

Zusammenfassung

Computer Graphics

<https://github.com/wuethrich44/zusammenfassung-cg>

Fabian Wüthrich

Simon Erni

Tom Enz

Raphael Hodel

24. Januar 2016

1 Farbe

Was ist Farbe?

- *physikalisch* Zusammensetzung des Lichts, das auf das Auge trifft.
- *physiologisch* Wahrnehmung dieses Lichts im Auge und Interpretation

1.1 Physikalische Grundlagen

Licht besteht aus elektromagnetischer Strahlung verschiedener Wellenlänge.

sichtbares Licht Intervall von 380 nm (blau) bis 780 nm (rot)

infrarot oberhalb von 780nm

ultraviolet unterhalb von 380nm

Längeneinheit: 1nm = 10Å (Angström). Ein Atom hat etwa den Durchmesser von 1Å

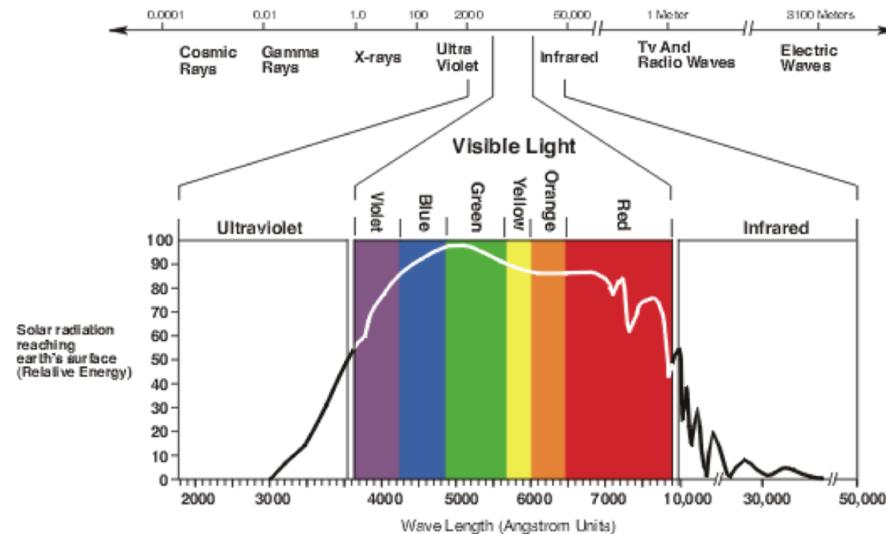


Abbildung 1.1: Farbspectrum

1.1.1 Begriffe

Spektralfarben haben genau eine Frequenz. Auch Spektralfarben haben Komplementärfarben.

Natürliches Licht ist ein Mix aus Frequenzen.

Spektrum ist die Verteilung der Frequenzen. Keine Frequenz sieht man Schwarz und wenn alle Frequenzen beteiligt, sieht man weiss. Wenn nur ein Teil der Frequenzen gibts eine Farbe.

Spektralverteilung Dadurch können verschiedene Lichtquellen charakterisiert/unterschieden werden. Diese gibt an wie viel Energie in jeder Wellenlänge abgestrahlt wird. Es gibt verschiedene Spektralverteilungen, die als *weiss* wahrgenommen werden.

Farbe einer Fläche Licht fällt von einer Lichtquelle auf ein Objekt und wird von diesem zum Auge (oder zur Kamera) reflektiert.

Komplementärfarben Sowohl der additiven als auch bei der subtraktiven Farbmischung gelten diejenigen Farben als komplementär, die miteinander gemischt einen neutralen Grauton ergeben. Beispiele: Rot → Cyan; Gelb → Blau; Grün → Magenta; Cyan → Rot; Blau → Gelb

Weitere Begriffe in der physikalischen Betrachtung Lichtstärke, Lichtstromdichte, Leuchtdichte, Spezifische Lichtausstrahlung

1.2 Farbwahrnehmung

Wir nehmen Farben anders wahr als sie z.B. durch ihre physikalischen Eigenschaften definiert sind.

Tristimulustheorie: Auf der Netzhaut befinden sich 3 Arten von Zellen (Zäpfchen), die auf Licht reagieren. Ihre höchste Empfindlichkeit liegt bei **rot, grün und blau**. Stäbchen und Zäpfchen bestimmen Auflösungsvermögen und Farbempfindlichkeit des Auges.

Stäbchen ($75 - 150 \times 10^6$) ermöglichen das Sehen bei niedriger Intensität. Nur Schwarz/Weiss-Sehen.

Zäpfchen ($6 - 7 \times 10^6$) sind Fotorezeptoren für Licht höherer Intensität und für Farben. Fotoropigmente sensibilisieren für rot (64%), grün (32%) und blau (4%). Lange, mittlere und kurze Zäpfchen für lange (rot), mittlere (grün) und kurze Wellenlängen (blau). Die Zäpfchen sprechen jedoch auf einen ganzen Bereich von Wellenlängen an, nicht nur genau auf die Wellenlängen von rot, grün und blau. Neutraler werden sie daher auch als S, M und L bezeichnet.

1.2.1 Wie sehen wir Farben

- Zäpfchen wirken wie ein Filter
- Farbe wird durch die Stärke der Antwort jeder Zäpfchensorte bestimmt: 3 Werte
- Damit kann nicht jede Spektraldichte repräsentiert werden
- Farben mit unterschiedlichen Spektren können also gleich aussehen: metamere Farben

WICHTIG:

Es ist also NICHT möglich jede Farbe mit rot, grün und blau zu mischen. Jedoch ein sehr grosser Anteil. Das Tristimulustheorie-Experiment zeigte, dass die Kurven der Wellenlängen auch negativ sein können. D.h. man müsste Farben auch subtrahieren - was nicht geht. Beim Tristimulus-Experiment hatten die Versuchspersonen drei Regler um die Farben Rot, Grün und Blau zu mischen. Der Versuchsleiter hatte einen Regler um durch sämtliche Spektralfarben zu gehen. Der Versuchsleiter stellte also eine Farbe ein und die Probanden mussten mit ihren drei Reglern versuchen, diese Farbe zu erreichen. Dies gelang nicht für alle Farben.

1.3 Farbsysteme

1.3.1 Normfarbtafel

Der Schnitt des 3D Farbraums mit der Ebene die durch $(1,0,0)$, $(0,1,0)$ und $(0,0,1)$ geht ergibt die Normfarbtafel von CIE. Sie enthält alle Farbtöne.

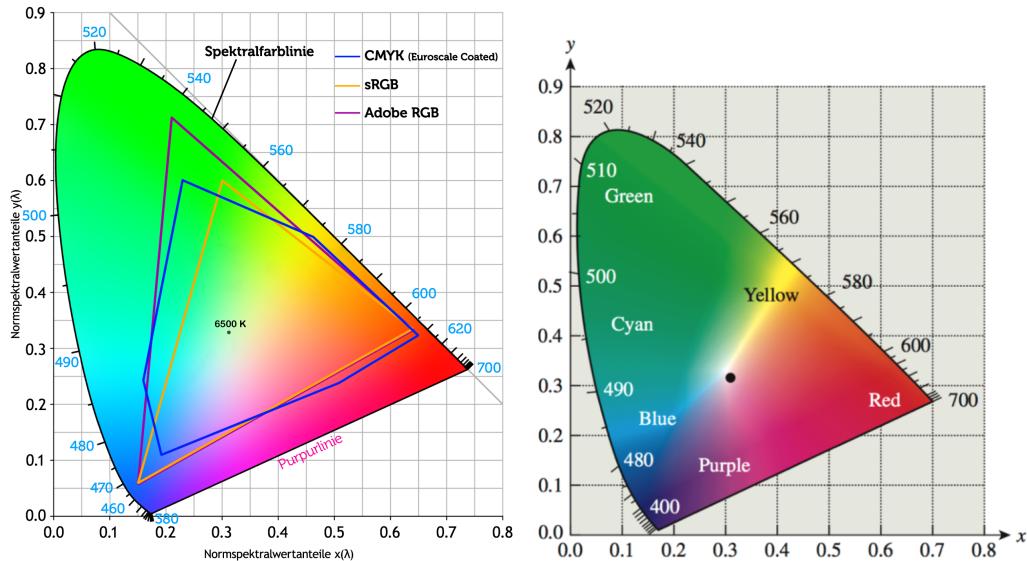


Abbildung 1.2: Normfarbtafel

Wichtige Eigenschaften

- Sie enthält alle sichtbaren Farben in vollen Intensität
- Spektralfarben befinden sich am Rand (siehe Spektralfarbenlinie in Abb. 1.2 links)
- Der gerade Teil der Umrandung heisst *Purpurlinie*
- **Mischen von Farben** Sind 2 Farben F_1 und F_2 auf der Tafel gegeben, so lassen sich mit ihnen alle Farben auf der Verbindungslinie zwischen F_1 und F_2 erzeugen.
- Dominante Wellenlänge kriegt man indem man die Farbe im Diagramm mit einem Punkt markiert. Dann zieht man vom Weiss eine Linie durch den Punkt bis an den Rand. Der Punkt am Rand ist dann die dominante Wellenlänge.
- Farben im unteren Teil des Diagramms besitzen keine dominante Wellenlänge und werden daher nicht spektral genannt.
- **Komplementärfarben** Farben die sich zu weiss addieren heissen Komplementärfarben. Das heisst, die Verbindungslinie zwischen zwei Komplementärfarben muss durch Weiss gehen.

Farben am Monitor

Ein bestimmter Monitor verwendet drei Farben (rot, grün, blau). Es können alle Farben in diesem Dreieck dargestellt werden (siehe Abb. 1.2 links) aber nicht alle der CIE Farbtafel.

1.3.2 Übersicht Farbsysteme

Die meisten Farbsysteme sind drei-dimensional und stellen eine Farbe also durch 3 Komponenten dar. Es gibt hardware orientierte Farbsysteme:

- RGB für Bildschirme
- CMY(K) für die Drucktechnik
- YIQ (NTSC, Fernsehen USA/Japan)
- YUV, YCrCb: Video/TV-Farbsysteme (PAL, NTSC, SECAM)

oder intuitive Farbsysteme wie

- HSV, HSI
- Munsell
- CIELab

1.3.3 RGB

Ist ein additives Farbsystem. Eine Farbe wird durch 3 Grundfarben rot (R), grün (G) und blau (B) dargestellt $C = (R, G, B)$. Wertebereich für RGB ist das Intervall $[0 \dots 1]$. Dies kann allerdings auf 8bit (1 Byte) $(0 \dots 255)$, 12bit oder 16bit abgebildet werden. Damit sind 16 Mio. Farben möglich.

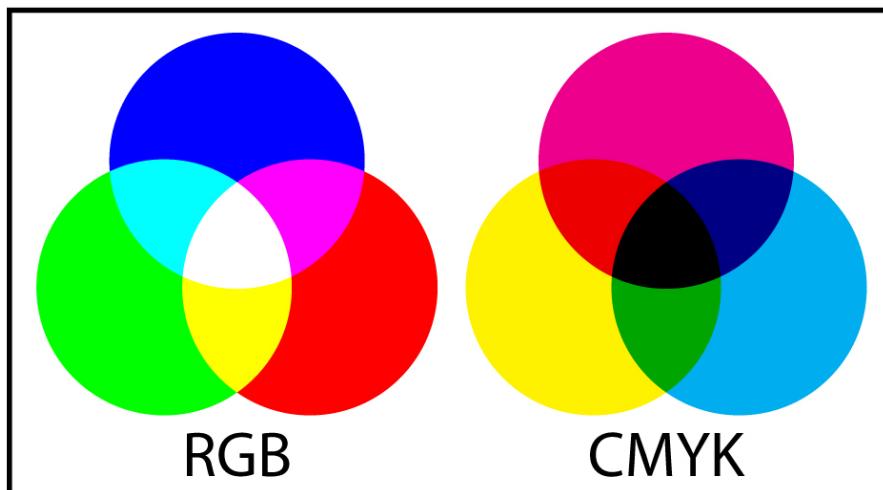


Abbildung 1.3: RGB (additives Farbsystem) vs. CMYK (subtraktives Farbssystem)

Additive Mischung

Bei der additiven Mischung von Farben empfängt das Auge die einzelnen Farben, der Farbeindruck entsteht durch die Addition der Spektren. Beispiel für additive Farbmischung:

- Monitore, Fernseher
- Scheinwerfer, Spotlights
- Kunst: Pointillismus

1.3.4 CMYK

Wird beim Drucken eine Farbe auf eine Oberfläche aufgetragen, so absorbiert sie ein Teil des Lichtes. Werden mehrere Farbe aufgetragen, so wird nur derjenige Teil des Lichtes reflektiert, der von keiner Farbe absorbiert wurde. Farben addieren sich also nicht, sondern werden vom Licht subtrahiert, man nennt das auch subtraktive Farbmischung.

CMY ist ein Farbsystem für die subtraktive Farbmischung, es besteht aus den Komplementärfarben von RGB : C = cyan, M = magenta und Y = yellow. Die nachfolgende Auflistung zeigt welche Farben absorbiert bzw. reflektiert werden. Möchte man z.B. dass Grün reflektiert werden müssen yellow und cyan gemischt werden (absorbieren Blau und Rot).

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

Der Zusammenhang zwischen RGB und CMY ist also:

$$\begin{pmatrix} C \\ M \\ Y \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} - \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

$$C = B + G = W - R$$

$$M = R + B = W - G$$

$$Y = G + R = W - B$$

Schwarz könnte zwar mittels C, M, Y gemischt werden, doch das Ergebnis ist schlecht (und auch zu teuer). Häufig wird deshalb noch Schwarz als vierte Farbe verwendet. Dieses System heisst CMYK (von blac**K**) und berechnet sich als:

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

$$C = C - K$$

$$M = M - K$$

$$Y = Y - K$$

1.3.5 HSV

HSV ist ein benutzerorientiertes Farbsystem, das sich nach der intuitiven Definition einer Farbe mittels Farbton, Sättigung und Helligkeit richtet:

H	Hue	Farbton, spektraler Teil der Farbe; bestimmt ob Farbe rot, gelb, grün, etc	in Grad von 0 bis 360 (R=0, G=120, B=240)
S	Saturation	Sättigung, bestimmt die Reinheit der Farbe.	zwischen 0 (vollständig ungesättigt = Grauton) und 1 (gesättigte Farbe)
V	Value	Helligkeit, Intensität.	zwischen 0 (schwarz) und 1 (volle Intensität)

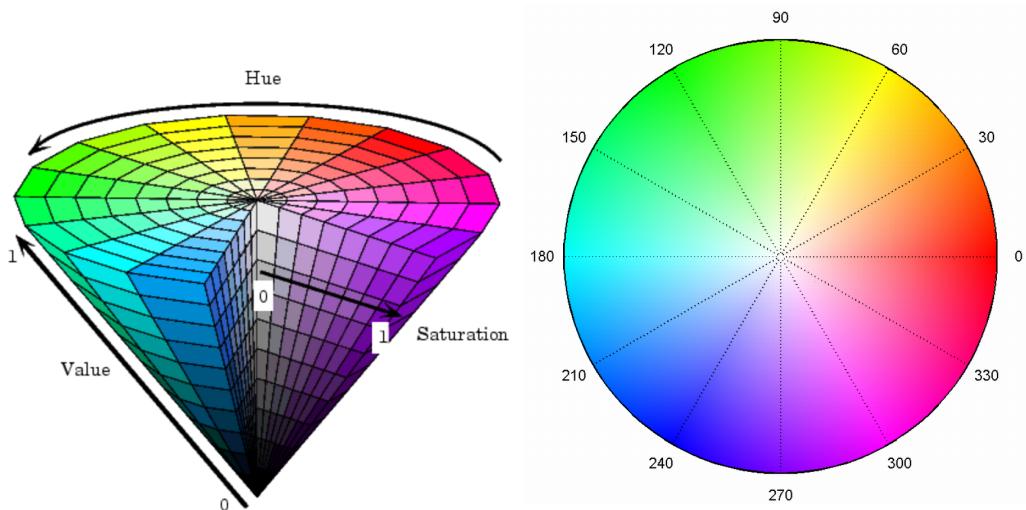


Abbildung 1.4: HSV

1.4 Halbtontechnik

1.4.1 Grauwerte

Der Grauwert einer RGB-Farbe ist nicht der Mittelwert von R, G und B sondern:

$$I = 0.299R + 0.587G + 0.114B$$

Farbe (in RGB)	Grauwert (in RGB)
(255,0,0)	(76,76,76)
(0,128,0)	(75,75,75)
(200,200,200)	(200,200,200)
(255,200,128)	(208,208,208)

1.4.2 Quantisierung

Einfache aber nicht akzeptable Lösung. Die im ursprünglichen Bild verwendeten Farben, werden auf die Anzahl vorhandenen Farben quantisiert.



