

# Computergrafik

Dr. Johannes Riesterer

22. April 2019



©Johannes Riesterer  
Vervielfältigung nur mit ausdrücklicher Erlaubnis des Autors

## **Vorwort**

Dieses Skript entstand aus einer Reihe von Vorlesung über Computergrafik, die ich sowohl an der dualen Hochschule in Stuttgart als auch an der Hochschule für Kunst Design und populäre Musik Freiburg gehalten habe.

# Inhaltsverzeichnis

<b>1</b>	<b>Mathematische Werkzeuge</b>	<b>5</b>
1.1	Lineare Algebra . . . . .	5
1.1.1	Vektoren und Matrizen . . . . .	5
1.2	Affiner Raum und affine Abbildungen . . . . .	13
1.2.1	Homogene Koordinaten und Projektionen . . . . .	15
1.3	Parameterdarstellungen . . . . .	17
1.4	Wahrscheinlichkeitstheorie und Monte Carlo Integration . . . . .	20
1.4.1	Markov Chain Monte Carlo . . . . .	20
1.4.2	Metropolis Hastings Algorithmus . . . . .	20
<b>2</b>	<b>Physikalische Grundlagen</b>	<b>20</b>
2.1	Photometrie . . . . .	20
2.2	Farbwahrnehmung und Farbmodelle . . . . .	24
<b>3</b>	<b>Kurven, Flächen und Netze</b>	<b>24</b>
3.1	Polygone, Netze und Elemente der diskreten Geometrie . . . . .	25
3.2	Datenstrukturen . . . . .	32
3.2.1	Subdivision . . . . .	35
3.3	Modellierung . . . . .	35
3.3.1	Bezier Kurven und Flächen . . . . .	35
<b>4</b>	<b>Echtzeitvisualisierung und OpenGL</b>	<b>40</b>
4.1	Geschichte . . . . .	40
4.2	GL-Pipeline . . . . .	40
4.2.1	Opengl Shaderpipeline . . . . .	42
4.2.2	Geometrie . . . . .	43
4.2.3	Rasterung . . . . .	45
4.3	Lokale Beleuchtungsmodelle . . . . .	50
4.3.1	Ideale und diffuse Reflexionen . . . . .	50
4.3.2	Lambert Modell . . . . .	50
4.3.3	Phong Modell . . . . .	51
4.4	Shader und standard Algorithmen . . . . .	54
4.4.1	Mittelung der Normalen . . . . .	54
4.4.2	Flat-Shading, Gouraud und Phong Shading . . . . .	54
4.4.3	Texturen und UV-Mapping . . . . .	55
4.4.4	Bumpmapping . . . . .	56
4.4.5	Displacementmapping . . . . .	57
4.4.6	Shadowmap . . . . .	58
4.4.7	Forward shading, Blending und Transparenz . . . . .	59
4.4.8	Deferred shading . . . . .	60
4.5	GLSL via WebGL . . . . .	62
4.5.1	Minimaler Shader . . . . .	62

<b>5 Raytracing</b>	<b>66</b>
5.1 Raytracing Algorithmen . . . . .	67
5.1.1 Die Rendergleichung (Erste und zweite Form) . . . . .	67
5.1.2 Pfadformulierung der Rendergleichung . . . . .	68
5.2 Raycasting . . . . .	68
5.3 "Klassisches"Raytracing . . . . .	68
5.4 BRDF-Shader . . . . .	69
5.5 Radiosity Verfahren . . . . .	70
5.6 Monte-Carlo Raytracing und Pathtracing . . . . .	73
5.7 Monte Carlo Methode . . . . .	73
5.7.1 Raymarching . . . . .	73
5.8 Klassifikation der Verfahren . . . . .	73
5.9 Datenstrukturen für Bereichsabfragen . . . . .	73
5.10 Labor . . . . .	73
5.10.1 Blender . . . . .	73
5.10.2 Echtzeitfähiges Raymarching in WebGL . . . . .	73
<b>6 Animation und Simulation</b>	<b>73</b>
6.1 Keyframe Animation . . . . .	73
6.2 Partikelsysteme . . . . .	73
6.3 Elemente der Kollisionserkennung . . . . .	73
6.4 Labor . . . . .	73
<b>Tabellenverzeichnis</b>	<b>74</b>
<b>Abbildungsverzeichnis</b>	<b>75</b>

# 1 Mathematische Werkzeuge

## 1.1 Lineare Algebra

### 1.1.1 Vektoren und Matrizen

Wir wollen zunächst den Vektorraum  $\mathbb{R}^n$  einführen. Hierbei ist  $n$  eigentlich immer 2, 3 oder 4. Zunächst ist der  $\mathbb{R}^n$  eine Menge, nämlich die Menge der  $n$ -dimensionalen Vektoren

$$\mathbb{R}^n := \left\{ \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \mid x_1, x_2, \dots, x_n \in \mathbb{R} \right\}.$$

Auf dieser Menge der Vektoren definiert man die Addition

$$+ : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}^n$$

$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} + \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} := \begin{pmatrix} x_1 + y_1 \\ x_2 + y_2 \\ \vdots \\ x_n + y_n \end{pmatrix}$$

und die sogenannte Skalarmultiplikation

$$\cdot : \mathbb{R} \times \mathbb{R}^n \rightarrow \mathbb{R}^n$$

$$\lambda \cdot \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} := \begin{pmatrix} \lambda \cdot x_1 \\ \lambda \cdot x_2 \\ \vdots \\ \lambda \cdot x_n \end{pmatrix}.$$

Das Element  $\lambda \in \mathbb{R}$  nennt man auch Skalar.

**Definition 1.** Der Vektorraum  $\mathbb{R}^n$  ist das Tripel  $(\mathbb{R}^n, +, \cdot)$ .

**Beispiel 1.**

$$\begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} + \begin{pmatrix} 3 \\ 2 \\ 1 \end{pmatrix} = \begin{pmatrix} 4 \\ 4 \\ 4 \end{pmatrix}$$

$$\begin{pmatrix} 1 \\ 1 \end{pmatrix} + \begin{pmatrix} -1 \\ -1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

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

$$-1 \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} -1 \\ -1 \\ -1 \end{pmatrix}$$

$$\pi \cdot \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix} = \begin{pmatrix} \pi \\ 2\pi \\ 3\pi \\ 4\pi \end{pmatrix}$$

**Bemerkung 1.** Für alle Skalare  $\lambda, \mu \in \mathbb{R}$  und Vektoren  $u, v \in \mathbb{R}^n$  gelten die Rechenregeln

$$\begin{aligned}\lambda \cdot (u + v) &= \lambda \cdot u + \lambda \cdot v \\ (\lambda + \mu) \cdot u &= \lambda \cdot u + \mu \cdot u.\end{aligned}$$

**Definition 2.** Für ein  $u, v \in \mathbb{R}^n$  sind die folgenden Kurz-Notationen üblich

$$\begin{aligned}-u &:= -1 \cdot u \\ u - v &:= u + (-v) := u + (-1 \cdot v)\end{aligned}$$

**Definition 3.** Eine  $n \times m$  Matrix ist ein Objekt der Form

$$(a_{ij})_{ij} := \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,m} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,m} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{i,1} & a_{i,2} & a_{i,3} & \cdots & a_{i,m} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n,1} & a_{n,2} & a_{n,3} & \cdots & a_{n,m} \end{pmatrix}$$

mit  $a_{i,j} \in \mathbb{R}$  für alle  $i = 1, \dots, n$  und  $j = 1, \dots, m$ .

**Bemerkung 2.** Ein Vektor der Dimension  $n$  ist eine  $n \times 1$ -Matrix.

**Definition 4.** Ist  $A = (a_{ij})_{ij}$  eine  $n \times m$  und  $B = (b_{kl})_{kl}$  eine  $m \times p$  Matrix so ist das Matrizenprodukt definiert als die  $n \times p$  Matrix

$$A \cdot B := \left( \sum_{j=1}^m a_{ij} \cdot b_{jl} \right)_{il}.$$

Sind  $A$  und  $B$  zwei  $n \times m$ -Matrizen so ist ihre Summe definiert durch

$$A + B := \left( a_{ij} + b_{ij} \right)_{ij}.$$

Für ein  $\lambda \in \mathbb{R}$  definieren wir die Skalarmultiplikation

$$\lambda \cdot A := \left( \lambda \cdot a_{ij} \right)_{ij}.$$

**Definition 5.** Die  $n$ -dimensionale Einheitsmatrix ist definiert durch

$$I_n := \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 \\ \vdots & & \ddots & & \vdots & \vdots \\ \vdots & & & \ddots & 0 & 0 \\ 0 & 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 1 \end{pmatrix} .$$

**Bemerkung 3.** Sind  $A$  und  $B$  beides  $n \times n$ -Matrizen, so ist im Allgemeinen

$$A \cdot B \neq B \cdot A .$$

Für die  $n$ -te Einheitsmatrix  $I_n$  gilt jedoch immer

$$A \cdot I_n = I_n \cdot A = A .$$

**Definition 6.** Für eine  $2 \times 2$ -Matrix definieren wir die Determinante

$$\det : M^{n \times n} \rightarrow \mathbb{R}$$

$$\det \left( \begin{pmatrix} a & b \\ c & d \end{pmatrix} \right) := ad - bc$$

**Satz 1.** Für eine  $n \times n$  Matrix  $A = (a_{ij})_{ij}$  definieren wir die Determinante durch die Rekursionsformel

$$\det(A) = \sum_{j=1}^n (-1)^{i+j} a_{ij} \det(A_{ij})$$

und  $\det(a) = a$  für eine  $1 \times 1$ -Matrix  $a$ . wobei  $A_{ij}$  die Matrix ist, die aus  $A$  durch Streichen der  $i$ -ten Zeile und der  $j$ -ten Spalte entsteht. Diese Definition ist unabhängig von der Wahl von  $i$ . (Entwickeln nach der  $i$ -ten Zeile).

**Satz 2.** Für alle  $n \times n$ -Matrizen  $A, B$  gilt

$$\det(A \cdot B) = \det(A) \cdot \det(B)$$

**Satz 3.** Sei  $A$  eine  $n \times n$  Matrix. Dann existiert genau dann eine Matrix  $A^{-1}$  mit

$$A \cdot A^{-1} = A^{-1} \cdot A = I_n ,$$

wenn  $\det(A) \neq 0$ .  $A^{-1}$  ist eindeutig bestimmt.

**Bemerkung 4.** Ist  $v$  ein  $n$ -dimensionaler Vektor und  $A$  eine  $m \times n$ -Matrix, so ist  $A \cdot v$  ein  $m$ -dimensionaler Vektor.

**Definition 7.** Ist  $A := (a_{ij})_{ij}$  eine  $n \times m$ -Matrix, so heißt die  $m \times n$ -Matrix  $A^t := (a_{ji})_{ij}$  die transponierte Matrix.

**Definition 8.** Ist insbesondere  $v = \begin{pmatrix} x_1 \\ \vdots \\ x_k \end{pmatrix}$  ein  $k$ -dimensionaler Vektor, so heißt  $v^t = (x_1 \quad \cdots \quad x_n)$  der transponierte Vektor, welcher auch eine  $1 \times n$ -Matrix ist.

**Satz 4.** Für alle  $n \times m$ -Matrizen  $A$  und  $m$ -dimensionale Vektoren  $u, v$  gilt

$$A \cdot (\lambda \cdot u + \mu \cdot v) = \lambda \cdot A \cdot u + \mu \cdot A \cdot v .$$

**Definition 9.** Vektoren  $v_1, \dots, v_k \in \mathbb{R}^n$  heißen linear unabhängig, falls für  $\lambda_i \in \mathbb{R}$ ,  $i = 1, \dots, k$  mit

$$\sum_{i=1}^k \lambda_i \cdot v_i = 0$$

stets  $\lambda_i = 0$  folgt für alle  $i = 1, \dots, k$ .

**Satz 5.** Die Vektoren  $v_1, \dots, v_k \in \mathbb{R}^n$  sind genau dann linear abhängig, wenn man mit Hilfe des Gaußalgorithmus in der Matrix

$$\begin{pmatrix} v_1^t \\ \hline \vdots \\ \hline v_k^t \end{pmatrix}$$

eine Nullzeile erzeugen kann.

**Bemerkung 5.** Für  $k > n$  sind  $v_1, \dots, v_k \in \mathbb{R}^n$  stets linear abhängig.

**Definition 10.** Für Vektoren  $v_1, \dots, v_k \in \mathbb{R}^n$  heißt die Menge

$$\text{span}(v_1, \dots, v_k) := \left\{ \sum_{i=1}^k \lambda_i \cdot v_i \mid \lambda_i \in \mathbb{R} \right\} \subseteq \mathbb{R}^n$$

der von ihnen aufgespannte lineare Unterraum. Diese Definition ist offensichtlich unabhängig von der Reihenfolge. Eine Menge von Vektoren  $\{w_1, \dots, w_l\}$  heißt Basis von  $\text{span}(v_1, \dots, v_k)$ , falls  $w_1, \dots, w_l$  linear unabhängig sind und  $\text{span}(w_1, \dots, w_l) = \text{span}(v_1, \dots, v_k)$  gilt.  $l$  heißt dann auch die Dimension von  $\text{span}(v_1, \dots, v_k)$ .

**Satz 6.** Ist  $B := \{b_1, \dots, b_n\}$  eine Menge linear unabhängiger  $n$ -dimensionaler Vektoren, so ist

$$\text{span}(b_1, \dots, b_n) = \mathbb{R}^n .$$

Wir nennen dann die geordnete Menge  $B$  eine Basis des  $\mathbb{R}^n$ .

**Definition 11.** Wir bezeichnen die Basis  $E := \{e_1, \dots, e_n\}$  des  $\mathbb{R}^n$  mit

$$e_i := \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \leftarrow i\text{-te Stelle}$$

als Standardbasis des  $\mathbb{R}^n$ .

**Definition 12.** Sei  $B := \{b_1, \dots, b_n\}$  eine Basis des  $\mathbb{R}^n$  und  $v \in \mathbb{R}^n$ . Dann gibt es nach dem letzten Satz Skalare  $\lambda_1, \dots, \lambda_n \in \mathbb{R}$ , so dass sich  $v$  als Linearkombination

$$v = \sum_{i=1}^n \lambda_i \cdot b_i$$

ausdrücken lässt. Schreibt man diese  $\lambda_i$  wieder in einen Vektor, so erhalten wir eine Abbildung

$$\begin{aligned} \theta_B : \mathbb{R}^n &\rightarrow \mathbb{R}^n \\ \begin{pmatrix} v_1 \\ \vdots \\ v_n \end{pmatrix} &\mapsto \begin{pmatrix} \lambda_1 \\ \vdots \\ \lambda_n \end{pmatrix}. \end{aligned}$$

Man nennt  $\theta_B(v)$  die Darstellung von  $v$  zur Basis  $B$ .

**Bemerkung 6.** Für die Standardbasis  $E$  des  $\mathbb{R}^n$  ist  $\theta_E(v) = v$  für alle  $v \in \mathbb{R}^n$ , also  $\theta_S = id$ .

**Definition 13.** Sei  $B := \{b_1, \dots, b_n\}$  eine Basis des  $\mathbb{R}^n$  und  $v \in \mathbb{R}^n$ . Dann definieren wir

$$M_B = \left( b_1 \mid b_2 \mid \dots \mid b_n \right)^{-1}.$$

**Satz 7.** Sei  $B := \{b_1, \dots, b_n\}$  eine Basis des  $\mathbb{R}^n$  und  $v \in \mathbb{R}^n$ . Dann ist

$$\theta_B(v) = M_B \cdot v$$

**Definition 14.** Seien  $B := \{b_1, \dots, b_n\}$  und  $B' := \{b'_1, \dots, b'_n\}$  zwei Basen des  $\mathbb{R}^n$ . Dann heißt  $M_B^{B'} := M_{B'} \cdot M_B^{-1}$  die Basiswechselmatrix von  $B$  nach  $B'$ . Wir haben also folgende Situation:

$$\begin{array}{ccc} \mathbb{R}^n & \xleftarrow{M_B^{-1}} & \mathbb{R}^n \\ \downarrow I_n & & \downarrow M_B^{B'} \\ \mathbb{R}^n & \xrightarrow{M_{B'}} & \mathbb{R}^n \end{array}$$

**Definition 15.** Die Abbildung

$$\langle \cdot, \cdot \rangle: \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}$$

$$\left\langle \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}, \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} \right\rangle := \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}^t \cdot \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} = (x_1 \quad \cdots \quad x_n) \cdot \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} = \sum_{i=1}^n x_i \cdot y_i$$

heißt Skalarprodukt.

**Satz 8.** Für alle  $u, v, w, l \in \mathbb{R}^n$  und  $\lambda, \mu, \tau, \nu \in \mathbb{R}$  gilt

$$\begin{aligned} \langle \lambda u + \mu v, \tau w \rangle &= \lambda \tau \langle u, w \rangle + \mu \tau \langle v, w \rangle \\ \langle \lambda u, \tau w + \nu l \rangle &= \lambda \tau \langle u, w \rangle + \lambda \nu \langle u, l \rangle \end{aligned}$$

**Definition 16.** Die Abbildung

$$\begin{aligned} \|\cdot\|: \mathbb{R}^n &\rightarrow \mathbb{R} \\ \|v\| &:= \sqrt{\langle v, v \rangle} \end{aligned}$$

heißt Norm.

**Definition 17.** Zwei vom Nullvektor verschiedene Vektoren  $u, v \in \mathbb{R}^n$  heißen orthogonal, falls  $\langle u, v \rangle = 0$  ist. Man sagt auch sie stehen senkrecht aufeinander und benutzt auch die Bezeichnung  $u \perp v$ .

**Satz 9.** Der Kosinus des Innenwinkel  $\varphi$  zweier Vektoren  $u, v$  lässt sich durch

$$\cos(\varphi) = \frac{\langle u, v \rangle}{\|u\| \cdot \|v\|}$$

**Definition 18.** Ein Vektor  $v \in \mathbb{R}^n$  heißt normal, falls  $\|v\| = 1$  ist. Ist  $w \in \mathbb{R}^n$  ein beliebiger Vektor, so heißt  $\frac{1}{\|w\|}w$  die Normalisierung von  $w$ .

**Definition 19.** Eine Basis  $B := \{b_1, \dots, b_n\}$  heißt Orthonormalbasis (kurz ONB), falls

$$\langle b_i, b_j \rangle = \begin{cases} 1 & \text{falls } i = j \\ 0 & \text{sonst} \end{cases}$$

gilt. Insbesondere sind alle  $b_i$  normal.

**Algorithmus 1.** Seien  $v_1, \dots, v_n$  linear unabhängige Vektoren. Dann lässt sich daraus durch folgenden Algorithmus eine ONB generieren:

$$\begin{aligned} b'_i &:= v_i - \sum_{j=1}^{i-1} \langle v_i, b_j \rangle b_j \\ b_i &:= \frac{1}{\|b'_i\|} b'_i \end{aligned}$$

und Rekursionsanfang  $b'_1 = v_1$ .

Im  $\mathbb{R}^3$  gibt es eine besonders einfache Methode aus zwei Vektoren einen Vektor zu generieren, der auf den Ausgangs-Vektoren senkrecht steht.

**Definition 20.** Für  $u = \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix}$  und  $v = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix}$  heißt

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

das Kreuzprodukt von  $u$  und  $v$ .

**Bemerkung 7.** Es gilt

- $\langle u \times v, u \rangle = \langle u \times v, v \rangle = 0$
- $u \times v = -(v \times u)$
- $u \times v = 0$  genau dann, wenn  $u$  und  $v$  linear abhängig sind.

**Bemerkung 8.** Ist  $B := \{b_1, \dots, b_n\}$  eine ONB, so gilt

$$M_B^{-1} = M_B^t$$

**Definition 21.** Eine Matrix  $O \in \mathbb{M}^{n \times n}$  heißt orthogonal, falls  $O^{-1} = O^t$  ist.

**Satz 10.** Eine Matrix  $O \in \mathbb{M}^{n \times n}$  ist genau dann orthogonal, falls

$$\det(O) \in \{-1, 1\}.$$

Ist  $\det(O) = 1$ , so nennen wir  $O$  eine Drehung und  $SO(n) := \{O \in \mathbb{M}^{n \times n} \mid \det(O) = 1\}$  die Drehgruppe (oder auch spezielle orthogonale Gruppe).

