Inhaltsverzeichnis

Kapitel 0

Einführung

0.1 Was ist Computergrafik?

Software, die einen Computer dazu bringt, eine grafische Ausgabe (oder kurz gesagt: Bilder) zu produzieren. Bilder können sein: Fotos, Schaltpläne, Veranschaulichung von Denkansätzen (mathematische Topologie).

0.2 Anwendungsgebiete

– Video-Spiele (auch „ernste“ Spiele)	3D,4D
– Simulation (z.B. bei der Pilotenausbildung)	4D
– Zeichnungen für wissenschaftliche Arbeiten (mit xfig, ipe)	2D
– Schaltpläne	2D
– Fotorealistische Bilder (kaum von echten Fotos zu unterscheiden)	3D
– Kartographie, GIS (Geographic Information Systems)	2D,(3D)
– CAD (Computer Aided Design, Gebäudebaupläne etc.)	3D,(2D)
– Medizin (CT, MRT, US, etc.)	3D,2D

2D-Computergrafik: in der 2D-Computergrafik werden Bilder durch einfache Objekte wie Linien, Kreise oder Polygone auf einer 2-dimensionalen Ebene beschrieben. Die Ausgabe einer Computergrafik ist im Allgemeinen 2-dimensional. Daher müssen 3D- oder 4D-Grafiken in 2D-Grafiken umgewandelt werden.

3D-Computergrafik: in der 3D-Computergrafik werden Bilder (Kugeln, Würfel, etc.) nicht in einer Ebene sondern im 3-dimensionalen Raum beschrieben. Als Ausgabe möglich sind stereoskopische 3D-Bilder, meistens sind es jedoch 2D-Grafiken. Der Entwurf und die Beschreibung einer 3D-Szene (Modellierung) sowie das Erstellen eines Bildes oder Filmes (Rendering) werden weiter unten beschrieben.

4D-Computergrafik: als vierte Dimension kommt die Zeit hinzu, also bewegte 3D-Bilder.

0.3 Beispiele

Beispiel 1: Mehrere Würfel auf reflektierender Oberfläche, mehrere Lichtquellen, Objekte haben glänzende oder matte Flächen (Abb. 1).

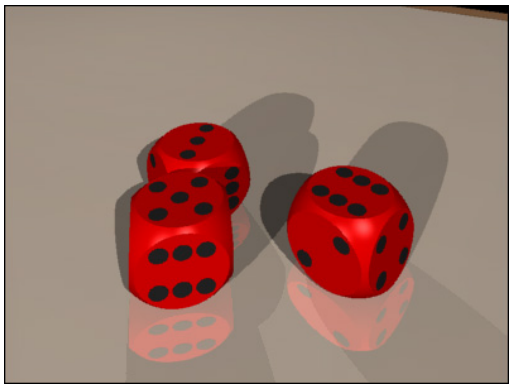


Abbildung 1: Würfel

Beispiel 2: Raum mit mehreren konvexen Spiegeln, die sich teilweise ineinander spiegeln, Objekte, eine Lichtquelle (Abb. 2).

Beispiel 3: Mehrere Körper vor einem Spiegel, u.a. eine gläserne Kugel, die Brechung erzeugt (Abb. 3).

Beispiel 4: Grünlicher Glaskegel auf Schachbrettmuster erzeugt Brechungen (Abb.4).

Abbildung 2: Regal im Raum mit Spiegeln

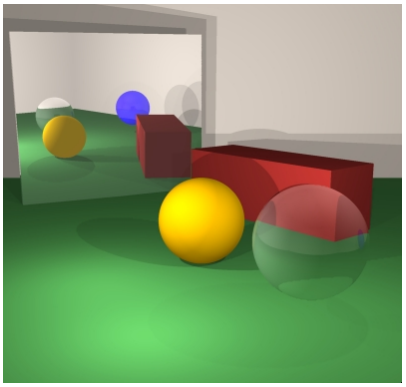


Abbildung 3: Geometrische Körper vor einem Spiegel

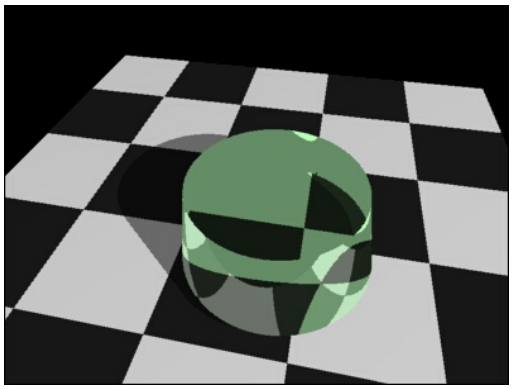


Abbildung 4: Glaskegel auf Schachbrett

Wie programmiert man solche Bilder? Einfach ist es natürlich jedes einzelne Pixel in eine Matrix abzulegen. Ziel dieser Vorlesung ist es jedoch auch, solche

3-dimensionalen Szenen im Computer abzuspeichern und sich daraus ein 2D-Bild ausgeben zu lassen. Dadurch wären zum Beispiel auch Veränderungen der Lichtquelle möglich.

0.4 Grobes Schema der 3D-Computergrafik

0.4.1 Modellierung, Eingabe numerischer Daten

Darstellung einer 3-dimensionalen Szene im Computer. Bekannt sind Informationen wie die geometrischen Daten (Form, Lage im Raum von Objekten), die Farbe und Transparenz der Objekte. Weiter weiß man, ob die Objekte eine glänzende oder matte Oberfläche haben. Bekannt ist auch die Textur, das ist das Muster der Oberflächen, z.B. ein Tapetenmuster, oder eine im Rechner dargestellte Wiese, deren Grashalme natürlich nicht einzeln dargestellt werden. Was muß man tun, um von der 3D-Szene ein 2D-Bild zu bekommen?

0.4.2 Rendering:

Es muß eine 2D-Projektion der geometrischen Daten vorgenommen werden. Das geht über die Anpassung der Daten an das Ausgabemedium (Bildschirm, Papier, etc.) und ggf. muß das Bild auf die Größe des Ausgabemediums zurecht geschnitten werden. Dazu gehört auch die Rasterisierung (Projektion der Farbe, Transparenz etc.) in „Pixel“ und das Entfernen verdeckter Kanten und Flächen. Benötigt werden hier vor allem viel Mathematik (lineare Algebra, analytische Geometrie), Kenntnisse über Brechung, Reflexion und „Raytracing“.

Definition 0.4.1 (Raytracing)**.** Auf deutsch Strahlenverfolgung, ist ein Verfahren zur Verdeckungsberechnung. Das Prinzip ist folgendes: von einem bestimmten Punkt (Augpunkt) wird ein Strahl durch ein Pixel der Bildebene gesendet und verfolgt, bis er auf ein anderes Objekt fällt. Je nach den Einflüssen der Brechung, Glanz, Lichtquellen wird das Pixel eingefärbt.

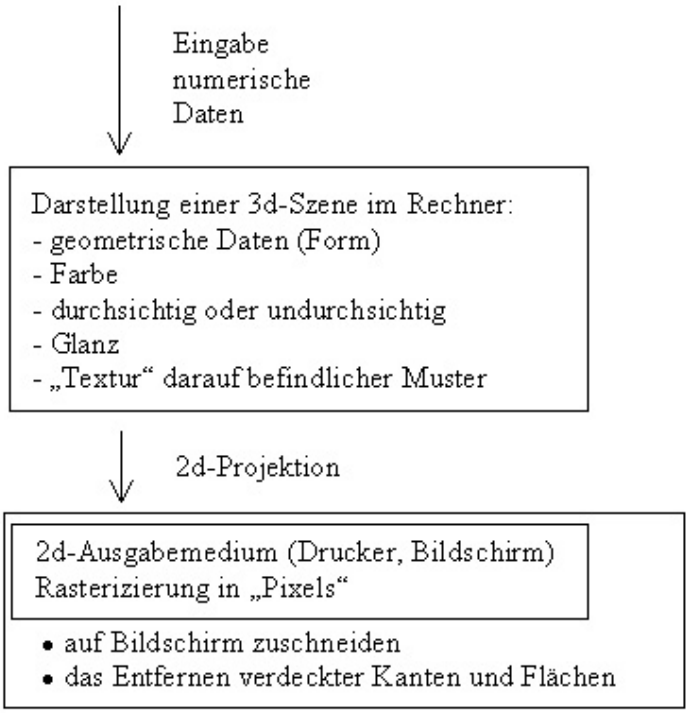


Abbildung 5: Modellierung und Projektion

Kapitel 1

Darstellung geometrischer Objekte

1.1 Punkte und Geraden im $\mathbb{R}^2$

1.1.1 Darstellung durch kartesische Koordinaten:

Punktdarstellung: Ein Punkt p wird dargestellt durch ein Tupel (a, b) mit $a, b \in \mathbb{R}$, d.h. $(a, b) \in \mathbb{R}^2$.

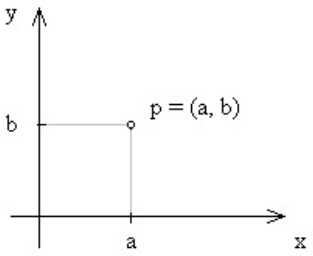


Abbildung 1.1: Punkt p im kartesischen Koordinatensystem

Bemerkung. Ein Punkt p ist unterschiedlich von einem Vektor $\boldsymbol{v}$. Der Vektor $\boldsymbol{v}$ reicht von einem Punkt zu einem anderen. Wenn der Vektor vom Nullpunkt (oder Ursprung) aus zu einem Punkt p geht, so nennt man ihn Ortsvektor $\boldsymbol{p}$ des Punktes p und schreibt dafür $\boldsymbol{p} = \begin{pmatrix} a \\ b \end{pmatrix}$.

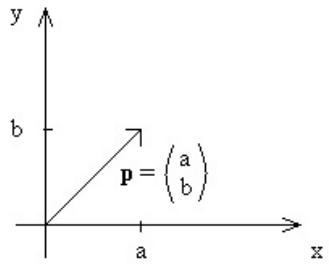


Abbildung 1.2: Ortsvektor $\boldsymbol{p}$ von Punkt p

Geradendarstellung: Ein Gerade wird durch ein Paar $(a, b) \in \mathbb{R}^2$ dargestellt, wobei gemeint ist:

$$g = \{(x, y) | y = ax + b\} \ .$$

Problem: hiermit sind keine „senkrechten“ Geraden darstellbar, a müsste in diesem Fall unendlich sein. Daher schreiben wir:

$$(a, b, c) \text{ mit } g = \{(x, y) | ax + by + c = 0\} \ .$$

Nachteil: Mehrdeutigkeit, $g_1 = (4, 2, 2)$ und $g_2 = (2, 1, 1)$ bezeichnen die

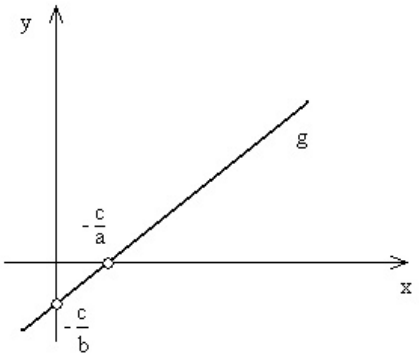


Abbildung 1.3: Gerade im kartesischen Koordinatensystem

gleiche Gerade, es gibt unendlich viele Darstellungen einer Gerade. Es läßt sich leicht herausstellen ob zwei Tripel die gleiche Gerade darstellen. Zwei Geraden $g_a = (x, y, z)$ und $g_b = (p, q, r)$ sind gleich, wenn ein beliebiges $\lambda \in \mathbb{R}$ existiert, so daß $\lambda(x, y, z) = (p, q, r)$ gilt (also $\lambda x = p, \lambda y = q, \lambda z = r$).

1.1.2 Darstellung durch homogene Koordinaten:

Homogene Koordinaten sind für viele Operationen in der Computergrafik günstiger. Bei Gebrauch von kartesischen Koordinaten benötigt man Fallunterscheidungen, ob zwei Geraden parallel sind oder nicht, das verlängert den Code. Außerdem kann man den gleichen Code zur Bestimmung des Schnittpunktes zweier Geraden sowie zur Berechnung einer Geraden durch zwei Punkte benutzen.

Punktdarstellung

Ein Punkt p mit den kartesischen Koordinaten (a, b) wird repräsentiert durch ein Tripel $(a, b, 1)$ oder durch jedes Tripel $(\lambda a, \lambda b, \lambda), \lambda \neq 0, \lambda \in \mathbb{R}$. Teilt man die homogenen Koordinaten λa und λb durch die dritte Koordinate λ, dann erhält man die kartesischen Koordinaten (a, b). Diese Punkte bilden die so genannte „projektive Ebene“ $\mathbb{P}^2$. Zusätzlich zu $\mathbb{R}^2$ hat man noch Punkte der Form $(a, b, 0)$, die man „unendlich ferne“ Punkte nennt. Unendlich ferne Punkte gibt es nicht im kartesischen Koordinatensystem, da man seine homogene nicht in kartesische Koordinaten umwandeln kann.

Geradendarstellung

Wie vorhin beschrieben bezeichnet ein Tripel (a, b, c) (wobei nicht alle gleich Null sein dürfen) die Gerade

$$g = \{(x, y) | ax + by + c = 0\} \ .$$

Allerdings beschreibt das Tripel $(\lambda a, \lambda b, \lambda c)$ für jedes $\lambda \neq 0, \lambda \in \mathbb{R}$ die gleiche Gerade. Warum benutzt man diese Methode, obwohl man 50% mehr Daten (Tripel statt Tupel) speichern muß? Weil geometrische Relationen dadurch leichter beschreibbar sind, der Punkt p zum Beispiel mit den homogenen Koordinaten (a, b, c) liegt auf der Geraden g mit den homogenen Koordinaten (r, s, t). Wir haben Vektor $\boldsymbol{p} = \begin{pmatrix} a \\ b \\ c \end{pmatrix}$ und $\boldsymbol{g} = \begin{pmatrix} r \\ s \\ t \end{pmatrix}$. Das Skalarprodukt $\boldsymbol{pg} = 0$, d.h. $ar + bs + ct = 0$.

Bemerkung. Auf der Geraden $g = (r, s, t)$ liegt auch der unendlich ferne Punkt $p_1 = (-s, r, 0)$. Der unendlich ferne Punkt $p_2 = (s, -r, 0)$ ist der selbe Punkt wie p_1, d.h. beide „Enden“ von g liegen auf dem selben Punkt. Die Gerade $g' = (s, r, t'), t' \neq 0, t' \in \mathbb{R}$ ist parallel zu g. Zwei „parallele“ Geraden

schneiden sich immer in einem unendlich fernen Punkt (im Gegensatz zu der Euklidischen Geometrie).

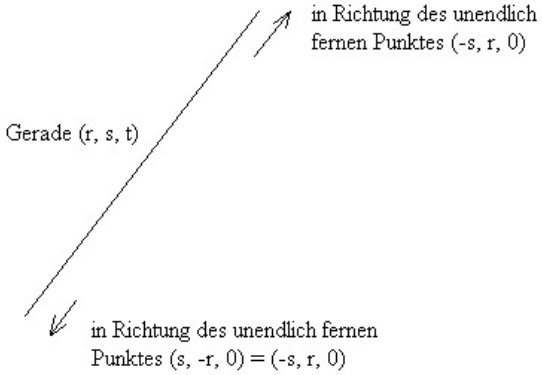


Abbildung 1.4: Gerade mit unendlich fernen Punkten

Was bedeutet das für die/den ProgrammiererIn? Wie schon oben angedeutet kann man hier ohne Fallunterscheidung den Schnittpunkt zweier Geraden bestimmen. Sind zwei Geraden parallel, haben sie einen unendlich fernen Punkt als Schnittstelle. Diese unendlich fernen Punkte sind auf der Zahlengerade nicht sichtbar, in der projektiven Ebene schon (siehe Punkt p in Abbildung 1.5).

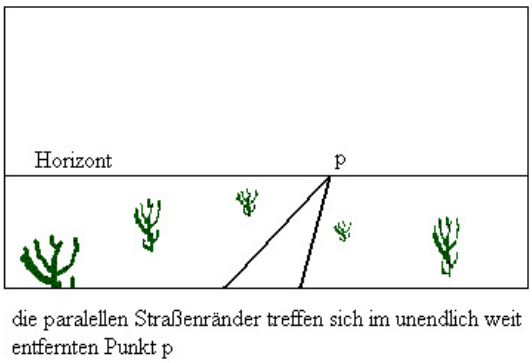


Abbildung 1.5: Parallele Geraden schneiden sich im Horizont

In der Abbildung 1.5 besteht der Horizont aus lauter unendlich fernen Punkten. Alle unendlich fernen Punkte liegen auf der Geraden mit den homogenen Koordinaten $(0, 0, 1)$, auch genannt die „unendlich ferne“ Gerade.

Bemerkung. In der Euklidischen Geometrie nennt man Geraden in $\mathbb{R}^2$, die keinen Schnittpunkt haben, Parallelen. In der projektiven Geometrie haben alle Geraden g_1 und g_2, $g_1 \neq g_2$, also auch zwei parallele Geraden, genau einen Schnittpunkt. Haben zwei nicht parallele Geraden einen „normalen“ Schnittpunkt, so haben zwei parallele Geraden im homogenen Koordinatensystem einen gemeinsamen unendlich fernen Punkt als Schnittpunkt. Alle unendlich fernen Punkte einer Ebene bilden zusammen die unendlich ferne Gerade. Normale Geraden und die unendlich ferne Gerade haben einen Schnittpunkt auf der unendlich fernen Geraden. Gäbe es Parallelität im euklidischen Sinne, wären alle Geraden parallel zur unendlich fernen Geraden. Hätten sie nämlich wie in der Euklidischen Geometrie keinen Schnittpunkt, dann hätten sie auch keinen Schnittpunkt mit der unendlich fernen Geraden, da ja genau dieser eine Punkt auf der unendlich fernen Geraden der Schnittpunkt zweier parallelen Geraden ist. Diese Schnittpunkte würden also wegfallen, somit wären auch die normalen Geraden parallel zu der unendlich fernen Geraden.

Schnittpunkt

Der Schnittpunkt p von zwei Geraden $g_1 : (a, b, c)$ und $g_2 : (d, e, f)$ ist $p = (x, y, z)$ mit

$$ax + by + cz = 0$$

und

$$dx + ey + fz = 0$$

Betrachtung der Matrix $m : \begin{pmatrix} a & b & c \\ d & e & f \end{pmatrix}$. Ist m vom Rang 1 (wenn linear abhängig), dann ist $g_1 = g_2$. Ist m vom Rang 2 (wenn linear unabhängig), dann ist ein Unterraum der Dimension 1 die Lösung, das entspricht einem Punkt im homogenen Koordinatensystem. Je zwei von einander verschiedene Geraden schneiden sich immer in einem Punkt (im Gegensatz zur Euklidischen Geometrie, s.o.).

Gerade durch zwei Punkte

Durch die beiden Punkte $p_1 : (a, b, c)$ und $p_2 : (d, e, f)$ läuft die Gerade $g : (x, y, z)$:

$$ax + by + cz = 0$$

und

$$dx + ey + fz = 0$$

Weiteres siehe oben.

1.1.3 Strecken

Eine Strecke ist ein Stück einer Geraden g, das zwischen zwei Punkten p und q liegt. Die Strecke wird durch diese zwei Punkte (ihre Endpunkte) repräsentiert.

$$\overline{pq} = s = \{\lambda \boldsymbol{p} + (1 - \lambda)\boldsymbol{q} | \lambda \in [0, 1]\}$$

1.1.4 Polygonzüge

Ein Polygonzug (auch *Kantenzug* genannt), besteht aus Kanten (Strecken) und wird durch die Punkte seiner Kanten (in deren Reihenfolge) repräsentiert.

$$p_1, p_2, ..., p_n$$

Die Kanten des Polygonzugs sind $\overline{p_1p_2}, \overline{p_2p_3}, ..., \overline{p_{n-1}p_n}$, wobei n die Länge des Polygonzugs beschreibt.

Ein Polygonzug heißt **einfach**, wenn es keine zwei sich schneidenden Kanten gibt. Die Ausnahme bilden die jeweils aufeinander folgenden Kanten $\overline{p_ip_{i+1}}$ und $\overline{p_{i+1}p_{i+2}}$, die sich im Punkt p_{i+1} schneiden.
Polygonzüge mit sich schneidenden Kanten heißen **nicht einfach**.

Kurven in der Ebene werden i.d.R. in der Computergraphik durch Polygonzüge approximiert.

1.1.5 Polygone

Polygone sind geschlossene Polygonzüge, solche also, bei denen der letzte Punkt auch der erste ist: $p_n = p_1$.

Ein Polygon heißt **einfach**, wenn sich (analog zu den Polygonzügen) keine zwei Kanten schneiden.
Polygone mit sich schneidenden Kanten heißen **nicht einfach**.

Einfache Polygone teilen die Ebene in zwei Teile: das **Innere** und das **Äußere**. Das Innere ist begrenzt durch die Kanten des Polygons, das Äußere ist unbegrenzt.

1.1.6 Parametrisierte Darstellung von Kurven

Kurven in der Ebene werden repräsentiert durch eine stetige Abbildung

$$f : I \to \mathbb{R}^2$$

$$f(t) = (x(t), y(t))$$

wobei

$$x, y : I \to \mathbb{R}$$

stetige Funktionen sind.

Beispiel 1: (siehe Abbildung 1.6)

$f : [-1, 1] \to \mathbb{R}^2$, wobei $x(t) = t^2, y(t) = t$
kurz: t^2, t mit $t \in [-1, 1]$

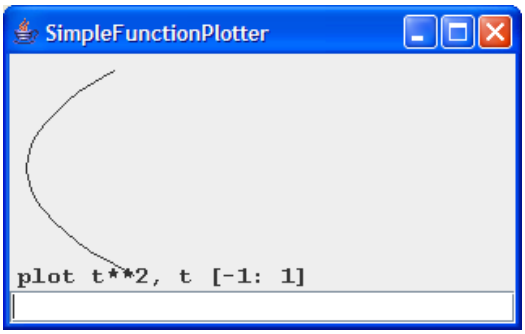


Abbildung 1.6: Beispiel 1

Beispiel 2 : $t, \sqrt{1-t^2}$ mit $t \in [-1, 1]$ (Halbkreis nach unten)

Beispiel 3 : $sin(t), cos(t)$ mit $t \in [0, 2\pi]$ (Kreis)

Beispiel 4 : $t * cos(30t), t * sin(30t)$ mit $t \in [0, 1]$ (Spiralform)

Beispiel 5 : $t * cos(\frac{1}{t}), t * sin(\frac{1}{t})$ mit $t \in [-1, 0]$

Beispiel 6 : $t - 2sin(t), 1 - 2cos(t)$ mit $t \in [0, 30]$ (*Zykloide*)

Beispiel 7 : t^3, t^2 mit $t \in [-1, 1]$ (siehe Abbildung 1.7)

Differenzierbarkeit

$f(t) = (x(t), y(t))$

Wenn die Ableitungen nach t existieren, dann ist

$(\frac{dx(t)}{dt}, \frac{dy(t)}{dt}) = f'(t)$

der Vektor tangential zu der Kurve an der Stelle $f(t)$ - falls $f'(t) \neq \boldsymbol{0}$.

Beispiel:
$f(t) = (t^2, t) \Rightarrow f'(t) = (2t, 1)$

Analog werden f'', f''', ... definiert.

Def.
Eine Kurve ist *i-fach stetig differenzierbar* genau dann, wenn $f^{(i)}(t)$ stetig ist und $f^{(i)}(t) = \boldsymbol{0}$ nirgends eintritt.

(C^i : Menge aller i-fach stetig differenzierbaren Kurven)

Beispielsweise ist t^3, t^2 nicht i-fach stetig differenzierbar, da bei $f'(t) = 3t^2, 2t$ gelten würde: $f'(0) = \boldsymbol{0}$.

1.1.7 Implizite Darstellung von Kurven

Die implizite Darstellung von Kurven wird in der Computergraphik nicht so häufig benutzt und in diesem Kurs nur angerissen.

$F : \mathbb{R}^2 \to \mathbb{R}$ Funktion (z.B. Polynom $F(x, y) = x^2 + y^2 - 1$)

Die Gleichung $F(x, y) = 0$ definiert - unter Umständen - eine zusammenhängende Kurve.

Zum Beispiel liefert $F(x, y) = x^2 + y^2 - 1 \Leftrightarrow \{(x, y) | x^2 + y^2 - 1 = 0\}$ einen Kreis mit dem Radius $r = 1$, wohingegen $F(x, y) = xy - 1$ keine zusammenhängende Kurve liefert, da xy mit $x = 0$ oder $y = 0$ in jedem Fall 0 liefert und somit $0 - 1$ nicht 0 ergeben kann, an den Achsen also ein „Bruch“ entsteht.

1.2 Punkte, Geraden, Ebenen im 3-dimensionalen Raum

1.2.1 Punkte

Punkte im 3-dimensionalen Raum sind darstellbar durch kartesische Koordinaten für die Achsen x, y, z:
$p = (a, b, c)$

Mittels *homogener Koordinaten* lassen sich Punkte in 3D darstellen (analog zur Darstellung in 2D):

$$p = (a, b, c, d)$$

, was $(\frac{a}{d}, \frac{b}{d}, \frac{c}{d})|d \neq 0$ entspricht.

Ist $d = 0$, so heißt ein Punkt *unendlich fern* (wie in 2D).

Die Darstellung durch homogene Koordinaten ist *nicht eindeutig*.

Alle Tupel $(\lambda a, \lambda b, \lambda c, \lambda d), \lambda \neq 0$ stellen den gleichen Punkt dar.

Punkte der Form $(a, b, c, 0), a \neq 0 \vee b \neq 0 \vee c \neq 0$ definieren unendlich ferne Punkte.

1.2.2 Ebenen

Ebenen können in der Form $\{(x, y, z)|a_1x+a_2y+a_3z+a_4 = 0\}$ (in kartesichen Koordinaten) mit
$a_1 \neq 0 \vee a_2 \neq 0 \vee a_3 \neq 0 \vee a_4 \neq 0$
dargestellt werden.

Das 4-Tupel ist *nicht eindeutig* bestimmt. (Analog zur Punktdarstellung) beschreibt jedes $(\lambda a_1, \lambda a_2, \lambda a_3, \lambda a_4)|\lambda \neq 0$ die gleiche Ebene.

1.2.3 Geraden

Geraden können durch zwei Punkte p und q identifiziert werden. Die Gerade existiert dann, wenn $p \neq q$ gilt.

Oft werden Geraden aber auch durch den Schnittpunkt zweier Ebenen

$$(a_1, a_2, a_3, a_4), (b_1, b_2, b_3, b_4)$$

identifiziert, also:

$$g = \{(x, y, z)|a_1x + a_2y + a_3z + a_4 = 0 = b_1x + b_2y + b_3z + b_4\}$$

1.3 Eigenschaften des 3-dimensionalen projektiven Raumes

- Durch je 3 verschiedene Punkte gibt es mindestens eine Ebene.

- Durch je 2 verschiedene Punkte gibt es genau eine Gerade.
- Je 2 verschiedene Ebenen schneiden sich in einer Geraden.
- Je 3 verschiedene Ebenen schneiden sich in mindestens einem Punkt.
- Eine Gerade g ist in einer Ebene E enthalten oder schneidet sie in genau einem Punkt.
- Zwei Geraden können einen leeren Schnitt haben („windschiefe Geraden“).

1.4 Strecken, Polygone, Polygonzüge im 3-dimensionalen Raum

Diese Formen können analog zur Definition im 2-dimensionalen Raum modelliert werden.

Polygone lassen sich ebenfalls analog zur 2D-Festlegung definieren. Allerdings sind hier auch Sonderformen möglich (siehe Abbildung 1.8). Wegen der Möglichkeit verschachtelter Polygone wird oft die Vorgabe gemacht, ebene Polygone zu benutzen.

Kurven: $\boldsymbol{p} : I \longrightarrow \mathbb{R}^3$ $(x(t), y(t), z(t)$, $I \in \mathbb{R}$ (Intervall).

zusätzlich: Polyeder
Polyeder werden als Teil des 3-dimensionalen Raums definiert, der komplett durch Ebenen begrenzt ist.

Beispiele für Polyeder sind n-seitige Würfel (z.B. der bekannte 6-seitige Würfel oder die Würfel (nicht im mathematischen Sinne, sondern im praktischen) des D&D-Systems). Polyeder können aber auch Löcher enthalten (siehe Abbildung 1.9), solange die Teilung des Raumes (Inneres, Äußeres) aufrecht bleibt.

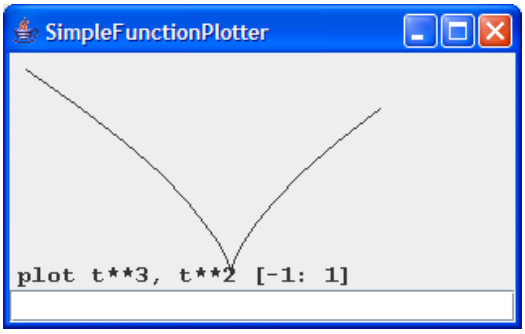


Abbildung 1.7: Beispiel 7

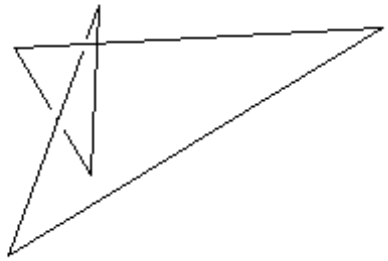


Abbildung 1.8: Polygonform im 3-dimensionalen Raum

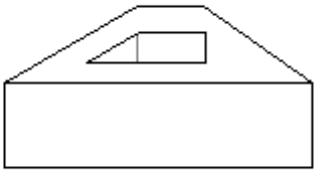


Abbildung 1.9: Polyeder „mit Loch“

1.5 Polyeder(polyhedron, polyhedra)

Idee: Polyeder sind Teilmengen des 3d Raumes, die von ebenen einfachen Polygonen (hier: „Polygon“ enthält auch sein Inneres) begrenzt sind.

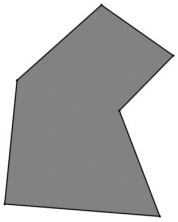


Abbildung 1.10: Polygon

1.5.1 Halbräume

Halbräume sind Mengen der Form $\{(x, y, z) \,| a_1x + a_2y + a_3z + a_4 \geq 0\}$

1.5.2 Konvexes Polyeder

Eine Menge $A \subset \mathbb{R}^d$ heißt konvex genau dann, wenn mit $p, q \in A$ auch die ganze Strecke $\overline{pq} \subset A$.

Ein konvexes Polyeder ist der Durchschnitt endlich vieler Halbräume, d.h. die Menge aller (x, y, z) mit

$$\begin{array}{ccccccc} a_{11}x & + & a_{12}y & + & a_{13}z & + & a_{14} \geq 0 \\ a_{21}x & + & a_{22}y & + & a_{23}z & + & a_{24} \geq 0 \\ \vdots & & \vdots & & \vdots & & \vdots \\ a_{k1}x & + & a_{k2}y & + & a_{k3}z & + & a_{k4} \geq 0 \end{array}$$

d.h. $A \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \geq \mathbf{0}$; $\qquad A = \begin{pmatrix} a_{11} & \cdots & a_{14} \\ \vdots & & \vdots \\ a_{k1} & \cdots & a_{k4} \end{pmatrix}$

Der maximale Teil eines konvexen Polyeders P, der in genau einer zu einem definierenden Halbraum gehörenden Ebene liegt, heißt Facette (Seitenfläche) von P.

Der maximale Teil eines konvexen Polyeders P, der in genau zwei zu einem definierenden Halbraum gehörenden Ebenen liegt, heißt Kante von P.

Der maximale Teil eines konvexen Polyeders P, der in mindestens drei zu einem definierenden Halbraum gehörenden Ebenen liegt, heißt Ecke von P.

Beispiele:

1. Tetraeder: $x \geq 0;\ y \geq 0;\ z \geq 0$
   $x + y + z \leq 1 \Leftrightarrow -x - y - z + 1 \geq 0$

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

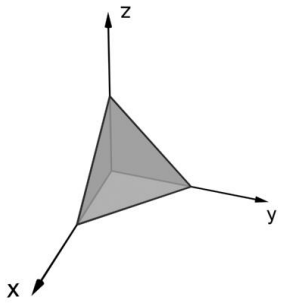


Abbildung 1.11: Tetraeder

Ein Tetraeder hat 4 Facetten, 6 Kanten und 4 Ecken:

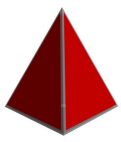


Abbildung 1.12: Facetten

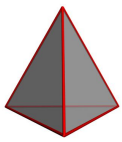


Abbildung 1.13: Kanten

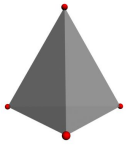


Abbildung 1.14: Ecken

2. Unbeschränktes Polyeder: $x \geq 0$; $y \geq 0$
   Unbeschränkte Polyeder heißen auch Polytope.
   Dieses Polyeder hat 2 Facetten, 1 Kante und 0 Ecken.

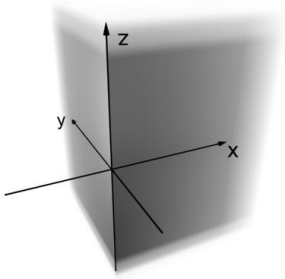


Abbildung 1.15: Polytop

3. reguläre konvexe Polygone, d.h. die Facetten sind alle kongruent und reguläre Polygone.
   regulär: Alle Seitenlängen und Winkel sind gleich groß.
   Es gibt unendlich viele reguläre Polygone.

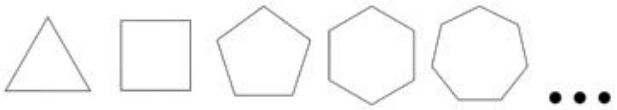


Abbildung 1.16: Polygone

In 3d gibt es fünf reguläre konvexe Polyeder (Platonische Körper):

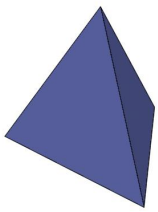


Abbildung 1.17: Tetraeder

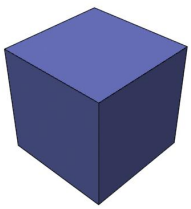


Abbildung 1.18: Hexaeder(Würfel)

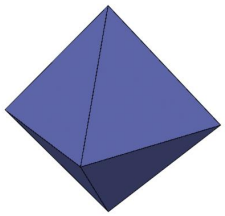


Abbildung 1.19: Oktaeder

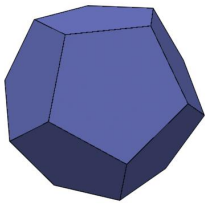


Abbildung 1.20: Dodekaeder

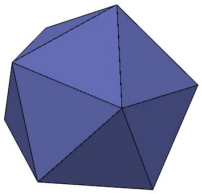


Abbildung 1.21: Ikosaeder

1.5.3 Polyeder allgemein

Polyeder können allgemein als endliche Vereinigung regelmäßiger zusammenhängender Polygone definiert werden.