Abbildung 1.5: Quantisierung

Beispiel Konvertierung der Intensitätsstufen:

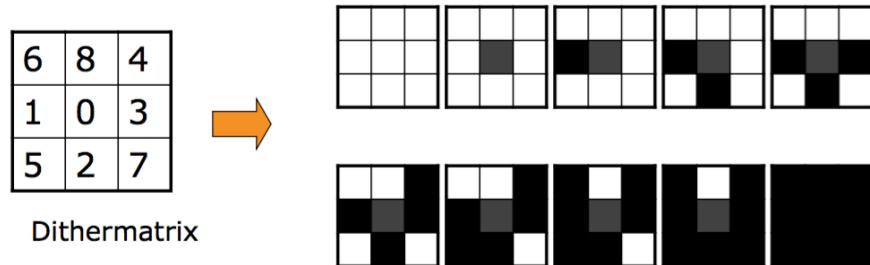
Gegeben ist ein Bild mit Graustufen von 0-255. Es soll nun in 10 Stufen (3x3 Matrix) dargestellt werden. $255/10 = 25.5$

30	206	250	240
6	56	120	141

1 ✓	8 ✓	9 ✓	9 ✓
0 ✓	2 ✓	4 ✓	5 ✓

1.4.3 Dithering

Dithering wird verwendet um Farben auf einem Gerät zu simulieren, das weniger Farben darstellen kann. Jeder Pixel wird durch eine $n \times n$ Matrix ersetzt. Die Bildgrösse wird also vergrössert. Die Anzahl Intenistätsstufen ergibt sich durch $(n \times n) + 1$. Das heisst bei: $2 \times 2 = 5 / 3 \times 3 = 10 / 4 \times 4 = 17$



Wichtige Regeln für Dithering

- Strukturen sollten möglichst vermieden werden. Keine horizontale Streifen erzeugen (3 4 5; 1 0 2; 5 7 8)
- Es sollten möglichst Kreise approximiert werden.
- Einmal gefärbte Punkte sollen im nächsten Level gefärbt bleiben (ist bei der Verwendung von Dither-Matrizen automatisch der Fall)

1.4.4 Dithering bei gleichbleibender Bildgrösse

Hierzu gibt es folgende Methoden:

Clustered dot dithering

Berechnen des Mittelwert einer $n \times n$ Region und ersetzen der Region durch die Dithermatrix

Dispersed dot dithering

Vergleich der einzelnen Pixel mit den Werten der Dithermatrix. Verwendet wird wieder eine $n \times n$ Matrix $D_{ij}^{(n)}$. Für jeden Punkt wird

$$\begin{aligned} i &= x \mod n \\ j &= y \mod n \end{aligned}$$

berechnet. Der Pixel wird gesetzt, falls seine Intensität grösser ist, als der Wert der Matrix.

$$I(x, y) > D_{ij}^{(n)}$$

Dieser Algorithmus wird Allgemein zur Reduktion der Graustufen/Farben verwendet, und nicht unbedingt nur zum Drucken von Bildern. Da ein Bildschirm individuelle Punkte viel besser darstellen kann als ein Drucker, muss die Matrix nicht allen oben beschriebenen Anforderungen genügen. Eine geeignete Matrix ist zum Beispiel die Bayes-Matrix

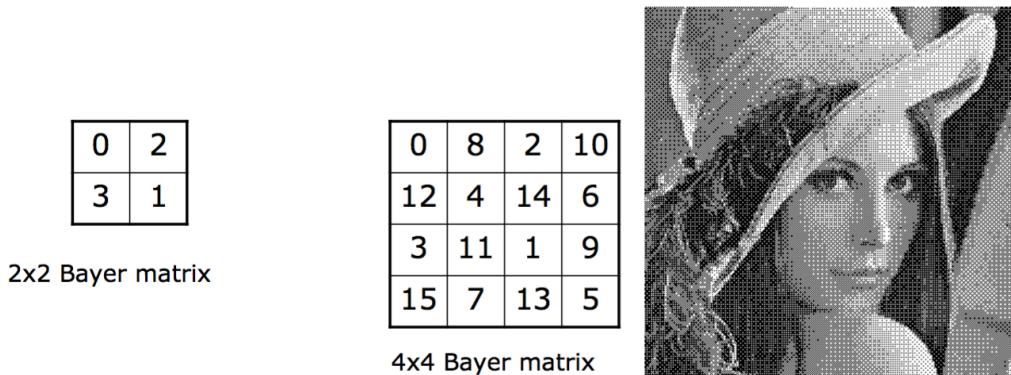


Abbildung 1.6: Bayes Matrix (links) & Dispersed Dot Dithering (rechts)

Dispersed Dot Dithering Quiz-Übung

Ein Bild ist in den Grauwerten von 0-255 gegeben und soll auf einem Bildschirm dargestellt werden der nur 2 Farben zur Verfügung hat. Die Grösse sollte dabei nicht verändert werden. Geben Sie eine gesetzten Pixel mit 1, ungesetzten mit 0 an. Man nehme die Ausgangstabelle:

Tabelle 1.1: Tabelle mit Pixelwerten von 0 - 255

30	200	250	240
6	56	120	141

Wir haben hier jetzt eine 2x2 Dithermatrix:

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

Wir müssen nun die Werte von der Ausgangstabelle 0 – 255 auf den Wertebereich der Matrix 0 – 3 herunterskalieren. Wir berechnen daher den Faktor, mit dem wir jeden Pixelwert multiplizieren um ihn auf diesen Wertebereich hinunterzubringen.

$$k = \frac{W_{max}}{n * n + 1}$$

W_{max} Maximalwert des Pixels (hier 255)

n Grösse (also Breite oder Höhe) der Matrix (hier 2)

k Resultierender Faktor

Der neue Wert eines Pixels berechnet sich dann einfach so:

$$W_{neu} = \frac{W_{alt}}{k}$$

Die erhaltenen Werte werden abgerundet. Wir erhalten dann folgende Zwischentabelle:

Tabelle 1.2: Tabelle für die Dithermatrix

0	3	4	4
0	1	2	2

Darüber legen wir nun einfach die Dithermatrix. Unsere Tabelle mit den Pixelwerten ist hier ja allerdings grösser als die Dithermatrix - dann legen wir sie einfach erneut hin, bis alles gefüllt ist. Mathematisch ausgedrückt setzen wir also hier eine 1 wenn folgender Ausdruck wahr ist:

$$i = x \bmod n$$

$$j = y \bmod n$$

x Ausgangs X-Position Pixel

y Ausgangs Y-Position Pixel

$$W_{x,y} > D_{i,j}$$

Also wenn der Wert der Zwischentabelle grösser ist als der Wert in der Dithermatrix. Das Modulo Zeugs ist nur zum Mapping von der x Position des Pixels auf die Position innerhalb der Dither Matrix.

Tabelle 1.3: Resultat

0	0	1	1
0	0	1	0

Error Diffusion

Anstatt Kreise verschiedener Grösse zu nehmen, können auch Punkte verschieden dicht angeordnet werden. (Frequenz-Verfahren). Die Idee dabei ist, dass der Fehler der durch das Setzen eines Pixels auf schwarz oder weiss gemacht wird, auf die umliegenden Pixel verteilt wird. Das Bild wird dabei sequentiell von oben nach unten und von links nach rechts durchlaufen. Der Fehler wird anhand der folgenden Gewichte auf die noch nicht besuchten Pixel verteilt:

Der Algorithmus lautet als (für jeden Pixel (x;y)):

Error-Diffusion (nach Floyd und Steinberg):

Vorgehen bei Aufgabe

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

$K = \text{Nearest}(I[x, y])$

$O[x, y] = K$

$\text{error} = I[x, y] - K$

$I[x+1, y] += 7 / 16 * \text{error}$

$I[x-1, y+1] += 1 / 16 * \text{error}$

$I[x, y+1] += 5 / 16 * \text{error}$

$I[x+1, y+1] += 3 / 16 * \text{error}$

1. Zu verarbeitenden Pixel anschauen
2. Bestimmen auf welchen Wert der Pixel gesetzt wird
3. Fehler berechnen (Fehler = Differenz aus Neuer Wert und Ursprünglicher Wert)
4. Den Rest gemäss Tabelle oben berechnen (Bsp. 7/16 * Fehler)
5. Rest gemäss Formel auf umliegende Felder verteilen (Neuer Wert des Feldes = ursprünglichen Wert - Rest)

1.5 Cheat-Sheet

1.5.1 Farbumrechnung

RGB -> HSV

The R,G,B values are divided by 255 to change the range from 0..255 to 0..1:

$$R' = R/255$$

$$G' = G/255$$

$$B' = B/255$$

$$C_{max} = \max(R', G', B')$$

$$C_{min} = \min(R', G', B')$$

$$\Delta = C_{max} - C_{min}$$

Hue calculation:

$$H = \begin{cases} 0^\circ & , \Delta = 0 \\ 60^\circ \times (\frac{G'-B'}{\Delta} \bmod 6) & , C_{max} = R' \\ 60^\circ \times (\frac{B'-R'}{\Delta} + 2) & , C_{max} = G' \\ 60^\circ \times (\frac{R'-G'}{\Delta} + 4) & , C_{max} = B' \end{cases}$$

Saturation calculation:

$$S = \begin{cases} 0 & , C_{max} = 0 \\ \frac{\Delta}{C_{max}} & , C_{max} \neq 0 \end{cases}$$

Value calculation:

$$V = C_{max}$$

CMYK -> RGB

The R,G,B values are given in the range of 0..255.

The red (R) color is calculated from the cyan (C) and black (K) colors:

$$R = 255 \times (1-C) \times (1-K)$$

The green color (G) is calculated from the magenta (M) and black (K) colors:

$$G = 255 \times (1-M) \times (1-K)$$

The blue color (B) is calculated from the yellow (Y) and black (K) colors:

$$B = 255 \times (1-Y) \times (1-K)$$

RGB -> CMYK

The R,G,B values are divided by 255 to change the range from 0..255 to 0..1:

$$R' = R/255$$

$$G' = G/255$$

$$B' = B/255$$

The black key (K) color is calculated from the red (R'), green (G') and blue (B') colors:

$$K = 1 - \max(R', G', B')$$

The cyan color (C) is calculated from the red (R') and black (K) colors:

$$C = (1 - R' - K) / (1 - K)$$

The magenta color (M) is calculated from the green (G') and black (K) colors:

$$M = (1 - G' - K) / (1 - K)$$

The yellow color (Y) is calculated from the blue (B') and black (K) colors:

$$Y = (1 - B' - K) / (1 - K)$$

HSV-> RGB

When $0 \leq H < 360$, $0 \leq S \leq 1$ and $0 \leq V \leq 1$:

$$C = V \times S$$

$$X = C \times (1 - |(H / 60^\circ) \bmod 2 - 1|)$$

$$m = V - C$$

$$(R', G', B') = \begin{cases} (C, X, 0) & , 0^\circ \leq H < 60^\circ \\ (X, C, 0) & , 60^\circ \leq H < 120^\circ \\ (0, C, X) & , 120^\circ \leq H < 180^\circ \\ (0, X, C) & , 180^\circ \leq H < 240^\circ \\ (X, 0, C) & , 240^\circ \leq H < 300^\circ \\ (C, 0, X) & , 300^\circ \leq H < 360^\circ \end{cases}$$

$$(R, G, B) = ((R' + m) \times 255, (G' + m) \times 255, (B' + m) \times 255)$$

1.5.2 Farbtabelle

Farbe	R	G	B	H °	S %	V %	C	M	Y	C	M	Y	K
Schwarz	0	0	0	0	0	0	1		1	0	0	0	1
Weiss	255	255	255	0	0	100	0		0	0	0	0	0
Rot	255	0	0		100	100	0		1	0	1	1	0
Grün	0	255	0	120	100	100	1		0	1	0	1	0
Blau	0	0	255	240	100	100	1		1	0	1	0	0
Yellow	255	255	0	600	100	100	0		0	0	0	1	0
Cyan	0	255	255	180	100	100	1		0	1	0	0	0
Magenta	255	0	255	300	100	100	0		0	0	1	0	0
Dunkelgrün	0	50	0	120	100	19.6	1	0.803921568627451	1	1	0	1	0.804
Braun	36	20	9	24	75	14.1	0.8588235294117648	0.9215686274509804	0.9647058823529412	0	0.444	0.75	0.859
Grau	190	190	190	0	0	74.5				0	0	0	0.255
Orange	255	165	0	39	100	100	0	0.3529411764705882	1	0	0.353	1	0

2 Projektive Geometrie

2.1 Hessesche Normalform

2.1.1 der Gerade

Die Gleichung

$$\frac{Ax + By + C}{\sqrt{A^2 + B^2}} = 0$$

heisst hessische Normalform der **Gerade**.

Wird die Gleichung erfüllt, liegt der Punkt auf der Geraden, andernfalls resultiert der Abstand zur Geraden (inkl. auf welcher Seite).

Definition

Die Gleichung wird genau von den Punkten erfüllt, die auf der Geraden (welche die Gleichung beschreibt) liegen. Liegt der Punkt nicht auf der Geraden, so liefert die Hesse'sche Normalenform den Abstand vom Punkt zur Geraden. Das Vorzeichen gibt an, ob der Punkt auf der Seite der Geraden liegt, in deren Richtung der Normalenvektor zeigt, oder ob er auf der anderen Seite der Geraden liegt.

Beispiel Geradengleichung

Wir betrachten die Geradengleichung

$$3x + y - 5 = 0$$

und bestimmen die HN:

$$\frac{3x + y - 5}{\sqrt{3^2 + 1^2}} \text{ resp. } \frac{3x + y - 5}{\sqrt{10}}$$

Wir testen die folgenden Punkte

$$S(3; -4) T(2; -5) U(2; 5)$$

Der Punkt S in die HN eingesetzt ergibt:

$$\frac{3 \cdot 3 + (-4) - 5}{\sqrt{10}} = \frac{9 - 9}{\sqrt{10}} = 0$$

, somit liegt S auf der Geraden.

Der Punkt T in die HN eingesetzt ergibt:

$$\frac{3 \cdot 2 + (-5) - 5}{\sqrt{10}} = \frac{6 - 10}{\sqrt{10}} = -1.265$$

, somit liegt T nicht auf der Geraden. Der Punkt T hat einen Abstand von 1,256 LE von der Geraden und liegt dem Normalenvektor entgegengesetzt.

Der Punkt U in die HN eingesetzt ergibt:

$$\frac{3 \cdot 2 + 5 - 5}{\sqrt{10}} = \frac{6}{\sqrt{10}} = 1.897$$

, somit liegt U nicht auf der Geraden. Der Punkt U hat einen Abstand von 1,897 LE von der Geraden und liegt auf der „positiven“ Seite der Geraden.

2.1.2 zur Ebene

Die Gleichung

$$\frac{Ax + By + Cz + D}{\sqrt{A^2 + B^2 + C^2}} = 0$$

heisst hessische Normalform der **Ebene**.

Definition

- Ergibt das Einsetzen eines Punktes in der HN = 0, so liegt der Punkt in der Ebene.
- Ergibt das Einsetzen eines Punktes in der HN $d > 0$, so liegt der Punkt auf der gleichen Seite des Normalenvektors $\vec{n}_E = \begin{pmatrix} A \\ B \\ C \end{pmatrix}$ mit Abstand d zur Ebene.
- Ergibt das Einsetzen eines Punktes in der HN $d < 0$, so liegt der Punkt auf der entgegengesetzten Seite des Normalenvektors $\vec{n}_E = \begin{pmatrix} A \\ B \\ C \end{pmatrix}$ mit Abstand $|d|$ zur Ebene.

Beispiel

Gegeben ist die Koordinatengleichung der Ebene E: $x - 4 \cdot y + 2 \cdot z + 9 = 0$
Gesucht der Abstand der Punkte $Q(3; -1; -4)$, $R(5; 6; -7)$ und $S(3; 7; 8)$ zur Ebene.

1. Schritt: Bestimmen der HN der Ebene

$$\text{HN: } \frac{Ax + By + Cz + D}{\sqrt{A^2 + B^2 + C^2}} = \frac{x - 4 \cdot y + 2 \cdot z + 9}{\sqrt{1^2 + (-4)^2 + 2^2}} = \frac{x - 4 \cdot y + 2 \cdot z + 9}{\sqrt{21}} = 0$$

2. Schritt: Einsetzen der Punkte:

$Q(3; -1; -4) :$

$$\frac{x - 4 \cdot y + 2 \cdot z + 9}{\sqrt{21}} = \frac{3 - 4 \cdot (-1) + 2 \cdot (-4) + 9}{\sqrt{21}} = \frac{8}{\sqrt{21}} = 1.746$$

Der Punkt liegt also dem Normalenvektor $\vec{n}_E = \begin{pmatrix} 1 \\ -4 \\ 2 \end{pmatrix}$ gleich gesetzten Seite der Ebene mit dem Abstand 1.746 LE zur Ebene.

$R(5; 6; -7) :$

$$\frac{5 - 4 \cdot 6 + 2 \cdot (-7) + 9}{\sqrt{21}} = \frac{-24}{\sqrt{21}} = -5.237$$

Der Punkt liegt also dem Normalenvektor \vec{n}_E entgegengesetzte Seite der Ebene mit dem Abstand 5.237 LE zur Ebene.

$S(3; 7; 8) :$

$$\frac{3 - 4 \cdot 7 + 2 \cdot 8 + 9}{\sqrt{21}} = \frac{0}{\sqrt{21}} = 0$$

Der Punkt liegt auf der Ebene (es ist ja auch unser ursprünglicher Punkt C.)

2.2 Skalare und Vektoren und das Kartesisches Koordinatensystem

Skalar ist eine reelle (oder komplexe) Zahl. Beispiele: Temperatur, Druck, Luftfeuchtigkeit.

Vektor hat einen (reellen) Betrag und eine Richtung. Beispiele: (Wind-) Geschwindigkeit (an einem Ort), Kraft (auf ein Objekt), Fliessgeschwindigkeit in Gewässern, Elektrisches Feld, Magnetfeld, Gravitationsfeld, etc.

2.2.1 Addition von Vektoren und Multiplikation mit einem Skalar

Gegeben sind die Vektoren: $\vec{a} = \begin{pmatrix} a_1 \\ a_2 \end{pmatrix}$ und $\vec{b} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$

Addition: Zwei Vektoren \vec{a} und \vec{b} addieren heisst entsprechende Komponenten addieren.

$$\vec{a} + \vec{b} = \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} = \begin{pmatrix} a_1 + b_1 \\ a_2 + b_2 \end{pmatrix}$$

Multiplikation mit Skalar: Einen Vektor \vec{a} mit einem Skalar $\lambda \in \mathbb{R}$ multiplizieren.

$$\lambda \vec{a} = \lambda \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} \lambda a_1 \\ \lambda a_2 \end{pmatrix}$$

2.2.2 Das Inverse eines Vektors und der Nullvektor

Inverse: Das Inverse $-\vec{a}$ des Vektors $\vec{a} = \begin{pmatrix} a_1 \\ a_2 \end{pmatrix}$ ist der Vektor mit den negativen Komponenten:

$$-\vec{a} = -\begin{pmatrix} a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} -a_1 \\ -a_2 \end{pmatrix}$$

Nullvektor: Der Nullvektor $\vec{0}$ ist ein Vektor dessen Komponenten alle verschwinden (also gleich Null

$$\text{sind}): \vec{0} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

Damit wird die Subtraktion des Vektors \vec{b} vom Vektor \vec{a} wie folgt definiert: $\vec{a} - \vec{b} = \vec{a} + (-\vec{b})$

Rechenregeln:

$$\vec{a} + \vec{b} = \vec{b} + \vec{a} \text{ Kommutativgesetz}$$

$$\vec{a} + (\vec{b} + \vec{c}) = (\vec{a} + \vec{b}) + \vec{c} \text{ Assoziativgesetz}$$

$$\vec{a} + \vec{0} = \vec{a} \text{ Existenz eines Neutralelements } \vec{0}$$

$$\vec{a} + -\vec{a} = \vec{0}$$

$$\lambda(\vec{a} + \vec{b}) = \lambda\vec{a} + \lambda\vec{b}$$

$$(\lambda + \mu)\vec{a} = \lambda\vec{a} + \mu\vec{a}$$

$$(\lambda\mu)\vec{a} = \lambda(\mu\vec{a}) = \mu(\lambda\vec{a})$$

2.2.3 Geometrische Interpretation

Zwei Vektoren sind gleich, wenn ihre Komponenten gleich sind! Achtung: In der Physik darf man z.B. Kraftvektoren nicht einfach verschieben!

