

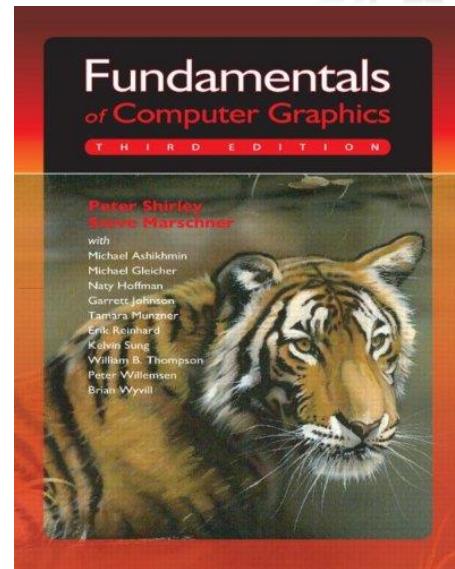
Computergrafik

Vorlesung im Wintersemester 2020/21
Kapitel 2: Raytracing

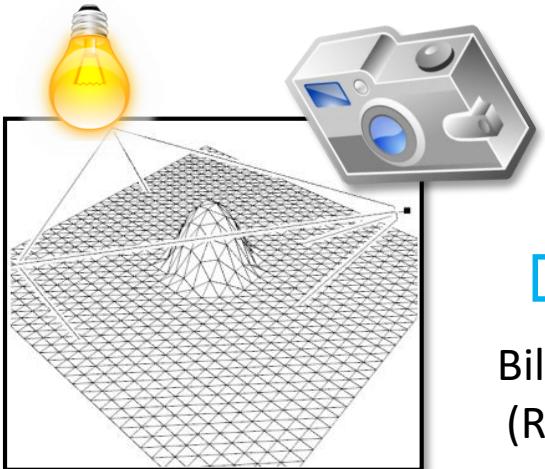
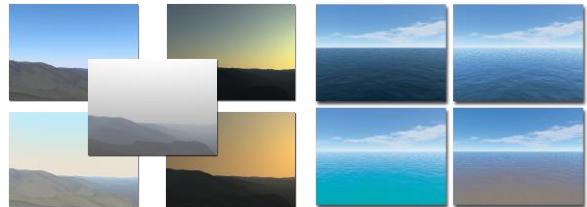
Prof. Dr.-Ing. Carsten Dachsbacher
Lehrstuhl für Computergrafik
Karlsruher Institut für Technologie



- **Fundamentals of Computer Graphics,**
P. Shirley, S. Marschner, 3rd Edition, AK Peters
→ Kapitel 4
- **Immersive Linear Algebra,**
J. Ström, K. Åström, T. Akenine-Möller,
<http://immersivemath.com/ila/index.html>
→ Kapitel 2-4



Computergrafik und Farbbilder



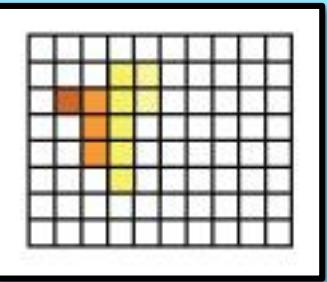
Bildsynthese
(Rendering)

Virtuelle Szene
(Geometrie, Material, Kamera, Lichtquellen, ...)



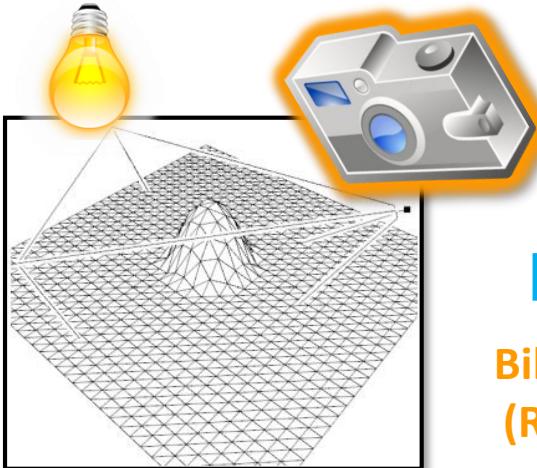
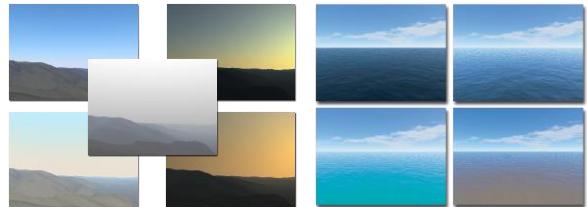
Monitor

Ansteuerung des
Monitors



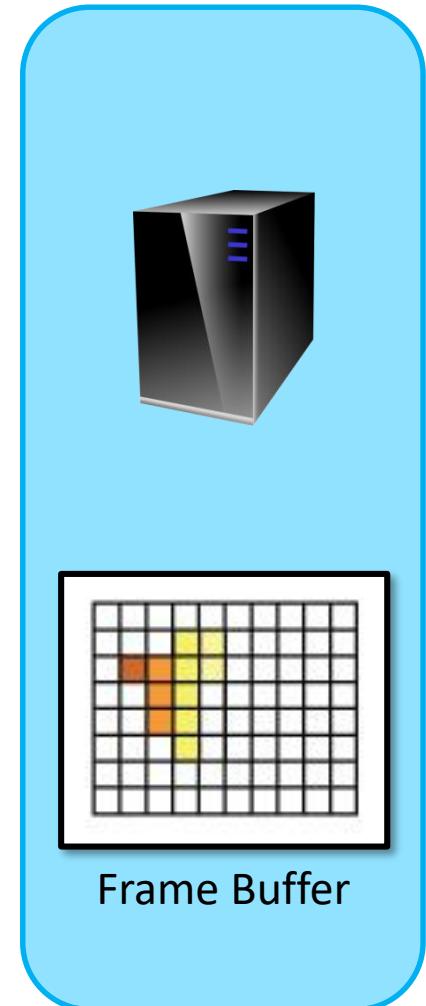
Frame Buffer

Computergrafik und Farbbilder



Bildsynthese
(Rendering)

Virtuelle Szene
(Geometrie, Material, Kamera, Lichtquellen, ...)

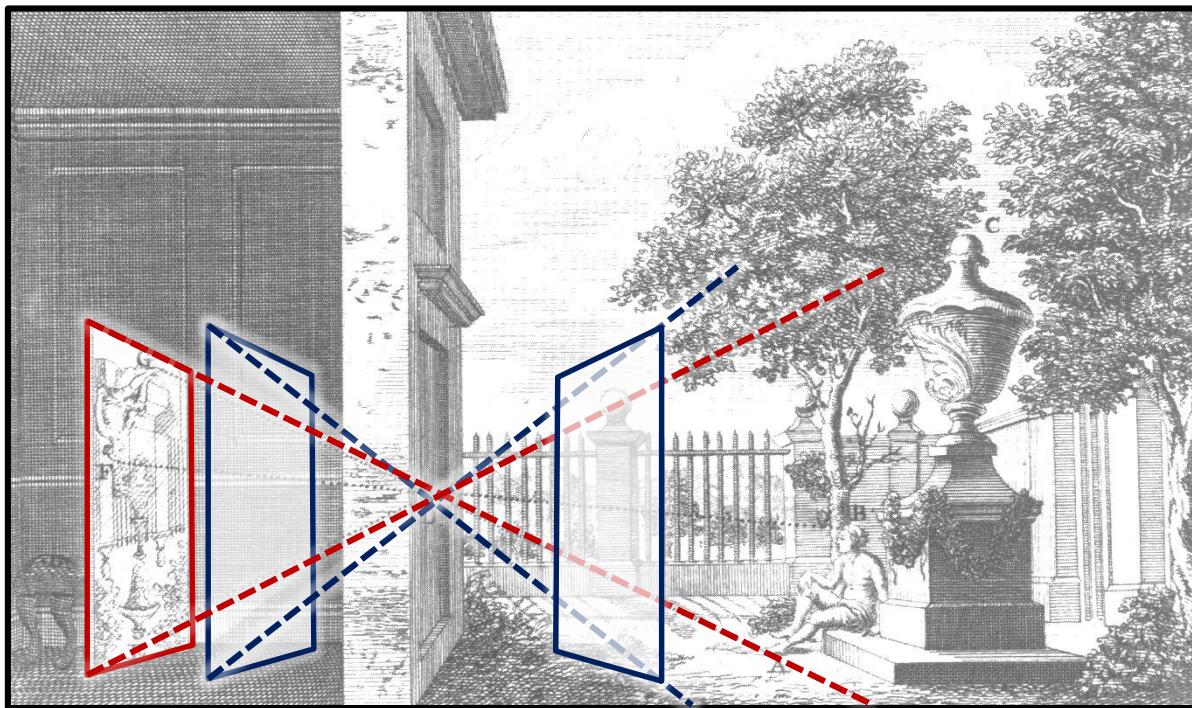


Monitor

Ansteuerung des
Monitors

Lochkamera

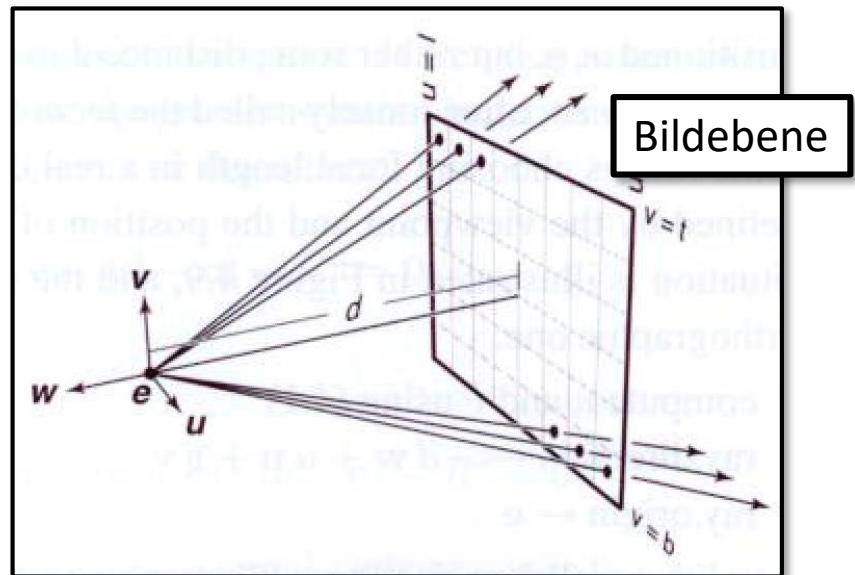
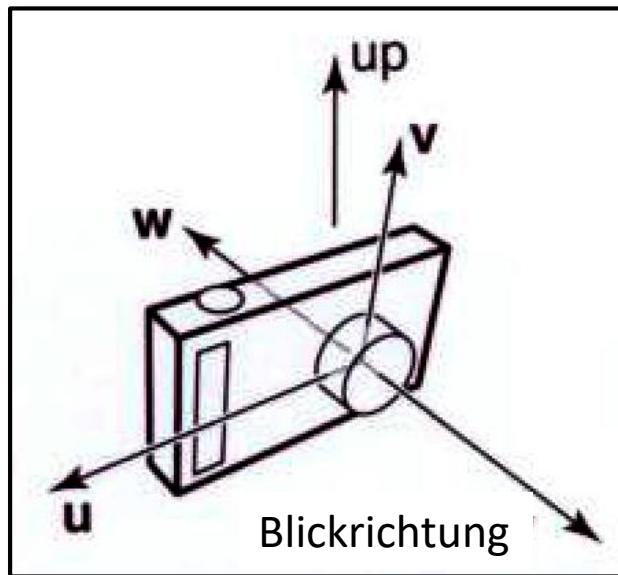
- ▶ in der CG verwendet man am einfachsten das Modell der Lochkamera
 - ▶ Kammer mit kleiner Öffnung → auf der gegenüberliegenden Wand entsteht ein spiegelverkehrtes Bild



- ▶ (im Prinzip) unbegrenzte Schärfentiefe
- ▶ Kamera definiert durch Position der Öffnung und Bildebene

Virtuelle Kamera

- ▶ in der CG verwendet man am einfachsten das Modell der Lochkamera
- ▶ eine virtuelle Kamera ist definiert durch
 - ▶ Position und Blickrichtung (oder Zielpunkt)
 - ▶ Orientierung der vertikalen Achse (hier **up**)
 - ▶ eine Bildebene mit Breite, Höhe und Abstand **vor** der Kamera
 - ▶ die Bildebene entspricht dem Bild der virtuellen Kamera



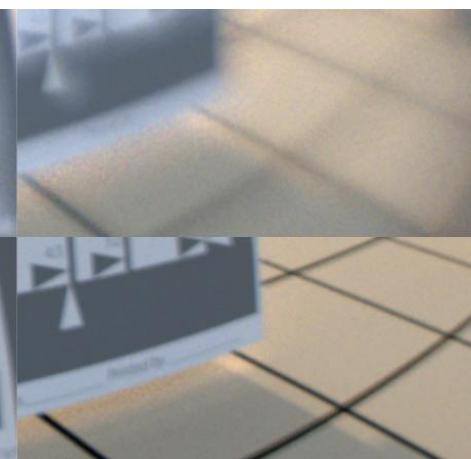
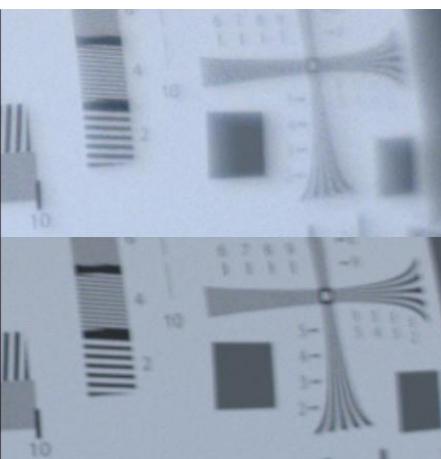
Bsp.: Abbildung durch Linsen (keine Lochkamera)



fisheye

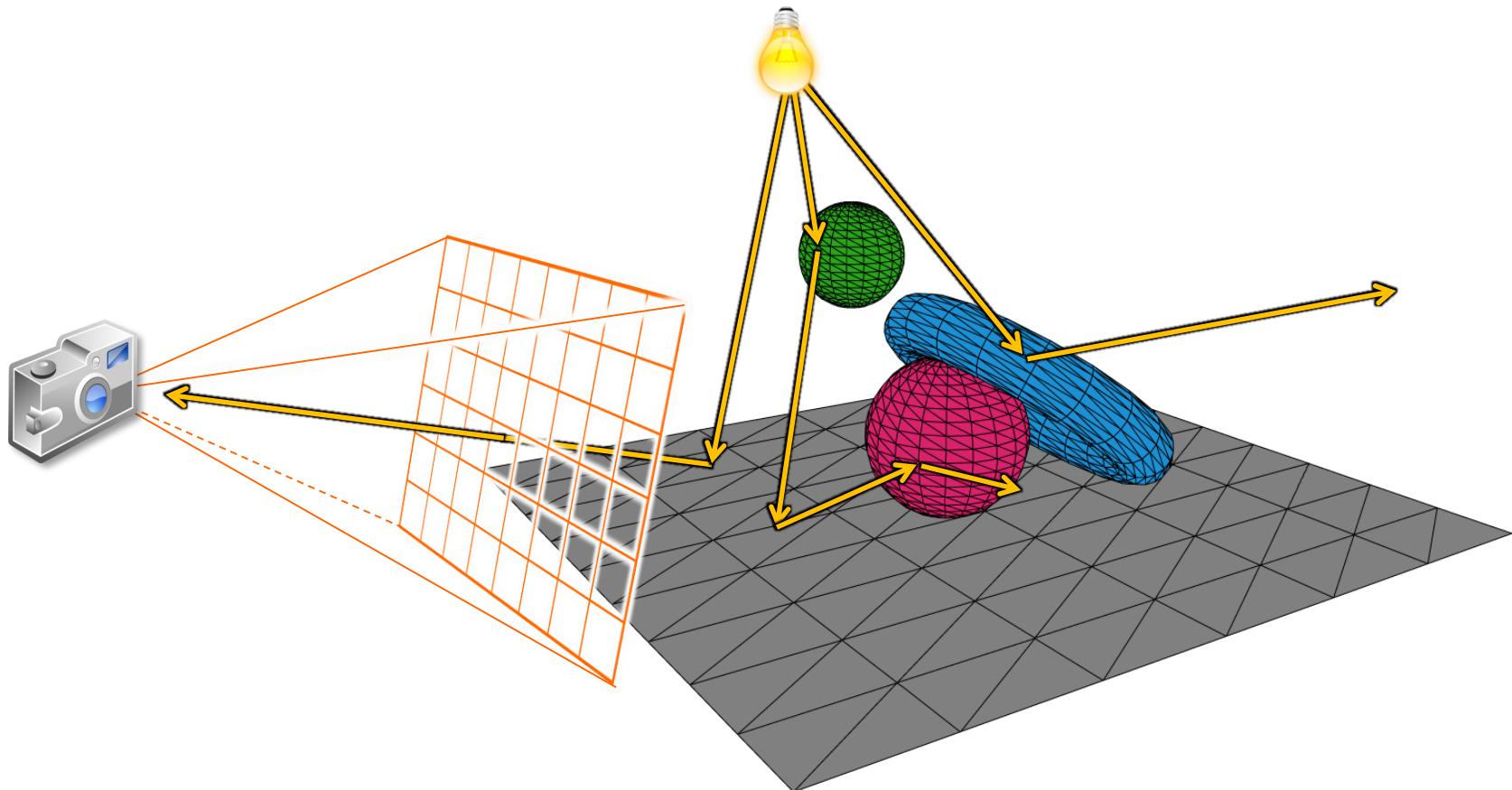


aspherical fisheye



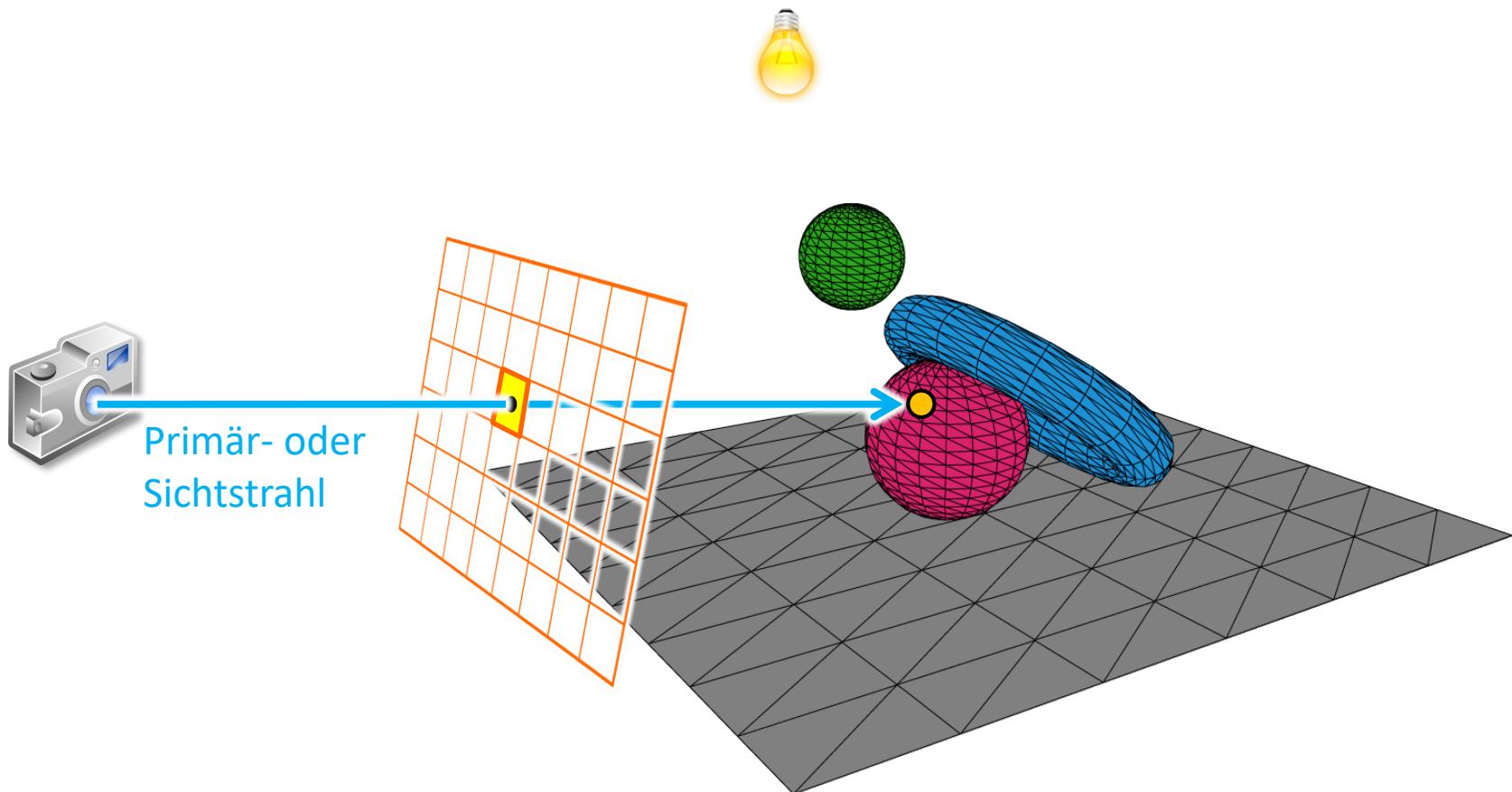
Grundidee

- ▶ Raytracing ist ein Verfahren zur Bildsynthese (engl. Rendering)
- ▶ geometrische Optik: Licht breitet sich entlang von Strahlen aus
- ▶ die Ausbreitung des Lichts wird – von der Kamera aus – zurückverfolgt



