

**Achtung:** Dieses Dokument ist *nur* eine Mitschrift der Vorlesung "Computergrafik" SoSe2010. Sie wurde während der Vorlesung angefertigt. Es wird aber seitens des Autors keine Garantie auf Vollständigkeit und Richtigkeit des Inhalts gegeben.

## **Mitschrift**

### **Computergrafik**

gehalten von Prof. Dr. Günther Rote

23. Juni 2010



# Inhaltsverzeichnis

<b>1</b>	<b>Einführung</b>	<b>5</b>
1.1	Organisatorisches . . . . .	5
1.1.1	Übungsblätter: . . . . .	5
1.1.2	Programmierung . . . . .	5
1.2	Übersicht . . . . .	5
1.2.1	Fahrplan . . . . .	5
<b>2</b>	<b>Koordinatensysteme, geometrische Transformationen</b>	<b>7</b>
2.1	kartesische Koordinaten . . . . .	7
2.2	Geometrische Transformationen . . . . .	7
2.3	Homogene Koordinaten . . . . .	8
2.3.1	Allgemeine affine Transformation in homogenen Koordinaten . . . . .	9
2.4	Die projektive Ebene . . . . .	9
2.4.1	Geraden in der projektiven Ebene . . . . .	9
2.4.2	Modelle der projektiven Ebene . . . . .	11
2.4.3	Projektive Punkte zu kartesischen Koordinaten . . . . .	12
2.5	Allgemeine projektive Transformationen . . . . .	13
2.6	Transformation im dreidimensionalen Raum . . . . .	15
2.6.1	Affine Transformation im dreidimensionalen Raum . . . . .	15
2.6.2	projektive Transformationen im dreidimensionalen Raum . . . . .	16
2.7	Projektionen und Perspektive . . . . .	16
2.8	Koordinaten in der Praxis . . . . .	17
2.9	„rendering pipeline“ – vom geometrischen Modell zum Rasterbild . . . . .	22
<b>3</b>	<b>Licht und Farben</b>	<b>23</b>
3.1	Farbsehen im menschlichen Auge . . . . .	23
<b>4</b>	<b>Rasterung von Strecken und Kreisen</b>	<b>25</b>
4.1	Strecken . . . . .	25
4.1.1	Bresenham-Algorithmus . . . . .	27
4.1.2	Midpoint Line Algorithmus . . . . .	28
4.2	Kreise . . . . .	29
4.3	Schwachstellen der Rasterung (Aliasing) . . . . .	32
4.4	Antialiasing . . . . .	32
<b>5</b>	<b>Helligkeit und Farbe in der Computergrafik</b>	<b>35</b>
5.1	Helligkeit . . . . .	35
5.2	Additive Farbsysteme . . . . .	35
5.2.1	Das RGB-Farbsystem . . . . .	35
5.2.2	Farbsechseck bzw. Farbkreis . . . . .	37
5.2.3	Andere Farbsysteme . . . . .	37
5.2.4	YCrCb/YPrPb-Farbsystem . . . . .	40
5.3	Subtraktive Farbmischung (z. B. beim Drucken) . . . . .	40
5.3.1	CMY-System . . . . .	41
5.3.2	CMYK-System (Vierfarbdruck) . . . . .	41
5.4	Digitale Farbdarstellung . . . . .	41
5.4.1	Farbpaletten . . . . .	41
<b>6</b>	<b>Beleuchtung</b>	<b>43</b>
6.1	Diffuse Reflexion . . . . .	43
6.2	Spiegelnde Reflexion . . . . .	45
6.3	Gesamtbeleuchtung . . . . .	47
6.4	Schattierung (Shading) . . . . .	47
6.4.1	Ausfüllen einer gerasterten Fläche . . . . .	49

6.5	Entfernen verdeckter und teilweise verdeckter Flächen . . . . .	50
6.5.1	Painter's Algorithmus . . . . .	50
6.5.2	Tiefenpuffer (z-Puffer) . . . . .	51
6.5.3	Darstellung einer Szene durch Überstreichen (Scanline-Algorithmus) . . . . .	51
6.6	Anzeigen von benachbarten Flächenstücken (Dreiecken) . . . . .	53
6.7	Lineare Interpolation in Weltkoordinaten . . . . .	54
6.8	Transformation von Normalvektoren bei affinen Transformationen . . . . .	55
<b>7</b>	<b>Texturen</b>	<b>57</b>
<b>8</b>	<b>Schatten</b>	<b>59</b>
<b>9</b>	<b>Clipping</b>	<b>61</b>
9.1	Algorithmus von Cohen-Sutherland . . . . .	61
<b>10</b>	<b>Geometrische Modellierung</b>	<b>63</b>
10.1	„Rendering pipeline“ . . . . .	63
10.2	Geometrische Modellierung . . . . .	63
10.2.1	Wie kann man geometrische Formen darstellen? . . . . .	63
10.2.2	Darstellung polyedrischer Flächen bzw. Polygongittern . . . . .	64
10.3	Darstellung von geometrischen Modellen in Grafikbibliotheken . . . . .	65
10.4	Constructive Solid Geometry (CSG) . . . . .	66
10.4.1	Grundlagen . . . . .	66
10.4.2	Darstellung in der Computergrafik . . . . .	67
<b>11</b>	<b>Strahlverfolgung (Raytracing)</b>	<b>69</b>
11.1	Gesetzmäßigkeiten . . . . .	69
11.1.1	Reflexion . . . . .	69
11.1.2	Brechung . . . . .	69
11.1.3	Zusammenfassung . . . . .	69
11.1.4	Einheiten . . . . .	70
11.1.5	Diffuse Reflexion . . . . .	70
11.2	Einsatz der physikalischen Grundlagen beim Raytracing . . . . .	71
11.3	Globale Beleuchtung . . . . .	71
11.3.1	Rendering-Gleichung . . . . .	71
11.3.2	Radiosity-Verfahren . . . . .	72
<b>12</b>	<b>Transparenz</b>	<b>75</b>
12.1	RGBA-Modell bzw. $(r, g, b, \alpha)$ . . . . .	75
12.2	Kombination mit Tiefenpuffer . . . . .	75
<b>13</b>	<b>Spline-Kurven und -Flächen</b>	<b>77</b>
13.1	Bézierkurven (Bézier-Splines) . . . . .	77
13.1.1	Der Algorithmus von de Casteljau . . . . .	79





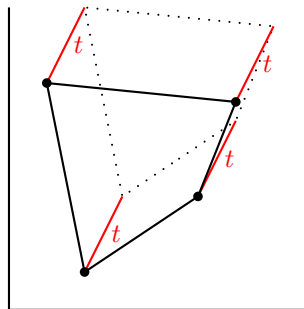
## 2 Koordinatensysteme, geometrische Transformationen

### 2.1 kartesische Koordinaten



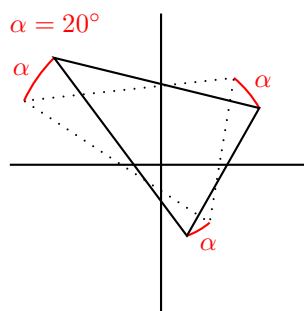
### 2.2 Geometrische Transformationen

- *Translation*:  $p \mapsto p + t$       $t \in \mathbb{R}^2$ , Translationsvektor



- *Rotation* (um den Ursprung  $\begin{pmatrix} 0 \\ 0 \end{pmatrix}$ ):

$$p \mapsto M \cdot p \quad M = \begin{pmatrix} \cos \alpha & -\sin \alpha \\ \sin \alpha & \cos \alpha \end{pmatrix}, \text{ Rotationsmatrix}$$



- *Rotation* um den Punkt  $c$ :  $p \mapsto M(p - c) + c = Mp + (c - Mc)$ ,      $c \mapsto c$
- *gleichförmige Skalierung*:

$$p \mapsto \lambda \cdot p = \begin{pmatrix} \lambda & 0 \\ 0 & \lambda \end{pmatrix} \cdot p, \quad \lambda \neq 0$$

$$\lambda = 1 \quad p \mapsto -p = \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix} \cdot p = \text{Spiegelung am Ursprung} = \text{Rotation um } 180^\circ$$

- *Ungleichförmige Skalierung*:

$$M = \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} \quad p \mapsto M \cdot p$$

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \lambda_1 x \\ \lambda_2 y \end{pmatrix}$$

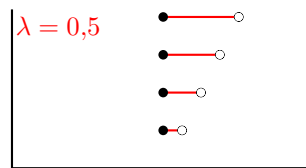
$M = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}$  resultiert in der Spiegelung an der  $x$ -Achse

$M = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$  resultiert in der Spiegelung an der  $y$ -Achse

- *Scherung*

$$M = \begin{matrix} \text{Scherung auf der } x\text{-Achse} \\ \begin{pmatrix} 1 & \lambda \\ 0 & 1 \end{pmatrix} \end{matrix} \left( \begin{matrix} \text{Scherung auf der } y\text{-Achse} \\ \text{oder} \end{matrix} \begin{pmatrix} 1 & 0 \\ \lambda & 1 \end{pmatrix} \right)$$

$$\begin{pmatrix} x \\ y \end{pmatrix} \mapsto \begin{pmatrix} 1 & \lambda \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x + \lambda y \\ y \end{pmatrix}$$



Flächeninhalt:

- Translationen, Rotationen, Scherungen und Spiegelungen ändern den Flächeninhalt nicht.
- Skalierung ändert den Flächeninhalt um den Faktor  $\lambda_1 \cdot \lambda_2$

**Definition** Eine Verknüpfung mehrerer dieser Transformationen bildet eine **affine Transformation**. Allgemein ist diese:

$$p \mapsto M \cdot p = b, \quad M \in \mathbb{R}^{2 \times 2}, b \in \mathbb{R}^2, \det M \neq 0$$

Der Flächeninhalt ändert sich um den Faktor  $\det M$

**Definition** Die Verknüpfung von Translation, Rotation und Spiegelung heißt **starre Bewegung** oder **Isometrie**. Allgemein ist diese:

$$p \mapsto Mp + t \text{ mit } \textbf{orthogonaler Matrix } M \text{ (d. h. } \det M = \pm 1)$$

die Isometrien zerfallen:

- **orientierungserhaltende** ( $\det M = 1$ ) und
- **orientierungsumkehrende** ( $\det M = -1$ ) Isometrien

## 2.3 Homogene Koordinaten

**Definition** **Homogene Koordinaten:** Statt  $p = \begin{pmatrix} x \\ y \end{pmatrix}$  verwendet man eine dritte Koordinate  $p = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$

**Konvention** Die Koordinaten  $\begin{pmatrix} x \\ y \\ z \end{pmatrix}$  und  $\begin{pmatrix} \lambda x \\ \lambda y \\ \lambda z \end{pmatrix}$  stellen denselben Punkt dar ( $\lambda \neq 0$ )

Der Punkt  $\begin{pmatrix} x \\ y \\ z \end{pmatrix}$  mit  $z \neq 0$  hat die kartesischen Koordinaten  $\begin{pmatrix} \frac{x}{z} \\ \frac{y}{z} \end{pmatrix}$



### 2.3.1 Allgemeine affine Transformation in homogenen Koordinaten

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \mapsto \underbrace{\begin{pmatrix} m_{11} & m_{12} & b_1 \\ m_{21} & m_{22} & b_2 \\ 0 & 0 & 1 \end{pmatrix}}_{M'} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \mapsto \begin{pmatrix} m_{11}x + m_{12}y + b_1 \\ m_{21}x + m_{22}y + b_2 \\ 1 \end{pmatrix}$$

Die Matrizen  $M'$  und  $\lambda M'$  beschreiben dieselbe Transformation ( $\lambda \neq 0$ )

$$p \mapsto M'p \text{ mit } M' = \begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ 0 & 0 & m_{33} \end{pmatrix} \text{ und } \det M' \neq 0$$

$$\det M' \neq 0 \Leftrightarrow m_{33} \neq 0 \wedge \begin{vmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{vmatrix} \neq 0$$

$\Rightarrow$  o. B. d. A. kann man auch  $m_{33} = 1$  annehmen (Dann kann man die dritte Zeile auch weglassen).

## 2.4 Die projektive Ebene

**Definition** Die (reelle) **projektive Ebene**  $P^2$  besteht aus den Äquivalenzklassen von Punkten  $\begin{pmatrix} x \\ y \\ z \end{pmatrix} \neq \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$ ,

wobei  $\begin{pmatrix} x \\ y \\ z \end{pmatrix}$  und  $\begin{pmatrix} \lambda x \\ \lambda y \\ \lambda z \end{pmatrix}$  denselben Punkt darstellen ( $\lambda \neq 0$ )

### 2.4.1 Geraden in der projektiven Ebene

Gerade in  $\mathbb{R}^2$  (kartesische Koordinaten):

$$y = ax + b \text{ (Gerade darf nicht senkrecht sein)}$$

$$ax + by = -c$$

$$\Updownarrow$$

Gerade in Homogenen Koordinaten

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \rightarrow \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

$$ax + by + c = 0 \Leftrightarrow \begin{pmatrix} a \\ b \\ c \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = 0$$

Allgemeine Gleichung einer Geraden in  $P^2$

$$\begin{pmatrix} a \\ b \\ c \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} = 0 \Leftrightarrow ax + by + cz = 0 \quad \begin{pmatrix} a \\ b \\ c \end{pmatrix} \neq \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

Wenn  $\begin{pmatrix} x \\ y \\ z \end{pmatrix}$  die Gleichung erfüllt, dann erfüllt auch  $\begin{pmatrix} \lambda x \\ \lambda y \\ \lambda z \end{pmatrix}$  die Gleichung.  $\begin{pmatrix} a \\ b \\ c \end{pmatrix}$  und  $\begin{pmatrix} \lambda a \\ \lambda b \\ \lambda c \end{pmatrix}$  stellen dieselbe Gerade dar.

**projektive Punkte**  $\begin{pmatrix} x \\ y \\ z \end{pmatrix} \neq \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$  Skalierung egal.

**projektive Gerade**  $\begin{pmatrix} a \\ b \\ c \end{pmatrix} \neq \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$  Skalierung egal

**Satz** Punkt  $\begin{pmatrix} x \\ y \\ z \end{pmatrix}$  liegt auf der Geraden  $\begin{pmatrix} a \\ b \\ c \end{pmatrix}$ :

$$\begin{pmatrix} a \\ b \\ c \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} = 0$$

**Satz** Zwei verschiedene Geraden schneiden sich in genau einem Punkt.

**Beweis** Gerade  $\forall \lambda : \begin{pmatrix} a_1 \\ b_1 \\ c_1 \end{pmatrix}, \begin{pmatrix} a_2 \\ b_2 \\ c_2 \end{pmatrix} \neq \lambda \begin{pmatrix} a_1 \\ b_1 \\ c_1 \end{pmatrix}$ . Schnittpunkt:

$$a_1 x + b_1 y + c_1 z = 0$$

$$a_2 x + b_2 y + c_2 z = 0$$

Koeffizientenmatrix  $A = \begin{pmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \end{pmatrix}$ ,  $\text{rg } A = 2$

$\Rightarrow$  Lösungsmenge ist eindimensional

$$L = \left\{ \lambda \begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix} \mid \lambda \in \mathbb{R} \right\} \text{ ist ein projektiver Punkt}$$

$$\begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix} \text{ kann als } \begin{pmatrix} a_1 \\ b_1 \\ c_1 \end{pmatrix} \times \begin{pmatrix} a_2 \\ b_2 \\ c_2 \end{pmatrix} = \begin{pmatrix} \begin{vmatrix} b_1 & b_2 \\ c_1 & c_2 \end{vmatrix} \\ \begin{vmatrix} c_1 & c_2 \\ a_1 & a_2 \end{vmatrix} \\ \begin{vmatrix} a_1 & a_2 \\ b_1 & b_2 \end{vmatrix} \end{pmatrix} \text{ berechnet werden (Kreuzprodukt)}$$



**Satz** Durch zwei verschiedene Punkte gibt es genau eine Geraden

**Beweis** gleich wie oben:  $\begin{pmatrix} a \\ b \\ c \end{pmatrix}$  mit  $\begin{pmatrix} x \\ y \\ z \end{pmatrix}$  vertauschen.

**Dualitätsprinzip** Man kann in einem Satz der projektiven Geometrie der Ebene „Punkte“ und „Geraden“ vertauschen und es bleibt ein gültiger Satz.

## 2.4.2 Modelle der projektiven Ebene

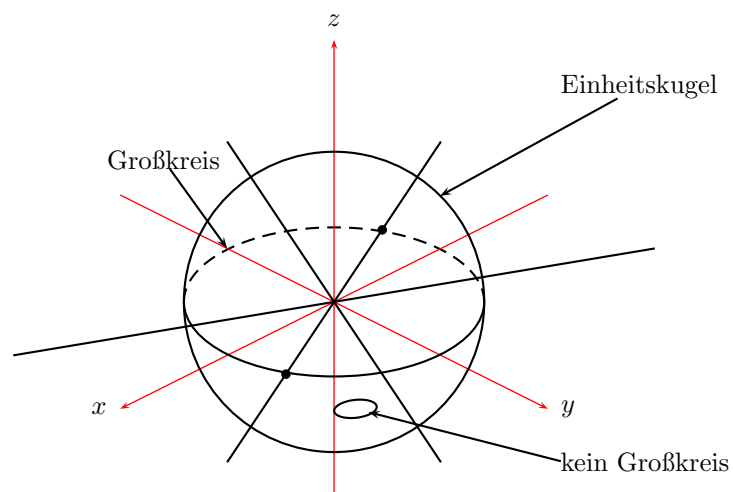
1. *Räumliches Modell der projektiven Ebene*  $\left\{ \lambda \begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix} \mid \lambda \in \mathbb{R} \right\} \dots$  Geraden durch den Ursprung im  $\mathbb{R}^3$  entsprechen den projektiven Punkten.



projektive Gerade  $\equiv$  Ebene durch den Ursprung

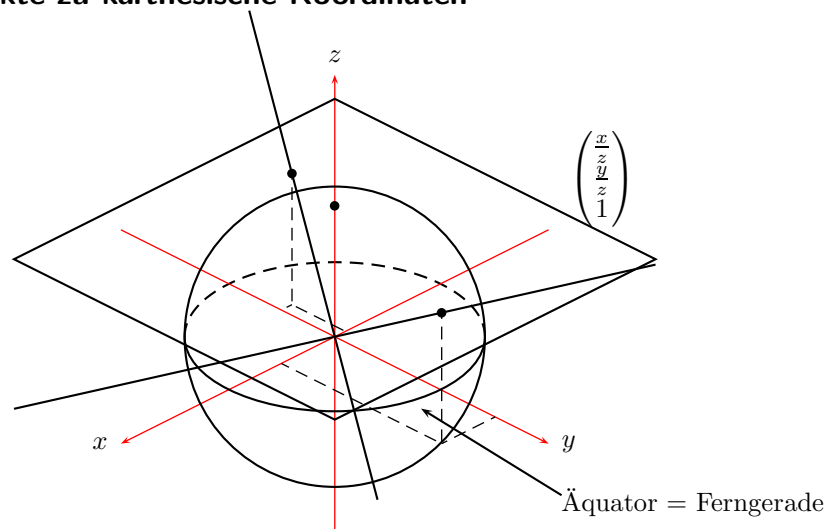
2. *Kugelmodell der projektiven Ebene* entsteht durch Schnitt des räumlichen Modells mit der Einheitskugel

$$S^2 = \left\{ \begin{pmatrix} x \\ y \\ z \end{pmatrix} \mid x^2 + y^2 + z^2 = 1 \right\}$$



projektiver Punkt  $\equiv$  Paar gegenüberliegender Punkte auf der Einheitskugel  
 projektive Gerade  $\equiv$  Großkreise

## 2.4.3 Projektive Punkte zu kartesischen Koordinaten



Schnitt der Geraden  $\begin{pmatrix} x \\ y \\ z \end{pmatrix} \cdot \lambda$  im  $\mathbb{R}^3$  mit Ebene  $z = 1$ :  $z \cdot \lambda = 1 \Rightarrow \lambda = \frac{1}{z}$

$$\rightarrow \begin{pmatrix} x \cdot \frac{1}{z} \\ y \cdot \frac{1}{z} \\ 1 \end{pmatrix}$$

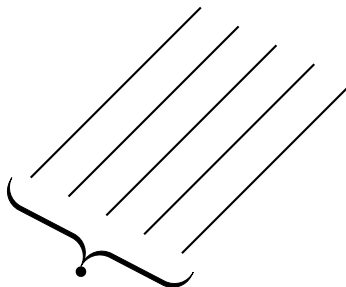
**Satz** Die Punkte  $\begin{pmatrix} x \\ y \\ z \end{pmatrix}$  mit  $z = 0$  haben *keine* Entsprechung in der euklidischen Ebene: Jede projektive Gerade hat als Bild in der euklidischen Ebene eine Gerade, mit einer Ausnahme: die Gerade  $\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$

**Definition** Die Punkte des projektiven Raumes, die keine euklidische Entsprechung haben, heißen **Fernpunkte**. Die Gerade  $\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$  **Ferngerade**.

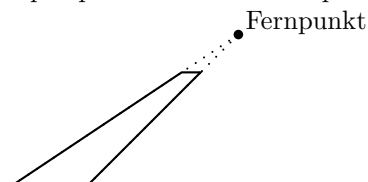
**Satz** Zwei Geraden der euklidischen Ebene sind genau dann *parallel*, wenn ihr Schnittpunkt ein Fernpunkt ist.