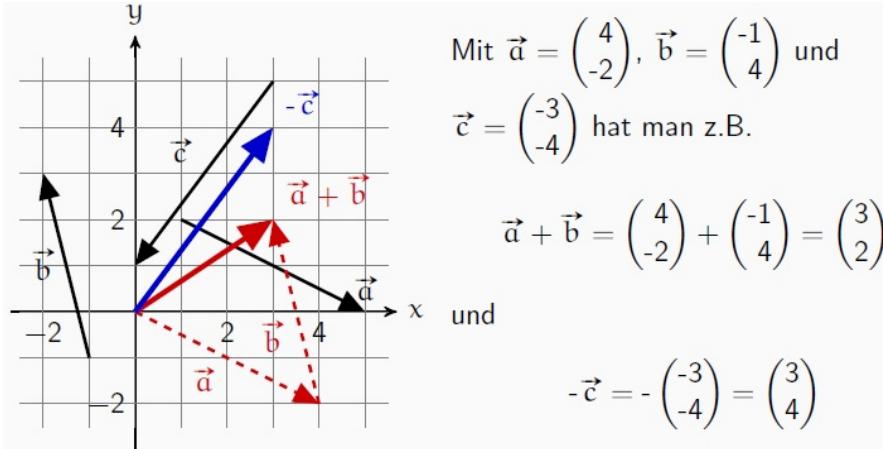


Abbildung 2.1: Geometrische Interpretation

Es folgt nun ein Example zur Berechnung von Vektoren. Bestimmen Sie $3\vec{a} - 2\vec{b} + \vec{c}$ sowohl grafisch wie auch rechnerisch (analytisch). Details der Vektoren sind in Abbildung 2.1 zu entnehmen.

$$3 \begin{pmatrix} 4 \\ -2 \end{pmatrix} - 2 \begin{pmatrix} -1 \\ 4 \end{pmatrix} + \begin{pmatrix} -3 \\ -4 \end{pmatrix} = \begin{pmatrix} 12 \\ -6 \end{pmatrix} - \begin{pmatrix} -2 \\ 8 \end{pmatrix} + \begin{pmatrix} -3 \\ -4 \end{pmatrix} = \begin{pmatrix} 11 \\ -18 \end{pmatrix}$$

Basisvektoren: Die Basisvektoren \vec{e}_x und \vec{e}_y sind orthogonal und haben die Länge 1, d.h. $\vec{e}_x \cdot \vec{e}_x = 0$ und $|\vec{e}_x| = 1$ sowie $|\vec{e}_y| = 1$.

2.3 Skalarprodukt

Das Skalarprodukt zweier Vektoren \vec{a} und \vec{b} ist wie folgt definiert:

$$\vec{a} \cdot \vec{b} = |\vec{a}| |\vec{b}| \cos(\phi)$$

In kartesischen Koordinaten gilt:

$$\vec{a} \cdot \vec{b} = \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} = a_1 b_1 + a_2 b_2$$

Es folgt ein Beispiel dazu:

Gegeben sind die Vektoren $\vec{a} = \begin{pmatrix} 4 \\ -2 \end{pmatrix}$ und $\vec{b} = \begin{pmatrix} -1 \\ 4 \end{pmatrix}$ man berechne nun das Skalarprodukt sowie den Winkel ϕ .

$$\vec{a} \cdot \vec{b} = \begin{pmatrix} 4 \\ -2 \end{pmatrix} + \begin{pmatrix} -1 \\ 4 \end{pmatrix} = 4(-2) + (-1)4 = -4 - 8 = -12$$

$$a = |\vec{a}| = \sqrt{a_1^2 + a_2^2} = \sqrt{4^2 + (-2)^2} = \sqrt{20}$$

$$b = |\vec{b}| = \sqrt{b_1^2 + b_2^2} = \sqrt{(-1)^2 + 4^2} = \sqrt{17}$$

$$\cos(\phi) = \frac{\vec{a} \cdot \vec{b}}{|\vec{a}| |\vec{b}|} = \frac{-12}{\sqrt{20} \sqrt{17}} = -0.6508$$

Daraus folgt mit $\arccos(\phi)$: $\phi = \arccos(-0.6508) = 130.6^\circ$

$|\vec{a}|$ ist die Länge des Vectors \vec{a} .

Rechengesetze

$$\vec{a} \cdot \vec{b} = \vec{b} \cdot \vec{a}$$

Kommutativgesetz

$$\vec{a} \cdot (\vec{b} + \vec{c}) = \vec{a} \cdot \vec{b} + \vec{a} \cdot \vec{c}$$

Distributivgesetz

$$\lambda(\vec{a} \cdot \vec{b}) = (\lambda\vec{a}) \cdot \vec{b} = \vec{a} \cdot (\lambda\vec{b})$$

Orthogonale Vektoren Zwei Vektoren \vec{a} und \vec{b} stehen genau dann senkrecht aufeinander, sind also orthogonal, falls ihr Skalarprodukt verschwindet respektive 0 ist.

$$\vec{a} \cdot \vec{b} = 0 \iff \vec{a} \perp \vec{b}$$

Angenommen ein der Vektor $\vec{a} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$ sei gegeben. Um einen dazugehörenden orthogonalen Vektor zu erhalten muss folgende Formel aufgelöst werden:

$$2b_1 + 1b_2 = 0$$

Ein möglicher Vektor wäre also $\vec{b} = \begin{pmatrix} -1 \\ 2 \end{pmatrix}$ oder ein Vielfaches davon!

2.4 Spatprodukt

Das Spatprodukt $[\vec{a}, \vec{b}, \vec{c}]$ der drei Vektoren \vec{a} , \vec{b} und \vec{c} ist das Skalar $[\vec{a}, \vec{b}, \vec{c}] = \vec{a} \cdot (\vec{b} \times \vec{c})$. Der Betrag des Spatprodukts $|[\vec{a}, \vec{b}, \vec{c}]|$ ist das Volumen des durch die drei Vektoren \vec{a} , \vec{b} und \vec{c} aufgespannten Spats.

Das Spatprodukt ist gleich der Determinante einer 3×3 -Matrix:

$$[\vec{a}, \vec{b}, \vec{c}] = \begin{vmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{vmatrix}$$

Die Berechnung erfolgt nach der Regel von Sarrus:

$$[\vec{a}, \vec{b}, \vec{c}] = \begin{array}{c|cc|cc} & + & + & + & \\ \begin{matrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{matrix} & \left| \begin{matrix} a_1 & a_2 \\ b_1 & b_2 \\ c_1 & c_2 \end{matrix} \right. & \left| \begin{matrix} a_1 & a_2 \\ b_1 & b_2 \\ c_1 & c_2 \end{matrix} \right. & \end{array}$$

$$[\vec{a}, \vec{b}, \vec{c}] = a_1 b_2 c_3 + a_2 b_3 c_1 + a_3 b_1 c_2 - a_3 b_2 c_1 - a_1 b_3 c_2 - a_2 b_1 c_3$$

Beachte: $[\vec{a}, \vec{b}, \vec{c}] = 0 \Leftrightarrow \vec{a}, \vec{b}$ u. \vec{c} sind komplanar (linear abhängig).

Abbildung 2.2: Spatprodukt

Rechenregeln für das Spatprodukt

Vertauschen von zwei Vektoren bewirkt einen Vorzeichenwechsel: z.B.

$$[\vec{a}, \vec{b}, \vec{c}] = -[\vec{b}, \vec{a}, \vec{c}]$$

Zyklisches Vertauschen der drei Vektoren ändert nichts:

$$[\vec{a}, \vec{b}, \vec{c}] = [\vec{b}, \vec{c}, \vec{a}] = [\vec{c}, \vec{a}, \vec{b}]$$

Multiplikation der Vektoren mit reellen Zahlen λ, μ, ν :

$$[\lambda\vec{a}, \mu\vec{b}, \nu\vec{c}] = \lambda\mu\nu[\vec{a}, \vec{b}, \vec{c}]$$

Addition zweier Vektoren

$$[\vec{a} + \vec{b}, \vec{c}, \vec{d}] = [\vec{a}, \vec{c}, \vec{d}] + [\vec{b}, \vec{c}, \vec{d}]$$

2.5 Transformation: Translation in 2D

Um eine Figur in eine Richtung zu verschieben, kann man alle ihre Punkte als Matrix zusammennehmen, diese in eine höhere Dimension nehmen (in homogene Koordinaten umschreiben) und die Translation in derselben Dimension vornehmen, indem man in der rechten Spalte der Einheitsmatrix die Verschiebungen vornimmt.

2.5.1 Beispiel Translation 2D

Gegeben sind die Punkte $A = (3, 1)$, $B = (6, 1)$ und $C = (5, 4)$ und die Verschiebungsrichtung $\vec{t} = \begin{pmatrix} 1 & 2 \end{pmatrix}$.

Man nimmt nun die Punkte alle zusammen in einer Matrix und erweitert diese die homogenen Koordinaten mit dem Wert 1 in der dritten Dimension (dargestellt unterhalb des Strichs in der Matrix):

$$Points = \begin{pmatrix} A_x & B_x & C_x \\ A_y & B_y & C_y \\ \hline 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 3 & 6 & 5 \\ 1 & 1 & 4 \\ \hline 1 & 1 & 1 \end{pmatrix}$$

Den Verschiebungsvektor macht man nun zur Matrix. Dazu wird die Einheitsmatrix der neuen homogenen Dimension genommen und als rechteste Spalte wird der Vektor \vec{t} eingesetzt. \vec{t} wird als zu

$$T = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix}$$

Nun muss man nur noch $T \cdot M$ berechnen:

$$\begin{aligned} T \cdot Points &= \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 3 & 6 & 5 \\ 1 & 1 & 4 \\ \hline 1 & 1 & 1 \end{pmatrix} \\ &= \begin{pmatrix} A'_x & B'_x & C'_x \\ A'_y & B'_y & C'_y \\ \hline 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 4 & 7 & 6 \\ 3 & 3 & 6 \\ \hline 1 & 1 & 1 \end{pmatrix} \end{aligned} \tag{2.1}$$

Die Punkte der neuen Koordinaten sind also $A' = (4, 3)$, $B' = (7, 3)$ und $C' = (6, 6)$.

2.6 Transformation: Rotation um einen Winkel Φ in 2D

Die Rotation um wird vorgenommen, indem folgende Matrix verwendet wird:

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

Die Umkehrung davon ist einfach die transponierte Matrix.

2.7 Transformation: Rotation in 3D

2.7.1 Rotation einen Winkel Φ um die z-Achse

$$\mathbf{R}_z(\Phi_z) = \begin{pmatrix} \cos(\Phi) & -\sin(\Phi) & 0 & 0 \\ \sin(\Phi) & \cos(\Phi) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Inverse davon ist: $\mathbf{R}_z^{-1}(\Phi_z) = \begin{pmatrix} \cos(\Phi) & \sin(\Phi) & 0 & 0 \\ -\sin(\Phi) & \cos(\Phi) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

2.7.2 Rotation einen Winkel Φ um die y-Achse

$$\mathbf{R}_y(\Phi_y) = \begin{pmatrix} \cos(\Phi) & 0 & \sin(\Phi) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\Phi) & 0 & \cos(\Phi) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Inverse davon ist: $\mathbf{R}_y^{-1}(\Phi_y) = \begin{pmatrix} \cos(\Phi) & 0 & -\sin(\Phi) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(\Phi) & 0 & \cos(\Phi) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

2.7.3 Rotation einen Winkel Φ um die x-Achse

$$\mathbf{R}_x(\Phi_x) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\Phi) & -\sin(\Phi) & 0 \\ 0 & \sin(\Phi) & \cos(\Phi) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Inverse davon ist: $\mathbf{R}_x^{-1}(\Phi_x) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\Phi) & \sin(\Phi) & 0 \\ 0 & -\sin(\Phi) & \cos(\Phi) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

2.7.4 Rotation um eine beliebige Achse

1. Rotation um den Winkel Φ um die z-Achse (hier Matrix D genannt)
2. Rotation um den Winkel Θ um die frühere x-Achse mit der Matrix C.
3. Rotation um den Winkel Ψ um die frühere z-Achse mit der Matrix B.

Man hat nun die Matrizen:

$$\mathbf{D} = \begin{pmatrix} c_\phi & s_\phi & 0 \\ -s_\phi & c_\phi & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad \mathbf{C} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & c_\theta & s_\theta \\ 0 & -s_\theta & c_\theta \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} c_\psi & s_\psi & 0 \\ -s_\psi & c_\psi & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Abbildung 2.3: Rotationsmatrizen

Diese kann man als eine Transformation zusammennehmen durch Multiplikation:

$$\mathbf{A} = \mathbf{BCD}$$

$$= \begin{pmatrix} c_\psi c_\phi - c_\theta s_\phi s_\psi & c_\psi s_\phi + c_\theta c_\phi s_\psi & s_\psi s_\theta \\ -s_\psi c_\phi - c_\theta s_\phi c_\psi & -s_\psi s_\phi + c_\theta c_\phi c_\psi & c_\psi s_\theta \\ s_\theta s_\phi & -s_\theta c_\phi & c_\theta \end{pmatrix}$$

Abbildung 2.4: Zusammengesetzte Rotationsmatrix

2.8 Transformation: Spiegelung an einer Geraden

Eine direkte Spiegelung ist nur möglich, wenn die Gerade durch den Ursprung geht. Tut sie dies nicht, muss zuerst eine Translation durchgeführt werden. Dies macht man folgendermassen.

Einer Geraden g sieht man gleich zu Beginn an, ob sie durch den Nullpunkt geht, nämlich wenn sie eine Verschiebungskonstante hat, also in der Form $g = ax + by + c$ mit $c \neq 0$ vorliegt.

Nach der Verschiebung findet die eigentliche Spiegelung statt. Dazu muss man zuerst herausfinden, in welchem Winkel die Gerade zum Ursprung steht. Dazu verwendet man den Tangens, weil bekannt ist, in welchem Verhältnis x zu y steht, nämlich mit der Steigung m (vergleiche dazu die Gleichung $y = mx + q$)! Man kann also den Arcus-Tangens verwenden, um δ herauszufinden:

$$\begin{aligned} \tan(\delta) &= m \\ \delta &= \arctan(m) \end{aligned} \tag{2.2}$$

Man könnte auch direkt in Kosinus und Sinus umwandeln mit der Formel:

$$\begin{aligned} \cos(2\delta) &= \frac{1 - \tan^2(\delta)}{1 + \tan^2(\delta)} \\ \sin(2\delta) &= \frac{2 \tan^2(\delta)}{1 + \tan^2(\delta)} \end{aligned} \tag{2.3}$$

Danach verwendet man die **Nullpunkt-Spiegelungsmatrix**:

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

Jetzt muss man nur noch die Translation rückgängig machen mit T^{-1} (Inverse von T). Zusammengesetzt ergibt sich die **Spiegelungsmatrix** $\sigma = T^{-1} * \sigma' * T$.

Die **Bildpunkte** erhält man durch Multiplikation der Punkte mit der Spiegelungsmatrix:

$$A' = \sigma * A$$

2.8.1 Beispiel

Gegeben ist die Gerade $g = 2x - 3y + 2$. Wir wollen die Spiegelung σ an dieser Geraden betrachten. Gesucht sind:

1. Die Matrix σ

Man bringt die Gerade erstmal in die Form $y = mx + q$:

$$\begin{aligned} 2x - 3y + 2 &= 0 \\ \Rightarrow y &= \frac{2x + 2}{3} \end{aligned} \tag{2.4}$$

Um nun einen Punkt der Geraden zu finden, muss man einfach für x einsetzen (ich wähle 2, da es so eine schöne Lösung gibt):

$$\begin{aligned} y &= \frac{2 * 2 + 2}{3} \\ &= \frac{6}{3} = 2 \end{aligned} \tag{2.5}$$

Ein Punkt, der auf der Geraden liegt, ist also $p_g = (2, 2)$.

