

2. FS - Graphikpipeline und Objektpräsentation

Rechtshändiges Koordinatensystem berechnen ob Punkt auf/hinter/vor der Ebene liegt:

Wenn man ein rechtshändiges Koordinatensystem vorausschickt und die Eckpunkte jedes Polygons (von vorne betrachtet) im mathematisch positiven Sinn (also gegen den Uhrzeigersinn) anordnet, dann gilt für einen Punkt (x, y, z) :

- wenn $Ax + By + Cz + D = 0$ dann liegt der Punkt **auf** der Ebene
- wenn $Ax + By + Cz + D < 0$ dann liegt der Punkt **hinter** der Ebene
- wenn $Ax + By + Cz + D > 0$ dann liegt der Punkt **vor** der Ebene

[2. Graphikpipeline und Objektrepräsentationen > Wichtige Begriffe](#)

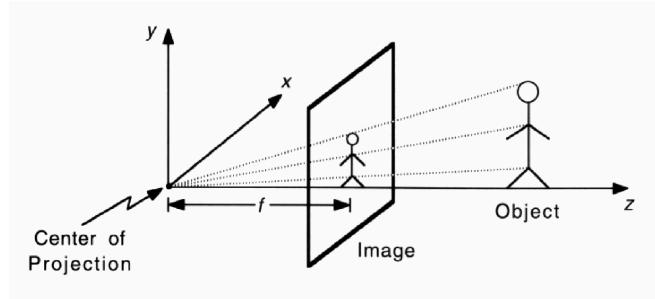
Berechnung des nach außen gerichteten Normalvektors:

$$N = (V2 - V1) \times (V3 - V1)$$

[2. Graphikpipeline und Objektrepräsentationen > Wichtige Begriffe](#)

2. FS - Bildaufnahme

Perspektivische Projektion



- Gegeben: 3D-Punkt $P = (X, Y, Z)$
- Projektionsgleichungen:
 - $x = \frac{f}{Z} \cdot X$
 - $y = \frac{f}{Z} \cdot Y$
- x und y : Position des projizierten Punktes auf der Bildecke

[2. Bildaufnahme > Mathematische Beziehung \(perspektivische Projektion\)](#)

Linsengleichung

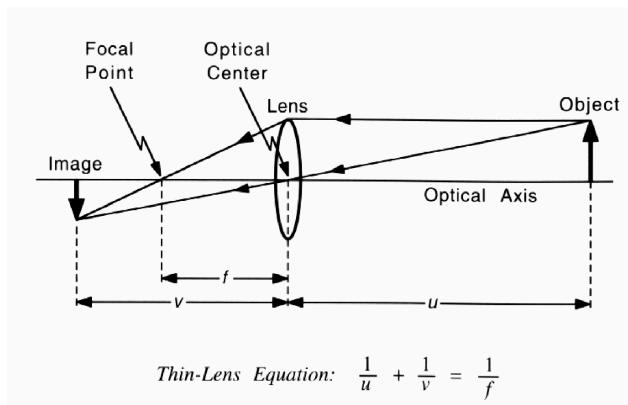


Abbildung 8: Linsengleichung

Liegt ein Objekt in einer endlichen Entfernung u , werden dessen Lichtstrahlen zu einem Bild hinter dem Brennpunkt in der Entfernung v fokussiert. Die zur Linsenachse senkrecht stehende Ebene wird Bildecke genannt. Die Beziehung zwischen der Entfernung u vom Objekt zur Linse und der Entfernung v von der Linse zur Bildecke wird durch die einfache Linsengleichung beschrieben, wie in Abbildung 8 dargestellt:

$$\frac{1}{u} + \frac{1}{v} = \frac{1}{f}.$$

Diese besagt, dass Objekte, die sehr weit weg sind ($u = \infty, v = f$), im Brennpunkt scharf abgebildet sind, alle Objekte, die näher sind, dahinter.

[2. Bildaufnahme > Linsengleichung](#)

3. FS - Transformationen

2D-Transformationen

Translation (Verschiebung)

Das Verschieben eines Punktes (x, y) um den Vektor (t_x, t_y) liefert den transformierten Punkt:

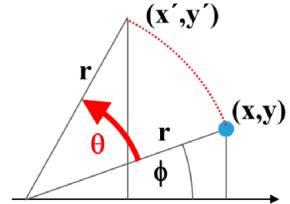
$$(x', y') = (x + t_x, y + t_y)$$

Rotation (Drehung)

Durch das Drehen eines Objektes mit dem Winkel θ um den Koordinatenursprung kommt der Punkt (x, y) wegen $x = r \cdot \cos\theta$ und $y = r \cdot \sin\theta \Rightarrow x' = r \cdot \cos(\phi + \theta) = r \cdot \cos\phi \cdot \cos\theta - r \cdot \sin\phi \cdot \sin\theta = x \cdot \cos\theta - y \cdot \sin\theta$ (und y' analog) auf

$$(x', y') = (x \cdot \cos\theta - y \cdot \sin\theta, x \cdot \sin\theta + y \cdot \cos\theta)$$

zu liegen.



Skalierung (Vergrößerung oder Verkleinerung)

Beim Skalieren eines Objektes um den Faktor s um den Ursprung $(0, 0)$ wird ein Punkt (x, y) auf

$$(x', y') = (s \cdot x, s \cdot y)$$

abgebildet. Wenn in x- und y-Richtung unterschiedliche Skalierungsfaktoren s_x und s_y verwendet werden, dann erhält man

$$(x', y') = (s_x \cdot x, s_y \cdot y).$$

Reflexion (Spiegelung)

Die Spiegelung an einer Koordinatenachse ist ein Sonderfall der Skalierung mit $s_x = -1$ oder $s_y = -1$.

