Graphikprogrammierung

Prof. Dr. Bernhard Möller

Universität Augsburg

SS06

Inhaltsverzeichnis

1 Einleitung

1.1 Begriffsdefinition

Computergrafik: realistische Darstellung realer oder imaginärer 3D-Szenerien

Dies erfolgt in der Regel in 3 Schritten:

1. Erstellen eines abstrakten Modells der Szenerie
2. Projektion des Modells auf den 2D-Bildraum
3. Ausgabe des 2D-Bilds auf passendem Gerät

1.2 Kurze Geschichte

bereits ab 1949	erste Anfänge
1962	erste 3D-Grafiken
1963	erstes interaktives Graphiksystem (SUTHERLAND) (Bildkomposition aus Standardelementen[1], Menüs mit Tastatur und Lichtgriffel bedient)
Anfang 1970er	erste kommerzielle CAD/CAM-Systeme
1972	erster Flugsimulator
Ab Mitte 1970er	Grafische Programmiersprachen Schattierung/Texturierung
1979	Reflexion und Transparenz
1980	fraktale Landschaften und erste Animationen
1981	Rendering (fotorealistische Darstellung)
1982	Morphing
1984	globale Beleuchtungsmodelle
1985	Erster graphischer ISO-Standard GKS (*Graphics Kernel System*)
1989	Bewegungsabtastung[2] (*Motion Capture*)

Seither wesentliche Fortschritte in der Animation und der realistischen Wiedergabe von virtuellen Charakteren (+ Hinzunahme von KI-Elementen).

[1] Würfel, Kugel, Kegel etc.
[2] von Schauspielern, für realistischere Darstellung von Körperbewegungen

1.3 Übersicht: Ein typisches Graphikfließband

1. Ebene der Anwendung/Szenerie
   - Durchmustern der Datenbank der Szenerie-Objekte
   - Bewegen der Objekte, Zielen mit der und Bewegen der Kamera
   - Sichtbarkeitsanalyse
   - Einstellen der Detailstufe (LOD, *level of detail*)
2. Geometrische Operationen
   - Transformationen (Translation, Drehung, Streckung, Scherung)
   - Übergang von Objektkoordinaten zu Weltkoordinaten
   - Übergang von Weltkoordinaten zu Kamerakoordinaten
   - Projektion auf das Sichtfenster (u. U. perspektivisch)
   - Entfernung verdeckter Flächen (*culling*)
   - Kappen auf Bildausschnitt (*clipping*)
   - Beleuchtung
3. Bilderstellung (*rendering*)
   - Rasterung und Anti-Alias
   - Farbinterpolation (*shading*)
   - Texturen
   - Nebel
   - Transparenz (*alpha translucency*)
   - Schatten
   - Tiefenpufferung (*z-buffering*)
   - Bildanzeige

Viele Schritte davon werden heute von den Graphikkarten im Rechner unterstützt.

2 Grundlegende Verfahren und Techniken

2.1 Koordinaten und Transformationen

2.1.1 Punkte und Vektoren

Punkte: Orte im Raum
Vektoren: „gerichtete Abstände“ zwischen Punkten

„Um wie viel und in welche Richtung muss ich Punkt X verschieben, um Punkt Y zu erhalten?“

Notation: $\overrightarrow{XY}$; $Y = X + \overrightarrow{XY}$; $\overrightarrow{XY} = Y - X$

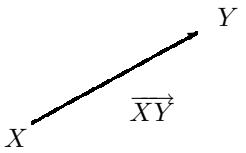


Abbildung 2.1.

Ist ein Koordinatenursprungspunkt Ω gegeben, so kann Punkt X mit seinem Ortvektor $\overrightarrow{\Omega X}$ identifiziert werden (vgl. Lineare Algebra).

Oft ist die Unterscheidung aber nützlich:

- Punkte entsprechen *absoluten Koordinaten*, im $\mathbb{R}^3 : X = (x_1, x_2, x_3)$
- Vektoren entsprechen *relativen Koordinaten*,
  im $\mathbb{R}^3 : \overrightarrow{XY} = \begin{pmatrix} y_1 - x_1 \\ y_2 - x_2 \\ y_3 - x_3 \end{pmatrix}$

Man kann diesen Sachverhalt wie folgt verallgemeinern:

Definition 2.1.1. Ein *affiner Raum* ist ein Paar (P, V) bestehend aus einer Menge P von *Punkten* und einer Menge V von *Vektoren*, die einen Vektorraum über einem geeigneten Körper $\mathbb{K}$ bilden.
Weiter gibt es eine Operation $+$: $P \times V \to P$, die *Verschiebung*, mit folgenden Eigenschaften:

1. $X + \mathbf{0} = X \quad \forall X \in P$ ($\mathbf{0} \in V$ ist Nullvektor);
2. $(X + u) + v = X + \underbrace{(u + v)}_{\text{Vektoraddition}} \quad \forall X \in P,\ u, v \in V$
3. $\forall X, Y \in P \quad \exists_1 u \in V : X + u = Y$
   Schreibweise: $u = \overrightarrow{XY} = Y - X$, dann gilt $Y = X + \overrightarrow{XY}$

Beispiel:

(a) $(\underbrace{\mathbb{R}^3}_{\text{Zeilen}}, \underbrace{\mathbb{R}^3}_{\text{Spalten}})$

$$(x_1, x_2, x_3) + \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} = (x_1 + u_1, x_2 + u_2, x_3 + u_3)$$

$$(y_1, y_2, y_3) - (x_1, x_2, x_3) = \begin{pmatrix} y_1 - x_1 \\ y_2 - x_2 \\ y_3 - x_3 \end{pmatrix}$$

(b) (V, V), wobei V ein Vektorraum (wieder Identifizierung von Punkten mit ihren Ortsvektoren), Verschiebung fällt dann mit der Vektoraddition zusammen.

Zwei wichtige Operationen auf Vektoren sind das *Skalarprodukt* und das *Vektorprodukt*.

Definition 2.1.2. : Ein Vektorraum V über dem Körper $\mathbb{R}$ heißt *euklidisch*, wenn auf ihm eine Operation $\langle \cdot, \cdot \rangle : V \times V \to \mathbb{R}$, das *Skalarprodukt* definiert ist mit folgenden Eigenschaften für alle $u, v, w \in V$ und $\lambda \in \mathbb{R}$:

- $\langle u, v \rangle = \langle v, u \rangle$ (Symmetrie)
- $\langle u, u \rangle > 0$, falls $u \neq 0$ (Definitheit)
- $\langle u + v, w \rangle = \langle u, w \rangle + \langle v, w \rangle$
- $\langle \lambda u, v \rangle = \lambda \langle v, u \rangle$

} ((Bi-)Linearität)

Speziell im $\mathbb{R}^n$ gilt: $\langle u, v \rangle = \sum_{i=1}^{n} u_i v_i$

Der Wert $\|u\| \stackrel{\text{def}}{=} \sqrt{\langle u, u \rangle}$ heißt *euklidische Norm* oder *Länge* von u.
Länge von u im $\mathbb{R}^2$: $\langle u, u \rangle = u_1^2 + u_2^2 = \|u\|^2$ (Pythagoras)

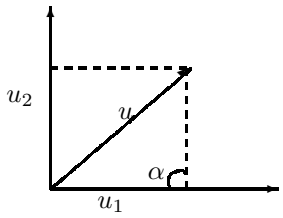


Abbildung 2.2.

Ist $\measuredangle(u, v)$ der Winkel zwischen u und v (wenn man sie am selben Punkt anträgt), so gilt mit $\alpha = \measuredangle(u, v)$: $\langle u, v\rangle = \|u\| \cdot \|v\| \cdot \cos\alpha$. Insbesondere gilt: $\langle u, v\rangle = 0 \Leftrightarrow u \perp v$ (oder $u = 0$ oder $v = 0$).

Im $\mathbb{R}^3$ ist daher jede Ebene **E** eindeutig bestimmt durch einen beliebigen Normalenvektor n und einen beliebigen Aufhängepunkt X. $E = \{X + v : \langle n, v\rangle = 0\}$. Oft wird der Normalenvektor noch als Einheitsvektor, d. h. mit $\|n\| = 1$ gewählt. Ist m ein beliebiger Normalenvektor $\neq 0$, so ist $\frac{m}{\|m\|}$ ein Einheits-Normalenvektor.

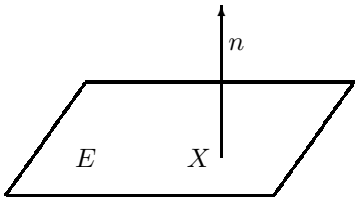


Abbildung 2.3.

Zur Gewinnung eines solchen Normalenvektors im $\mathbb{R}^3$ dient das Vektorprodukt.

Definition 2.1.3. : Das *Vektorprodukt* ist eine Operation $\times : \mathbb{R}^3 \times \mathbb{R}^3 \to \mathbb{R}^3$ mit den Eigenschaften für $u, v \in \mathbb{R}^3$:

- $(u \times v) \perp u$ und $(u \times v) \perp v$
- u, v und $u \times v$ bilden ein rechtshändiges System
- $\|u \times v\|$ = Fläche des von u und v gebildeten Parallelogramms, d. h. $\|u \times v\| = \|u\| \cdot \|v\| \cdot \sin\alpha$

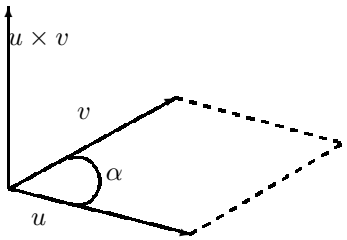


Abbildung 2.4.

In Koordinaten[3] gilt

$$\begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} \times \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} = \begin{pmatrix} u_2 \cdot v_3 - u_3 \cdot v_2 \\ u_3 \cdot v_1 - u_1 \cdot v_3 \\ u_1 \cdot v_2 - u_2 \cdot v_1 \end{pmatrix}$$

[3]Merkregel: In jeder Zeile tauchen nur jeweils die zwei anderen Zeilenindizes auf (in „aufsteigender“ Reihenfolge modulo 3), in der Form einer zweireihigen Determinante

Einige Gesetze:

- $u \times v = -(v \times u)$ (also *nicht* kommutativ!)
- $u \times v = 0$ wenn $u||v$
- $u \times (v + w) = u \times v + u \times w$
- $(u + v) \times w = u \times w + v \times w$

Ein Assoziativgesetz gilt nicht, da

- $u \times (v \times w) = \langle u, w \rangle \cdot v - \langle u, v \rangle \cdot w$
- $(u \times v) \times w = \langle u, w \rangle \cdot v - \langle v, w \rangle \cdot u$

Durch Kombination von Skalar- und Vektorprodukt lässt sich das Volumen des von u, v und w gebildeten Spats (Parallelepipeds[4]) berechnen als

$$|(u \times v) \cdot w| = |u \cdot (v \times w)|$$

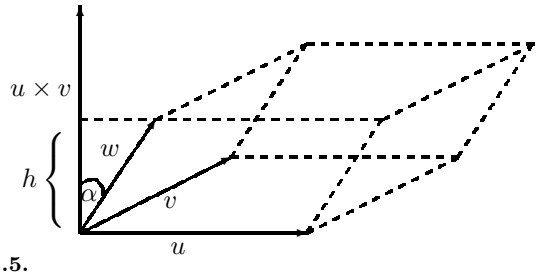


Abbildung 2.5.

[4]ein von 6 Parallelogrammen begrenzter Körper

2.1.2 Affine Kombinationen

Es ist sinnvoll, auch für Punkte des $\mathbb{R}^3$ eine Addition und eine skalare Multiplikation zu definieren:

$$(x_1, x_2, x_3) + (y_1, y_2, y_3) \stackrel{\text{def}}{=} (x_1 + y_1, x_2 + y_2, x_3 + y_3)$$

$$\lambda(x_1, x_2, x_3) \stackrel{\text{def}}{=} (\lambda \cdot x_1, \lambda \cdot x_2, \lambda \cdot x_3)$$

Dann ist $\frac{1}{2}(X + Y)$ der Mittelpunkt der Strecke XY (Abbildung 2.6.(1)) und $\frac{1}{3}(X + Y + Z)$ (Abbildung 2.6.(2)) der Schwerpunkt (Schnittpunkt der Seitenhalbierenden[5]) des Dreiecks XYZ.

Multipliziert man diese Ausdrücke aus, so erhalten sie die Form $\sum_{i=1}^{n} \lambda_i x_i$, wobei die Nebenbedingung $\sum_{i=1}^{n} \lambda_i = 1$ gilt. Eine solche Summe heißt eine *affine Kombination* der X_i.

Gilt zusätzlich noch $\lambda_i \in [0, 1]$ für alle $i = 1, \ldots, n$, so heißt sie *Konvexkombination*. Die Menge aller Konvexkombinationen von $X_1, \ldots X_n$ heißt die *konvexe Hülle* der $X_1, \ldots, X_n$ (Abbildung 2.6.(3))

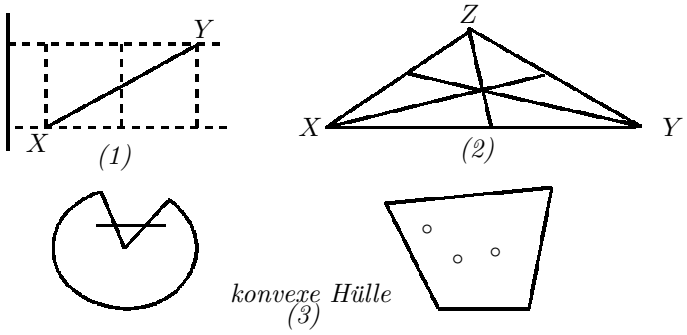


Abbildung 2.6.

Eine Menge von Skalaren λ_i, die den Bedingungen für Konvexkombinationen genügt, also

$$\lambda_i \in [0, 1], \qquad \sum_{i=1}^{n} \lambda_i = 1$$

heißt eine *Teilung der Eins*.

[5] Seitenhalbierende halbieren die Fläche (betrachte die halbierte Seite als Grundseite), damit bei massiven Körpern auch die Masse

Beispiel.

(a) Für beliebiges $t \in [0,1]$ bilden t, $1-t$ eine Teilung der Eins. Die Menge

$$XY = \{tX + (1-t)Y : t \in [0,1]\}$$

also die konvexe Hülle von X und Y, ist die Strecke XY (Verbindungsstrecke zwischen X und Y).

(b) Eine Konvexkombination $P = \lambda_1 X_1 + \lambda_2 X_2 + \lambda_3 X_3$ von drei Punkten liegt im Inneren des Dreiecks $X_1 X_2 X_3$:

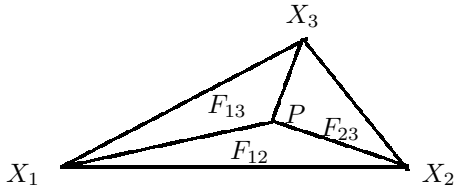


Abbildung 2.7.

Ist F die Gesamtfläche des Dreiecks, d. h. $F = F_{12} + F_{23} + F_{13}$, so gilt $\lambda_1 = \frac{F_{23}}{F}, \lambda_2 = \frac{F_{13}}{F}, \lambda_3 = \frac{F_{12}}{F}$.

Sie beschreiben also die relativen Flächenanteile und heißen daher auch die *baryzentrischen*[6] *Koordinaten* von P bzgl. X_1, X_2, X_3.

Der Schwerpunkt vom Dreiecks X_1, X_2, X_3 hat die baryzentrischen Koordinaten $\lambda_1 = \lambda_2 = \lambda_3 = \frac{1}{3}$, teilt also das Dreieck in drei flächengleiche Teildreiecke auf. □

Affine Kombinationen sind gut mit Verschiebungen (Translationen) verträglich.

Lemma 2.1.4. *Sei w ein Verschiebungsvektor und $\sum_{i=1}^{n} \lambda_i X_i$ eine affine Kombination. Dann gilt:*

$$\sum_{i=1}^{n} \lambda_i (X_i + w) = \left(\sum_{i=1}^{n} \lambda_i x_i \right) + w$$

Beweis. $\sum \lambda_i (X_i + w) = \sum (\lambda_i x_i + \lambda_i w) = \sum \lambda_i x_i + \underbrace{(\sum \lambda_i)}_{=1} \cdot w$ □

Für allgemeine Linearkombinationen gilt das **nicht**:

$$(X + w) + (Y + w) = X + Y + 2w \neq (X + Y) + w$$

wenn $w \neq 0$.
(Translation ist *keine* lineare Abbildung, vgl. Kapitel 2.1.3).

[6] Baryzentrum = Massenschwerpunkt

Wir wollen nun noch einige weitere Anwendungen von Konvexkombinationen sehen.

Beispiel. *Morphing*, d.h. schrittweise, quasi kontinuierliche, Überführung einer geometrischen Figur in eine andere.
Gegeben seien zwei Punktfolgen A_i, B_i, $(i = 1, \ldots, n)$, interpretiert als Polygonzüge A und B. Wir wollen A schrittweise in B überführen.

Abbildung 2.8.

Ein Parameter $t \in [0, 1]$ steuert die Überführung.
Definiere Zwischenpunkte $P_i(t) = (1 - t)A_i + tB_i$.
Es gilt: $P_i(0) = A_i$, $P_i(1) = B_i$.
Lässt man nun t in mehreren Schritten von 0 nach 1 laufen und zeigt jeweils das Zwischenpolygon $P(t)$, bestehend aus den $P_i(t)$ an, so „verwandelt“ sich A schrittweise in B. □

Beispiel. *Bézier-Kurven* dienen zur „glatten“ Darstellung von Freiformlinien, die gegebene Punkte verbinden. Für Bézier-Kurven 2. und 3. Grades verwendet man quadratische bzw. kubische Teilungen der Eins, die parametrisierte Kurvendarstellungen ergeben. Die Punkte, aus denen die entsprechenden Konvexkonstruktionen gebildet werden, heißen die *Kontrollpunkte* der jeweiligen Kurven.
Grad 2: $P(t) = (1 - t)^2 A_0 + 2(1 - t)tA_1 + t^2 A_2$
Grad 3: $Q(t) = (1 - t)^3 A_0 + 3(1 - t)^2 tA_1 + 3(1 - t)t^2 A_2 + t^3 A_3$.
Dass dabei tatsächlich Teilungen der Eins verwendet werden, erhält man mit der binomische Fomel wie folgt für beliebiges n und beliebiges $t \in [0, 1]$:

$$1 = 1^n = ((1 - t) + t)^n = \sum_{i=0}^{n} B_i^n,$$

wobei die

$$B_i^n = \left(\begin{array}{c} n \\ i \end{array} \right) (1 - t)^{n-i} t^i$$

Bernstein-Polynome heißen. □

Graphische Darstellung:

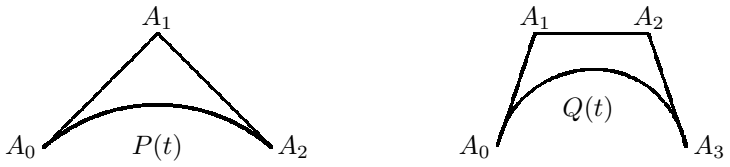


Abbildung 2.9.

2.1.3 Homogene Koordinaten und affine Abbildungen

Aus der Linearen Algebra ist bekannt, dass die grundlegenden Transformationen Drehung, Spiegelung, Streckung und Scherung lineare Abbildungen sind und soweit durch Matrizen beschrieben werden können. Wie wir gesehen haben, gilt das für Translationen nicht. Um die Implementierung einfach zu machen, möchte man aber gern alles durch Matrizen darstellen. Das gelingt mit dem Trick der homogenen Koordinaten.
Die Translation um den Vektor u schreibt sich koordinatenweise:

$$y_i = x_i + u_i = 1 \cdot x_i + u_i \cdot 1$$

Damit das als Anwendung einer Matrix auf einen Vektor gedeutet werden kann, muß die rechte Seite als Skalarprodukt eines Vektors, der von x abhängt, mit einer Matrixteile, die von u abhängt, gesehen werden. Das gelingt, indem man statt n-dimensionalen Vektoren und $m \times n$-Matrizen solche von einer Dimension höher verwendet:

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ ? \end{pmatrix} = \left(\begin{array}{ccc|c} 1 & 0 & 0 & u_1 \\ 0 & 1 & 0 & u_2 \\ 0 & 0 & 1 & u_3 \\ \hline ?_1 & ?_2 & ?_3 & ?_4 \end{array}\right) = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ 1 \end{pmatrix}$$

Die richtige Wahl für die ? ergibt sich daraus, dass zur weiteren Transformation y als 4. Koordinate ebenfalls wieder 1 haben muß, unabhängig von den x_i. Damit muß die 4. Matrixzeile (0001) lauten.
Die entstandene 4D-Matrix wird mit $T(u_1, u_2, u_3)$ bezeichnen.
Wir verwenden nun nur noch Matrizen mit obiger 4. Zeile.
Für sie gilt allgemein:

$$\left(\begin{array}{c|c} A & u \\ \hline \vec{0} & 1 \end{array}\right) \begin{pmatrix} x \\ 1 \end{pmatrix} = \begin{pmatrix} Ax + u \\ 1 \end{pmatrix}$$

so dass nun jede affine Abbildung durch eine lineare dargestellt werden kann. Wir geben nun die Form von A für die wichtigsten Transformationen an; für alle außer der Translation gilt $u = 0$.

Streckung (Skalierung)

$$A = \begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{pmatrix}; \quad \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \mapsto \begin{pmatrix} \lambda_1 x_1 \\ \lambda_2 x_2 \\ \lambda_3 x_3 \end{pmatrix}$$

Die zugehörige 4D-Matrix heiße:

$$S(\lambda_1, \lambda_2, \lambda_3)$$

Unterschiedliche Streckungsfaktoren entlang den Koordinatenachsen sind möglich; der Sonderfall $\lambda_1 = \lambda_2 = \lambda_3 = \lambda$ heißt *gleichmäßige* Streckung (zugehörige 4D-Matrix $S(\lambda)$). Negative λ_i erzeugen Spiegelungen an den Koordinatenebenen.

Beispiel $\lambda_1 = -1, \lambda_2 = \lambda_3 = 1$

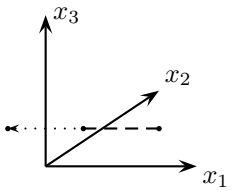


Abbildung 2.10.

Scherung

Im $\mathbb{R}^2$, parallel zur x_1-Achse:

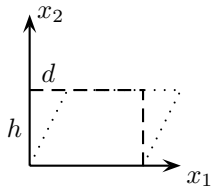


Abbildung 2.11.

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} x_1 + \sigma x_2 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 & \sigma \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \quad \text{mit } \sigma = \frac{d}{h}$$

Wie kommt man auf σ? Es muss gelten:

$$\begin{pmatrix} 1 & \sigma \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ h \end{pmatrix} = \begin{pmatrix} \sigma h \\ h \end{pmatrix} = \begin{pmatrix} d \\ h \end{pmatrix}$$

Allgemein im $\mathbb{R}^3$:

$$A = \begin{pmatrix} 1 & \sigma_1 & \sigma_2 \\ \sigma_3 & 1 & \sigma_4 \\ \sigma_5 & \sigma_6 & 1 \end{pmatrix}$$

Zugehörige 4D-Matrix:

$$SH(\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5, \sigma_6)$$

Drehung (Rotation)

Drehung um die x_3-Achse um φ im Gegenuhrzeigersinn:

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

Zugehörige 4D-Matrix:

$$R_{x_3}(\varphi)$$

analog für die anderen Achsen.

Drehung um eine normierte Drehachse $n = \begin{pmatrix} n_1 \\ n_2 \\ n_3 \end{pmatrix}$ durch den Ursprung und um φ:

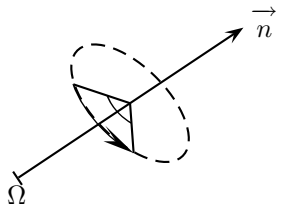


Abbildung 2.12.

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

mit $s = \sin\varphi, c = \cos\varphi, t = 1 - c$. Zugehörige 4D-Matrix: $R(\varphi, n_1, n_2, n_3)$

Allgemeinere Drehungen setzt man aus diesen Grundformen und Translationen zusammen.
Beispiel: Es soll eine Drehung um die Parallele zur x_3-Achse durch den Punkt $Z = (z_1, z_2, z_3)$ beschrieben werden. Man verschiebt zuerst durch die Translation $T(-z_1, -z_2, -z_3)$ den Raum so, dass Z in den Ursprung kommt, führt dort die Drehung $R_{x_3}(\varphi)$ aus und verschiebt anschließend mit $T(z_1, z_2, z_3)$ alles wieder zurück. Die Gesamttransformation wird also durch die Produktmatrix

$$T(z_1, z_2, z_3) \cdot R_{x_3}(\varphi) \cdot T(-z_1, -z_2, -z_3)$$

beschrieben:

$$\begin{pmatrix} 1 & 0 & 0 & z_1 \\ 0 & 1 & 0 & z_2 \\ 0 & 0 & 1 & z_3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} c & -s & 0 & 0 \\ s & c & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & -z_1 \\ 0 & 1 & 0 & -z_2 \\ 0 & 0 & 1 & -z_3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 & 0 & z_1 \\ 0 & 1 & 0 & z_2 \\ 0 & 0 & 1 & z_3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} c & -s & 0 & -z_1c + z_2s \\ s & c & 0 & -z_1s - z_2c \\ 0 & 0 & 1 & -z_3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} c & -s & 0 & -z_1c + z_2s + z_1 \\ s & c & 0 & -z_1s - z_2c + z_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

wobei wiederum $s = \sin\varphi$, $c = \cos\varphi$.

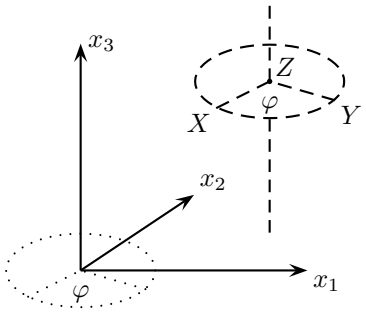
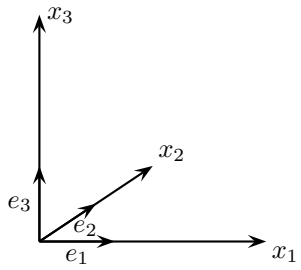


Abbildung 2.13.

2.1.4 Wechsel des Koordinatensystems

Wir stellen den Sachverhalt des vorigen Beispiels in einen allgemeineren Zusammenhang.



Sei $\overrightarrow{e}_i$ der Einheitsvektor entlang der x_i-Achse. Ein Punkt (x_1, x_2, x_3) wird dargestellt durch den Ortsvektor

$$x_1\overrightarrow{e_1} + x_2\overrightarrow{e_2} + x_3\overrightarrow{e_3} = (\overrightarrow{e_1}\ \overrightarrow{e_2}\ \overrightarrow{e_3}) \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

Wir wollen zu einem neuen Koordinatensystem übergehen, das durch drei linear unabhängige Vektoren f_1, f_2, f_3 bestimmt ist. Wie drücken sich Punkte im f-System aus?
Definiere $K : \mathbb{R}^3 \to \mathbb{R}^3$ durch $\overrightarrow{e_1} \mapsto \overrightarrow{f_1}$, $\overrightarrow{e_2} \mapsto \overrightarrow{f_2}$, $\overrightarrow{e_3} \mapsto \overrightarrow{f_3}$. Zu K gehört die Matrix

$$K = (\overrightarrow{f_1}\ \overrightarrow{f_2}\ \overrightarrow{f_3})$$

Da die $\overrightarrow{f_i}$ linear unabhängig sind, ist K umkehrbar, die Umkehrabbildung ist gegeben durch die inverse Matrix K^{-1}.
Um die Koordinaten von $X = (x_1, x_2, x_3)$ (im e-Koordinatensystem) bezüglich des f-Koordinatensystems zu bestimmen, rechnen wir

$$\overrightarrow{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = KK^{-1} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = (\overrightarrow{f_1}\ \overrightarrow{f_2}\ \overrightarrow{f_3})K^{-1} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

Daraus lesen wir, analog zu oben ab, daß X bezüglich der f-Basis die Koordinaten $K^{-1}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$ hat.

Hat umgekehrt ein Punkt bezüglich der f-Basis die Koordinaten $\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}$, so hat er bezüglich der e-Basis die Koordinaten $K\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}$.