**Satz 11.** Sei  $O \in \mathbb{M}^{n \times n}$  eine orthogonale Matrix, dann gilt für alles  $v, w \in \mathbb{R}^n$

$$\langle O \cdot v, O \cdot w \rangle = \langle v, w \rangle$$

und somit insbesondere

$$\|O \cdot v\| = \|v\|.$$

**Definition 22.** Eine  $2 \times 2$ -Drehmatrix ist eine Matrix der Form

$$\begin{pmatrix} \cos(\varphi) & \pm \sin(\varphi) \\ \mp \sin(\varphi) & \cos(\varphi) \end{pmatrix}$$

für ein  $\varphi \in [0, 2\pi]$ .

**Bemerkung 9.** Eine  $2 \times 2$ -Drehmatrix ist eine orthogonale Matrix.

**Definition 23.** Eine elementare  $3 \times 3$ -Drehmatrix ist eine Matrix der Form

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\varphi) & \pm \sin(\varphi) \\ 0 & \mp \sin(\varphi) & \cos(\varphi) \end{pmatrix}, \begin{pmatrix} \cos(\varphi) & 0 & \pm \sin(\varphi) \\ 0 & 1 & 0 \\ \mp \sin(\varphi) & 0 & \cos(\varphi) \end{pmatrix}, \begin{pmatrix} \cos(\varphi) & \pm \sin(\varphi) & 0 \\ \mp \sin(\varphi) & \cos(\varphi) & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

für ein  $\varphi \in [0, 2\pi]$ .

**Satz 12.** Jede Drehung  $O \in SO(3)$  lässt sich zerlegen in ein Produkt

$$O = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & \pm \sin(\phi) \\ 0 & \mp \sin(\phi) & \cos(\phi) \end{pmatrix} \cdot \begin{pmatrix} \cos(\psi) & 0 & \sin(\psi) \\ 0 & 1 & 0 \\ -\sin(\psi) & 0 & \cos(\psi) \end{pmatrix} \cdot \begin{pmatrix} \cos(\xi) & \sin(\xi) & 0 \\ -\sin(\xi) & \cos(\xi) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Die Winkel  $\phi, \psi, \xi$  heißen Eulerwinkel.

**Bemerkung 10.** Die Zerlegung  $O \in SO(3)$  einer Drehung in obiges Produkt ist nicht eindeutig. Ein anschauliches Beispiel dafür liefert der sogenannte "Gimbal lock".  $SO(3)$  ist also nicht das Produkt von drei Intervallen sondern es ist  $SO(3) = S^3 / \{\pm 1\}$ .



Abbildung 1: Kardansche Aufhängung und Gimbal lock

**Bemerkung 11.** Man kann bei der Produktzerlegung auch andere elementare Drehmatrizen (elementare Drehachsen) wählen, wobei eine spezielle Wahl zu den in der Luft und Raumfahrt verwendeten "Roll, Nick, Gier" Winkeln führen.



Abbildung 2: Roll, Nick und Gier Winkel

## 1.2 Affiner Raum und affine Abbildungen

Der Affine Raum  $\mathbb{A}^n$  ist ein Tupel

$$(\mathbb{R}^n, (\mathbb{R}^n, +, \cdot))$$

zusammen mit den Abbildung

$$\begin{aligned} - : \mathbb{R}^n \times \mathbb{R}^n &\rightarrow (\mathbb{R}^n, +, \cdot) \\ \overline{PQ} &:= Q - P \end{aligned}$$

und

$$\begin{aligned} + : \mathbb{R}^n \times (\mathbb{R}^n, +, \cdot) &\rightarrow \mathbb{R}^n \\ \begin{pmatrix} P_1 \\ \vdots \\ P_n \end{pmatrix} + \begin{pmatrix} v_1 \\ \vdots \\ v_n \end{pmatrix} &:= \begin{pmatrix} P_1 + v_1 \\ \vdots \\ P_n + v_n \end{pmatrix}. \end{aligned}$$

Die Elemente (Vektoren) aus  $\mathbb{R}^n$  nennt man auch Punkte in Abgrenzung zu den Vektoren aus  $(\mathbb{R}^n, +, \cdot)$ . Für Punkte  $P, Q \in \mathbb{R}^n$  ist also  $\overline{PQ}$  ein Vektor, auch Verbindungsvektor genannt.

**Definition 24.** Ist  $B := \{b_1, \dots, b_n\}$  eine Basis des Vektorraums  $(\mathbb{R}^n, +, \cdot)$  und  $P \in \mathbb{A}$  ein Punkt, so nennen wir das Tupel  $(P, B)$  eine affine Basis. Für jeden Punkt  $Q$  gibt es dann also Skalare  $\lambda_1, \dots, \lambda_n$  mit

$$Q = P + \sum_{i=1}^n \lambda_i \cdot b_i.$$

Der Punkt  $\begin{pmatrix} \lambda_1 \\ \vdots \\ \lambda_n \end{pmatrix}$  heißt die Darstellung von  $Q$  bezüglich der affinen Basis  $(P, B)$ .

**Definition 25.** Abbildungen der Form

$$\begin{aligned}\phi : \mathbb{A}^n &\rightarrow \mathbb{A}^n \\ \phi(P) &:= A \cdot P + t\end{aligned}$$

mit  $A \in M^{n \times n}$  und  $t \in (\mathbb{R}^n, +, \cdot)$  heißen affine Abbildungen. Insbesondere heißt eine affine Abbildung mit  $A = I_n$  und  $t \neq 0$  Translation.

**Bemerkung 12.** Eine Affine Abbildung

$$\begin{aligned}\phi : \mathbb{A}^n &\rightarrow \mathbb{A}^n \\ \phi(P) &:= A \cdot P + t\end{aligned}$$

ist genau dann invertierbar, falls  $\det(A) \neq 0$  ist und die Inverse Abbildung ist dann

$$\begin{aligned}\phi^{-1} : \mathbb{A}^n &\rightarrow \mathbb{A}^n \\ \phi^{-1}(P) &:= A^{-1} \cdot P - A^{-1} \cdot t.\end{aligned}$$

**Definition 26.** Sind  $(P, B := \{b_1, \dots, b_n\})$  und  $(P', B' := \{b'_1, \dots, b'_n\})$  zwei affine Basen und definieren wir die Abbildung

$$\begin{aligned}\theta_{(P,B)} : \mathbb{A}^n &\rightarrow \mathbb{A}^n \\ \theta_{(P,B)}(Q) &:= M_B \cdot Q - M_B \cdot P,\end{aligned}$$

so erhalten wir analog zu der Situation in Vektorräumen

$$\begin{array}{ccc} \mathbb{A}^n & \xleftarrow{\theta_{(P,B)}^{-1}} & \mathbb{A}^n \\ \downarrow id & & \downarrow \theta_{(P,B)}^{(P',B')} \\ \mathbb{A}^n & \xrightarrow{\theta_{(P',B')}} & \mathbb{A}^n \end{array}$$

mit  $\theta_{(P,B)}^{(P',B')}(Q) := \theta_{(P',B')}\left(\theta_{(P,B)}^{-1}(Q)\right)$ .

**Definition 27.** Der Abstand von  $P, Q \in \mathbb{A}$  ist definiert durch

$$\begin{aligned}d : \mathbb{A}^n \times \mathbb{A}^n &\rightarrow \mathbb{R}^n \\ d(P, Q) &:= \|\overline{PQ}\|.\end{aligned}$$

### 1.2.1 Homogene Koordinaten und Projektionen

**Definition 28.** Der projektive Raum ist definiert als

$$\begin{aligned}\mathbb{P}^n &:= \mathbb{R}^{n+1} - \{0\} / \sim \\ v \sim w &\Leftrightarrow v = \lambda w \text{ für ein } \lambda \neq 0 \in \mathbb{R}.\end{aligned}$$

**Satz 13.**  $\mathbb{P}^n := S^n / \{\pm 1\}$ ,  $\mathbb{P}^1 = S^1$ .

Wir haben die Abbildung

$$\begin{aligned}\mathbb{A}^n &\rightarrow \mathbb{P}^n \\ \begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_n \end{pmatrix} &\mapsto \begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_n \\ 1 \end{pmatrix}\end{aligned}$$

und nennen das Bild eines Punktes unter dieser Abbildung die homogenen Koordinaten. Auf der Menge der homogenen Koordinaten haben wir die Umkehrabbildung

$$\begin{aligned}\mathbb{P}^n - \left\{ \begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_n \\ p_{n+1} \end{pmatrix} \mid p_{n+1} = 0 \right\} &\rightarrow \mathbb{A}^n \\ \begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_n \\ p_{n+1} \end{pmatrix} &\mapsto \begin{pmatrix} \frac{p_1}{p_{n+1}} \\ \frac{p_2}{p_{n+1}} \\ \vdots \\ \frac{p_n}{p_{n+1}} \end{pmatrix}.\end{aligned}$$

Die Matrizenmultiplikation

$$\begin{aligned}\mathbb{R}^{n+1} &\rightarrow \mathbb{R}^{n+1} \\ v &\mapsto A \cdot v\end{aligned}$$

setzt sich wegen der Eigenschaft  $A \cdot (\lambda v) = \lambda A \cdot v$  zu einer Abbildung

$$\begin{aligned}\mathbb{P}^n &\rightarrow \mathbb{P}^n \\ p &\mapsto A \cdot p\end{aligned}$$

fort. Wir können damit und mit der Definition der Matrix-Vektor-Multiplikation eine affine Abbildung

$$\begin{aligned}\phi : \mathbb{R}^n &\rightarrow \mathbb{R}^n \\ \phi(v) &:= A \cdot v + t\end{aligned}$$

in homogenen Koordinate ausdrücken durch eine Matrizenmultiplikation

$$\begin{pmatrix} v \\ 1 \end{pmatrix} \mapsto \begin{pmatrix} A & t \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} v \\ 1 \end{pmatrix} .$$

**Definition 29.** *Die Abbildung*

$$persp_{xy} : \mathbb{A}^3 \rightarrow \mathbb{A}^2$$

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \mapsto \begin{pmatrix} \frac{X}{\frac{Z}{d}+1} \\ \frac{Y}{\frac{Z}{d}+1} \end{pmatrix}$$

die Zentralprojektion auf die  $X - Y$ -Ebene mit Augendistanz  $d$  beziehungsweise ist

$$\overline{persp}_{xy} : \mathbb{A}^3 \rightarrow \mathbb{A}^2$$

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \mapsto \begin{pmatrix} \frac{X}{\frac{Z}{d}} \\ \frac{Y}{\frac{Z}{d}} \end{pmatrix}$$

die Zentralprojektion auf die Ebene parallel zur  $X - Y$ -Ebene mit Abstand  $d$  und Augenpunkt im Ursprung.

Die Motivation folgt direkt aus dem Strahlensatz.

**Bemerkung 13.** Die Zentralprojektion auf die  $X - Y$ -Ebene mit Augendistanz  $d$  lässt sich nicht durch Multiplikation mit einer  $2 \times 3$ -Matrix realisieren.

Definieren wir die Matrizen

$$K_{persp_{xy}} := \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{d} & 1 \end{pmatrix}$$

und

$$K_{orth_{xy}} := \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

so können wir die Zentralprojektion auf die  $X - Y$ -Ebene mit Augendistanz  $d$  durch die Hintereinanderausführung folgender Abbildungen darstellen:

$$\begin{aligned} persp_{xy} : \mathbb{R}^3 &\rightarrow \mathbb{A}^3 \rightarrow \mathbb{A}^3 \rightarrow \mathbb{A}^2 \rightarrow \mathbb{R}^2 \\ \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} &\mapsto \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \mapsto K_{persp_{xy}} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ \frac{z}{d} + 1 \end{pmatrix} \\ &\mapsto K_{orth_{xy}} \cdot \begin{pmatrix} x \\ y \\ z \\ \frac{z}{d} + 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ \frac{z}{d} + 1 \end{pmatrix} \mapsto \begin{pmatrix} \frac{x}{\frac{z}{d} + 1} \\ \frac{y}{\frac{z}{d} + 1} \end{pmatrix} \end{aligned}$$

Analog für die andere Projektion mit der Matrix

$$K_{\overline{persp}_{xy}} := \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{d} & 0 \end{pmatrix}.$$

### 1.3 Parameterdarstellungen

**Definition 30.** Eine Kurve ist eine Abbildung

$$c : I \subset \mathbb{R} \rightarrow \mathbb{R}^3$$

$$c(t) := \begin{pmatrix} x(t) \\ y(t) \\ z(t) \end{pmatrix}$$

bei der die Funktionen  $x, y, z : I \rightarrow \mathbb{R}$  stetig sind. Sie heisst differenzierbar, falls  $x, y, z$  differenzierbar sind und die Ableitung ist dann definiert als

$$c' : I \subset \mathbb{R} \rightarrow \mathbb{R}^3$$

$$c'(t) := \begin{pmatrix} x'(t) \\ y'(t) \\ z'(t) \end{pmatrix}.$$

**Beispiel 2.** Die Kurve  $c : [0, 2\pi] \rightarrow \mathbb{R}^3$ ,  $c(t) := \begin{pmatrix} r \cos(t) \\ r \sin(t) \\ 0 \end{pmatrix}$  beschreibt einen Kreis mit Radius  $r$ , der in der XY-Ebene liegt. Die Ableitung ist

$$c'(t) = \begin{pmatrix} (r \cos(t))' \\ (r \sin(t))' \\ 0 \end{pmatrix} = \begin{pmatrix} -r \sin(t) \\ r \cos(t) \\ 0 \end{pmatrix}.$$

**Beispiel 3.** Die Kurve  $c : \mathbb{R} \rightarrow \mathbb{R}^3$ ,  $c(t) := \begin{pmatrix} \cos(t) \\ \sin(t) \\ t \end{pmatrix}$  beschreibt eine sogenannte Helix. Die Ableitung ist

$$c'(t) = \begin{pmatrix} \cos'(t) \\ \sin'(t) \\ t' \end{pmatrix} = \begin{pmatrix} -\sin(t) \\ \cos(t) \\ 1 \end{pmatrix}.$$



Abbildung 3: Eine Helix. Quelle:Wikipedia

**Definition 31.** Ist  $c : I \subset \mathbb{R} \rightarrow \mathbb{R}^3$  eine stückweise differenzierbare Kurve, so heißt

$$l(c) := \int_I ||c'(t)|| dt$$

ihre Länge.

**Definition 32.** Ein Fläche ist eine injektive Abbildung

$$s : U \subset \mathbb{R}^2 \rightarrow \mathbb{R}^3$$

$$s(u, v) := \begin{pmatrix} x(u, v) \\ y(u, v) \\ z(u, v) \end{pmatrix}$$

bei der die Abbildungen  $x, y, z : U \subset \mathbb{R}^2 \rightarrow \mathbb{R}$  stetig sind. Sie heißt differenzierbar, falls die partiellen Ableitungen

$$\frac{\partial}{\partial u} s(u, v) = \begin{pmatrix} \frac{\partial}{\partial u} x(u, v) \\ \frac{\partial}{\partial u} y(u, v) \\ \frac{\partial}{\partial u} z(u, v) \end{pmatrix}$$

und

$$\frac{\partial}{\partial v} s(u, v) = \begin{pmatrix} \frac{\partial}{\partial v} x(u, v) \\ \frac{\partial}{\partial v} y(u, v) \\ \frac{\partial}{\partial v} z(u, v) \end{pmatrix}$$

existieren. Die Ebene

$$T_s(u, v) := \{s(u, v) + \lambda \cdot \frac{\partial}{\partial u} s(u, v) + \mu \cdot \frac{\partial}{\partial v} \mid \lambda, \mu \in \mathbb{R}\}$$

heißt Tangentialebene am Punkt  $(u, v)$  und der Vektor

$$n(u, v) := \frac{\partial}{\partial u} s(u, v) \times \frac{\partial}{\partial v} s(u, v),$$

welcher Senkrecht auf dieser Ebene steht, die Normale.

**Definition 33.** Ist

$$\begin{aligned} s : U \subset \mathbb{R}^2 &\rightarrow \mathbb{R}^3 \\ s(u, v) &:= \begin{pmatrix} x(u, v) \\ y(u, v) \\ z(u, v) \end{pmatrix} \end{aligned}$$

eine Fläche und  $S := s(U)$  die von Ihr erzeugte Oberfläche, so definiert man das OberflächenIntegral

$$\int_S d\omega := \int_U \|n(u, v)\| \cdot dU.$$

Man nennt  $d\omega$  beziehungsweise  $\|n(u, v)\|$  das infinitessimale Flächenelement. Ist  $f : S \rightarrow \mathbb{R}$  eine Funktion, so definiert man analog

$$\int_S f d\omega := \int_U f(s(u, v)) \cdot \|n(u, v)\| \cdot dU.$$

**Satz 14.** Ist  $U = U_1 \times U_2 \in \mathbb{R}^2$  und  $f : U \rightarrow \mathbb{R}$  eine integrierbare Funktion, so gilt

$$\int_U f dU = \int_{U_1} \int_{U_2} f dU_2 dU_1.$$

**Beispiel 4.** Die Sphäre  $S^2 \subset \mathbb{R}^3$  ist definiert durch

$$\begin{aligned} s : [0, \pi] \times [0, 2\pi] &\rightarrow \mathbb{R}^3 \\ s(u, v) &:= \begin{pmatrix} \sin(u) \cos(v) \\ \sin(u) \sin(v) \\ \cos(u) \end{pmatrix} \end{aligned}$$

Das Volumen berechnet sich mit

$$\begin{aligned} \frac{\partial}{\partial u} s(u, v) &= \begin{pmatrix} \cos(u) \cos(v) \\ \cos(u) \sin(v) \\ -\sin(u) \end{pmatrix} \\ \frac{\partial}{\partial v} s(u, v) &= \begin{pmatrix} -\sin(u) \sin(v) \\ \sin(u) \cos(v) \\ 0 \end{pmatrix} \\ \left\| \frac{\partial}{\partial u} s(u, v) \times \frac{\partial}{\partial v} s(u, v) \right\| &= \sin(u) \end{aligned}$$

zu

$$\int_{S^2} d\omega = \int_{[0,\pi) \times [0,2\pi)} \sin(u) d(u \times v) = \int_{[0,2\pi)} \int_{[0,\pi)} \sin(u) du dv = 4\pi$$

## 1.4 Wahrscheinlichkeitstheorie und Monte Carlo Integration

**Definition 34.** Ein reller Wahrscheinlichkeitsraum ist ein Tupel  $(\Omega, \rho)$  bestehend aus

- Einer Menge  $\rho \subset \mathbb{R}^n$  deren Teilmengen Ereignisse genannt werden.
- Eine Funktion  $\mathbb{P} : \Omega \rightarrow \mathbb{R}$  mit  $\int_{\omega} \rho(\omega) d\omega = 1$  welche auch Wahrscheinlichkeitsdichte genannt wird.

Eine Abbildung  $X : \Omega \rightarrow \mathbb{R}$  wird Zufallsvariable genannt.

**Definition 35.** Ist  $X : \Omega \rightarrow \mathbb{R}$  eine Zufallsvariable, dann heißt

$$\mathbb{E}[X] := \int_{\Omega} X(\omega) \cdot \rho(\omega) d\omega \quad (1)$$

Erwartungswert.