Alle anderen Transformationen können durch Hintereinanderausführen der beschriebenen einfachen Transformationen erreicht werden. Diese Abbildungen (mit Ausnahme der Translation) lassen sich auch durch Transformationsmatrizen darstellen. Dabei werden die Punkte als Vektoren dargestellt, um mit ihnen die Matrixoperationen durchführen zu können:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} \quad \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} \quad \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} \quad (x', y') = (x + t_x, y + t_y) \quad \dots?$$

(a) Skalierung

(b) Rotation (gegen Uhrzeigersinn)

(c) Spiegelung um x-Achse

(d) Verschiebung

3. Transformationen > Einfache 2D-Transformationen

Homogene Koordinaten:

- $x = \frac{x'}{h}$
- $y = \frac{y'}{h}$

3. Transformationen > Homogene Koordinaten

Transformationen als Matrizen

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

(a) 2D-Skalierung

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

(b) 2D-Rotation

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

(c) 2D-Translation

3. Transformationen > Grundlegende Transformationen

Kurznotation der Transformationen

- $T(tx, ty)$ = Translation um Vektor (tx, ty)
- $R(\theta)$ = Rotation um Winkel θ
- $S(s_x, s_y)$ = Skalierung in x- und y-Richtung

Inverse Transformationen

- $T^{-1}(tx, ty) = T(-tx, -ty)$
- $R^{-1}(\theta) = R(-\theta)$
- ** $S^{-1}(sx, sy) = S(1/sx, 1/sy)$

3. Transformationen > **Kurznotation für Transformationen**

3D Transformationen

3D Transformationen

Alle 2D-Konzepte lassen sich leicht auf 3D erweitern. Man benötigt wieder eine homogene Komponente, so dass 4x4-Matrizen auf 4-dimensionalen Vektoren operieren. Später werden wir sehen, dass man auch Projektionen auf diese Art formulieren kann.

Hier sind einmal die wichtigsten 3D-Transformationen:

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \quad \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \quad \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(a) 3D-Skalierung

(b) 3D-Translation

(c) Spiegelung um yz-/xz-/xy-Ebene

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \quad \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \quad \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

(d) 3D-Rotation um x-Achse

(e) 3D-Rotation um y-Achse

(f) 3D-Rotation um z-Achse

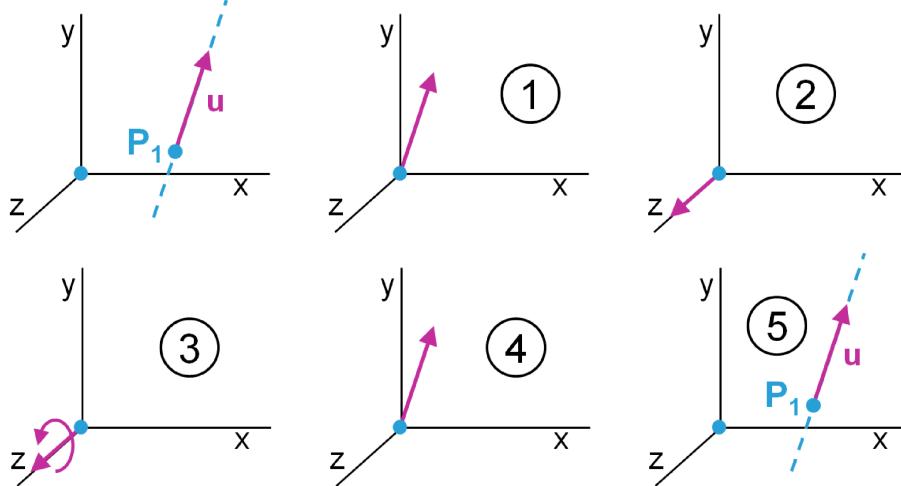
Die Namen für diese einfachen Transformationsmatrizen sehen nun so aus:

- $\mathbf{T}(\mathbf{t}_x, \mathbf{t}_y, \mathbf{t}_z)$ = Translation um den Vektor (t_x, t_y, t_z)
- $\mathbf{R}_x(\theta)$ = Rotation um den Winkel θ um die x-Achse; y- und z-Achse analog
- $\mathbf{S}(s_x, s_y, s_z)$ = Skalierung um die Faktoren s_x, s_y und s_z .

Als Beispiel für eine komplexere Transformation wollen wir eine

Drehung um den Winkel θ um eine beliebige Achse im Raum

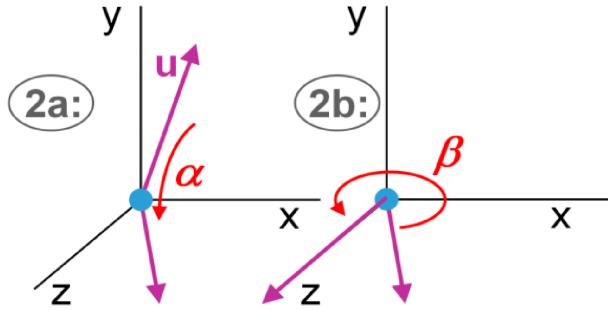
herleiten. Die Achse sei durch einen Punkt $P_1(x_1, y_1, z_1)$ und einen Richtungsvektor u gegeben.



1. Schritt = Punkt P_1 in den Koordinatenursprung verschieben: $T(-x_1, -y_1, -z_1)$
2. Schritt = Vektor u in die z-Achse drehen
 - (a) Vektor u um die x-Achse in die xz-Ebene drehen: $R_x(\alpha)$
Sei $u = (a, b, c)$, dann ist $u' = (0, b, c)$ die Projektion von u auf die yz-Ebene. Der Drehungswinkel α um die x-Achse ergibt sich aus $\cos\alpha = c/d$ mit $d = \sqrt{(b^2 + c^2)}$
 - (b) Vektor u um die y-Achse in die z-Achse drehen: $R_y(\beta)$
Der Drehungswinkel β um die y-Achse ergibt sich aus $\cos\beta = d$ (bzw. $\sin\beta = -a$)
3. Schritt = Drehung um θ um die z-Achse ausführen: $R_z(\theta)$
4. Schritt = Vektor u in die ursprüngliche Richtung zurückdrehen: zuerst $R_y(-\beta)$, dann $R_x(-\alpha)$
5. Schritt = Punkt P_1 an die ursprüngliche Position zurückverschieben: $T(x_1, y_1, z_1)$