**Satz** Die Punkte, die auf der Ferngeraden liegen, sind genau die Fernpunkte

**Satz** Es gibt zu jeder Schaar paralleler Geraden genau einen Fernpunkt.



Anschaulich ist ein Fernpunkt äquivalent zu perspektivischen Sammelpunkten:



## 2.5 Allgemeine projektive Transformationen

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \mapsto M \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} \text{ mit } M = \mathbb{R}^{3 \times 3}, \det M \neq 0$$

(Punkte bleiben Punkte, Geraden bleiben Geraden, Inzidenz bleibt erhalten)

**Definition** Affine Transformationen sind jene Transformationen, bei denen die Fernpunkte Fernpunkte bleiben.

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \mapsto M \begin{pmatrix} x \\ y \\ 0 \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} \stackrel{!}{=} \begin{pmatrix} x' \\ y' \\ 0 \end{pmatrix}$$

$$M = \begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{pmatrix}$$

$$\forall x, y : m_{31}x + m_{32}y + m_{33} \cdot 0 = 0 \Rightarrow m_{31} = m_{32} = 0$$

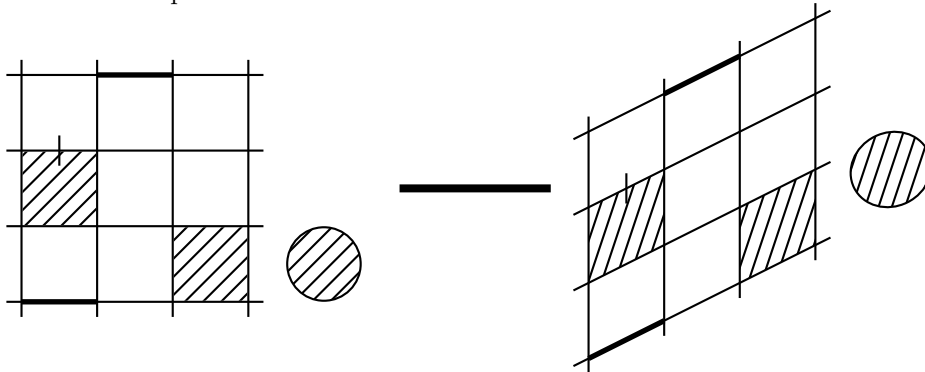
$$\det M \neq 0$$

$$\det M = \underbrace{m_{33}}_{\neq 0} \cdot \underbrace{\begin{vmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{vmatrix}}_0 \Rightarrow \text{o. B. d. A. } m_{33} = 1$$

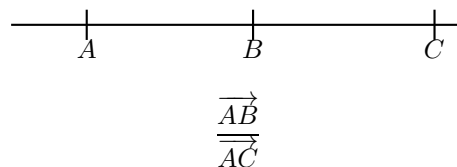
$$\begin{pmatrix} x \\ y \end{pmatrix} \mapsto \overbrace{\begin{pmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}}^{\text{lineare Transformation}} + \overbrace{\begin{pmatrix} m_{13} \\ m_{23} \end{pmatrix}}^{+ \text{ Translation}}$$

Affine Transformation:

- parallele Geraden bleiben parallel



- erhalten das Teilverhältnis auf parallelen Geraden



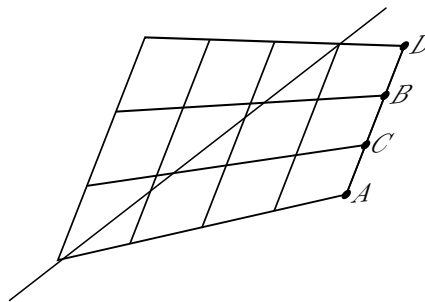
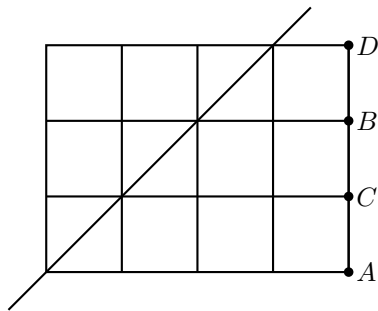
Starre Bewegungen (Isometrien, euklidische Transformationen):

$$M = \begin{pmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{pmatrix} \text{ ist orthogonal } M^T = M^{-1} \text{ erhalten Längen, Winkel und Flächen}$$

**Doppelverhältnis**

$$\boxed{\frac{\overrightarrow{AC}}{\overrightarrow{BC}} : \frac{\overrightarrow{AD}}{\overrightarrow{BD}}} = \text{Doppelverhältnis}$$

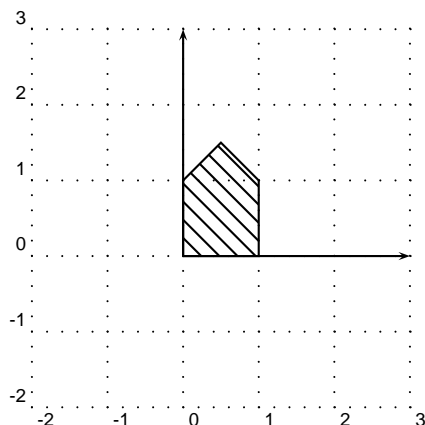
**Bemerkung** projektive Transformationen erhalten das sogenannte Doppelverhältnis



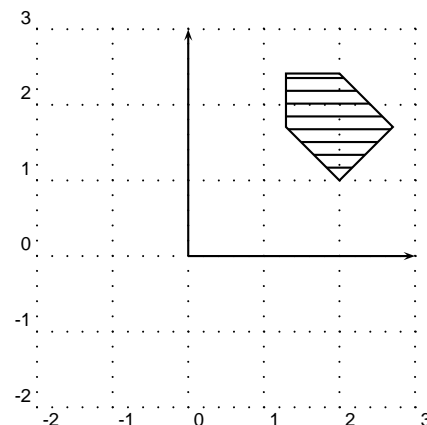
**Ausblick** projektiver Raum; wird beschrieben durch homogene Koordinaten  $\begin{pmatrix} x \\ y \\ z \\ u \end{pmatrix} \neq \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$ . Kartesische Koordinaten  $\begin{pmatrix} x \\ y \\ z \end{pmatrix}$  entsprechen homogenen Koordinaten  $\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$  oder  $\begin{pmatrix} \lambda x \\ \lambda y \\ \lambda z \\ \lambda \end{pmatrix}$  ( $\lambda \neq 0$ , bel.).

**Bemerkung 1** Transformation  $x \mapsto Mx$  kann man auf zwei Arten interpretieren:

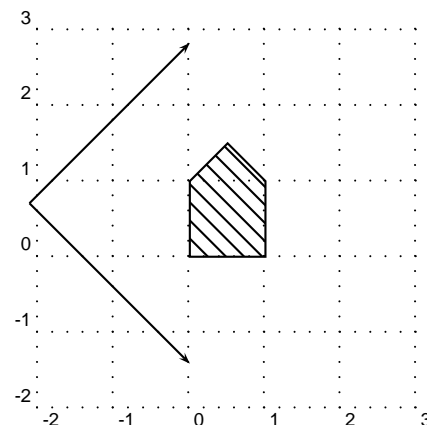
- Wende die transformation  $M$  auf Objekte an. Objekte werden bewegt, Standpunkt/Koordinatensystem bleibt fest.
- Drücke die unveränderte Lage eines Objektes in einem neuen Koordinatensystem aus.



$x \mapsto Mx$



$x \mapsto Mx$



Rechnerisch macht dies keinen Unterschied.

**Bemerkung 2** geometrische Transformationen können verknüpft; Reihenfolge ist wichtig!

$$y = M_1 x$$

$$z = M_2 y$$

$$z = \underbrace{M_2 M_1}_{\text{Matrizenmultiplikation}} x$$

Inverse Transformation wird durch die inverse Matrix ausgedrückt:

$$x = M_1^{-1} y$$

**Bemerkung 3** Bei uns stehen Koordinaten in *Spaltenvektoren*  $\begin{pmatrix} x \\ y \\ z \end{pmatrix}$   $\begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$

$\Rightarrow$  Transformation  $\equiv$  Multiplikation mit einer Matrix *von links*

$$Mx = y$$

Alternative: Zeilenvektoren

$\Rightarrow$  Transformation  $\equiv$  Multiplikation mit einer *von rechts* mit der *transponierten Matrix*

$$y^t = x^t M^t = (Mx)^t$$

Diese Schreibweise ist an sich intuitiver (da die Rechnung in der Reihenfolge der Anwendung aufgeschrieben wird), aber mathematisch unüblich:

$$M_2 M_1 x = z \iff x^t M_1^t M_2^t = z^t$$

## 2.6 Transformation im dreidimensionalen Raum

### 2.6.1 Affine Transformation im dreidimensionalen Raum

- allgemeine affine Transformationen:

$$\begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix} \text{ mit } \begin{vmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{vmatrix} \neq 0$$

- Isometrien (starre Bewegungen):

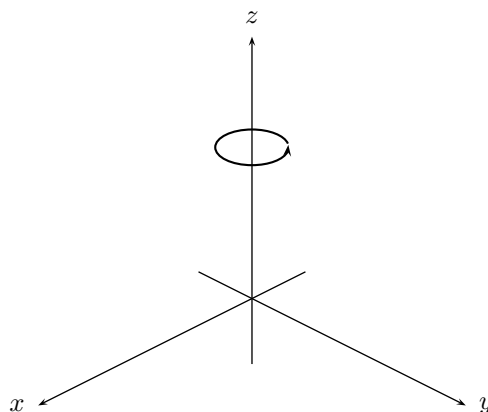
$$\begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{pmatrix} \text{ ist eine orthogonale Matrix}$$

- orientierungserhaltende  $\det M = +1$  [Rotation um eine Achse (+ Translation)]
- orientierungsumkehrende  $\det M = -1$  [Spiegelung an einer Ebene, Spiegelung an einem Punkt, Drehspiegelung ...]

#### Beispiele

- Drehung um die  $z$ -Achse:

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



- Spiegelung an der  $xy$ -Ebene:

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

- Spiegelung am Nullpunkt:

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

## 2.6.2 projektive Transformationen im dreidimensionalen Raum

$$x \mapsto Mx, \quad M \in \mathbb{R}^{4 \times 4}, \det M \neq 0$$

## 2.7 Projektionen und Perspektive

- Sehen mit dem menschlichen Auge



- Lochkamera



- Projektionen



**Projektionen** Verbinde gegebene Punkte mit einem festen *Projektionszentrum* (kann auch ein Fernpunkt sein) und schneide die Strahlen mit einer Ebene (= *Projektionsebene*)

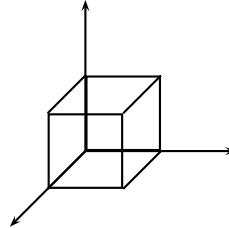
1. Projektionszentrum im Endlichen: Zentralprojektion
2. Projektionszentrum ein Fernpunkt: Parallelprojektion (Parallele Geraden bleiben parallel)



- a) Wenn die Projektion senkrecht auf den Projektionsstrahlen steht, spricht man von *orthographischer Projektion*



- b) andernfalls von *schiefer Projektion*



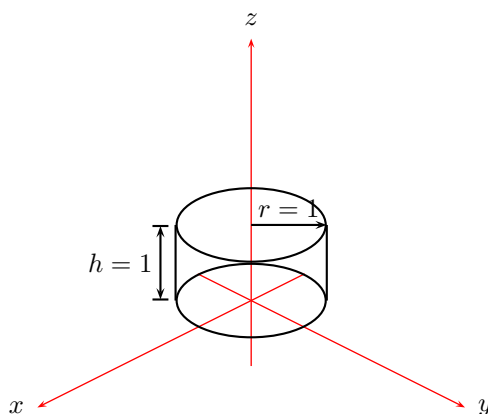
**zu 1. Zentralprojektion** Der *Hauptpunkt* ist der Punkt der Projektionsebene, der dem Auge am nächsten liegt. Ein projiziertes Bild vermittelt den exakten wirklichkeitsgetreuen Eindruck genau dann, wenn man sich so davor stellt, dass das Auge direkt vor dem Hauptpunkt  $H$  liegt und den richtigen Abstand  $d$  und im richtigen Abstand zum Bild, mit dem das Bild berechnet wurde

- Parallele Geraden können in der Projektion zu schneidenden Geraden werden
- Das Bild des entsprechenden Fernpunktes heißt *Fluchtpunkt* (vanishing point)
- Die Fluchtpunkte der horizontalen Gerade liegen auf dem *Horizont* (die Fluchtgerade durch die alle horizontalen Ebenen gehen).
- Wenn die Projektionsgerade senkrecht ist, dann liegt der Hauptpunkt auf dem Horizont  
 $\Rightarrow$  Senkrechte Geraden bleiben dann parallel (und senkrecht)



## 2.8 Koordinaten in der Praxis

- Objektkoordinaten

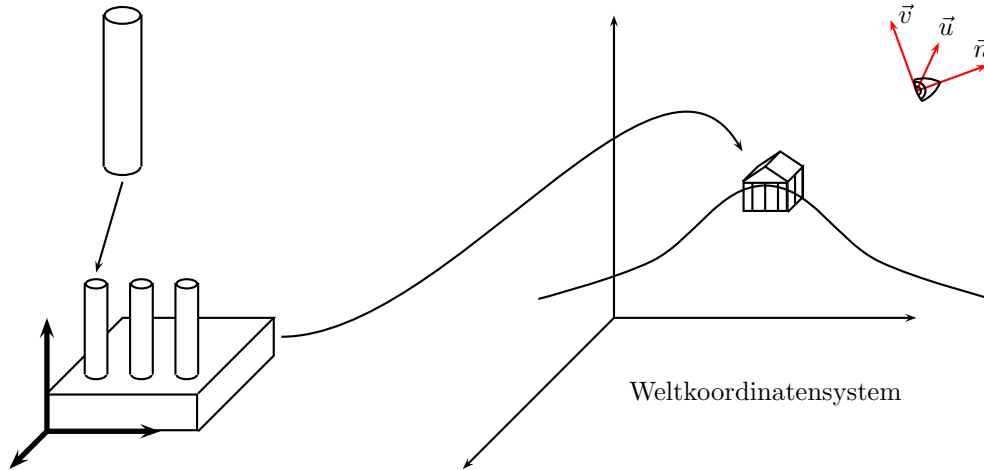


„Standardzylinder“  $x^2 + y^2 \leq 1, 0 \leq z \leq 1$

Durch die affine Transformationen wird die Form des Zylinders angepasst und der Zylinder an die passende Stelle (in einem größeren Modell / in der Umgebung) gesetzt

$$M = \begin{pmatrix} r & & & \\ & r & & \\ & & h & \\ & & & 1 \end{pmatrix} \dots \text{Skalierung} \rightarrow \text{Radius } r \text{ Höhe } h$$

*Translation* (+Rotation) von mehreren Kopien. Zylinder wird Teil eines größeren Objektes mit einem eigenen Koordinatensysteme



- **Weltkoordinaten**

Ein globales Koordinatensystem, das für alle Berechnungen als Referenz dient.

- **Augenkoordinaten** (Kamerakordinaten)

- Ursprung = Augpunkt

- 3 orthogonale Achsen:

- $\vec{n}$  = „Blickrichtung“ vom Objekt zum Betrachter

- $\vec{u}$  = „Horizontale Richtung“ von links nach rechts

- $\vec{v}$  = „Senkrechte Richtung“ von unten nach oben

Die Projektionsebene ist orthogonal zu  $\vec{n}$ . Auf der Projektionsebene wird ein rechteckiges Bild erzeugt, dessen Kanten an  $\vec{u}$  und  $\vec{v}$  ausgerichtet sind.

$$\begin{pmatrix} x_{\text{Welt}} \\ y_{\text{Welt}} \\ z_{\text{Welt}} \\ w_{\text{Welt}} \end{pmatrix} \mapsto M_{AW} \begin{pmatrix} x_{\text{Welt}} \\ y_{\text{Welt}} \\ z_{\text{Welt}} \\ w_{\text{Welt}} \end{pmatrix} = \begin{pmatrix} x_{\text{Auge}} \\ y_{\text{Auge}} \\ z_{\text{Auge}} \\ w_{\text{Auge}} \end{pmatrix}$$

Weltkoordinaten  $x, y, z$  bilden ein Rechtssystem. Augenkoordinaten  $u, v, n$  bilden ein Rechtssystem.

$\Rightarrow M_{AW}$  ist Rotation+Translation

$$M_{AW}^{-1} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} x_{\text{Auge}} \\ y_{\text{Auge}} \\ z_{\text{Auge}} \\ 1 \end{pmatrix}$$

$$M_{AW}^{-1} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} x_u \\ y_u \\ z_u \\ 0 \end{pmatrix} = \text{Vektor } \vec{u} \text{ in Weltkoordinaten}$$

$$M_{AW}^{-1} \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} x_v \\ y_v \\ z_v \\ 0 \end{pmatrix} = \text{Vektor } \vec{v} \text{ in Weltkoordinaten}$$

$$M_{AW}^{-1} \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} x_n \\ y_n \\ z_n \\ 0 \end{pmatrix} = \text{Vektor } \vec{n} \text{ in Weltkoordinaten}$$

$$\begin{aligned}
 & \text{orthogonal} \\
 M_{AW}^{-1} &= \begin{pmatrix} \boxed{\begin{matrix} x_u & x_v & x_n \\ y_u & y_v & y_n \\ z_u & z_v & z_n \end{matrix}} & \begin{matrix} x_{\text{Auge}} \\ z_{\text{Auge}} \\ y_{\text{Auge}} \end{matrix} \\ & \quad \begin{matrix} 0 & 0 & 0 \end{matrix} & 1 \end{pmatrix} \\
 M_{AW} &= \begin{pmatrix} x_u & y_u & z_u & * \\ x_v & y_v & z_v & * \\ x_n & y_n & z_n & * \\ 0 & 0 & 0 & 1 \end{pmatrix} \\
 M_{AW} \begin{pmatrix} x_{\text{Auge}} \\ y_{\text{Auge}} \\ z_{\text{Auge}} \\ 1 \end{pmatrix} &= \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}
 \end{aligned}$$

Der Punkt  $\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$  in Augenkoordinaten wird auf  $A + \vec{n}$  in Weltkoordinaten abgebildet.

$$M_{AW}^{-1} \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} x_{\text{Auge}} + x_n \\ y_{\text{Auge}} + y_n \\ z_{\text{Auge}} + z_n \\ 1 \end{pmatrix} \quad M_{AW}^{-1} \left[ \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} - \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \right] = \begin{pmatrix} x_n \\ y_n \\ z_n \\ 0 \end{pmatrix}$$

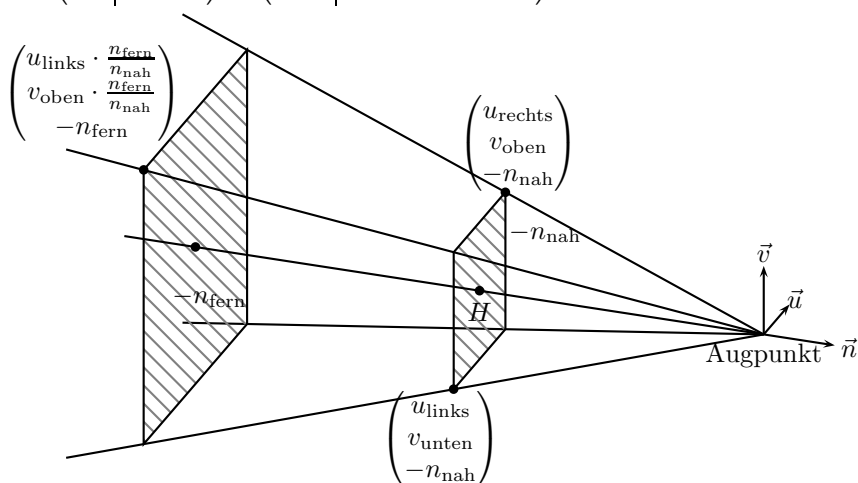
$A = \begin{pmatrix} x_u & x_v & x_n \\ y_u & y_v & y_n \\ z_u & z_v & z_n \end{pmatrix}$  Spalten sind die kartesischen Weltkoordinaten von  $\vec{u}, \vec{v}, \vec{n}$

$$M_{AW}^{-1} = \left( \begin{array}{ccc|c} & & & x_{\text{Auge}} \\ & A & & z_{\text{Auge}} \\ & & & y_{\text{Auge}} \\ 0 & 0 & 0 & 1 \end{array} \right)$$

$$M_{AW} = \left( \begin{array}{ccc|c} & A^T & & -A^T \begin{pmatrix} x_{\text{Auge}} \\ z_{\text{Auge}} \\ y_{\text{Auge}} \end{pmatrix} \\ 0 & 0 & 0 & 1 \end{array} \right)$$

Probe:

$$\left( \begin{array}{c|c} A & \begin{pmatrix} x_{\text{Auge}} \\ z_{\text{Auge}} \\ y_{\text{Auge}} \end{pmatrix} \\ \hline 0 & 1 \end{array} \right) \cdot \left( \begin{array}{c|c} A^T & -A^T \begin{pmatrix} x_{\text{Auge}} \\ z_{\text{Auge}} \\ y_{\text{Auge}} \end{pmatrix} \\ \hline 0 & 1 \end{array} \right) = \left( \begin{array}{c|c} A \cdot A^T = I' & 0 \\ \hline 0 & 1 \end{array} \right) = I \quad \square$$



Projektionsrechteck liegt in der Ebene  $n = n_{\text{nah}}$  und bedeckt dort den Bereich

$$[u_{\text{links}}, u_{\text{rechts}}] \times [v_{\text{unten}}, v_{\text{oben}}]$$

Der Sichtbare Bereich ist alles was hinter diesem Rechteck liegt. Zusätzlich wird alles abgeschnitten, was hinter der Ebene  $n = n_{\text{fern}}$  liegt.

$\Rightarrow$  Pyramidenstumpf (view frustum)

- **Normalisierte Gerätekoordinaten** (normalized device coordinates, NDC) Der sichtbare Pyramidenstumpf wird durch projektive Transformation auf den Würfel  $[-1, +1]^3$  transformiert.  $x, y, z$  bilden ein Linkssystem.



$$\begin{pmatrix} * \\ * \\ -n_{\text{nah}} \\ 1 \end{pmatrix} \mapsto \begin{pmatrix} * \\ * \\ -1 \\ +1 \end{pmatrix} \cdot \lambda \quad (\text{Ebene } z = -1)$$

$$\begin{pmatrix} * \\ * \\ -n_{\text{fern}} \\ 1 \end{pmatrix} \mapsto \begin{pmatrix} * \\ * \\ 1 \\ 1 \end{pmatrix} \cdot \lambda_2$$

Fernpunkt auf der  $z$ -Achse:

$$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \mapsto \begin{pmatrix} 0 \\ 0 \\ * \\ 0 \end{pmatrix}$$

Horizontale Linien (Richtung  $u$ ) bleiben parallel und horizontal (Richtung  $x$ ):

$$\begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \mapsto \begin{pmatrix} * \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Vertikale Linien

$$\begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \mapsto \begin{pmatrix} 0 \\ * \\ 0 \\ 0 \end{pmatrix}$$

Transformation

$$\begin{pmatrix} u \\ v \\ n \\ w \end{pmatrix} \mapsto \begin{pmatrix} * & 0 & * & 0 \\ 0 & * & * & 0 \\ 0 & 0 & * & * \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} u \\ v \\ n \\ w \end{pmatrix} = \begin{pmatrix} * & 0 & * & 0 \\ 0 & * & * & 0 \\ 0 & 0 & a & b \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} u \\ v \\ n \\ w \end{pmatrix}$$

$$\begin{pmatrix} * \\ * \\ -n_{\text{nah}} \\ 1 \end{pmatrix} \mapsto \begin{pmatrix} * \\ * \\ -1 \\ 1 \end{pmatrix} \cdot \lambda = \begin{pmatrix} * \\ * \\ -n_{\text{nah}}a + b \\ n_{\text{nah}} \end{pmatrix} \Rightarrow -n_{\text{nah}}a + b = -n_{\text{nah}}$$

$$\begin{pmatrix} * \\ * \\ -n_{\text{fern}} \\ 1 \end{pmatrix} \mapsto \begin{pmatrix} * \\ * \\ 1 \\ 1 \end{pmatrix} \cdot \lambda = \begin{pmatrix} * \\ * \\ -n_{\text{fern}}a + b \\ n_{\text{fern}} \end{pmatrix} \Rightarrow -n_{\text{fern}}a + b = n_{\text{fern}}$$

$$\begin{array}{rcl}
 -n_{\text{nah}}a + b & = & -n_{\text{nah}} \\
 -n_{\text{fern}}a + b & = & n_{\text{fern}} \\
 \hline
 a(-n_{\text{nah}} + n_{\text{fern}}) & = & -n_{\text{nah}} - n_{\text{fern}} \\
 a & = & -\frac{n_{\text{fern}} + n_{\text{nah}}}{n_{\text{fern}} - n_{\text{nah}}} \\
 \Rightarrow b & = & -\frac{2 \cdot n_{\text{fern}} \cdot n_{\text{nah}}}{n_{\text{fern}} - n_{\text{nah}}}
 \end{array}$$
  

$$\Rightarrow M = \begin{pmatrix} \frac{2n_{\text{nah}}}{u_{\text{rechts}} - u_{\text{links}}} & 0 & \frac{u_{\text{rechts}} + u_{\text{links}}}{u_{\text{rechts}} - u_{\text{links}}} & 0 \\ 0 & \frac{2n_{\text{nah}}}{v_{\text{oben}} - v_{\text{unten}}} & \frac{u_{\text{rechts}} - u_{\text{links}}}{v_{\text{oben}} + v_{\text{unten}}} & 0 \\ 0 & 0 & \frac{v_{\text{oben}} - v_{\text{unten}}}{n_{\text{fern}} + n_{\text{nah}}} & -\frac{2n_{\text{fern}} \cdot n_{\text{nah}}}{n_{\text{fern}} - n_{\text{nah}}} \\ 0 & 0 & \frac{n_{\text{fern}} - n_{\text{nah}}}{-1} & 0 \end{pmatrix}$$
  