Sei nun M eine Transformationsmatrix im f-System. Dann wird ihre Wirkung im e-System durch das Produkt KMK^{-1} beschrieben. Auf diese Weise ist die Matrix im letzten Beispiel entstanden.
Besonders einfach ist die Bestimmung von K^{-1}, wenn f ein Orthonormalsystem und damit K eine Orthogonalmatrix ist, denn dann ist $K^{-1} = K^T$.

2.1.5 Objekt- und Weltkoordinaten

Bei der Modellierung von Szenerien werden Koordinaten in mehreren unterschiedlichen Rollen verwendet.
Zunächst beschreibt man jedes einzelne Objekt, etwa durch Angabe der Koordinaten von Punkten, Linien und Flächen des Objekts relativ zu einem festen Ursprungspunkt, der zum Objekt gehören kann oder auch nicht. Diese Koordinaten heißen *Objektkoordinaten*; sie sind insbesondere auch für zusammengesetzte Objekte interessant (vgl. Abschnitt 2.1.8). Bei einem Quader könnte der Ursprung einer der Eckpunkte sein, bei einem Zylinder der Mittelpunkt einer der Deckflächen (oder der Zylindermittelpunkt, d.h. der Mittelpunkt der Zylinderachse), bei einem Kegel der Mittelpunkt der Bodenfläche, bei einer Kugel oder Kugelfläche der Kugelmittelpunkt (im letzten Fall ist der Ursprung nicht Teil des Objekts).
Szenerien werden dann aus Einzelobjekten zusammengesetzt. Dabei wählt man einen Ursprung Ω für die gesamte Szenerie und gibt dann zunächst für jedes Objekt O die Koordinaten seines Ursprungs Ω_O in Bezug auf den Szenerie-Ursprung Ω an. Die Koordinaten bezüglich Ω heißen *Szenerie-* oder kürzer *Weltkoordinaten*. Die Weltkoordinaten eines Punkts P im Objekt O erhält man, indem man zu den Objektkoordinaten von P bezüglich Ω_O die Weltkoordinaten von Ω_O addiert.

2.1.6 Zwei Sichten von Transformationsfolgen

Beim Aufbau komplexer Szenerien wechseln Definitionen von Körpern als Mengen von Ortsvektoren ab mit der Angabe von Transformationsmatrizen. Die mathematische Bedeutung ist, dass die Komposition aller Matrizen vor einer Körperdefinition auf diesen Körper angewandt wird.
Dieses Denken „von rechts nach links“ möchte man bei der Aufschreibung von links nach rechts aber eigentlich nicht durchführen. Gibt es also auch eine vernünftige Deutung von Transformationsfolgen von links nach rechts?
Dabei helfen uns die Betrachtungen aus Abschnitt 2.1.4 über Wechsel des Koordinatensystems. Eine 3D-Matrix $K = (\overrightarrow{f_1}\overrightarrow{f_2}\overrightarrow{f_3})$ kann auch gedeutet werden als Übergang in ein neues Koordinatensystem mit den Basisvektoren $\overrightarrow{f_i}$. Also

kann eine Folge KLM von Matrizen als fortgesetzter Wechsel des Koordinatensystem gemäß K, L, M gedeutet werden. Damit lassen sich komplexe Transformationsfolgen oft leichter verstehen und schreiben. Die Implementierung, z.B. in OpenGL, sorgt dann dafür, dass am Ende alles korrekt in Weltkoordinaten zurückgerechnet wird.

Beispiel
Im 2D betrachten wir die Folge TR, wobei T eine Translation um den Vektor $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ und R eine Rotation um den Ursprung um 45° im Gegenuhrzeigersinn ist.
Deutung von rechts nach links (erst zeichnen, dann transformieren)

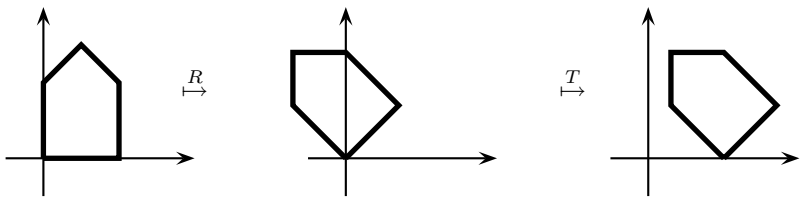


Abbildung 2.14.

Deutung von links nach rechts (zuerst Koordinatensysteme ändern, dann zeichnen)

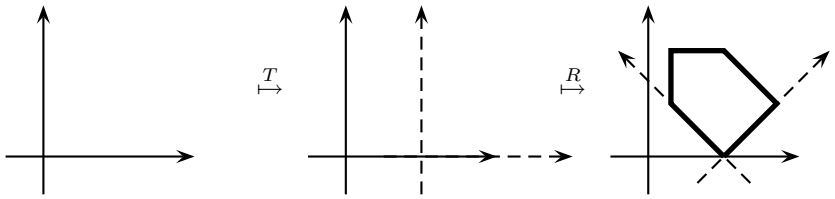


Abbildung 2.15.

□

Diese Sicht wird uns vor allem bei der Darstellung hierarchischer Objekte helfen.

2.1.7 Kugelkoordinaten

Besonders im Zusammenhang mit Drehungen ist manchmal eine andere Koordinatendarstellung günstig. Man beschreibt in *Kugelkoordinaten* einen Punkt durch seinen Abstand r vom Ursprung und zwei Winkel ϑ und φ:

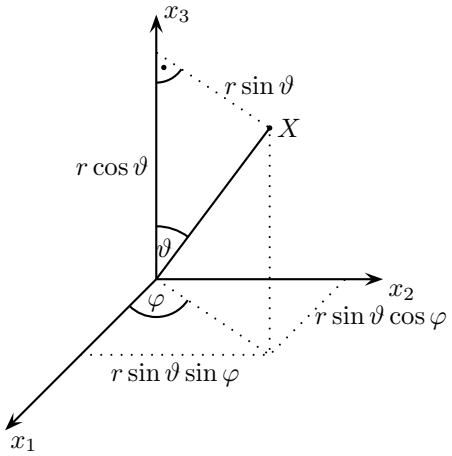


Abbildung 2.16.

Die Umrechnungsformeln sind wie folgt:

$$(r, \vartheta, \varphi) \mapsto (r \sin \vartheta \cos \varphi, r \sin \vartheta \sin \varphi, r \cos \vartheta)$$

$$x = (x_1, x_2, x_3) \mapsto (\|x\|, \arccos \frac{x_3}{\|x\|}, \varphi)$$

mit

$$\varphi = \begin{cases} \frac{\pi}{2} & x_1 = 0 \wedge x_2 \geq 0 \\ -\frac{\pi}{2} & x_1 = 0 \wedge x_2 < 0 \\ \arctan \frac{x_2}{x_1} & \text{sonst} \end{cases}$$

Als Übung leite man die Rotationsmatrix $R_{x_3}(\varphi)$ mittels der Kugelkoordinatendarstellung her.

2.1.8 Hierarchien und Szenengraphen

Oft sind Objekte aus mehreren Teilobjekten zusammengesetzt. Wird das Gesamtobjekt bewegt, so auch seine Teilobjekte, wobei die Teile noch zusätzlich eigene Bewegungen ausführen können.

Beispiel Fahrrad
Beim Schieben bewegt sich das gesamte Fahrzeug, wobei sich die Räder noch relativ zum Rahmen drehen. □
Das modelliert man durch hierarchische Strukturierung der Objekte, entweder baumartig oder, zur Effizienzsteigerung, in azyklisch gerichteten Graphen, wenn gleichartige Teilobjekte mehrfach auftreten.
Transformationen eines Objekts wirken dann gleichartig auch auf alle seine Teilobjekte. Zusätzliche Transformationen eines Teilobjekts vererben sich dann weiter nach unten auf dessen Teilobjekte, nicht jedoch auf übergeordnete Objekte.

Beispiel Sonnensystem
Hauptobjekt: Sonne (Mittelpunkt = Ursprung)
Teilobjekte: Planeten und Monde

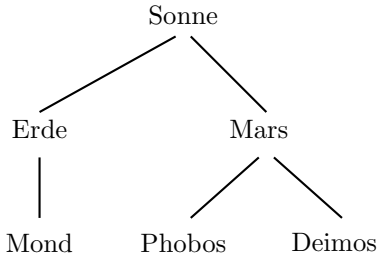


Abbildung 2.17.

Die Planeten drehen sich samt ihren Monden um die Sonne, wobei aber noch Eigenrotationen der Monde um den jeweiligen Planeten hinzukommen. Anders als beim Fahrrad können hier keine Teilobjekte mehrfach verwendet werden, da die Größen und Rotationsgeschwindigkeiten unterschiedlich sind. □

2.2 Projektionen und Kameramodelle

Nachdem alle Objekte einer Szenerie mittels der besprochenen Transformationen im 3D-Weltkoordinatensystem platziert sind, muss die Szenerie aus verschiedenen Blickwinkeln auf eine 2D-Bildfläche projiziert werden. Dazu verwendet man eine virtuelle Kamera, die wie folgt definiert ist:

- Ihre Position ist durch die Koordinaten eines Punkts gegeben.
- Der Bildausschnitt ist rechtwinklig.
- Ihr Schärfebereich ist unendlich groß.

Man verwendet zwei Typen von Projektionen, die *Parallel-* und die *Zentralprojektion*:

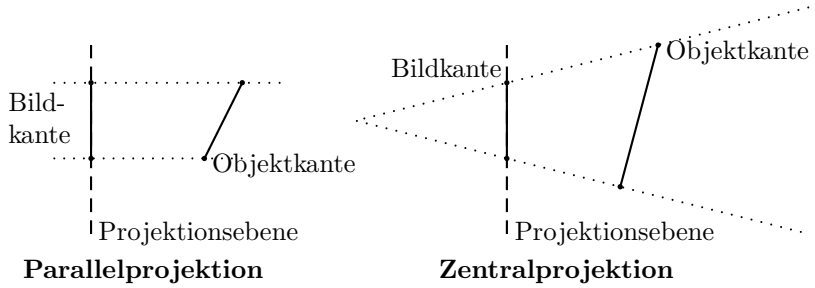


Abbildung 2.18.

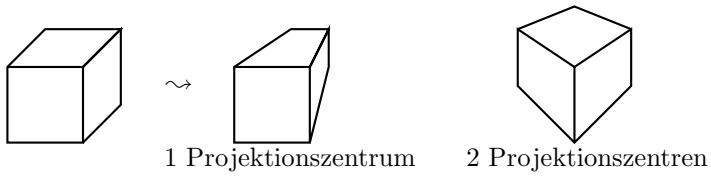


Abbildung 2.19. *Beispiel für Zentralprojektion*

2.2.1 Parallelprojektion

Hier gibt es zwei Untertypen:

- bei *orthogonaler* Projektion stehen die Projektionsstrahlen senkrecht auf der Projektionsebene (vgl. obiges Beispiel)
- bei *schiefwinkliger* Projektion nicht.

Die Projektionsrichtung wird durch einen normierten Vektor $\overrightarrow{p}$ gegeben. Für $\overrightarrow{p}$, die parallel zu den Koordinatenachsen sind, ergeben sich als Bilder die sechs *Hauptrisse* (Grundriss, Aufriss, Seitenriss) eines Objekts.

Beispiel
Transformationsmatrix für die Draufsicht (Grundriss) Wir behandeln zuerst den einfachen Fall, dass die Projektionsebene mit der x_1x_2-Ebene zusammenfällt und dort auch das x_1x_2-Koordinatensystem gewählt wird. Dann werden bei der Projektion einfach alle x_3-Koordinaten auf 0 abgebildet, die übrigen bleiben gleich:

$$V_D = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Das D in V_D steht hierbei für „Draufsicht“.

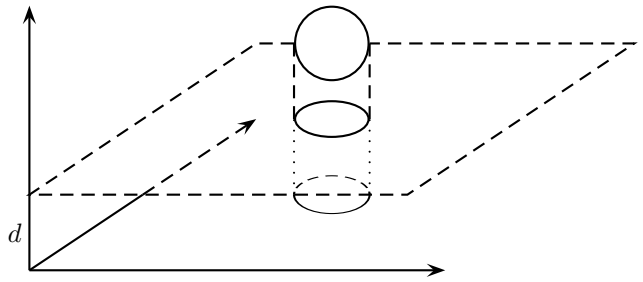


Abbildung 2.20. *Parallelprojektion Grundriss*

Wählt man eine andere Ebene parallel zur x_1x_2-Ebene, etwa die Ebene $x_3 = d$, so werden alle x_3-Koordinaten auf d abgebildet. Soll schließlich der Ursprung des Bildkoordinatensystems nicht $(0, 0, d)$ sein, sondern (b, c, d), so ist noch ein Translationsanteil hinzuzufügen.

$$V_D = \begin{pmatrix} 1 & 0 & 0 & b \\ 0 & 1 & 0 & c \\ 0 & 0 & 0 & d \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

□

Für den Fall, dass $\overrightarrow{p}$ nicht parallel zu einer Koordinatenachse ist, legt man durch einen zusätzlichen Vektor $\overrightarrow{w}$ fest, wo bezüglich der Kamera „oben“ ist;

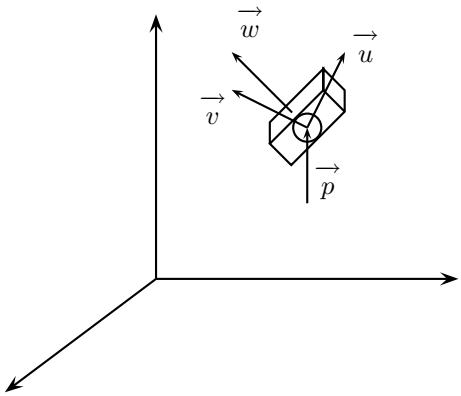


Abbildung 2.21.

dieser Vektor muss nicht auf $\overrightarrow{p}$ senkrecht stehen. Wir müssen nun die Achsenrichtungen des Bildkoordinatensystems bestimmen; sie sollen ein normiertes rechtshändiges System bilden. Verabredungsgemäß soll die Normale der Bildebene immer parallel zu $\overrightarrow{p}$ sein; als Richtung der 3. Achse wählt man also

$$\overrightarrow{n} = -\overrightarrow{p}$$

Der Vektor $\overrightarrow{u}$ soll auf $\overrightarrow{n}$ und $\overrightarrow{w}$ senkrecht stehen:

$$\overrightarrow{u} = \frac{\overrightarrow{w} \times \overrightarrow{p}}{\| \overrightarrow{w} \times \overrightarrow{p} \|}$$

Als dritte Richtung wählen wir

$$\overrightarrow{v} = \overrightarrow{n} \times \overrightarrow{u} = -(\overrightarrow{n} \times \overrightarrow{u})$$

(als Produkt normierter Vektoren bereits normiert).

Nun betrachten wir wieder den Fall, dass der Ursprung des Bildkoordinatensystems mit dem Weltkoordinatensystem zusammenfällt. Die Transformation K ist gegeben durch

$$K = \begin{pmatrix} u_1 & v_1 & n_1 & 0 \\ u_2 & v_2 & n_2 & 0 \\ u_3 & v_3 & n_3 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Die Koordinaten eines Punktes mit Weltkoordinaten (x_1, x_2, x_3) erhält man durch $K^{-1} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$. Weil K nach Konstruktion eine Orthonormalmatrix ist, gilt

$$K^{-1} = K^T = \begin{pmatrix} u_1 & u_2 & u_3 & 0 \\ v_1 & v_2 & v_3 & 0 \\ n_1 & n_2 & n_3 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Die gesamte Projektion auf die $\overrightarrow{u}\overrightarrow{v}$-Ebene ist dann durch $V_D \cdot K^T$ gegeben.

Beispiel

$$\overrightarrow{p} = \frac{\sqrt{2}}{2} \cdot \begin{pmatrix} -1 \\ -1 \\ 0 \end{pmatrix} \qquad \overrightarrow{w} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$$

Dann folgt

$$\overrightarrow{u} = \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix} \qquad \overrightarrow{v} = \frac{\sqrt{2}}{2} \cdot \begin{pmatrix} -1 \\ 1 \\ 0 \end{pmatrix} \qquad \overrightarrow{n} = \frac{\sqrt{2}}{2} \cdot \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$$

Also

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

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

$$P = V_D \cdot K^T = \begin{pmatrix} 0 & 0 & -1 & 0 \\ -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Projektion eines Einheitswürfels:

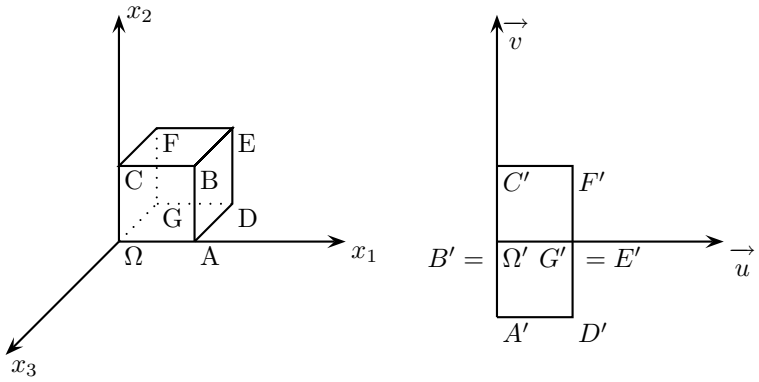


Abbildung 2.22.

Für $\overrightarrow{p} = \frac{\sqrt{3}}{3} \cdot \begin{pmatrix} -1 \\ -1 \\ -1 \end{pmatrix}$ und $\overrightarrow{w} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$ ergibt sich

$$\overrightarrow{u} = \frac{\sqrt{2}}{2} \cdot \begin{pmatrix} -1 \\ 1 \\ 0 \end{pmatrix} \qquad \overrightarrow{v} = \frac{\sqrt{6}}{6} \cdot \begin{pmatrix} -1 \\ -1 \\ 2 \end{pmatrix} \qquad \overrightarrow{n} = \frac{\sqrt{3}}{3} \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

$$K = \begin{pmatrix} -\frac{\sqrt{2}}{2} & -\frac{\sqrt{6}}{6} & \frac{\sqrt{3}}{3} & 0 \\ \frac{\sqrt{2}}{2} & -\frac{\sqrt{6}}{6} & \frac{\sqrt{3}}{3} & 0 \\ 0 & \frac{\sqrt{6}}{3} & \frac{\sqrt{3}}{3} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad P = \begin{pmatrix} -\frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & 0 & 0 \\ -\frac{\sqrt{6}}{6} & -\frac{\sqrt{6}}{6} & \frac{\sqrt{6}}{3} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

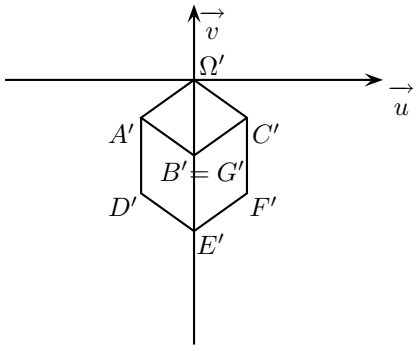


Abbildung 2.23.

$$A' = \begin{pmatrix} -\frac{\sqrt{2}}{2} \\ -\frac{\sqrt{6}}{6} \end{pmatrix} \qquad B' = \begin{pmatrix} 0 \\ -\frac{\sqrt{6}}{3} \end{pmatrix} \qquad C' = \begin{pmatrix} \frac{\sqrt{2}}{2} \\ -\frac{\sqrt{6}}{6} \end{pmatrix}$$

$$D' = \begin{pmatrix} -\frac{\sqrt{2}}{2} \\ -\frac{\sqrt{6}}{2} \end{pmatrix} \qquad E' = \begin{pmatrix} 0 \\ -\frac{2\sqrt{6}}{3} \end{pmatrix} \qquad F' = \begin{pmatrix} \frac{\sqrt{2}}{2} \\ -\frac{\sqrt{6}}{2} \end{pmatrix} \qquad G' = \begin{pmatrix} 0 \\ -\frac{\sqrt{6}}{3} \end{pmatrix}$$

□

2.2.2 Zentralprojektion

Bei dieser Projektionsart berücksichtigt man auch perspektivische Verzerrung. Das bringt mit sich, dass diese Projektionsart nicht parallelentreu ist. Damit kann sie aber keine affine Abbildung sein:

Lemma 2.2.1. *Jede affine Abbildung ist parallelentreu*

Beweis. Jede affine Abbildung hat die Form

$$f(\overrightarrow{x}) = A\overrightarrow{x} + \overrightarrow{t}.$$

Betrachte nun zwei parallele Geraden, d. h. Geraden mit gleichem Richtungsvektor $\overrightarrow{u}$:

$$g = \left\{\lambda\overrightarrow{u} + \overrightarrow{v} : \lambda \in \mathbb{R}\right\} \qquad h = \left\{\lambda\overrightarrow{u} + \overrightarrow{w} : \lambda \in \mathbb{R}\right\}$$

Wir berechnen die Bilder von g und h:

$$f(\lambda\overrightarrow{u} + \overrightarrow{v}) = A(\lambda\overrightarrow{u} + \overrightarrow{v}) + \overrightarrow{t} = A\lambda\overrightarrow{u} + A\overrightarrow{v} + \overrightarrow{t} = \lambda A\overrightarrow{u} + (A\overrightarrow{v} + \overrightarrow{t})$$

$$f(\lambda\overrightarrow{u} + \overrightarrow{w}) = \lambda A\overrightarrow{u} + (A\overrightarrow{w} + \overrightarrow{t})$$

Beide Bildgeraden haben also den gleichen Richtungsvektor $A\overrightarrow{u}$ und sind somit wieder parallel. □

Wir kommen also zur Beschreibung der Zentralprojektion nicht mit dem bisherigen Typ von 4D-Matrizen aus. Glücklicherweise zeigt sich, dass man aber mit allgemeineren Matrizen arbeiten kann, so dass die Einheitlichkeit der Implementierung gewahrt bleibt.
Man verwendet dazu auch allgemeinere homogene Koordinaten als bisher: Es sind jetzt beliebige Zahlen $\neq 0$ als 4. Koordinaten zulässig.
Ein 4D-Vektor $\begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$ steht für den 3D-Vektor $\begin{pmatrix} \frac{x}{w} \\ \frac{y}{w} \\ \frac{z}{w} \end{pmatrix}$.
Also ist w ein Skalierungsfaktor, der natürlich $\neq 0$ sein muss, damit die Divisionen möglich sind.
Bisher hatten wir stets $w = 1$ verwendet, was zu obiger Interpretation passt.
Der Übergang von 3D-Koordinaten zu homogenen 4D-Koordinaten erfolgt durch

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \mapsto \begin{pmatrix} wx \\ wy \\ wz \\ w \end{pmatrix}$$

mit beliebigem $w \neq 0$.
Wir betrachten nun den Spezialfall der Zentralprojektion mit Projektionszentrum im Ursprung und einer Bildebene parallel zur xy-Ebene in Entfernung d in negativer z-Richtung vom Ursprung.

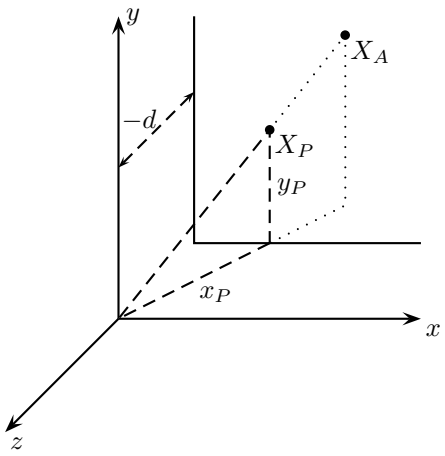


Abbildung 2.24.

Wir bestimmen das Bild eines Punkts $X_A = (x_A, y_A, z_A)$. Dazu wird X_A mit dem Projektionszentrum Ω verbunden; der Durchstoßpunkt $X_P = (x_P, y_P, z_P)$ von $X_A\Omega$ durch die Bildebene ist der Bildpunkt.
Nach dem Vierstreckensatz[7] erhält man, wegen $z_P = -d$,

$$\frac{x_P}{x_A} = \frac{-d}{z_A} \quad \text{und analog} \quad \frac{y_p}{y_A} = \frac{-d}{z_A}$$

$$\text{Also gilt} \begin{pmatrix} x_P \\ y_P \\ z_P \end{pmatrix} = \begin{pmatrix} \frac{-dx_A}{z_A} \\ \frac{-dy_A}{z_A} \\ -d \end{pmatrix}$$

Beim Übergang zu homogenen Koordinaten skalieren wir noch gleichmäßig um den Faktor $-z_A$, um eine einheitliche Form aller Koordinaten zu erhalten:

$$\begin{pmatrix} x_P \\ y_P \\ z_P \\ w_P \end{pmatrix} = \begin{pmatrix} dx_A \\ dy_A \\ dz_A \\ -z_A \end{pmatrix} = \begin{pmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & d & 0 \\ 0 & 0 & -1 & 0 \end{pmatrix} \cdot \begin{pmatrix} x_A \\ y_A \\ z_A \\ 1 \end{pmatrix}$$

Man beachte die völlig neuartige 4. Zeile der entstandenen Transformationsmatrix!
Da in unserer neuen Sicht von homogenen Koordinaten eine gleichzeitige Skalierung in allen 4 Werten die 3D-Bedeutung nicht ändert, kann man auch die durch d dividierte Matrix

$$P_d = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -\frac{1}{d} & 0 \end{pmatrix}$$

verwenden, die wesentlich einfacher zu handhaben ist.

[7] Strahlensatz

Beispiel Zentralprojektion eines achsenparallelen Würfels der Kantenlänge 2, zentriert um die z-Achse im Abstand 4 vom Ursprung auf eine Bildebene im Abstand $d = 2$.

$$\begin{array}{ll} A = (-1,-1,-4) & E = (-1,-1,-6) \\ B = (1,-1,-4) & F = (1,-1,-6) \\ C = (1,1,-4) & G = (1,1,-6) \\ D = (-1,1,-4) & H = (-1,1,-6) \end{array}$$

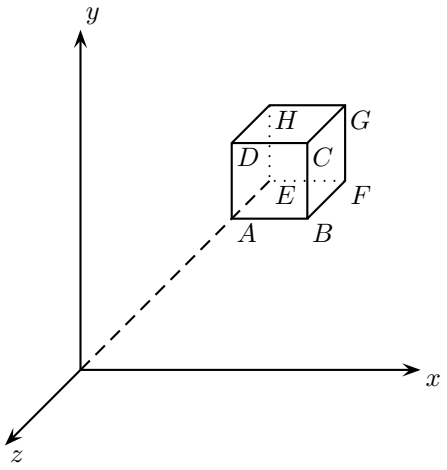


Abbildung 2.25.

Bereinigte Transformationsmatrix ($d = 2$):

$$P_2 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -\frac{1}{2} & 0 \end{pmatrix}$$

$$A' = P_2 \cdot \begin{pmatrix} -1 \\ -1 \\ -4 \\ 1 \end{pmatrix} = \begin{pmatrix} -1 \\ -1 \\ -4 \\ 2 \end{pmatrix}$$

4D/3D-Koordinaten der Bildpunkte:

$A' = (-1, -1, -4, 2)$,	$A' = (-\frac{1}{2}, -\frac{1}{2}, -2)$,
$B' = (1, -1, -4, 2)$,	$B' = (\frac{1}{2}, -\frac{1}{2}, -2)$,
$C' = (1, 1, -4, 2)$,	$C' = (\frac{1}{2}, \frac{1}{2}, -2)$,
$D' = (-1, 1, -4, 2)$,	$D' = (-\frac{1}{2}, \frac{1}{2}, -2)$,
$E' = (-1, -1, -6, 3)$,	$E' = (-\frac{1}{3}, -\frac{1}{3}, -2)$,
$F' = (1, -1, -6, 3)$,	$F' = (\frac{1}{3}, -\frac{1}{3}, -2)$,
$G' = (1, 1, -6, 3)$,	$G' = (\frac{1}{3}, \frac{1}{3}, -2)$,
$H' = (-1, 1, -6, 3)$,	$H' = (-\frac{1}{3}, \frac{1}{3}, -2)$,

Darstellung auf der Bildebene:

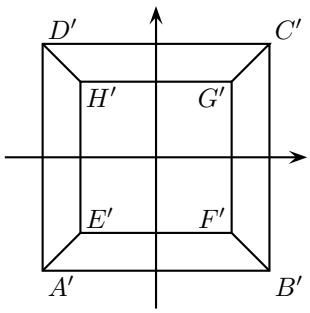


Abbildung 2.26.