Die resultierende Matrix berechnet sich also so:

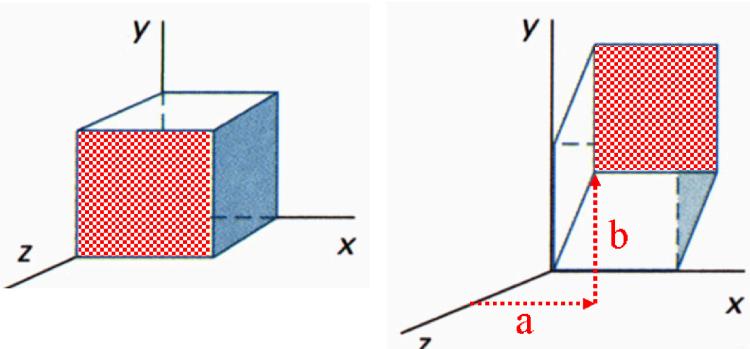
$$\begin{aligned} R(\theta) &= T^{-1}(-x_1, -y_1, -z_1) \cdot R_x - 1(\alpha) \cdot R_y - 1(\beta) \cdot R_z(\theta) \cdot R_y(\beta) \cdot R_x(\alpha) \cdot T(-x_1, -y_1, -z_1) = \\ &= T(x_1, y_1, z_1) \cdot R_x(-\alpha) \cdot R_y(-\beta) \cdot R_z(\theta) \cdot R_y(\beta) \cdot R_x(\alpha) \cdot T(-x_1, -y_1, -z_1) \end{aligned}$$



Die **Scherung in 3D** ist ebenfalls einfach darstellbar:

Eine Scherung parallel zur xy-Ebene um den Wert a in x-Richtung und den Wert b in y-Richtung erreicht man mittels

$$\begin{bmatrix} 1 & 0 & a & 0 \\ 0 & 1 & b & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Eine Scherung mit einer anderen Fixebene als einer der Koordinatenhauptebenen kann man sich leicht ableiten.

3. Transformationen > 3D Transformationen

3. FS - Bildcodierung und Kompression

Speicherplatz von Bild

- Wie viel Speicherplatz benötigt man für die Speicherung des Bildinhaltes bei einem **RGB Farbbild** der Größe **1.024x768**, wenn pro Farbkanal **4.096** verschiedene Werte kodiert werden sollen?

- $1.024 \times 768 = \textcolor{red}{786.432}$ Pixel
- $4.096 = 2^{12} \Rightarrow \textcolor{red}{12}$ Bit/Pixel
- RGB = **3** Farbkanäle

$$\text{Size} = L \times N \times B_{\text{c}}$$

$$\Rightarrow \textcolor{red}{786.432} \cdot \textcolor{red}{12} \cdot \textcolor{red}{3} = 28.311.552 \text{ Bit}$$

$$\Rightarrow 28.311.552 : \textcolor{red}{8} = 3.538.944 \text{ Byte}$$

$$\Rightarrow 3.538.944 : \textcolor{red}{1.024} = \textcolor{black}{3.456} \text{ KiloByte (KB)}$$



[3. Bildcodierung und Kompression > Speichergröße eines Rasterbildes](#)

2D - Diskrete Cosinus Transformation (DCT)

$$F(u, v) = \alpha(u) \cdot \alpha(v) \sum_{x=0}^7 \sum_{y=0}^7 f(x, y) \cos\left(\frac{(2x+1)u\pi}{16}\right) \cos\left(\frac{(2y+1)v\pi}{16}\right)$$

- u und v : **Horizontale** bzw. **vertikale Ortsfrequenz** ($0 \leq u, v < 8$).
- $f(x, y)$: Pixelwert am Punkt (x, y) .
- $F(u, v)$: DCT-Koeffizient, der das Signal in Frequenzkomponenten zerlegt.
- $\alpha(u)$ und $\alpha(v)$: **Skalierungsfaktoren** zur Wahrung der Orthonormalität.

[3. Bildcodierung und Kompression > Diskrete Cosinus Transformation \(DCT\)](#)

4. FS - Punktoperationen

Punktoperationen

Der neue Wert $I'(u, v)$ wird durch eine Funktion f des Originalwertes $I(u, v)$ bestimmt:

$$I'(u, v) = f(I(u, v))$$

[4. Punktoperationen > Was sind Punktoperationen?](#)

Affine Punktoperationen

$$I'(u, v) = a \cdot I(u, v) + b$$

[4. Punktoperationen > Unterklassen der Punktoperatoren](#)

Invertierung

$$I'(u, v) \rightarrow -I(u, v) + q = q - I(u, v)$$

[4. Punktoperationen > Identitätsfunktion und Invertierung](#)

Schwellwertoperation

$$p_{\text{th}} : \quad I'(u, v) \leftarrow f_{\text{th}}(I(u, v)) = \begin{cases} p_0, & \text{für } I(u, v) < p_{\text{th}} \\ p_1, & \text{für } I(u, v) \geq p_{\text{th}} \end{cases}$$

[4. Punktoperationen > Schwellwertoperation](#)

Kontrast und Helligkeit

$$I'(u, v) = I(u, v) \cdot 1.5$$

[4. Punktoperationen > Kontrast und Helligkeit](#)

Histogrammnormalisierung

$$I'(u, v) = \frac{(I(u, v) - q_{min})}{(q_{max} - q_{min})} \cdot q$$

[4. Punktoperationen > **Histogrammnormalisierung**](#)

Akkumuliertes Histogramm

$$Ha(x) = \sum_{k=0}^x H(k)$$

[4. Punktoperationen > **Vermeidung von Ausreißern durch Quantile**](#)

Histogrammausgleich

$$H_n(x) = \frac{q}{H_a(q)} \cdot H_a(x)$$

[4. Punktoperationen > **Berechnung des Histogrammausgleichs**](#)

5. FS - Rasterisierung

Linien

Linien werden in der Form $y = m * x + b$ angegeben wobei m den Anstieg beschreibt und $(0, b)$ den Schnittpunkt der y Achse.