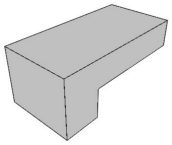


Abbildung 1.22: Polyeder

Definition von W. Nef:
Die kleinste Menge von Teilmengen des $\mathbb{R}^3$, die alle Halbräume enthält und unter $\cap$, $\cup$ und Komplement abgeschlossen ist, heißt Polyeder.

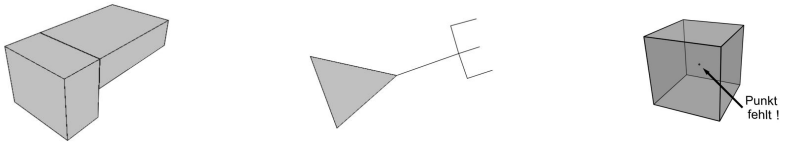


Abbildung 1.23: Polyeder nach Nef

1.6 Flächen im $\mathbb{R}^3$

Definition durch Parametriesierung:

$\mathbf{a}: G \longrightarrow \mathbb{R}^3$, wobei $G \subset \mathbb{R}^3$ zusammenhängendes Gebiet
$(u,v) \longmapsto (x(u,v), y(u,v), z(u,v)) = \mathbf{a}(u,v)$

Die Fläche ist dann die Menge:

$$\{(x(u,v), y(u,v), z(u,v)) \,|(u,v) \in G\}$$

(Bildmenge von $\mathbf{a}$).

Beispiele:

1. $G = \underbrace{[0,\pi]}_{u\in} \times \underbrace{[0,1]}_{v\in}$
   $\mathbf{a}(u,v) = (\cos(u), \sin(u), v)$

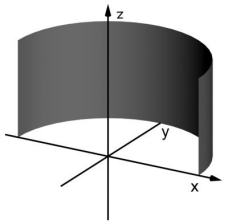


Abbildung 1.24: Beispiel 1

2. $\mathbf{a}(u,v) = (u, v, \sin(u)\cdot\sin(v))$
3. $\mathbf{a}(u,v) = (u, v, u^2 + v^2)$ Rotationsparaboloid
4. Oberfläche einer Kugel
5. $\mathbf{a}(u,v) = (u, v, u^2 - v^2)$
6. $\mathbf{a}(u,v) = (u^3 - v^2, uv, u^2 + v^2)$

Ein Netzt von Linien der Form $u = const, v = const$ auf G erzeugt auf der Fläche F ein Netz von Kurven: „krummlinige Koordinaten“.

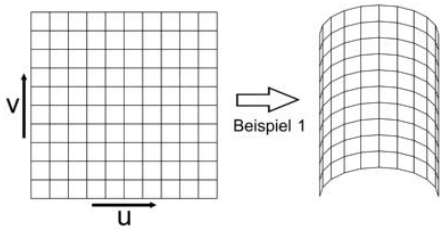


Abbildung 1.25: krummlinige Koordinaten

1.6.1 glatte Flächen

Eine Fläche mit Parametern $\mathbf{a} : G \longrightarrow \mathbb{R}^3$ heißt r-fach differenzierbar („glatt“) genau dann, wenn $\mathbf{a}$ r-fach stetig differenziebar ist und die Vektoren $\frac{\partial \mathbf{a}}{\partial u}(u,v)$ und $\frac{\partial \mathbf{a}}{\partial v}(u,v)$ linear unabhängig sind für alle u,v.

Beispiel 1:

$$\begin{aligned} \mathbf{a}(u,v) &= (\cos(u), \sin(u), v) \\ \frac{\partial \mathbf{a}}{\partial u}(u,v) &= (\sin(u), \cos(u), 0) \\ \frac{\partial \mathbf{a}}{\partial v}(u,v) &= (0,0,1) \end{aligned}$$

$\frac{\partial \mathbf{a}}{\partial u}(u_0,v_0)$: Tangentenvektor an die Kurve $v = v_0$, u variiert
$\frac{\partial \mathbf{a}}{\partial v}(u_0,v_0)$: Tangentenvektor an die Kurve $u = u_0$, v variiert
Die Ebene durch (u_o, v_o), die von diesen Vektoren aufgespannt wird, heißt Tangentialebene an F in (u_o, v_o), also $\mathbf{a}(u_o,v_o) + \lambda \frac{\partial \mathbf{a}}{\partial u}(u,v) + \mu \frac{\partial \mathbf{a}}{\partial v}(u,v)$, $\lambda, \mu \in \mathbb{R}$.

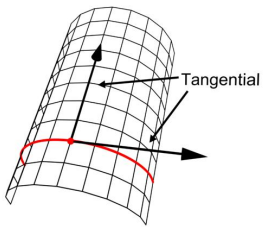


Abbildung 1.26: Tangentialebene

1.7 Modellierung von Kurven und Flächen mittels B-Splines

Das Wort Spline, übersetzt mit „längliches, dünnes Stück Holz oder Metall“ kommt aus dem Schiffbau, wobei dünne, elastische Holz- oder Metalllatten an einzelnen Punkten durch Nägel fixiert, zur Festlegung der Kontur unter Spannung benutzt werden.
Hier bezeichnet ein Spline n-ten Grades eine Funktion, die stückweise aus Polynomen mit maximalem Grad n zusammengesetzt ist.

B-Splines sind die meist genutzte Möglichkeit Splines anzunähern. Dies geschieht durch Approximation vorgegebene Punkte mit Hilfe von Gewichtsfunktionen, wobei der erste und letzte Punkt Anfangs- und Endpunkt der Kurve sein können. Gegeben sind n+1 vom Benutzer gewählte Punkte in der Ebene, $p_0, \cdots, p_n$, die sogenannten Kontrollpunkte. Gesucht ist eine möglichst glatte Kurve (etwa C^2-stetig), die in der Nähe dieser Kontrollpunkte verläuft und sich durch Verschieben der Kontrollpunkte lokal verändern läßt. Neben den B-Splines gibt es unter anderen die Bezier-Splines. Die B-Splines haben jedoch gegenüber den Bezier-Splines den Vorteil, daß der Grad der Polynome unabhängig von der Anzahl der Kontrollpunkte und eine lokale Kontrolle möglich ist.
Anwendung finden B-Splines in CAED (computer aided geometry design) Programmen. Dort werden sie z.B. für das Modellieren von Karosserien verwendet.

C^0 bedeutet, dass es sich um stetige Kurven handelt, C^1-Kurven sind mindestens einmal stetig differenzierbar und die Richtung ändert sich stetig. Bei C^2-Kurven ändern sich die Richtung und Krümmung stetig. Die 2.Ableitung einer Kurve entspricht der Krümmung.

Als Ansatz für B-Spline-Kurven nehmen wir parametrisierte Kurven. Damit lautet die Kurvendarstellung

$$\boldsymbol{c}(t) = \sum_{i=0}^{n} B_{i,k}(t) \cdot \boldsymbol{p}_i \quad , \quad t \in [0,1], \tag{1.1}$$

wobei $B_{i,k}(t) : \mathbb{R} \to \mathbb{R}$ mit i=0,...,n Funktionen in C^{k-2} sind. Die $B_{i,k}(t)$ werden auch B-Spline-Basisfunktionen genannt. Für jedes t ist $\boldsymbol{c}(t)$ die gewichtete Summe der Kontrollpunkte.

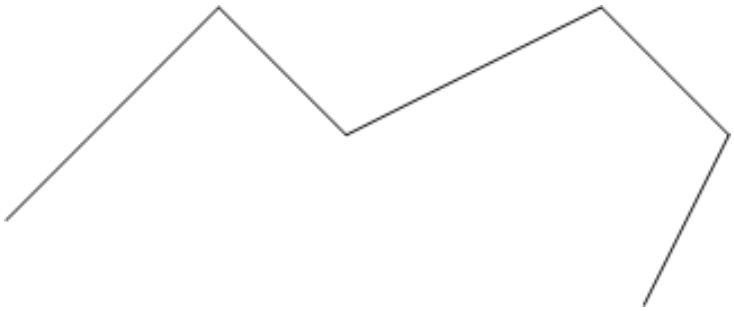


Abbildung 1.27: C^0-stetige Kurve

Abbildung 1.28: C^1-stetige Kurve aus Strecken und Viertelkreis

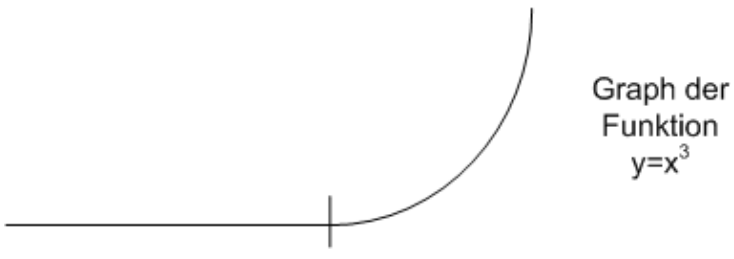


Abbildung 1.29: C^2-stetige Kurve

Die Funktionen sollen gewissen Anforderungen genügen. Es sollte

$$\sum_{i=0}^{n} B_{i,k}(t) = 1 \tag{1.2}$$

und

$$\boldsymbol{c}(t) \in C^{k-2} \tag{1.3}$$

gelten.
Alle B-Spline-Basisfunktionen der Ordnung o können rekursiv aus den Basisfunktionen der Ordnung o-1 gewonnen werden. Also gilt

$$B_{i,1}(t) = \begin{cases} 1 & t \in [t_i, t_{i+1}] \\ 0 & sonst \end{cases}$$

$$B_{i,k}(t) = \frac{t - t_i}{t_{i+k-1} - t_i} \cdot B_{i,k-1}(t) + \frac{t_{i+k} - t}{t_{i+k} - t_{i+1}} \cdot B_{i+1,k-1}(t).$$

Das Intervall [0,1] wird dabei in n+k Teilintervalle geteilt.

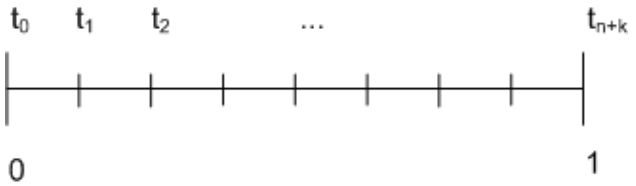


Abbildung 1.30: Unterteilung des Intervalls in n+k Teilintervalle

Die B-Spline-Basisfunktionen $B_{i,1}(t)$ der Ordnung 1 sind Rechteckfunktionen der Höhe 1 über dem Intervall $[t, t_{i+1}]$. Sie sind stückweise konstant und außerhalb des Intervalls identisch Null. Die Funktionen $B_{i,1}(t) \in C^{-1}$ sind unstetig.

B-Spline-Basisfunktionen $B_{i,2}(t)$ der Ordnung 2, k ist also 2, sind Dreiecksfunktionen über dem Intervall $[t, t_{i+2}]$. Die Rekursionsgleichung lautet:

$$B_{i,2}(t) = \frac{t - t_i}{t_{i+1} - t_i} \cdot B_{i,1}(t) + \frac{t_{i+2} - t}{t_{i+2} - t_{i+1}} \cdot B_{i+1,1}(t)$$

Diese Funktionen sind stückweise linear und außerhalb des Intervalls identisch Null. Die $B_{i,2}(t) \in C^0$ sind stetig.

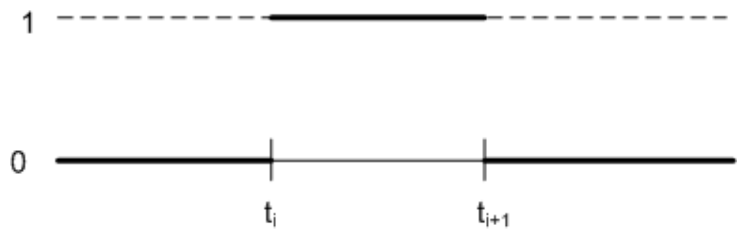
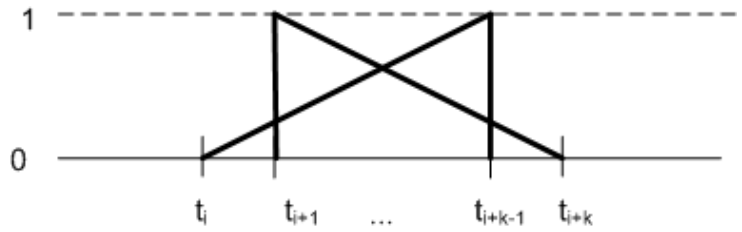


Abbildung 1.31: B-Spline-Basisfunktionen $B_{i,1}(t)$ der Ordnung 1



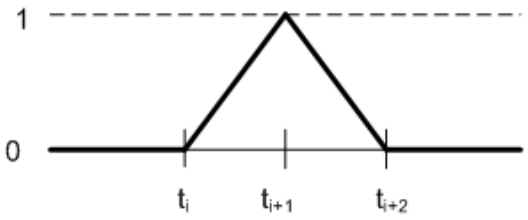


Abbildung 1.32: B-Spline-Basisfunktionen $B_{i,2}(t)$ der Ordnung 2

Für $k = 3$ sind die Basisfunkionen $B_{i,3}(t)$ der Ordnung 3 Hütchenfunktionen mit der Rekursionsgleichung

$$B_{i,3}(t) = \frac{t - t_i}{t_{i+2} - t_i} \cdot B_{i,2}(t) + \frac{t_{i+3} - t}{t_{i+3} - t_{i+1}} \cdot B_{i+1,2}(t).$$

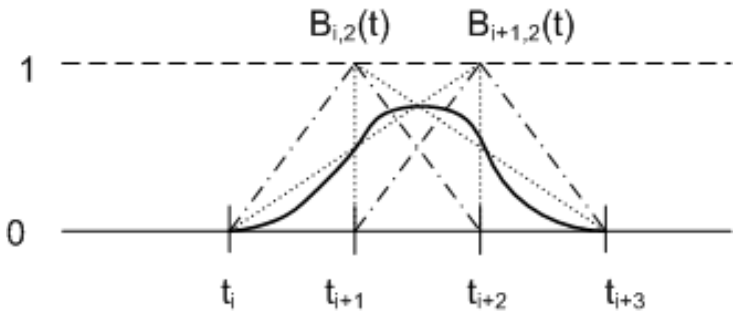


Abbildung 1.33: B-Spline-Basisfunktionen $B_{i,3}(t)$ der Ordnung 3

Die $B_{i,3}(t) \in C^1$ sind stückweise quadratisch und an den Grenzen der Stücke gehen sie glatt ineinander über. Außerhalb des Intervalls $[t_i, t_{i+3}]$ sind sie identisch Null.
$B_{3,3}(t)$ ist symmetrisch zu $B_{1,3}(t)$ und $B_{4,3}(t)$ zu $B_{0,3}(t)$.

Für alle anderen Basisfunktionen gilt es analog, so dass die $B_{i,k}(t)$ stückweise polynomiell vom Grad $k - 1 \in C^{k-2}$ und identisch Null außerhalb von $[t_i, t_{i+k}]$ sind.
Um zu erreichen, dass die Basisfunktionen eine Teilung der 1 bilden, also dass $\sum_{i=0}^{n} B_{i,k}(t) = 1$ für alle t gilt, setzen wir $t_0 = t_1 = t_2 = \cdots = t_{k-1} = 0$ und analog $t_{n+1} = t_{n+2} = \cdots = t_{n+k} = 1$.
Im Inneren wäre es erfüllt gewesen, aber am Anfang und Ende des Intervalls [0,1] würde es sonst noch nicht stimmen.
Der Punkt P_i mit den Koeffizienten $B_{i,k}(t)$ hat nur auf das Intervall $[t_i, t_{i+k}]$ Einfluss. Umgekehrt ist es so, dass für $t \in [t_i, t_{i+1}]$ nur die Punkte $P_{i-k+1}, ..., P_i$ relevant sind. Punkte auf der Kurve $\boldsymbol{c}(t) = \sum_{i=0}^{n} B_{i,k}(t) \cdot \boldsymbol{p}_i$ sind eine Konvexkombination dieser Punkte. Eine Konvexkombination ist die Linearkombination, wobei die Summe der Koeffizienten gleich 1 ist. Punkte in dem Dreieck sind genau die Punkte, die sich durch die Konvexkombination der Eckpunkte bilden lassen.

Die Konvexkombination von $P_1, ..., P_n$ bilden die konvexe Hülle von $P_1, ..., P_n$,

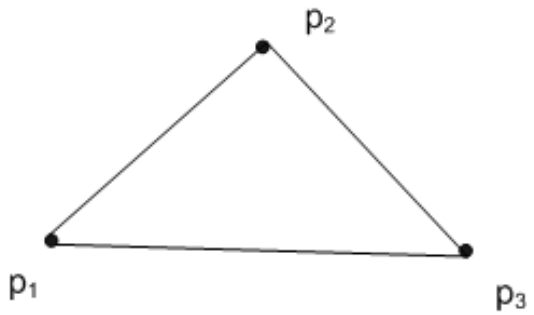


Abbildung 1.34: Konvexe Hülle der Punkte $P_1, \cdots, P_3$

d.h. die kleinste konvexe Menge, die $P_1, ..., P_n$ enthält.

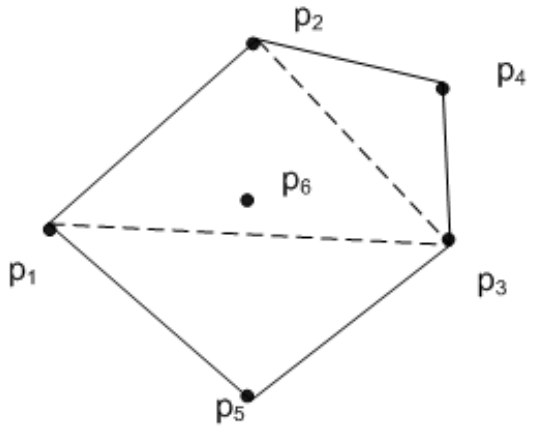


Abbildung 1.35: Konvexe Hülle der Punkte $P_1, \cdots, P_6$

Also liegt die Kurve $\boldsymbol{c}(t)$ für $t \in [t_i, t_{i+1}]$ in der konvexen Hülle von $P_{i-k+1}, ..., P_i$. Die Kurve insgesamt liegt in der konvexen Hülle der Kontrollpunkte.

1.7.1 B-Splineflächen im 3dim. Raum

$$a : [0,1]^2 \to \mathbb{R}^3$$

$$\boldsymbol{a}(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{n} \boldsymbol{p}_{ij} \cdot B_{i,k}(u) \cdot B_{j,k}(v),$$

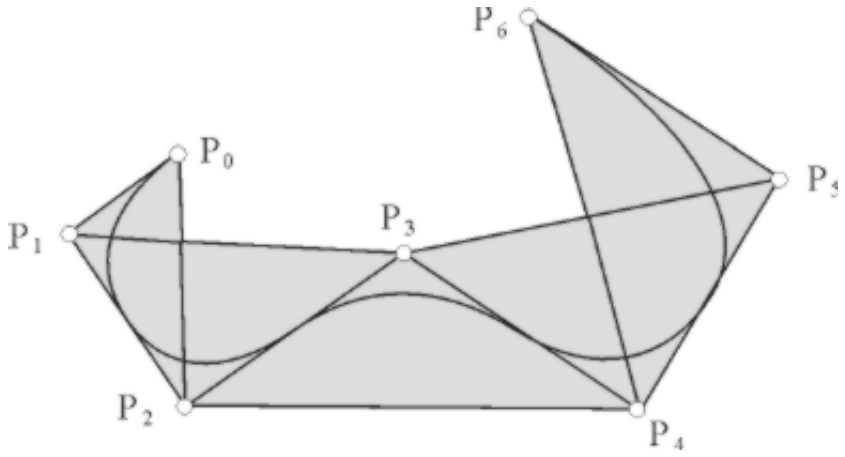


Abbildung 1.36: Kurve liegt in der konvexen Hülle des Kontrollpolygons

wobei p_{ij} die Kontrollpunkte sind und $0 \leq i, j \leq n$ gilt.

1.8 Modellieren von 3d - Objekten

1.8.1 Drahtmodell (wire frame model)

Die Darstellungselemente des Drahtmodells sind Kurven (im einfachsten Fall: Strecken). Es werden nur 1-dimensionale Kanten (Strecken, Kurven) eines Objekts angegeben.

Bsp:

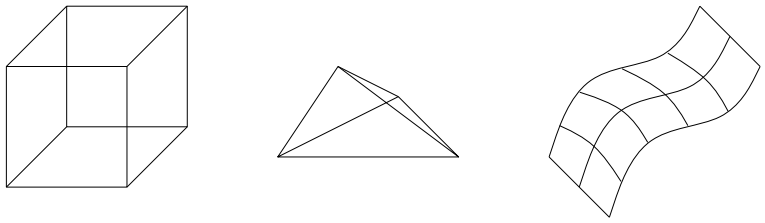


Abbildung 1.37: Beispiele

Die kombinatorische Struktur wird mithilfe von Graphen, also Ecken und Kanten, modelliert. Diese Informationen reichen jedoch nicht aus, um das Objekt darstellen zu können. Zusätzlich zur kombinatorischen Struktur muss die geometrische Struktur des Objekts beschrieben werden. Wenn alle Kanten Strecken sind, reicht es aus, jeder Ecke ein Punkt im 3d-Raum zuzuordenen. Gibt es auch Kurven müssen diese zu den jeweiligen Kanten angegeben werden (z.B. parametrisiert).

Vorteil: Das Drahtmodell ist einfach darzustellen und wiederzugeben. Zudem ist es sehr platzeffizient.

Nachteile:

- Die Flächen sind nicht dargestellt, daher werden Kanten, die eigentlich verdeckt sein sollten, nicht entfernt.
- Mehrdeutigkeit (siehe Abbildung 1.38)
- unmögliche Objekte (siehe Abbildung 1.39)

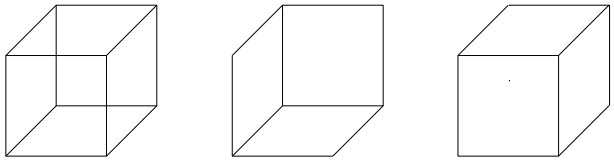


Abbildung 1.38: Mehrdeutigkeit

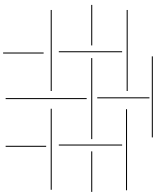


Abbildung 1.39: unmögliche Objekte

1.8.2 Flächenmodelle:

In diesem Modell werden die 3d-Objekte durch ihre Oberfläche dargestellt. Diese kann wieder mit Splines oder durch ebene Polygone (in diesem Fall wären die 3d-Objekte Polyeder; die Objekte können so beliebig approximiert werden) modelliert werden. Konkret kann dieses auf zwei Darstellungsarten geschehen:

Hierarchische Darstellung:

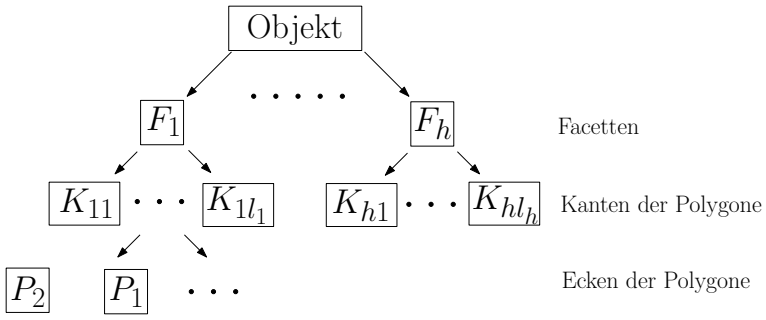


Abbildung 1.40: Hierarchische Darstellung

Darstellung durch einen Graph:

In dieser Darstellung sind die Knoten die Facetten des Objekts und die Kantenmenge ist wie folgt definiert: Es gibt eine Kante (im Graphen) zwischen F_1 und $F_2 \Leftrightarrow F_1$ und F_2 haben eine gemeinsame Kante (im Objekt).

Bsp.:

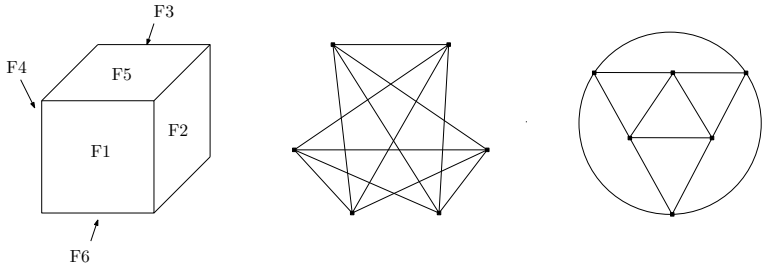


Abbildung 1.41: Würfel und zwei isomorphe Oktaeder

1.8.3 Volumenmodelle

CSG (constructive solid geometry)

Idee: Eine Menge von Grundkörpern (z.B. Quader, Zylinder, Kugeln, Kegel, Halbräume...) wird spezifiziert durch geometrische Daten.

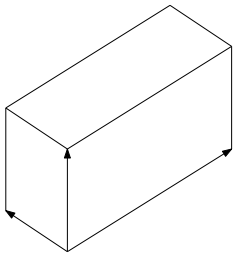


Abbildung 1.42: Parallelepiped

z.B. Parallelepiped (Verallgemeinerung eines Quaders) angegeben durch p, $\boldsymbol{u}_1$, $\boldsymbol{u}_2$, $\boldsymbol{u}_3$

Kompliziertere Objekte/Szenen können dann durch Mengenoperationen ($\cup$, $\cap$, $\setminus$) aus Grundbausteinen dargestellt werden.

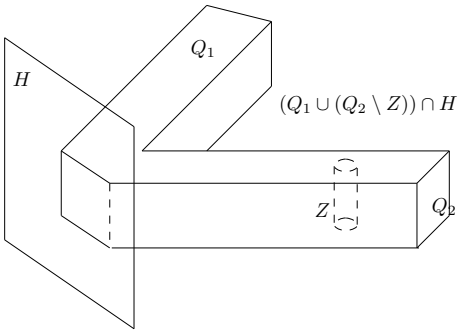


Abbildung 1.43:

Zum Beispiel wurden die Objekte im CG - Praktikum (siehe Webseite) durch Quadriken modelliert.

Quadriken Eine Quadrik ist allgemein definiert durch:

$$a_1x^2 + a_2y^2 + a_3z^2 + a_4xy + a_5xz + a_6yz + a_7x + a_8y + a_9z + a_{10} = 0$$

Will man nicht die Oberflächen sondern die Volumen der Objekte beschrieben, so wählt man anstatt des $=$ ein $\leq$.

Beispiel 1: Eine Kugeloberfläche mit Radius 1 um den 0-Punkt wird als Quadrik wie folgt definiert: $a_1 = a_2 = a_3 = 1$ und $a_{10} = -1$, alle anderen $a_i = 0$:

$$x^2 + y^2 + z^2 - 1 = 0$$

Beispiel 2: Rotations-Paraboloid, Abbildung 1.44:

$$x^2 + y^2 - z = 0$$

Beispiel 3: Doppelter Kegel, Abbildung 1.45:

$$x^2 + y^2 - z^2 = 0$$

Beispiel 4: Zylinder

$$x^2 + y^2 - a = 0$$

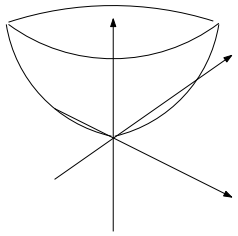


Abbildung 1.44: Rotations-Paraboloid

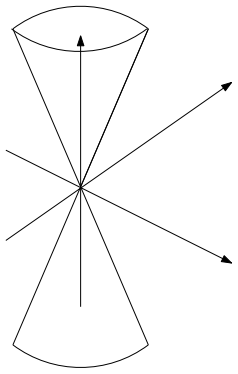


Abbildung 1.45: Doppelter Kegel

Beispiel 5: Spezieller Fall: Ebene: $a_1x + a_2y + a_3z + a_4 = 0$

Zu den Quadriken als Grundelemente hat man zur Erzeugung komplizierterer Objekte wieder die Mengenoperationen zur Verfügung.

Octtrees

Um 3d - Objekt M (annähernd) zu beschreiben:

1. Man schließt das Objekt M in einen Würfel W ein, so dass das ganze Objekt in ihm enthalten ist. Dies heisst 0-te Näherung.
2. Teile W in 8 Teilwürfel der halben Kantenlängen. Von diesen gib es
   (a) Würfel, die kein Teil des Objekts enthalten (leer),
   (b) Würfel, die von dem Objekt ausgefüllt sind (voll) und
   (c) Würfel, die teilweise mit dem Objekt gefüllt sind.

3. Teile die Würfel dritter Art rekursiv auf.

4. Die Rekursion wird abgebrochen, wenn die Approximation hinreichend genau ist.

Analog verwendet man zur Modellierung von 2d-Objekten Quadtrees.

1.9 $2\frac{1}{2}$ -dimensionale Darstellungen

Auf einer Fläche F (2 dimensional) wird eine Operation ausgeführt
Zum Beispiel wir eine Verschiebung um den Vektor $\mathbf{t}$ durchgeführt. Gemeint ist der Körper, der überstrichen wird, wenn man F um den Vektor $\mathbf{t}$ verschiebt(*Abb*1):

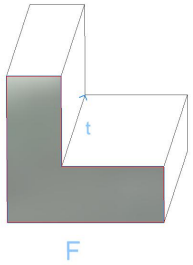


Abbildung 1.46: Verschiebung

Oder die Rotation einer Fläche um eine Achse(*Abb*2.):

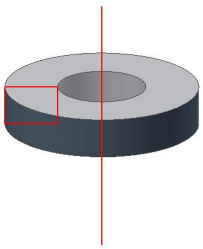


Abbildung 1.47: Rotation

Auch $1\frac{1}{2}$ dimensionale Darstellung von 2D Objekten sind möglich:

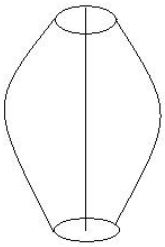


Abbildung 1.48: Silhouette

Kapitel 2

Transformationen und Projektionen

2.1 2D Transformationen

2.1.1 Translation(Verschiebung)

Abbildung $R^2 \to R^2$, die zu jedem Punkt/Vektor einen festen Vektor $\mathbf{t} = \begin{pmatrix} t_x \\ t_y \end{pmatrix}$ addiert:

$f_{\mathbf{t}} : R^2 \to R^2$
$\mathbf{v} \mapsto \mathbf{v} + \mathbf{t}$

Erhält Längen von Strecken, Winkel, Umlaufssinn, ...
aber nicht den Ursprung.

2.1.2 Lineare Abbildung

Eine lineare Abbildung ist von der Form

$f : R^2 \to R^2$
$\mathbf{v} \mapsto A * \mathbf{v}$, wobei A eine $2 \times 2-$ Matrix
$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$

Bedeutung: Eine lineare Abbildung ist eine Rotation und Skalierung, aber keine Verschiebung des Koordinatensystems*Abb*1.1.

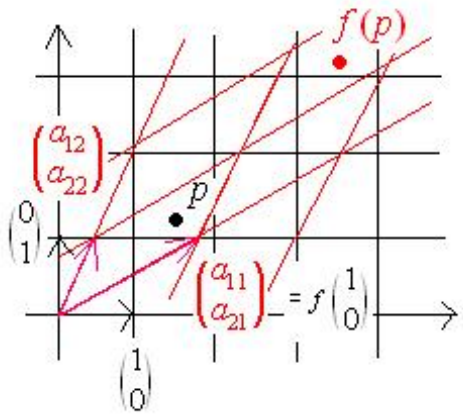


Abbildung 2.1: Bedeutung der linearen Abbildung

Erhalten bleibt der Ursprung.

2.1.3 Rotation(Drehung)

Die Rotation um einen Winkel α gegen den Uhrzeigersinn um den Ursprung ist die lineare Abbildung.
$r_\alpha : R^2 \to R^2$
$\mathbf{v} \mapsto R_\alpha * \mathbf{v}$,
wobei $R_\alpha = \begin{pmatrix} cos\alpha & -sin\alpha \\ sin\alpha & cos\alpha \end{pmatrix}$

Die Drehung um einen beliebigen Punkt p wird folgendermaßen durchgeführt:

- Translation um -**p**
- Drehung um Ursprung
- Translation um **p**

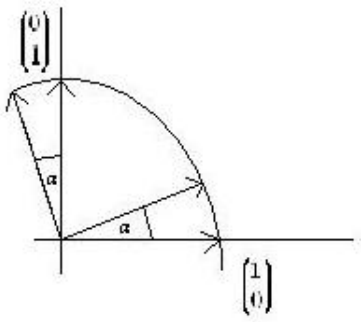


Abbildung 2.2: Rotation um Winkel α

Die allgemeine Formel für die Rotation ist also: $\mathbf{v} \mapsto R_\alpha(\mathbf{v} - \mathbf{p}) + \mathbf{p}$

I.a. ist dies keine lineare Abildung, da der Ursprung nicht zwangsläufig erhalten bleibt.

Rotation ist längen- und winkeltreu!
Umlaufsinn bleibt erhalten:

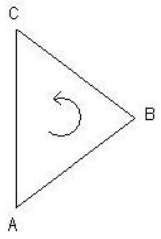


Abbildung 2.3: Umlaufsinn vor Rotation

2.1.4 starre Bewegung

Die starre Bewegung ist eine Hintereinanderausführung von Translation und Rotation (oBdA: um Ursprung)
s: $R^2 \to R^2$ s: $\mathbf{v} \mapsto R_\alpha(\mathbf{v} + \mathbf{t})$

Sie ist Längen- und winkeltreu und der Umlaufsinn bleibt erhalten.