**Satz 15** ((Schwaches) Gesetz der großen Zahlen). Ist  $X : \Omega \rightarrow \mathbb{R}$  eine Zufallsvariable mit Wahrscheinlichkeitsdichte  $\rho : \Omega \rightarrow [0, 1]$  und  $\{\omega_1, \dots, \omega_N\}$  eine Stichprobe für  $\rho$ , so gilt:

$$\frac{1}{N} \sum_{i=0}^N X(\omega_i) \xrightarrow{N \rightarrow \infty} \mathbb{E}[X] \text{ (in Wahrscheinlichkeit)} \quad (2)$$

**Satz 16.** Ist  $f : S \subset \Omega \rightarrow \mathbb{R}$  eine Funktion, so gilt für eine beliebige Wahrscheinlichkeitsdichte  $\rho : \Omega \rightarrow [0, 1]$  und eine Stichprobe  $\{\omega_1, \dots, \omega_N\}$

$$\frac{1}{N} \sum_{i=0}^N \frac{f(\omega_i)}{\rho(\omega_i)} \xrightarrow{N \rightarrow \infty} \int_{\Omega} f(\omega) d\omega \text{ (in Wahrscheinlichkeit).} \quad (3)$$

### 1.4.1 Markov Chain Monte Carlo

**Definition 36.** Markov Kette

### 1.4.2 Metropolis Hastings Algorithmus

## 2 Physikalische Grundlagen

### 2.1 Photometrie

Die Radiantergie  $Q$  ist die Lichtenergie. Sie wird durch einen Strom von Photonen erzeugt. Die Energie eines Photons ist durch  $E = h \cdot f$  geben, wobei  $h$

das konstante Planksche Wirkungsquantum und  $f$  die Frequenz der Welle ist (Welle-Teilchen Dualismus).

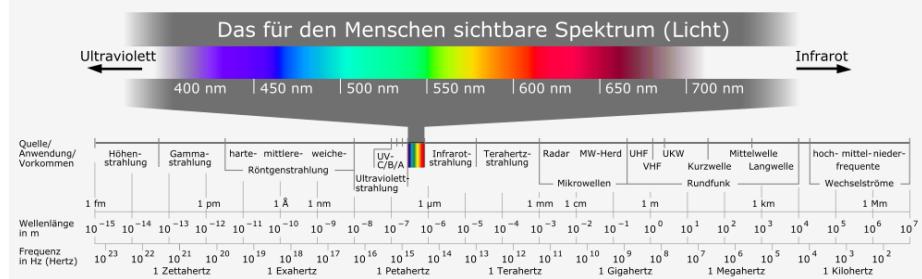


Abbildung 4: Spektrum

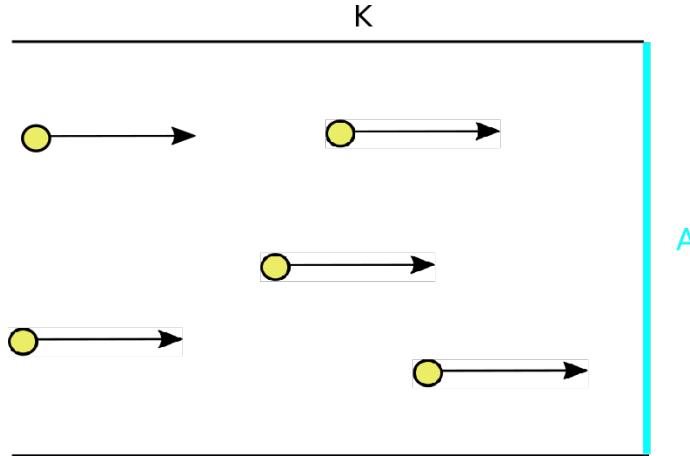


Abbildung 5: Partikelstrom in einem Kanal

Durch einen Kanal  $K$  bewegen sich Teilchen mit gleichförmiger Geschwindigkeit  $L$  (Lichtgeschwindigkeit) und Dichte  $\eta$ . Die Anzahl der Teilchen, die die Fläche  $A$  bezüglich eines Zeitintervall  $[0, t]$  passieren, ist gegeben durch

$$D_A([0, t]) := L(A) * \text{Flächeninhalt}(A) * t \quad (4)$$

$$L(A) := \eta * \|L\| \quad (5)$$

Betrachtet man die allgemeinere Situation eines Flächenstückes  $B$  in einem Teilchenstrom  $L$ , so ist die Anzahl gegeben durch

$$D_B([0, t]) := L(B) * \cos(\alpha) * \text{Flächeninhalt}(B) * t \quad (6)$$

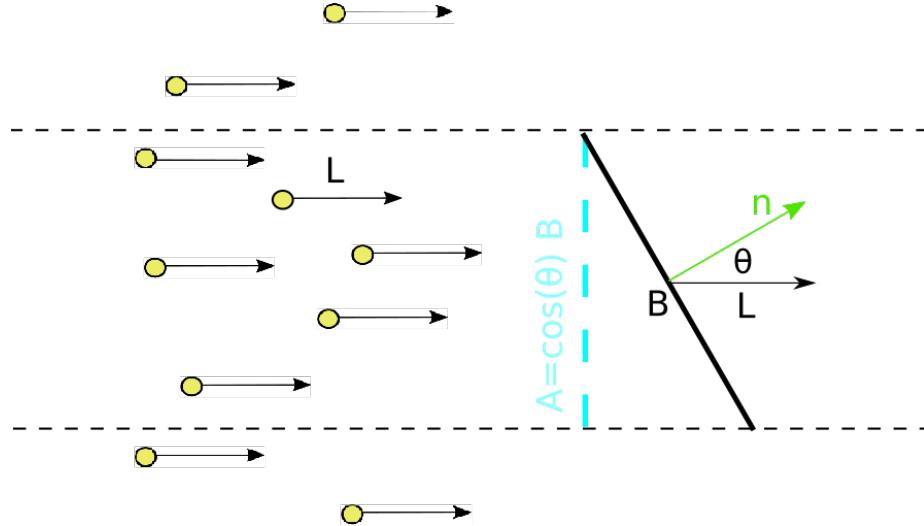
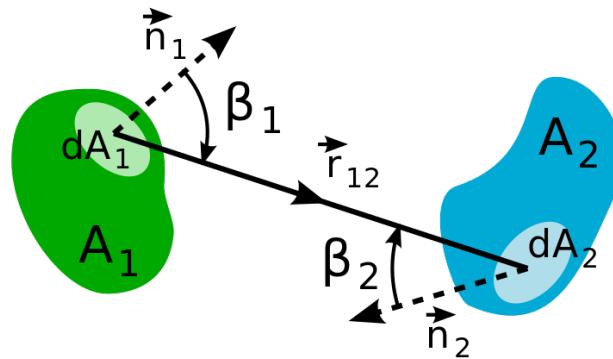


Abbildung 6: Partikelstrom

Lässt man  $B \rightarrow x$  gegen einen Punkt konvergieren, so erhalten wir die Strahlungsdichte am Punkt  $x$  in Richtung  $n$  als Grenzprozess  $L(B) \rightarrow L(x, n)$ .

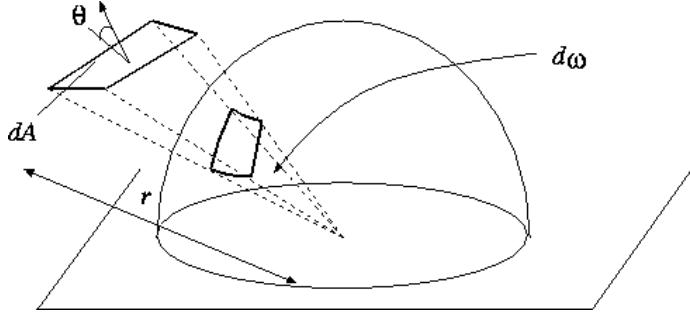
Angewendet auf Flächen erhalten wir das photometrische Grundgesetz. Es beschreibt die Strahlungleistung, die von einem Flächenelement auf ein anderes Flächenelement übertragen wird.



**Satz 17.** Die Strahlungsleistung  $\phi := \frac{\partial Q}{\partial t}$ , die von einer abstrahlenden Fläche  $A_2$  auf eine Fläche  $A_1$  übertragen wird, berechnet sich durch

$$\phi = \int_{A_1} \int_{\pi_s(A_2)} L(x, \omega) \cdot \cos(\beta_1) d\omega dA_1 , \quad (7)$$

wobei  $\theta$  der Winkel zwischen der Flächennormale am Punkt  $x$  und der Richtung  $\omega$  ist und  $\pi_s(A_2)$  das sphärische Bild von  $A_2$  ist.



Mit der Beziehung

$$d\omega = \frac{1}{r^2} \cdot \cos(\theta) dA \quad (8)$$

erhalten wir die Darstellung

$$\phi = \int_{A_1} \int_{A_2} \frac{1}{r_{12}^2} \cdot L(x, \bar{x}\bar{y}) \cdot \cos(\beta_1) \cdot \cos(\beta_2) dA_2 dA_1 , \quad (9)$$

**Definition 37.** Eine Oberfläche, dessen Strahldichte für alle Punkte und Richtungen konstant ist, also  $L(x, \omega) = c$  für alle  $x \in O$  und  $\omega \in S^2$ , bezeichnet man auch als Lambertschen Strahler.

**Definition 38.** Die Irradiance  $E$  ist definiert durch

$$E(x) := \int_{H^2} L_i(x, \omega) \cos \theta d\omega , \quad (10)$$

also die auftreffende Strahlungsleistung pro Flächeneinheit.

**Definition 39.** Die Radiosity  $B$  ist definiert durch

$$B(x) := \int_{H^2} L_o(x, \omega) \cos \theta d\omega , \quad (11)$$

also die ausgehende Strahlungsleistung pro Flächeneinheit.

**Definition 40.** Die bidirektionale Reflektanzverteilungsfunktion (engl. Bidirectional Reflectance Distribution Function, BRDF) ist eine Funktion  $f_r(x, \omega_i, \omega_r)$ , die das Reflexionsverhalten der Oberfläche eines Materials beschreibt. Sie hat als Eingabe die ausgehende Richtung  $\omega_r$  und die eingehende Richtung  $\omega_i$  am Punkt  $x$ . Sie liefert den Quotienten aus Strahlungsdichte und Bestrahlungsstärke für die ausgehende Richtung  $\omega_r$  und die eingehende Richtung  $\omega_i$  am Punkt  $x$ . Sie gibt somit die Abhängigkeit des reflektierten Lichts von der einfallenden Lichtstärke an:

$$\underbrace{L_r(x, \omega_r)}_{\substack{\text{Reflektierte Strahlung} \\ \text{in Richtungen } \omega_r}} = \underbrace{\int_{H^2} f_r(x, \omega, \omega_r)}_{\substack{\text{Reflektionseigenschaft} \\ \text{des Materials}}} \cdot \underbrace{L_i(x, \omega) \cdot \cos(\theta)}_{\substack{\text{Eingehende Strahlung} \\ \text{aus Richtung } \omega}} d\omega \quad (12)$$

Summation über alle  
eingehenden Richtungen  $\omega$

Letztere Gleichung wird Reflectance-Gleichung genannt.

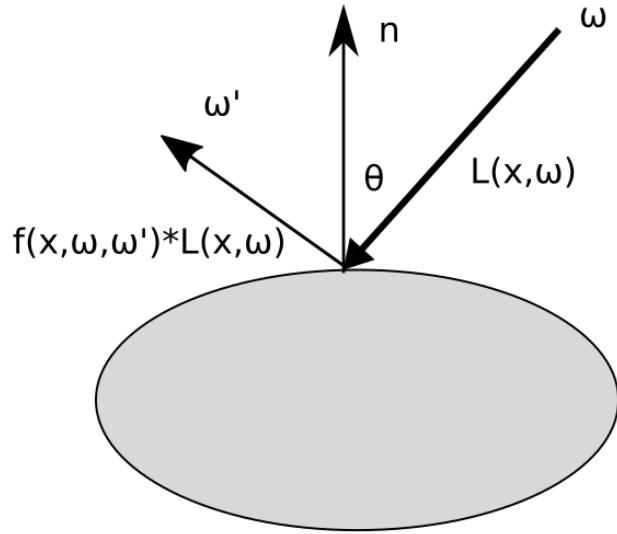


Abbildung 7: BRDF Funktion

## 2.2 Farbwahrnehmung und Farbmodelle

### 3 Kurven, Flächen und Netze

Die rechnergestützte Beschreibung von Kurven und Flächen ist ein wichtiges Gebiet der Computergrafik. Es ermöglicht die Berechnung geometrischer und physikalischer Eigenschaften von Körpern so wie deren graphische Darstellung. Eine zentrale Rolle spielen hierbei geometrische Objekte, die durch Punkte, Kanten und einfache Flächen, wie zum Beispiel Dreiecke oder Vierecke, beschrieben werden können. Man spricht dann auch entsprechend von Dreiecks- bzw. Vier-ecksnetzen. Häufig sind dabei nicht alle möglichen Konstruktionen zulässig und es hat sich ein Struktur herausgebildet, die für die Computergrafik besonders

geeignet ist. Zum einen, weil seine grafische Darstellung besonders schnell und einfach möglich ist und zum anderen, weil viele Berechnungen bestimmte Voraussetzungen benötigen, man denke da zum Beispiel an die Oberfläche oder das Volumen eines Körpers. Von Bedeutung sind in diesem Kontext auch geeignete Datenstrukturen, mit deren Hilfe sich diese Strukturen und gängige Berechnungen effizient verarbeiten lassen. Ein weiterer wichtiger Aspekt ist das generieren und modellieren solcher Strukturen. Hierbei treten häufig sogenannte Interpolationsprobleme auf. Sie entstehen aus dem Wunsch heraus, Kurven und Flächen nur durch die Angabe von Punkten zu generieren.

### 3.1 Polygone, Netze und Elemente der diskreten Geometrie

**Definition 41.** Ein geschlossenes Polygon ist ein Paar  $P := (E_P, K_P)$ , wobei

$$E_P := \{e_0, \dots, e_k\} \subset \mathbb{A}^3,$$

eine geordnete Menge von paarweise verschiedenen Punkten, die auch Ecken genannt werden, und

$$K_P := \left\{ (e_0, e_1), \dots, (e_{k-1}, e_k), (e_k, e_0) \right\} \subset \mathbb{A}^3 \times \mathbb{A}^3$$

die zugehörige Menge der gerichteten Kanten ist. Ist  $(e_{l-1}, e_l) \in K_P$  eine gerichtete Kante, so bezeichnen wir mit

$$|(e_{l-1}, e_l)| := \{e_{l-1} + t \cdot \overrightarrow{e_{l-1} e_l} \mid t \in [0, 1]\}$$

ihre geometrische Realisierung oder einfach Kante und mit

$$|P| := \bigcup_{(e_{l-1}, e_l) \in K_P} |(e_{l-1}, e_l)|$$

die geometrische Realisierung des Polygons oder geometrisches Polygon.

Ein (geschlossenes) Polygon heißt **einfach**, falls der Schnitt zweier geometrischer Kanten entweder leer oder genau ein Punkt aus  $E$  ist. Es heißt **planar**, falls alle Punkte  $e \in E_P$  in einer Ebene liegen.

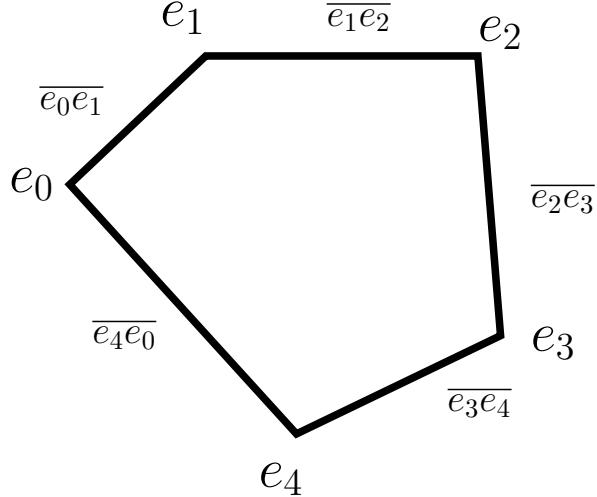


Abbildung 8: Geschlossenes Polygon

**Definition 42.** Wir nennen ein geschlossenes Polygon  $P'$  eine **Nummerierung** von  $P$ , falls  $E_{P'} = E_P$  (als Mengen) und  $|P'| = |P|$  ist.

Wie wir gleich sehen werden, definiert die Reihenfolge der Punkte eines einfachen, geschlossenen, planaren Polygons eine Orientierung. Um diesen Sachverhalt präzise definieren zu können, benötigen wir einen Satz, der auf den ersten Blick als selbstverständlich erscheint aber rein mathematisch betrachtet tatsächlich schwer zu beweisen ist. Es handelt sich um den Jordanschen Kurvensatz:

**Satz 18.** Sei  $P$  ein einfaches, geschlossenes, planares Polygon und  $U$  die Ebene, in der alle seine Punkte liegen. Dann unterteilt die geometrische Realisierung  $|P|$  die Ebene  $U$  in genau zwei Gebiete, ein beschränktes, das das Innere des Polygons genannt wird, und ein unbeschränktes, das das Äußere des Polygons genannt wird.

**Definition 43.** Wir bezeichnen das Innere eines Polygons  $P$  mit  $\overset{\circ}{|P|}$ .

Mit Hilfe des Jordanschen Kurvensatzes können wir nun sagen, was die Durchlaufrichtung einer Kante ist und schließlich ob ein Polygon mit oder gegen den Uhrzeigersinn orientiert ist.

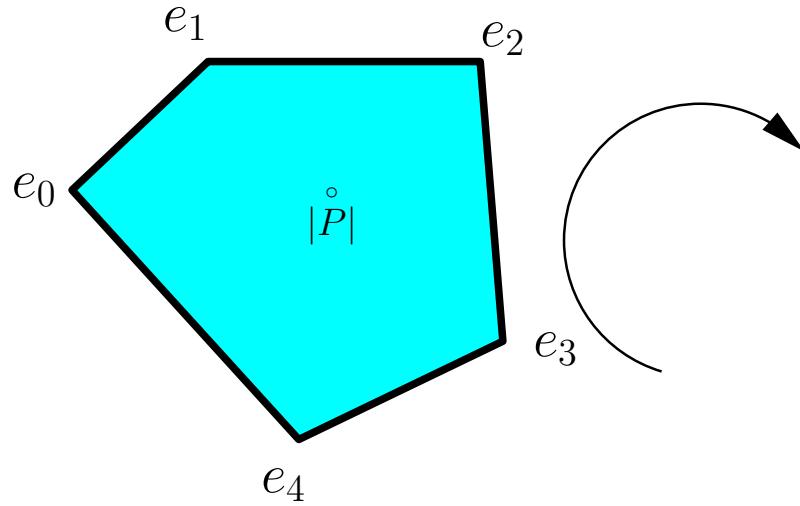
**Definition 44.** Sei  $P$  ein einfaches, geschlossenes, planares Polygon. Eine gerichtete Kante  $(e_{l-1}, e_l) \in E_P$  hat **positive Durchlaufrichtung**, falls beim Durchlaufen der Kante  $|(e_{l-1}, e_l)|$  von  $e_{l-1}$  nach  $e_l$  das innere des Polygons stets rechts von der Kante liegt und **negative Durchlaufrichtung**, falls es stets links von ihr liegt.

