

CG Mathespick

- Vektor schreibt man als $\begin{pmatrix} 3 \\ 4 \end{pmatrix}$ und in einer Zeile: $(3 \ 4)^T$. T steht für transponiert.
- **Ortsvektor** ist von Ursprung zu Punkt und **Richtungsvektor** ist irgendwo
- Einheitsvektoren gehen eine Einheit nach rechts oder oben $e_x = (1 \ 0)^T$ / $e_y = (0 \ 1)^T$
- Linearkombination ist Vektor der ein Vielfaches vom Einheitsvektor sind $p = 3 \cdot e_x + 4 \cdot e_y$
- $\sin = GK / HP$, $\cos = AK/HP$, $\tan = GK / AK$

Gerade beschreiben

- **Punkt-Punktform**: $r = P1 + t (P2 - P1)$ und t ist eine beliebige reelle Zahl
- **Punkt-Richtungsform**: $r = P1 + t \cdot n$ und n ist der Richtungsvektor kriegt man wenn man Endpunkt (P2) minus Anfangspunkt (P1) macht
- Ab hier nur 2D:
 - **Steigungsform**: $y = mx + b$ und die Steigung m ist y / x
 - **Punkt-Richtungsform**: $(y - y_0) = m(x - x_0)$ Steigung siehe oben und (x_0/y_0) ist ein Punkt auf der Gerade
 - **Allgemeine Geradengleichung**: $ax + by + c = 0$

Aus zwei Punkten P1 (3/4) und P2 (5/3) auf allgemeine Geradengleichung:

Punkte in Homogene Koordinaten umwandeln: $P1 = \begin{pmatrix} 3 \\ 4 \\ 1 \end{pmatrix}$ und $P2 = \begin{pmatrix} 5 \\ 3 \\ 1 \end{pmatrix}$

Kreuzprodukt nehmen: $P1 \times P2 = \begin{pmatrix} 1 \\ 2 \\ -11 \end{pmatrix}$ (Mit TR `crossp(a, b)`)

Dann einsetzen: $1 \cdot x + 2 \cdot y - 11 = 0$

Bei parallelen Gerade Skalarprodukt = 0 (weil Winkel 0 ist)

Matrix Multiplikation

$$\begin{pmatrix} 3 & 2 & 1 \\ 1 & 0 & 2 \end{pmatrix} \cdot \begin{pmatrix} 1 & 2 \\ 0 & 1 \\ 4 & 0 \end{pmatrix} = \begin{pmatrix} 3 \cdot 1 + 2 \cdot 0 + 1 \cdot 4 & * \\ * & * \end{pmatrix} = \begin{pmatrix} 7 & * \\ * & * \end{pmatrix}$$

$$\begin{pmatrix} 3 & 2 & 1 \\ 1 & 0 & 2 \end{pmatrix} \cdot \begin{pmatrix} 1 & 2 \\ 0 & 1 \\ 4 & 0 \end{pmatrix} = \begin{pmatrix} 7 & 3 \cdot 2 + 2 \cdot 1 + 1 \cdot 0 \\ * & * \end{pmatrix} = \begin{pmatrix} 7 & 8 \\ * & * \end{pmatrix}$$

$$\begin{pmatrix} 3 & 2 & 1 \\ 1 & 0 & 2 \end{pmatrix} \cdot \begin{pmatrix} 1 & 2 \\ 0 & 1 \\ 4 & 0 \end{pmatrix} = \begin{pmatrix} 7 & 8 \\ 1 \cdot 1 + 0 \cdot 0 + 2 \cdot 4 & * \end{pmatrix} = \begin{pmatrix} 7 & 8 \\ 9 & * \end{pmatrix}$$

$$\begin{pmatrix} 3 & 2 & 1 \\ 1 & 0 & 2 \end{pmatrix} \cdot \begin{pmatrix} 1 & 2 \\ 0 & 1 \\ 4 & 0 \end{pmatrix} = \begin{pmatrix} 7 & 8 \\ 9 & 1 \cdot 2 + 0 \cdot 1 + 2 \cdot 0 \end{pmatrix} = \begin{pmatrix} 7 & 8 \\ 9 & 2 \end{pmatrix}$$

Homogene Koordinaten 2D

Homogene Koordinaten wird ein Vektor einfach um eine Komponente erweitert. Wenn $P = \begin{pmatrix} x \\ y \end{pmatrix}$ dann hat er die homogenen Koordinaten $\begin{pmatrix} x \cdot w \\ y \cdot w \\ w \end{pmatrix}$. Möchte man Sachen machen die nicht durch den Ursprung gehen, einfach der Punkt oder die Gerade in den Ursprung verschieben und dann wieder zurück (Objekt natürlich auch verschieben).

Matrix für **Translation** (ist eine Euklidische Transformation): Matrix für **Skalierung** (Ähnlichkeiten, Streckung):

$$\begin{pmatrix} x' \\ y' \\ w' \end{pmatrix} := \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} x' \\ y' \\ w' \end{pmatrix} := \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Zentrum der **Streckung** bleibt bei Skalierung gleich. Wenn (u v) das Zentrum sind gilt:

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}$$

Matrix für **Rotation** (ist eine Euklidische Transformation):

$$\begin{pmatrix} x' \\ y' \\ w' \end{pmatrix} := \begin{pmatrix} \cos(\beta) & -\sin(\beta) & 0 \\ \sin(\beta) & \cos(\beta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Matrix für **Spiegelung** an einer Gerade durch den Ursprung (Winkel ist von X-Achse bis Gerade):

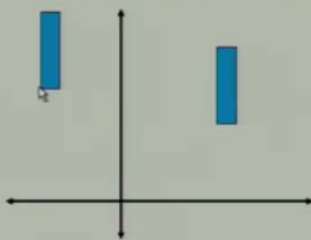
$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos(2\delta) & \sin(2\delta) & 0 \\ \sin(2\delta) & -\cos(2\delta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Matrix für **Scherrung** in x-Richtung \neq (Affine Transformationen):

$$\begin{pmatrix} x' \\ y' \\ w' \end{pmatrix} := \begin{pmatrix} 1 & m & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Verknüpfung von Transformationen

- assoziativ: $A \cdot B \cdot C = (A \cdot B) \cdot C = A \cdot (B \cdot C)$
- nicht kommutativ: $A \cdot B \neq B \cdot A$
- Drehung um 90° + Verschieben um (4,3) \neq Verschieben um (4,3) + Drehung um 90°



Verknüpfung immer von rechts nach links!