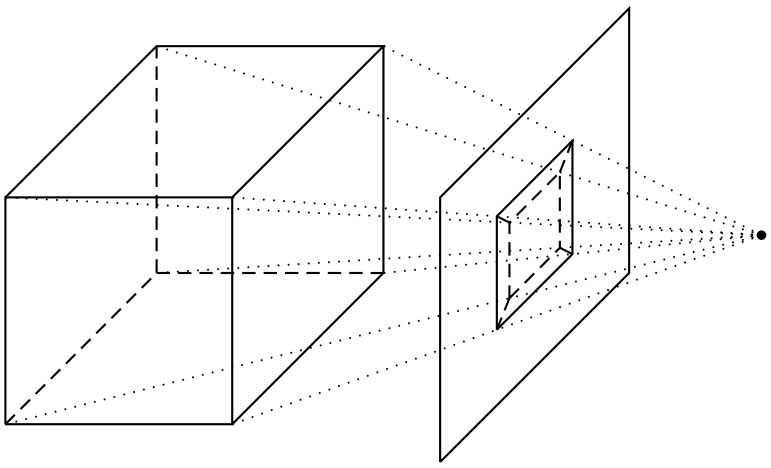


Abbildung 2.27. □

Der allgemeine Fall der Zentralprojektion ergibt sich aus dem speziellen wieder durch geeignete Koordinatentransformation.

Beispiel Nun soll die Bildebene mit der xy-Ebene zusammenfallen, wobei das Projektionszentrum wieder im Abstand d vor der Bildebene auf der z-Achse liegen soll.

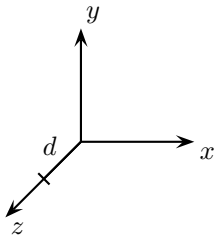


Abbildung 2.28.

Das erreicht man durch eine einfache Translation

$$T = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Als Gesamttransformation für die Zentralprojektion erhält man

$$TP_dT^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -\frac{1}{d} & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -d \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{1}{d} & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -d \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{1}{d} & 1 \end{pmatrix} \qquad \square$$

Wir verzichten auf die Behandlung weiterer Fälle; die Technik wurde bereits bei der Parallelprojektion demonstriert. Wir führen lediglich einige wichtige Sprechweisen für den allgemeinen Fall ein. Gegeben sei eine Projektionsebene Π und ein *Zentral-* oder *Augpunkt* Z. Die abzubildenden Objekte denkt man sich auf einer zu Π senkrechten *Grundebene* Γ aufgestellt. Der Normalenvektor zu Π in Richtung Z sei $\overrightarrow{n}$. Die Gerade durch Z in Richtung $\overrightarrow{n}$ schneidet Π im *Hauptpunkt* H; der Abstand $\|\overrightarrow{ZH}\|$ heißt *Augdistanz*.

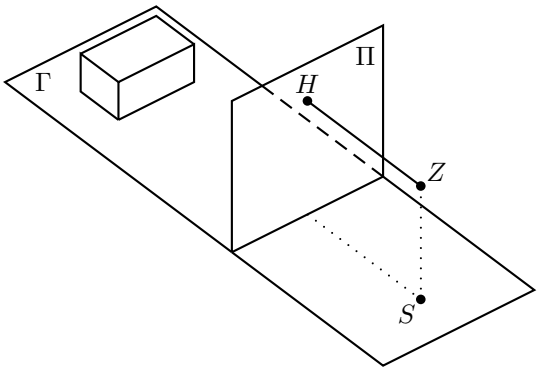


Abbildung 2.29.

Der Fußpunkt des Lots von Z auf Γ ist der *Standpunkt* S; der Abstand $\sigma = \|\overrightarrow{ZS}\|$ heißt *Standhöhe*. Für großes σ erhält man die *Vogelperspektive*, für kleines σ die *Froschperspektive*. Die Parallele zu Γ durch H in Π heißt *Horizont*.
Die Bilder aller Parallelen, die Π schneiden, laufen auf einen gemeinsamen Punkt zu, der ihr *Fluchtpunkt* heißt. Er ist der Durchstoßpunkt derjenigen Parallelen f, die durch Z verläuft. In der Draufsicht:

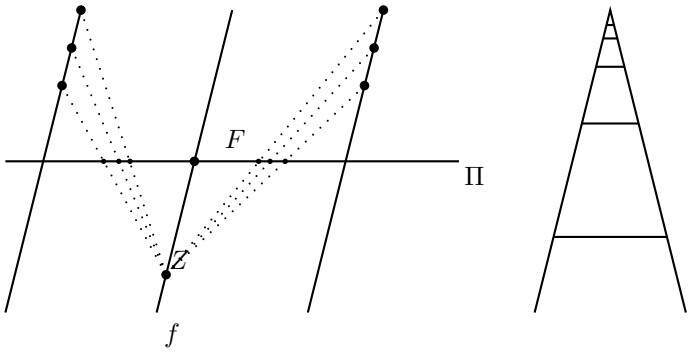


Abbildung 2.30.

Ganz f wird durch die Projektion auf den Fluchtpunkt F abgebildet, die übrigen Parallelen auf Halbgeraden ohne Endpunkt.
Nur unter den zu Π parallelen Geraden werden Parallelen wieder auf Parallelen abgebildet.
Die Schar der Geraden senkrecht zu Π heißt Schar der *Tiefengeraden*; ihr Fluchtpunkt ist der Hauptpunkt H.
Je nachdem, wie die Bildebene zum Weltkoordinatensystem liegt, ergeben sich bei der Projektion quaderförmiger Objekte unterschiedlich viele Fluchtpunkte für die Quaderkanten, nämlich so viele wie Π Koordinatenachsen schneidet:

- Zweipunktperspektive: Bildebene verläuft senkrecht, schneidet aber die anderen Koordinatenachsen:

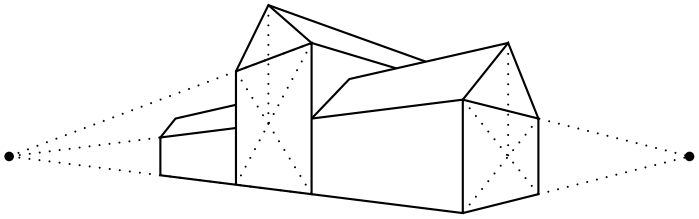


Abbildung 2.31.

- Dreipunktperspektive, Π schneidet alle 3 Achsen

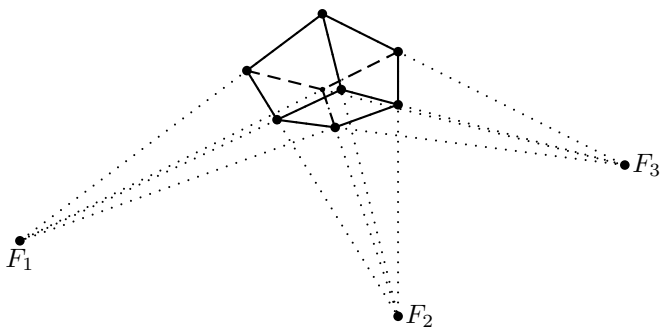


Abbildung 2.32.

2.2.3 Die virtuelle Kamera

Zur direkteren Definition einer Kamera gibt man meist etwas andere Parameter an, als die bisher besprochenen (unabhängig von der Projektionsart):

- den Augpunkt Z (*eye point*),
- einen Hauptpunkt H (*point of interest*),
- den Oben-Vektor $\overrightarrow{w}$,

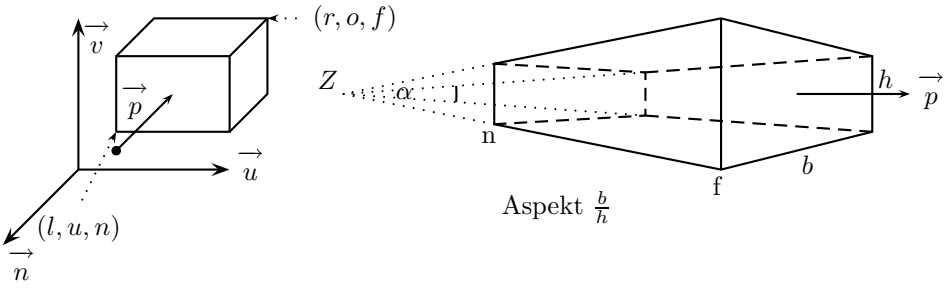
alles in Weltkoordinaten.

Der Projektionsvektor $\overrightarrow{p}$ und der Normalenvektor $\overrightarrow{n}$ der Bildebene ergeben sich als

$$\overrightarrow{p} = \frac{\overrightarrow{ZH}}{\|\overrightarrow{ZH}\|} \qquad \overrightarrow{n} = -\overrightarrow{p}$$

Die beiden anderen Koordinatenrichtungen $\overrightarrow{u}$ und $\overrightarrow{v}$ des Bildkoordinatensystems werden berechnet wie bei der Parallelprojektion beschrieben.

Außerdem wird ein Ausschnitt aus der gesamten Welt definiert, das *Sichtvolumen* oder *Frustum*. Damit können sehr weit von der Kamera entfernte und unmittelbar vor ihr liegende Objekte ausgeblendet werden. Bei der Parallelprojektion ist das Sichtvolumen ein Quader, bei der Zentralprojektion ein Pyramidenstumpf, jeweils zentriert um die $\overrightarrow{n}$-Achse.
In beiden Fällen gibt man die Position der Deckflächen durch zwei Werte n („nah“) und f („fern“) an, die aussagen, in welchen Parallelebenen zur $\overrightarrow{u}\,\overrightarrow{v}$-Ebene die Deckflächen liegen. Bei der Parallelprojektion gibt man außerdem die $\overrightarrow{u}$- und $\overrightarrow{v}$-Koordinaten der linken unteren und der rechten oberen Ecke der Deckfläche an, bei der Zentralprojektion den Öffnungswinkel des Pyramidenstumpfs in $\overrightarrow{v}$-Richtung und den *Aspekt*, d. h. das Verhältnis der Seitenlängen der Deckflächen.



Frustum bei Parallelprojektion Frustum bei Zentralprojektion

Abbildung 2.33.

Natürlich muss $|f| > |n|$ gelten.
Eine Hauptaufgabe wird nun sein, zu berechnen, welche Teile welcher Weltobjekte im Sichtvolumen liegen; die Techniken hierzu heißen *Streichen* (*culling*) und *Kappen* (*clipping*) und werden in den folgenden Abschnitten besprochen.

2.3 Kappen und Rasterung

2.3.1 Pixel und ihre Koordinaten

Wir behandeln zunächst den Übergang vom Projektionsbild zur Bildschirmdarstellung. Der rechteckige Ausschnitt aus der Projektionsebene, der am Schirm dargestellt werden soll, also die Vorderfläche des Frustums, heißt auch *Fenster*. Das Darstellungsziel ist ein zweidimensionales diskretes Raster von Bildelementen, genannt *Pixel*. Es wird von der Grafikkarte des Rechners verwaltet. Meist nimmt man an, dass die Pixel alle gleich große Rechtecke mit gleichem Seitenverhältnis sind. Wir betrachten sogar nur quadratische Pixel. Außerdem trägt jedes Pixel nur eine Farbe.
Diese Farbinformation wird in einem zweidimensionalen Feld, dem *Bild(schirm)-puffer (framebuffer)* gehalten und durch binär codierte Ganzzahlwerte dargestellt. Die *Farbtiefe*, d. h. die Anzahl der darstellbaren Farben, hängt von der Anzahl der verwendeten Bits ab.
Als *(virtuelle) Bildschirmauflösung* bezeichnet man die Dimensionen Breite und Höhe des Bildpuffers. Ein gängiges Seitenverhältnis ist 4:3 ($640 \times 480, 800 \times 600, 1024 \times 768, 1280 \times 960, 1600 \times 1200$).
Die Pixel werden über das *Geräte-Koordinatensystem* angesteuert. Es verwendet nur natürlichzahlige Koordinaten. Der Ursprung liegt in der linken oberen Ecke, die x-Werte laufen von links nach rechts, die y-Werte von oben nach unten.
Zur Beschleunigung wird der Bildschirmpuffer heute direkt im Speicher der Grafikkarte gehalten. Da die virtuelle Auflösung meist veränderlich ist, muss in einem Zwischenschritt das Bild noch in die *physikalische Auflösung* des Bildschirms umgerechnet werden; auch dies geschieht direkt in der Grafikkarte. Je nach dem Verhältnis von virtueller und physikalischer Auflösung haben die Pixel unterschiedliche physikalische Größen. Bei extremen Verhältnissen kommt es zu Qualitätsverlust in der Darstellung. Wir verstehen im Folgenden unter „Auflösung“ stets die virtuelle.

2.3.2 Die Fenster/Ausschnitt-Transformation

Oft will man das Fenster nicht auf dem gesamten zur Verfügung stehenden Schirm abbilden, sondern auf einem Teilbereich, dem *Ausschnitt (viewport)*, gegeben durch die Gerätekoordinaten $x_{\min}, x_{\max}, y_{\min}, y_{\max}$.
Wir geben an, wie man die Koordinaten (x, y) eines Fensterpunkts $P = (u, v)$ im Ausschnitt bestimmt (Window-Viewport-Transformation):

1. Translation des Fensters in den Ursprung des Geräte-Koordinatensystems (die Fensterecken seien durch $u_{\min}, u_{\max}, v_{\min}, v_{\max}$ bestimmt):

$$u' = u - u_{\min} \qquad v' = v - v_{\min}$$

2. Skalierung des Fensterbereichs auf die Ausschnittgröße:

$$u'' = \frac{x_{\max} - x_{\min}}{u_{\max} - u_{\min}} \cdot u' \qquad v'' = \frac{y_{\max} - y_{\min}}{v_{\max} - v_{\min}} \cdot v'$$

3. Translation des skalierten Bereichs an die Ausschnittposition:

$$x = u'' + x_{\min} \qquad y = -v'' + y_{\max}$$

Bei Schritt 2 kommt es zu Verzerrungen, wenn die Seitenverhältnisse von Fenster und Ausschnitt nicht übereinstimmen.
Ein nichttriviales Problem ist die Rundung der in Schritt 3 entstandenen Koordinaten auf im Geräte-Koordinatensystem verwendbare ganzzahlige Werte (Genaueres dazu später).

2.3.3 Kappen und Streichen

Die Definition des Sichtvolumens begrenzt bereits vor der Projektion die Anzahl der Objekte, die dargestellt werden müssen. Das *Streichen* blendet die übrigen Objekte aus, wozu auch verdeckte Flächen gehören. Das *Kappen* beschneidet die Objekte, die das Frustum treffen, auf den Teil, der im Frustum liegt.

2.3.3.1 Streichen verdeckter Flächen

Beim *Rückseiten-Streichen (backface culling)* werden die vom Augpunkt Z aus nicht sichtbaren Rückseiten von Polygonen ausgeblendet. Bei polygonalen Objekten reduziert das die Szenenkomplexität im Mittel um 50%.
Ein einfaches Verfahren dazu beruht auf der Annahme, dass polygonale Deckflächen so orientiert sind, dass ihr Normalenvektor ins Äußere des jeweiligen Körpers zeigt. Dann sind vom Augpunkt Z aus genau diejenigen Flächen sichtbar, deren Normalenvektor $\overrightarrow{n}$ mit dem Verbindungsvektor $\overrightarrow{p} = \overrightarrow{PZ}$ von Z zu einem beliebigen Flächenpunkt P einen Winkel mit Betrag $< 90°$ einschließt:

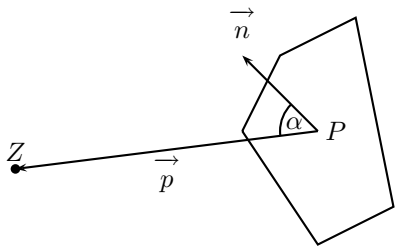


Abbildung 2.34. *backface culling*

Wird der Winkel gleich 90°, so entartet das Polygon in der Projektion zu einer Strecke.
Wegen $\left\langle \overrightarrow{n}, \overrightarrow{p} \right\rangle = \| \overrightarrow{n} \| \cdot \| \overrightarrow{p} \| \cdot \cos\alpha$ wird also das Polygon ausgeblendet, wenn $\left\langle \overrightarrow{n}, \overrightarrow{p} \right\rangle < 0$. Als P kann man eine Polygonecke wählen, $\overrightarrow{n}$ ergibt sich als Vektorprodukt zweier Polygonkanten. Damit ist dieser Test sehr effizient zu realisieren.
In OpenGL wird dieses Verfahren durch `glEnable(GL_CULL_FACE)` aktiviert. Mit `glCullFace(GL_BACK)` wird eingestellt, dass nur Rückseiten ausgeblendet werden. Das ist dann adäquat, wenn nur undurchsichtige Objekte beteiligt sind und vollständig geschlossene Oberflächen haben.
Rückseiten dürfen nicht immer gestrichen werden. Liegt der Augpunkt im Inneren eines großen Objekts, so sind die Vorderseiten der Polygone außen und

die Rückseiten innen, so dass in diesem Fall die Vorderseiten gestrichen werden müssen. Dazu dient `glCullFace(GL_FRONT)`. Um nur die Polygonumrisse anzuzeigen verwendet man `glCullFace(GL_FRONT_AND_BACK)`.
Das Verfahren funktioniert nicht mehr, wenn die Innenseite eines nichtgeschlossenen Objekts sichtbar ist; auch im Zusammenhang mit Beleuchtung kann es zu Problemen führen.
Eine andere Art „innen“ und „außen“ zu unterscheiden, die auch in OpenGL benützt wird, arbeitet so:
Eine Polygonseite gilt als außen liegend, wenn in der Projektion die Bilder der Ecken im Gegenuhrzeigersinn angeordnet sind. Diese Voreinstellung kann durch `glFrontFace(GL_CW)` (CW $\hat{=}$ clockwise) geändert und durch `glFrontFace(GL_CCW)` (CCW $\hat{=}$ counter-clockwise) wiederhergestellt werden.

2.3.3.2 Kappen von Strecken in 2D

Für die Lage von Strecken relativ zum Fenster gibt es drei Möglichkeiten:

1. Beide Endpunkte liegen im Fenster. Dann muss die Strecke vollständig gezeichnet werden.
2. Beide Endpunkte liegen oberhalb, unterhalb oder seitlich vom Fenster. Dann wird die Strecke nicht gezeichnet.
3. Andernfalls muss die Strecke an den Fensterrändern gekappt werden.

Für die Tests in 1 und 2 genügen einfache Vergleiche mit den Extremkoordinaten des Fensters, für 3 das Schneiden von Geraden.
Obwohl der Rechenaufwand gering ist, kann bei komplexen Szenerien mit sehr vielen Strecken der Gesamtaufwand zu hoch sein. Eine effiziente Lösung dieses Problems bietet der Cohen-Sutherland-Algorithmus.
Dazu werden die Fensterränder imaginär ins Unendliche verlängert, so dass die Ebene in neun Teile zerfällt:

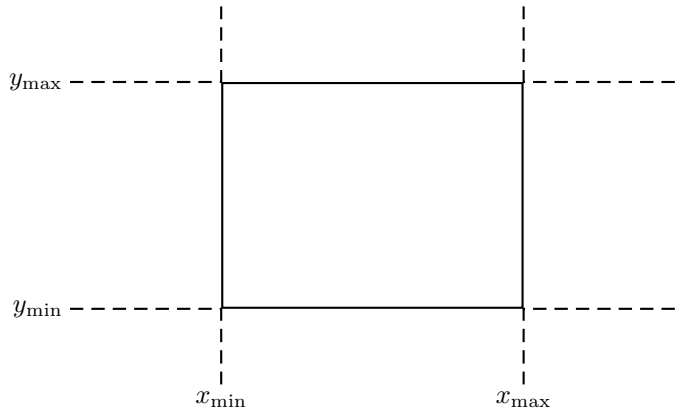


Abbildung 2.35. Cohen-Sutherland-*Algorithmus (Die verlängerten Fensterränder heißen* Fenstergrenzen*)*

Einem Punkt $X = (x, y)$ der Ebene ordnet man nun einen 4-Bit-Code zu, nach folgender Vorschrift:

Bit $0 \stackrel{\wedge}{=} x < x_{min}$	X links vom Fenster
Bit $1 \stackrel{\wedge}{=} x > x_{max}$	X rechts vom Fenster
Bit $2 \stackrel{\wedge}{=} y < y_{min}$	X unterhalb des Fensters
Bit $3 \stackrel{\wedge}{=} y > y_{max}$	X oberhalb des Fensters

Damit haben genau die Punkte im Fenster den Code 0000. Sei nun C_X der Code von X.
Es gilt

$$C_X | C_Y = 0 \Leftrightarrow C_X = 0 \wedge C_Y = 0$$

$$C_X \ \& \ C_Y \neq 0 \Leftrightarrow \exists i : C_{Xi} \neq 0 \wedge C_{Yi} \neq 0$$

Seien nun X und Y die Endpunkte der betrachteten Strecke.

1. Gilt $C_X | C_Y = 0$, so liegt XY im Fenster und wird gezeichnet
2. Gilt ansonsten $C_X \ \& \ C_Y \neq 0$, so liegen beide Punkte ganz auf einer Seite außerhalb des Fensters und XY wird nicht gezeichnet.
3. Andernfalls schneidet XY die Fensterränder und wenigstens einer der Endpunkte (oBdA X) liegt außerhalb des Fensters. Die Bits $\neq 0$ von C_X geben an, welche Fensterränder auf jeden Fall von XY geschnitten werden. Dies wird aus folgendem Bild deutlich (Bit 3 ganz links, Bit 0 ganz rechts stehend)

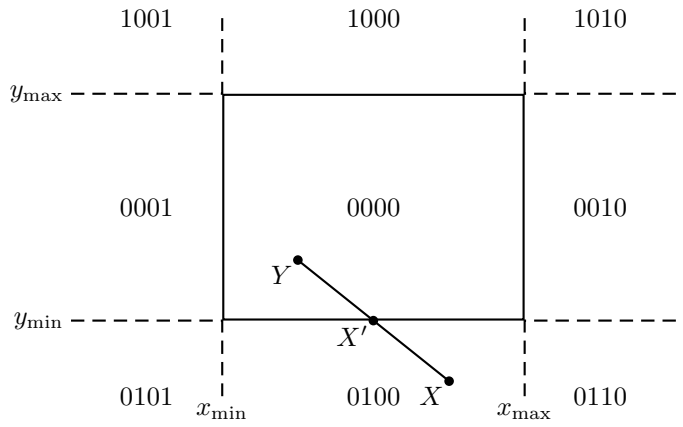


Abbildung 2.36. *Strecke, die Fensterränder schneidet*

Man berechnet nun den Schnittpunkt X' von XY mit dem durch das rechteste Bit $\neq 0$ gegebenen Rand und setzt das Verfahren mit der Strecke $X'Y$ fort. Nach maximal 4 Schritten der Bauart 3 terminiert das Verfahren.

2.3.3.3 Kappen von Polygonen in 2D

Zusätzlich zum reinen Kappen der Polygonseiten am Fenster muss hier berücksichtigt werden, dass Teile der Fensterränder als neue Polygonkanten hinzukommen können.

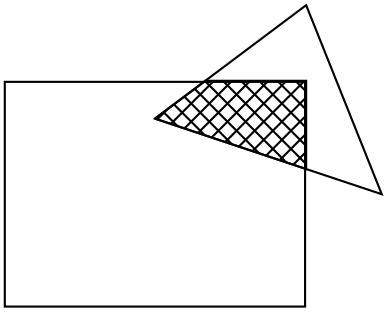


Abbildung 2.37. *Neue Polygonkanten im Fenster*

Außerdem kann ein nicht-konvexes Polygon beim Kappen in mehrere Teile zerfallen:

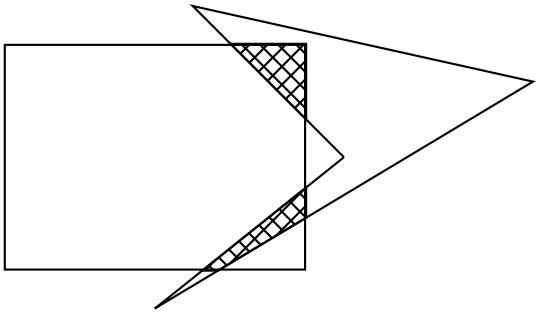


Abbildung 2.38. *Teile eines Polygons im Fenster*

Zur korrekten Erfassung aller Fälle dient der SUTHERLAND-HODGMAN-Algorithmus. Er beschneidet das Polygon reihum an allen vier Fenstergrenzen. Ein solcher Vorgang läuft ab wie folgt: Sämtliche Polygonkanten werden der Reihe nach gerichtet durchlaufen. Man hat dann folgende vier Fälle, falls die Kante nicht auf der Fenstergrenze f verläuft:

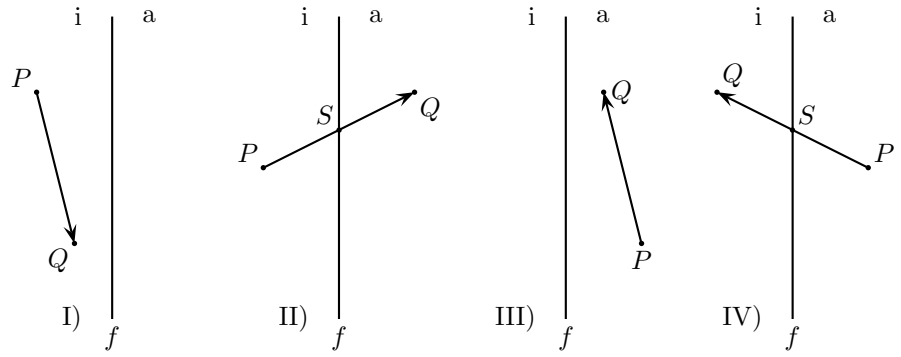


Abbildung 2.39. *Mögliche Schnittpunkte mit der Fenstergrenze, i=innen, a=außen*

P sei bereits behandelt. Was geschieht mit der Kante PQ?

Fall I) Q wird ins Ausgabepolygon übernommen

Fall II) Statt Q wird der Schnittpunkt S von PQ mit f übernommen.

Fall III) Q wird gestrichen, d. h. keine zugehörige Ecke im Ausgabepolygon erzeugt.

Fall IV) Als zusätzlicher Punkt wird S ins Ausgabepolygon aufgenommen.

Für unser voriges Beispiel ergibt sich folgendes, wenn die Reihenfolge der Fenstergrenzen oben-rechts-unten-links ist:

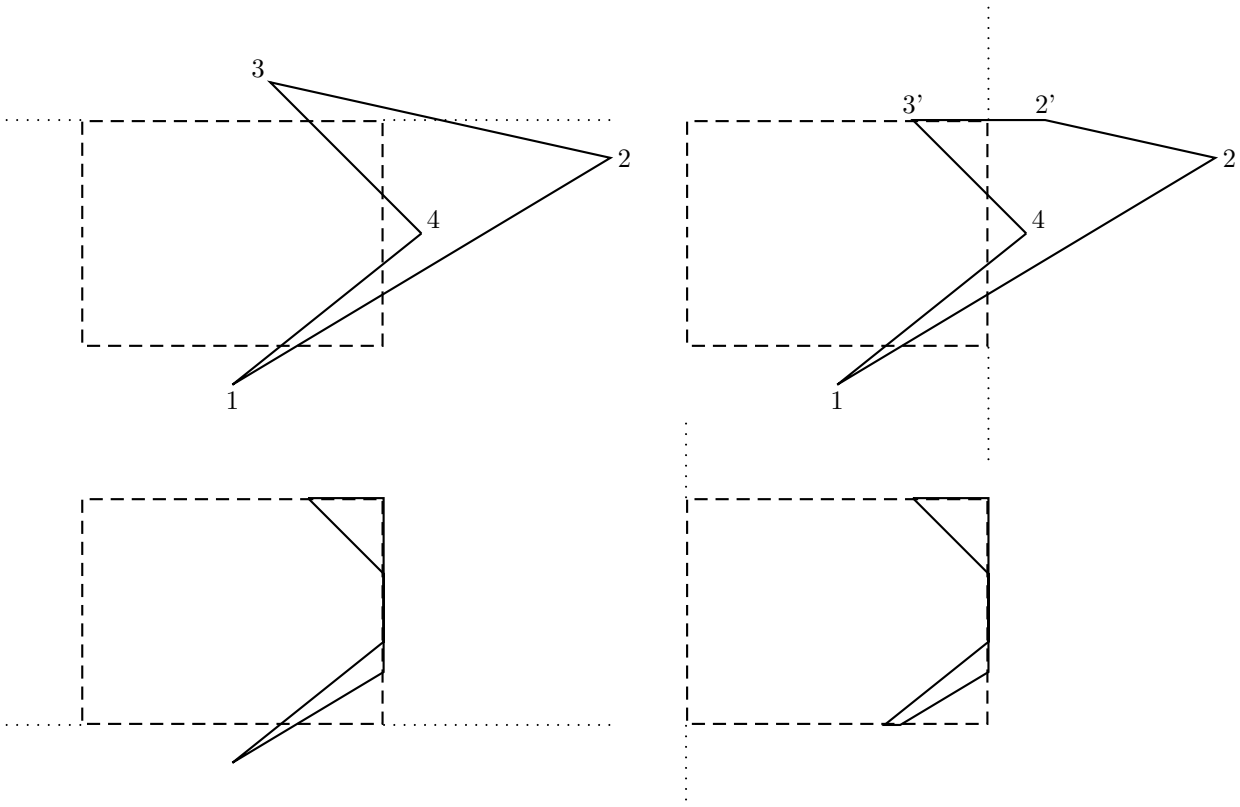


Abbildung 2.40. *Streichung von Streckenteilen, die nicht im Fenster liegen*

Der letzte Beschneidungsschritt an der linken Grenze ändert nichts mehr.
Man beachte, dass einige Kanten des gekappten Polygons auf den Fensterrändern verlaufen.

