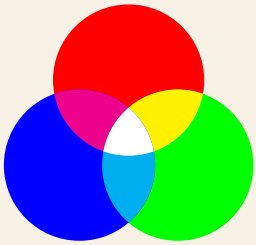
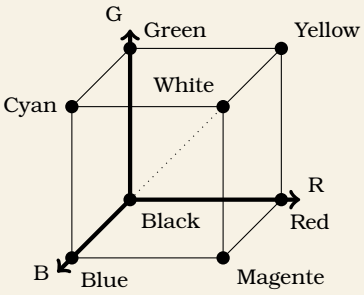
# Farbmodelle

## RGB

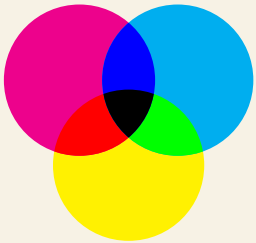
- additiver Farbraum
- Verwendung in Monitoren
- Aufbau: (Rot, Grün, Blau)

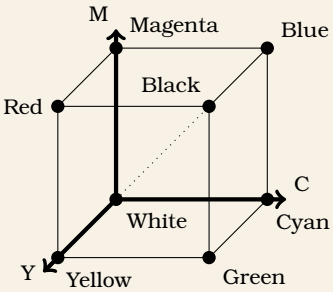





## CMY

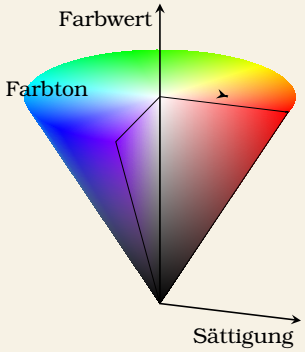
- subtraktiver Farbraum
- Verwendung in Druckern
- Aufbau: (Cyan, Magenta, Gelb)
- CMYK: (Cyan, Magenta, Gelb, Schwarzanteil)
- RGB ⇒ CMY: C = 1-R, M = 1-G, Y = 1-B
- CMY ⇒ CMYK: K = min(CMY), C = C-K, ...






## HSV

- Orientiert an menschlicher Wahrnehmung
- RGB ⇒ HSV möglich
- Verwendung: Farbauswahl in Grafikprogrammen
- Aufbau: (Farbwinkel, Sättigung, Intensität)
- Beispiele: Schwarz(x,0%,0%), Weiß(x,0%,100%), Rot(0°,x%,y%), Grün(120°,x%,y%), Blau(240°,x%,y%)





feste Sättigung

# Transformation

## Grundlagen

Die Affine Transformation ist definiert durch:

$$x \to Ax + t$$

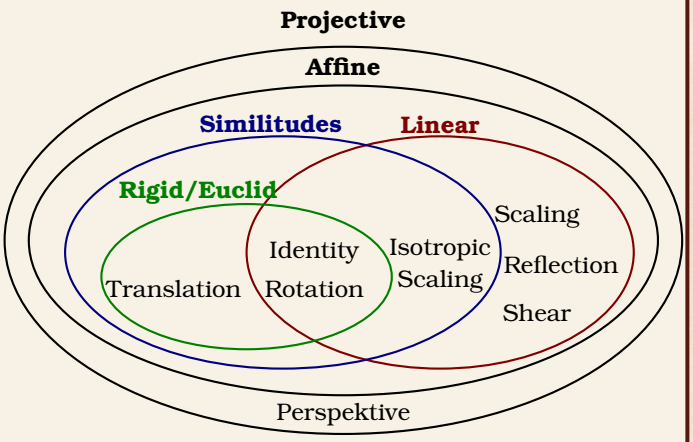
Um dies auf Koordinaten anzuwenden, überführt erstellt man homogene Koordinaten:

$$(x, y) \to (x, y, 1)$$

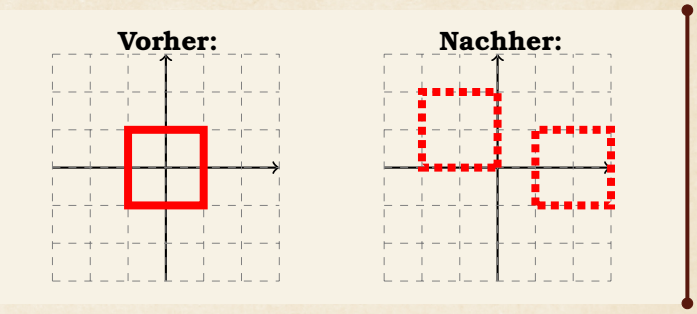
Die affine Transformation wird dann so berechnet:

$$\begin{pmatrix} A_{11} & A_{12} & t_1 \\ A_{21} & A_{22} & t_2 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

## Transformationshierachie




## Transpositionsmatrizen




### Erklärung

Die Einheitsmatrix ist der Ausgangspunkt. In der letzten Spalte werden die additiven Werte zur Verschiebung des Mittelpunktes des Objektes eingetragen. Gleichzeitige Verschiebungen in unterschiedlichen Richtungen ist möglich!

### 2D

Allgemein:

$$\begin{pmatrix} 1 & 0 & \pm X \\ 0 & 1 & \pm Y \\ 0 & 0 & 1 \end{pmatrix}$$

Beispiel x++:

$$\begin{pmatrix} 1 & 0 & 2 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

### 3D

Allgemein:

$$\begin{pmatrix} 1 & 0 & 0 & \pm X \\ 0 & 1 & 0 & \pm Y \\ 0 & 0 & 1 & \pm Z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

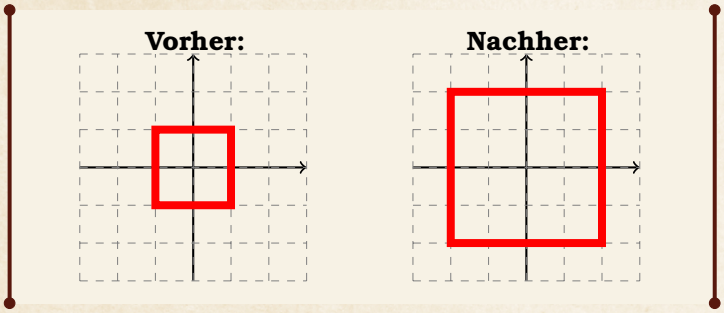
Beispiel:

$$\begin{pmatrix} 1 & 0 & 0 & 3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -2 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



MadebyJulianKotzur

## Scalierungsmatrizen




### Erklärung

Bei der Skalierungsmatrix werden in der Diagonalen die x,y,z-Werte statt der 1 auf den gewünschten Faktor gesetzt. Bei einem Faktor größer 1 wird vergrößert, ansonsten verkleinert. Negative Faktoren sollten nicht verwendet werden. Faktorisierungen für die einzelnen Koordinaten können kombiniert werden!