Grundidee

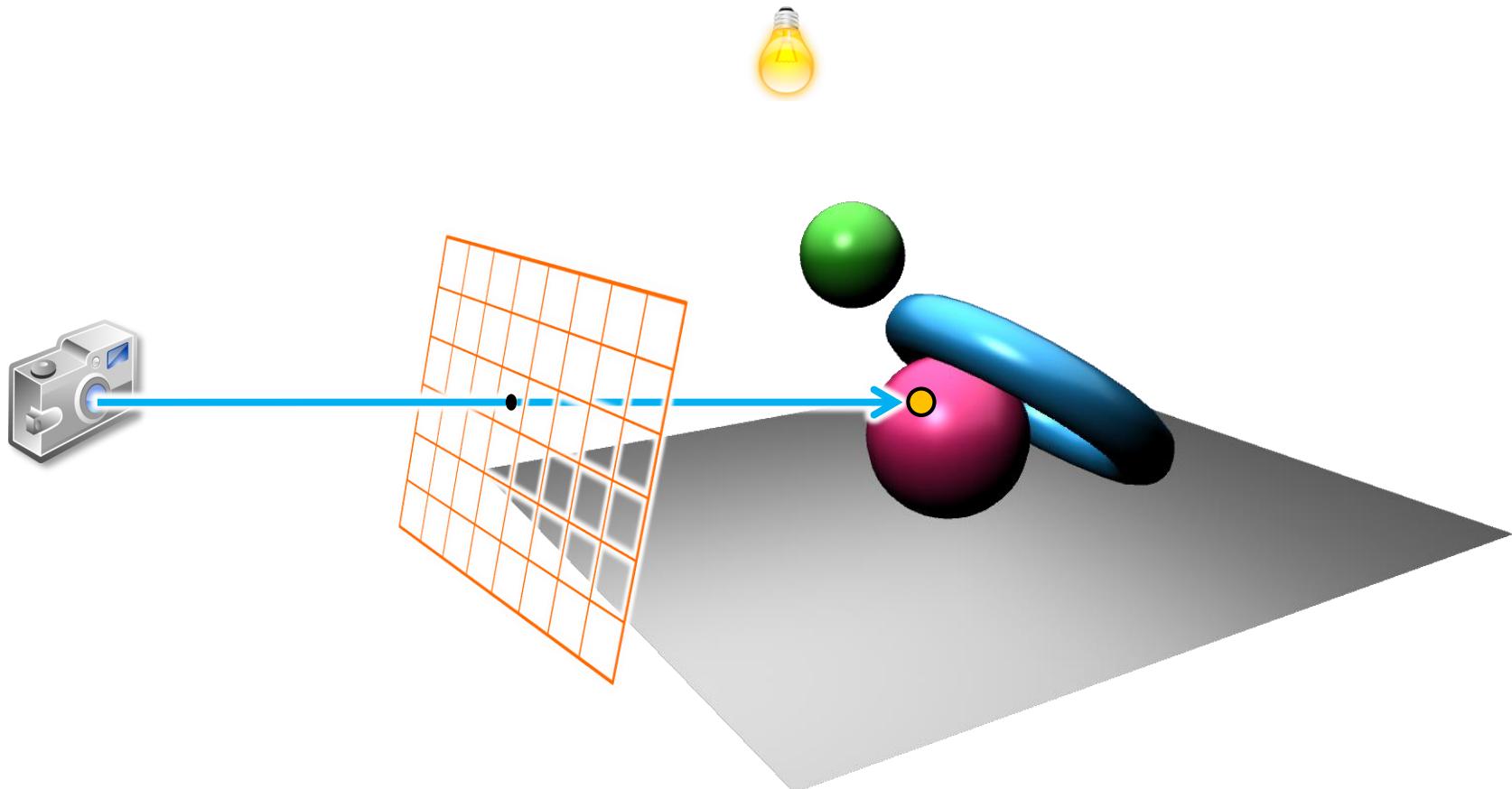
- betrachte jeden Pixel des Bildes: verfolge einen Strahl durch diesen Pixel und finde die nächste Fläche entlang des Strahls



Primär- oder
Sichtstrahl

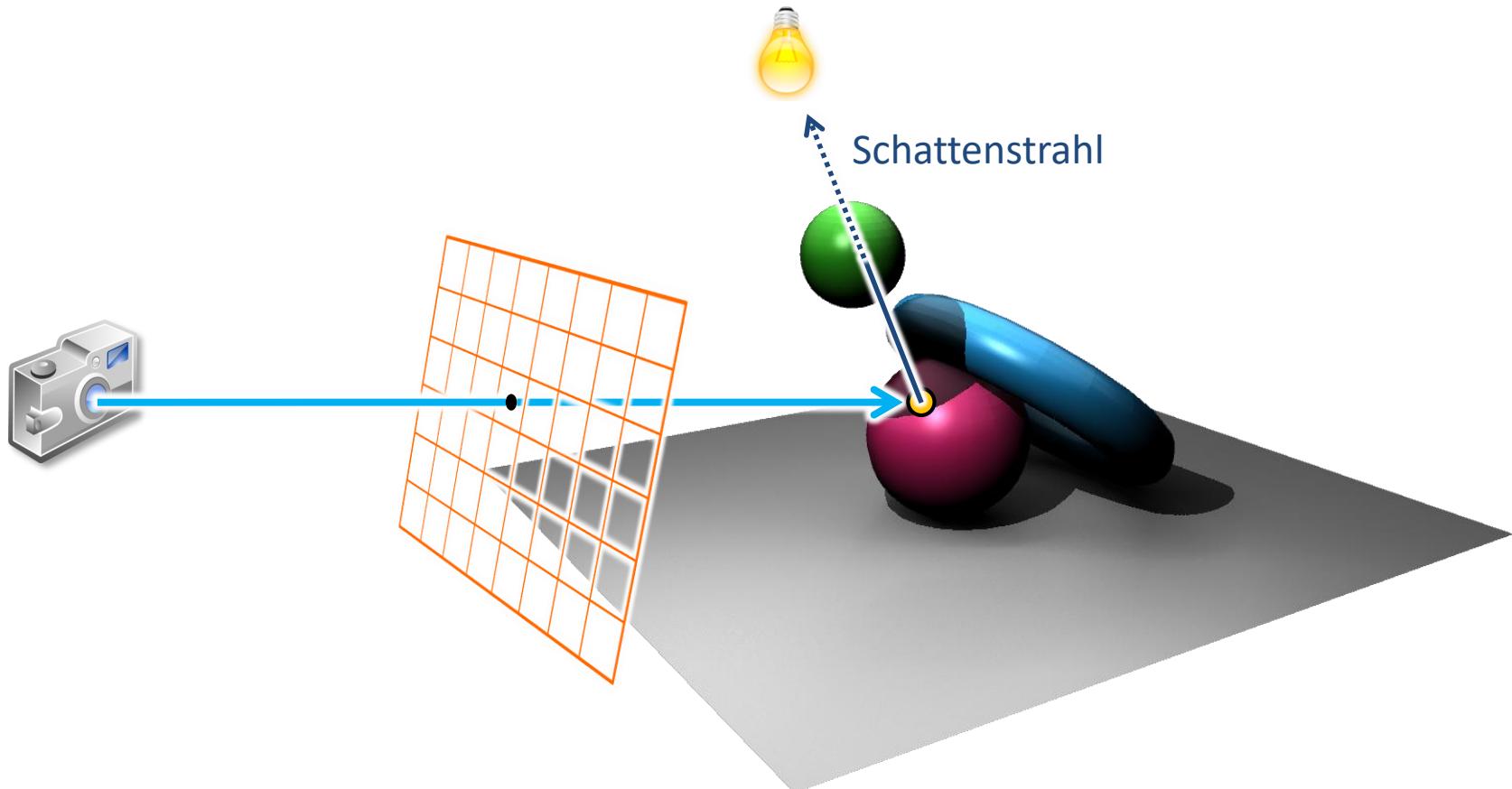
Grundidee

- betrachte jeden Pixel des Bildes: verfolge einen Strahl durch diesen Pixel und finde die nächste Fläche entlang des Strahls
- berechne die Schattierung der Fläche



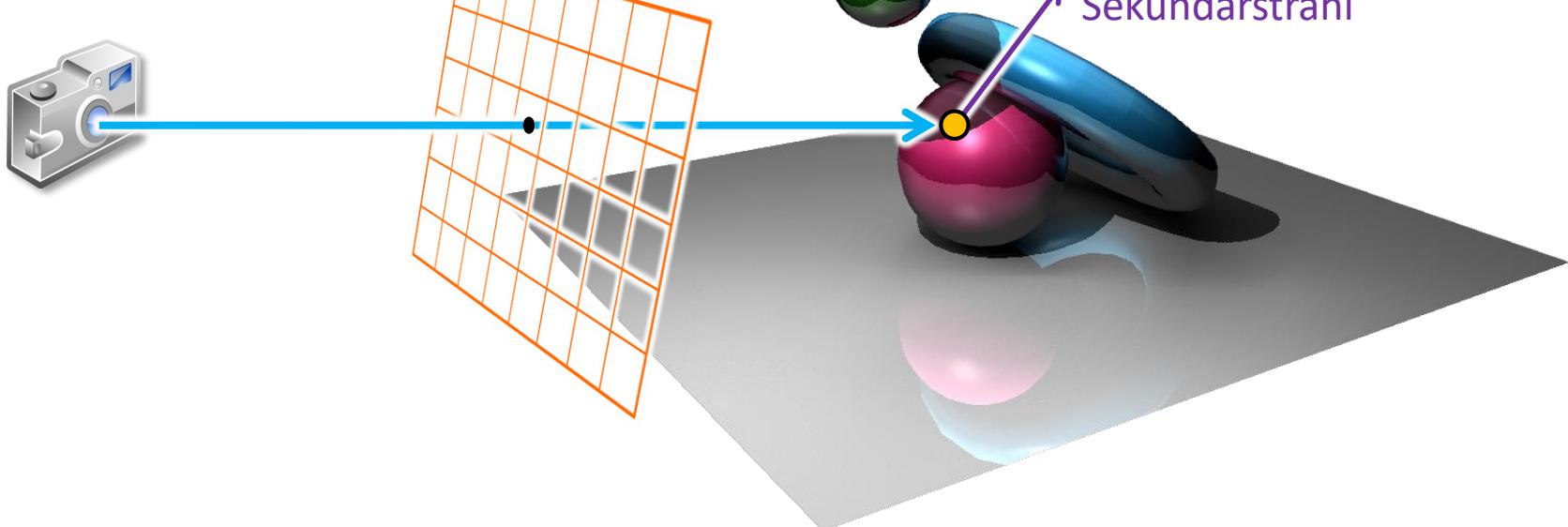
Grundidee

- betrachte jeden Pixel des Bildes: verfolge einen Strahl durch diesen Pixel und finde die nächste Fläche entlang des Strahls
- berechne die Schattierung der Fläche



Grundidee

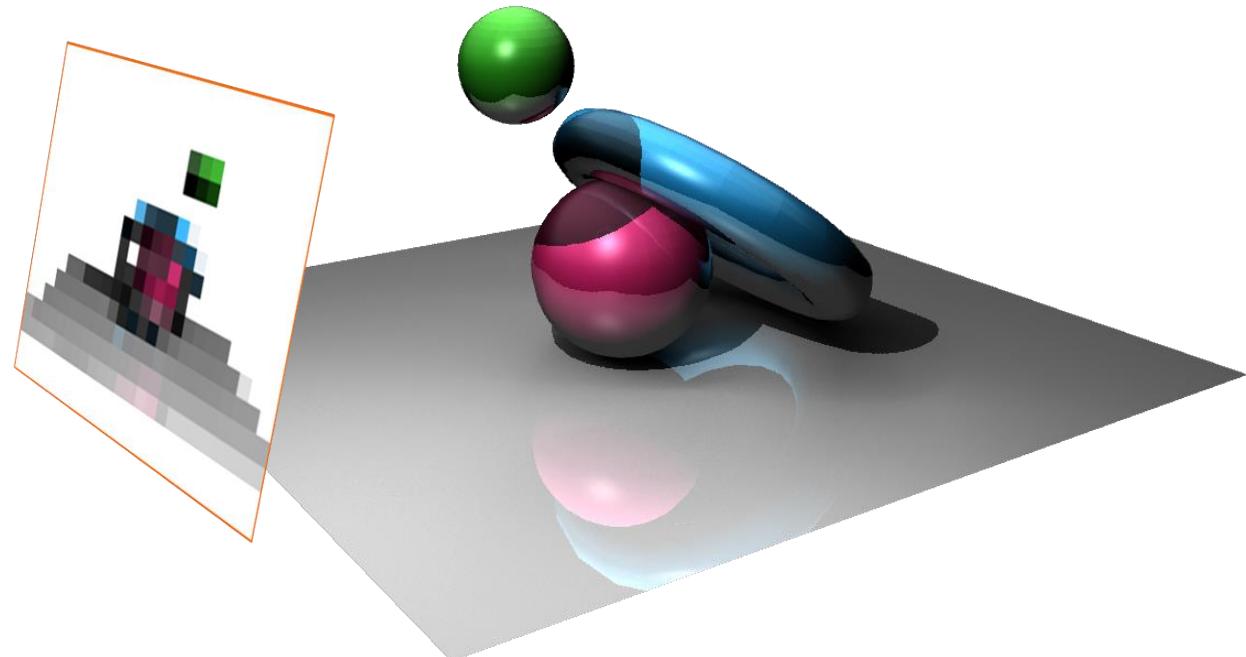
- betrachte jeden Pixel des Bildes: verfolge einen Strahl durch diesen Pixel und finde die nächste Fläche entlang des Strahls
- berechne die Schattierung der Fläche
- setze Strahlverfolgung fort, wenn Flächen spiegeln oder transmittieren



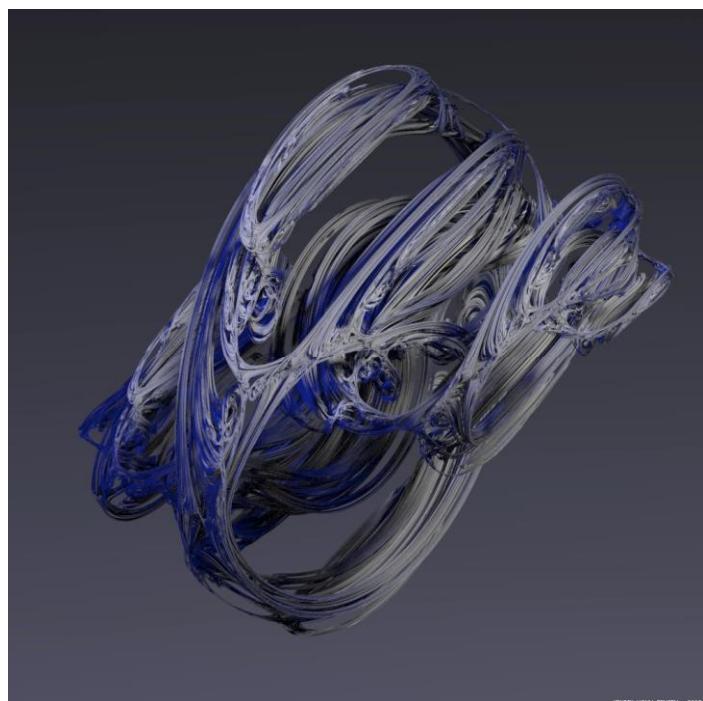
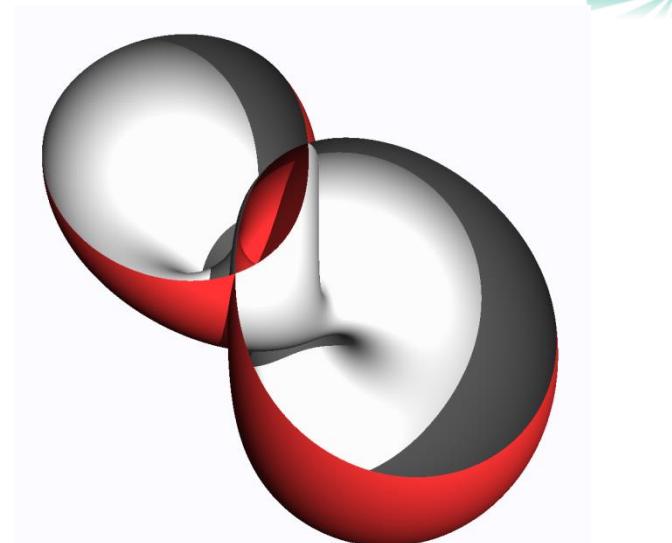
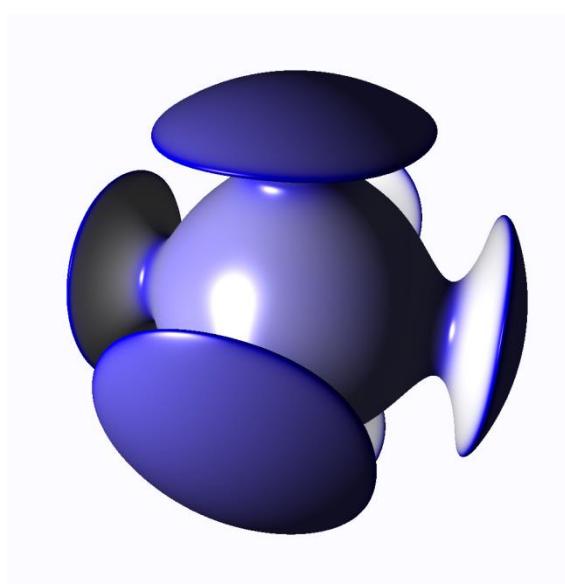
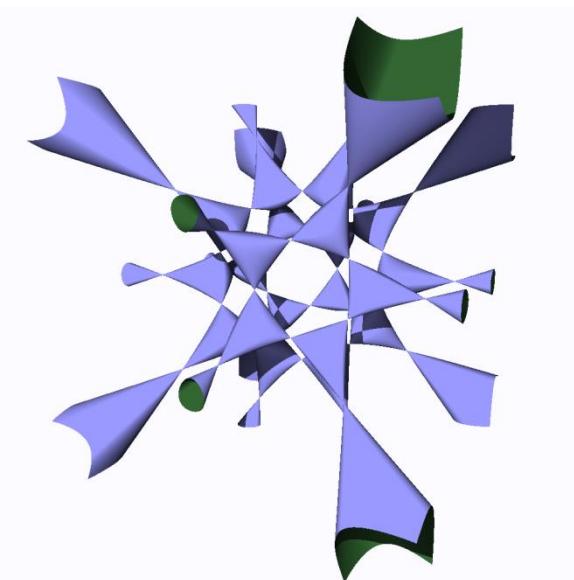
Raytracing

Grundidee

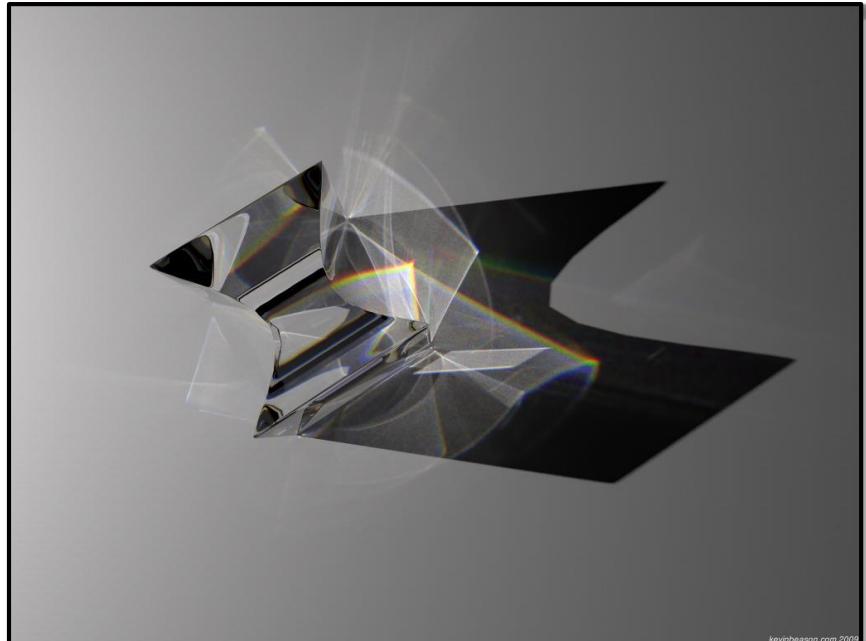
- betrachte jeden Pixel des Bildes: verfolge einen Strahl durch diesen Pixel und finde die nächste Fläche entlang des Strahls
- berechne die Schattierung der Fläche
- setze Strahlverfolgung fort, wenn Flächen spiegeln oder transmittieren



Raytracing Beispiele



Raytracing Beispiele



Raytracing Beispiele

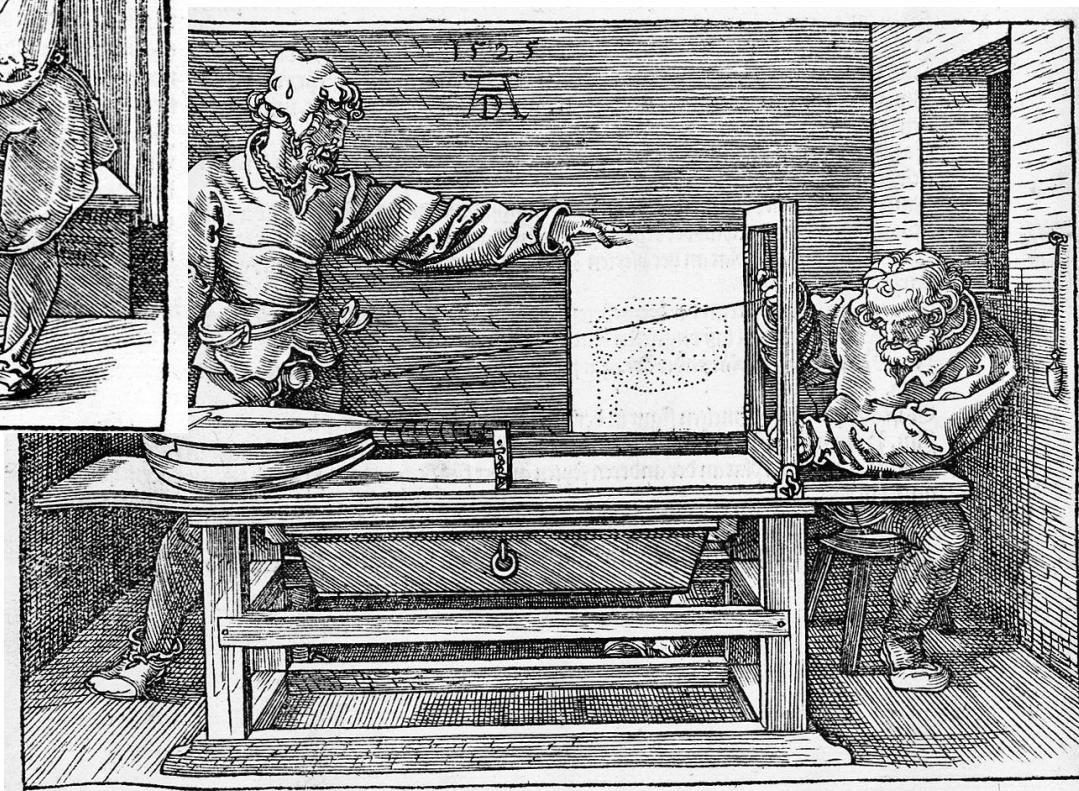


Raytracing?



