

# Computergrafik

Mitschrift von

Markus Vieth

Steffen Eiden

Lukas Birklein

17. Januar 2017



# Vorwort

Dieses Skript basiert auf unserer Mitschrift der Vorlesung Computergrafik und VR im WS 2016/17 an der JGU Mainz (Dozent: Prof. Dr. E. Schömer).

Es handelt sich nicht um eine offizielle Veröffentlichung der Universität.

Wir übernehmen keine Gewähr für die Fehlerfreiheit und Vollständigkeit des Skripts.

Fehler können unter [Github](#) gemeldet werden. Die aktuelle Version dieses Skriptes ist ebenfalls auf [Github](#) zu finden.



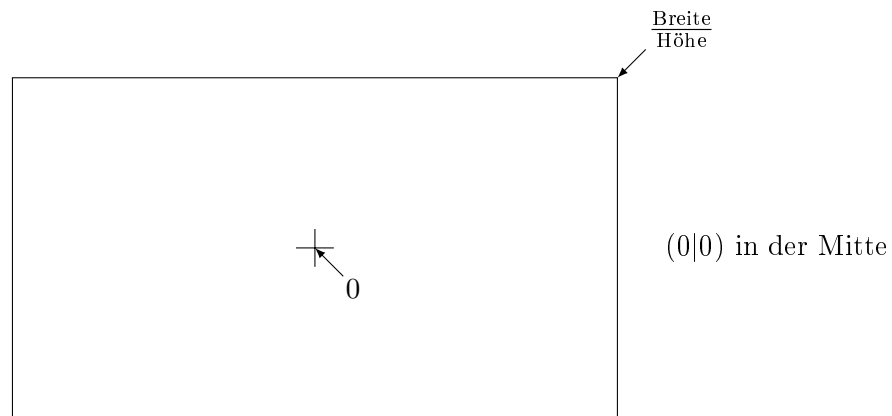
# Inhaltsverzeichnis

<b>Vorwort</b>	<b>i</b>
<b>1 Koordinatensysteme</b>	<b>1</b>
1.1 Normalisiertes Koordinatensystem . . . . .	1
1.2 Bildschirmkoordinaten $\Leftrightarrow$ Weltkoordinaten . . . . .	1
1.3 Homogene Koordinaten . . . . .	2
1.4 Transformationen . . . . .	3
1.4.1 Rotation . . . . .	3
1.4.2 + Verschiebung . . . . .	3
1.4.3 Skalierung . . . . .	3
1.4.4 Translation . . . . .	3
1.4.5 Hintereinanderausführung von Translation, Rotation und Skalierung . . . . .	4
1.5 Invertierung von $M$ . . . . .	4
1.6 Qt . . . . .	5
<b>2 VBO</b>	<b>6</b>
2.1 Baryzentrische Koordinaten . . . . .	7
2.2 Texturen . . . . .	8
2.2.1 Mipmap . . . . .	8
<b>3 3D-Objekte</b>	<b>10</b>
3.1 Orthogonalprojektion . . . . .	10
3.2 Perspektivische Projektion . . . . .	11
3.3 Objekte drehen aka Blickwinkel ändern . . . . .	14
<b>4 Qt Formen</b>	<b>15</b>
4.1 OFF-Format . . . . .	16
<b>5 Beleuchtung</b>	<b>17</b>
5.1 Ein einfaches Beleuchtungsmodell . . . . .	17
5.1.1 Lambertsches Gesetz . . . . .	17
<b>6 Einschübe</b>	<b>18</b>
6.1 Berechnung des Normalenvektors für parametrisierte Flächen . . . . .	18
6.2 Tiefenbuffer . . . . .	18
<b>7 Beleuchtung (Fortsetzung)</b>	<b>19</b>
7.1 Flatshading . . . . .	19
<b>8 Einschub: virtual Trackball</b>	<b>21</b>
8.1 Formel von Rodrigues . . . . .	21
8.1.1 dyadisches Produkt . . . . .	22

<b>9 Beleuchtung (Fortsetzung)</b>	<b>23</b>
9.1 Phong Lichtmodell . . . . .	23
9.1.1 Phong . . . . .	23
<b>10 Oberflächen</b>	<b>25</b>
10.1 Texturen . . . . .	25
10.2 Cube-Mapping . . . . .	26
<b>11 Volume Rendering mit 3D-Texturen</b>	<b>27</b>
11.1 DVT . . . . .	27
11.1.1 Lambert-Beer-Gesetz . . . . .	27
11.2 Raycasting mittels front-to-back rendering . . . . .	28
<b>12 Einschub: Algebraische Flächen</b>	<b>29</b>
<b>13 Shadow-Mapping</b>	<b>30</b>

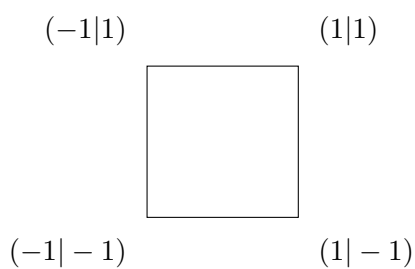
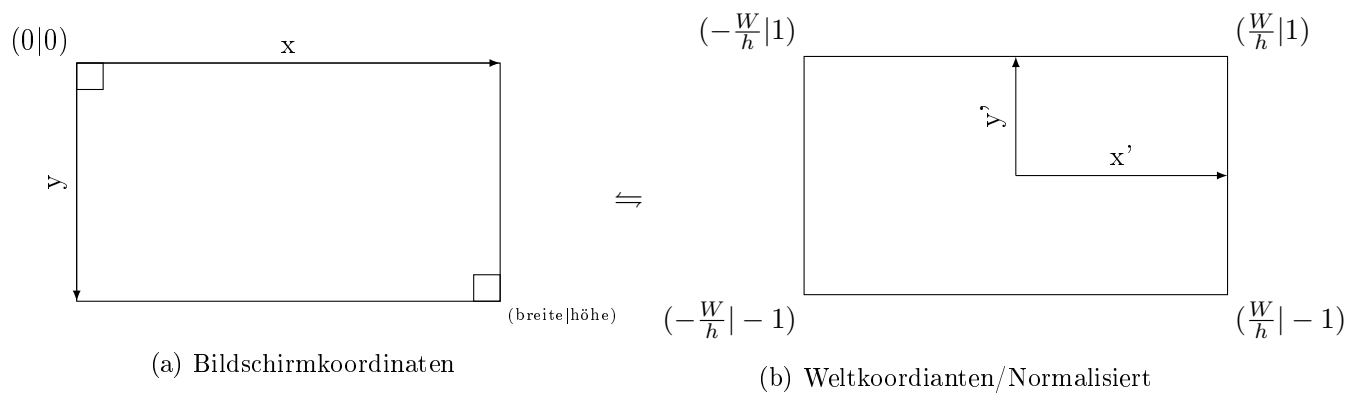
# 1 Koordinatensysteme

## 1.1 Normalisiertes Koordinatensystem



glViewport: Ausschnitt wo gezeichnet wird.