### 2D

Allgemein:

$$\begin{pmatrix} *X & 0 & 0 \\ 0 & *Y & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Beispiel größer:

$$\begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

### 3D

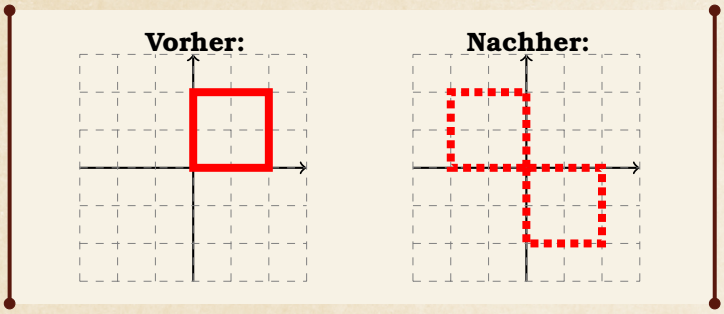
Allgemein:

$$\begin{pmatrix} *X & 0 & 0 & 0 \\ 0 & *Y & 0 & 0 \\ 0 & 0 & *Z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Beispiel kleiner:

$$\begin{pmatrix} 0.5 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

## Reflektionsmatrizen




### Erklärung

Ausgehend von der Eingangsmatrix setzt man die Achse, an der man Spiegeln will, auf $-1$. Es können mehrere Achsenspiegelungen kombiniert werden!

### 2D

Spiegelung X-Achse:

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

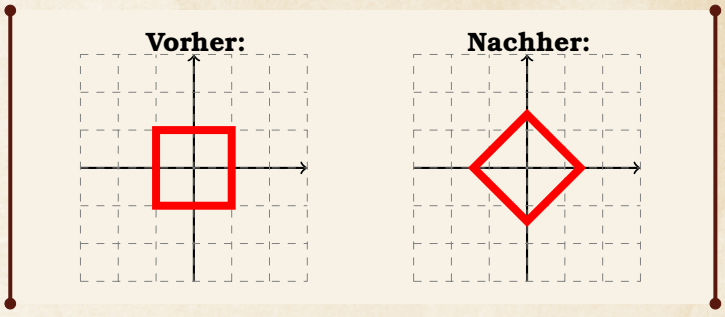
Spiegelung Y-Achse:

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

### 3D

Funktioniert äquivalent wie im zweidimensionalem. Beispiel: Bei der Spiegelung an der XY-Ebene wird $Z = -1$ gesetzt.

## Rotationsmatrizen




### Erklärung

Man benötigt eine speziell definierte Matrix zur Rotation, welche mit sinus und cosinus Werten arbeitet. Bei mehrfacher Rotation in einem Schritt muss der Winkel angepasst werden! (Rotation ist gegen den Uhrzeigersinn!)

### 2D

Allgemein:

$$\begin{pmatrix} \cos(\phi) & -\sin(\phi) & 0 \\ \sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Beispiel 90°:

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

### 3D

Rotation x-Achse:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) & 0 \\ 0 & \sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

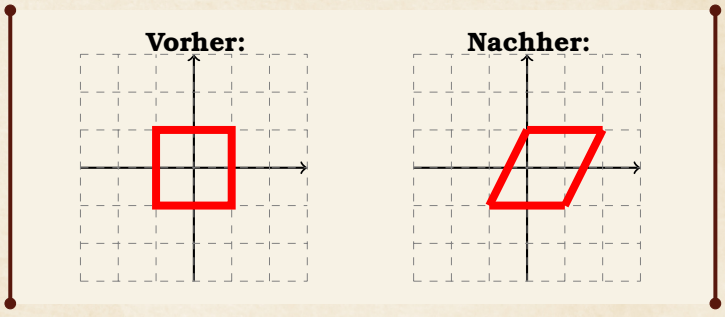
Rotation y-Achse:

$$\begin{pmatrix} \cos(\phi) & 0 & \sin(\phi) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\phi) & 0 & \cos(\phi) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotation z-Achse:

$$\begin{pmatrix} \cos(\phi) & -\sin(\phi) & 0 & 0 \\ \sin(\phi) & \cos(\phi) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

## Scherungsmatrizen




### Erklärung

Man nehme ein Objekt, was am Ursprung liegt, z.b. ein Quadrat oder einen Würfel. Schert man nun in x-Achsen Richtung, dann verschiebt man die y-Achse in x-Richtung. Schert man nun das Quadrat um 0.5 in x-Richtung, so wird aus dem y-Spaltenvektor der Matrix $(0.5, 1, 0)^T$. Möchte man in 3D nun um 1 horizontal in x-Richtung verschieben, ändert man nur den x-Wert des y-Spaltenvektors. Also immer überlegen, welche Achse man in welche Richtung kippt!

## 2D

x-Richtung
Nach rechts/links:

$$\begin{pmatrix} 1 & n & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

y-Richtung
Nach oben/unten:

$$\begin{pmatrix} 1 & 0 & 0 \\ n & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

## 3D

Scherung x-Achse:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ n_y & 1 & 0 & 0 \\ n_z & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Scherung y-Achse:

$$\begin{pmatrix} 1 & n_x & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & n_z & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Scherung z-Achse:

$$\begin{pmatrix} 1 & 0 & n_x & 0 \\ 0 & 1 & n_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

# Weitere 3D Rotationen

## Euler Rotation

- Keine Interpolation
- Sei die Rotationswinkel für x,y,z durch $\alpha, \beta, \gamma$ gegeben, dann ist die Rotationsmatrix:

$$R = R_1(\alpha)R_2(\beta)R_3(\gamma)$$

## Axis and Angle

- Interpolation möglich
- Rotation an der n-ten Koordinate mit Winkel $\alpha$ ist gegeben durch:

$$R = P + (1 - P) \cdot cos(\alpha) + Q \cdot sin(\alpha)$$

## Quaternions

- Interpolation möglich
- Rotation an der n-ten Koordinate mit Winkel $\alpha$ ist gegeben durch:

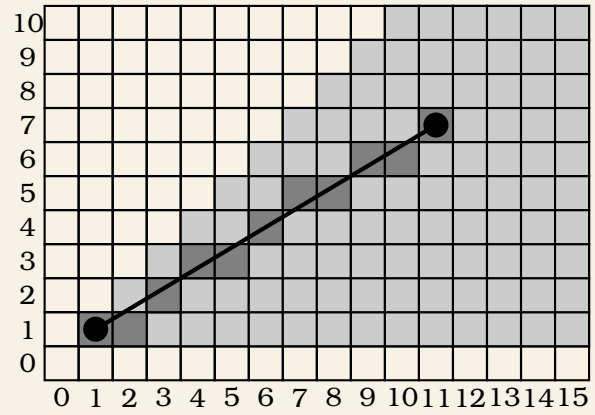
$$q = (cos(\alpha/2), n \cdot sin(\alpha/2))$$

# Anmerkungen

- In eine Transformationmatrix können keine Variablen eingesetzt werden. Demnach ist soetwas wie: $(x, y, z) -> (x^2, y^2, z^2)$ nicht möglich!

# Rasterization

## Line Rasterization



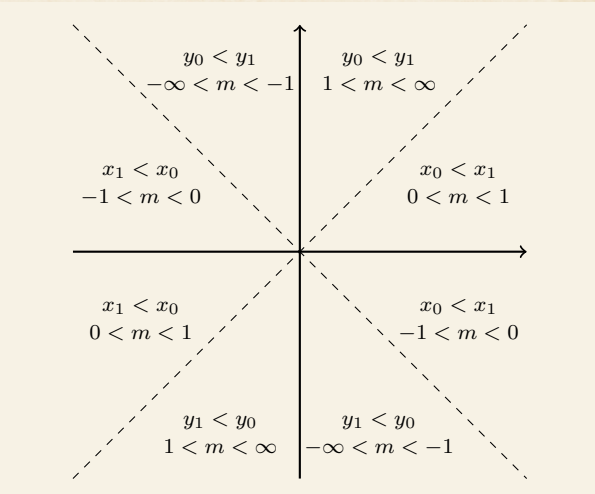

Bruteforce: Linie durchziehen. Pro x-Wert wird das Kästchen genommen, zu dessen Mittelpunkt die Linie am nächsten ist. In Grenzfällen muss in die Geradengleichung eingesetzt werden und y gerundet werden.

### Bresenham Algorithmus

Die Grundlage bildet der folgende einfache Algorithmus, der auf dem gleichen Einschränkungen wie der Brute Force Algorithmus funktioniert.

```
0  x, y = x0, y0;
1  dx, dy = x1 - x0 , y1-y0;
2  D, dDE, dDNE = dx-2*dy, -2*dy, 2*(dx-dy);
3  while (x <= x1) {
4    setPixelS(x,y);
5    x = x+1;
6    if(D < 0) {
7      y = y+1;
8      D = D + dDNE;
9    }else{
10     D = D + dDE;
11 } }
```

Dieser Algorithmus betrachtet noch nicht alle Oktanten:




Um alle Oktanten abzudecken, muss man folgendes am Code hinzufügen:

- Berechne $m = (y1 - y0)/(x1 - x0)$
- Wenn $|m| > 1$: Vertausche x und y, sodass x = y und y = x für alle Startpunkte gilt. Mache setPixel(y,x)
- Wenn $x_0 > x_1$: Tausche Start und Endpunkt
- Wenn $m < 0$: y– statt y++
- Nutze $dx = x_1 - x_0$ und $dy = |y_1 -_y 0|$

Resultierender Code:

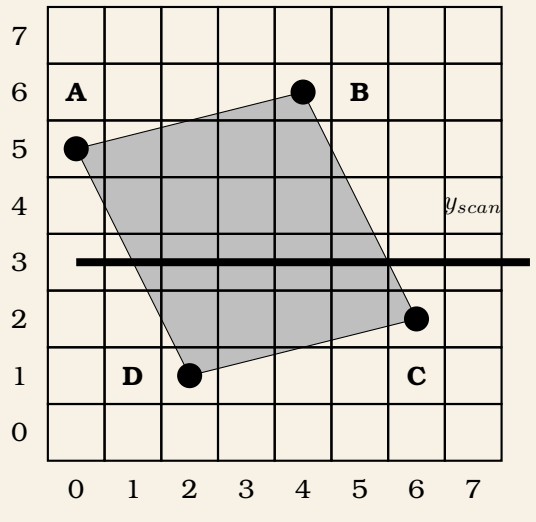
```
0  m = (y1 -y0)/(x1 -x0);
1  if(Math.abs(m)>1){
2    tmp,y0,x0 = y0,x0,tmp;
3    tmp,y1,x1 = y1,x1,tmp;
4  }
5  if(x0 > x1){
6    tmp,x1,x0 = x1,x0,tmp;
7    tmp,y1,y0 = y1,y0,tmp;
8  }
9  step = 1;
10 if(m < 0) step = -1;
11 x, y = x0, y0;
12 dx, dy = x1 - x0 , y1-y0;
13 D, dDE, dDNE = dx-2*dy, -2*dy, 2*(dx-dy);
14 while (x <= x1) {
15   if(Math.abs(m)>1) setPixel(y,x);
16   else setPixelS(x,y);
17   x = x+1;
18   if(D < 0) {
19     y = y+step;
20     D = D + dDNE;
21   }else{
22     D = D + dDE;
23 } }
```

- Für eine Linie der Länge $d = \sqrt{(x1 - x0)^2 + (y1 - y0)^2}$ werden immer gleich viele Pixel, nämlich das größere von $x1 - x0 + 1$ Pixel und $y_1 - y_0 + 1$ Pixel.
- Es werden immer mind. 1 Pixel benötigt (Start = Ende)

### Antialiasing

Statt nur einem Pixel pro Iterationsschritt, werden mehrere Pixel gezeichnet. Dabei wird deren Intensität so angepasst, dass diese Zusammen die stärke von dem einzelnen Pixel ohne Antialiasing haben. Man nennt es auch Kantenglättung.

## Polygon Rasterization - Scanline




### Allgemein

Beim Scanline Algorithmus wird ein Polygon von oben nach unten/ unten nach oben anhand einer Linie, parallel zur x-Achse, abgetastet. Dabei werden die Schnittpunkte mit dem Polygon festgehalten. Anhand der Schnittpunkte werden dann die Pixel des Polynoms berechnet.