Aus Endpunkten (x_0, y_0) und (x_1, y_1) kann man sich m und b berechnen:

$$m = \frac{(y_1 - y_0)}{(x_1 - x_0)}$$

$$b = y_0 - m * x_0$$

[5. Rasterisierung > Linienalgorithmen](#)

Bresenham-Verfahren

Bei Linien lässt sich der nächste Punkt so berechnen:

$$y = m * (x_k + 1) + b$$

(lässt sich ganz einfach aus Linearen Funktionen erschließen)

Hier werden dann nicht die genauen y Werte berechnet sondern lediglich die Entscheidung getroffen, ob y_k oder y_{k+1} näher zum exakten y -Wert liegt.

Abstand zu y_k ist:

$$d_{lower} = y - y_k = m * (x_k + 1) + b - y_k$$

Abstand zu y_{k+1} ist:

$$d_{upper} = (y_{k+1}) - y = y_k + 1 - m * (x_k + 1) - b$$

Nun berechnet man sich die Differenz zwischen d_{lower} und d_{upper} :

$$d_{lower} - d_{upper}$$

- Wenn diese Differenz negativ ist, dann nimmt man den unteren Punkt (x_{k+1}, y_k)
- Wenn positiv den oberen (x_{k+1}, y_{k+1})

Optimierung durch Entscheidungsvariable

Um keine Fließkommaoperationen (Multiplikation/Division) durchführen zu müssen, wird eine **Entscheidungsvariable** eingeführt. Dazu setzt man:

$$m = \frac{\Delta y}{\Delta x} \quad \text{mit} \quad \Delta x = x_1 - x_0, \quad \Delta y = y_1 - y_0$$

Multipliziert man die obige Differenz mit Δx , ergibt sich:

$$p_k = \Delta x \cdot (d_{lower} - d_{upper}) = 2\Delta y \cdot x_k - 2\Delta x \cdot y_k + c$$

Diese Entscheidungsvariable hat dasselbe Vorzeichen wie $d_{lower} - d_{upper}$, benötigt aber keine Division mehr.

Rekursive Berechnung der nächsten Entscheidungsvariable:

$$p_{k+1} = p_k + 2\Delta y - 2\Delta x \cdot (y_{k+1} - y_k)$$

Das bedeutet: Die neue Entscheidungsvariable lässt sich **einfach aus der vorherigen berechnen**, je nachdem, ob y erhöht wurde oder nicht – also ganz ohne Neuberechnung des exakten y -Werts.

Startwert:

$$p_0 = 2\Delta y - \Delta x$$

[5. Rasterisierung > Bresenham-Verfahren](#)

Mischen von Farben

B ... Hintergrundfarbe

F ... Vordergrundfarbe

$$F : P = t * F + (1 - t) * B$$

[5. Rasterisierung > Attribute von \(2D-\) Polygonen und Flächen](#)

Barzentrische Koordinaten berechnen

Für ein Dreieck mit P_0, P_1, P_2 , berechne für einen Punkt $P = (x, y)$:

$$g_{ij}(x, y) = (y_i - y_j)x + (x_j - x_i)y + (x_i y_j - x_j y_i)$$

Dann:

- $\alpha = \frac{g_{12}(x, y)}{g_{12}(x_0, y_0)}$

- $\beta = \frac{g_{20}(x, y)}{g_{20}(x_1, y_1)}$

- $\gamma = \frac{g_{01}(x, y)}{g_{01}(x_2, y_2)}$

Punkt liegt im Dreieck, wenn:

$$0 < \alpha < 1, \quad 0 < \beta < 1, \quad 0 < \gamma < 1$$

Kantendefinition (für benachbarte Dreiecke):

Damit Kanten **nicht doppelt** gezeichnet werden:

- Nur Pixel rendern, wenn Mittelpunkt **echt im Inneren liegt**
- Für Kanten auf dem Rand: z. B. **nur „unten“ und „rechts“ rendern**, nicht „oben“ oder „links“ → eindeutige Kantenregel

[5. Rasterisierung > Berechnen der baryzentrischen Koordinaten](#)

5. FS - Lokale Operationen

Pixel Nachbarschaften

Koordinaten der vier D-Nachbarn:

$$(u-1, v), (u+1, v), (u, v-1), (u, v+1)$$

Koordinaten der diagonalen Nachbarn:

$$(u-1, v-1), (u+1, v+1), (u-1, v+1), (u+1, v-1)$$

[5. Lokale Operationen > Nachbarschaften](#)

Filter

Gaußfilter

Beispiel für eine Filtermaske (für $\sigma = 0.5$):

$$F_{Gauss} = \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Laplacefilter

Filtermatrix:

$$F_{Laplace} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

[5. Lokale Operationen > Tiefpassfilter](#) und [5. Lokale Operationen > Differenzfilter](#)

Formale Eigenschaften linearer Filtern

$$I'(u, v) = \sum_i \sum_j I(u-i, v-j) \cdot F(-i, -j)$$

- Die ursprüngliche lineare Filterdefinition entspricht einer **linearen Korrelation**, da hier **keine Spiegelung** der Filtermatrix erfolgt.

Kommutativität:

$$I * F = F * I$$

Linearität:

- Skalierung eines Bildes:

$$(a \cdot I) * F = a \cdot (I * F)$$
- Addition zweier Bilder:

$$(I_1 + I_2) * F = (I_1 * F) + (I_2 * F)$$

$$(I_1 + I_2) * F = (I_1 * F) + (I_2 * F)$$

Assoziativitt:

$$A * (B * C) = (A * B) * C$$

→ Filter können **beliebig kombiniert** und **umgruppiert** werden.

