate system transform p_c .

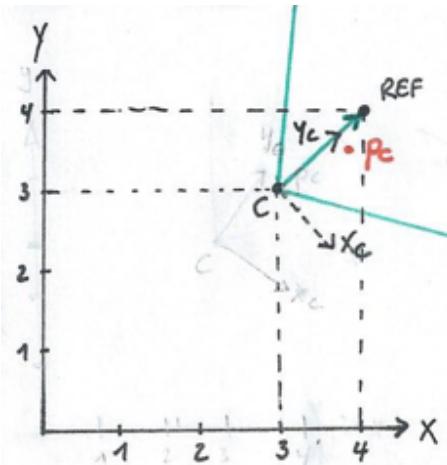


Abbildung 3.5: Übung: Kamerakoordinatensysteme.

$$\text{Lösung 3 } x_c = \frac{1}{\sqrt{2}}(1, -1) // \text{perpendicular to } y_c$$

$$y_c = \text{normalize}(ref - eye) = \frac{1}{\sqrt{2}}(1, 1)$$

$$\text{translation} = eye = (3, 3)$$

$$T = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 3 \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 3 \\ 0 & 0 & 1 \end{pmatrix}$$

$$p = T \cdot p_c = T \cdot (0.25, 1, 1) \approx (3.88, 3.53, 1)$$

3.2 Kamera-Transformation

Eine der zentralen Aufgabe der Graphics Processing Unit (GPU), also der Grafikkarte, ist es, aus einer dreidimensionalen Szene, ein Bild zu generieren. Dazu müssen insbesondere Punkte aus der 3D Welt (also aus Weltkoordinaten) in Bildkoordinaten umgerechnet werden. Diese Umrechnung geschieht im Rahmen der Rendering Pipeline der Grafikkarte. Einer der zentralen Schritte ist dabei die Model-View-Projektion. Diese Transformation wird durch eine (homogene) 4×4 Matrix beschrieben, die sich aus drei Teilmatrizen zusammensetzt. Diese Zusammensetzung wird hier hergeleitet.

3.2.1 Model-Transformation

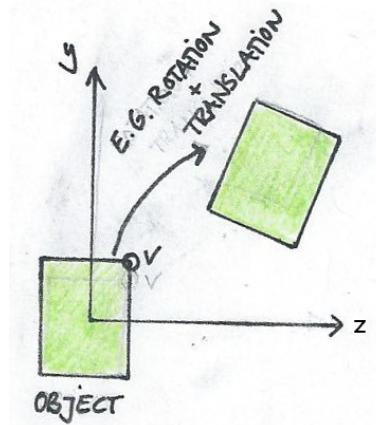


Abbildung 3.6: Model-Transformation.

Der erste Schritt wird durch die Model-Matrix beschrieben. Diese lässt sich am besten durch den bereits eingeführten Szenengraph beschreiben. Zunächst befinden sich die Objekte der Szene in ihrer Ursprungslage. Diese wird meist durch den Erzeuger der 3D-Modelle festgelegt, also beispielsweise einen Modellierer. Im Weltkoordinatensystem der Szene werden die Objekte aber meist transformiert (z.B. verschoben, rotiert oder skaliert). Im Szenengraph werden diese Transformationen repräsentiert, die aber im Kern lediglich passende Matrizen kapseln. Abbildung 3.6 zeigt den Prozess der Transformation eines Objektes (grün) von Objektkoordinaten (um den Ursprung) in Weltkoordinaten.

Neben den Objekten der Szene spielt die virtuelle Kamera eine zentrale Rolle. Wir beschreiben die Pose der Kamera durch den Augpunkt (eye) einen Referenzpunkt (ref) und einen Oben-Vektor (up). In der hier zur Veranschaulichung gewählten 2D-Darstellung spielt der Up-Vektor keine Rolle. Die Kamera hat zwei weitere zentrale Eigenschaften: die *near* und die *far*-Ebene. Diese geben die Grenzen der Sichtbarkeit an. Die *near* Ebene schneidet dabei Szenenobjekte weg, die zu nah (oder gar hinter dem) Augpunkt sind, die *far* Ebene stellt eine Grenze in der Ferne da. Es ergibt sich damit das Sichtbarkeitsvolumen, das mit gelb markiert ist. Wir betrachten hier insbesondere immer einen Punkt der Szene (\mathbf{v}). Es werden aber alle Punkte (Vertices) der Szene exakt gleich behandelt, die Transformation ist immer dieselbe.

Um den Punkt \mathbf{v} von seinen Objektkoordinaten in die Weltkoordinaten zu transformieren, multipliziert man den Punkt mit der Model-Matrix \mathbf{M} :

$$\mathbf{v}' = \mathbf{M}\mathbf{v}$$

Die Matrix \mathbf{M} ergibt sich direkt aus den Transformationen, die zur Positionierung des Objektes in der Szene verwendet wurden.

Hinweis: Im Framework für das Praktikum wird die Model-Matrix im Wurzelknoten der Szene mit der Identität initialisiert:

`getRoot().traverse(RenderMode.REGULAR, Matrix.createIdentityMatrix4());` Transformationsknoten im Szenengraph verändern die Model-Matrix. Ein Translationsknoten etwa multipliziert eine homogene Matrix, die eine Translation repräsentiert, gegen den alte Model-Matrix:

```
translation = Matrix.createTranslationMatrixGl(translation);
super.traverse(mode, modelMatrix.multiply(translation));
```

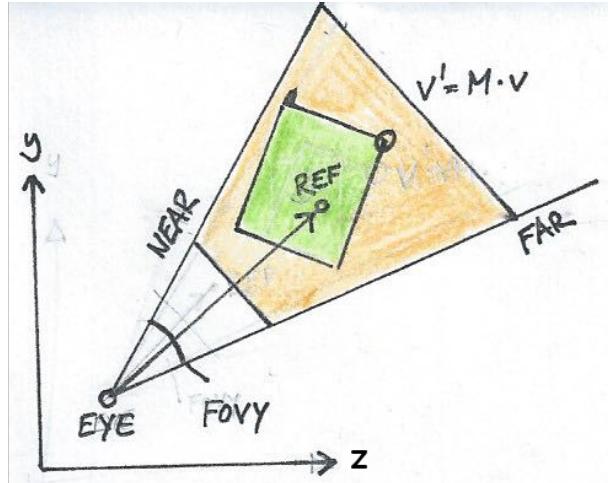


Abbildung 3.7: Model-Transformation und virtuelle Kamera.

Beispiel: Wir betrachten ein Beispiel für das wir den gesamten Transformationsablauf von Objektkoordinaten bis zu Pixelkoordinaten im Ergebnisbild nachvollziehen. Die Szene ist in Abbildung 3.8 dargestellt.

Wir beginnen mit einem Punkt des Objektes in Ruhelage: $p = (-1, 1, 0)$. Dies entspricht dem Punkt $p = (-1, 1, 0, 1)$ in homogenen Koordinaten. Der erste Schritt der Transformation ist die Transformation in Weltkoordinaten durch die Model-Matrix. In diesem Beispiel setzt sich die Modelmatrix aus einer Translationkomponenten (Verschieben um $(5, 3, 0)$) und einer Rotationskomponente (um -45° um die z-Achse) zusammen. \mathbf{M} ist also:

$$\mathbf{M} = \begin{pmatrix} \cos(-45) & -\sin(-45) & 0 & 5 \\ \sin(-45) & \cos(-45) & 0 & 3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Für die Transformation in Weltkoordinaten ergibt sich:

$$\mathbf{p}_{\text{welt}} = \mathbf{M} \cdot \mathbf{p} = (5, 4.414, 0, 1)$$

3.2.2 View-Transformation

Im zweiten Schritt wird die View-Transformation durchgeführt. Dazu wird die Blickrichtung der Kamera auf die Richtung der z -Achse überführt. Nach Anwendung dieser Transformation blickt die virtuelle Kamera also entlang der z -Achse. Es wird aber in der Praxis nicht die Kamera transformiert, sondern es werden wieder die Objekte der Szene transformiert. Der Augpunkt der Kamera ist nach

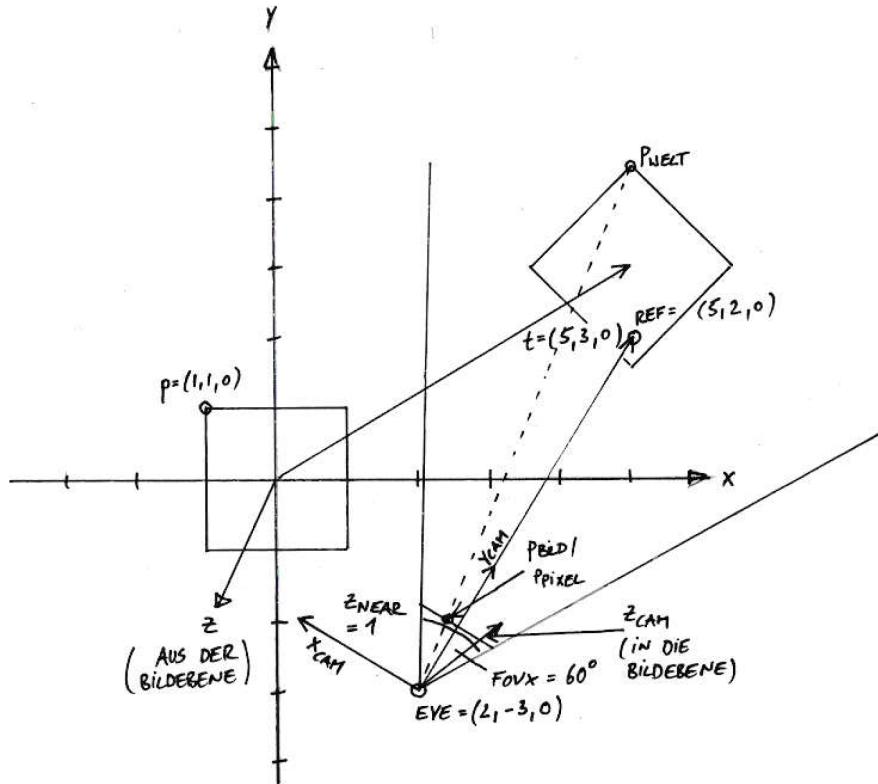


Abbildung 3.8: Beispiel: Transformation von Objektkoordinaten in Pixelkoordinaten.