$m^{-1}$ bzw. 1/m ist gegeben durch:

$$m = \frac{y_{upper} - y_{lower}}{x_{upper} - x_{lower}}$$

Ist $m = \infty$, dann ist $1/m = 0$ und die Kante ist vertikal. Ich kann damit ganz normal weiter rechnen! Ist $m = 0$, dann ist $1/m = \infty$ und die Kante ist horizontal. Ich kann diese Kante ignorieren!

### Aktive Edge Table

Enthält alle Kanten des Polynoms, die von der Scanline geschnitten werden. Bei Eckpunkten kommen beide dazugehörigen Kanten in die AET. Kommt der Algorithmus an einen neuen Eckpunkt, an dem eine Kante zu Ende ist, wird diese beendete Kante aus dem AET genommen. AET's sind aufsteigend nach $x_{inters}$ sortiert! Beispiel für AET aus yscan = 3 der Grafik:

| *edge* | $x_{inters}$ | $y_{upper}$ | $m^{-1}$ | Next |
|---|---|---|---|---|
| BC | 5.5 | 6 | -0.5 | DA |
| DA | 1 | 5 | -0.5 | NULL |

### Edge Table

- Enthält alle nicht verwendeten Kanten
- Aufsteigend sortiert nach $y_{lower}$
- Beispiel für ET aus yscan = 3 der Grafik:

| *edge* | $y_{lower}$ | $x_{lower}$ | $y_{upper}$ | $m^{-1}$ | Next |
|---|---|---|---|---|---|
| AB | 5 | 0 | 6 | 4 | NULL |

### Ausgewählte Pixel

Der Scanline Algorithmus gibt für jeden y-Wert ein oder mehrere Intervalle von Schnittpunkten aus. Sei $[x_0, x_n]$ ein solches Intervall. Es werden alle x-Werte gesetzt, wenn $x_0$ und $x_n$ ganzzahlig sind. Wenn die Intervallgrenzen nicht ganzzahlig sind, müssen diese beiden Pixel speziell definiert sein. Dafür werden die Intervallgrenzen z.b. immer aufgerundet.

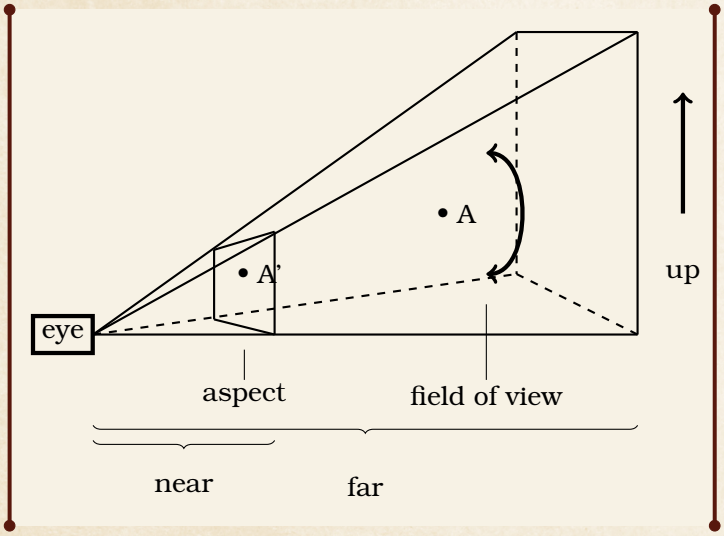
### Gouraud Shading

Lineare Interpolation eines Polynoms entlang der Kanten. Scanline kann Grundlage bilden. Dazu müssen dann noch die Attribute der Pixel (Farbwerte) verwaltet werden und man kann Gouraud Shading betreiben.

### Check this out

https://www.mathematik.uni-marburg.de/~thormae/lectures/graphics1/code_v2/RasterPoly/index.html

# Viewing and Projection




- Aspect Ratio: Breite / Höhe des Aspekts
- Field of View: Winkel des Kegels in Blickrichtung
- Near muss nicht beim Aspekt enden
- Nur Kegelinnenbereich zw. Far ohne Near ist sichtbar

## Viewing Transformation

Bildet Worldspace auf Kameraspace ab. Dafür braucht man die Kameraposition e (eye), Blickrichtung g und einen up-Vektor t. Die Blickrichtung kann auch aus einem Punkt und der Kameraposition berechnet werden. Benötigte Mathematik:

Kreuzprodukt: $\begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \times \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} a_2 \cdot b_3 - a_3 \cdot b_2 \\ a_3 \cdot b_1 - a_1 \cdot b_3 \\ a_1 \cdot b_2 - a_2 \cdot b_1 \end{pmatrix}$

Vektornorm: $||v|| = \sqrt{x^2 + y^2 + z^2}$

Zu Berechnen:

- 2D: $w = -g/||g||$, für u eine Koordinate von w negieren
- 3D: $w = -g/||g||$, $u = \frac{t \times w}{||t \times w||}$, $v = w \times u$

Die resultierende Transformationsmatrix für 3D:

$$\begin{pmatrix} u_x & u_y & u_z & -u^T e_x \\ v_x & v_y & v_z & -v^T e_y \\ w_x & w_y & w_z & -w^T e_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

## Projektionsmatrizen

Orthogonale Projection auf $z = 0$-Ebene:

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

Ursprungsprojektion auf $z = 1$-Ebene:

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

Diese Matrizen hat man aber noch kein Frustrum mit Aspekt und Field of View. Mit den folgenden Projektionsmatrizen wird der sichtbare Bereich im 3 Dimensionalen auf einen Einheitscubus abgebildet. Damit kann insbesonders die Verzerrung der perspektivischen Betrachtung dargestellt werden.
Vorgehensweise: Die homogenisierte Matrix wird an einen homogenisierten Punkt multipliziert und dann normalisiert. Das Ergebnis liegt dann innerhalb des Einheitscubus. Die Darstellung des projizierten Punktes auf dem Aspekt sind dann einfach die x,y-Koordinaten aus dem Ergebnis. Der Wert der tiefe ist irrelevant.
Benötigte Parameter: Near n, Far f z-Koordinaten und vom Aspekt die left l, right r, top t und bottom b x,y-Koordinaten
Um die dazugehörigen zweidimensionalen Matrizen zu erhalten, verwirft man die erste Zeile und Spalte.

### Orthogonal

Blickt man in z-Richtung, dann hat man in der letzten Zeile eine -1. Blickt man entgegen der z-Richtung hat man eine 1.