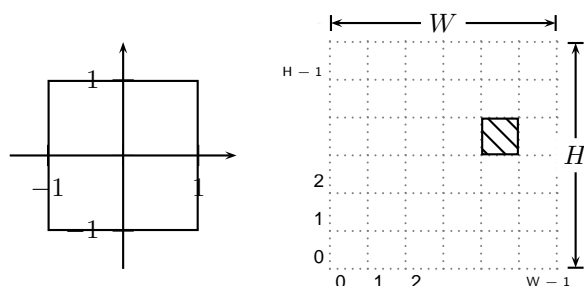
$$M^{-1} = \begin{pmatrix} \frac{u_{\text{rechts}} - u_{\text{links}}}{2n_{\text{nah}}} & 0 & 0 & \frac{u_{\text{rechts}} - u_{\text{links}}}{2n_{\text{nah}}} \\ 0 & \frac{v_{\text{oben}} - v_{\text{unten}}}{2n_{\text{nah}}} & 0 & \frac{v_{\text{oben}} - v_{\text{unten}}}{2n_{\text{nah}}} \\ 0 & 0 & 0 & -1 \\ 0 & 0 & \frac{1}{2n_{\text{fern}}} - \frac{1}{2n_{\text{nah}}} & \frac{1}{2n_{\text{fern}}} + \frac{1}{2n_{\text{nah}}} \end{pmatrix}$$

- **Rasterkoordinaten** - Koordinaten auf dem Bildschirm  
von normalisierten Gerätekoordinaten (NDC) zu Rasterkoordinaten:

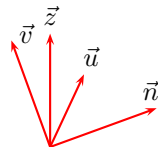
**zur Erinnerung** NDC Linkssystem



1. Projektion:  $z$ -Koordinaten weglassen. ( $z$  gibt Informationen über die Tiefe, größerer  $z$ -Wert ist weiter hinten)
2. Skalierung des  $x$ - $y$ -Quadrates und Runden auf  $W \times H$ -Gitter



Pixelkoordinaten:  $\begin{pmatrix} \lfloor (x+1) \cdot \frac{W}{2} \rfloor \\ \lfloor (y+1) \cdot \frac{H}{2} \rfloor \end{pmatrix}$



Berechnung des Augkoordinatensystems  
Gegeben ist der Einheitsvektor  $\vec{n}$  (und der Augpunkt)

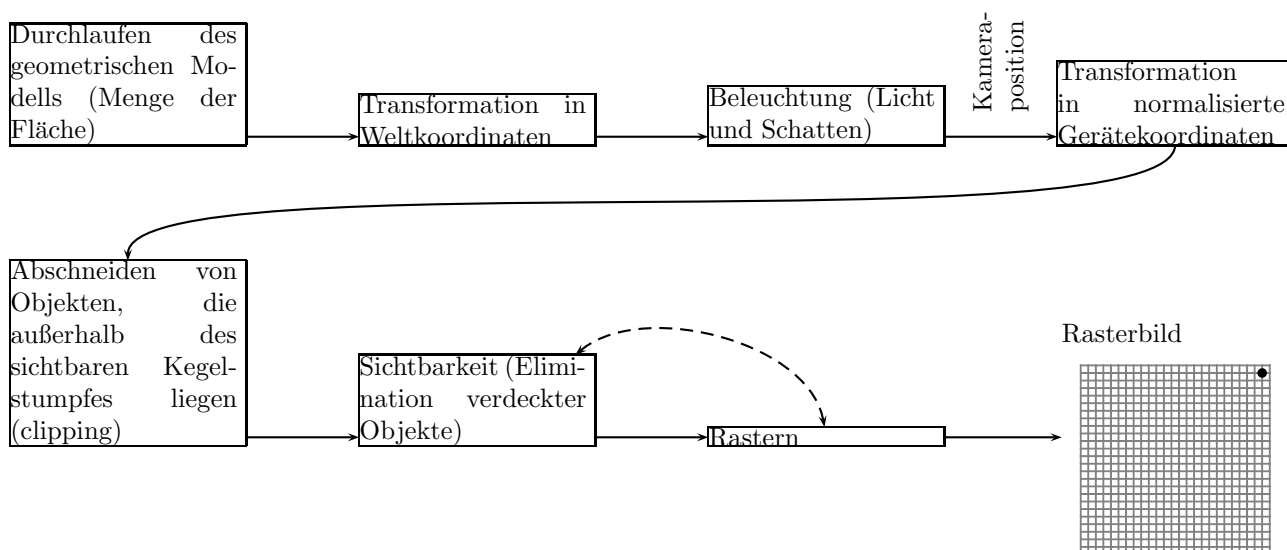
Setze  $\vec{u}_0 := \vec{z} \times \vec{n}$

$\vec{z} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$  senkrecht nach oben

$$\vec{u} := \frac{\vec{u}_0}{\|\vec{u}_0\|}$$

$$\vec{v} := \vec{n} \times \vec{u}$$

## 2.9 „rendering pipeline“ – vom geometrischen Modell zum Rasterbild



nur *eine* mögliche Organisation; andere Reihenfolgen sind möglich

# 3 Licht und Farben

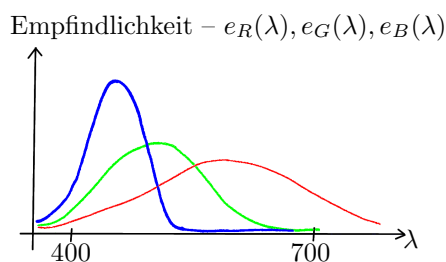
**Definition** (sichtbares) **Licht** sind elektromagnetische Wellen verschiedener Wellenlänge (ca. zwischen 400–700 nm) Die Wellenlänge  $\lambda$  entscheidet über die **Farbe**. Das meiste Licht ist eine Mischung von verschiedenen Wellenlängen.



Wenn Licht auf einen Gegenstand trifft, dann wird es in unterschiedlichem Maß zurückgeworfen, je nach Wellenlänge Wenn Licht einen filter durchdringt, ist es analog (subtraktive Farbmischung).

## 3.1 Farbsehen im menschlichen Auge

Es gibt drei Arten von lichtempfindlichen *Zapfen* (R, G, B)



Erregung der „roten“ Zapfen bei einer Lichtquelle mit Intensitätsfunktion  $f(\lambda)$

$$r = \int f(\lambda) \cdot e_R(\lambda) \, d\lambda$$

analog „grün“:  $g = \int f(\lambda) \cdot e_G(\lambda) \, d\lambda$

analog „blau“:  $b = \int f(\lambda) \cdot e_B(\lambda) \, d\lambda$

- Verschiedene Lichtquellen mit verschiedenen spektraler Zusammensetzung erzeugen den gleichen Farbeindruck, wenn sie die gleichen  $(r, g, b)$ -Werte hervorbringen.
- Dreidimensionaler Farbraum, aber nicht alle  $(r, g, b)$ -Werte erreichbar (Wenn  $g > 0 \Rightarrow r > 0$  oder  $b > 0$ ,  $(r, g, b) = (0, 1, 0)$  gibt es nicht)
- Wenn man  $f(\lambda)$  mit einem Skalar  $c > 0$  multipliziert, dann ändert sich nur die Helligkeit, nicht die Farbe. Entsprechend wird  $(r, g, b)$  mit einem Skalar multipliziert.
- Normalisierung auf  $r + b + g = 1$  führt auf einen zweidimensionalen Farbraum, bei dem die Helligkeit konstant ist.

In der Computergrafik tut man so, als ob es nur *drei* verschiedene Wellenlängen (Grundfarben) gibt: **R**ot, **G**rün und **B**lau







# 4 Rasterung von Strecken und Kreisen

## 4.1 Strecken

- Rendering pipeline wurde durchlaufen
- Koordinaten sind ganzzahlige Pixelkoordinaten



**Problemstellung** Gegeben sind zwei Punkte  $p_1 = (x_1, y_1)$  und  $p_2 = (x_2, y_2)$ . Zeichne Strecke zwischen  $p_1$  und  $p_2$ .  $x_1, y_1, x_2, y_2$  sind ganzzahlig.

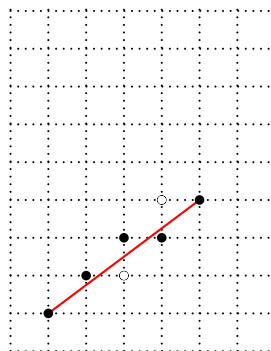
**Vereinfachung** Strecken die durch den Nullpunkt gehen  $(0,0), p_1 = (x_1, y_1)$

- Beschränkung auf Strecken im ersten Quadranten des Koordinatensystems. Alle anderen Strecken (im Quadranten II, III, IV) können durch Spiegelung an  $x$ - oder  $y$ -Achse erzeugt werden
- Beschränkung aufersten Oktanten, alle anderen Strecken erhält man wie oben durch Vertauschen von  $x$ - und  $y$ -Koordinate

**Vereinfachte Problemstellung** Gegeben ist ein Punkt  $p_1 = (x_1, y_1)$  mit ganzzahligen Koordinaten.  $x_1 \geq y_1, x_1 \geq 0, y_1 \geq 0$ . Zeichne Strecke zwischen  $(0,0)$  und  $(x, y)$ .

$$\text{Geradengleichung: } g(x) = \frac{y_1}{x_1}x$$

$$\text{Steigung: } m = \frac{y_1}{x_1}, 0 \leq m \leq 1$$



**Idee** Werte für jeden Wert  $i$  zwischen 0 und  $x_1$  die Funktion  $g$  aus. Runde das Ergebnis  $g(i)$  und nimm diesen Wert als  $j$ -Wert

```
double m = y1/x1;
for (i = 0; i <= x1, i++) {
    j = round(i*m); // auf- bzw. abrunden
    setPixel(i,j);
}
```

- ersetze `j = round(i * m);` durch `y=y+m*j; j = round(y);`
- im  $i$ -ten Schritt  $y_i = m \cdot i$

- im  $(i + 1)$ -ten Schritt  $y_{i+1} = m \cdot (i + 1)$
- $y_{i+1} - y_i = m(i + 1 - 1) = m$
- $0 \leq m \leq 1 \Rightarrow y_{i+1} \leq y_i + 1$
- Wert von  $j$  steigt pro Schleifendurchlauf um höchstens 1.

$$y_{i+1} = y_i + m$$

$$j = \text{round}(y_i)$$

$$y_i = j + r_i$$

$$y_{i+1} = y_i + m = j + \underbrace{r_i + m}_{r_{i+1}}$$

$$m = \frac{y_1}{x_1}$$

$$\text{Rest: } -\frac{1}{2} \leq r_i \leq \frac{1}{2}$$

```
double m = y1/x1;
double r = 0;
int j = 0;
for (i = 0; i <= x1; i++) {
    r = r + m; // y = y + m;
    if (r >= 1/2) { // j = round(y)
        j = j+1,
        r = r - 1;
    }
    setPixel(i, j);
}
```

z. B:

$$y = 0.6$$

$$y = j_{alt} + r_{alt}$$

$$y = j_{neu} + r_{neu}$$

$$r_{alt} = 0.6$$

$$j_{neu} = j_{alt} + 1$$

$$r_{neu} = r_{alt} + 1$$

Immer noch ein Problem: **double**-Werte sind zu ungenau

Wir wissen, dass  $x_1$  und  $y_1$  ganzzahlig sind.

$\Rightarrow$  Multiplikation mit  $2x_1$  liefert **int**-Werte

```
int m = 2*y1;
int r = 0;
int j = 0;
setPixel(0,0);
for(int i = 1; i <= x1; i++) {
    r = r + m;
    if (r >= x1) {
        j = j+1;
        r = r - 2*x1;
    }
    setPixel(i, j);
}
```

$$r_{i+1} + 2y_1 \geq x_1 \Leftrightarrow r \geq x_1 - 2y_1$$

```
int j = 0;
int m = 2*y1;
int r = 2*y1 - x1;
int c1 = m - 2*x1;
int c2 = m;
for (int i = 1; i <= x1; i++) {
    if (r >= 0) {
        j++;
        r = r + c1;
    } else {
        r = r + c2;
    }
    setPixel(i, j);
}
```

### 4.1.1 Bresenham-Algorithmus



Wähle eine Spalte  $i + 1$  das Pixel, das am nächsten an  $q_{i+1}$  liegt, also

$$\begin{aligned} d'_{i+1} \leq d''_{i+1} &\Leftrightarrow \text{wähle } j_{i+1} = j_i \\ d''_{i+1} < d'_{i+1} &\Leftrightarrow \text{wähle } j_{i+1} = j_i + 1 \end{aligned}$$

oder äquivalent

$$\begin{aligned} d'_{i+1} - d''_{i+1} \leq 0 &\Leftrightarrow j_{i+1} = j_i \\ d'_{i+1} - d''_{i+1} > 0 &\Leftrightarrow j_{i+1} = j_i + 1 \end{aligned}$$

$y$ -Koordinaten von  $q_{i+1}$   $\frac{y_1}{x_1}(i + 1)$

$$\begin{aligned} d'_{i+1} &= \frac{y_1}{x_1}(i + 1) - j_i & d''_{i+1} &= j_i + 1 - \frac{y_1}{x_1}(i + 1) \\ d'_{i+1} - d''_{i+1} &= 2\frac{y_1}{x_1}(i + 1) - 2j_i - 1 & & \text{ | multipliziere mit } x_1 \end{aligned}$$

(ändert nichts an der Bedingung  $\leq 0$ )

Wir erhalten eine Fehler-/Entscheidungsvariable für  $(i + 1)$ -te Spalte

$$\begin{aligned} e_{i+1} &= x_i(d'_{i+1} - d''_{i+1}) = 2y_1(i + 1) - 2j_i x_1 - x_1 \\ e_{i+1} \leq 0 &\Leftrightarrow j_{i+1} = j_i \\ e_{i+1} > 0 &\Leftrightarrow j_{i+1} = j_i + 1 \end{aligned}$$

Betrachte Differenz zwischen aufeinanderfolgenden Entscheidungsvariablen

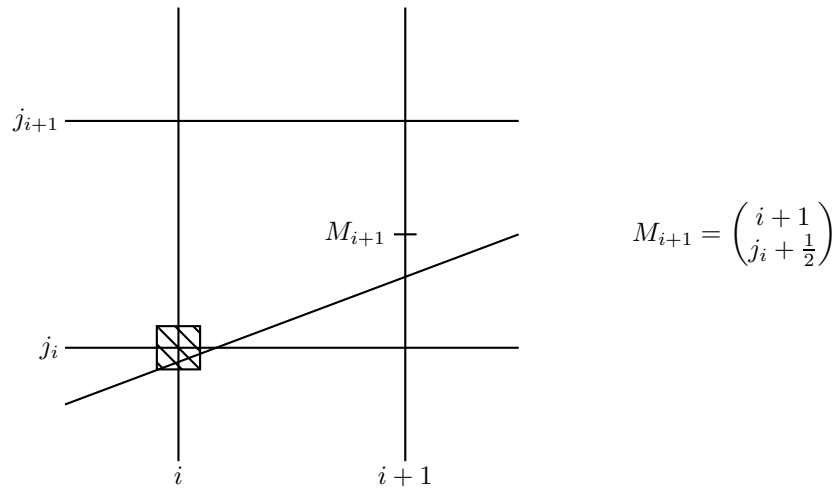
$$\begin{aligned} e_{i+1} - e_i &= \underline{2y_1(i + 1)} - 2j_i x_1 - \underline{x_1} - \underline{2y_1 i} + 2j_{i-1} x_1 + \underline{x_1} \\ e_{i+1} - e_i &= 2y_1 - 2x_1 \underbrace{(j_i - j_{i-1})} \\ &= \begin{cases} 0, & \text{falls } e_i \leq 0 \\ 1, & \text{falls } e_i > 0 \end{cases} \end{aligned}$$

Also

$$\begin{aligned} e_i > 0, j_i &= j_{i-1} + 1 \text{ und } e_{i+1} = e_i + 2y_1 - 2x_1 \\ e_i \leq 0, j_i &= j_{i-1} \text{ und } e_{i+1} = e_i + 2y_1 \end{aligned}$$

Anfangswerte:  $i = 0, j_0 = 0, e_1 = 2y_1 - x_1$

## 4.1.2 Midpoint Line Algorithmus



Wenn  $M_{i+1}$  über der Strecke liegt  $\Rightarrow$  wähle  $j_{i+1} = j_i$  Wenn  $M_{i+1}$  unter der Strecke liegt  $\Rightarrow$  wähle  $j_{i+1} = j_i + 1$

$$\text{Geradengleichung: } y = \frac{y_1}{x_1}x \Leftrightarrow y_1x - x_1y = 0$$

$$(x, y) \text{ liegt über der Geraden} \Leftrightarrow y > \frac{y_1}{x_1}x \Leftrightarrow y_1x - x_1y < 0$$

$$(x, y) \text{ liegt unter der Geraden} \Leftrightarrow y < \frac{y_1}{x_1}x \Leftrightarrow y_1x - x_1y > 0$$

Setze  $F(x, y) = xy_1 - x_1y$ , dann gilt also:

- $(x, y)$  über Geraden  $\Leftrightarrow F(x, y) < 0$
- $(x, y)$  auf Geraden  $\Leftrightarrow F(x, y) = 0$
- $(x, y)$  unter Geraden  $\Leftrightarrow F(x, y) > 0$

Entscheidungsvariable

$$d_{i+1} = F(M_{i+1}) = (i+1)y_1 - x_i \left( j_i + \frac{1}{2} \right) \leq 0 \Leftrightarrow j_{i+1} = j_i$$

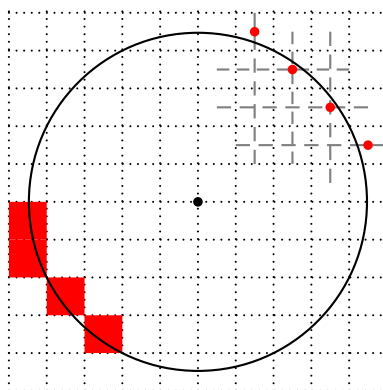
$$d_{i+1} = F(M_{i+1}) = (i+1)y_1 - x_i \left( j_i + \frac{1}{2} \right) > 0 \Leftrightarrow j_{i+1} = j_i + 1$$

$$\begin{aligned} d_{i+1} - d_i &= y_1(i+1) - x_1 \left( j_i + \frac{1}{2} \right) - y_1i + x_1 \left( j_{i+1} + \frac{1}{2} \right) \\ &= y_1 - x_1 \underbrace{(j_i - j_{i-1})} \\ &= \begin{cases} 0, & \text{falls } d_i \leq 0 \\ 1, & \text{falls } d_i > 0 \end{cases} \\ d_1 &= y_1 - \frac{x_1}{2} \end{aligned}$$

Für Ganzzahligkeit multipliziere mit 2

$$\begin{aligned} e_i &= 2d_i \Rightarrow e_{i+1} - e_i = 2y_1 - 2x_1(j_i - j_{i-1}) \\ e_1 &= 2y_1 - x_1 \end{aligned}$$

## 4.2 Kreise



### Annahme

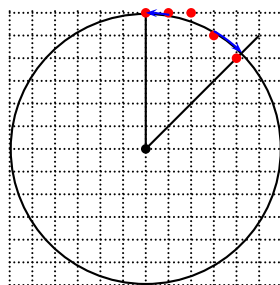
- Radius  $r$  ist ganzzahlig
- Mittelpunkt  $(c_x, c_y)$  ist ein Gitterpunkt

Kreisgleichung:

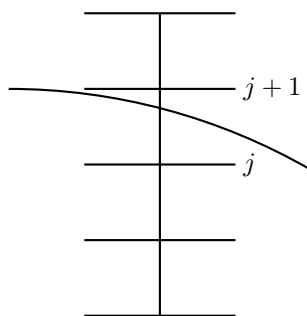
$$(x - c_x)^2 + (y - c_y)^2 = r^2$$

Betrachte den Fall, wo der Mittelpunkt  $(0,0)$  ist (anschließend alles um  $(c_x, c_y)$  verschieben). Wir zeichnen den Bereich  $x \geq 0, y \geq 0, y \geq x$  auf diesem Achtelkreis zeichnen wir auf jeder senkrechten Gittergeraden *einen* Punkt.

Ausnutzung der Symmetrie:



$x = i$  ist fest, Kreis verläuft zwischen  $(i, j)$  und  $(i, j + 1)$ .



Welchen dieser beiden Punkte soll man auswählen?

- (1). Wähle den Punkt, der kleineren *Abstand* vom Kreis hat
- (2). Berechne den Schnittpunkt mit der Geraden  $x = i$  und  $r$  und runde zum nächsten Gitterpunkt.
  - (Äquivalent: vergleiche den *senkrechten Abstand* zum Kreis)
  - (Äquivalent: Liegt  $(i, j + \frac{1}{2})$  über oder unter dem Kreisbogen?)
- (3). Wähle den Punkt der die *Kreisgleichung*  $x^2 + y^2 = r^2$  am besten erfüllt.

$$|x^2 + y^2 - r^2| \rightarrow \text{MIN}$$

Für Geraden sind alle drei Bedingungen äquivalent

Für (3). gibt es beliebig viele Varianten:

$$\sqrt{x^2 + y^2} = r \Rightarrow |r - \sqrt{x^2 + y^2}| \rightarrow \text{MIN führt auf (1)}$$

$$(x^2 + y^2)^2 = r^4 \Rightarrow |r^4 - (x^2 + y^2)^2| \rightarrow \text{MIN führt auf (1)}$$



**Satz** Wenn der Mittelpunkt  $(c_x, c_y)$  und der Radius  $r$  ganzzahlig sind, dann sind Bedingung (1),(2) und (3) äquivalent.

Algebraische Formulierung von (1), (2), (3):

- Punkt  $(i, j + 1)$  liegt außerhalb

$$i^2 + (j + 1)^2 \geq r^2$$

- Punkt  $(i, j)$  liegt innerhalb

$$i^2 + (j + 1)^2 < r^2$$

$$(1). \quad r - \sqrt{i^2 + j^2} \underset{<}{\overset{\geq}{\cong}} \sqrt{i^2 + (j + 1)^2} - r$$

$$(2). \quad i^2 + \left(j + \frac{1}{2}\right)^2 \underset{<}{\overset{\geq}{\cong}} r^2$$

$$(3). \quad r^2 - (i^2 + j^2) \underset{<}{\overset{\geq}{\cong}} i^2 + (j + 1)^2 - r^2$$

$$(3) \iff \begin{aligned} 2r^2 &\underset{<}{\overset{\geq}{\cong}} i^2 + j^2 + 2j + 1 + i^2 + j^2 \\ &= 2i^2 + 2j^2 + 2j + 1 \end{aligned}$$

$$\iff i^2 + j^2 + j - r^2 + \frac{1}{2} \underset{<}{\overset{\geq}{\cong}} 0$$

$\Rightarrow$  „ $\cong$ “ kommt nicht vor für  $i, j \in \mathbb{Z}$

$$(2) \iff i^2 + j^2 + j + \frac{1}{4} - r^2 \underset{<}{\overset{\geq}{\cong}} 0$$

Für  $i, j, r \in \mathbb{Z}$  sind (2) und (3) äquivalent:

$\overbrace{i^2 + j^2 + j - r^2}^{g(i, j+1), \text{ s. Algorithmus}} \geq 0$	$\Rightarrow$ Zeichne $(i, j)$
$\leq -1$	$\Rightarrow$ Zeichne $(i, j + 1)$

**Behauptung**

$$(a). \quad i^2 + \left(j + \frac{1}{2}\right)^2 \geq r^2 \Rightarrow \sqrt{i^2 + (j + 1)^2} - r > r - \sqrt{i^2 + j^2}$$

$$(b). \quad i^2 + j^2 + j + \frac{1}{2} \leq r^2 \Rightarrow \sqrt{i^2 + (j + 1)^2} - r < r - \sqrt{i^2 + j^2}$$

Daraus folgt, dass Regel (1) mit den beiden anderen Regeln (2), (3) konsistent ist.