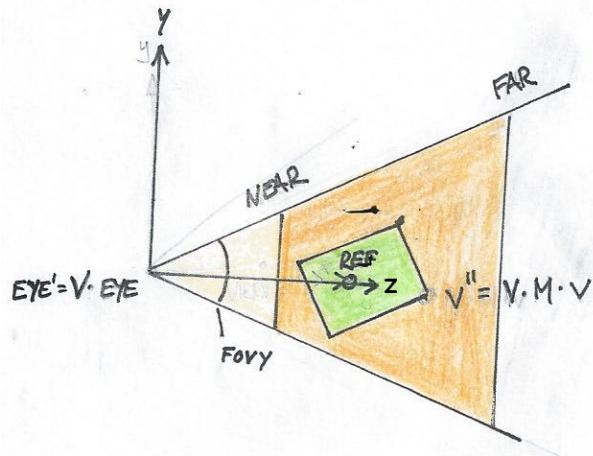


Abbildung 3.9: View-Transformation.

der Transformation im Ursprung zu finden, die Szenenobjekte sind entlang der z -Achse aufgereiht. Die View-Transformation wird durch die View-Matrix \mathbf{V} festgelegt. \mathbf{V} lässt sich direkt aus den extrinsischen Kameraparametern (eye , ref , up) bestimmen. Dazu bestimmt man aus den drei Vektoren ein orthonormales, rechthändiges Koordinatensystem, wobei die Blickrichtung ($\text{ref} - \text{eye}$) darin der z -Achse entspricht und der Ursprung im Augpunkt liegt, also:

$$\mathbf{z} = \frac{\mathbf{ref} - \mathbf{eye}}{\|\mathbf{ref} - \mathbf{eye}\|}$$

$$\mathbf{x} = \mathbf{up} \times \mathbf{z}$$

$$\mathbf{y} = \mathbf{z} \times \mathbf{x}$$

Die drei Koordinatenvektoren schreibt man als Spalten in die Transformationsmatrix \mathbf{v} und ergänzt den Augpunkt als Verschiebung:

$$\mathbf{V} = \begin{pmatrix} \mathbf{x} & \mathbf{y} & \mathbf{z} & \mathbf{eye} \\ 0 & 0 & 0 & 1 \end{pmatrix}^{-1}$$

Für die Transformation des Objektpunktes \mathbf{v} ergibt sich:

$$\mathbf{v}'' = \mathbf{VMv}$$

Nach der View-Transformation lassen sich die *near* und *far* Ebenen einfach durch die Entfernung (und damit z -) Werte beschreiben.

Hinweis: Im Praktikumsprojekt hängt der Aufbau der View-Matrix vom Anwendungsfall ab. Im Modul *opengl* wird die View-Matrix aus den extrinischen Kameraparametern generiert, die durch eine Touch-Steuerung angepasst werden können:

```
...
cam.setViewMatrixFromEyeRefUp();
```

Im Modul *vuforia* ergibt sich die View-Matrix direkt aus dem Kamera-Tracking:

```
Matrix44F vuforiaViewMatrix = Tool.convertPose2GLMatrix(result.getPose());
...
super.traverse(mode, modelMatrix.multiply(M));
```

Beispiel: Verfolgen wir weiter das Beispiel: hier ergibt sich: $\mathbf{eye} = (2, 3, 0)$, $\mathbf{ref} = (5, 2, 0)$, $\mathbf{up} = (0, 0, 1)$ und damit

$$\begin{aligned} \mathbf{z} &= \frac{(5, 2, 0) - (2, 3, 0)}{\|(5, 2, 0) - (2, 3, 0)\|} = (0.514, 0.857, 0) \\ \mathbf{x} &= (0, 0, 1) \times \mathbf{z} = (-0.857, 0.514, 0) \\ \mathbf{y} &= \mathbf{z} \times \mathbf{x} = (0, 0, 1) \end{aligned}$$

$$V = \begin{pmatrix} -0.857 & 0 & 0.514 & 2 \\ 0.514 & 0 & 0.857 & 3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} -0.857 & 0.514 & 0 & 3,258 \\ 0 & 0 & 1 & 0 \\ 0.514 & 0.857 & 0 & 1.543 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Damit berechnet sich der Punkt \mathbf{p} in Kamerakoordinaten zu:

$$\mathbf{p}_{\text{cam}} = \mathbf{V} \cdot \mathbf{p}_{\text{welt}} = (1.242, 0, 7.901, 1)$$

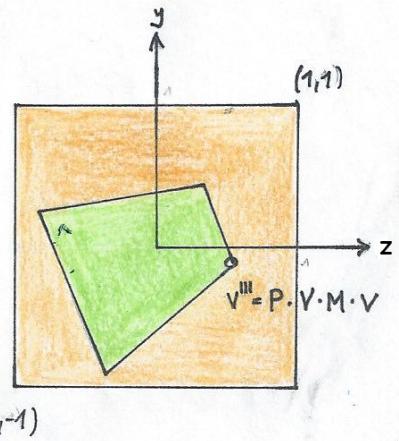


Abbildung 3.10: Perspektivische Transformation.

3.2.3 Perspektivische Transformation

Der dritte Schritt ist die perspektivische Transformation. Diese setzt sich aus zwei Komponenten zusammen. Der Projektion auf die Bildebene (man kann sich die Bildebene durch die *near* Ebene vorstellen) und der Skalierung auf den Einheitswürfel von $(-1, -1, -1)$ bis $(1, 1, 1)$. Das Volumen des Einheitswürfels entspricht dann genau dem Teil der Szene, der im Ergebnisbild (sieht man von Verdeckungen ab) sichtbar ist. Betrachten wir zunächst die Projektion:

Wir leiten die Projektion anhand des Stahlensatzes her (siehe Abbildung 3.11). Der Punkt $\mathbf{p} = (y, z)$ wird auf die Bildebene mit dem z -Wert z_0 (*near*-Ebene, also $z_0 = z_{near}$) projiziert. Dort hat er dann die projizierten Koordinaten $\mathbf{p}' = (y', z_0)$.

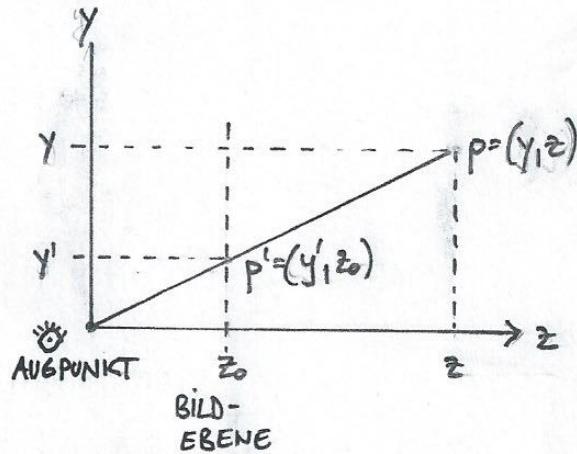


Abbildung 3.11: Projektion.

Nach dem Strahlensatz ergibt sich, dass

$$\frac{z}{z_0} = \frac{y}{y'},$$

also

$$y' = \frac{y z_0}{z}$$

Analog gilt für x' (im 3D):

$$x' = \frac{xz_0}{z}$$

Verwendet man eine homogene Darstellung, dann lässt sich dieser Zusammenhang in Matrix-Schreibweise formulieren:

$$\begin{pmatrix} x' \\ y' \\ z_0 \\ 1 \end{pmatrix} = \mathbf{P} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} z_0 & 0 & 0 & 0 \\ 0 & z_0 & 0 & 0 \\ 0 & 0 & z_0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix},$$

da

$$\begin{pmatrix} x'z \\ y'z \\ z_0z \\ z \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ z_0 \\ 1 \end{pmatrix}.$$

Wir nennen die Matrix \mathbf{P} Zentralprojektionsmatrix.

Beispiel: Wenden wir die perspektivische Projektion auf unser Beispiel an, dann ergibt sich für eine Near-Clipping-Entfernung $z_{near} = z_0 = 1$:

$$\mathbf{P} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Damit ergibt sich für den Punkt \mathbf{p} im Bildkoordinatensystem:

$$\mathbf{p}_{\text{bild}} = \mathbf{P} \cdot \mathbf{p}_{\text{cam}} = (1.242, 0, 7.901, 7.901)$$

Achtung: hier muss man noch die perspektivische Division, also die Division durch die w -Koordinate durchführen, und erhält:

$$\mathbf{p}_{\text{bild}} = (0.157, 0, 1, 1)$$

3.2.4 Pixel

Der letzten schritt ist die Umrechnung der Bildschirmkoordinaten in die Pixel-Koordinaten des Bildschirms. Also Information dafür steht die Auflösung des Bildschirms bereit: Breite $w \times$ Höhe h . Das Verhältnis zwischen Breite und Höhe bezeichnen wir als Aspekt: $aspekt = \frac{w}{h}$. Außerdem benötigt man den Öffnungswinkel. Üblicherweise wird der Öffnungswinkel in y-Richtung (oben-unten) angegeben: $fovy$. Den Öffnungswinkel in x-Richtung heisst fov_x . Der Pixel mit den Koordinaten $(0, 0)$ befindet sich in der linken untere Ecke, die x-Koordinaten steigen von links nach rechts, die y-Koordinaten steigen von unten nach oben.

Damit ergibt sich die folgende Matrix \mathbf{K} :

$$\mathbf{K} = \begin{pmatrix} f & 0 & 0 & w/2 \\ 0 & f & 0 & h/2 \end{pmatrix}$$

Dazu benötigen wir demnach noch die fokale Länge f (siehe Abbildung 3.12).

$$f_y = \frac{h}{2 \cdot \tan(\frac{fovy}{2})}$$

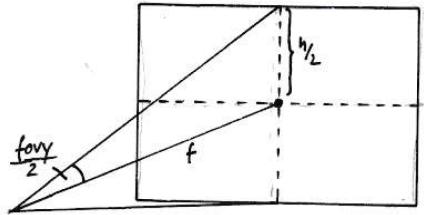


Abbildung 3.12: Bestimmung der fokalen Distanz (*focal distance*) aus der Höhe der Bildebene h in Pixeln und des vertikalen Öffnungswinkels $fovy$.