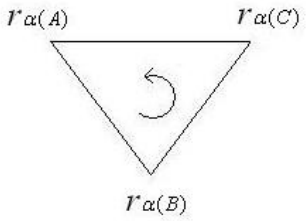


Abbildung 2.4: Umlaufsinn nach Rotation

2.1.5 Spiegelung(Reflexion)

Spiegelung an der x-Achse(*Abb*1.5) ist eine lineare Abbildung mit Matrix $\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$

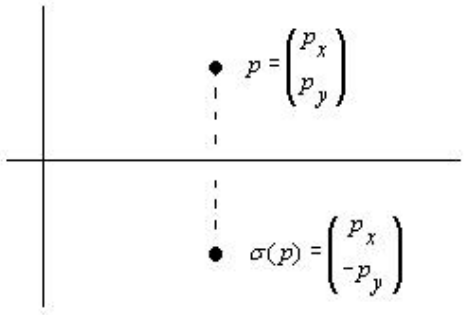


Abbildung 2.5: Spiegelung an der x-Achse

An einer beliebigen Geraden g: σ_g ist das folgendermaßen realisierbar:

- Translation, so dass Bild g' von g durch Ursprung (Vektor $\mathbf{t}$)
- Drehung so dass Bild g" von g' = x-Achse (Winkel α)
- Spiegelung an x-Achse
- inverse Drehung (um -α)
- inverse Translation (um -$\mathbf{t}$)

Spiegelung erhält Längen und Winkel, dreht Umlaufsinn um.

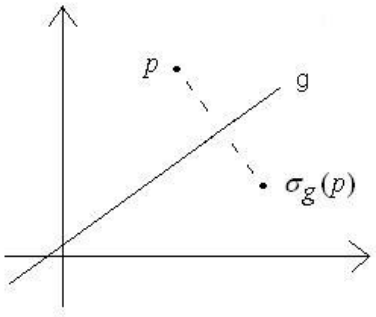


Abbildung 2.6: Spiegelung an Gerade g

2.1.6 Isometrie(Kongruenzabbildung)

Isometrie ist die Hintereinanderausführung von starren Bewegungen und Spiegelungen.
Sie ist längen- und winkeltreu und wird dadurch charakterisiert.
Zwei Objekte heißen kongruent, wenn eines das Bild des anderen unter Isometrie darstellt.

2.1.7 (zentrische) Streckung

Die Streckung um Faktor a $\in$ R am Ursprung ist eine lineare Abbildung mit der Matrix $\begin{pmatrix} a & 0 \\ 0 & a \end{pmatrix}$

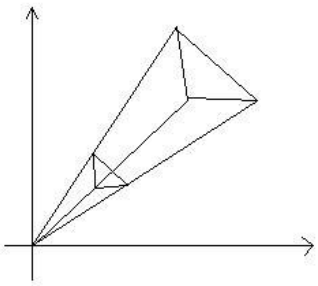


Abbildung 2.7: Streckung

Sie erhält die Winkel, nicht aber Längen, falls a $\neq$ 1.

Dies ist auch an beliebigem Punkt $\mathbf{p}$ möglich.

$\mathbf{v} \mapsto \mathbf{p} + a(\mathbf{v} - \mathbf{p})$

2.1.8 Ähnlichkeitsabbildung

Eine Ähnlichkeitsabbildung ist die Hintereinanderausführung von Isometrie und Streckung.
Sie ist Winkeltreu.
Zwei Objekte heißen ähnlich wenn eines das Bild des anderen unter Ähnlichkeitsabbildung ist.

2.1.9 Affine Abbildung

Eine affine Abbildung ist die Hintereinanderausführung von linearer Abbildung und Translation.
a:$R^2 \to R^2$
$\mathbf{v} \mapsto A * \mathbf{v} + \mathbf{t}$
für feste Matrix A, festen Vektor $\mathbf{t}$

Sie umfasst alle bisherigen Klassen.
I.a. ist es keine lineare Abbildung, aber in homogenen Koordinaten als Matrix-Vektor-Produkt darstellbar.
$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$
$\mathbf{t} = \begin{pmatrix} t_x \\ t_y \end{pmatrix}$

kartesische Koordinaten:
$\begin{pmatrix} v_x \\ v_y \end{pmatrix} \mapsto \begin{pmatrix} a_{11}v_x + a_{12}v_y + t_x \\ a_{21}v_x + a_{22}v_y + t_y \end{pmatrix}$

homogene Koordinaten:
$$\begin{pmatrix} v_x \\ v_y \\ 1 \end{pmatrix} \mapsto \begin{pmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} v_x \\ v_y \\ 1 \end{pmatrix} = \begin{pmatrix} a_{11}v_x + a_{12}v_y + t_x \\ a_{21}v_x + a_{22}v_y + t_y \end{pmatrix}$$

z.B. einzelne Translation

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

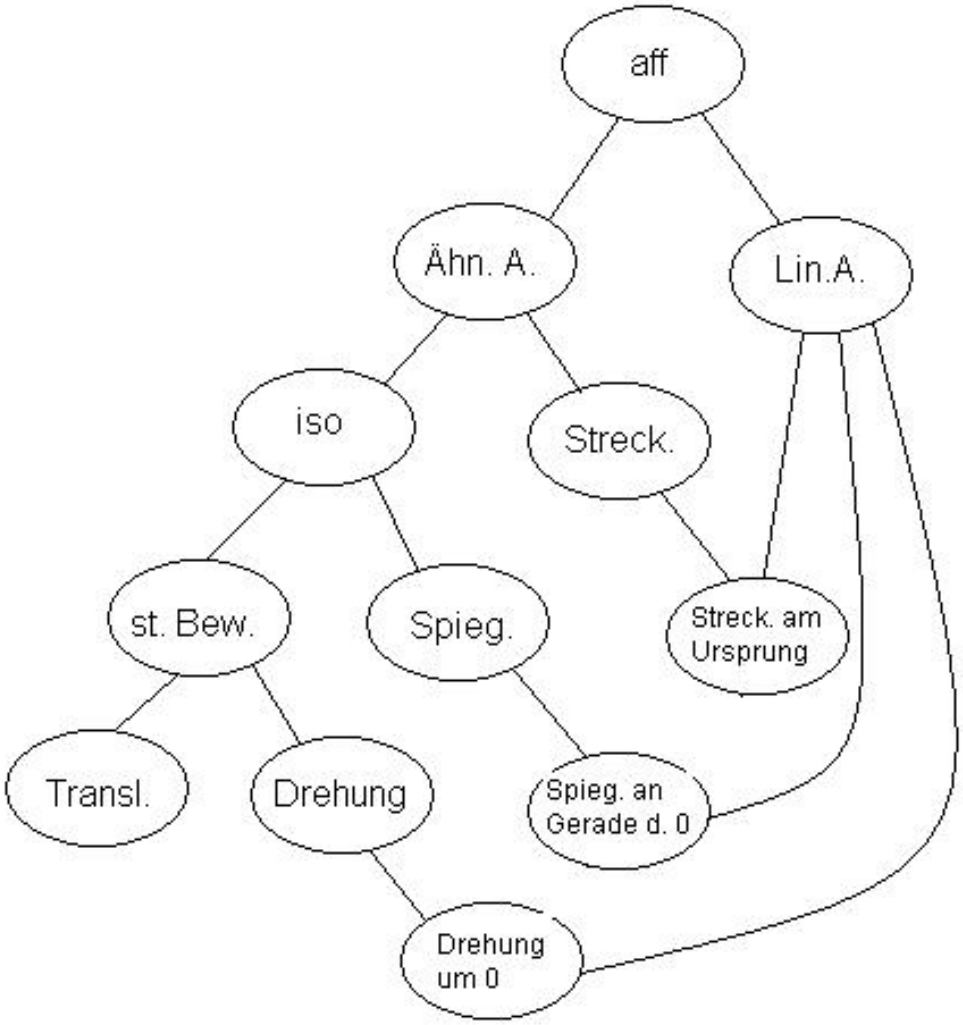


Abbildung 2.8: Hierarchie

2.2 3D - Transformationen

2.2.1 Translation

analog zu 2D

$f_{\mathbf{t}} : R^3 \to R^3$
$\mathbf{v} \mapsto \mathbf{v} + \mathbf{t}$

2.2.2 lineare Abbildung

analog zu 2D

$f : R^3 \to R^3$
$\mathbf{v} \mapsto A * \mathbf{v}$, wobei A eine $3 \times 3-$ Matrix

Eigenschaften:

- Bild einer Ebene ist
  - eine Ebene oder
  - eine Gerade oder
  - ein Punkt
- Parallelität von Geraden oder Ebenen bleibt erhalten, falls Bild der Geraden (Ebenen) wieder Gerade(Ebene) ist.
- Konvexes Polyeder wird abgebildet auf ein konvexes Polyeder.
- Teilungsverhältnis einer Strecke bleibt erhalten(falls Strecke nicht zu einem Punkt kollabiert).

2.2.3 Affine Abbildungen

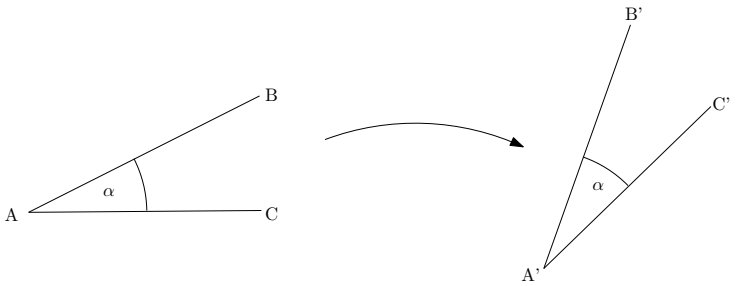
Abbildungen der Form $\mathbf{v} \mapsto A \cdot \mathbf{v} + \mathbf{t}$, wobei A eine 3×3-Matrix und $\mathbf{v}$ ein Vektor ist, haben auch die Eigenschaften, die bei den linearen Abbildungen genannt wurden, sind jedoch i.A. keine linearen Abbildungen. Allerdings sind sie bei Darstellung in homogenen Koordinaten wieder als Matrix-Vektor-Produkt darstellbar:

$$\begin{pmatrix} & A & & t_x \\ & & & t_y \\ & & & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} v_x \\ v_y \\ v_z \\ 1 \end{pmatrix} = \begin{pmatrix} u_x + t_x \\ u_y + t_y \\ u_z + t_z \\ 1 \end{pmatrix}, \text{wobei } \mathbf{u} = \begin{pmatrix} u_x \\ u_y \\ u_z \end{pmatrix} = A \cdot \mathbf{v}$$

Es gilt: Seien A, B, C, D vier Punkte, die nicht in einer Ebene liegen („affin unabhängig“) und A', B', C', D' vier weitere Punkte. Dann gibt es genau eine affine Abbildung mit $f(A) = A', \ldots, f(D) = D'$.

Spezialfälle

Isometrien sind längentreue (und damit winkeltreue) affine Abbildungen.



Sei

$$\mathbf{v} \mapsto A \cdot \mathbf{v} + \mathbf{t}$$

eine Isometrie. Die Einheitsvektoren

$$\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

werden abgebildet auf

$$\mathbf{a_1} + \mathbf{t}, \mathbf{a_2} + \mathbf{t}, \mathbf{a_3} + \mathbf{t}$$

wobei die $\mathbf{a_i}$ die Spalten von A sind.
Da A längentreu ist, ist $\|\mathbf{a_i}\| = 1$ für $i = 1, 2, 3$, und wegen der Winkeltreue

müssen die $\mathbf{a_i}$ paarweise senkrecht aufeinander stehen, also $\mathbf{a_i} \cdot \mathbf{a_j} = 0$ für $i \neq j$ und $\mathbf{a_i} \cdot \mathbf{a_j} = 1$ für $i = j$ gelten. Damit ist

$$A \cdot A^T = I = A^T \cdot A$$

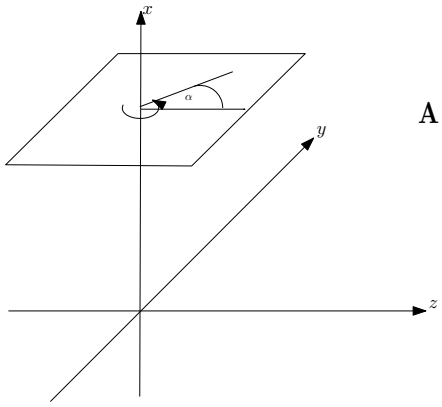
also $A^{-1} = A^T$, woraus sich

$$\det(A) = \det(A^T) = \det(A^{-1}) = \det(A)^{-1}$$

ergibt und damit $\det(A)^2 = 1$, also $\det(A) \in \{-1, 1\}$.
Wir halten fest: Sei A die zu einer Isometrie gehörige Matrix. Dann stehen die Spaltenvektoren senkrecht aufeinander und haben Länge 1 („Orthonormalsystem “) und $\det(A) \in \{-1, 1\}$.
Die Isometrien mit $\det(A) = 1$ heißen *gerade Isometrien* oder *starre Isometrien*, die mit $\det(A) = -1$ heißen *ungerade Isometrien*.

Rotationen sind spezielle starre Bewegungen. Rotation um den Winkel α gegen den Uhrzeigersinn um die z-Achse:

$$\mathbf{v} \mapsto A \cdot \mathbf{v}$$



$$\mathbf{A} := \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

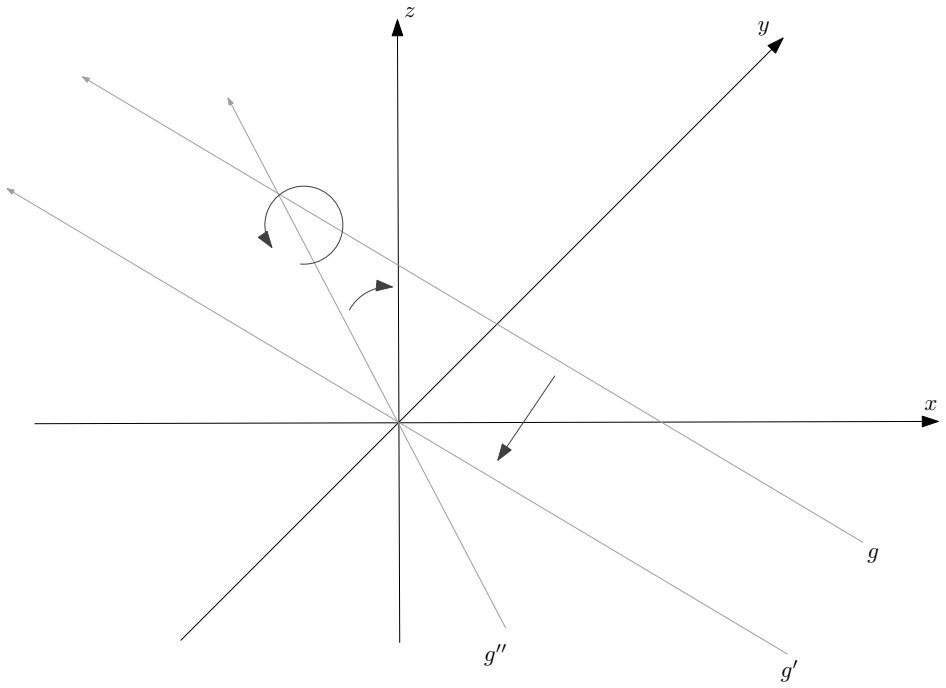
Analog für Rotationen um die x-Achse

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{pmatrix}$$

und für Rotationen um die y-Achse

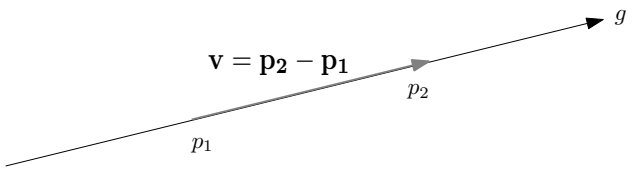
$$\begin{pmatrix} \cos\alpha & 0 & \sin\alpha \\ 0 & 1 & 0 \\ -\sin\alpha & 0 & \cos\alpha \end{pmatrix}$$

Eine Rotation um eine beliebige Gerade um einen Winkel α realisieren wir durch 5 Schritte:



1. Translation, sodaß das Bild g' von g durch den Ursprung verläuft.
2. Rotation um die x-Achse, sodaß das Bild g'' von g' in der x-z-Ebene liegt.
3. Rotation um die y-Achse, sodaß das Bild g''' von g'' gleich der z-Achse ist (orientierungserhaltend).
4. Rotation um den Winkel α um die z-Achse.
5. Schritte 3, 2, 1 wieder invertieren.

Die Rotationsachse sei gegeben als eine Gerade g durch zwei Punkte p_1, p_2 mit Orientierung von p_1 nach p_2.



$$p_i = \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix}, i = 1, 2; \mathbf{v} = \mathbf{p_2} - \mathbf{p_1}; \mathbf{u} = \frac{\mathbf{v}}{\|\mathbf{v}\|} = \begin{pmatrix} a \\ b \\ c \end{pmatrix}$$

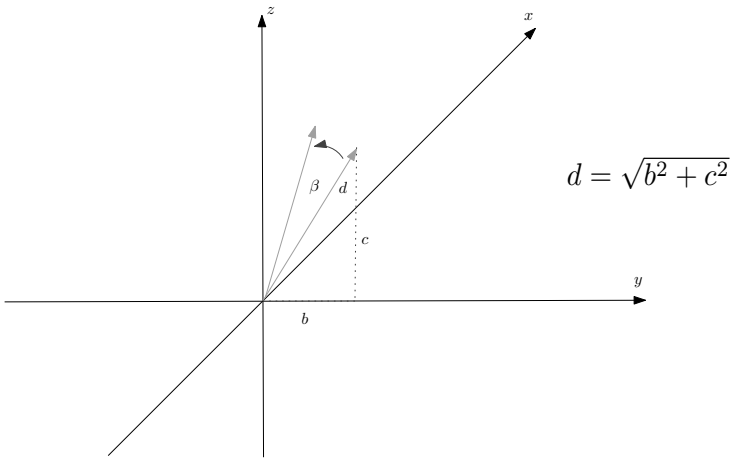
Also: $\mathbf{v}$ ist ein Richtungsvektor von p_1 nach p_2 und $\mathbf{u}$ die entsprechende normierte Version.

1. Translation T um den Vektor

$$\begin{pmatrix} -x_1 \\ -y_1 \\ -z_1 \end{pmatrix}$$

   Der Punkt p_1 wird also auf den 0-Punkt verschoben. Die so entstandene Gerade verläuft nun durch den Ursprung.

2. Rotation um die x-Achse um den Winkel β ergibt die Matrix



$$d = \sqrt{b^2 + c^2}$$

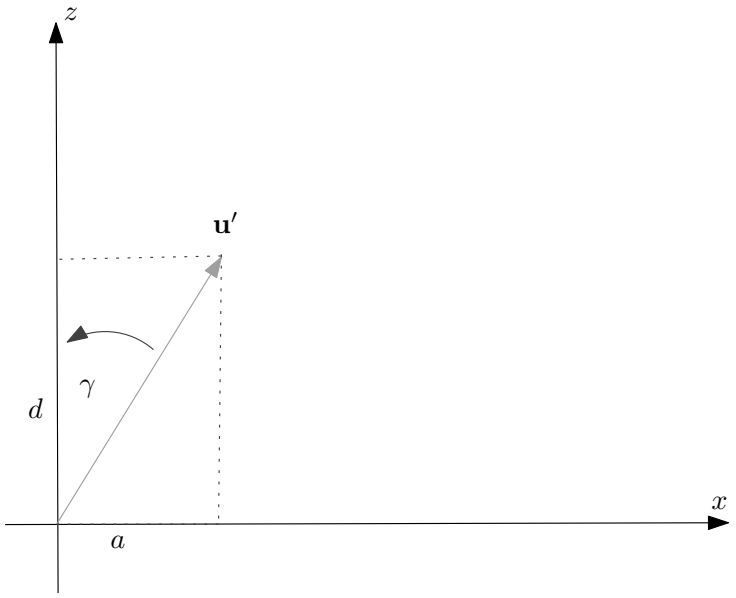
$$R_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & c/d & -b/d \\ 0 & b/d & c/d \end{pmatrix}$$

   mit $\cos\beta = c/d$ und $\sin\beta = b/d$.

3. Also haben wir nach dem zweiten Schritt die Gerade

$$\mathbf{u}' = R_x \cdot \mathbf{u} = \begin{pmatrix} a \\ 0 \\ d \end{pmatrix}$$

Nun müssen wir die entstandene Gerade noch um die y-Achse drehen, sodass sie mit der z-Achse übereinstimmt. Dabei benutzen wir, dass der Winkel γ zwischen der Geraden und der y-Achse $\cos\gamma = d$ und $\sin\gamma = a$ erfüllt:



$$R_y = \begin{pmatrix} d & 0 & a \\ 0 & 1 & 0 \\ -a & 0 & d \end{pmatrix}$$

4. Nun führen wir mit Hilfe einer Drehmatrix die schon bekannte Rotation um die z-Achse durch:

$$R_z = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

5. Invertieren wir die Schritte und multiplizieren alle entstandenen Matrizen, so haben wir die Matrix für die Drehung um die obige Gerade g.

Insgesamt haben wir also

$$\mathbf{a} \mapsto R_x^{-1} \cdot R_y^{-1} \cdot R_z \cdot R_y \cdot R_x \cdot (\mathbf{a} - \mathbf{p_1}) + \mathbf{p_2}$$

Spiegelung an der x-y-Achse ist eine Isometrie. Die x- und y-Koordinaten bleiben erhalten, die z-Koordinate wird invertiert.

$$\mathbf{v} = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix}, \mathbf{v}' = \begin{pmatrix} v_x \\ v_y \\ -v_z \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix} \cdot \mathbf{v}$$

Spiegelung an beliebiger Ebene [Selbst überlegen]

Spiegeln am 0-Punkt

$$\mathbf{v}' = \begin{pmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{pmatrix} \cdot \mathbf{v}$$

Spiegeln an der x-Achse

$$\mathbf{v}' = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{pmatrix} \cdot \mathbf{v}$$

Das entspricht einer Rotation um 180° an der x-Achse. (Nachrechnen!)

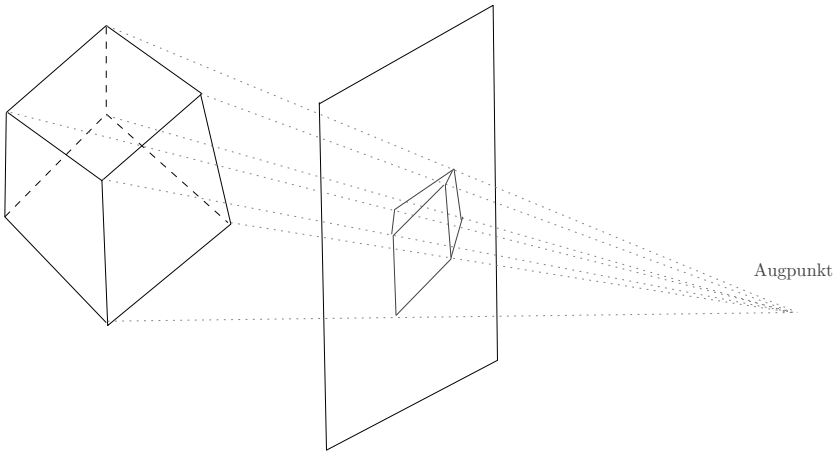
Ähnlichkeitsabbildung Diese sind zusammengesetzt aus einer Streckung am Ursprung

$$\mathbf{v} \mapsto \lambda \cdot \mathbf{v} \text{ oder } \mathbf{v} \mapsto \begin{pmatrix} \lambda & 0 & 0 \\ 0 & \lambda & 0 \\ 0 & 0 & \lambda \end{pmatrix} \cdot \mathbf{v}$$

und einer Isometrie.

2.3 Projektionen

Hier von $\mathbb{R}^3$ nach $\mathbb{R}^2$. Allgemeine Situation:



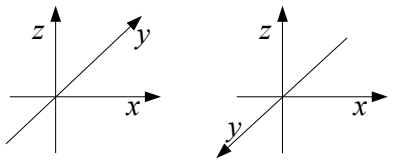
Diese Projektion heißt *Zentralprojektion* (auch perspektivische Projektion). Ein Punkt p in $\mathbb{R}^3$ wird abgebildet auf den Schnittpunkt der Geraden durch A und p mit der Bildebene E.

2.3.1 Rechtshändiges und linkshändiges Koordinatensystem

Die Koordinatenachsen im dreidimensionalen Raum lassen sich auf zwei verschieden Arten anordnen: Linkshändig und Rechtshändig (s. Abbildung 2.9). Um die Anordnung der Koordinatenachsen zur Erhalten streckt man Daumen, Zeige- und Mittelfinger einer Hand in verschiedene Richtungen. Dabei gilt folgende Zuordnung:

Daumen	x-Achse
Zeigefinger:	y-Achse
Mittelfinger	z-Achse

In der Mathematik sind rechtshändige Koordinatensysteme üblich.



(a) rechtshändig (b) linkshändig

Abbildung 2.9: Koordinatensysteme

2.3.2 Zentralprojektion

Beobachtung 2.3.1. *Bei der Zentralprojektion gilt: Je weiter Objekte von der Bildebene entfernt sind, umso kleiner ist das Bild.*

Vereinfachung: Um die Rechnungen zu Erleichtern nehmen wir folgende Vereinfachungen vor:

- Bildebene = x-y-Ebene
- Augpunkt liegt bei $(0, 0, -a)$ für eine Konstante $a > 0$.

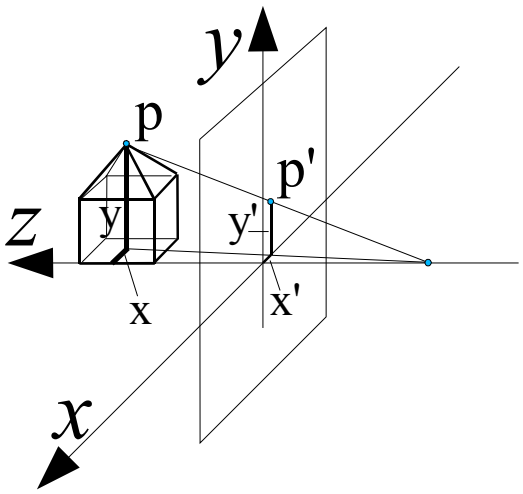


Abbildung 2.10: Zentralprojektion eines geometrischen Objektes auf die x-y-Ebene mit Augpunkt auf der z-Achse

Formeln für die (vereinfachte) Zentralprojektion: Ziel: Wir möchten zu einem gegebenen Weltpunkt $P = (x, y, z)^T$ den Punkt $P' = (x', y')^T$ (jeweils karthesische Koordinaten) in der x-y-Ebene berechnen, auf den er projeziert wird (s. auch Abbildung 2.10).

Aus dem Strahlensatz ergibt sich:

$$\frac{x'}{x} = \frac{a}{a+z} \qquad \frac{y'}{y} = \frac{a}{a+z}$$

Daraus folgt:

$$x' = \frac{a}{a+z}x \qquad y' = \frac{a}{a+z}y$$

Die Abbildung für die Zentralprojektion $\mathbb{R}^3 \to \mathbb{R}^2$ ist also:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \mapsto \begin{pmatrix} \frac{a}{a+z}x \\ \frac{a}{a+z}y \end{pmatrix}$$

Diese Abbildung ist nicht linear. Mit homogenen Koordinaten lässt sie sich

jedoch als Matrix-Vektor-Produkt darstellen:

$$\begin{pmatrix} a & 0 & 0 & 0 \\ 0 & a & 0 & 0 \\ 0 & 0 & 1 & a \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} ax \\ ay \\ a+z \end{pmatrix}$$

Definition 2.3.2 (Parallelprojektion)**.** Bei $a \to \infty$ werden die Sehstrahlen parallel und wir erhalten die Projektion

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \mapsto \begin{pmatrix} x \\ y \end{pmatrix}$$

Diese Projektion heißt *Parallelprojektion*.

Verhalten von zwei parallelen Geraden unter der Zentralprojektion: Wir betrachten nun, was passiert, wenn man auf zwei parallele Geraden g_1 und g_2 im $\mathbb{R}^3$ eine Zentralprojektion anwendet.

$$g_1 : \boldsymbol{p} + \lambda \boldsymbol{v} \qquad g_2 : \boldsymbol{q} + \lambda \boldsymbol{v} \qquad \lambda \in \mathbb{R}$$
$$\boldsymbol{p} = (p_1, p_2, p_3, 1)^T \qquad \boldsymbol{q} = (q_1, q_2, q_3, 1)^T$$

g_1 und g_2 sind parallel. Sie haben deshalb den gleichen Richtungsvektor $\boldsymbol{v}$, aber unterschiedliche Stützvektoren $\boldsymbol{p}$ und $\boldsymbol{q}$.

In homogenen Koordinaten erhalten wir die folgende Darstellung:

$$\boldsymbol{p} + \lambda \boldsymbol{v} = \begin{pmatrix} p_1 + \lambda v_1 \\ p_2 + \lambda v_2 \\ p_3 + \lambda v_3 \\ 1 \end{pmatrix} = \begin{pmatrix} p_1 \\ p_2 \\ p_3 \\ 1 \end{pmatrix} + \lambda \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ 0 \end{pmatrix}$$

$$\text{Andere Darstellung (für } \lambda \neq 0\text{):} \qquad \frac{1}{\lambda} \begin{pmatrix} p_1 \\ p_2 \\ p_3 \\ 1 \end{pmatrix} + \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ 0 \end{pmatrix}$$

Für $\lambda \to \infty$ erhalten wir $(v_1, v_2, v_3, 0)^T$, also einen unendlich fernen Punkt. Dieser ist unabhängig von p. Für g_1 und g_2 erhalten wir also jeweils den gleichen unendlich fernen Punkt: Den Schnittpunkt von g_1 und g_2.

Unter der Zentralprojektion erhalten wir:

$$\begin{pmatrix} a & 0 & 0 & 0 \\ 0 & a & 0 & 0 \\ 0 & 0 & 1 & a \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ 0 \end{pmatrix} = \begin{pmatrix} av_1 \\ av_2 \\ v_3 \end{pmatrix}$$

Für $v_3 \neq 0$ (d.h. der Vektor liegt nicht in der x-y-Ebene) ist das ein endlicher Punkt mit karthesischen Koordinaten

$$\begin{pmatrix} a\frac{v_1}{v_3} \\ a\frac{v_2}{v_3} \end{pmatrix}$$

Unter bestimmten Umständen werden also unendlich ferne Punkte von der Zentralprojektion auf endliche Punkte abgebildet. Diese Bilder der unendlich fernen Punkte werden auch *Fluchtpunkte* genannt und spielen beispielsweise in der Malerei eine wichtige Rolle.

Es gibt Projektionen mit einem Fluchtpunkt (s. Abbilundg 2.11), mit zwei Fluchtpunkten (s. Abbilundg 2.12) und sogar welche mit drei Fluchtpunkten Diese nennt man auch 1-, 2-, und 3-Punkt-Projektionen.

Die Allgemeine Zentralprojektion mit beliebigem Augpunkt und beliebieger Bildebene wird in den Übungen behandelt (4. Übungszettel, Aufgabe 1).

2.3.3 Parallelprojektion

Definition 2.3.3. Eine Parallelprojektion heißt orthogonal, wenn alle Sehstrahlen senkrecht zur Bildebene stehen.

Bemerkung. Bisher haben wir uns mit einer orthogonalen Parallelprojektion beschäftigt, da die Bildebene die x-y-Ebene und der Augpunkt der unendlich ferne Punkt auf der z-Achse war.

Bemerkung. Orthogonale Parallelprojektionen werden in der Praxis tatsächlich benutzt, z.B. bei technischen Zeichnungen.

Bezeichnungen

Parallelprojektion in	x-y-Ebene:	Grundriss
	y-z-Ebene:	Aufriss
	x-z-Ebene:	Seitenriss

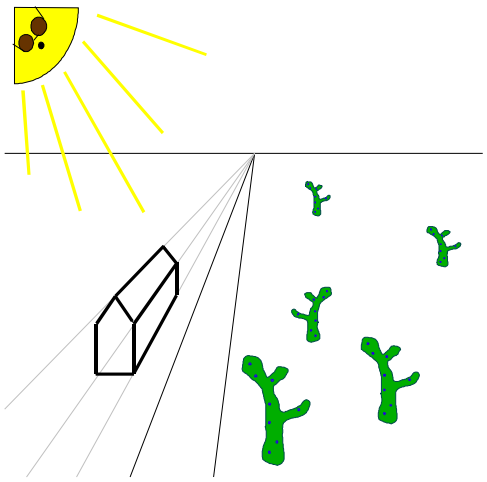


Abbildung 2.11: Einpunktprojektion: Der Rand der Straße und die Verlängerungen der Hausmauerkanten treffen sich an einem unendlich fernen Punkt. Dieser liegt auf der unendlich fernen Gerade (dem Horizont)

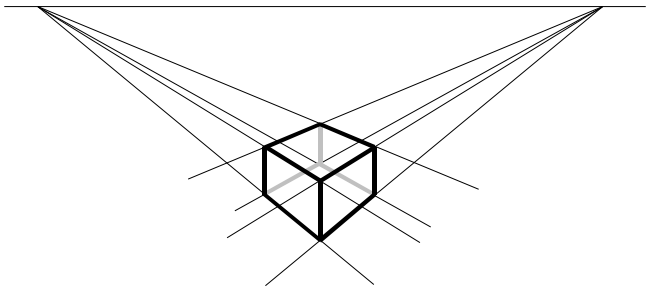
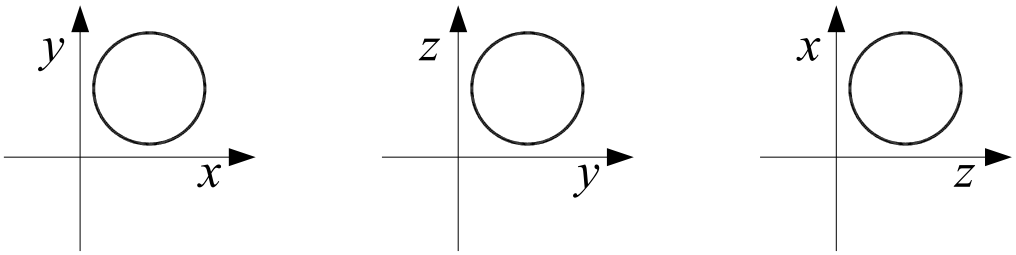


Abbildung 2.12: Zweipunktprojektion eines Würfels

Bemerkung. Die drei Projektionen genügen nicht um ein Objekt zu rekonstruieren. Die Projektionen in Abbildung 2.13 könnten sowohl von einer Kugel, als auch von dem Schnitt von 3 Zylindern (je einer mit Mittelachse in x-, y- und z-Richtung) stammen.