Das heisst, dass die Translationsmatrix folgendermassen aussieht (so wird der Punkt in den Ursprung verschoben):

$$T_{p_g} = \begin{pmatrix} 1 & 0 & -2 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{pmatrix}$$

Wir kennen das Verhältnis von der Gegenkatete zur Ankatete, nämlich:

$$\tan(\delta) = \frac{2}{3}$$

Eingesetzt können wir nun berechnen:

$$\begin{aligned} \cos(2\delta) &= \frac{1 - \frac{2^2}{3}}{1 + \frac{2^2}{3}} = \frac{1 - \frac{4}{9}}{1 + \frac{4}{9}} = \frac{\frac{5}{9}}{\frac{13}{9}} = \frac{5}{13} \\ \sin(2\delta) &= \frac{2 \cdot \frac{2}{3}}{1 + \frac{2^2}{3}} = \frac{\frac{4}{3}}{1 + \frac{4}{9}} = \frac{\frac{12}{9}}{\frac{13}{9}} = \frac{12}{13} = \frac{12}{13} \end{aligned} \tag{2.6}$$

Nun muss man dies nur noch in die Spiegelungsmatrix einsetzen:

$$\sigma' = \begin{pmatrix} \frac{5}{13} & \frac{12}{13} & 0 \\ \frac{12}{13} & -\frac{5}{13} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Schliesslich kann man "da realSpiegelungsmatrix σ folgendermassen berechnen:

$$\sigma = T^{-1}\sigma'T = \begin{pmatrix} \frac{5}{13} & \frac{12}{13} & -\frac{8}{13} \\ \frac{12}{13} & -\frac{5}{13} & \frac{12}{13} \\ 0 & 0 & 1 \end{pmatrix}$$

2. Die Bildpunkte von $A = (8, 1)$ und $B = (-65.4, 0.2)$.

Man setzt die Punkte in die Matrix:

$$Points = \begin{pmatrix} 8 & -65.4 \\ 1 & 0.2 \\ 1 & 1 \end{pmatrix}$$

Danach berechnet man $\sigma * Points$ und erhält das Resultat:

$$ResultMatrix = \begin{pmatrix} 3.38... & -25.58... \\ 7.92... & -59.52... \\ 1 & 1 \end{pmatrix}, A' = \begin{pmatrix} 3.38... \\ 7.92... \end{pmatrix}, B' = \begin{pmatrix} -25.58... \\ -59.52... \end{pmatrix}$$

2.9 Vektorprodukt

Definition

Das **Vektorprodukt** $\vec{a} \times \vec{b}$ zweier Vektoren \vec{a} und \vec{b} ist ein Vektor (wie der Name sagt) mit den Eigenschaften:

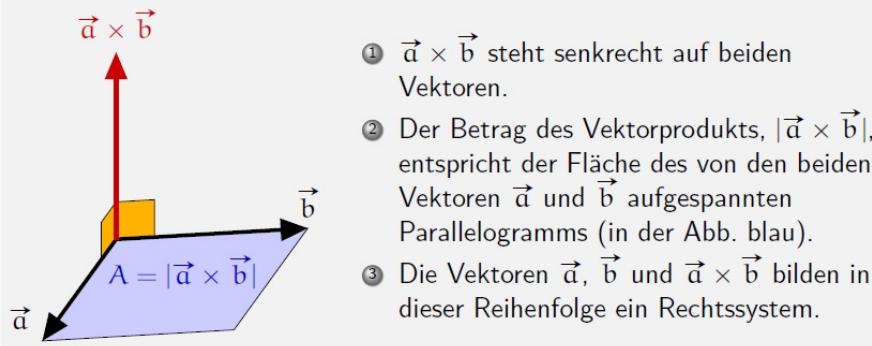


Abbildung 2.5: Definition vom Vektorprodukt

Berechnen Sie das Vektorprodukt der Vektoren $\vec{a} = \begin{pmatrix} 1 \\ 2 \\ 8 \end{pmatrix}$ und $\vec{b} = \begin{pmatrix} 4 \\ 3 \\ 5 \end{pmatrix}$ mit Hilfe der Regel von Sarrus.

$$\vec{a} \times \vec{b} = \begin{array}{r} 1 \quad 2 \quad 8 \\ 4 \quad 3 \quad 5 \end{array} \xrightarrow{\text{Rot ist Subtraktion}} \begin{array}{r} 2 \cdot 5 - 3 \cdot 8 \\ 8 \cdot 4 - 1 \cdot 5 \\ 1 \cdot 3 - 4 \cdot 2 \end{array} = \begin{pmatrix} -14 \\ 27 \\ -5 \end{pmatrix}$$

Blau ist Multiplikation

Abbildung 2.6: Vektorprodukt Example 1

Wie gross ist die Fläche des Dreiecks, welches von den beiden Vektoren $\vec{a} = \begin{pmatrix} 1 \\ 2 \\ 8 \end{pmatrix}$ und $\vec{b} = \begin{pmatrix} 4 \\ 3 \\ 5 \end{pmatrix}$ aufgespannt wird?

Siehe Vektorprodukt Example 1

$$A = \frac{|\vec{a} \times \vec{b}|}{2} = \frac{\sqrt{(-14)^2 + 27^2 + (-5)^2}}{2}$$

Abbildung 2.7: Vektorprodukt Example 2

2.9.1 Rechenregeln für das Vektorprodukt

$$\vec{a} \times \vec{b} = -\vec{b} \times \vec{a} \text{ Anti-Kommutativgesetz}$$

$$\vec{a} \times (\vec{b} + \vec{c}) = \vec{a} \times \vec{b} + \vec{a} \times \vec{c} \text{ Distributivgesetz}$$

$$\lambda(\vec{a} \times \vec{b}) = (\lambda \vec{a}) \times \vec{b} = \vec{a} \times (\lambda \vec{b})$$

3 Scan Konvertierung

Scan Konvertierung = Rasterung. Bezeichnet den Vorgang vom Übersetzen von Linien & Polygone in Pixel.

3.1 Rasterung Linie

Man nehme als erstes Beispiel eine einfache Linie. Mathematisch könnte man diese ja so beschreiben:

$$y(x) = mx + b$$

Gerastert müsste diese dann so aussehen wie in Abbildung 3.1. Pro Spalte haben wir ja immer nur 1

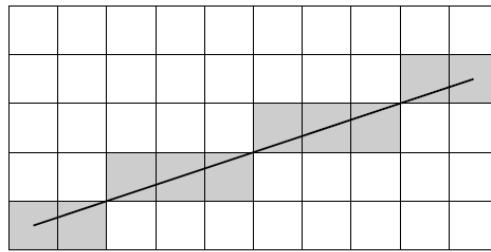


Abbildung 3.1: Gerasterte Linie

Pixel gesetzt hier, d.h wir müssen jeweils nur herausfinden, welche Zeile den geringsten Abstand zur korrekten Linie hat.

3.1.1 Digital Differential Analyzer

Beim Digital Differential Analyzer (DDA) berechnen wir zuerst den genauen Y-Wert, ganz nach der Formel $y(x) = mx + b$. Dann runden wir dies auf die nächste ganze Zahl und haben den Wert, welcher Pixel gezeichnet werden soll. Das machen wir für jede Spalte - ist also entsprechend aufwändig zu berechnen, da für jeden Pixel eine Multiplikation gemacht werden muss. Besser wäre es, ausgehend vom Startpixel zu rechnen – denn in der Formel erhöht sich x ja immer um 1, das heisst es wird einfach immer $+m$ ausgegeben. Wir haben dann also:

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

Das Verfahren ist aber immer noch mühsam, weil hier mit Gleitkommazahlen gerechnet wird und diese dann erst noch gerundet werden! Für Performancefreaks in der Computergrafik ein Graus.

3.1.2 Mittelpunktschema - Linie

Eine Linie kann ja eigentlich auch als folgende Gleichung dargestellt werden:

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

Die Distanz von einem Punkt (x, y) zu dieser Linie wäre dann: (MEP?)

$$d = \frac{|ax + by + c|}{\sqrt{a^2 + b^2}}$$

Das heisst für alle x,y Werte ergibt die Gleichung 0 wenn sie auf der Linie sind - klar.

Nehmen wir jetzt zusätzlich noch an, dass die Linie steigt, und zwar nicht stärker als 1. Wenn wir wieder Spalte für Spalte anschauen, kann es also nur sein, dass der nächste gezeichnete Pixel der direkt rechts vom jetzigen ist, oder dann der rechts schräg oben. Um dann zu entscheiden, welcher Punkt jetzt gezeichnet wird, setzen wir einen Punkt zwischen die beiden möglichen Pixel und setzen den in die Gleichung ein. Wenn diese etwas positives zurückliefert, dann ist die Linie oberhalb des Mittelpunktes und wir setzen den Pixel oben rechts. Ansonsten setzen wir ihn einfach rechts. In der Beispielgrafik 3.2 bedeutet *E* east, also rechts und *NE* north east - oben rechts.

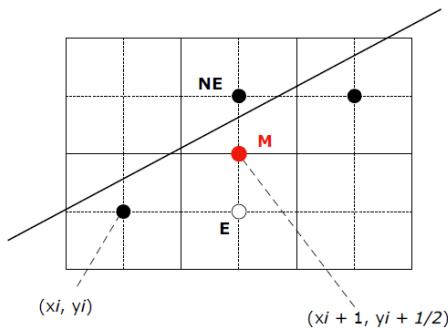


Abbildung 3.2: Mittelpunktschema Grafik

Wenn man es implementiert, hat man üblicherweise 2 Punkte - also (x_0, y_0) und (x_1, y_1) . Dann gilt es nur noch, diese Werte in den untenstehenden Algorithmus einzufügen und los gehts.

<pre>// Initialisierung Dx = x1 - x0; Dy = y1 - y0; DE = 2 * Dy; DNE = 2*(Dy - Dx); d = 2*Dy - Dx; y = y0;</pre>	<pre>// Berechnung WritePixel(x0, y0) for x = x0+1 to x1 do if d <= 0 then d = d + DE; else d = d + DNE; y = y + 1; end WritePixel(x, y); end</pre>
---	--

Abbildung 3.3: Mittelpunktschema Algorithmus

3.2 Mittelpunktschema - Kreise

Dasselbe wie für Linien. Zudem können wir sagen, wenn wir ja (x, y) berechnet haben, haben wir auch $(x, -y), (-x, y), (-x, -y), (y, x), (-y, x), (y, -x)$ und $(-y, -x)$ – also alle möglichen Kombinationen von +, -, x und y. Die mathematisch korrekte Formel für die Kreise wäre (r = Radius):

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

Der Punkt (x_i, y_i) liegt gerade im zweiten Quadranten. Das heisst der Kreis sieht zur Zeit so aus, wie in Abbildung 3.4. Das heisst wir sind wie bei der Linie beim Punkt (x_i, y_i) und müssen jetzt entscheiden, zeichnen wir den nächsten Pixel rechts von uns (east - *E*), oder halt rechts unten (south

east - SE). Die Distanz d berechnet sich hier so, also eigentlich einfach die normale Kreisformel, wo einfach noch ein $+1$ eingesetzt wurde beim x - Wert und beim y - Wert ein $-\frac{1}{2}$, da der Kreis ja etwas fällt.

$$d = (x_i + 1)^2 + (y_i - \frac{1}{2})^2 - r^2$$

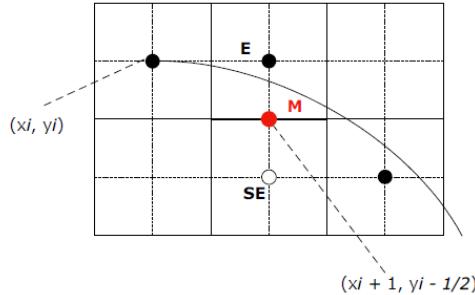


Abbildung 3.4: Mittelpunktschema Grafik für den Kreis

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

Abbildung 3.5: Mittelpunktschema Algorithmus für den Kreis

3.3 Füllen von Flächen mit Farbe

Sehr easy. Einfach dieser rekursive Algorithmus. Wenn man innerhalb einer Bedingung das macht,

<pre>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, y+1, oldColor, newColor); FloodFill(x-1, y, oldColor, newColor); FloodFill(x+1, y, oldColor, newColor); end end</pre>
--

Abbildung 3.6: Füll Algorithmus

dann einfach `ReadPixel(x,y) != boundaryColor`.

3.4 Füllen von Polygonen

Hier verwendet man den Scanlinien Algorithmus. Das heisst, wir gehen wie bei einem alten Röhrenfernseher von oben nach unten und zeichnen Zeilenweise die Pixel ins Polygon, siehe Abbildung 3.7. Damit wir wissen, von wo nach wo die Pixel gezeichnet werden sollen, müssen wir ja die Schnitt-

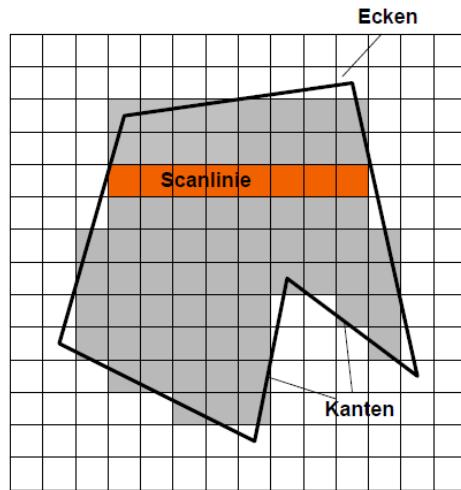


Abbildung 3.7: Scanlinien Algorithmus Grafik

punkte der Kanten berechnen, welche die aktuelle Scanlinie schneiden. Damit das effizient geschieht, sind sie bereits vorsortiert in Kantentabellen:

1. Kantentabelle *edgetable*

Alle Kanten des Polygons sind nach minimaler y Koordinate y_{min} sortiert. Wenn zwei Kanten denselben y_{min} haben, werden sie nach x sortiert.

2. Aktive Kanten (AET)

Hier werden alle Kanten gespeichert, die von der aktuellen Scanlinie geschnitten werden, dies sortiert nach x.

```
Erzeuge ET
Initialisiere AET = empty
y = 0
repeat
    Addiere alle Kanten ET(y) zu AET
    Sortiere AET nach x
    Zeichne Spans
    y = y + 1
    Entferne Kanten mit ymax = y aus AET
    Aktualisiere den x Wert aller Kanten in AET
until AET == empty and ET == empty
```

Abbildung 3.8: Scanlinien Algorithmus

3.5 Dreiecke zeichnen - à la Brute Force

Das ist furzeinfach. Wir haben oben ja gesehen, dass wir mit der Liniengleichung bestimmen können, ob ein Punkt oberhalb oder unterhalb einer Linie ist. Ein Dreieck besteht ja eigentlich nur aus 3

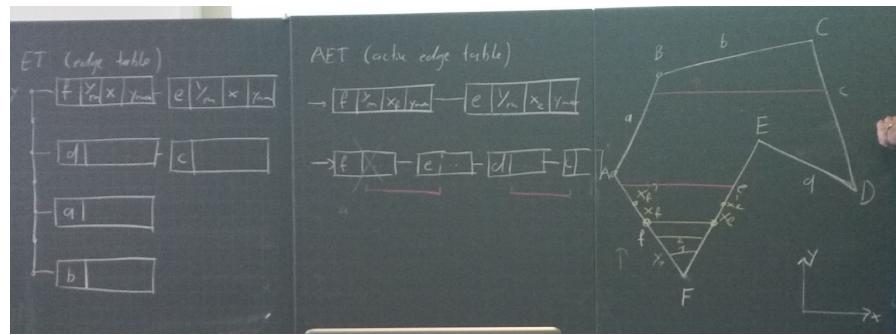


Abbildung 3.9: Scanlinien Algorithmus

Linien, das heisst wir prüfen für jeden Pixel, ob die Gleichung für alle Dreieckslien aufgeht - wenn ja zeichnen wir ihn, sonst lassen wirs sein. Die in Abbildung 3.10 eingefärbten Regionen bezeichnen die Regionen, welche *innerhalb* der Kante des Dreiecks liegen. Dort, wo sich alle Farben überlappen, dort werden die Pixel entsprechend gezeichnet.

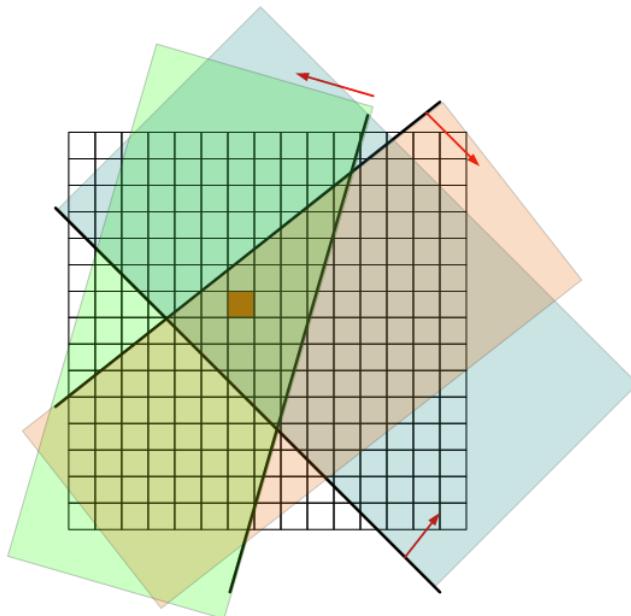


Abbildung 3.10: Dreiecke Zeichnen Brute Force Algorithmus

3.6 Anti Aliasing

Wir haben hier Linien gezeichnet, die allerdings einen extremen Treppeneffekt hätten, wenn wir sie 1:1 auf dem Monitor darstellen würden. Wir schauen kurz 2 Techniken an, die diesen Treppeneffekt minimieren.

3.6.1 Prefiltering

Hier berechnet man die Fläche der Linie auf dem Pixel und wählt proportional dazu die Intensität der Farbe.

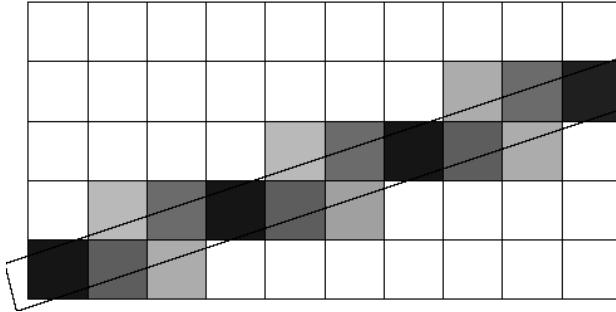


Abbildung 3.11: Prefiltering Algorithmus

3.6.2 Supersampling

Das Prinzip von Supersampling ist, dass man viel mehr Pixel berechnet, als eigentlich nötig wären. Nehmen wir das 2x2 Super Sampling, wo wir 4x mehr Pixel generieren und diese dann zusammenrechnen, wie in Abbildungen 3.12, 3.13 gezeigt. Im Detail berechnen wir die Geometrie unserer

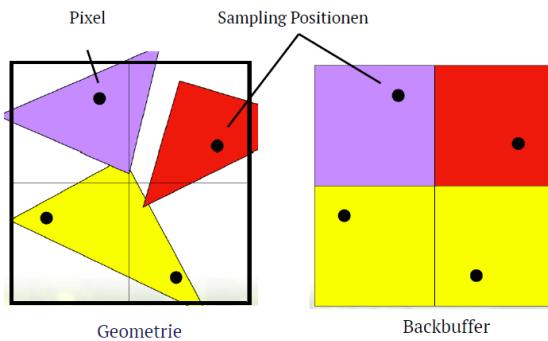


Abbildung 3.12: Supersampling 1

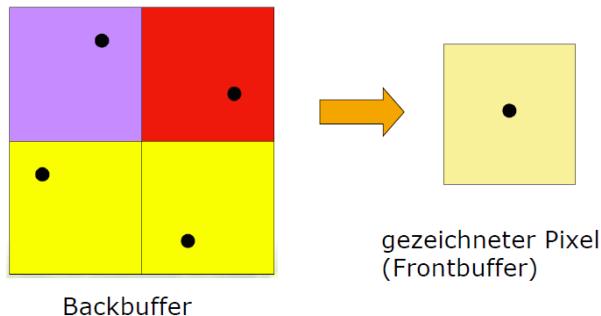


Abbildung 3.13: Supersampling 2

Objekte definieren sog. Sampling Positionen. Die Farbe an diesen sampling Positionen bestimmt dann die Farbe des ganzen Pixels - welcher im Backbuffer abgelegt wird und eben eine 4x höhere Auflösung hat als gezeigt. Die Farben des Backbuffers werden zum Schluss zusammengerechnet und es wird genau 1 Pixel gezeichnet, der Frontbuffer.