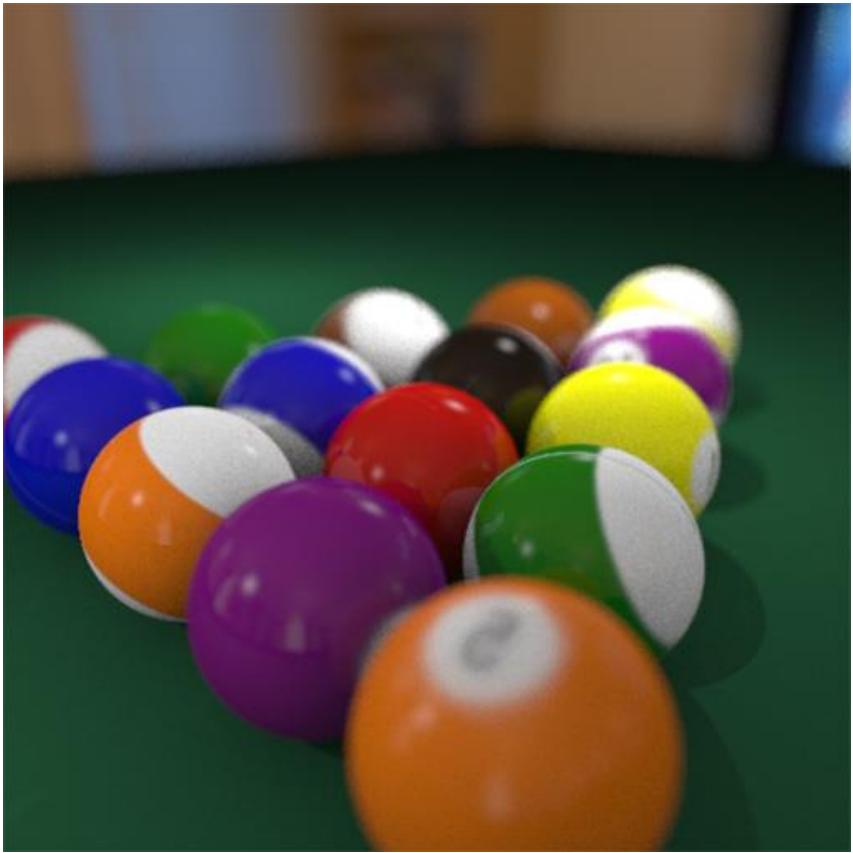
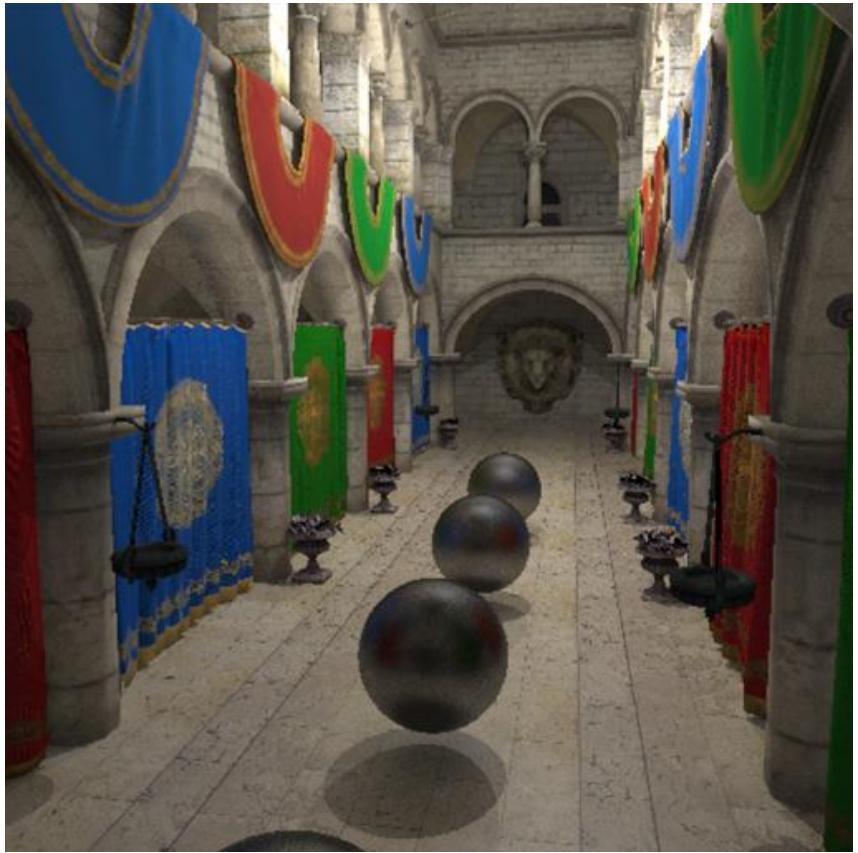
Bilder:

https://de.wikisource.org/wiki/Underweysung_der_Messung,_mit_dem_Zirckel_und_Richtscheyt,_in_Linien,_Ebenen_unnd_gantzen_corporon



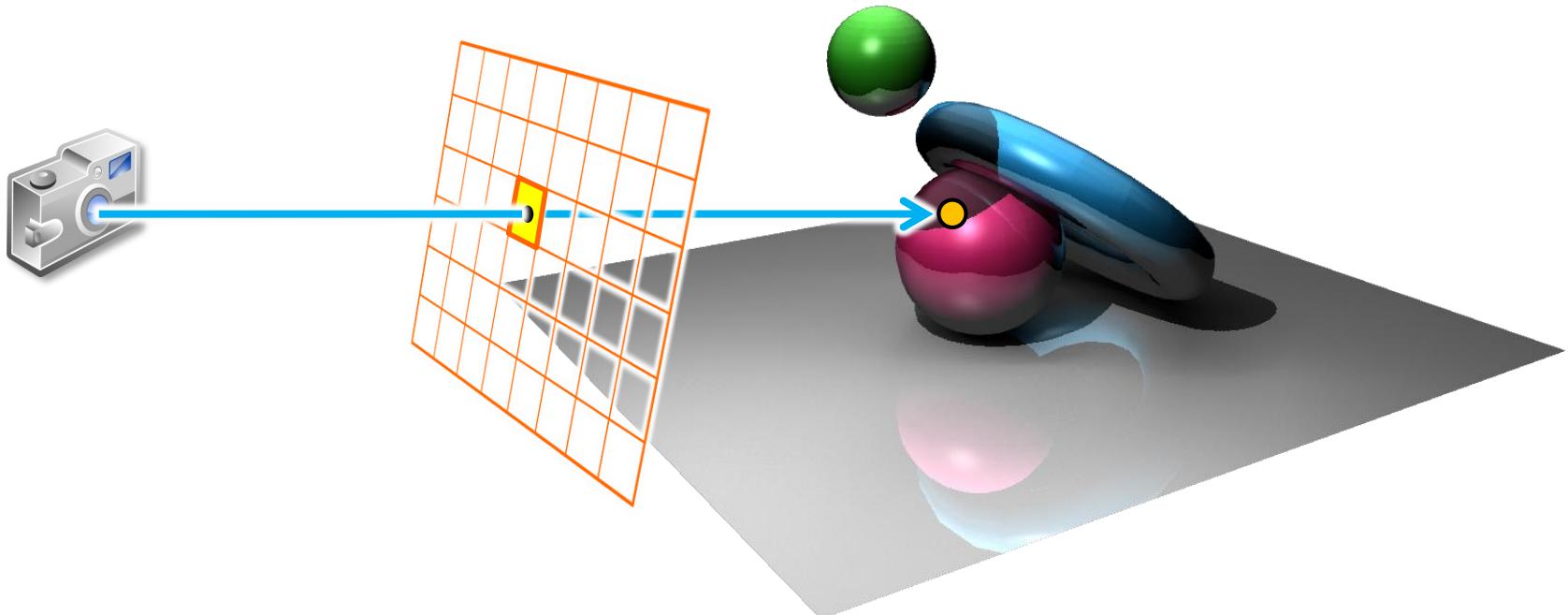
Albrecht Dürer: „Underweysung der messung mit dem zirckel und richtscheyt, in Linien ebnen unnd gantzen corporen“, 1525

Raytracing in der Übung



Schritte (die ein Raytracing-Programm implementieren muss)

- Erzeugung der Sichtstrahlen durch jeden Pixel (ray generation)
- Schnittberechnung (ray casting, ray intersection):
finde Dreieck, das den Sichtstrahl am nächsten zur Kamera schneidet
- Schattierung und Beleuchtungsberechnung (shading)
- Sekundärstrahlen für Spiegelung und Transmission



Analytische Geometrie

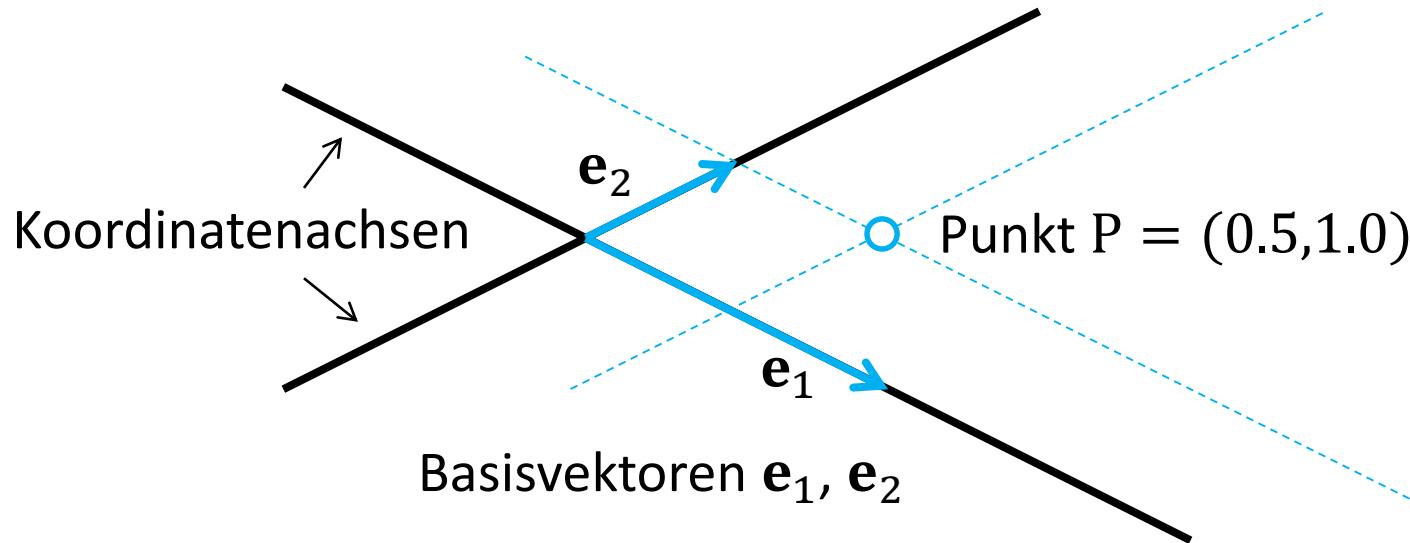
- die Geometrie ist ein Grundpfeiler der Computergrafik
 - Euklid (300 v.Chr.), René Descartes (*1596 +1650)
- ein kleines bisschen analytische Geometrie...
 - Koordinatensysteme, Vektoren und Matrizen
 - (Halb-)Geraden, Ebenen, ...
 - Ziel: Auffrischen der Kenntnisse, wichtige Konzepte für die Computergrafik
 - benötigen wir bspw. für Erzeugung Primärstrahlen, Schnittberechnung, Schattierung
- Konventionen



$\alpha, \beta, \gamma, \dots$	reelle Zahlen, Skalare	a, b, c, \dots	reelle Zahlen
i, j, k, \dots	ganze Zahlen	a, b, c, ...	Vektoren
P, Q, R, \dots	Punkte	P, Q, R, ...	Ortsvektoren
s, t, u, v, \dots	Parameterwerte		

Koordinatensysteme

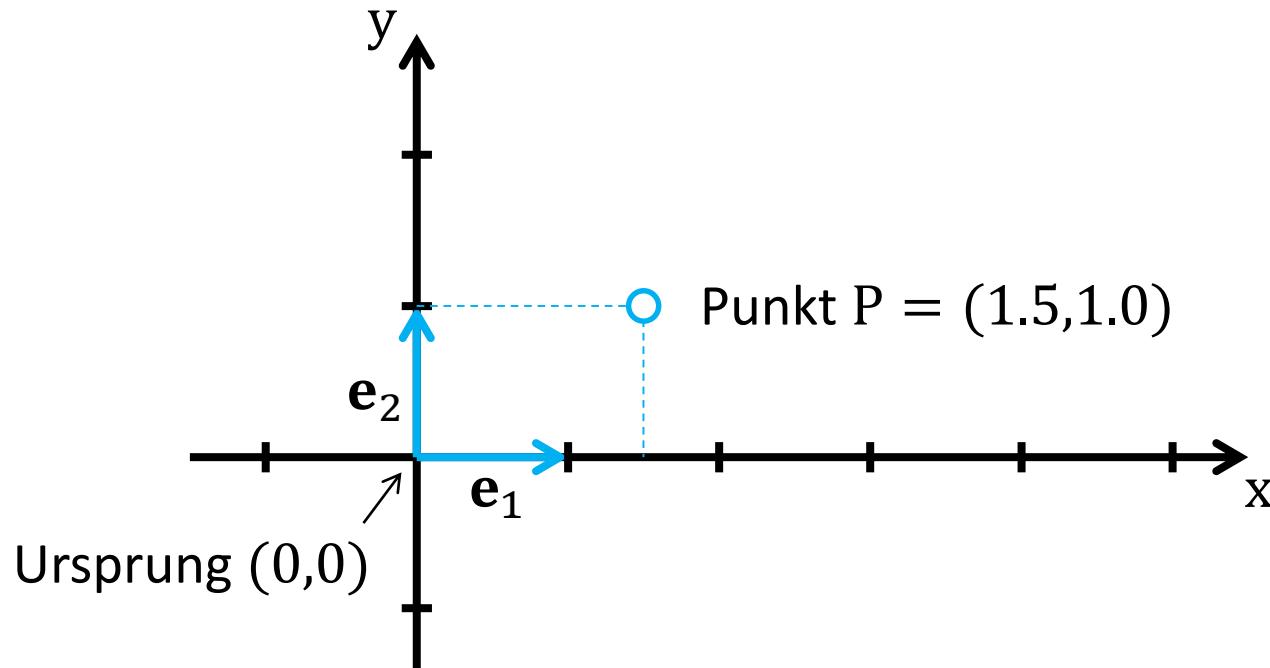
Koordinatensystem (geradlinig, schiefwinklig)



- ▶ nicht notwendigerweise rechtwinklig
- ▶ Basisvektoren nicht zwingend gleich lang
- ▶ Lage eines Punktes in der Ebene durch 2 Koordinaten eindeutig festgelegt
- ▶ Verallgemeinerung in höhere Dimensionen leicht möglich

Koordinatensysteme

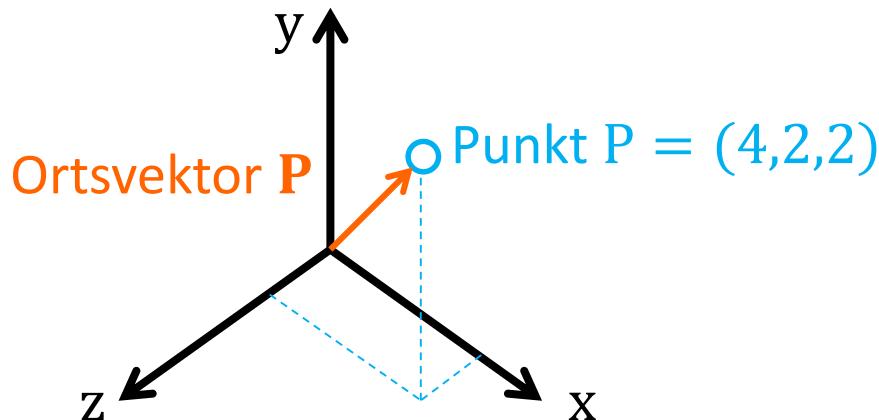
Kartesisches Koordinatensystem (nach René Descartes)



- Koordinatenachsen senkrecht
- Einheitsvektoren gleicher Länge (Einheit zur Längenmessung, z.B. m)
- Basisvektoren e_1, e_2, \dots bilden eine Orthonormalbasis
- wird nichts anderes gesagt, dann beziehen wir uns auf kartesische Koord.
und ein rechtshändiges System

Vektor, Punkt und Ortsvektor

- ein Vektor beschreibt anschaulich eine Länge und Richtung („Pfeil“)
- geben i.d.R. Richtungen oder Verschiebungen an



- ein Ortsvektor \mathbf{P} bezeichnet den Vektor \overrightarrow{OP}
 - von einem Bezugspunkt (typ. Koordinatenursprung O)
 - zu einem Aufpunkt mit den Punktkoordinaten $P = (p_1, p_2, \dots, p_n)$
 - p_i sind die Gewichtungsfaktoren der Linearkombination der aufspannenden Vektoren, d.h. $\mathbf{P} = p_1 \cdot \mathbf{e}_1 + p_2 \cdot \mathbf{e}_2 + p_3 \cdot \mathbf{e}_3$

Vektor, Punkt und Ortsvektor

- beachte den semantischen Unterschied
 - wir addieren zwei (Richtungs-)Vektoren,
z.B. Hintereinanderausführung von Verschiebungen
 - wir addieren keine Ortsvektoren
es sei denn, als Zwischenoperation, z.B. Berechnung eines
Mittelpunkts oder des Differenzvektors

- Rechenoperationen
 - Länge eines Vektors: $|\mathbf{a}| = \sqrt{\sum_{i=1}^n a_i^2}$
 - Einheitsvektor gdw. $|\mathbf{a}| = 1$
 - Vektoraddition: $\mathbf{a} \pm \mathbf{b} = (a_1 \pm b_1, a_2 \pm b_2, \dots, a_n \pm b_n)$
 - Skalierung: $\lambda \mathbf{a} = (\lambda a_1, \lambda a_2, \dots, \lambda a_n)$

Vektoren

Skalarprodukt (inneres Produkt, engl. dot product)

► Definition

► $\mathbf{a} \cdot \mathbf{b} = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$

► als Matrixmultiplikation

► $\langle \mathbf{a}, \mathbf{b} \rangle = \mathbf{a}^T \mathbf{b} = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$

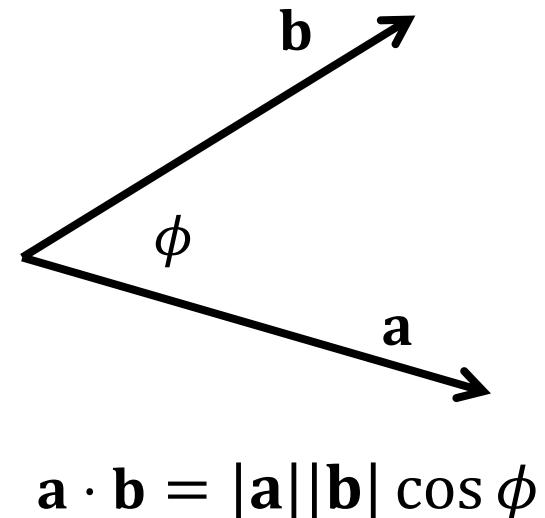
► Eigenschaften

► $\mathbf{a} \cdot \mathbf{b} = |\mathbf{a}| |\mathbf{b}| \cos \phi$

► $\mathbf{a} \cdot \mathbf{a} = |\mathbf{a}|^2$

► $\mathbf{a} \cdot \mathbf{b} = \mathbf{b} \cdot \mathbf{a}$ (kommutativ)

► $\mathbf{a} \cdot (\mathbf{b} \pm \mathbf{c}) = \mathbf{a} \cdot \mathbf{b} \pm \mathbf{a} \cdot \mathbf{c}$ (distributiv)



Vektoren

Kreuzprodukt (auch äußeres/vektorielles Produkt, engl. cross product)

- definiert in 3D-Vektorräumen

► $\mathbf{a} \times \mathbf{b} = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \times \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} a_2 b_3 - a_3 b_2 \\ a_3 b_1 - a_1 b_3 \\ a_1 b_2 - a_2 b_1 \end{pmatrix} = \mathbf{n} |\mathbf{a}| |\mathbf{b}| \sin \phi$

- $|\mathbf{n}| = 1$ und \mathbf{n} steht senkrecht auf \mathbf{a} und \mathbf{b}

- Graßmann, Lagrange, ... Identitäten

► $\mathbf{a} \cdot (\mathbf{a} \times \mathbf{b}) = \mathbf{b} \cdot (\mathbf{a} \times \mathbf{b}) = 0$

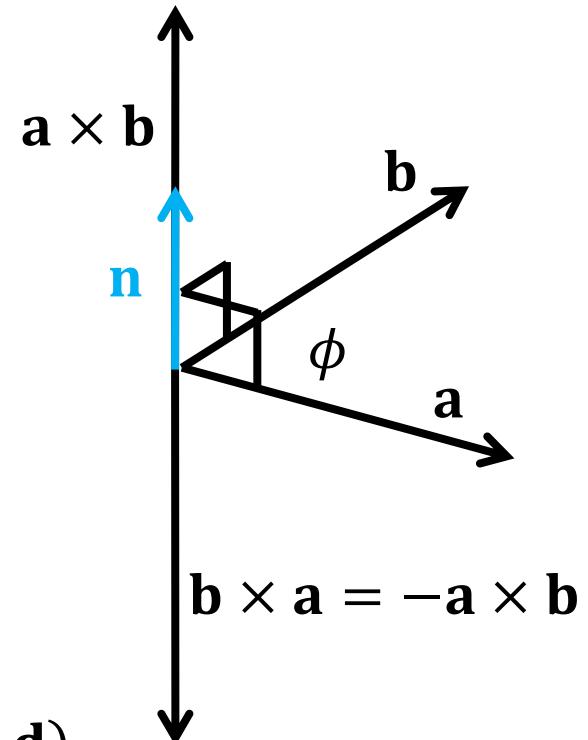
► $\mathbf{a} \cdot (\mathbf{b} \times \mathbf{c}) = (\mathbf{a} \times \mathbf{b}) \cdot \mathbf{c}$

► $\mathbf{a} \times (\mathbf{b} + \mathbf{c}) = \mathbf{a} \times \mathbf{b} + \mathbf{a} \times \mathbf{c}$

► $\mathbf{a} \times (k\mathbf{b}) = k(\mathbf{a} \times \mathbf{b})$

► $\mathbf{a} \times (\mathbf{b} \times \mathbf{c}) = (\mathbf{a} \cdot \mathbf{c}) \cdot \mathbf{b} - (\mathbf{a} \cdot \mathbf{b}) \cdot \mathbf{c}$

► $(\mathbf{a} \times \mathbf{b}) \cdot (\mathbf{c} \times \mathbf{d}) = (\mathbf{a} \cdot \mathbf{c})(\mathbf{b} \cdot \mathbf{d}) - (\mathbf{b} \cdot \mathbf{c})(\mathbf{a} \cdot \mathbf{d})$



Vektoren

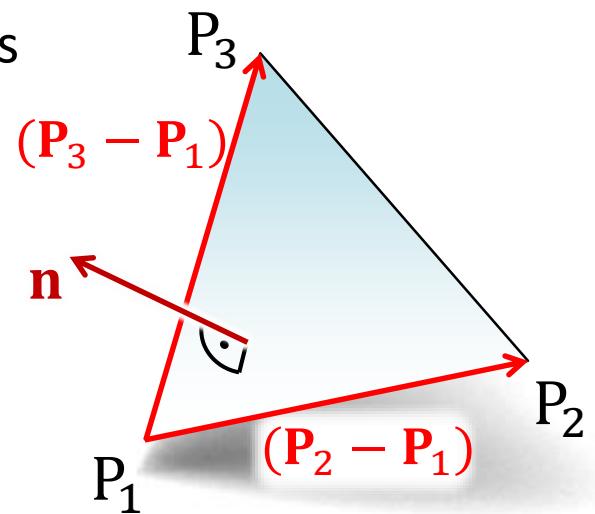
Kreuzprodukt (auch äußeres/vektorielles Produkt, engl. cross product)

- definiert in 3D-Vektorräumen

- $\mathbf{a} \times \mathbf{b} = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \times \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} a_2 b_3 - a_3 b_2 \\ a_3 b_1 - a_1 b_3 \\ a_1 b_2 - a_2 b_1 \end{pmatrix} = \mathbf{n} |\mathbf{a}| |\mathbf{b}| \sin \phi$
- $|\mathbf{n}| = 1$ und \mathbf{n} steht senkrecht auf \mathbf{a} und \mathbf{b}

- Anwendungsbeispiel: Berechnung der Oberflächennormale und Fläche eines Dreiecks

- gegeben: $\Delta(P_1, P_2, P_3)$
- $\mathbf{n}' = (\mathbf{P}_2 - \mathbf{P}_1) \times (\mathbf{P}_3 - \mathbf{P}_1)$
- $\mathbf{n} = \mathbf{n}' / |\mathbf{n}'|$
- $A_{\Delta} = \frac{1}{2} |\mathbf{n}'| \quad \leftarrow$ das Vorzeichen wird gleich interessant!