(a) Grundriss (b) Aufriss (c) Seitenriss

Abbildung 2.13: Die orthogonalen Parallelprojektionen einer Kugel bzw. des Schnittes von 3 Zylindern in die Koordinatenebenen

Definition 2.3.4. Eine Parallelprojektion bei der Bildebene und Sehstrahlen nicht senkrecht zueinander stehen heißt *schiefe Parallelprojektion* oder *axonometrische Projektion.*

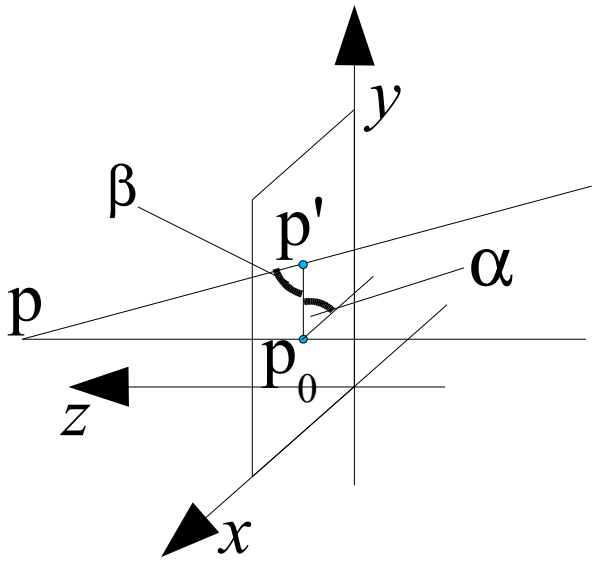
Formeln für die schiefe Parallelprojektion: Sei $P = (x, y, z)$ ein Punkt im $\mathbb{R}^3$. Wir möchten nun dessen Bild $P' = (x', y')$ unter der (schiefen) Parallelprojektion bestimmen. (s. Abb. 2.14).

P_0 sei der Punkt in der x-y-Ebene, den man erhält, wenn man auf P die orthogonale Parallelprojektion anwendet, also $P_0 = (x, y, 0)$. Die Sehstrahlen treffen unter den Winkeln α und β auf die x-y-Ebene. Dabei ist β der Winkel zwischen den Strecken PP' und $P'P_0$ und α der Winkel zwischen der Strecke $P'P_0$ und einer zur x-Achse parallelen Geraden, die durch P_0 verläuft (s. Abbildungen 2.14b u. 2.14c). Die Länge der Strecke $P'P_0$ bezeichnen wir mit L.

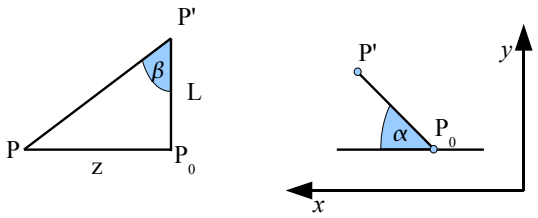
Aus der Definition des Kotangens folgt:

$$\frac{L}{z} = \cot \beta =: d$$

Da $z \cdot d = L$ ergibt sich nun aus der Definition des Sinus und des Kosinus:



(a) 3D-Ansicht



(b) Ebene durch P, P' und P_0 mit Winkel β

(c) x-y-Ebene mit Winkel α

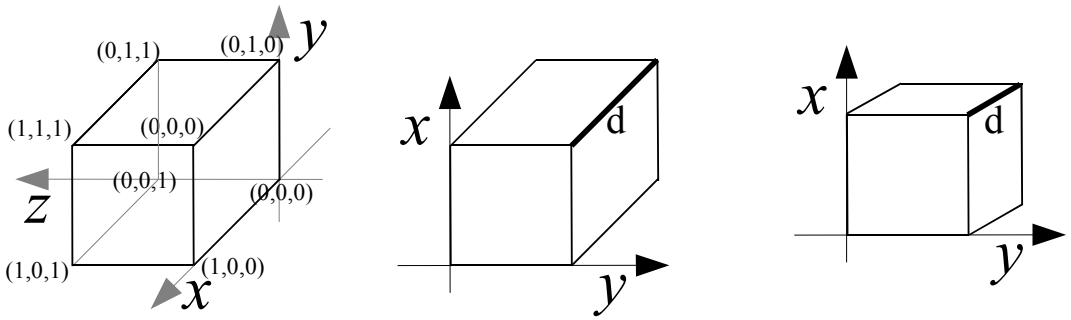
Abbildung 2.14: Schiefe Parallelprojektion

$$x' = x + z \cdot d \cdot \cos\alpha \qquad y' = y + z \cdot d \cdot \sin\alpha$$

Damit lässt sich die Abbildung in homogenen Koordinaten als Matrix-Vektor-Produkt schreiben:

$$\begin{pmatrix} 1 & 0 & d\cos\alpha & 0 \\ 0 & 1 & d\sin\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix}$$

Definition 2.3.5. Ist $d = 1$, also $\beta = 45°$ und $\alpha = 45°$, so spricht man von der *Kavalliersperspektive*.
Ist $d = \frac{1}{2}$, also $\beta = \arctan 2 \approx 63.5°$ und $\alpha = 30°$, so spricht man von der *Kabinettsperspektive* (s. Abbildung 2.15).

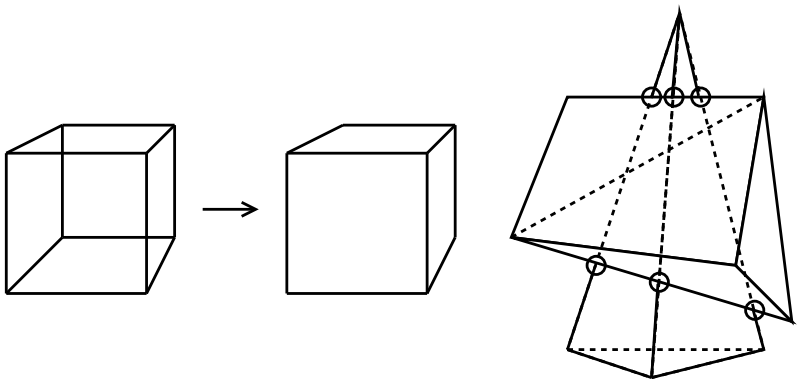


(a) 3D-Objekt (b) Kavalliersperspektive (c) Kabinettsperspektive

Abbildung 2.15: 3D-Objekt in verschiedenen Parallelprojektionen

Kapitel 3

Entfernen von verdeckten Kanten und Flächen



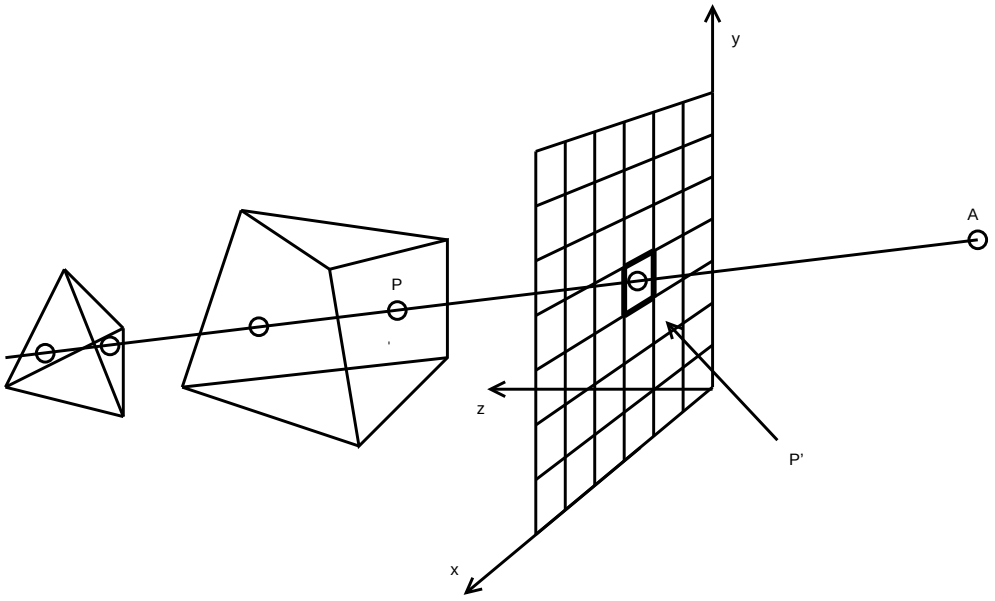
Annahmen Die Objekte der Szene sind als Volumen- oder Flächenmodelle gegeben (kein Drahtgittermodell). Die Flächen sind undurchsichtig (nicht transparent), und haben verschiedene Farben (oder Grauwerte).

3.1 Tiefenpufferalgorithmus (Z-Buffer)

Voraussetzungen: Verwenden von Zentral- oder Parallelprojektion. Das Ausgabemedium besteht aus Pixeln (Bildpunkten).

Problem: Welche Farbe soll ein Pixel P' des Ausgabemediums (üblicherweise der Bildschirm) bekommen?

Ansatz: Die Farbe derjenigen Fläche F, die als erste vom Strahl $\overrightarrow{AP'}$ getroffen wird. Der Schnittpunkt P des Strahls mit F hat von allen Schnittpunkten des Strahls mit den Flächen der Szene die kleinste z-Koordinate. Falls kein solcher Punkt existiert, nimmt man für P' die vorher definierte Hintergrundfarbe.



Aus den Überlegungen ergibt sich nun folgender Algorithmus:

Algorithm 1: Tiefenpufferalgorithmus

Daten:
`// momentane Farbe des Pixels` (x', y')
Array Farbe[x',y']
`// min. z-Koordinate der bisher untersuchten Schnittpunkte`
Array Tiefe[x',y']
für alle *Pixel* $P' = (x', y')$ *des Bildschirms* **tue**
  Tiefe[x',y']:= ∞
  Farbe[x',y']:= Hintergrundfarbe
  für alle *Flächen* F *in der Szene* **tue**
    Bestimme den Schnittpunkt $P = (x, y, z)$ von $\overrightarrow{AP'}$ mit F
    wenn P *existiert und* $0 \leq z <$Tiefe[x',y'] **dann**
      Tiefe[x',y']:= z
      Farbe[x',y']:= Farbe von F
    Ende
  Ende
Ende

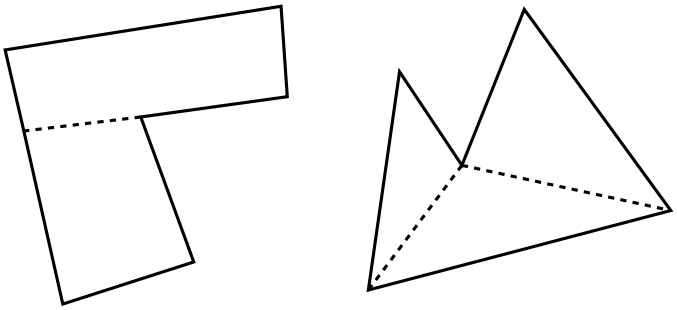
Laufzeitanalyse: Die Laufzeit ist $O(g \cdot n)$, mit g = Anzahl der Pixel des Ausgabemedium und n = Anzahl der Flächen der aktuellen Szene.

Beispiel: Rendern einer Szene eines 3D Spiels in der Auflösung 1280x1024. Die Szene enthält ca. 10000 Flächen. Das sind dann also ca. 13 Milliarden Rechenoperationen pro Bild. Das ist natürlich zu viel!

Algorithm 2: verbesserter Tiefenpufferalgorithmus

Daten:
`// momentane Farbe des Pixel` (x', y')
Array Farbe[x',y']:= Hintergrundfarbe
`// min. z-Koordinate der bisher untersuchten Schnittpunkte`
Array Tiefe[x',y']:= ∞
für alle *Flächen F in der Szene* **tue**
  Schritt A: Bestimme den Bereich F' des Bildschirms, auf den F abgebildet wird
  Schritt B:
  für alle *Pixel* $P' = (x', y') \in F'$ **tue**
    Bestimme den Schnittpunkt $P = (x, y, z)$ von $\overrightarrow{AP'}$ mit F.
    wenn P *existiert und* $0 \leq z <$Tiefe[x',y'] **dann**
      Tiefe[x',y']:= z
      Farbe[x',y']:= Farbe von F
    Ende
  Ende
Ende

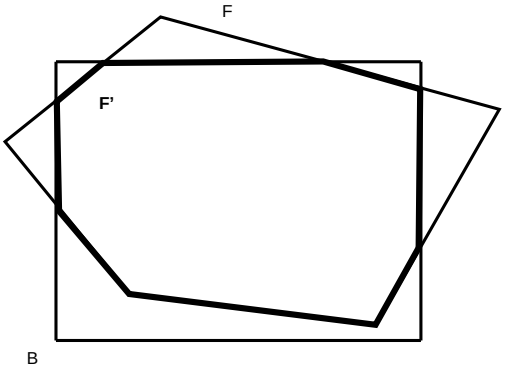
Einzelheiten: Bei einer Szene aus Polyedern (die Flächen sind also einfache Polygone) empfiehlt sich das Zerlegen in konvexe Polygone, oder sogar Dreiecke (Triangulierung). Dies ist für jedes einfache Polygon möglich.



Wir können also annehmen: Jede Fläche F ist ein konvexes Polygon.

Schritt A F' bestimmen:

1. Projiziere F auf die Bildebene. Das dabei entstehende Polygon P sei gegeben durch die Folge seiner Ecken.
2. Abschneiden (engl. clipping) von P an den Rändern des Bildbereiches B.



Überprüfe für jede Kante von P, ob sie

- ganz außerhalb von B liegt,
- ganz innerhalb von B liegt, oder
- teilweise außen, teilweise innen von B liegt.

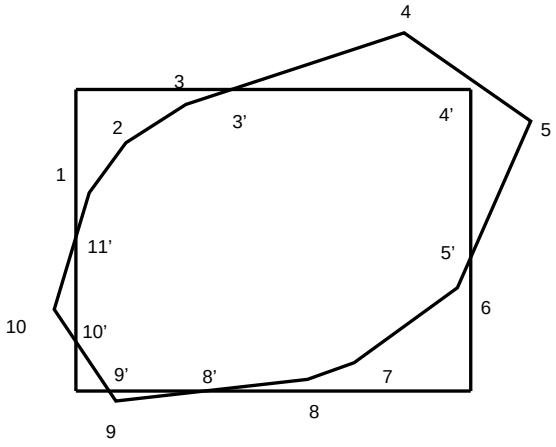
Beispiel: Folge der Ecken von P sei:

$$\underbrace{1\ 2\ 3}_{innen}\ 4\ 5\ \underbrace{6\ 7\ 8}_{innen}\ 9\ 10$$

Durch das Abschneiden erhält man ein Polygon F' mit Ecken

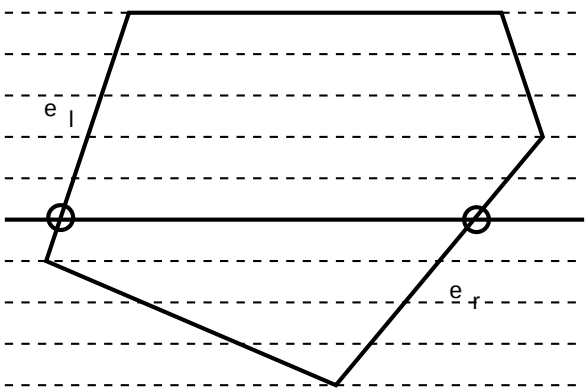
$$1\ 2\ 3\ 3'\ 4'\ 5'\ 6\ 7\ 8\ 8'\ 9'\ 10'\ 11'.$$

Dabei wurden die Punkte außerhalb von B entweder durch Schnittpunkte oder Bildschirmecken ersetzt.



Schritt B Besuchen der Pixel von F': Beginne beim untersten Punkt von F' (kleinste y-Koordinate), und besuche die Pixel zeilenweise.

- Erstelle Listen L_l und L_r der linken und rechten Kanten von F', geordnet von unten nach oben.
- Für jede Zeile betrachte alle Pixel x, die rechts von e_l und links von e_r liegen. Dabei bezeichnen e_l und e_r die momentan aktuellen Kanten. Beim Übergang zur nächsthöheren Zeile ist außerdem festzustellen, ob die Endpunkte von e_l bzw. e_r überschritten wurden. Sollte dies der Fall sein, so müssen e_l bzw. e_r entsprechend aktualisiert werden.



Bemerkung. Da alle Flächen eben sind, ist die z-Koordinate des Schnittpunkts von F mit $\overrightarrow{AP'}$ linear in den Bildschirm-Koordinaten x' und y'. Mit $\Delta z := z(1, y')$ ist also

$$\begin{aligned} z_0 &= z(x', y') \\ z_1 &= z(x' + 1, y') = z(x', y') + z(1, y') = z(x', y') + \Delta z, \end{aligned} \tag{3.1}$$

Dabei ist Δz eine Konstante, die nur von der aktuellen Fläche F abhängt. Es ist somit möglich, die Schnittpunkte beim Durchlaufen der Zeile durch Addition mit einer Konstanten zu aktualisieren, was viel weniger rechenaufwendig ist, als ein Gleichungssystem zu lösen.

Bemerkung. Der oben beschriebene Algorithmus funktioniert im wesentlichen auch für gekrümmte Flächen. Allerdings können sich hier Darstellungsfehler ergeben, die auf Rundungsfehler zurückzuführen sind.

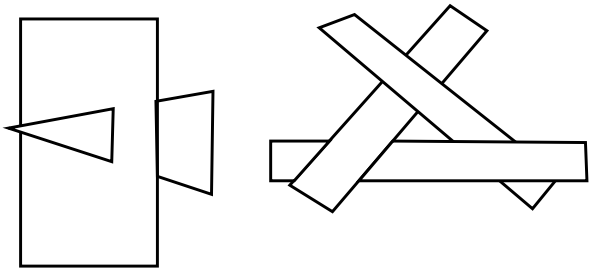
3.2 „Maleralgorithmus“ (Painter’s Algorithm)

Idee: Ordne die Flächen der Szene nach Tiefe. Zeichne diese nun nacheinander auf den Bildschirm, angefangen mit der tiefsten (am weitesten entfernten) Fläche. Überschreibe dabei die vorherigen Farbwerte.

Vorteile: Das Feld „Tiefe“ und der Test auf Tiefe werden nicht mehr benötigt.

Nachteile: Im Allgemeinen können Flächen nicht der Tiefe nach angeordnet werden. Sich schneidene Flächen müssten daher an Schnittlinien zerteilt werden. Außerdem können sich mehrere Flächen gegenseitig überdecken

(s.u.). In der Praxis muß man daher bei der Erstellung von Szenen darauf achten, diese Problemfälle zu vermeiden.



3.3 BSP - Binäre Raumaufteilung (binary space partition)

Bei BSP wird eine Ebene (bzw. ein d-dimensionaler Raum) durch eine Gerade (bzw. eine Hyperebene) in zwei Halbebenen (bzw. Halbräume) geteilt, die dann einzelnen für sich betrachtet werden. Ist der enthaltene Teil der Szene noch zu *kompliziert* wird der Raum weiter geteilt bis die Teile der Szene *einfach* sind.

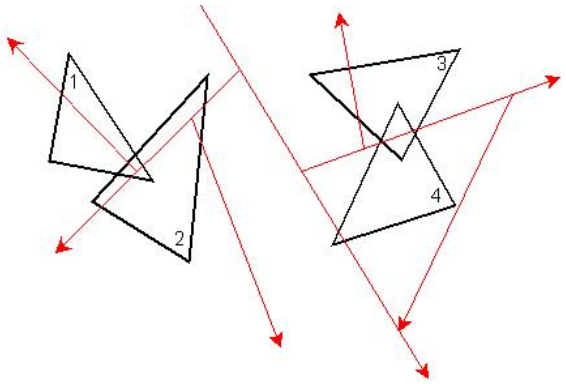


Abbildung 3.1: BSP

Die Aufteilung kann durch einen Baum illustriert werden. In der folgenden Abbildung stehen die Zahlen an den Knoten für die Polygone, die sich zumindest teilweise in der jeweiligen Halbeben befinden. Rechts von der ersten aufteilenden Gerade in der obigen Abbildung befinden sich z.B die Polygone 1,2 und 4. Die Aufteilung endet wenn in einer Halbebene nur noch ein oder kein Polygon mehr ist. Die Szene ist dann *einfach*:

Eine Szene wird als *einfach* bezeichnet wenn:

- alle Polynome vollständig außerhalb sind
- alle Polynome bis auf eines ist vollständig außerhalb
- ein Polynom überdeckt den ganzen Bereich und liegt **vor** allen anderen

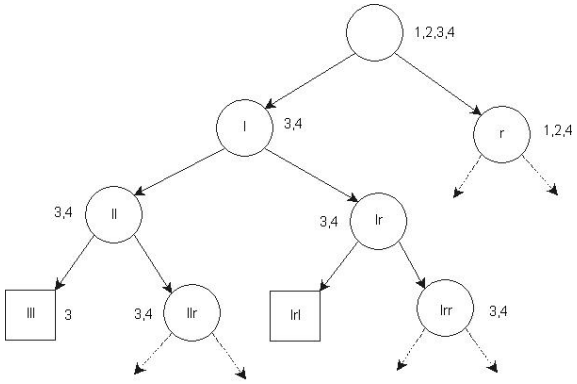


Abbildung 3.2: Baumdarstellung

Dabei kann das letzte Kriterium weggelassen werden. Jedoch muss dann die Szene auch mehrfach geteilt werden. Das Kriterium kann bleiben, sobald zwischen den Polynomen eine Reihenfolge herrscht.

Unter Umständen könnte der Algorithmus nicht terminieren, sodass in der Praxis abgebrochen wird, wenn die Größe eines Bereichs (konvexes Polygon) kleiner oder gleich einem Pixel ist.

3.3.1 Wahl der aufteilenden Geraden

Eine einfache Methode zum Setzen der aufteilenden Geraden ist die Szene abwechselnd durch horizontale und vertikale Geraden zu teilen. Dabei gehen diese Geraden stets von der Mitte des zu teilenden Fensters aus. Die Teile der Szene werden so lange geteilt, bis sie *einfach* sind, wobei zwei Rekursionsschritte einer Ebene eines **Quadtree** entsprechen.

Eine andere Methode zur Aufteilung wäre, die Geraden entlang der Polygonseiten zu legen. Dadurch terminiert der Algorithmus schneller.

Zum Schluss werden die einfachen Bereiche dann auf dem Bildschirm ausgegeben.

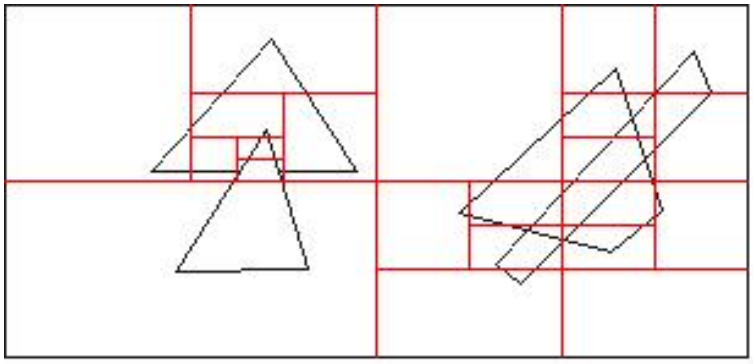


Abbildung 3.3: Aufteilung durch vertikale/horizontale Geraden

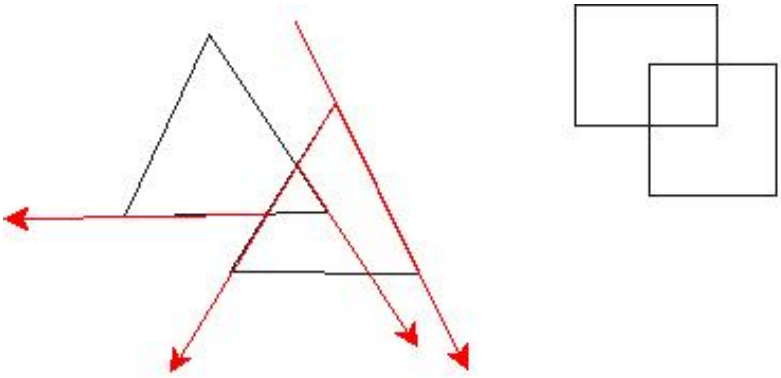


Abbildung 3.4: Aufteilung entlang der Kanten

3.4 Sweepline-Verfahren (scan line)

Der Sweepline-Algorithmus ist ein Bildraumalgorithmus, das heißt er operiert auf der Projektion der Szene ins 2-dimensionale. Die Idee ist eine vertikale Linie, die *Sweepline*, über die Szene zu schicken und sich die Schnittpunkte der Polygone mit der Sweepline in einer geordneten Liste zu merken. Man *fegt* also mit einer vertikalen Gerade von links nach rechts über die Szene.

An gewissen Stellen, den *Ereignispunkten*, muss die Liste aktualisiert werden. Dazu werden die folgenden beiden Datenstrukturen verwendet:

- **SLS** (Sweepline status):
  Die Datenstruktur für die momentane Konfiguration auf der Sweepline.

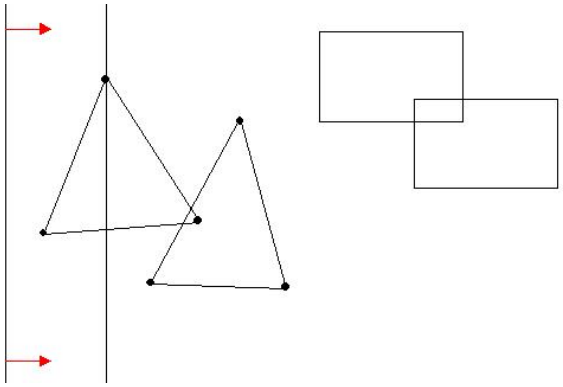


Abbildung 3.5: Sweepline-Verfahren

- **EPS** (Event point scedule):
  Enthält die Ereignispunkten und wird ständig aktualisiert.

Nach der Initialisierung ist SLS leer und in EPS befinden sich die Endpunkte der Kanten der zu bearbeitenden Polygone. Beim Durchlauf enthält die SLS die geordnete Folge der Kanten die gerade geschnitten werden. Die Kanten sind gerichtet, sodass das begrenzte Polygon **rechts** von der Kante liegt. Die geordnete Liste der Flächen hinter einem Intervall nennen wir hier $FL(s)$.

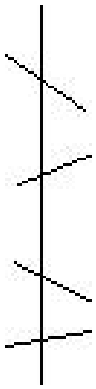


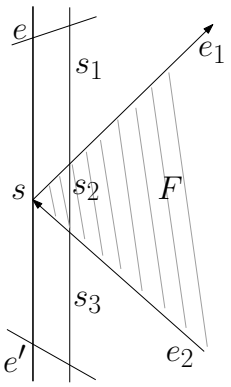
Abbildung 3.6: Intervall s

3.4.1 Typen von Ereignispunkten

Wir betrachten vorerst nur die Ereignoistypen für sich nicht durchdringende Flächen.

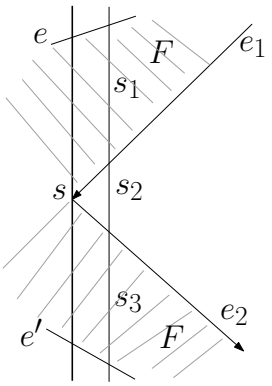
Endpunkte von Kanten

a) Anfang einer Fläche:



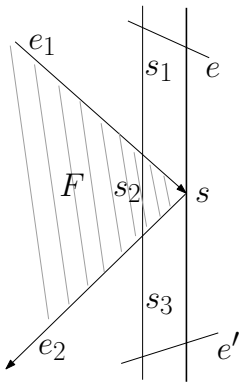
- Füge e_1 und e_2 in SLS ein; s sei das Intervall in das sie eingefügt wurden.
- Dadurch wird s aufgeteilt in die drei neuen Intervalle s_1, s_2 und s_3.
- $FL(s_1) \leftarrow FL(s)$;
- $FL(s_2) \leftarrow FL(s) \cup F$ an der richtigen Stelle eingefügt
- $FL(s_3) \leftarrow FL(s)$;
- Falls F die oberste Fläche ist werden e_1 und e_2 als *sichtbar* markiert („Flag“ für jede Kante).
- Die Schnittpunkte von e und e_1 sowie e' und e_2 werden, falls existent, als Ereignispunkte in EPS eingefügt.

b) Anfang eines Splits



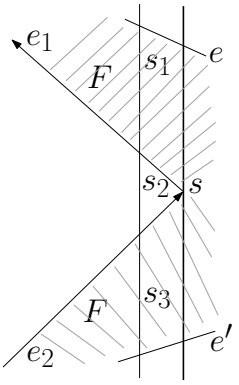
- Eine Fläche endet in P
- $FL(s_1) \leftarrow FL(s)$;
- $Fl(s_2) \leftarrow Fl(s)\ \{F\}$
- $FL(s_3) \leftarrow FL(s)$
- Rest ist analog zu a)

c) Ende einer Fläche



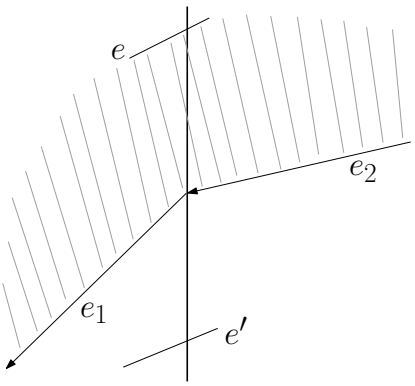
- SLS $\leftarrow$ SLS $\setminus \{e_1, e_2\}$
- teste rechts von Sweepline ob Schittpunkt von (e, e') existiert und füge ihn (falls er existiert) in EPS ein
- $Fl(s) \leftarrow Fl(s_1)$
- falls e_1 sichtbar war zeichne e_1 vom (gemerkten) sichtbaren Anfangspunkt

d) Ende eines Splits



– Gleiches wie bei Fall c)

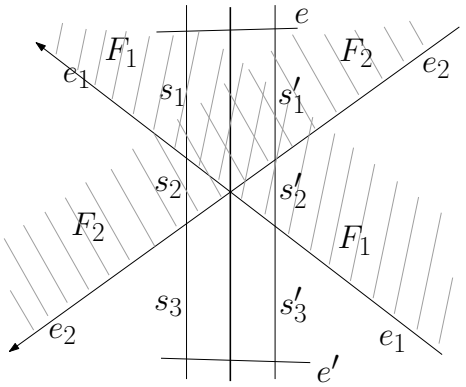
e) Knick



– Füge Schnittpunkt von e und e_2 in EPS ein, falls dieser existiert.
– Füge Schnittpunkt von e' und e_2 in EPS ein, falls dieser existiert.
– Falls e_1 die oberste Kante ist, so wird diese gezeichnet

Schnittpunkte von Kanten

a) Schnitt zweier Flächen



- $FL(s_1') \leftarrow FL(s_1)$;
- $FL(s_2') \leftarrow FL(s_2) \setminus \{F_2\} \cup \{F_1\}$
- $FL(s_3') \leftarrow FL(s_3)$;
- tausche Reihenfolge von e_1 und e_2 in SLS
- berechne Schnittpunkt von e_2 und e sowie von e_1 und e' und füge sie (falls existent) in EPS ein
- teste ob e_1 sichtbar wird, falls ja markieren und Punkt merken
- falls e_2 sichtbar war und jetzt unsichtbar wird, zeichne entsprechenden Teil von e_2

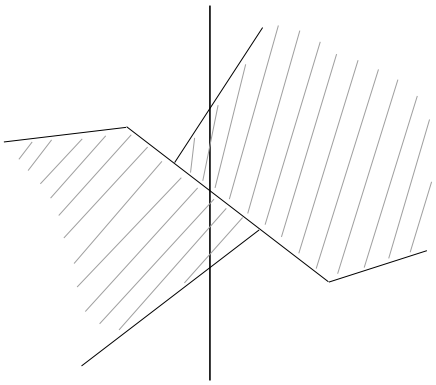
- andere Fälle werden in einer Übungsaufgabe behandelt

3.4.2 Gesamtalgorithmus

- Initialisierung SLS $\leftarrow \varnothing$
- EPS $\leftarrow$ Endpunkte von Kanten im Bildraum
- $Fl(s) \leftarrow H$ H - Hintergrund, Ebene hinter der ganzen Szene
- Abarbeitung aller Eventpunkte mit entsprechenden Operationen
- am Schluss werden alle sichtbaren Teile von Kanten gezeichnet

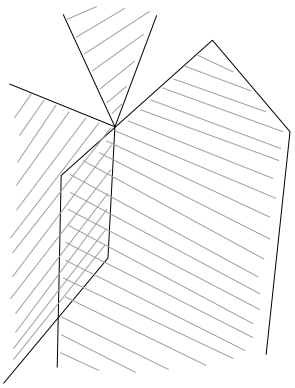
3.4.3 Entartete Fälle

a) Kante gehört zu mehr als einer Fläche



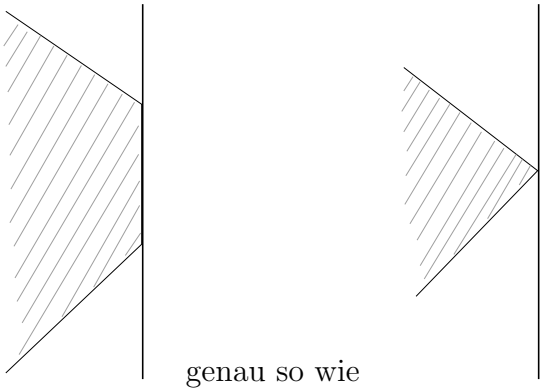
– Kanten eventuell doppelt (oder mehrfach) im SLS, dazwischen Segment der Länge 0

b) Punkt inzident zu mehr als 2 Kanten



– bei einem solchen Ereignispunkt alle von links kommenden Kanten aus dem SLS nehmen und nach rechts abgehende in der richtigen Reihenfolge (Steigung) wieder einfügen

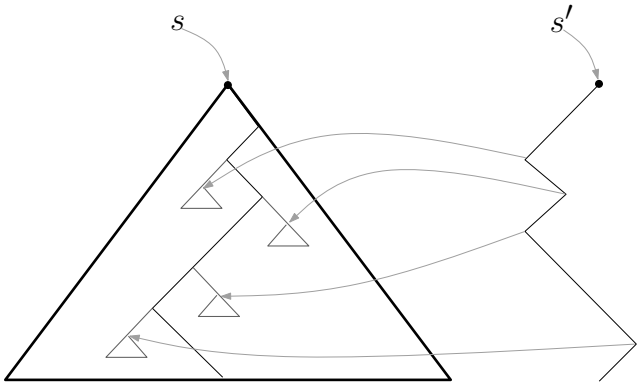
c) Kante parallel zur Sweepline



3.4.4 Laufzeit des Algortihmus

insgesamt n Kanten, k Schnittpunkte von Kanten (im Bildraum)
$2n + k$ Ereignispunkte
Für jeden konstant viele Operationen der Form

1. in SLS: suchen, einfügen, streichen $\longmapsto$ (bal.) Suchbäume $O(\log n)$ (eigentlich $O(\log(2n + k))$
2. in EPS: Minimum streichen, einfügen $\longmapsto$ Heap $O(\log n)$
3. in $Fl(s)$: kopieren, einfügen, streichen $\longmapsto$ (bal.) Suchbäume Zeiger auf Wurzel



kopieren:

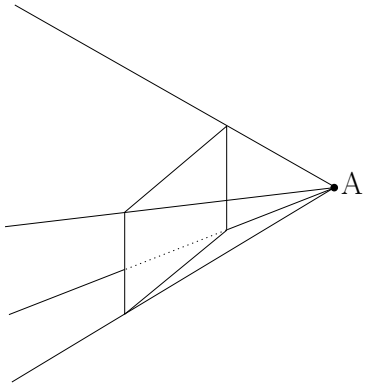
- neuer Zeiger auf die Wurzel
- einfügen oder streichen in s, nicht in s'

- kopiere Weg entlang dessen sich etwas ändert, von dort aus Zeiger auf Teilbäume der alten Strukturen $O(\log(n))$ pro Operation

insgesamt $O((n+k)\log n)$

kann benutzt werden zum Entfernen verdeckter Flächen
sweepline $\Leftrightarrow$ 1 Zeile der Bildschirms
Farbe der obersten Fläche von Fl(s)

3.4.5 Heuristiken zur Vereinfachung



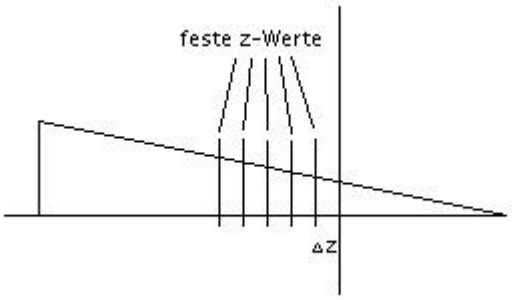
Bildschirm

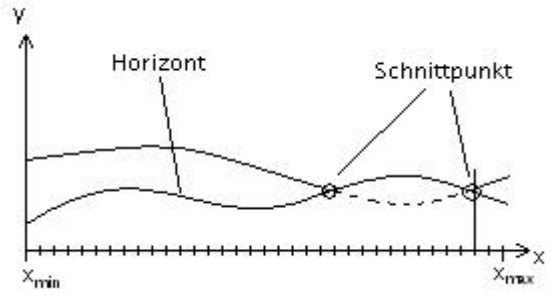
1. die Objekte in der 3D-Szene außerhalb dieses Bereichs sind sowieso nicht sichtbar: also entfernen

2. „Nach hinten“ orientierte Flächen kann man ignorieren.
   Dies ist der Fall wenn der Normalenvektor der Fläche in positive z-Richtung zeigt (bei einer Szene von Polyedern).