**Definition 45.** Ein einfaches, geschlossenes, planares Polygon  $P$  heißt im Uhrzeigersinn orientiert, falls eine und damit alle gerichteten Kanten positive

*Durchlaufrichtung haben und entsprechend gegen den Uhrzeigersinn orientiert bei negativer Durchlaufrichtung.*

$$P = \{e_0, e_1, e_2, e_3, e_4\}$$

$$K_P = \{(e_0, e_1), (e_1, e_2), (e_2, e_3), (e_3, e_4), (e_4, e_0)\}$$



$$P' = \{e_0, e_4, e_3, e_2, e_1\}$$

$$K_{P'} = \{(e_0, e_4), (e_4, e_3), (e_3, e_2), (e_2, e_1), (e_1, e_0)\}$$

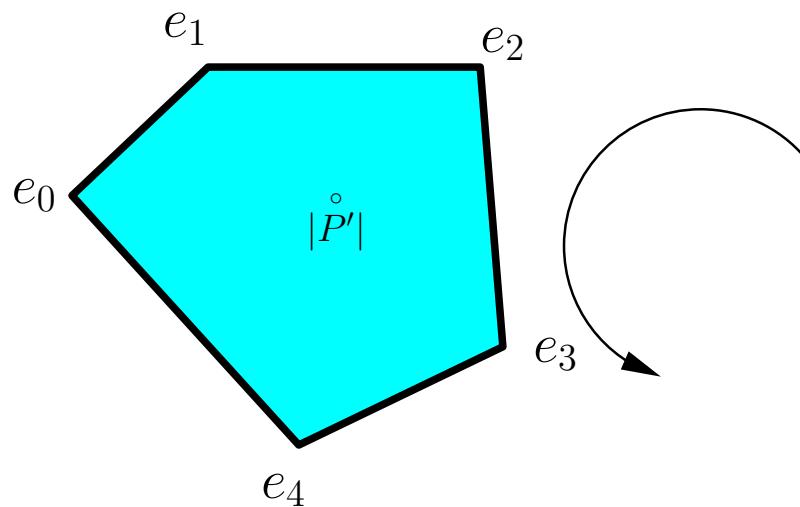


Abbildung 9: Gerichtete Polygone

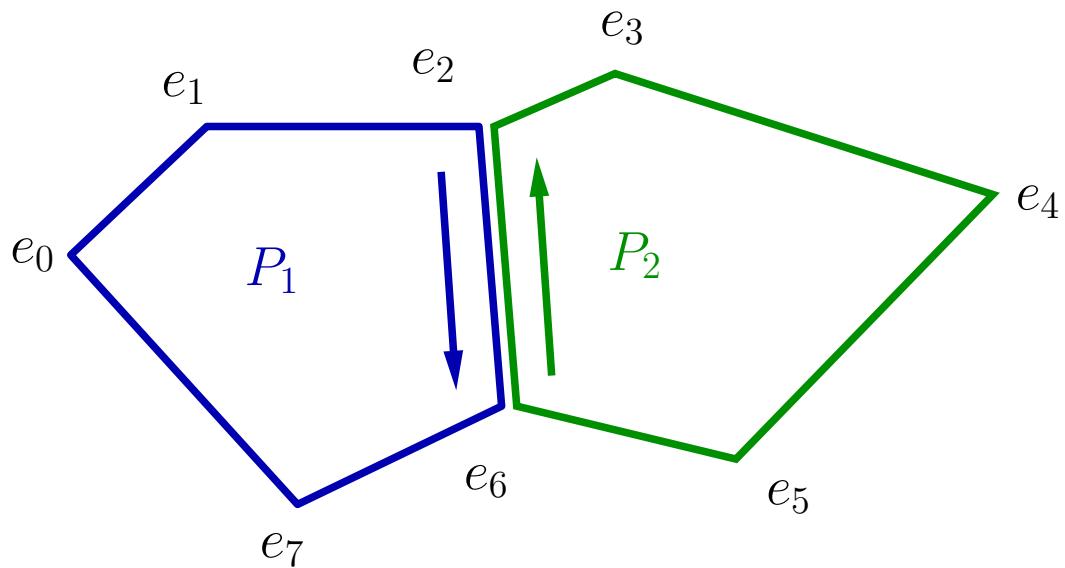
Netze bestehen nun im wesentlichen aus an den Kanten zusammengeklebten, einfachen, geschlossenen, planaren Polygonen.

**Definition 46.** Sei  $N := \bigcup P_i$  die Vereinigung endlich vieler, einfacher, geschlossener, planarer Polygone,  $E_N := \bigcup E_{P_i}$  und  $K_N := \bigcup K_{P_i}$  die Vereinigung der Ecken beziehungsweise der Kantenmengen. Analog zu der Definition eines Polygons definieren wir die geometrischen Realisierungen  $|K_N| := \bigcup |P_i|$  und  $|N| := \bigcup |\overset{\circ}{P}_i| \bigcup |K_N|$

- Dann heißt  $N$  **Netz**, falls Der Schnitt zweier Polygone entweder leer, ein Punkt in  $E_N$  oder eine Kante in  $K_N$  ist.
- Die Menge aller Kanten, die nur zu einem Polygon gehören, heißt Rand von  $N$  und wird mit  $\partial N$  bezeichnet.
- Ein Netz heißt **geschlossen**, falls es keinen Rand gibt, in Zeichen  $\partial N = \emptyset$ .
- Die Polygone des Netzes werden auch als **Facetten** bezeichnet.

Durch die Orientierung von Polygonen können wir nun den Begriff der Orientierung und der Orientierbarkeit eines Netzes einführen.

**Definition 47.** Ein Netz  $N$  heißt orientierbar, falls man alle seine Polygone so nummerieren kann, dass jede gemeinsame Kante zweier Polygone jeweils entgegengesetzt gerichtet ist.

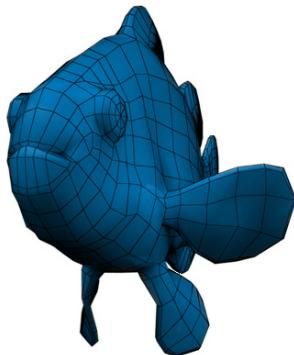


$$P_1 = \{e_0, e_1, e_2, e_6, e_7\}$$

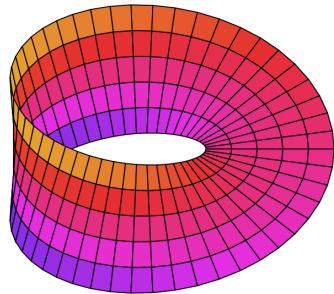
$$P_2 = \{e_2, e_3, e_4, e_5, e_6\}$$

Abbildung 10: Orientierbare Fläche

**Bemerkung 14.** Ist eine Fläche orientierbar und nummeriert man die Polygone so, dass benachbarte Kanten entsprechend der Definition immer entgegengesetzt gerichtet sind, so sind entweder alle seine Polygone im oder alle seine Polygone gegen den Uhrzeigersinn orientiert.



(a) Orientierbarer Fisch namens Wanda



(b) Nicht orientierbares Möbiusband

**Definition 48.** Eine Orientierung ist die Wahl einer Klasse von Nummerierungen, so dass alle Polygone entweder im oder gegen den Uhrzeigersinn orientiert sind.

**Bemerkung 15.** Ein Netz hat entweder genau zwei oder keine Orientierung.

**Definition 49.** Sei  $N$  ein orientierbares Netz mit einer Orientierung so gewählt, dass alle Polygone gegen den Uhrzeigersinn orientiert sind. Wählt man für jedes Polygon  $P_i \in N$  drei benachbarte Ecken  $e_{j-1}^{P_i}, e_j^{P_i}$  und  $e_{j+1}^{P_i}$ , so erhalten wir mit  $n_{P_i} = e_j^{P_i} e_{j+1}^{P_i} \times e_j^{P_i} e_{j-1}^{P_i}$  einen Vektor, der senkrecht auf der Ebene steht in der das Polygon liegt. Wir nennen die Menge  $\{P_0, \dots, P_n\}$  das äußere Normalenfeld des Netzes.

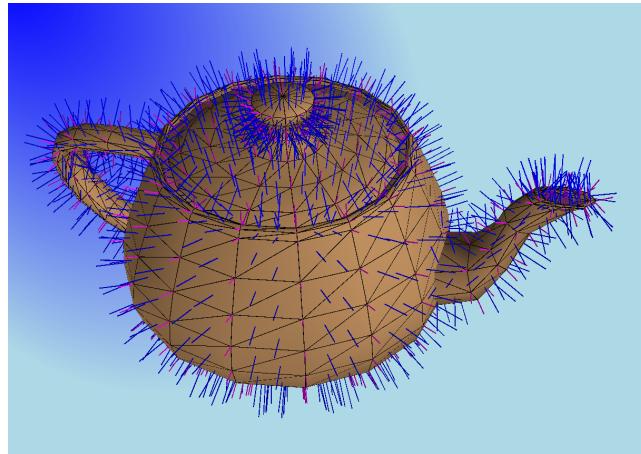


Abbildung 11: Das äußere Normalenfeld

Eine wichtige Größe für ein Netz ist die sogenannte Eulercharakteristik, welche in direkter Beziehung zum sogenannten Geschlecht eines Netzes steht.

**Definition 50.**  $N := \bigcup P_i$  ein Netz. Dann ist die Eulercharakteristik definiert als

$$\chi(N) := \#(E_N) - \#(|K_N|) + \#(N)$$

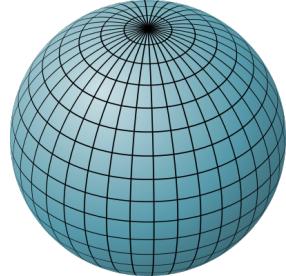
Sie ist also die Anzahl der Punkte, minus die Anzahl der geometrischen Kanten, plus die Anzahl der Facetten.

Das Geschlecht eines geschlossenen Netzes ist anschaulich gesprochen die Anzahl an Löchern, die durch das geometrische Netz hindurch gehen. Dies lässt sich mit einem Aufwand mathematisch präzise definieren, jedoch für unsere Zwecke reicht eine Definition durch Beispiele.

**Definition 51.** Sei  $N$  ein geschlossenes Netz. Dann bezeichnen wir mit

$$g(N) := \text{Anzahl der Löcher in } |N|$$

Beispiele:



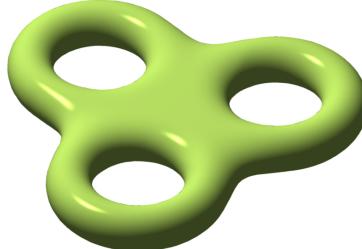
(a) Geschlecht 0



(b) Geschlecht 1



(c) Geschlecht 2



(d) Geschlecht 3

Abbildung 12: Geschlossene Netze der Geschlechter 0 bis 3. Quelle: Wikipedia

Der Zusammenhang zwischen der Eulercharakteristik und dem Geschlecht wurde im allgemeinen Fall von dem Mathematiker Henri Poincaré und vorher in einem Spezialfall von Leonhard Euler bewiesen.

**Satz 19.** Für ein geschlossenes Netz  $N$  gilt  $\chi(N) = 2 - 2g(N)$ .

**Bemerkung 16.** Nach dem Satz lässt sich das Geschlecht und somit die Anzahl der Löcher via  $g(N) = \frac{2-\chi(N)}{2}$  berechnen.

### 3.2 Datenstrukturen

**Definition 52.** Eine Liste ist eine Datenstruktur, in der man Objekte unter Beachtung der Reihenfolge speichern kann. Für eine Liste von Objekten  $O_1, \dots, O_n$  verwenden wir die Notation

$$E = (O_1, \dots, O_n)$$

Jedes Element kennt seinen Vorgänger und Nachfolger und man kann direkt auf das  $i$ -te Element zugreifen:

$$\begin{aligned} E[i] &:= O_i \\ \text{prev}(E[i]) &:= O_{i-1} \\ \text{next}(E[i]) &:= O_{i+1} \end{aligned}$$

Dabei bezeichnet  $\text{prev}(E[i])$  den Vorgänger und  $\text{next}(E[i])$  den Nachfolger, wobei

$$\forall l \in \mathbb{N} : \{E[l] = \text{NULL} \mid (l > n) \vee (l < 0)\}$$

$i$  wird auch als Index des  $i$ -ten Listenelements bezeichnet.

**Definition 53** (Eckenliste). In einer Liste  $E = (e_1, e_2, \dots, e_n)$  werden Referenzen auf die Ecken in Form von Indizes der Eckenliste gespeichert. Eine Referenz im  $\mathbb{R}^3$  könnte so aussehen:

$$e_k = \begin{pmatrix} x_k \\ y_k \\ z_k \end{pmatrix}$$

Die  $i$ -te Facette wird als Liste  $F_i = (i_1, \dots, i_l)$  von Referenzen auf die Eckenliste gespeichert. Die  $k$ -te Ecke der  $i$ -ten Facette kann so z.B. mit  $E[F_i[k]]$  referenziert werden.

Vor- und Nachteile:

- |   |  |
|---|--|
| + Einfach zu implementieren                             | nicht enthalten / schwierig zu berechnen   |
| + Orientierung kann durch Konvention gespeichert werden | - Die Zugehörigkeit einer Ecke zu einer Kante muss immer berechnet werden                        |
| - Nachbarschaftsbeziehungen                             | - Point-in-Mesh, Abstandsberechnungen, Schnittprobleme und ähnliches fast unmöglich zu berechnen |

**Definition 54** (Kantenliste). In einer Liste  $E = (e_1, e_2, \dots, e_n)$  werden Referenzen auf die Ecken gespeichert. In einer Liste  $K$  werden die Kanten in einer zwei Elementen Liste von Indizes auf die Eckenliste abgespeichert:

$$K = ((i_{e_1}, i_{e_n}), \dots, (i_{e_1}, i_{e_m}))$$

Die  $k$ -te Facette wird als Liste von Indizes auf die Kantenliste gespeichert:

$$F_k = (j_1, \dots, j_l)$$

In einer Liste  $M$  werden die Indizes auf Facetten in einer zwei Elementen Liste abgespeichert, die die entsprechende Kante in der Kantenliste als Kante haben:

$$M = ((i_1, i_2), \dots, (l_1, l_2))$$

$M[i] \in M$  sind also zwei Facetten, die  $K[i] \in K$  als Kante haben. Der erste Eintrag der Facette befindet sich links und der zweite rechts von der Kante. Ist die Kante eine Randkante, so wird der andere Wert auf  $-1$  gesetzt.

Vor- und Nachteile:

- + Nachbarschaftsbeziehungen der Polygone werden gespeichert
- Orientierung der Kanten geht verloren (schwierig zu speichern)

**Definition 55** (Halfedge). In einer Liste  $E = (e_1, e_2, \dots, e_n)$  werden Referenzen auf die Ecken gespeichert. Es wird eine Datenstruktur namens Halfedge eingeführt. Eine Halfedge  $e$  besitzt<sup>1</sup>:

- Indizes auf den Anfangs- und Endpunkt ( $i_b$  und  $i_e$ )
- Referenzen auf die gegenüberliegende Halfedge  $\text{twin}(e)$   
(Bei dieser sind die Anfangs- und Endpunkte vertauscht.)
- Referenzen auf die vorangehende  $\text{prev}(e)$  und die folgende  $\text{next}(e)$  Halfedge
- Referenz auf die Facette, die sie berandet

Die unendliche oder äußere Zelle wird wieder mit  $-1$  bezeichnet. Eine Facette ist dann eine Liste mit Referenzen von Halfedges, beziehungsweise reicht auch die Referenz auf eine Halfedge.

Vor- und Nachteile:

- + Alle kombinatorischen Daten werden effizient gespeichert
- + Auch die Orientierung der Kanten bleibt erhalten
- Die Implementierung ist komplex
- Overhead durch große Datens Mengen
- Nur für orientierbare Netze nutzbar

---

<sup>1</sup>Siehe auch Abbildung 13 auf der nächsten Seite

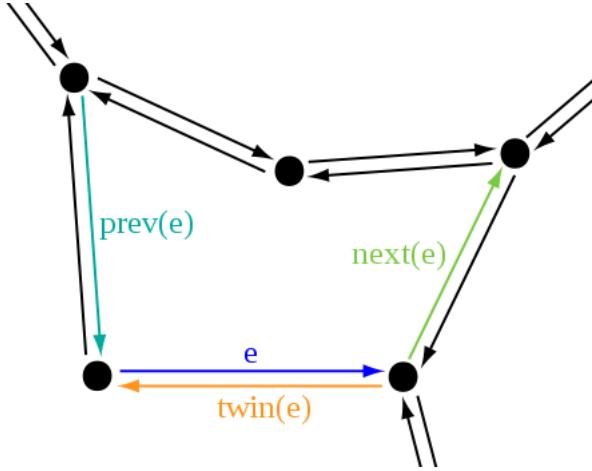


Abbildung 13: Halfedge. Quelle:Wikipedia

### 3.2.1 Subdivision

## 3.3 Modellierung

### 3.3.1 Bezier Kurven und Flächen

**Definition 56.** Die Bernsteinpolynome vom Grad  $n$  sind definiert als

$$B_i^n(t) := \binom{n}{i} (1-t)^{n-i} t^i$$

mit  $i = 0, \dots, n$ ,  $t \in [0, 1]$  und dem Binomialkoeffizient

$$\binom{n}{i} := \frac{n!}{i!(n-i)!} = \frac{n(n-1)\cdots 1}{i(i-1)\cdots 1(n-i)(n-i-1)\cdots 1}.$$

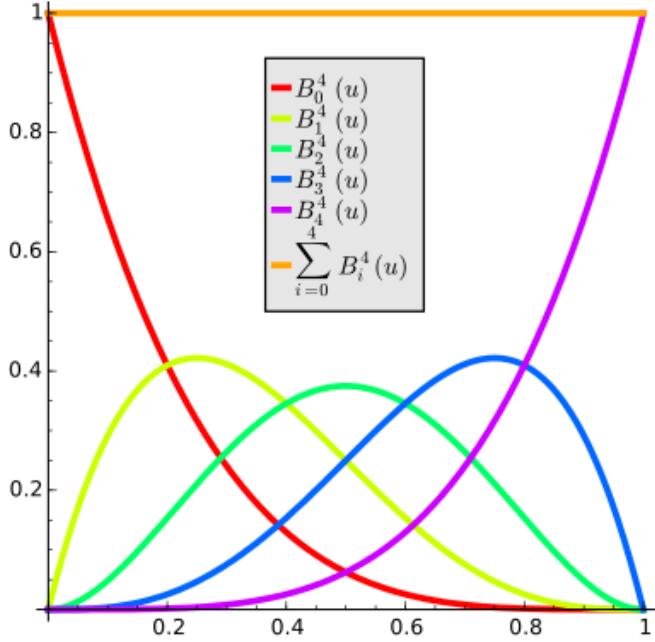


Abbildung 14: Die Bernsteinpolynome  $B_i^4$  und deren Summe. Quelle:Wikipedia

**Bemerkung 17.** Die Bernsteinpolynome vom Grad  $n$  bilden eine Basis des Vektorraums der Polynome vom Grad  $n$  im Intervall  $[0, 1]$ .

**Satz 20.** Es gilt die Rekursionsformel

$$B_i^n(t) = (1-t) \cdot B_i^{n-1}(t) + t \cdot B_{i-1}^{n-1}(t)$$

mit  $B_0^0(t) = 1$  und  $B_n^i(t) = 0$  für  $i < 0$  oder  $i > n$ .

*Beweis.* Folgt fast direkt aus der Rekursionsformel des Binomialkoeffizienten

$$\binom{n}{i} = \binom{n-1}{i} + \binom{n-1}{i-1}$$

$$\begin{array}{ccccccc} & & & 1 & & & \\ & & & 1 & 1 & 1 & \\ & & & 1 & 2 & 1 & \\ & & & 1 & 3 & 3 & 1 \\ & & & 1 & 4 & 6 & 4 & 1 \\ & & & 1 & 5 & 10 & 10 & 5 & 1 \end{array}$$

Abbildung 15: Pascalsches Dreieck

□

**Definition 57.** Seien  $b_0, \dots, b_n \in \mathbb{R}^3$ . Dann heißt die Kurve

$$B^n(t) := \sum_{i=0}^n B_i^n(t) \cdot b_i, t \in [0, 1]$$

eine Bezierkurve vom Grad  $n$ . Die  $b_i$  werden auch Kontrollpunkte genannt. Für ein beliebiges Intervall  $[a, b]$  definieren wir

$$B_{[a,b]}^n(t) := B^n\left(\frac{t-a}{b-a}\right), t \in [a, b].$$

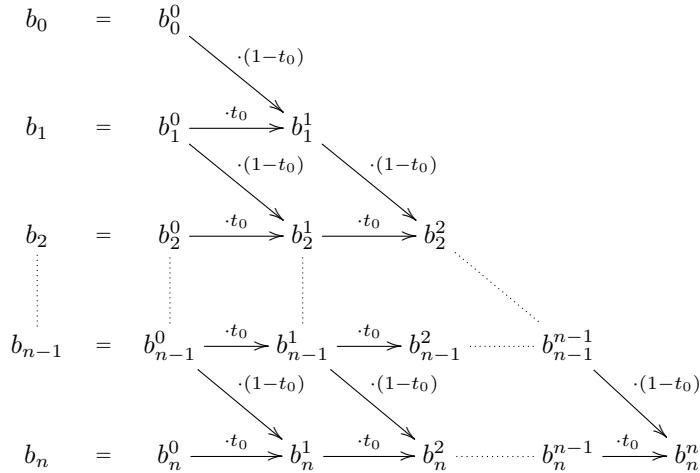
**Satz 21.** Eine Bezierkurve hat die Ableitung

- $(B^n)'(t) = n \cdot \sum_{j=0}^{n-1} B_j^{n-1}(t) \cdot (b_{j+1} - b_j)$ , und nach der Kettenregel
- $(B_{[a,b]}^n)'(t) = \frac{1}{b-a}(B^n)'(\frac{t-a}{b-a})$  für ein beliebiges Parameterintervall.