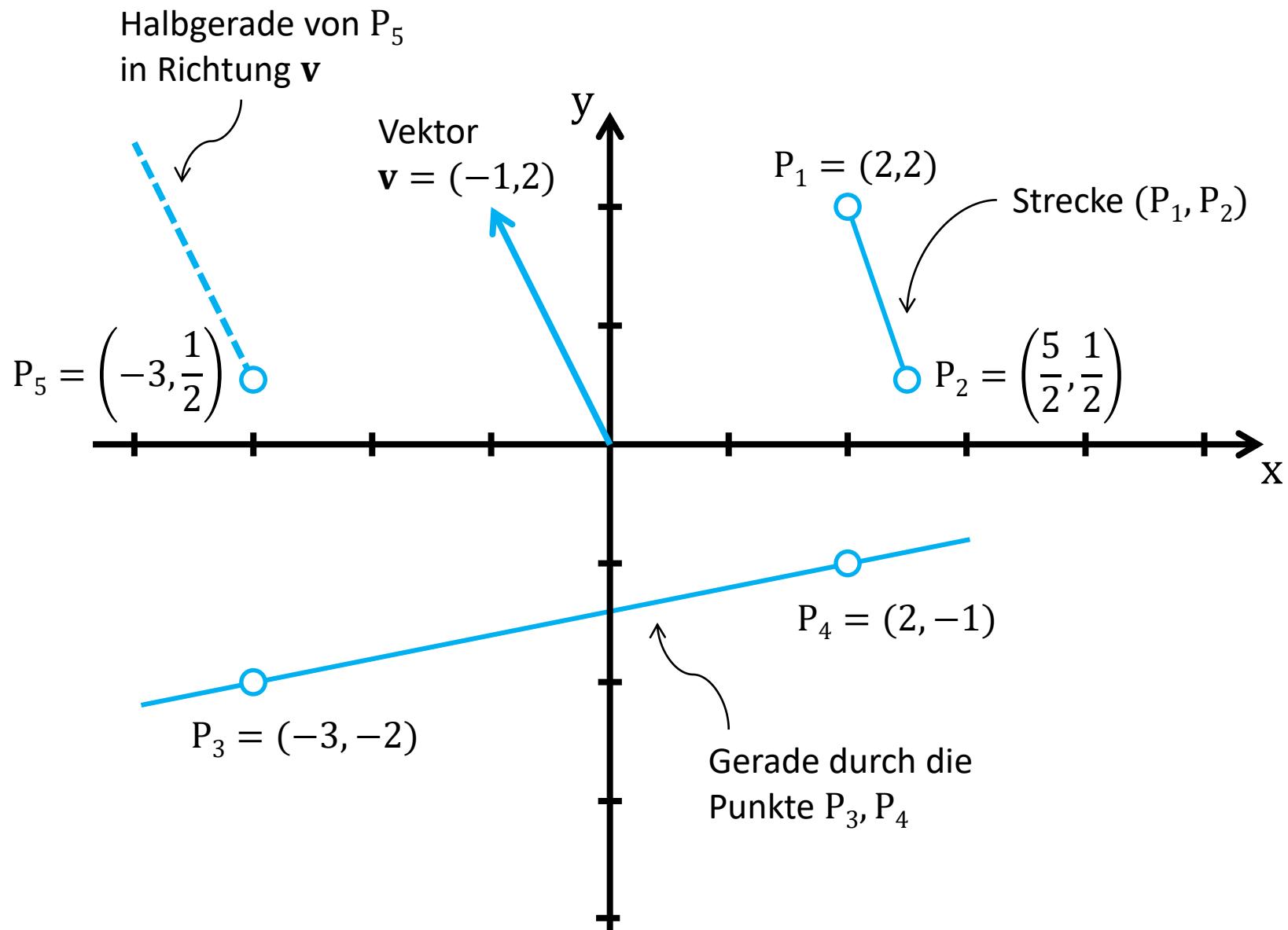


Parameterdarstellungen

- eine Gerade ist eindeutig bestimmt durch zwei Punkte P und Q ($P \neq Q$)
 - als Richtung der Gerade bezeichnet man den Vektor von P nach Q
 - Gerade
$$g(t) = P + t \cdot (Q - P), t \in \mathbb{R}$$
 - Halbgerade
$$g(t) = P + t \cdot (Q - P), t \in \mathbb{R}_0^+ \text{ (also } t \geq 0\text{)}$$
 - Strecke
$$g(t) = P + t \cdot (Q - P), t \in \mathbb{R}, 0 \leq t \leq 1$$

- eine Ebene ist eindeutig bestimmt durch drei Punkte P, Q, R , die nicht auf einer Geraden liegen (nicht kollinear)
 - die aufspannenden Vektoren der Ebene sind $(Q - P)$ und $(R - P)$
 - Ebene
$$g(s, t) = P + s \cdot (Q - P) + t \cdot (R - P), s, t \in \mathbb{R}$$
 - Parallelogramm
$$g(s, t) = P + s \cdot (Q - P) + t \cdot (R - P), s, t \in \mathbb{R}_0^+ \wedge s, t \in [0; 1]$$
 - Dreieck
$$g(s, t) = P + s \cdot (Q - P) + t \cdot (R - P), s, t \in \mathbb{R}_0^+ \wedge s + t \leq 1$$

Geraden, Strecken, ...



Baryzentrische Koordinaten

► Definition:

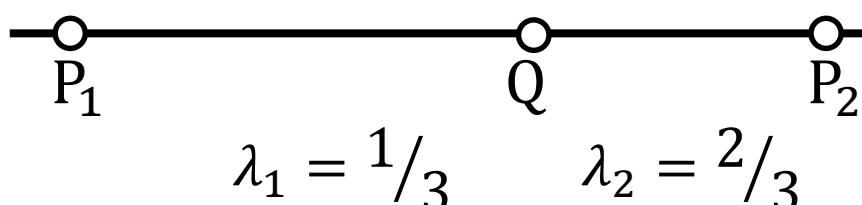
- gegeben k Punkte $P_1, \dots, P_k \in \mathbb{R}^n$ mit $k \leq n + 1$
- wenn ein Punkt Q in der Form

$$Q = \lambda_1 P_1 + \lambda_2 P_2 + \cdots + \lambda_k P_k$$

mit $\lambda_1 + \lambda_2 + \cdots + \lambda_k = 1$ (Affinkombination) dargestellt werden kann, so bezeichnet man $(\lambda_1, \lambda_2, \dots, \lambda_k)$ als die **baryzentrischen Koordinaten** von Q bzgl. der Basispunkte P_1, \dots, P_k

► Beispiel $k = 2, n = 2$

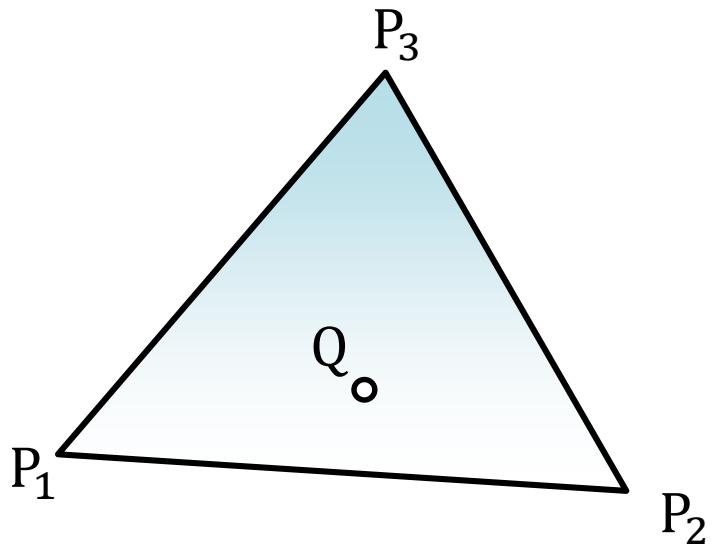
- alle Punkte $Q = \lambda_1 P_1 + \lambda_2 P_2$ mit $\lambda_1 + \lambda_2 = 1$ liegen auf der Geraden durch P_1 und P_2
- $Q = \lambda_1 P_1 + \lambda_2 P_2 = (1 - \lambda_2)P_1 + \lambda_2 P_2 = P_1 + \lambda_2(P_2 - P_1)$, wegen $\lambda_1 = 1 - \lambda_2$



...wann liegt Q auf der Strecke (P_1, P_2) ?

Baryzentrische Koordinaten

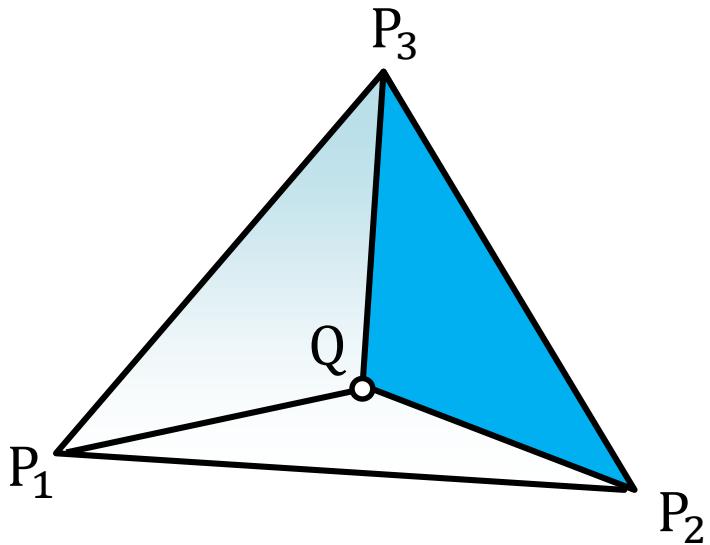
- Beispiel $k = 3, n = 2$ (Dreieck im \mathbb{R}^2)
 - alle Punkte Q mit $Q = \lambda_1 P_1 + \lambda_2 P_2 + \lambda_3 P_3$ und $\lambda_1 + \lambda_2 + \lambda_3 = 1$
liegen innerhalb des Dreiecks $\Delta(P_1, P_2, P_3)$ gdw. $\lambda_1, \lambda_2, \lambda_3$ positiv sind



- wie berechnet man $\lambda_1, \lambda_2, \lambda_3$?
 - es gilt natürlich wieder: $\lambda_1 = 1 - \lambda_2 - \lambda_3$
 - gegeben $Q, P_1, P_2, P_3 \rightarrow$ lineares Gleichungssystem mit
zwei Gleichungen ($n = 2$) und zwei Unbekannten λ_2, λ_3

Baryzentrische Koordinaten

- Beispiel $k = 3, n = 2$ (Dreieck im \mathbb{R}^2)
- alle Punkte Q mit $Q = \lambda_1 P_1 + \lambda_2 P_2 + \lambda_3 P_3$ und $\lambda_1 + \lambda_2 + \lambda_3 = 1$
liegen innerhalb des Dreiecks $\Delta(P_1, P_2, P_3)$ gdw. $\lambda_1, \lambda_2, \lambda_3$ positiv sind



- geometrische Interpretation der baryzentrischen Koordinaten

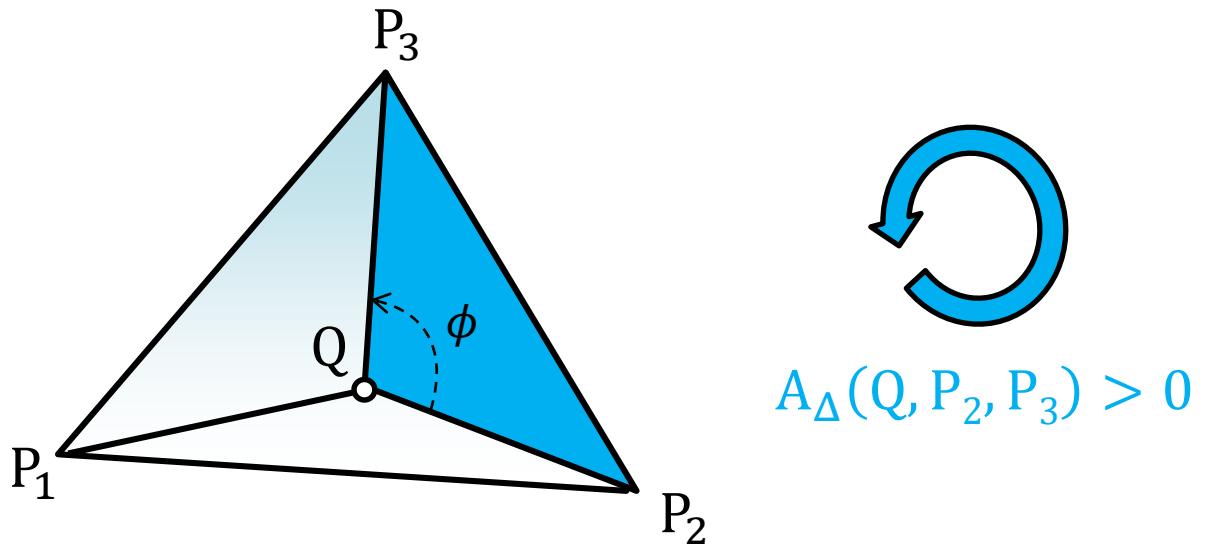
$$\lambda_1 = \frac{A_{\Delta}(Q, P_2, P_3)}{A_{\Delta}(P_1, P_2, P_3)} \quad \lambda_2 = \frac{A_{\Delta}(P_1, Q, P_3)}{A_{\Delta}(P_1, P_2, P_3)} \quad \lambda_3 = \frac{A_{\Delta}(P_1, P_2, Q)}{A_{\Delta}(P_1, P_2, P_3)}$$

- offensichtlich: $\lambda_1 + \lambda_2 + \lambda_3 = 1$



Baryzentrische Koordinaten

- Beispiel $k = 3, n = 2$ (Dreieck im \mathbb{R}^2)
- alle Punkte Q mit $Q = \lambda_1 P_1 + \lambda_2 P_2 + \lambda_3 P_3$ und $\lambda_1 + \lambda_2 + \lambda_3 = 1$
liegen innerhalb des Dreiecks $\Delta(P_1, P_2, P_3)$ gdw. $\lambda_1, \lambda_2, \lambda_3$ positiv sind



- geometrische Interpretation der baryzentrischen Koordinaten

$$\lambda_1 = \frac{A_{\Delta}(Q, P_2, P_3)}{A_{\Delta}(P_1, P_2, P_3)} \quad \lambda_2 = \frac{A_{\Delta}(P_1, Q, P_3)}{A_{\Delta}(P_1, P_2, P_3)} \quad \lambda_3 = \frac{A_{\Delta}(P_1, P_2, Q)}{A_{\Delta}(P_1, P_2, P_3)}$$

- $A_{\Delta}(Q, P_2, P_3) = \frac{1}{2} |P_2 - Q| |P_3 - Q| \sin \phi$

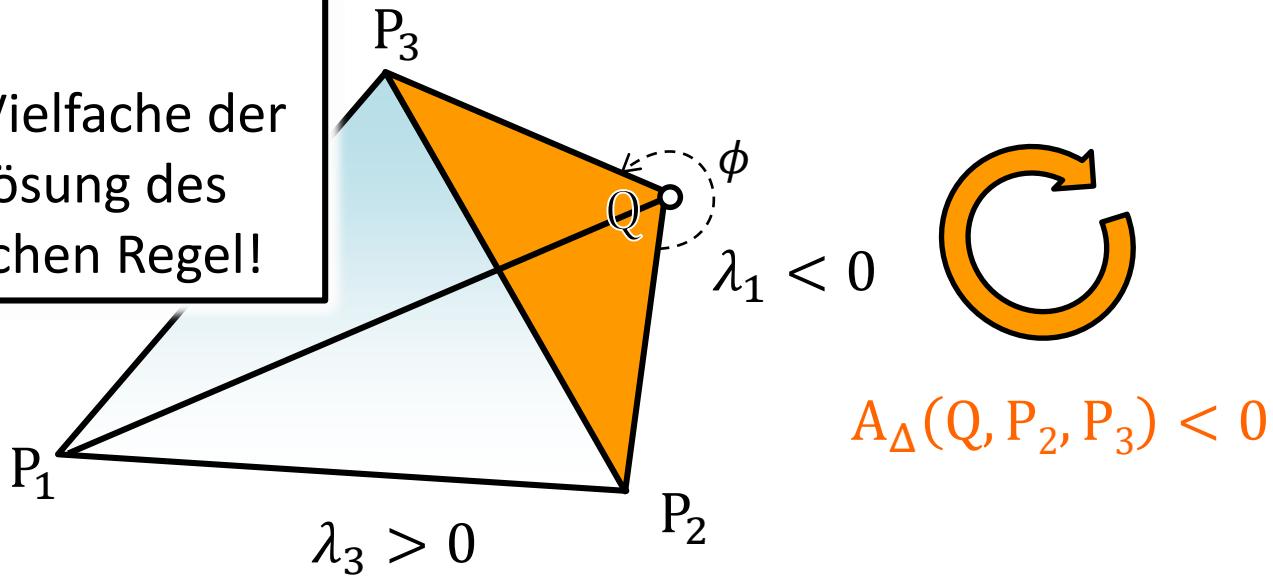


Baryzentrische Koordinaten

- Beispiel $k = 3, n = 2$ (Dreieck im \mathbb{R}^2)
 - alle Punkte Q mit $Q = \lambda_1 P_1 + \lambda_2 P_2 + \lambda_3 P_3$ und $\lambda_1 + \lambda_2 + \lambda_3 = 1$ liegen innerhalb des Dreiecks $\Delta(P_1, P_2, P_3)$ gdw. $\lambda_1, \lambda_2, \lambda_3$ positiv sind

Bemerkung:

Flächeninhalte sind Vielfache der Determinanten bei Lösung des LGS mit der Cramerschen Regel!



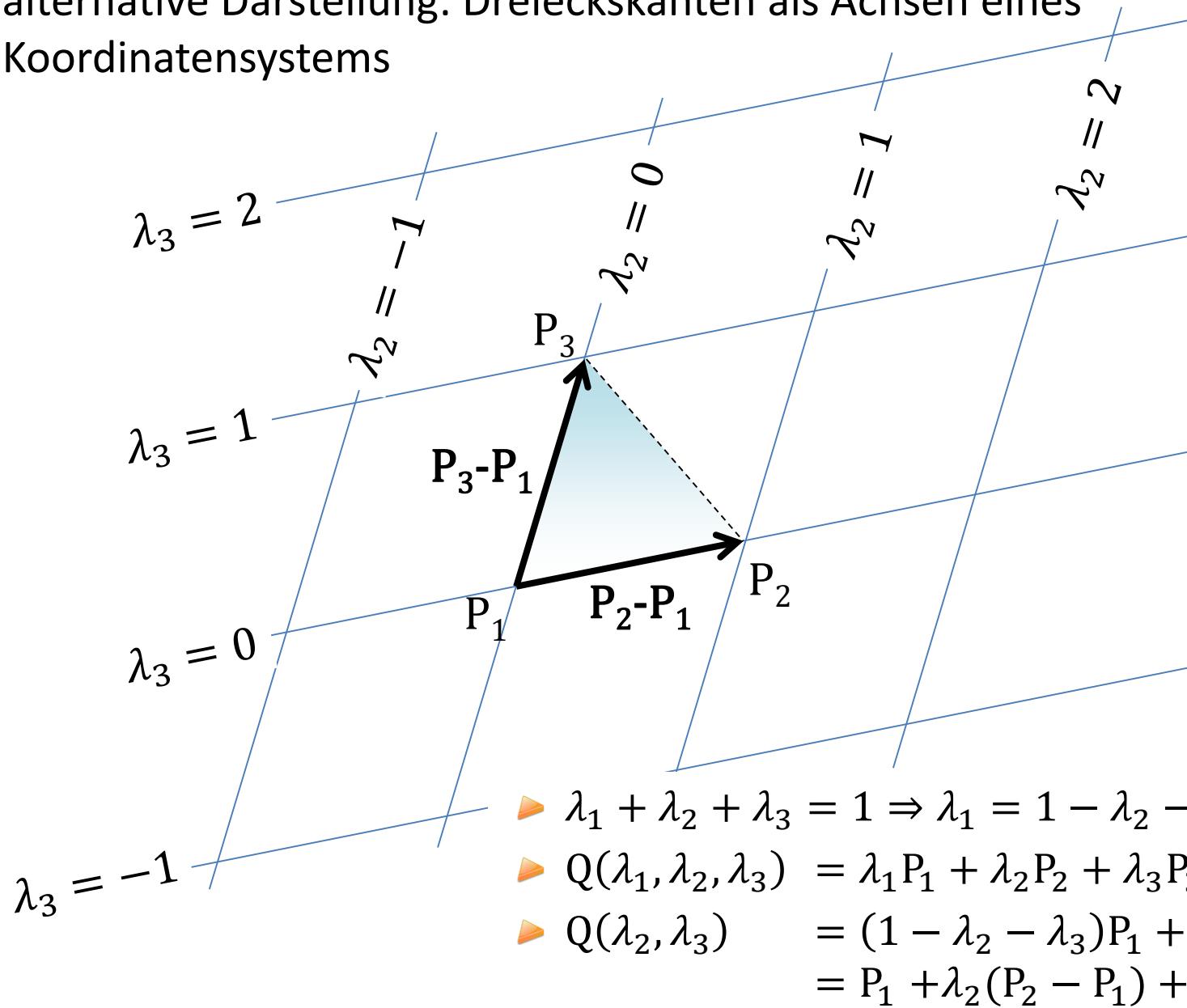
- geometrische Interpretation der baryzentrischen Koordinaten

$$\lambda_1 = \frac{A_{\Delta}(Q, P_2, P_3)}{A_{\Delta}(P_1, P_2, P_3)} \quad \lambda_2 = \frac{A_{\Delta}(P_1, Q, P_3)}{A_{\Delta}(P_1, P_2, P_3)} \quad \lambda_3 = \frac{A_{\Delta}(P_1, P_2, Q)}{A_{\Delta}(P_1, P_2, P_3)}$$

- $A_{\Delta}(Q, P_2, P_3) = \frac{1}{2} |P_2 - Q| |P_3 - Q| \sin \phi < 0$ und daher $\lambda_1 + \lambda_2 + \lambda_3 = 1$

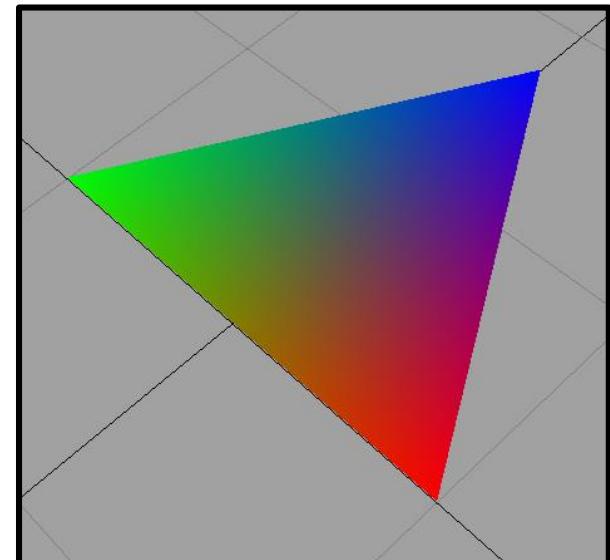
Baryzentrische Koordinaten

- alternative Darstellung: Dreiecksseiten als Achsen eines Koordinatensystems



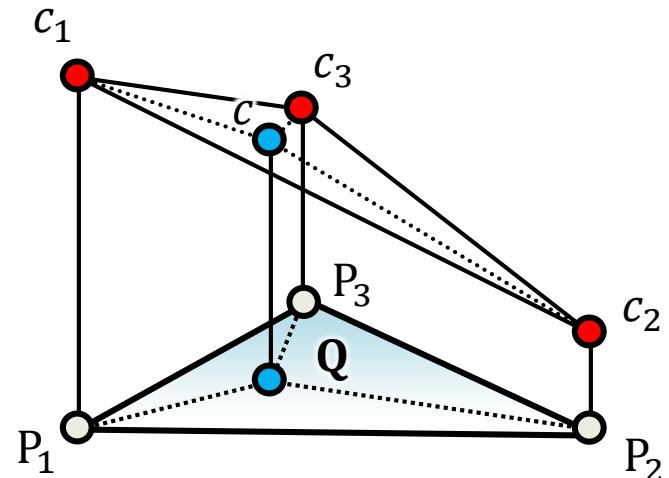
Anwendungen: Baryzentrische Koordinaten

- Beispiel: teste, ob ein Punkt Q innerhalb eines Dreiecks $\Delta(P_1, P_2, P_3)$ liegt
 - Lösung: berechne $\lambda_1, \lambda_2, \lambda_3$, z.B. über Teilflächen und Dreiecksfläche
 - Punkt $Q \in \Delta(P_1, P_2, P_3)$ gdw. $\lambda_1, \lambda_2, \lambda_3 \geq 0$
- Beispiel: lineare Interpolation von (Farb-)Werten
 - gegeben: eine Farbe c_1, c_2, c_3 (als RGB-Tripel) zu jedem Eckpunkt eines Dreiecks $\Delta(P_1, P_2, P_3)$
 - gesucht: interpolierte Farbe c_Q an einem Punkt Q auf dem Dreieck
 - Lösung: berechne $\lambda_1, \lambda_2, \lambda_3$,
dann ist $c_Q = \lambda_1 c_1 + \lambda_2 c_2 + \lambda_3 c_3$
- **Interpolation mit baryzentrischen Koordinaten = lineare Interpolation**