**Beweis** (von (b).) Die Funktion  $f(t) = \sqrt{t}$  ist für  $t > 0$  konkav



$$\text{(daraus folgt: } f\left(\frac{u+v}{2}\right) \geq \frac{f(u) + f(v)}{2})$$

Eine differenzierbare Funktion  $f$  ist konkav  $\Leftrightarrow f'$  monoton fallen  $\Leftrightarrow f'' \leq 0$

$$f'(t) = \frac{1}{2\sqrt{t}} \searrow \text{ konkav}$$

$$u = i^2 + j^2$$

$$v = i^2 + (j+1)^2 = i^2 + j^2 + 2j + 1$$

$$\frac{u+v}{2} = i^2 + j^2 + j + \frac{1}{2}$$

$$r \underset{\text{N.V.}}{\geq} \sqrt{i^2 + j^2 + j + \frac{1}{2}} \geq \frac{1}{2} \left[ \sqrt{i^2} + \sqrt{i^2 + (j+1)^2} \right]$$

$$2r \geq \sqrt{i^2 + j^2} + \sqrt{i^2 + (j+1)^2}$$

**Beweis** (von (a).) Funktion  $h(y) = \sqrt{i^2 + y^2}$  ist *konvex*

$$h'(y) = \frac{1 \cdot 2y}{2\sqrt{i^2 + y^2}} = \frac{1}{\sqrt{\frac{i^2}{y^2} + 1}} \nearrow \text{ monoton wachsend}$$

$$g\left(\frac{u+v}{2}\right) \leq \frac{1}{2}(g(u) + g(v)) \quad u = j, v = j+1$$

$$r^2 \underset{\text{N.V.}}{\leq} \sqrt{i^2 + \left(j + \frac{1}{2}\right)^2} \leq \frac{1}{2} \left( \sqrt{i^2 + j^2} + \sqrt{i^2 + (j+1)^2} \right)$$

**Algorithmus** beginnt mit  $(0, r)$  und zeichnet Punkte von links nach rechts.

- letzter gezeichneter Punkt =  $(i-1, j)$
- soll nächster  $(i, j)$  oder  $(i, j-1)$  gezeichnet werden?

$$g(i, j) = i^2 + (j-1)^2 + (j-1) - r^2 = i^2 + j^2 - 2j + 1 + j - r^2$$

$$g(i, j) := i^2 + j^2 - j - r^2 \begin{cases} \geq 0 & \Rightarrow \text{zeichne } (i, j-1) \\ \leq -1 & \Rightarrow \text{zeichne } (i, j) \end{cases}$$

$$g(i+1, j) - g(i, j) = (i+1)^2 - i^2 = 2i + 1$$

$$\begin{aligned} g(i, j-1) - g(i, j) &= (j-1)^2 - (j-1) - j^2 + j \\ &= j^2 - 2j + 1 - j + 1 - j^2 + j = -2j + 2 \end{aligned}$$



```

i := 0; j := r, g := -r; // Invariante: g = g(i,j)

loop
  SetPixel(i,j)
  g := g + 2i + 1; i := i + 1
  if g ≥ 0 then j := j - 1; g := g - 2j
until j < i

SetPixel(cx+i,cy+j)
SetPixel(cx+j,cy+i)
SetPixel(cx-i,cy+j)
SetPixel(cx-j,cy+i)
SetPixel(cx-i,cy-j)
SetPixel(cx-j,cy-i)
SetPixel(cx+i,cy-j)
SetPixel(cx+j,cy-i)

```

Die Pixel in den 4 Himmelsrichtungen werden doppelt gezeichnet.

### 4.3 Schwachstellen der Rasterung (Aliasing)

- merkbare Sprünge bei fast achsenparallelen Geraden

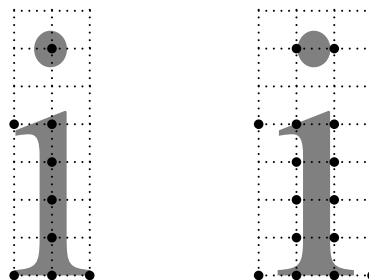


- unterschiedliche Helligkeitsverteilung in Abhängigkeit von der Steigung



- horizontale/vertikale Linie hat 1 Pixel pro Längeneinheit
  - schräge Linie ( $45^\circ$ ) hat 1 Pixel pro  $\sqrt{2}$  Längeneinheiten
- ⇒ schräge Linien erscheinen dünner

- Buchstaben können verschieden breit werden

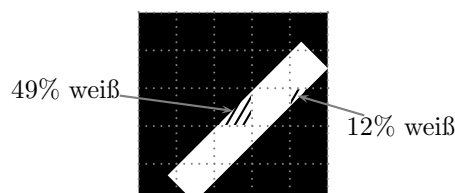


**Lösung (Antialiasing)** Verschiedene Graustufen für Pixel, statt nur schwarz und weiß

### 4.4 Antialiasing

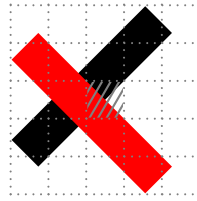
verschiedene Ansätze:

1. Betrachte den Bildpunkt als quadratische Fläche und nicht als Punkt, betrachte eine Kurve (1-dimensional) als Fläche (2-dimensional). z. B. Strecke als Rechtecke



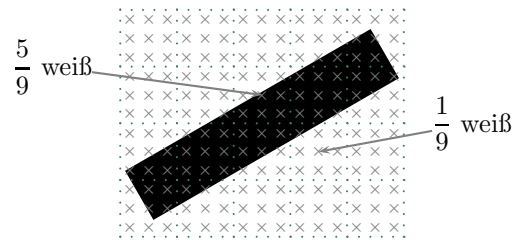


Pixel wird entsprechend hell- oder dunkelgrau gefärbt. (im Allgemeinen als proportionale Mischung aller Farben, die im Pixelquadrat vorkommen).



Aufwendig zu rechnen.

2. Supersampling: Man rechnet mit einer höheren Pixeldichte als tatsächlich vorhanden.



die verfeinerten Pixel werden „binär“ zugeordnet (in Fläche und außerhalb) und entsprechend gesetzt. Die Werte der tatsächlichen Pixel ergeben sich als Mittelwert der in ihnen enthaltenen verfeinerten Pixel.

3. Glättung:

#### Idee



Helligkeitswerte „strahlen“ auf die Nachbarn ab.

- a) berechne die Pixelwerte zunächst ohne Antialiasing.
- b) Verteile die Helligkeit von jedem Pixel auf seinen Nachbarn nach einem festen Schema.

z. B.

$\frac{1}{36}$	$\frac{4}{36}$	$\frac{1}{36}$
$\frac{4}{36}$	$\frac{16}{36}$	$\frac{4}{36}$
$\frac{1}{36}$	$\frac{4}{36}$	$\frac{1}{36}$

3. lässt sich auch mit 2. kombinieren



# 5 Helligkeit und Farbe in der Computergrafik

## 5.1 Helligkeit

**Definition** Helligkeit in der schwarz/weiß-Grafik bezeichnet einen Grauwert auf der Skala zwischen schwarz und weiß

$$\text{schwarz} = 0, \quad \text{weiß} = 1 \quad (5.1)$$

$$\text{oder schwarz} = 0, \quad \text{weiß} = 255 \quad (8\text{-Bit-Integer}) \quad (5.2)$$

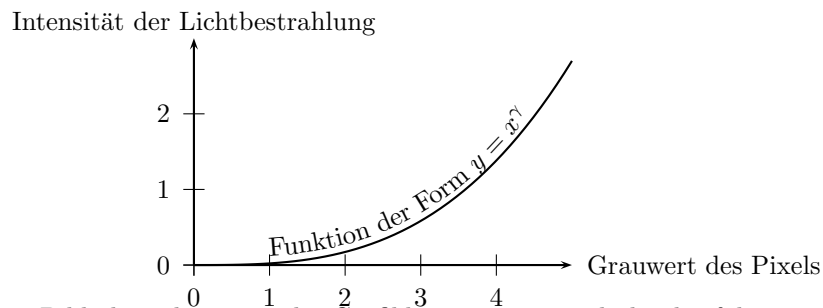
**Problem** tatsächliche Mischung 50% weiß und 50% schwarz  $\rightarrow$  sehr hellesgrau

**Weber-Fechner'sches Gesetz** beschreibt die Nichtlinearität der Sinneswahrnehmungen (für Optik und Akustik gleichermaßen). Vergrößerung der Energie um einen *konstanten Faktor* wird als Vergrößerung des Reizes in *konstanten Schritten* wahrgenommen.

**Beispiel** Eine Vergrößerung von 10 auf 12 Energieeinheiten wird genauso groß wahrgenommen wie eine Vergrößerung von 5000 auf 6000.

**Beispiel** Lautstärke wird in deziBel (dB) gemessen: Logarithmus aus der Schallenergie (oder Druck?). (Logarithmus aus konstanten Faktoren konstante Differenzen).

**Beispiel** Eine Oktave in der Musik (z. B. Abstand zwischen tiefen C und hohen C) entspricht einer Verdoppelung der Schallfrequenz.



„ $\gamma$ -Korrektur“, wird vom Bildschirm bzw. von der Grafikkarte automatisch durchgeführt.

## 5.2 Additive Farbsysteme

### 5.2.1 Das RGB-Farbsystem

In der Computergrafik geht man von einem 3-Komponenten-Modell aus: Farbe ist aus 3 Grundfarben zusammengemischt

rot (R), grün (G), blau (B)

(In Wirklichkeit: unendlich viele Grundfarben, für jede sichtbare Wellenlänge eine)



Auf einem Bildschirm sind Lichtpunkte (Phosphore) in drei Farben R, G, B nahe aneinander gitterförmig angeordnet. Die Bildpunkte werden unabhängig von einander angesteuert.

Eine Farbe in der Computergrafik ist durch 3 RGB-Werte zwischen 0 und 1 (0,1, ..., 255) charakterisiert. 24 bit pro Pixel,  $2^{24} \approx 16$  Millionen Farben

Farbe	$(r, g, b)$
rot	$(1, 0, 0)$
grün	$(0, 1, 0)$
blau	$(0, 0, 1)$
gelb (Y)	$(1, 1, 0)$
magenta (M)	$(1, 0, 1)$
cyan (C)	$(0, 1, 1)$
schwarz (K)	$(0, 0, 0)$
weiß	$(1, 1, 1)$
Grauwerte	$(x, x, x)$ alle drei Werte sind gleich

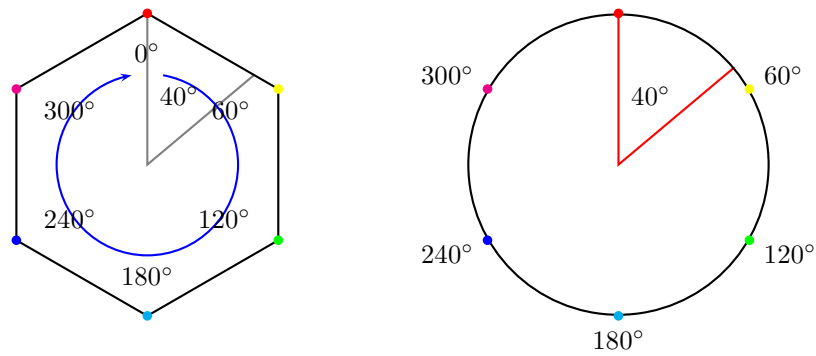


- Das RGB-System ist für die intuitive Behandlung von Farben nicht geeignet
- die Grauwerte bilden die Grauachse  $K$ -Weiß im Farbwürfel.
- die übrigen Ecken bilden das Farbsechseck RYGCMB. Auf diesem Sechseck liegen die „reinsten“/„stärksten“ Farben, alle anderen Farben kann man durch beimischen von Grau konstruieren.



Diese „stärksten Farben“ sind die Farben die mindestens eine Komponente 0 und mindestens eine Komponente 1 haben (Farben ohne Grau/Weiß/Schwarz-Anteil).

### 5.2.2 Farbsechseck bzw. Farbkreis



- Die Punkte auf diesem Sechseck werden häufig durch einen Winkel ( $0^\circ - 360^\circ$ ) parametrisiert.
- Startpunkt willkürlich ( $R = 0^\circ$ ,  $Y = 60^\circ$ , ...)
- Dieser Parameter heißt „Farbton“, „Unbuntart“ (engl. *hue* ( $H$ )).

$$\begin{aligned}
 40^\circ \text{ entspricht dann } & \frac{1}{3} \cdot R + \frac{2}{3} \cdot Y \left( \frac{1}{3} \cdot 0^\circ + \frac{2}{3} \cdot 60^\circ \right) \\
 & = \frac{1}{3} \cdot (1, 0, 0) + \frac{2}{3} \cdot (1, 1, 0) = \left( 1, \frac{2}{3}, 0 \right)
 \end{aligned}$$

Wir erhalten ein neues Farbsystem HSV

### 5.2.3 Andere Farbsysteme

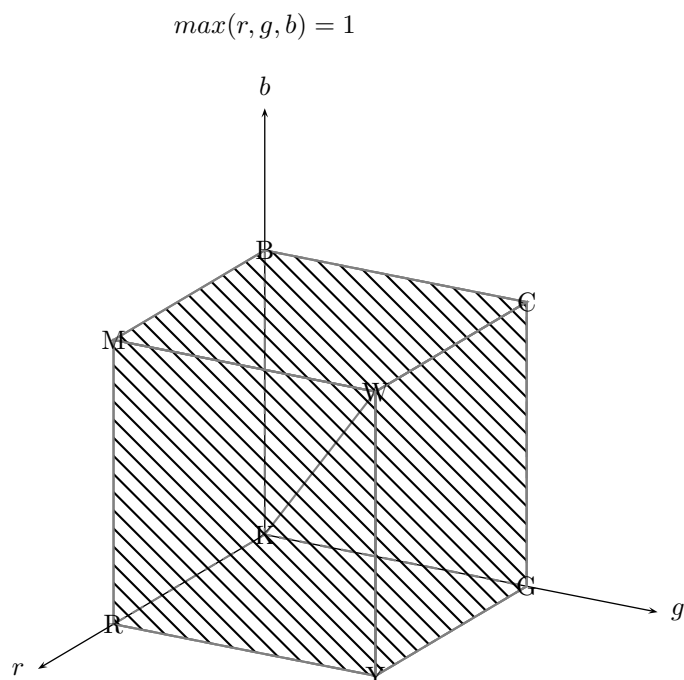
#### HSV-System

**hue** (Farbton)  $0^\circ \leq H \leq 360^\circ$

**saturation** (Sättigung)  $0 \leq S \leq 1$

**value** ( $\approx$  Helligkeit)  $0 \leq B \leq 1$

$V = 1$  sind die Farben auf den drei Deckseiten des Würfels: mindestens einer der drei Werte ist 1



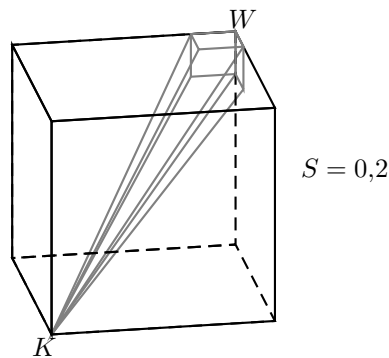
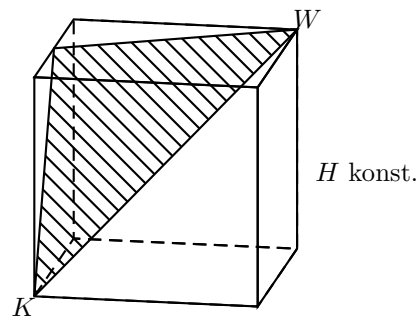
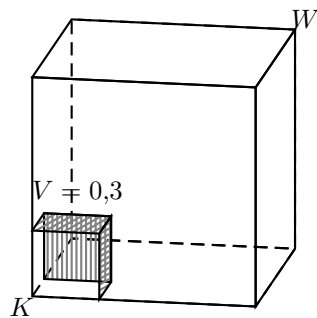
Umrechnung HSV  $\rightarrow$  RGB:

$\begin{pmatrix} r'' \\ g'' \\ b'' \end{pmatrix}$  sei die reine Farbe, die  $H$  entspricht.

$$\begin{pmatrix} r' \\ g' \\ b' \end{pmatrix} = S \begin{pmatrix} r'' \\ g'' \\ b'' \end{pmatrix} + (1 - S) \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

Ergebnis  $\begin{pmatrix} r \\ g \\ b \end{pmatrix} = V \begin{pmatrix} r' \\ g' \\ b' \end{pmatrix}$

Umrechnung RGB  $\rightarrow$  HSV



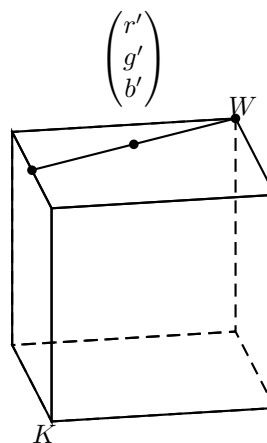
$$V = \max(r, g, b)$$

$$\begin{pmatrix} r' \\ g' \\ b' \end{pmatrix} = \begin{pmatrix} r \\ g \\ b \end{pmatrix} \cdot \frac{1}{V}$$

$$S = 1 - \min(r', g', b')$$

$$\begin{pmatrix} r'' \\ g'' \\ b'' \end{pmatrix} = \left[ \begin{pmatrix} r' \\ g' \\ b' \end{pmatrix} - (1 - S) \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \right] \cdot \frac{1}{S}$$

reine Farbe  $\rightarrow$  Umwandlung in  $H$



$\begin{pmatrix} r'' \\ g'' \\ b'' \end{pmatrix}$  ist tatsächlich eine reine Farben

1. Wir wissen  $\max(r', g', b') = 1$  z. B.  $r' = 1$

$$r'' = [1 - (1 - S)1] \cdot \frac{1}{S} = S \frac{1}{S} = 1$$

2. Nehmen wir nun an  $(r', g', b') = g' = 1 - S$

$$g'' = [g' - (1 - S)1] \cdot \frac{1}{S} = [-1 + S + 1 - S] \frac{1}{S} = 0$$

Für  $V = 0$  setze  $S, H$  beliebig.

Für  $V \neq 0, S = 0$ , setze  $H$  beliebig.



Nachteil:

$$R = (1, 0, 0)$$

$$Y = (1, 1, 0)$$

$$W = (1, 1, 1)$$

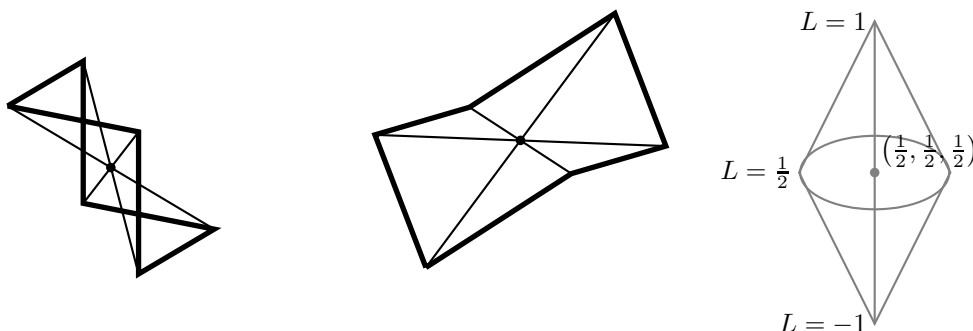
haben denselben  $V$ -Wert.

### HSL-System

**hue** (Farbton)  $0^\circ \leq H \leq 360^\circ$

**saturation** (Sättigung)  $0 \leq S \leq 1$

**lightness** (oder „luminance“)  $L = \frac{1}{2}$  enthält das reine Farbensechseck und den Graupunkt  $(\frac{1}{2}, \frac{1}{2}, \frac{1}{2})$



Umrechnung HSL  $\rightarrow$  RGB:

$\begin{pmatrix} r'' \\ g'' \\ b'' \end{pmatrix}$  sei die reine Farbe, die  $H$  entspricht.

$$\begin{pmatrix} r' \\ g' \\ b' \end{pmatrix} = S \begin{pmatrix} r'' \\ g'' \\ b'' \end{pmatrix} + (1 - S) \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

$$\text{Für } 0 \leq L \leq \frac{1}{2} : \begin{pmatrix} r \\ g \\ b \end{pmatrix} = \begin{pmatrix} r' \\ g' \\ b' \end{pmatrix} \cdot 2L$$

$$\text{Für } \frac{1}{2} \leq L \leq 1 : \begin{pmatrix} r \\ g \\ b \end{pmatrix} = \begin{pmatrix} r' \\ g' \\ b' \end{pmatrix} \cdot (2 - 2L) + \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \cdot (2L - 1)$$



$S = 1$  Farben auf allen 6 Seitenflächen des Würfels

## 5.2.4 YCrCb/YPrPb-Farbsystem

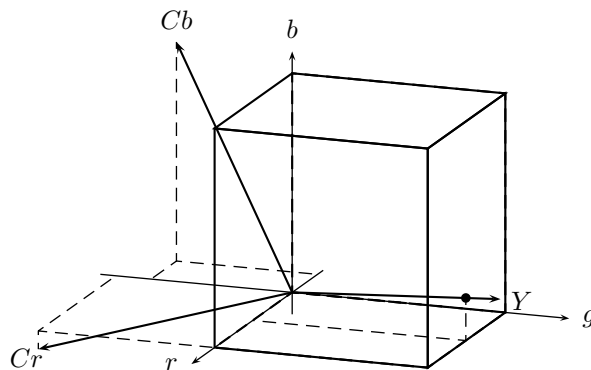
Einsatz beim Farbfernsehen

$$Y = 0,299 \cdot R + 0,587 \cdot G + 0,114 \cdot B \quad (\text{Helligkeit})$$

$$\left. \begin{array}{l} Cr = \dots \\ Cb = \dots \end{array} \right\} \text{lineare Ausdrücke in } R, G, B \text{ (Chromanz, Farbigkeit)}$$

$$0 \leq R, G, B \leq 1$$

$$0 \leq Y \leq 1 - \frac{1}{2} \leq Cr, Cb \leq +\frac{1}{2}$$



## 5.3 Subtraktive Farbmischung (z. B. beim Drucken)

3 Grundfarben



$C$  = cyan,  $B, G$  wird durchgelassen,  $R$  wird absorbiert

$M$  = magenta,  $B, R$  wird durchgelassen,  $G$  wird absorbiert

$Y$  = gelb,  $R, G$  wird durchgelassen,  $B$  wird absorbiert

$C + M$  nur Blau bleibt übrig

$C + M + Y$  = schwarz

### 5.3.1 CMY-System

$$0 \leq C, M, Y \leq 1$$



### 5.3.2 CMYK-System (Vierfarbdruck)

Zusätzlich  $K$  =schwarz. (Das Schwarz von  $K$  wird dunkler als von  $C + M + Y$  oder um Druckfarbe zu sparen.)

$$K' := \min(C, M, Y)$$

$$C' := C - K'$$

$$M' := M - K'$$

$$Y' := Y - K'$$

(möglichst viel Farbe durch  $K$  ersetzen)

## 5.4 Digitale Farbdarstellung

heutzutage:

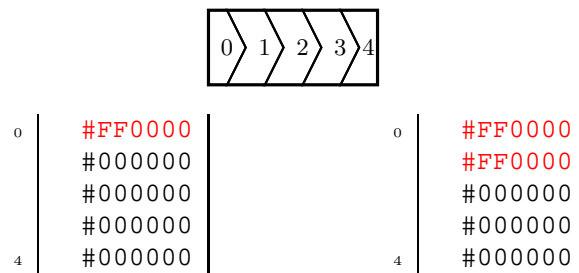
- 8 Bit pro Farbkanal ( $r, g, b$ ) ... 24 Bit pro Bildpunkt  
 $\Rightarrow 2^{24} = 16$  Mio. Farben (True Color)
  - 4-Kanal-Darstellung ( $r, g, b, \alpha$ )  $\alpha$  ist für Transparenz:
    - $\alpha = 0$ ... durchsichtig; Farbe wird vom Hintergrund genommen,
    - $\alpha = 1$ ... Farbe wird von ( $r, g, b$ ) bestimmt
    - $0 < \alpha < 1$ ... teilweise transparent
- $\rightarrow 32$  bit pro Pixel

### 5.4.1 Farbpaletten

Es wird nicht für jeden Bildpunkt eine unabhängige Farbe gespeichert (24 Bit) sondern ein Index in einer Tabelle (8 Bit, = Farbpaletten).