[5. Lokale Operationen > Formale Eigenschaften lineare Filter](#)

Separierbarkeit

- Ein Filterkern F kann als **Faltungsprodukt kleinerer Filterkerne** beschrieben werden:

$$F = F_1 * F_2 * \dots * F_n$$
- Besonders nützlich: Trennung in **zwei eindimensionale Filter**:

Beispiel:

- $F_x = [1 \quad 2 \quad 1]$
- $F_y = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}$

Kombiniert:

$$F_{xy} = F_x * F_y$$

- Vorteil: **Reduktion der Rechenkomplexitt**
 - Normal: $3 \times 5 = 15$ Operationen pro Pixel
 - Separiert: $5 + 3 = 8$ Operationen pro Pixel

[5. Lokale Operationen > Konsequenz Separierbarkeit](#)

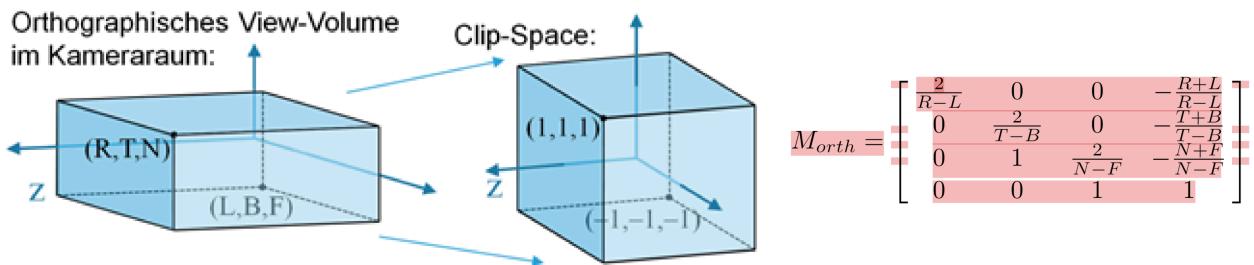
6. FS - Viewing

Viewport Transformation

$$\begin{bmatrix} x_{screen} \\ y_{screen} \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{n_x}{2} & 0 & 0 & \frac{n_x}{2} \\ 0 & \frac{n_y}{2} & 0 & \frac{n_y}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

[6. Viewing > Viewport-Transformation](#)

Projektionstransformation



Bemerkung: Eine Parallelprojektion kann auch schräg auf eine Abbildungsebene erfolgen (zum Beispiel beim Schattenwurf), diese Variante wird hier nicht berücksichtigt.

[6. Viewing > Projektionstransformation](#)

Kamera Transformation

e ... eye position

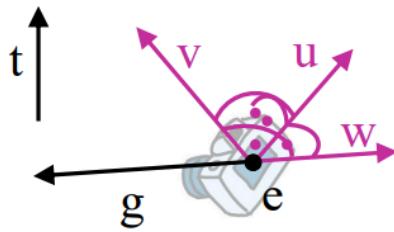
g ... gaze direction (positive w-axis points to the viewer)

t ... view-up vector

$$w = -\frac{g}{|g|}$$

$$u = \frac{t \times w}{|t \times w|}$$

$$\mathbf{v} = \mathbf{w} \times \mathbf{u}$$



6. Viewing > Kamera Transformation

Orthographisches Viewing

Viewing: Camera + Projection + Viewport

TU
WIEN

$$\begin{bmatrix} x_{\text{screen}} \\ y_{\text{screen}} \\ z \\ 1 \end{bmatrix} = M_{\text{vp}} \cdot M_{\text{orth}} \cdot M_{\text{cam}} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

viewport transformation *projection transformation* *camera transformation*

pixels on the screen world coordinates

6. Viewing > Orthographisches Viewing

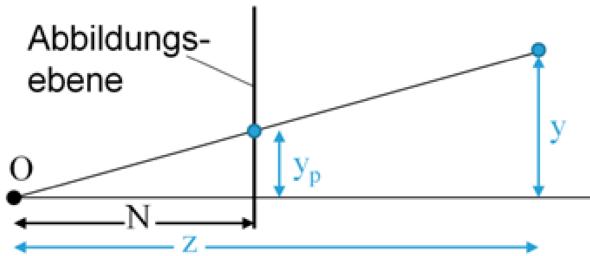
Perspektive

Abbildung eines Punktes (x, y, z) auf die Ebene:

$$(x, y, z) \rightarrow \left(\frac{x \cdot N}{z}, \frac{y \cdot N}{z}, N \right)$$

- Das lässt sich durch eine Matrix P darstellen:

$$P = \begin{bmatrix} N & 0 & 0 & 0 \\ 0 & N & 0 & 0 \\ 0 & 1 & N + F & -F * N \\ 0 & 0 & N & 1 \end{bmatrix}$$



$$y_p = \frac{N}{z} y \quad x_p = \frac{N}{z} x$$

- Ein Punkt $(x, y, z, 1)$ wird mit der **Projektionsmatrix** P multipliziert
- Ergebnis der Multiplikation: $(x \cdot N, y \cdot N, z \cdot (N+F) - F \cdot N, z)$
- Danach erfolgt das **Homogenisieren** (Normalisieren): Division durch die letzte Komponente z
- Ergebnis nach Division:

$$\left(\frac{x \cdot N}{z}, \frac{y \cdot N}{z}, (N + F) - \frac{F \cdot N}{z}, 1 \right)$$

Alternative Methode:

- **Einsetzen** der Projektionsmatrix P an der richtigen Stelle in der Gesamtviewingberechnung
- Dadurch wird eine **Gesamtmatrix** erzeugt, die die Transformation von Modellkoordinaten (x, y, z) zu Gerätekordinaten $(x_{\text{screen}}, y_{\text{screen}})$ in einem Schritt ausführt

$$\begin{bmatrix} x_{\text{screen}} \\ y_{\text{screen}} \\ z \\ 1 \end{bmatrix} = M_{vp} * \overbrace{(M_{orth} * P * M_{cam} * M_{mod})}^{M_{per}} * \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