Oder, falls der Öffnungswinkel in x -Richtung gegeben ist:

$$f_x = \frac{w}{2 \cdot \tan(\frac{fovx}{2})}$$

Normalerweise gilt: $f = f_x = f_y$.

Beispiel Nehmen wir an, dass wir einen Bildschirm mit einer Auflösung von 1024×768 verwenden und die Kamera einen horizontalen Öffnungswinkel von $fovx = 60^\circ$ hat. Dann ergibt sich eine Matrix

$$\mathbf{K} = \begin{pmatrix} 886.810 & 0 & 0 & 512 \\ 0 & 886.810 & 0 & 384 \end{pmatrix}$$

Damit ergibt sich für den Punkt \mathbf{p} auf der Bildebene der Pixel

$$\mathbf{p}_{\text{pixel}} = \mathbf{K} \cdot \mathbf{p}_{\text{bild}} = (651.412, 384)$$

Übung 3.6 In folgendem Koordinatensystem ist eine Kamera im Ursprung mit Blickrichtung entlang der z -Achse. Die near-Ebene liegt bei $z_{near} = 1$ und die far-Ebene liegt bei $z_{far} = 5$ (siehe Abbildung 3.14). Der Punkt $p = (3, 3)$ soll nun perspektivisch transformiert werden. Das View-Volumen ist bei $y = -5/ + 5$ begrenzt (Clipping). Bestimmen Sie zunächst die Projektion auf die Bildebene z_{near} mit Zentralprojektion. Führen Sie nun noch die Transformation in den Einheitswürfel durch.

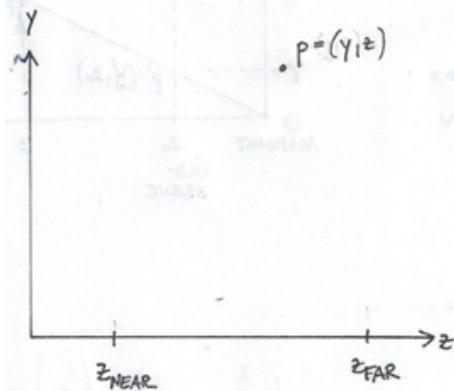


Abbildung 3.13: .

Lösung 4 $y' = y * z_{near} / z = 3 * 1 / 3 = 1$

Einheitswürfel:

$$z'' = 2 * (z - z_{NEAR}) / (z_{FAR} - z_{NEAR}) - 1$$

$$\begin{aligned}
 &= 2 * (3 - 1) / (5 - 1) - 1 = 4/4 - 1 = 0 \\
 y'' &= 2 * (y - y_{MIN}) / (z_{MAX} - y_{MIN}) - 1 \\
 &= 2 * (3 - (-5)) / (5 - (-5)) - 1 = 16/10 - 1 = 0.6
 \end{aligned}$$

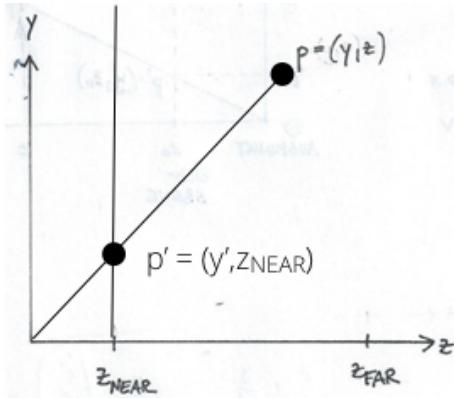


Abbildung 3.14: .

3.3 Tracking

Exercise: Tracking

We want to compute the pose of a marker in 2D. The marker has the corner points $c_1 = (3, 1)$ and $c_2 = (4, 2)$. The intrinsic camera matrix is:

$$K = \begin{pmatrix} 128 & 128 \\ 0 & 1 \end{pmatrix}$$

The marker corners are identified in the camera image plane at $x_1 = (150)$ und $x_2 = (100)$. We start with an initial guess: $eye = (4, 1)$, $y_c = (1, -1)$.

Question: What is the error in the optimization error function?

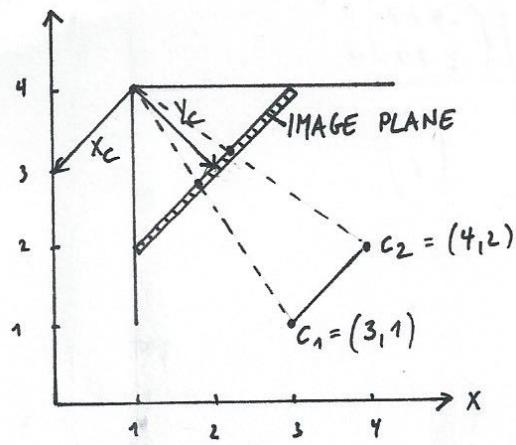


Abbildung 3.15: Übung: Marker Tracking.

Solution:

$$\begin{aligned}
T_{cam} &= \begin{pmatrix} -1 & 1 & 1 \\ -1 & -1 & 4 \\ 0 & 0 & 1 \end{pmatrix}, T_{cam}^{-1} = \frac{1}{2} \begin{pmatrix} -1 & -1 & 5 \\ 1 & -1 & 3 \\ 0 & 0 & 1 \end{pmatrix} \\
c_1^{cam} &= T_{cam}^{-1} c_1 = \begin{pmatrix} 0.5 \\ 2.5 \\ 0.5 \end{pmatrix}, c_2^{cam} = T_{cam}^{-1} c_2 = \begin{pmatrix} -0.5 \\ 2.5 \\ 0.5 \end{pmatrix} \\
c_1^{img} &= P c_1^{cam} = \begin{pmatrix} 0.5 \\ 2.5 \\ 2.5 \end{pmatrix} = \begin{pmatrix} 0.2 \\ 1 \\ 1 \end{pmatrix}, c_2^{img} = P c_2^{cam} = \begin{pmatrix} -0.5 \\ 2.5 \\ 2.5 \end{pmatrix} = \begin{pmatrix} -0.2 \\ 1 \\ 1 \end{pmatrix} \\
c_1^{pixel} &= K c_1^{x,1,img} = \begin{pmatrix} 153.6 \\ 1 \end{pmatrix}, c_2^{pixel} = K c_2^{x,1,img} = \begin{pmatrix} 102.4 \\ 1 \end{pmatrix} \\
f &= (150 - 153.6)^2 + (100 - 102.4)^2 = 18.72
\end{aligned}$$

4 Kurven

4.1 Mathematische Grundlagen: Basisfunktionen

Eine Basis muss nicht aus Vektoren bestehen. In anderen Aufgabenstellungen ist es sinnvoll, Funktionen - also Basisfunktionen - zu verwenden, um eine Basis zu definieren.

Betrachten wir ein lineares Interpolationsproblem: Gegeben sind zwei Werte v_0 und v_1 . Über einen Parameter λ aus dem Wertebereich $[0; 1]$ soll zwischen den beiden Werten interpoliert werden. Für $\lambda = 0$ soll den Werte v_0 angenommen werden und für $\lambda = 1$ soll der Wert v_1 angenommen werden. Dazwischen sollen es einen linearen Übergang zwischen v_0 und v_1 geben. In der Mitte ($\lambda = 0.5$) ergibt sich der Mittelwert aus v_0 und v_1 , also $\frac{v_0+v_1}{2}$.

Wie lässt sich dieses Interpolationsproblem als Funktion $f(\lambda)$ über dem Parameter λ beschreiben?

Eine Lösung ist es, für die beiden Werte v_0 und v_1 jeweils geeignete Basisfunktionen zu finden, nämlich $b_0(\lambda) = 1 - \lambda$ und $b_1(\lambda) = \lambda$. Für das Interpolationsproblem ergibt sich demnach:

$$\begin{aligned} f(\lambda) &= v_0 \cdot b_0(\lambda) + v_1 \cdot b_1(\lambda) \\ &= v_0 \cdot (1 - \lambda) + v_1 \cdot \lambda \end{aligned}$$

oder allgemein

$$f(\lambda) = \sum_{i=0}^{m-1} \vec{v}_i \cdot b_i(\lambda)$$

mit $v_i \in \mathbb{R}^n$ und $b_i(\lambda) \in \mathbb{R}$. Es kann also nicht nur mit Skalaren wie im vorherigen Beispiel sondern mit Punkten beliebiger Dimension gerechnet werden.

Übung 4.1 Wie lautet der interpolierte Wert zwischen v_0 und v_1 an der Stelle $\lambda = 0.5$, wenn man alternativ die folgenden Basisfunktionen verwendet: $b_0(\lambda) = 1 - \lambda^2$ und $b_1(\lambda) = \lambda^2$?

5 Beleuchtung und Texturen

Literaturangaben für dieses Kapitel: [Sha98], [Hec86]

6 Animation

6.1 Einführung

Animationen spielen in der Computergrafik und damit auch in Augmented Reality eine zentrale Rollen. Viele Objekte in der Realität bewegen sich dynamisch. Es gibt viele Ansätze, um diese Dynamik umzusetzen. In diesem Abschnitt beschäftigen wir uns nur mit einen sehr kleinen Ausschnitt aus dem Spektrum: Skelett-basierte Animationen. Diese werden meist bei der Animation von (Menschähnlichen) Charakteren eingesetzt, lassen sich aber auch in vielen anderen Fällen verwenden.

Bei der Verwendung von Animationen gibt es zwei Komponenten, die zusammengeführt werden. Zum einen die Festlegung der Animation im Vorfeld und dann die Darstellung und Verwendung der Animation in der Grafik-Anwendung.