Hessesche Normalform (HNF)

Braucht man z.B. um zu schauen ob ein Punkt links oder rechts von Gerade ist. Wenn d positiv ist Punkt unterhalb der Gerade. Wenn d negativ ist der Punkt oberhalb der Gerade. n ist senkrecht auf der Gerade.

Von der Koordinatengleichung zur HNF

Koordinatengleichung der Gerade g:

$$ax + by + c = 0 \quad (1)$$

Mit $n_x = \frac{a}{\sqrt{a^2 + b^2}}$, $n_y = \frac{b}{\sqrt{a^2 + b^2}}$ und $d = -\frac{c}{\sqrt{a^2 + b^2}}$, wird aus (2)

$$n_x x + n_y y - d = 0.$$

Somit ist

$$\vec{n} = \begin{pmatrix} n_x \\ n_y \end{pmatrix} = \frac{1}{\sqrt{a^2 + b^2}} \begin{pmatrix} a \\ b \end{pmatrix}$$

ein (normierter) Normalenvektor und

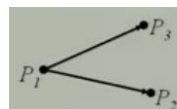
$$d = -\frac{c}{\sqrt{a^2 + b^2}}$$

der (vorzeichenbehaftete) Abstand der Gerade g vom Ursprung.

30 / 6

Ebene

Aus drei Punkten eine Ebene:


$$\begin{array}{l} \vec{P_2 - P_1} \times \vec{P_3 - P_1} \\ Ax + By + Cz + D = 0 \end{array} \quad \begin{pmatrix} A \\ B \\ C \end{pmatrix}$$

Um D zu erhalten einfach einen Punkt einsetzen z.B. P1. Wenn wir Gegenuhrzeigersinn rechnen schaut Normalenvektor in Richtung des Betrachters.

Homogene Koordinaten 3D

Matrix für Translation:

$$T(t_x, t_y, t_z) = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotation um z-Achse (gleich wie in 2D):

$$R_z(\delta) = \begin{pmatrix} \cos(\delta) & -\sin(\delta) & 0 & 0 \\ \sin(\delta) & \cos(\delta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Matrix für Skalierung:

$$S(s_x, s_y, s_z) = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotation um x-Achse:

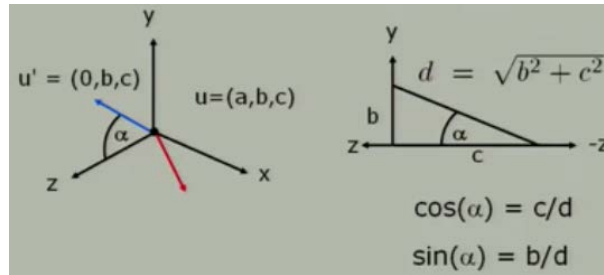
$$R_x(\delta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\delta) & -\sin(\delta) & 0 \\ 0 & \sin(\delta) & \cos(\delta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotation um y-Achse:

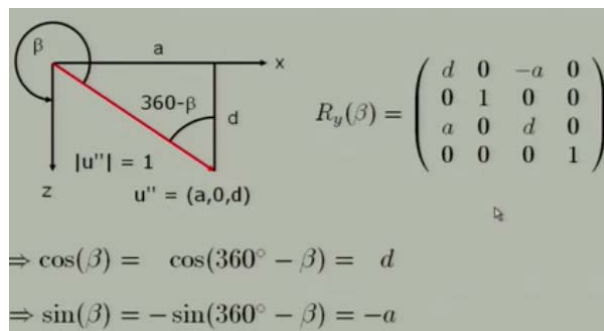
$$R_y(\delta) = \begin{pmatrix} \cos(\delta) & 0 & \sin(\delta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\delta) & 0 & \cos(\delta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotation um beliebige Achse:

0. Drehachse in Vektor verwandeln (P2-P1) und normieren (Vektor durch Länge teilen)
1. Drehachse in Ursprung translateren (Translation um -x, -y -z verschieben)
2. Rotation um x-Achse damit Drehachse in xz-Ebene zu liegen kommt. Der Sinus und Cosinus kann man in die Drehmatrix einsetzen. Beim neuen Vektor ist die mittlere Komponente (y) gleich 0.



3. Rotation um y-Achse damit Drehachse auf z-Achse zu liegen kommt. D haben wir oben berechnet und a kennen wir von vorher.



4. Rotation um z-Achse mit gegebenem Winkel
5. Alles Rückgängig machen

Projektion

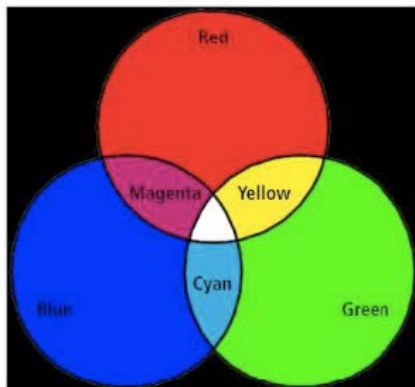
- Parallel Projektion: Augenpunkt im Unendlichen
- Zentral/Perspektivische Projektion: Augenpunkt in endlichem Abstand

Die perspektivische Projektion mit Zentrum im Nullpunkt und mit Bildebene $\varepsilon : ax + by + cz + d = 0$ wird durch Multiplikation mit der folgenden homogenen Matrix vermittelt:

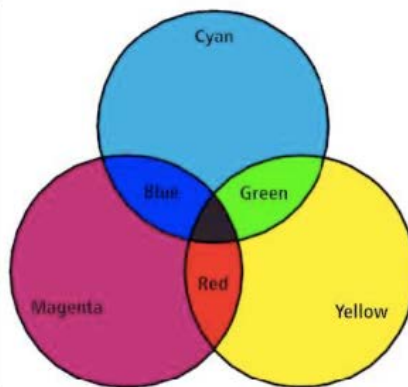
$$H = \begin{pmatrix} -d & 0 & 0 & 0 \\ 0 & -d & 0 & 0 \\ 0 & 0 & -d & 0 \\ a & b & c & 0 \end{pmatrix}$$

Farbe

- Stäbchen – Schwarz – Weiss (75 -150 Mio.)
- Zäpfchen – Farbe (6-7 Mio.)
- Normfarbtafel: Spektralfarben am Rand, Mischen zwei Farbe mit Linie verbinden alle zwischendrin, Dominante Wellenlänge Farbe über Weiss mit Rand verbinden, Verbindungslinie von zwei Komplementärfarben muss durch weiss gehen
- CMYK zu RGB einfach minus rechnen. K ist Grauwert meist $\min(R,G,B)$. Wenn alle RGB gleich nur K setzen.
- Rot gut bei wenig Licht