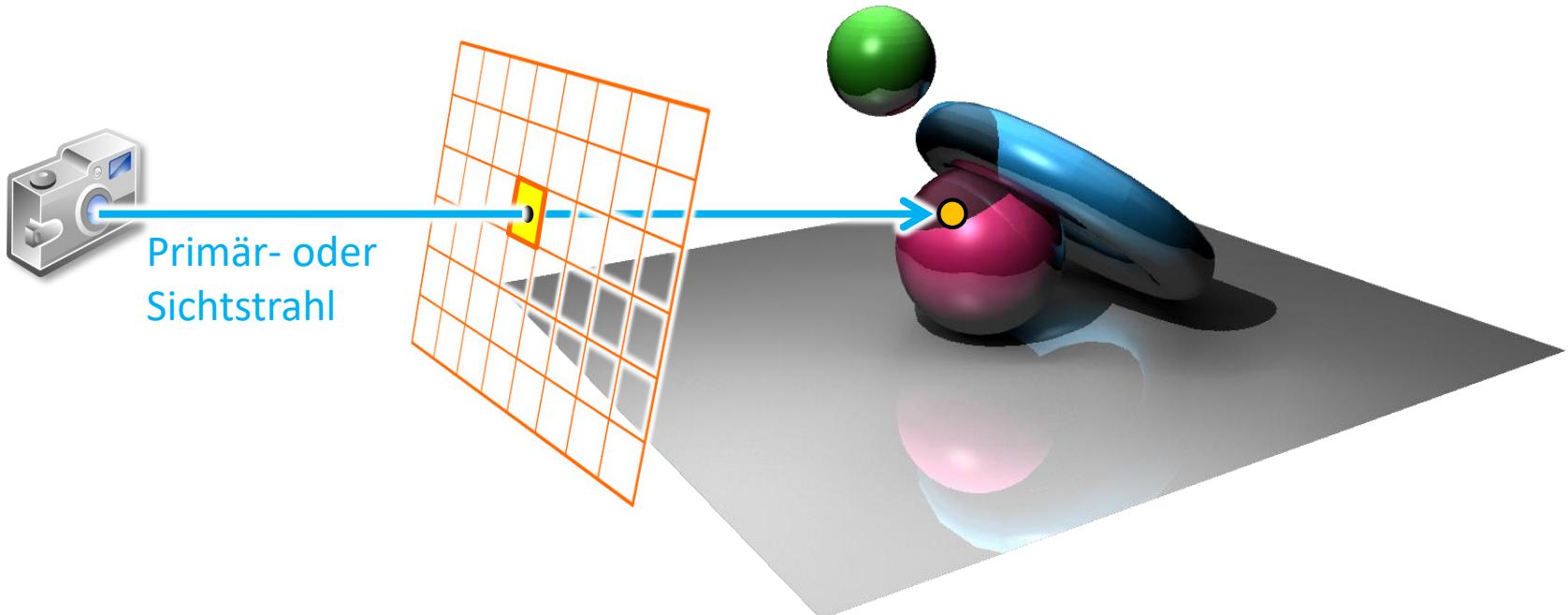
Anwendungen: Baryzentrische Koordinaten

- Beispiel: teste, ob ein Punkt Q innerhalb eines Dreiecks $\Delta(P_1, P_2, P_3)$ liegt
 - Lösung: berechne $\lambda_1, \lambda_2, \lambda_3$, z.B. über Teilflächen und Dreiecksfläche
 - Punkt $Q \in \Delta(P_1, P_2, P_3)$ gdw. $\lambda_1, \lambda_2, \lambda_3 \geq 0$
- Beispiel: lineare Interpolation von (Farb-)Werten
 - gegeben: eine Farbe c_1, c_2, c_3 (als RGB-Tripel) zu jedem Eckpunkt eines Dreiecks $\Delta(P_1, P_2, P_3)$
 - gesucht: interpolierte Farbe c_Q an einem Punkt Q auf dem Dreieck
 - Lösung: berechne $\lambda_1, \lambda_2, \lambda_3$,
dann ist $c_Q = \lambda_1 c_1 + \lambda_2 c_2 + \lambda_3 c_3$
- **Interpolation mit baryzentrischen Koordinaten = lineare Interpolation**



Schritte

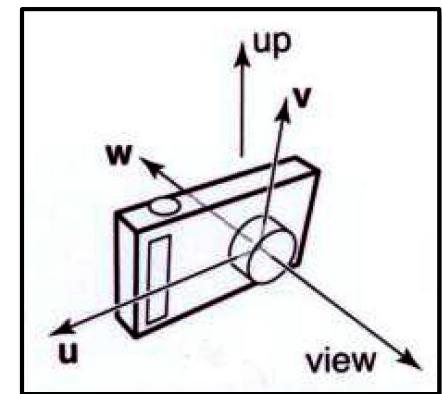
- Erzeugung der Sichtstrahlen durch jeden Pixel (ray generation)
- Schnittberechnung (ray casting):
finde Dreieck, das den Sichtstrahl am nächsten zur Kamera schneidet
- Schattierung und Beleuchtungsberechnung (shading)
- Sekundärstrahlen für Spiegelung und Transmission



Primär- oder
Sichtstrahl

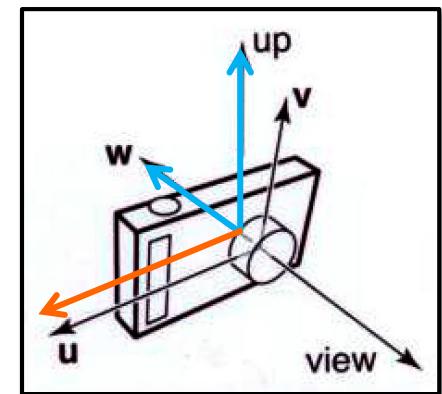
Erzeugung von Sichtstrahlen (ray generation)

- virtuelle Kamera (Lochkamera-Modell) definiert durch
 - ▶ Position (= Projektionszentrum, „eye“) e
 - ▶ Zielpunkt z
 - ▶ „Up-Vektor“ up ($|up| = 1$)
 - ▶ negative Blickrichtung $w = (e - z)/|e - z|$
 - ▶ $\Rightarrow u = up \times w$ und $v = w \times u$
 - ▶ Normalisiere die Vektoren u und v



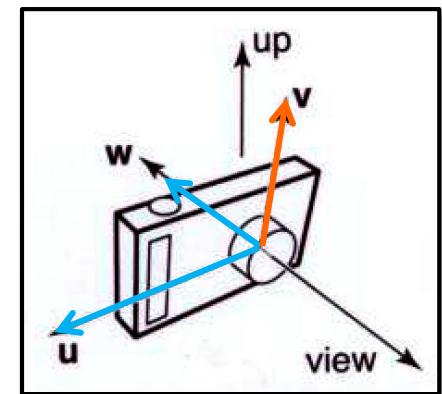
Erzeugung von Sichtstrahlen (ray generation)

- virtuelle Kamera (Lochkamera-Modell) definiert durch
 - Position (= Projektionszentrum, „eye“) e
 - Zielpunkt z
 - „Up-Vektor“ up ($|up| = 1$)
 - negative Blickrichtung $w = (e - z)/|e - z|$
 - $\Rightarrow u = up \times w$ und $v = w \times u$
 - Normalisiere die Vektoren u und v



Erzeugung von Sichtstrahlen (ray generation)

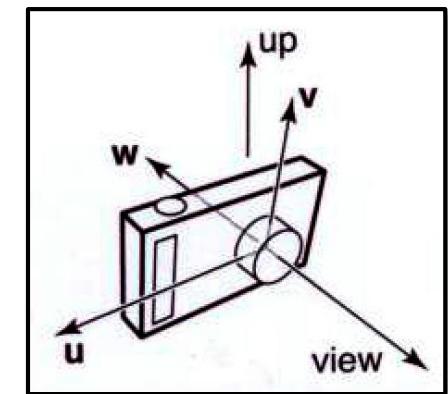
- virtuelle Kamera (Lochkamera-Modell) definiert durch
 - ▶ Position (= Projektionszentrum, „eye“) e
 - ▶ Zielpunkt z
 - ▶ „Up-Vektor“ up ($|up| = 1$)
 - ▶ negative Blickrichtung $w = (e - z)/|e - z|$
 - ▶ $\Rightarrow u = up \times w$ und $v = w \times u$
 - ▶ Normalisiere die Vektoren u und v



Erzeugung von Sichtstrahlen (ray generation)

- ▶ virtuelle Kamera (Lochkamera-Modell) definiert durch

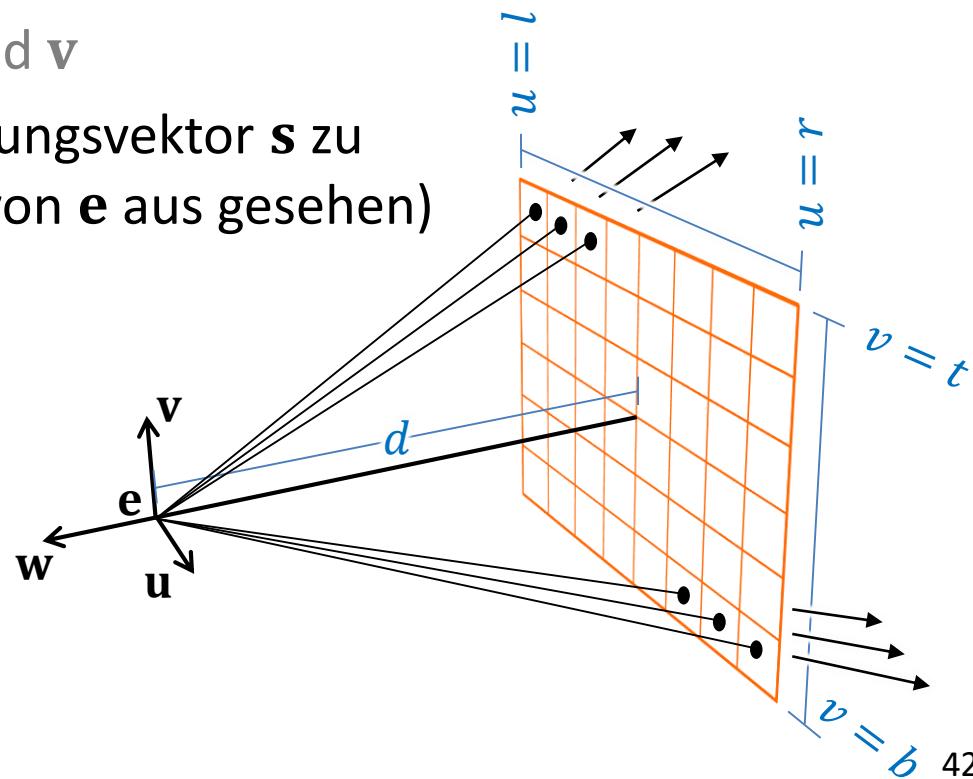
- ▶ Position (= Projektionszentrum, „eye“) e
- ▶ Zielpunkt z
- ▶ „Up-Vektor“ up ($|up| = 1$)
- ▶ negative Blickrichtung $w = (e - z)/|e - z|$
- ▶ $\Rightarrow u = up \times w$ und $v = w \times u$
- ▶ Normalisiere die Vektoren u und v



- ▶ Erzeugen von Sichtstrahlen: Richtungsvektor s zu Punkten auf der Bildebene (also von e aus gesehen)

- ▶ Bildebene gegeben durch
Abstand zur Kamera: d
linker/rechter Rand: l und r
unterer/oberer Rand: b und t

- ▶ $s = u \cdot u + v \cdot v - d \cdot w$,
mit $u \in [l, r]$ und $v \in [b, t]$



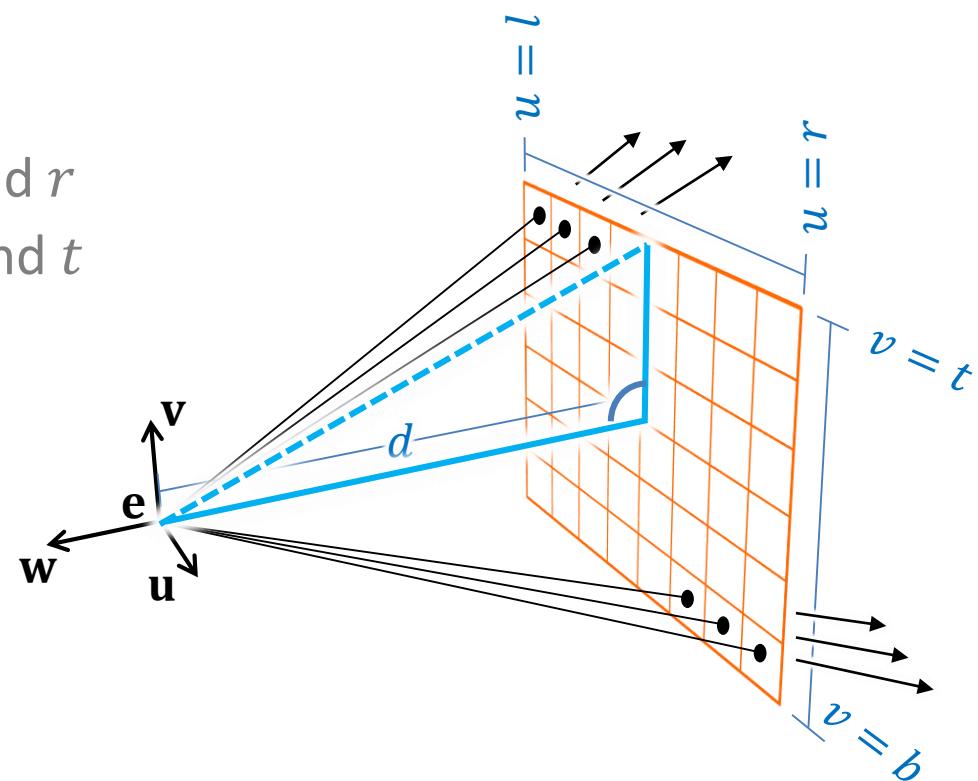
Erzeugung von Sichtstrahlen (ray generation)

- Bildebene gegeben durch

- ▶ Abstand zur Kamera: d
- ▶ linker/rechter Rand: l und r
- ▶ unterer/oberer Rand: b und t

- Richtungsvektor s zu Zielpunkten auf der Bildebene

- ▶ $s = u \cdot \mathbf{u} + v \cdot \mathbf{v} - d \cdot \mathbf{w}$, mit $u \in [l, r]$ und $v \in [b, t]$



- Bsp. typisches Sichtfeld ist 90° und symmetrisch, d.h. $l = -r \wedge b = -t$
- ▶ $\tan 90^\circ/2 = t/d$ (d wird festgelegt) und
- ▶ $(r - l)/(t - b) = r/t$ „Aspect Ratio“
- ▶ Aspect Ratio = Verhältnis Breite zu Höhe des Bildschirms
(typische Werte: 4:3, 16:9, 16:10, 21:9, ...)

Erzeugung von Sichtstrahlen (ray generation)

► Richtungsvektor s

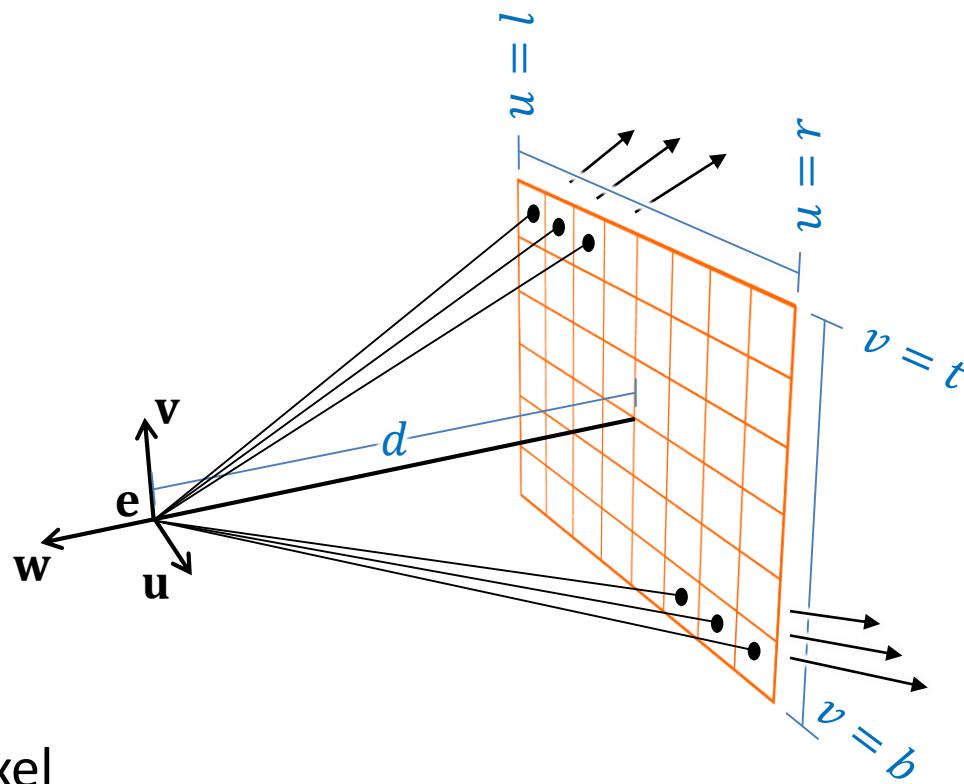
- $s = u \cdot \mathbf{u} + v \cdot \mathbf{v} - d \cdot \mathbf{w}$,
mit $u \in [l, r]$ und $v \in [b, t]$

► Strahlgleichung $\mathbf{r}(t) = \mathbf{e} + ts$

- $t = 0 \rightarrow \mathbf{e}$ (Kameraposition)
- $t = 1 \rightarrow \mathbf{e} + s$ (Pixelmitte)
- i.d.R. $\mathbf{r}(t) = \mathbf{e} + t\mathbf{d}$,
mit $\mathbf{d} = s/|s|$

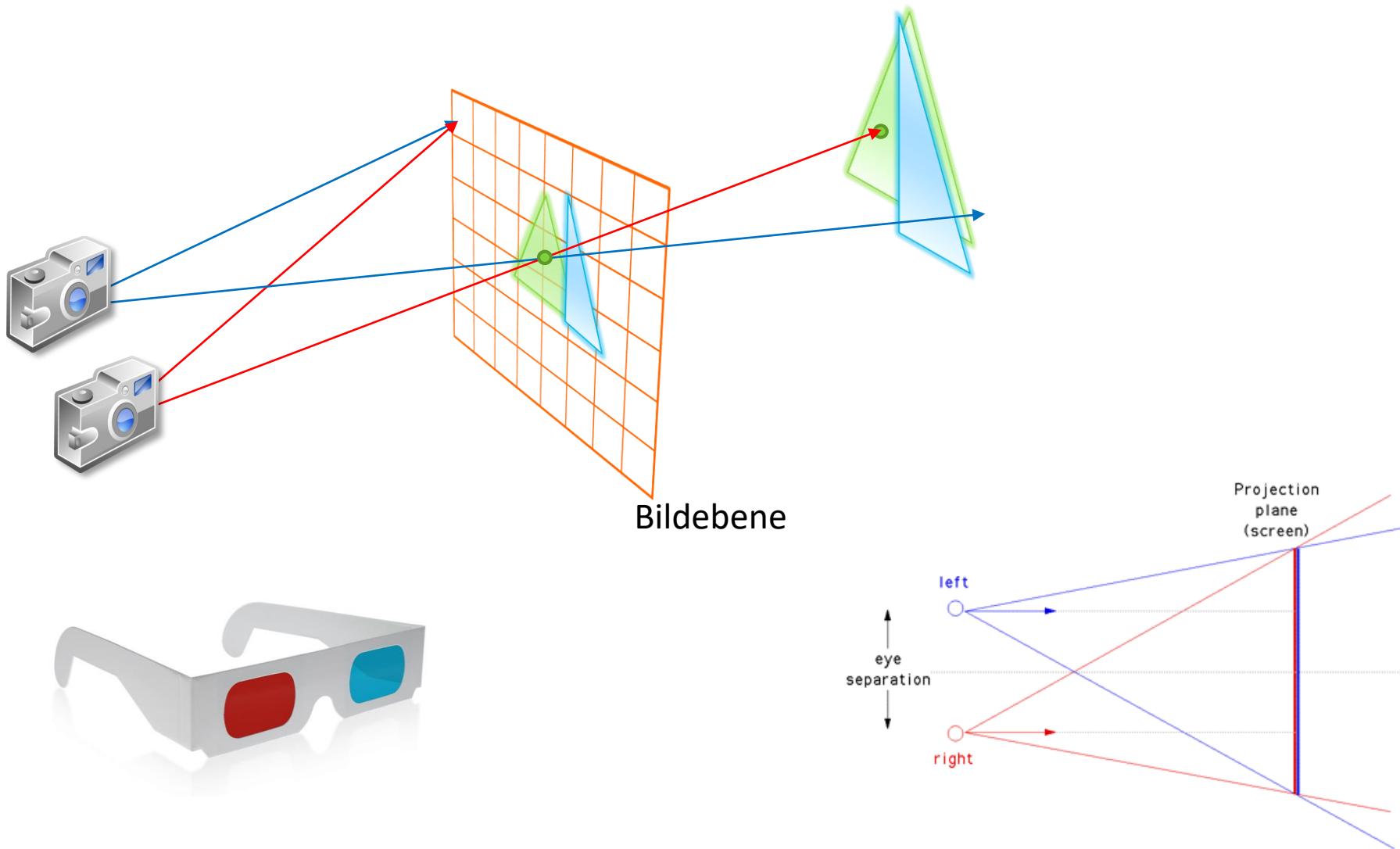
► Pseudocode: ein Zielpunkt pro Pixel

```
for ( y = 0; y < height; y++ ) {
    for ( x = 0; x < width; x++ ) {
        u = l + (r-l) * (x+0.5) / width;
        v = t + (b-t) * (y+0.5) / height;
        ...
    }
}
```