4 Clipping

Wenn man in einem Raum auf ein Objekt schaut, so sieht man nur z.B. nur einen Teil davon. Der andere Teil wäre schon da, ist aber nicht sichtbar. In einer 3D Rendering Engine wäre es jetzt unnötig, dazu alle Lichteffekt usw. schon zu berechnen, deswegen schneidet man es weg. Das nennt man Clipping.

4.1 3D vs 2D Clipping

Clipping geschieht in mehreren Stufen. Nachdem die 3D Szene aufgebaut ist, werden 3D Objekte entfernt, welche z.B. nicht im View Bereich der Kamera sind. Das wäre 3D Clipping. 2D Clipping bedeutet, dass nachdem alle Beleuchtung usw. berechnet wurde, noch der Teil rausgeschnitten wird, der nicht auf den Bildschirm passt - bevor das Bild dann schlussendlich gerastert und dargestellt wird.

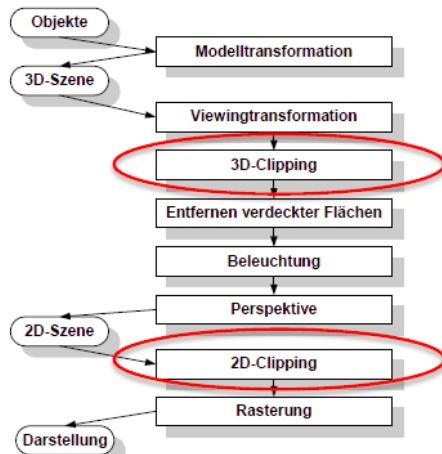


Abbildung 4.1: Grafikpipeline

4.2 Clipping Variationen

Man kann auf verschiedene Arten allgemein nun Clippen (3D).

1. Scissoring

Die 'dümmste' Variante des Clippings. Hier wird das 3D Clipping einfach übersprungen und nur am Schluss einfach die gezeichneten, welche gerade so im Fenster liegen.

2. Temporärer Buffer

Das gesamte Objekt wird auf Vorrat in einem temporären Buffer gezeichnet, welcher bei der Verwendung des Zeichenobjekts ins Zielbild kopiert wird. Dieses Verfahren wird beispielsweise für Buchstaben einer Schrift angewandt. Da die Objekte nicht mehr Polygone sondern fertig gezeichnetes Bilder sind, lassen sich diese sehr einfach schneiden.

3. Analytische Berechnung

Man berechnet irgendwie, was darin liegt. Man siehe unten.

4.3 Linien Clipping

Gehen wir von einem ganz einfachen Fall aus - dem Clipping einer Linie. Wir haben also ein Rechteck, welches z.B. den Canvas darstellt. Hindurch geht eine Linie. Wir möchten nun berechnen, welche Punkte gezeichnet werden müssen, resp. eine Linie haben, welche genau innerhalb des gezeichneten Rechtecks ist. Dazu gibt es die nachfolgenden Varianten.

4.3.1 Brute Force

Wenn mindestens ein Endpunkt der Linie ausserhalb des Rechtecks liegt (also wenn die Linie einfach etwas herausgeht aus dem Rechteck), dann müsste man alle Schnittpunkte berechnen, die die Linie mit dem Kappungs-Rechteck hat. Das scheint aber für CG Anwendungen zu unperformant zu sein.

4.3.2 Cohen-Sutherland

Nette Herren, die sich folgende Methode ausgedacht haben:

1. Nimm den Anfangs- & Endpunkt der Linie. Schau, in welchem Bereich der Punkt liegt:

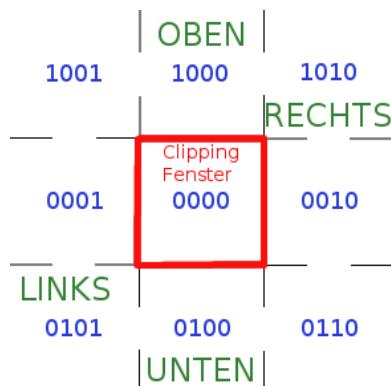


Abbildung 4.2: Bereiche der Grafik für den Algorithmus von Cohen Sutherland

Liegt der Punkt oben rechts vom Clipping Fenster, so hat der Punkt den Wert *1010*. Ist er rechts unten, so ist der Wert *0110*. Der Wert wird also bitweise bestimmt, in der Reihenfolge:

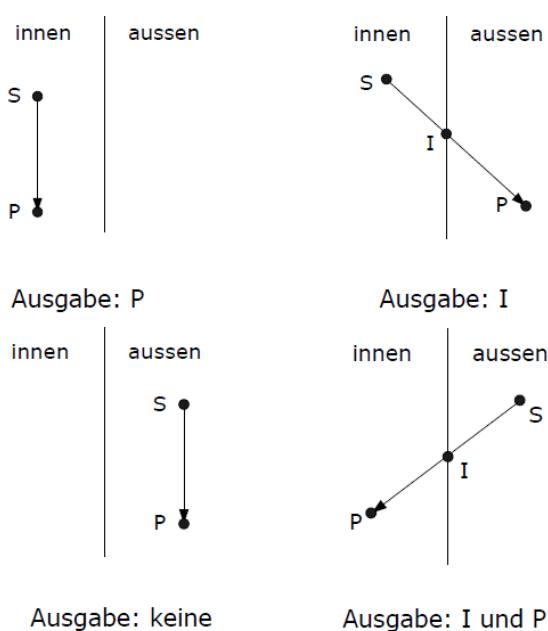
- a) Oben (1000)
 - b) Unten (0100)
 - c) Rechts (0010)
 - d) Links (0001)
2. Ist der Code bei beiden Punkten *0000*, so liegen ja beide Punkte im Clipping Fenster und die Linie wird akzeptiert und so gezeichnet.
 3. Ansonsten wird Bit für Bit der Wert verglichen und geschaut, ob bei beiden der Wert 1 ist. Als Beispiel hat der erste Endpunkt den Code 1001 und der zweite Endpunkt den Code 0101. Wenn man die beiden Codes verbindet kriegt man 0001. Es ist also ein Wert 1 und somit geht die Linie nicht durch das Clipping-Fenster. Eigentlich wird geschaut, ob beide Punkte links, rechts, oben oder unten des Rechtecks sind. Denn wenn das der Fall ist, so geht die Linie nämlich sicher nicht durch das Clipping Fenster und die Linie kann abgelehnt werden.

4. Wenn die Linie aber durch das Vergleichen nicht abgelehnt werden konnte, so wird der Schnittpunkt mit einer beliebigen Seite des Clipping Fensters berechnet und die Linie so *entzweigeschnitten*. Einen Teil der Linie kann dann verworfen werden, der andere Teil der Linie wird nochmals durch den Algorithmus gejagt, bis dann alles im Rechteck vorhanden ist.

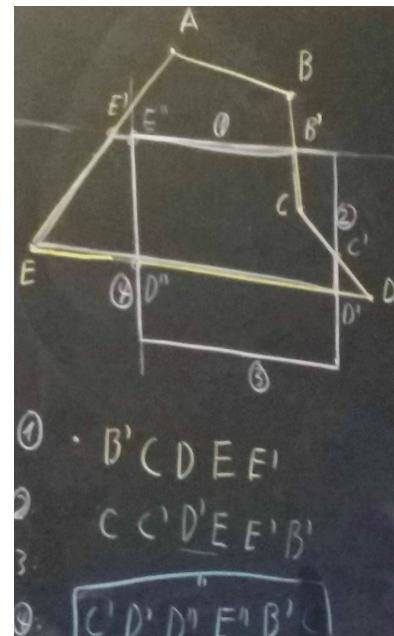
4.4 Polygone Clippen

Das wird schon etwas aufwändiger. Wir haben hier einen Algorithmus von den Herren **Sutherland-Hodgeman**, der eigentlich nur für Geraden geht - aber da ein Rechteck ja eigentlich einfach aus 4 Geraden besteht, machen wir das Ganze einfach 4x - dann passt auch.

Ein Polygon besteht ja aus n-Punkten. Wir gehen jetzt von einem Startpunkt aus und gehen dann von Punkt zu Punkt. Wir schauen dann jeweils einfach für den Startpunkt S und den nächsten Punkt des Polygons P , ob diese Punkte innerhalb oder ausserhalb des Polygons liegen. Dies wird für alle Punkte gemacht, das heisst jeder Punkt ist mal Startpunkt und jeder Punkt ist mal Endpunkt. Die Ausgabe des Algorithmus bestimmt dann, welcher Punkt behalten wird. Wenn die Punkte die Linien schneiden, wird einfach der Schnittpunkt dazu berechnet und entsprechend auch gespeichert. Es wird immer nur eine Gerade angeschaut d.h. am Anfang sind auch Punkt drin die gar nicht im Polygon liegen.



(a) Verschiedene Fälle (S = Startpunkt)



(b) Beispiel

Abbildung 4.3: Sutherland Hodgman Algorithmus

5 Visible Surface

Es geht darum, unsichtbare Flächen nicht darstellen zu lassen - sonst sieht es ja komisch aus.

5.1 Backface Culling

Wenn die Objekte geschlossen sind, dann kann einfach der Normalenvektor der Fläche betrachtet werden - denn wenn ein Vektor wegzeigt muss diese Fläche liegen dann ja auf der Rückseite und muss logischerweise nicht gezeichnet werden.

5.2 Tiefensortierung

Sortiere die Polygone und zeichne sie dann von hinten nach vorne, sodass die zuletzt gezeichneten Polygone die anderen überdecken. Wird auch als *Maleralgorithmus* bezeichnet.

5.2.1 Vorteile

1. Auch für transparente Objekte möglich
2. Einfach für Spezialfälle (2.5D -> Photoshop-Bildebene)

5.2.2 Nachteile

1. Ineffizient für viele Objekte $O(n^2)$
2. Wird nicht direkt Hardware unterstützt
3. Nicht für jedes Objekt direkt möglich - unpassende Objekte müssen z.B. noch weiter zerschnitten werden

5.2.3 Details

Im Detail sieht das dann so aus: Von jedem Polygon wird der Punkt genommen, der am nächsten zur Kamera ist. Anhand dieser Entfernen werden die Polygone zuerst einmal grob sortiert. Wenn jetzt sich irgendwo Polygone überschneiden, also z.B. Polygon A ist zwischen 2 und 10 von der Kamera entfernt und Polygon B zwischen 8 und 11, dann wird noch weiteres gemacht.

Polygon A ist zur Zeit jetzt ja näher bei der Kamera. Falls nun irgend eine folgende Bedingung zutrifft, werden sie nicht vertauscht, ansonsten schon.

1. Überlagern sich die x-Ausdehnungen nicht?

Auf gut Deutsch: Sind die Objekte vollständig nebeneinander?

2. Überlagern sich die y-Ausdehnungen nicht?

Sind die Objekte vollständig hintereinander? Abbildung 5.1 zeigt ein Beispiel, wo die Objekte vollständig nebeneinander sind, aber nicht vollständig hintereinander.

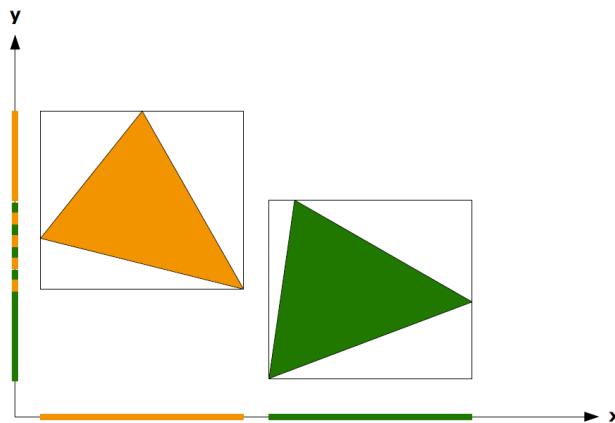


Abbildung 5.1: Beispiel Ausdehnung von 2 Objekten

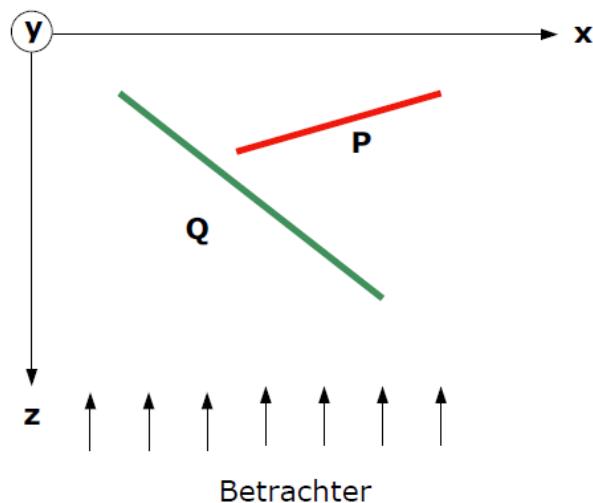


Abbildung 5.2: Q überdeckt P komplett - Q zeichnen

3. Wird das hintere Objekt vom Vorderen komplett überdeckt?
4. Liegt das vordere Objekt komplett auf der Betrachterseite vom vorderen Objekt?
5. Überlappen sich die Polygone nicht auf der Projektion in die xy Ebene?
Nehmen wir an, die beiden Objekte sind die zwei Dreiecke - wenn wir jetzt nur von vorne auf diese Objekte schauen, sind diese ja 2-Dimensional. Wenn sich diese nicht schneiden, wie im Beispiel in Abbildung 5.4, dann ist diese Bedingung erfüllt.

5.3 Z Buffer

Auf Grund der Nachteile vom Tiefensortieren hat man einen ganz einfachen Algorithmus entwickelt. Dieser zeichnet alle Polygone, berechnet also die R, G und B Werte und zusätzlich noch die Entfernung zur Kamera - der Z Wert. Beim Zeichnen der Pixel wird dann einfach geprüft, ob der aktuelle Pixel näher ist.

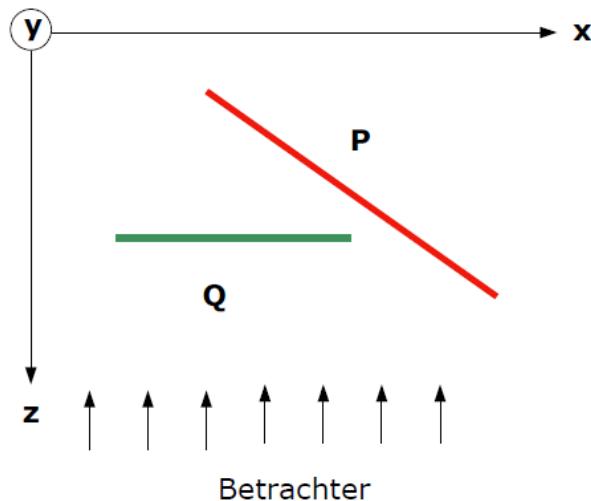
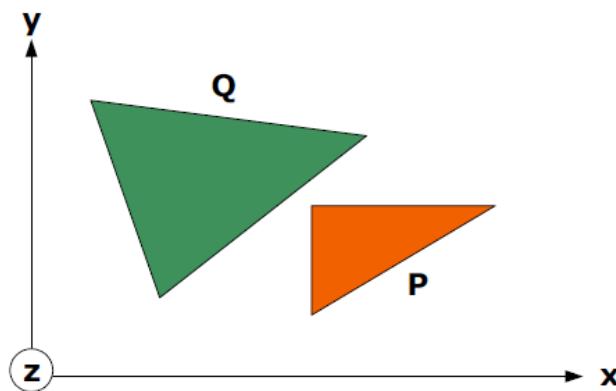


Abbildung 5.3: Q liegt komplett vor P - zuerst P und dann Q zeichnen



Ansicht vom Betrachter aus

Abbildung 5.4: Betrachter schaut von vorne - wenn sich P und Q nicht überlappt ist egal wie man sie zeichnet

5.3.1 Vorteile

1. Hardwareunterstützt
2. Polygone können in beliebiger Reihenfolge gezeichnet werden
3. Zeitkomplexität ist $O(n)$, aber häufig sogar konstant ab einer gewissen Anzahl Polygone.

5.3.2 Nachteile

1. Rundungsprobleme
2. Was ist, wenn zwei Pixel denselben z Wert haben? (Dann gibts Artefakte vom überlappen)
3. (grosser Speicherbedarf - heute nicht mehr sooo problematisch)

5.3.3 Berechnung von z

Z lässt sich aus der Ebenengleichung berechnen

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

oder inkrementell entlang einer Scanlinie

$$z_{neu} = \frac{-D - A(x_{alt} + 1) - By}{C} = z_{alt} - \frac{A}{C}$$

5.4 Warnock Algorithmus

Hier wird der Bildbereich angeschaut und entschieden, ob er *einfach* zu zeichnen ist. Ist er das nicht, wird er in 4 Unterbereiche unterteilt, für welche dann wieder einzeln entschieden wird, ob sie *einfach* sind. Das ist dementsprechend rekursiv.

5.4.1 Einfache Bereiche

Ein Bereich ist einfach, falls...

1. ... er kein oder nur 1 Polygon enthält.
2. ... nur ein Polygon beinhaltet, das am nächsten ist und den Bereich auch vollständig füllt.
3. ... er nur 1 Pixel gross ist.

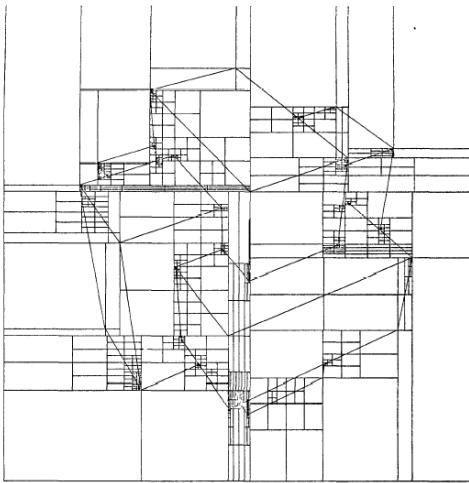


Figure 25 Subdivision by Warnock's algorithm. The object contains three intersecting bricks. In this example subdivision occurs at a vertex if possible.