Magenta = Red + Blue
Cyan = Blue + Green
Yellow = Green + Red



Magenta = White - Green
Cyan = White - Red
Yellow = White - Blue

Tintenfarbe	absorbiert	reflektiert
Cyan	Rot	Grün und Blau
Magenta	Grün	Blau und Rot
Gelb	Blau	Rot und Grün
Schwarz	Alles	Nichts

RGB to HSV

$$\begin{aligned}
 MAX &:= \max(R, G, B), \quad MIN := \min(R, G, B) \\
 H &:= \begin{cases} 0, & \text{falls } MAX = MIN \Leftrightarrow R = G = B \\ 60^\circ \cdot \left(0 + \frac{G-B}{MAX-MIN}\right), & \text{falls } MAX = R \\ 60^\circ \cdot \left(2 + \frac{B-R}{MAX-MIN}\right), & \text{falls } MAX = G \\ 60^\circ \cdot \left(4 + \frac{R-G}{MAX-MIN}\right), & \text{falls } MAX = B \end{cases} \\
 \text{falls } H < 0^\circ \text{ dann } H &:= H + 360^\circ \\
 S_{HSV} &:= \begin{cases} 0, & \text{falls } MAX = 0 \Leftrightarrow R = G = B = 0 \\ \frac{MAX-MIN}{MAX}, & \text{sonst} \end{cases} \\
 V &:= MAX \\
 L &:= \frac{MAX + MIN}{2}
 \end{aligned}$$

HSV to RGB

$$\begin{aligned}
 h_i &:= \left\lfloor \frac{H}{60^\circ} \right\rfloor; \quad f := \left(\frac{H}{60^\circ} - h_i \right) \\
 p &:= V \cdot (1 - S); \quad q := V \cdot (1 - S \cdot f); \quad t := V \cdot (1 - S \cdot (1 - f)) \\
 (R, G, B) &:= \begin{cases} (V, t, p), & \text{falls } h_i \in \{0, 6\} \\ (q, V, p), & \text{falls } h_i = 1 \\ (p, V, t), & \text{falls } h_i = 2 \\ (p, q, V), & \text{falls } h_i = 3 \\ (t, p, V), & \text{falls } h_i = 4 \\ (V, p, q), & \text{falls } h_i = 5 \end{cases}
 \end{aligned}$$

Dithering

- Kann Farben simulieren (z.B. Schwarz-Weiss Drucker) es gibt Quantisierung, Dithering und Error Diffusion
- Grauwertänderung von 1 zu 0.5 wird gleich wahrgenommen wie 0.5 zu 0.25
- Grauwert: $I = 0.299r + 0.587g + 0.114b$ (Wenn wieder RGB muss überall der selbe Wert drin stehen)
- Quantisierung: Auf nächsten zur Verfügung stehenden Farbwert runden
- Dithering ersetzt einen Grauwertpixel durch Dithermatrix

6	8	4
1	0	3
5	2	7

Wenn Dithering verwendet werden soll, muss das Bild zuerst auf die Anzahl Intensitätsstufen konvertiert werden, die mit der Dithermatrix dargestellt werden können.

Gegeben sei das folgende Bild mit Graustufen von 0-255. Dieses sollte nun mit 10 Stufen dargestellt werden. Wie sieht das Bild aus?

$$200 * 10 / 256 = 7.81 \text{ und dann abrunden}$$

Bild:

30	200	250	240
6	56	120	141

Resultat:

1 <input checked="" type="checkbox"/>	7 <input checked="" type="checkbox"/>	9 <input checked="" type="checkbox"/>	9 <input checked="" type="checkbox"/>
0 <input checked="" type="checkbox"/>	2 <input checked="" type="checkbox"/>	4 <input checked="" type="checkbox"/>	5 <input checked="" type="checkbox"/>

- Dithering bei gleicher Auflösung gibt es zwei Methoden:
 - Clustered dot dithering: Berechnen des Mittelwert einer $n \times n$ Region und ersetzen der Region durch die Dithermatrix
 - Dispersed dot dithering: Dieses Bild mit Wertebereich von 0 – 255 Grauwerten, soll in nur 2 Farben dargestellt werden. Die Dithermatrix kann 5 Stufen darstellen. Dann das gleiche wie oben machen z.B. $30 * 5 / 256 = 0.58$ abrunden = 0.

30	200	250	240
6	56	120	141

$$D^{(2)} = \begin{pmatrix} 0 & 3 \\ 1 & 2 \end{pmatrix}$$

0	3	4	4
0	1	2	2

0	0	1	1
0	0	1	0

Dann mappt man die Dither-Matrix zweimal auf die neue Tabelle und vergleicht die Werte. Falls Wert in neuer Tabelle grösser als in Dithermatrix wird Wert gesetzt.

- Error Diffusion

Bei der Error Diffusion wird der Fehler, der beim Setzen eines Pixels gemacht wird auf die umliegenden Pixel verteilt.

Betrachten wir das Bild:

255	175	100	50
132	145	90	250

Auf einem Bildschirm oder Drucker mit 2 Intensitäten stehen nun nur die beiden Intensitäten 0 und 255 zur Verfügung.

Nun wird der nächste Pixel (rot markiert) verarbeitet.

Frage 1: Soll ich 0 oder 255 nehmen? 180 -> 255 (Fehler: 75) Fehler wird auf umliegende Pixel aufaddieren (siehe unten)

Auf welchen Wert wird der Pixel gesetzt und wie gross ist der Fehler, der dabei gemacht wird?