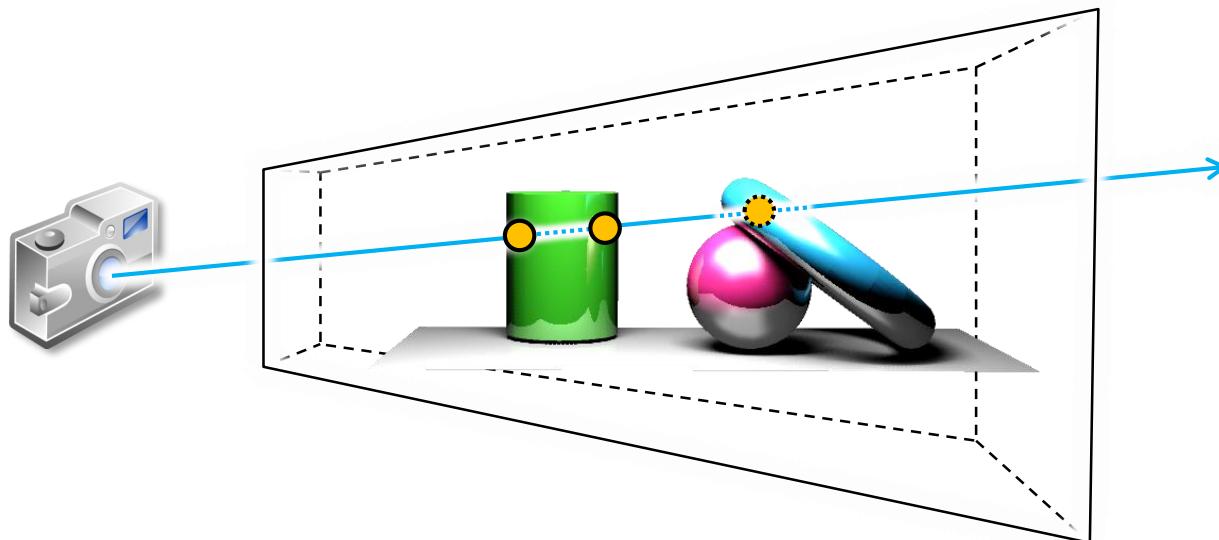
Raytracing, Stereo-Rendering

- Grundidee: für jedes Auge ein Bild



Schritte

- ▶ Erzeugung der Sichtstrahlen durch jeden Pixel (ray generation)
- ▶ Schnittberechnung (ray casting):
finde geometrisches Primitiv/Objekt (und Schnittpunkt damit),
das den Sichtstrahl am nächsten zur Kamera schneidet
- ▶ Schattierung und Beleuchtungsberechnung (shading)
- ▶ Sekundärstrahlen für Spiegelung und Transmission



Raytracing Pseudocode

```
► for ( y = 0; y < height; y++ ) {
    for ( x = 0; x < width; x++ ) {
        u = 1 + (r-1) * (x+0.5) / width;
        v = t + (b-t) * (y+0.5) / height;
        s = ...;
        d = normalize( s );

        // finde nächsten Schnittpunkt
        intersection = NULL;
        float t = FLOAT_MAX;

        for ( each object ) {
            t' = intersect( object, e, d );
            if ( t' > 0 && t' < t ) {
                intersection = object;
                t = t';
            }
        }
    }
}
```



► Achtung: uns interessieren nur Schnittpunkte vor der Kamera ($t' > 0$), und von diesen der nächste in Strahlrichtung ($t' < t$)

► Parameterdarstellung

alle Punkte eines geometrischen Gebildes (Linie, Ebene etc.) kann man durch Einsetzen aller zulässigen Parameterwerte gewinnen

- Bsp.: Kreis in 2D: $x(t) = r \cdot \cos t, y(t) = r \cdot \sin t, t \in [0..2\pi)$

► Explizite Darstellung

- Bsp.: Kurve in 2D: $y = f(x)$, Fläche in 3D: $z = f(x, y)$

► Implizite Darstellung

die Punkte die zu einem geometrischen Gebilde gehören bilden die Lösungsgesamtheit eines Systems von Gleichungen

- Bsp.: Gerade in 2D

$$G = \{(x, y) | ax + by + c = 0 \wedge a, b, c \in \mathbb{R}\}$$

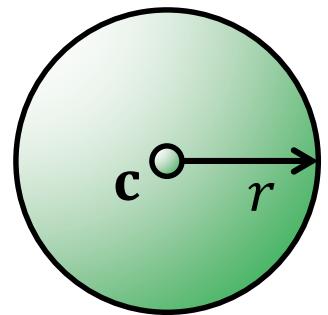
$$ax + by + c > 0$$

$$ax + by + c < 0$$

Implizite Darstellung von Kugel und Torus

- Kugel: alle Punkte auf der Kugeloberfläche haben den Abstand r vom Mittelpunkt $\mathbf{c} = (c_x, c_y, c_z)$

$$\begin{aligned} K &= \{(x, y, z) \mid |(x - c_x, y - c_y, z - c_z)| = r\} \\ &= \left\{ (x, y, z) \mid |(x - c_x, y - c_y, z - c_z)|^2 = r^2 \right\} \end{aligned}$$



- Torus (nur damit Sie es mal gesehen haben):

- implizit:

$$(x^2 + y^2 + z^2 + b^2 - a^2)^2 = 4b^2(x^2 + y^2)$$

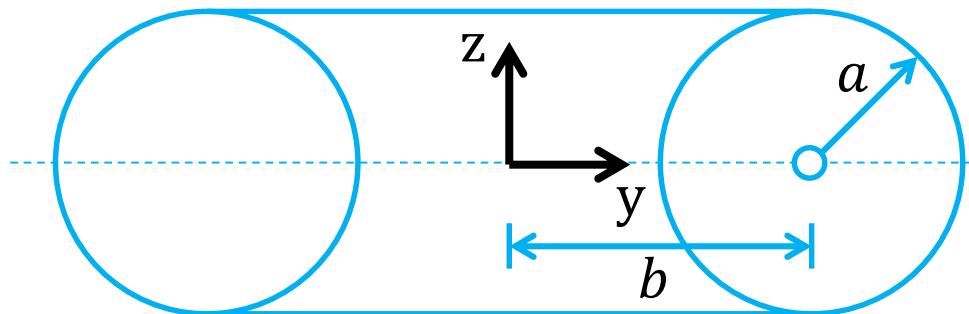
- parametrisch:

$$x = -\sin(u)(b + a \cdot \cos(v))$$

$$y = \cos(u)(b + a \cdot \cos(v))$$

$$z = a \cdot \sin(v)$$

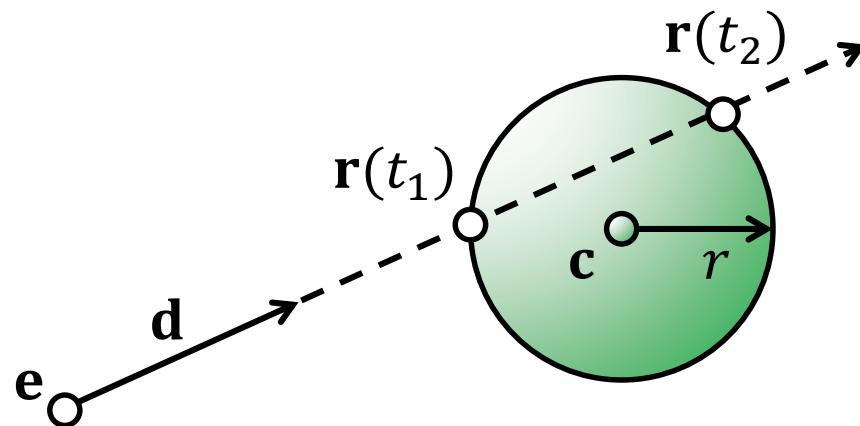
$$u, v \in [0..2\pi)$$



Schnittpunktberechnung

Strahl-Kugel-Schnitt

- ▶ Strahlgleichung (parametrisch) $\mathbf{r}(t) = \mathbf{e} + t\mathbf{d}$
- ▶ Kugel (implizite Darstellung) $|\mathbf{x} - \mathbf{c}|^2 - r^2 = 0$
- ▶ Einsetzen:
 - ▶ $|\mathbf{r}(t) - \mathbf{c}|^2 - r^2 = 0$
 - ▶ $|\mathbf{e} + t\mathbf{d} - \mathbf{c}|^2 - r^2 = 0 \quad (|\mathbf{x}|^2 = \mathbf{x} \cdot \mathbf{x})$
 - ▶ $(\mathbf{e} + t\mathbf{d} - \mathbf{c}) \cdot (\mathbf{e} + t\mathbf{d} - \mathbf{c}) - r^2 = 0$
 - ▶ $\underbrace{(\mathbf{e} - \mathbf{c}) \cdot (\mathbf{e} - \mathbf{c}) - r^2}_{const.} + \underbrace{2(t\mathbf{d} \cdot (\mathbf{e} - \mathbf{c}))}_{t(2\mathbf{d} \cdot (\mathbf{e} - \mathbf{c}))} + \underbrace{(t\mathbf{d}) \cdot (t\mathbf{d})}_{t^2(\mathbf{d} \cdot \mathbf{d})} = 0$



Schnittpunktberechnung

Strahl-Kugel-Schnitt

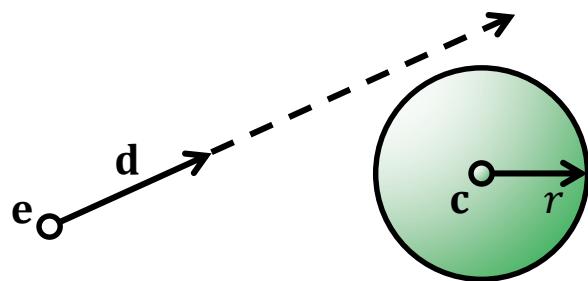
► $\underbrace{(\mathbf{e} - \mathbf{c}) \cdot (\mathbf{e} - \mathbf{c}) - r^2}_{\text{const.}} + \underbrace{2(t\mathbf{d} \cdot (\mathbf{e} - \mathbf{c}))}_{t(2\mathbf{d} \cdot (\mathbf{e} - \mathbf{c}))} + \underbrace{(t\mathbf{d}) \cdot (t\mathbf{d})}_{t^2(\mathbf{d} \cdot \mathbf{d})} = 0$

► quadratische Gleichung: Mitternachtsformel

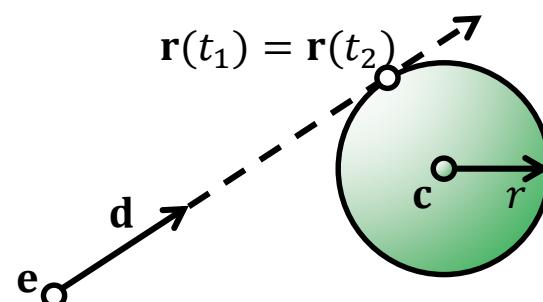
► $t_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

mit $a = \mathbf{d} \cdot \mathbf{d}$, $b = 2\mathbf{d} \cdot (\mathbf{e} - \mathbf{c})$, $c = (\mathbf{e} - \mathbf{c}) \cdot (\mathbf{e} - \mathbf{c}) - r^2$

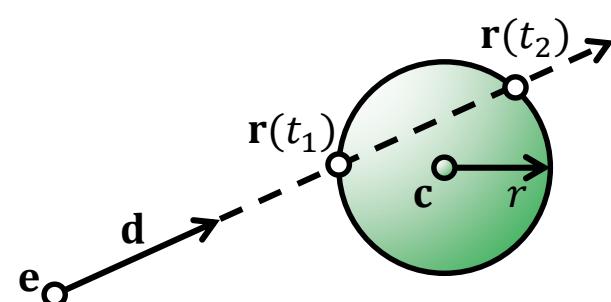
► Diskriminante $b^2 - 4ac$ (Achtung: bei 2 Lösungen kann $t_1 < 0$ sein)



Diskriminante < 0
→ keine Lösung



Diskriminante $= 0$
→ eine Lösung



Diskriminante > 0
→ zwei Lösungen

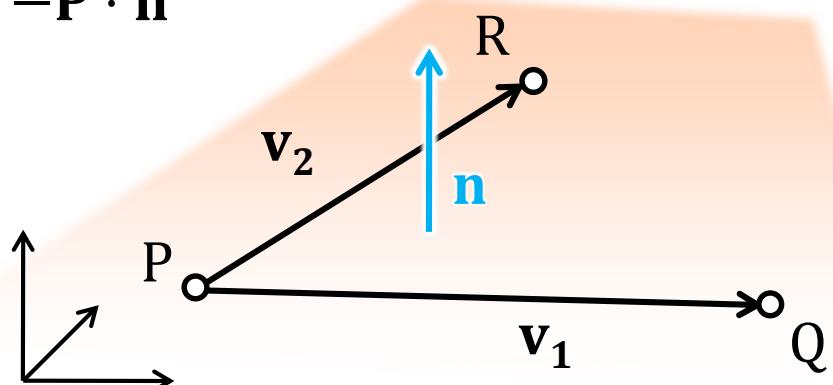
Implizite und Parameter-Darstellung einer Ebene

- eine Ebene E im \mathbb{R}^3 hat die implizite Form

$$E = \{(x, y, z) | ax + by + cz + d = 0 \wedge a, b, c, d \in \mathbb{R}, \neg a, b, c = 0\}$$

- Parameterdarstellung \rightarrow implizite Darstellung

- geg. Ebene durch drei nicht-kollineare Punkte P, Q, R
(alternativ: Punkt P und Normale \mathbf{n})
- ges. Werte a, b, c, d der Ebenengleichung
- aufspannende Vektoren: $\mathbf{v}_1 = Q - P$ und $\mathbf{v}_2 = R - P$
- Normale der Ebene: $\mathbf{n} = \mathbf{v}_1 \times \mathbf{v}_2$
- $a = n_x, b = n_y, c = n_z$ und $d = -P \cdot \mathbf{n}$



Implizite und Parameter-Darstellung einer Ebene

- eine Ebene E im \mathbb{R}^3 hat die implizite Form

$$E = \{(x, y, z) | ax + by + cz + d = 0 \wedge a, b, c, d \in \mathbb{R}, \neg a, b, c = 0\}$$

- Parameterdarstellung \rightarrow implizite Darstellung

- geg. Ebene durch drei nicht-kollineare Punkte P, Q, R
(alternativ: Punkt P und Normale \mathbf{n})
- ges. Werte a, b, c, d der Ebenengleichung

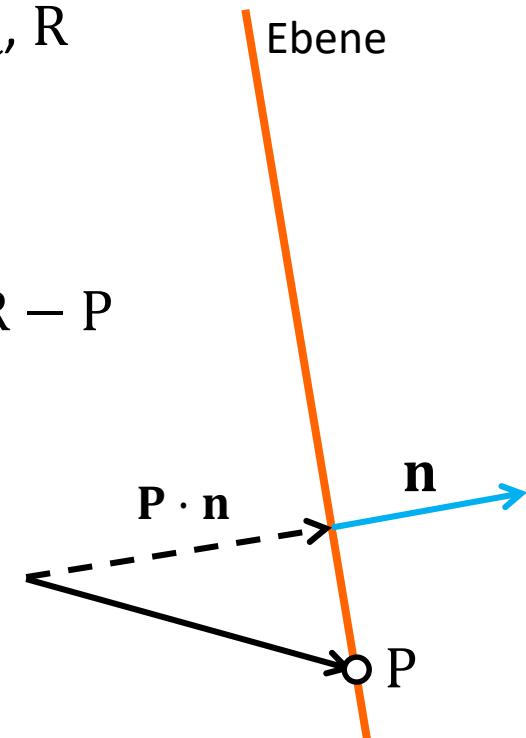
- aufspannende Vektoren: $\mathbf{v}_1 = Q - P$ und $\mathbf{v}_2 = R - P$

- Normale der Ebene: $\mathbf{n} = \mathbf{v}_1 \times \mathbf{v}_2$

- $a = n_x, b = n_y, c = n_z$ und $d = -P \cdot \mathbf{n}$

- ist $|\mathbf{n}| = 1$, dann spricht man von
der Hesse-Normalform (HNF)

- Überführung in HNF durch teilen von a, b, c, d durch $|\mathbf{n}|$



Schnittpunktberechnung mit Ebene

Strahl-Ebene-Schnitt

- ▶ Strahl (parametrische Repr.) $\mathbf{r}(t) = \mathbf{e} + t\mathbf{d}, |\mathbf{d}| = 1$
- ▶ Ebene (implizite Repr., HNF) $\mathbf{x} \cdot \mathbf{n} - d = 0, |\mathbf{n}| = 1$
- ▶ Normalenvektor: \mathbf{n}
- ▶ Abstand vom Ursprung: d (wg. $|\mathbf{n}| = 1$)
(Achtung: Vorzeichen anders als auf letzter Folie)

- ▶ Lösung durch Einsetzen

▶ Einsetzen:

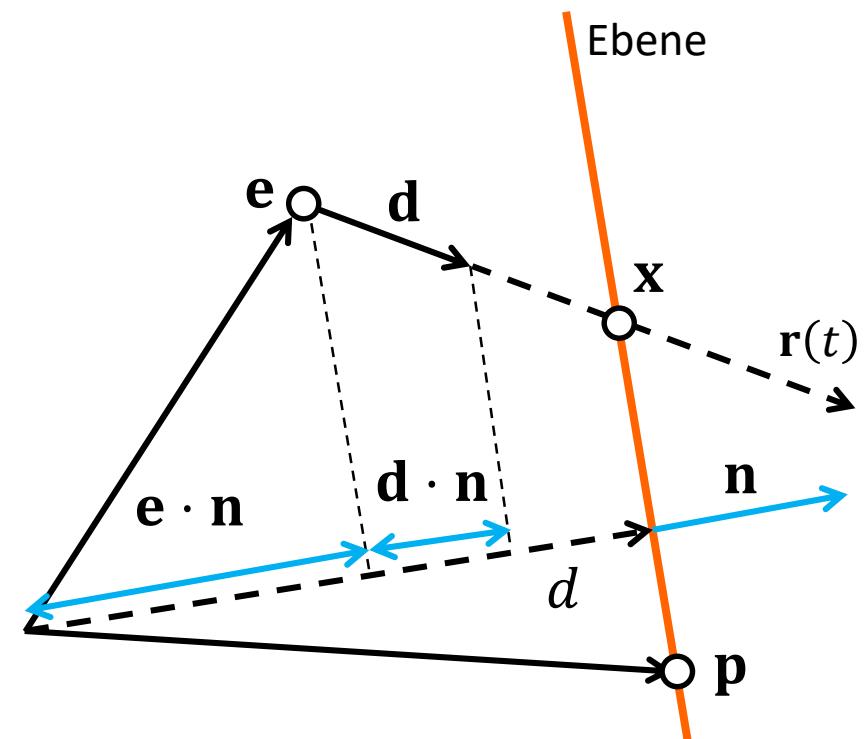
$$(\mathbf{e} + t\mathbf{d}) \cdot \mathbf{n} - d = 0$$

$$\mathbf{e} \cdot \mathbf{n} + t(\mathbf{d} \cdot \mathbf{n}) - d = 0$$

▶ Auflösen ergibt: $t = \frac{d - \mathbf{e} \cdot \mathbf{n}}{\mathbf{d} \cdot \mathbf{n}}$

▶ $\mathbf{d} \cdot \mathbf{n} = 0 \Leftrightarrow$

Strahl und Ebene sind parallel



Schnittpunktberechnung mit Dreieck

Strahl-Dreieck-Schnitt (anschaulich, so macht man es aber nicht!)

- ▶ baryzentrische Koordinaten eines potentiellen Schnittpunkts

- ▶ $Q = \lambda_1 P_1 + \lambda_2 P_2 + \lambda_3 P_3$

mit $\lambda_1 + \lambda_2 + \lambda_3 = 1$

- ▶ $\lambda_3 = \Delta(P_1 Q P_2) / \Delta(P_1 P_2 P_3)$, etc.

- ▶ Punkt in Dreieck, falls alle λ_i größer als Null

- ▶ möglicher Schnitttest:

- ▶ Ebene durch Dreieck legen

- ▶ berechne Schnittpunkt mit Ebene

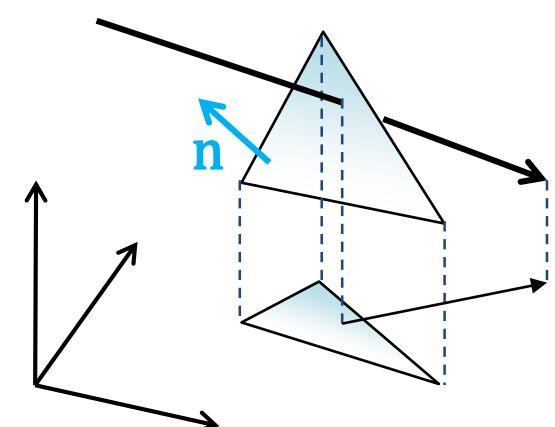
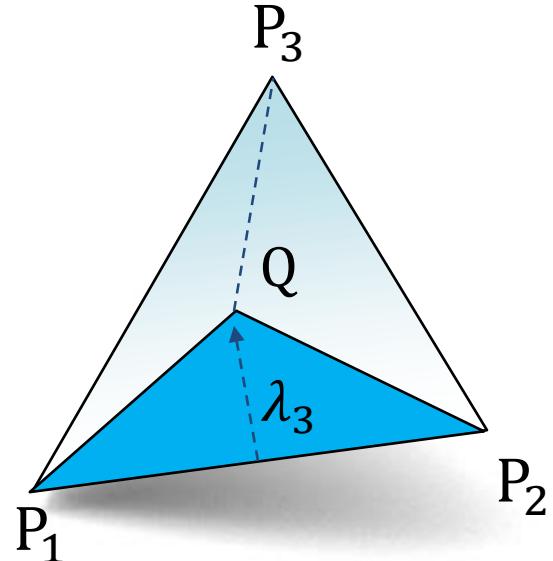
- ▶ berechne baryzentrische Koordinaten

- ▶ z.B. auch über 2D-Punkte in Ebene:

Projektion in Hauptebene

mit max. Normalenkoordinate

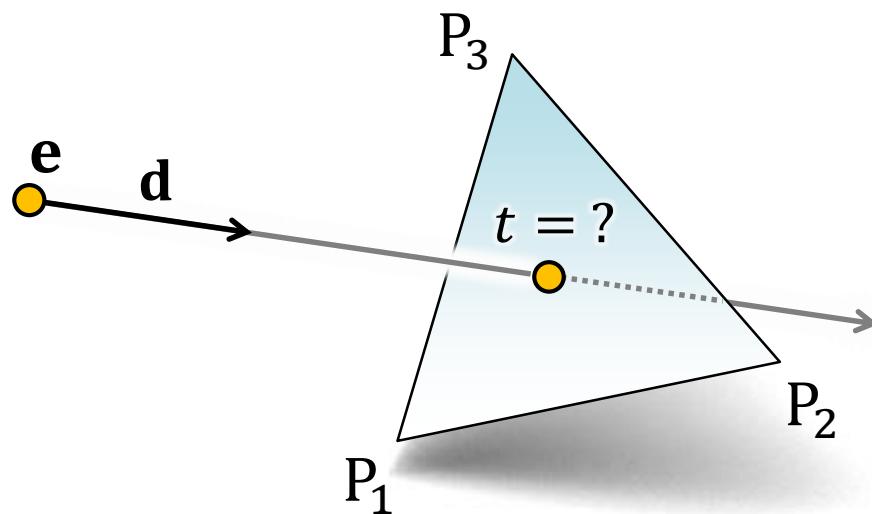
- ▶ teste $\lambda_1, \lambda_2, \lambda_3$ auf Positivität



Schnittpunktberechnung mit Dreieck

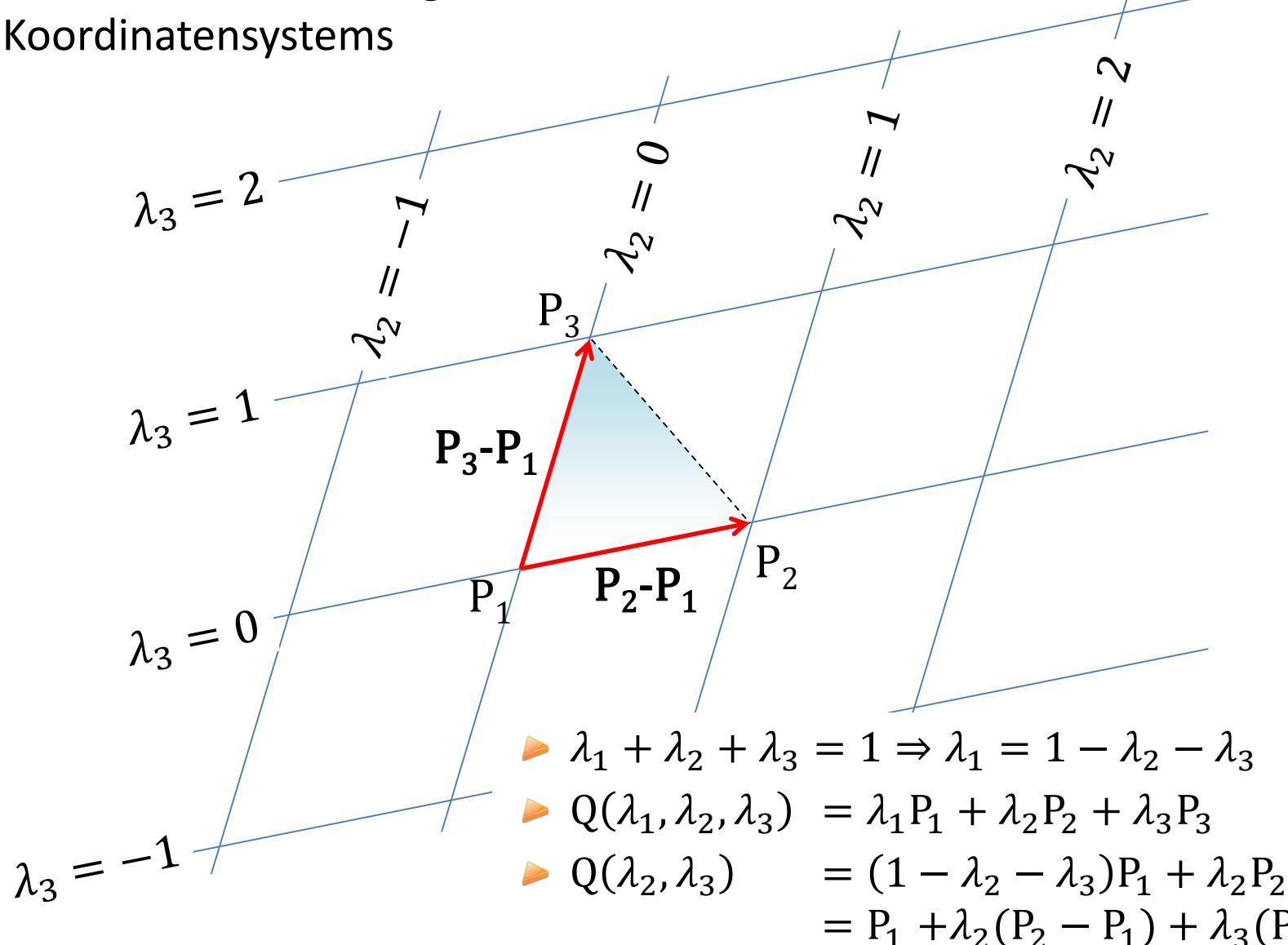
Strahl-Dreieck-Schnitt (direkt, so wird es gemacht!)