Abbildung 5.5: Ein Beispiel zu Warnock

5.5 Various

Ein Ausdehnungsbereich ist ein möglichst kleines Rechteck (Bounding Box), welches das Objekt vollständig enthält. Wenn sich solche Ausdehnungsbereiche nicht schneiden, so schneiden sich logischerweise auch die Objekte nicht.

Objektraum = alle Objekte

Bildraum = alle Pixel

6 Beleuchtung

6.1 Einführung

Die Eigenschaften eines Objekts werden bestimmt durch seine Beleuchtungseigenschaften. Das heisst, ein glänzendes Objekt könnte z.B. aus Plastik bestehen, ein mattes Objekt aus Ton oder ähnlichen. Konkret sind dies folgende Eigenschaften:

1. Farbe
2. Reflexion
3. Transparenz
4. Struktur (matt / glänzend)
5. Spiegelung
6. Textur

Nachfolgend werden einige Beleuchtungsmodelle beschrieben. Es wäre zu aufwändig, die physikalisch korrekte Beleuchtung zu berechnen, deswegen nimmt man Vereinfachungen des Beleuchtungsmodells. Nachfolgend werden einige solcher Vereinfachungen dargestellt.

6.2 Lambert Beleuchtungsmodell - Diffuse Reflektion

Bei einem matten Objekt reflektiert die Oberfläche gleichmässig in alle Richtungen. Für die Kamera heisst das nun, dass egal aus welchem Winkel das Objekt betrachtet wird, die Farbe an einem Punkt immer dieselbe sein wird. Diese Farbe an einem Punkt wird also bestimmt durch:

1. Farbe des Lichts
2. Intensität des Lichts
3. Farbe des Materials

Die Intensität des Lichts ist abhängig vom Winkel, in dem es auftrifft - ist z.B. im Winter ja auch so, dann scheint die Sonne in einem steileren Winkel - und schon wird es kalt.

Formal ausgedrückt ist die Energie einer beleuchteten Fläche proportional zum Cosinus zwischen Lichtrichtung und Flächennormalen.

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

wobei

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

\vec{N} Normalvektor der Fläche

\vec{L} Richtung zur Lichtquelle

I_d Reflektierte Intensität

I_L Intensität der Lichtquelle

k_d Farbe der Fläche

6.3 Phong Modell

Phong wird für glänzende Reflektionen verwendet. Siehe Abbildung 6.1 - wir haben eine Fläche eines

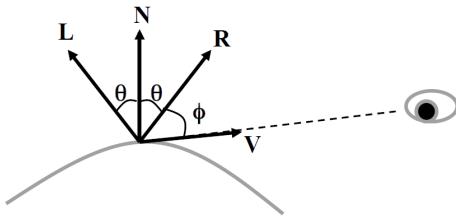


Abbildung 6.1: Phong Beleuchtungsmodell

Objekts. Diese hat einen Normalenvektor - hier N . Das Licht kommt aus der Richtung von L , hat also einen Winkel zum Normalenvektor - hier θ . Jetzt ist es wie beim Billard - Eintrittswinkel = Austrittswinkel. Wenn wir also nun genau im Reflektionswinkel auf das die Fläche schauen würden, dann hätten wir die volle Reflektion des Objekts. Je weiter wir aber weg sind von diesem *Austrittsvektor* - in der Grafik ist diese Distanz als Winkel ϕ angegeben, desto schwächer wird auch die Reflektion.

Deswegen können wir folgende Formel aufstellen - die jetzt hoffentlich etwas verständlich ist:

$$I_s = I_L \cdot k_s \cdot \cos^{n_s} \phi$$

I_s Intensität des reflektierten Strahls - Vektor V

I_L Intensität des einfallenden Lichts - Vektor L

k_s Beleuchtungskoeffizient (abhängig davon, wie stark das Objekt welche Farbe reflektiert)

n_s Wie nahe ist die Oberfläche an einer 'ideal reflektierenden' - üblicherweise zwischen 2 und 100

ϕ Winkel zwischen idealer Reflektionsrichtung und Betrachtungsrichtung

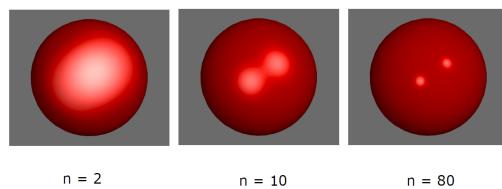


Abbildung 6.2: Beispiele für verschiedene Werte von n_s

6.3.1 Berechnung der Reflektionsrichtung

Wir haben 2 Vektoren, den Vektor vom einfallenden Licht L und der Normalenvektor der Fläche N . Ganz wichtig ist es hierbei, dass beide Vektoren *normalisiert* sind, d.h. dass deren Länge (oder Betrag) = 1 ist, sonst funktioniert die folgende Formel nicht und man rechnet mal eine halbe Stunde herum, jaja. Der Reflektionsvektor ist hier dann R .

$$R = 2 \cdot N(N \cdot L) - L$$

Die \cdot bedeuten einfach das Skalarprodukt. Jetzt muss man nur noch den Winkel zwischen dem Vektor R , und dem Vektor der Betrachtung V herausfinden. Auch hier: beide müssen die Länge 1 haben.

$$\cos^{-1}(V \cdot R)$$

Sehr simpel. Um Vektoren übrigens zu normalisieren macht man das:

$$V_{\text{normiert}} = \frac{V}{|V|}$$

$|V|$ ist die Länge (oder der Betrag) des Vektors. Die bestimmt man, wenn man z.B. einen Vektor (a, b, c) hat, so:

$$\sqrt{a^2 + b^2 + c^2}$$

6.4 Abschwächung des Lichts

In der Physik nimmt die Energie des Lichts mit zunehmender Entfernung quadratisch ab. f ist hier dann nur ein Faktor, der dann in die Beleuchtungsintensität hineinmultipliziert wird und d die Distanz

$$f = \frac{1}{d^2}$$

In der Computergrafik sieht das aber doof aus, weil die Lichtintensität zu schnell abnimmt. Besser man nimmt so was:

$$\frac{1}{c_1 + c_2 d + c_3 d^2}$$

Wobei die Faktoren $c_1 \dots c_n$ frei wählbar sind.

6.5 Schattierung

Wo wird die Beleuchtung überhaupt berechnet?

6.5.1 Konstante Schattierung

Pro Polygon wird nur eine Farbe berechnet - sieht aber dann nicht so schön aus wenn die Objekte gekrümmt sind.

6.5.2 Gouraud Schattierung

Die Farbe wird an den Eckpunkten der Polygone berechnet, die Fläche wird dann linear interpoliert gefüllt.

6.5.3 Phong Schattierung

Hier wird die Beleuchtung für jeden Pixel berechnet, denn hier wird nicht die Farbe sondern der Normalenvektor jeweils innerhalb des Polygons interpoliert. Sieht am schönsten aus.

7 Kurven und Flächen

7.1 Kurven in der Ebene

Gar nicht so schwierig! Wir haben irgendein Intervall, also eine Zahlenreihe von a nach b. Mathematisch ausgedrückt wäre das dann

$$[a, b] \rightarrow \mathbb{R}$$

All diese Zahlen in diesem Intervall setzen wir einfach in eine Funktion hinein und voila - eine Kurve. Jetzt gibt es eine implizite Art, eine Kurve darzustellen und eine explizite Art.

7.1.1 Implizite Darstellung

Am Beispiel eines Kreises wäre die implizite Darstellung:

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

Das heisst in einer impliziten Darstellung können wir z.B. nicht einfach die Zahlen von a nach b einsetzen, sondern die Gleichung ist einfach für alle zutreffenden Punkte = 0. Wenn wir Glück haben, ist die implizite Darstellung umwandelbar in eine explizite.

7.1.2 Explizite Darstellung

Eigentlich einfach eine Funktion. Für den oberen Halbkreis:

$$y = \sqrt{r^2 - x^2}$$

und den unteren Halbkreis:

$$y = -\sqrt{r^2 - x^2}$$

7.1.3 Parameter Darstellung

t bedeutet hier die *Zeit* und geht von a nach b. Beispiel mit dem Kreis, wo die *Zeit* von 0 bis 2π geht:

$$X(t) = \begin{pmatrix} x_1(t) \\ x_2(t) \end{pmatrix} = \begin{pmatrix} r \cdot \cos(t) \\ r \cdot \sin(t) \end{pmatrix}$$

Die Parameterdarstellung ist nicht unbedingt eindeutig, also auch z.B.

$$X(t) = \begin{pmatrix} r \cdot \cos(2t) \\ r \cdot \sin(2t) \end{pmatrix}$$

beschreibt einen Kreis, wobei hier die *Zeit* von 0 bis π geht - also er wird einfach 'schneller' durchlaufen.

7.2 Kurven im Raum

Dasselbe, einfach in grün. Eh 3D. Man hat einfach anstatt einen 2D Vektor in der Parameterdarstellung dann einen 3D Vektor.

7.2.1 Länge der Kurve im Raum

Das lässt sich ganz einfach mit einem Integral der ersten Ableitung der Kurve berechnen.

$$L = \int_a^b |f'(t)| dt$$

7.3 Kurvendarstellung

7.3.1 Polynomiale Darstellung

Sagen wir, wir haben eine Ausgangskurve, die wir irgendwie im Computer effizient darstellen möchten. In der Abbildung 7.1 sind diese durch die gestrichelten Linien dargestellt. Nehmen wir die naive

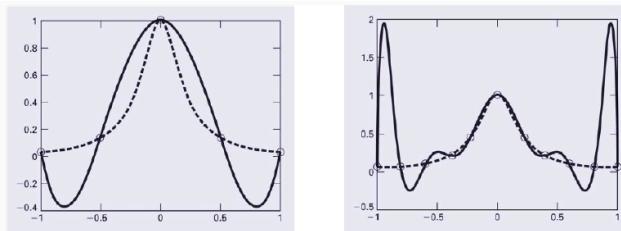


Abbildung 7.1: Möglichkeit der Darstellung von Kurven

Variante und versuchen, die Kurve durch ein einfaches Polynom darzustellen, also irgendwas in der Form von:

$$f(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n$$

Leider nützt uns diese Variante nicht wirklich. Denn die durchgezogenen Linien zeigen, dass diese nie wirklich an die echte Kurve herankommt - mit mehr definierten Punkten wird es sogar noch schlimmer - siehe der rechte Teil der Grafik. Wir müssen also etwas besseres haben.

Beispielrechnung - Methode der unbestimmten Koeffizienten

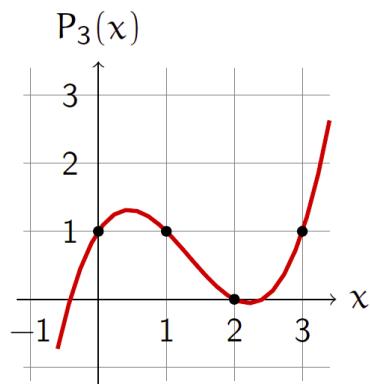


Abbildung 7.2: Beispiel Kurve

Wir können also folgende Punkte aus der Kurve lesen: $(0, 1)$, $(1, 1)$, $(2, 0)$, $(3, 1)$. Wir können also für die Gleichung

$$f(x) = a_0 + a_1x + a_2x^2 + a_3x^3$$

für den ersten Punkt $x = 0, y = 1$ wäre die Gleichung also:

$$1 = a_0 + a_1 \cdot 0 + a_2 \cdot 0 + a_3 \cdot 0 = a_0$$

Wir können das auch in einer Matrix darstellen;

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 \\ 1 & 3 & 9 & 27 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \end{pmatrix}$$

und lösen die dann auf mittels Maple oder so - weil von Hand ist ja uncool & lahm. Nachteil an dieser Methode ist, dass es eben lahm ist, man muss wenn sich ein Punkt ändert immer das komplette Gleichungssystem wieder lösen. Die Lösung wäre auf jeden Fall:

$$f(x) = 1 + \frac{3}{2}x - 2x^2 + \frac{1}{2}x^3$$

Beispielrechnung - Methode nach Lagrange

Wollen wir keine Gleichungssysteme lösen, so lässt uns Lagrange von der Qual erlösen. Schlechter Reim - ist aber trotzdem so.

Nehmen wir wieder wie Kurve aus Abbildung 7.2. Wir haben wieder die bekannten Punkte, wobei der erste Punkt dann $x_0 = 0$ und $f(x_0) = 1$ wäre usw.

Wir definieren dann Gleichungen wie in Abbildung 7.3. Die Funktion $L_0(x)$ hat dann die Eigen-

$$L_0(x) = \frac{(x - x_1)(x - x_2)(x - x_3)}{(x_0 - x_1)(x_0 - x_2)(x_0 - x_3)}, \quad L_1(x) = \frac{(x - x_0)(x - x_2)(x - x_3)}{(x_1 - x_0)(x_1 - x_2)(x_1 - x_3)},$$

$$L_2(x) = \frac{(x - x_0)(x - x_1)(x - x_3)}{(x_2 - x_0)(x_2 - x_1)(x_2 - x_3)}, \quad L_3(x) = \frac{(x - x_0)(x - x_1)(x - x_2)}{(x_3 - x_0)(x_3 - x_1)(x_3 - x_2)}.$$

Abbildung 7.3: Lagrange Gleichungen

schaft, dass sie an der Stelle $x_0 = 1$ ist und an allen anderen Stützstellen, also $L_0(x_1) = L_0(x_2) = \dots = 0$ ist. Die Funktion $L_1(x)$ hat dieselbe Eigenschaft, einfach ist sie an der Stelle $x_1 = 0$. Dasselbe für die weiteren Gleichungen. Jetzt können wir einfach unsere Gleichung aufstellen:

$$f(x) = L_0(x)f(x_0) + L_1(x)f(x_1) + L_2(x)f(x_2) + L_3(x)f(x_3)$$

Was dann auch schon die Lösung ist.

Erklärung dazu; Mit den aufgestellten L_n Gleichungen können wir ja einfach die Stützstellen beliebig 'verschieben' - was wir zum Schluss mit den Faktoren $f(x_0)$ auch machen. (Anmk. Autor - evt kann das jemand noch besser erklären, ich schreib jetzt einfach mal weiter im Text).

7.3.2 Methode nach Newton

7.4 Approximierende Splines

Wir definieren einfach irgendwelche Punkte im Raum und bestimmen jeweils die Funktion zwischen den Punkten - diese ist eine Polynomiale Funktion max. 3 Grades. Die Kurve muss dann nicht durch die Punkte gehen.

Gesucht ist ein Polynom 3. Grades durch die Punkte $(x_0, f(x_0)) = (0, 1)$, $(x_1, f(x_1)) = (1, 1)$, $(x_2, f(x_2)) = (2, 0)$ und $(x_3, f(x_3)) = (3, 1)$.

x_i	$f[x_i] = f(x_i)$	$f[x_{i+1}, x_i]$	$f[x_{i+2}, x_{i+1}, x_i]$	$f[x_{i+3}, x_{i+2}, x_{i+1}, x_i]$
0	1	$\frac{1-1}{1-0} = 0$	$\frac{-1-0}{2-0} = -\frac{1}{2}$	
1	1	$\frac{0-1}{2-1} = -1$	$\frac{1-(-1)}{3-1} = 1$	$\frac{1-\left(-\frac{1}{2}\right)}{3-0} = \frac{1}{2}$
2	0	$\frac{1-0}{3-2} = 1$		
3	1			

Die Koeffizienten stehen auf der ersten (schräg nach unten laufenden) Zeile:

$$f(x) = 1 + 0(x-0) - \frac{1}{2}(x-0)(x-1) + \frac{1}{2}(x-0)(x-1)(x-2).$$

Abbildung 7.4: Newton

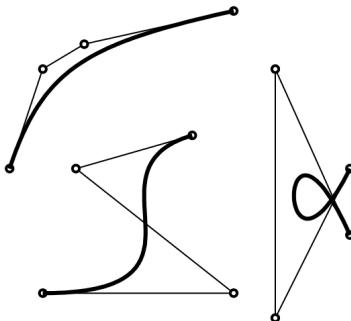


Abbildung 7.5: Splines

7.4.1 Lineare Interpolation - Lineare Bézier Splines

Wir haben einfach eine Linie mit den Anfangs P_0 und Endpunkten P_1 . Jetzt müssen wir diese wahnsinnig schwierig beschreiben. Hier ist immer $t \leq t \leq 1$.

$$1. P(t) = (1-t)P_0 + t \cdot P_1$$

$$2. P(t) = (P_1 - P_0) \cdot t + P_0$$

$$3. P(t) = (P_1, P_0) \begin{pmatrix} -1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} t \\ 1 \end{pmatrix}$$

7.4.2 Quadratische Bézier Splines

Quadratische Bézier Splines sind eigentlich dasselbe wie die linearen Bézier Splines - einfach wird zwischen den linearen Splines nochmals interpoliert - yo dawg I heard you like interpolieren. In der Abbildung 7.6 sieht man das schön. Grafisch kann man das dort auch super erklären - denn vom Punkt in der Mitte zwischen P_0 und P_1 wird einfach eine Linie gezogen zum Punkt in der Mitte

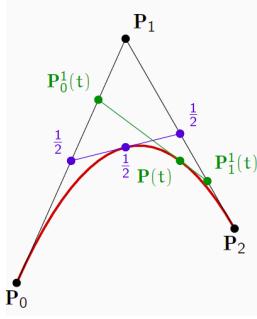


Abbildung 7.6: Quadratische Bézier Splines

zwischen P_1 und P_2 und dann in der Mitte dieser Linie ist dann der mittlere Punkt der fertigen Kurve. Wir rechnen also die 2 linearen Repräsentationen der Linien zusammen und das geht so:

Man hat zwei lineare Bézier Splines, die so definiert sind:

$$P_0^1(t) = (1 - t)P_0 + t \cdot P_1$$

$$P_1^1(t) = (1 - t)P_1 + t \cdot P_2$$

Jetzt setzt man diese einfach in die normale Bézier Repräsentation ein - also eine Funktion in die Funktion einfügen.

$$P(t) = (1 - t)P_0^1(t) = t \cdot P_1^1(t)$$

Dann muss man nur noch auflösen et voila:

$$P(t) = (1 - t)^2 P_0 + 2(1 - t)t \cdot P_1 + t^2 P_2$$

Das, meine Kinder, wäre der ganze Zauber. Tschüss & bis zum nächsten Mal!