2.3.3.4 Streichen und Kappen in 3D

Werden diese Operationen *vor* der Projektion ausgeführt, resultiert in der Regel eine erhebliche Effizienzsteigerung, da weniger und einfachere Objekte dargestellt werden müssen.
Zur weiteren Vereinfachung wird dabei die gesamte Szenerie so transformiert, dass das Sichtvolumen eine Standardform bekommt. Bei der Parallelprojektion ist das der Quader zwischen den Ecken (-1, -1, 0) und (1, 1, -1); bei der Zentralprojektion ist es der normierte Pyramidenstumpf mit quadratischem Querschnitt, Pyramidenspitze im Ursprung, der aus dem Frustum durch Verschieben der fernen Deckfläche an die Stelle $z = -1$ und durch Öffnen auf den Winkel 90° entsteht:

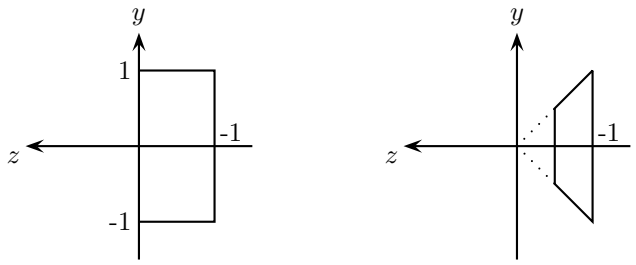


Abbildung 2.41. *Standardformen*

Dieser Stumpf wird dann in einem zweiten Schritt auf den Standardquader der Parallelprojektion transformiert, so dass Streichen und Kappen einheitlich für beide Projektionsarten erfolgen können. Natürlich muss anschließend wieder rücktransformiert werden.
Wir leiten die entsprechenden Transformationsmatrizen her.
Für die Parallelprojektion kann die Matrix N_{par} so zusammengesetzt werden:

1. Verschiebe den Augpunkt in den Ursprung.
2. Verändere bei schiefwinkliger Projektion durch eine Scherung die Projektionsrichtung so, dass sie parallel zur Z-Achse ist.
3. Bringe den entstandenen Quader durch Translation und Skalierung auf die angegebene Standardform.

Bei der Scherung hat man folgende Situation:

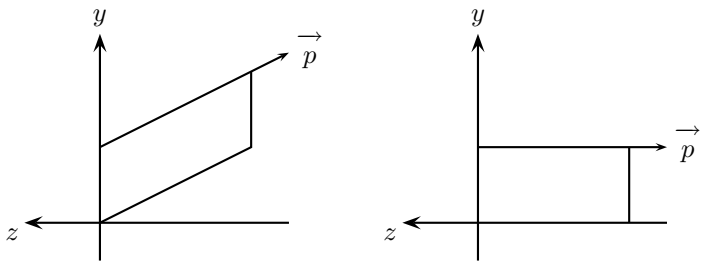


Abbildung 2.42. *Situation bei der Scherung*

Die Matrix muss folgende Gestalt haben:

$$SH_{par} = \begin{pmatrix} 1 & 0 & s_x & 0 \\ 0 & 1 & s_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Ist

$$\overrightarrow{p} = \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} \quad \text{und} \quad \overrightarrow{p}' = \begin{pmatrix} 0 \\ 0 \\ p_z \\ 1 \end{pmatrix} = SH_{par} \cdot \overrightarrow{p}$$

so muss gelten

$$p_x + s_x p_z = 0 \qquad p_y + s_y p_z = 0$$

also

$$s_x = -\frac{p_x}{p_z} \qquad s_y = -\frac{p_y}{p_z}$$

Die anschließende Standardisierung wird durch die Matrix

$$S\left(\begin{array}{ccc} \frac{2}{x_{\max}-x_{\min}} & \frac{2}{y_{\max}-y_{\min}} & \frac{1}{f-b} \end{array}\right) \cdot T\left(\begin{array}{ccc} \frac{x_{max}+x_{min}}{2} & -\frac{y_{\max}+y_{\min}}{2} & -f \end{array}\right)$$

erreicht, wenn der gescherte Quader so liegt:

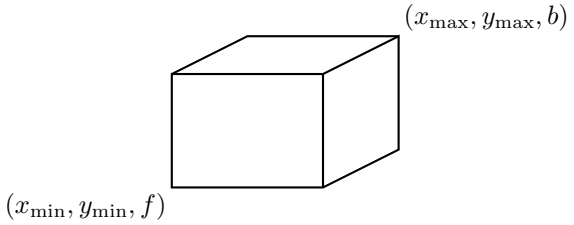


Abbildung 2.43. *gescherter Quader*

Bei der Zentralprojektion geht man so vor:

1. Verschiebe den Augpunkt in den Ursprung.
2. Bringe das Frustum durch Translation und Skalierung in Standardform.

Nach dem 1. Schritt hat man folgende Situation:

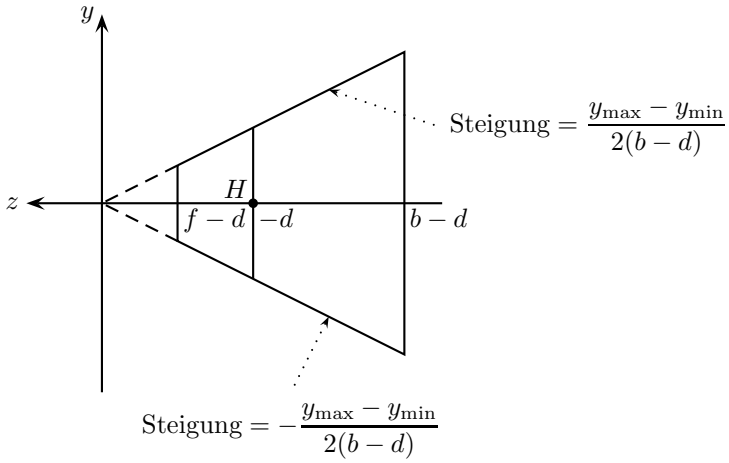


Abbildung 2.44. *Situation nach dem ersten Schritt der Scherung*

Nun wird in x- und y-Richtung so skaliert, dass die Steigungen der Grenzebenen ± 1 sind. Das gelingt mit den Faktoren $\frac{2(b-d)}{x_{max}-x_{min}}$ und $\frac{2(b-d)}{y_{max}-y_{min}}$.
Anschließend wird mit einer gleichmäßigen Skalierung mit dem Faktor $\frac{1}{b-d}$ die ferne Ebene auf $z = -1$ gebracht. Dabei entsteht $z_{min} = -\frac{f-d}{b-d}$.
Schließlich kann der normalisierte Pyramidenstumpf mit der Matrix

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{1+z_{min}} & -\frac{z_{min}}{1+z_{min}} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

auf den Standardquader der Parallelprojektion transformiert werden.
Die Algorithmen von COHEN-SUTHERLAND und SUTHERLAND-HODGMAN zum Strecken und Kappen lassen sich nun leicht vom 2D-Fall auf den 3D-Fall übertragen.

2.3.3.5 Hüllkörper

Um Rechenschritte zu sparen, verwendet man beim Streichen und Kappen zunächst nicht die exakten, oft sehr komplexen Objekte, sondern einfachere, sie umhüllende, wie Vierecke/Quader oder Kreise/Kugeln. Diese *Hüllkörper (bounding volumes* werden konvex gewählt.
Bei polygonalen Objekten ergibt sich ein Hüllquader aus den Minima und Maxima der Eckkoordinaten. Dabei wird zwar u. U. viel an Genauigkeit „verschenkt“, aber der Hüllquader ist leichter zu bestimmen als ein angepasster Hüllkörper. Es empfiehlt sich, einmal bestimmte Hüllkörper bei Transformationen stets mitzutransformieren.
Zur Bestimmung einer Hüllkugel berechnet man zunächst einen Mittelpunkt durch Schwerpunktbildung; der Radius ist dann das Maximum der Abstände der Objektpunkte von diesem Punkt.
Beim Streichen prüft man nun zuerst, ob der Hüllkörper eines Objekts gestrichen werden kann; dann entfällt natürlich auch das Objekt selbst. Beim Kappen kann ein Objekt unverändert bleiben, wenn sein Hüllkörper ganz im Sichtvolumen liegt.

2.3.4 Rasterung und Zeilenkonversion

Rasterung bedeutet die Auswahl geeigneter diskreter Gerätekoordinaten nach der Fenster-Ausschnitt-Transformation. Bei der *Zeilenkonversion (scan conversion)* wird das abstrakte kontinuierliche Bild entlang den Zeilen des Ausgabemediums (z. B. Bildschirm) diskretisiert. Eine *Zeile (scan line)* ist die Folge aller Pixel mit gleicher y-Koordinate im Gerätekoordinatensystem.
Zur einfacheren Beschreibung der Verfahren nehmen wir in diesem Kapitel an, dass die y-Koordinate nicht wie im Gerätekoordinatensystem von oben nach unten wächst, sondern, wie in der Mathematik üblich, von unten nach oben. Außerdem lassen wir vorübergehend auch Bruchzahlen als Koordinatenwerte zu. Den Übergang zum echten Gerätekoordinatensystem erreicht man durch Spiegelung und Rundung der Koordinatenwerte. Außerdem beschränken wir uns auf Schwarz-Weiß-Bilder.

2.3.4.1 Rasterung von Strecken

Auf dem Raster des Gerätekoordinatensystems soll die Strecke P_1P_e mit $P_1 = (x_1, y_1), P_e = (x_e, y_e)$ gezeichnet werden. Dabei sollen P_1, P_e selbst bereits Rasterpunkte sein, d. h. ganzzahlige Koordinaten haben.
Der direkte Ansatz wäre so:

1. stelle die Gleichung der Geraden durch P_1, P_e auf.
2. Bestimme für alle ganzzahligen Werte x_i zwischen x_1 und x_e die zugehörigen y_i-Werte und runde sie zu y_i'.
3. Färbe die Pixel (x_i, y_i') schwarz.

Die Geradengleichung hat die Form $y = mx + d$ mit der Steigung $m = \frac{dy}{dx} = \frac{y_e - y_1}{x_e - x_1}$. Der Achsenabschnitt d ergibt sich aus $y_1 = mx_1 + d$ zu $d = \frac{y_1 x_e - x_e y_1}{dx}$.
Dieses Verfahren hat viele Nachteile:

- Die y_i werden direkt, jeweils mit einer Gleitpunktmultiplikation, berechnet.
- Es muss jeweils gerundet werden.
- Bei positiven Steigungen > 1 liegen die y_i zu weit auseinander; es entsteht keine optisch zusammenhängende Linie mehr.

Für ein verbessertes Verfahren betrachten wir zuerst nur Steigungen m mit $0 \leq n \leq 1$. Dann reicht für jeden x_i-Wert genau ein y_i-Wert aus, um eine optisch zusammenhängende Linie zu erzeugen. Außerdem nehmen wir an, dass die Gerade durch den Ursprung verläuft (d. h. $d = 0$). Alle übrigen Fälle kann man durch Translation und Spiegelung auf diesen Fall zurückführen.
Die Gleitpunktmultiplikationen spart man durch Fortschaltungstechnik ein. Es gilt ja (auch bei $d \neq 0$)

$$y_{i+1} = mx_{i+1} + d = m(x_i + 1) + d = mx_i + d + m = y_i + m$$

Nun muss man nur noch das zeitaufwendige Runden loswerden. Das gelingt durch den *Midpoint-Line-Algorithmus* der auf ein Verfahren von Bresenham 1965 zurückgeht.
Ein Einzelschritt des Verfahrens verläuft so:

- Pixel $P_i = (x_i, y_i)$ sei bereits als zur Darstellung der Strecke gehörig erkannt. Nun ist $P_{i+1} = (x_{i+1}, y_{i+1})$ zu bestimmen.
- Es gilt $x_{i+1} = x_i + 1$ und wegen $0 \leq m \leq 1$ auch $y_{i+1} \in \{y_i, y_i + 1\}$:

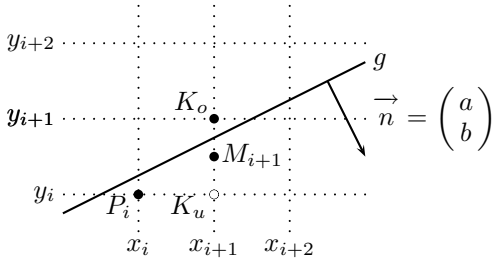


Abbildung 2.45. *Midpoint-Line-Algorithmus*

- Also ist zu entscheiden, welcher der Koordinatenpunkte $K_u = (x_{i+1}, y_i)$ und $K_o = (x_{i+1}, y_i + 1)$ gewählt wird.
- Liegt der Mittelpunkt M_{i+1} von $K_u K_o$ unterhalb g, wählt man K_o, liegt er oberhalb, wählt man K_u, sonst (d. h. M_{i+1} liegt auf s) wählt man einen beliebigen der beiden.

Die Hauptarbeit liegt nun darin, alle benötigten Rechnungen in Ganzzahlarithmetik durchzuführen.
Zuerst aber verwenden wir eine andere Darstellung von g durch ihre Normalenform

$$ax + by + c = 0$$

wobei a und b die Koordinaten des nach unten zeigenden Normalenvektors $\overrightarrow{n}$ von g sind (daher muss $a \geq 0 \wedge b < 0$ gelten). Man erhält diese Form aus der vorigen, indem man mit dx durchmultipliziert und alles auf eine Seite bringt:

$$dy \cdot x - dx \cdot y + dx \cdot d = 0$$

d. h. $a = dy, b = -dx, c = dx \cdot d$.
Für beliebigen Punkt $P = (x, y)$ sei nun $t(P) = ax + by + c$.
Dann gilt

$$t(P) \left\{ \begin{array}{c} > 0 \\ = 0 \\ < 0 \end{array} \right\} \Leftrightarrow \left\{ \begin{array}{c} P \text{ unterhalb } g \\ P \text{ auf } g \\ P \text{ oberhalb } g \end{array} \right.$$

Damit kann der vorher erwähnte Test für M_{i+1} durchgeführt werden. Es gilt $M_{i+1} = (x_i + 1, y_i + \frac{1}{2})$.
Wie kann man $t(M_{i+2})$ durch Fortschaltung ohne Multiplikation aus $t(M_i)$ bestimmen? Hier gibt es zwei Fälle:

- Wird in Schritt i Punkt K_u gewählt, folgt

  $$M_{i+2} = (x_i + 2, y_i + \tfrac{1}{2})$$

  und

  $$\begin{aligned} t(M_{i+2}) &= a(x_i + 2) + b(y_i + \tfrac{1}{2}) + c \\ &= a(x_i + 1) + b(y_i + \tfrac{1}{2}) + c + a \\ &= t(M_{i+1}) + a = t(M_{i+1}) + dy \end{aligned}$$

- Wird in Schritt i Punkt K_o gewählt, folgt

  $$M_{i+2} = (x_i + 2, y_i + \tfrac{3}{2})$$

  und

  $$\begin{aligned} t(M_{i+2}) &= a(x_i + 2) + b(y_i + \tfrac{3}{2}) + c \\ &= a(x_i + 1) + b(y_i + \tfrac{1}{2}) + c + a + b \\ &= t(M_{i+1}) + a + b = t(M_{i+1}) + dy - dx \end{aligned}$$

Als Ausgangswert brauchen wir $M_2 = (x_1 + 1, y_1 + \frac{1}{2})$ mit

$$\begin{aligned} t(M_2) &= a(x_1 + 1) + b(y_1 + \tfrac{1}{2}) + c \\ &= ax_1 + by_1 + c + a + \tfrac{b}{2} \\ &= t(P_1) + a + \tfrac{b}{2} = a + \tfrac{b}{2} = dy - \tfrac{dx}{2} \end{aligned}$$

da P_1 auf g liegt, also $t(P_1) = 0$ gilt.
Jetzt stört noch das Halbieren die reine Ganzzahlarithmetik. Da aber nur das Vorzeichen der $t(M_i)$ benötigt wird, kann man einfach alles mit 2 durchmultiplizieren, d. h. mit $f(M_i) = 2 \cdot t(M_i)$ rechnen. Das ergibt

$$f(M_2) = 2 \cdot dy - dx$$

$$f(M_{i+2}) = f(M_{i+1}) + \begin{cases} 2 \cdot dy & \text{falls } f(M_{i+1}) < 0 \text{ d. h. } K_u \text{ gerade} \\ 2 \cdot dy - 2 \cdot dx & \text{falls } f(M_{i+1}) \geq 0 \text{ d. h. } K_o \text{ gerade} \end{cases}$$

Beispiel
$P_1 = (2, 3)$ $P_e = (15, 8)$
$dx = 13$ $dy = 5$ $2 \cdot dy = 10$ $2 \cdot dy - 2 \cdot dx = -16$

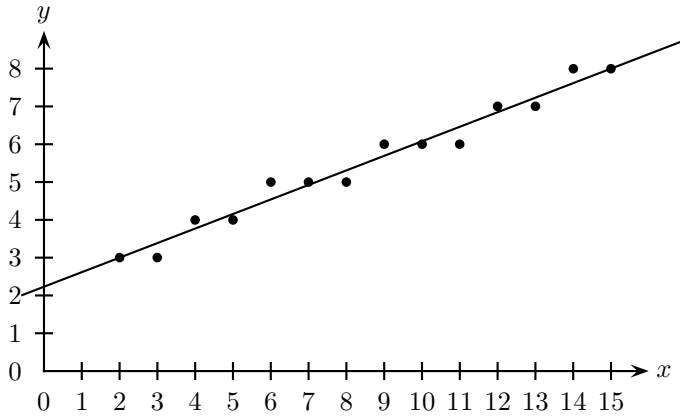


Abbildung 2.46. *Beispiel zum Midpoint-Line-Algorithmus*

i	x_{i+1}	$f(M_{i+1})$	u/o	y_{i+1}
1	3	-3	u	3
2	4	7	o	4
3	5	-9	u	4
4	6	1	o	5
5	7	-15	u	5
6	8	-5	u	5
7	9	5	o	6
8	10	-11	u	6
9	11	-1	u	6
10	12	9	o	7
11	13	-7	u	7
12	14	3	o	8
13	15	-13	u	8

Wenn die Strecken nicht schwarz, sondern farbig gezeichnet werden, lässt man oft den Endpunkt weg, da der ja Anfang einer weiteren Strecke, etwa der nächsten Polygonkante sein kann, die u. U. eine andere Farbe trägt. Damit werden unkontrollierbare Farbüberschreibungen vermieden.

2.3.4.2 Abmildern von Treppeneffekten (Anti-Aliasing)

Sind die Pixel relativ groß, können die beim Midpoint-Line-Algorithmus entstehenden Annäherungen eine sehr starke Treppenstruktur statt eines quasikontinuierlichen Verlaufs zeigen.
Das kann abgemildert werden, wenn man statt mit reinem Schwarz-Weiß mit Grauwerten arbeitet. Man benutzt dann den Abstand der Pixel von der idealisierten Strecke, um ihre Grauwerte zu bestimmen. Je weiter die Pixelmitte von der Strecke entfernt ist, desto mehr ähnelt die Farbe des Pixels der Hintergrundfarbe. Um visuelle Lücken zu schließen, werden nun auch zusätzliche Pixel mit einbezogen, die vom Midpoint-Line-Algorithmus gar nicht betrachtet werden.

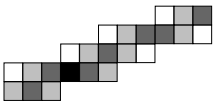


Abbildung 2.47. *Beispiel für Anti-Aliasing*

Allerdings führt das zu einem Schärfeverlust (Weichzeichnereffekt).

2.3.4.3 Rasterung von Polygonen

Wie bereits beim Kappen genügt es hier nicht, nur die Polygonkanten zu rastern, weil ja auch die Pixel im Polygoninneren gefärbt werden müssen.
Hier arbeitet man vorteilhaft zeilenorientiert. Mit dem zu besprechenden Verfahren können sogar konkave oder durchlöcherte Polygone korrekt dargestellt werden. Wichtig ist, effizient entscheiden zu können, ob ein Pixel innerhalb oder außerhalb des betrachteten Polygons liegt. Dazu hat man folgende Regeln (das betrachtete Polygon sei P):

1. Bestimme für die aktuelle Rasterzeile Z die Schnittpunkte mit allen Kanten von P und ordne sie nach steigender x-Koordinate in einer Liste an. Dabei werden Schnittpunkte mit horizontalen Kanten nicht in die Liste aufgenommen. Läuft Z durch eine Ecke von P, d. h. schneidet Z beide beteiligten Kanten, so tritt der Schnittpunkt in der Liste doppelt auf.

2. Durchlaufe nun die geordnete Liste L der Schnittpunkte von links nach rechts. Zwischen zwei solchen Schnittpunkten bekommen alle Pixel im Inneren von P die Polygonfarbe. Dazu genügt es festzustellen, ob links von diesen Pixeln ungerade viele Schnittpunkte liegen. Man merkt sich also beim Durchlaufen der Zeile einen Zustand mit den Werten g bzw. u (gerade bzw. ungerade Zahl von Schnittpunkten links vom betrachteten Pixel).

   Folgende Fälle treten auf, wenn das aktuelle Pixel ein noch nicht bearbeiteter Schnittpunkt $S = (x, y)$ ist:

(a) S ist Ecke von P.
Bei der Zählung g/u wird S nur für diejenige Kante von P gezählt, für die es eine Ecke mit minimaler y-Koordinate ist; ein Vorkommen von S, in dem es Ecke mit maximaler y-Koordinate ist, wird aus L gelöscht.

(b) x ist nicht ganzzahlig. Liegt der Bereich links von S außerhalb von P, wird das Pixel mit Koordinate $\lceil x \rceil$ als innerhalb von P definiert. Liegt der Bereich innerhalb von P, wird das Pixel mit Koordinate $\lfloor x \rfloor$ als innerhalb von P definiert.

(c) x ist ganzzahlig. Liegt der Bereich links von S außerhalb von P, wird das Pixel mit Koordinate x als innerhalb von P definiert. Liegt der Bereich innerhalb von P, wird das Pixel mit Koordinate x als außerhalb von P definiert.

(d) Bei jedem Wechsel von g auf u wird mindestens ein Pixel bearbeitet.

Beispiel

a) Zeile 11: $L = [18.0, 18.0]$

Nach Regel 1 werden beide Punkte aus L gestrichen, also keine inneren Pixel von P in dieser Zeile.

b) Zeile 7: $L = [2.0, 4.0, 8.0, 11.0, 20.0]$ (die horizontale Kante ON wird nicht berücksichtigt)

Der Wert für O wird nach Regel 1 aus L gestrichen. Als innere Punkte verbleiben nach 3. die Pixel mit x-Koordinaten 2-3 und 11-19.

c) Zeile 6: $L = [2.0, 6.0, 6.0, 20.0]$ Innerhalb des Polygons: 2-5 und 6-19.

d) Zeile 1: $L = [4.0, 4.0, 9.0, 12.0]$ Innerhalb des Polygons: 4 und 9-12. □

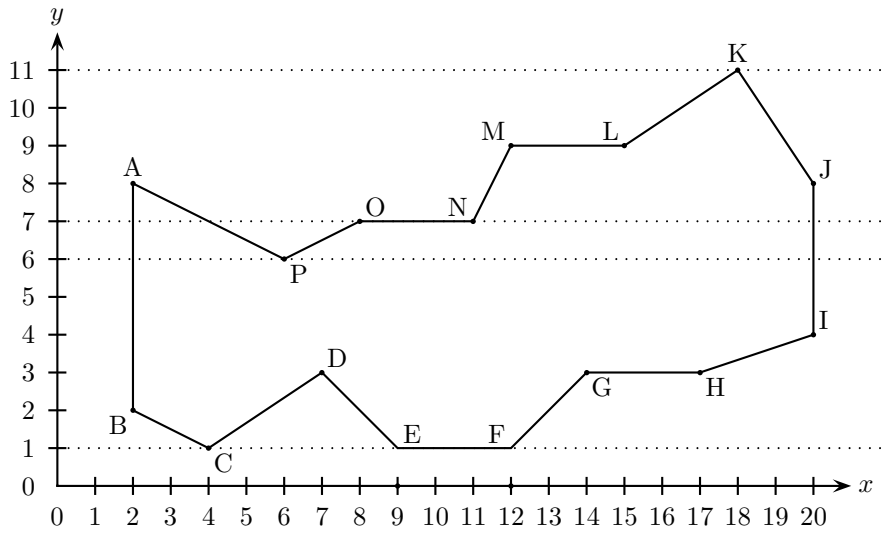


Abbildung 2.48. *Rastern von Polygonen*

Die Unsymmetrie in den Fällen 2. und 3. dient dazu, aneinander grenzende Polygone korrekt zu behandeln, analog zum Fall aneinander grenzender Strecken in 2.3.4.1.
Die Schnittpunktberechnungen müssen nicht für jede Zeile neu durchgeführt werden, sondern können inkrementell von einer Zeile zur nächsten erfolgen.
Dazu nützt man aus, dass die Steigung S einer Polygonkante K sich aus den Koordinaten ihrer Endpunkte $P_a = (x_a, y_a)$ und $P_e = (x_e, y_e)$ mit $y_a < y_e$ ergibt als $s = \frac{y_e - y_a}{x_e - x_a}$. Andererseits gilt für die Schnittpunkte $P_i = (x_i, y_i)$ und $P_{i+1} = (x_{i+1}, y_{i+1})$ der Zeilen i und $i+1$ mit K, dass auch $s = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} = \frac{1}{x_{i+1} - x_i}$ ist. Also folgt $x_{i+1} = x_i + \frac{1}{s}$.
Natürlich treten auch hier beim Rastern Treppeneffekte auf, die ähnlich wie in 2.3.4.2 abgemildert werden.

2.4 Sichtbarkeit

In diesem Abschnitt geht es um Verdeckungen der Objekt(teil)e im Sichtvolumen untereinander. Um alle Fälle einheitlich behandeln zu können, transformiert man vor der Verdeckungsanalyse das Sichtvolumen mit den Methoden von 2.3.3.4 auf Standardform. Es gibt zwei Arten der Analyse:

1. *Objektraumorientierte* Verfahren arbeiten auf den abstrakten Objekten. Damit sind sie unabhängig vom speziellen Ausgabegerät und können außerdem die volle Genauigkeit der Gleitpunktarithmetik nutzen.