**Satz 22** (Algorithmus von de Casteljau). Sei  $B^n(t) := \sum_{i=0}^n B_i^n(t) \cdot b_i$  eine Bezierkurve. Für ein  $t_0 \in [0, 1]$  definieren wir rekursiv

$$b_i^k := \begin{cases} b_i & i = 0, \dots, n \\ (1-t_0) \cdot b_{i-1}^{k-1} + t_0 \cdot b_i^{k-1} & i = 1, \dots, n \quad k = 1, \dots, i \end{cases}$$

was sich schematisch folgendermaßen darstellen lässt:



Dann gilt  $b_n^n = B^n(t_0)$ .

$$t_0 = \frac{1}{2}$$

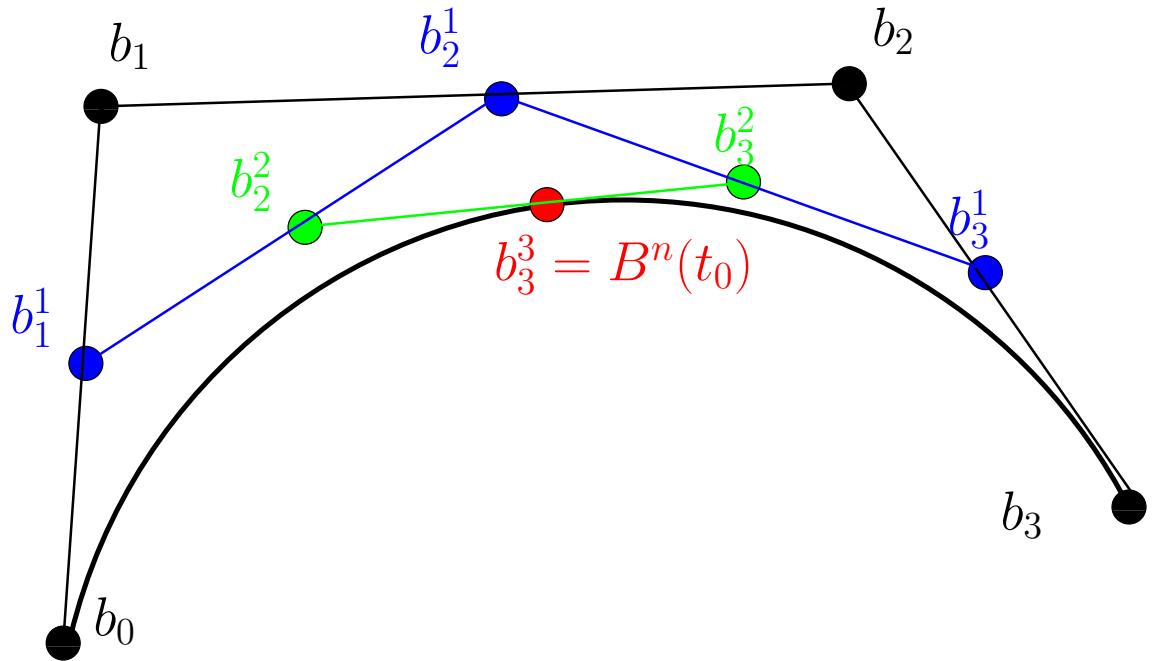


Abbildung 16: Auswertung einer Beziekurve zum Zeitpunkt  $t_0$  durch iterative Streckenteilung nach de Casteljau

**Definition 58** (Patching). Seien  $B^n(t)$  und  $(B^*)^m(t)$  Bezierkurven. Man spricht von einem  $C^0$ -Patching (an der Stelle  $B^n(1)$ ), falls  $B^n(1) = (B^*)^m(0)$  gilt. Stimmen auch die Ableitungen überein, also  $(B^n)'(1) = ((B^*)^m)'(0)$ , dann spricht man von einem  $C^1$ -Patching und stimmen noch für  $k > 1$  auch die höheren Ableitungen  $(B^n)^k(1) = ((B^*)^m)^k(0)$  überein, so spricht man von einem  $C^k$ -Patching.

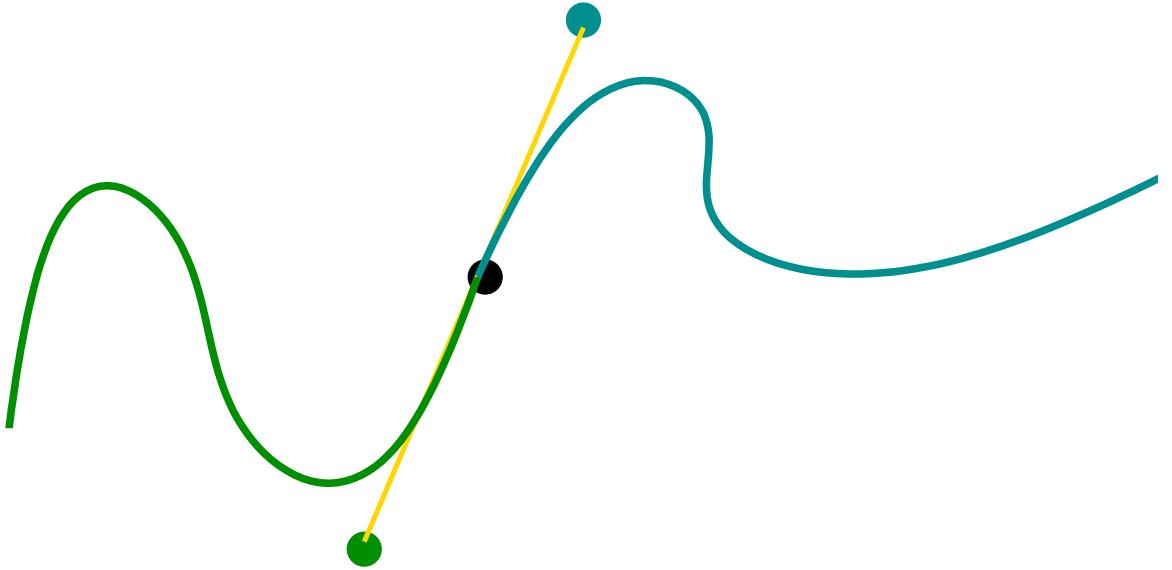


Abbildung 17: Bezier Patch

**Bemerkung 18.** Zwei Bezierkurven  $B^n(t) := \sum_{i=0}^n B_i^n(t) \cdot b_i$  und  $(B^*)^m(t) := \sum_{i=0}^m B_i^m(t) \cdot b_i^*$  bilden genau dann einen  $C^1$ -Patch (an der Stelle  $B^n(1)$ ), wenn  $B^n(1) = (B^*)^m(0)$  und  $(B^n(1) - b_{n-1}) = -\frac{m}{n}((B^*)^m(0) - (b^*)_1)$  gilt.

**Definition 59.** Ist  $B^n(t)$  eine Bezierkurve, so bilden die Bezierkurven  $(B^*)^n(t) := \sum_{i=0}^n B_i^n(t/2) \cdot b_i^i$  und  $(B^{**})^n(t) := \sum_{i=0}^n B_i^n(\frac{t+1}{2}) \cdot b_n^{n-i}$  einen  $C^1$ -Patch.

**Definition 60.** Ersetzt man in einer Bezierkurve

$$B^m(v) := \sum_{j=0}^m B_j^m(v) \cdot b_j$$

vom Grad  $m$  die Kontrollpunkte  $b_j$  durch  $m + 1$  Bezierkurven

$$b_j(u) = \sum_{i=0}^n B_i^n(u) \cdot b_{ij}$$

vom Grad  $n$  mit jeweils  $n + 1$  Kontrollpunkten  $b_{ij}$  für  $i = 0, \dots, n$ , so erhält man eine Fläche

$$F(u, v) = \sum_{i=0}^n \sum_{j=0}^m B_i^n(u) B_j^m(v) \cdot b_{ij},$$

welche auch Tenorprodukt-Fläche der Bezierkurven oder einfach Bezierfläche genannt wird.



Abbildung 18: Ein Rendering der Utah Teekanne, eines der weit verbreitetsten 3D-Modelle in der Computergrafik. Sie wurde mit Bezierflächen modelliert.

## 4 Echtzeitvisualisierung und OpenGL

### 4.1 Geschichte

OpenGL entstand ursprünglich aus dem von Silicon Graphics (SGI) entwickelten IRIS GL. Im sogenannten Fahrenheit-Projekt versuchten Microsoft und SGI ihre 3D-Standards zu vereinheitlichen, das Projekt wurde jedoch wegen finanzieller Schwierigkeiten auf Seiten von SGI abgebrochen.

Der OpenGL-Standard wird vom OpenGL ARB (Architecture Review Board) festgelegt. Das ARB existiert seit 1992 und besteht aus einer Reihe von Firmen. Stimmberchtigte Mitglieder sind die Firmen 3DLabs, Apple, AMD/ATI, Dell, IBM, Intel, Nvidia, SGI und Sun (Stand Nov. 2004). Weiter mitwirkende Firmen sind Evans and Sutherland, Imagination Technologies, Matrox, Quantum3D, S3 Graphics, Spinor GmbH, Tungsten Graphics, und Xi Graphics. Microsoft, eines der Gründungsmitglieder, hat das ARB im März 2003 verlassen.

Neue Funktionen in OpenGL werden meist zuerst als herstellerspezifische Erweiterungen eingeführt und gehen dann den Weg über herstellerübergreifende Erweiterungen und ARB-Erweiterungen zu Kernfunktionalität. Dies erlaubt es, neueste Möglichkeiten der Grafikhardware zu nutzen und dennoch OpenGL abstrakt genug zu halten.

Quelle: <https://de.wikipedia.org/wiki/OpenGL>

### 4.2 GL-Pipeline

Eine Computergrafik-Pipeline besteht im Wesentlichen aus den folgenden Schritten:

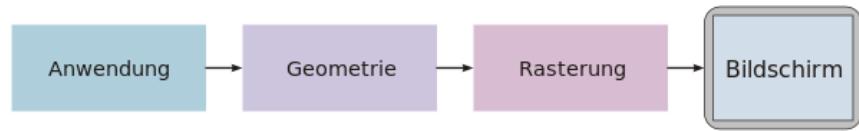


Abbildung 19: Computergrafik-Pipeline

Eine solche Pipeline wird auch von OpenGL implementiert. Es besteht jedoch die Besonderheit, dass die Geometrie und die Rasterung in Hardware realisiert ist und man durch kleine Programme, sogenannte Shader, auf diese Hardware zugreifen und Manipulationen vornehmen kann beziehungsweise seit OpenGL 2.0 sogar muss.

#### 4.2.1 Opengl Shaderpipeline

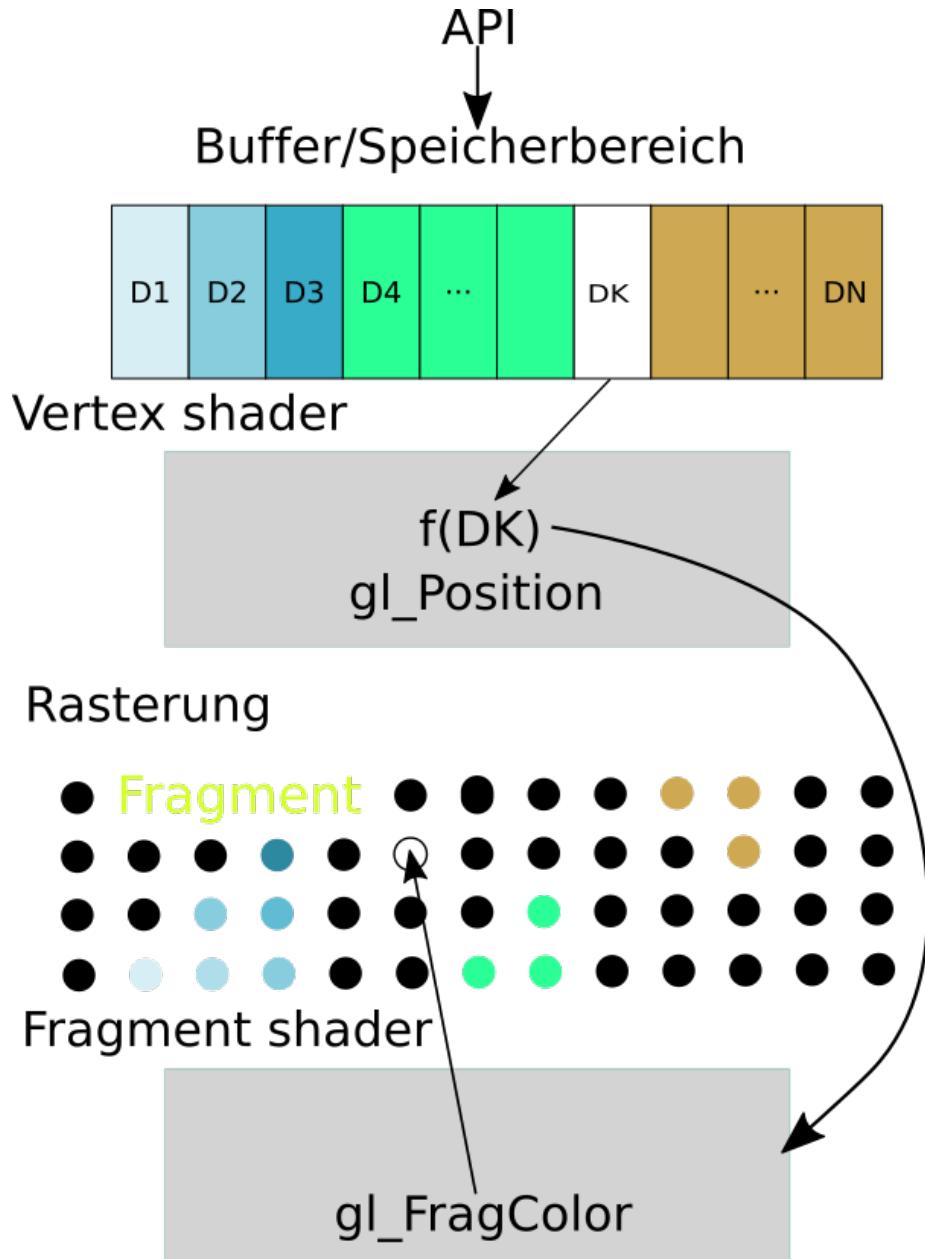


Abbildung 20: Zentralprojektion

#### 4.2.2 Geometrie

Die Geometrieverarbeitung besteht aus der Hintereinanderausführung der folgenden affinen/homogenen Abbildungen und Algorithmen:

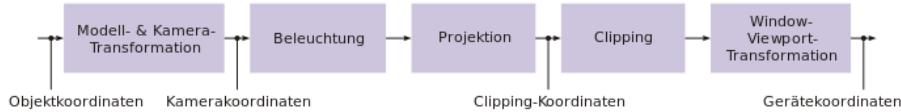


Abbildung 21: Geometrie-Pipeline

#### Modell und Kameratransformationen

Die Transformationen von Objektkoordinaten in das Kamerakoordinatensystem werden durch affine Transformationen beschrieben, welche durch Multiplikation durch homogene  $4 \times 4$ -Matrizen realisiert werden.

#### Projektion

Die Projektion wird durch Matrizen ähnlich der Projektionsmatrix  $K_{persp_{xy}}$  und  $K_{orth_{xy}}$  aus Abschnitt 1.2.1 realisiert. Es werden jedoch noch Translationen, Rotationen und Stauchungen dazwischen geschaltet, die ebenfalls als  $4 \times 4$ -Matrizen realisiert werden können und mit denen diese Matrix multipliziert werden. Man erreicht damit, dass der Kegelstumpf zwischen der vorderen (*nearplane*) und der hinteren (*farplane*) Projektionsebene in den  $[-1, 1] \times [-1, 1] \times [-1, 1]$  Würfel abgebildet wird, welcher auch Sichtvolumen genannt wird.

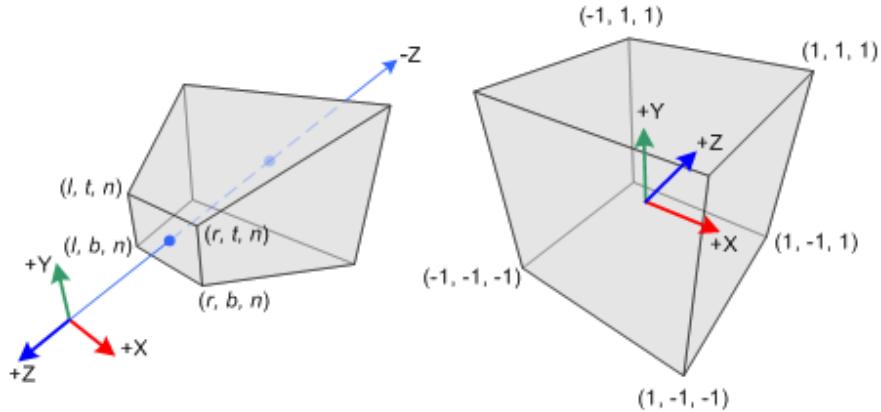


Abbildung 22: Projektion und Sichtvolumen

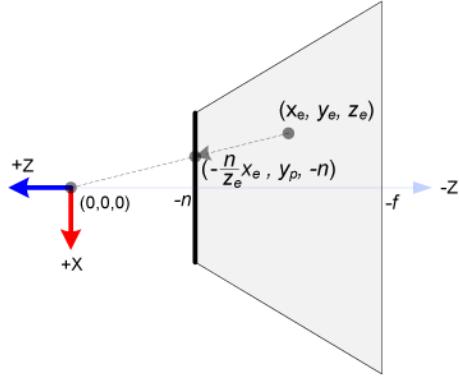


Abbildung 23: Zentralprojektion

Die Multiplikation all dieser Matrizen wird auch die MODEL-VIEW-PROJECTION-MATRIX genannt.

In obigen Fall ist die Projektionsmatrix gegeben durch

$$P := \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{t+b} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t-b}{t-b} & 0 \\ 0 & 0 & \frac{-f-n}{f-n} & \frac{-2 \cdot f \cdot n}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}.$$

und insbesondere falls  $r = -l$  und  $t = -b$  ist

$$P := \begin{pmatrix} \frac{n}{r} & 0 & 0 & 0 \\ 0 & \frac{n}{t} & 0 & 0 \\ 0 & 0 & \frac{-f-n}{f-n} & \frac{-2 \cdot f \cdot n}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}.$$

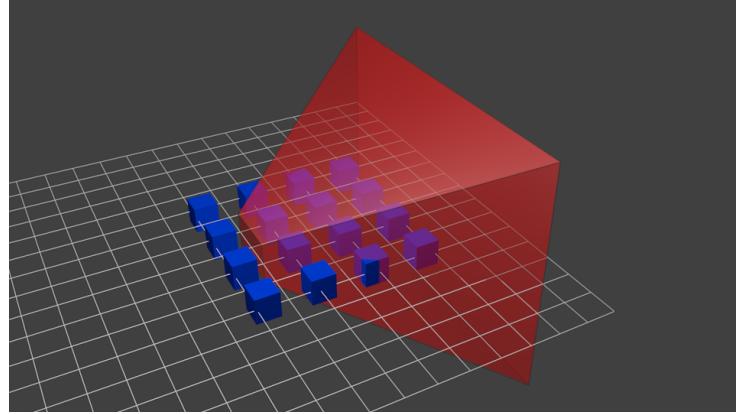


Abbildung 24: Zentralprojektion

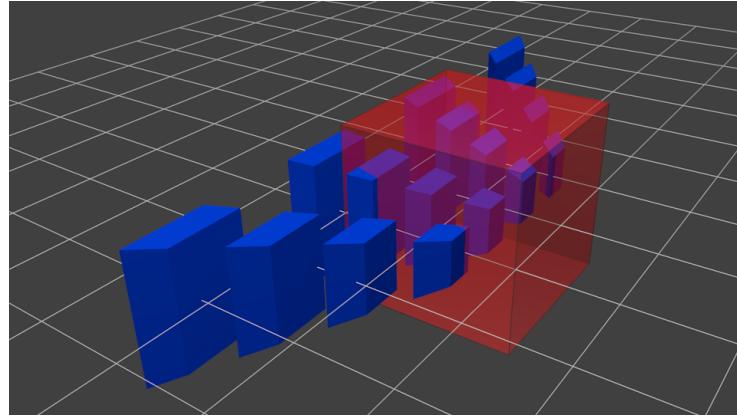


Abbildung 25: Zentralprojektion

#### Windows-Viewport-Transformation

Zuletzt müssen die zweidimensionalen Punkte noch mit Hilfe einer zweidimensionalen affinen Transformation in das Koordinatensystems des Anzeigenfensters auf dem Ausgabegerät transformiert werden. Diese Transformation wird auch Windows-Viewport-Transformation genannt.