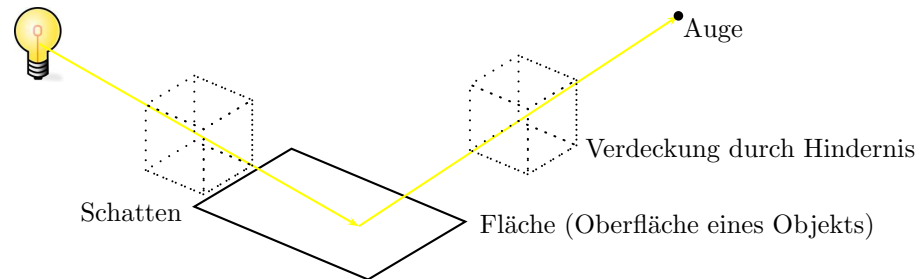
Animation ganz einfach und schnell möglich.



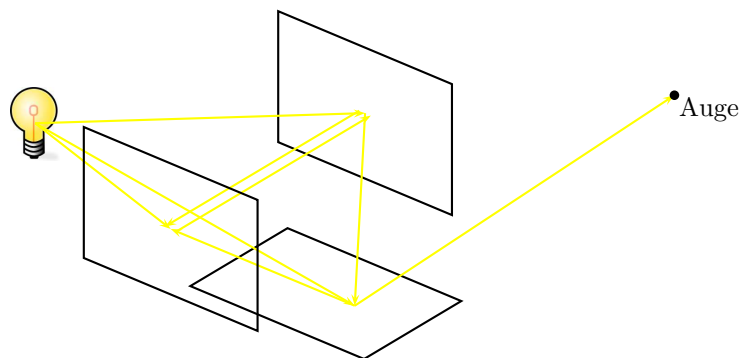
Heutzutage eher historisch.

# 6 Beleuchtung

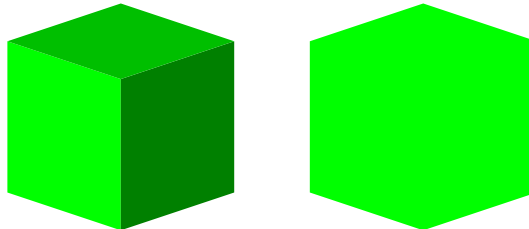
Standardfall:



In Wirklichkeit sehr komplex, in der Computergrafik aber eher selten angewandt:



Beleuchtung ist wichtig um einen räumlichen Eindruck zu erzeugen



Flächenbeschaffenheit:

- 1) diffuse Reflexion; „sieht aus allen Richtungen gleich aus“
  - 2) spiegelndes Reflexion;
- } treten auch kombiniert auf

## 6.1 Diffuse Reflexion

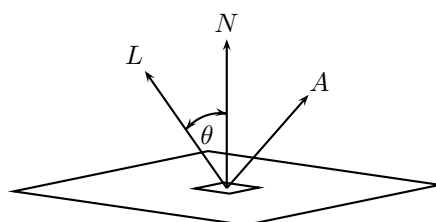
$L, N, A$  sind Einheitsvektoren:

$N$ ...Normalvektor der Fläche

$L$ ...Vektor zur Lichtquelle

$A$ ...Vektor zum Auge

$\theta$ ...Einfallswinkel (zwischen  $L$  und  $N$ )



Wie erscheint ein Punkt (Flächenstück) für das Auge?

$I$  = Intensität

zusätzliche Daten:

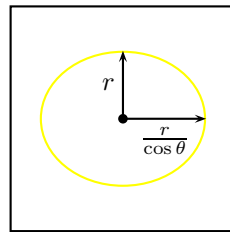
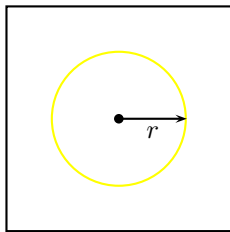
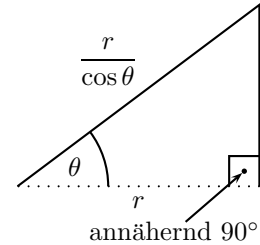
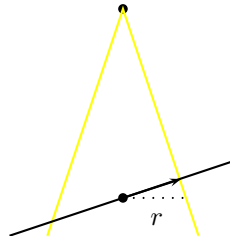
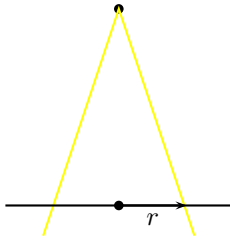
$r_L$ ...Abstand zur Lichtquelle

$I_L$ ...Intensität der Lichtquelle

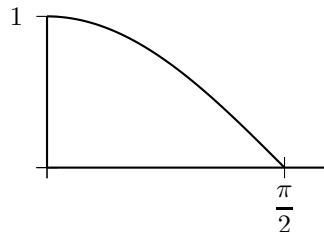
$K_D$ ...diffuser Reflexionskoeffizient der Fläche

Abhängigkeit von  $\theta$  (Lambert'sches Gesetz)

$$I = \cos \theta \cdot (\dots) \quad \cos \theta \langle N, L \rangle$$



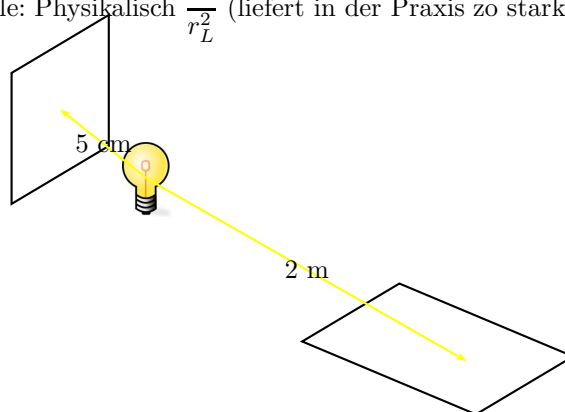
**Begründung:** Die Menge an Lichtenergie, die auf eine Fläche auftrifft, hängt vom räumlichen Winkel ab, unter dem die Fläche von der Lichtquelle aus erscheint.



Die gleiche Menge an Licht, die vorher auf eine Fläche von  $r^2\pi$  gefallen ist, fällt jetzt auf eine Fläche von  $r \cdot r \frac{1}{\cos \theta} \cdot \pi \dots$ , pro Flächeneinheit kommt um den Faktor  $\cos \theta$  beim Auge weniger an.

$\Rightarrow I$  hängt *nicht* von  $A$  ab. (Das ist spezifisch für die diffuse Reflexion)

Abstand zur Lichtquelle: Physikalisch  $\frac{1}{r_L^2}$  (liefert in der Praxis zu starke Kontraste)



Mit der Formel  $\frac{1}{r_L^2}$  würde das einen Faktor 400 bedeuten. Man nimmt stattdessen eine Formel der Gestalt:

$$\frac{1}{C_1 + C_2 r_L^2 + C_3 r_L^2} \quad \text{für passende Konstanten } C_1, C_2, C_3$$

- Formel für diffuse Reflexion

$$I = I_L \frac{1}{C_1 + C_2 r_L^2 + C_3 r_L^2} \cos \theta \cdot K_D \quad \left(0 \leq \theta \leq \frac{\pi}{2}\right)$$

- Diese Formel wird für jede Farbkomponente **R**, **G**, **B** angewandt

$$\begin{aligned} I^R &= I_L^R \frac{1}{C_1 + C_2 r_L^2 + C_3 r_L^2} \cos \theta \cdot K_D^R \\ I^G &= I_L^G \frac{1}{C_1 + C_2 r_L^2 + C_3 r_L^2} \cos \theta \cdot K_D^G \\ I^B &= I_L^B \frac{1}{C_1 + C_2 r_L^2 + C_3 r_L^2} \cos \theta \cdot K_D^B \end{aligned}$$

- In Wirklichkeit müsste man das für jede Wellenlänge  $\lambda$  getrennt ausrechnen.

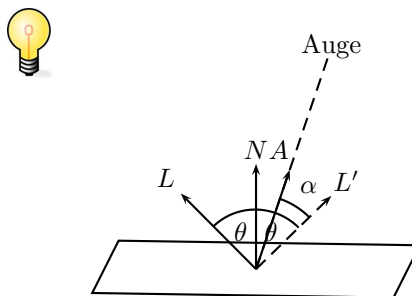
$$\cos \theta \text{ durch } \max(0, \langle N, L \rangle)$$

- Für unendlich ferne *Lichtquellen* lässt man den Term  $\frac{1}{C_1 + C_2 r_L^2 + C_3 r_L^2}$  weg.  $I_L$  hat eine andere Bedeutung.  $L$  ist dann konstant.
- Man betrachtet auch „diffuses Licht“. Es kommt aus allen Richtungen gleich stark.

$$I = I_{L,D} \cdot K_D \quad \text{für } R, G, B \text{ getrennt}$$

- Um die Helligkeit eines Flächenstücks (in  $R, G, B$ ) zu berechnen, werden die Ergebnisse der verschiedenen Lichtquellen und der diffusen Beleuchtung aufsummiert.

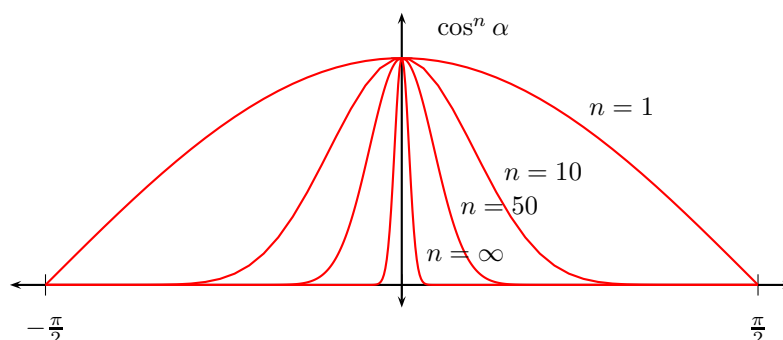
## 6.2 Spiegelnde Reflexion

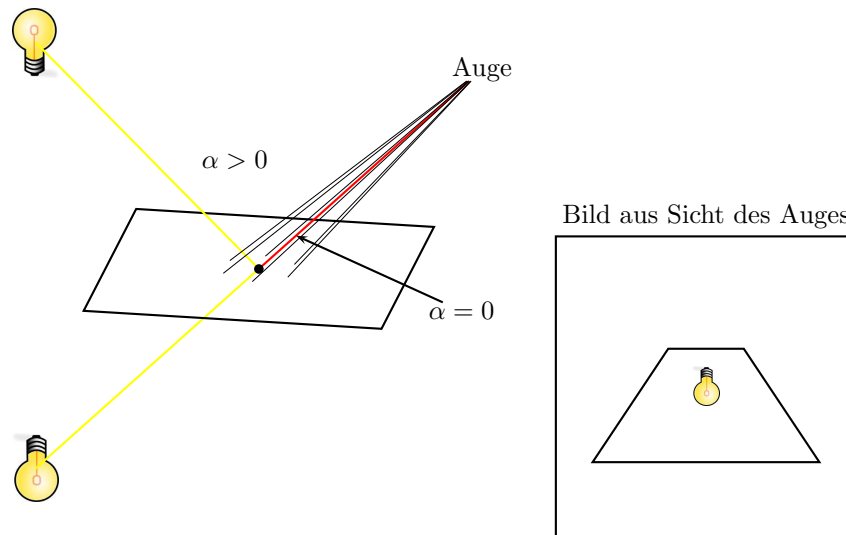


$L'$  ist der an  $N$  gespiegelte Vektor, die der „idealen Spiegelung“

$$\alpha = \angle(A, L')$$

Modell von BUI-TONG PHONG: Intensität ist proportional zu  $\cos^n \alpha$





Formel für spiegelnde Reflexion:

$$I^X = I_L^X \cdot \frac{1}{C_0 + C_1 r + C_2 r^2} \cdot K_S \cdot \cos^n \alpha, \quad X = R, G, \text{ oder } B$$

$I_L$ ... Intensität der Lichtquelle ( $I_L^R, I_L^G, I_L^B$ )

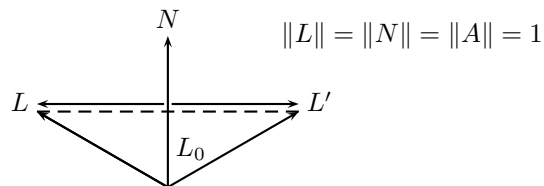
$\frac{1}{C_0 + C_1 r + C_2 r^2}$ ... Abhängigkeit von der Entfernung der Lichtquelle

$K_S$ ... spiegelnde Reflexionskoeffizient *unabhängig von der Farbe!*

$n$ ... Exponent: gibt an wie glatt die Spiegelung ist

$\frac{1}{C_0 + C_1 r + C_2 r^2}$ ... Abhängigkeit von der Entfernung der Lichtquelle

$I^R, I^G, I^B$ ... Ergebnis-Intensität aus Sicht des Auges



$L_0$ ... Projektion von  $L$  auf  $N$

$$= \langle L, N \rangle \cdot N$$

$$L' = -(L - L_0) + L_0 = 2L_0 - L, \|L'\| = 1$$

$$\cos \alpha = \langle A, L' \rangle = \langle A, 2L_0 \rangle - \langle A, L \rangle$$

$$= 2\langle L, N \rangle \langle N, A \rangle - \langle A, L \rangle$$

$$= \cos \alpha$$

**Näherungsmodell** (einfacher zu rechnen)

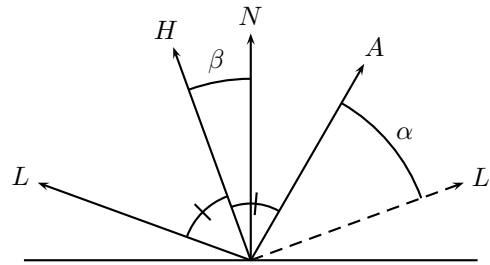
$H$  der Vektor, der symmetrisch zwischen  $A$  und  $L$  liegt.

$$H_0 = A + L$$

$$H = \frac{H_0}{\|H_0\|}$$

Statt  $\cos \alpha$  nimmt man  $\cos \beta, \beta = \angle(H, N)$

$$\cos \beta = \langle H, N \rangle \quad \beta = 0 \Leftrightarrow \alpha = 0$$



$\beta = \frac{\alpha}{2}$ , wenn  $L, N, A$  in einer Ebene liegen. Im Raum ist das nicht immer der Fall.

## 6.3 Gesamtbeleuchtung

- mehrere Lichtquellen an bestimmten Orten (bzw. aus bestimmten Richtungen) mit Intensität  $I_L^R, I_L^G, I_L^B$
- eine diffuse Lichtquelle mit Intensität  $I_D^R, I_D^G, I_D^B$

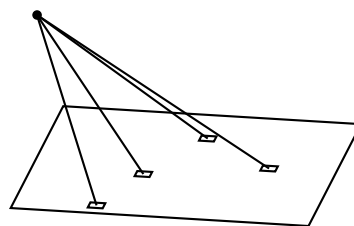
Für jede Fläche:

- diffuse Reflexionskoeffizienten  $K_D^R, K_D^G, K_D^B$
- spiegelnde Reflexionskoeffizienten  $K_S$ , Exponent  $n$
- möglicherweise eine Eigenleuchtintensität  $I_E^R, I_E^G, I_E^B$  (z. B. Leuchtschirm, glühendes Ofenrohr, flächig leuchtende Lichtquelle)

Für jedes Flächenstück addiert man alle Beleuchtungskomponenten zusammen (falls die Fläche sichtbar vom Auge ist).

$$\begin{aligned}
 I^R = & \sum \text{diffuse Reflexionen von allen Lichtquellen, die das Flächenstück beleuchten.} \\
 & + I_D^R \cdot K^R \\
 & + \text{spiegelnde Reflexionen von allen Lichtquellen, die das Flächenstück beleuchten.} \\
 & + I_E^R
 \end{aligned}$$

analog für  $I^G, I^B$



eigentlich liefert die Rechnung in jedem Punkt der Fläche ein anderes Ergebnis.

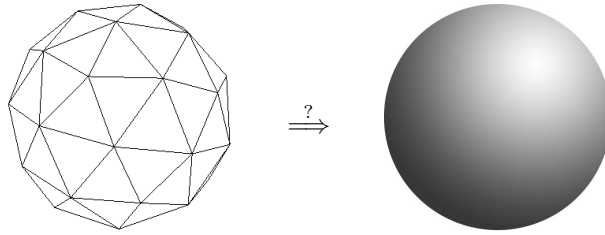
**Problem** Die Intensität kann in einzelnen Punkten des Bildes  $> 1$  werden. 2 Möglichkeiten

1. herunterskalieren aller Werte  
z. B.  $(r; g; b) = (3; 1; 0,5) \rightarrow (1, \frac{1}{3}, \frac{1}{6})$
2. zu große Werte werden auf 1 gesetzt (dadurch können auch Farben verfälscht werden)  
z. B.  $(r; g; b) = (3; 1; 0,5) \rightarrow (1, 1, 0,5)$   
(sehr rotes orange) (helles gelb)

## 6.4 Schattierung (Shading)

Alle folgenden Rechnungen in Weltkoordinaten

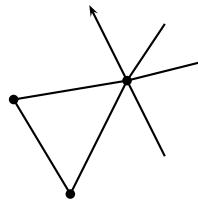
**Definition** Unter *Shading* versteht man die Anwendung der Beleuchtungsregeln auf jeden Punkt einer Fläche, sodass sich eine abgestufte Farbverteilung ergibt, die realistisch aussieht.



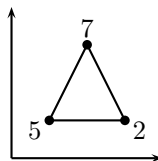
- Gekrümmte Flächen können durch ein genügend feines *Dreiecksnetz* (oder Vierecksnetz) approximiert werden.
- Große flache Flächen können in kleinere Dreiecke zerlegt werden.

GOURAUD-Schattierung für ein Dreiecksnetz:

- Berechne die Beleuchtung für jede Ecke eines Dreiecks, mit einem Normalvektor, der für jede Ecke fest ist (unabhängig von dem Dreieck zu dem es gehört). z. B. der Normalvektor der glatten Fläche, die durch das Dreiecksnetz approximiert wird.

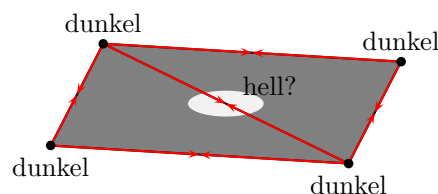


- Interpoliere die Beleuchtung linear auf jedem Dreieck



$$f(x, y) = ax + by + c$$

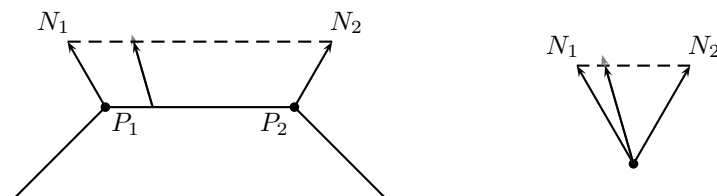
Die Gouraud-Schattierung, versagt z. B. bei Glanzlichtern (können verschwinden oder unnatürlich vergrößert werden) und sehr großen Flächen:



PHONG-Schattierung:

~~diffuse Lichtquelle~~ *Umgebungslicht* (ambient light)

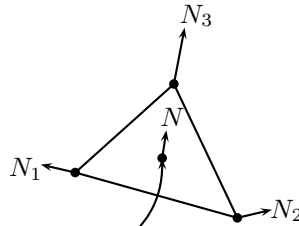
- An den Ecken werden die Normalen genommen. Dazwischen werden die Normalenrichtungen angenähert interpoliert und mit diesen Normalen in jedem Punkt die Beleuchtungsrechnung durchgeführt





Am Punkt  $P = \lambda P_2 + (1 - \lambda)P_1$  wird der folgende Normalvektor genommen:

$$N = \frac{\lambda N_2 + (1 - \lambda)N_1}{\|\lambda N_2 + (1 - \lambda)N_1\|}$$



$$P = \lambda_1 P_1 + \lambda_2 P_2 + \lambda_3 P_3 \rightarrow N = \frac{\lambda_1 N_1 + \lambda_2 N_2 + \lambda_3 N_3}{\|\lambda_1 N_1 + \lambda_2 N_2 + \lambda_3 N_3\|}, \quad (\lambda_1 + \lambda_2 + \lambda_3 = 1)$$

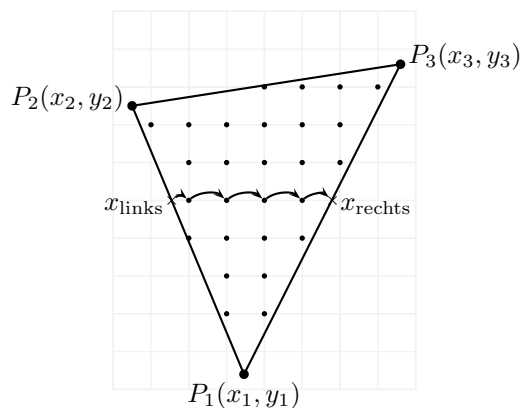
Aufwendiger zu rechnen als Gouraud-Schattierung

### 6.4.1 Ausfüllen einer gerasterten Fläche

**Annahme** Fläche ist ein Dreieck

$$y_1 \leq y_2 \leq y_3$$

$P_2$  links von  $P_1 P_3$



in  $P_1, P_2, P_3$  sind Intensitätswerte  $I_1, I_2, I_3$  berechnet worden, diese sollen linear interpoliert werden. (z. B. bei Gouraud-Schattierung) eigentlich sollte man die Interpolation in Weltkoordinaten machen, und für  $R, G, B$  getrennt.