2. *Bildraumorientierte* Verfahren arbeiten im Gerätekoordinatensystem. Sie sind damit geräteabhängig und sind in ihrer Genauigkeit durch die diskrete Geräteauflösung beschränkt

2.4.1 Objektraumorientierte Verfahren

Im Prinzip muss hier für alle Paare von Objekten geprüft werden, ob sie sich schneiden und sich somit teilweise verdecken. Die Komplexität kann reduziert werden, indem man die Objekte zuerst auf die Bildebene projiziert und die Schnitte zwischen den Projektionen bestimmt. Besonders einfach wird das bei polygonalen Objekten. Eine weitere Reduktion der Komplexität wird durch Verwendung von Hüllkörpern erreicht: Schneiden sich die Hüllkörper zweier Körper nicht, so auch nicht die Körper selbst.
Allerdings ist die Analyse der Projektion u. U. zu grob; die Projektionen zweier Hüllkörper können sich schneiden, obwohl die Projektionen der Körper selbst getrennt liegen:

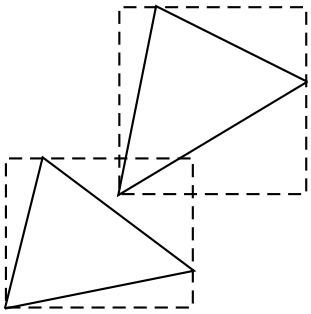


Abbildung 2.49. *Hüllkörper schneidet Hüllkörper eines anderen Körpers*

2.4.2 Bildraumverfahren: Tiefenpufferung

Die Grobstruktur solcher Verfahren sieht so aus: Man ermittelt für jedes Pixel P das Objekt, dessen Schnittpunkt mit der durch P laufenden Projektionsgeraden am nächsten an der Bildebene liegt. Dann wird P in der Farbe dieses Schnittpunkts gezeichnet
Das weitest verbreitete Verfahren ist die *Tiefenpufferung (Z-buffering)*, das auch in OpenGL unterstützt wird. Es stammt von Catmull 1974. Wir setzen voraus, dass das Sichtvolumen bereits auf Standardform transformiert ist.
Der *Tiefenpuffer (Z-Puffer)* ist ein zweidimensionales Feld ZP mit der Auflösung des Bildpuffers BP. Er enthält für jedes Pixel zwischen 8 und 32 Bit Tiefeninformation. Nun werden die Pixel zeilenweise bearbeitet.

1. Initialisiere BP mit der Hintergrundfarbe.
2. Initialisiere ZP mit der z-Koordinate der hinteren Deckfläche des Sichtvolumens, also dem kleinsten möglichen z-Wert.
3. Führe nun für alle Polygone im Sichtvolumen jeweils eine Rasterzeilenkonversion durch:
   (a) Bestimme für jedes Pixel (x, y) in der Projektion des Polygons die z-Koordinate $z(x, y)$ des zugehörigen Polygonpunkts.
   (b) Ist $z(x, y) > ZP(x, y)$, so setze $BP(x, y)$ auf die Polygonfarbe und überschreibe $ZP(x, y)$ mit $z(x, y)$.

Anschließend enthält BP das korrekte Bild der Szene und ZP dessen Tiefenverteilung.
Die Berechnung der Werte $z(x, y)$ kann bei ebenen Polygonen wieder inkrementell geschehen. Die Ebene, in der das Polygon liegt, sei gegeben durch die Gleichung $ax + by + cz + d = 0$. Dann gilt

$$z(x, y) = \frac{-d - ax - by}{c}$$

Geht man nun zeilenweise vor, verändert sich in jedem Schritt x zu $x + dx$ und es gilt

$$z(x + dx, y) = \frac{-d - a(x + dx) - by}{c} = \frac{-d - ax - by - a \cdot dx}{c}$$

$$= z(x, y) - \frac{a \cdot dx}{c}$$

Also genügt zur Bestimmung des nächsten z-Werts eine einfach Subtraktion.
Vorteile der Tiefenpufferung:

- Das Ergebnis ist unabhängig von der Bearbeitungsreihenfolge der Polygone; damit brauchen diese nicht vorsortiert zu werden.
- Das Verfahren ist linear in der Anzahl und Eckenzahl der Polygone und damit auch für sehr komplexe Szenerien geeignet.
- Es ist einfach in Soft- und Hardware zu implementieren und gehört heute zum Standard der Grafikkarten.
- Wie oben gezeigt, kann es durch inkrementelle Zeilenberechnung zusätzlich beschleunigt werden.
- Das entstehende Tiefenprofil kann auch zur Hardware-unterstützten Schattenberechnung genutzt werden: Beim Rendering hilft es zu entscheiden, ob ein Punkt von einer Lichtquelle beleuchtet wird oder in ihrem Schatten liegt

Nachteile der Tiefenpufferung:

- Wegen der rasterorientierten Arbeitsweise entstehen oft unschöne Treppeneffekt; die Abhilfe ist aufwendig.

- Die Auflösung und Genauigkeit von *ZP* bestimmt die Diskretisierung der Bildtiefe. Weit entfernte Objekte mit kleinen Details werden nur ungenau dargestellt. Außerdem hängt die Qualität des Ergebnisses sehr stark von der Wahl der Deckflächen des Frustums ab.
- Bei Zentralprojektion erzeugt die Verzerrung nichtlineare Zusammenhänge zwischen den Tiefenwerten in Szenerie und Projektion. Daher werden weiter entfernte Objekte u. U. in ihren *ZP*-Werten identifiziert und fallen im Bild zusammen.
- Der Speicheraufwand ist erheblich: Bei Auflösung 1600×1400 und einer z-Tiefe von 16 Bit ergeben sich 4,2 MB Speicher für *ZP*! Als Abhilfe kann das Bild in Teilbilder, z. B. Streifen, zerlegt werden.

Ein Verfahren zu letzterem Problem ist *zeilenweise z-Pufferung*. Hier enthält *ZP* nur die Tiefeninformation für genau eine Rasterzeile.
Weitere Verfahren zur Sichtbarkeitsanalyse werden in späteren Abschnitten besprochen.

2.5 Parameterdarstellung von Kurven und Flächen

Bisher hatten wir nur vergleichsweise einfache Kurven und Flächen wie Kreis- und Ellipsenbögen oder Quadriken verwendet. Gerade wenn aber Bahnkurven für komplexe Bewegungsabläufe definiert werden sollen, braucht man Beschreibungsmittel, die darüber hinausgehen. Weitere Anwendungen finden sich in der Freiformgeometrie, wenn mehrere vorgegebene Punkte durch möglichst "elegante" und "glatte" Kurven- oder Flächenstücke verbunden werden sollen.
In diesem Abschnitt besprechen wir als sehr flexibles Hilfsmittel hierfür die *Parameterdarstellung* von Kurven und Flächen. Bei Kurven werden Punkte des $\mathbb{R}^2$ oder $\mathbb{R}^3$ in Abhängigkeit von einem reellwertigen Parameter $t \in [a, b]$ für ein Intervall $[a, b] \subseteq \mathbb{R}$ beschrieben. In vielen Anwendungen, etwa bei der Modellierung von Bewegungsabläufen, kann t als die Zeit interpretiert werden; oft ist es aber, wie beim Morphing in Abschnitt 2.1, ein Maß für eine gewisse Deformation oder Abweichung von einem Grundwert.
Eine *Kurve in Parameterdarstellung* im $\mathbb{R}^2$ ist also eine Abbildung $K : [a, b] \longrightarrow \mathbb{R}^2$; man schreibt dann oft auch $K(t) = (x(t), y(t))$, wobei x, y selbst reellwertige Funktionen $[a, b] \longrightarrow \mathbb{R}$ sind. Für 3D-Kurven hängt dann auch noch die z-Koordinate von t ab.

Beispiel. Der Kreis K mit Radius r um den Ursprung hat die Parameterdarstellung

$$K(t) = (r \cos t, r \sin t)$$

mit $t \in [0, 2\pi]$. Hier stellt t den Winkel zwischen dem Ortsvektor des jeweiligen Kreispunkts und der x-Achse dar.Das ist ein Spezialfall der Ellipse E mit Halbachsen r_1, r_2 und Mittelpunkt im Ursprung:

$$E(t) = (r_1 \cos t, r_2 \sin t)$$

Ein komplexeres Beispiel ist die *Blütenblattkurve* B mit

$$B(t) = (\cos 4t \cos t, cos4t \sin t)$$

Hier spielt t gewissermaßen eine Doppelrolle: Das jeweils erste Vorkommen hat eher zeitartigen Charakter, das zweite eher winkelartigen. Die Kurve entsteht, wenn beim Zeichnen eines Kreises das Bezugssystem relativ zur Umgebung rotiert.
Als Beispiel für eine 3D-Kurve diene die Schnecken/schraubenlinie oder Helix H; sie entsteht durch Zeichnen eines Kreises unter Verschiebung in der dritten Dimension:

$$H(t) = (r \cos t, r \sin t, ht)$$

wobei h die "Verschiebungsgeschwindigkeit" ist. Bei jedem Durchlauf von t durch ein Intervall der Länge 2π macht die "Schraube" eine Umdrehung. Da sie dabei die Höhe h überwindet, heißt h auch die *Ganghöhe* der Schraube. □

Oft wählt man als Parameterintervall $[0, 1]$. Die *Parameter-Transformation* von einem beliebigen Intervall $[a, b]$ auf $[0, 1]$ gelingt durch die Vorschrift

$$\alpha(t) = \frac{t - a}{b - a}$$

die Umkehrung ist

$$\alpha^{-1}(t) = a + t(b - a)$$

Bedeutet $K(t) = (x(t), y(t))$ die Bahnkurve eines Objekts in Abhängigkeit von der Zeit t, so gibt der Vektor $K'(t) = (x'(t), y'(t))$, die erste Ableitung von K, die Bewegungsrichtung des Objekts an; $K'(t)$ ist der Richtungsvektor der Tangenten an K im Punkt $K(t)$ und sein Betrag ist die Geschwindigkeit des Objekts in diesem Punkt seiner Bahn. Analog gibt die zweite Ableitung $K''(t)$ Richtung und Größe der Bewegungsänderung im Punkt $K(t)$ an; der Betrag dieses Vektors ist also die Beschleunigung im Punkt $K(t)$. Gelegentlich braucht man auch die *Kurvennormale* im Punkt $K(t)$; das ist ein Vektor der auf der Tangenten senkrecht steht, also z.B. $(-y'(t), x(t))$.

Beispiel. Für den Kreis K gilt $K'(t) = (-r \sin t, r \cos t)$. Damit ist der Normalenvektor $(-r \cos t, -r \sin t)$ in jedem Punkt parallel zum Ortsvektor des Punktes, was ja auch zur Anschauung passt. □

Mit weiteren Hilfsmitteln aus der Analysis kann man auch die *Krümmung*, den *Krümmungsradius* und die *Bogenlänge* zwischen zwei Kurvenpunkten bestimmen; wir gehen hierauf aber nicht genauer ein.
Parameterisierte Flächen F im $\mathbb{R}^3$ erhält man in der Form

$$F(s,t) = (x(s,t), y(s,t), z(s,t))$$

mit $s \in [a,b], t \in [c,d]$. Wir werden das später im Zusammenhang mit Texturabbildungen genauer besprechen.

2.6 Bézierkurven und -flächen

Als wichtiges Beispiel für parametrisierte Kurven besprechen wir die *Bézierkurven*. Sie wurden in den 1960er Jahren ursprünglich für gute Formgebung im Automobilbau entwickelt. Inzwischen sind sie aus Zeichenprogrammen nicht mehr wegzudenken; andere Anwendungen finden sich beim Entwurf neuer Schriftarten, wenn "schön" geschweifte Linien gewünscht sind. Wir beschränken uns in diesem Abschnitt auf Bézierkurven; Bézierflächen werden wir im Zusammenhang mit Texturen besprechen. Es sei noch vermerkt, dass OpenGL keine direkte Unterstützung für Béziertechniken bietet; die Verfahren müssen koordinatenweise ausprogrammiert werden.

2.6.1 Definition und grundlegende Eigenschaften

Eine Bézierkurve der Ordnung $n \geq 1$ wird durch eine Folge $P_0, \ldots, P_n$ von $n+1$ Punkten im $\mathbb{R}^2$ festgelegt. P_0 und P_n heißen ihre *Ankerpunkte*, die übrigen ihre *Kontrollpunkte*. Die Kontrollpunkte steuern, wie die Kurve zwischen ihren Ankerpunkten verläuft. Das von den Anker- und Kontrollpunkten gebildete Polygon heißt das *Kontrollpolygon* der Bézierkurve, die Kanten P_iP_{i+1} $(i = 0, \ldots, n-1)$ seine *Kontrollkanten*. In der Praxis begnügt man sich meist mit Ordnungen ≤ 3, da man komplexere Kurven oft durch Zusammensetzen solcher einfacher Bézierkurven approximieren kann.
Ihre Definition haben wir bereits in Abschnitt 2.1 beim Morphing gesehen:

$$C^n(P_0, \ldots, P_n)(t) = \sum_{i=0}^{n} P_i B_i^n(t)$$

wobei die *Bernsteinpolynome* B_i^n definiert sind als

$$B_i^n(t) = \left(\begin{array}{c} n \\ i \end{array} \right) (1-t)^{n-i} t^i$$

mit $t \in [0, 1]$.
Die Spezialfälle für $n = 1, 2, 3$ heißen *lineare, quadratische* bzw. *kubische* Bézierkurven; die Graphen der beiden letzteren sind in Abb. 2.9 zu sehen.
Allgemein haben die Bézierkurven folgende Eigenschaften:

- Die Koordinatenfunktionen einer Bézierkurve der Ordnung n sind Polynome vom Grad n.
- Die Kurve hat die Ankerpunkte als Endpunkte.
- Die Kurve verläuft innerhalb der konvexen Hülle des Kontrollpolygons und folgt in ihrer Gestalt dem Kantenzug der Kontrollkanten.
- Eine Gerade schneidet die Kurve höchstens so oft wie sie ihr Kontrollpolygon schneidet.
- An den Ankerpunkten hat die Kurve die Geraden P_0P_1 bzw. $P_{n-1}P_n$ als Tangenten.
- Liegen alle Anker- und Kontrollpunkte auf einer Geraden, entartet die Bézierkurve zu einer Geraden.

- Um eine Bézierkurve affin zu transformieren, genügt es, ihr Kontrollpolygon dieser Transformation zu unterwerfen.

Wir rechnen die Behauptung über die Tangenten an den Ankerpunkten kurz nach. Als Ableitung von C^n ergibt sich mit Produkt- und Kettenregel für die Ableitung

$$C^n(P_0,\ldots,P_n)'(t) = \sum_{i=0}^{n} P_i B_i^{n\prime}(t) = \sum_{i=0}^{n} P_i \left(\begin{array}{c} n \\ i \end{array} \right) ((n-i)(1-t)^{n-i-1}(-1)t^i + i(1-t)^{n-i}t^{i-1})$$

Für $t = 0$ gilt $C^n(P_0,\ldots,P_n)(t) = P_0$ und wir müssen die Ableitung an diesem Punkt bestimmen.

$$C^n(P_0,\ldots,P_n)'(0) = \sum_{i=0}^{n} P_i \left(\begin{array}{c} n \\ i \end{array} \right) ((n-i)(-1)0^i + i0^{i-1}) = P_0 \left(\begin{array}{c} n \\ 0 \end{array} \right)(-n) + P_1 \left(\begin{array}{c} n \\ 1 \end{array} \right) 1 = n(P_1 - P_0)$$

da für alle $i > 1$ die entsprechenden Potenzen von 0 selbst Null sind und alles auslöschen. Also ist die Tangente in P_0 tatsächlich parallel zur Geraden P_0P_1. Der Beweis für den anderen Ankerpunkt P_n verläuft analog.
Wir geben noch einige Eigenschaften der Bernsteinpolynome an:

- Sie bilden einer Zerlegung der Eins (vgl. Abschnitt 2.1):

$$\sum_{i=0}^{n} B_i^n(t) = 1$$

- $B_i^n(t)$ ist ein Polynom vom Grad n und für $t \in [0,1]$ gilt $B_i^n(t) \geq 0$.
- Es gilt folgende Rekursionsbeziehung:

$$B_i^n(t) = (1-t)B_i^{n-1}(t) + tB_{i-1}^{n-1}(t)$$

  Außerdem ist $B_0^n(t) = (1-t)^n$ und $B_n^n(t) = t^n$.

2.6.2 Zusammenfügen von Bézierkurven

Da die Tangentenrichtung an den Ankerpunkten bekannt ist, kann man eine Beziérkurve $C^m(P_0,\ldots,P_m)$ mit einer anderen $C^n(Q_0,\ldots,Q_n)$ "glatt" verkleben, wenn $P_m = Q_0$ und $P_{m-1}P_m = -Q_0Q_1$ ist. Diesen Vorgang nennt man *Endpunktinterpolation*.
Mit dieser Methode können komplexe Kurven besser modular behandelt werden, indem man m und n jeweils relativ klein wählt und dafür lieber viele Teilkurven verwendet.

2.6.3 Der Algorithmus von de Casteljau

Die direkte Auswertung der definierenden Formel für die Beziérkurven ist, besonders bei größerer Ordnung und bei Bestimmung von vielen Kurvenpunkten, recht aufwendig. De Casteljau hat einen Algorithmus angegeben, der einfachere Operationen verwendet und sich auch sehr gut geometrisch ausführen lässt. Er basiert auf obiger Rekursionsformel für die Bernsteinpolynome. Man rechnet für $n > 1$ so:

$$
\begin{aligned}
& C^n(P_0, \ldots, P_n)(t) \\
= \; & \sum_{i=0}^{n} P_i B_i^n(t) \\
= \; & P_0 B_0^n(t) + \sum_{i=1}^{n-1} P_i B_i^n(t) + P_n B_n^n(t) \\
= \; & P_0 (1-t)^n + \sum_{i=1}^{n-1} P_i ((1-t) B_i^{n-1}(t) + t B_{i-1}^{n-1}(t)) + P_n t^n \\
= \; & P_0 (1-t)^n + \sum_{i=1}^{n-1} (1-t) P_i B_i^{n-1}(t) + \sum_{i=1}^{n-1} t P_i B_{i-1}^{n-1}(t)) + P_n t^n \\
= \; & (1-t) P_0 B_0^{n-1} + \sum_{i=1}^{n-1} (1-t) P_i B_i^{n-1}(t)) + \sum_{i=1}^{n-1} t P_i B_{i-1}^{n-1}(t) + t P_n B_{n-1}^{n-1}) \\
= \; & (1-t) P_0 B_0^{n-1} + \sum_{i=1}^{n-1} (1-t) P_i B_i^{n-1}(t)) + \sum_{i=0}^{n-1} t P_{i+1} B_i^{n-1}(t) + t P_n B_{n-1}^{n-1}) \\
= \; & (1-t) P_0 B_0^{n-1} + \sum_{i=1}^{n-1} (1-t) P_i B_i^{n-1}(t)) + \sum_{i=0}^{n-2} t P_{i+1} B_i^{n-1}(t) + t P_n B_{n-1}^{n-1}) \\
= \; & \sum_{i=0}^{n-1} (1-t) P_i B_i^{n-1}(t)) + \sum_{i=0}^{n-1} t P_{i+1} B_i^{n-1}(t) \\
= \; & \sum_{i=0}^{n-1} ((1-t) P_i + t P_{i+1}) B_i^{n-1}(t) \\
= \; & C^{n-1}(Q_0, \ldots, Q_{n-1})(t)
\end{aligned}
$$

wobei $Q_i = (1-t)P_i + tP_{i+1}$ gilt, d.h. Q_i die Strecke P_iP_{i+1} im Verhältnis $(1-t) : t$ teilt. Für $n = 1$ gilt

$$C^1(P_0, P_1)(t) = (1-t)P_0 + tP_1$$

Daraus ergibt sich folgendes iteratives Verfahren:

- Sind $n > 1$ Kontrollkanten vorhanden, so teile alle im Verhältnis $(1-t) : t$ und verbinde die entstehenden Punkte. Es entsteht ein Kantenzug mit $n-1$ Kanten.
- Wiederhole das Verfahren, falls möglich.
- Ist schließlich nur noch eine Kante übrig, teile sie im gleichen Verhältnis; der zugehörige Punkt ist der Kurvenpunkt der ursprünglichen Kurve zum Parameterwert t.

3 Bildsynthese

In diesem Kapitel werden wir über Farben, Beleuchtung und Schattierung sowie über Texturen sprechen. Außerdem werden wir kurz auf das Anti-Aliasing eingehen.

3.1 Wahrnehmung, Licht und Farbe

3.1.1 Licht und Farbe

Das für den Menschen wahrnehmbare Licht hat Wellenlängen zwischen 390 nm und 800 nm, die sich wie folgt aufteilen:

- Violett 390-430 nm
- Blau-Violett 460-480 nm
- Cyan 480-490 nm
- Grün 490-530 nm
- Gelb 550-580 nm
- Orange 590-640 nm
- Rot 630-800 nm

Im Allgemeinen strahlen Lichtquellen ein ganzes Kontinuum aus diesem Spektrum ab, nur Laser können eine spezielle Wellenlänge „rein“ abstrahlen. Die Farbe Weiß ist eine gleichmäßige Mischung aller Wellenlängen. Eine Farbe lässt sich durch drei Kenngrößen beschreiben:

- den *Farbton (hue)*, der bestimmt ist durch die dominanten Lichtwellenlängen;
- die *Helligkeit (luminance)*, die gesamte Energie in der Mischfarbe; sie ist proportional zur Fläche unter der Verteilungskurve;

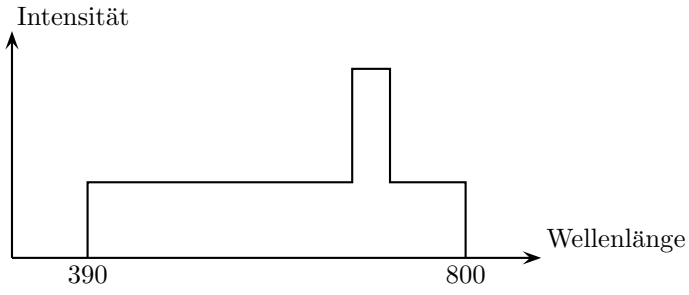


Abbildung 3.50.

- die *Sättigung (saturation)*, d. h. das Verhältnis der Intensität der dominanten Wellenlänge zur Gesamtintensität.

Die Ausbreitung des Lichts wird in der Computergrafik meist durch Lichtstrahlen von der Lichtquelle zu den Szenerieobjekten modelliert.

3.1.2 Die menschliche Wahrnehmung

Das Licht wird in der Netzhaut des menschlichen Auges durch zwei Arten von Rezeptoren, den *Stäbchen* und den *Zäpfchen*, wahrgenommen. Die Stäbchen sind für das Hell-Dunkel-Sehen zuständig, die Zäpfchen für das Farbensehen. Es gibt ca. 120 Mio. Stäbchen und ca. 64 Mio. Zäpfchen, also hat man das Verhältnis 19:1. Außerdem reagieren die Stäbchen wesentlich empfindlicher auf Lichtreize als die Zäpfchen. Insgesamt reagiert das Auge daher viel stärker auf Helligkeitsreize als auf Farbreize.
Der Mensch ist ein *Trichromat*, d. h. ein Drei-Farben-Seher. Es gibt drei Arten von Zäpfchen, die je für einen Wellenlängenbereich besonders empfindlich sind. Alle Farben, die das Auge wahrnehmen kann, können auf eine additive Mischung dieser drei Grundfarben zurückgeführt werden.
Für die Lichtwahrnehmung gilt außerdem LECHNERs *Gesetz*: Die Beziehung zwischen der ins Auge einfallenden und der wahrgenommenen Lichtintensität ist nicht linear, sondern annähernd logarithmisch. Das hat z. B. für das Schwarz-Weiß- oder Grauwert-Sehen zur Folge, dass kleine Intensitätsunterschiede in dunklen Regionen stärker wahrgenommen werden als in hellen.
Weiter gibt es den *Mach-Band-Effekt*: Abrupte Intensitätsänderungen innerhalb eines Bildes werden durch die Interaktion der Lichtrezeptoren im Auge zusätzlich betont. Das kann bei der Bildsynthese zu unschön überzeichneten Kanten führen.

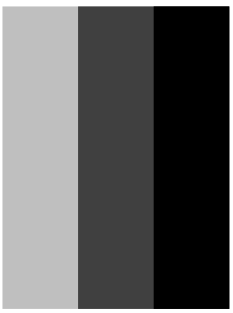


Abbildung 3.51. *Mach-Band-Effekt*

Schließlich spielen psychologische Aspekte bei der Farbwahrnehmung eine Rolle: Mit Rot, Orange und Gelb wird Wärme assoziiert; diese Farben lassen Objekte außerdem größer und näher erscheinen. Mit Blautönen verbindet man Kühle sowie Wasser und Himmel; diese Farben lassen Objekte kleiner und ferner erscheinen.

3.1.3 Farbmodelle

Gestützt auf die Trichromateneigenschaft des Menschen stellen viele Farbmodelle eine Farbe als Tripel von Werten, d. h. als Elemente eines dreidimensionalen Farbraums, dar.
Als *Primärfarben* eines Modells bezeichnet man die Farbbasis, d. h. die Farben, aus denen alle anderen zusammengesetzt sind. *Sekundärfarben* sind alle Mischfarben aus genau zwei Primärfarben. Die *Skala (gamut)* eines Farbmodells ist die Gesamtheit der darstellbaren Farben, also der gesamte Farbraum. Die Skalen unterschiedlicher Modelle sind in der Regel unterschiedlich.

3.1.3.1 Das RGB-Modell

Die Primärfarben sind hier Rot = $(1, 0, 0)$, Grün = $(0, 1, 0)$ und Blau = $(0, 0, 1)$, die Sekundärfarben Gelb = $(1, 1, 0)$, Cyan = $(0, 1, 1)$ und Magenta = $(1, 0, 1)$. Weiter ist Schwarz = $(0, 0, 0)$ und Weiß = $(1, 1, 1)$. Den würfelförmigen Farbraum hatten wir bereits in Zusammenhang mit OpenGL besprochen. Dieses Modell passt auf die meisten Farbmonitore und ist daher wohl das wichtigste in der Computergrafik.
Es hat aber einige Nachteile:

- Der RGB-Farbraum deckt nicht alle vom Menschen wahrnehmbaren Farben ab.
- Das Modell ist bezüglich Farbwahrnehmung nicht linear. Bei hohen Bit-Tiefen für die Farbauflösung (ab der True-Color-Auflösung mit 8 Bit pro Pixel und Grundfarbe) kann es dazu kommen, dass in einigen Bereichen des Farbwürfels benachbarte Farbwerte vom Auge nicht mehr unterscheidbar sind, in anderen Bereichen dagegen schon. Hier muss man bei Interpolation zwischen Farben vorsichtig sein.
- Es kann schwierig sein, die RGB-„Koordinaten“ einer gewünschten Farbe (etwa Kastanienbraun) zu finden.
- Das Abschwächen einer Farbe erfordert ungleiche Änderungen in den RGB-Koordinaten und ist daher ebenfalls schwierig

3.1.3.2 Das HSV-Modell

Dieses Modell ist wahrnehmungsorientiert, da jede Farbe durch Ton (**h**ue), Sättigung (**s**aturation) und Intensität (**v**alue) beschrieben wird. Der Farbraum ist eine sechseckige Pyramide. S und V sind auf Koordinatenachsen angetragen, während H den Winkel zwischen der S-Achse und dem Lot vom Farbpunkt auf die V-Achse angibt (Zylinderkoordinaten).

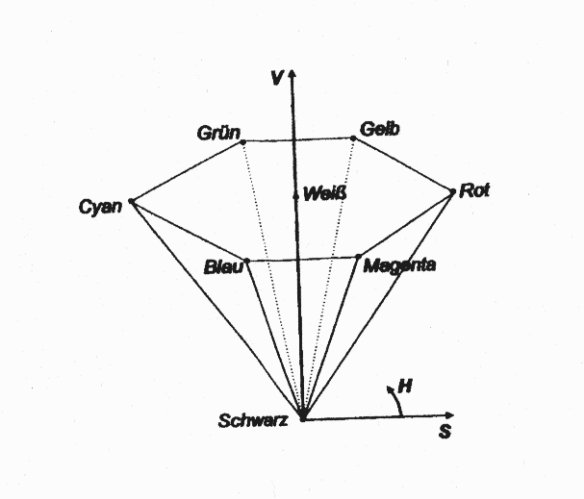


Abbildung 3.52. *Das HSV-Modell*

Es gilt $0° \leq H \leq 360°$ mit den Farbtönen Rot = 0°, Gelb = 60° usw. Weiter ist $V \in [0, 1]$, wobei 1 volle Intensität bedeutet. Für $V = 0$ sind die anderen Koordinaten unerheblich, die Farbe ist schwarz. $S \in [0, 1]$, auch *Chroma* genannt, misst den Weißanteil in der Farbe. Die Grautöne liegen auf der V-Achse.
In diesem Modell lassen sich Farbabschwächungen leicht durch Verändern der S-Koordinate erreichen.
HSV und RGB können ineinander umgerechnet werden. Außerdem ergibt eine Parallelprojektion des RGB-Würfels in Richtung der Raumdiagonalen Weiß-Schwarz genau die Deckfläche der HSV-Pyramide.