#### 4.2.3 Rasterung

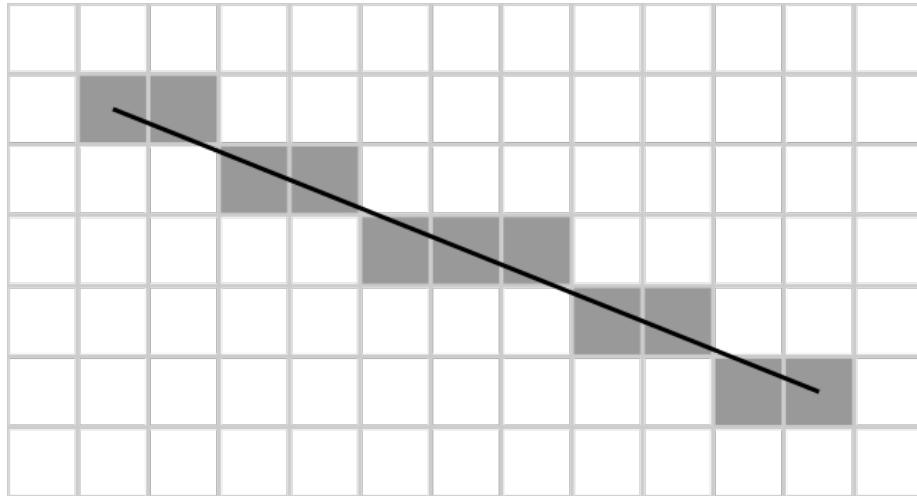


Abbildung 26: Rasterung nach Bresenham Algorithmus

Als Rasterung bezeichnet man das Transformieren kontinuierlicher, zweidimensionaler Daten auf diskrete Pixel. Anhand der gegebenen Daten muss also ent-

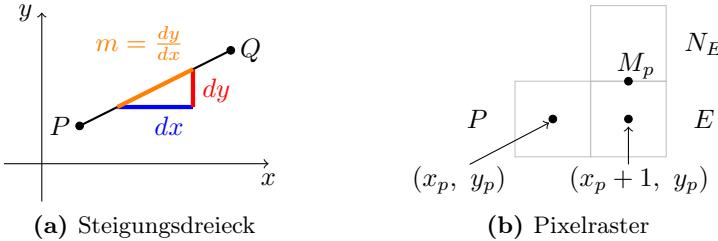


Abbildung 27: Schrittbeobachtung Bresenham Algorithmus

schieden werden, welche Farbe ein Pixel des Ausgabegerätes erhält. Wir haben also eine Funktion  $\text{Frame} - \text{Buffer} : \mathbb{N} \times \mathbb{N} \rightarrow \text{Farbe}$ . Die Pixel, die einem Dreieck zugeordnet werden, bezeichnet man auch als Fragment.

Im Bereich Echtzeitvisualisierung sind alle gängigen Rasterverfahren für Dreiecke und andere primitiven im wesentlichen Abwandlungen und Weiterentwicklungen von Algorithmen, die von Bresenham eingeführt wurden. Wir wollen uns den Bresenham Algorithmus für Linien/Strecken dazu exemplarisch anschauen.

**Algorithmus 2** (Bresenham für Strecken mit positiver Steigung). *Die Strecke  $S = \overline{PQ} \in \mathbb{R}^n$  soll auf ein ( $n$ -dimensionales Pixel-) Raster abgebildet werden (Siehe Abbildung 26). Wir beschränken uns hier auf die Berechnung im  $\mathbb{R}^2$ .  $P$  und  $Q$  seien gegeben mit*

$$P = \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} \quad Q = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} \quad \forall x_i, y_i \in \mathbb{R}$$

Die Geradengleichung lässt sich einfach bestimmen (Abbildung 27a):

$$\begin{aligned} y &= mx + b \\ m &\text{ mit} \\ m &= \frac{dy}{dx} \\ dx &= x_1 - x_0 \\ dy &= y_1 - y_0 \end{aligned}$$

Die Abbildung auf ein Raster ist für Steigungen  $m \in \{0, 1\}$  trivial. Anders für  $0 < m < 1$ . Dafür betrachten wir in einem Schritt benachbarte Pixel (Abbildung 27b):

- Ausgangspixel  $P(x_p, y_p)$
- Nachbar  $E(x_p + 1, y_p)$
- „Nachfolgender“ Nachbar  $N_E(x_p + 1, y_p + 1)$

Der Mittelpunkt zwischen  $N$  und  $E$  wird festgelegt durch

$$M_p = \left( x_p + 1, y_p + \frac{1}{2} \right)$$

Damit kann entschieden werden welches benachbarte Pixel zur Strecke gehört.  
Liegt nun

$$M_p \begin{cases} \text{oberhalb } \overline{PQ} & \Rightarrow \text{ wähle } E \\ \text{unterhalb } \overline{PQ} & \Rightarrow \text{ wähle } NE \end{cases}$$

Bresenham hat eine effiziente Lösung dieses Problems aufgezeigt. Zunächst lässt sich die Geradengleichung in eine Funktion  $F$  umformen, die von zwei Variablen abhängt:

$$\begin{aligned} y &= \frac{dy}{dx}x + b \\ dx \cdot y &= dy \cdot x + dx \cdot b \\ 0 &= dy \cdot x - dx \cdot y + dx \cdot b \\ F(x, y) &:= dy \cdot x - dx \cdot y + dx \cdot b \end{aligned}$$

Das arithmetische Entscheidungskriterium lautet also

$$\begin{aligned} (x, y) \in y = mx + b &\Leftrightarrow F(x, y) = 0 \\ \text{bzw.} \\ F(x, y) &\begin{cases} = 0 & (x, y) \text{ liegt auf } S \\ > 0 & (x, y) \text{ liegt unterhalb von } S \\ < 0 & (x, y) \text{ liegt oberhalb von } S \end{cases} \end{aligned}$$

Damit kann die Entscheidungsfunktion  $D_p$  definiert werden

$$\begin{aligned} D_p &:= 2 \cdot F(M_p) \\ &= 2 \cdot F(x_p + 1, y_p + \frac{1}{2}) \\ &= dy(2x_p + 2) - dx(2y_p + 1) + dx \cdot 2b \end{aligned}$$

1. Fall:  $D_p < 0 \Rightarrow$  Nachfolgepixel ist  $E$

$$\begin{aligned} D_E &= 2 \cdot F(M_E) \\ &= 2 \cdot F(x_p + 2, y_p + \frac{1}{2}) \\ &= dy \cdot (2x_p + 4) - dx \cdot (2y_p + 1) + 2 \cdot b \cdot dx \\ &= D_p + \Delta_E \\ \text{mit} \quad \Delta_E &:= 2 \cdot dy \end{aligned}$$

2. Fall:  $D_p \geq 0 \Rightarrow$  Nachfolgepixel ist  $NE$

$$\begin{aligned} D_{NE} &= 2 \cdot F(M_{NE}) \\ &= D_p + \Delta_{NE} \\ \text{mit} \quad \Delta_{NE} &:= 2dy - 2dx \end{aligned}$$

Eine Implementierung könnte wie folgt aussehen:

```

1 bresenham(x0, y0, x1, y1){
2     dx = x1 - x0;
3     dy = y1 - y0;
4     d = 2*dy - dx; /*F(x0 + 1, y0 + 1/2) | F(x0, y0)=0*/
5     deltaE = 2 * dy;
6     deltaNE = 2 * (dy - dx);

```

```

7      x = x0;
8      y = y0;
9      writепixel(x0, y0);
10     while (x < x1) {
11         if (d < 0) {
12             d += deltaE;
13             x++;
14         } else {
15             d += deltaNE;
16             x++; y++;
17         }
18         writепixel(x, y);
19     }
20 }
```

Das Rastern erzeugt oft harte, kantige und ausgefranste Übergänge. Diese Effekte bezeichnet man auch als Aliasing. Algorithmen, die diesen Aliasing-Effekten entgegenwirken nennt man auch Antialiasing.

*tems la gresle et le tonner  
tems la gresle et le tonner*



Abbildung 28: Eine Schrift mit und ohne Antialiasing

## Interpolation

GLSL interpoliert Werte innerhalb eines Fragmentes linear zwischen den im Vertex-Shader gesetzten Werten. Dies kann zum Beispiel mit Hilfe eines Sweepeline-Algorithmus realisiert werden.

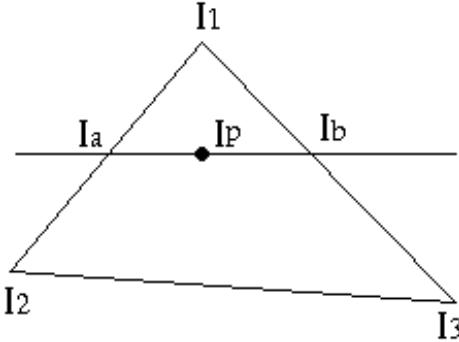


Abbildung 29: Scanline-Verfahren beim Gouraud-Shading

## Sichtbarkeitsprobleme

### Clipping

Beim 3D-Clipping werden alle Polygone verworfen, die nach der Transformation in Clipping-Koordinaten vollständig ausserhalb des Sichtvolumens  $[-1, 1] \times [-1, 1] \times [-1, 1]$  liegen. Das 2D-Clipping findet nach der Projektion der Polygone statt. Polygone, die über das Viewportfenster hinaus ragen, werden an den Fensterkanten abgeschnitten. Der wichtigste Schritt ist hierbei das Abschneiden von Strecken an diesen Kanten. Wir betrachten dazu exemplarisch den Cohen-Sutherland-Algorithmus:

**Algorithmus 3** (Cohen-Sutherland).

### Culling/Rückseitenentfernung

Als Culling bezeichnet man das Entfernen von Polygonen anhand Ihrer Orientierung, beziehungsweise Anhand der zugehörigen Normale. Man definiert, welche Orientierung eine Rückseite darstellt und verwirft dann Polygone, deren Normale positives beziehungsweise negatives Skalarprodukt mit der Blickrichtung aufweisen. So werden Beispielsweise Rückseiten, die nicht sichtbar sind, nicht weiter in der Pipeline verarbeitet.

### z-Buffer

Der Z-Buffer enthält für jedes Pixel  $(u, v)$  nach der Rasterung einen Wert zwischen  $-1$  und  $1$ . Dieser Wert ist der Abstand zum nächsten Punkt eines Polygons in Clipping-Koordinaten von diesem Pixel auf der Projektionsebene. Wir haben somit eine Funktion

$$Z - Buffer : \mathbb{N} \times \mathbb{N} \rightarrow [-1, 1].$$

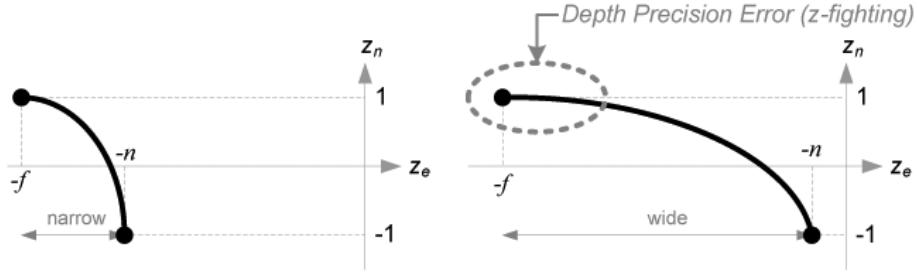


Abbildung 30: Z-Buffer-War

**Algorithmus 4** (z-Buffer Algorithmus). Setze Alle Einträge im Z-Buffer auf 1 (Hintergrund). Setze alle Einträge im Framebuffer auf die Hintergrundfarbe.

$\forall$  Polygone  $P$ :

begin:

Transformiere das Polygon in Clipping-Koordinaten.

Transformiere das Polygon in Rasterkoordinaten.

$\forall$  erhaltenen Pixel  $(u, v)$ :

begin:

$pz :=$  Z-Koordinate des Polygons in Clipping-Koordinaten zum entsprechenden Pixel

$zz :=$  Eintrag im Z-Buffer für das Pixel  $(u, v)$

if  $pz < zz$ :

begin:

$Z - Buffer(u, v) := pz$

$Frame - Buffer(u, v) := Farbe(P, (u, v))$

end

end

end

### 4.3 Lokale Beleuchtungsmodelle

#### 4.3.1 Ideale und diffuse Reflexionen

#### 4.3.2 Lambert Modell

Das Lambertsche Modell beschreibt, wie durch den perspektivischen Effekt die Strahlungsstärke mit flacher werdendem Abstrahlwinkel abnimmt. Sei nun  $L$  der Punkt, an dem sich eine punktförmige Lichtquelle befindet und  $P \in N$  ein Punkt eines Netzes  $N$  mit Normale  $n$  (also insbesondere gilt  $\|n\| = 1$ ). Dann berechnet sich die Lichtintensität  $I_d$  an diesem Punkt durch

$$I_d := I_l \cdot I_m \cdot \max\left(0, \left\langle n, \frac{1}{\|\overrightarrow{PL}\|} \cdot \overrightarrow{PL} \right\rangle\right)$$

wobei  $I_l \in [0, 1]$  die Lichtintensität der Lichtquelle,  $I_m \in [0, 1]$  eine Materialkonstante und  $\max(a, b)$  das Maximum der Zahlen  $a$  und  $b$  ist. Um die gesamte

Lichtintensität zu berechnen wird noch ein konstanter, sogenannter ambienter Lichtanteil  $I_a$  hinzugefügt

$$I := I_d + I_a . \quad (13)$$

$I_a$  ist wie gesagt eine Konstante. Der Eindruck von ambientem Licht, also Licht, das aus keiner erkennbaren Richtung kommt und alles gleichmäßig erhellt, entsteht durch Lichtstrahlen, die sehr oft an Gegenständen reflektiert werden. Der Wert  $I$  muss gegebenenfalls auf das Intervall  $[0, 1]$  beschränkt werden.



Abbildung 31: Lambert Modell diffuser Reflektion

#### 4.3.3 Phong Modell

Das Phong-Modell erweitert das Lambert-Modell um den Effekt von spiegelnden Reflexionen. Die Idee ist, dass die Helligkeit zusätzlich zunimmt, je mehr die Betrachtungsrichtung dem perfekt reflektiertem Lichtstrahl entspricht.

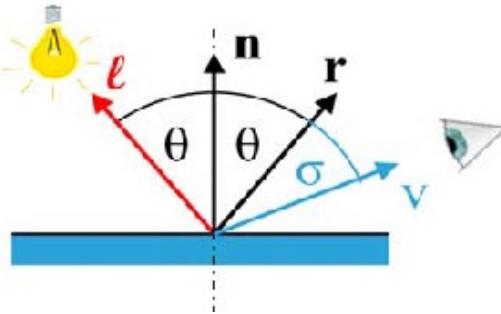


Abbildung 32: Die einzelnen Komponenten des Phong-Modells

Ist  $v$  der normierte Vektor in Richtung Kamera und  $l$  der normierte Vektor in Richtung Lichtquelle, dann definiert man die reflektierende Lichtintensität durch

$$I_s := I_l \cdot I_m \cdot s \cdot \max\left(0, \langle r, v \rangle\right)^h. \quad (14)$$

$I_l$  ist hierbei wieder die Intensität der Lichtquelle und  $I_m$  eine Materialkonstante. Beide liegen in einen Wertebereich von  $[0, 1]$ .  $h$  ist eine ganze Zahl im Wertebereich  $[0, \infty)$  und bestimmt die "härte" der Reflexion.  $s$  ist eine Fließkommazahl im Wertebereich  $(0, 1]$  und beeinflusst den Glanz der Oberfläche. Der Vektor  $r$  lässt sich berechnen durch

$$r = -l + 2 \cdot \langle n, l \rangle \cdot n \quad (15)$$

wobei  $n$  die Normale an dem betrachteten Oberflächenpunkt ist.

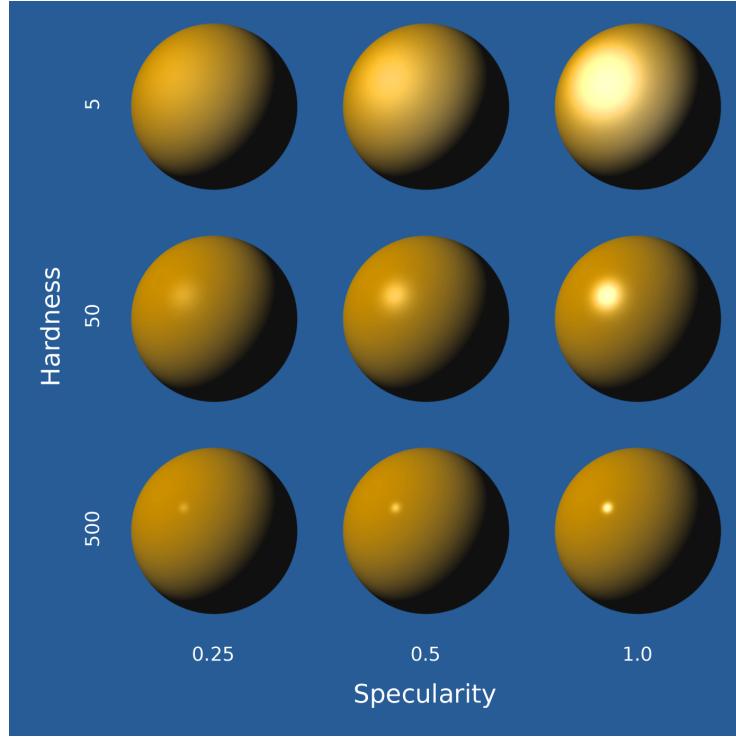


Abbildung 33: Phong Modell spiegelnder Reflektion

Im Phong-Modell wird nun die endgültige Lichtentität wie folgt zusammengesetzt:

$$I := I_s + I_d + I_a \quad (16)$$

wobei  $I_d$  die diffuse Lichtintensität aus dem Lambert-Modell und  $I_a$  den konstanten, ambienten Anteil bedeutet. Gegebenenfalls muss  $I$  wieder auf das Intervall  $[0, 1]$  beschränkt werden.

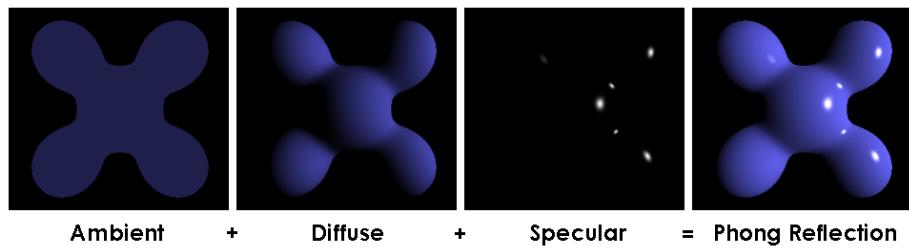


Abbildung 34: Die einzelnen Komponenten des Phong-Modells

Die Möglichkeiten, ein bestimmtes Material zu beschreiben, sind beim Phong-Modell beschränkt. Es gibt komplexere lokale Beleuchtungsmodelle, wie zum Beispiel das Cook-Toriente-Modell, die eine detaillierte Materialbeschreibung zulassen.

## 4.4 Shader und standard Algorithmen

### 4.4.1 Mittelung der Normalen

Stoßen mehrere Flächen mit Normalen  $\{\bar{N}_1, \bar{N}_2, \dots, \bar{N}_k\}$  an einem Vertex  $V$  zusammen, so bezeichnen wir mit  $\bar{N}_V := \frac{1}{k} \sum \bar{N}_i$  die gemittelte Normale.