7.4.3 Kubische Bézier Splines

Hehe, jetzt dachtest Du schon du bist fertig. Aber es wird nicht schwieriger. Wir haben jetzt einfach 4 Kontrollpunkte (P_0, \dots, P_3) statt 3, also haben wir alle Definitionen:

$$P_0^1(t) = (1 - t)P_0 + t \cdot P_1$$

$$P_1^1(t) = (1 - t)P_1 + t \cdot P_2$$

$$P_2^1(t) = (1 - t)P_2 + t \cdot P_3$$

Jetzt müssen wir alle 4 Gleichungen zusammenfügen:

$$P_0^2(t) = (1 - t)P_0^1(t) = t \cdot P_1^1(t)$$

$$P_1^2(t) = (1 - t)P_1^1(t) = t \cdot P_2^1(t)$$

Und diese dann nochmals in die mix schmeissen:

$$P(t) = (1 - t)P_1^2(t) = t \cdot P_2^1(t)$$

Was dann wieder aufgelöst:

$$P(t) = (1 - t)^3 P_0 + 3(1 - t)^2 \cdot t \cdot P_1 + 3(1 - t)t^2 P_2 + t^3 P_3$$

ergibt. Dem aufmerksamen Leser wird nicht entgangen sein, dass diese Bézier Splines Eigenschaften vom Pascal'schen Dreieck haben.

7.4.4 Bernsteinpolynome

Nicht aufgeben! Kurz ein Schluck Wasser und weiter gehts. Es ist jetzt einfach das Ganze Bézier Splines Zeugs - einfach nochmals verallgemeinert. Wir vorhin angetönt - dem aufmerksamen Leser wird ja nicht entgangen sein, dass die Formel der Bézier Kurve irgendwie so ein Muster ähnlich dem Pascal'schen Dreieck hat. Die Bausteine von Bézier Splines sind ja auch immer dieselben - also

$$\begin{array}{ccccccc} & & & & 1 & & \\ & & & & 1 & 1 & \\ & & & & 1 & 2 & 1 \\ & & & & 1 & 3 & 3 & 1 \\ & & & & 1 & 4 & 6 & 4 & 1 \\ & & & & 1 & 5 & 10 & 10 & 5 & 1 \\ & & & & 1 & 6 & 15 & 20 & 15 & 6 & 1 \\ & & & & 1 & 7 & 21 & 35 & 35 & 21 & 7 & 1 \\ & & & & \ddots \end{array}$$

Abbildung 7.7: Pascal'sches Dreieck

irgendwas mit $(1 - t)$ und t selbst kommt auch irgendwie immer vor. Nehmen wir nochmals das Beispiel vom Kubischen Bézier Spline und zerlegen die End-Formel in die Summanden und nehmen noch kurz die Punkte nochmals raus:

$$\begin{aligned} & 1 \cdot (1 - t)^3 \\ & 3 \cdot (1 - t)^2 \cdot t \\ & 3 \cdot (1 - t) \cdot t^2 \\ & 1 \cdot t^3 \end{aligned}$$

Die Faktoren zu Beginn des Terms stellen genau die Binomialkoeffizienten dar. Hier nochmals die Binomialkoeffizienten-Berechnung:

$$\binom{n}{i} = \frac{n!}{i!(n-i)!}$$

Ersetzen wir die Faktoren doch gleich einmal:

$$\begin{aligned} & \binom{3}{0} \cdot (1 - t)^3 \\ & \binom{3}{1} \cdot (1 - t)^2 \cdot t \\ & \binom{3}{2} \cdot (1 - t) \cdot t^2 \\ & \binom{3}{3} \cdot t^3 \end{aligned}$$

Das wärs auch schon gewesen. Denn jetzt können wir daraus die Formel für die einzelnen Teile sagen:

$$B_i^3(t) = \binom{3}{i} (1 - t)^{3-i} t^i$$

Und das sind dann schon die Bernsteinpolynome. Für i setzte man dann einfach eine Zahl zwischen 0 und 3 und man erhält dann den Term an der entsprechenden Position.

Jetzt haben wir aber doch am Anfang die Punkte herausgenommen - wir wollen diese jetzt natürlich wieder hineinnehmen. Dazu müssen wir sie nur eben mit dem entsprechenden Term wieder multiplizieren. Die ganze Kubische Bézier Kurve lässt sich dann so schreiben:

$$P(t) = \sum_{i=0}^3 B_i^3(t) \cdot P_i$$

Jetzt gehen wir ja nur bis 3 - können wir auch bis n gehen? Klar - das sind dann gerade die Bézier Kurven n. Ordnung.

$$P(t) = \sum_{i=0}^n B_i^n(t) \cdot P_i$$

Wobei dann die Bernsteinpolynome, also der Teil mit dem B , so definiert sind:

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

Eigenschaften von Bézier Kurven

1. Die Bézier Kurve liegt immer innerhalb der konvexen Hülle des Linienzuges
2. Die Anzahl der Kontrollpunkte P_0, \dots, P_n bestimmt den Grad n der Bézier Kurve.
3. Die Änderung eines Kontrollpunktes bewirkt eine Änderung der gesamten Kurve.
4. Indem man die stetigen Funktionswerte und Ableitungen voraussetzt, lassen sich Bézier Kurven höheren Grades durch Bézier Kurven niedrigeren Grades zusammensetzen.

Wahnsinnige Mathematische Wortspiele - man kanns auch einfach sagen. Zum ersten Punkt lässt sich sagen, dass in einem konvexen Gebiet jeder Punkt von jedem direkt erreichbar ist, ohne das Gebiet verlassen zu müssen. Also man stelle sich eine Insel im Meer vor. Diese ist konvex, wenn man von jedem x-beliebigen Punkt *trockenen Fusses* an jeden anderen x-beliebigen Punkt kommt. Und die konvexe Hülle beschreibt dann einfach das kleinste Gebiet, welches konvex ist und alle Kontrollpunkte enthält. Abbildung 7.8 zeigt in Grau diese konvexe Hülle.

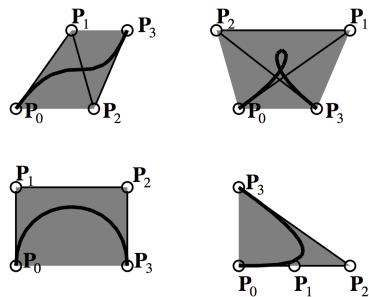


Abbildung 7.8: Konvexe Hülle

Die Zusammensetzung von Bézier Kurven setzt ja voraus, dass diese stetig sind. Wir haben folgende Eigenschaften, dass es stetig ist:

1. Die Segmente haben an den Endpunkten keine Sprünge / Lücken
2. Die Segmente haben an den Endpunkten dieselbe erste Ableitung - also nicht eine Kurve in Form von: \wedge .

3. Die Segmente haben an den Endpunkten dieselbe zweite Ableitung. Also nicht z.B. eine gerade Linie und gleich anschliessend ein Halbkreis. Das wäre, wie man geradeaus Autofährt und dann direkt eine 180 Grad Kurve kommt und man in unendlich kurzer Zeit das Steuerrad nach rechts drehen sollte, damit man die Kurve 'perfekt' fährt. Also nicht so eine Kurve wie in Abbildung 7.9.

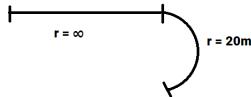


Abbildung 7.9: Zweite Ableitung ist hier nicht gleich

Zeichnen von Bézier Kurven

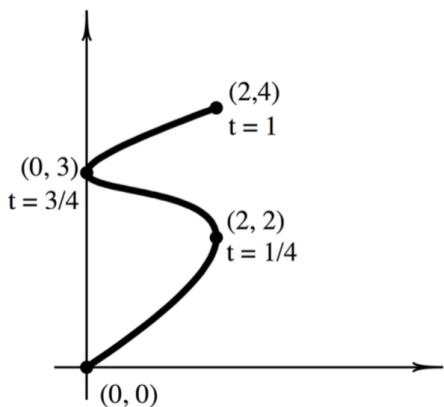
Am besten stellt man 2 Gleichungen auf, eine, die von der t -Variable abhängig ist und die X-Position ergibt, und die andere welche die Y-Position ergibt. Dann kann man in jedem Taschenrechner für Werte von t von 0 bis 1 diese Gleichungen einfach numerisch lösen und die entsprechenden Punkte einzeichnen und verbinden. Zur Kontrolle empfiehlt sich der Besuch auf folgender Website: <https://www.desmos.com/calculator/cahqdxeshd>

7.4.5 B Splines & NURBS

Laut meinen Notizen nicht Teil der MEP.

7.5 Lösungen Übungen

7.5.1 Aufgabe 1 - Interpolierende Kurve durch vier Punkte



Bestimmen Sie eine interpolierende Kurve 3. Grades der Form

$$\mathbf{P}(t) = \mathbf{c}_0 + \mathbf{c}_1 t + \mathbf{c}_2 t^2 + \mathbf{c}_3 t^3$$

durch die Punkte $\mathbf{P}_0 = (0,0)^T$, $\mathbf{P}_1 = (2,2)^T$, $\mathbf{P}_2 = (0,3)^T$ und $\mathbf{P}_3 = (2,4)^T$ mit Hilfe der Methode der unbestimmten Koeffizienten.

Abbildung 7.10: Übungsbeschreibung Aufgabe 1

Der Graph auf der linken Seite beschreibt eine unmöglich durch eine Funktion darstellbaren Graphe dar. Da aber Kurven ja keine Funktionen sind, ist uns das egal und wir beschreiben die gewünschten Faktoren durch Vektoren - analog einer parametrisierten Darstellung einer Linie durch Vektoren.

Wir haben also folgende Ausgangsgleichung:

$$P(t) = \begin{pmatrix} c_{0x} \\ c_{0y} \end{pmatrix} + \begin{pmatrix} c_{1x} \\ c_{1y} \end{pmatrix} t + \begin{pmatrix} c_{2x} \\ c_{2y} \end{pmatrix} t^2 + \begin{pmatrix} c_{3x} \\ c_{3y} \end{pmatrix} t^3$$

Wir können aus dieser Gleichung z.B. für $P(0) = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ zwei einzelne Gleichungen herleiten, eine für die x-Komponente, die andere für die y-Komponente. Wir haben aber ja 4 Punkte, d.h. wir können 8 Gleichungen aufstellen. In die Matrix übertragen sehen diese dann so aus:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & \frac{1}{4} & \frac{1}{4}^2 & \frac{1}{4}^3 \\ 1 & \frac{3}{4} & \frac{3}{4}^2 & \frac{3}{4}^3 \\ 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} c_{0x} \\ c_{1x} \\ c_{2x} \\ c_{3x} \end{pmatrix} = \begin{pmatrix} 0 \\ 2 \\ 0 \\ 2 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & \frac{1}{4} & \frac{1}{4}^2 & \frac{1}{4}^3 \\ 1 & \frac{3}{4} & \frac{3}{4}^2 & \frac{3}{4}^3 \\ 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} c_{0y} \\ c_{1y} \\ c_{2y} \\ c_{3y} \end{pmatrix} = \begin{pmatrix} 0 \\ 2 \\ 3 \\ 4 \end{pmatrix}$$

Mit einem CAS ergibt sich dann die Lösung von:

$$P(t) = \begin{pmatrix} 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 18 \\ 12 \end{pmatrix} t + \begin{pmatrix} -48 \\ -\frac{56}{3} \end{pmatrix} t^2 + \begin{pmatrix} 32 \\ \frac{32}{3} \end{pmatrix} t^3$$

7.5.2 Aufgabe 2 - Bézier-Kurve durch vier Punkte

Bestimmen Sie die (kubische) Bézier-Kurve durch die Punkte $\mathbf{P}_0(0,0)$, $\mathbf{P}_1(2,-4)$, $\mathbf{P}_2(5,6)$ und $\mathbf{P}_3(9,0)$ und skizzieren Sie diese.

Abbildung 7.11: Übungsbeschreibung Aufgabe 2

Man muss einfach folgende Gleichung aufstellen:

$$P(t) = (1-t)^3 \cdot \begin{pmatrix} 0 \\ 0 \end{pmatrix} + 3(1-t)^2 \cdot t \begin{pmatrix} 2 \\ -4 \end{pmatrix} + 3(1-t)t^2 \begin{pmatrix} 5 \\ 6 \end{pmatrix} + t^3 \begin{pmatrix} 9 \\ 0 \end{pmatrix}$$

Diese Gleichung kann man noch in der Matrixform darstellen. Wieso das nützlich ist sieht man gleich. Wichtig hier ist die Matrix auf der rechten Seite, diese ist durch die kubische Bézier-Funktion bereits vorgegeben. In die linke Matrix füllt man einfach die bekannten Punkte ein.

$$P(t) = \begin{pmatrix} 0 & 2 & 5 & 9 \\ 0 & -4 & 6 & 0 \end{pmatrix} \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} t^3 \\ t^2 \\ t \\ 1 \end{pmatrix}$$

Jetzt kann man die ersten zwei Matrizen miteinander multiplizieren (Taschenrechner hilft hier) und man erhält:

$$P(t) = \begin{pmatrix} 0 & 3 & 6 & 0 \\ -30 & 42 & -12 & 0 \end{pmatrix} \begin{pmatrix} t^3 \\ t^2 \\ t \\ 1 \end{pmatrix}$$

Dann muss man noch diese zusammenrechnen - et voila:

$$P(t) = \begin{pmatrix} 3t^2 + 6t \\ -30t^3 + 42t^2 - 12t \end{pmatrix}$$

Wenn man es noch grafisch darstellen möchte, einfach per Taschenrechner ein paar Werte für t , am besten zwischen 0 und 1, ausrechnen und darstellen. Dann sollte es etwa so aussehen:

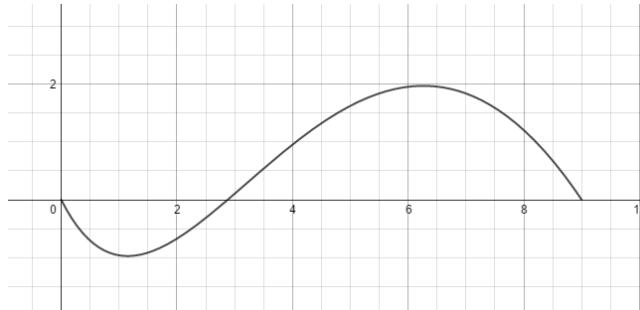


Abbildung 7.12: Grafische Lösung Bezier Aufgabe 2

7.5.3 Aufgabe 3 - Nochmals: Bézier-Kurve durch vier Punkte

Betrachten Sie die vier Punkte aus Aufgabe 1. Welche kubische Bézier-Kurve geht exakt durch diese vier Punkte? Hinweis: Einzige Variablen sind $\mathbf{P}_1 = (P_x, P_y)^T$ und $\mathbf{P}_2 = (Q_x, Q_y)^T$. Wenn Sie den Ansatz

$$\mathbf{P}(t) = \mathbf{P}_0(1-t)^3 + 3\mathbf{P}_1t(1-t)^2 + 3\mathbf{P}_2t^2(1-t) + \mathbf{P}_3t^3$$

für $t = 1/4$ und $t = 3/4$ auswerten und verlangen, dass

$$\mathbf{P}(1/4) = \mathbf{P}_1 \quad \text{und} \quad \mathbf{P}(3/4) = \mathbf{P}_2$$

gilt, erhalten Sie vier Gleichungen in den gesuchten Unbekannten P_x , P_y , Q_x und Q_y . Beachte: es gibt viele Bézier-Kurven durch \mathbf{P}_1 und \mathbf{P}_2 welche in \mathbf{P}_0 starten und in \mathbf{P}_3 enden.

Abbildung 7.13: Übungsbeschreibung Aufgabe 3

Hier hat man ja durch die normale Bézier Gleichung folgende Gleichung vorgegeben, wobei die mittleren Kontrollpunkte ja unbekannt sind.

$$P(t) = \begin{pmatrix} 0 \\ 0 \end{pmatrix} (1-t)^3 + 3 \begin{pmatrix} P_x \\ P_y \end{pmatrix} t(1-t)^2 + 3 \begin{pmatrix} Q_x \\ Q_y \end{pmatrix} t^2(1-t) + \begin{pmatrix} 2 \\ 4 \end{pmatrix} t^3$$

Zusätzlich kann man dann folgende Gleichungen aufstellen, indem man ja diese Constraints setzt:

$$P\left(\frac{1}{4}\right) = \begin{pmatrix} 2 \\ 2 \end{pmatrix}$$

$$P\left(\frac{3}{4}\right) = \begin{pmatrix} 0 \\ 3 \end{pmatrix}$$

Daraus ergeben sich folgende Gleichungen:

$$P_x\left(\frac{1}{4}\right) = 2 = 3P_x\frac{1}{4}\left(1 - \frac{1}{4}\right)^2 + 3Q_x\frac{1}{4}^2\left(1 - \frac{1}{4}\right) + 2\frac{1}{4}^3$$

$$\begin{aligned}
P_y \left(\frac{1}{4} \right) &= 2 = 3P_y \frac{1}{4} \left(1 - \frac{1}{4} \right)^2 + 3Q_y \frac{1}{4}^2 \left(1 - \frac{1}{4} \right) + 4 \frac{1}{4}^3 \\
P_x \left(\frac{3}{4} \right) &= 0 = 3P_x \frac{3}{4} \left(1 - \frac{3}{4} \right)^2 + 3Q_x \frac{3}{4}^2 \left(1 - \frac{3}{4} \right) + 2 \frac{3}{4}^3 \\
P_y \left(\frac{3}{4} \right) &= 3 = 3P_y \frac{3}{4} \left(1 - \frac{3}{4} \right)^2 + 3Q_y \frac{3}{4}^2 \left(1 - \frac{3}{4} \right) + 4 \frac{3}{4}^3
\end{aligned}$$

Das kann man noch vereinfachen, aber so viele Formeln in Latex ist mühsam. Woraus sich mit einem schlauen CAS folgende Lösung ergibt:

$$P = \begin{pmatrix} 6 \\ 4 \end{pmatrix}, Q = \begin{pmatrix} -4 \\ \frac{16}{9} \end{pmatrix}$$

Jetzt kann man wieder das Schema aus Aufgabe 2 verwenden, um noch die komplette Lösung zu erhalten. Wir haben ja jetzt 4 bekannte Kontrollpunkte:

$$P_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, P_2 = \begin{pmatrix} 6 \\ 4 \end{pmatrix}, P_3 = \begin{pmatrix} -4 \\ \frac{16}{9} \end{pmatrix}, P_4 = \begin{pmatrix} 2 \\ 4 \end{pmatrix}$$

Diese eingesetzt in die Matrizen:

$$P(t) = \begin{pmatrix} 0 & 6 & -4 & 2 \\ 0 & 4 & \frac{16}{9} & 4 \end{pmatrix} \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} t^3 \\ t^2 \\ t \\ 1 \end{pmatrix}$$

Erste beiden Matrizen zusammenrechnen:

$$P(t) = \begin{pmatrix} 32 & -48 & 18 & 0 \\ \frac{32}{3} & -\frac{56}{3} & 12 & 0 \end{pmatrix} \begin{pmatrix} t^3 \\ t^2 \\ t \\ 1 \end{pmatrix}$$

Und zum Schluss noch die beiden anderen auch:

$$P(t) = \begin{pmatrix} 32t^3 - 48t^2 + 18t \\ \frac{32}{3}t^3 - \frac{56}{3}t^2 + 12t \end{pmatrix}$$

8 Texture Mapping

Grundidee vom Texturemapping ist es, dass man Bilder etwas realistischer gestalten kann, ohne dass man mehr Polygone hinzufügen muss. Hat man z.B. in einer Szene ein Bild an der Wand, so kann z.B. einfach ein Rechteck als Polygon genommen werden und jetzt grob gesagt ein PNG darüber gespannt werden. Dann hat man ein detailliertes Bild, ohne von den darunterliegenden Polygonen abhängig zu sein. Oder siehe Minecraft - Blöcke mit Texturen darauf.