Wert:

Fehler:

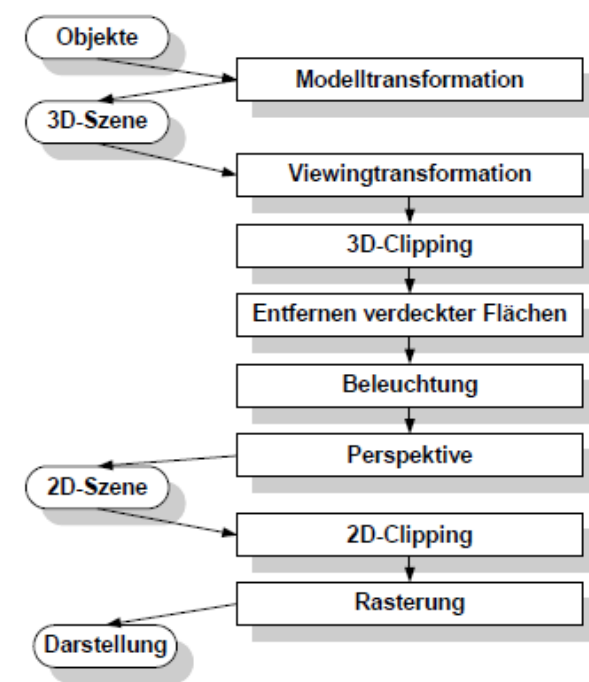
		7/16
1/16	5/16	3/16

Frage 2:

Wie sind die Werte der Pixel, nachdem der Pixel gesetzt und der Fehler auf die anderen Pixel verteilt wurde:

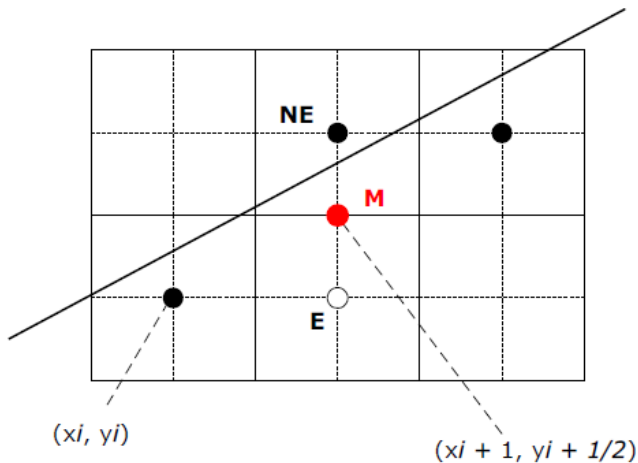
<input type="text" value="255"/>	<input type="text" value="255"/>	<input type="text" value="135"/>	<input type="text" value="50"/>
<input type="text" value="137"/>	<input type="text" value="170"/>	<input type="text" value="105"/>	<input type="text" value="250"/>

Grafik Pipeline



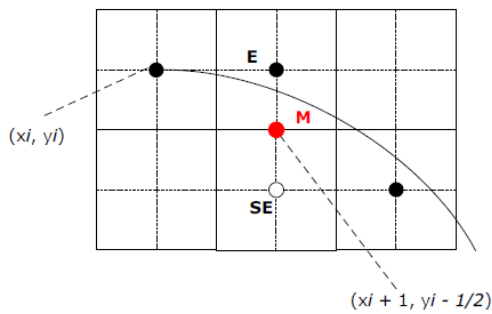
Scan Konvertierung

- Digital Differential Analyzer: x und y mit Steigung berechnen und y runden
- Mittelpunktschema



// Initialisierung	// Berechnung
$Dx = x_1 - x_0;$ $Dy = y_1 - y_0;$ $DE = 2 * Dy;$ $DNE = 2 * (Dy - Dx);$ $d = 2 * Dy - Dx;$ $y = y_0;$	$WritePixel(x_0, y_0)$ for $x = x_0 + 1$ to x_1 do if $d \leq 0$ then $d = d + DE;$ else $d = d + DNE;$ $y = y + 1;$ end $WritePixel(x, y);$ end

- Bei Kreis zeichnet man Viertelkreis und hat dann den Rest: $(x, -y)$, $(y, -x)$, $(-x, -y)$, $(-x, y)$, $(-y, x)$ und $(-y, -x)$



// Initialisierung	// Berechnung
$x = 0;$ $y = R;$ $d = 1 - R;$	$WritePixel(x, y);$ while $y > x$ do if $d < 0$ then $d = d + 2 * x + 3;$ else $d = d + 2 * (x - y) + 5;$ $y = y - 1;$ end $x = x + 1;$ $WritePixel(x, y);$ end

Füllen

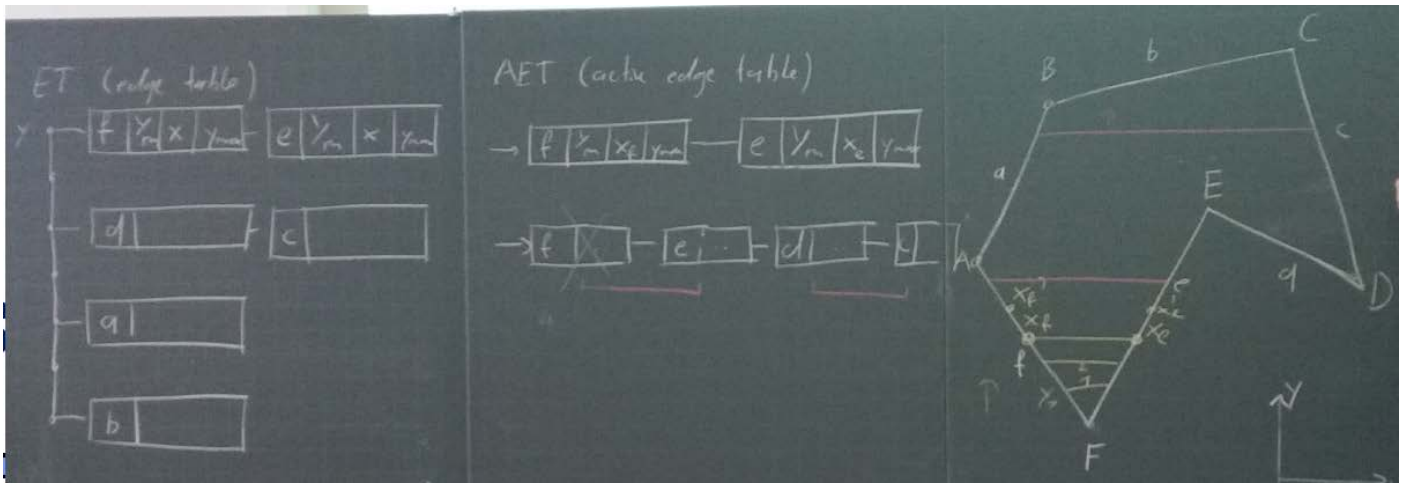
- Beim Füllen gibt 4er Flood (füllt nicht alles) und 8er Flood (kann überlaufen)

```

proc FloodFill(Int x, Int y,
                Color oldColor, Color newColor)
    if ReadPixel(x, y) == oldColor then
        WritePixel(x, y, newColor);
        FloodFill(x, y - 1, oldColor, newColor);
        FloodFill(x - 1, y, oldColor, newColor);
        FloodFill(x, y + 1, oldColor, newColor);
        FloodFill(x + 1, y, oldColor, newColor);
    end
end