3.1.3.3 Das HLS-Modell

Auch dieses Modell ist wahrnehmungsorientiert, es ist eng mit dem HSV-Modell verwandt. L steht für „lightness“ oder „luminance“. Der Farbraum ist hier eine sechsseitige Doppelpyramide. Sie entsteht aus der HSV-Pyramide, indem man den Punkt „Weiß“ aus der Deckfläche herauszieht:

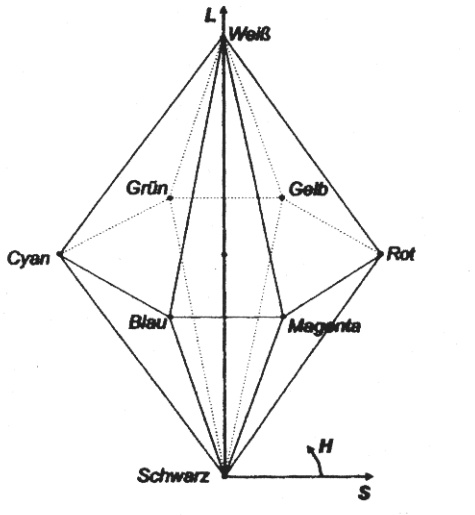


Abbildung 3.53. *Das HLS-Modell.*

Es gilt $L \in [0, 1]$. Für $L = 1$ (Weiß) und $L = 0$ (Schwarz) sind die anderen Koordinaten unerheblich.

3.1.3.4 Additive und subtraktive Modelle

Bei Beleuchten einer Stelle mit verschiedenfarbigen Lichtquellen *addieren* sich deren Farbtöne, es entsteht ein hellerer Ton; Bei Mischung aller Grundfarben ergibt sich Weiß.
Nach Bemalen einer Stelle mit Farbe werden dagegen aus dem auffallenden Licht alle anderen Farben absorbiert; das ist also ein *subtraktiver* Vorgang.
Zur Ausgabe etwa auf Farbdruckern verwendet man daher subtraktive Farbmodelle.

3.1.3.5 Das CMY(K)-Modell

In diesem subtraktiven Modell sind die Primärfarben Cyan, Magenta und Gelb. Die Gewichte der Farbanteile liegen wieder in $[0, 1]$. Die Sekundärfarben sind Rot (Magenta und Gelb), Blau (Cyan und Magenta) und Grün (Cyan und Gelb). Mischt man alle drei Primärfarben, entsteht theoretisch Schwarz. In der Praxis ist diese Mischung aber oft nicht dunkel genug. Daher fügt man oft Schwarz als künstliche vierte Primärfarbe hinzu und erhält so das CMYK-Modell (K für blac**k**); der zugehörige Druckvorgang heißt *Vierfarbenprozess*. RGB und CMY lassen sich leicht ineinander umrechnen:

$$(r, g, b) = (1, 1, 1) - (c, m, y)$$

3.1.3.6 Komplementärfarben

Zwei Farben werden physiologisch als harmonisch empfunden, wenn sie sich zu Weiß addieren. Damit hat man die Komplementärpaare (Rot, Cyan), (Grün, Magenta) und (Blau, Gelb). In subtraktiven Farbräumen werden zwei Farben als komplementär angesehen, wenn ihre Mischung ein Grauton ist.

3.1.3.7 Der CIE-Farbraum

Alle bisherigen Farbmodelle haben den Nachteil, dass sie nicht alle vom Menschen wahrnehmbaren Farben darstellen können.Tatsächlich gelingt das mit drei wahrnehmbaren Primärfarben nur, wenn man auch negative Gewichte zulässt. Der CIE-Standard, 1931 von der *Commission Internationale de l'Éclairage* verabschiedet, löst diese Probleme. Er ist geräteunabhängig und verwendet drei künstliche Primärfarben X, Y und Z bei additiver Farbmischung. Er dient unter anderem zum Vergleich der Farbmodelle; sie lassen sich alle in ihn umrechnen.

3.1.3.8 Farbinterpolation

Alle erwähnten Modelle sind nichtlinear, d. h. eine gleich große Parameteränderung hat an verschiedenen Stellen verschieden große Auswirkungen auf die Wahrnehmung. Daher ist Farbinterpolation eine nichttriviale Aufgabe. Benötigt wird sie z. B. beim Schattieren, bei einigen Anti-Alias-Verfahren, beim Überblenden von Bildern und bei der Visualisierung.
Da auch der Zusammenhang zwischen den Farbmodellen (z. B. zwischen RGB und HSV/HLS) nicht linear (d. h. affin) sein muss, ist es nicht gleichgültig, in welchem Modell man die Interpolation durchführt. Aus Geschwindigkeits- und Gewohnheitsgründen wählt man für die ersten drei der obigen Aufgaben meist RGB.
Echt lineare Abhängigkeiten erreicht man erst in speziellen Farbmodellen, z. B. dem LAB-Modell. Die Umrechnungen sind allerdings sehr kompliziert und erfolgen indirekt über das CIE-Modell. Trotzdem ist dieses Modell inzwischen Bestandteil von Standardsoftware zur Bildbearbeitung.

3.2 Beleuchtung, Reflexion und Transmission

In diesem Abschnitt geht es um folgende Themen:

- Modelle von Lichtquellen
- Reflexion an Objekten
- Transmission von Licht durch Objekte (Transparenz)
- Materialeigenschaften (Oberflächenstrukturen)

Folgende Hauptansätze existieren:

- *Fotorealistische Grafik*, d. h. Erzeugung möglichst wirklichkeitsgetreuer Bilder. Dabei müssen viele physikalische Gesetzmäßigkeiten der Lichtausbreitung nachgebildet werden, was zu hoher Rechenzeit führt. Damit ist dieser Ansatz für interaktive und Echtzeitanwendungen in der Regel zu aufwendig, weil die Algorithmen auch nicht von der Hardware direkt unterstützt werden. Zwei wesentliche Verfahren sind Strahlverfolgung (*ray tracing*), die Rückverfolgung von Lichtstrahlen vom Auge des Betrachters aus, und die Analyse des *Strahlungsaustausches (radiosity)*, d. h. der Wechselwirkung des Lichts mit allen Objektoberflächen in der Szenerie (dieses Verfahren ist unabhängig vom Standort des Betrachters).
- Weniger exakte, „empirische“ Verfahren haben bedeutend kürzere Rechenzeiten bei immer noch guter Wirkung der erzeugten Bilder; sie sind somit echtzeitfähig. Zwei Standardverfahren, die wir auch im Detail besprechen werden, sind *Gouraud-Schattierung* und *Phong-Beleuchtung*. Sie werden in allen modernen Grafikkarten verwendet.
- Man kann sich aber auch völlig von der Forderung nach wirklichkeitsnahem Aussehen lösen. Hierher gehören rechnererzeugte *Spezialeffekte*, die *nichtfotorealistische Grafik* (z. B. Erzeugung handskizzenähnlicher Bilder) und das *Cartoon-Shading*, bei dem Farbverläufe so verfremdet werden, dass die Bilder denen in Comics ähneln.

3.2.1 Die Strahlenoptik (geometrische Optik)

Wir listen hier die wichtigsten Eigenschaften von Lichtstrahlen auf:

- Ausgangsort ist eine punktförmige Lichtquelle, die nach allen Richtungen Strahlen aussendet.
- In einem homogenen Medium breitet sich Licht allseitig und geradlinig aus. Das führt zu scharfen Schattengrenzen.
- Lichtstrahlen können sich kreuzen ohne sich zu stören.
- Lichtwege sind umkehrbar.
- Außerhalb seiner Bahn übt ein Lichtstrahl keine Wirkung aus.
- Bei Auftreffen eines Lichtstrahls auf eine Oberfläche kann er teilweise reflektiert, absorbiert oder durch die Fläche transmittiert werden; bei der Transmission kann es zu Brechungen kommen.

Reflexion und Brechung beschreibt man mit Hilfe eines lokalen Koordinatensystems. Sein Ursprung Ω ist der Auftreffpunkt des betrachteten Strahls L auf der betrachteten Oberfläche F. Der Normalenverktor zu F im Punkt Ω sei $\overrightarrow{n}$, der Richtungsvektor von L, von Ω aus entgegen der Einfallsrichtung aufgetragen, sei $\overrightarrow{l}$.

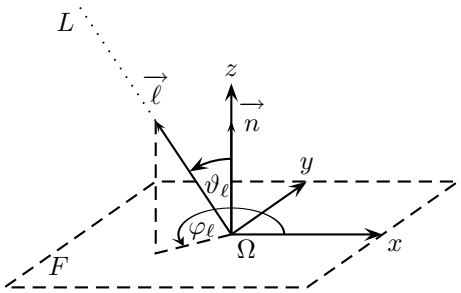


Abbildung 3.54. *Koordinatensystem zur Beschreibung von Reflexion und Brechung*

Es erweist sich als günstig, hier mit Kugelkoordinaten zu arbeiten. Außerdem sollen $\overrightarrow{l}$ und $\overrightarrow{n}$ normiert sein. Also hat der Endpunkt von $\overrightarrow{l}$ die Kugelkoordinaten $(1, \varphi_l, \vartheta_l)$.

3.2.2 Das Reflexionsgesetz

Bei *idealer Reflexion* liegt der reflektierte Strahl R wieder in der von L und $\overrightarrow{n}$ bestimmten Ebene. Sein normierter Richtungsvektor $\overrightarrow{r}$ ist also durch den Endpunkt mit den Kugelkoordinaten $(1, \varphi_l - 180°, \vartheta_l)$ bestimmt, da der Einfallswinkel gleich dem Ausfallswinkel ist.
Hat der Endpunkt von $\overrightarrow{l}$ die kartesischen Koordinaten (l_x, l_y, l_z), so hat der von $\overrightarrow{r}$ die Koordinaten $(-l_x, -l_y, l_z)$ (Punktspiegelung des Lotfußpunkts von $\overrightarrow{l}$ auf die xy-Ebene am Ursprung Ω).
Ideal reflektierende Oberflächen existieren allerdings in der Natur nicht. Tatsächlich wird der Lichtstrahl bei *realer* oder *unvollkommener Reflexion* aufgespalten und in mehrere Richtungen gestreut; es entsteht ein „Streuballon“ um die ideale Reflexionsrichtung. Dabei ist aber der Lichtanteil in der idealen Reflexionsrichtung immer noch am größten.
Eine weitere idealisierte Form ist die *ideal* oder *vollkommen diffuse Reflexion*, bei der ein einfallender Strahl gleichmäßig in alle Richtungen gestreut wird, sodass eine Streuungshalbkugel um Ω entsteht.
In der Realität hat man verschiedenste Zwischenformen zwischen idealer und ideal diffuser Reflexion.
Zur genauen Darstellung des Reflexionsverhaltens muss man aus φ_l und ϑ_l für alle Raumpunkte P berechnen, welcher Anteil von L in P ankommt. Die zugehörige Funktion heißt *bidirektionale Reflexionsverteilung (bi-directional reflection distribution function BRDF).* In den seltensten Fällen ist sie geschlossen darstellbar; meist muss sie durch Messungen bestimmt werden. Für die zu besprechenden lokalen Beleuchtungsmodelle verwendet man hinreichend einfache BRDFs.

3.2.3 Das Brechungsgesetz

Geht Licht von einem Medium in ein anderes über, so ändert sich seine Geschwindigkeit. Je optisch dichter ein Medium ist, desto langsamer breitet sich das Licht darin aus. Daher werden Lichtstrahlen am Übergang zwischen den Medien gebrochen, und zwar umso stärker, je flacher sie auftreffen.
Die *Brechungszahl* n_M eines Mediums M ist $n_M = \frac{c}{c_M}$, wobei c die Lichtgeschwindigkeit im Vakuum und c_M die in M ist. Für Einfalls- und Ausfallswinkel bei Beleuchtung an der Grenze zwischen Medien M_1 und M_2 gilt das *Gesetz von Snellius*:

$$\frac{\sin\vartheta_1}{\sin\vartheta_2} = \frac{n_{M_1}}{n_{M_2}} = \frac{c_{M_2}}{c_{M_1}}$$

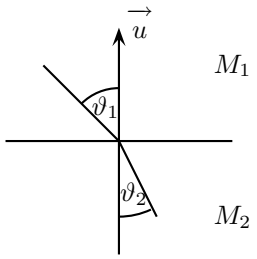


Abbildung 3.55. *Gesetz von Snellius*

Dieses Verhältnis heißt der *Brechungskoeffizient* für diesen Übergang.
Der ausfallende Strahl liegt wieder in einer Ebene mit der Richtung $\vec{l}$ des einfallenden Strahls und dem Normalenvektor $\vec{n}$ auf die Grenzfläche im Auftreffpunkt Ω.
Tritt ein Lichtstrahl in ein Medium ein und dann wieder aus, kommt es zur *Doppelbrechung*:

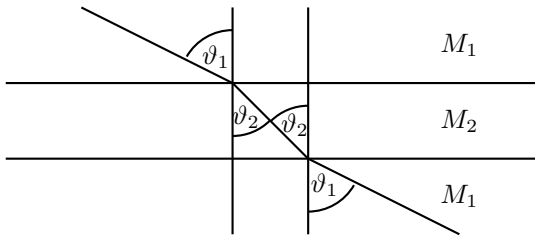


Abbildung 3.56. *Doppelbrechung*

Schließlich kann es beim Übergang von einem optisch dünneren M_1 zu einem dichteren M_2 zur *Totalreflexion* kommen, nämlich dann, wenn der „Austrittswinkel“ größer als 90° wird, weil dann das gebrochene Licht ganz in M_1 verbleibt. Der Winkel ϑ_1, für den $\vartheta_2 = 90°$ wird, heißt *Grenzwinkel* der Totalreflexion.
Die beschriebene ideale Brechung kommt in der Natur so nicht vor; tatsächlich hat man an der Grenzfläche eine Mischung aus Reflexion und Brechung, die auch stark von den Transparenzeigenschaften der Medien abhängt.

3.2.4 Weitere Einflussfaktoren

Neben Reflexion und Brechung spielen *Absorption* und *Emission* eine Rolle. Die Absorption hängt vom Oberflächenmaterial, aber auch von der Wellenlänge des einfallenden Lichts ab; sie bestimmt auch die Helligkeit des Objekts. Emittiert ein Objekt Licht (z. B. glühende Kohle), agiert es zusätzlich als Lichtquelle.
Die wichtigsten Fragestellungen für ein beleuchtetes Objekt:

- Wieviel des einfallenden Lichts wird in welche Richtungen reflektiert?
- Wieviel wird in welche Richtungen in das Objekt hinein gebrochen?
- Wieviel durch das Objekt transmittiertes Licht wird in welche Richtungen gebrochen?
- Wieviel Licht wird absorbiert?
- Wieviel Licht wird in welche Richtungen emittiert?
- Wie wird das Licht bei Durchgang durch ein Medium durch Streuung abgeschwächt?

Die meisten Beleuchtungsmodelle berücksichtigen allerdings nur einen Teil dieser Aspekte.

3.2.5 Arten von Lichtquellen

- *Punktlichtquellen* strahlen gleichmäßig in alle Richtungen aus. Sie sind durch ihren Mittelpunkt und ihre RGB-Intensitäten bestimmt.
- Bei *entfernungsabhängigen Punktlichtquellen* ist der Lichtanteil an einem Auftreffpunkt umso schwächer, je weiter der Punkt von der Quelle entfernt ist, und zwar ungekehrt proportional zum Quadrat der Entfernung. Damit wird modelliert, daß sich das ausgesandte Licht mit zunehmender Entfernung auf immer größere Kugelschalen verteilt.
- Ein *Spotlicht* reduziert ein Punktlicht auf einen Lichtkegel; zusätzliche Parameter sind dessen Achsenrichtung und Öffnungswinkel.
- Ein *Richtungslicht* ist eine Punktlichtquelle im Unendlichen. Damit fallen seine Strahlen parallel ein und für alle Objekte ergibt sich, im Gegensatz zu den vorigen Quellen, der selbe Richtungsvektor zum Licht hin.
- Die bisherigen Lichtquellen erzeugen harte Schatten. Soll das abgemildert werden, kann man mit mehreren benachbarten Punktenlichtern arbeiten. Sind das sehr viele, entsteht eine *Flächenlichtquelle*. Das kann aber zu sehr aufwendigen Berechnungen führen.

3.3 Beleuchtung und Schattierung

Die Beleuchtung einer Szenerie kann lokal oder global modelliert werden

- Ein *lokales Beleuchtungsmodell* berechnet die Intensität bzw. Farbe eines Objektpunkts abhängig vom direkten Lichteinfall aus einer oder mehreren Quellen; es berücksichtigt nun die *direkte Beleuchtung*. Verdeckung durch andere Objekte und Schattierung werden nicht berücksichtigt. Das bekannteste solche Modell stammt von Phong.
- Ein *globales Beleuchtungsmodell* berücksichtigt zusätzlich das indirekt einfallende Licht, das nach Reflexion(en) an oder Transmission(en) durch die eigene oder andere Oberflächen eintrifft; es berücksichtigt *direkte* und *indirekte Beleuchtung*. Hier werden auch globale Effekte wie Verdeckungen und Spiegelungen einbezogen. Solche Modelle entstehen oft durch Erweiterung lokaler Modelle. Die bekanntesten sind *Ray-Tracing* und *Radiosity*.

Die *Schattierung (shading model)* bestimmt, wann ein Beleuchtungsmodell angewandt wird. Beim Ray-Tracing wird das Modell auf jedes Pixel angewandt, das zu einem Objektpunkt gehört. Dagegen werten spezielle Schattierungsverfahren für Polygon-Netze, sog. *interpolative* Techniken, das Beleuchtungsmodell nur an ausgewählten Punkten aus. Intensitäts- und Farbwerte für die Zwischenpunkte werden durch Interpolation ermittelt. Die bekanntesten Verfahren dazu sind Flächenschattierung (flat shading) und die Verfahren von Gouraud und Phong.

3.3.1 Das Beleuchtungsmodell von Lambert

Wir betrachten nun eine Punktquelle und nicht farbiges, d. h. nur in Graustufen variierendes Licht. Das *Lambert-Modell* simuliert die Beleuchtung ideal diffus reflektierender Oberflächen. Die auf eine Fläche einfallende Helligkeit hängt von der Neigung der Fläche zu den Lichtstrahlen ab: Nur der Querschnitt des Teilkegels, der eine Fläche A trifft, ist maßgebend. Das ist die Projektion $A_\perp$ von A auf die Ebene senkrecht zum Richtungsvektor des Lichts (bezogen auf den Mittelpunkt der Fläche). Ist ϑ der Einfallswinkel des Lichts, so gilt $|A_\perp| = |A| \cos\vartheta$, im Querschnitt:

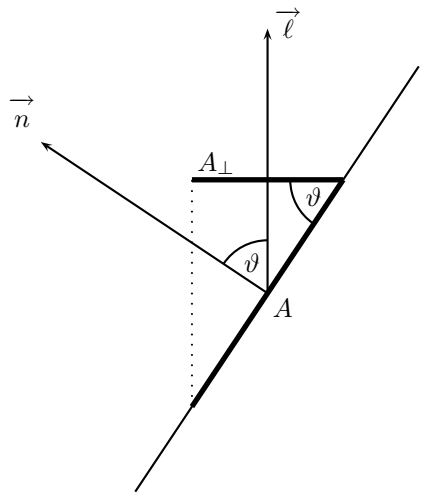


Abbildung 3.57. *Einfallswinkel im Querschnitt*

Ist $\overrightarrow{l}$ der Richtungsvektor des Lichts und $\overrightarrow{n}$ der Normalenvektor der Fläche, beide normiert, so hat man $\cos\vartheta = \langle \overrightarrow{l}, \overrightarrow{n} \rangle$. Da wir uns nur für die Beleuchtung der Vorderseite interessieren, werden nur Winkel $\vartheta \in [-90°, 90°]$ betrachtet, also negative Werte von $\langle \overrightarrow{l}, \overrightarrow{n} \rangle$ ausgeschlossen. Damit ist die einfallende Intensität proportional zu $\max(0, \langle \overrightarrow{l}, \overrightarrow{n} \rangle)$.
Bei ideal diffuser Reflexion ist die Intensität I_d des ausfallenden Lichts proportional zur Intensität I_l des einfallenden. Der *diffuse Reflexionskoeffizient* r_d gibt an, welcher Anteil des einfallenden Licht diffus gestreut wird. Damit gilt im Lambert-Modell

$$I_d = I_l \cdot r_d \cdot \max(0, \langle \overrightarrow{l}, \overrightarrow{n} \rangle)$$

Maximale Reflexion erfolgt bei $\vartheta = 0°$, minimale bei $\vartheta = 90°$, da dann der Strahl an der Oberfläche entlang läuft.

3.3.2 Das Beleuchtungsmodell von Phong

3.3.2.1 Der Grundansatz

Im Phong-Modell (1975) wird zusätzlich zum Lambert-Modell noch eine unvollkommene Reflexion einbezogen. Der höchste Reflexionsanteil fällt in die ideale Reflexionsrichtung $\overrightarrow{r}$. Je größer dann der Winkel α zwischen der Betrachtungsrichtung $\overrightarrow{v}$ und $\overrightarrow{r}$ ist, desto kleiner ist die reflektierte Intensität in dieser Richtung. Das wird durch eine Potenz von $\cos\alpha = \langle \overrightarrow{r}, \overrightarrow{v} \rangle$ ausgedrückt:

$$I_s = I_l \cdot r_s \cdot \max(0, \langle \overrightarrow{r}, \overrightarrow{v} \rangle^n)$$

Dabei ist

- I_s die Intensität des reflektierten Strahls,
- I_l die Intensität des einfallenden Lichts,

- n der *Spiegelungsexponent*,
- r_s der *spiegelnde Reflexionskoeffizient*.

r_s gibt an, welcher Anteil des Lichts überhaupt reflektiert wird. n drückt aus, wie nahe die Oberfläche an einer ideal reflektierenden ist; es liegt typischerweise zwischen 1 und 1000. Da für $q \in [0,1[$ gilt $q^n \to 0$ für $n \to \infty$, nähert sich für große n die Funktion $f(\alpha) = \cos^n \alpha$ immer mehr der Sprungfunktion

$$g(\alpha) = \begin{cases} 1 & \alpha = 0° \\ 0 & \text{sonst} \end{cases}$$

auf $[-90°, 90°]$ an; g beschreibt gerade die ideale Reflexion.
Für kleinere n ergibt sich dagegen als Abbild einer Lichtquelle auf einem Objekt ein nach außen schwächer werdendes Lichtfeld, das als *Glanzpunkt (highlight)* bezeichnet wird.

3.3.2.2 Ambientes Licht

Sowohl im Lambert- wie im Phong-Modell bleiben nicht direkt beleuchtete Oberflächenteile schwarz, da auf sie kein reflektierbares Licht auftrifft.
Daher nimmt man oft ein allgemeines Hintergrundlicht, die *ambiente Beleuchtung*, an. Sie modelliert z. B. die allgemeine Helligkeit bei Tag.
Das führt zu einem weiteren Summanden für die Intensität:

$$I_a = I_A \cdot r_a$$

Dabei ist I_A die Gesamtintensivität des ambienten Lichts, und r_a, der *ambiente Reflexionskoeffizient*, gibt an, wie stark dieses Licht wirkt.

3.3.2.3 Das Gesamtmodell

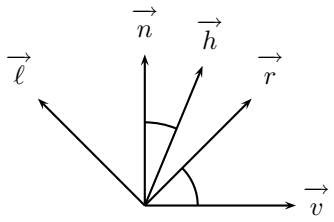
Als Gesamtintensität nach Phong ergibt sich für einfarbiges Licht

$$I = I_a + I_s + I_d = I_A \cdot r_a + I_l \cdot r_d \cdot \max(0, \langle \overrightarrow{l}, \overrightarrow{n} \rangle) + I_l \cdot r_s \cdot \max(0, \langle \overrightarrow{r}, \overrightarrow{v} \rangle^n)$$

Dabei sind die Parameter r_a, r_d, r_s und n abhängig vom bzw. charakteristisch für das Material der betrachteten Oberfläche.
Farbiges Licht modelliert man als drei separate monochrome Lichtquellen mit den Primärfarben RGB und erhält Intensitäten I_r, I_g und I_b, die sich nach obiger Formel bestimmen. Dabei ist n allen drei Anteilen gemeinsam, während die Reflexionskoeffizienten nach Farben variieren können.
Die diffusen Koeffizienten $(r_{d_r}, r_{d_g}, r_{d_b})$ sind hauptverantwortlich für die Grundfarbe eines Objekts; sie bestimmen seine Farbe bei Beleuchtung mit weißem Licht. Reflektiertes Licht hat oft die Farbe des einfallenden Lichts. Z. B. ist der Glanzpunkt bei gelbem Licht auf einem Apfel wieder gelb. Um das zu modellieren muss man $r_{d_r} = r_{d_g} = r_{d_b}$ wählen.
Die Koeffizienten für einige typische Oberflächenmaterialien lauten so:

Material	r_{ar}, r_{ag}, r_{ab}	r_{dr}, r_{dg}, r_{db}	r_{sr}, r_{sg}, r_{sb}	n
Schwarzes Plastik	0.00, 0.00, 0.00	0.01, 0.01, 0.01	0.50, 0.50, 0.50	32
Messing	0.33, 0.22, 0.03	0.78, 0.57, 0.11	0.99, 0.94, 0.81	28
Bronze	0.21, 0.13, 0.05	0.71, 0.43, 0.18	0.39, 0.27, 0.17	26
Kupfer	0.19, 0.07, 0.02	0.70, 0.27, 0.08	0.26, 0.14, 0.09	13
Gold	0.25, 0.20, 0.07	0.75, 0.61, 0.23	0.63, 0.56, 0.37	51
Silber	0.19, 0.19, 0.19	0.51, 0.51, 0.51	0.51, 0.51, 0.51	51

Da die Parameter des Modells empirisch bestimmt werden, kann die Summe der Einzelterme für eine einzelne Farbkomponente einen Wert größer als Eins ergeben; diese „Überbelichtung“ wird durch Abschneiden auf Eins korrigiert. Das gilt auch für das Zusammenwirken mehrerer Lichtquellen.
Aus Effizienzgründen verwendet man oft (auch in OpenGL) eine Abwandlung des Phong-Modells. Statt des Winkels zwischen $\overrightarrow{r}$ und $\overrightarrow{v}$ benützt man den Winkel zwischen $\overrightarrow{n}$ und dem *Mittelvektor (halfway vector)* $\overrightarrow{h} = \frac{\overrightarrow{l} + \overrightarrow{v}}{\| \overrightarrow{l} + \overrightarrow{v} \|}$; die sich ergebende Ungenauigkeit wird durch einen veränderten Spiegelungsexponenten n kompensiert.



Da die Vektoren des Modells alle dreidimensional sind, muss die Berechnung der Beleuchtung vor der Projektion erledigt werden. Die Beleuchtungswerte werden dann beim Rastern zur Bestimmung der Farbwerte der Pixel verwendet.

3.3.3 Übersicht über globale Beleuchtungsmodelle

3.3.3.1 Strahlverfolgung

Dieses Verfahren nützt aus, dass Strahlwege umkehrbar sind. Es arbeitet im Bildraum und verfolgt für jedes Pixel den Strahl vom Augpunkt zurück in die Szene; er heißt *Primärstrahl* P. Am ersten Schnittpunkt S von P mit einem Objekt wird mit einem lokalen Beleuchtungsmodell berechnet, wieviel Licht in Richtung P zurückgeworfen wird und das Pixel entsprechend eingefärbt. Dazu benötigt man allerdings zuerst die Information, wieviel Licht woher in S eintrifft. Um das zu berechnen, schickt man, gemäß Reflexions- und Brechungsgesetz, *Sekundärstrahlen* von S aus in die Szenerie hinein und verfährt mit ihnen rekursiv genauso weiter. Damit wird klar, wieso das Verfahren so rechenintensiv ist: die Anzahl der rekursiven Aufrufe vervielfacht sich in jedem Schritt, was zu einer exponentiellen Gesamtkomplexität führt. Der Vorteil des Verfahrens liegt in der exakten Behandlung von Spiegelungen von Objekten in anderen und von Schatten.
Ein bekanntes und mächtiges Werkzeug zur Erzeugung von Bildern nach dem Strahlverfolgungsverfahren ist POV (persistence of vision, `http://www.povray.org`).