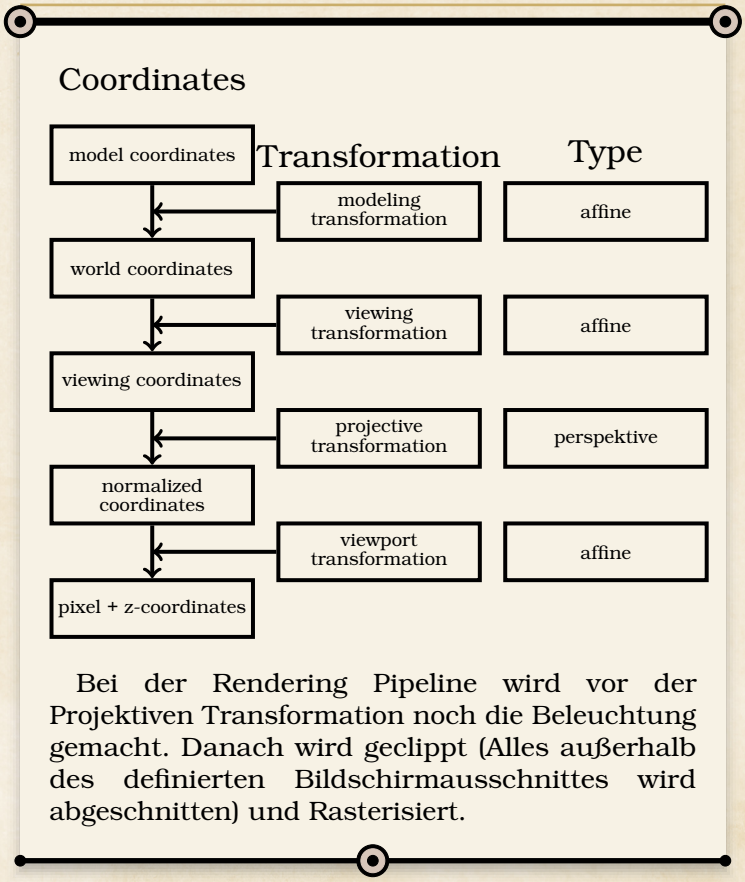
$$\begin{pmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{2}{n-f} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & \pm 1 \end{pmatrix}$$

### Perspektivisch

Blickt man in z-Richtung, dann hat man in der letzten Zeile eine 1. Blickt man entgegen der z-Richtung hat man eine -1.

$$\begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{l+r}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{b+t}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & \pm 1 & 0 \end{pmatrix}$$

## Viewing Pipeline



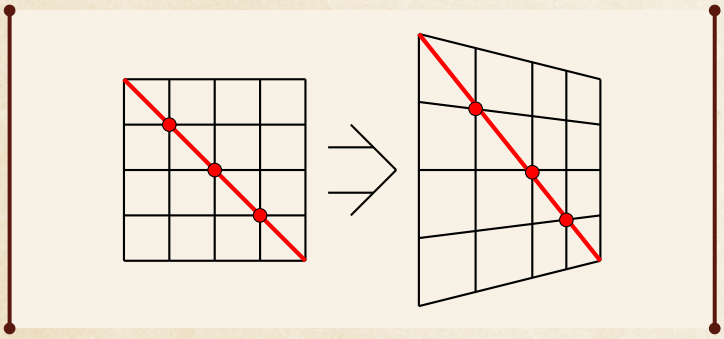

Bei der Rendering Pipeline wird vor der Projektiven Transformation noch die Beleuchtung gemacht. Danach wird geclippt (Alles außerhalb des definierten Bildschirmausschnittes wird abgeschnitten) und Rasterisiert.

# Texturen

## Grundlegendes

- Parametrisierung: Ein Texture Mapping von 2d in 3d oder andersrum finden
- Aliasing bei Texturen: Viele Pixel aus der Rasterisierung werden durch schlechts Texture sampling auf wenig Pixel des Bildschirms abgebildet. Das führt zu einem falschen Bild (Schachbrett Muster: Mehrere Weiße/schwarze Pixel direkt nebeneinander)
- Antialising: Mip-Mapping und Interpolation der Grenzregionen
- Pixel Footprint: Bereich der Welt, der auf einen Pixel des Bildschirms projeziert wird.

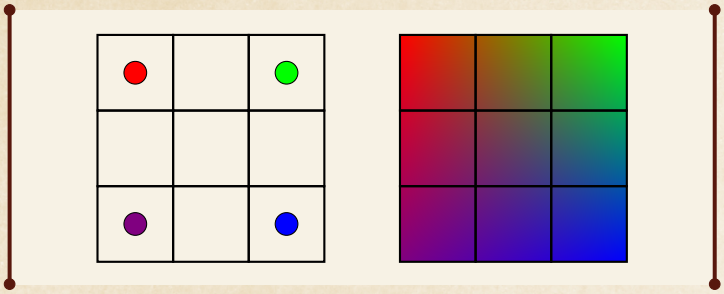
## Perspektivische Texturen



## Interpolation

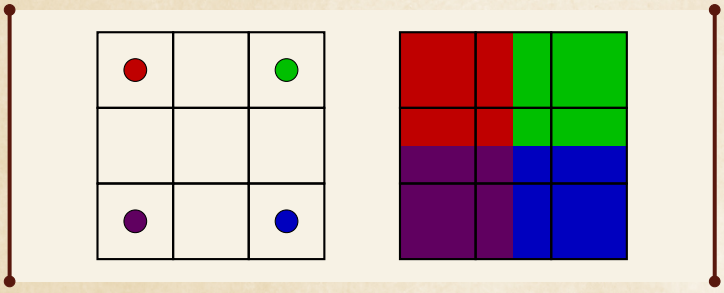
! Interpolation mit unterschiedlichen Farbmodellen erzeugt unterschiedliche Ergebnisse !

### Bilineare Interpolation



Farbwerte werden abhängig zur Entfernung zur nächst gelegenen Farbe berechnet. Will man z.b. den Farbwert des Mittelpunktes eines Quadrates berechnen, dann nimmt man die Summe der Multiplikation der Farbwerte mit 0.25.

### Nearest Neighbor Interpolation



Ein Pixel bekommt die Farbe, welche am nächsten zu diesem Pixel ist.