Setpixel( $x, y, I$ )  
 ↑  
 ganzzahlig

Wir füllen das Dreieck zeilenweise

```
FuelleZeile( $x_{\text{links}}, x_{\text{rechts}}, y, I_{\text{links}}, \Delta I$ )
    //  $\Delta I = I(x+1, y) - I(x, y)$  = unabhaengig von  $x$  und  $y$ ,
    // weil  $I$  eine lineare Funktion ist
     $x := \lceil x_{\text{links}} \rceil$ 
     $I := I_{\text{links}} + \Delta I \cdot (x - x_{\text{links}})$ 
    while  $x \leq x_{\text{rechts}}$ 
        Setpixel( $x, y, I$ )
         $x := x + 1$ 
         $I := I + \Delta I$ 
```

äußere Schleife:

## 1. untere Hälfte

$$\Delta x_{\text{links}} = \frac{x_2 - x_1}{y_2 - y_1}$$

$$\Delta x_{\text{links}} = \frac{x_3 - x_1}{y_3 - y_1}$$

$$\Delta I_{\text{links}} = \frac{I_2 - I_1}{y_2 - y_1}$$

$$\Delta I = \frac{(I_1 + \Delta I_{\text{rechts}}) - (I_1 + \Delta I_{\text{links}})}{(x_1 + \Delta x_{\text{rechts}}) - (x_1 + \Delta x_{\text{links}})}$$

$$= \frac{\Delta I_{\text{rechts}} - \Delta I_{\text{links}}}{\Delta x_{\text{rechts}} - \Delta x_{\text{links}}}$$

Ziel: auf der Geraden  $P_1P_2$  gilt  $x = y_1 + (y - y_1) \cdot \Delta x_{\text{links}}$

Ziel: auf der Geraden  $P_1P_2$  gilt  $I = I_1 + (y - y_1) \cdot \Delta I_{\text{links}}$

$$\text{mit } \Delta I_{\text{rechts}} = \frac{I_3 - I_1}{y_3 - y_1}$$

```

y = ⌈y₁⌉;  x_links    := x₁ + (y - y₁) · Δx_links
           x_rechts   := x₁ + (y - y₁) · Δx_rechts
           I_links    := I₁ + (y - y₁) · ΔI_links
while y ≤ y₂
  FuelleZeile(x_links, x_rechts, y, I_links, ΔI)
  y := y + 1
  x_links := x_links + Δx_links
  x_rechts := x_rechts + Δx_rechts
  I_links := I_links + ΔI_links

```

## 2. obere Hälfte

$$\Delta x_{\text{links}} := \frac{x_3 - x_2}{y_3 - y_2}; \quad \Delta I_{\text{links}} := \frac{I_3 - I_2}{y_3 - y_2}$$

$$y := y + 1; \quad x_{\text{rechts}} := x_{\text{rechts}} + \Delta x_{\text{links}}$$

$$x_{\text{links}} := x_2 + (y - y_2) \cdot \Delta x_{\text{links}}$$

$$I_{\text{links}} := I_2 + (y - y_2) \cdot \Delta I_{\text{links}}$$

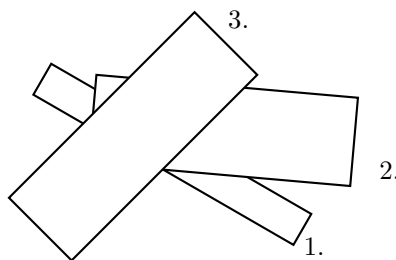
```

while y ≤ y₃
  FuelleZeile(x_links, x_rechts, y, I_links, ΔI)
  y := y + 1
  x_links := x_links + Δx_links
  x_rechts := x_rechts + Δx_rechts
  I_links := I_links + ΔI_links

```

## 6.5 Entfernen verdeckter und teilweise verdeckter Flächen

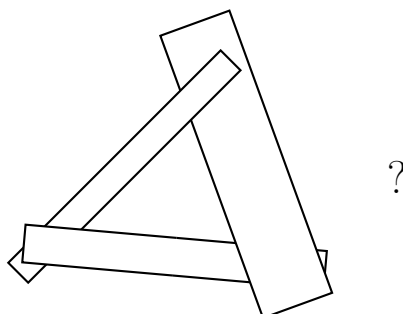
1. Painter's Algorithmus
2. Tiefenpuffer
3. Überstreichen (swap)



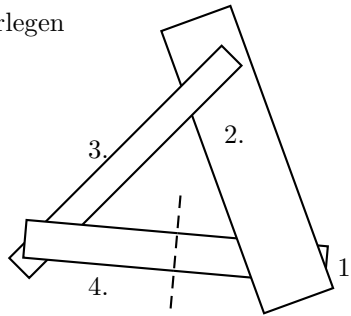
## 6.5.1 Painter's Algorithmus

Finde eine Reihenfolge der Flächen nach vorn, „male“ die Flächen in dieser Reihenfolge.

**Problem** zyklische Abhängigkeiten:



**Lösung** Flächen in kleinere Flächen zerlegen



### 6.5.2 Tiefenpuffer (z-Puffer)

Jeder Bildpunkt speichert zusätzlich einen  $z$ -Wert (größere Werte sind weiter hinten)

$$\text{Setpixel}(x, y, z, I) \text{ bzw. } \text{Setpixel}(x, y, z, I^R, I^G, I^B)$$

Vergleiche  $z$  mit dem für  $(x, y)$  gespeicherten  $z$ -Wert, wenn kleiner, dann überschreibe  $z, I$ , andernfalls ignoriere.

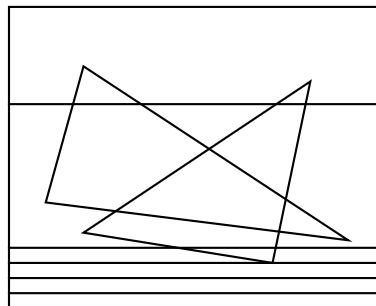
Beim Zeichnen eines ebenen Flächenstückes ist  $z$  linear von  $x$  und  $y$  abhängig.  $z$  interpoliert zwischen den 3 Ecken  $P_i(x_i, y_i, z_i), i = 1, 2, 3$

**Nachteil** (von beiden Verfahren) Aufwand ist proportional zu gezeichneten Gesamtfläche und nicht zur sichtbaren Fläche (nicht sichtbare Flächen werden auch gezeichnet bzw. zu zeichnen versucht).

### 6.5.3 Darstellung einer Szene durch Überstreichen (Scanline-Algorithmus)

**Vereinfachte Annahme** Szene besteht aus Dreiecken.

- Der Algorithmus beruht auf der gleichen Idee wie der Algorithmus zum Ausfüllen gerasterter Flächen (s. Abschnitt 6.4.1)
  - Überstreiche Szene mit horizontalen Geraden (von unten nach oben, mit steigendem  $y$ )
    - \* Für jede Rasterzeile gehe Pixel von links nach rechts durch (mit steigendem  $x$ )
- Prinzip: Der Algorithmus aus Abschnitt 6.4.1 wird für alle Dreiecke parallel angewendet



**Vorverarbeitung** Wir erstellen eine Kantenliste (Edge List, EL), sortiert nach den Koordinaten des unteren Endpunktes der Kante, zuerst nach der  $y$ -Koordinate sortiert, bei gleicher  $y$ -Koordinate nach  $x$ -Koordinate (horizontal Kanten brauchen nicht gespeichert werden).

**Bearbeitung einer Zeile** Wir benötigen zur Abarbeitung einer Zeile die Menge der von der Zeile geschnittenen Dreiecke und für diese dann die nötigen Parameter, wie sie für die Fülle-Zeile-Funktion aus Abschnitt 6.4.1 benutzt wurden.

Wir verwalten die Daten in einer „Aktive-Kanten-Liste“ (AEL). Darin enthalten sind alle Kanten, die die aktuelle Scanline schneiden. Sie sind sortiert nach der  $x$ -Koordinate des Schnittpunktes mit der Scanline. Zu jeder Kante speichern wir die Informationen, die wir benötigen. Um das Dreieck für diese Zeile zu rastern und die entsprechenden Werte für die Zeile auszurechnen.

z. B.

- zugehöriges Dreieck
- linke oder rechte Begrenzung
- $y$ -Koordinate des oberen Endpunktes (um die Kante aus der AEL entfernen zu können)
- $x$ -Koordinate Schnittpunktes mit der Scanline (entspricht  $x_{\text{links}}$  bzw.  $x_{\text{rechts}}$ )
- $\Delta x$  (entspricht  $\Delta x_{\text{links}}$  bzw.  $\Delta x_{\text{rechts}}$ )

die folgenden Daten brauchen nur in einer linken Begrenzung gespeichert werden:

- $z_{\text{links}}, I_{\text{links}}, \Delta z_{\text{links}}, \Delta I_{\text{links}}, \Delta z, \Delta I$  (Notation s. Abschnitt 6.4.1)

und entsprechende Werte für andere zu interpolierende Daten.

**Änderung der AEL beim Übergang von  $y$  auf  $y + 1$**  wie bei der äußeren Schleife in Abschnitt 6.4.1.

- inkrementiere  $x, z_{\text{links}}, I_{\text{links}}$  um  $\Delta x, \Delta z_{\text{links}}, \Delta I_{\text{links}}$

Außerdem kann es nötig sein Kanten aus der Liste zu löschen (Kante endet zwischen  $y$  und  $y + 1$ ), neue Kanten aufzunehmen (Kante beginnt zwischen  $y$  und  $y + 1$ ) oder die Reihenfolge der Kanten zu ändern (Kanten schneiden sich zwischen  $y$  und  $y + 1$ )

**1. Kante endet zwischen  $y$  und  $y + 1$**

Vergleiche  $y$ -Koordinate des oberen Endpunktes einer Kante in der AEL mit der aktuellen Scanline-Koordinate

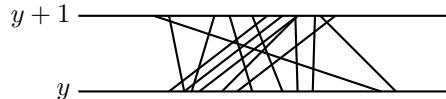
**2. Kante beginnt zwischen  $y$  und  $y + 1$**

Vergleiche  $y$ -Koordinate des unteren Endpunktes der ersten noch nicht verwendeten Kante in der EL mit aktuellen Scanline-Koordinaten.

**3. Kanten schneiden sich zwischen  $y$  und  $y + 1$**

prinzipiell vergleiche die neuen  $x$ -Werte zweier Kanten die in der alten AEL benachbart waren

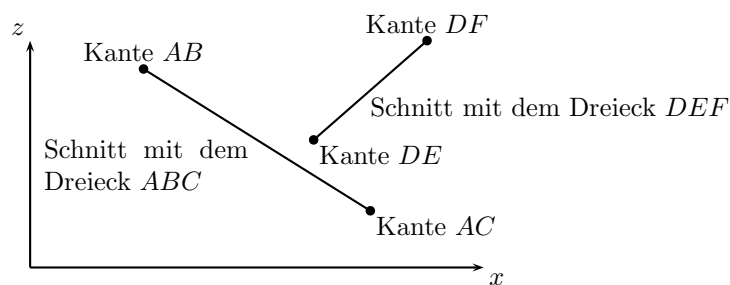
**Problem** Die Scanline wird nicht Kontinuierlich



Es können sich auch nichtbenachbarte Kanten zwischen  $y$  und  $y + 1$  schneiden, wenn die dazwischen liegenden Kanten ebenfalls geschnitten werden. Wir müssen also nicht nur bei  $y$  benachbarte Kanten vergleichen, sondern auch die nahe einer Kreuzung entstehenden neuen Nachbarn.

Dieser Ansatz läuft auf ein Bubble-Sort- oder Insertion-Sort-Prinzip hinaus. Dann ist Neusortieren mit effizienten Sortierv Verfahren.

**Füllen einer Zeile** Zeile wird von links nach rechts durchlaufen. Beachten wir dabei die Tiefeninformation entspricht dies einem Überstreichen einer Ebene  $y = \text{const}$  mit einer Geraden  $x = \text{const}$



in dieser Ebene haben wir Kante, die den Schnitten von Dreiecken mit der Ebene entsprechen. Der Dreiecksschnitt, dessen  $z$ -Koordinate für ein gegebenes  $x$  am kleinsten ist, bestimmt die Färbung des Pixels.

Analog zur AEL legen wir eine aktive Dreiecksliste (ATL, auch active span list) an, um der die von der aktiven Geraden  $x = \text{const}$  geschnittenen Dreiecksschnitte nach  $z$ -Koordinate sortiert sind und die zugehörigen Werte für die Interpolation gespeichert sind. Das sind  $z, I$  und  $\Delta z$ , sowie  $\Delta I$  und  $x_{\text{rechts}}$ .

Modifizierung der ATL beim Übergang von  $x$  auf  $x + 1$

1. Dreieck einfügen:  $x_{\text{links}}$  liegt zwischen  $x$  und  $x + 1$  für eine linke Kante in der AEL
2. Dreieck löschen:  $x_{\text{rechts}}$  liegt zwischen  $x$  und  $x + 1$
3. Zwei Dreiecke schneiden sich ( $z$ -Werte tauschen Reihenfolge)

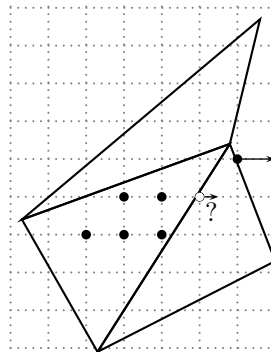
zu zeichnen ist die aktuelle Farbe des Dreiecks, das in der ATL aktuell an der ersten Stelle steht.

**Varianten, Sonderfälle, ...**

1. Es ist häufig sinnvoll, sobald zwei Kanten in der AEL benachbart werden, den Schnittpunkt zu berechnen und diesen, falls er existiert, in einem *Event-Schedule* einzufügen. Dieser verwaltet die „Zeit“-punkte, an denen sich die Reihenfolge der Kanten ändert in einer Prioritätswarteschlange.
2. Vereinfachungen bei sich nicht schneidenden Dreiecken
3. Kombination mit dem Tiefenpuffer
  - z. B. tiefenpuffer für eine Zeile des Bildes
    - Überstreiche Szene zeilenweise
    - Vor jedem Wechsel in eine neue Zeile leere Tiefenpuffer
    - für jede Zeile: Zeichne alle Dreiecke in der aktuellen AEL (in beliebiger Reihenfolge) nach dem Tiefenpuffer-Prinzip

Analog: Tiefenpuffer für ein Pixel

Größe des Tiefenpuffers	Struktur des Algorithmus
Breite $\times$ Höhe $\times$ Genauigkeit	Für alle Dreiecke Für alle Zeilen Für alle Spalten
Breite $\times$ Genauigkeit	Für alle Zeilen Für alle Dreiecke Für alle Spalten
Genauigkeit	Für alle Zeilen Für alle Spalten Für alle Dreiecke

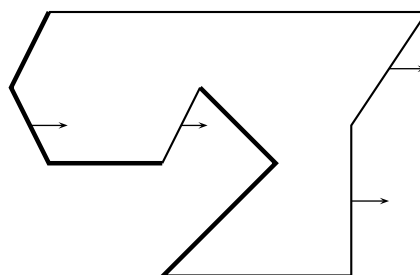
**6.6 Anzeigen von benachbarten Flächenstücken (Dreiecken)**

Gehört der Rand eines Dreiecks zum Dreieck?

Zu welchem Dreieck?

**Gesucht** Eine konsistente Regel, die bei mehreren aneinanderstoßenden Flächen festlegt, zu welcher Fläche jeder Bildpunkt gehört.

**Eine Möglichkeit** Ein Pixel, das auf dem Rand liegt wird (in Gedanken) horizontal nach rechts verschoben. Es gehört zu der Fläche, wo es dann landet. Wenn es dabei auf einer horizontalen Kante liegt, dann wird es zur oberen Fläche gerechnet.



**Voraussetzung** Exakte Arithmetik

## 6.7 Lineare Interpolation in Weltkoordinaten

Interpolation von Helligkeitswerten (Gouraud-Schattierung), von Normalenrichtungen (Phong-Schattierung), bei Texturen immer in *Weltkoordinaten*! Nicht NDC.

**Gegeben** Eine Strecke  $P_1P_2$  in NDC (oder Bildschirmkoordinaten) als Bild einer Strecke  $P_1P_2$  in Weltkoordinaten.

An den Endpunkten sind Werte  $I_1, I_2$  gegeben

(Intensitätswerte  $I_1^R, I_1^G, I_1^B, \dots$ , oder Normalenrichtungen  $N_1^x, N_1^y, N_1^z, \dots$ )

Wir wollen diese Werte für die dazwischen liegenden Punkte der Strecke in Weltkoordinaten interpolieren.

Wir kennen die Transformationsmatrix  $M = M^{\text{Welt, NDC}}$ :

$$M \begin{pmatrix} x_{\text{Welt}} \\ y_{\text{Welt}} \\ z_{\text{Welt}} \\ w_{\text{Welt}} \end{pmatrix} = \begin{pmatrix} x_{\text{NDC}} \\ y_{\text{NDC}} \\ z_{\text{NDC}} \\ w_{\text{NDC}} \end{pmatrix} \quad P_1 = \begin{pmatrix} x_1^{\text{NDC}} \\ y_1^{\text{NDC}} \\ z_1^{\text{NDC}} \\ 1 \end{pmatrix} \quad P_2 = \begin{pmatrix} x_2^{\text{NDC}} \\ y_2^{\text{NDC}} \\ z_2^{\text{NDC}} \\ 1 \end{pmatrix}$$

$$P_1 \text{ in homogenen Weltkoordinaten} = M^{-1} \begin{pmatrix} x_1^{\text{NDC}} \\ y_1^{\text{NDC}} \\ z_1^{\text{NDC}} \\ 1 \end{pmatrix} = \begin{pmatrix} x_1^{\text{Welt}} \\ y_1^{\text{Welt}} \\ z_1^{\text{Welt}} \\ w_1^{\text{Welt}} \end{pmatrix} \hat{=} \begin{pmatrix} x_1^{\text{Welt}}/w_1^{\text{Welt}} \\ y_1^{\text{Welt}}/w_1^{\text{Welt}} \\ z_1^{\text{Welt}}/w_1^{\text{Welt}} \\ 1 \end{pmatrix} \text{ in kartesischen Weltkoordinaten}$$

**Annahme:**  $I : \mathbb{R}^3 \rightarrow \mathbb{R}$  linear  $I(P_1) = I_1$   $I(P_2) = I_2$

Betrachte eine „homogene Erweiterung“:  $\hat{I} : \mathbb{R}^4 \rightarrow \mathbb{R}$  von  $I$

$$\hat{I} \left( \begin{pmatrix} \lambda x \\ \lambda y \\ \lambda z \\ \lambda w \end{pmatrix} \right) = \lambda \cdot \hat{I} \left( \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} \right) \quad \hat{I} \left( \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \right) = I \left( \begin{pmatrix} x \\ y \\ z \end{pmatrix} \right)$$

1. Die Funktion  $\hat{I}$  ist linear;
2. Aus  $\hat{I}$  kann man  $I$  ausrechnen:

$$I \left( \text{Punkt} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} \text{ in homogenen Koordinaten} \right) = \frac{1}{w} \cdot \hat{I} \left( \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} \right)$$

$$\begin{aligned} I \left( \text{Punkt in homogenen Koordinaten} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} \right) &= \\ I \left( \text{Punkt in homogenen Koordinaten} \begin{pmatrix} x/w \\ y/w \\ z/w \\ 1 \end{pmatrix} \right) &= \hat{I} \left( \begin{pmatrix} x/w \\ y/w \\ z/w \\ 1 \end{pmatrix} \right) = \frac{1}{w} \cdot \hat{I} \left( \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} \right) \\ I \left( \text{Punkt in kartesischen Koordinaten} \begin{pmatrix} x/w \\ y/w \\ z/w \end{pmatrix} \right) &= \end{aligned}$$

1. Berechne  $P_1$  und  $P_2$  in homogenen Weltkoordinaten:

$$\begin{pmatrix} x_1^{\text{Welt}} \\ y_1^{\text{Welt}} \\ z_1^{\text{Welt}} \\ w_1^{\text{Welt}} \end{pmatrix} = M^{-1} \cdot \begin{pmatrix} x_1^{\text{NDC}} \\ y_1^{\text{NDC}} \\ z_1^{\text{NDC}} \\ 1 \end{pmatrix} \quad \text{und} \quad \begin{pmatrix} x_2^{\text{Welt}} \\ y_2^{\text{Welt}} \\ z_2^{\text{Welt}} \\ w_2^{\text{Welt}} \end{pmatrix} = M^{-1} \cdot \begin{pmatrix} x_2^{\text{NDC}} \\ y_2^{\text{NDC}} \\ z_2^{\text{NDC}} \\ 1 \end{pmatrix}$$

2. Berechne  $\hat{I}_1$  und  $\hat{I}_2$  an diesem Punkt in  $\mathbb{R}^4$

$$I(P_1) = I_1 = \frac{1}{w} \cdot \hat{I}_1$$

$$\hat{I}_1 = w_1^{\text{Welt}} \cdot I_1, \quad \hat{I}_2 = w_2^{\text{Welt}} \cdot I_2$$

3. Lineare Interpolation  $P(\lambda) = \lambda_2 + (1 - \lambda)P_1$  in NDC

**Gesucht**  $I(\lambda) = I(P(\lambda))$

$$\begin{pmatrix} x(\lambda) \\ y(\lambda) \\ z(\lambda) \\ 1 \end{pmatrix} = \lambda \begin{pmatrix} x_2 \\ y_2 \\ z_2 \\ 1 \end{pmatrix} + (1 - \lambda) \begin{pmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{pmatrix}$$

$$\hat{I} = \lambda \cdot \hat{I}_2 + (1 - \lambda) \hat{I}_1$$

$$w^{\text{Welt}}(\lambda) = \lambda \cdot w_2^{\text{Welt}} + (1 - \lambda)w_1^{\text{Welt}} \quad I(\lambda) = \frac{\hat{I}(\lambda)}{w^{\text{Welt}}(\lambda)}$$

**Zusammenfassung**  $w_1^{\text{Welt}}, w_2^{\text{Welt}} = \text{letzte Komponente von } M^{-1} \begin{pmatrix} \dots \end{pmatrix}$

$$I(\lambda) = \frac{\overbrace{w_2^{\text{Welt}} \cdot I_2}^{\hat{I}_2} \cdot \lambda + \overbrace{w_1^{\text{Welt}} \cdot I_1}^{\hat{I}_1} \cdot (1 - \lambda)}{w_2^{\text{Welt}} \lambda + w_1^{\text{Welt}} (1 - \lambda)}$$

**Einbau in Füllalgorithmus** zusätzlich

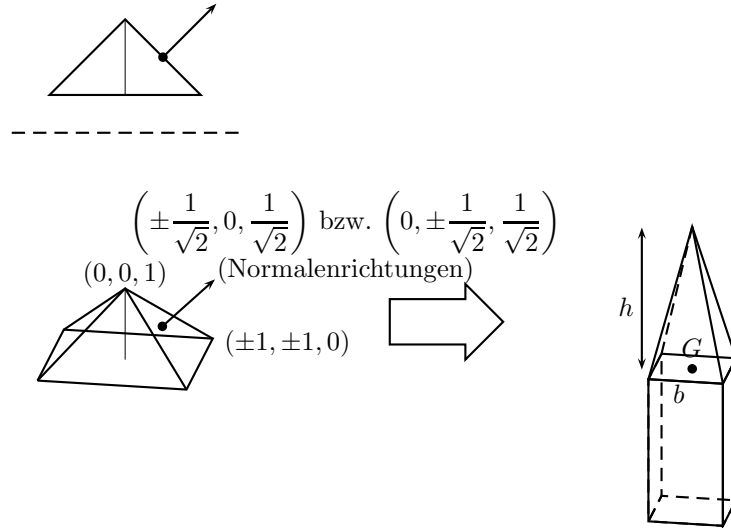
```
FuelleGerade( $x_1, x_2, y, z_1, z_2, w_1, w_2, I_1, I_2$ )
     $\hat{I}_1 = w_1 I_1$ 
     $\hat{I}_2 = w_2 I_2$ 
     $\Delta \hat{I} = \dots$ 
     $\Delta w = \dots$ 
     $\vdots$ 
    while  $x < x_2$ 
        Setpixel( $x, y, \dots, I$ )
         $x := x + 1$ 
         $z := z + \Delta z$ 
         $\hat{I} := \hat{I} + \Delta \hat{I}$  // Zaehler
         $w := w + \Delta w$  // Nenner
         $I = \frac{\hat{I}}{w}$ 
```

analog für mehrere Größen  $I^R, I^G, I^B$  nimmt man  $\hat{I}^R, \hat{I}^G, \hat{I}^B, w$  und  $\Delta w$  braucht man nur einmal.