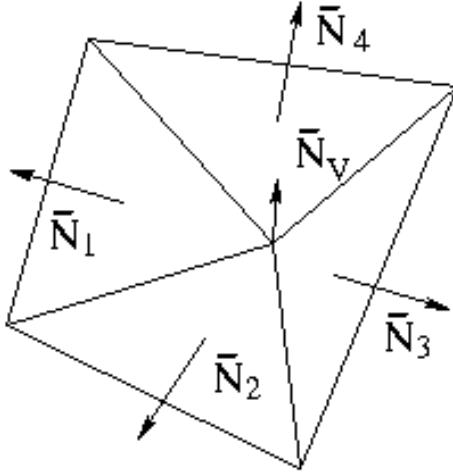


Abbildung 35: Verschiedene Normalen von angrenzenden Flächen

### 4.4.2 Flat-Shading, Gouraud und Phong Shading

Bezeichnen wir mit  $\bar{N}_{V_1}, \bar{N}_{V_2}, \bar{N}_{V_3}$  die gemittelten Normalen der Vertices  $V_1, V_2$  und  $V_3$  eines Dreiecks und mit  $L(N)$  die mit Hilfe eines Lichtmodells berechnete Lichtintensität in Abhängigkeit eines Normalenvektors  $N$ . Des weiteren bezeichnen wir mit  $Int(V, w_1, w_2, w_3)$  die lineare Interpolation zwischen den Werten bzw. Vektoren  $w_1, w_2, w_3$  an einem Punkt  $V$  innerhalb des Dreiecks  $V_1, V_2, V_3$ .

Man spricht vom Flat-Shading, wenn die gleiche Lichtintensität für das gesamte Dreieck (Fragment) verwendet wird. Es gibt zwei prominente Möglichkeiten, die Helligkeitswerte zu interpolieren, das Gouraud und das Phong Shading. Sei dazu  $V$  ein Punkt im Inneren des Dreiecks  $V_1, V_2, V_3$ . Die Lichtintensität an diesem Punkt berechnet sich dann beim Gouraud Shading durch

$$I_{Gouraud}(V) := Int(V, L(\bar{N}_{V_1}), L(\bar{N}_{V_2}), L(\bar{N}_{V_3})) \quad (17)$$

und beim Phong Shading durch

$$I_{Phong}(V) := L(Int(V, \bar{N}_{V_1}, \bar{N}_{V_2}, \bar{N}_{V_3})) \quad (18)$$



Abbildung 36: Flat, Gouraud und Phong-Shading

#### 4.4.3 Texturen und UV-Mapping

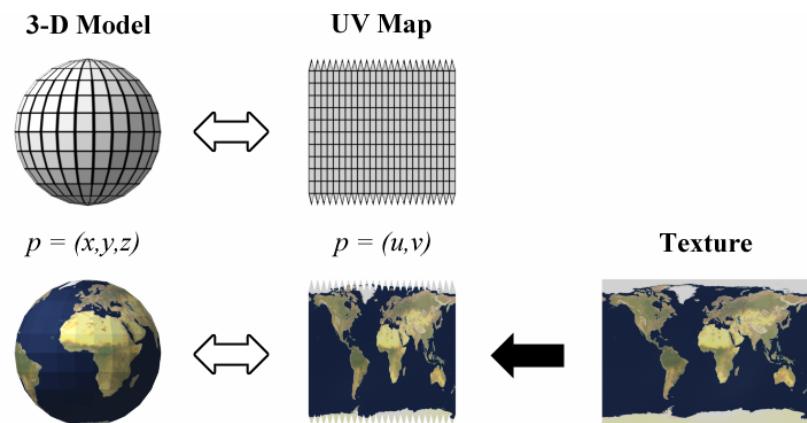


Abbildung 37: uv mapping



Abbildung 38: uv mapping

#### 4.4.4 Bumpmapping

Die relativen Höheninformationen liegen in einer Textur in Form von Graustufen vor – der sogenannten Heightmap. Jeder Grauwert steht für eine bestimmte Höhe. Normalerweise ist Schwarz (Wert 0) die „tiefste“ Stelle und Weiß (Wert: 255) die „höchste“. Bei der Lichtberechnung, die wesentlich von der Normale abhängt, wird die gegebene Normale so abgeändert, als gäbe es eine Vertiefung bzw. eine Erhöhung entsprechend der Höheninformation. Da Höhenunterschiede nur durch Veränderung der Helligkeit vorgegaukelt werden, die Flächen aber glatt bleiben, treten einige sichtbare Fehler auf: Bei flachem Betrachtungswinkel wirkt die Struktur stark verzerrt. Die Silhouette bleibt so eben wie beim ursprünglichen Objekt. Es wird ein glatter Schatten geworfen. Bumps werfen keine Schatten aufeinander.

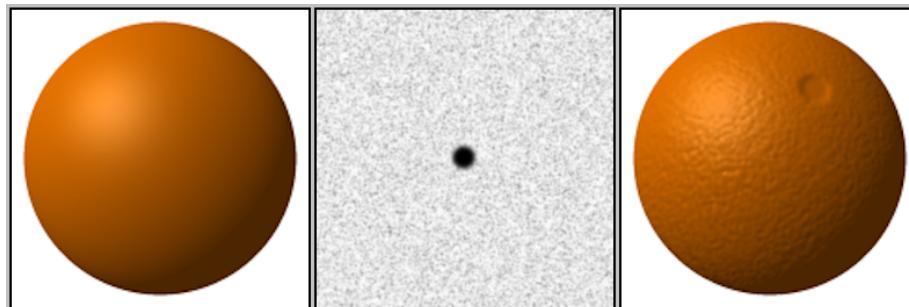


Abbildung 39: uv mapping

#### 4.4.5 Displacementmapping

Wie beim Bumpmapping wird eine Height-map als Textur mitgegeben. Im Gegensatz dazu werden die Punkte des Gitternetzes (Vertices) entsprechend diesen Texturinformationen entlang ihrer Normalen, das heißt senkrecht zur Oberfläche, tatsächlich verschoben. So ist es beispielsweise möglich, ein Höhenrelief durch das Anwenden einer Displacement Map auf eine ebene (planare) Oberfläche zu übertragen und dieser damit eine rauhe Struktur zu verleihen. Zusätzlich zur Verschiebung kann in Abhängigkeit von der Dichte des Drahtgitters dessen Verfeinerung notwendig werden. Man spricht dann von Tesselation.

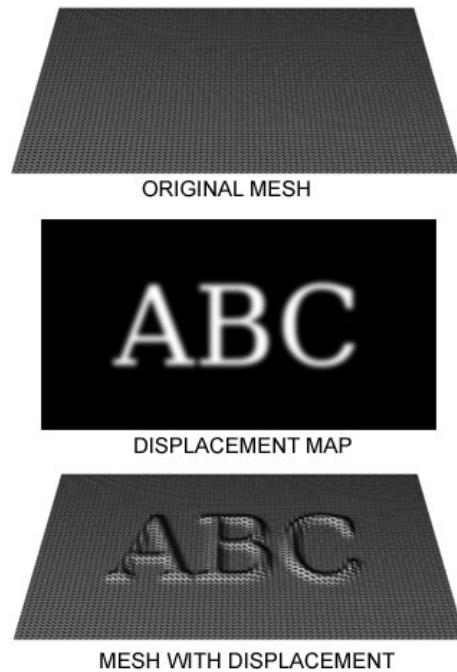


Abbildung 40: Displacementmapping

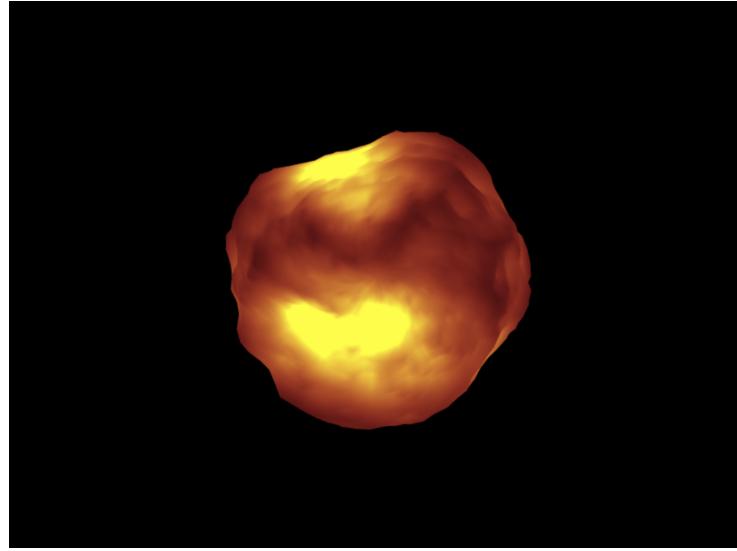
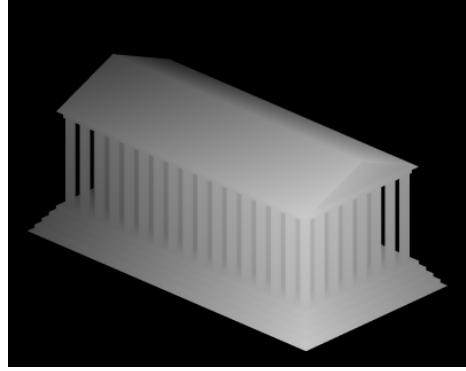


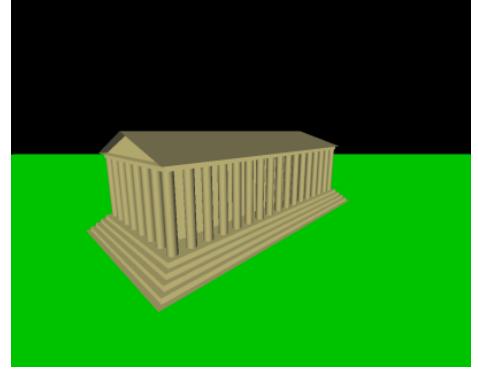
Abbildung 41: Lavaball

#### 4.4.6 Shadowmap

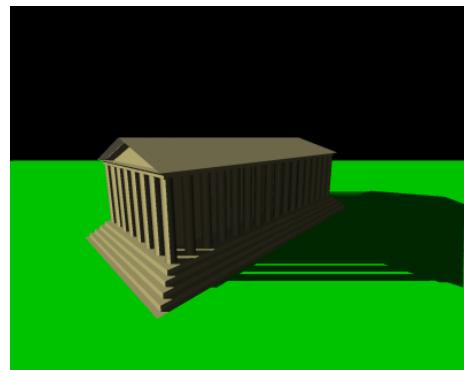
Das Rendern einer Szene mit Schatten unter Zuhilfenahme von einer Shadow Map geschieht im Wesentlichen in zwei Schritten. Als erstes wird die Szene aus der Sicht des Lichts gerendert und die Tiefeninformation (z-Buffer) in eine Textur gerendert. Anschließend wird die Szene normal gerendert, wobei für jedes Pixel der Abstand des zugehörigen Vertex zur Lichtquelle (in Clipping Koordinaten) mit dem Wert der Shadow Map verglichen wird. Ist der Wert in der Shadow Map kleiner, wird das Pixel schattiert gerendert.



(a) Tiefeninformation aus Sicht des Lichtes gerendert



(b) Rendering ohne Shadow map



(c) Rendering mit Shadow map

Probleme bereitet der sogenannte z-Buffer War. Abhilfe schafft eine Transformation der z-Buffer-Werte durch eine geeignete Funktion.

#### 4.4.7 Forward shading, Blending und Transparenz

Beim Forward shading wird die Lichtberechnung für jedes Vertex in dem entsprechenden Shader durchgeführt. Im Gegensatz dazu vergleiche deferred shading. Werden mehrere Render passes durchgeführt, so kann man über das Blending festlegen, wie die doppelten Fragmente gemischt werden. Transparenz kann man zum Beispiel realisieren, indem man die transparenten Objekte mit entsprechendem Blending über die vorher gerenderten Szene rendert. Ein lineares Blending wird zum Beispiel durch die Funktion

$$p_{neu} = \alpha \cdot p_1 + (1 - \alpha)p_2 \quad (19)$$

realisiert. Mehrere Lichtquellen können entweder direkt im Shader oder durch mehrfaches Rendern der Szene mit den verschiedenen Lichtquellen und entspre-

chendem Blending realisiert werden.

#### 4.4.8 Deferred shading

Der Begriff Deferred Shading (zu dt. "verzögertes Schattieren") oder auch Deferred Lighting ("verzögerte Beleuchtung") beschreibt eine Technik, mit deren Hilfe die Geometrieverarbeitung von der Lichtberechnung getrennt werden kann. Dies erlaubt hunderte Lichtquellen in einer hoch-komplexen Szene.

Um eine Szene mit einem traditionellen Forward-Renderer zu beleuchten, wird normalerweise jedes Objekt der Szene mit den entsprechenden Beleuchtungsparametern der Lichtquelle gezeichnet. Die Lichtberechnungen werden für jedes Vertex aufgerufen.

Deferred Shading verfolgt einen anderen Ansatz: Für jeden Pixel werden die Daten aus dem G-Buffer ausgelesen und die Beleuchtung entsprechend dieser Daten im Fragmentshader einer minimalen, bildschirmfüllenden Geometrie berechnet.

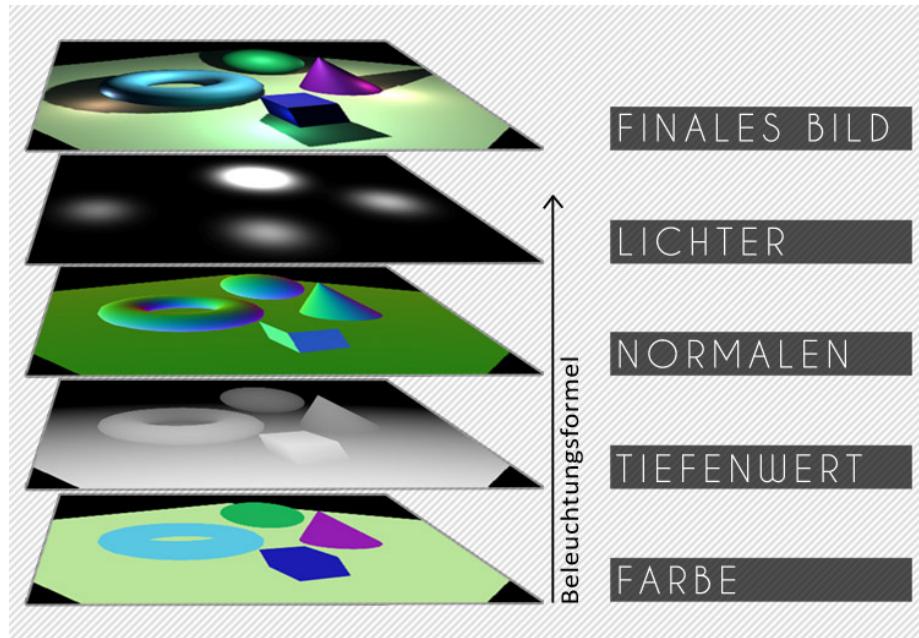


Abbildung 42: G-Buffer beim deferred shading

Mit dieser Methode sind hunderte Lichter gleichzeitig in einer hoch-komplexen Szene möglich. Der Vorteil ist, dass die Lichtberechnungen somit nur für wirklich sichtbaren Pixel des tatsächlichen Ausgabe-Bildes berechnet werden. Das Verfahren ist nur dann sinnvoll, wenn man es mit einer sehr komplexen Geometrie und sehr vielen Lichtquellen zu tun hat.

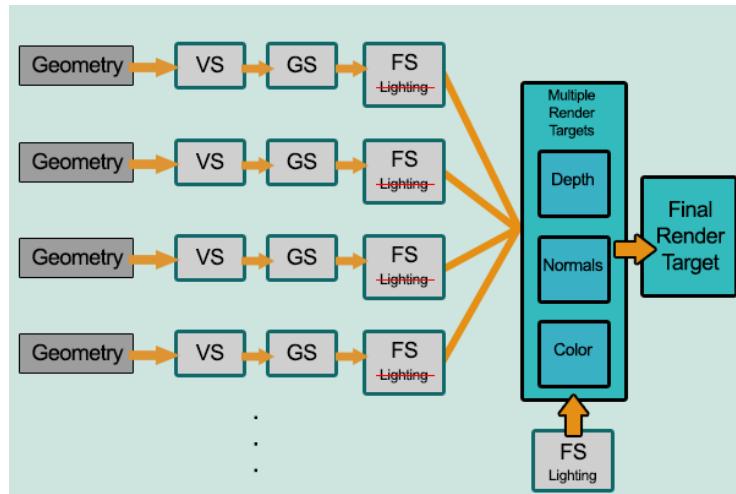


Abbildung 43: Pipeline beim deferred rendering



Abbildung 44: Szene gerendert mit forward shading.



Abbildung 45: Szene gerendert mit deferred shading. Der verbesserte Eindruck des Tageslichtes wird durch eine sehr hohe Anzahl von Lichtquellen erzeugt.

## 4.5 GLSL via WebGL

### 4.5.1 Minimaler Shader

Minimaler Shader auf bildschirmfüllender Minimal-Geometrie

```
1 <!DOCTYPE html>
2 <!--
3 To change this license header, choose License Headers
in Project Properties.
4 To change this template file, choose Tools | Templates
5 and open the template in the editor.
6 -->
7 <html>
8   <head>
9     <title>TODO supply a title</title>
10    <meta charset="UTF-8">
11    <meta name="viewport" content="width=device-
12      width,initial-scale=1.0">
13  </head>
14  <body>
15    <canvas id="glcanvas"></canvas>
16
17    <script id="2d-vertex-shader" type="x-shader/x
-vertex">
        attribute vec2 a_position;
```

```

18         uniform float t;
19         varying float T;
20         void main() {
21             // gl_Position = vec4(a_position, 0.0, t)
22             ;
23             T = t;
24             gl_Position = vec4(a_position[0],
25                 a_position[1], 0.0, 1.0);
26         }
27     </script>
28
29     <script id="2d-fragment-shader" type="x-shader
30         /x-fragment">
31         precision mediump float;
32         varying float T;
33         void main() {
34             gl_FragColor = vec4(0.0 ,1.0,0.0,1.0);
35
36         }
37     </script>
38
39
40         var shaderScript;
41         var shaderSource;
42         var vertexShader;
43         var fragmentShader;
44         var program;
45         var canvas;
46         var gl;
47         var buffer;
48         window.onload = init;
49
50
51         var someProgress;
52
53         function init() {
54             // Get A WebGL context
55             canvas = document.getElementById("glcanvas
56             ");
57             canvas.width = 400;
58             canvas.height = 300;
59

```

```

60
61     gl = canvas.getContext("webgl");
62
63
64     gl.viewport(0, 0, gl.drawingBufferWidth,
65                 gl.drawingBufferHeight);
66
67     //compile and link shaders
68     shaderScript = document.getElementById("2d
69         -vertex-shader");
70     shaderSource = shaderScript.text;
71
72     vertexShader = gl.createShader(gl.
73         VERTEX_SHADER);
74     gl.shaderSource(vertexShader, shaderSource
75         );
76     gl.compileShader(vertexShader);
77
78     shaderScript = document.getElementById("2d
79         -fragment-shader");
80     shaderSource = shaderScript.text;
81     fragmentShader = gl.createShader(gl.
82         FRAGMENT_SHADER);
83     gl.shaderSource(fragmentShader,
84         shaderSource);
85     gl.compileShader(fragmentShader);
86
87
88
89
90
91
92     // Create an empty buffer
93     buffer = gl.createBuffer();
94
95     someProgress = 1.0;
96
97     //start gameloop
98     render();