3.5 Methode des gleitenden Horizonts

Für Flächen die durch eine Gleichung $F(x, y, z) = 0$ gegeben sind, lässt sich die Methode des gleitenden Horizonts anwenden. Dabei werden – die sichtbaren Teile – der Schnitte der Fläche mit parallelen Ebenen gezeichnet. Für Ebenen parallel zur x-y-Ebene sind dies Kurven der Form $F(x, y, z_i) = 0$ Für Flächen F(x, y, z) = 0 Kurzum für feste z-Werte - hier: f(y) = (x, z)





Werte z_1 ... z_n
Für i = 1 ... n
    Für $x = x_{min}$ ... x_{max}
        Berechne f(x, z)
        Falls x >HMAX[x]
            Wenn Punkt (x, y) sichtbar, dann verbinde mit Vorgänger
                Falls Vorgänger nicht sichtbar,
                berechne Schnittpunkt und verbinde mit diesem
            Aktualisiere HMAX[x]

Kapitel 4

Beleuchtung und Schatten

Es gibt u.A. folgende mögliche Lichtquellen:

- diffuses (ambientes) Licht: richtungsloses Licht
- paralleles Licht: Licht von sehr weit entfernten Ojekten, z.B. Sonnenlicht
- Licht aus Punktquellen: radiales Licht

Des weiteren wird Licht reflektiert, gebrochen, gestreut.

Im Computerbereich gibt es Modelle, welche teilweise nicht den physikalischen Gesetzen entsprechen, die sich aber in der Computergrafik bewährt haben.

4.1 Betrachtung von einfarbigem Licht

Die Helligkeit eines Bildpunktes wird durch die Intensität I des Lichts ausgedrückt.

1. Intrinsisches, immamentes Licht:
   Einfachstes Lichtmodell zum Darstellen von Objekten – die Objekte strahlen mit eigener Intensität.

$$I = k_i$$

k_i: objekteigene konstante Intensität

Wird im Folgenden aber nicht weiter betrachtet.

2. Ambiente Reflexion:
   Diffuses (richtungsloses) Licht fällt auf die Objekte und wird von der Oberfläche in alle Richtungen gleichermaßen reflektiert. Man bezeichnet dies auch als indirekte Beleuchtung oder Hintergrundbeleuchtung.

$$I = I_a * k_a \qquad k_a \in [0, 1]$$

   I_a: Intensität des ambienten Lichts

   k_a: ambienter Reflexionskoeffizient

3. Diffuse Reflexion
   Gerichtetes Licht (von einer punktförmigen Lichtquelle oder paralleles Licht) fällt auf die Objekte und wird von der Oberfläche in alle Richtungen gleichermaßen reflektiert.

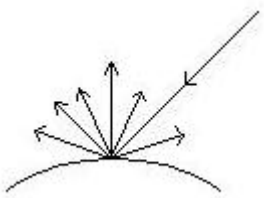
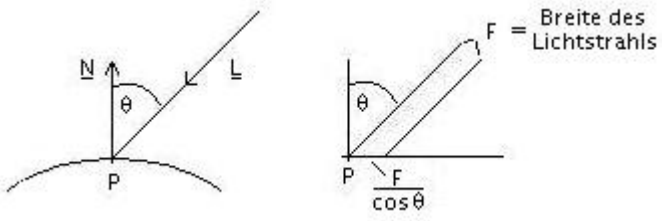


Abbildung 4.1: Entstehung diffuser Reflexion

Die Lichtintensität in P ist proportional zum Kosinus des Einstrahlwinkels Θ - bei Verkleinerung des Einstrahlwinkels verringert sich die Größe der bestrahlten Fläche. Bei selbem Licht hat man demzufol-



ge eine größere Intensität bei kleinerem Einstrahlwinkel und kleinerer Fläche, als bei größerem Winkel und größerer Fläche.

N: Einheitsnormalenvektor im Punkt P

L: Einheitsvektor des einfallenden Lichts

Θ: Einstrahlwinkel zwischen N und L

$$I = I_p * k_d * \underbrace{\cos\Theta}_{0 \quad wenn \quad -90° > \Theta > 90°}$$

$$I = I_p * k_d * < N, L >$$

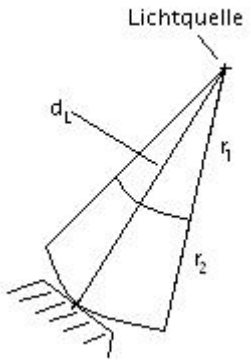
I_p: Intensität der Lichtquelle

k_d: diffuser Reflexionskoeffizient

Punktförmige Lichtquellen strahlen kein paralleles Licht aus. Mit zunehmender Entfernung schwächt sich die Intensität um einen Abschwächungsfaktor f_{ab} ab.

$$I = I_p * k_d * f_{ab} * < N, L >$$

Aber wie "wählt" man f_{ab}?



Eine Möglichkeit wäre z.B. $f_{ab} = 1/d_L^2$ mit d_L = Entfernung zur Lichtquelle.

Doch die erzielten Ergebnisse überzeugen nicht. Mit

$$f_{ab} = max(\frac{1}{c_1 + c_2 * d + c_3 * d^2}, 1)$$

erzielt man bessere Ergebnisse, wengleich man in jedem Falle c_1, c_2 und c_3 immer so wählen sollte, dass in etwa das gewünschte Ergebnis erzielt wird.

Ein weiterer Aspekt, den man beachten könnte, ist atmosphärischer Abschwächung - schwächer werdendes Licht bei zunehmendem Abstand des Betrachters vom Objekt.

4. Ambiente und diffuse Reflexion:

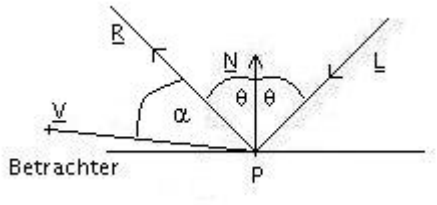
   Die Kombination beider Reflexionen, welche dem "natürlichen Vorkommen" bereits näher kommen.

$$I = I_a * k_a + I_p * k_d * < N, L >$$

$$I = I_a * k_a + I_p * k_d * f_{ab} * < N, L >$$

5. Spiegelnde Reflexion:

   Licht wird in verschiedenen Richtungen verschieden stark reflektiert - "gespiegelt". Am stärksten in Richtung R.



   R: Einheitsvektor des reflektierten Lichts

   V: Einheitsvektor des Beobachters bezogen auf den Punkt P

   α: Ausstrahlwinkel zwischen R und V

4.1.1 Modell von Phong (1975)

Von Phong Bui-Tuong stammt ein weit verbreitetes Beleuchtungsmodell für nicht perfekte Reflektoren (z.B. den Apfel). Dabei wird davon ausgegangen, daß die gerichtete Reflexion bei α=0 maximal ist und mit steigendem α rapide abnimmt. Diese rapide Abnahme wird durch $cos^n\alpha$ approximiert. n ist ein Materialkoeffizient, dessen typische Werte zwischen 1 und mehreren Hundert liegen. Kleine Werte erzeugen eine langsame Intensitätsabnahme, große Werte simulieren ein kleines, scharfes Glanzlicht. Die Intensität des beim Betrachter ankommenden Lichts ist proportional zu $\cos^n \alpha$.

Hier nun das Zusammenspiel aller drei Reflexionsarten:

$$I = I_a * k_a + f_{ab} * I_p * (k_d * \cos\Theta + k_s * \cos^n \alpha)$$

$$I = I_a * k_a + f_{ab} * I_p * (k_d * < N, L > + k_s * < R, V >^n)$$

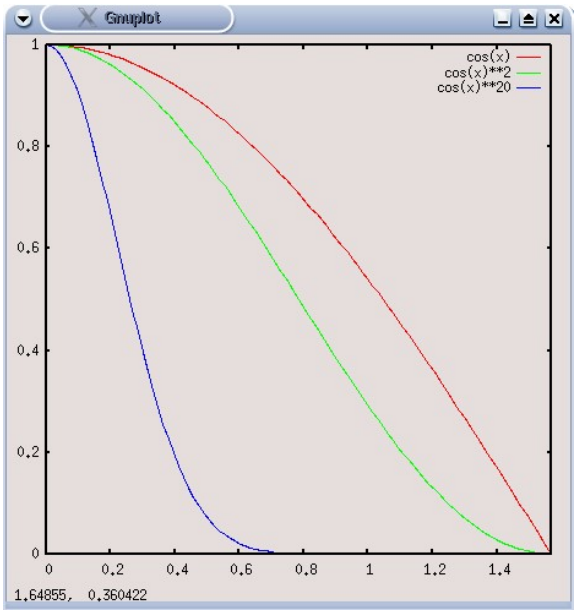


Abbildung 4.2: Veranschaulichung des Modells von Phong

k_s: Spiegelreflexionskoeffizient

n: Spiegelreflexionsexponent

Bei mehreren Lichtquellen stellt sich das wie folgt dar:

$$I = I_a * k_a + \sum_{i=1}^{m} f_{abi} * I_{pi} * (k_d * < L_i, N > + k_s * < R, V >^n)$$

4.2 Farben

Für jede Farbe/Wellenlänge λ gibt es eine extra Intensität.

$$I_\lambda = I_{a\lambda} * k_a * O_{d\lambda} + f_{ab} * I_{p\lambda} * (k_d * O_{d\lambda} * < N, L > + k_s * O_{s\lambda} * < R, V >^n)$$

$O_{d\lambda}$: diffuse Farbabgabe (der Farbe λ)

$O_{s\lambda}$: spiegelnde Farbabgabe (der Farbe λ)

Im Allgemeinen werden 3 Farben benutzt, um alle anderen darzustellen. Oftmals kommt das RGB-Farbsystem für 3 Wellenlängen zum Einsatz. Für jeweils rotes, grünes und blaues Licht werden die I_λ's berechnet. Daraus setzen sich dann die anderen Farben zusammen.

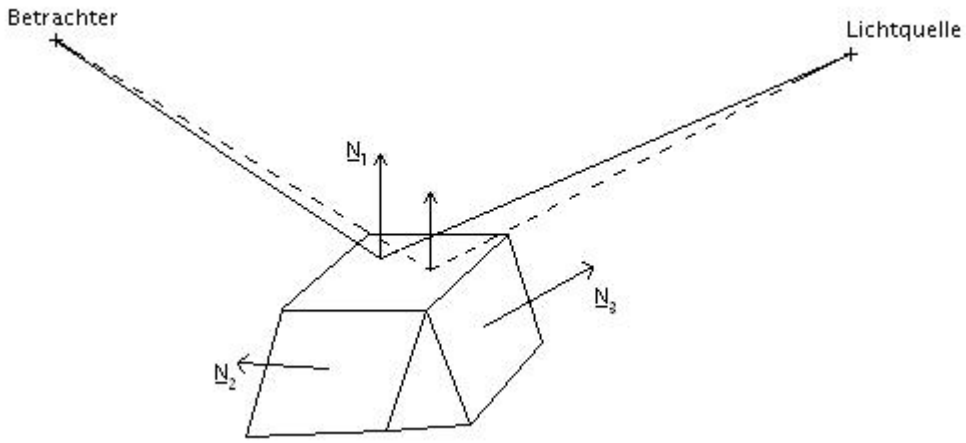


Abbildung 4.3: Problem: Ein- und Ausstrahlwinkel an jedem Punkt verschieden

4.3 Beleuchtung eines Polygons

Bei der ganzen Betrachtung ergibt sich folgendes Problem: Da die Winkel der eintreffenden Lichtstrahlen überall verschieden sind, müsste man die Lichtintenstät für jeden Punkt eigens berechnen (vgl. Fig. 3).

Das ist zuviel Aufwand.

Um diesen zu verringern, gibt es mehrere Möglichkeiten:

1. Konstante Beleuchtung: Man bestimmt für einen Punkt der Fläche die Intensität und wendet diese auf das gesamte Polygon an.

   ⇒ "Lichtquelle und Betrachter sind unendlich weit weg."

   Nachteil: Wenn ein Körper mit glatter Oberfläche z.B. eine Kugel, durch ein Polyeder angenähert wird, dann würde man eine "kantige" Kugeloberfläche, bestehend aus einer Vielzahl kleiner Flächen sehen(vgl. Fig 4).

2. lineare Interpolation: Man teilt die Flächen in Dreiecke (Triangulierung), bestimmt die Intensität an den Endpunkten und interpoliert linear (vgl. Fig 5).

Problem: man hat noch immer sichtbare Kanten

- bei konstanter Beleuchtung

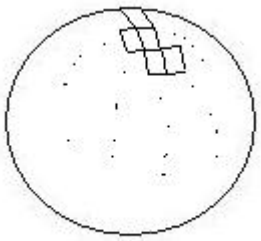


Abbildung 4.4: kantige Kugeloberfläche

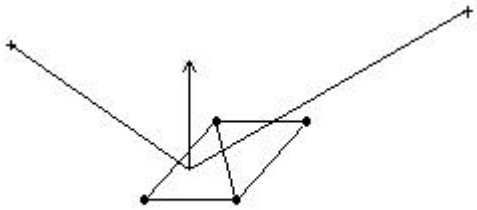


Abbildung 4.5:

- bei linearer Interpolation
- sogar, wenn jedes Pixel einzeln berechnet wird

Das lässt sich auf den "Mach-Effekt" zurückführen. Das Auge verstärkt Unterschiede, selbst wenn diese nur gering sind. Berühren sich also zwei Flächen mit leicht unterschiedlicher Intensität an einer Kante, so wird der geringe Unterschied trotzdem stärker hervorgehoben.

Lösung: Man mischt jeweils benachbarte Polygone miteinander $\Rightarrow$ "Gouraud-Shading".

4.4 Glättung von Kanten

Es wurden verschiedene Aspekte zur Beleuchtung von Modellen und Szenen vorgestellt. Es gibt zwei Arten von Licht, das Hintergrundlicht und Licht von Lichtquellen, wobei hier zu beachten ist, ob die Objekte diffus oder spiegelnd reflektieren. Probleme treten jedoch auf, wenn wir polyedrische Objekte haben, vor allem, wenn glatte Objekte aus Polyedern aproximiert werden.

Problem 4.4.1. *Falls polyedrische Objekte beleuchtet werden, kann es passieren, dass Kanten sichtbar bleiben, die in der Realität nicht existieren. Verstärkt wird dieses Problem durch den Mach-Effekt, also dadurch, dass das Auge oder der Sehnerv die Warnehmung von Kanten noch verstärken. Zur Glättung solcher Kanten gibt es zwei bekannte Verfahren, die hier vorgestellt werden.*

4.4.1 Gouraud-Shading

Definition 4.4.2 (Gouraud-Shading). Jeder Ecke v der polyedrischen Flächen wird ein eindeutiger Normalenvektor (N.V.) $\boldsymbol{N}$ zugeordnet.

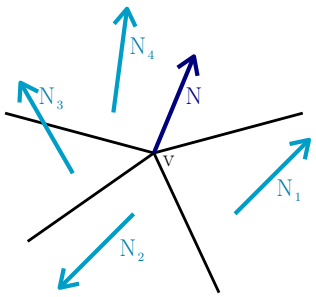


Abbildung 4.6: $\boldsymbol{N_1}$, $\boldsymbol{N_2}$, $\boldsymbol{N_3}\ldots$ N.V., der an v anstoßenden Facetten.

Man ordnet $\boldsymbol{N}$ die Summe (das arithmetische Mittel) der $\boldsymbol{N_i}$ (Normalvektoren der an die Ecke angrenzenden Flächen) mit $i = 1, 2, 3\ldots$ zu und normiert.

$$\boldsymbol{N} := \frac{1}{\left|\sum_{i=1}^{k} \boldsymbol{N_i}\right|} \cdot \sum_{i=1}^{k} \boldsymbol{N_i}$$

Anschließend wird die Intensität an den Ecken ausgerechnet, zum Beispiel mit dem Phong-Beleuchtungsmodell. Die Intensität wird an den restlichen

Punkten durch Interpolation (angenommen die Fläche ist trianguliert) berechnet.

Bemerkung. Der Normalenvektor entspricht der partiellen Ableitung. An Ecken sind aber mehrere verschiedene Tangentialebenen, also verschiedene Normalenvektoren möglich.

Bemerkung. Eine Gewichtung der verschiedenen $\boldsymbol{N_i}$ zum Beispiel über die Größe der Winkel mit denen die verschiedenen Facetten an der Ecke aufeinander treffen wäre auch möglich.

Die Interpolation lässt sich zum Beispiel in Verbindung mit Sweeplines durchführen. In Abbildung 4.7 werden die Intensitäten I_n einiger Punkter eines Dreiecks bestimmt.

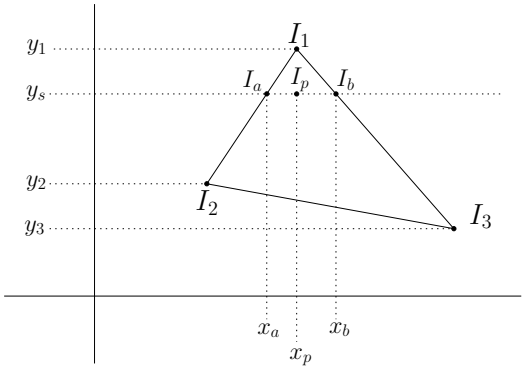


Abbildung 4.7: Interpolation über die Intensität (Gouraud-Shading) mittels Sweeplines

$$I_a = I_1 + (I_s - I_i) \cdot \frac{y_1 - y_s}{y_1 - y_2}$$

$$I_b = I_1 + (I_3 - I_1) \cdot \frac{y_1 - y_s}{x_b - x_a}$$

$$I_p = I_b + (I_a - I_b) \cdot \frac{x_b - x_p}{x_b - x_a}$$

Problem 4.4.3. *Gouraud-Shading entfernt Kanten. Es gibt jedoch auch Kanten, die sichtbar bleiben sollen.*

Sollen k Kanten an einer Ecke erscheinen, werden je $k + 1$ Normalvektoren den Kanten zugeordnet und für die Interpolation der Flächen je einer davon genutzt. Zum Beispiel werden zwei Normalvektor für die Interpolation der Flächen benutzt, wobei einer für die Interpolation der “vorderen” und einer

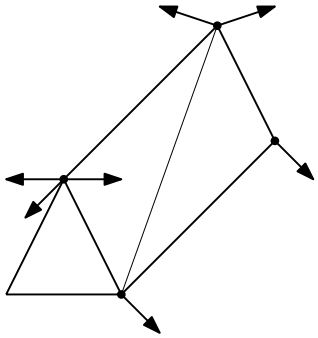


Abbildung 4.8: Wenn Kanten nicht geglättet werden sollen, arbeitet man mit mereren N.V.

für die "hintere" Fläche genutzt wird (wobei die "vordere" und "hintere" Fläche durch die zu verbleibende Kante geteilt werden).

Problem 4.4.4. *Probleme treten bei spiegelnden Reflektionen des Lichtes auf, da die Glanzeffekte verloren gehen. Hier hilft das Phong-Shading.*

4.4.2 Phong-Shading

Phong-Shading und Phong-Beleuchtungsmodell sind beide von Phong erfunden worden, sind jedoch zwei verschiedene Dinge.

Das Phong-Shading löst das Problem der sichtbaren Kanten, kann jedoch auch Glanzeffekte darstellen, was mit dem Gouraud-Shading nicht möglich ist.

Definition 4.4.5 (Phong-Shading). Die Normalvektoren der Ecken werden wie beim Gouraud-Shading festgelegt. Für die Punkte der Kanten und im Inneren der Facetten werden Normalvektoren aus den Normalvektoren der Ecken Interpoliert. Für diese Normalvektoren werden dann die Intensitäten nach einem Beleuchtungsmodell berechnet.

Bemerkung. Das Phong-Shading liefert zwar deutlich bessere Ergebnisse bei spiegelnden Reflektionen als das Gouraud-Shading, hat jedoch eine wesentlich höhere Laufzeit. Die asymptotische Laufzeit ist zwar gleich (O(n), bei n zu berechnenden Pixeln), der Aufwand je zu berechnendem Punkt, jedoch höher.

Ausblick Die Vorlesung wird sich im weiteren Verlauf mit

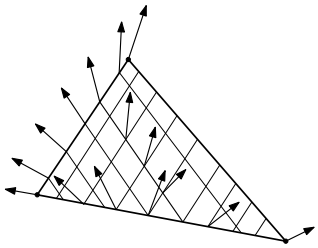


Abbildung 4.9: Interpolation der Normalvektoren (Phong-Shading).

Plate II.29 *Shutterbug.* Individually shaded polygons with diffuse reflection (Sections 14.4.2 and 16.2.3). (Copyright © 1990, Pixar. Rendered by Thomas Williams and H.B. Siegel using Pixar's PhotoRealistic RenderMan™ software.)

Abbildung 4.10: entnommen aus: J.D.Foley, A.Van-Dam, S.K.Feiner and J.F.Hughes: Computer Graphics - Principles and Practice (2nd Edition in C). Addison-Wesley, 1996

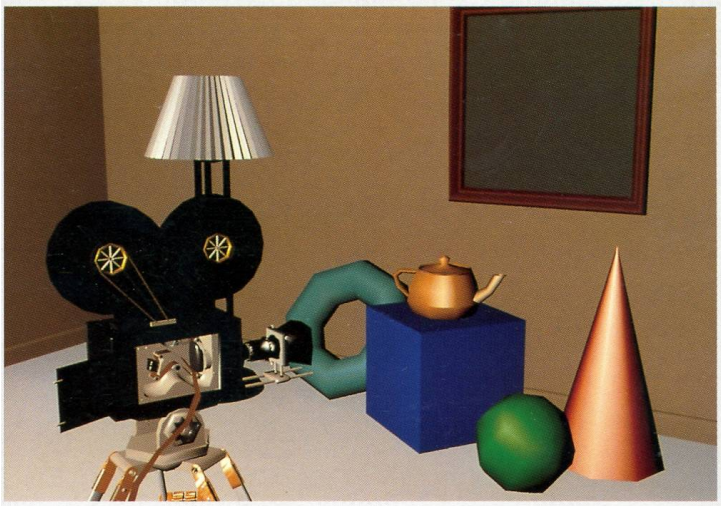


Plate II.31 *Shutterbug.* Gouraud shaded polygons with specular reflection (Sections 14.4.4 and 16.2.5). (Copyright © 1990, Pixar. Rendered by Thomas Williams and H.B. Siegel using Pixar's PhotoRealistic RenderMan™ software.)

Abbildung 4.11: entnommen aus: J.D.Foley, A.Van-Dam, S.K.Feiner and J.F.Hughes: Computer Graphics - Principles and Practice (2nd Edition in C). Addison-Wesley, 1996

Plate II.32 *Shutterbug.* Phong shaded polygons with specular reflection (Sections 14.4.4 and 16.2.5). (Copyright © 1990, Pixar. Rendered by Thomas Williams and H.B. Siegel using Pixar's PhotoRealistic RenderMan™ software.)

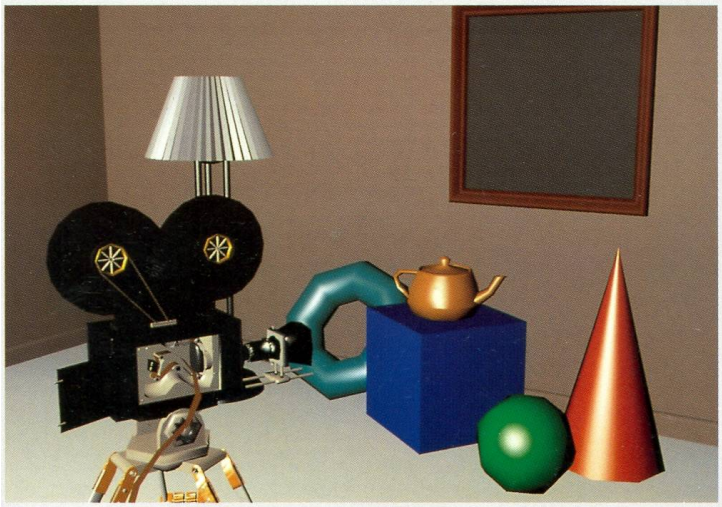


Abbildung 4.12: entnommen aus: J.D.Foley, A.Van-Dam, S.K.Feiner and J.F.Hughes: Computer Graphics - Principles and Practice (2nd Edition in C). Addison-Wesley, 1996

- Schatten
- Transparenz
- Texturen, Mustern, Bildern und
- Spiegelungen

beschäftigen.

4.5 Schatten

Bereiche, die von der Lichquelle aus nicht sichtbar sind, liegen im Schatten. Bestimmen lassen sich diese Bereiche mit den üblichen HSE-Algorithmen (hidden-surface-elimination), also z.B. dem Tiefenpuffer-Algorithmus.

Die Intensität (unter Nutzung einer oder mehrfacher Lichtquellen) berechnet sich dann, wie folgt:

$$I = I_a \cdot k_a + \sum_i s^i \cdot \left[f_{ab}^i \cdot I_p^i \cdot \left(k_d \cdot \boldsymbol{N} \cdot \boldsymbol{L^i} + k_s \cdot \left(\boldsymbol{R^i} \cdot \boldsymbol{V} \right)^n \right) \right] \tag{4.1}$$

wobei

$$s^i = \begin{cases} 0 \text{ falls der betrachte Punkt im Schatten liegt} \\ 1 \text{ sonst.} \end{cases}$$

Beispiel Zur Berechnung der Schatten in einer Szene mit einer Lichtquelle, mittels Tiefenpuffer-Algorithmus werden zwei Tiefenpuffer gebraucht. Der eine Tiefenpuffer berechnet die Sichtbarkeit, der andere die Schatten. Da der Tiefenpuffer-Algorithmus davon ausgeht, dass der Augpunkt einen negativen z-Wert hat, brauchen wir ein zweite Koordinatensystem, bei dem diese Bedingung erfüllt ist. Für einen Punkt $p(x, y, z)$ berechnen wir also Koordinaten (x', y', z'), die in einem fiktiven Koordinatensystem liegen, in dem die Lichtquelle einen negativen Wert hat. Für jeden dargestellten Punkt (x_0, y_0, z_0) berechnen wir jetzt die zugehörigen Werte (x'_0, y'_0, z'_0). Hat der Tiefenpuffer, der die Schattenberechnet an dieser Stelle einen Wert $< z'_0$, so liegt der Punkt im Schatten. Sonst (Wert im Tiefenpuffer $= z'_0$) liegt er im Licht.

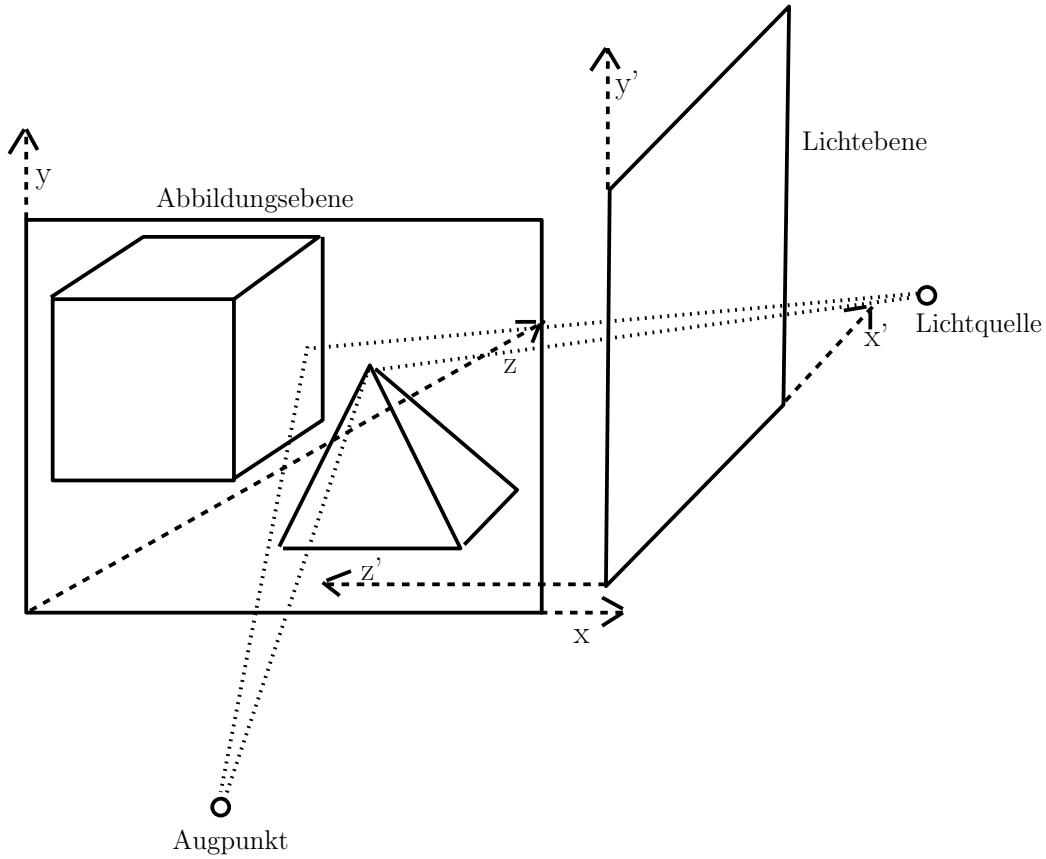


Abbildung 4.13: Beispiel zur Schattenberechnung mittels Tiefenpuffer

4.6 Oberflächendetails(OD)

Viele Oberflächen sind nicht einfarbig sondern mit Bildern oder Mustern (Textur) versehen. Z.B. Bilder, Tapeten, Fußboden in Bild 4.14.

Plate II.35 *Shutterbug.* Texture mapping (Sections 14.4.7, 16.3.2, 17.4.2, and 17.4.3). (Copyright © 1990, Pixar. Rendered by Thomas Williams and H.B. Siegel using Pixar's PhotoRealistic RenderMan™ software.)

Abbildung 4.14: Bild entnommen aus dem Buch: J.D.Foley, A.Van-Dam, S.K.Feiner and J.F.Hughes: Computer Graphics - Principles and Practice (2nd Edition in C). Addison-Wesley, 1996

Einfachste Möglichkeit: Oberflächendetail(OD)-Polygone: Bilder, Türen, Fenster.
OD werden auf eine Grundfläche(z.B. Wand) im Bild gebracht. Beim HSE braucht man sie noch nicht zu berücksichtigen, erst bei der Ausleuchtung der einzelnen Pixel muss man die OD-Farbe beachten. Wenn man sie wiedergibt und modelliert hat, erst HSE machen, dann z.B. nach Phong-Modell ausleuchten und die Farbe und Intensität bestimmen (durch geeignete Parameterwahl kann man die Spiegelung etc. anpassen).
Eine Ausnahme sind durchsichtige Fenster, Details dahinter müssen natürlich dargestellt werden.

4.7 Textur-Abbildung

Wir gehen davon aus, dass wir eine Textur als Bild gegeben haben. Genauso könnte man eine Erzeugungprozedur für die Textur benutzen, dies würde sich zum Beispiel beim Erstellen eines Karomusters anbieten, da wir nicht jedes einzelne Pixel der leicht beschreibbaren Textur angeben/abspeichern müssten.