3.3.3.2 Strahlungsanalyse

Dieses Modell beschreibt das Gleichgewicht von emittierter, reflektierter und absorbierter Lichtenergie in einem geschlossenen System durch ein System von Gleichungen gemäß den Hauptsätzen der Thermodynamik. Die Hauptannahmen sind:

- alle Flächen der Szenerie reflektieren ideal diffus;
- alle Flächen sind homogen;
- die Szenerie ist abgeschlossen, Energie wird weder zu- noch abgeführt.

Im Gegensatz zur Strahlverfolgung wird hier nicht punkt- sondern flächenweise gerechnet. Dabei sind auch Flächenlichtquellen gut behandelbar.
Die wesentlichen Gleichungen besagen für jede Fläche:

$$\text{abgestrahlte Energie} = \text{emittierte Energie} + \text{reflektierte Energie}$$

Die reflektierte Energie ergibt sich aus den abgestrahlten Energien der anderen Flächen gemäß Reflexions- und Brechungsgesetz. Die Lösung der Gleichungen ergibt dann für jede Fläche der Szenerie die Energiedichte, die der auf dem Bildschirm darzustellenden Farbe entspricht. Diese Werte sind unabhängig vom Standort des Betrachters; sie müssen nur dann neu berechnet werden, wenn sich Objektfarben oder die Intensitäten von Lichtquellen ändern. Die entstehenden linearen Gleichungssysteme lassen sich mit höchstens kubischem Aufwand in der Anzahl der Flächen lösen. Damit ist das Verfahren für diffus reflektierende Szenerien wesentlich effizienter als Strahlverfolgung. Derzeit wird an Kombinationen der beiden Verfahren gearbeitet.

3.4 Schattierung von Polygonnetzen

Im Prinzip könnte man mit lokalen Modellen für jeden Punkt einer Fläche die Farbe berechnen. Das wäre aber viel zu ineffizient und auch „an der falschen Stelle genau“. Denn die verwendeten Polygonnetze sind ja ohnehin nur Approximationen der realen Objekte in der Szenerie. Vielmehr reicht es aus, die Beleuchtung einiger ausgewählter Punkte exakt zu berechnen und für die übrigen Interpolation zu verwenden. Eine wichtige Teilaufgabe dabei ist es, „uneigentliche“ Kanten, die durch die Approximation entstehen, in der Darstellung möglichst wieder verschwinden zu lassen.
Bei den zu besprechenden Verfahren spielen neben den bereits erwähnten Flächennormalen auch *Eckennormalen* eine Rolle, besonders bei der Kantenglättung. Hier betrachtet man nur Ecken, bei denen die Normalen der angrenzenden Flächen nicht allzu stark voneinander abweichen. Dabei ist darauf zu achten, dass alle an die Ecke grenzenden Flächen einheitlich orientiert sind, also deren Ecken stets im Uhrzeiger- oder Gegenuhrzeigersinn durchlaufen werden. Dann ist die Eckennormale das arithmetische Mittel der Normalen der angrenzenden Flächen.

3.4.1 Flächenschattierung (*flat shading*)

Hier wählt man für jede Fläche einen Punkt, ermittelt dessen Farbe und färbt damit die ganze Fläche gleichmäßig ein. Zur Berechnung verwendet man hier die Flächennormale im gewählten Punkt (z. B. einem Eckpunkt).
Dabei bleiben die Kanten des Polygonnetzes in der Regel sichtbar und die Objekte werden eckig dargestellt. Eine einigermaßen „runde“ Ansicht ergibt sich nur bei Verwendung sehr vieler sehr kleiner Flächen.
Trotzdem hat das Verfahren noch eine hohe Bedeutung, weil es einfach und effizient implementierbar ist. Man verwendet es gern in Vor- oder Entwurfsansichten in Modellierpaketen oder wenn es, wie in PDAs, noch keine Hardwareunterstützung für die Grafik gibt.

3.4.2 Gouraud-Schattierung

Hier wertet man das Beleuchtungsmodell nur an den Ecken des Netzes anhand der Eckennormalen aus. Für die Pixel, die die inneren Flächenpunkte darstellen, wird der Farbwert durch Interpolation aus den Eckwerten gewonnen.

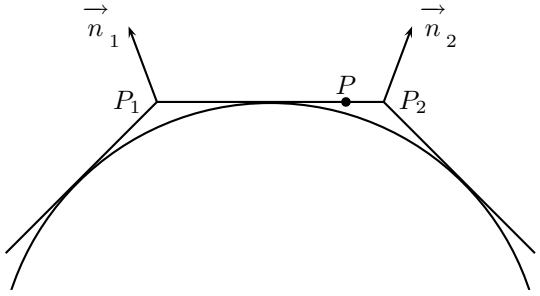


Abbildung 3.58. *Gouraud-Schattierung*

Der Farbwert des Punkts P ermittelt sich aus denen von P_1 und P_2 gemäß dem Streckenverhältnis $\frac{|PP_2|}{|PP_1|}$.
Der entstehende Farbverlauf ist auch an den Kanten stetig, allerdings aber nicht glatt (in dem Sinne, dass auch die 1. Ableitung noch stetig wäre). Das Auge nimmt solche Kanten, ähnlich wie beim Mach-Bandeffekt, wahr. Daher muss man auch hier gelegentlich zu einem feineren Netz greifen.
Ein Hauptnachteil des Verfahrens ist, dass es Glanzpunkte nicht gut darstellt. Weil das Beleuchtungsmodell nur an den Ecken ausgewertet wird, gehen Glanzpunkte in Flächenmitte „verloren“, während sie in Eckennähe über alle angrenzenden Flächen „verschmiert“ werden. Das kann wieder durch feinere Tesselierung kompensiert werden.
Trotzdem ist Gouraud-Schattierung heute das Standardverfahren in Grafikkarten.

3.4.3 Phong-Schattierung

Auch dieses Verfahren dient zur Kantenglättung. Hier wird das Beleuchtungsmodell wieder für alle Punkte auf einer Fläche ausgewertet. Allerdings verwendet man nicht die echte Flächennormale, sondern eine interpolierte Richtung zwischen den nächstliegenden Eckennormalen, weil diese besser der Normalen zur approximierten Fläche entspricht.

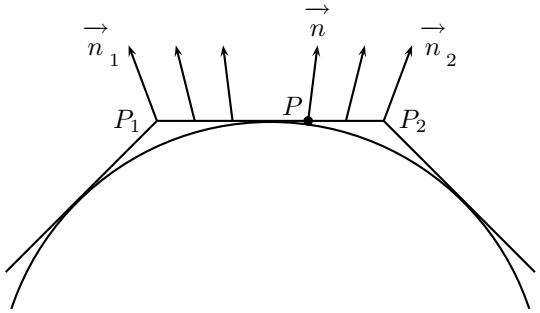


Abbildung 3.59. *Phong-Schattierung*

Dabei wird das Erscheinungsbild der approximierten krummen Fläche gut angenähert. Auch Glanzpunkte werden sehr gut dargestellt. Allerdings ist das Verfahren sehr rechenintensiv und nicht für das pixelorientierte Arbeiten geeignet, da es die Information über die 3D-Normalenvektoren benötigt. Es ist daher nicht gut von der Hardware zu unterstützen, sondern hat seinen Platz eher in globalen Verfahren wie der Strahlverfolgung.

3.4.4 Schattierung in OpenGL

OpenGL unterstützt nur Flächen- und Gouraud-Schattierung. Sie werden mit

```
glShadeModel(GL_FLAT)
```

bzw.

```
glShadeModel(GL_SMOOTH)
```

eingestellt.

3.5 Strukturierung von Flächen

Man könnte Oberflächenstrukturen zwar im Prinzip durch hinreichend feine Tesselierung darstellen. Dabei würde aber die Modell- und auch die Rechenkomplexität viel zu sehr steigen. Daher sieht man andere Möglichkeiten zur Strukturierung vor.

3.5.1 Texturen

Die Grundidee geht auf Catmull 1974 zurück. Man trägt dabei im Wesentlichen ein zweidimensionales Bild auf eine Fläche auf. Da Bilder oft rechteckig vorliegen, ist eine Hauptaufgabe, geeignete Verzerrungen auf die vorgegebene Flächenform zu erreichen. Außerdem ist ein großes Bild über viele aneinanderstoßende Flächen zu verteilen.
Als Erweiterungen lässt man zu, auch andere Flächenattribute über Texturen zu manipulieren. Außerdem betrachtet man auch ein- oder dreidimensionale Texturobjekte.

3.5.1.1 Texturkoordinaten

Wir betrachten nun zweidimensionale Texturen. Sie können durch zwei Koordinaten $s, t \in [0,1]$ und die zugeordneten Farbwerte beschrieben werden[8]. Die Menge der Farbwerte sei F. Also ist eine *Textur* eine Abbildung $tx : [0,1] \times [0,1] \to F$. Die Koordinatenmenge $[0,1]^2$ heißt *Texturraum*, die Paare $(s,t) \in [0,1]^2$ heißen *Texturkoordinaten*.
Zum Anbringen einer Textur auf einer Oberfläche muss für jeden Flächenpunkt eine Texturkoordinate (s,t) angegeben und dem Punkt der Farbwert $tx(s,t)$ zugeordnet werden. Wir benötigen also eine *Texturabbildung* $ta : K \to [0,1]^2$, wenn $K \subseteq \mathbb{R}^3$ die Menge der Punktkoordinaten der Fläche im Objektkoordinatensystem ist.
Für Polygonnetze gibt man ta oft nur für die Eckpunkte an und interpoliert daraus wieder, wie bei der Schattierung, die Werte für die Flächenpunkte.
Die Texturwerte müssen dann mit den eigentlichen Farbwerten verrechnet werden. Das kann neben reiner Farbänderung auch zu Änderungen der Transparenz führen. Eine übliche Kombination bestimmt die diffuse Objektfarbe über die Textur und die spiegelnden Farbanteile über die normale Beleuchtungsberechnung.
Man kann auch mehrere Texturen zu einer *Multitextur* kombinieren.

3.5.1.2 Bitmap-Texturen

Bitmaps sind pixelbasiert, also diskret, während wir Texturen ja als kontinuierliche Funktionen definiert haben. Damit muss bei Verwendung einer Bitmap b der Größe $m \times n$ gerundete Indizierung verwendet werden:

$$tx(s,t) = b[rd(s \cdot (m-1))][rd(t \cdot (n-1))],$$

wobei $rd : \mathbb{R} \to \mathbb{Z}$ die Rundungsfunktion ist. Die einzelnen Pixel einer Bitmaptextur werden auch *Texel* genannt.

[8] s und t können dabei durchaus unterschiedliche Maßstäbe bedeuten.

Das obige Verfahren lässt sich gut von der Hardware unterstützen. Oft beschränkt man die Größen m und n auf Zweierpotenzen.

3.5.1.3 Prozedurale Texturen

Hier werden die Werte von tx über eine mathematische Funktion berechnet. Für einfache Muster wie schachbrettartige Rasterung ist das leicht. Für stein- oder holzartige Maserungen dagegen ist es eine Mischung aus Mathematik und Kunst.

3.5.1.4 Texturabbildungen

Für einfache Flächen wie Rechtecke, Zylinder- oder Kugelflächen lässt sich die Texturabbildung affin beschreiben.
Für ein achsenparalleles Rechteck der Breite b und der Höhe h in der xy-Ebene ergibt sich, wenn eine Ecke im Ursprung und das Rechteck im ersten Quadranten liegt,

$$s(x,y,z) = \frac{x}{b} \qquad t(x,y,z) = \frac{y}{h}$$

für $x \in [0,b], y \in [0,h], z \in 0$.
Entspricht das Verhältnis $\frac{b}{h}$ nicht dem der Maßstäbe für s und t, wird die Textur dabei verzerrt.
Ein Zylindermantel kann zu einem Rechteck abgewickelt werden und lässt sich daher analog behandeln, indem man die Punkte auf der Fläche durch Zylinderkoordinaten darstellt. Sei r der Zylinderradius und h die Höhe. Die Punkte auf einem auf der xy-Ebene stehenden Zylinder haben die Koordinaten $(r\cos\varphi, r\sin\varphi, z)$ mit $\varphi \in [0, 2\pi]$ und $z \in [0,h]$, und sind somit durch φ und z vollständig beschrieben.
Will man z. B. eine Textur mittig auf eine Fläche mit $\frac{h}{2}$ als Höhe und einem Viertel des Zylinderumfangs als Breite platzieren, ergibt sich

$$s(\varphi,z) = \frac{2\varphi}{\pi} \text{ und } t(\varphi,z) = \frac{2(z-\frac{h}{4})}{h}$$

für $\varphi \in [0, \frac{\pi}{2}]$ und $z \in [\frac{h}{4}, \frac{3h}{4}]$.
Zum Texturieren eines „ rechteckigen“ Kugelausschnitts verwendet man Kugelkoordinaten. Für $\varphi \in [\varphi_l, \varphi_r]$ und $\vartheta \in [\vartheta_u, \vartheta_o]$ ergibt sich

$$s(\varphi,\vartheta) = \frac{\varphi - \varphi_l}{\varphi_r - \varphi_l} \qquad t(\varphi,\vartheta) = \frac{\vartheta_u - \vartheta}{\vartheta_u - \vartheta_o}$$

Hierbei wird das Texturbild natürlich verzerrt, und zwar umso stärker, je weiter oben auf der Kugel der Bildausschnitt aufgetragen wird.
Für allgemeinere Flächen, die nicht selbst analytisch durch Koordinatenfunktionen beschrieben sind, verwendet man oft das folgende zweistufige Verfahren: Man wählt ein der Fläche „ähnliches“ einfaches Zwischenobjekt, für das die Texturabbildung exakt angegeben werden kann, also z. B. einen Quader, Zylinder oder eine Kugel. Mit diesem Objekt wird das eigentliche umhüllt und dann die Textur des Hüllobjekts H geeignet auf das Originalobjekt O projiziert.
Folgende Projektionsarten für einen Punkt P von O sind gängig:

- Ideale Reflexion eines Strahls von H an P und Ablesen der Textur am Austrittspunkt des reflektierten Strahls aus H.
- Verfolgen des Strahls vom Mittelpunkt von O durch P bis zu H.
- Verfolgen des O-Normalenvektors in P bis zu H.
- Verfolgen eines H-Normalenvektors bis zu P.

Mit diesem Verfahren hat der Anwender eine recht gute Kontrolle, wie seine Textur auf dem Originalobjekt wirken wird.

3.5.1.5 Texturfilterung

Die in 3.5.1.2 beschriebene Verwendung von Bitmaps hat in dieser einfachen Form einige Nachteile.
Wenn sich z. B. die Werte zweier benachbarter Texturkoordinaten stark unterscheiden, kann bei einer nur leichten Kameraveränderung durch das unterschiedliche Rundungsverhalten der Texturwert einer Oberfläche plötzlich „springen“, was bei Animationen zu unschönen Flackereffekten führen könnte.
Besser wäre es, für Koordinaten, deren Bilder zwischen zwei diskreten Texturkoordinaten liegen, zwischen den entsprechenden Texturwerten zu interpolieren.
Ein weiteres Problem ist, dass Pixel und Texel unterschiedliche Form und Größe haben. Ein quadratisches Texel kann durch die Texturabbildung auf ein nichtquadratisches und sogar nichtplanares Gebiet abgebildet werden. Außerdem wird es anschließend noch projiziert. Es kommt also nur in Ausnahmefällen als quadratischer Bereich auf dem Ausgabegerät an. Umgekehrt entspricht ein quadratisches Pixel nur in Ausnahmefällen einem quadratischen Texturbereich.
Überdeckt ein Pixel mehrere Texel, muss dieser Teil der Textur „verkleinert“ werden (*minification*). Muss umgekehrt ein Texel mehrere Pixel „versorgen“, muss es vergrößert werden (*magnification*).
Schließlich muss noch auf Vergrößerung/Verkleinerung der Objekte selbst Rücksicht genommen werden. Dazu speichert man beim *mip-mapping* eine Textur in verschieden großen, bereits vorgefilterten Auflösungsstufen, zwischen denen dann je nach Bedarf umgeschaltet wird. Dieses Verfahren wird heute auch von der Hardware unterstützt.

3.5.2 Weiterführende Verfahren

3.5.2.1 Bump-Mapping

Dieses Verfahren dient dazu, höckerige oder porige Oberflächen darzustellen, ohne jede einzelne Erhöhung oder Vertiefung geometrisch zu modellieren. Stattdessen simuliert man die Strukturen durch Änderungen an der Beleuchtung. Das Verfahren von Blinn 1978 funktioniert so: Zu einem Punkt P der betrachteten Oberfläche werden in der Tangentialebene durch P zwei Störvektoren gewählt, die zur Normalen durch P addiert werden und damit den diffusen Reflexionsanteil verändern. Als Störgewichte dienen die partiellen Ableitungen $\frac{\partial tx}{\partial s}$ und $\frac{\partial tx}{\partial t}$ der Texturfunktion, die z. B. durch Differenzenquotienten angenähert werden können. Allerdings bleiben beim Bump-Mapping die Ränder glatt.

3.5.2.2 Displacement-Mapping

Hierbei wird nicht die Beleuchtung manipuliert, sondern es werden tatsächlich Oberflächenpunkte verschoben. Im einfachsten Fall geschieht das in Richtung der Normalen und der Texturwert gibt an, wie weit. Also stellt die Textur in diesem Fall ein Höhenfeld dar.
Damit der Effekt stark genug wird, müssen allerdings zuerst genügend viele Zwischenpunkte auf den Oberflächen berechnet werden, die dann verschoben werden können. Damit kann dieses Verfahren sehr aufwendig werden.

3.5.2.3 Schattenpufferung

Im Rest dieses Abschnitts besprechen wir, wie über erweiterte Texturbehandlung auch einige globale Effekte, wie Schatten und Spiegelungen, dargestellt werden können.
Die *Schattenpufferung* ist ein zweistufiges Verfahren zur Schattenberechnung. Sein Kern ist die Verwendung des Tiefenpuffers, um zu bestimmen, welche Punkte von Lichtquellen aus sichtbar sind (die übrigen liegen im Schatten). Für eine einzelne Lichtquelle L geht das so: Im ersten Schritt wird mit Tiefenpufferung ein Tiefenprofil der Szenerie erstellt, der *Schattenpuffer* SP.
Für jeden Objektpunkt P wird nun der Abstand d von P zu L mit dem SP-Wert in dieser Richtung verglichen. Ist d kleiner, so liegt P bezüglich L im Schatten. SP wird durch gewöhnliche Tiefenpufferung erstellt, wobei allerdings der Bildpuffer nicht verändert wird. Da SP nicht vom Augpunkt abhängt, können die Werte abgespeichert und wiederverwendet werden, wenn sich nur der Augpunkt verändert.
Im zweiten Schritt wird die Szenerie dann dargestellt. Man verwendet wieder einen Tiefenpufferansatz: Liegt ein Punkt P aus Kamerasicht am weitesten vorne und steht somit zur Darstellung an, wird mit SP geprüft, ob P im Schatten von L liegt. Wenn ja, wird er nur ambient beleuchtet, ansonsten von L aus mit dem jeweiligen Beleuchtungsmodell.
Der Aufwand steigt linear mit der Zahl der Lichtquellen. Alle Stufen des Verfahrens werden von der Hardware unterstützt. Allerdings ist es für hohe SP-Auflösungen und mehrere Lichtquellen wieder sehr speicheraufwendig. Außerdem ist es sehr anfällig für Abtastfehler bei der Schattenberechnung.

3.5.2.4 Reflection-Mapping

Hierbei werden neben den Glanzpunkten auch Spiegelungen von Objekten in Oberflächen dargestellt. Auch bei diesem Verfahren war Blinn 1976 maßgeblich beteiligt.
Beim einfachsten Verfahren, dem *Chrome-Mapping*, wird das Bild einer unscharfen Textur über eine (fast beliebige) Texturabbildung auf eine stark spiegelnde Oberfläche aufgebracht. Im Gegensatz zur reinen Texturierung ist die Bindung aber nicht fest; die Texturkoordinaten auf dem Objekt können sich bei Bewegungen von Objekt oder Kamera ändern. Es entstehen verwaschene Spiegelungen wie an chromähnlichen Oberflächen.
Beim *Environment-Mapping* soll dagegen das gespiegelte Objekt klar erkennbar sein. Man benutzt eine Projektion der Welt, die *Umgebungsabbildung*. Will man ein kleines spiegelndes Objekt mit einem Mittelpunkt P modellieren, so legt man zuerst um P eine Kugel, auf die man die ganze sichtbare Welt projiziert.

Damit definiert die Kugelfläche eine 2D-Textur. Das spiegelnde Objekt wird nun dargestellt, indem man für jeden seiner Oberflächenpunkte mittels der idealen Reflexionsrichtung vom Augpunkt aus den Wert der Kugeltextur abliest. Das ist zwar nur für ein punktförmiges Objekt exakt, aber für kleine spiegelnde Objekte immer noch ausreichend, wenn die gespiegelten Objekte weit entfernt sind. In OpenGL wird die automatische Berechnung der entsprechenden Texturkoordinaten unterstützt.
In einer Erweiterung des Verfahrens nimmt man statt der Kugel einen Würfel; außerdem spaltet man zur genaueren Berechnung die Lichtstrahlen in Pyramiden mit kleinem Öffnungswinkel auf.
Beide Varianten werden auch von der Hardware unterstützt.

3.6 Schattenberechnung

Schatten sind für die realistische Darstellung von Szenerien sehr wichtig, da der Mensch aus dem Alltag gewohnt ist, Schatten zu sehen, so dass Bilder ohne Schatten sehr unnatürlich wirken. Außerdem erleichtern es Schatten, die räumliche Anordnung einer Szene zu begreifen.

3.6.1 Grundlagen

Man unterscheidet zwei Arten von Schatten:

- *Kernschatten (umbra)*, d. h. völliges Fehlen von Licht. Er entsteht, wenn nur eine Lichtquelle vorhanden ist.
- *Schlag-* oder *Halbschatten (penumbra)*, d. h. nur teilweises Fehlen von Licht. Schlagschatten entstehen, wenn mehrere Lichtquellen oder eine nicht genau punktförmige Lichtquelle beteiligt sind.

Als Illustration könnten die bekannten Bilder zur Erklärung von Mond- und Sonnenfinsternissen dienen:

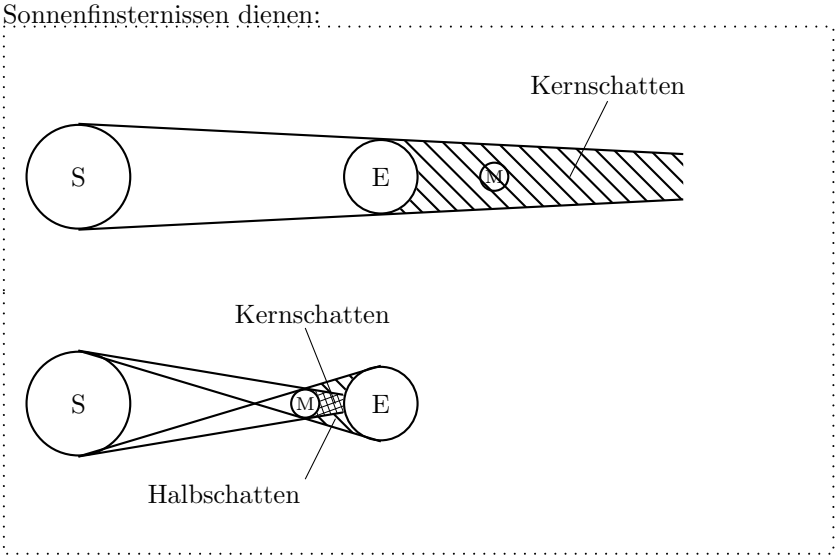


Abbildung 3.60. *Illustration: Mond-/Sonnenfinsternis*

In der Realität gibt es selten reine Kernschatten, da oft von anderen Körpern als dem verdeckenden reflektiertes Streulicht hinzukommt. Daher versucht man, nicht „harte“ Kernschatten mit einem scharfen Rand darzustellen, sondern „weiche“ Schatten mit einem unscharfen, verlaufenden Rand. Dazu arbeitet man nicht mit einem einzigen Licht, sondern mit einem Bündel von mehreren, eng beieinander liegenden.

3.6.2 Einfache Schattenalgorithmen

3.6.2.1 Vorgefertigte Schatten (pre-rendered shadows)

Bei diesem Verfahren werden die verschiedenen Schatten, die ein Objekt werfen kann, mit einem externen Programm vorberechnet und in der eigentlichen

Anwendung als Texturen an die entsprechenden Stellen aufgebracht. Das funktioniert dann, wenn das Objekt einfach ist – etwa nahezu kugel- oder quaderförmig – und nur über einfache Oberflächen bewegt wird. Dieses Verfahren wurde z. B. in den Spielen „Sonic Adventure“ 1998 und „Rayman“ 2000 verwendet. Besonders leicht ist es zu benutzen, wenn die Grafikhardware Multitexturen unterstützt, weil der Programmierer dann nicht selbst die „gewöhnliche“ und die Schattentextur zusammenmischen muss.
Das Verfahren ist zwar schnell und leicht zu implementieren, hat aber den Nachteil, dass die Schatten nicht wirklich dynamisch – also realistisch veränderlich – sind.

3.6.2.1.1 Ebene Schatten (planar shadows)

Hier macht man sich zunutze, dass Schattenwurf bezüglich einer Punktlichtquelle als Zentralprojektion aufgefasst werden kann (vgl. Aufg. 1 von Übungsblatt 7). Um den Schatten eines Objekts O auf einer Ebene E zu berechnen, zeichnet man unter Verwendung der bekannten Transformationsmatrizen eine schwarz gefärbte Projektion von O auf E.
Damit sind die Schatten dynamisch und trotzdem schnell zu berechnen.
Allerdings ist an zwei Stellen Vorsicht geboten:

1. Befindet sich die Lichtquelle L zwischen O und E, darf kein Schatten erzeugt werden, da man sonst einen *Antischatten* erhalten würde.

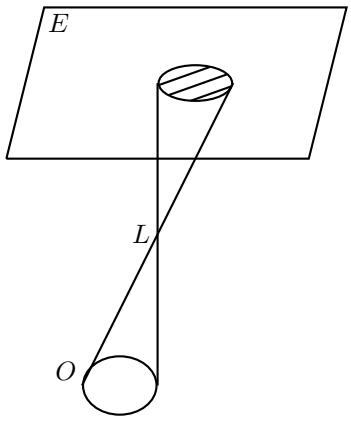


Abbildung 3.61. *Antischatten*

2. In realen Szenerien hat man es nicht mit (unendlichen) Ebenen zu tun, sondern mit endlich begrenzten Flächen, an deren Rändern der Schatten gegebenenfalls gekappt werden muss, damit er nicht unrealistischerweise in Nachbarflächen hineinragt:

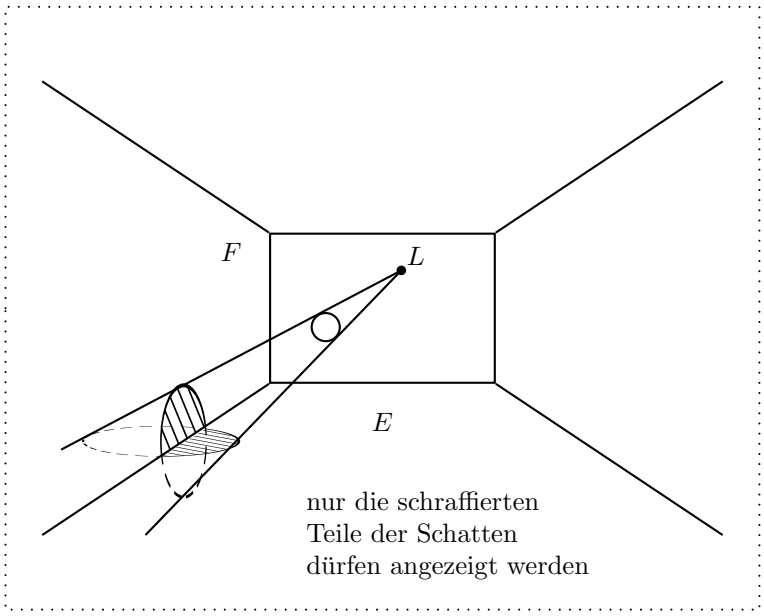


Abbildung 3.62. *gekappter Schatten*

Zur Lösung von 2. verwendet man den sogenannten *Schablonenpuffer (stencil buffer)*. Er erlaubt es, bestimmte Teile des Bildpuffers gegen Veränderung zu sperren, wirkt also tatsächlich wie eine Schablone beim Aufsprühen von Farben. Beim Zeichnen eines Schattens behandelt man dann alle beteiligten Flächen reihum wie folgt:

- Sperre alle Nachbarflächen der Fläche
- Zeichne nun den Schatten auf die zur Fläche gehörige Ebene.
- Hebe die Sperre wieder auf

Sind schließlich alle Schatten gezeichnet, muss natürlich das beleuchtete Objekt selbst auch noch dargestellt werden.
Das Verfahren kann dadurch beschleunigt werden, dass während des Schattenzeichnens Beleuchtung und Texturierung ausgeschaltet werden.
Das bisherige Verfahren erzeugt harte Kernschatten. Um Schlagschatten zu erzeugen, verwendet man als Schattenfarben nicht Schwarz, sondern dunkle Grautöne mit einer gewissen Transparenz. Wo sich mehrere Schlagschatten überlappen, entsteht dann ein entsprechender Kernschatten.
Weiche Schatten erhält man mit dieser Technik, wenn man ein Punktlicht durch mehrere eng beieinander liegende Lichtquellen ersetzt. Allerdings steigt damit natürlich der Rechenaufwand.