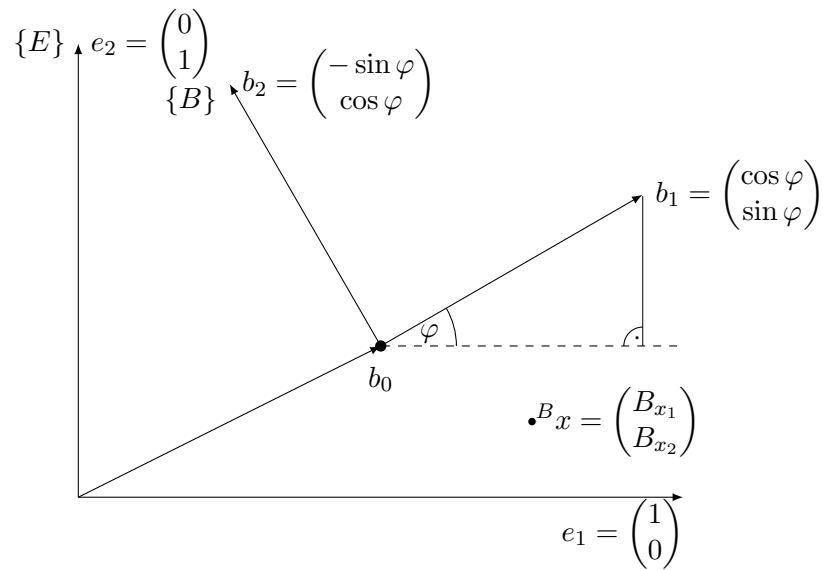
## 1.2 Bildschirmkoordinaten $\Leftrightarrow$ Weltkoordinaten



$$x' = ax + b$$

$$y' = cy + d$$

## 1 Koordinatensysteme



$b_0$ : E-Koordinaten des Ursprungs von System B

$b_1, b_2$  E-Koordinaten der Basisvektoren von System B

$${}^E x = b_0 + {}^B x_1 \cdot b_1 + {}^B x_2 \cdot b_2$$

$$[b_1, b_2] = R \in \mathbb{R}^{2 \times 2}, \quad |b_1| = |b_2| = 1, \quad \overbrace{b_1^T \cdot b - 2}^{\text{Standardskalarprodukt}} = 0$$

$$R^T \cdot R = \mathbb{E}, \det(R) = 1 \text{ (Rechtssystem)}$$

$$\Rightarrow {}^E x = b_0 + {}^E R \cdot {}^B x$$

### 1.3 Homogene Koordinaten

$${}^E x^1 = \begin{pmatrix} {}^E x_1 \\ {}^E x_2 \\ 1 \end{pmatrix} = \begin{pmatrix} {}^E x \\ 1 \end{pmatrix} \in \mathbb{R}^3$$

$$= \left( \begin{array}{cc|c} {}^E R_B & b_0 \\ \hline 0 & 1 \end{array} \right) \begin{pmatrix} {}^B x \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} \cos \varphi & -\sin \varphi & b_{0_1} \\ \sin \varphi & \cos \varphi & b_{0_2} \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} {}^B x_1 \\ {}^B x_2 \\ 1 \end{pmatrix}$$

Allgemein:

$${}^E x^1 = {}^E M_B \cdot {}^B \hat{x}$$

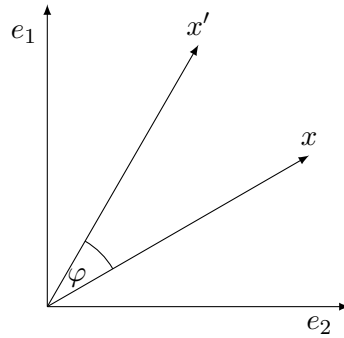
1.  ${}^E M_B$  beschreibt die Transformationsmatrix von Koordinaten aus System B in das System E
2.  ${}^E M_B$  kann auch interpretiert werden, als die (starre) Transformation, die E in B überführt.



## 1.4 Transformationen

### 1.4.1 Rotation

z.B.



$$x' = R \cdot x$$

### 1.4.2 + Verschiebung

$$x' = Rx + z$$

$$\rightsquigarrow \hat{x}' = Mx \text{ mit } M = \left( \begin{array}{c|c} R & t \\ \hline 0 & 1 \end{array} \right)$$

$$x' = R(x + t)$$

### 1.4.3 Skalierung

$$S = \begin{pmatrix} s_1 & 0 \\ 0 & s_2 \end{pmatrix} \quad x' = Sx = \begin{pmatrix} x'_1 = s_1 x_1 \\ x'_2 = s_2 x_2 \end{pmatrix}$$

homogenisiert

$$\hat{S} = \left( \begin{array}{c|c} S & 0 \\ \hline 0 & 1 \end{array} \right)$$

mit  $s_2 = -1$  Spiegelung um  $x_1$

### 1.4.4 Translation

Homogen:

$$\hat{T} = \left( \begin{array}{cc|c} 1 & 0 & t_1 \\ 0 & 1 & t_2 \\ \hline 0 & 0 & 1 \end{array} \right)$$

## 1 Koordinatensysteme

### 1.4.5 Hintereinanderausführung von Translation, Rotation und Skalierung

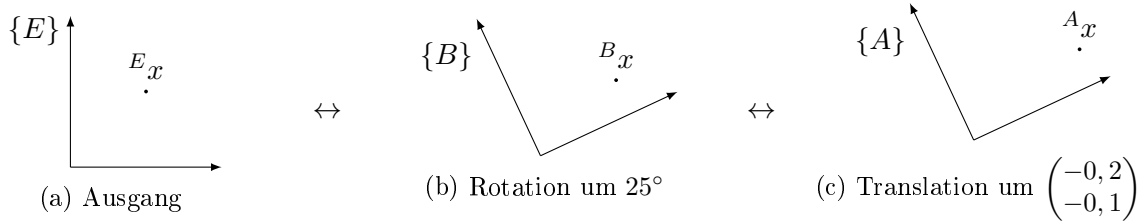
z.B.

$$x''' = S(R(x + t))$$

$$\rightsquigarrow x' = x + t; \quad x'' = Rx'; \quad x''' = Sx''$$

$$x''' = \underbrace{\hat{S}\hat{R}\hat{T}\hat{x}}_{\text{Leserichtung}} \quad \text{alles homogenisiert}$$

^ wird oft weggelassen, wenn klar.



$${}^B M_A {}^A x = {}^B x$$

$${}^E x = {}^E M_B {}^B x$$

$$\Rightarrow {}^E x = \underbrace{{}^E M_B {}^B M_A}_{{}^E M_A} {}^A x$$

z.B.

gegeben

$${}^E M_B \quad {}^E M_A$$

gesucht

$${}^B M_A$$

$${}^E M_A = {}^E M_B {}^B M_A$$

$$\Rightarrow {}^B M_A = {}^E M_B^{-1} \cdot {}^E M_A$$

$${}^E M_B^{-1} = {}^B M_E$$

## 1.5 Invertierung von $M$

Sei  $M = \left( \begin{array}{c|c} R & t \\ \hline 0 & 1 \end{array} \right)$   $\begin{matrix} R^{-1} \\ \uparrow \\ \text{Drehung um } -\varphi \end{matrix} = R^T$  u.a. auch, da Rotation ( $R^T R = 1$ )

$$Mx = Rx + t = x'$$

$$\rightsquigarrow R^T(x' - t) = x$$

$$\rightsquigarrow R^T x' - R^T t$$

$$\rightsquigarrow M^{-1} = \left( \begin{array}{c|c} R^T & -R^T t \\ \hline 0 & 1 \end{array} \right)$$

## 1.6 Qt

```

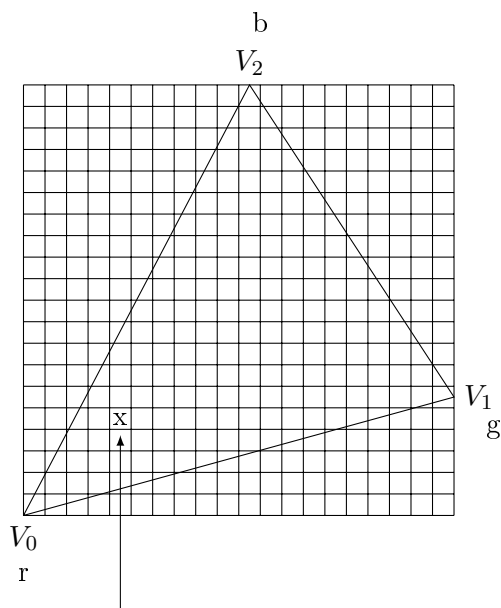
1  QMatrix4x4 M;
2  M.setToIdentity();    //  $M = \mathbb{1}$ 
3  M.rotate( $\varphi$ , 0, 0, 1); // Rotation um die z-Achse
4                          //  $M = M \cdot R$ 
5  M.scale( $s_1$ ,  $s_2$ );     //  $M = M \cdot S$ 
6  M.translate( $t_1$ ,  $t_2$ ); //  $M = M \cdot T$ 
7  //  $M = \underbrace{\mathbb{1} \cdot R \cdot S \cdot T}_{\text{Leserichtung für Transformation}}$ 
8  // Jeder Befehl wird zuerst ausgeführt! LIFO!
9  Mx'
```