Wir benutzen ein eigenes Textur-Koordinatensystem um die Textur mit Höhe v und Breite u anzugeben. Einzelne Punkte in einer Textur werden auch Texel genannt welche auf eine Oberfläche adäquat abgebildet werdeb können.(s. Bild 4.15)

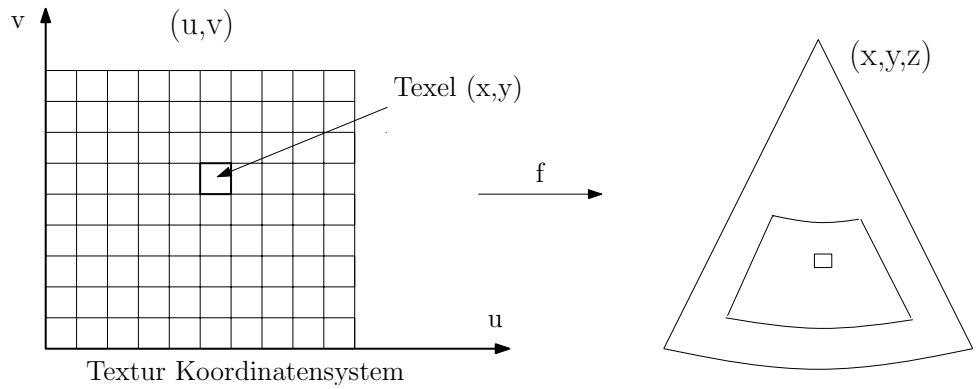


Abbildung 4.15: Punkt (x,y) der Textur wird abgebildet auf Punkt (x,y,z) auf dem Kegel

Beispiel.

$$(u, v) \in [0, 1]^2$$

Parametrisierte Kegel

$$(s, t) \mapsto (t \cos s, t \sin s, t) s \in [0, 2\pi] t \in [0, 1]$$
$$s = \frac{1}{2} u 2\pi$$
$$t = \frac{1}{2} v.$$

Abbildung von 2 ins 3 Dimensionale (am Einfachsten wenn Kegel parametrisiert). Wenn man nur die Hälfte haben will, könnte man s=$\frac{1}{2}u2\pi$ t=$\frac{1}{2}v$.

Um ein Pixel P des Bildraums einzufärben bestimmen wir den Bereich B_P im Objektraum, der auf P abgebildet wird.(s.Bild 4.16) Wir bestimmen Bereich

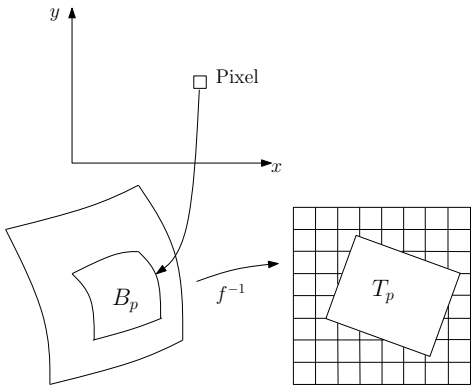


Abbildung 4.16:

T_P in Texturvorlage, der auf B_P abgebildet wird und geben P die mitlere Farbe von T_P.

4.8 Durchsichtige Objekte

Mögliche Probleme bei durchsichtigen Objekten: gefärbte Fenstergläser würden Objektfarben dahinter verändern (Mischfarben, s.Bild 4.17) und dicke Gläser würden Licht brechen.

- direkter Durchblick (z.B. Fenster)
- Brechung
- durchscheinende Objekte (z.B. Milchglas,lässt Licht durch aber keine Details der Objekte dahinter)

Überlagern wir alle Effekte erzeugen wir Realismus.
Einfachheitshalber ignorieren wir oft die Brechung wenn sie zu vernachlässigen ist (z.B. bei Fenstern).

Wie bestimmen wir die Intensität des Lichts der Wellenlänge λ bei Überschneidung von Objekten?
Möglichkeit: lineare Interpolation der Intensitäten im Schnittbereich.

Sei I^i_λ Intensität von λ auf O_i. Wir mischen die Farben und haben den Faktor

$$(1 - k^\lambda_{t_1})I^1_\lambda + k^\lambda_{t_1} I^2_\lambda,$$

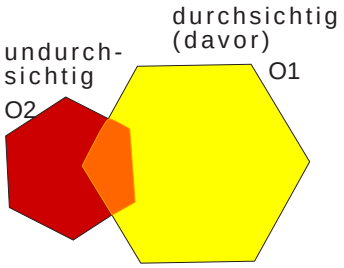


Abbildung 4.17: Durch die Überlagerung erscheint das Objekt dahinter in einer Mischfarbe

wobei $k_{t_1}^{\lambda} \in [0,1]$ der Durchlässigkeitskoeffizient von O_1 ist (wie stark wird Licht der Wellenlänge λ durchgelassen).
$k_{t_1}^{\lambda} = 0 : O_1$ undurchlässig für Farbe λ
$k_{t_1}^{\lambda} = 1 : O_1$ völlig durchlässig für Farbe λ
Realistischer wäre den ambienten und diffusen Anteil von O_2 gemäß der Formel durchzulassen und den spiegelnden Anteil von O_1 zu addieren.

Es könnten sich auch mehrere Oberflächen überlappen. Durch Implementierung des Sweepline-Algorithmus können wir für jedes der Überlappungsintervalle die Mischung der Farben durchlässiger Objekte berechnen.(s.Bild 4.18)

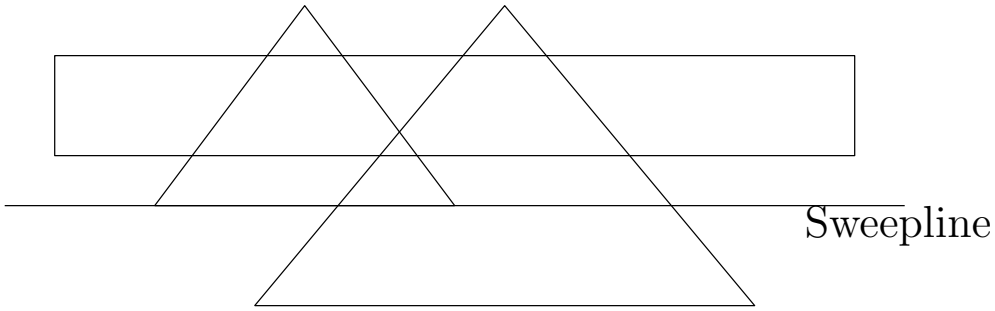


Abbildung 4.18: Sweepline scannt nach Ecken

4.9 Brechung

Brechung bezeichnet die Richtungsänderung einer Welle aufgrund einer lokalen Änderung ihrer Ausbreitungsgeschwindigkeit, die durch den Brechungs-

index $\eta_{i\lambda}$ beschrieben wird.

Treffen Licht- oder andere elektromagnetische Wellen von einem Medium (zum Beispiel Luft) auf ein anderes (zum Beispiel Glas), dessen Brechungsindex sich von dem des ersten unterscheidet, wird ein Teil des Lichts reflektiert, ein anderer erfährt eine Ablenkung gemäß dem Snelliusschen Brechungsgesetz. (s.Bild 4.19)

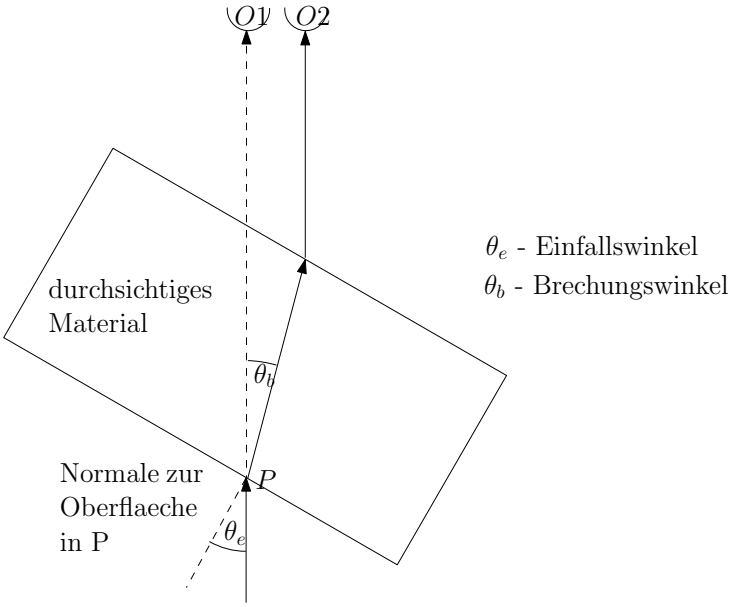


Abbildung 4.19: Brechung

es gibt: $\eta_{2\lambda}$

$$\frac{\sin\theta_e}{\sin\theta b} = \frac{\eta_2\lambda}{\eta_1\lambda}$$

Snelliussches Gesetz. $\eta_{i\lambda}$-Brechungsindex des Materials i für Licht der Farbe (wellen länge)λ

Z.B. Vakumm hat Index 1 (für alle Farben). Wasser hat Index 1,33 für gelbes Licht, Natrium $\lambda = 10^{-10}m$

$\boldsymbol{E}$-Sehstrall in Richtung Augpunkt
$\boldsymbol{T}$-Richtung des gebrochenen Sehstrahls für Wellenlänge λ
$\boldsymbol{N}$-Oberflächennormale
$\boldsymbol{M}$-tangential zur Oberfläche in gleicher Ebene wie $\boldsymbol{E}$ und $\boldsymbol{T}$
Alle Vektoren sind normiert auf Länge 1. (s.Bild 4.20)

also:

$$\boldsymbol{T} = -\cos\theta_b \boldsymbol{N} + \sin\theta_b \boldsymbol{M}$$
$$\boldsymbol{M} = \frac{\boldsymbol{N}\cos\theta_e - \boldsymbol{E}}{\sin\theta_e}$$

also:

$$\boldsymbol{T} = \frac{\sin\theta_b}{\sin\theta_e}(\boldsymbol{N}\cos\theta_e - \boldsymbol{E}) - \cos\theta_b \boldsymbol{N}.$$

Nach dem Schnellschen Brechungsgesetz: $\frac{\sin\theta_b}{\sin\theta_e} = \frac{\eta_{2\lambda}}{\eta_{1\lambda}} = \eta_{2,1,\lambda}$. Damit haben wir

$$\boldsymbol{T} = (\eta_{2,1,\lambda}\cos\theta_e - \cos\theta_b)\boldsymbol{N} - \eta_{2,1,\lambda}\boldsymbol{E},$$

wobei $\cos\theta_b = \sqrt{1-\sin^2\theta_b} = \sqrt{1-\eta_{2,1,\lambda}^2\sin^2\theta_e} = \sqrt{1-\eta_{2,1,\lambda}(1-(\boldsymbol{N}\boldsymbol{E})^2)}$.

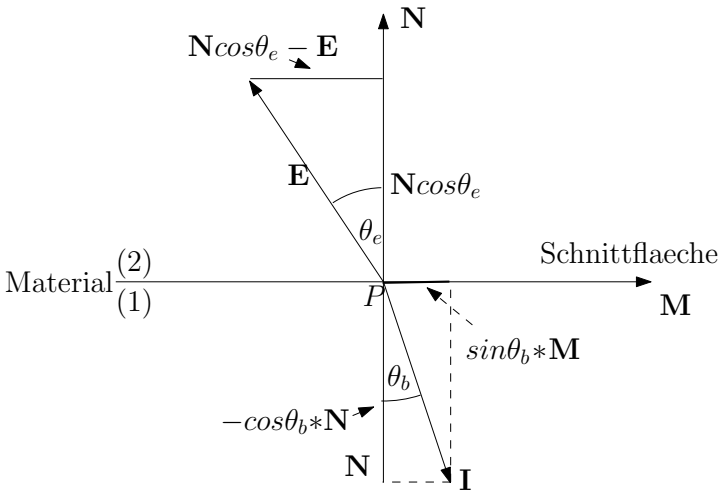


Abbildung 4.20:

Kapitel 5

Globale Beleuchtung

Bisher haben wir nur Licht von Lichtquellen berücksichtigt. Gegenstände werden aber auch durch indirektes Licht beleuchtet, das durch diffuse oder direkte Reflexion entsteht. Effekte wie Spiegelungen und Brechungen müssen dann beachtet werden.

Wir betrachten zwei Methoden:

- Ray Tracing
- Radiosity

5.1 Ray Tracing

Idee: Erweiterung von HSE-Methoden (Abb. 5.1)

Von z aus wird durch jedes Pixel P' ein Strahl **R** gelegt. Dieser trifft (als erstes) einen Punkt P auf der Oberfläche eines Objekts O. In die Beleuchtung von P' wird das Licht an Stelle P mit einbezogen aber auch, ob O spiegelnd oder durchsichtig ist, oder im Schatten liegt.

5.1.1 Schatten, Reflexion und Brechung

Schatten

Zusätzlich wird von P aus eine Strahl zu jeder Lichtquelle L geschickt. Falls er ein Objekt trifft, liegt P bezüglich L im Schatten. Die einfachste Form

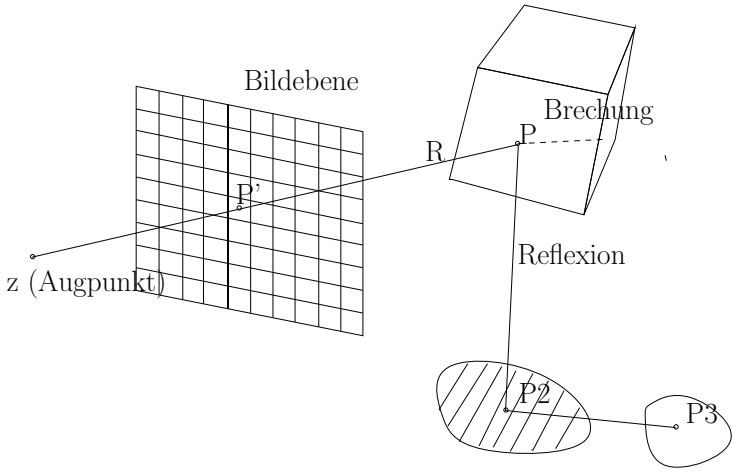


Abbildung 5.1: Ray-Tracing: Das Licht wird vom Augpunkt aus zurückverfolgt

des Ray Tracings berücksichtigt nicht, dass das Objekt durchsichtig oder reflektierend sein kann. Für realistische Bilder möchte man aber auch diese Effekte einbeziehen.

Spiegelung und Brechung

Falls O spiegelnd ist, schicke einen Sekundärstrahl von P aus, gemäß Reflexionsgesetz und berechne dessen Intensität rekursiv.

Falls O durchsichtig ist, schicke einen Sekundärstrahl von P aus, gemäß Brechungsgesetz und berechne dessen Intensität rekursiv.

Diffuses Licht von matten Objekten wird beim Ray Tracing nicht berücksichtigt.

5.1.2 Berechnung der Intensität

Baum von Strahlen

Die Strahlen werden zurückverfolgt. Bei Objekten, die reflektieren und spiegeln, teilt sich der Strahl auf und man muss beide Strahlen weiterverfolgen. Dadurch entsteht ein Baum von Strahlen. Siehe Abb. (5.3)

Blätter: falls Strahl kein Objekt trifft (Hintergrundfarbe) oder ein weder

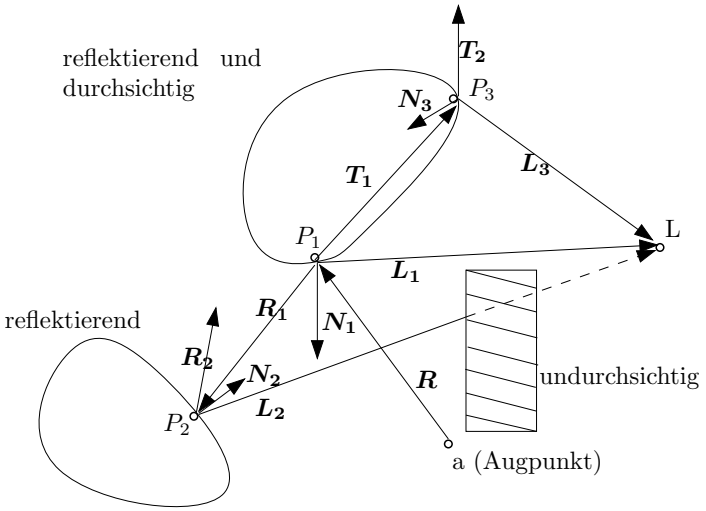


Abbildung 5.2: Beispiel mit Reflexion, Brechung und Schatten
$\boldsymbol{N}^{(i)}$ Oberflächennormalen $\boldsymbol{R}^{(i)}$ reflektierte Strahlen
$\boldsymbol{T}^{(i)}$ gebrochene Strahlen $\boldsymbol{L}^{(i)}$ Schattenstrahlen

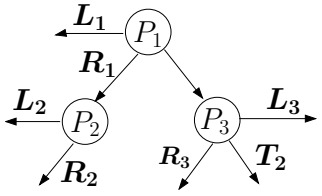


Abbildung 5.3: Baum von Strahlen

spiegelndes noch durchsichtiges Objekt getroffen wird.

Baum kann (theoretisch) unendlich tief sein (z.B. gegenüberliegende Spiegel, Lichtleiter) (Abb. 5.4)

In der Praxis wird die Tiefe auf 3 oder 4 beschränkt.

Formel für die Beleuchtungsintensität eines Punktes q an der Oberfläche für eine Farbe λ nach dem Phong-Modell

$$I_\lambda = I_{a\lambda} \cdot k_a \cdot O_{d\lambda} + \sum_i^k s^{(i)} \cdot f_{ab}^{(i)} \cdot I_{p\lambda} \cdot [k_d \cdot O_{d\lambda} \cdot (\boldsymbol{N} \cdot \boldsymbol{L}^{(i)}) + k_s \cdot O_{s\lambda} \cdot (\boldsymbol{R}^{(i)} \cdot \boldsymbol{V})^n] + k_s \cdot I_{s\lambda} + k_t \cdot I_{t\lambda} \tag{5.1}$$

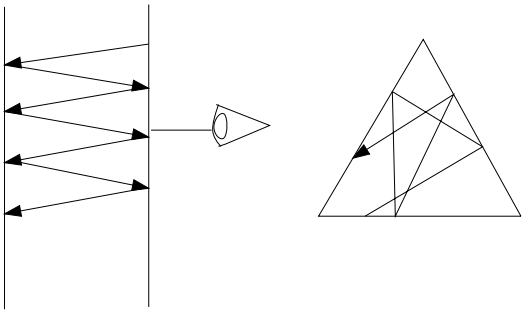


Abbildung 5.4: unendlich viele Reflexionen durch Spiegel

$k_t \in [0,1]$ Koeffizient für die Durchsichtigkeit des Objekts wobei

- $k_t = 0 \Leftrightarrow$ Objekt undurchsichtig
- $k_t = 1 \Leftrightarrow$ Objekt völlig durchsichtig

$$\left.\begin{array}{ll} I_{s\lambda} & \text{Intensität des reflektierten Strahls (von q aus)} \\ I_{t\lambda} & \text{Intensität des gebrochenen Strahls} \end{array}\right\} \text{ rekursiv berechnen}$$

Durch die Rekursion wird der Baum von unten nach oben berechnet. In jedem Knoten wird Gleichung (5.1) ausgewertet und das Ergebnis zur Berechnung des darüberliegenden Knotens verwendet.

5.1.3 Effizienz

Bei $m \times m$ Gitter, Strahlenbaum der Tiefe k und l Lichtquellen ist die Anzahl der zu berechnenden Knoten:

$$m^2 \cdot (2^{k+1} - 1) \cdot (l + 1)$$

Für jeden muss die nächste getroffene Oberfläche berechnet werden. Für n Oberflächen:

$$m^2 \cdot 2^k \cdot l \cdot n$$

Bsp.: $m = 1000$ $k = 3$ $n = 2$ $n = 100 \rightarrow 16 \cdot 10^8$ Schritte (Schnitt von Strahl mit Oberfläche)

Verringerung der Laufzeit

- für primäre Strahlen Tiefenpuffer verwenden (Item buffer)
- Arbeit mit bounding boxes
- Suche im Baum abbrechen, wenn die Intensität zu klein wird

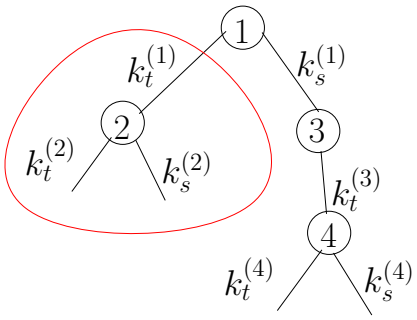


Abbildung 5.5: bessere Effizienz mit branch and bound

$$k_t(1)(... + \underbrace{k_s^{(2)} I_{s\lambda}}_{=1} + \underbrace{k_t^{(2)} I_{t\lambda}}_{=1})$$

Annehmen I ist 1. Falls Ergebnis sehr klein, den rot markierten Unterbaum ignorieren (Abb. 5.5)

- "Lichtpuffer" um Schatten zu bestimmen (Tiefenpuffer anwenden). Das liefert ein Raster von Punkten, die vom Licht getroffen werden oder nicht getroffen werden. (Abb. 5.6)

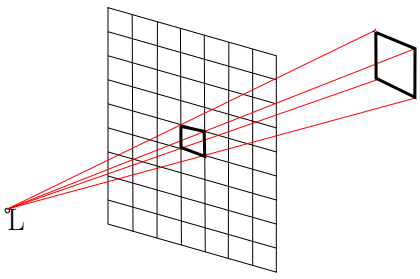


Abbildung 5.6: 3d-Raum wird durch Raster in Pyramiden zerlegt, in denen jeweils nur das vorderste Objekt beleuchtet wird.

5.2 Radiosity

5.2.1 Beschreibung des Modells

Bei Radiosity betrachtet man die Abstrahlung von Licht aller Oberflächen und die daraus resultierende gegenseitige Beleuchtung der Flächen. Das bedeutet, dass jede Fläche eine Lichtquelle sein kann. Die Verwendung dieses Modells führt zu wesentlich realistischeren Ausleuchtungen als ein konstanter ambienter Lichtanteil. Zum Beispiel führt es zur farbigen Beleuchtung von Objekten in der Nähe einer angestrahlten farbigen Oberfläche.

5.2.2 Berechnung der Intensitäten

Im Folgenden ist immer statt der Energie die Energie pro Zeiteinheit gemeint. Angenommen die Szene besteht aus n Flächenstücken $f_1, \ldots, f_n$, welche (ziemlich) eben, undurchsichtig, diffus reflektierend und „hinreichend“ klein sind, die Flächeninhalte dieser seien $A_1, \ldots, A_n$, dann gilt für i=1, ... n

$$B_i = E_i + \rho_i \sum_{j=1}^{n} B_j \underbrace{F_{j,i} \frac{A_j}{A_i}}_{=F_{i,j}}$$

B_i ... Strahlung der Fläche f_i pro Flächeneinheit und Zeit $[\frac{W}{m^2}]$
E_i ... eigene Strahlung
ρ_i ... Reflexionsfähigkeit der Fläche f_i=$\frac{\text{Energie des reflektierten Lichts}}{\text{Energie des einfallenden Lichtes}}$
$F_{j,i}$ ... Formfaktor, also der Anteil, der von der j-ten Fläche ausgestrahlten Energie, der die i-te Fläche f_i erreicht. Dabei werden die gegenseitige Lage und Hindernisse berücksichtigt.

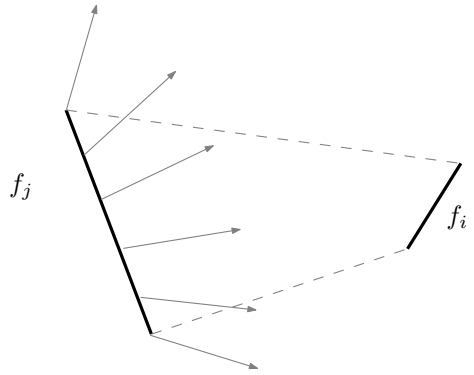


Abbildung 5.7: schematische Darstellung der Beleuchtung der Fläche f_i durch die Fläche f_j

B_j	Energie, die eine Flächeneinheit von f_j verlässt.
$B_j F_{j,i}$	Energie, die eine Flächeneinheit von f_j nach ganz f_i strahlt
$B_j F_{j,i} \frac{A_j}{A_i}$	Energie, die ganz f_j nach einer Flächeneinheit von f_i strahlt

Dabei gelten gleiche Verhältnisse für die Strahlung von f_i nach f_j wie von f_j nach f_i Also gilt:

$$A_i F_{i,j} = A_j F_{j,i} \Leftrightarrow F_{i,j} = \frac{A_j}{A_i} F_{j,i}$$

Damit erhält man für B_i

$$B_i = E_i + \rho_i \sum_{j=1}^{n} B_j F_{i,j} \qquad i = 1, \ldots, n$$

Man kann diese Formel als Gleichungssystem schreiben:

$$\begin{bmatrix} 1-\rho_1 F_{11} & -\rho_1 F_{12} & \cdots & -\rho_1 F_{1n} \\ -\rho_2 F_{21} & \ddots & & \vdots \\ \vdots & & & -\rho_{n-1} F_{n-1n} \\ -\rho_n F_{n1} & \cdots & -\rho_n F_{nn-1} & 1-\rho_n F_{nn} \end{bmatrix} \begin{bmatrix} B_1 \\ \vdots \\ \\ B_n \end{bmatrix} = \begin{bmatrix} E_1 \\ \vdots \\ \\ E_n \end{bmatrix}$$

Mit diesem Gleichungssystem kann man die B_i berechnen, wenn die Formfaktoren und Reflexionsfähigkeiten für alle f_i bekannt sind. Bei farbigem Licht muss man dieses Gleichungssystem für jede Wellenlänge einzeln berechnen (z.B. beim RGB-System 3 mal). Daraus erhält man die Intensität für jedes f_i in der Szene. Die Szene kann dann mit HSE auf den Bildschirm abgebildet werden. Eventuell benutzt man Gouraud-Shading damit keine unerwünschten Kanten auftreten.

Berechnung der Formfaktoren

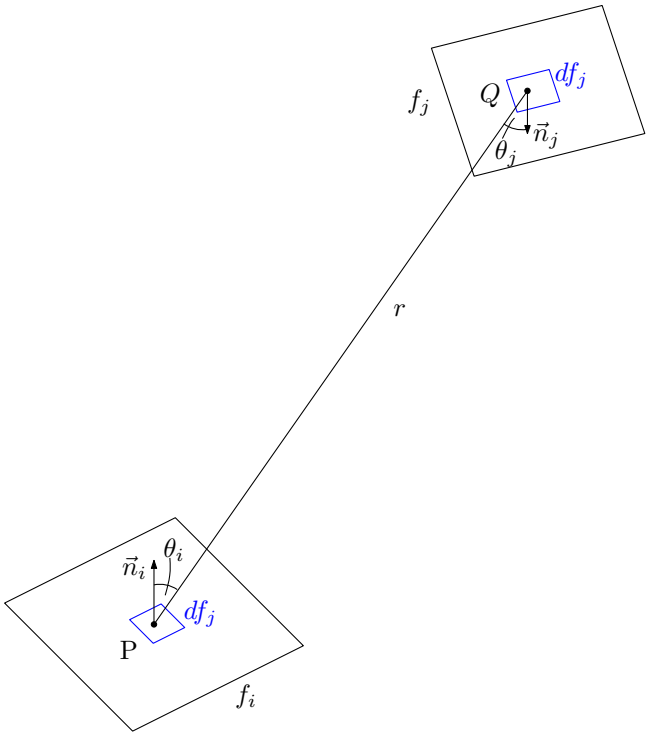


Abbildung 5.8: Definition der Winkel, Normalenvektoren und Flächenstücke für zwei Flächen im Raum

$\mathrm{d}f_i$	...	infinitesimales Flächenstück von f_i bei P
$\mathrm{d}f_j$	...	infinitesimales Flächenstück von f_j bei Q
$\vec{n_i}$	...	Normalenvektor von f_i bei P
$\vec{n_j}$	...	Normalenvektor von f_j bei Q
θ_i	...	Winkel zwischen $\vec{n_i}$ und der Strecke $\overline{PQ}$
θ_j	...	Winkel zwischen $\vec{n_j}$ und der Strecke $\overline{QP}$
r	...	Abstand zwischen P und Q

Unter diesen Voraussetzungen gilt für den Formfaktor von $\mathrm{d}f_i$ nach $\mathrm{d}f_j$

$$F_{\mathrm{d}i\,\mathrm{d}j} = \frac{\cos\theta_i \cos\theta_j}{\pi r^2} H_{PQ} \mathrm{d}f_i$$

H_{PQ} ... Blockierungsfaktor $\begin{cases} 0 & \text{wenn zwischen P und Q eine andere Fläche liegt} \\ 1 & \text{sonst} \end{cases}$

Erklärung der Formel

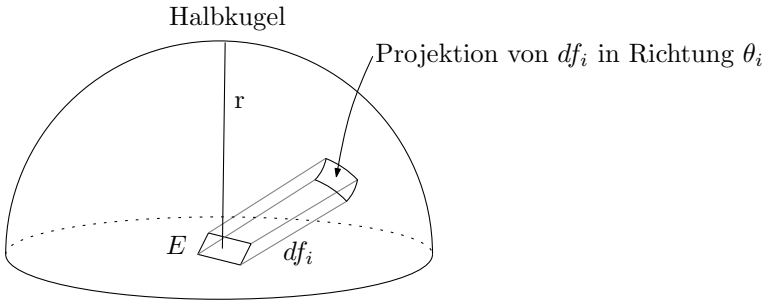


Abbildung 5.9: Projektion von $\mathrm{d}f_i$ auf die Halbkugel mit Radius r

Projeziert man $\mathrm{d}f_i$ in Richtung θ_i auf ein Halbkugel über $\mathrm{d}f_i$ erhält man für die projezierte Fläche $r^2 \cos\theta_i \mathrm{d}f_i$
Die durchdringende Energie pro Flächeneinheit und Zeiteinheit ist $cE\cos\theta$

c ... Proportionalitätsfaktor
E ... von $\mathrm{d}f_i$ ausgestrahlte Energie pro Zeiteinheit

Für c muss gelten

$$\int_{\mathrm{O(K)}} c\, E \cos\theta \, \mathrm{d}F = E$$

O(K) ... Oberfläche der Halbkugel
$\mathrm{d}F$ ... Ring der Breite $r\mathrm{d}\theta$ und des Umfangs $r\sin\theta 2\pi$ also

$$\mathrm{d}F = 2\pi r^2 \sin\theta \mathrm{d}\theta$$

siehe Abbildung 5.10

Das Integral entspricht der gesamten Energie , die durch die Oberfläche der Halbkugel pro Zeiteinheit dringt.

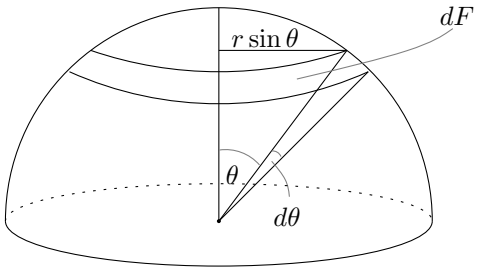


Abbildung 5.10: Konstruktion des Flächenstücks dF auf der Halbkugel

Man erhält:

$$\begin{aligned} 1 &= c\pi r^2 \int_{\theta=0}^{\frac{\pi}{2}} 2\sin\theta\cos\theta \mathrm{d}\theta \\ &= c\pi r^2 \left[\sin^2\theta\right]_0^{\frac{\pi}{2}} \\ &= c\pi r^2 \end{aligned}$$

$\Leftrightarrow c = \frac{1}{\pi r^2}$

Der durchdringende Anteil pro Flächeneinheit der Kugel beträgt $\frac{1}{\pi r^2}\cos\theta$. Mit der Projektion diese Anteils auf die Fläche durch $\mathrm{d}f_j$ bekommt man einen Faktor $\cos\theta_j$, und der Anteil, der auf $\mathrm{d}f_j$ auftrifft hat den Faktor $\cos\theta_j \mathrm{d}f_j$.

Damit erhält man für den Anteil, der von $\mathrm{d}f_i$ ausgestrahlten Energie pro Zeiteinheit, der auf $\mathrm{d}f_j$ auftrifft.

$$F_{\mathrm{d}i\,\mathrm{d}j} = \frac{\cos\theta_i \cos\theta_j}{\pi r^2} H_{ij}\,\mathrm{d}f_j$$

H_{ij} ... Blockierungsfunktion $H_{ij}(P,Q) = H_{PQ}$

Bei Integration über $\mathrm{d}f_j$ erhält man den Anteil der von $\mathrm{d}f_i$ ausgestrahlten Energie, der pro Zeiteinheit auf f_j trifft.

$$F_{\mathrm{d}i\,j} = \int_{f_j} \underbrace{\frac{\cos\theta_i \cos\theta_j}{\pi r^2} H_{ij}}_{F_{\mathrm{d}i\,\mathrm{d}j}}\,\mathrm{d}f_j$$

Bei Integration über $\mathrm{d}f_i$ erhält man daraus den Anteil der von f_i ausgestrahlten Energie, der pro Zeiteinheit auf f_j trifft.

$$\boxed{F_{ij} = \frac{1}{A_i}\int_{f_i} F_{\mathrm{d}i\,j}\,\mathrm{d}f_i = \frac{1}{A_i}\int_{f_i}\int_{f_j} \frac{\cos\theta_i\cos\theta_j}{\pi r^2} H_{ij}\,\mathrm{d}f_j\,\mathrm{d}f_i}$$

5.2.3 Heuristiken

Radiosity ist eine Methode zum Berechnen der ambienten Strahlung.

$$B_i = E_i + \rho_i \cdot \sum_{j=0}^{n} B_j \cdot F_{j,i} \cdot \frac{A_j}{A_i}$$

wobei E_i die selbstausgestrahlte Energie ist und $F_{j,i}$ der Fromfaktor, der bestimmt wieviel Energie tatsächlich bei E_i ankommt.

Wenn man die verschiedenen Formfaktoren berechnet, läßt sich das Gleichungssystem der B_i's lösen.