```

- Bei Scanlinien Algorithmus hat Kantentabelle (nach minimalem y sortiert, bei gleichem y wird nach x sortiert) und aktive Kantentabelle (alle Kanten welche von der Scanlinie geschnitten werden)



Erzeuge ET

Initialisiere AET = empty

$y = 0$

repeat

Hole alle Kanten an der Position y aus der ET und schreibe sie in die AET

Addiere alle Kanten $ET(y)$ zu AET

Sortiere AET nach x

Wenn Kanten dazukommen immer nach x sortieren

Zeichne Spans

$y = y + 1$

Wenn y grösser als y_{\max} einer Kante wird, entferne die Kante

Entferne Kanten mit $y_{\max} = y$ aus AET

Aktualisiere den x Wert aller Kanten in AET

Alle x -Werte anmalen

until AET == empty and ET == empty

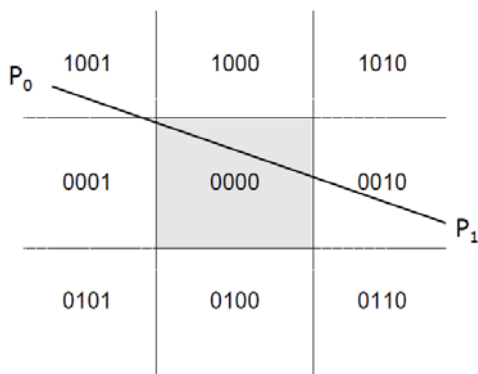
- Anti-Aliasing: Prefiltering (Je mehr von der Linie im Pixel ist desto dunkler) oder Supersampling (Pixel aufteilen und schauen was für Farbe dann zu einem Pixel mischen)

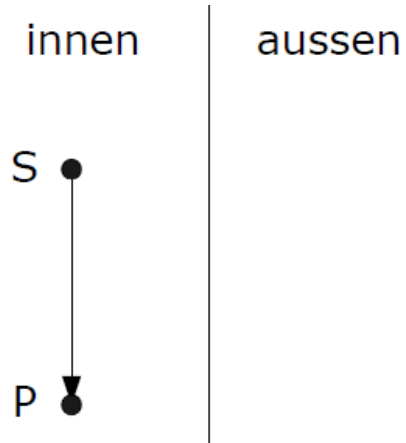
Clipping

Es gibt folgende Verfahren: Scissoring (Alle Pixel berechnen aber nur die zeichnen die im Fenster sind), Temporäre Buffer (z.B. Schrift können in Buffer vorberechnet werden und so besser geschnitten werden) und das Berechnen von Teilen.

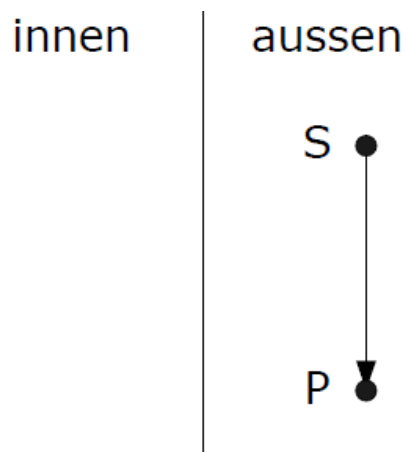
Cohen-Sutherland

Wenn Code verundet wird und 1 ergibt wird die Linie abgelehnt.

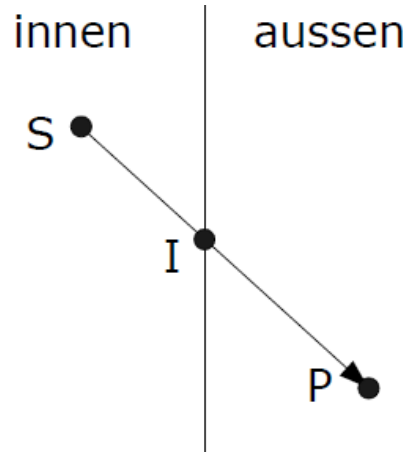




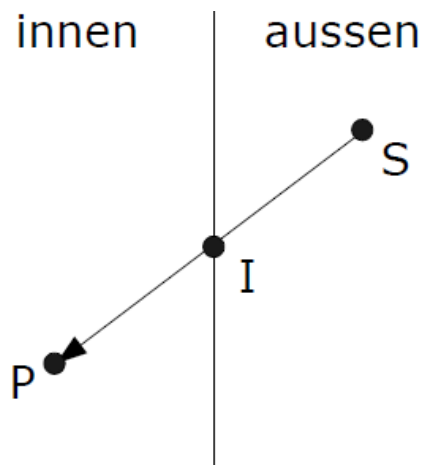
Ausgabe: P



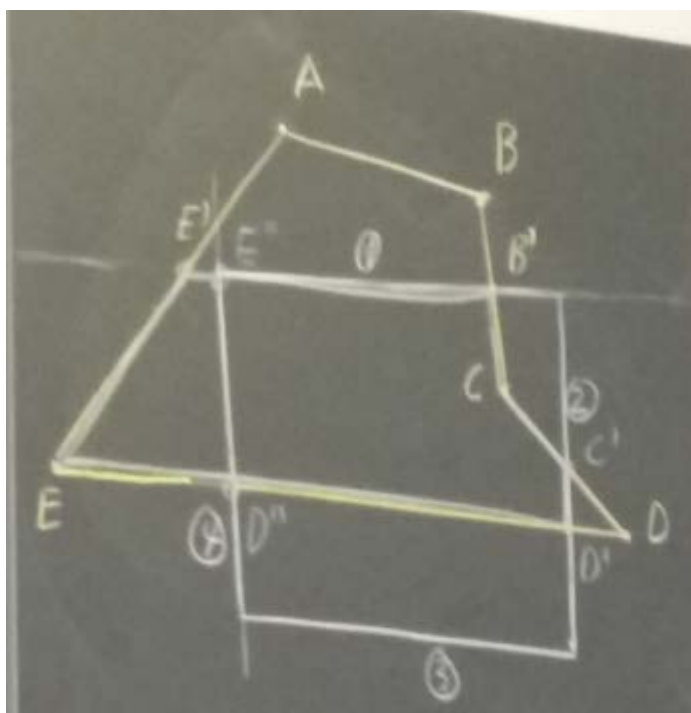
Ausgabe: keine



Ausgabe: I



Ausgabe: I und P



Es wird immer nur eine Gerade angeschaut!