6. Viewing > Perspektive

6. FS - Kantenfilterung

Gradienten/Kantenfilterung

$$\nabla I = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial I}{\partial x} \\ \frac{\partial I}{\partial y} \end{bmatrix}$$

- G_x und G_y sind die Gradientenkomponenten in horizontaler bzw. vertikaler Richtung

$$|\nabla I| = \sqrt{G_x^2 + G_y^2} \quad \text{und} \quad \theta = \text{atan2}(G_y, G_x)$$

[6. Kantenfilterung > Kantenfilterung](#)

Laplace-Operator

$$\nabla^2 f(x, y) = \frac{\partial^2 f(x, y)}{\partial x^2} + \frac{\partial^2 f(x, y)}{\partial y^2}$$

wobei:

$$\frac{\partial f}{\partial x^2}(x, y) = f(x+1, y) + f(x-1, y) - 2f(x, y)$$

in x Richtung und analog in y Richtung.

Fürs Filtern von einem Bild, zuerst Laplace Filter und dann das Ergebnis vom Ursprungsbild subtrahieren:

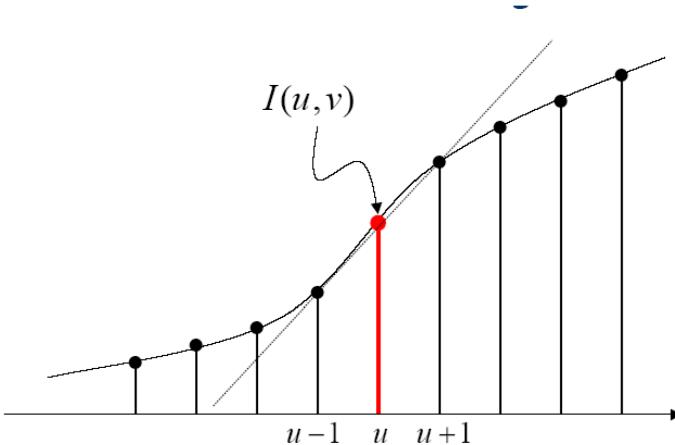
$$I' = I - w * (H^L * I)$$

- H^L ist Laplace Filter
- w bestimmt Intensität der Schärfung

[6. Kantenfilterung > Laplace-Operator](#)

Differenzenapproximation

$$\frac{df}{du}(u) \approx \frac{f(u+1) - f(u-1)}{2}$$



$$\frac{\partial I(u, v)}{\partial u} \approx I(u+1, v) - I(u-1, v)$$

Bei diskreten Signalen
nur Approximation möglich!

- Dies entspricht dem **Anstieg einer Geraden**, die durch die **benachbarten Abtastwerte** verläuft.

[6. Kantenfilterung > Lösung Differenzenapproximation](#)

Gradienten und Kanten in zweidimensionalen Bildern

Ein Bild wird als zweidimensionale Funktion $I(u, v)$ betrachtet

- Kanten im Bild = abrupte lokale Änderungen in Intensität oder Farbe
- Starke Änderungen = hohe Ableitungswerte → Hinweis auf Kante
- Erste Ableitung misst Stärke der Intensitätsänderung
- Für diskrete Funktionen ist die Ableitung nicht definiert → Approximation notwendig
- In 1D wird die erste Ableitung durch den Differenzenquotienten geschätzt:

$$\frac{df}{du}(u) \approx \frac{f(u+1) - f(u-1)}{2}$$

- In 2D: partielle Ableitungen entlang der Koordinatenachsen:

$$\frac{\partial I}{\partial u}(u, v), \quad \frac{\partial I}{\partial v}(u, v)$$

- Gradient Vektor:

$$\nabla I = \begin{pmatrix} \frac{\partial I}{\partial u} \\ \frac{\partial I}{\partial v} \end{pmatrix}$$

- Betrag des Gradienten (Kantenstärke):

$$\|\nabla I\| = \sqrt{\left(\frac{\partial I}{\partial u}\right)^2 + \left(\frac{\partial I}{\partial v}\right)^2}$$

- Betrag des Gradienten ist rotationsinvariant
- Horizontale Ableitung kann geschätzt werden durch linearen Filter:

$$H_{Dx} = \frac{1}{2} \cdot [-1 \quad 0 \quad 1]$$

- Vertikale Ableitung entsprechend:

$$H_{Dy} = \frac{1}{2} \cdot \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$$

- Filterantwort ist richtungsabhängig
- Horizontale Filter erkennen vertikale Kanten, vertikale Filter horizontale Kanten
- In flachen Bildregionen (konstante Intensität) ist die Filterantwort *null*

[6. Kantenfilterung > **Gradienten und Kanten in zweidimensionalen Bildern**](#)

Kantendetektion

- Gradienten in x- und y-Richtung:

$$D_x(u, v) = H_x * I$$

$$D_y(u, v) = H_y * I$$

- Kantenstärke:

$$E(u, v) = \sqrt{D_x(u, v)^2 + D_y(u, v)^2}$$

- Kantenrichtung:

$$\Phi(u, v) = \tan^{-1} \left(\frac{D_y(u, v)}{D_x(u, v)} \right)$$

[6. Kantenfilterung > Mathematische Formulierungen](#)

Weitere Kantenoperatoren

Roberts-Operator (ältester Kantenoperator)

- Sehr kleine Filtergröße: 2×2
- Schätzt Ableitungen entlang der Diagonalen
 - $H_{R1} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}, H_{R2} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$

Kirsch-Operator

Filter für **acht verschiedene Richtungen** im Abstand von **45°**

- Bietet höhere Präzision durch mehrere **enger aufgestellte Filter** für spezifische Richtungen
- Diese Acht Richtungen sehen so aus:

$$H_1^K = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, H_2^K = \begin{pmatrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{pmatrix}, H_3^K = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}, H_4^K = \begin{pmatrix} 0 & -1 & -2 \\ 1 & 0 & -1 \\ 2 & 1 & 0 \end{pmatrix},$$