## 2 VBO

	Kartesische Koord.	Farben	Textur-Koord.	Normal
$v_0$	$x_0, y_0, z_0, w_0$	$r_0, g_0, b_0$	$s_0, t_0$	$u_0, v_0, w'_0$
$v_1$				
$\vdots$				
$v_{n-1}$				

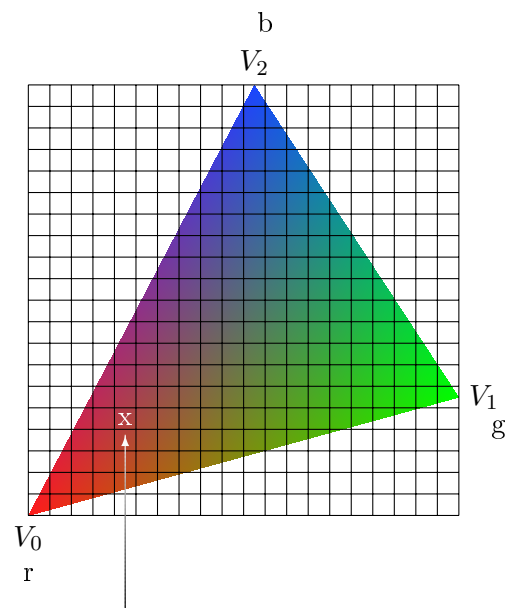
Entweder alles in einen Buffer  $\Rightarrow$  Array of Structs

Oder für jede Klasse einen eigenen Buffer  $\Rightarrow$  Struct of Array



interpolierte Farbe z.B. Mischung

(a) Beispiel Raster



interpolierte Farbe z.B. Mischung

(b) Beispiel Mischung der Farben

Shader nimmt die Attribute von den Randpunkten und prozessiert diese auf die Pixel im inneren des Dreiecks.

## 2.1 Baryzentrische Koordinaten

`varying` im Vertexshader

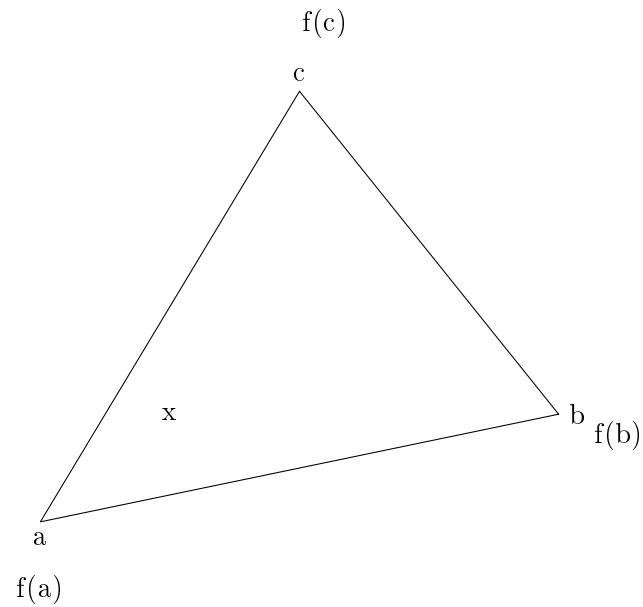


Abbildung 2.2: Baryzentrisches Koordinatensystem

$$\begin{pmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \\ 1 \end{pmatrix}$$

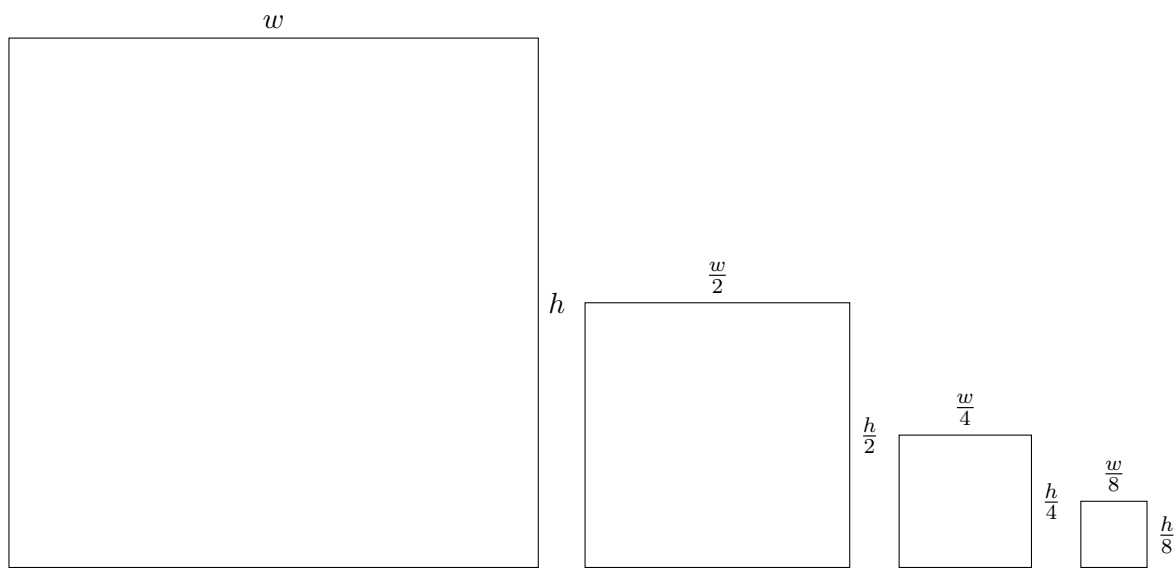
$$x = \alpha \cdot a + \beta \cdot b + \gamma \cdot c \wedge \alpha + \beta + \gamma = 1$$

$$\Rightarrow f(x) = \alpha \cdot f(a) + \beta \cdot f(b) + \gamma \cdot f(c)$$

(wird bei Qt durch das Keyword „Varying“ ausgelöst)

## 2.2 Texturen

### 2.2.1 Mipmap



$$S = \sum_{i=0}^{\infty} \left(\frac{1}{4}\right)^i = \frac{1}{1 - \frac{1}{4}} = \frac{4}{3}$$

Bildschirmpixel

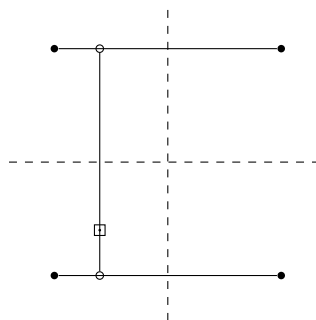
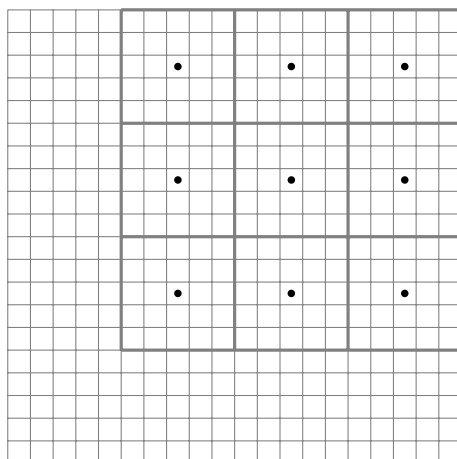
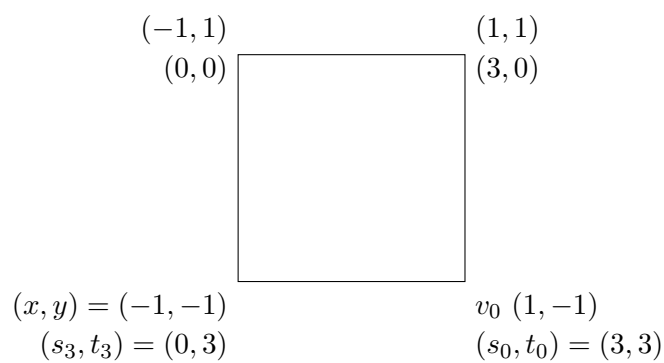


Abbildung 2.3: bilineare Interpolation



In der Regel sind Texturkoordinaten in  $[0, 1]^2$ , wenn größer wird sie periodisch verwendet.