## Perspektivische Interpolation

- Warum: Tiefenwert bei Interpolation miteinbeziehen
- 1. Schritt: Texturkoordinaten der Eckpunkte berechnen: Farbwerte homogenisieren und resultierende Werte durch z-Weltkoordinate teilen
- 2. Schritt: Baryzentrische Koordinaten des zu berechnenden Punktes bestimmen. (Mittelpunkt: (1/3,1/3,1/3) )
- 3. Schritt: Interpolation des zu berechnenden Punktes. Summe der Multiplikation der baryzentrichen Koordinaten mit den Farbwerten des dazugehörigen Punktes
- 4. Schritt: Dehomogenisierung des Ergebnisses

**Beispiel:**

- Gegeben: $A(0,0,1)(0,0), B(2,5,2)(0,1), C(4,1,2)(1,0)$
- Gesucht: Textur an $(2,2)$ mit baryz. Koords. $(\frac{1}{3},\frac{1}{3},\frac{1}{3})$

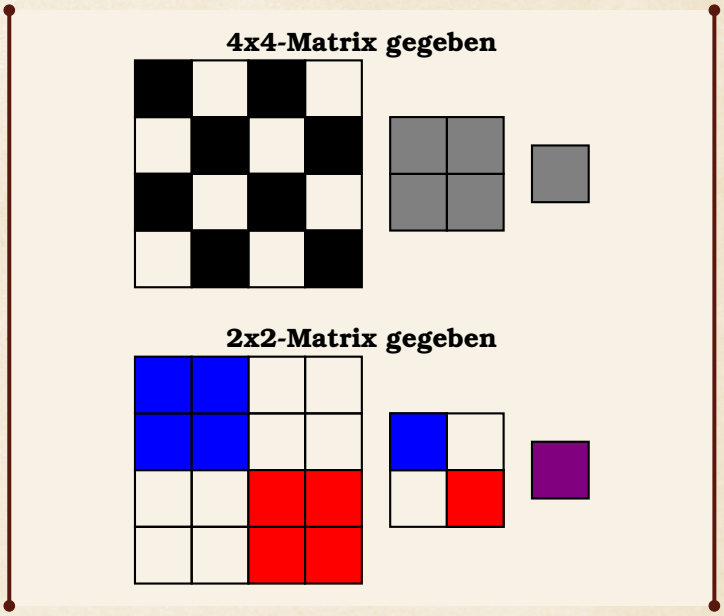
$$A(0,0,1) \to A'(0,0,1), B(0,1,1) \to B'(0,\frac{1}{2},\frac{1}{2})$$

$$C(1,0,1) \to C'(\frac{1}{2},0,\frac{1}{2})$$

$$\frac{1}{3}\cdot A' + \frac{1}{3}\cdot B' + \frac{1}{3}\cdot C' = (\frac{1}{6},\frac{1}{6},\frac{4}{6})$$

$$\frac{1}{6}\backslash\frac{4}{6} = \frac{1}{6}\cdot\frac{6}{4} = \frac{6}{24} = \frac{1}{4} \to (\frac{1}{4},\frac{1}{4})$$

## Mip Mapping




- Erzeugt aus einer gegebenen Textur Texturen mit weniger Pixel. Letztere können dann z.b. für entferntere Objekte genutzt werden
- Höher auflösende Texturen können nicht erzeugt werden, sind aber logisch herleitbar
- Zur Texturerzeugung gibt es viele Möglichkeiten, bekannt: Interpolation und Boxfiltern
- Vorgehen Beispiel: Man legt die 2x2 Textur auf die 4x4 Textur. Zu jedem Pixel $p_0$ der 2x2 Textur gibt es nun 4 Pixel $p_i$ aus der 4x4 Textur. Ergebnis: $p_0 = 1/4 * \sum(r_i, g_i, b_i)$.
- Trilinear Interpolation: Für einen smotheren Übergang zw. zwei Mipmapping Stufen. Erzeuge mit bilinearer Interpolation die beiden Werte für die Stufen. Dann interpoliere die beiden Werte nocheinmal.

# Visibility

## Occlusion

Vordere Objekte verdecken dahinterliegende

### Painters Algorithm

Objekte werden von hinten nach vorne sortiert und dementsprechend gerendert. Dauert lange und kann nicht alles darstellen (z.b. Ein Objekt durchdringt teilweise ein vor ihm liegendes)

### Z-Buffer

Neben den Farbwerten wird pro Pixel eines Objekts auch ein z-Wert gespeichert. Zu Beginn hat man nur die Hintergrundfarbe und jeder Pixel des Bildes einen unendlich kleiner z-Wert. Beim Rendern werden die Pixel eines Objektes betrachtet, und wenn dieser größer als der dazugehörige Pixel im Bild ist, wird letzterer geupdated.

### Back Face Culling

Zur Beschleunigung der Berechnung: Für die einzelnen Dreiecke wird deren Richtungsvektor mit dem des Betrachters verglichen. Ist der Skalarprodukt der beiden Vektoren größer als 0, ist das Dreieck sichtbar und wird gerendert, ansonsten nicht.

## Transparenz

Transparenz ist meistens ein vierter Wert $\alpha$, welcher an die Farbe angehängt wird. (z.b. aus RGB wird RGBA mit (1,0,0,0.2) für ein leicht Rotes Bild). Ist $\alpha = 1$ ist das Objekt undurchsichtig.

### Alpha Blending

Man legt ein Transparentes Objekt O auf ein bereits existierendes Bild B. Reihenfolge ist relevant! Für einen Pixel P sieht das dann so aus:

$$P = (1 - \alpha_O) \cdot B + \alpha_O \cdot O$$

### Occlusion mit Transparenz

Mögliche Vorgehensweisen:

- Painters Algorithm
- Zuerst z-Buffer für Undurchsichtige Objekte, dann Painters Algorithm für transparente
- Schnell aber schlecht: z-Buffer für undurchsichtige, dann transparente ohne Sortierung rendern

# Tonemapping

In der Realität gibt es einen großen Helligkeitsbereich, auf Bildschirmen aber einen viel kleineren. Beim Tonemapping passt man den Kontrastumfang so an, dass man ein Bild auf einem Bildschirm realistisch darstellen kann. Es gibt viele Möglichkeiten fürs Tonemapping, diese lassen sich in globale und lokale Vorgehensweisen einteilen.

# Virtual Reality

HMD = Head mounted Display, stellt virtuelle Realitäten dar. Augmented Reality hat ein transparentes Display, Bilder werden in die Realität projiziert.

## Stereo Rendering

Man hat nicht eine Kamera, sondern zwei, die leicht unterschiedliche Bilder zeigen. Dies wird Stereo Rendering genannt.