1. - B' C D E E' (Ausgabe des Algorithmus)

2. C C' D' E E' B'

3. C C' D' E E' B' (alles innerhalb es ändert sich nichts)

4. C' D' D'' - E'' B' C

Algorithmen für verdeckte Linien und Flächen

Backface Culling (WebGL), Tiefensortierung (Painters Algorithms), Z-Buffer (WebGL), Warnock Algorithmus, Raycasting / Raytracing

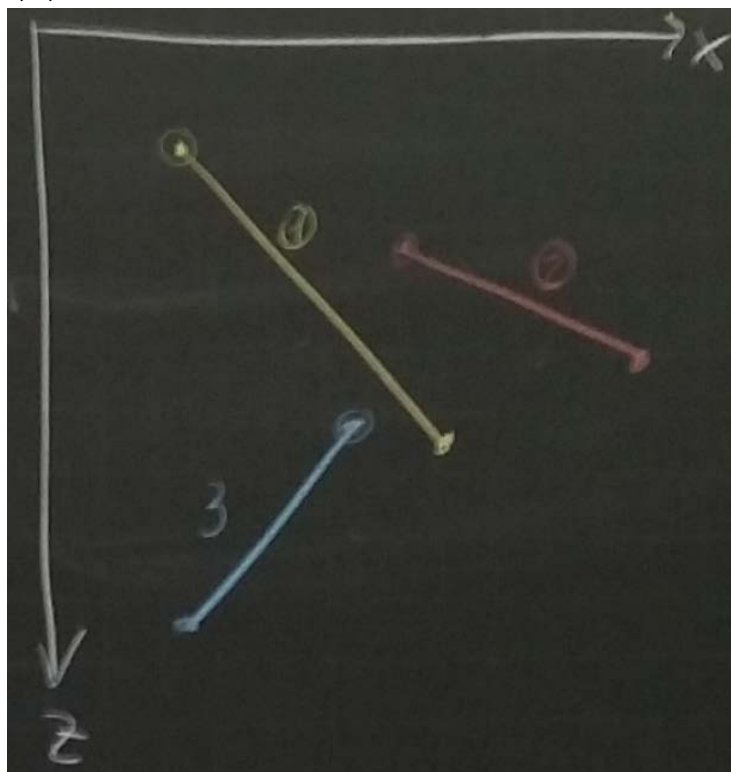
- Backface Culling: Wenn Normalenvektor in gleiche Richtung schaut wie Kamera dann muss Sie nicht gezeichnet werden, weil sie auf der Rückseite des Objektes ist.
 - Tiefensortierung Voraussetzung: transparente Objekte möglich, ineffizient für viele Objekte $O(n^2)$ nicht Hardware unterstützt
 - Polygon P ist in der Sortierung vor Q (es wird zuerst gezeichnet)
 - die Reihenfolge bleibt erhalten, falls eine der 5 folgenden Bedingungen erfüllt ist
1. Überlagern sich die x-Ausdehnungen nicht?
 2. Überlagern sich die y-Ausdehnungen nicht?
 3. Liegt P ganz auf der vom Betrachter abgewandten Seite von Q? (Also Q verdeckt P -> zeichne Q)
 4. Liegt Q ganz auf der Betrachterseite von P? (Also liegt Q komplett vor P -> zeichne zuerst P und dann Q)
 5. Überlappen sich die Polygone nicht auf der Projektion in die xy Ebene? (Wenn man von z auf die xy-Ebene schaut und sie sich nicht überlagern ist es egal wie man sie zeichnet.

Es ist auch möglich, dass keine gültige Reihenfolge gefunden wird, dann muss ein Polygon entlang der Ebene des anderen geschnitten werden

- Beispiel:
 1. Sortiere nach minimaler z-Koordinate Reihenfolge: 1, 2, 3

Man kann es so nicht zeichnen, weil wenn z.B. zuerst gelb und dann rot gezeichnet wird, dann wird das rote Polygon nicht vom gelben Polygon überlagert

2. Betrachte 1 (P) vs. 2 (Q) keine der 5 Bedingungen ist erfüllt also tauschen wir die Reihenfolge: 2, 1, 3
3. Betrachte 2 (P) vs. 1 (Q) Bedingung 3 trifft zu (P liegt ganz auf der abgewandten Seite von Q). Reihenfolge bleibt: 2, 1, 3
4. Betrachte 2 (P) vs. 3 (Q) Bedingung 1 trifft zu x überlappen sich nicht Reihenfolge bleibt: 2, 1, 3
5. Betrachte 1 (P) vs. 3 (Q) Bedingung 4 trifft zu weil Q ganz auf der Betrachtungsseite von P liegt. Reihenfolge bleibt: 2, 1, 3



- Z-Buffer: Pro Pixel wird die Farbe und die Tiefe (z) gespeichert

```
// Initialisierung
for y = 0 to YMAX do
  for x = 0 to XMAX do
    color[x,y] = Background
    depth[x,y] = MAXDEPTH
  end
end
```

Alles was wir zeichnen muss näher liegen als die MAXDEPTH

berechnen

```
// Polygone zeichnen
for alle Polygone q do
  for alle pixel p(x,y) in q do
    z = CalculateDepth(q,x,y);
    if z > depth[x,y] then
      depth[x,y] = z;
      color[x,y] = Color of q
    end
  end
end
```

Initialisieren auf MAXDEPTH

255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255

+

Sind alle näher (kleineres z) wird in Init hereingeschrieben

64	64	64	64	64	64	64	64
64	64	64	64	64	64	64	
64	64	64	64	64	64		
64	64	64	64				
64	64	64					
64	64						
64							
64							

=

64	64	64	64	64	64	64	255
64	64	64	64	64	64	255	255
64	64	64	64	64	255	255	255
64	64	64	64	255	255	255	255
64	64	64	255	255	255	255	255
64	64	255	255	255	255	255	255
64	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255

64	64	64	64	64	64	64	255
64	64	64	64	64	64	255	255
64	64	64	64	64	255	255	255
64	64	64	64	255	255	255	255
64	64	64	255	255	255	255	255
64	64	255	255	255	255	255	255
64	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255

+

127							
127	127						
127	127	127					
127	127	127	127				
127	127	127	127	127			

=

64	64	64	64	64	64	64	255
64	64	64	64	64	64	255	255
64	64	64	64	64	255	255	255
64	64	64	64	255	255	255	255
64	64	64	255	255	255	255	255
64	64	127	255	255	255	255	255
64	127	127	127	255	255	255	255
127	127	127	127	127	255	255	255

Der kleinere Wert wird immer verwendet

z-Buffer ist hardware unterstützt, konstante Berechenzeit, bei gleichen z-Werten Rundungsfehler.