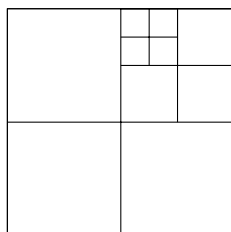
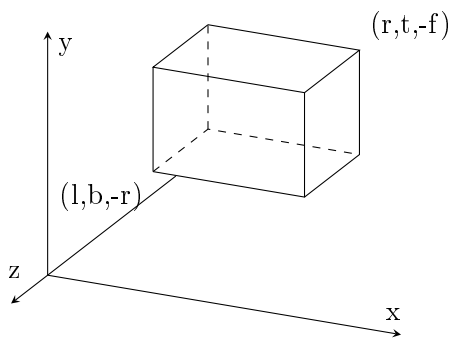


Abbildung 2.4: Maps Mipmapping

**Denkanstoß:** Welche Kette von Transformationen braucht man um zu einem Fixpunkt zu zoomen?

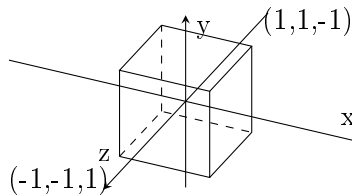
## 3 3D-Objekte

### 3.1 Orthogonalprojektion



$$\begin{aligned}x &\in [l, r] \\y &\in [b, t] \\z &\in [-f, -n]\end{aligned}$$

Sichtquader  $\rightarrow$  Einheitsquader



$$\begin{aligned}x' &\in [-1, 1] \\y' &\in [-1, 1] \\z' &\in [-1, 1]\end{aligned}$$

$$\begin{aligned}x' &= \alpha \cdot x + \beta \\l &\mapsto -1, \quad r \mapsto 1\end{aligned}$$

$$(1)$$

$$-1 = \alpha \cdot l + \beta$$

$$(2)$$

$$1 = \alpha \cdot r + \beta$$

$$(2) - (1)$$

$$2 = \alpha \cdot r - \alpha \cdot l \Rightarrow \alpha = \frac{2}{r-l}$$

$$1 = \frac{2 \cdot r}{r-l} + \beta$$

$$\beta = 1 - \frac{2r}{r-l} = \frac{r-l-2r}{r-l} = -\frac{r+l}{r-l}$$

Analog für  $y'$  und  $z'$



$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix}_{\text{NDS}} = \underbrace{\begin{pmatrix} \frac{2}{r-l} & 0 & 0 & \frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & \frac{t+b}{t-b} \\ 0 & 0 & \frac{-2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{pmatrix}}_O \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

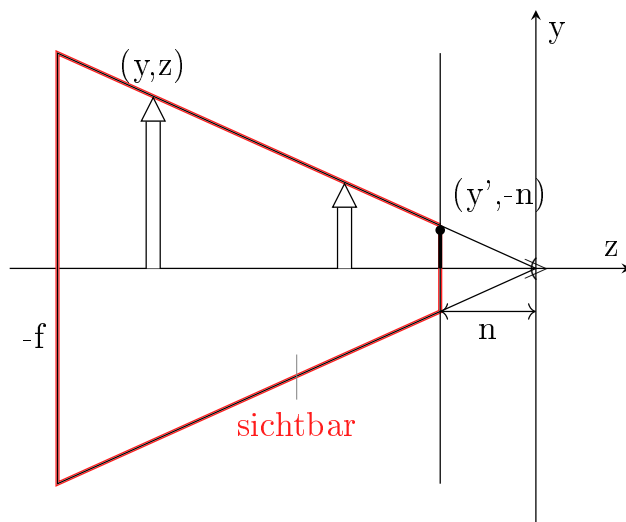
$$z' = -\frac{2}{f-n}z - \frac{f+n}{f-n}$$

$$z = n \quad z^* = \frac{2n - (f+n)}{f-n} = \frac{n-f}{f-n} = -1$$

$$-n \mapsto -1, \quad -f \mapsto 1$$

`Qmatrix4x4.ortho(1,n,b,t,n,f);` liefert  $O$

## 3.2 Perspektivische Projektion

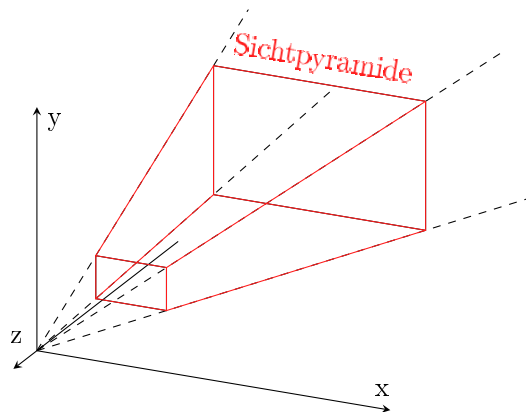


$$\frac{y'}{-n} = \frac{y}{z}$$

$$y' = -\frac{n \cdot y}{z}$$

### 3 3D-Objekte

Sichtpyramide  $\rightarrow$  Einheitswürfel



$$y' = -\frac{n \cdot y}{z}$$

$$\cap$$

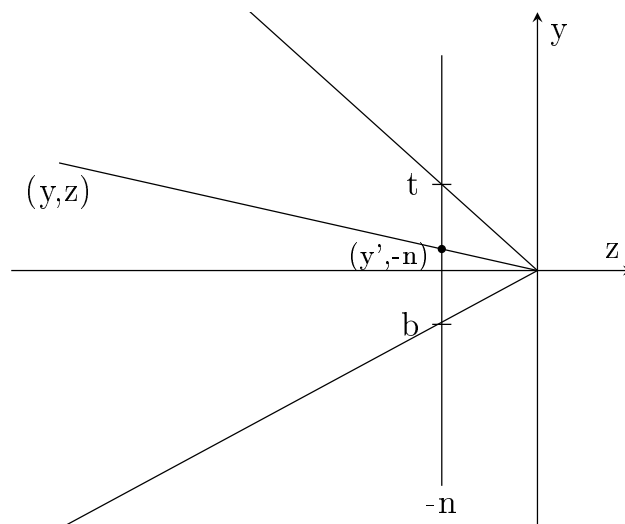
$$[b, t] \mapsto [-1, 1]$$

$$y'' = \alpha \cdot y' + \beta$$

$$y'' = \frac{2}{t \cdot b} \cdot y' - \frac{t+b}{t-b}$$

$$y'' = \frac{2n}{t-b} \cdot \frac{y}{-z} - \frac{t+b}{t-b}$$

analog für  $x''$ ,  $z''$



$$\begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} \xrightarrow{\text{Dehomogenisierung}} \begin{pmatrix} \frac{x}{w} \\ \frac{y}{w} \\ \frac{z}{w} \end{pmatrix}$$

homogene Koord.  Kartesische koord.

Homogenisierungsmatrix:

$$\begin{pmatrix} x'' \\ y'' \\ z'' \\ w'' \end{pmatrix} = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & -\frac{t+b}{t-b} & 0 \\ 0 & 0 & \alpha & \beta \\ 0 & 0 & -1 & 0 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

$$y'' = \frac{2n}{t-b} \cdot y + \frac{t+b}{t-b} \cdot z$$

$$w'' = -z$$

$$\frac{y''}{w''} = \frac{2n}{t-b} \frac{y}{(-z)} + \frac{t+b}{t-b} \frac{z}{(-z)}$$

$$z''' = \frac{z''}{w''} = \frac{\alpha \cdot z + \beta}{-z} = -\alpha - \frac{\beta}{z}$$

$$-n \mapsto -1, \quad -f \mapsto 15$$

$$-\alpha - \frac{\beta}{-n} = -1$$

$$-\alpha - \frac{\beta}{-f} = 1$$

$$-\alpha + \frac{\beta}{n} = -1(1)$$

$$-\alpha + \frac{\beta}{f} = 1(2)$$


---

$$\frac{\beta}{f} - \frac{\beta}{n} = 2(2) - (1)$$

$$\beta \left( \frac{1}{f} - \frac{1}{n} \right) = 2$$

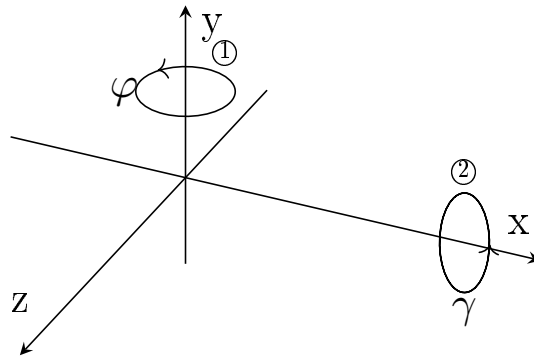
$$\beta \left( \frac{n-f}{fn} \right)$$

$$\beta = \frac{-2nf}{f-n}$$

$$\alpha = \frac{\beta}{f} - 1 = -\frac{2n-(f-n)}{f-n} = \frac{f+n}{f-n}$$

### 3.3 Objekte drehen aka Blickwinkel ändern

$$p' = \underset{\substack{\textcircled{2} \\ \uparrow \\ \text{Drehung um} \\ \text{die Welt-x-Achse}}}{R_{\vartheta,x}} \cdot \underset{\substack{\textcircled{1} \\ \uparrow \\ \text{Drehung um} \\ \text{die Welt-y-Achse}}}{R_{\varphi,y}} \cdot p$$