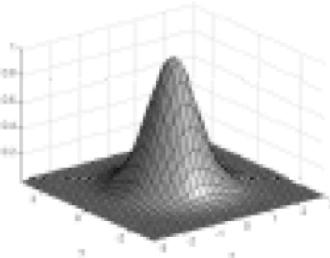
$$H_5^K = \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix}, H_6^K = \begin{pmatrix} 2 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -2 \end{pmatrix}, H_7^K = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}, H_8^K = \begin{pmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{pmatrix}$$

Die **Kantenstärke E^K** an der Stelle **(u,v)** ist als **Maximum** der einzelnen Filterergebnisse definiert, d.h. $E^K(u,v) = \max(D_1(u,v), D_2(u,v), \dots, D_8(u,v))$, und der am stärksten ansprechende Filter bestimmt auch die zugehörige **Kantenrichtung**. Derartige **Kompass-Operatoren** bieten allerdings kaum **Vorteile** gegenüber einfacheren Operatoren, wie z.B. dem Sobel-Operator. Ein Vorteil des Kirsch-Operators ist, dass er **keine Wurzelberechnung** benötigt.

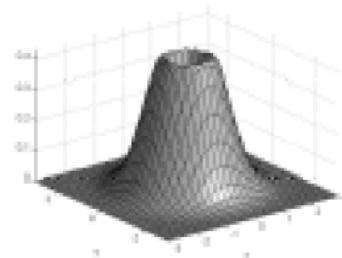
6. Kantenfilterung > Weitere Kantenoperatoren

2D-Gauß-Funktion

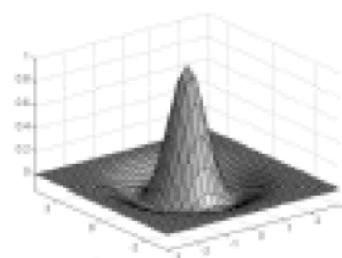
$$f(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$



Gauss' Bell
Function



First derivative



Minus second derivative
„Mexican Hat“

Wendet man den Laplace-Operator auf die Gauß-Funktion an, erhält man die **kontinuierliche Repräsentation des LoG**:

$$g(x, y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = -\frac{1}{\pi\sigma^4} e^{-\frac{x^2+y^2}{2\sigma^2}} \left(1 - \frac{x^2+y^2}{2\sigma^2}\right)$$

Die diskrete Approximation sollte für Kernel ungerader Kantenlänge durchgeführt werden, wobei der Ursprung des Kernels jeweils **in der Mitte** liegt. Für die Kantenlänge 5 entspricht dies dem unten abgebildeten Filter H^{LoG} . Die Abbildung zeigt ein Anwendungsbeispiel, die detektierten Kanten sind jetzt Ein Pixel breit.

$$H^{\text{LoG}} = \begin{pmatrix} 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & -2 & -1 & 0 \\ -1 & -2 & 16 & -2 & -1 \\ 0 & -1 & -2 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 \end{pmatrix}$$

6. Kantenfilterung > 2D-Gauß-Funktion

7. FS - Clipping und Antialiasing

Schnittpunktberechnung mit vertikalen Fensterkanten

Schnittpunktberechnungen mit vertikalen Fensterkanten:

- Für die linke Kante:

$$y = y_0 + m(x_{wmin} - x_0) \quad y = y_0 + m(x_{wmin} - x_0)$$

- Für die rechte Kante:

$$y = y_0 + m(x_{wmax} - x_0) \quad y = y_0 + m(x_{wmax} - x_0)$$

- Schnittpunktberechnungen mit horizontalen Fensterkanten:**

- Für die untere Kante:

$$x = x_0 + \frac{(y_{wmin} - y_0)}{m} x = x_0 + m(y_{wmin} - y_0)$$

- Für die obere Kante:

$$x = x_0 + \frac{(y_{wmax} - y_0)}{m} x = x_0 + m(y_{wmax} - y_0)$$

[7. Clipping und Antialiasing > Clippen von Linien Cohen-Sutherland-Verfahren](#)

Faltung

- Faltung:** Kombiniert zwei Funktionen und ergibt das integralgewichtete Summenprodukt der beiden.
 - Formel:** $f_1 * f_2(x) = \int_{-\infty}^{\infty} f_1(\tau) f_2(x - \tau) d\tau$
- Faltungstheorem:**
 - Multiplikation zweier Funktionen im Ortsraum entspricht der Faltung ihrer Spektren im Frequenzraum:
$$f_1 f_2 = F_1 * F_2$$
 - Faltung im Ortsraum entspricht der Multiplikation der Spektren im Frequenzraum:
$$f_1 * f_2 = F_1 F_2$$

[7. Clipping und Antialiasing > 2. Faltung](#)

7. FS - Globale Operationen

Kosinus- und Sinusfunktionen

- **Kosinusfunktion:**
 - $\cos(x)$ hat den Wert **1** am Ursprung ($\cos(0) = 1$)
 - Durchläuft von $x = 0$ bis $x = 2\pi$ eine volle Periode
- **Sinusfunktion:**
 - $\sin(x)$ hat den Wert **0** am Ursprung ($\sin(0) = 0$)
 - Durchläuft ebenfalls von $x = 0$ bis $x = 2\pi$ eine volle Periode

[7. Globale Operationen > Kosinus- und Sinusfunktionen](#)

Frequenz und Periode

Für $\cos(x)$ innerhalb einer Strecke der Länge $T = 2\pi$ ist die Anzahl der Perioden **1**:

$$\omega = \frac{2\pi}{T} = 1$$

[7. Globale Operationen > Frequenz und Perioden](#)

Addition von Cos/Sin Funktionen

$$C = \sqrt{A^2 + B^2}, \quad \phi = \tan^{-1} \left(\frac{B}{A} \right)$$

Erweiterung auf beliebige Funktionen

$$g(x) = \sum_{k=0}^{\infty} A_k \cos(k\omega_0 x) + B_k \sin(k\omega_0 x)$$

- **Fourierkoeffizienten:**
 - A_k, B_k : Bestimmen das Gewicht der jeweiligen Kosinus- und Sinusfunktionen.
 - Frequenzen der beteiligten Funktionen: Ganzzahlige Vielfache der Grundfrequenz ω_0 .
- **Fourieranalyse:** Berechnung der Fourierkoeffizienten aus der gegebenen Funktion $g(x)$.