Wir betrachten hier die Animation von Dreiecksnetzen. Vereinfachend nehmen wir an, dass sich die Topologie des Netzes nicht verändert. Das bedeutet, dass sich die Anzahl der Dreiecke im Laufe der Animation nicht verändert. Außerdem bleiben die Dreiecks-Indizes (für Vertices und Texturkoordinaten) konstant. Auch die Anzahl der Vertices bleibt bestehen, es können sich aber die Positionen der Vertices verändern. Die Aufgabe des Animationssystems ist es also, in jedem Frame die *richtigen* Position der Vertices zu bestimmen.

Naiv könnte man annehmen, dass man einfach in der Vorbereitung der Animation für jede Veränderung über die Zeit und für jeden Vertex die Position festlegt und abspeichert. Bei der Verwendung musste man dann zu jedem Zeitpunkt den passenden Satz von Vertexpositionen laden und darstellen. Diese Form der Animation fällt in den Bereich der *Keyframe-Animation*. Der Ansatz hat zwei zentrale Nachteile. Zum einen ist das Erstellen der Animationen sehr aufwändig, da die Positionen aller Vertices in jedem Zeitschritt (ggf. manuell) festgelegt werden müssen. Außerdem hat der Ansatz einen sehr hohen Speicheraufwand, da sehr viele unterschiedliche Vertex-Positionen gespeichert werden müssen. Die Frage nach effizienten Speicheransätzen ist ein aktives Forschungsgebiet → *Animationskompression*. Wir betrachten hier ein anderes Vorgehen.

6.2 Skelett und Knochen

Die Grundlagen zu diesem Abschnitt gehen zurück auf die Arbeiten in [MTLT88].

Eine Idee zur Überwindung der genannten Nachteile leitet sich auch der Natur ab. Die Bewegung eines Menschen lässt sich vereinfachend auch von seiner Oberfläche (Haut) auf die Bewegung seiner Knochen reduzieren. Tatsächlich beschränkt sich die Bewegung dann auf Rotationen an den Gelenken (engl. *joint*) entlang der starren Knochen (engl. *bone*). Diesen Ansatz machen wir uns hier zunutze. Anstelle der Animation des Dreiecksnetzes reduziert wird das Dreiecksnetz auf eine Skelettstruktur (engl. *skeleton*) und animieren das Skelett (= Festlegung der Rotationen an den Gelenken). Anschließend wird die Bewegung der Knochen auf das umschliessende Dreiecksnetz übertragen (siehe Abbildung 6.1).

Die Animation des Skeletts ist damit natürlich immernoch nicht gelöst. Wir betrachten das Problem genauer im Abschnitt 6.3. Zunächst beschäftigen wir uns mit der Frage, wie aus dem Skelett die Bewegung des Dreiecksnetzes abgeleitet wird. Zum Verständnis ist dazu wichtig, dass sowohl beim Skelett als auch bei dem umschliessenden Dreiecksnetz zwischen einer Ruhelage (engl. *rest state*) und der aktuellen Lage unterschieden wird. Als Eingabe zu dem Ansatz wird also das Skelett in Ruhelage

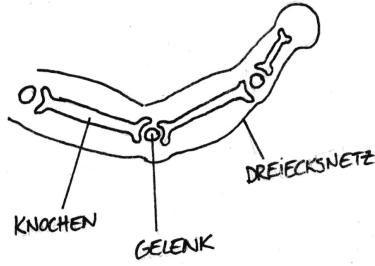


Abbildung 6.1: Skelett-basierte Animation mit einem Skelett aus Knochen, die durch Gelenke miteinander verbunden sind und vom Dreiecksnetz, das eigentlich animiert werden soll, umschlossen werden.

und mit Animation über die Zeit erwartet. Das Dreiecksnetz liegt nur in der Ruhelage vor. Dessen Lage im Laufe der Animation muss berechnet werden (siehe Abbildung 6.2).

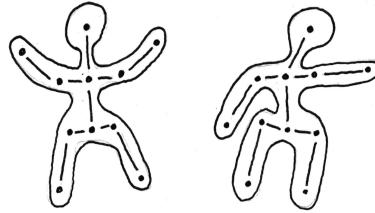


Abbildung 6.2: Ruhelage des Skeletts und des Dreiecksnetzes (links) und deformierte Lage im Laufe der Animation (rechts).

Ein Knochen hat einen Start- und einen Endpunkt. Mit jedem Knochen assoziieren wir ein Koordinatensystem. Dies setzt sich zum einen aus einer Rotation und zum anderen aus einer Translation zusammen. Auf die Position des Startpunktes hat der Knochen selber keinen Einfluss. Diese ergibt sich automatisch aus der Lage der Knochen, an denen der Knochen hängt. Die Rotation ist einfach die Rotation des Gelenkes an denen der Knochen beginnt. Die Position des Endpunktes ergibt sich aus der Startpunkt-Position und der Kombination aus der Knochen-Rotation und der Länge des Knochens, repräsentiert als Verschiebungsmatrix. Auch das Skelett hat damit als Aufbau eine Baumstruktur: Abbildung 6.3.

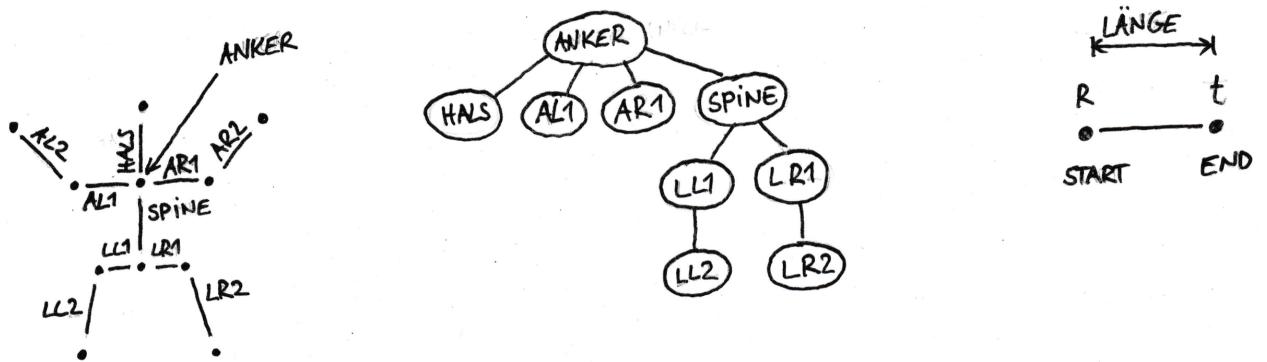


Abbildung 6.3: Skelett eines humanoiden Charakters mit benannten Knochen (links), dazugehörige Baumstruktur (Mitte), Knochen mit Koordinatensystem aus Rotation R am Start-Gelenk und Translation t zum End-Gelenk.

Um nun einen Vertex v zusammen mit einem Knochen B zu bewegen, verwenden wir das folgende

Vorgehen: Zunächst transformieren wir v in Ruhelage in das Koordinatensystem von B :

$$\delta = \mathbf{T}_B^{-1} \cdot \mathbf{v}$$

\mathbf{T}_B ist dabei die Transformation am Startpunkt von B in Ruhelage. Diese beinhaltet die kombinierten Transformationen aller Knochen weiter oben im Skelettbau (jeweils $\mathbf{R} \cdot \mathbf{T}_t$ mit homogener Translationsmatrix \mathbf{T}_t) und die Rotation von B . In Abbildung 6.3 wäre das beispielweise für den Knochen B^{LR2} :

$$\mathbf{T}_{LR2} = \mathbf{T}_{\text{Anker}} \cdot \mathbf{T}_{\text{Spine}} \cdot \mathbf{T}_{LR1} \cdot \mathbf{R}_{LR2} = \mathbf{T}_{\text{Anker}} \cdot \mathbf{R}_{\text{Spine}} \cdot \mathbf{T}_{t,\text{Spine}} \cdot \mathbf{R}_{LR1} \cdot \mathbf{T}_{t,LR1} \cdot \mathbf{R}_{LR2}$$

$\mathbf{T}_{\text{Anker}}$ ist dabei meist einfach die Translationsmatrix zur Position des Ankers.

Hat man nun die Koordinaten δ des Vertex \mathbf{v} im Koordinatensystem von B in Ruhelage bestimmt, lässt sich daraus einfach die transformierte Position \mathbf{v}' von \mathbf{v} für einen transformierten Knochen B' bestimmen:

$$\mathbf{v}' = \mathbf{T}_{B'} \cdot \delta$$

Hätte man nur einen Knochen, wäre das Problem damit gelöst: Alle Vertices \mathbf{v} des Dreiecksnetzes werden zu jedem Zeitschritt transformiert:

$$\mathbf{v}' = \mathbf{T}_{B'} \cdot \mathbf{T}_B^{-1} \cdot \mathbf{v}$$

In der Praxis hat man aber mehrere Knochen und es muss zunächst festgelegt werden, mit der Transformation welches Knochens ein Vertex bewegt werden muss. Wir betrachten dazu zwei Ansätze:

- nächster Knochen
- gewichtet über alle Knochen

6.2.1 Nächster Knochen

Der naheliegende Ansatz ist es, den Knochen zu verwenden, der dem Vertex am nächsten liegt. Wir müssen also den Abstand eines Vertex \mathbf{v} vom Knochen B bestimmen. Wir nehmen an, dass B den Startpunkt \mathbf{p} und den Endpunkt \mathbf{q} hat. Damit lässt sich eine Geradengleichung aufstellen:

$$g : \mathbf{p} + \lambda \cdot (\mathbf{q} - \mathbf{p})$$

Die Punkte des gesuchten Segments (also des Knochens) liegen im λ -Intervall $[0, 1]$. Oftmals normalisiert man den Richtungsvektor einer Geradengleichung:

$$g : \mathbf{p} + \lambda \cdot \frac{\mathbf{q} - \mathbf{p}}{\|\mathbf{q} - \mathbf{p}\|}$$

Die Punkte des gesuchten Segments liegen dann im λ -Intervall $[0, \|\mathbf{q} - \mathbf{p}\|]$. Für $\lambda = 0$ erreicht man den Startpunkt \mathbf{p} , für $\lambda = \|\mathbf{q} - \mathbf{p}\|$ erreicht man den Endpunkt \mathbf{q} .