### 3D-Brille

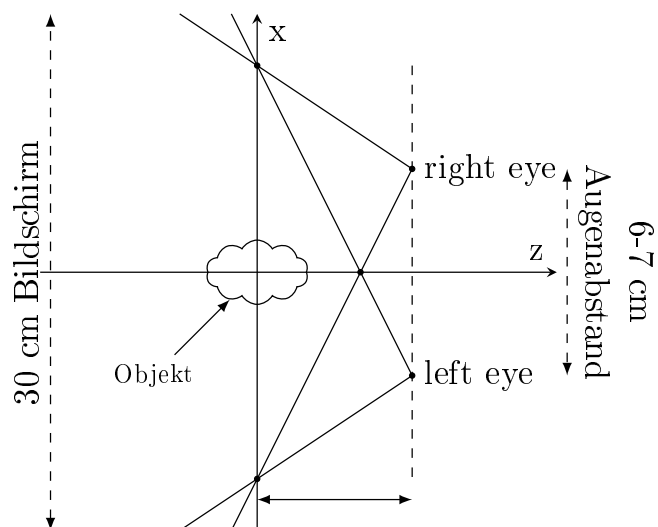
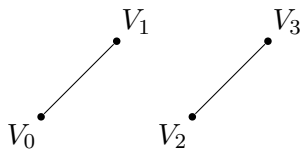
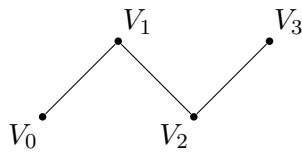


Abbildung 3.1: Streckenangaben sind beispielhaft

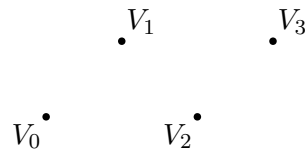
## 4 Qt Formen



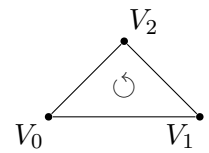
(a) GL\_LINES



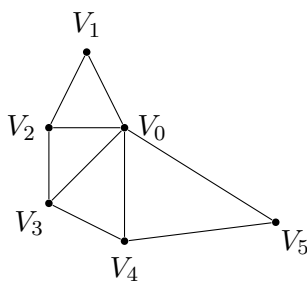
(b) GL\_LINE\_STRIP



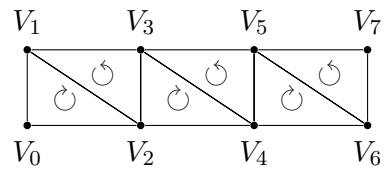
(c) GL\_POINTS



(d) GL\_TRIANGLE

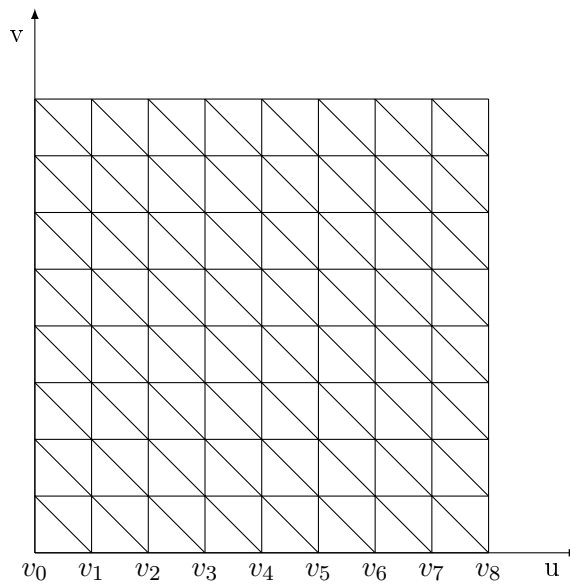


(e) GL\_TRIANGLE\_FAN



(f) GL\_TRIANGLE\_STRIP

Um den GL\_TRIANGLE\_STRIP abzuschließen wird der letzte Knoten (hier  $V_7$ ) zweimal gesendet.



$$f : \mathbb{R}^2 \rightarrow \mathbb{R}^3$$

$$(u, v) \rightarrow (x, y, z)$$

**Vertex Buffer Objects**  $\leftarrow$  Koordinatenzusatzinformationen

**Index Buffer Objects**  $\leftarrow$  Indices der Vertexe

4.1 OFF-Format

<i>n</i>	<i>f</i>	<i>e</i>
<i>x</i> <sub>0</sub>	<i>y</i> <sub>0</sub>	<i>z</i> <sub>0</sub>
<i>x</i> <sub>1</sub>	<i>y</i> <sub>1</sub>	<i>z</i> <sub>1</sub>
	⋮	
3	<i>i</i> <sub>1</sub>	<i>i</i> <sub>2</sub> <i>i</i> <sub>3</sub>
	⋮	

Index Face Set  
GL\_ELEMENT\_ARRAY\_BUFFER

# 5 Beleuchtung

## 5.1 Ein einfaches Beleuchtungsmodell

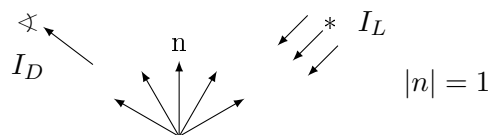
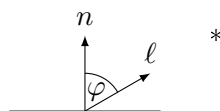


Abbildung 5.1: Diffuse Reflexion

### 5.1.1 Lambertsches Gesetz

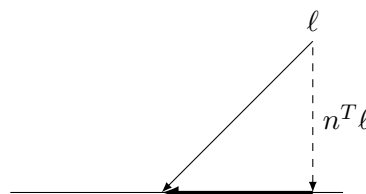


$$I_O = I_L \cdot \ell^T \cdot n$$

$$|n| = |\ell| = 1$$

$\ell$  zeigt zur Lichtquelle.

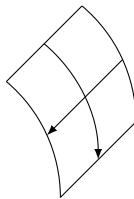
$$\cos(\varphi) = \frac{\ell^T \cdot n}{|n||\ell|} \left( = \frac{\langle \ell, n \rangle}{|n||\ell|} \right)$$



## 6 Einschübe

### 6.1 Berechnung des Normalenvektors für parametrisierte Flächen

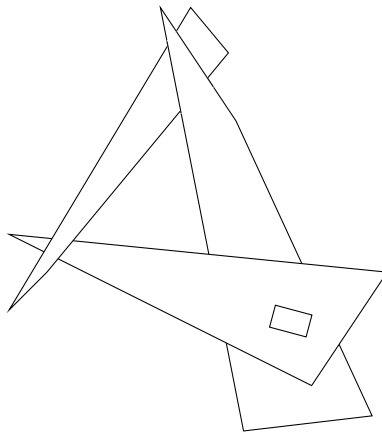
$$(u, v) \rightarrow (x, y, z) = f$$



$$\frac{\partial f(u, v)}{\partial u} \times \frac{\partial f(u, v)}{\partial v} = n$$

Im Allgemeinen gilt  $|n| \neq 1$

### 6.2 Tiefenbuffer



`glEnable(GL_DEPTH_TEST)` - Tiefentest aktivieren.