Damit können 3D-Effekte erzeugt werden:

- 3D-Brille mit Polarisation-Filter: Zwei Bilder gleichzeitig. Links und Rechts wird dementsprechend Polarisiert
- Shutter-Glasses: Immer ein Bild fürs rechte und dann fürs Linke Auge. Das andere Glass der Brille wird dann jeweils abgedunkelt (doppelte Framerate benötigt)
- Rot-Blau Brillen: Ähnlich wie Polarisierung
- HMD: Ein 'Bildschirm' pro Auge

### Bilderzeugung bei HMD

Man nimmt zwei Kameras und der überlappende Bereich der Kameras ist das resultierende Bild.

### Tiefenwahrnehmung mit HMD

**Möglich:**

- (Stereo)-Parallax: Aus der Diskrepanz zwischen der Abbildung für das Rechte und Linke Auge wird tiefe wahrgenommen
- Motion-Parallax: Wenn man den Kopf bewegt, bewegen sich nahe Objekte schneller als ferne!
- Color: Farben in der ferne werden bläulicher
- Schattierung: Schatten liefert Struktur und Tiefe eines Objekts
- Größe: Entfernte Objekte sind kleiner
- Schatten: Liefern Infors zur Position eines Objekts

**Nicht Möglich:**

- Focus: Durch das Fokussieren von Objekten können wir tiefe wahrnehmen. Nicht fokussierte Objekte werden unscharf. Bei HMD fokussiert das Auge aber auf das Display und nicht auf die dargestellten Objekte.

## HMD - Probleme

### Tracking

Ausrichtung und Position im Raum müssen festgestellt und verarbeitet werden. Beispiele für Sensoren:

- Accelerometer: Misst Beschleunigung
- Gyroscope: Misst Rotationsbeschleunigung
- Optical Tracking: Mehrere Sensoren im Raum messen Position anhand von Lasern

### Latency

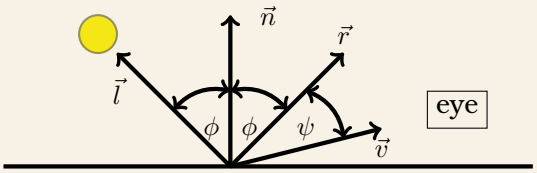
Benötigte Zeit um das Bild zu erneuern. Mögliche Ursachen der Verzögerung:

- Tracking: Bestimmungzeit von Position und Ausrichtung
- Rendering: Zeit für Bilderzeugung
- Display Latency: Zeit für Bildschirmaktualisierung

# Lightning and Shading

- Ein Lichtstrahl kann absorbiert, reflektiert, gestreut, gebrochen oder weitergeleitet werden
- Lokale Beleuchtung: Nur Pixel und Licht wird betrachtet. z.b. Phong Lighting
- Globale Beleuchtung: Neben Pixel und Licht werden auch andere Objekte mit einbezogen. z.b. Ray Tracing
- Directional Light source: Alle Lichtstrahlen haben den gleichen Vektor und sind parallel
- Point Light source: Differente Lichtstrahlen, da radialsymmetrisch hervorgehend aus der Quelle
- Berechnungen: Vektoren sollten vor dem Einsetzen in die Formeln normiert werden!!!

## Phong Lighting




$$L_{total} = L_{ambient} + L_{diffuse} + L_{specular}$$

Benötigte Mathematik Skalarmultiplikation:

$$\begin{pmatrix} a_1 \\ b_1 \end{pmatrix} \circ \begin{pmatrix} a_2 \\ b_2 \end{pmatrix} = a_1 \cdot a_2 + b_1 \cdot b_2$$

Alle Vektoren sollten normalisiert werden:

$$\begin{pmatrix} a \\ b \end{pmatrix} \overset{normalize}{\Rightarrow} \frac{1}{\sqrt{a^2 + b^2}} \begin{pmatrix} a \\ b \end{pmatrix}$$

### Ambient Light

$$L_{ambient} = k_{ambient} + I_{ambient}$$

- $I_{ambi}$: Lichtintensität des ambienten Lichts
- $k_{ambi}$: ambienter Reflektionskoeffizient
- Modellierung der indirekten Beleuchtung
- Globale Beleuchtung, aber wesentlich schlechter als Ray Tracing

### Diffuse Reflection

$$L_{diff} = k_{diff} \cdot I_{in} \cdot cos(\phi) = k_{diff} \cdot I_{in} \cdot (\vec{n} \circ \vec{l})$$

- $k_{diff}$: Farbe der Oberfläche
- $I_{in}$: Intensität der Lichtquelle, default = 1
- $\vec{n}$: Normalenvektor des Punktes
- $\vec{l}$: Lichtvektor zur Lichtquelle
- Wenn das Skalarprodukt negativ ist: setzte es auf 0
- Blickwinkel des Beobachters ist egal!
- lokale Betrachtung

### Specular Reflection

$$L_{spec} = k_{spec} \cdot I_{in} \cdot cos(\psi)^{n_s} = k_{spec} \cdot I_{in} \cdot (v \circ r)^{n_s}$$

- $k_{spec}$: Farbwert der Oberfläche
- $I_{in}$: Intensität der Lichtquelle, default = 1
- $n_s$: Reflektionsfaktor der Oberfläche
- $\vec{v}$: Vektor zum Beobachter. Berechnung: $(x_{eye} - x_{punkt}, y_{eye} - y_{punkt}, ...)$
- $\vec{r}$: Reflektierung von $\vec{l}$. Berechnung: $r = 2 \cdot (n \circ l) \cdot n - l$. Abkürzung bei Reflektion an Achse: Die Achsen-Koordinate des Lichtstrahls negieren
- Blickwinkel des Beobachters ist wichtig!
- lokale Betrachtung

## Torrance-Sparrow Light Model

- Physikalisch basiertes Lichtmodell, welches anhand von 3 Faktoren Rauheiten und derren Abschattung berücksichtigt
- 1. D: Wahrscheinlichkeitsfunktion der Neigungswinkel beschreibt Rauheit
- 2. F: Der Fresnelterm beschreibt, wie Licht von jeder der Facetten reflektiert wird
- 3. G: geometrische Abschwächungsfaktor dient der Berücksichtigung von Beschattung und Verdeckung der Facetten

## Shading

Berechnung der Farbe eines beliebigen Punktes auf einem Objekt.