Zunächst bestimmt man also den Abstand von der Geraden g (siehe nächster Unterabschnitt). Bei der Suche schränkt man aber λ auf den Bereich $[0, \|\mathbf{q} - \mathbf{p}\|]$ ein. Liegt also beispielsweise der nächste Punkt von \mathbf{v} zu g bei $\lambda < 0$, dann wird mit $\lambda = 0$ weitergearbeitet (analog für $\lambda > \|\mathbf{q} - \mathbf{p}\|$).

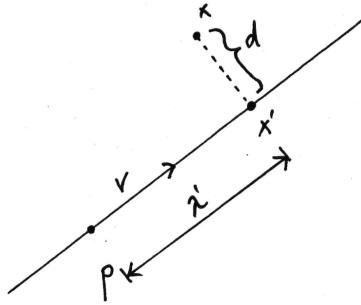


Abbildung 6.4: Berechnung des Abstandes d zwischen dem Punkt \mathbf{x} und der Geraden $g : \mathbf{p} + \lambda \cdot \mathbf{v}$ über die Projektion \mathbf{x}' mittels Skalarprodukt.

6.2.2 Einschub: Abstand Punkt-Gerade

[Siehe auch den Abschnitt zur Einführung des Skalarproduktes] Wir nehmen hier an, dass der Richtungsvektor \mathbf{v} der Geraden normiert ist: $\|\mathbf{v}\| = 1$. Zur Berechnung des Abstandes eines Punktes \mathbf{x} von einer Geraden $g : \mathbf{p} + \lambda \cdot \mathbf{v}$ bestimmt man zunächst den von \mathbf{x} lotrecht auf die Gerade projizierten Punkt \mathbf{x}' . Dazu macht man sich wieder das Skalarprodukt zunutze:

$$\lambda' = ((\mathbf{x} - \mathbf{p}) \cdot \mathbf{v})$$

und

$$\mathbf{x}' = \mathbf{p} + \lambda' \cdot \mathbf{v}$$

Der Abstand d ergibt sich dann direkt aus der Projektion \mathbf{x}' :

$$d = \|\mathbf{x} - \mathbf{x}'\|$$

6.2.3 Gewichtet über alle Knochen

Noch bessere Ergebnisse erzielt man aber, wenn man sich für jeden Vertex \mathbf{v} auf nicht nur einen Knochen festlegt. Stattdessen verwendet man besser alle Knochen (oder die nächsten n Knochen). Die jeweiligen Transformationsergebnisse \mathbf{v}' werden dann für jeden Knochen gewichtet. Je näher der Knochen am Vertex liegt (je kürzer der Abstand zwischen Knochen und \mathbf{v} ist), desto größer ist das Gewicht:

$$\mathbf{v}' = \frac{\sum_i \omega_i \cdot \mathbf{T}_{B'_i} \cdot \mathbf{T}_{B_i}^{-1} \cdot \mathbf{v}}{\sum_i \omega_i}$$

Diese Berechnung sieht komplizierter aus, als sie ist. Man transformiert den Vertex \mathbf{v} ganz genauso wie zuvor eingeführt. Der einzige Unterschied ist, dass man das nacheinander für alle Knochen B_i macht, die Ergebnisse jeweils mit den Gewicht $\omega_i = \omega(d_i)$ skaliert und aufeinander addiert. Am Ende muss noch durch die Summe aller Gewichte geteilt (normiert) werden.

Die Gewichtsfunktion hängt natürlich vom Abstand d_i des Vertex \mathbf{v} vom Knochen B_i ab. Eine mögliche Wahl ist die Normalverteilung

$$\omega(d) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{d^2}{2\sigma^2}}$$

Hier benötigt man einen Wert für die Standardabweichung σ . Entweder kann man ihn tatsächlich aus allen (kürzesten) Entfernungen aller Vertices von den Knoten bestimmen oder man schätzt einen passenden Wert.

6.3 Inverse Kinematik

Mit dem bestehenden Ansatz lässt sich ohne Aufwand Forwärts-Kinematik umsetzen. Legt man die Rotationen an den Gelenken fest, so ergibt sich durch die kombinierten Transformationen vom Wurzelknoten bis zu einem Gelenk automatisch dessen Position. Um beispielsweise die Position einer Hand in einem humanoiden Skelett zu bestimmen, multipliziert man alle Transformationen der Gelenke und Knochen vom Rumpf bis zu Hand mit dem Vector $(0,0,0,1)$.

Das herausfordernde Problem ist die inverse Kinematik. Gibt man die Position der Hand vor, welche Transformationen auf dem Weg von der Wurzel bis zur Hand müssen dann verwendet werden? Dieses Problem stellt sich häufig bei der Animation von Skeletten. Um beispielsweise eine Gehen-Animation zu erstellen, setzt man dann nur die Position der Hände und der Füße, alle weiteren Parameter (Rotationen an den Gelenken) werden mittels inverser Kinematik bestimmt.

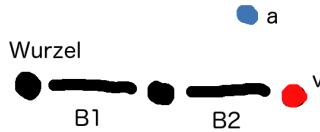


Abbildung 6.5: Beispiel zur Herleitung Inverse Kinematik mit zwei Knochen B_1 und B_2 . Der rote Punkt stellt den Endpunkt von B_2 dar, der blaue Punkt ist der Anfasserpunkt.

Um das Prinzip der inversen Kinematik herzuleiten betrachten wir das folgende Beispiel: Ein Skelett besteht aus einem Anker/Rumpf, von dem aus ein Knochen B_1 ausgeht auf den wieder ein Knochen B_2 folgt (siehe Abbildung 6.5). Das Ende von B_2 verwenden wir als *Anfasser*. Dessen Position wird von außen vorgegeben (z.B. durch ein Modellierungsprogramm). Die Längen der Knochen sind natürlich konstant, damit sind auch $\mathbf{T}_{1,t}$ und $\mathbf{T}_{2,t}$ fix. Die beiden Rotationen \mathbf{R}_1 an B_1 und \mathbf{R}_2 an B_2 müssen bestimmt werden. Dazu müssen die Rotationsmatrizen parametrisiert werden. Zur Parametrisierung von Rotationsmatrizen gibt es unterschiedliche Ansätze wie Euler-Winkel oder Quaternionen. Wir vereinfachen das Problem hier ein wenig indem wir annehmen, dass die Gelenke jeweils nur einen Freiheitsgrad haben. B_1 kann nur um die y-Achse rotieren, B_2 nur um die z-Achse. Es ergeben sich also zwei Rotationsmatrizen mit je einem Parameter: $\mathbf{R}_{1,y}(\alpha)$ und $\mathbf{R}_{2,z}(\beta)$. Insgesamt ergibt sich für die Berechnung des Endpunktes von B_2 :

$$\mathbf{v} = \mathbf{R}_{1,y}(\alpha) \cdot \mathbf{T}_{1,t} \cdot \mathbf{R}_{2,z}(\beta) \cdot \mathbf{T}_{2,t} \cdot (0, 0, 0, 1) = f(\alpha, \beta)$$

Das Ziel der Inversen Kinematik ist es nun, dass \mathbf{v} möglichst nah an den Anfasserpunkt \mathbf{a} kommt. Dazu betrachten wir die Fehlerfunktion

$$\epsilon(\alpha, \beta) = (\mathbf{v} - \mathbf{a})^2 = (f(\alpha, \beta) - \mathbf{a})^2.$$

Gesucht sind die Parameter α und β für die der Abstand zwischen \mathbf{v} und \mathbf{a} minimal wird:

$$\arg \min_{\alpha, \beta} \epsilon(\alpha, \beta),$$

6.3.1 Optimierung

Wir suchen nun die Parameter, die dazu führen, dass das Ende des zweiten Knochen dem Anfasserpunkt am nächsten kommt. Dazu betrachten wir ein – vielleicht das grundlegendste — Verfahren: Gradientenabstieg.

Wir haben es hier mit einer Fehlerfunktion $\epsilon(\alpha, \beta)$ zu tun, dessen Minimum sich nicht ohne weiteres direkt finden lässt. Wir berechnen das Minimum daher iterativ und nähern uns schrittweise mit dem Gradientenabstiegsverfahren der Lösung an. Dazu machen wir uns zunutze, dass der Gradient der Funktion $\nabla\epsilon$ in die Richtung des steilsten Anstiegs der Funktion zeigt. Bewegt man sich also in die negative Richtung, dann bewegt man sich in Richtung des nächsten lokalen Minimums. Für geeignete Funktionen und geeignete Startpunkte des iterativen Verfahrens konvergieren wir damit zum gesuchten Minimum:

$$\delta_{i+1} = \delta_i - s \cdot \nabla\epsilon,$$

mit Parametervektor $\delta = (\alpha, \beta)$ und Schrittweite s . Die Bestimmung des optimalen Werts von s (so groß wie möglich, ohne dass das Verfahren instabil wird) ist eine Wissenschaft für sich, die wir hier nicht weiter vertiefen.

Bleibt die Frage, wie der Gradient $\nabla\epsilon$ bestimmt wird. Hier bietet sich als allgemein gültige Lösung der Differenzenquotient. Betrachten wir zunächst den Gradienten von ϵ genauer:

$$\nabla\epsilon = \begin{pmatrix} \frac{\partial\epsilon}{\partial\alpha} \\ \frac{\partial\epsilon}{\partial\beta} \end{pmatrix}.$$

Der Gradient ist also der Vektor der partiellen Ableitung in Richtung der einzelnen Parameter, in unserem Beispiel α und β . Die partielle Ableitung nach α über den Differenzenquotienten lautet dann

$$\frac{\partial\epsilon}{\partial\alpha} = \frac{\epsilon(\alpha + h, \beta) - \epsilon(\alpha, \beta)}{h}$$

für eine geeignete Schrittweite h . Die partielle Ableitung nach β ergibt sich analog.