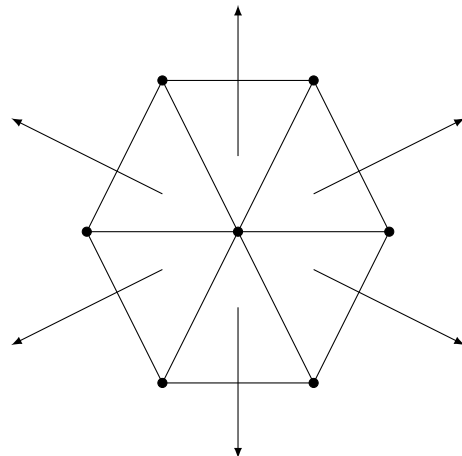
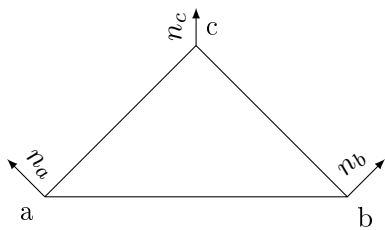


## 7 Beleuchtung (Fortsetzung)

### 7.1 Flatshading

Alle Knoten einer Fläche habe die gleiche Normale

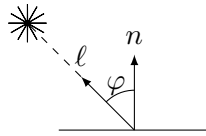
Normale ermitteln + normieren (u.U. Gewichten z.B. Fläche oder Winkel)



## 7 Beleuchtung (Fortsetzung)

### Lambert

$$I_D = I_L \cdot (n^T \cdot \ell)$$



$$x' = M \cdot x$$

$$E = \{x | n^T x = n_0\}$$

$$M \in \mathbb{R}^{3 \times 3}$$

$$\{Mx | n^T x = n_0\}$$

$$\text{z.B. } M = R$$

$$R^{-1} = R^T$$

$$R : \text{Rotation}$$

Transformationsregel

$$x' = \mathbb{R}x$$

$$n' = \mathbb{R}n$$

### Behauptung

$$x' = Mx$$

$$\text{für Rotationen } (R^{-1})^T = \mathbb{R}$$

$$n' = (M^{-1})^T n$$

### Beweis

$$\begin{aligned} n'^T x' &= \left( (M^{-1})^T n \right)^T Mx = n^T M^{-1T} \cdot Mx = n^T M^{-1} Mx \\ &= n^T x = n_0 \end{aligned}$$

**Qt**  $(M^{-1})^T$ . `QMatrix4x4::normalMatrix`

## 8 Einschub: virtual Trackball

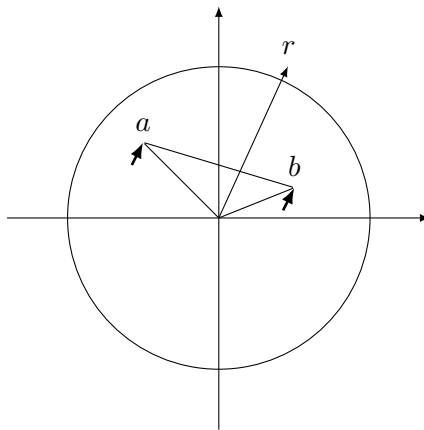


Abbildung 8.1: Virtueller Trackball

$$a = (x_a, y_a, \sqrt{1 - x_a^2 - y_a^2})$$

$$b = (x_b, y_b, \sqrt{1 - x_b^2 - y_b^2})$$

$$r = a \times b \pm \text{normieren}$$

Bewege  $a$  nach  $b$  auf einem Großkreis

Normale der Großkreisebene ist  $\frac{a \times b}{|a \times b|} = r \hat{=}$  Rotationsachse

**Winkel**

$$\cos \sphericalangle(a, b) = \frac{a \cdot b}{|a \cdot b|}$$

### 8.1 Formel von Rodrigues

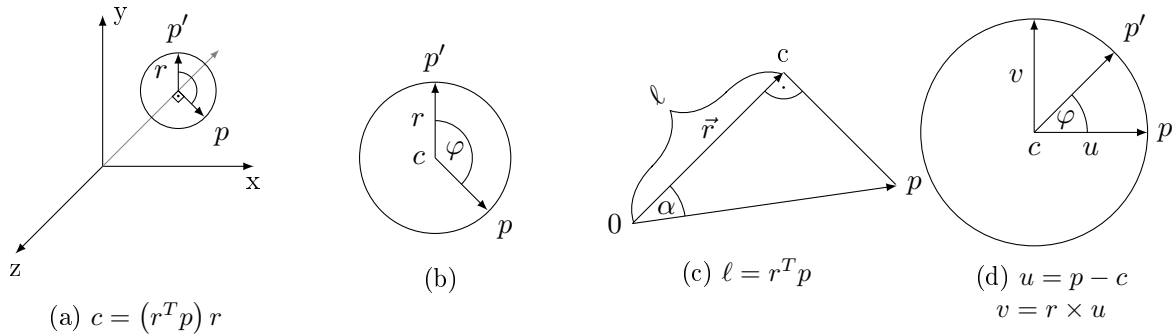
Achse, Winkel  $\rightarrow$  Rotationsmatrix

$$(r, \varphi) \rightarrow R$$

$$|r| = 1$$

```
1 QMatrix4x4 R;
2 R.rotate(  $\underbrace{f}_{\text{deg}}$ , r);
```

## 8 Einschub: virtual Trackball



zu c

$$\begin{aligned} \ell &= \cos \alpha \cdot |p| \\ &= \underbrace{\frac{r^T p}{|r| |p|}}_{=1} |p| \end{aligned}$$

$$\begin{aligned} Rp = p' &= c + \cos \varphi u + \sin \varphi v \\ &= c + \cos \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \sin \varphi \begin{pmatrix} 0 \\ 1 \end{pmatrix} \\ &= c + \cos \varphi (p - c) + \sin \varphi (r \times (p - c)) \\ &= (1 - \cos \varphi) c + \cos \varphi p + \sin \varphi (r \times p) \\ &= (1 - \cos \varphi) (rr^T) p + \cos \varphi p + \sin \varphi (r^x p) \end{aligned}$$

da  $r \times c = 0$

$$(AB)C = A(BC)$$

$$r^T r = \begin{pmatrix} r_1^2 & r_1 r_2 & r_1 r_3 \\ r_2 r_1 & r_2^2 & r_2 r_3 \\ r_3 r_1 & r_3 r_2 & r_3^2 \end{pmatrix}$$

$$\boxed{= (1 - \cos \varphi) rr^T + \cos \varphi I + \sin \varphi r^x}$$

### 8.1.1 dyadisches Produkt

$$(rr^T)^T = r^{TT} r^T = rr^T \Rightarrow \text{symmetrisch}$$

$$r \times p = Ap$$

$$\begin{pmatrix} r_1 \\ r_2 \\ r_3 \end{pmatrix} \times \begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix} = \begin{pmatrix} r_2 p_3 - r_3 p_2 \\ r_3 p_1 - r_1 p_3 \\ r_1 p_2 - r_2 p_1 \end{pmatrix} = \begin{pmatrix} 0 & -r_3 & r_2 \\ r_3 & 0 & -r_1 \\ -r_2 & r_1 & 0 \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix}$$

$A = \text{Axiator von } r$

$$r^x := A$$

Qt

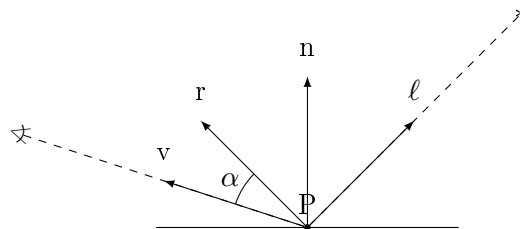
$$R - e^{\varphi r^x} \quad e^A = \sum_{n=0}^{\infty} \frac{1}{n!} \cdot A^n$$

$$\begin{aligned} r^{xT} &= -r^x \\ \text{sch}(A) &= \sum_{n=0}^{\infty} \frac{1}{(2n+1)!} A^{2n+1} \cdot (-1)^n \end{aligned}$$