**Bemerkung** Die  $z$ -Koordinate kann man direkt in NDC interpolieren

## 6.8 Transformation von Normalvektoren bei affinen Transformationen

z. B. Objektkoordinaten auf Weltkoordinaten



$$T = \begin{pmatrix} b/2 & 0 & 0 & g_x \\ 0 & b/2 & 0 & g_y \\ 0 & 0 & h & g_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$T = \left( \begin{array}{c|c} A & b \\ \hline 0 & 1 \end{array} \right) \quad \begin{pmatrix} x \\ y \\ z \end{pmatrix} \mapsto A \underbrace{\begin{pmatrix} x \\ y \\ z \end{pmatrix}} + b$$

### Ebene im Urbildraum

$$E = \left\{ \vec{x} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \mid \vec{n}^T \cdot \vec{x} = c \right\} \quad \vec{n} \dots \text{Normalenvektor, } c \in \mathbb{R}$$

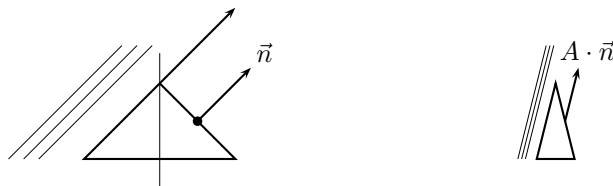
$$T(E) = \left\{ \underbrace{A \cdot \vec{x} + b}_{=\bar{x}} \mid \vec{n}^T \cdot \vec{x} = c \right\}$$

$$Ax + b = \bar{x} \quad x = A^{-1}(\bar{x} - b)$$

$$\begin{aligned} T(E) &= \{ \bar{x} \mid n^T A^{-1}(\bar{x} - b) = c \} \\ &= \{ \bar{x} \mid n^T A^{-1} \bar{x} = c + n^T A^{-1} b \} \\ &= \left\{ \bar{x} \mid \underbrace{\left( (A^{-1})^T n \right)^T}_{\text{Normalvektor der Ebene } T(E)} \bar{x} = \bar{c} \right\} \end{aligned}$$

Bei einer Affinen Transformation mit  $3 \times 3$ -Transformationsmatrix  $A$  muss man Normalvektoren nicht mit  $A$ , sondern von links nach rechts mit  $(A^{-1})^T$  multiplizieren und anschließend *normieren*. (Vektoren als Spalten betrachten)

Falls  $A$  orthogonal ist, dann ist  $A^T = A^{-1}$ ,  $(A^{-1})^T = A$





# 7 Texturen

(Abbildungen folgen später)

Aufbringen eines Musters auf eine Fläche.

Das Muster – die Textur – ist als zweidimensionales *Bild* gegeben (Pixelarray), und eine Abbildung von diesem Bild auf eine Fläche im Raum.

Die Bildfläche muss nicht unbedingt flach sein:

1. gekrümmt
2. Dreiecksgitter

Dann muss man das Bild in ein dreiecksgitter mit derselben kombinatorischen Struktur zerlegen. Die Abbildung erfolgt dann durch lineare Interpolation auf jedem Dreieck.

Bei beiden Lösungen existiert das Problem der Verzerrung.

Einfaches Modell für Architektur: Fassade wird auf die Hasuwand aufgeklebt.

**Problem** Schatten sind möglicherweise nicht richtig.

Bei der Darstellung benötigt man die umkehrabbildung von der Fläche in Weltkoordinaten (bzw. von der projizierten Fläche in NDC) auf die Textur.

Bei der linearen Interpolation von *Dreiecken* verwendet man gerne *baryzentrische Koordinaten*

$$(\lambda_1, \lambda_2, \lambda_3) \text{ mit } \lambda_1 + \lambda_2 + \lambda_3 = 1$$

stellt den Punkt  $\lambda_1 A_1 + \lambda_2 A_2 + \lambda_3 A_3$  dar.

$$\lambda_1, \lambda_2, \lambda_3 \geq 0 \Leftrightarrow \text{Punkt im Dreieck}$$

Die Ecken des Dreiecks auf dem Objekt haben barizentrische Koordinaten

$$(\lambda_1, \lambda_2, \lambda_3) = (1, 0, 0), (0, 1, 0), (0, 0, 1)$$

dazwischen wird linear *interpoliert* (in Weltkoordinaten). Dann nimmt man in der Textur den Punkt an der Stelle  $\lambda_1 B_1 + \lambda_2 B_2 + \lambda_3 B_3$ .

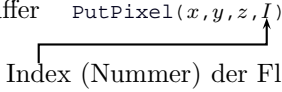
Mögliche Probleme dadurch, das Auflösung des erzeugten bilder nicht mit der Auflösung der Textur übereinstimmt.

- a) Auflösung der Textur zu klein  
⇒ Bild wirkt grob gepixelt (mögliche Lösung: Interpolation)
- b) Auflösung der Textur ist größer  
⇒ Es kann zu Aliasing-Effekten kommen. (mögliche Auswege s. Abschnitt 4.4)



## 8 Schatten

Für die Berechnung von Schatten kann man die gleichen Methoden verwenden, wie zur Bestimmung sichtbarer Flächen:

- von der Lichtquelle aus „sichtbar“  $\Rightarrow$  beleuchtet.
  - von der Lichtquelle aus „verdeckt“  $\Rightarrow$  Schatten.
1. Berechne (für jede Lichtquelle unabhängig) in einem getrennten Puffer das „Bild“, dass von der Lichtquelle aus „gesehen“ wird.  
z. B. mit dem Tiefenpuffer `PutPixel( $x, y, z, I$ )`  
  
Index (Nummer) der Fläche, die gerade dargestellt wird.
  2. Erzeuge das Bild, dass vom Auge aus sichtbar ist Pixel für Pixel.

$P$  sei der Punkt (in Weltkoordinaten), der gerade dargestellt wird. Er gehört zum Objekt mit Nummer  $n$ . Bestimme das Bild von  $P$  im Bildpuffer, der zur Lichtquelle gehört und vergleiche den Index, der dort gespeichert ist.

Wenn  $= n \Rightarrow$  Punkt  $P$  ist von dieser Lichtquelle beleuchtet.



# 9 Clipping

Eine Strecke auf ein Rechteck zurechtstutzen.

## 9.1 Algorithmus von Cohen-Sutherland

$$\mathbb{R}^2 \rightarrow \{0, 1\}^4 \quad (4 \text{ Bits})$$

$$x < x_{\min}$$

$$x > x_{\max}$$

$$y < y_{\min}$$

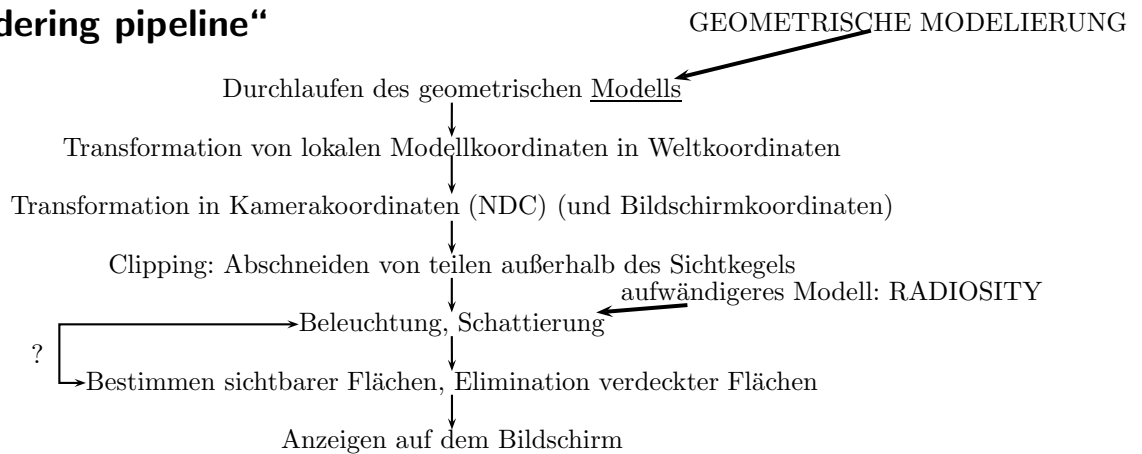
$$y > y_{\max}$$

1. Wenn  $A_1$  und  $A_2$  ein gemeinsames 1-Bit haben.  
 $\Rightarrow$  Strecke  $A_1A_2$  schneidet Rechteck nicht.
2. Wenn beide Codes 0000 sind.  
 $\Rightarrow$  Strecke liegt zur Gänze im Inneren.
3. Bestimme eine Stelle, wo sich beide Codes unterscheiden,
4. Schneide die Gerade mit der entsprechenden horizontalen oder vertikalen Geraden.
5. Ersetze den Endpunkt mit dem 1-Bit durch den Schnittpunkt.
6. Wiederhole



# 10 Geometrische Modellierung

## 10.1 „Rendering pipeline“



nur eine typische Reihenfolge!

Alle diese Operationen werden heute von Hardwareausgeführt

**Alternative** für Beleuchtung, Schattierung und Bestimmung sichtbarer Flächen: Raytracing (Strahlverfolgung)

## 10.2 Geometrische Modellierung

### 10.2.1 Wie kann man geometrische Formen darstellen?

**Möglichkeit 1)** *geometrische Grundformen* (Kugel, Zylinder, Polygone) werden zusammengesetzt

Grundmodell: triangulierte Flächen.

Mit triangulierten Flächen lässt sich im Prinzip alles genügend fein approximieren

(flache Polygone mit mehr als 3 Ecken sind in der Regel auch zugelassen)

**Möglichkeit 2)** *Implizite Flächen* Fläche ist durch eine Gleichung gegeben

$$(x^2 + y^2)^2 + 3z^3xy - 28xz^2 = 0$$

**Möglichkeit 3)** parametrische Fläche

z. B. Kugel:

$$\begin{aligned} x &= \cos \varphi \cos \vartheta \\ y &= \sin \varphi \cos \vartheta \\ z &= \sin \vartheta \\ 0 &\leq \varphi \leq 2\pi & -\pi &\leq \vartheta \leq \pi \end{aligned}$$

Operation	Dreiecksgitter	implizite Fläche	parametrische Fläche
Erzeugen eines Punktes auf der Fläche	sehr leicht	schwierig	sehr leicht
Durchlaufen aller Punkte der Fläche	sehr leicht	sehr schwierig	leicht
Schnitt der Fläche mit einem Sichtstrahl (z. B. beim Raytracing)	durchschnittlich <sup>1</sup> (Modell durchlaufen)	leicht (Lösen eines Gleichungssystems in einer Variablen)	sehr schwierig (Gleichungssystem mit 3 Variablen)
geometrische Manipulationen	sehr leicht	schwierig	durchschnittlich
Speicherverbrauch	ist abhängig von der Genauigkeit	kompakt	kompakt

Umwandlung von impliziten oder parametrischen Flächen in ein Dreiecksgitter: „Gittererzeugung“ („Meshing“)

### 10.2.2 Darstellung polyedrischer Flächen bzw. Polyongittern

1. **primitiv:** Liste von Dreiecken, Vierecken, etc.

$$\begin{aligned}
 D_1 &: (x_1, y_1, z_1), (x_2, y_2, z_2), (x_3, y_3, z_3) \\
 D_2 &: (x_4, y_4, z_4), (x_5, y_5, z_5), (x_6, y_6, z_6) \\
 &\vdots
 \end{aligned}$$

**Problem:** in einem typischen Dreiecksgitter kommen die gleichen Punkte mehrfach vor. Es ist Verschwendung, die gleichen Daten mehrfach zu speichern bzw. zu übertragen.

2. Trennung der Koordinaten von der kombinatorischen Struktur:

$$\begin{array}{ll}
 P_1 : (x_1, y_1, z_1) & D_1 : 1, 2, 4 \\
 P_2 : (x_2, y_2, z_2) & D_2 : 2, 3, 4 \\
 \vdots & \vdots
 \end{array}$$

Flächen sind typischerweise *orientiert*: Dreieck  $P_1, P_2, P_3$  und Dreieck  $P_1, P_3, P_2$  sind Vorder- und Rückseite des gleichen Dreiecks im Raum, um sie können unterschiedliche Materialeigenschaften haben.

Um noch mehr zu sparen, werden Dreiecke zu alternierenden Dreiecksstreifen bzw. Dreiecksfächern zusammengefasst. (und Vierecke zu Vierecksstreifen)

Für  $n$  Dreiecke benötigt man nur  $n + 2$  Punkte. Für isolierte Dreiecke werden  $3n$  Punkte benötigt.

#### Datenstruktur zum Speichern der kombinatorischen Struktur einer polyedrischen Fläche

- Doubly-connected edge list (*DECL*, doppelt-verkettete Kantenliste)
- Quad-edge data structure
- „winged-edge“ data structure

Jede Kante wird durch zwei *Halbkanten* repräsentiert, eine in jede Richtung

Jede Halbkante hat 3 Zeiger:

- $l(e), r(e), \dots$  linke und rechte Nachbarkante mit demselben Startknoten

<sup>1</sup>weder leicht noch schwer



- $u(e)$ ... Umkehrkante

$e$	$r(e)$	$l(e)$	$u(e)$
1	2	3	4
2	$\emptyset$	1	5
3	1	$\emptyset$	6
4	7	8	1
$\vdots$	$\vdots$	$\vdots$	$\vdots$

- Jede Kante liegt an bis zu 23 Flächen an, eine links und eine rechts.
- Durchlaufen aller Kanten, die von einem Knoten ausgehen
- Wir benötigen eine erste Kante  $e_1$

$$\begin{aligned} e_2 &= r(e_1) \\ e_3 &= r(e_2) \\ &\vdots \end{aligned}$$

bis  $e_i = e_1$  oder  $e_i = \emptyset$

Durchlaufen aller Kanten einer Fläche, die rechts von  $e_1$  liegt.

$$\begin{aligned} e_2 &= l(u(e_1)) \\ e_3 &= l(u(e_2)) \\ &\vdots \end{aligned}$$

bis  $e_i = e_1$

$$\begin{aligned} l(r(e)) &= e && \text{falls } r(e) \neq 0 \\ r(l(e)) &= e && \text{falls } l(e) \neq 0 \\ u(u(e)) &= e \end{aligned}$$

## 10.3 Darstellung von geometrischen Modellen in Grafikbibliotheken

typische Funktionsaufrufe in einer Grafikbibliothek wie OpenGL

```
startPolygon();
  setzeDiffuseFarbe(r,g,b);
  // ... weitere Materialeigenschaften
  Knoten(x1,y1,z1);
  Knoten(x2,y2,z2);
  Knoten(x3,y3,z3);
endPolygon();
startPolygon();
  Knoten(x1,y1,z1);
  Knoten(x6,y6,z6);
  Knoten(x5,y5,z5);
  Knoten(x3,y3,z3);
  Knoten(x2,y2,z2);
endPolygon();
// ...
```

Modellkoordinaten  $\rightarrow$  Weltkoordinaten  $\rightarrow$  ...

CT = „current transformation matrix“  
aktuelle Transformationsmatrix

**typischer Ablauf:**

- Am Anfang wird auf CT die Identität initialisiert.
- eine Folge von geometrischen Transformationen; die entsprechenden Transformationen werden *von rechts* an CT dranmultipliziert.
- Dazwischen werden die aktuellen Werte von CT auf einem Stapel gespeichert (Push) bzw. wiederhergestellt (Pop)

```

setze af Einheitsmatrix           // CT = I
  multiply(T3)                   // CT = T3
    multiply(T2)                 // CT = T3 · T2
      Beschreibe Turm
        push
          multiply(T1)           // CT = T3 · T2 · T1
            beschreibe Zinne...
          pop
        push
          multiply(T1)           // CT = T3 · T2 · T1'
            beschreibe Zinne...
          pop
        :
      :
    :
  :
:

```

## 10.4 Constructive Solid Geometry (CSG)

### 10.4.1 Grundlagen

Objekte werden als gefülltes Volumen betrachten, nicht als Randflächen.

- **Grundkörper:** Würfel, Zylinder; konvexe Polyeder
- **Boolesche Operationen:**  $\cup$ ,  $\cap$ ,  $\setminus$  (eigentlich Mengenoperationen) z. B:

Man möchte nur reguläre Mengen betrachten. Eine Menge  $K$  ist regulär, wenn sie der Abschluss ihres Inneren ist:

$$K = \overline{K^\circ}$$

**zur Erinnerung:** Das Innere der Menge  $A$  ist definiert mit

$$A^\circ = \{x \in A \mid \exists \varepsilon > 0 : \text{Kreis mit Radius } \varepsilon \text{ um } x \subseteq A\}$$

Der Abschluss von  $A$  ist definiert mit

$$\overline{A} = \mathbb{R}^2 \setminus (\mathbb{R}^2 - A)^\circ$$

(bzw. mit  $\mathbb{R}^k$  im Raum)

Der Rand von  $A$  ist definiert mit

$$\partial A := \overline{A} - A^\circ$$

$\Rightarrow$  regularisierte Boolesche Operationen

$$A \cup^* B := \overline{(A \cup B)^\circ}$$

$$A \cap^* B := \overline{(A \cap B)^\circ}$$

$$A \setminus^* B := \overline{(A \setminus B)^\circ}$$

### 10.4.2 Darstellung in der Computergrafik

$$A \cup^* [(B \cap^* C) \setminus^* (D \cap^* E)]$$

Umwandlung in eine explizite Darstellung des Randes ist aufwändig.

Man muss Schnitte zwischen den Grundobjekten bestimmen. Anschließend muss man berechnen, welche der ausgeschnittenen Oberflächenstücke den Rand des Ergebnisses bilden.

**Schnitt mit einer Geraden** (z. B. einem Sichtstrahl beim Raytracing) ist relativ einfach:

- Schneide den Strahl mit allen Grundbausteinen. sortiere die Schnittpunkte, bestimme den ersten Schnittpunkt, wo der Strahl in den Körper eintritt.



# 11 Strahlverfolgung (Raytracing)

gut geeignet für spiegelnde und transparente Körper.

A) „**Forward raytracing**“ Schicke Lichtstrahlen (Lichtteilchen) von der Lichtquelle aus und verfolge ihren Weg.

Die allermeisten Teilchen werden irgendwo absorbiert, bevor sie ins Auge gelangen.

⇒ extrem aufwändig.

B) „**Backward raytracing**“

1. Bestimme Helligkeit mit einem normalen Beleuchtungsmodell

2. Verfolge Sichtstrahlen vom Auge ausgehend zurück, solange sie nur spiegelnde oder transparenter Flächen treffen.

Bei Spiegelung kann man entweder den Strahl vor dem Auftreffen (Auge) oder nach dem Auftreffen (Objekt) an der Spiegelungsebene spiegeln → aus dem geknickten Sichtstrahl wird eine Gerade.

## 11.1 Gesetzmäßigkeiten

### 11.1.1 Reflexion

$\alpha = \beta$ : Einfallswinkel = Ausfallswinkel

### 11.1.2 Brechung

Übergang zwischen zwei Medien  $M_1$  und  $M_2$ , die verschiedene optische Dichte  $D_1$  bzw.  $D_2$  haben.

$$\frac{\cos \alpha}{\cos \beta} = \underbrace{\frac{D_2}{D_1}}_{\text{fest für 2 gegebene Materialien}}$$

Die Brechung tritt fast nie in Reinform auf. Es tritt meist immer Reflexion gleichzeitig auf

Wenn  $D_1 > D_2$  ist, dann kann in der Formel  $\cos \beta > 1$  herauskommen.

→ Dann wird der Strahl reflektiert (Totalreflexion)

### 11.1.3 Zusammenfassung

Ein Lichtstrahl trifft auf eine Fläche

- ein Teil wird diffus reflektiert
- ein Teil wird glänzend reflektiert
- ein Teil wird gespiegelt
- ein Teil wird (bei einem transparentem Stoff) durchgelassen und dabei gebeugt
- der Rest wird absorbiert.

### 11.1.4 Einheiten

In welchen Einheiten wird ein „Lichtstrahl“ gemessen? (Die bisherige „Intensität“ ist keine physikalische Einheit.)

**Lichtstrom** Energie an sichtbarem Licht pro Zeit

$$\text{Einheit} = 1 \text{ Lumen} = 1 \text{ lm}$$

Die analoge Einheit bei der Strahlungsmessung = 1 W, Lumen ist eingeschränkt auf sichtbares Licht, bezogen auf die Helligkeitsempfindlichkeit des Auges, abhängig von der Wellenlänge.

**Lichtstärke** Wieviel Licht strahlt eine punktförmige Lichtquelle in eine bestimmte Richtung aus?

$$\text{Lichtstärke} = \frac{\text{Lichtstrom}}{\text{Raumwinkel}}$$

$$\text{Einheit} = 1 \text{ Candela} = 1 \text{ cd}$$

Eine Lichtquelle, die mit 1 cd in alle Richtungen strahlt, hat einen Lichtstrom von  $4\pi$  lm.

[Raumwinkel: Der Raumwinkel, unter dem man ein Objekt sieht, *ist* die Fläche der Projektion auf die Einheitskugel (analog zum Ebenenwinkel)]

**Beleuchtungsstärke** Auf eine Fläche einfallender Lichtstrom, pro Fläche

$$\text{Einheit} = 1 \text{ Lux} = 1 \text{ lx} = 1 \frac{\text{lm}}{\text{m}^2}$$

Bei der Planung von Beleuchtungsanlagen wichtig

**Leuchtdichte** (engl. luminance) Wieviel Licht wird von einer ausgedehnten Lichtquelle pro Flächeneinheit in eine bestimmte Richtung abgestrahlt

$$\frac{\text{Lichtstärke}}{\text{Fläche}} = \frac{\text{Lichtstrom}}{\text{Raumwinkel} \cdot \text{Fläche}} = \frac{\text{Beleuchtungsstärke}}{\text{Raumwinkel}}$$

Bild

$$\text{Einheit} = 1 \frac{\text{cd}}{\text{m}^2}$$

- Die Leuchtdichte ist für den visuellen Helligkeitseindruck maßgeblich.
- Die Leuchtdichte bleibt entlang eines Sichtstrahls konstant (sofern das Licht nicht durch Rauch, etc. abgeschwächt wird.)
- Bei den elektromagnetischen Strahlen ist die äquivalente Einheit die Strahlungsdichte

### 11.1.5 Diffuse Reflexion

Bei der diffusen Reflexion gilt das LAMBERT'sche Gesetz:

Die Leuchtdichte, die in eine bestimmte Richtung geschickt wird, die einen Winkel  $\alpha$  mit der Normalen bildet ist proportional zu  $\cos \alpha$ .