8.1 Prozedurale Texturen

Wenn man z.B. eine Holzkiste hat, so ist es nicht möglich oder zu mühsam, eine exakte PNG Datei zu erstellen. Bei Blöcken in Minecraft mag das ja gehen, aber sobalds dann komplexere Objekte werden, kann man auf prozedural generierte Texturen zurückgreifen. Dabei werden die Holzstrukturen, oder Marmor usw dynamisch generiert und es ist egal, wie gross das Objekt schlussendlich ist.

8.2 Textur Korrdinaten

Der Pixel x,y vom Bild ist dann schlussendlich wo auf dem Objekt? Das macht man mit Transformationen - ist ja klar. Wie aber genau?

8.2.1 Kugel

Wenn wir eine Kugel haben, z.B. wie die Erde ja eine ist, dann können wir mit Längen und Breiten Angaben arbeiten. Das heisst, dass wir die Winkel ϕ und θ frei wählen können, mit der folgenden Formel kriegen wir dann alle möglichen Koordinaten der Kugel:

$$\begin{aligned}x &= r \cdot \cos\theta \cdot \sin\phi \\y &= r \cdot \sin\theta \cdot \cos\phi \\z &= r \cdot \cos\phi\end{aligned}$$

Das Mapping auf die Texturkoordinaten (u,v) geschieht dann mit diesen Formeln:

$$\begin{aligned}u &= \frac{\theta}{2\pi} \\v &= \frac{\phi}{2\pi}\end{aligned}$$

8.2.2 Polygon

Wenn wir aber etwas anderes als eine Kugel haben, z.B. einen Dinosaurier den wir gern mit einer Textur überziehen möchten, so müssen wir für jedes Dreieck die entsprechenden Texturkoordinaten angeben. Die Fläche innerhalb des Dreiecks auf der Textur wird dann genommen und dann pro dargestelltem Pixel (also im Fragment Shader) interpoliert. Ein Ursprungspixel (auch *Texel* genannt), kann dann auf mehrere Pixel oder auf weniger Pixel reduziert werden. Wenn es reduziert wird, so

wird einfach der Farb-Durchschnitt genommen. Häufig werden diese Vergrösserungen usw. schon vorberechnet, dann muss man es nicht 10x machen.

Perspektive

Wenn einfach linear interpoliert wird, dann gibt es perspektivische Fehler, wie in Abbildung 8.1 dargestellt. Das hat den Ursprung darin, dass die Weltkoordinaten nicht die gleichen Abstände ha-

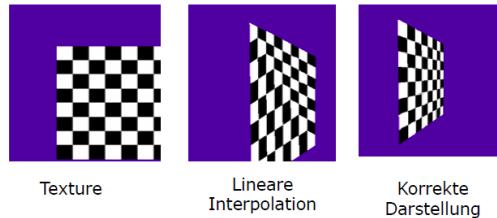


Abbildung 8.1: Fehler bei der linearen Interpolation von Texturen

ben wie die Kamerakoordinaten. Das gute ist, dass diese perspektivische Korrektur in den heutigen Grafikkarten bereits vorimplementiert ist.

8.3 Environment Mapping

Man möchte z.B. die Umgebung in einem anderen Objekt spiegeln lassen. Dazu könnte man die gesamte Umgebung in 3D ebenfalls aufbauen und das Raytracing verwenden - was sehr ressourcenintensiv wäre. Andererseits könnte ja man einfach sich vorstellen, man ist im innern einer Kugel und projiziert auf dieses Innere eine Textur, welche dann die Umgebung darstellt. Somit vereinfacht sich die Berechnung der Umgebungsreflexion auf ein Objekt erheblich.

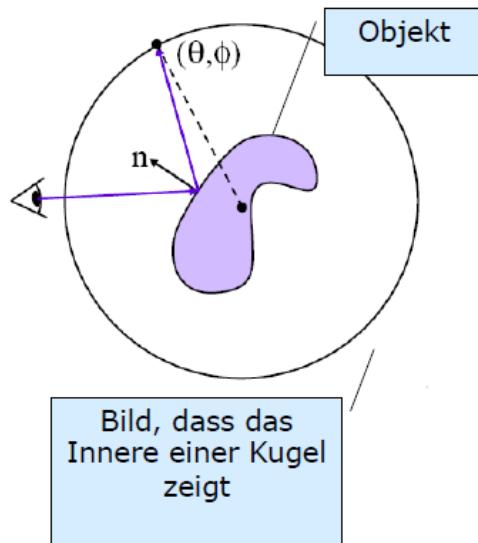


Abbildung 8.2: Environment Mapping Beispiel

8.4 Bump Mapping

Um eine etwas raue Oberfläche zu simulieren, könnte man tausende Polygone zusätzlich berechnen lassen - zu aufwändig. Daher verändert man einfach den Normalenvektor, mit dem man die Reflexionen des Lichts verändert zufällig.

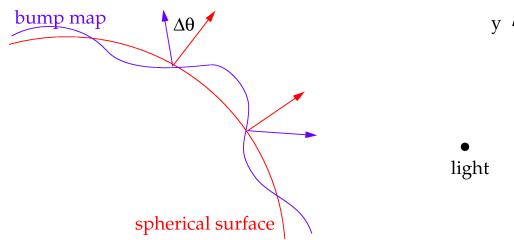


Abbildung 8.3: Bump Map Prinzip

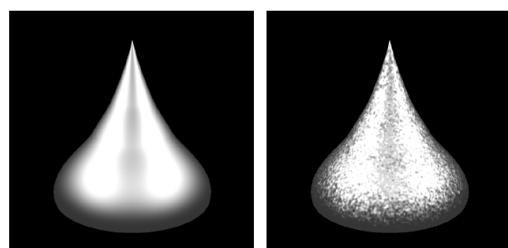


Abbildung 8.4: Bump Map Beispiel

9 Raytracing

9.1 Grundidee

Raytracing erzeugt fotorealistische Bilder, indem es Spiegelung und Transparenz berechnet. Grundsätzlich generiert man für jeden Pixel eine Linie in das Bild hinein und schaut dann, welches Objekt diese Linie zuerst schneidet. Wenn das erste Objekt gefunden wurde, dann berechne die Farbe. Simpel as that.

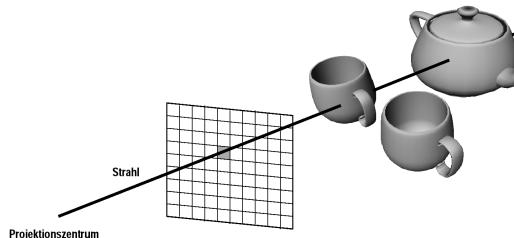


Abbildung 9.1: Raytracing Grundidee

9.2 Rekursives Raytracing

Mit dem ersten Strahl auf das Objekt ist noch nicht viel Realität in den PC geholt worden. Erst mit dem rekursiven Raytracing kommt das (Siehe Abbildung 9.2).

Wir haben also den ersten Strahl vom Betrachter aus - unseren *Primärstrahl*. Dieser geht vom Betrachter direkt auf den betrachteten Pixel. Von dort aus machen wir weitere Strahlen - einmal ein Reflexionsstrahl, der ähnlich wie der Primärstrahl einfach das nächste Objekt sucht und schaut, was dort für eine Farbe ist und diese dann dem ersten Punkt hinzufügt. Dann gibt es noch die Schattenstrahlen, die von der Lichtquelle aus berechnen, wie sehr diese jetzt den Punkt beleuchten. Dann gibt es den Transmissionsstrahl, der durch das Objekt hindurch geht, und die Farbe holt, die am anderen Ende des Objekts ist. Wir haben also:

1. **Primärstrahl** - Der erste Strahl vom Betrachter zum Objekt
2. **Schattenstrahl** - Der Strahl vom betrachteten Punkt zur Lichtquelle
3. **Reflektionsstrahl** - Der Strahl der Reflektion
4. **Transmissionsstrahl** - Der Strahl der durch das Objekt hindurch geht

Jetzt kommt es natürlich nur noch darauf an, wie tief man diese Strahlen verfolgt. Irgendwann sind sie nur noch so schwach, dass sie fast keinen Einfluss mehr darauf haben, wie das Bild aussieht, deswegen hört man auch irgendwann auf - und auch natürlich wegen der Rechenzeit ist es wichtig, diese *Strahlentiefe* zu begrenzen.

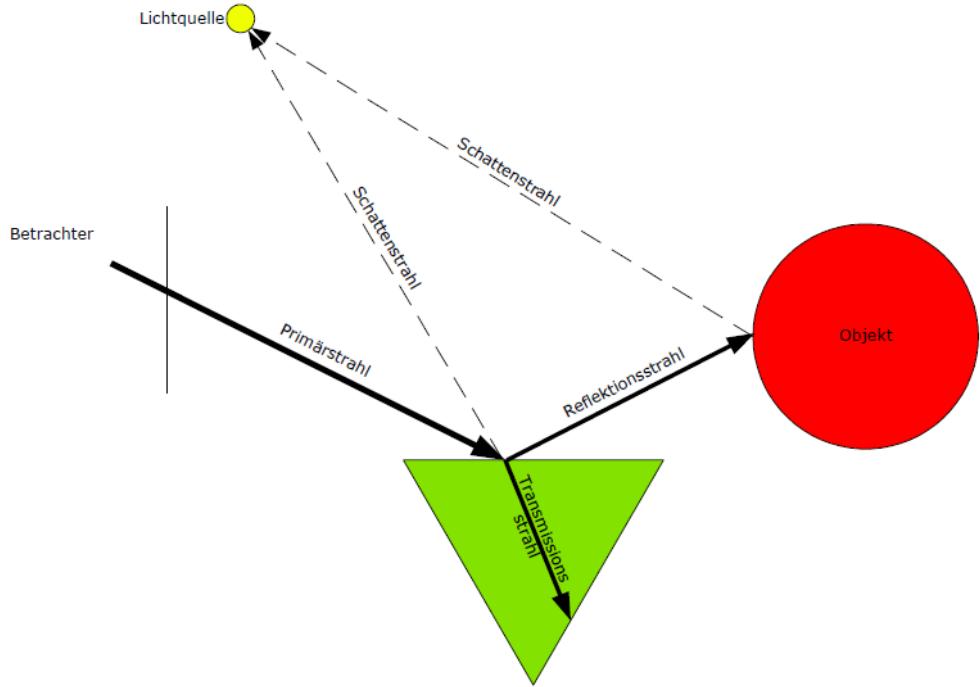


Abbildung 9.2: Rekursives Raytracing

9.2.1 Aliasing

Weil man nicht so schöne Bilder erhält mit dem Raytracing an sich, verwendet man noch ein adaptives Supersampling. Man macht also bis zu 16 Rays pro Pixel und fügt diese dann zu einem Pixel wieder zusammen.

10 Radiosity

Wir verwenden ein globales Beleuchtungsmodell für die korrekte diffuse Beleuchtung. Wichtig so für Architektur oder Industriedesign - da sollte es irgendwie realistisch aussehen. Das ganze nennt sich auch *Global Illumination*. Man unterscheidet hier nicht zwischen Leuchtquellen und beleuchteten Flächen - weil diese ja auch wieder reflektieren. Es ist also eigentlich ein geschlossenes System aus Sicht der Energieerhaltung. Im Gegensatz zu Raytracing wird hier das gesamte Modell schon einmal vorberechnet und beleuchtet, sodass der Betrachtungsstandort dann schlussendlich egal ist.

10.1 Detailkonzept

Wir haben ja verschiedene Objekte in einer Szene. Wir unterteilen jetzt die Oberflächen dieser Objekte in *Patches*. Ein Patch hat immer dieselbe Lichtintensität - ist also im Prinzip ein 'Licht-Pixel' irgendwie. Der Leuchtet dann auf andere Patches und andere Patches leuchten auf ihn. Das Ganze modellieren wir mit Hilfe von linearen Gleichungen. Da wir diese Gleichungen nie im Detail anschaut haben, werde ich sie hier mal nicht aufführen. Wir könnten Gleichungssysteme aber mit diesen Verfahren lösen:

1. Jacobi Iteration
2. Gauss Seidel Relaxation
3. Southwell Iteration

Ist aber grundsätzlich sehr Aufwändig so ein Global Illumination Modell zu berechnen. Vor allem der Austausch zwischen den Patches, der sog. *Formfaktor* ist am schwierigsten, z.B. hat er ein Doppelintegral und alles. Zudem muss zwischen jedem Patch berechnet werden, ob sie überhaupt untereinander sichtbar sind, wie viel Licht da ausgetaucht wird (Winkel usw), ... Deswegen nimmt man dort häufig geometrische Approximationen vom Ganzen, also z.B. mit Halbkugeln oder Würfel.

10.2 Optimierungen

10.2.1 Progressive Radiosity

Man zeigt schon früh einmal ein approximatives Resultat des Bildes und berechnet dann immer weiter die Details.

10.2.2 Hierarchical Radiosity

Die Patches zu verteilen ist ja sehr schwierig - macht man zu viele hat man zu lange zum Berechnen, zu wenig und es sieht doof aus. Also macht man hierarchical radiosity und nimmt einfach dort, wo es nötig ist ein paar mehr patches und sonst nicht. Es ist eher nötig, wenn viel Licht ausgetauscht wird oder die Patches nahe beieinander sind.

11 Codebeispiele

11.1 Vertexshader

11.1.1 2D

11.1.2 3D

```
1 attribute vec3 aVertexPosition;
2 attribute vec3 aVertexNormal;
3 attribute vec4 aVertexColor;
4
5 uniform mat4 uPerspectiveMatrix;
6 uniform mat4 uCameraMatrix;
7
8 uniform mat4 uTransformationMatrix;
9 uniform mat3 uNormalMatrix;
10
11 varying vec4 vColor;
12 varying vec3 vLighting;
13
14 void main () {
15     vColor = aVertexColor;
16     vec4 position = vec4 ( aVertexPosition , 1.0 );
17
18     vec3 ambientLight = vec3(0.5, 0.5, 0.5);
19     vec3 directionalLightColor = vec3(1, 1, 1);
20     vec3 directionalVector = vec3(0, 1, 0);
21     vec3 transformedNormal = normalize(uNormalMatrix * aVertexNormal);
22
23     float directional = max(dot(transformedNormal, directionalVector), 0.0);
24     vLighting = ambientLight + (directionalLightColor * directional);
25
26     gl_Position = uPerspectiveMatrix*uCameraMatrix*uTransformationMatrix*position;
27 }
```

11.2 Fragmentshader

11.2.1 2D

11.2.2 3D

```
1 precision mediump float;
2
3 varying vec4 vColor;
4 varying vec3 vLighting;
5
6 void main () {
```

```

7     gl_FragColor = vec4(vColor.rgb * vLighting, vColor.a);
8 }
```

11.3 Setup-Methode

```

1 function setupAttributes(shaderProgram) {
2     // finds the index of the variable in the program
3     aVertexPositionId = gl.getAttribLocation(shaderProgram, "aVertexPosition");
4     gl.enableVertexAttribArray(aVertexPositionId);
5
6     aVertexNormalId = gl.getAttribLocation(shaderProgram, "aVertexNormal");
7     gl.enableVertexAttribArray(aVertexNormalId);
8
9     aVertexColorId = gl.getAttribLocation(shaderProgram, "aVertexColor");
10    gl.enableVertexAttribArray(aVertexColorId);
11
12    uTransformationMatrixId = gl.getUniformLocation(shaderProgram, "uTransformationMatrix");
13    uPerspectiveMatrixId = gl.getUniformLocation(shaderProgram, "uPerspectiveMatrix");
14    uCameraMatrixId = gl.getUniformLocation(shaderProgram, "uCameraMatrix");
15    uNormalMatrixId = gl.getUniformLocation(shaderProgram, "uNormalMatrix");
16 }
```

11.4 Startup-Methode

```

1 function startup() {
2     canvas = document.getElementById("gameCanvas");
3     gl = createGLContext(canvas);
4
5     setupAttributes(loadAndCompileShaders(gl, "VertexShader.vert", "FragmentShader.
6         frag"));
7
8     gl.frontFace(gl.CCW); // defines how the front face is drawn
9     gl.cullFace(gl.BACK); // defines which face should be culled
10    gl.enable(gl.CULL_FACE);
11    gl.enable(gl.DEPTH_TEST);
12
13    //Background Color
14    gl.clearColor(0.0, 0.0, 0.0, 1.0);
15
16    perspective = mat4.create();
17    mat4.frustum(perspective, -1.02, 1.02, -1.02, 1.02, 0.2, 100);
18    gl.uniformMatrix4fv(uPerspectiveMatrixId, false, perspective);
19
20    camera = mat4.create();
21    mat4.lookAt(camera, [0, 0, 1.2], [0, 0, 0], [0, 1, 0]);
22    gl.uniformMatrix4fv(uCameraMatrixId, false, camera);
23
24    window.requestAnimationFrame(drawAnimated);
25 }
```

11.5 Draw-Methode

```
1 function draw() {  
2     gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);  
3     ...  
4 }
```