Alternative: Quaternionen

## 9 Beleuchtung (Fortsetzung)

### 9.1 Phong Lichtmodell

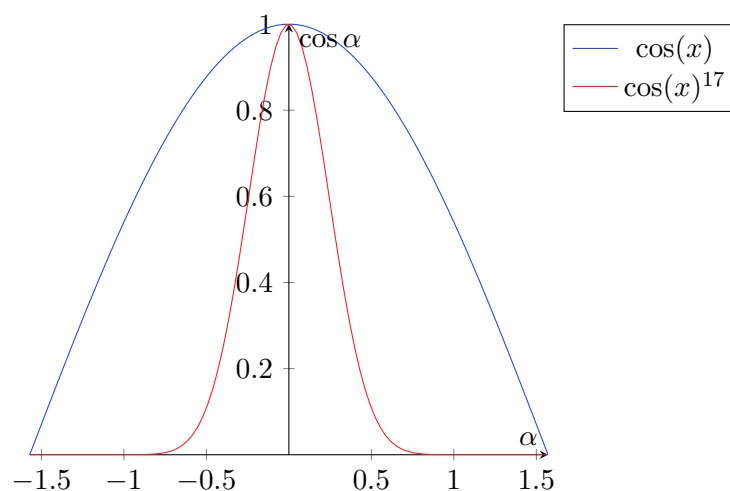


$$|n| = |\ell| = |r| = |v| = 1$$

$$r = 2n(n^T \ell) - \ell$$

S = Shininess

$$I_S = I_L (\cos \alpha)^S = I_L (r^T v)^S, \quad I_D = I_L (n^T \ell)$$



#### 9.1.1 Phong

$$I_{\text{Color}} = I_{\text{Ambient, Color}} + I_{\text{Diffuse, Color}} + I_{\text{Specular, Color}}$$

$$\text{Color} \in \{\text{Red, Green, Blue}\}$$

```

1 void main() {
2     vec3 normal = normalize(vNormal);
3     vec3 lightDir = normalize(lightPos - vPos);
4     vec3 reflectDir = reflect(lightDir, normal);
5     vec3 viewDir = normalize(-vPos);

7     float lambertian = max(dot(lightDir, normal), 0.0.);

```

## 9 Beleuchtung (Fortsetzung)

```
8     float specular = 0.0;
10    if ( lambertian > 0.0) {
11        float specAngle = max(dot(reflectDir, viewDir), 0.0);
12        specular = pow(specAngle, uShininess);
13    }
14    gl_FragColor = vec4(uAmbient + lambertian * uDiffuse + specular * uSpecular, 1.0);
15 }
```

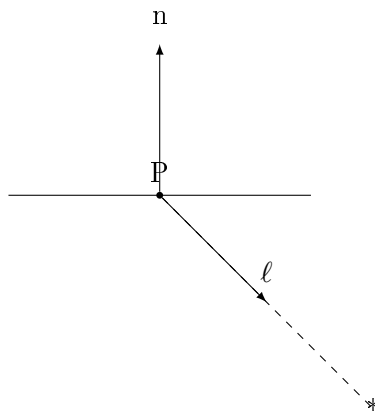
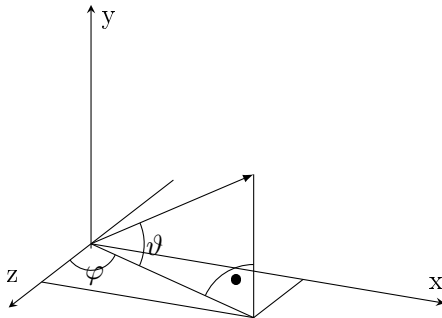


Abbildung 9.1: Zu ignorierende Lichtquelle

# 10 Oberflächen

## 10.1 Texturen



$$(\varphi, \vartheta) \mapsto \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \cos \vartheta \cdot \sin \varphi \\ \sin \vartheta \\ \cos \vartheta \cdot \cos \varphi \end{pmatrix}$$

$$0 \leq \varphi \leq 2\pi$$

$$-\frac{\pi}{2} \leq \vartheta \leq \frac{\pi}{2}$$

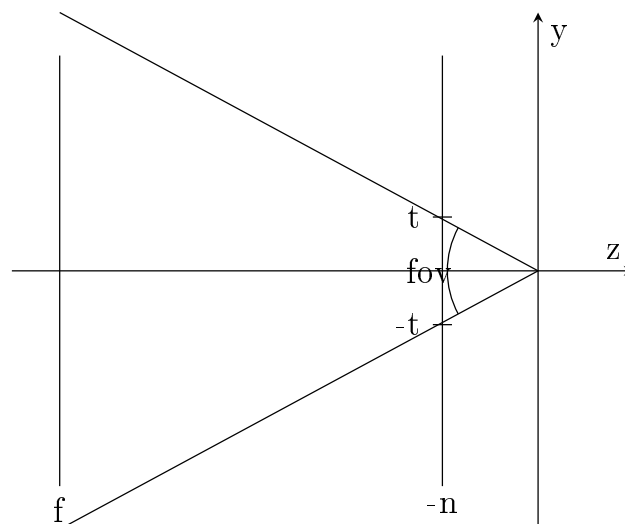
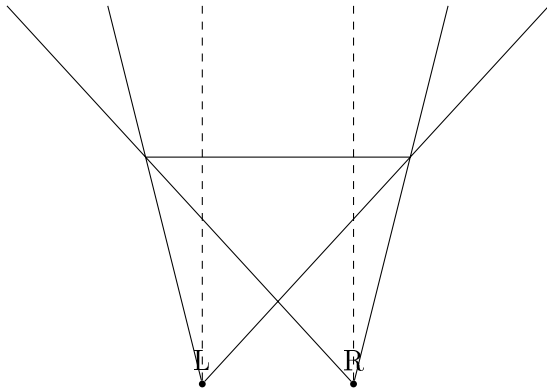


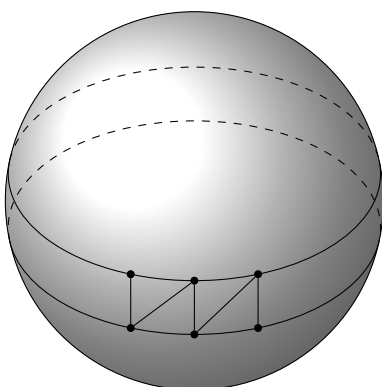
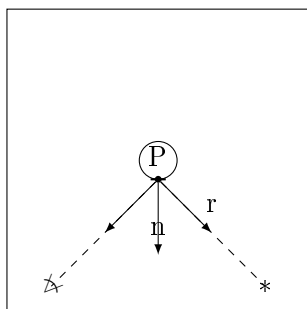
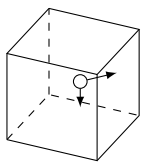
Abbildung 10.1: Field of view

```
perspective(fov, aspectratio, n, f);
```