[7. Globale Operationen > Addition von Kosinus- und Sinusfunktionen](#)

Fourierintegral

- **Fourierintegral:** Erweiterung auf **nicht periodische Funktionen**, die ebenfalls als Summe von Sinus- und Kosinusfunktionen dargestellt werden können:

$$g(x) = \int_0^\infty A_\omega \cos(\omega x) + B_\omega \sin(\omega x) d\omega$$

- **Koeffizienten** A_ω und B_ω :
 - Beschreiben die Amplitude der entsprechenden **Kosinus- bzw. Sinusfunktion** bei der Frequenz ω .
 - Bestimmung der Koeffizienten:

$$A_\omega = \frac{1}{\pi} \int_{-\infty}^{\infty} g(x) \cos(\omega x)$$

$$B_\omega = \frac{1}{\pi} \int_{-\infty}^{\infty} g(x) \sin(\omega x)$$

- **Spektrum der Funktion:**
 - $A(\omega)$ und $B(\omega)$ sind **kontinuierliche Funktionen**, die das **Spektrum** der Frequenzen im Signal repräsentieren.

Fouriertransformation und Fourier-Spektrum

- **Fouriertransformation:**
 - Vereinfacht die Darstellung, indem die Ausgangsfunktion $g(x)$ und das Spektrum als **komplexwertige Funktionen** betrachtet werden.
- **Fourierspektrum** $G(\omega)$:
 - Wird als **komplexe Funktion** dargestellt:

$$G(\omega) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} (\cos(\omega x) - i \sin(\omega x)) g(x)$$

[7. Globale Operationen > Fourierintegral – Nicht periodische Funktionen](#)

Diskrete Fourier-Transformationen

DFT: Vorwärtstransformation

Für ein diskretes Signal $g(u)$ der Länge M (mit $u = 0, \dots, M-1$), wird das **Fourierspektrum** $G(m)$ für $m = 0, \dots, M-1$ durch die **Vorwärtstransformation** berechnet:

$$G(m) = \frac{1}{\sqrt{M}} \sum_{u=0}^{M-1} g(u) e^{-i \frac{2\pi mu}{M}}$$

DFT: Inverse Transformation

Die **inverse DFT** zur Rekonstruktion des Signals $g(u)$ aus dem Spektrum $G(m)$:

$$g(u) = \frac{1}{\sqrt{M}} \sum_{m=0}^{M-1} G(m) e^{i \frac{2\pi mu}{M}}$$

- **Beide Transformationen sind identisch:** Vorwärts- und inverse DFT sind mathematisch symmetrisch.

Eigenschaften des Fourierspektrums

- Sowohl das **Signal** $g(u)$ als auch das **Fourierspektrum** $G(m)$ sind **komplexwertige Vektoren** der Länge M .
- **Betrag des Fourierspektrums (Magnitude):**

$$\|G(m)\| = \sqrt{G_{\text{real}}^2(m) + G_{\text{imag}}^2(m)}$$

[7. Globale Operationen > Diskrete Fourier-Transformation \(DFT\)](#)

Convolution Theorem

Faltung zweier Funktionen im Zeitbereich gleich der Punktweise-Multiplikation ihrer Fouriertransformierten im Frequenzbereich.

$$\mathcal{F}\{f * g\}(\omega) = F(\omega) \cdot G(\omega) Ff * g$$

[7. Globale Operationen > Convolution Theorem](#)

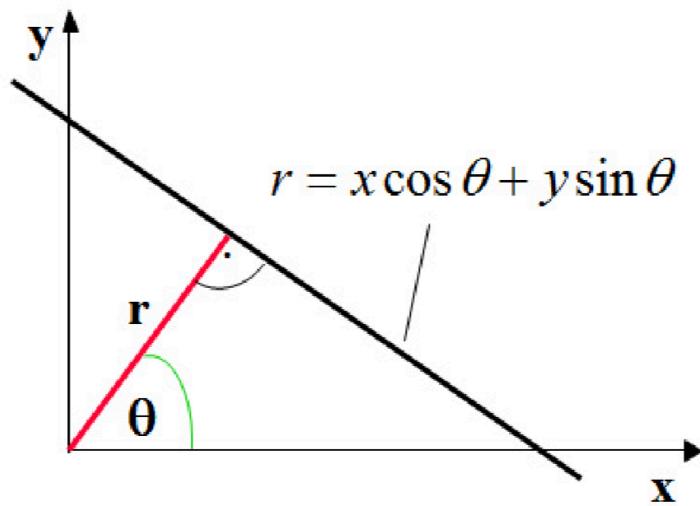
Hough Transformation

Andere Darstellung von Linien

Statt [5. FS - Rasterisierung > Linien](#) Darstellung wird hier die Hessesche Normalform verwendet:

$$r = x \cos(\theta) + y \sin(\theta)$$

- **Parameter:**
 - r : Normalabstand der Geraden zum Ursprung.
 - θ : Winkel des Normalabstands zur x-Achse.



zsmf by xmozz

8. FS - Sichtbarkeitsverfahren

Perspektivische Projektion

- **Berechnung:** Der Blickpunkt (x, y, z) wird in die Ebenengleichung eingesetzt.
- **Formel:** $Ax + By + Cz + D < 0 \Rightarrow$ Polygon ist unsichtbar.

[8. Sichtbarkeitsverfahren > 3. Berechnungsverfahren](#)