Insgesamt ergibt sich also der folgende Algorithmus:

```
Initialisiere Parametervektor δ // z.B. alter Stand oder 0
WHILE nicht konvergiert
    Bestimme ∇ε
    δ = δ - s · ∇ε
END WHILE
```

Also Konvergenzbedingung prüft man üblicherweise ob

- entweder das Ergebnis sich nicht weiter verändert
- oder eine maximale Anzahl von Iterationen durchgeführt wurde
- oder der Anfasspunkt hinreichend genau erreicht wurde.

Das Verfahren zur Durchführung der Inversen Kinematik wurde nun ausschliesslich am Beispiel des Skelettes mit zwei Knochen und zwei Parametern durchgeführt. Selbstverständlich lässt es sich aber mit den gleichen Ansätzen auf komplexere Aufbauten anwenden.

7 Datenstrukturen

7.1 Mathematische Grundlagen: Geometrische Elemente

7.1.1 Gerade/Strahl

Eine Gerade ist eindeutig durch zwei nicht identische Punkte \mathbf{p} und \mathbf{q} bestimmt. Es gibt verschiedene Möglichkeiten, eine solche Gerade zu repräsentieren. Die gängigste ist die parametrisierte Form. Dazu berechnet man zunächst einen Richtungsvektor $\vec{v} = \mathbf{q} - \mathbf{p}$. Mit diesem Richtungsvektor ergibt sich die parametrische Geradendarstellung

$$g : \mathbf{x} = \mathbf{p} + \lambda \vec{v}$$

für beliebige $\lambda \in \mathbb{R}$ und $\mathbf{x}, \vec{v}, \mathbf{p}, \mathbf{q} \in \mathbb{R}^n$. Alle Punkte \mathbf{x} , die sich mit einem Wert von λ konstruieren lassen, sind demnach Teil der Geraden.

Ein Sonderfall von Geraden sind Strahlen. Hier geht man davon aus, dass nur solche Punkte Teil des geometrischen Elementes sind, bei denen gilt: $\lambda \geq 0$.

Übung 7.1 Gegeben sind zwei Geraden

$$g_1 = \begin{pmatrix} 1 \\ 2 \end{pmatrix} + s \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

und

$$g_2 = \begin{pmatrix} -1 \\ -1 \end{pmatrix} + t \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

Prüfen Sie zunächst, ob sich die beiden Geraden schneiden. Berechnen Sie dann den Schnittpunkt zwischen g_1 und g_2 .

7.1.2 Ebene

Eine Ebene ist definiert durch drei Punkte \mathbf{a} , \mathbf{b} und \mathbf{c} , die nicht auf einer gemeinsamen Geraden liegen. Auch für die Ebene lässt sich eine parametrisierte Darstellung angeben. Dazu berechnet man zunächst die aufspannenden Vektoren $\vec{u} = \mathbf{b} - \mathbf{a}$ und $\vec{v} = \mathbf{c} - \mathbf{a}$. Damit lässt sich die Ebene schreiben als:

$$E : \mathbf{x} = \mathbf{a} + s \vec{u} + t \vec{v}$$

für $s, t \in \mathbb{R}$ und $\mathbf{x}, \vec{u}, \vec{v}, \mathbf{a}, \mathbf{b}, \mathbf{c} \in \mathbb{R}^n$.

Eine alternative Repräsentation von Ebenen ist die Hesse-Normalform. Diese lautet:

$$E : \mathbf{x} \cdot \vec{n} - d = 0,$$

wobei \vec{n} der Normalenvektor der Ebene in Einheitslänge ist und d sich als $d = \vec{n} \cdot \mathbf{p}$ berechnet. Den Normalenvektor kann man z.B. aus \vec{u} und \vec{v} mit dem Kreuzprodukt berechnen. Diese Darstellung kann z.B. dazu verwendet werden, einfach den Abstand eines Punktes von der Ebene zu berechnen:

$$\text{dist}(E, \mathbf{p}) = \mathbf{p} \cdot \vec{n} - d.$$

Diese Abstandsfunktion liefert einen positiven Wert, falls sich \mathbf{p} auf der Seite der Ebene befindet, in die der Normalenvektor zeigt, andernfalls einen negativen Wert.

Übung 7.2 Berechnen Sie den Schnittpunkt zwischen der Ebene

$$E : \mathbf{x} \cdot \begin{pmatrix} 1 \\ 1 \end{pmatrix} - 1 = 0,$$

und der Geraden

$$g = \begin{pmatrix} 1 \\ 2 \end{pmatrix} + s \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

8 Simulation

8.1 Partikel

Wir beginnen die Betrachtung von Simulationsmethoden mit dem einfachsten Körper, den Partikel. Ein (Masse-)Partikel wird durch drei Informationen bestimmt: Position \mathbf{p} , Geschwindigkeit \mathbf{v} und Masse m . Wir nehmen hier an, dass die Masse des Partikels konstant bleibt, Position und Geschwindigkeit hingegen ändern sich über die Zeit. Daher geben wir sie auch zeitabhängig an: Position $\mathbf{p}(t)$ und Geschwindigkeit $\mathbf{v}(t)$. Meist startet die Simulation der Bewegung eines Partikels in einem Ursprungszustand

$$\mathbf{x}(t_0) = \begin{pmatrix} \mathbf{p}(t_0) \\ \mathbf{v}(t_0) \end{pmatrix}$$

Von außen wirken verschiedene Kräfte auf einen Partikel, die die Bewegung des Partikels beeinflussen. Der Zusammenhang zwischen Kraft \mathbf{F} und Geschwindigkeit \mathbf{v} ist gegeben durch das zweite Newtonsche Gesetz

$$\mathbf{F} = \mathbf{a} \cdot m$$

mit der Beschleunigung \mathbf{a} . Außerdem ist die Geschwindigkeit die zeitliche Ableitung der Position und die Beschleunigung die zeitliche Ableitung der Geschwindigkeit, also

$$\mathbf{p}'(t) = \mathbf{v}(t), \mathbf{v}'(t) = \mathbf{a}(t)$$

In der Physik verwendet man meist eine andere Notation für diesen Zusammenhang:

$$\dot{\mathbf{p}}(t) = \mathbf{v}(t), \dot{\mathbf{v}}(t) = \mathbf{a}(t) = \ddot{\mathbf{p}}(t)$$

Will man nun die Bewegung eines Partikels über die Zeit verfolgen beginnt man mit dem Zustand $\mathbf{x}(t_0)$ und löst schrittweise die zwei gekoppelten Differenzialgleichungen für $\mathbf{p}(t)$ und $\mathbf{v}(t)$, also:

$$\dot{\mathbf{x}}(t) = \begin{pmatrix} \mathbf{v}(t) \\ \mathbf{F}(t)/m \end{pmatrix}$$

8.1.1 Beispiel: Feuer

Wir betrachten das Beispiel der Simulation einer Flamme. Die Flamme wiederum wird durch eine Menge von Partikeln repräsentiert. Jeder der Partikel agiert unabhängig und durchläuft drei Phasen: Erschaffung mit Startzustand, Bewegung unter Einfluss externer Kräfte, Zerstörung. Dieser Zusammenhang ist in Abbildung 8.1 dargestellt: Der Partikel startet mit dem Zustand $\mathbf{x}(t_0)$. Zu Beginn hat er eine Startgeschwindigkeit $\mathbf{v}(t_0)$. Ab dann gibt es zwei Kräfte, die die Bewegung des Partikels beeinflussen: Die Erddziehung \mathbf{F}_G und der Wind, der eine waagerechte Kraft \mathbf{F}_{WIND} induziert. Die Erddziehung lässt sich aus der Masse m des Partikels und der Erdbeschleunigung $g \approx 9.81 \frac{m}{s^2}$

berechnen – hier: $\begin{pmatrix} 0 \\ -9.81 \\ 0 \end{pmatrix}$.

¹<https://de.wikipedia.org/wiki/Schwerefeld>

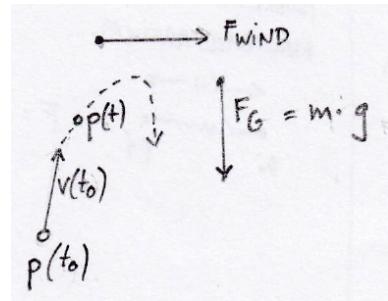


Abbildung 8.1: Simulation eines Feuer-Partikels. Als externe Kräfte wirken die Erdanziehung und Wind.

8.1.2 Integration

Die Aufgabe in einem Simulationssystem ist es nun, aus dem Startzustand und den Kräften zu jedem Zeitpunkt $\mathbf{F}(t)$ die Bewegung (Position und Geschwindigkeit) des Partikels über die Zeit zu bestimmen. Diesen Vorgang nennt man Integration. Wir führen hier eine diskrete Integration durch. Das heißt, der Zustand wird nicht kontinuierlich, sondern immer zu festen Zeitschritten iterativ bestimmt. Dies wird häufig auch durch eine Index-Notation dargestellt:

$$\mathbf{x}_n \rightarrow \mathbf{x}_{n+1}$$

mit Startzustand \mathbf{x}_0 . Der Zustand \mathbf{x}_n ist der Zustand zum Zeitpunkt $t_n = t_0 + n \cdot \Delta t$, wobei Δt der Zeitabstand zwischen zwei Zuständen ist.

In der Literatur findet man eine Reihe unterschiedlicher Integrationsverfahren, mit unterschiedlichen Vor- und Nachteilen. Generell unterscheidet man explizite und implizite Verfahren. Wir betrachten hier zunächst nur das einfachste explizite Verfahren, das explizite Euler-Verfahren. Dieses Verfahren leiten wir zunächst her.

Wir betrachten die Funktion in Abbildung 8.2. Insbesondere befinden wir uns bei der Integration an der Stelle t_n . Dort haben wir den Zustand $x_n = x(t_n)$. t_{n+1} ergibt sich aus t_n durch die Schrittweite Δt . Um nun von x_n zu x_{n+1} zu kommen, bilden wir die Differenz aus x_{n+1} und x_n . Diese ergibt sich als Annäherung aus der Steigung der Funktion an der Stelle x_n (Tangente an die Funktion an der Stelle t_n).