Bei der Betrachtung von der Seite wird der gleiche Raumwinkel aus der Sicht des Betrachters von einer größeren Fläche überdeckt (s. dazu auch Abschnitt 6.1, S. 44).

$$\begin{aligned}
\alpha &= 0^\circ & 2 \tan \beta & \\
& & \cos \alpha (\tan(\alpha + \beta) - \tan(\alpha - \beta)) & \\
\lim_{\beta \rightarrow 0} \frac{\cos \alpha \tan(\alpha + \beta) - \tan(\alpha - \beta)}{2 \tan \beta} &= \cos \alpha \lim_{\beta \rightarrow 0} \frac{\frac{1}{\cos(\alpha + \beta)} - \frac{-1}{\cos(\alpha - \beta)}}{\frac{2}{\cos^2 \beta}} \\
&= \cos \alpha \frac{\frac{2}{\cos(\alpha)}}{\frac{2}{\cos^2 \beta}} = \cos \alpha
\end{aligned}$$

Dieser effekt gleicht sich mit dem Lambert'schen Gesetz aus. Daher erscheint eine diffuse Fläche aus allen Richtungen *gleich hell*.

## 11.2 Einsatz der physikalischen Grundlagen beim Raytracing

Beim Raytracing wird der gespiegelte Strahl (oder gebrochene Strahl) zurückverfolgt, bis er auf eine diffuse Fläche trifft.

Rekursives Raytracing an jedem Punkt, wo der Strahl auftrifft, werden die verschiedenen Anteile (diffus, gespiegelt, gebrochen) aufgesammelt.

```

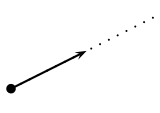
function trace(Strahl r; Faktor f; Schnitttiefe t)
  // Strahl = 1 Punkt und eine Richtung
  berechne den Punkt P, wo der Strahl das Objekt trifft.
  if P = ∅ return (Hintergrundhelligkeit)
  f := f · e-l·D // l = Laenge der Strecke bis zu P
                                     // D = Daempfungsfaktor (fuer Nebel / Rauch)
                                     // D = 0 ... klare Luft, Dser Schritt ist optional

  sum := 0
  fuer jede Lichtquelle Q:
    Betrachte die Strecke PQ
    Wenn frei von Hindernissen;
    berechne Helligkeit nach dem Phong-Modell
    sum := sum + Helligkeit · f
  spiegele den Strahl an der Oberflaeche → Strahl r';
  sum := sum + trace(r', f · Spiegelungskoeffizient, t + 1);
                                     1

  falls transparent:
    brich den Strahl an der Flaeche → Strahl r
    sum := sum + trace(r', Brechungskoeffizient · f, t + 1);
                                     2

  return sum

```



## 11.3 Globale Beleuchtung

### 11.3.1 Rendering-Gleichung

Die **Rendering-Gleichung** bildet die tatsächlichen Verhältnisse bei der Beleuchtung am genauesten ab.

#### Variablen

- $p, p', p'', p'''$ ... Punkte auf den Oberflächen der Objekte
- $\overline{pp'}$ ... Sehstrahlen.
- $f(p', p'')$ ... Leuchtdichte auf dem Strahl  $p'p''$

#### Gegeben

- $\rho(p', p'', p''')$ ... Welchen Anteil des Lichtes auf  $p'p''$  wird in Richtung  $p''p'''$  zurückgeworfen?

<sup>1</sup>hängt von der Geometrie und der Oberflächenbeschaffenheit von  $P$  ab

<sup>2</sup>Welcher Anteil des Lichts, das aus Richtung  $r'$  kommt wird in Richtung  $r$  weitergeleitet

**Beispiel** perfekter Spiegel:

$$\rho(p', p'', p''') = \begin{cases} 1, & \text{wenn die Winkelhalbierende von } p'p''p''' \text{ senkrecht auf der Fläche steht} \\ 0, & \text{sonst} \end{cases}$$

**Rendering-Gleichung**

$$f(p'', p''') = \int_{p': p'p'' \text{ ist sichtbar}} f(p', p'') \rho(p', p'', p''') \underbrace{dp'}_{\text{Flächenintegral}} + \underbrace{e(p'', p''')}_{\substack{\text{Wass wird von} \\ p'' \text{ als Lichtquelle} \\ \text{in Richtung } p''' \\ \text{ausgestrahlt?}}}$$

Integralgleichung.  $f$  ist eine Funktion von 4 Variablen

$$f(p'', p''', \lambda) = \int \dots \rho(p', p'', p''', \lambda) \dots, \lambda \in \mathbb{R} \text{ bzw. } \lambda \in \{R, G, B\}$$

$\Rightarrow$  In wirklichkeit auch von  $\lambda$  abhängig.

### 11.3.2 Radiosity-Verfahren

Vereinfachung der Rendering-Gleichung

- Betrachtet alle Wechselwirkungen zwischen Flächen
- nur diffuse Flächen
- Die Flächen werden in kleine Flächenstücke zerlegt; auf die Art wird das Problem diskreditiert  
 $\rightarrow$  großes lineares Gleichungssystem

**Formfaktor  $f_{ij}$ :** Welcher Anteil des Lichtstroms, der von Fläche  $A_i$  ausgesendet wird, kommt bei der Fläche  $A_j$  an?

**Variablen**

- $b_i$ ... Beleuchtungsstärke der Fläche  $A_i$ . Wieviel Lichtstrom wird pro Flächeneinheit ausgeschickt?
- $\rho_i$ ... Reflektionskoeffizient der Fläche  $A_i$
- $e_i$ ... Eigenstrahlungslichtstärke von  $A_i$ , wenn  $A_i$  eine Lichtquelle ist.

Summe des eintreffenden Lichtes auf  $A_i$ :

$$\sum_j b_j \cdot |A_j| \cdot f_{ji}$$

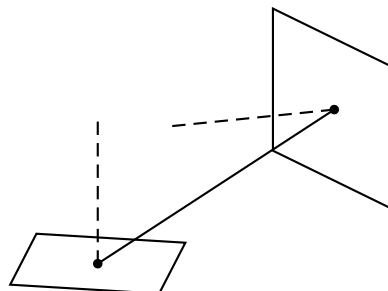
Ausgeschicktes Licht auf  $A_i$ :

$$b_i \cdot |A_i| = \rho_i \cdot \sum_j b_j \cdot |A_j| \cdot f_{ji} + e_i \cdot |A_i|$$

Dies muss man für  $R, G, B$  bzw. für jede Wellenlänge  $\lambda$  separat lösen.

**Formfaktoren**

Hängen von der Geometrie ab.



$$\begin{aligned} p &\in A_i, q \in A_j \\ \alpha, \beta &\text{ Winkel zu den Flächennormalen} \\ r &= \|p - q\| \end{aligned}$$



**Annahme**  $A_i$  und  $A_j$  relativ klein, Sicht ist nicht unterbrochen ...  $r, \alpha, \beta$  schwanken nur wenig.

**Näherungsformel** (Fehler gering wenn  $p$  und  $q$  weit aus einander)

$$f_{ij} \approx \cos \alpha \cdot \cos \beta \cdot \frac{1}{r^2} \cdot |A_j| \cdot \frac{1}{\pi}$$

**Exakte Formel**

$$f_{ij} = \int_{p \in A_i} \int_{q \in A_j} \frac{\cos \alpha(p, q) \cdot \cos \beta(p, q) \cdot s(p, q)}{\|p - q\|^2} dq dp \cdot \frac{1}{|A_i|} \cdot \frac{1}{\pi}$$

$$s(p, q) = \begin{cases} 1, & p \text{ und } q \text{ sehen sich} \\ 0, & \text{sonst} \end{cases}$$

$$\boxed{f_{ij} \cdot |A_i| = f_{ji} \cdot |A_j|}$$

$$b_i \cdot |A_i| = \rho_i \cdot \sum_j b_j \cdot |A_j| \cdot f_{ji} + e_i \cdot |A_i|$$

$$b_i \cdot \cancel{|A_i|} = \rho_i \cdot \sum_j b_j \cdot \cancel{|A_i|} \cdot f_{ij} + e_i \cdot \cancel{|A_i|}$$

$$\Rightarrow \boxed{b_i = \rho_i \sum_j b_j \cdot f_{ij} + e_i} \quad \text{Radiosity-Gleichungssystem}$$

lineares Gleichungssystem in  $n$  Variablen  $b_1, \dots, b_n$  ( $n = \# \text{Flächenstücke}$ ),  $O(n^2)$  Koeffizienten  $f_{ij}$   $0 \leq \rho_i \leq 1$

**Geometrische „Berechnung“ von  $f_{ij}$**

$A_i$  ist ein kleines Flächenstück

Projektion auf die obere Einheitshalbkugel um  $A_i$

$\hat{=}$  Multiplikation mit  $\cos \beta \cdot \frac{1}{r^2}$

$\hat{=}$  scheinbare Größe von  $A_j$  aus Sicht von  $A_i$

Anschließende Projektion senkrecht auf die Ebene durch  $A_i \hat{=}$  Multiplikation mit  $\cos \alpha$

gesamter Fläche des Kreises =  $\pi$

$$\sum_i f_{ij} \stackrel{!}{=} 1$$

Wahl des Konstanten Faktors  $\frac{1}{\pi}$  wird durch die physikalische Forderung  $\sum_j = 1$  erzwungen.

Das Gleichungssystem löst man am besten iterativ:

- Beginne mit einer beliebigen Ausgangslösung  $\vec{b}^{(0)}$  z. B. durch Phong-Beleuchtung berechnet. oder

$$\vec{b}^{(0)} = \begin{pmatrix} b_1^{(0)} \\ \vdots \\ b_n^{(0)} \end{pmatrix} \cdot \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}$$

- Setze  $b^{(k)}$  rechts ein und erhalte die nächste Näherungslösung

$$\boxed{b_i^{(k+1)} = \sum_j b_j^{(k)} \cdot f_{ij} + e_i}, \quad i = 1, \dots, n$$

Das konvergiert umso schneller, je kleiner die  $\rho_i$ -Werte sind.

$$x = Ax \qquad A \in \mathbb{R}^{n \times n} \qquad b \in \mathbb{R}^n \qquad x \in \mathbb{R}^n$$

$$x^{(0)}, x^{(1)} = Ax^{(0)} + b, x^{(2)} = Ax^{(1)} + b$$

GAUSS-SEIDEL-Verfahren (iteratives Verfahren zur Lösung des Gleichungssystems). Konvergiert unter gewissen Bedingungen. z. B. wenn die Summe der Absolutbeträge in jeder Zeile von  $A < 1$  ist.

Bei der Radiosity-Gleichung ist dies fast der Fall:  $\leq 1$ .

Falls alle  $\rho_i < 1$  sind:  $< 1$

**Zusammenfassung**

1. Flächen in kleine Stücke zerlegen

- a) gleichförmiges Gitter

- b) angepasst an den erwarteten Lichtaustausch.

Wo Flächen nah an anderen Flächen sind wird die Zerlegung feiner.

- c) adaptiv.

Man rechnet zuerst mit einer gröberen Zerlegung und verfeinert sie dort, wo das Ergebnis stark schwankt

2. Formfaktoren bestimmen (am zeitaufwändigsten)

Können auch mit Hilfe von Grafikpuffern berechnet werden.

$$\int_{p \in A_i} \int_{q \in A_j} \dots$$

Für ein festes  $p$  ähnelt die Berechnung des Integranden der Grafikpufferberechnung mit Kamera  $p$ . Diese Berechnung wird von Grafik-Hardware unterstützt

3. Gleichungssystem lösen

4. mit den so bestimmten (diffusen) Helligkeiten kann man dann die Szene darstellen

# 12 Transparenz

teildurchsichtige Medien. z. B. Glas mit einem Bild, mit Staub, Nebel, Wolken, Rauch.

## 12.1 RGBA-Modell bzw. $(r, g, b, \alpha)$

Ein halbdurchsichtiges Bild wird durch eine vierte Größe  $\alpha$  (zusätzlich zu  $R, G, B$ ) für jeden Bildpunkt

$\alpha = 1$	vollständig undurchsichtig
$\alpha = 0$	vollständig durchsichtig
$\alpha = 0,3$	30% wird von diesem Bild beige-steuert, 70% kommt von dem was dahinter liegt

z. B. Bild eines Mauszeigers (idealisiert)

in Wirklichkeit natürlich „pixelig“, Übergänge zwischen den kanten mit Antialiasing geglättet:

$$\left. \begin{array}{l} (r, g, b, \alpha) \\ (\bar{r}, \bar{g}, \bar{b}) \text{ (Hintergrund)} \end{array} \right\} \text{Ergebnis} = (\alpha \cdot r + (1-\alpha) \cdot \bar{r}, \alpha \cdot g + (1-\alpha) \cdot \bar{g}, \alpha \cdot b + (1-\alpha) \cdot \bar{b})$$

$$\text{Ergebnis} = \alpha^1 rgb^1 + (1-\alpha^1) \alpha^2 rgb^2 + (1-\alpha^1)(1-\alpha^2) \alpha^3 rgb^3 + (1-\alpha^1)(1-\alpha^2)(1-\alpha^3) \underbrace{rgb}_{\text{Vektor aus 3 Komponenten}}$$

← mehrere Transparente Schichten

$$\left( \underbrace{\frac{\alpha^1 rgb^1 + (1-\alpha^1) \alpha^2 rgb^2}{\alpha^1 + (1-\alpha^1) \alpha^2}}_{rgb^{neu}}, \underbrace{\alpha^1 + (1-\alpha^1) \alpha^2}_{\alpha^{neu}} \right)$$

Effekt auf eine dahinter liegende Fläche  $\overline{rgb}$

$$\alpha^{neu} rgb^{neu} + (1-\alpha^{neu}) \overline{rgb}$$

Wichtig: durchlässige Medien in der richtigen Reihenfolge kombinieren (z. B. vorne nach hinten, oder hinten nach vorne)

## 12.2 Kombination mit Tiefenpuffer

$(rgb^1, z^1, \alpha^1)$   $(rgb^2, z^2, \alpha^2)$  einer dieser Werte ist im Tiefenpuffer an einer bestimmten Stelle gespeichert, der andere soll dort hingeschrieben werden

$$\begin{aligned} z^1 &< z^2 \text{ (o. B. d. A.)} \\ rgb^{neu} &:= \frac{\alpha^1 rgb^1 + (1-\alpha^1) \alpha^2 rgb^2}{\alpha^1 + (1-\alpha^1) \alpha^2} \\ \alpha^{neu} &:= \alpha^1 + (1-\alpha^1) \alpha^2 \\ z^{neu} &:= z^1 \end{aligned}$$

(eigentlich unzureichend, aber notwendig, um wenigstens den undurchsichtigen Fall richtig zu behandeln)

Diese Methode funktioniert, wenn man höchstens *ein* halbdurchlässiges Pixel ( $0 < \alpha < 1$ ) zeichnen möchte oder wenn *alle* Objekte von vorne nach hinten oder von hinten nach vorne eingefügt werden.



# 13 Spline-Kurven und -Flächen

z. B. Kreis

implizite Darstellung	explizite Darstellung	Parameterdarstellung
$x^2 + y^2 = 1$	$y = \sqrt{1 - x^2}$ $y = -\sqrt{1 - x^2}$	$y = \cos \alpha$ $x = \sin \alpha$
<hr/>		
	<i>Spezialfall</i>	<i>Alternative (rationale Darstellung)</i>
	$x = t$	$x = \frac{2t}{1+t^2}$ $t = \tan \frac{\alpha}{2}$
	$y = \sqrt{1 - t^2}$	$y = \frac{1-t^2}{1+t^2}$ $-\infty < t < \infty$

- Parameterdarstellung ist gut für das Nachfahren der Kurve; Darstellung als Folge von Punkten
- implizite Darstellung ist gut für Raytracing

**Definition Spline-Kurven** sind parametrische Kurven, wo sich Parameterfunktionen Polynome sind (es gibt auch rationale Splines). Die Kurven sind durch Leitpunkte (Kontrollpunkte) festgelegt. Dadurch sind sie einigermaßen intuitiv manipulierbar.

## 1. Bézier-Splines

$k + 1$  Kontrollpunkte definieren einen Bézier-Spline der Ordnung  $k$  (Polynome vom Grad  $\leq k$ )

## 2. B-Splines

Ordnung  $k$ : stückweise Polynome der Ordnung  $k$ , Kontrollpunkte  $< \infty$

## 3. Hermite-Splines

interpolieren zwischen 2 Endpunkten mit vorgegebenen Tangentenrichtungen und Geschwindigkeiten  
... kubische Splines

## 13.1 Bézierkurven (Bézier-Splines)

- Ordnung =  $d$  = Grad der Polynome
- $d + 1$  Kontrollpunkte  $\vec{P}_0, \vec{P}_1, \dots, \vec{P}_d$
- BERNSTEINpolynome:

$$B_i^d = \binom{d}{i} \cdot t^i \cdot (1 - t)^{d-i} \quad i = 0, 1, \dots, d \quad 0 \leq t \leq 1$$

$$B_0^1(t) = 1 \cdot t^0 (1 - t)^1 = 1 - t$$

$$B_1^1(t) = \binom{1}{1} \cdot t^1 \cdot (1 - t)^0 = t$$

$$B_0^2(t) = \binom{2}{0} \cdot t^0(1-t)^2 = (1-t)^2 = 1 - 2t + t^2$$

$$B_1^2(t) = \binom{2}{1} \cdot t^1 \cdot (1-t) = 2t(1-t) = 2t - 2t^2$$

$$B_2^2(t) = \binom{2}{2} \cdot t^2 \cdot (1-t)^0 = t^2$$

$$B_0^3(t) = (1-t)^3$$

$$B_1^3(t) = \binom{3}{1} \cdot t^1 \cdot (1-t)^2 = 3t(1-t) = 3t(1-t)^2$$

$$B_2^3(t) = \binom{3}{2} \cdot t^2 \cdot (1-t) = 3t^2(1-t) = 3t^2 - 3t^3$$

$$B_3^3(t) = t^3$$

Eigenschaften der Bernsteinpolynome:

$$1. B_i^d(t) \geq 0 \text{ für } 0 \leq t \leq 1$$

$$2. \sum_{i=0}^d B_i^d(t) = 1$$

**Beweis**

$$\sum \binom{d}{i} t^i (1-t)^{d-i} = [t + (1-t)]^d = 1^d = 1$$

3. Rekursion:

$$\boxed{B_i^d(t) = t \cdot B_{i-1}^{d-1}(t) + (1-t)B_i^{d-1}(t)} \quad 1 \leq i \leq d-1 \quad (13.1)$$

$$B_d^d(t) = t \cdot B_{d-1}^{d-1}(t) = t^d B_0^d(t) \quad = t \cdot B_{d-1}^{d-1}(t) + (1-t)B_0^{d-1}(t) = (1-t) \quad (13.2)$$

**Beweis** (von (1))

$$\begin{aligned} & t \cdot \binom{d-1}{i-1} \cdot t^{i-1}(1-t)^{(d-1)-(i-1)} + (1-t) \binom{d-1}{i} \cdot t^i(1-t)^{d-1-i} \\ &= \binom{d-1}{i-1} t^i(1-t)^{d-i} + \binom{d-1}{i} t^i(1-t)^{d-i} = t^i(1-t)^{d-i} \underbrace{\left[ \binom{d-1}{i-1} + \binom{d-1}{i} \right]}_{\binom{d}{i}} \end{aligned}$$

**Bemerkung**  $(B_0^d(t), \dots, B_d^d(t))$  sind die Wahrscheinlichkeiten bei der Binomialverteilung mit Erfolgswahrscheinlichkeit  $p = t$

$B_i^d(p)$  = Wahrscheinlichkeit, dass bei  $d$  Wiederholungen genau  $i$  Erfolge auftreten.

$\Rightarrow$  Splinekurve  $\vec{P}(t)$ :

$$\boxed{\vec{P}(t) = B_0^d \cdot \vec{P}_0 + B_1^d(t) \vec{P}_1 + \dots + B_d^d(t) \cdot \vec{P}_d} \quad 0 \leq t \leq 1$$

$d = 1$ :

$$\vec{P}(t) = (1-t) \cdot \vec{P}_0 + t \vec{P}_1 = \vec{P}_0 + t(\vec{P}_1 - \vec{P}_0)$$

Eigenschaften:

1. Kurve beginnt bei  $P_0$  und endet bei  $P_d$ . Die übrigen Punkte werden im allgemeinen nicht durchlaufen.
2. die Tangentenrichtung am Anfang und am Ende ist  $\overrightarrow{P_0 P_1}$  bzw.  $\overrightarrow{P_{d-1} P_d}$  (Beweis s. Übung, Aufgabe 29).
3. Die Kurve verläuft in der Konvexen Hülle der Kontrollpunkte.

4. Die Punkte  $\vec{P}_0, \dots, \vec{P}_d$  können  $\in \mathbb{R}^2, \in \mathbb{R}^3, \dots, \mathbb{R}^n$  sein. Bézierkurven *im Raum* werden auf die gleiche Art definiert.
5. Verminderung der Variation:

Eine Gerade in der Ebene (eine Ebene im Raum) schneidet die Bézierkurve höchstens so oft wie das Kontrollpolygon.

### 13.1.1 Der Algorithmus von de Casteljau

zur Berechnung von  $P(t)$

$O(d^2)$  Operationen

$$\mathcal{B}^d(t)(P_0, \dots, P_d) := \sum_{i=0}^d B_i^d(t) \cdot P_i$$

**Beweis** des Casteljau-Algorithmus:

$$\begin{aligned} \mathcal{B}^d(t)(P_0, \dots, P_d) &\stackrel{?}{=} (1-t)\mathcal{B}^{d-1}(P_0, \dots, P_{d-1}) + t\mathcal{B}^{d-1}(P_1, \dots, P_d) \\ &= (1-t) \sum_{i=0}^{d-1} B_i^{d-1}(t)P_i + \underbrace{\sum_{i=0}^{d-1} B_i^{d-1}(t)P_{i+1}}_{= \sum_{j=1}^d B_{j-1}^{d-1}(t)P_j} \\ &= (1-t) \sum_{j=0}^{d-1} B_j^{d-1}(t)P_j + \sum_{j=1}^d B_{j-1}^{d-1}(t)P_j && = (*) \text{ verwenden} \end{aligned}$$

#### Algorithmus

**Eingabe**  $P_0, P_1, \dots, P_d = P_0^0, P_1^0, \dots, P_d^0 \quad 0 < t < 1$

**Ausgabe**  $P(t)$

```

for  $l := 1$  to  $d$ 
    for  $i := 0$  to  $d - l$ 
         $P_i^l := (1-t)P_i^{l-1} + tP_{i+1}^{l-1}$ 
return  $P_0^d = P(t)$ 

```

#### Zerlegung von Bézierkurven

- $P_0, P_0^1, P_0^2, \dots, P_0^d$  bilde das Kontrollpolygon für die Kurve  $P(x), 0 \leq x \leq t$
- $P_0^d, P_1^{d-1}, \dots, P_{d-1}^1, P_d^0$  bilden das Kontrollpolygon für die Kurve  $P(x), t \leq x \leq 1$
- häufigster Fall:  $t = \frac{1}{2}$