3.6.3 Komplexe Schattenalgorithmen

Die bisherigen Verfahren sind nur für größere ebene Flächen gut geeignet. Oft hat man aber komplex geformte Objekte, die gegenseitig auf einander Schatten werfen. Hierfür gibt es zwei wesentliche Verfahren:

1. Für jedes Objekt wird sein *Schattenraum (shadow volume)* bestimmt, der dann innen nicht beleuchtet wird.

2. Bei der *Schattenabbildung (shadow mapping)* wird für jedes von der Kamera aus sichtbare Pixel geprüft, ob es auch vom Licht aus sichtbar ist.

Während bei den vorigen Verfahren die voll beleuchtete Szenerie nach und nach teilweise abgedunkelt wurde, wird hier zuerst bestimmt, was nicht beleuchtet werden soll.

3.6.3.1 Das Schattenraumverfahren

Der Schattenraum eines Objekts O bezüglich einer Lichtquelle L besteht aus der Silhouette von O, von der aus die Strahlen von L ins Unendliche verlängert werden. Ist O ein Dreieck, ist der Schattenraum also ein ins Unendliche reichender Pyramidenstumpf.
Wie bestimmt man für allgemeinere Körper die Silhouette? Für Polygonnetze hat man folgendes Verfahren: Bestimme für alle Paare benachbarter Flächen wie in Abschnitt 2.3.3.1, ob sie beleuchtet sind. Ihre gemeinsame Kante gehört genau dann zur Silhouette, wenn die eine Fläche beleuchtet und die andere unbeleuchtet ist.
Damit ist aber auch klar, dass das Verfahren für komplexe Polygonnetze sehr rechenaufwendig ist. Oft genügt es jedoch, mit vereinfachten approximierenden Netzen zu rechnen.
Zum tatsächlichen Durchführen des Verfahrens kann man wieder den Schablonenpuffer verwenden. Dieser speichert für jedes Pixel üblicherweise ein Byte Information, die für eine ganze Reihe von Vergleichsoperationen genutzt werden kann, und zwar auch noch abhängig von der Information im Tiefenpuffer.
Für die Schattenpufferung verfolgt man im Schablonenpuffer mit, wie oft auf jedem Strahl zum Auge Grenzflächen des Schattenraumes überschritten werden.
Der genaue Ablauf ist wie folgt:

1. Stelle die gesamte Szene nur mit ambienter Hintergrundbeleuchtung dar, wobei der Tiefenpuffer eingeschaltet ist.

2. Setze alle Einträge im Schablonenpuffer SP auf 0.

3. Lasse jetzt nur noch Schreiben in SP zu und sperre alle anderen Puffer.

4. Stelle nun, von der Kamera aus gesehen, die Vorderseite des Schattenraums dar und erhöhe dabei die SP-Werte aller Pixel um 1, wenn diese Seite dort vor allen bisher gezeichneten Flächen liegt (Eintritt in den Schattenraum).

5. Stelle nun die Rückseite des Schattenraums dar und erniedrige dabei die SP-Werte aller Pixel um 1, wenn die Rückseite dort vor allen bisher gezeichneten liegt (Verlassen des Schattenraums).

6. Schalte nun die Hintergrundbeleuchtung aus und die eigentlichen Lichtquellen an, ebenso Mischen.

7. Erlaube nun wieder Schreiben in den Bildpuffer BP (der Tiefenpuffer bleibt gesperrt).

8. Konfiguriere SP so, dass nur auf BP-Pixel mit SP-Wert 0 geschrieben wird; das sind genau die, die nicht im Schatten liegen.

9. Stelle nun die gesamte Szene noch einmal dar.

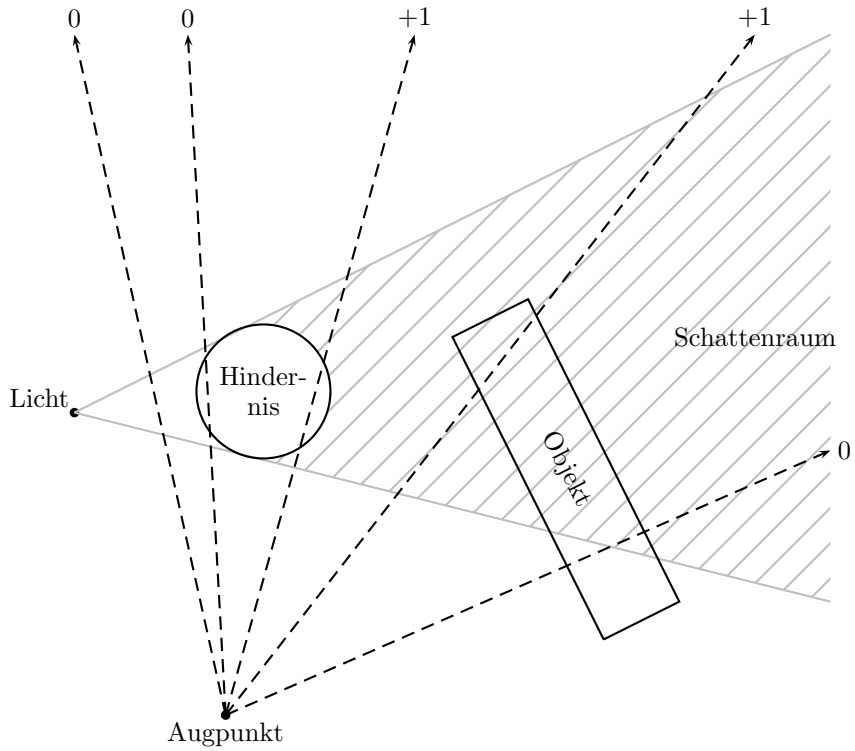


Abbildung 3.63.

Das Verfahren funktioniert auch, wenn mehrere Schattenräume auftreten.

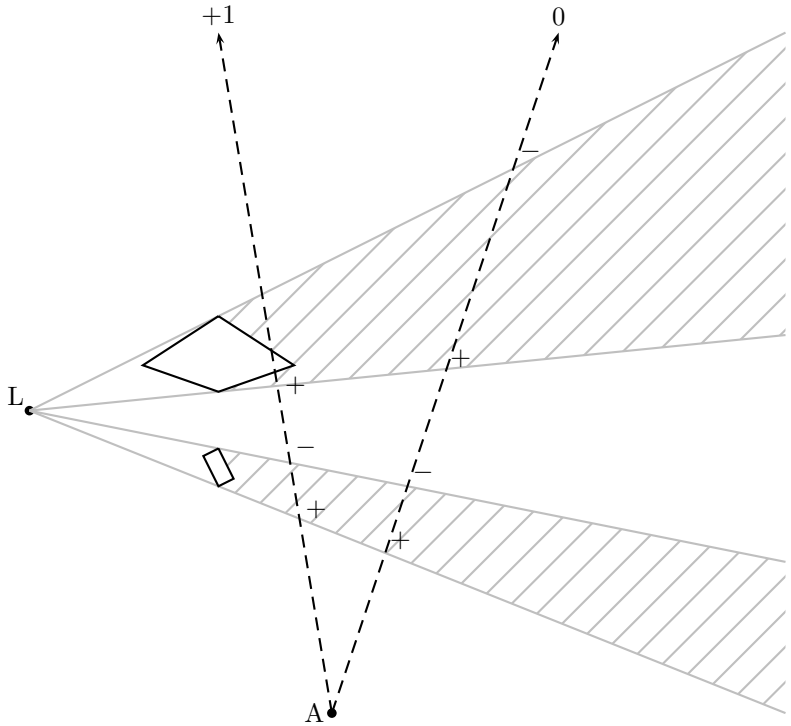


Abbildung 3.64.

Allerdings ist es nicht mehr korrekt, wenn der Augpunkt selbst im Schatten liegt:

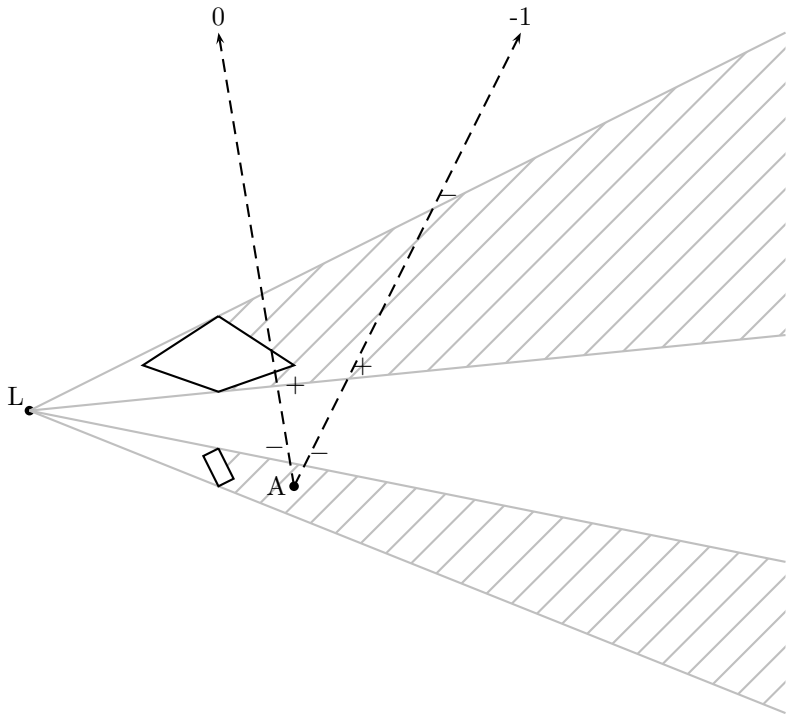


Abbildung 3.65.

In diesem Fall kann man ein duales Verfahren (*Carmack's Reverse*) einsetzen:

- Zeichne erst die Rückseite des Schattenraums und erhöhe SP dort, wo sie hinter einer anderen Fläche liegt.
- Zeichne dann die Vorderseite und erniedrige SP dort, wo sie hinter einer anderen Fläche liegt

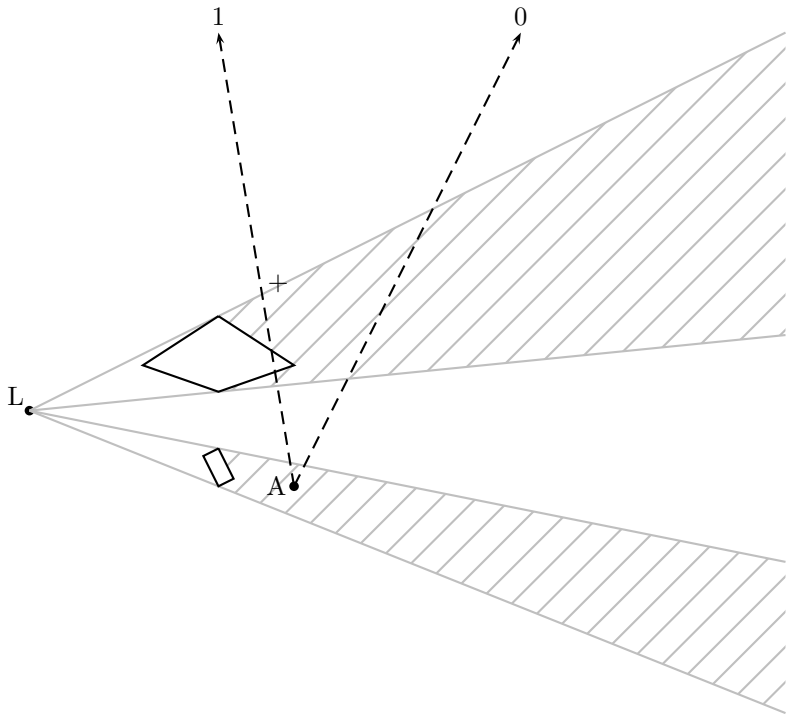


Abbildung 3.66.

Damit dieses Verfahren korrekt ist, muss der Schattenraum an beiden Enden mit Deckeln versehen werden (was wir bisher nicht angenommen hatten), die zur Rückseite zählen:

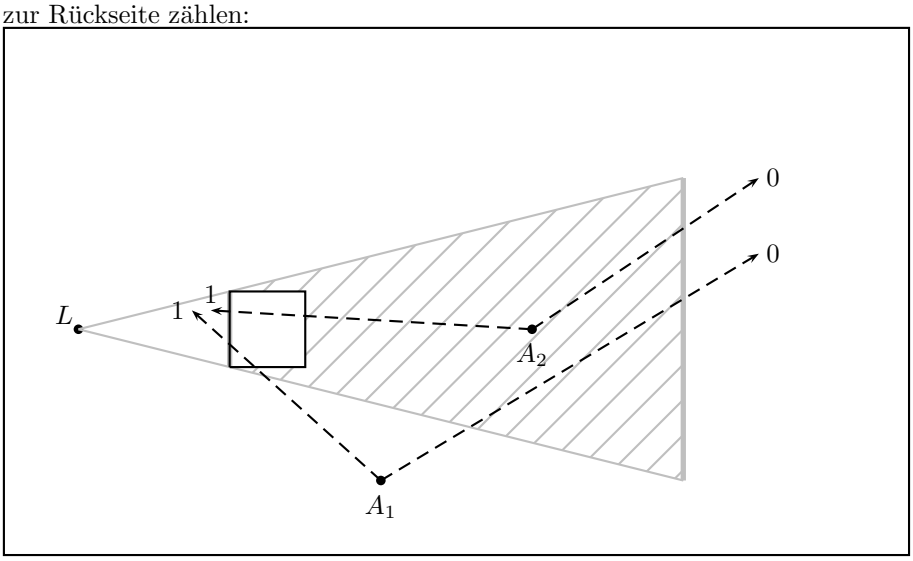


Abbildung 3.67.

Dabei liegt der rechte Deckel eigentlich im Unendlichen und erfordert eine Sonderbehandlung. Weitere Probleme ergeben sich, wenn die Szenerie auf das Frustum gekappt werden muss. Die Details würden hier aber zu weit führen.

3.6.3.2 Das Schattenabbildungsverfahren

Dieses Verfahren wurde bereits in 3.5.2.3 unter dem Namen „Schattenpufferung“ besprochen.

3.7 Anti-Alias-Verfahren

Wir hatten Treppeneffekte bereits beim Rastern von Bildern kennengelernt. Aber auch beim Wiederholen verkleinerter Texturen können sich durch Rasterungseffekte unschöne Interferenzerscheinungen ergeben. In diesem Abschnitt sollen einige Techniken zum Beheben solcher Fehler gezeigt werden. Als Vorbereitung brauchen wir aber etwas Signaltheorie.

3.7.1 Signale und Abtastung

Ein *Signal* ist eine Funktion von einem gewissen Koordinatenbereich (etwa der eindimensionalen Zeit oder dem zweidimensionalen Texturraum) in einen gewissen Wertebereich. Beide Bereiche können dabei diskret oder kontinuierlich sein.
Aus kontinuierlichen Signalen kann man durch *Abtastung (sampling)* diskrete gewinnen. Wir beschreiben das für den Fall eines 1D-Koordinatenbereichs, der Einfachheit halber $\mathbb{R}$, und $\mathbb{R}$ als Wertebereich. Wichtiges Hilfsmittel ist dabei die in 3.3.2.1 erwähnte Sprungfunktion, auch *Diracsche Deltafunktion* genannt:

$$\delta(t) = \begin{cases} 1 & t = 0 \\ 0 & \text{sonst} \end{cases}$$

Ist nun $s : \mathbb{R} \to \mathbb{R}$ ein kontinuierliches Signal, so erhält man daraus das „diskretisierte“ Signal $s_d : \mathbb{R} \to \mathbb{R}$ bei Verwendung von Abtastpunkten mit gleichmäßigen Abstand $T > 0$, der *Abtastperiode*, als

$$s_d(t) = s(t) \cdot k(t)$$

mit $k(t) = \sum_{n \in \mathbb{Z}} \delta(t - n \cdot T)$. Durch Multiplikation mit der „Kammfunktion“ k

Abbildung 3.68. *Die Kammfunktion*

hat s_d höchstens an den ganzzahligen Vielfachen von T Werte $\neq 0$. Ein echt diskretes Signal $s_d : \mathbb{N} \to \mathbb{R}$ erhält man als $s_d(n) = s(n \cdot T)$.
Das *Abtasttheorem* liefert Bedingungen, unter denen kontinuierliche Signale ohne Fehler in diskrete und zurück überführt werden können. Zu seiner Formulierung brauchen wir einige Begriffe aus der *Fourier-Analyse* reeller Funktionen. Jedes Signal $s : \mathbb{R} \to \mathbb{R}$ lässt sich darstellen als

$$s(t) = \int_{-\infty}^{\infty} S(f) \cdot [\cos(2\pi f t) + i \cdot \sin(2\pi f t)] df$$

d. h. als Überlagerung reiner Sinus- und Cosinusschwingungen, wobei $S(f)$ die (komplexwertige) „Stärke“ der Schwingungen mit Frequenz f ist.
Umgekehrt ergibt sich $S(f)$ durch die *Fourier-Transformation*

$$S(f) = \int_{-\infty}^{\infty} s(t) \cdot [\cos(2\pi f t) - i \cdot \sin(2\pi f t)] dt$$

Wir schreiben auch $S = FT(s)$ und $s = IFT(S)$; die vorige Gleichung bezeichnet man als *inverse Fourier-Transformation*. Der Betrag $|S(f)|$ ist die Amplitude der Schwingung mit Frequenz f, der Drehwinkel von $S(f)$ gibt die Phasenverschiebung an. Meist interessiert man sich aber nur für die Amplitude $|S(f)|$. Ein Signal s heißt *bandbegrenzt*, wenn $|S(f)|$ nur in einem beschränkten Intervall von 0 verschieden ist. Das Supremum dieses Intervalls heißt dann die *Grenzfrequenz* f_g von s.

Abtasttheorem (Shannon/Nyquist):
Gilt bei diskreter Abtastung mit Abtastperiode T für ein bandbegrenztes Signal

$$f_a = \frac{1}{T} > 2 \cdot f_g,$$

so kann das kontinuierliche Signal aus der diskreten Abtastung fehlerfrei zurückgewonnen werden. Der Wert $2 \cdot f_g$ heißt *Nyquist-Frequenz*.
Um zu verstehen, warum bei Verletzung dieser Bedingung Abtastfehler entstehen, skizzieren wir kurz den Rekonstruktionsprozess.
Ein *Filter* schwächt bestimmte unerwünschte Frequenzen im Frequenzspektrum eines Signals ab. Das gefilterte Signal entsteht, wie bei der diskreten Abtastung, durch Multiplikation des ursprünglichen Signals mit der Filterfunktion, von der man üblicherweise annimmt, dass ihr *Träger*, d.h. die Menge der Argumente mit Wert $\neq 0$, beschränkt ist.
Filter können koordinaten- oder frequenzbezogen definiert werden. Soll z. B. nur ein bestimmtes Frequenzband durchgelassen werden, wählt man eine Rechtecksfunktion auf dem Frequenzraum. Ihre invers Fouriertransformierte ist die Funktion

$$\mathrm{sinc}(t) = \frac{\sin \pi t}{\pi t},$$

bei der die Werte mit wachsendem $|t|$ rasch gegen 0 gehen.
Bei einem *Tiefpassfilter* liegt das zugehörige Rechteck symmetrisch zum Koordinatenursprung (was die mathematische Behandlung vereinfacht, denn negative Frequenzen sind ja eigentlich physikalisch nicht sinnvoll):

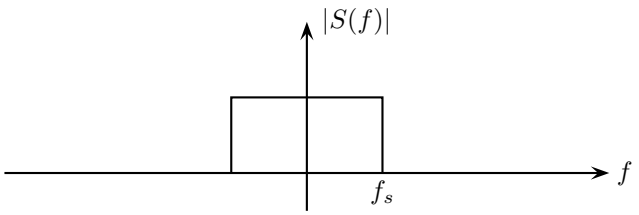


Abbildung 3.69. *Tiefpassfilter*

Es werden also alle Frequenzen unterhalb der Frequenz f_s durchgelassen, die anderen abgeschnitten.
Das gefilterte Signal erhält man, wenn man die Fouriertransformierte des Signals mit der Filterfunktion multipliziert (analog zur Dirac-Abtastung) und dann wieder rücktransformiert.

Es zeigt sich, dass man auf den Signalen eine Operation definieren kann, die den selben Effekt hat wie Transformation-Multiplikation-Rücktransformation. Das ist die so genannte *Faltung*

$$(f * g)(t) = \int_{-\infty}^{\infty} f(x)g(t-x)dx$$

Sie berechnet einen gewichteten Mittelwert der f-Werte. Die Gewichtsfunktion g, die an der y-Achse gespiegelt und dann jeweils an die Stelle t verschoben wird, heißt auch der *Kern* der Faltung (oder des Filters, wenn g die Filterfunktion beschreibt). Die Faltung ist kommutativ. Außerdem gilt, wenn $F = FT(f)$ und $G = FT(g)$,

$$FT(f * g) = F \cdot G \qquad IFT(F * G) = f \cdot g$$

Wegen dieser Zusammenhänge kann man nun Operationen in der Koordinaten- oder Frequenzsicht ausführen, je nachdem, was bequemer ist.
Zwei weitere Eigenschaften sind die folgenden:

- Die Fouriertransformierte einer Kammfunktion mit Periode T ist eine Kammfunktion mit Periode $\frac{1}{T}$
- Faltung eines Signals mit einer Kammfunktion überlagert das Signal periodisch auf sich selbst.

Mit diesen Hilfsmitteln können wir nun das Verfahren skizzieren, das das Abtasttheorem konstruktiv beweist.

- Diskretes Abtasten entspricht der Multiplikation des Ausgangssignals s mit einer Kammfunktion der Abtastperiode $T = \frac{1}{f_a}$.
- In der Frequenzsicht wird daraus die Faltung des Frequenzspektrums $S = FT(s)$ mit einer Kammfunktion der Periode $\frac{1}{T} = f_a$, also eine unendliche Wiederholung des Frequenzspektrums.
- Ist nun s bandbegrenzt mit der Grenzfrequenz f_g, so hat der Träger von S maximal die Breite $2 \cdot f_g$.
- Wählt man also $f_a > 2 \cdot f_g$, so liegen die durch die Faltung entstehenden „Kopien“ von S von einander getrennt.
- Daher kann man S und damit s zurückgewinnen, indem man die Faltung (d. h. das diskret abgetastete Signal) durch einen idealen Tiefpassfilter mit Schwellenfrequenz $f_s = f_g$ schickt.

Ist dagegen die Bedingung des Abtasttheorems verletzt, so überlagern sich die Kopien von S und es kommt zu unauflösbaren Mehrdeutigkeiten, weil im gefilterten Signal gewisse Frequenzen einmal im Original und gleichzeitig als *Alias* aus einer Kopie auftreten. Dabei kann sich z. B. eine hohe Frequenz gleichzeitig auch noch als niedrige „verkleiden“ und somit das gefilterte Signal verfälschen.

Beispiel

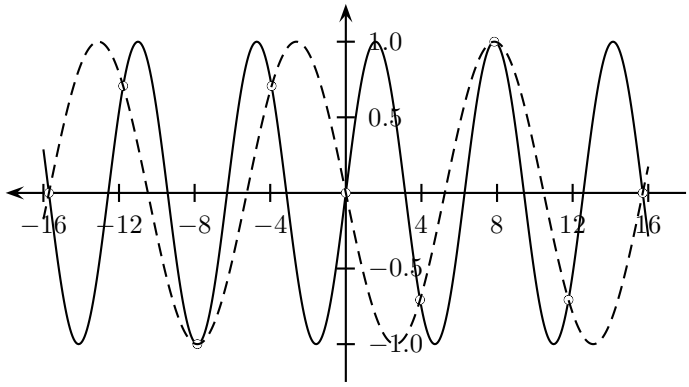


Abbildung 3.70. $s(x) = \sin(x)$, *Abtastpunkte* $\{\frac{5n\pi}{4} \mid n \in \mathbb{N}\}$

Die Abtastfrequenz ist kleiner als die doppelte Grenzfrequenz (nicht jeder Ausschnitt aus der Kurve der vollen Periodenlänge enthält zwei Abtastpunkte). Daher ist auch die gestrichelte Kurve eine mögliche Rekonstruktion ($f(x) = -\sin\frac{3x}{5}$). □

Bei zu geringer Abtastfrequenz spricht man von *Unterabtastung*; die durch die Verfälschungen hinzukommenden uneigentlichen Objekte heißen *Artefakte*.

Eine Grundvoraussetzung für das Abtasttheorem ist die Bandbegrenztheit der Signale. Leider ist das in der Praxis kaum gegeben. Besonders sehr scharfe Kanten/Übergänge führen zu einem unbeschränkten Frequenzspektrum. Als Modell kann die Sprungfunktion δ dienen: Ihre Fouriertransformierte ist eine konstante Funktion! Ebenso hat die Fouriertransformierte der Rechtecksfunktion, die wieder die Funktion *sinc* ist, einen unbeschränkten Träger. Damit das Abtasttheorem anwendbar wird, kann man solche Signale zuerst durch einen Tiefpaßfilter schicken, wodurch aber natürlich die Schärfe der Kanten verwischt wird.

Hat man diese Möglichkeit nicht, etwa weil eine Schachbretttextur direkt, d. h. ungefiltert, aufgebracht werden soll, kommt es dann zu u. U. sehr stark ausgeprägten Artefakten. Die Abtastfrequenz bei Bildern entspricht der Bildschirmauflösung, nämlich ℓ Zeilen pro cm. Die zugehörige Grenzfrequenz ist $f_g = \frac{\ell}{2}$. Artefakte treten also auf, wenn die charakteristischen Bildänderungen in Längen kleiner als $\frac{1}{f_g} = \frac{2}{\ell}$ = doppelter Abstand benachbarter Pixelmitten erfolgen.

Hat man also sprunghafte Änderungen in den Bildern, kann das Erhöhen der Auflösung die Artefakte zwar abmildern, aber nicht beseitigen. Dies gelingt nur, wenn die Bildschirmauflösung feiner gemacht werden kann, als die des menschlichen Auges, d. h. feiner als 0.5' Bogenmaß oder ca. 240Pixel/cm, wenn der Betrachter 30 - 60cm vom Schirm entfernt ist.

3.7.2 Entfernen von Artefakten durch Verwackeln (Jittering)

Bei dieser Technik wird das Bild wiederholt leicht versetzt berechnet; die verschiedenen Versionen werden im Akkumulationspuffer (*accumulation buffer*) *AP* gemischt. die störenden Artefakte heben sich dabei gewissermaßen gegenseitig auf.
Bei perspektivischer Projektion ist das Verfahren etwas aufwendiger als bei Parallelprojektion, weil für jede Bildsicht ein leicht versetztes Frustum definiert werden muss. Mit der selben Technik lassen sich dann aber auch Tiefenunschärfen miterzeugen, die für manche Anwendungen realistischer sind.
Genaues zum Arbeiten mit dem *AP* findet sich im Red Book und den Beispielen.

Index