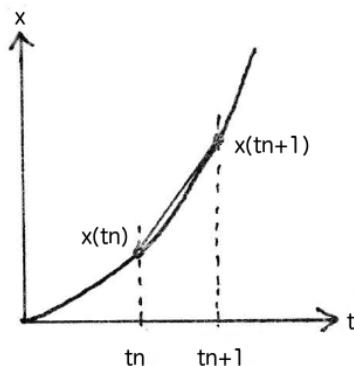


Abbildung 8.2: Der Funktionswert der Funktion an der Stelle t_{n-1} wird aus der Steigung und aus dem Funktionswert an der Stelle t_n geschätzt.

Die Steigung lässt sich nach dem Differenzenquotienten berechnen als

$$x'(t) = \frac{x(t + \Delta t) - x(t)}{\Delta t}$$

oder umgestellt

$$x(t + \Delta t) = x(t) + x'(t)\Delta t.$$

Die Ableitung eines Partikelzustandes

$$x'(t)$$

kennen wir aus der Einleitung:

$$x'(t) = \dot{x}(t) = \begin{pmatrix} v(t) \\ a(t) = F(t)/m \end{pmatrix}$$

Das explizite Euler-Verfahren ist aber natürlich nur eine Annäherung an das korrekte Ergebnis. Streng genommen liefert es nur für lineare Funktionen ein korrektes Ergebnis, das ist in der Praxis nur selten der Fall. Daher muss man sich mit der Frage beschäftigen, wie groß der Fehler ist, der in jedem Integrationsschritt in den Zustand einfliest. Wir betrachten die Details hier nicht, Informationen dazu findet man aber in jeder seriösen Literatur zu Integrationsverfahren. Wir begnügen uns mit der Erkenntnis, dass kleinere Schritte einen geringeren Fehler verursachen. Leider wird das Verfahren und damit die gesamte Simulation durch kleine Schrittweiten langsamer.

Außerdem sei noch eine kleine Einordnung zu den impliziten Integrationsverfahren gegeben. Während man bei den expliziten Verfahren die Veränderung am aktuellen Zeitpunkt t_n verwendet, um auf den nächsten Zeitschritt zu schliessen, verwendet man bei den impliziten Verfahren die Veränderung am kommenden Zeitpunkt t_{n+1} . Aus der Integrationsvorschrift (explizit)

$$x_{n+1} = x_n + \Delta t \cdot x'_n$$

wird dann in einem impliziten Verfahren

$$x_{n+1} = x_n + \Delta t \cdot x'_{n+1}.$$

Zur Lösung dieses Systems ist es meist notwendig, zumindest ein lineares Gleichungssystem zu lösen. Im Gegenzug ist das Verfahren dann stabiler und erlaubt größere Schrittweiten.

8.1.3 Simulation eines Partikelsystems

Also: Um mit dem expliziten Euler-Verfahren von einem Zustand $\mathbf{x}(t)$ zu seinem Nachfolgezustand $\mathbf{x}(t + \Delta t)$ zu kommen, addiert man auf die Position die mit der Schrittweite skalierte Geschwindigkeit

$$\mathbf{p}(t + \Delta t) = \mathbf{p}(t) + \Delta t \cdot \mathbf{v}(t)$$

und auf die Geschwindigkeit die mit der Schrittweite skalierte Beschleunigung.

$$\mathbf{v}(t + \Delta t) = \mathbf{v}(t) + \Delta t \cdot \mathbf{a}(t) = \Delta t \cdot \frac{\mathbf{F}(t)}{m}$$

Die Kraft $F(t)$ ergibt sich aus der Summe der externen Kräfte, z.B. Schwerkraft, Wind, Federkräfte,

...

In Pseudocode sieht das so aus:

```

// Initialisierung
FOR ALL p in partikelListe
    p.pos = Startposition
    p.vel = Startgeschwindigkeit
END FOR

// Simulation
WHILE not done
    FOR ALL p in partikelListe
        p.pos += d * p.vel
        p.vel += d * F / p.m
    END FOR
END WHILE

```

Dem Pseudocode wird folgende Notation zugrundegelegt:

- Partikel p mit Position $p.\text{pos}$ und Geschwindigkeit $p.\text{vel}$.
- Schrittweite d
- Kraft F

8.1.4 Simulation von Textilien

Neben dem Beispiel (Simulation von Feuer) betrachten wir noch ein zweites Phänomen, das sich mit einer Partikel-Simulation beschreiben lässt: die Simulation von Textilien oder Stoffen. Der zugrundeliegende Ansatz geht auf die Arbeiten von [TPBF87] und [BW98] zurück.

Zum Verständnis des Ansatzes ist es notwendig, sich den Aufbau und das Verhalten einer Feder zu vergegenwärtigen (Abbildung 8.3).

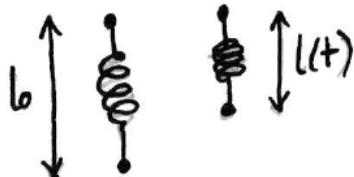


Abbildung 8.3: Bei der Stauchung einer Feder mit der Ruhelänge l_0 ergibt sich zum Zeitpunkt t die Länge $l(t)$ (analog bei der Streckung).

Ohne die Einwirkung externer Kräfte befindet sich eine Feder in ihrer Ruhelage. Dann hat sie die Länge l_0 . An den beiden Endpunkten wirkt keine Kraft; die Feder ruht. zieht man die Feder nun auseinander oder staucht sie zusammen, dann verändert sich die Länge der Feder. Aus l_0 wird $l(t)$. Durch Stauchung wird die Länge kürzer, durch Auseinanderziehen wird sie länger. Die Feder versucht dann aber, in ihre Ruhelage zurückzukommen: an ihren beiden Enden wirkt eine Kraft, die der ursprünglichen Kraft, die die Deformation verursacht hat, entgegenwirkt. Die Stärke der Kraft hängt von der Stärke der Deformation zusammen: wurde die Feder weiter auseinandergezogen, dann ist die Kraft, die dem entgegenwirkt, größer. Der Zusammenhang zwischen der Federlänge und der wirkenden Kraft wird durch das Hooksche Gesetz beschrieben:

$$F(t) = k \cdot (l(t) - l_0).$$

Die Konstante k heißt Federkonstante und beschreibt die Materialeigenschaften der Feder.

Achtung: Streng genommen gilt das Hooksche Gesetz nur in einem gewissen Bereich $|l(t) - l_0| < \epsilon$. Deformiert man die Feder weiter, dann ist sie zerstört und kehrt nicht in die Ruhelage zurück. Wir betrachten hier nur den Bereich, in dem das Gesetz gilt.

Bei der Simulation von Textilien machen wir uns die Simulation von Federn zunutzen. Wir nehmen an, dass ein Textil aus einem Gitter von Partikeln besteht. Jeder Partikel ist über Federn mit Partikeln in seinem Umfeld verbunden: Abbildung 8.4.

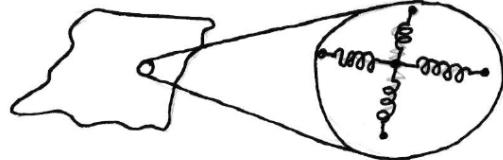


Abbildung 8.4: Bei der Simulation von Stoffen simuliert man ein Gitter aus durch Federn verbundenen Massepunkten.

Betrachten wir die Verbindungen zwischen Partikeln genauer, ergibt sich ein System, das aus drei verschiedenen Federtypen besteht. Diese drei Federtypen können unterschiedliche Federkonstanten haben und damit die Materialeigenschaften des Textils steuern: Struktur, Scherung und Biegung. Abbildung 8.5 zeigt den Aufbau: Die Strukturfedern sind immer mit den vier direkten Nachbarn im Gitter verbunden. Die Scherungsfedern verlaufen diagonal durch das Gitter und verbinden mit den entsprechenden vier Nachbarpartikeln. Die Biegungsfedern überspringen je einen Nachbarn und sind demnach mit den übernächsten Partikeln verbunden.

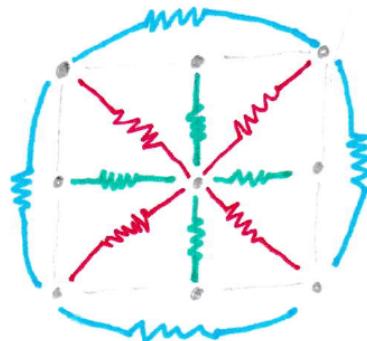


Abbildung 8.5: Bei der Simulation von Stoffen kommen drei Typen von Federn zum Einsatz: Strukturfedern (zu den direkten Nachbarpartikeln, grün), Scherungsfedern (zu den diagonalen Nachbarn, rot) und Biegungsfedern (zu den übernächsten Nachbarn, blau). Die Partikel sind als graue Punkte dargestellt.

Bringt man nun die Federkräfte und das Partikelgitter zur Simulation eines Textils zusammen, dann ergibt sich der folgende Zusammenhang:

$$F_{i,j}(t) = k(l(t) - l_0) \frac{p_j - p_i}{\|p_j - p_i\|}$$

Die Kraft, die eine Feder zwischen den Partikeln i und j induziert, zeigt damit immer in die (normierte) Richtung von der Partikel-Position p_i zur Partikelposition p_j . Im Ruhezustand des Textils sind alle Federn entspannt, das heißt, ihre Länge entspricht ihrer Ruhelänge. Im Laufe der Simulation sorgen externe Kräfte dafür, dass sich die Positionen der Partikel verändern. Damit verändern

sich auch die Längen $l(t)$ der Federn und die Federkräfte wirken so, dass sie diese Deformationen auszugleichen versuchen.

8.2 Simulation von Starrkörpern

Partikel haben zunächst kein Volumen. Will man Körper mit einem Volumen repräsentieren, muss man einen Schritt weiter gehen. Wir betrachten dabei zunächst Starrkörper, also Körper, deren Volumen und Form sich nicht ändern. Die notwendigen Grundlagen lassen sich sehr gut bei [WB97] nachlesen. Eine anschauliche Einführung findet man hier².