```

```

99      }
100
101     function render() {
102
103         // recursive gameloop/animation frame
104         // offered by browser
105         window.requestAnimationFrame(render);
106
107         // clear background and paint it red
108         gl.clearColor(1.0, 0.0, 0.0, 1.0);
109
110         gl.clear(gl.COLOR_BUFFER_BIT);
111
112
113
114         // use shader program
115         gl.useProgram(program);
116
117
118
119         // bind geometry to shader program
120         gl.bindBuffer(gl.ARRAY_BUFFER, buffer);
121
122         gl.bufferData(
123             gl.ARRAY_BUFFER,
124
125             // data array values
126             new Float32Array([
127                 -1.0, -1.0,
128                 1.0, -1.0,
129                 -1.0, 1.0,
130                 -1.0, 1.0,
131                 1.0, -1.0,
132                 1.0, 1.0]),
133
134         gl.STATIC_DRAW);
135
136         positionLocation = gl.getAttribLocation(
137             program, "a_position");
138         gl.enableVertexAttribArray(
139             positionLocation);
140
141         gl.vertexAttribPointer(positionLocation,
142             2, gl.FLOAT, false, 0, 0);

```

```

141
142
143     var tLocation = gl.getUniformLocation(
144         program, "t");
145     gl.uniform1f(tLocation, someProgress);
146
147     someProgress += 0.01;
148
149     gl.drawArrays(gl.TRIANGLES, 0, 6);
150   }
151
152
153
154
155 </body>
156 </html>

```

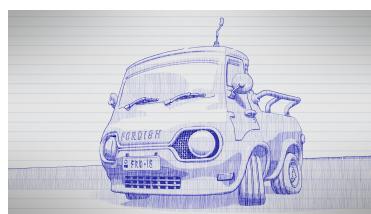
## 5 Raytracing



(a)



(b)



(c)



(d)

## 5.1 Raytracing Algorithmen

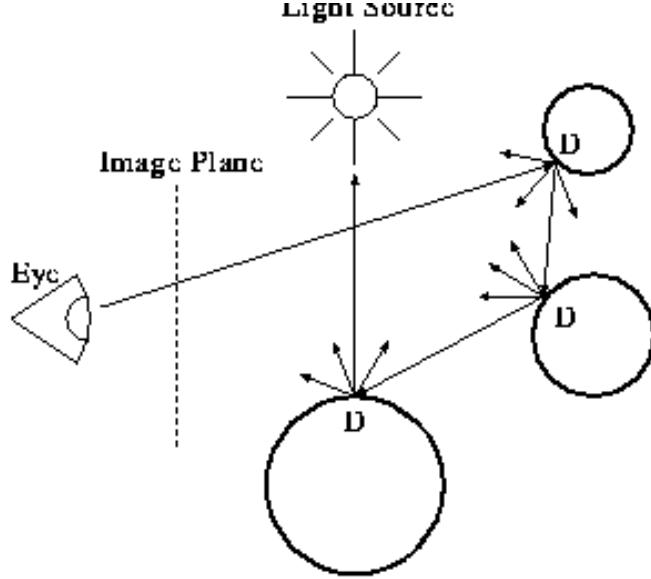


Abbildung 46: Strahlenverfolgung

### 5.1.1 Die Rendergleichung (Erste und zweite Form)

Durch Hinzunahme des von dem Oberflächenpunkt emittierten Lichtes  $L_e(x, \omega_o)$  zu dem an diesem Punkt reflektiert Lichtes erhalten wir die sogenannte Rendergleichung

$$L_o(x, \omega_o) = L_e(x, \omega_o) + \int_{H^2} f_r(x, \omega, \omega_0) \cdot L_i(x, \omega) \cdot \cos(\theta) d\omega, \quad (20)$$

wobei  $L_e(x, \omega_o)$  die von der Oberfläche emittierte Strahlungsleistung ist.

**Bemerkung 19.** Für transparente Oberflächen muss über  $S^2$  integriert werden.

Definiert man die Menge  $\Omega$  als die Menge aller Flächen in der Szene und

$$V(x, y) := \begin{cases} 1 & \text{falls } \overline{xy} \cap (\Omega - \{x, y\}) = \emptyset \\ 0 & \text{sonst} \end{cases} \quad (21)$$

so ist  $L_i(x, \omega_y) = V(x, y) \cdot L_o(y, \omega_x)$  (Energieerhaltung). Zusammen mit der Beziehung

$$d\omega = \frac{1}{\|x - y\|^2} \cdot \cos(\theta_y) dA_y \quad (22)$$

und der Definition

$$G(x, y) := V(x, y) \frac{\cos(\theta_x) \cdot \cos(\theta_y)}{\|x - y\|^2} \quad (23)$$

erhält man die Rendergleichung in zweiter Form

$$L_o(x, \omega_o) = L_e(x, \omega_o) + \int_{\Omega} f_r(x, \bar{y}, \omega_o) \cdot L_o(y, \bar{y}) \cdot G(x, y) \cdot dA_y . \quad (24)$$

### 5.1.2 Pfadformulierung der Rendergleichung

Definiert man den sogenannten Transport Operator

$$(T \circ L)(x, \omega) := \int_{H^2} f_r(x, \omega, \omega_0) \cdot L(x, \omega) \cdot \cos(\theta) d\omega , \quad (25)$$

so lässt sich die Rendergleichung schreiben als

$$L = L_e + T \circ L . \quad (26)$$

oder mit der Identitätsabbildung  $id$  als

$$L_e = (id - T) \circ L . \quad (27)$$

Wir erhalten damit

$$L = (id - T)^{-1} \circ L_e . \quad (28)$$

wobei

$$(id - T)^{-1} = \sum_{i=0}^{\infty} T^i . \quad (29)$$

eine konvergente Reihe ist, falls  $f_r(x, \omega, \omega_0) < 1$  gilt. Letztere Bedingung bedeutet anschaulich, dass kein Material perfekt reflektierend ist. Die Gleichung ist eine Verallgemeinerung der geometrischen Reihe auf Operatoren.

## 5.2 Raycasting

### 5.3 "Klassisches" Raytracing

Das klassische Raytracing nach Whitted ist durch folgenden rekursiven Algorithmus gegeben:

•

## 5.4 BRDF-Shader

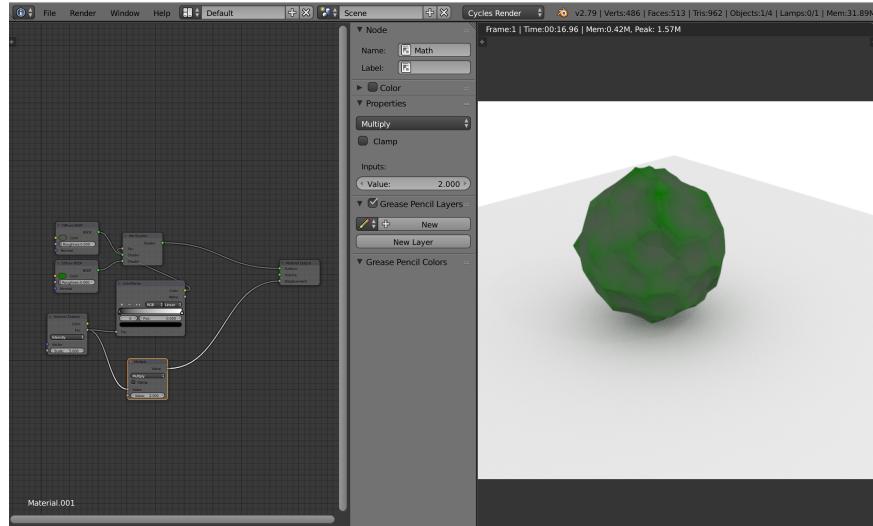


Abbildung 47: Grafische BRDF-Shader Programmierung in Blender

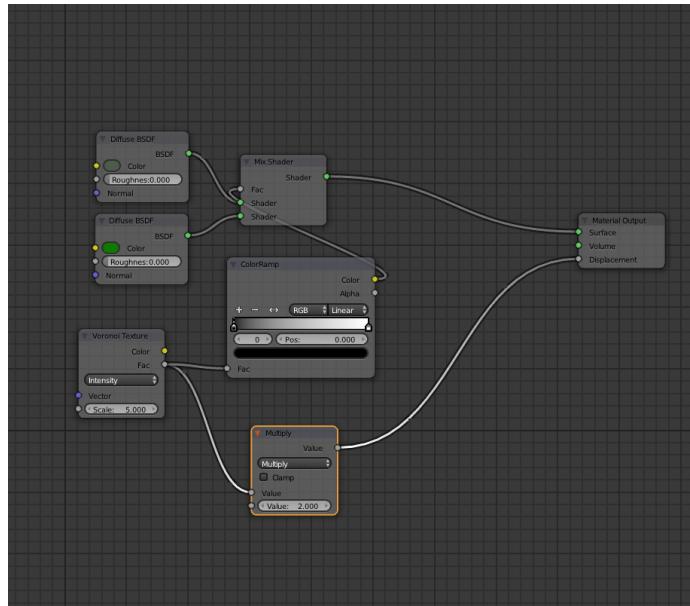


Abbildung 48: Grafische BRDF-Shader Programmierung in Blender

## 5.5 Radiosity Verfahren

Das Radiosity Verfahren löst die Rendergleichung näherungsweise im Fall rein diffuser Oberflächen beziehungsweise rein Lambertscher Strahler. In diesem Fall sind  $L(x, \omega) = L(x)$  und  $f_r(x, \omega, \omega') = f(x)$  konstant für alle Richtungen  $\omega$  und  $\omega'$  und mit der daraus resultierenden Beziehung

$$B(x) = \int_{H^2} L(x) \cos \theta d\omega \quad (30)$$

$$\Leftrightarrow B(x) = L(x) \underbrace{\int_{H^2} \cos \theta d\omega}_{=\pi} \quad (31)$$

$$\Leftrightarrow \frac{B(x)}{\pi} = L(x) . \quad (32)$$

erhält man durch Einsetzen in die Rendergleichung die Radiosity Gleichung

$$B_o(x) - B_e(x) - f(x) \cdot \int_{\Omega} B_o(y) \cdot G(x, y) \cdot dA_y = 0 . \quad (33)$$

Eine Möglichkeit diese approximativ zu lösen ist die Galerkin-Methode, die eine Näherungsweise Lösung dieses Problem auf die Lösung eines linearen Gleichungssystems reduziert. Hierzu unterteilt (approximiert) man alle Oberflächen in  $\Omega$  mit endlich vielen planeren Flächenstücken  $F_i$ , welche auch Patches genannt werden. Des weiteren definiert man die sogenannten Basisfunktionen

$$\phi_i(x) := \begin{cases} 1 & \text{falls } x \in F_i \\ 0 & \text{sonst} \end{cases} \quad (34)$$

und Approximiert alle beteiligten Funktionen durch Linearkombinationen dieser Funktionen, was zu der Gleichung

$$\sum_i B_i \phi_i = \sum_i E_i \phi_i + (\sum_i f_i \phi_i) \cdot \sum_i \int_{F_i} B_i \phi_i \cdot G(x, y) \cdot dA_y \quad (35)$$

führt. Multipliziert man beide Seiten mit  $\phi_j$  und Integriert anschliessend beide Seiten über  $F_j$  erhält man die diskrete Radiosity Gleichung

$$B_j = E_j + f_j \sum_i F_{ji} \cdot B_i , \quad (36)$$

wobei  $A_j$  der Flächeninhalt des Patches  $F_j$  und

$$F_{ji} := \frac{1}{A_j} \int_{F_j} \int_{F_i} G(x, y) \cdot dA_y \quad (37)$$

ist. Dies führt auf das lineare Gleichungssystem  $M \cdot B = E$  mit

$$M := (m_{ij}) \text{ mit } m_{ij} = \begin{cases} 1 - f_i \cdot F_{ij} & \text{für } i = j \\ -f_i \cdot F_{ij} & \text{sonst} \end{cases} \quad (38)$$

$$B := \begin{pmatrix} B_1 \\ \vdots \\ B_n \end{pmatrix} \quad (39)$$

$$E := \begin{pmatrix} E_1 \\ \vdots \\ E_n \end{pmatrix} \quad (40)$$

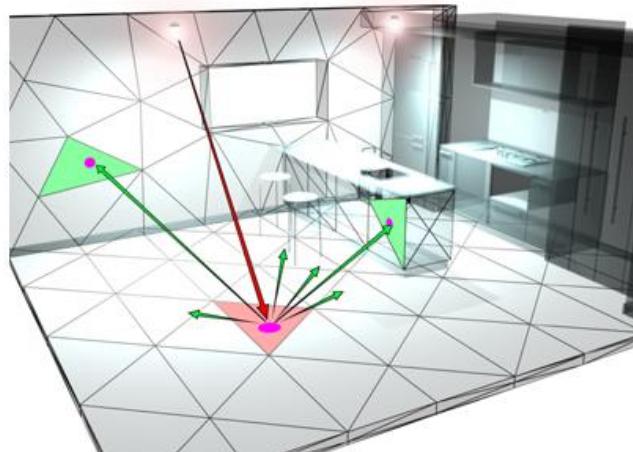


Abbildung 49: Patches und Übertragung der Lichtenergie zwischen einzelnen Patches

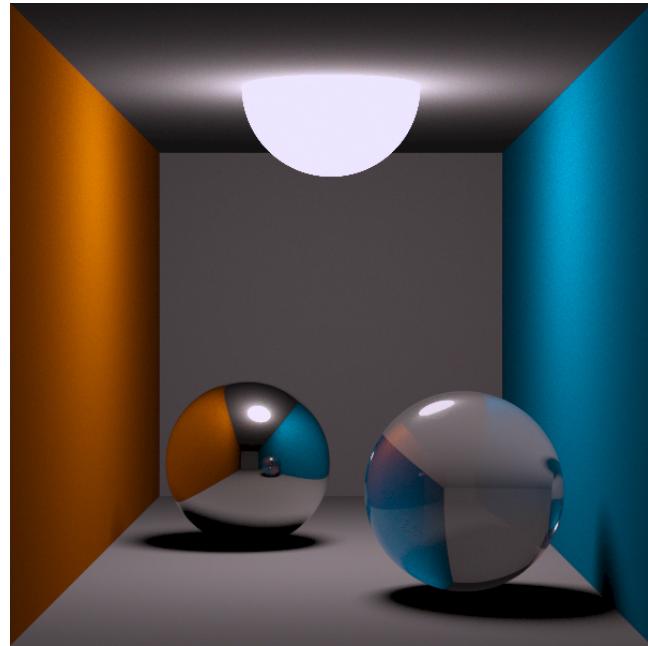


Abbildung 50: Rendering der Cornell Box mit rein diffusem Beleuchtungsmodell

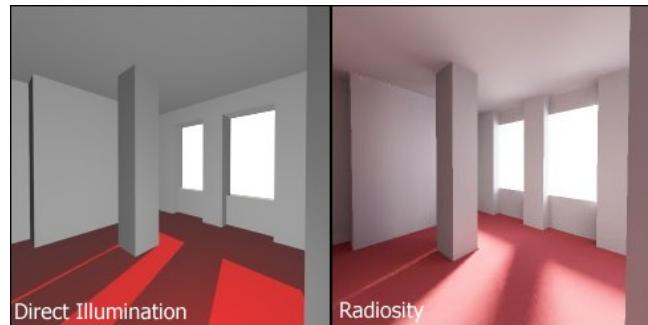
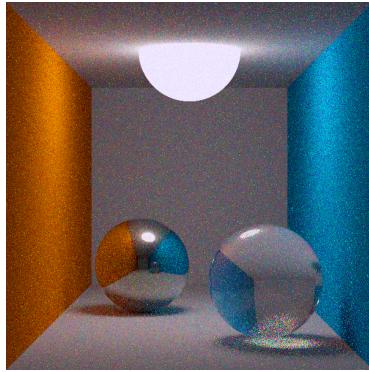
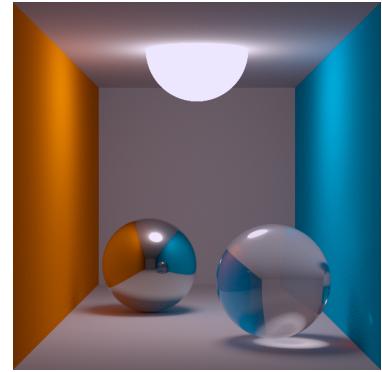


Abbildung 51: Direkte Beleuchtung im Vergleich mit dem Radiosity Verfahren.

## 5.6 Monte-Carlo Raytracing und Pathtracing



(a) Rendering der Cornell Box mit Pathtracing und wenig Samples



(b) Rendering der Cornell Box mit Pathtracing und vielen Samples

## 5.7 Monte Carlo Methode

Die Monte Carlo Methode bezeichnet die Anwendung des Gesetzes der großen Zahlen auf die Rendergleichung.

### 5.7.1 Raymarching

### 5.8 Klassifikation der Verfahren

### 5.9 Datenstrukturen für Bereichsabfragen

### 5.10 Labor

#### 5.10.1 Blender

#### 5.10.2 Echtzeitfähiges Raymarching in WebGL

## 6 Animation und Simulation

### 6.1 Keyframe Animation

### 6.2 Partikelsysteme

### 6.3 Elemente der Kollisionserkennung

### 6.4 Labor

## **Tabellenverzeichnis**

## Abbildungsverzeichnis

1	Kardansche Aufhängung und Gimbal lock . . . . .	12
2	Roll, Nick und Gier Winkel . . . . .	13
3	Eine Helix. Quelle:Wikipedia . . . . .	18
4	Spektrum . . . . .	21
5	Partikelstrom in einem Kanal . . . . .	21
6	Partikelstrom . . . . .	22
7	BRDF Funktion . . . . .	24
8	Geschlossenes Polygon . . . . .	26
9	Gerichtete Polygone . . . . .	28
10	Orientierbare Fläche . . . . .	30
11	Das äußere Normalenfeld . . . . .	31
12	Geschlossene Netze der Geschlechter 0 bis 3. Quelle:Wikipedia . .	32
13	Halfedge. Quelle:Wikipedia . . . . .	35
14	Die Bernsteinpolynome $B_i^4$ und deren Summe. Quelle:Wikipedia .	36
15	Pascalsches Dreieck . . . . .	36
16	Auswertung einer Beziekurve zum Zeitpunkt $t_0$ durch iterative Streckenteilung nach de Casteljau . . . . .	38
17	Bezier Patch . . . . .	39
18	Ein Rendering der Utah Teekanne . . . . .	40
19	Computergrafik-Pipeline . . . . .	41
20	Zentralprojektion . . . . .	42
21	Geometrie-Pipeline . . . . .	43
22	Projektion und Sichtvolumen . . . . .	43
23	Zentralprojektion . . . . .	44
24	Zentralprojektion . . . . .	44
25	Zentralprojektion . . . . .	45
26	Rasterung nach Bresenham Algorithmus . . . . .	45
27	Schrittbeachtung Bresenham Algorithmus . . . . .	46
28	Eine Schrift mit und ohne Antialiasing . . . . .	48
29	Scanline-Verfahren beim Gouraud-Shading . . . . .	49
30	Z-Buffer-War . . . . .	50
31	Lambert Modell diffuser Reflexion . . . . .	51
32	Die einzelnen Komponenten des Phong-Modells . . . . .	52
33	Phong Modell spiegelnder Reflexion . . . . .	53
34	Die einzelnen Komponenten des Phong-Modells . . . . .	53
35	Verschiedene Normalen von angrenzenden Flächen . . . . .	54
36	Flat, Gouraud und Phong-Shading . . . . .	55
37	uv mapping . . . . .	55
38	uv mapping . . . . .	56
39	uv mapping . . . . .	56
40	Displacementmapping . . . . .	57
41	Lavaball . . . . .	58
42	G-Buffer beim deferred shading . . . . .	60
43	Pipeline beim deferred rendering . . . . .	61

44	Szene gerendert mit forward shading. . . . .	61
45	Szene gerendert mit deferred shading. Der verbesserte Eindruck des Tageslichtes wird durch eine sehr hohe Anzahl von Lichtquel- len erzeugt. . . . .	62
46	Strahlenverfolgung . . . . .	67
47	Grafische BRDF-Shader Programmierung in Blender . . . . .	69
48	Grafische BRDF-Shader Programmierung in Blender . . . . .	69
49	Patches und Übertragung der Lichtenergie zwischen einzelnen Patches . . . . .	71
50	Rendering der Cornell Box mit rein diffusem Beleuchtungsmodell	72
51	Direkte Beleuchtung im Vergleich mit dem Radiosity Verfahren.l .	72