$F_{i,j} = \frac{1}{A_i} \int\limits_{f_i} \int\limits_{f_j} \frac{\cos\Theta_i \cdot \cos\Theta_j}{\Pi r^2} H_{i,j} df_j df_i$

Das für jeden Formfaktor zu berechnen ist zu aufwändig. Es gibt eine einfachere Methode wobei man $F_{di,j}$ berechnet:

$$F_{di,j} = \frac{\text{Fläche von} f''_{i,j}}{\underbrace{\text{Grundfläche}}_{=\Pi}}$$

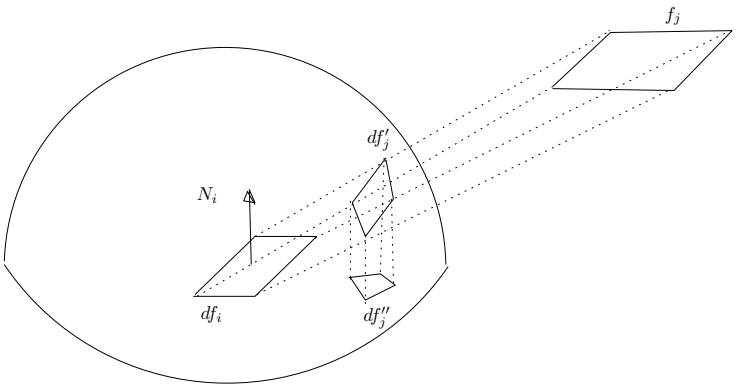


Abbildung 5.7: Radiosity an Halbkugel

df'_i ist die Projektion von f_i auf die Halbkugel

df''_i ist die Projektion von f'_j auf die Grundfläche (liefert Faktor Θ_i)

Dann läßt sich $F_{i,j}$ mit $F_{i,j} = \frac{1}{A_i} \int\limits_{f_i} F_{di,j} df_i$ berechnen.

Weiterhin ist es immer noch sehr aufwändig die Projektion von f''_j zu berechnen, da von einer Halbkugel aus projeziert werden muss. Deshalb nimmt man zur Vereinfachung der Berechnung oft einen Halbwürfel, also die eine Hälfte eines Würfels der Seitenlänge 2.

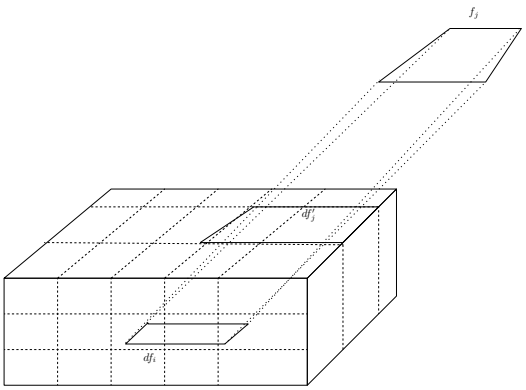


Abbildung 5.8: Radiosity an Halbwürfel

df_i' sind die Gitterzellen, die von den Projektionsstrahlen von f_j nach df_i getroffen werden.

Daraus bestimmt man den Anteil jeder getroffenen Gitterzelle an $F_{di,j}$ aufsummieren. $F_{di,j}$ ist nur approximativ, da auch nur teilweise getroffene Gitterzellen komplett berechnet werden und ausserdem zur Vereinfachung ein Würfel für die Projektionen benutzt wird.

Radiosity liefert nur diffuses Licht, keine Reflektion und keine Transparenz. Also muß diese noch extra berechnet werden.

Man kann diesen Makel durch einen zweiten Durchgang beheben:

Im ersten Durchgang berechnet man die Radiosity mit einer Spiegelwelt. Die Spiegelwelt besteht aus den Objekten, die man in den Spiegeln sieht. Diese werden als Quasi-Reale Objekte eingebunden, dann werden die Spiegel entfernt und dann wird Radiosity berechnet.

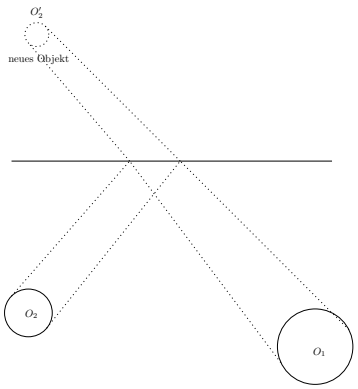


Abbildung 5.9: Zusätzliche Objekte beim Radiosity durch Spiegel

Beim zweiten Durchgang wird ganz normal Raytracing berechnet. Das liefert die Spiegelungen und die Brechungseffekte.

Die Radiosity-Pipeline:

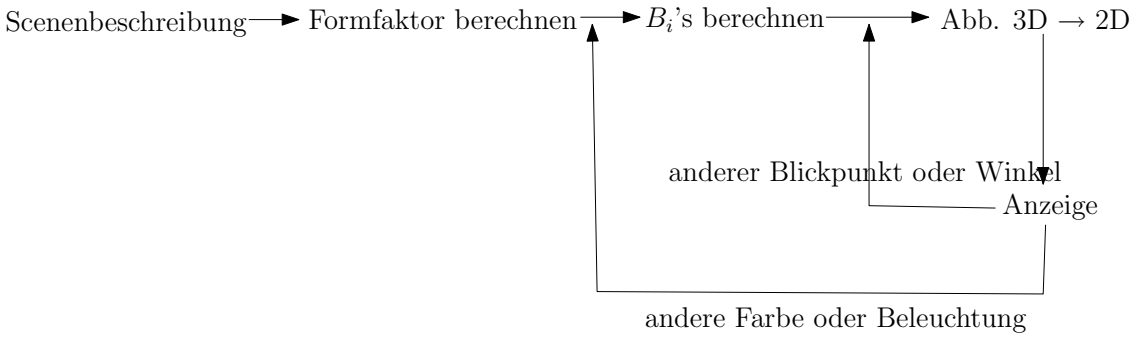


Abbildung 5.10: Radiosity-Pipeline

Kapitel 6

Verschiedenes

6.1 Fraktale

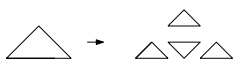
Fraktale sind Objekte die im mathematischen Sinne selbstähnlich sind. Das heißt, wenn man sie in einer bestimmten Art und Weise zerschneiden kann und dann mehrere verkleinerte Originale bekommt.

(a) Strecke

Eine Strecke kann man in zwei Hälften der halben Grösse teilen.

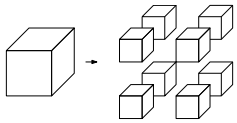
(b) Dreieck

Ein Dreieck kann man in 4 kleinere Dreicke teilen, die dann halb so groß wie das Original sind.



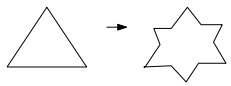
(c) Würfel

Einen Würfel kann man in 8 Würfel der halben Grösse teilen.



(d) Kochschneeflocke

Eine Kochschneeflocke wird dadurch erzeugt, dass man den Ersetzungsvorgang unendlich oft durchführt. Dabei wird jede Strecke durch 4 Strecken, die ein drittel so lang wie die Originalstrecke sind, ersetzt.



Es ist sinnvoll zu sagen, wenn ein Objekt O in k ähnliche Teile der Größe α von O zerfallen, dann ist die Dimension $Dim(O) = \log_{\frac{1}{\alpha}} k$.

Das bedeutet:

(a) mit $k = 2$ und $\alpha = \frac{1}{2}$ hat Dimension $Dim(O) = \log_2 2 = 1$

(b) mit $k = 4$ und $\alpha = \frac{1}{2}$ hat Dimension $Dim(O) = \log_2 4 = 2$

(c) mit $k = 8$ und $\alpha = \frac{1}{2}$ hat Dimension $Dim(O) = \log_2 8 = 3$

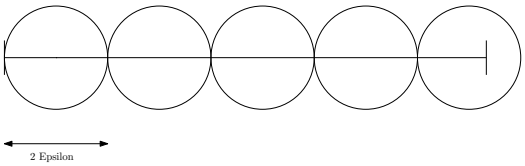
(d) mit $k = 4$ und $\alpha = \frac{1}{3}$ hat Dimension $Dim(O) = \log_3 4 = 1.26$

Definition 6.1.1 (fraktale Dimension). Sei $A \subset \mathbb{R}^d$ beschränkt und für $\varepsilon > 0$ sei $N_A(\varepsilon)$ die minimale Anzahl der d-dimensionalen Kugeln vom Radius ε, mit denen A überdeckt werden kann. Die fraktale Dimension von A ist dann definiert als $\dim A = \lim_{\varepsilon \to 0} \log_{\frac{1}{\varepsilon}} N_A(\varepsilon)$.

Bemerkung. Eine d-dimensionale Kugel um den Ursprung ist definiert als $\{(x_1, x_2, ..., x_d) \,|x_1^2 + x_2^2 + ... + x_d^2 = \varepsilon^2\}$. Die Erweiterung auf beliebige Kugeln enthält man durch Ersetzen von x_n durch $(x_n - c_n)$, wobei c_n die Verschiebung der Kugel in Richtung n ist.

Beispiele

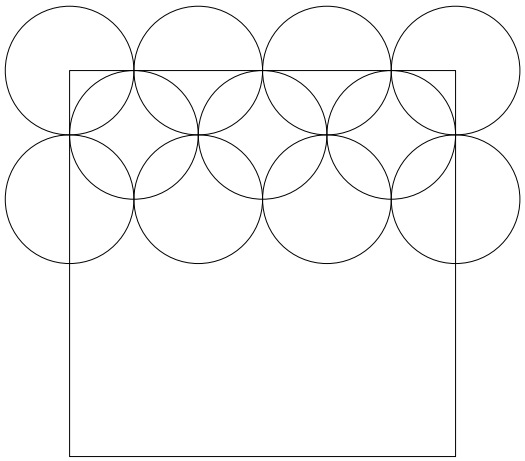
Strecke $A \subset \mathbb{R}^1$ (genauso für $\mathbb{R}^d$):



Länge $l \Rightarrow N_A(\varepsilon) = \frac{\lceil l \rceil}{2\varepsilon} = c_A \cdot \frac{1}{\varepsilon}$, wobei c_A eine Konstante ist, die von A abhängt. Die fraktale Dimension der Strecke A ist nach Definition

$\dim A = \lim_{\varepsilon \to 0} \log_{\frac{1}{\varepsilon}} c_A \cdot \frac{1}{\varepsilon} = \lim_{\varepsilon \to 0} \log_{\frac{1}{\varepsilon}} c_A + 1 = 1$, da $\log_{\frac{1}{\varepsilon}} c_A = \frac{\ln c_A}{ln\left(\frac{1}{\varepsilon}\right)}$ und $\varepsilon \to 0$.

Quadrat $A \subset \mathbb{R}^2$, Seitenlänge S

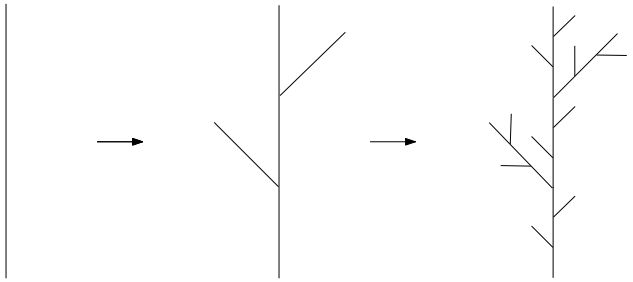


$N_A(\varepsilon) = \frac{\lceil l \rceil}{2\varepsilon} = c_A \cdot \frac{1}{\varepsilon^2}$

6.1.1 Bedeutung für Computergraphik

Fraktale als CG-Objekte: Modellierung von Selbstähnlichkeiten in der Natur, z.B. Blumenkohl, Pflanzen allgemein, felsiges Gebirge, Wolken etc.

Beispiel: Zeichnen eines Astes mittels einer Iterationsvorschrift



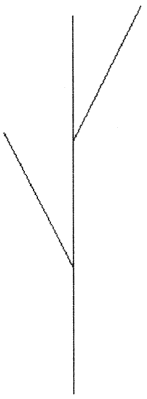
Das Bild wird definiert durch die Rekursionstiefe und den Astwinkel. Ein realistischeres Ergebnis erhält man, wenn man zusätzlich eine Wachstumswahrscheinlichkeit miteinbezieht. Allgemein kann man zufällige Veränderungen bezüglich gewissen Wahrscheinlichkeitsverteilungen in die Iterationsvorschrift einbauen.

Programmierbeispiel:

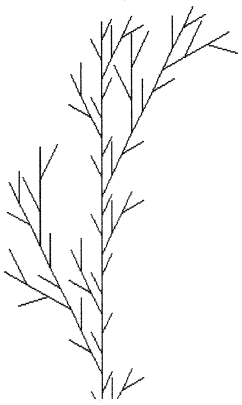
```
program tree;

   uses Graph;

   var k,GraphDriver,GraphMode: integer;
       alpha,c,s : real;


   procedure Bau(k:integer;  px,py,qx,qy:real);

        var dx,dy,ux,uy,vx,vy,wx,wy,zx,zy: real;

        begin if k=0 then line(trunc(px),trunc(py),trunc(qx),trunc(qy)) else

           begin dx:=(qx-px)/3;dy:=(qy-py)/3;
                 ux:=px+dx; uy:=py+dy;
                 vx:=ux+dx; vy:=uy+dy;
                 dx:=1.5*dx; dy:=1.5*dy;
                 wx:=ux+c*dx+s*dy; wy:=uy-s*dx+c*dy;
                 zx:=vx+c*dx-s*dy; zy:=vy+s*dx+c*dy;

                 Bau(k-1,px,py,ux,uy);
                 Bau(k-1,ux,uy,vx,vy);
                 Bau(k-1,vx,vy,qx,qy);
                 Bau(k-1,ux,uy,wx,wy);
                 Bau(k-1,vx,vy,zx,zy)
           end;
         end;


   begin
      GraphDriver:=0; InitGraph(GraphDriver,GraphMode,'/turbopas');

      write('Rekursionstiefe? Astwinkel(Grad)?:');
      read(k,alpha);
      c:= cos(alpha/180*3.14); s:=1-c*c;

      Bau(k,315,470,315,0)

   end.
```

Ergebnisse:

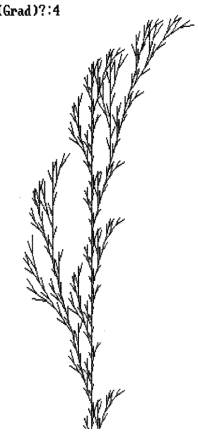
Rekursionstiefe? Astwinkel(Grad)?:1 45



Rekursionstiefe? Astwinkel(Grad)?:3
44



Rekursionstiefe? Astwinkel(Grad)?:4
34



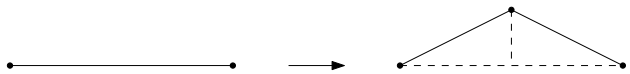
Rekursionstiefe? Astwinkel(Grad)?Wachstumswahrscheinlichkeit?:6 45 0.9

Rekursionstiefe? Astwinkel(Grad)?Wachstumswahrscheinlichkeit?:6 45 0.6

Beispiel Gebirge

Die Strecke wird in zwei Teilstrecken unterteilt, deren y-Koordinate zufällig

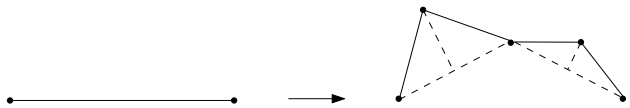
variiert.



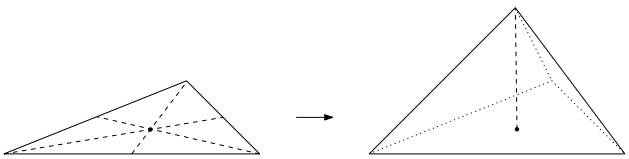
Die Koordinaten des mittleren Punktes sind dann

$x_{neu} = \frac{x_i + x_{i+1}}{2}$, $y_{neu} = P\left(x_{i-1} - x_i\right) \cdot R\left(x_{neu}\right)$

wobei $\mathbb{R} \in [0, 1]$ eine Zufallszahl. Als Ergebnis erhält man ein weniger regelmäßiges Gebilde:



Analoges Vorgehen ist auch im 3-Dimensionalen möglich: jede Dreiecksfläche könnte z.B. in eine Pyramide mit der Spitze über dem Mittelpunkt der Grundfläche ausgebaut werden, wobei die Höhe der Pyramide zufällig ist.



Die Vorgehensweise ist auch geeignet zur Modellierung von Objekten, die in der Grobstruktur regelmäßig und im Detail zufällig sind (z.B. Blumenbeete).

"Rendering" (Wiedergabe) von Fraktalen ist oft schwierig und zeitraubend (hoher Aufwand z.B. für Raytracing und HSE). Deswegen definiert man oft einfachere umschließende Objekte und bildet Fraktale als Textur im Bildraum ab.

6.1.2 Erinnerung: Kontextfreie Grammatiken

$G = (V, T, P, S)$, wobei

- V Nichtterminalzeichen
- T Terminalzeichen
- $P \subset V \times (V \cup T)^*$ Produktionen $A \to \alpha$
- $S \in V$ Startsymbol (Axiom)

Ableitung: $S \Rightarrow_G \alpha_1 \Rightarrow_G ... \Rightarrow_G \alpha_n \; S \Rightarrow_G^* w \in T^* :\Longleftrightarrow w \in L(G)$

Beispiele: $E \to E + E \,|E * E|\, (E) \,|a|\, b$
$a * ((a + b) * (b + a))$

$S \to (S) \,|SS|\, \varepsilon$

L-Systeme

(Lindenmayer-System, 1968)

Produktionen wie bei kontextfreien Grammatiken. Die Terminalzeichen sind $(,) , [,]$ (können auch mehr sein). In jedem Ableitungsschritt müssen sämtliche Nichtterminalsymbole ersetzt werden.

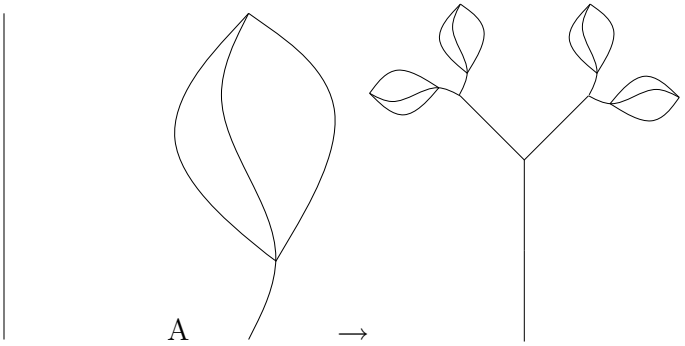
z.B. Axiom A, $A \to B\,[A]\,(A)$, $B \to BB$

Dies ist sogar ein *DOL*-System, also deterministisch: es gibt nur eine Produktion pro Variable.

$A \Rightarrow B\,[A]\,(A) \Rightarrow BB\,[B\,[A]\,(A)]\;(B\,[A]\,(A))$

Die Produktionen erzeugen Strings. Es ist also noch eine geometrische Interpretation notwendig, um die Objekte zeichen zu können.

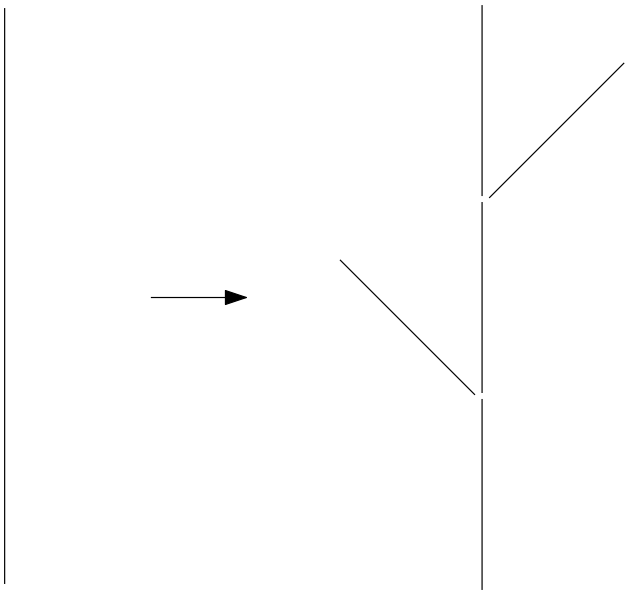
Dafü wird jedem Nichtterminalzeichen ein geometrisches Objekt zugeordnet, z.B.



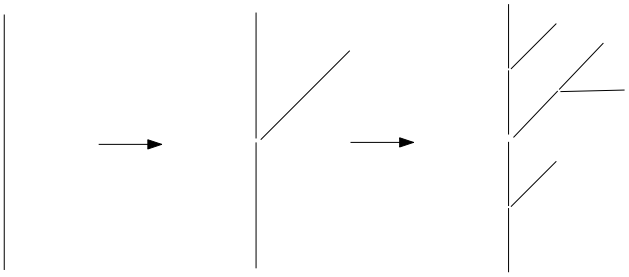
[...] bedeutet Verzweigung nach links,
(...) bedeutet Verzweigung nach rechts.

Lässt man das Objekt "von unten nach oben" wachsen, erhält man für $\Rightarrow BB\,[B\,[A]\,(A)]\;(B\,[A]\,(A))$ das obrige Bild.

Weitere Beispiele:



$A \to A\,(A)\,A\,[A]\,A$ fraktale Dimension: $\dim = \log_3 5 = 1,46..$



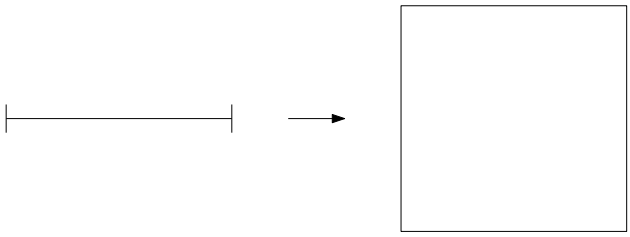
$A \to A\,(A)\,A$ $\dim = \log_2 3 = 1,58..$

Weitere Beispiele sind unter http://www.wooster.edu/cs/studentCoursework/theory/L_Systems2004.html und http://de.wikipedia.org/wiki/Lindenmayer-Systeme zu finden.

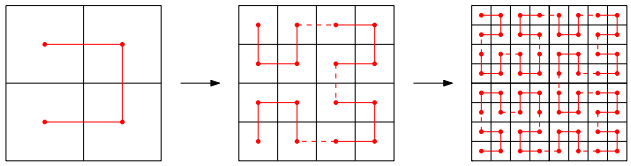
6.1.3 Raumfüllende Kurven

stetige (surjektive) Abbildungen $f\,[0,1] \rightarrow [0,1]^2$ (genauso für d Dimensionen)

Ziel: eine Kurve, die alle Punkte der Fläche enthält.

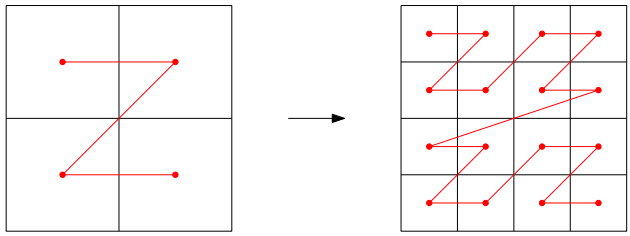


Beispiel: Hilbert-Kurve



Fraktale Dimension: 2. Durch unendliche Fortsetzung der Teilung erreicht man letztendlich jeden Punkt der Fläche. Nach k Verteilungen ist der Abstand eines Punktes von der Kurve höchstens 2^{-k}. Werden die Koordinaten eines beliebigen Punktes in der Fläche in Binärform gegeben, z.B. (x,y) = (0,1011, 0,0110), kann man die Lage des Punktes in den jeweiligen Vierteln, 16-tel etc. rekonstruieren. Bei der Verfeinerung der Kurve soll die Reihenfolge der besuchten Viertel beibehalten werden.

Eine ähnliche Kurve liefert die Z-Konstruktion, die jedoch Unstetigkeiten beinhaltet:



6.2 Scan-Konvertierung (Scan Conversion)

Scan-Konvertierung ist die Rasterung von einfachen Objekten (Geraden, Kreisen, Kurven). Als Ausgabemedium dient meist der Bildschirm, der aus einem Pixelraster besteht (Pixel haben ganzzahlige Koordinaten im Bildschirmkoordinatensystem, siehe Abbildung 6.1). Auf diesem Raster werden nun Objekte gezeichnet. Wie können wir zum Beispiel eine Gerade, die ja kontinuierlich ist, auf dem Bildschirmraster zeichnen, damit sie „gut“ aussieht? Wir nehmen an, dass die Pixel den Gitterpunkten entsprechen und entweder auf schwarz oder weiß (Startwert) gesetzt werden können. Wie „zeichnen“ wir

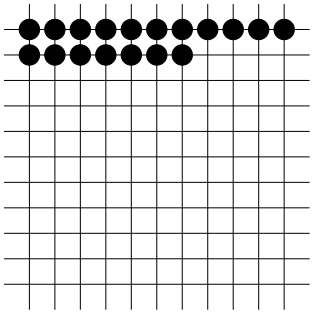


Abbildung 6.1: Pixel: Bildschirmkoordinaten als Gitterpunkte

eine Gerade $y = mx + B$? Annahme: wir zeichen „fein“, d.h. falls die Steigung $m \in [-1, 1]$ (Abbildung 6.2), schwärzen wir pro Spalte ein Pixel, sonst pro Zeile ein Pixel. Das Problem hierbei ist, dass steilere Geraden schwächer

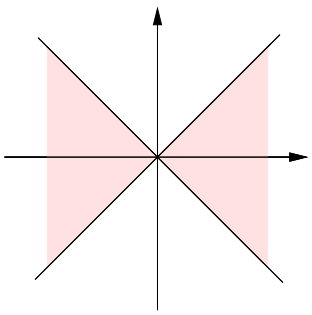


Abbildung 6.2: $m \in [-1, 1]$

abgebildet werden als flachere Geraden. Die Lösung ist, dass wir die Pixel nicht schwarz, sondern je nach Steigung mit einem entsprechenden Grauwert einfärben. Somit sehen alle Geraden gleich intensiv aus (Abbildung 6.3).

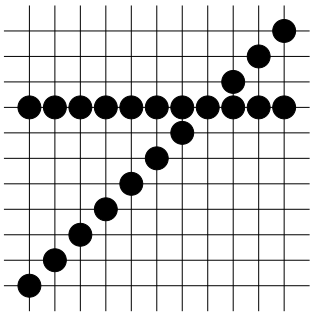


Abbildung 6.3: Verschiedene Intensitäten

6.2.1 Zeichnen einer Geraden

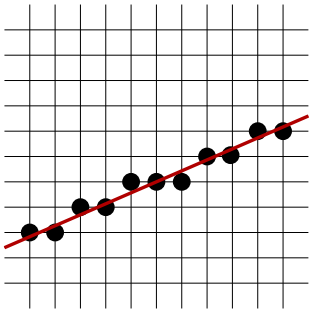


Abbildung 6.4: Gerade durch Pixel dargestellt

Naiver Ansatz

Ein naiver Algorithmus zum Zeichnen einer Geraden sieht folgendermaßen aus. Für die Steigung $m \in [-1, 1]$ (der andere Fall ist symmetrisch, einfach die x_i mit den y_i vertauschen) sind die $x_0, x_1, x_2, ...$ ganzzahlig und $x_{i+1} = x_i + 1$ ($\Delta x = 1$, da die Spalten aufeinander folgen). Die Gleichung $y_i = mx_i + B$ wird mit Gleitkomma-Rechnung berechnet und auf den nächsten ganzzahligen Wert gerundet. Wir schwärzen die Punkte $(x_i, \widetilde{y_i})$, wobei $\widetilde{y_i}$ der nächste ganzzahlige Wert zu y_i ist. Jedoch haben wir hier zwei Gleitkomma-Operationen und Rundungen pro Iteration, das dauert einfach zu lange.

Besserer Ansatz

Eine Verbesserung im Algorithmus ergibt sich, wenn wir uns den Wert des Vorgängers merken.

$$\begin{aligned} y_{i+1} &= mx_i + B \\ &= m(x_i + 1) + B \\ &= y_i + m \end{aligned}$$

Hier haben wir nur noch eine Gleitkomma-Addition und Rundung pro Iteration. Im nächsten Abschnitt werden wir sehen, dass man es noch besser machen kann.

6.2.2 Bresenham-Scan (Variante)

Gerade

Wir nehmen an, dass $m \in [0, 1]$, der anderer Fall ist symmetrisch. Die Gerade ist gegeben durch zwei Punkte (x_o, y_0) und (x_1, y_1), diese beiden Punkte haben jeweils ganzzahlige Bildschirmkoordinaten. In Abbildung 6.5 ist der Punkt P ein bereits gezeichneter Punkt der Geraden. Die Punkte E und NE sind die Kandidaten für den nächsten zu zeichnenden Punkt. M ist der Mittelpunkt zwischen NE und E. Der Punkt Q ist der Schnittpunkt der Geraden mit der Strecke $\overline{ENE}$. Welcher der beiden Punkte wird nun

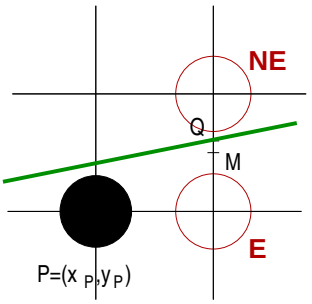


Abbildung 6.5: Gerade zeichnen mit Bresenham-Scan

genommen? E, wenn Q zwischen E und M liegt, NE sonst. In unserem Fall also der Punkt NE.

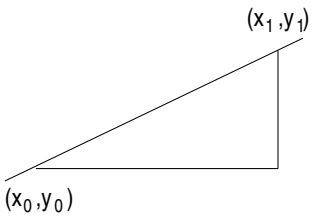


Abbildung 6.6: Gerade durch zwei Punkte gegeben

Implizite Form der Geraden Zum Berechnen der Geradengleichung

$$F(x, y) = ax + by + c = 0$$

betrachten wir die Abbildung 6.6 und setzen

$$\begin{aligned} a &= -dy = y_0 - y_1 \\ b &= dx = x_1 - x_0 \\ c &= B * dx \end{aligned}$$

Bis auf B, welches noch zu berechnen bleibt, sind die Werte bekannt. Dann ist

$$F(x, y) \begin{cases} < 0, \text{ falls}(x, y)\text{über der Geraden liegt} \\ = 0, \text{ falls}(x, y)\text{auf der Geraden liegt} \\ > 0, \text{ falls}(x, y)\text{unter der Geraden liegt} \end{cases}$$

Wir definieren eine Entscheidungsvariable d:

$$\begin{aligned} d := F(M) &= F(x_p + 1, y_p + \frac{1}{2}) \\ &= a(x_p + 1) + b(y_p + \frac{1}{2}) + c \end{aligned}$$

Jetzt wird das Vorzeichen von d getestet. Ist $d > 0$, wählen wir NE, ist $d \leqslant 0$, wählen wir E als nächsten Punkt. Wir überlegen uns, wie sich M und d im nächsten Schritt ändern? Hierzu Abbildung 6.7.

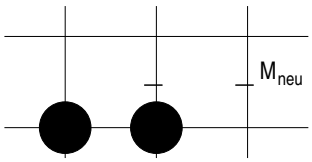


Abbildung 6.7: Nächster Schritt

Falls E gewählt wurde $\Rightarrow$

$$\begin{aligned} d_{neu} &= F(x_p + 2, y_p + \frac{1}{2}) \\ &= a(x_p + 2) + b(y_p + \frac{1}{2}) + c \\ &= d_{alt} + a \\ &= d_{alt} - dy \end{aligned}$$

Wir erhalten d_{neu} aus d_{alt} mit einer ganzzahligen Addition. $\Delta_E := -dy$.

Falls NE gewählt wurde $\Rightarrow$

$$\begin{aligned} d_{neu} &= F(x_p + 2, y_p + \frac{3}{2}) \\ &= a(x_p + 2) + b(y_p + \frac{3}{2}) + c \\ &= d_{alt} + a + b \\ &= d_{alt} + dx - dy \end{aligned}$$

$\Delta_{NE} := dx - dy$. Da dx und dy ganzzahlige Konstanten sind, können wir Δ_{NE} vorher berechnen, erhalten also auch hier eine ganzzahlige Addition.

Initialisierung: bei (x_0, y_0) („linkestes“ Pixel der Geraden im Bild) ist der

- erste M-Punkt: $(x_0 + 1, y_0 + \frac{1}{2})$
- erste Wert von d : $F(x_0{+}1, y_0{+}\frac{1}{2}) = F(x_0, y_0){+}a{+}\frac{b}{2}$, mit $F(x_0, y_0) = 0$, da (x_0, y_0) auf der Geraden liegt. Also ist $d_{start} = a + \frac{b}{2} = \frac{dx}{2} - dy$, was im Allgemeinen nicht ganzzahlig ist.

Wenn wir statt d einfach $d' = 2d$ benutzen, umgehen wir das Problem. Wir prüfen ja nur, ob d positiv oder negativ ist. Als Initialisierung also: $d'_{start} = dx - 2dy$ und als Inkrement:

$$\begin{aligned} \Delta_E &= -2dy \\ \Delta_{NE} &= 2(dx - dy) \end{aligned}$$

6.2.3 Kreis

Wir betrachten oBdA Kreise um den Ursprung, denn durch die nun gegebene Symmetrie ergeben sich einige Erleichterungen bei der Berechnung des Kreises.

$$x^2 + y^2 = R^2$$
$$y = \pm\sqrt{R^2 - x^2}$$

Zur Vereinfachung lassen wir $-\sqrt{R^2 - x^2}$ weg und bekommen den Halbkreis.

$$y = +\sqrt{R^2 - x^2}$$

Im nächsten Schritt betrachten wir nur den Viertelkreis

$$y = +\sqrt{R^2 - x^2}, x \in [0, R]$$

. Im Grunde reicht es aus, den oberen Achtelkreis im ersten Quadranten zu betrachten (siehe Abbildung 6.8). Um einen Kreis zu zeichnen, zeichnen wir mit jedem (x, y) insgesamt acht Punkte: $(\pm x, \pm y)$, $(\pm y, \pm x)$. Die Steigung

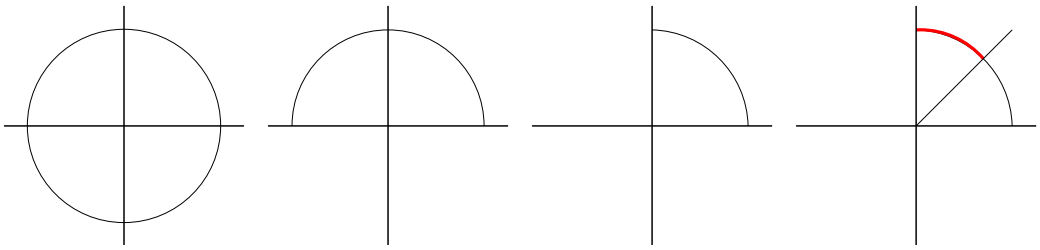


Abbildung 6.8: Kreis, Halbkreis, Viertel- und Achtelkreis

des Achtelkreises liegt zwischen -1 und 0, d.h. wir zeichnen ein Pixel pro Spalte (s.o.). In Abbildung 6.9 ist P das zuletzt gezeichnete Pixel. E und SE sind die Möglichkeiten für das nächste Pixel, die drei Pixel in der Spalte ganz rechts die Möglichkeiten für das übernächste Pixel.