Alle Eigenschaften eines Partikels (Masse, Position, Geschwindigkeit) hat auch ein Starrkörper. Hinzu kommt aber, dass ein solcher Volumenkörper zusätzlich rotieren kann. Wir nehmen hier vereinfachen an, dass die Dichte des betrachteten Körpers konstant über das gesamte Volumen ist und sich der Körper daher immer um seinen Schwerpunkt \mathbf{p} dreht (Winkel ϕ): Abbildung 8.6. Wir betrachten Starrkörpersimulationen zunächst im 2D, weil das die Herleitung der Rotationen deutlich vereinfacht. Für die Erweiterung auf den 3D sei auf [WB97] verwiesen.

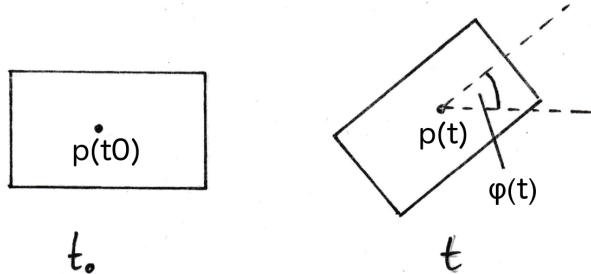


Abbildung 8.6: Starrkörper im Ruhezustand an Position p_0 (links) und nach linearer ($p(t)$) und angulärer ($\phi(t)$) Bewegung (rechts).

Zunächst müssen wir uns noch einmal vergegenwärtigen, was der Unterschied zwischen der Bewegung eines Partikels und eines Starrkörpers ist: die Ausdehnung und damit das Volumen. Um also die Masse eines Starrkörpers zu bestimmen, könnte man sich vorstellen, dass der Starrkörper durch n Partikel mit der Masse m_i , die jeweils ein Teilvolumen repräsentieren, zusammengesetzt ist. Der Schwerpunkt des Körpers ergäbe sich dann als

$$\mathbf{p}_0 = \frac{m_i \mathbf{r}_i}{n},$$

wobei \mathbf{r}_i der Vektor zum Massenpunkt m_i innerhalb des Körpers ist. Durch die Diskretisierung in n Partikel erhalten wir aber nur eine Approximation. Der genaue Schwerpunkt ergibt sich aus dem Integral über das Volumen:

$$\mathbf{p}_0 = \frac{1}{m} \int_V \rho(\mathbf{r}) \mathbf{r} dV$$

mit Dichte $\rho(\mathbf{r})$ und der Stelle \mathbf{r} im Körper und Masse des Körpers m .

Die Größen, die aus der Repräsentation von Partikeln übernommen wurden (Position und Geschwindigkeit \mathbf{v}_{lin}), stellen lineare Bewegungen dar. Ein Starrkörper kann neben der linearen Bewegung zusätzlich eine anguläre (= Rotations-) Geschwindigkeit haben. Die Rotation findet sich bereits in Abbildung 8.6: ϕ . Zu klären ist noch, wie die Veränderung der Rotation/Orientierung über die Zeit aussieht.

²<https://www.toptal.com/game/video-game-physics-part-i-an-introduction-to-rigid-body-dynamics>, abgerufen am 18.09.2017

Dazu führen wir die Rotationsgeschwindigkeit \mathbf{v}_{ang} ein (mit der Einheit Radians pro Sekunde). Damit überhaupt Rotationsgeschwindigkeiten auftreten, muss zunächst eine Rotationskraft wirken. Diese wird als *Drehmoment* (mit dem Zeichen τ) bezeichnet. Auch für das Drehmoment können wir analog zu $\mathbf{F} = m \cdot \mathbf{a}$ das zweite Newtonsche Gesetz aufstellen:

$$\tau = I \cdot \alpha.$$

Das Pendant zur Beschleunigung \mathbf{a} ist hier die Rotationsbeschleunigung α und das Gegenstück zur Masse ist hier das *Trägheitsmoment* I .

Das Trägheitsmoment hängt von den den Eigenschaften des Körpers ab und ist allgemein definiert als

$$I = \int_V \rho(\mathbf{r}) \mathbf{r}^2 dV.$$

Für einfache geometrische Formen kann man das Trägheitsmoment in Tabellen nachschlagen, etwa hier³. Für ein Rechteck mit Höhe h und Breite b ergibt sich

$$I = \frac{m(b^2 + h^2)}{12}.$$

Wenn nun eine Kraft \mathbf{F} auf einen Starrkörper wirkt, dann kann er eine Drehmoment hervorrufen. Wie, hängt von der Stelle ab, an der die Kraft auf dem Körper ansetzt (Abbildung 8.7):

$$\tau = ||\mathbf{r}|| |\mathbf{F}| \sin \theta$$

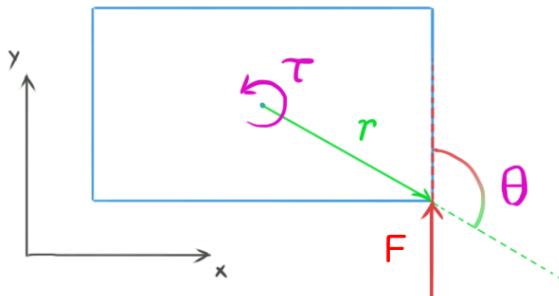


Abbildung 8.7: Die Kraft F wirkt auf den Körper an der Stelle r und induziert damit ein Drehmoment.) τ .

Dieser Zusammenhang sollte uns stark an die Definition des Kreuzproduktes erinnern, und tatsächlich:

$$\tau = \mathbf{r} \times \mathbf{F}.$$

Da wir hier nur den 2D-Fall betrachten reduziert sich die Formulierung zu

$$\tau = r_x \cdot F_y - r_y \cdot F_x.$$

Insgesamt ergibt sich also für einen Starrkörper der folgende Zustandsvektor

³Trägheitsmomente für einfache geometrische Körper (Wikipedia): https://en.wikipedia.org/wiki/List_of_moments_of_inertia, abgerufen am 19.09.2017

$$\mathbf{x} = \begin{pmatrix} \mathbf{p}(t) \\ \mathbf{v}_{\text{lin}}(t) \\ \phi(t) \\ v_{\text{ang}}(t) \end{pmatrix}$$

mit linearer Geschwindigkeit $\mathbf{v}_{\text{lin}}(t)$ und angulärer Geschwindigkeit $v_{\text{ang}}(t)$
Die zeitliche Ableitung des Zustandes ist

$$\dot{\mathbf{x}} = \begin{pmatrix} \mathbf{v}_{\text{lin}}(t) \\ \mathbf{F}(t)/m \\ v_{\text{ang}}(t) \\ \tau(t)/I \end{pmatrix}.$$

Damit lässt sich die Simulation, beispielsweise wieder mit dem expliziten Euler-Verfahren, analog zu dem Vorgehen für Partikel umsetzen.

8.3 Simulation Deformierbarer Körper

Die logische Erweiterung von Starrkörpern ist die Simulation von deformierbaren Körpern. Auch hier gibt es natürlich viele verschiedene Ansätze. Ein Ansatz, der auf der Repräsentation von Partikeln aufbaut, sind die Netz-freien Ansätze. Dabei setzt sich das Volumen des Körpers, der simuliert wird, aus einem Gitter von Partikeln (auch Phyxel genannt) zusammen. Die grundlegende Idee dazu ist in [MKN⁺04] beschrieben, der gleiche Autor erläutert das Vorgehen noch anschaulicher in [GP07].

Literaturverzeichnis

- [BW98] BARAFF, David ; WITKIN, Andrew: Large Steps in Cloth Simulation. In: *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*. New York, NY, USA : ACM, 1998 (SIGGRAPH '98). – ISBN 0-89791-999-8, 43–54
- [DBGJ13] DÖRNER, R. (Hrsg.) ; BROLL, W. (Hrsg.) ; GRIMM, P. (Hrsg.) ; JUNG, B. (Hrsg.): *Virtual und Augmented Reality (VR / AR)*. Bd. 1. Springer Vieweg, 2013
- [Fis05] FISCHER, G.: *Lineare Algebra*. Vieweg, 2005 (Vieweg-Studium : Grundkurs Mathematik). <http://books.google.de/books?id=rUlGDEFCRxkC>. – ISBN 9783834800312
- [GP07] GROSS, Markus ; PFISTER, Hanspeter: *Point-Based Graphics*. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 2007. – ISBN 0123706041, 9780080548821
- [Hec86] HECKBERT, Paul S.: Survey of Texture Mapping. In: *IEEE Comput. Graph. Appl.* 6 (1986), November, Nr. 11, 56–67. <http://dx.doi.org/10.1109/MCG.1986.276672>. – DOI 10.1109/MCG.1986.276672. – ISSN 0272–1716
- [MKN⁺04] MÜLLER, M. ; KEISER, R. ; NEALEN, A. ; PAULY, M. ; GROSS, M. ; ALEXA, M.: Point Based Animation of Elastic, Plastic and Melting Objects. In: *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. Aire-la-Ville, Switzerland, Switzerland : Eurographics Association, 2004 (SCA '04). – ISBN 3-905673-14-2, 141–151
- [MTLT88] MAGNENAT-THALMANN, N. ; LAPERRI‘ERE, A. ; THALMANN, D.: Joint-Dependent Local Deformations for Hand Animation and Object Grasping. In: *Proceedings of Graphics Interface '88*. Toronto, Ontario, Canada : Canadian Man-Computer Communications Society, 1988 (GI '88). – ISSN 0713–5424, 26–33
- [Sha98] SHANNON, Claude E.: Communication in the Presence of Noise. In: *PROCEEDINGS OF THE IEEE, VOL. 86, NO. 2, FEBRUARY 1998*, 1998
- [TPBF87] TERZOPOULOS, Demetri ; PLATT, John ; BARR, Alan ; FLEISCHER, Kurt: Elastically Deformable Models. In: *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*. New York, NY, USA : ACM, 1987 (SIGGRAPH '87). – ISBN 0-89791-227-6, 205–214
- [WB97] WITKIN, Andrew ; BARAFF, David: Physically Based Modeling: Principles and Practice. In: *Siggraph '97 Course notes*, 1997