### Flat Shading

- Pro Primitiv gibt es nur eine Normale
- Diese Normale ist ein orthogonaler Vektor zur Ebene des Primitiv
- Das ganze Primitiv hat die Farbe, welche anhand dieser Normalen mittels Phong Lightning berechnet wird

### Gouraud Shading

- Farbwerte werde z.b. mit Phong Lightning nur für die Eckpunkte berechnet
- Farbwerte innerhalb eines Objekts werden mittels Interpolation erzeugt

### Phong Shading

- Für jeden Punkt wird eine Normale durch Interpolation der Normalen der Eckpunkte erzeugt
- Anhand dieser Normale wird dann mittels Phong Lightning die Farbe des Punktes berechnet

### Anmerkung

Bei Flat und Gouraud Shading kann Licht Highlighting verschwinden oder verzerrt werden. Bei Phong Shading geschieht dies nicht.

# Ray Tracing

### Ray-Definition

$$p(t) = \vec{e} + t \cdot \vec{d} = \begin{pmatrix} e_1 \\ e_2 \\ \vdots \end{pmatrix} + t \cdot \begin{pmatrix} d_1 \\ d_2 \\ \vdots \end{pmatrix}$$

- Rays werden über einen Aufhängepunkt $\vec{e}$ und Richtungsvektor $\vec{d}$ definiert
- Rasterisierung: Pro Pixel ein Ray mit Schnittpunkt
- Tiefe/Zoom: Mehrere Rays, die sich im fokusierten Punkt schneiden

### Antialiasing

Mehrere Rays werden nach vorgegebenem Schema durch einen Pixel geschickt. Beispielschemata:


## Schnittpunkte

### Linie

Vektoren gleichsetzen und Gleichungssystem lösen. Lösung in dazugehörige Ausgangsgleichung einsetzen.

### Dreiecke

$$\vec{e} + t \cdot \vec{d} = A + \beta \cdot (B - A) + \gamma(C - A)$$

$$\begin{pmatrix} \vdots & \vdots & \vdots \\ d & A-B & A-C \\ \vdots & \vdots & \vdots \end{pmatrix} \cdot \begin{pmatrix} t \\ \beta \\ \gamma \end{pmatrix} = \begin{pmatrix} \vdots \\ A-e \\ \vdots \end{pmatrix}$$

- Dreieck ABC mit baryzentrischen Koords $(\alpha, \beta, \gamma)$
- Gesucht sind die drei unbekannten $t, \beta, \gamma$

### Polygone

1. Schnittpunkt des Rays mit Polygonebene:

$$p = \vec{e} + \frac{(p_1 - \vec{e}) \circ \vec{n}}{\vec{d} \circ \vec{n}} \cdot \vec{d}$$

2. Teste ob p im Polygon liegt.

- $p_1$ = Polygonpunkt, $\vec{n}$ = Vektor auf Polygon(-Ebene)
- Schritt 2 ist im allgemeinen schwierig
- Beim Kreis in 3D: Entfernung zw. Schnittpunkt und Mittelpunkt berechnen. Ist diese kleiner gleich dem Radius, liegt der Schnittpunkt im Kreis

### Kugeln

$$a = 1, \quad b = 2 \cdot \vec{d} \circ (\vec{e} - c), \quad c = (\vec{e} - c) \circ (\vec{e} - c) - r^2$$

$$t = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} = \frac{-b \pm \sqrt{b^2 - 4c}}{2}$$

- c = Mittelpunkt, r = Radius

## Zusatzeffekte

- **Shadow:** ShadowRay von Schnittpunkt zur Lichtquelle emittieren. Trifft dieser Ray an einem !anderen!(sonst Selbstschattierung) Punkt ein Objekt, liegt der Ausgangspunkt im Schatten
- **Reflection:** Am Schnittpunkt wird ein Ray in die reflektierende Richtung geschickt. Der Wert dieses Rays wird mit einem Reflektionskoeffizient auf den Wert des Schnittpunktes gerechnet.
- **Refraction (Brechung):** Am Schnittpunkt wird ein Ray mit einem Brechungsfaktor auf die Ausgangsrichtung emittiert. Beispiel: Wasser
- **Reflection + Refraction:** Wasserdarstellung durch Kombination aus Reflection und Refraction anhand der Fresnel Gleichung verbessern

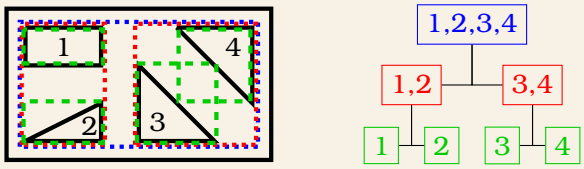
**Probleme beim Erzeugen der Zusatzeffekte:**

- **BRDFs:** Bidirektionale Reflexionsverteilungsfunktion definiert Oberflächeneigenschaften für die Anzahl der emittierten Rays, deren Richtung und Einberechnungsfaktoren
- **Area Lights:** Größere Lichtflächen mit nicht parallelen Lichtstrahlen sind problematisch. Eine Lösung ist, mehrere Rays zum Licht zu schicken

## Beschleunigung

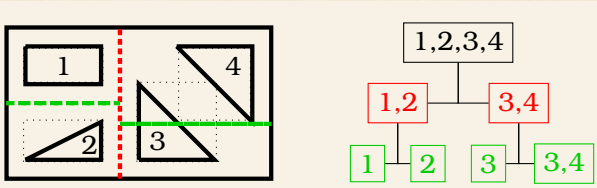
- Pro Pixel $p$ muss für jedes Objekt $n$ betrachtet werden, ob es einen Schnittpunkt gibt $\Rightarrow O(p \cdot n)$
- Mit Optimierungsstruktur auf $O(p \cdot log(n))$ reduzierbar
- Optimierungsstrukturen müssen vor dem Ray Tracing aufwändig erzeugt werden
- Grundidee: Geometrische Binäre Suche
- Nutzt AABB: Axis Aligned Bounding Box

### Bounding Volume Hierarchy




- Implementierung ist meist rekursiv
- Pro Stufe werden möglichst kleine Bounding Boxen um die 'etwa' gleiche Anzahl an Objekten erzeugt
- Es gibt freie Fläche, die nicht abgedeckt sind
- Mehrere Boxen können den gleichen Bereich abdecken $\Rightarrow$ Baumknoten auch
- Eher Effizient bei wenigen Objekten

### k-d Trees




- Pro Stufe wird der Raum entlang einer Ache so geteilt, dass in beiden Seiten etwa gleich viele Objekte sind
- Boxen/Knoten überdecken nicht gleiche Bereiche
- Objekte können in mehreren Knoten sein