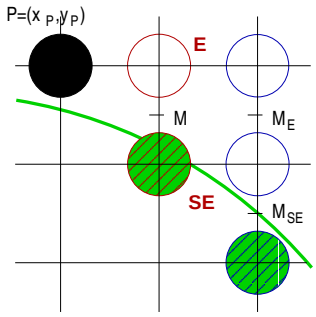


Abbildung 6.9: Kreiszeichnen

$$F(x,y) = x^2 + y^2 - R^2 \begin{cases} = 0 (x,y) \text{auf dem Kreis} \\ > 0 (x,y) \text{außerhalb des Kreises} \\ < 0 (x,y) \text{innerhalb des Kreises} \end{cases}$$

Wir fragen uns, ob M innerhalb oder außerhalb des Kreises liegt? Falls $F(M) > 0 \Rightarrow SE$ ist „näher“ am Kreis.

$$d_{alt} := F(M) = (x_p + 1)^2 + (y_p - \frac{1}{2})^2 - R^2 \begin{cases} < 0 \rightarrow E \text{ ist nächster Punkt} \\ \geqslant 0 \rightarrow SE \text{ ist nächster Punkt} \end{cases}$$

Falls E als nächster Punkt gewählt wurde $\Rightarrow$

$$\begin{aligned} d_{neu} = F(M_E) &= F(x_p + 2, y_p - \frac{1}{2}) \\ &= F(M) + 2(x_p + 1) + 1 \\ &= F(M) + 2x_p + 3 \end{aligned}$$

Mit $F(M) = d_{alt}$ und $\Delta_E = 2x_p + 3$. Im Gegensatz zur Geraden ist Δ_E hier keine Konstante sondern eine Funktion, die von x abhängig ist.

Falls SE als nächster Punkt gewählt wurde $\Rightarrow$

$$\begin{aligned} d_{neu} = F(M_{SE}) &= F(x_p + 2, y_p - \frac{3}{2}) \\ &= F(M) + 2(x_p + 1) + 1 - 2(y_p - \frac{1}{2}) + 1 \\ &= F(M) + 2x_p - 2y_p + 5 \\ &= d_{alt} + 2x_p - 2y_p + 5 \end{aligned}$$

In diesem Fall ist $\Delta_{SE} = 2x_p - 2y_p + 5$.

Initialisierung: (Annahme R ist positiv ganzzahlig)

$$\begin{aligned} P_0 &= (0, R) \\ M &= (1, R - \frac{1}{2}) \\ d_{start} = F(M) &= 1 + (R - \frac{1}{2})^2 - R^2 = \frac{5}{4} - R \end{aligned}$$

Problem: $\frac{5}{4}$ ist nicht ganzzahlig. Wir nehmen stattdessen eine neue Entscheidungsvariable d' mit $d' = d - \frac{1}{4}$ daraus folgt als Initialwert $d_{start} = 1 - R$. Test ob $d < 0 \Leftrightarrow d' < -\frac{1}{4} \Leftrightarrow d' < 0$ da d' ganzzahlig ist.

Differenzen 2. Ordnung

Wie oben schon gesagt ändern sich beim Kreis die Δ_E und Δ_{SE} in Abhängigkeit der aktuellen Pixelposition. Das bedeutet einen erhöhten Aufwand bei der Berechnung. Zur Beschleunigung des Algorithmus erstellen wie für die Inkremente die Differenzen 2. Ordnung.

Falls E gewählt wurde $\Rightarrow (x_p, y_p) \rightarrow (x_p + 1, y_p)$ mit $\Delta_E = 2x_p + 3$

$$\Delta_{Eneu} = 2(x_p + 1) + 3 = \Delta_E + 2$$

Der Unterschied zwischen den Δ_E ist eine Konstante (2. Ableitung), d.h. wir brauchen nur eine ganzzahlige Addition zur Aktualisierung von Δ_E. Analog: $\Delta_{SE} = 2x_p - 2y_p + 5$

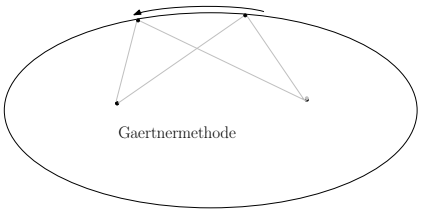
$$\Delta_{SEneu} = 2(x_p + 1) - 2y_p + 5 = \Delta_{SE} + 2$$

Falls SE gewählt wurde $\Rightarrow (x_p, y_p) \rightarrow (x_p + 1, y_p - 1)$, dann

$$\begin{aligned} \Delta_{Eneu} &= \Delta_E + 2 \\ \Delta_{SEneu} &= \Delta_{SE} + 4 \end{aligned}$$

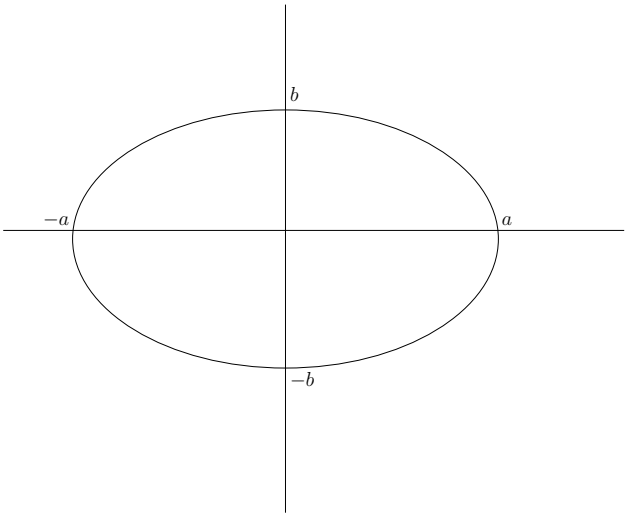
damit benötigen wir pro Iteration eine ganzzahlige Addition für je Δ_E, Δ_{SE}, dann berechnen wir noch d. Im Wesentlichen sind das drei ganzzahlige Additionen.

6.2.4 Ellipsen



Ellipsen mit Mittelpunkt 0 und achsenparallel (o.B.d.A.) sind gegeben durch folgende Gleichung:

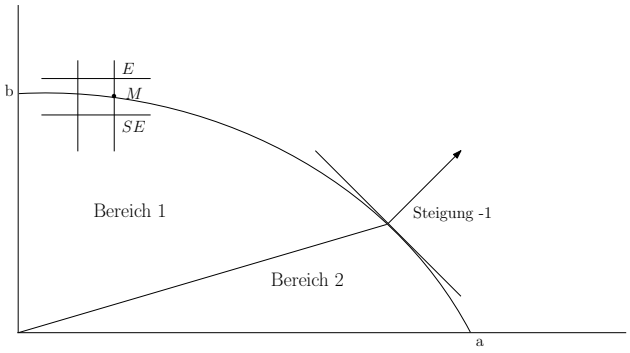
$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$$



Daraus ergibt sich folgende implizite Form:

$$F(x, y) = b^2x^2 + a^2y^2 - a^2b^2$$

wobei $F(x, y) = 0$ gerade die Punkte auf der Ellipse beschreibt. Für $F(x, y) < 0$ liegen die Punkte innerhalb und für $F(x, y) > 0$ außerhalb dieser. Es genügt hier aus Symmetriegründen wieder, nur die Viertelellipse im 1. Quadranten zu betrachten.



Bereich

- **Bereich 1:** Hier ist $\frac{\partial F}{\partial y} \geq \frac{\partial F}{\partial x}$
- **Bereich 2:** Hier ist $\frac{\partial F}{\partial y} \leq \frac{\partial F}{\partial x}$

Wir sind beim Übergang von **1** nach **2**, falls $2a^2y \leq 2b^2x$. Wir wechseln zum Punkt P, falls

$$a^2(y_P - 1/2) \leq b^2(x_P + 1) \qquad (1)$$

mit $d = F(M)$. Also: falls $d < 0$ gehe nach E, sonst gehe nach SE.
Im Bereich **1** ist für $d_{alt} = F(x_P + 1, y_P - 1/2)$

$$d_{neu} = \begin{cases} F(x_P + 2, y_P - 1/2) = d_{alt} + \underbrace{b^2(2x_P + 3)}_{=:\Delta_E} & (2) \\ F(x_P + 2, y_P - 3/2) = d_{alt} + \underbrace{b^2(2x_P + 3) + a^2(-2y_P + 2)}_{=:\Delta_{SE}} & (3) \end{cases}$$

wobei der erste Fall eintritt, falls der nächste Punkt E ist, der zweite, falls der nächste Punkt SE ist.
Initialisierung:

$$d = F(1, b - 1/2) = b^2 + a^2(b - 1/2)^2 - a^2b^2 = \underbrace{b^2 + a^2(-b + 1/4)}_{abrunden}$$

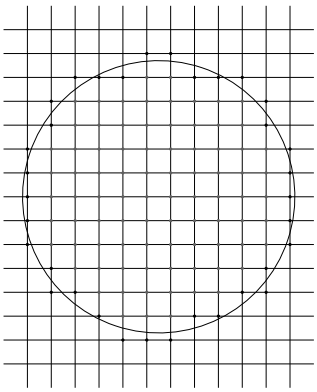
Der Algortihmus funktioniert also insgesamt wie folgt:

- d initialisieren
- Iteration
  falls $d < 0$: gehe nach E und aktualisiere gemäß (2)
  sonst: gehe nach SE und aktualisiere gemäß (3)
  prüfe, ob (1) noch erfüllt ist

- wenn in 2 angekommen
  d neu initialisieren
  analog zu oben verfahren (nun mit SE und S)

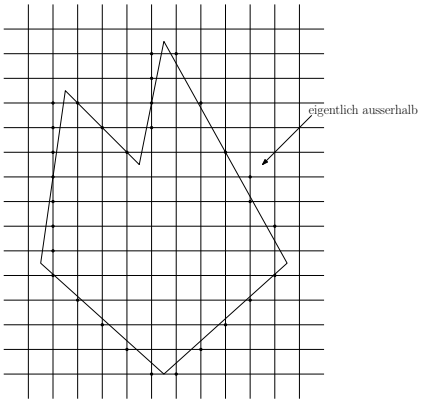
Auch hier können wir, wie beim Kreis, mit Differenzen zweiter Ordnung arbeiten. Dies ist sinnvoll bei d-Werten und auch beim Gradienten in (1).

6.3 Ausfüllen von Objekten

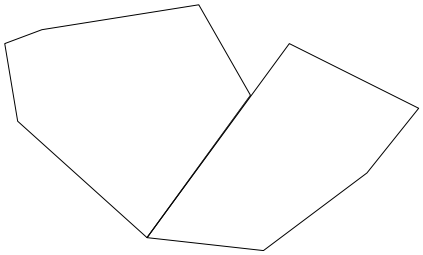


Allgemeine Idee: Zum Ausfüllen eines Objektes mit konstanter Farbe oder Grauwert: Mit Scanmethoden den Rand zeichnen, dann Zeile für Zeile die dazwischenliegenden Pixel färben.
Bei nicht konvexen Objekten müssen wir die Zwischenräume abwechselnd färben und nicht färben.

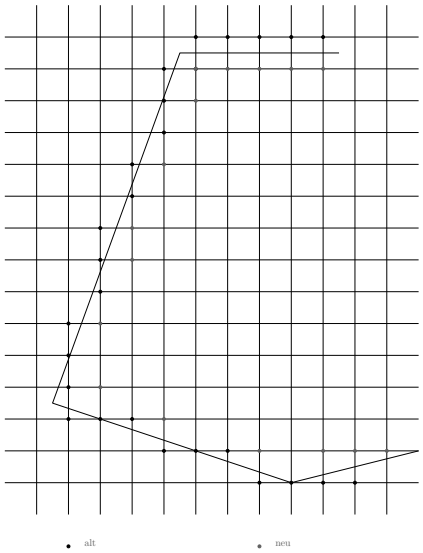
6.3.1 Polygone



Hier werden wir die einzelnen Kanten Scan-konvertieren und wie eben beschrieben verfahren. Dabei werden aber möglicherweise Pixel, die eigentlich außerhalb der Figur liegen, gezeichnet.

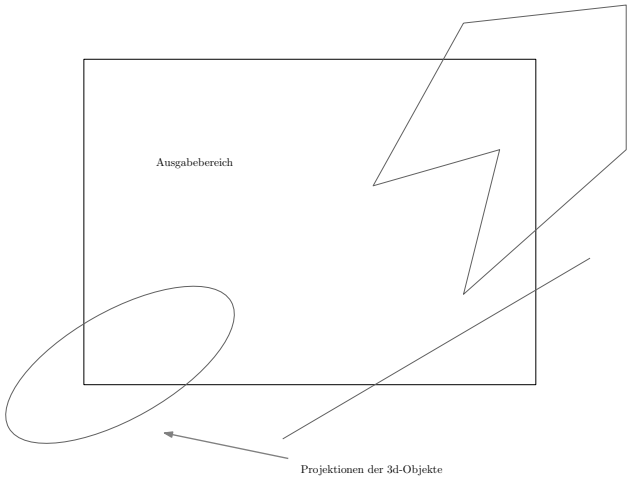


Dann können Probleme entstehen, wenn zwei oder mehr Polygone in einer Kante zusammenstoßen. Deswegen werden wir beim Scan des Randes nur Pixel innerhalb des Polygons zeichnen. Wir müssen also den Scan-Algorithmus entsprechend modifizieren. (Randbemerkung, da trivial)



6.4 Clipping (Zuschneiden)

Versucht man z.B. in Java außerhalb des Fensters zu zeichnen, so wird dies erkannt und ignoriert. Es bleibt aber weiterhin eine Effizienzfrage.

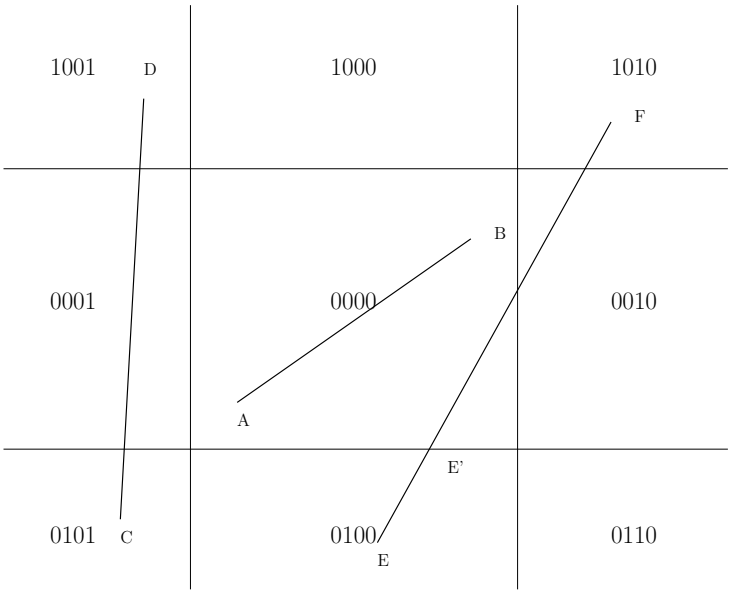


6.4.1 Zuschneiden von Geraden und Strecken

Hier benutzen wir den Algorithmus[1] von Cohen/Sutherland. Dazu teilen wir die Ebene in 9 Bereiche auf:

- 1. Bit 1: $y > y_{max}$
- 2. Bit 1: $y < y_{min}$
- 3. Bit 1: $x > x_{max}$
- 4. Bit 1: $x < x_{min}$

[1] eigentlich eher „den Trick...“



Je nach Signatur der Endpunkte der Strecke können wir nun

- Die Strecke trivial akzeptieren, falls die Signatur beider Endpunkte 0000 ist. (AB)
- Die Strecke trivial verwerfen, falls das komponentenweise „und“ beider Siganturen an mindestens einer Stelle eine 1 hat. (CD)
- Für alle anderen Strecken nehmen wir die Endpunkte außerhalb des Fensters und bestimmen mit Hilfe der Signatur die Begrenzungsgeraden, die geschnitten werden (max. 2). Nun berechnen wir die Schnittpunkte s_{p_i} mit diesen und teilen die Strecken dort auf. Dann fahren wir mit dem gleichen Verfahren fort. (EF)

6.5 Antialiasing

Die Rasterisierung eines Bildes erzeugt oft künstliche Effekte oder Muster (Artefakte).
Beispiele:

1. Treppeneffekt:
   Ein Beispiel für ein Verfremdung ist die Rasterisierung von Stecken auf Bildschirmen. Ein Bildschirm besteht aus endlich vielen Pixeln und macht es daher notwendig Geraden punktmäßig abzutasten. Verwendet man hierfür nur die beiden Zustände „Pixel liegt auf Strecke“ und „Pixel liegt nicht auf Strecke“ entstehen treppenartige Gebilde, welche die Strecken darstellen.

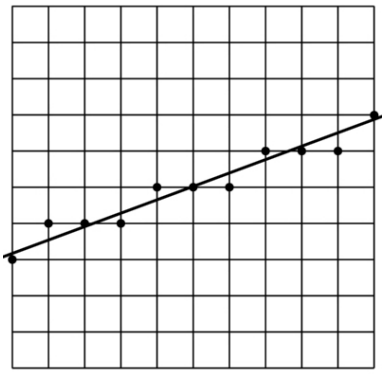
   

   Abbildung 6.10: Treppeneffekt

   Die Verfremdung entsteht dadurch, dass nur endlich viele Punkte abgetastet werden.

2. Kutschenräder im Western:
   In Westernfilmen kommt es öfter zu dem Phänomen, dass sich Kutschenräder scheinbar in die entgegengesetzte Richtung drehen. Das liegt daran, dass ein Film aus einer endlichen Menge von Bildern besteht und somit wieder nur endlich viele Punkte, in diesem Falle Zeitpunkte abgetastet werden. Drehen sich die Räder der Kutsche nun gerade so schnell, dass sich eine Speiche genau kurz vor der Position einer anderen Speiche im vorhergehenden Bild befindet, so sieht es so aus, als drehe sich das Rad andersherum.

3. ein unendliches Schachbrett:
   Betrachtet man ein unendlich großes Schachbrett, so sieht das Muster

im vorderen Bereich normal aus. Im hinteren Bereich, wo die Darstellung der Spielfelder immer kleiner wird, kommt es zur Bildung von neuen Mustern.

Solche Phänomene werden „Aliasing“ (Verfremdung) genannt.

Wie kann der Treppeneffekt behoben werden?
Pixel können auch mit Grauwerten zwischen Schwarz (0) und Weiß (1) gefärbt werden.
statt Gerade / Strecke:
Strich B mit gewisser Breite

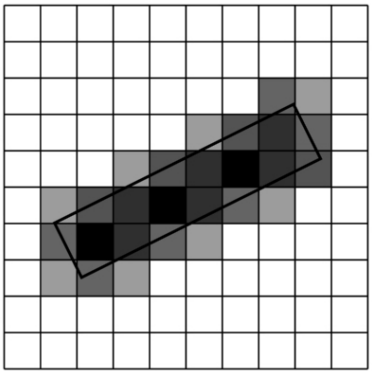


Abbildung 6.11: Anti-Aliasing

Pixel sind diesmal die Zwischenräume.

Idee: Der Grauwert einer Zelle Z ist gleich dem Anteil von $B \cap Z$ zur Fläche von Z. Dies wird als “unweighted area sampling“ bezeichnet.

unweighted: Die Zelle erhält den Grauwert nur nach dem Flächenanteil von $B \cap Z$ unabhängig davon, wie weit dieser Schnitt von Mittelpunkt der Zelle entfernt ist.

Beim „ **weighted** area sampling“ wird dies berücksichtigt.

Der Anteil von B ist in diesen beiden Zellen gleich. In der rechten Zelle sind jedoch die Punkte von $B \cap Z$ im Mittel weiter entfernt vom Zentrum als in der linken Zelle. Dies bedeutet, dass die linke Zelle dunkler gefärbt wird als die rechte.

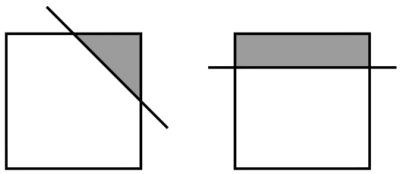


Abbildung 6.12: weighted area sampling

Falls das Gewicht $w(p)$ eines Punktes in $B \cap Z$ linear vom euklidischen Abstand zum Zentrum z abhängt, so bildet die Gewichtsfunktion einen **Kegel**.

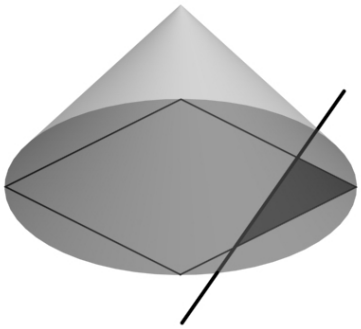


Abbildung 6.13: Kegel

formal:

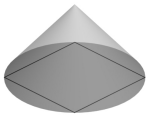
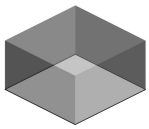
χ_B ist die charakteristische Funktion

$$\chi_B(p) = \begin{cases} 1 \text{ falls } p \in B \\ 0 \text{ falls } p \notin B \end{cases}$$

$$g(z) = \alpha \cdot \int_Z w(p) \cdot \chi_B(p) \, \mathrm{d}Z \tag{6.1}$$

$g(z)$: Grauwert, den Z erhält
α : Normierungsfaktor, damit $\in [0,1]$, also $\alpha \cdot \int_Z w(p) \, \mathrm{d}Z = 1 \Rightarrow \alpha = \frac{1}{\int w(p) dZ}$

unweighted area sampling fällt auch in dieses Schema für $w(p) = 1$:

Statt  haben wir  .

Besser wäre es, wenn wir w unabhängig von der Zelle Z definieren (Kegel um den Ursprung)

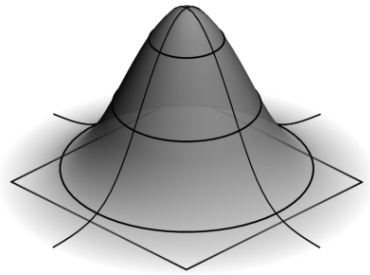
$$w(p) = \begin{cases} 1 - \|p\|_2 \text{ falls } \|p\|_2 \leq a \\ 0 \text{ sonst} \end{cases}$$

(1) wird dann zu

$$g(z) = \alpha \cdot \int_Z w(p - z) \cdot \chi_B(p) \, \mathrm{d}Z$$

$\frac{1}{\alpha} g = w * \chi_B$ heißt Faltung (Konvolution) von w und χ_B

Statt Box oder Kegel nimmt man oft auch die 2d - Gaußsche Normalverteilung:



$$w(p) = \frac{1}{2\pi\sigma^2} \cdot e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Zugrunde liegende Theorie (zunächst im Eindimensionalen):

Eine Funktion $f : \mathbb{R} \to \mathbb{R}$ [bei Bildern $f : \mathbb{R}^2 \to \mathbb{R}$] möchte man darstellen durch **Basisfunktionen** z. B. folgender Form:

$f_0(t) := 1$
$f_1(t), f_2(t), \ldots, f_n(t) := \sin(n \cdot t)$

Eine solche Funktion kann z.B. so aussehen:

$f(t) = \sum_{i=0}^{k} a_1 \cdot f_i(t)$

Ein besserer Ansatz ist jedoch der Übergang zu den Komplexen Zahlen $\mathbb{C}$:

$$f : \mathbb{R} \to \mathbb{C}$$

Wir nehmen Basisfunktionen der Form $f_\omega(t) = e^{i2\pi\omega t}$, $\omega \in \mathbb{R}$
ω ist hierbei die Frequenz.

$e^{i\varphi} = \cos\varphi + i\sin\varphi$

Die Funktion f soll dargestellt werden als Linearkombination der $f_\omega(t)$:

$f(t) = \int_{-\infty}^{\infty} F(\omega) \cdot e^{i2\pi\omega t} \mathrm{d}\omega$
$e^{i2\pi\omega t}$: Basisfunktion mit Frequenz ω
$F(\omega)$: zugehöriger Koeffizient

Alle „interessanten“ Funktionen sind auf diese Art darstellbar.

Es gilt:

$$F(\omega) = \int_{-\infty}^{\infty} f(t) \cdot e^{-i2\pi\omega t} \mathrm{d}t$$

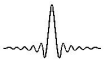
F heißt **Fourrier-Transformierte** von f.

$F = \Im(f) \qquad f = \Im^{-1}(F)$

$F(\omega)$ heißt Koeffizient von f bezüglich der Frequenz ω und besagt, „wie stark diese Frequenz in f vorkommt“.
F heißt auch (Frequenz-)**Spektrum** von f.

Es gilt (Faltungssatz):

$$\Im(f * g) = \Im(f) \cdot \Im(g)$$

$sinc(u) = \frac{\sin(u)}{u}$ ist die Fourrier-Transformierte von  , d.h. Faltung mit bewirkt Multiplikation der Frequenz mit  nach dem Faltungssatz, d.h. tiefe Frequenzen bleiben erhalten und die hohen werden abgeschwächt. Dies wird auch als **Tiefpassfilter** bezeichnet.

Auf ein Bild bezogen stellen tiefe Frequenzen „sanfte“ Übergänge dar und hohe Frequenzen „abrupte“ Übergänge.

6.6 Animation

Kamera als bewegtes Objekt: Dieses Verfahren nennt man „walk through“ oder „fly-by“ und soll dem Betrachter das Gefühl geben die Szene zu durchschreiten oder durch sie hindurch zu fliegen, eine Animation simuliert also die Bewegung eines Betrachters durch die Szene.

First-Person Computerspiele nutzen 4 bis 5 Freiheitgarde:

- Position des Augpunkt hat 2 bis 3 Freiheitgrade.
- Blickrichtung hat meistens 2 Freiheitsgrade.

Die Blickrichtung kann als Punkt auf 2d Sphäre dargestellt werden, s. Bild 6.14.

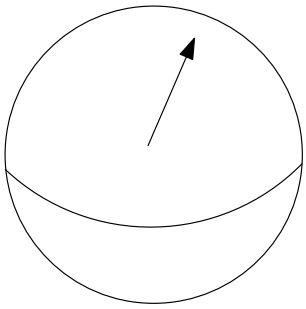


Abbildung 6.14:

Die Bewegungen der Kamera sind starre Bewegungen, dabei findet meist keine Rotation um die Blickrichtung statt.

Gegliederte Strukturen: Das Objekt wird modelliert durch starre Teile verbunden durch (verschiedenartige) Gelenke (s. Bild 6.15). Sie ergeben eine Hierarchie von Gliedern. Bewegungen beschreibt man relativ zum übergeordneten Glied in der Hierarchie.

Beispiel. Menschliches Bein (s. Bild 6.16)

- Hüfte bewegt sich innerhalb der Szene.
- Oberschenkel rotiert um die Hüfte (1 Freiheitsgrad beim Gehen)
- Unterschenkel rotiert um das Knie (1 Freiheitsgrad)

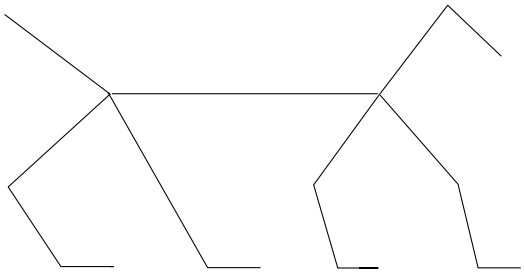


Abbildung 6.15:

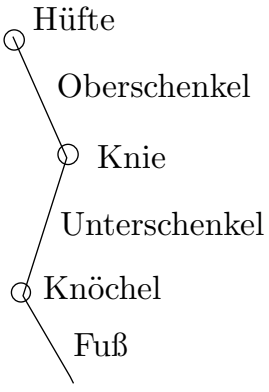
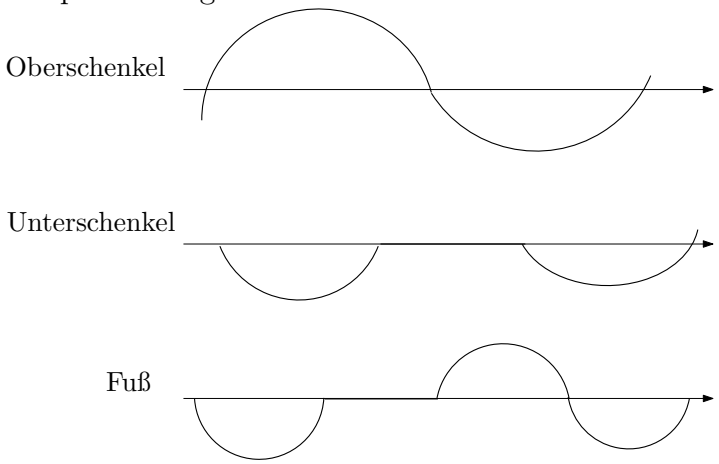


Abbildung 6.16:

- Fuß rotiert um den Knöchel (1 Freiheitsgrad beim Gehen)

Animation: in der Hierarchie von oben nach unten relative Bewegungen angeben.

Skript für ein gehendes Bein:



Oft gibt es noch zusätzliche Schwerigkeiten durch biegsame Glieder, z.B. Wirbelsäule

Zeitliches Aliasing: Bsp: Speichen eines Wagenrads. (s. Bild 6.17)

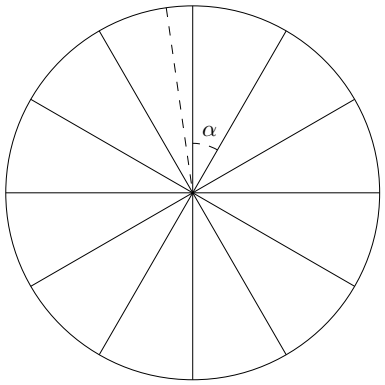


Abbildung 6.17:

Falls das Rad gerade so schnell ist, dass sich eine Speiche um α zwischen zwei Frames bewegt, so scheint das Rad zu stehen. In unserem Beispiel sind es 12 Speichen, also bei 24 fps Aufnahme und 2 Umdrehungen pro Sekunde würde das Rad zu stehen scheinen. Falls Geschwindigkeit ein wenig langsamer ist, scheint es rückwärts zu laufen.

Zeitliches Aliasing kann man vermeinden durch:

- Tiefpassfilter (auch zeitlich möglich),
- mit höherer Frequenz aufnehmen als wiedergeben.

6.7 OpenGL

OpenGL (Open Graphics Library) ist eine Spezifikation für ein plattform- und programmiersprachenunabhängiges API (Application Programming Interface) zur Entwicklung von 3D-Computergrafik. Diese Systemunabhängigkeit hat zur Folge, dass sich die Bibliothek nicht z.B. um die Verwaltung von Zeichenoberflächen (Fenster) kümmert. Diese müssen mit Hilfe dafür vorgesehener, betriebssystemabhängiger Bibliotheken zur Verfügung gestellt werden.

Der OpenGL-Standard beschreibt etwa 250 Befehle, die die Darstellung komplexer 3D-Szenen in Echtzeit erlauben. Einfache Objekte wie Punkte, Strecken sowie Polygone sind schon vordefiniert. Mit einer kleinen Erweiterung von OpenGL ist es auch möglich Kugeln und Zylinder zu rendern.

Die verschiedenen benötigten Projektionen sind vorhanden und das Zeichnen auf dem Bildschirm, das hidden surface removal, ist durch das sogenannte Culling implementiert (es gibt auch einen Z-Buffer). Zudem sind Gourand- sowie auch Flat-Shading implementiert. Beim Flat Shading gibt man jeder Fläche (Polyeder) eine Intensität.

Die Erscheinung der Objekte wie Transparenz, Farbe und Textur kann definiert werden. Zudem kann die 3D-Szene mit Lichtquellen bestückt werden.

Die folgende Viewing Pipeline zeigt vereinfacht welche Schritte in OpenGL zur Darstellung einer 3d - Szene gemacht wird.

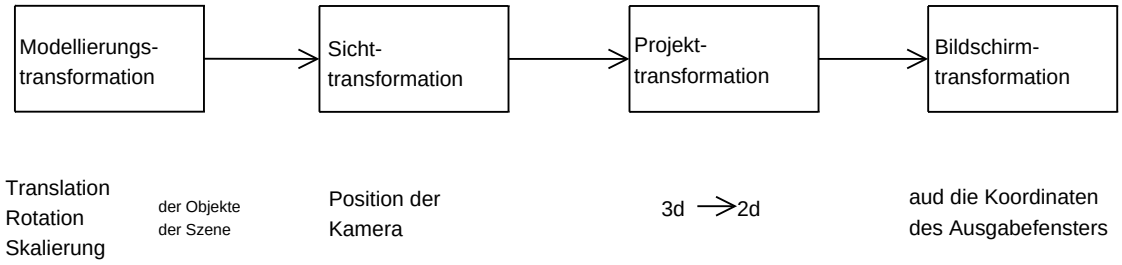


Abbildung 6.18: Viewing Pipeline

Die Modellierung der Objekte geschieht hierarchisch. Wenn man z.B. einen Tisch erzeugen will, so definiert man einen Quader durch die Koordinaten seiner Ecken und bringt ihn durch Translation und Skalierung in die gewünschte Position. Die Tischbeine können dann durch Zylinder modelliert werden, die ebenfalls an die gewünschte Position gebracht werden. All dies ist implementiert mithilfe eines Stacks und dargestellt durch eine Transformationsmatrix.

Es gibt auch die Möglichkeit der Animation (durch wiederholte Neupositionierung der Objekte) und Interaktion mit dem Anwender. Die Bibliothek stellt Funktionen bereit, die dies bequem unterstützen.