## 10 Oberflächen



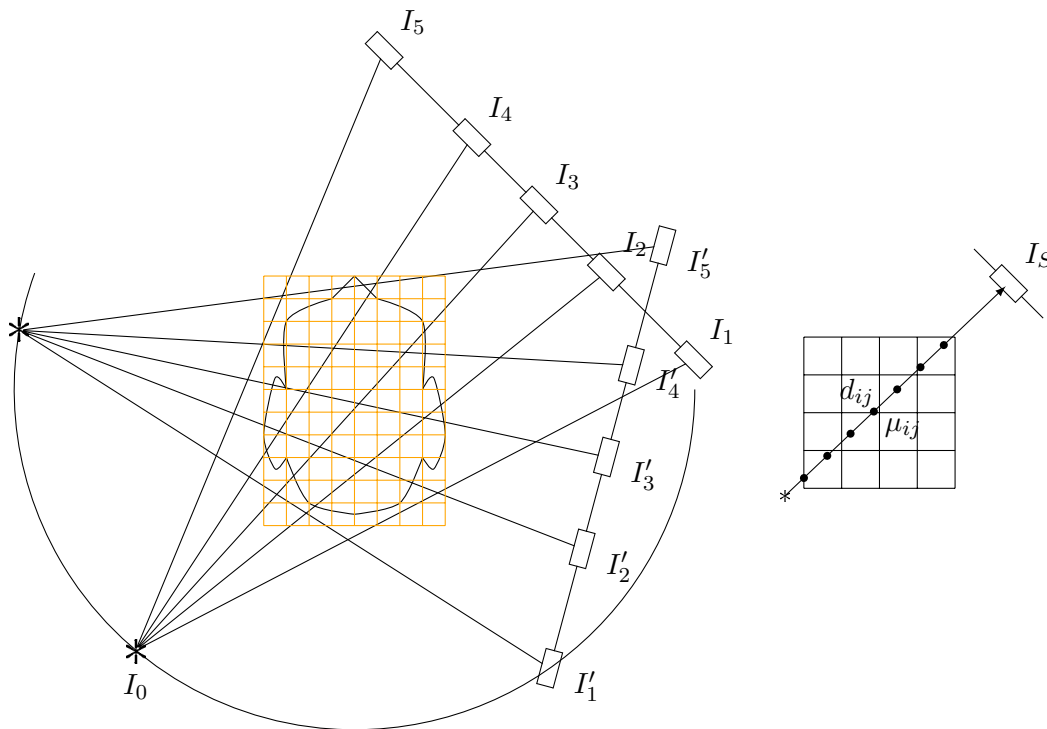
### 10.2 Cube-Mapping





# 11 Volume Rendering mit 3D-Texturen

## 11.1 DVT



### 11.1.1 Lambert-Beer-Gesetz

$$I^{\text{out}} = I^{\text{in}} \cdot e^{-\mu \cdot d}$$

Schwächungskoeffizienten

$$I_S = I_0 \cdot e^{\sum_{\square(i,j) \cap S \neq \emptyset} \mu_{ij} \cdot d_{ij}}$$

$$\ln \frac{I_0}{I_S} = \sum_{\square(i,j) \cap S \neq \emptyset} \mu_{ij} \cdot d_{ij}$$

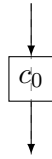
$$I = \bigcap_{\mathbb{R}^{360.000.000 \times 128.000.000}} D \cdot \mu$$

### Inverses Problem

**Gegeben:** Gemessene Intensitäten  $I_S$  für alle Strahlen, die die Bildebene treffen für hinreichend viele Aufnahmegerichtungen.

## 11 Volume Rendering mit 3D-Textures

**Gesucht:** Schwächungskoeffizienten für alle Voxel des zu rekonstruierenden Volumens.



$$c^{\text{out}} = c^{\text{in}} \cdot (1 - \alpha_i) + c_i \alpha_i$$

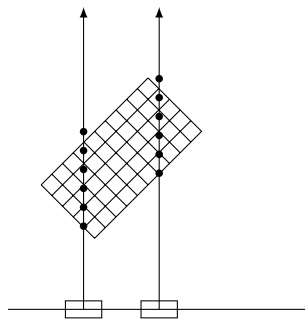
$\alpha_i$  = Deckkraft der Farbe  $c_i$

$1 - \alpha_i \hat{=}$  Transparenz



$$c^{\text{out}} = \sum_{i=1}^n c_i \alpha_i \prod_{j=i+1}^n (1 - \alpha_j)$$

## 11.2 Raycasting mittels front-to-back rendering



Auf der Sichtlinie werden in regelmäßigen Abständen im Körper Messpunkte gesetzt. Für jeden Messpunkt wird nun, wie vorher beschrieben, die Farbe und Leuchtkraft bestimmt. Anschließend werden diese zu einer Farbe für den entsprechenden Pixel auf dem Bildschirm zusammengefasst. Wir beginnen dabei mit dem entferntesten und enden mit dem Messpunkt am nächsten. Dadurch werden Voxel, welche hinter „soliden“ Voxeln liegen in der Farbwahl nicht betrachtet.

## 12 Einschub: Algebraische Flächen

$$f(x, y, z) = 0$$

$$\text{z.B. } \{(x, y, z) \mid x^2 + y^2 + z^2 - 1 = 0\}$$

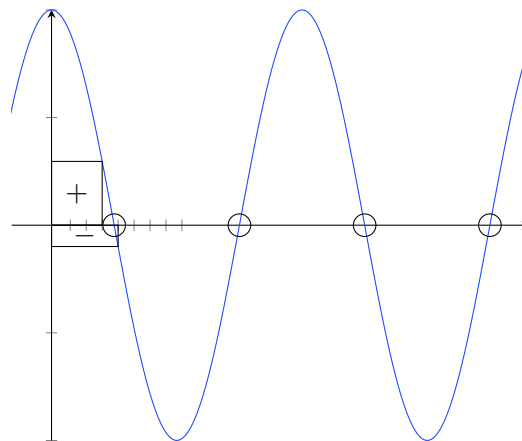
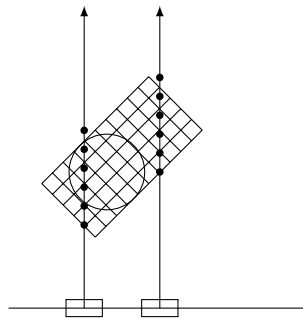
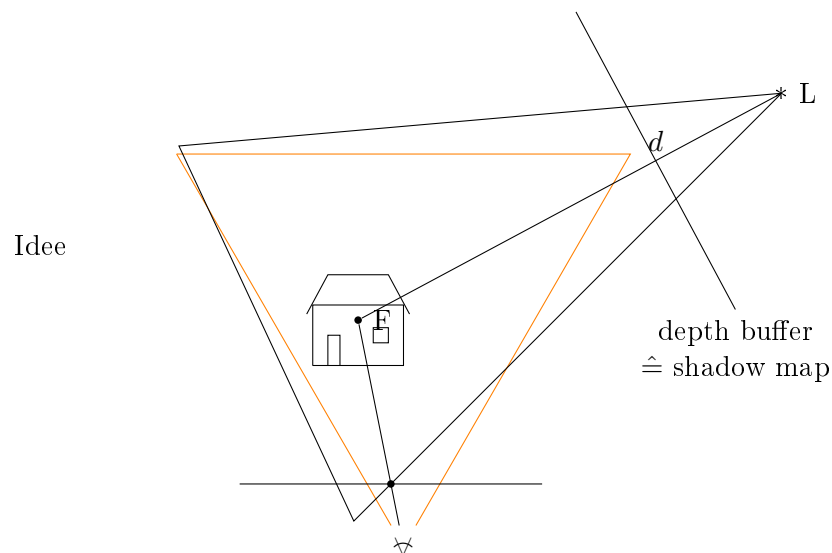


Abbildung 12.1: Finden von Nullstellen durch Suche nach Vorzeichenwechsel

Wie beim Raycasting wird der Bereich in Abschnitte unterteilt und diese einzeln untersucht, folgt auf ein + ein – (oder umgekehrt), liegt dazwischen eine Nullstelle.

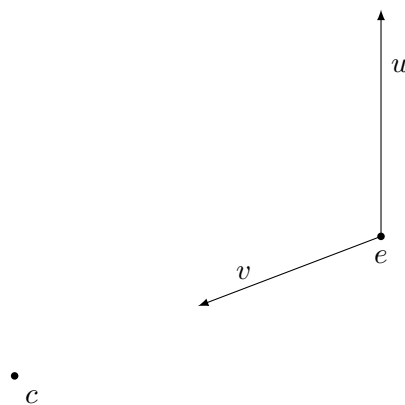
$$p \quad \nabla f(p) = n$$

# 13 Shadow-Mapping



1. Rendere die Szene aus der Sicht der Lichtquelle in eine Tiefenkarte (shadow map)
2. Rendere die Szene aus der Sicht der Kamera, wobei jedes Fragment  $F$  überprüft, ob sein Abstand zur Lichtquelle  $L$  größer ist als der Abstand des ersten Schnittpunktes des Strahles von  $L$  zu  $F$  mit der Szene. Diese Abstandsinformation findet sich in der Shadow map.

`lookat(eye, center, up)`



$$v = \frac{c - e}{|c - e|}, \quad r = \frac{v \times u}{|v \times u|}, \quad S = \frac{r \times v}{|r \times v|}$$

$$\text{Welt}_x = \text{Welt}_{M_{\text{Cam}}} \cdot \text{Cam}$$

$$\begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = \overset{||}{\left( \begin{array}{|c|} \hline r \\ \hline 0 \end{array}, \begin{array}{|c|} \hline s \\ \hline 0 \end{array}, \begin{array}{|c|} \hline v \\ \hline 0 \end{array}, \begin{array}{|c|} \hline e \\ \hline 1 \end{array} \right)}$$