$$z = \frac{-D - Ax - By}{C}$$

- Warnock Algorithmus: Zeichnet rekursiv nur einfache Bereiche. Ein Bereich ist einfach wenn:
 - Der Bereich kein Polygon enthält
 - Der Bereich nur ein Polygon enthält
 - Der Bereich ein Polygon enthält, das am nächsten liegt und den Bereich vollständig ausfüllt
 - Der Bereich nur aus einem Pixel besteht

Beleuchtung

- Korrektes Modell aufwendig, deshalb werden diffuse und spiegelnde Reflexion simuliert
- Raytracing und Radiosity verwenden zum Teil aufwendigere Beleuchtungsmodelle
- Farbe eines Objektes ist abhängig von Licht (Farbe und Intensität), Material (z.B. Porös) und Einfallswinkel
- Bei Diffuser Reflexion (Lambert) wird Licht gleichmässig abgestrahlt

Term für die diffuse Reflexion:

Farbe von Objekt

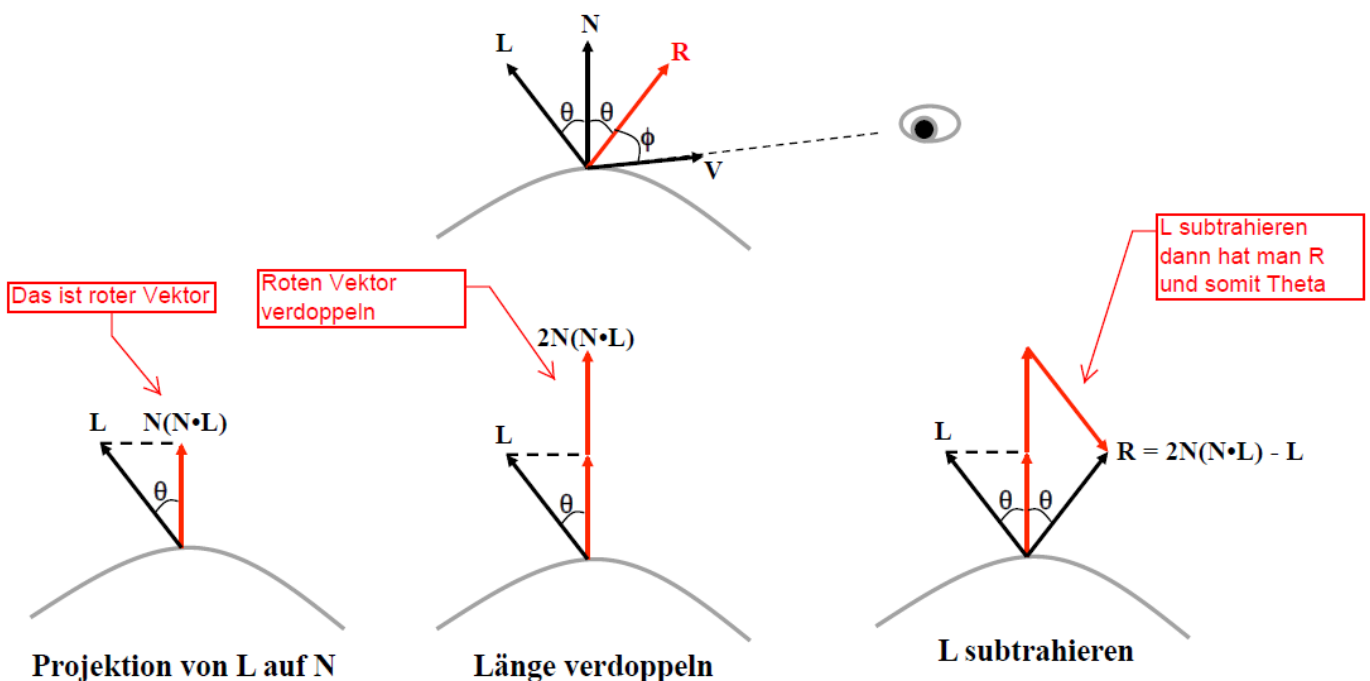
$$I_d = I_L \cdot k_d \cdot \cos \varphi$$

mit

$$\cos \varphi = \vec{N} \cdot \vec{L}$$

wobei N die Flächennormale,
 L die Richtung zur Lichtquelle,
 I_d die reflektierte Intensität und
 I_L die Intensität der Lichtquelle bezeichnet

- Bei der Spiegelnde Reflexion (Phong Modell) nimmt die Lichtintensität ab je grösser der Winkel vom Betrachter zur idealen Reflektionsrichtung ist.



Schattierung

- Die Schattierung bestimmt an welchen Orten die Beleuchtung berechnet wird
- Es gibt konstante Schattierung, Gouraud Schattierung und Phong Schattierung
- Konstante Schattierung:
 - Pro Polygon wird nur eine Farbe berechnet
 - Sieht blöd aus bei gekrümmten Objekten
- Gouraud Schattierung
 - Farbe wird für jeden Eckpunkt des Polygons berechnet
 - Im Innern wird durch lineare Interpolation die Farbe vermischt
- Phong Schattierung
 - Es wird die Beleuchtung für jedes Pixel berechnet
 - Macht man im Fragment Shader

Texture Mapping

- Problem ist Koordinatentransformationen zwischen Texture-, Modell und Kamera-Koordinatensystem
- Wenn man eine Kugel mit einer rechteckigen Textur abdecken möchte kann man folgende Formel verwenden:

Dieser Winkel ist für Kreis

Dieser Winkel ist für Kugel

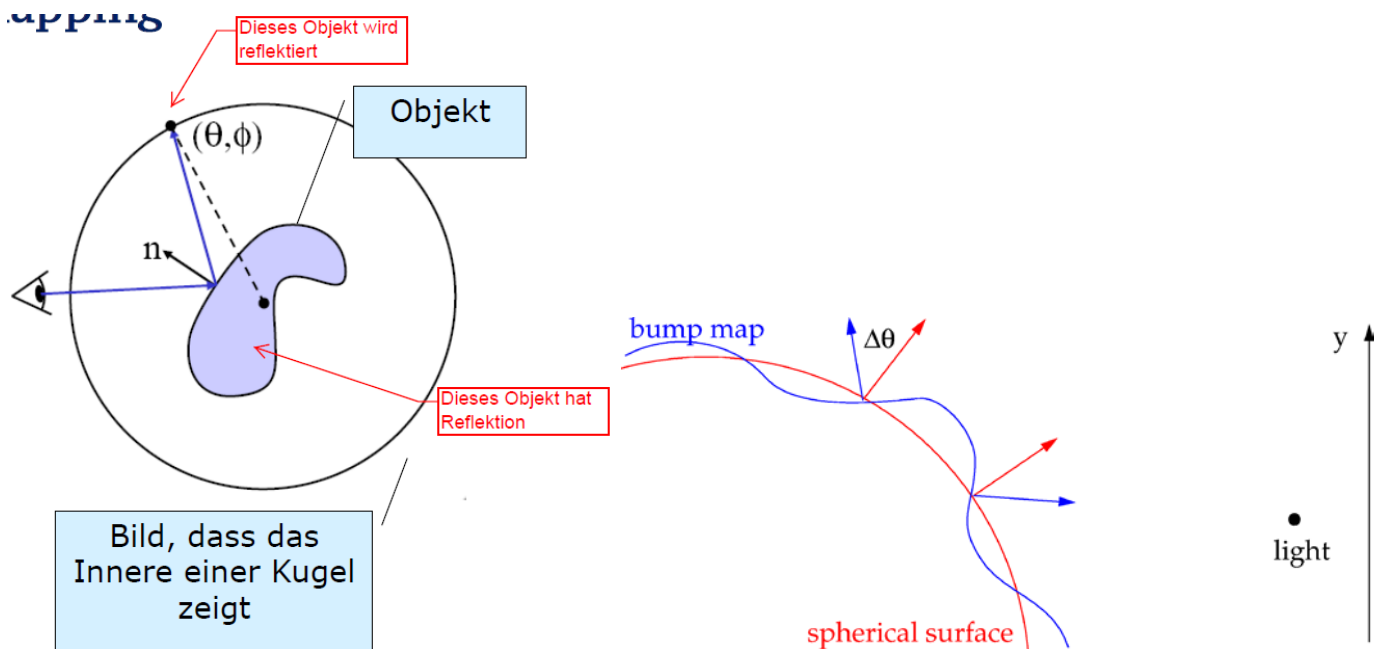
$$x = r \cos \phi \sin \varphi$$

$$y = r \sin \phi \cos \varphi$$

$$z = r \cos \varphi$$

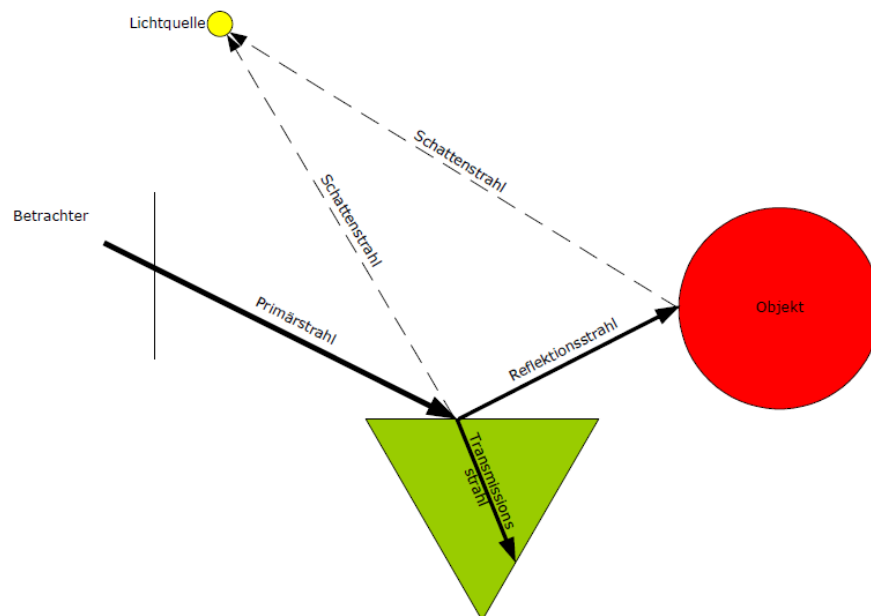
$$u = \frac{\phi}{2\pi} \quad v = \frac{\varphi}{2\pi}$$

- X, y, z sind die Koordinaten der Kugel und u, v sind die Koordinaten auf der Textur. Würde man einfach die x und y Koordinaten vom Kreis nehmen, würde die Textur verzerrt.
- Bei Vertices hat man anstatt die Farbe eines Vertices einfach die Texturkoordinante
- Texel ist immer Textur und Pixel ist das Polygon -> Magnification 1 Texel auf mehrere Pixel / Minification mehrere Texel auf ein Pixel
- Environment Mapping: Umgebung spiegelt sich in Objekt, Die Textur hat die Form einer Kugel
- Bump Mapping: Unebenheiten auf dem Objekt, der Normalenvektor wird verändert sodass die Beleuchtung anders wird. Das simuliert die Unebenheiten.



Raytracing

- Erzeugung von *photorealistischen Bildern* unter Berücksichtigung von Spiegelung und Transparenz
- Pro Pixel wird ein Strahl gelegt und das Objekt ermittelt, das zuerst geschnitten wird.
- Weitere Strahlen werden verwendet um zusätzliche Effekte zu erzielen, wie Spiegelung, Transparenz mit Refraktion und Schatten



Radiosity

- Globales Beleuchtungsmodell für korrekte diffuse Beleuchtung. Wird in Architektur und Design angewendet
- Keine Unterscheidung zwischen beleuchteten Flächen und Lichtquellen
- Berechnung des Energieflusses in einem geschlossenen System: Energieerhaltung
- Die Oberflächen der Objekte werden in *Patches* unterteilt. Ein Patch ist ein Polygon mit konstanter Lichtintensität. Der Lichttransfer zwischen den Patches wird durch ein System von linearen Gleichungen modelliert. Durch Lösung der Gleichungen wird die Intensität und Farbe pro Patch berechnet.

Web GL

- Vertex Processing: Berechnet die Position eines Vertex, Vorberechnungen von weiteren Daten für Fragment Processing möglich (Programm kann Attributes und Uniforms an Vertex übergeben)
- Fragment Processing: Berechnet die Farbe des Pixels (Füllt das Polygon). Programm kann Uniforms direkt an Fragment Shader übergeben. Vom Vertex an Fragment können Varying übergeben werden.
- Im Vertex Shader wird ProjectionMatrix mit ModelViewMatrix (Translation) multipliziert und mit dem Positionsvektor verrechnet
- Es gibt orthographische, frustum (near muss > 0 sein) und perspektivische Projektionen
- Der View Port bestimmt wo das Bild im Fenster angezeigt wird.