- ▶ Strahlgleichung (allg.): $\mathbf{r}(t) = \mathbf{e} + t\mathbf{d}$, $|\mathbf{d}| = 1$
- ▶ Dreieck mit Eckpunkten P_1, P_2, P_3
- ▶ existiert ein Schnittpunkt? wenn ja, für welches t ?



Zur Erinnerung: Baryzentrische Koordinaten

- alternative Darstellung: Dreiecksseiten als Achsen eines **schiefwinkligen** Koordinatensystems

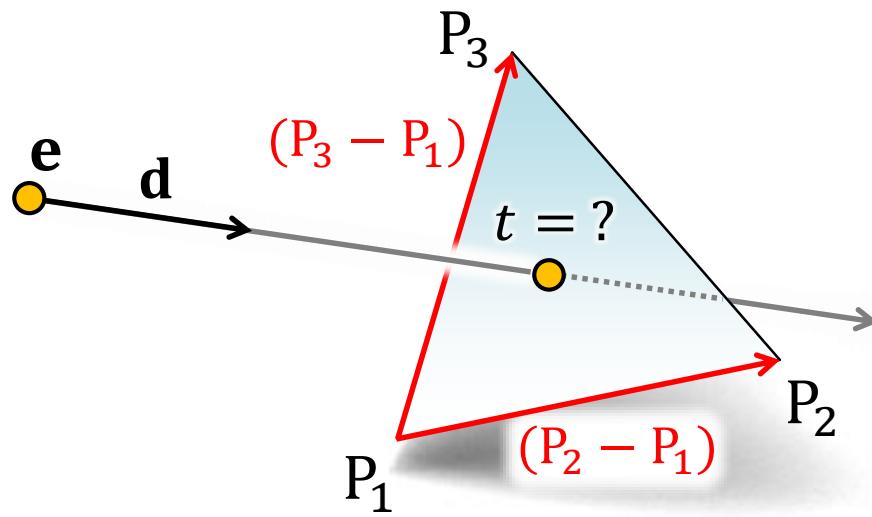


Schnittpunktberechnung

Strahl-Dreieck-Schnitt

- jeder Punkt in der Ebene des Dreiecks lässt sich beschreiben durch

$$Q(\lambda_2, \lambda_3) = P_1 + \lambda_2(P_2 - P_1) + \lambda_3(P_3 - P_1)$$



Interpretation:

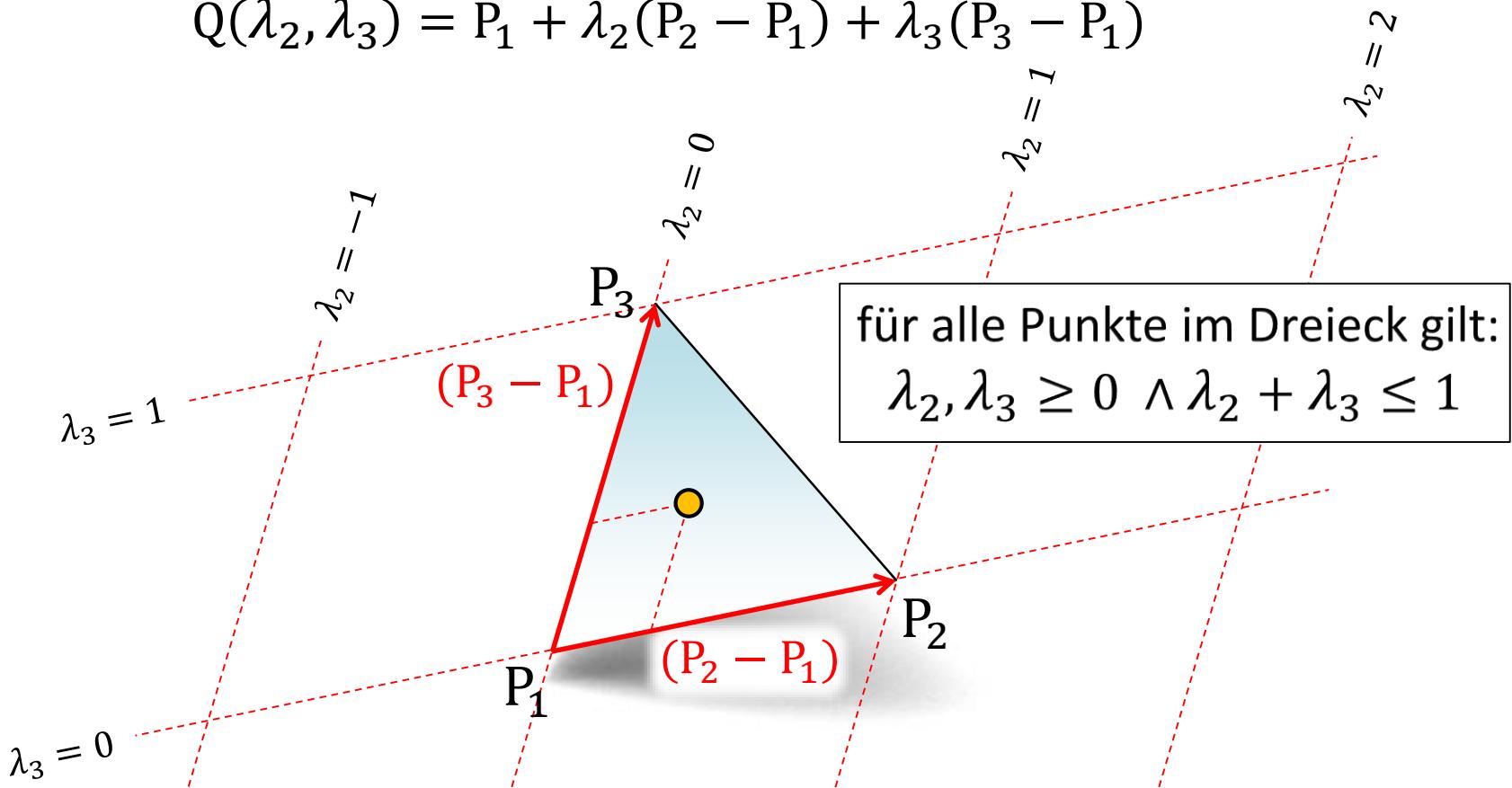
Dreiecksanten als Achsen eines schiefwinkligen Koordinatensystems

Schnittpunktberechnung mit Dreieck

Strahl-Dreieck-Schnitt

- jeder Punkt in der Ebene des Dreiecks lässt sich beschreiben durch

$$Q(\lambda_2, \lambda_3) = P_1 + \lambda_2(P_2 - P_1) + \lambda_3(P_3 - P_1)$$



Interpretation:

Dreiecksanten als Achsen eines schiefwinkligen Koordinatensystems

Schnittpunktberechnung mit Dreieck

Strahl-Dreieck-Schnitt

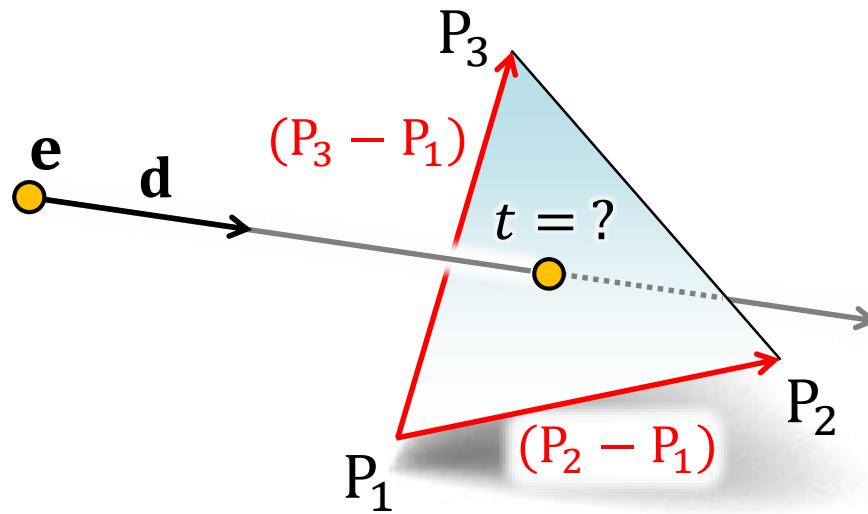
- Gleichsetzen $\mathbf{r}(t) = \mathbf{Q}(\lambda_2, \lambda_3)$

$$\mathbf{e} + t\mathbf{d} = \mathbf{P}_1 + \lambda_2(\mathbf{P}_2 - \mathbf{P}_1) + \lambda_3(\mathbf{P}_3 - \mathbf{P}_1)$$

► Lösung existiert \Leftrightarrow Schnittpunkt mit der Ebene

► 3 Gleichungen, 3 Unbekannte t, λ_2, λ_3 :

$$e_x + t d_x = P_{1,x} + \lambda_2(P_{2,x} - P_{1,x}) + \lambda_3(P_{3,x} - P_{1,x}), \dots$$



- Schnittpunkt mit dem Dreieck, wenn $\lambda_2, \lambda_3 \geq 0 \wedge \lambda_2 + \lambda_3 \leq 1$, in Richtung des Strahls, wenn $t > 0$
- Lösung effizient mittels Cramerscher Regel (Determinanten)

Weitere Schnitttests

► <http://realtimerendering.com/intersections.html>

	ray	plane	sphere	cylinder	cone	triangle	AABB	OBB	frustum	polyhedron	
ray	Gems p. 304; Gems p. 343; TGS; RTCD p. 198; SoftSurfer; RTCD p. 616; RT3 p. 781	IRT p. 90, 88; SG; GTWeb; Gems p. 482; TGS; RTCD p. 175; SoftSurfer (more)	IRT p. 39, 91; Gems p. 388; Heid (pt 2)(4); GTWeb; 3DG p. 16; GTCG p. 501; TGS; RTCD p. 127, 177; RT2 p. 568; RT3 p. 738	IRT p. 91; Gems IV p. 388; Heid (pt 2)(4); GTWeb; GTCG p. 507; TGS; RTCD p. 194	IRT p. 91; Gems V p. 227; Heid (pt 2)(4); GTWeb; GTCG p. 512	Moller-Trumbore (pt 2)(1); IRT p. 83, 102; Gems p. 343; Heid (pt 2)(4); GTWeb; 3DG p. 17; GTCG p. 485; TGS; RTCD p. 159, 184; Loeb (pt 1)(2); Chirikov (pt 1)(2); Lagae (pt 10)(4); SoftSurfer; RTCD p. 578; RT3 p. 746; Havoc TCVG June 2009	IRT p. 65, 104; Gems p. 393; Heid (pt 2)(4); GTWeb; 3DG p. 20; Terdiman (optimized Woo); Schroeder; GTCG p. 626; TGS; RTCD p. 179; Mahovsky (pt 3)(1); Williams (pt 10)(1); Eisemann (pt 12)(code); RTCD p. 572; RT3 p. 742	(IRT p. 104; Gems II p. 247); Gems p. 626; Heid (pt 2)(4); GTCG p. 630; TGS; (IRT p. 104; Gems II p. 247)	(IRT p. 104; Gems II p. 247); GTCG p. 630; TGS; RTCD p. 179; RT3 p. 572; RT3 p. 743	IRT p. 104; Gems II p. 247; GTCG p. 630; TGS; SoftSurfer	ray
plane	IRT p. 90, 88; SG; GTCD p. 482; TGS; RTCD p. 175; SoftSurfer (more)	GTWeb; SG; GTCG p. 529; RTCD p. 207	distance of center to plane <= radius; Heid (pt 2)(4); GTWeb; GTCG p. 548; TGS; RTCD p. 160, 219	GTWeb; GTCG p. 531; TGS;	GTWeb; GTCG p. 532; RTCD p. 164	Check if all vertices are on one side; (Moller (pt 2)(2)); GTCG p. 534; SoftSurfer	Gems IV p. 74; GTCG p. 634; TGS; RTCD p. 161, 222; RT2 p. 587; RT3 p. 759	GTWeb; GTCG p. 635; TGS; RTCD p. 161; RT2 p. 589; RT3 p. 757			plane
sphere	IRT p. 39, 91; Gems p. 388; Heid (pt 2)(4); GTWeb; 3DG p. 16; GTCG p. 501; TGS; RTCD p. 127, 177; RT2 p. 568; RT3 p. 738	distance of center to plane <= radius; Heid (pt 2)(4); GTWeb; GTCG p. 501; TGS; RTCD p. 160, 219	If radii A+B >= center/axis distance; Heid (pt 2)(4); GTWeb; GTCG p. 548; TGS; RTCD p. 88, 215, 223; RT3 p. 763	If radii A+B >= center/axis distance; Heid (pt 2)(4); GTWeb; GTCG p. 562; TGS; RTCD p. 114	If radii A+B >= axis/axis distance; Heid (pt 2)(4); GTWeb; GTCG p. 646; TGS; RTCD p. 114	Heid (pt 2)(4); GTWeb; Karanassos (pt 4)(1); TGS; RTCD p. 135, 167, 226	Gems p. 335; Gems V p. 375; GTCG p. 644; TGS; RTCD p. 130, 165, 228; RT2 p. 599; RT3 p. 763	TGS; RTCD p. 132, 166	GTWeb; Assignment; GPG p. 433; 3DG p. 445; RT2 p. 609; RT3 p. 774	3DG p. 462; RTCD p. 142	sphere
(capped) cylinder	IRT p. 91; Gems IV p. 388; Heid (pt 3)(4); GTWeb; GTCG p. 501; TGS; RTCD p. 194	GTWeb; GTCG p. 531; TGS;	If radii A+B >= center/axis distance; Heid (pt 2)(4); (GTCG p. 602); TGS; RTCD p. 114	GTWeb; GTCG p. 602	Heid (pt 2)(4); TGS	TGS	TGS	GPG p. 380		(capped) cylinder	
(capped) cone	IRT p. 91; Gems V p. 227; Heid (pt 2)(4); GTWeb; GTCG p. 512	GTWeb; GTCG p. 563; RTCD p. 184	GTWeb; (GTCG p. 602)	(GTCG p. 602)	Heid (pt 2)(4); GTWeb; GTCG p. 583					(capped) cone	
triangle (polygon)	Moller-Trumbore (pt 2)(1); IRT p. 83, 102; Gems IV p. 24; Heid (pt 2)(4); GTWeb; 3DG p. 17; GTCG p. 485; TGS; RTCD p. 159, 184; Loeb (pt 1)(2); Chirikov (pt 1)(2); Lagae (pt 10)(4); SoftSurfer; RTCD p. 578; RT3 p. 746	Check if all vertices are on one side; (Moller (pt 2)(2)); GTCG p. 534; SoftSurfer	Heid (pt 2)(4); GTWeb; Karanassos (pt 4)(1); TGS; RTCD p. 135, 167, 226	Heid (pt 2)(4); GTWeb; GTCG p. 583	Heid (pt 2)(4); GTWeb; GTCG p. 583	Moller (pt 2)(2); Heid (pt 2)(4); Gems p. 375; Moller; GPG p. 393; GTCG p. 539; TGS; RTCD p. 135, 172; Shen (pt 8)(1); Guigui (pt 8)(1); GPG p. 394; RT2 p. 590; RT3 p. 757	Gems III p. 238; Gems V p. 375; GTCG p. 539; TGS; RTCD p. 169; Moller; RT2 p. 596; RT3 p. 760	GTWeb; TGS	homogeneous clipping; Gems p. 84	generalized clipping	triangle (polygon)
axis-aligned bounding box	IRT p. 65, 104; Gems p. 393; Smits; 3DG p. 20; Terdiman (optimized Woo); Schroeder; GTCG p. 626; TGS; RTCD p. 179; Mahovsky (pt 3)(1); Williams (pt 10)(1); Eisemann (pt 12)(4); RTCD p. 572; RT3 p. 742	Gems IV p. 74; GTCG p. 634; TGS; RTCD p. 161, 222; RT2 p. 587; RT3 p. 763	Gems p. 335; Gems; GTCG p. 644; TGS; RTCD p. 130, 165, 228; RT2 p. 599; RT3 p. 763	TGS		A. LO <= B, HI & A. HI >= B, LO; Gems p. 375; 3DG p. 445; GTCG p. 637; TGS; RTCD p. 169; Moller; RT2 p. 596; RT3 p. 760	(Gems V p. 83); (Gems V p. 375); (GTCG p. 639); Assignment; 3DG p. 445; GTCG p. 639; TGS; RTCD p. 169; Moller; RT2 p. 596; RT3 p. 777	(Gems IV p. 83); (Gems V p. 375); (GTCG p. 639); Assignment; 3DG p. 445; GTCG p. 639; TGS; RTCD p. 169; Moller; RT2 p. 596; RT3 p. 771		axis-aligned bounding box	
oriented bounding box	IRT p. 104; Gems II p. 247; GTWeb; Gomes; GTCG p. 630; TGS; RTCD p. 179; RT2 p. 572; RT3 p. 743	GTWeb; Gomes; GTCG p. 635; TGS; RTCD p. 161; RT2 p. 588; RT3 p. 757	TGS; RTCD p. 132, 166	TGS	GTWeb; TGS	(Gems V p. 378); GTWeb; Gomes; GTCG p. 639; TGS; RTCD p. 602; RT3 p. 777	GTWeb; Gomes; GTCG p. 639; TGS; RTCD p. 602; RT3 p. 777		(Gems IV p. 83)	oriented bounding box	
viewing frustum	(IRT p. 104; Gems II p. 247)		GTWeb; Assignment; GPG p. 380; 3DG p. 302; RTCD p. 609; RT3 p. 774	GPG p. 380		homogeneous clipping; Gems p. 84	(Gems IV p. 83)	(Gems IV p. 83)	(Gems IV p. 83)	viewing frustum	
convex polyhedron	IRT p. 104; Gems II p. 247; Gems p. 388; Panzica (pt 8)(4); RTCD p. 198; SoftSurfer		3DG p. 462; RTCD p. 142		generalized clipping	Gems IV p. 74; Gems V p. 378	(Gems IV p. 83)	(Gems IV p. 83)	(Gems IV p. 83)	convex polyhedron	

Raytracing Pseudocode

```

▶ for ( y = 0; y < height; y++ ) {
    for ( x = 0; x < width; x++ ) {
        u = 1 + (r-1) * (x+0.5) / width;
        v = t + (b-t) * (y+0.5) / height;
        s = ...;
        d = normalize( s );
    }

    // finde nächsten Schnittpunkt
    intersection = NULL;
    float t = FLOAT_MAX;

    for ( each object ) {
        t' = intersect( object, e, d );
        if ( t' > 0 && t' < t ) {
            intersection = object;
            t = t';
        }
    }
    ...
}
}

```



▶ Achtung: uns interessieren nur Schnittpunkte vor der Kamera ($t' > 0$), und von diesen der nächste in Strahlrichtung ($t' < t$)

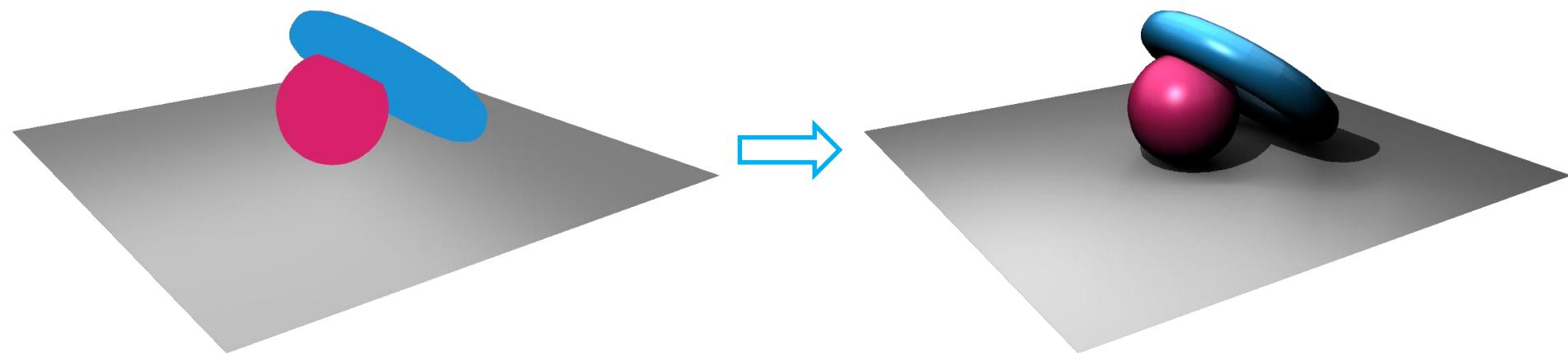
Raytracing ist
„embarrassingly parallel“:

#pragma omp parallel for

typische Parallelisierung:
Multithreading über der Bildebene
und SIMD/Vektorisierung innerhalb

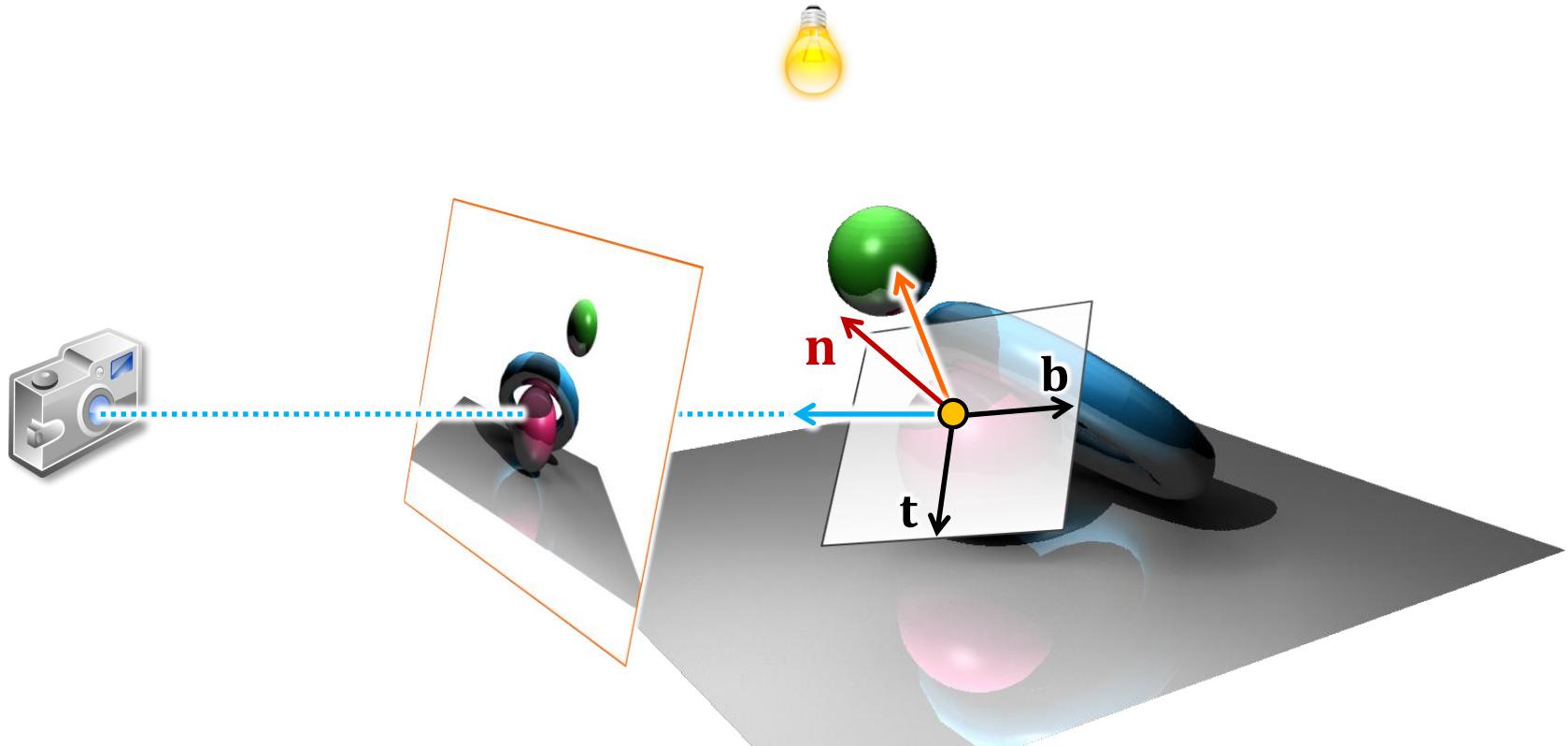
Schritte

- ▶ Erzeugung der Sichtstrahlen durch jeden Pixel (ray generation)
- ▶ Schnittberechnung (ray casting):
finde Dreieck, das den Sichtstrahl am nächsten zur Kamera schneidet
- ▶ Schattierung und Beleuchtungsberechnung (shading)
- ▶ Sekundärstrahlen für Spiegelung und Transmission



Schritte

- ▶ Erzeugung der Sichtstrahlen durch jeden Pixel (ray generation)
- ▶ Schnittberechnung (ray casting):
finde Dreieck, das den Sichtstrahl am nächsten zur Kamera schneidet
- ▶ Schattierung und Beleuchtungsberechnung (shading)
- ▶ Sekundärstrahlen für Spiegelung und Transmission



Materialeigenschaften

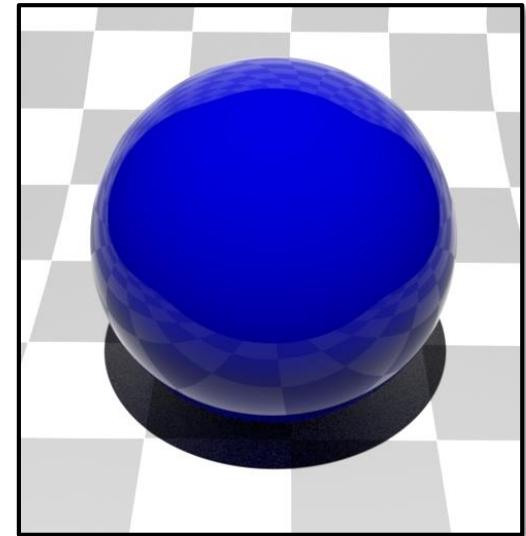
- ▶ woher kommt das Aussehen von Materialien?
- ▶ Schattierung ist essentiell für dreidimensionalen Eindruck und ist ein Ergebnis von
 - ▶ Emission der Lichtquellen (Intensität, Farbe, Position, ...)
 - ▶ Oberflächeneigenschaften (Material, Rauheit, Orientierung zur LQ, ...)



Licht-Material-Interaktion

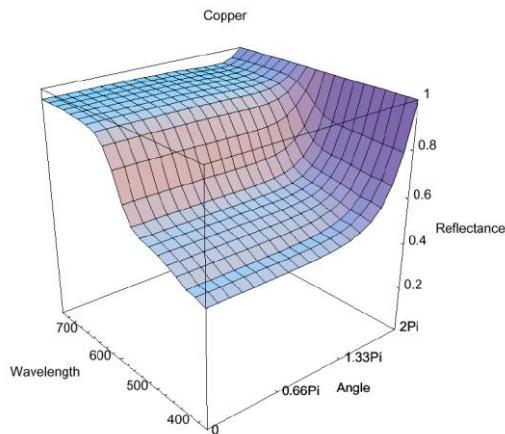
- was passiert, wenn Licht auf eine Oberfläche trifft?
 - ▶ ein Teil wird reflektiert
 - ▶ ein Teil dringt ins Material ein
 - ▶ das Verhältnis ist abhängig vom Einfallswinkel und von den Brechungsindizes beider Materialien (sog. Fresnel-Effekt)

Bilder: Lafortune

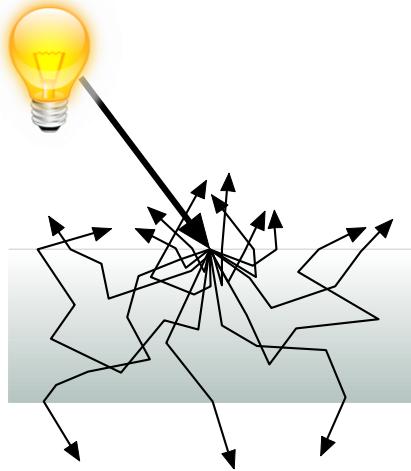
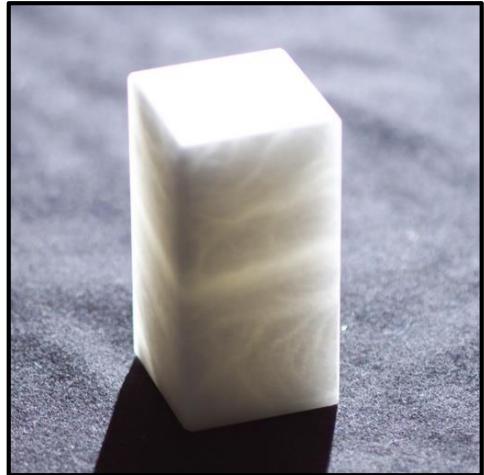


Licht-Material Interaktion

- eindringendes Licht wird entweder (sehr schnell) absorbiert: Metalle



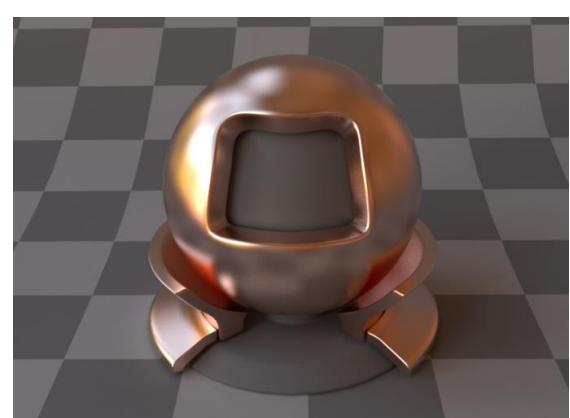
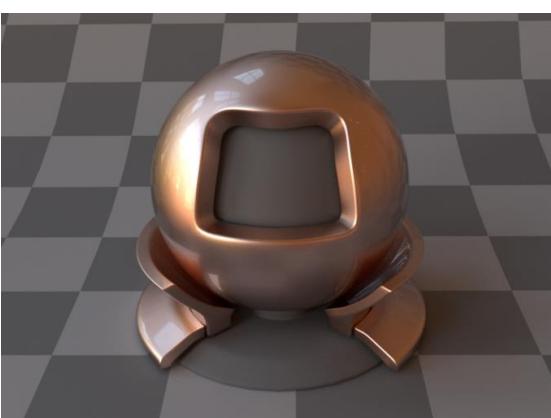
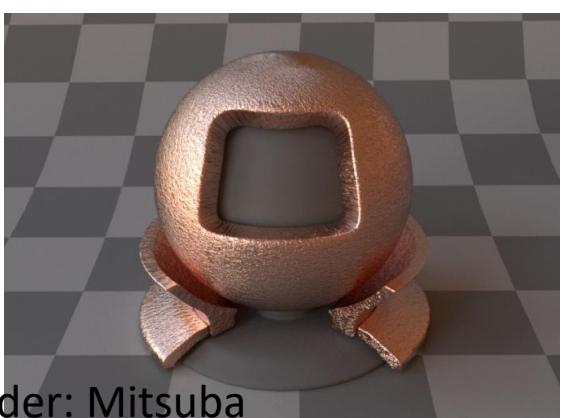
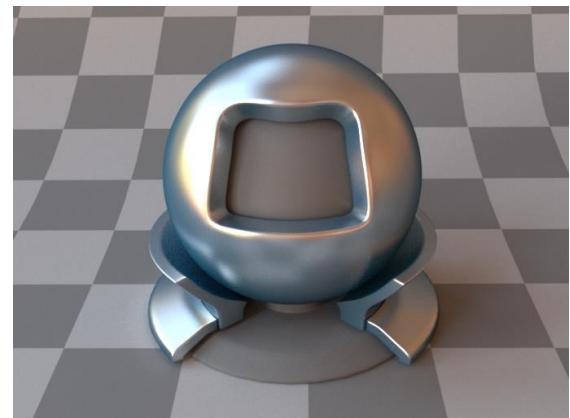
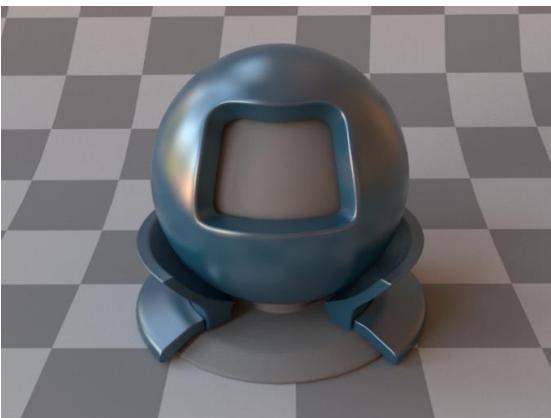
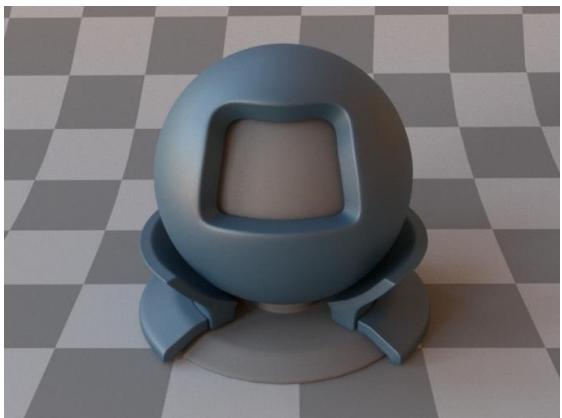
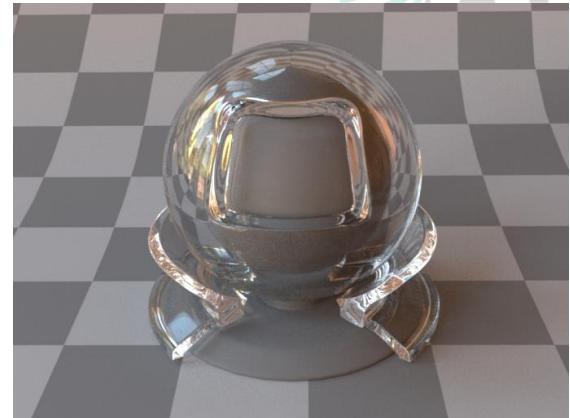
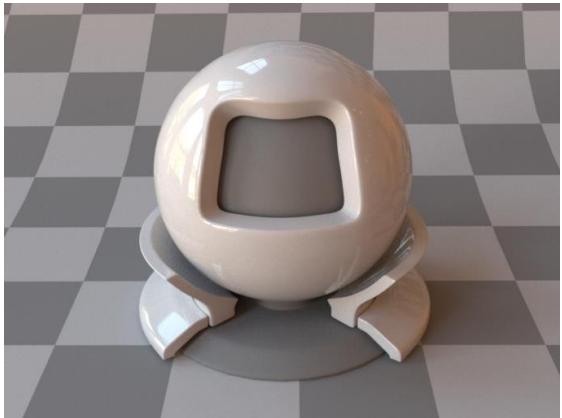
- ...oder verlässt das Material an anderer Stelle wieder



www.wikipedia.de

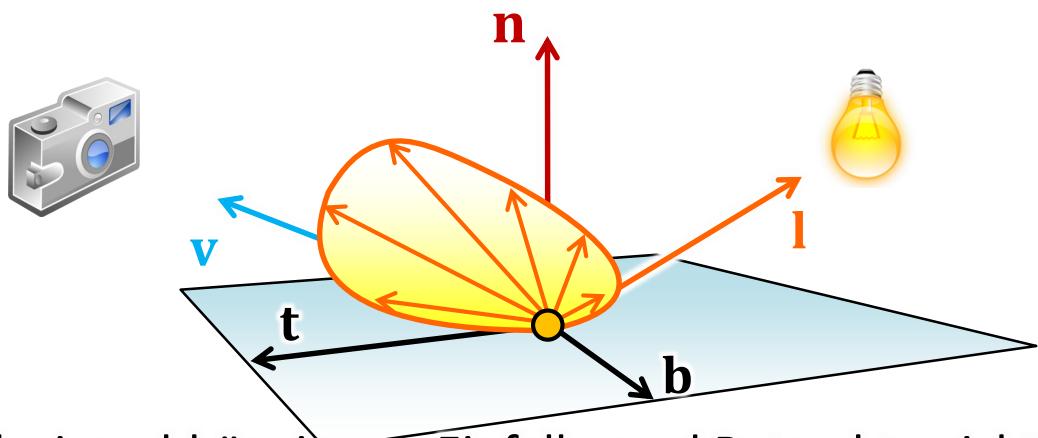
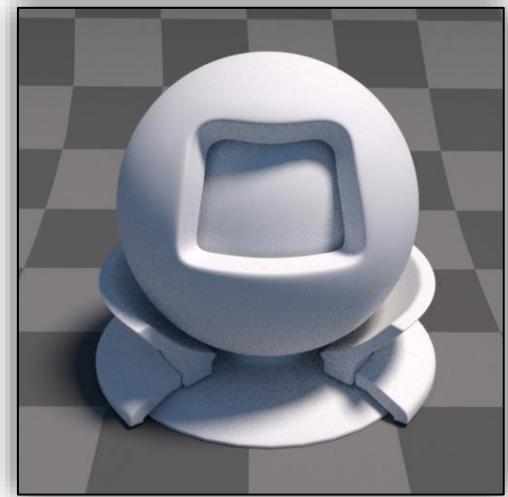
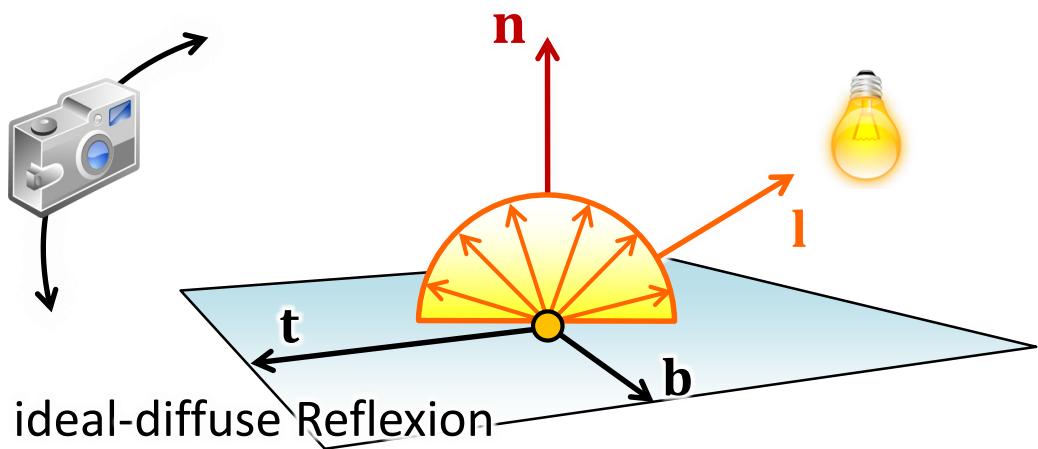
- wir befassen uns im Folgenden nur mit der Reflexion an der Oberfläche!

Mehr als nur Oberflächenreflexion

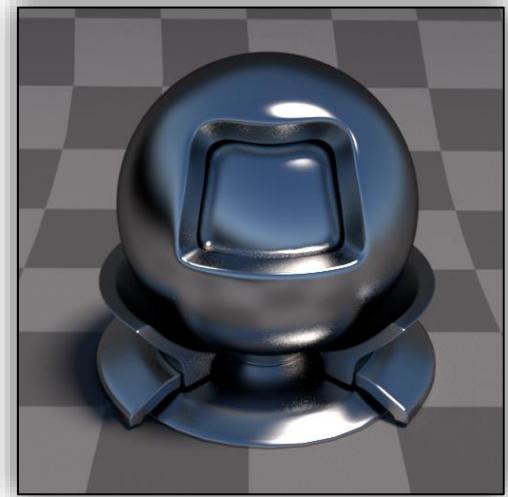


Einführung: Lokales Beleuchtungsmodell

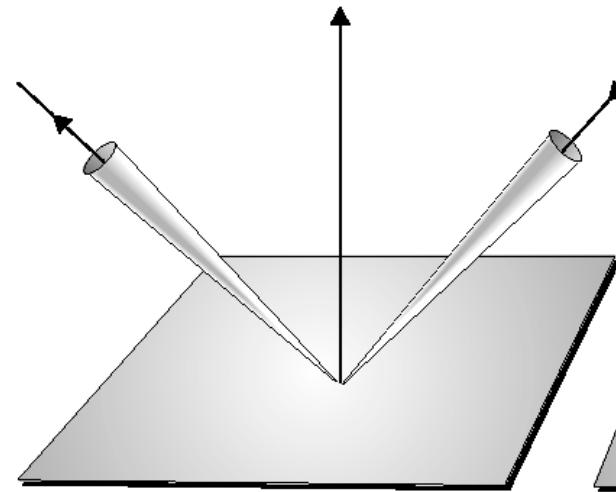
Beispiele



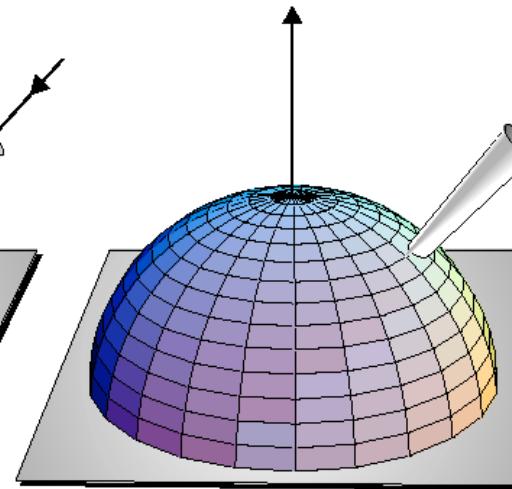
Reflexion abhängig von Einfalls- und Betrachterrichtung



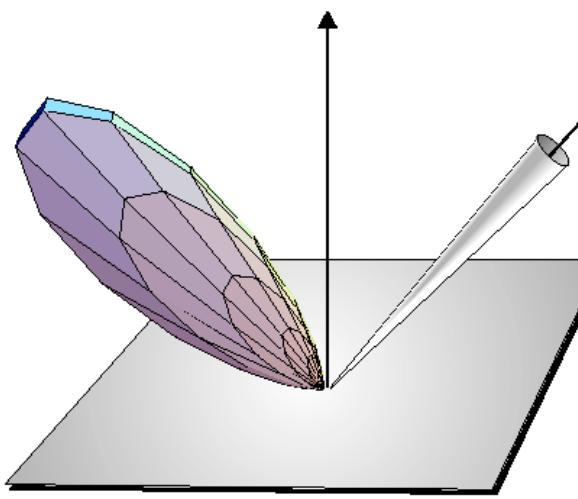
Begriffe und Darstellung



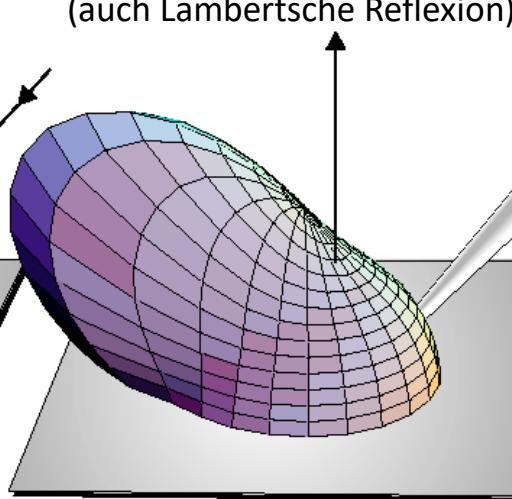
perfekt spiegelnd (engl. specular)



ideal-diffuse
(auch Lambertsche Reflexion)



glänzend, imperfect spiegelnd
(engl. glossy, seltener rough specular)



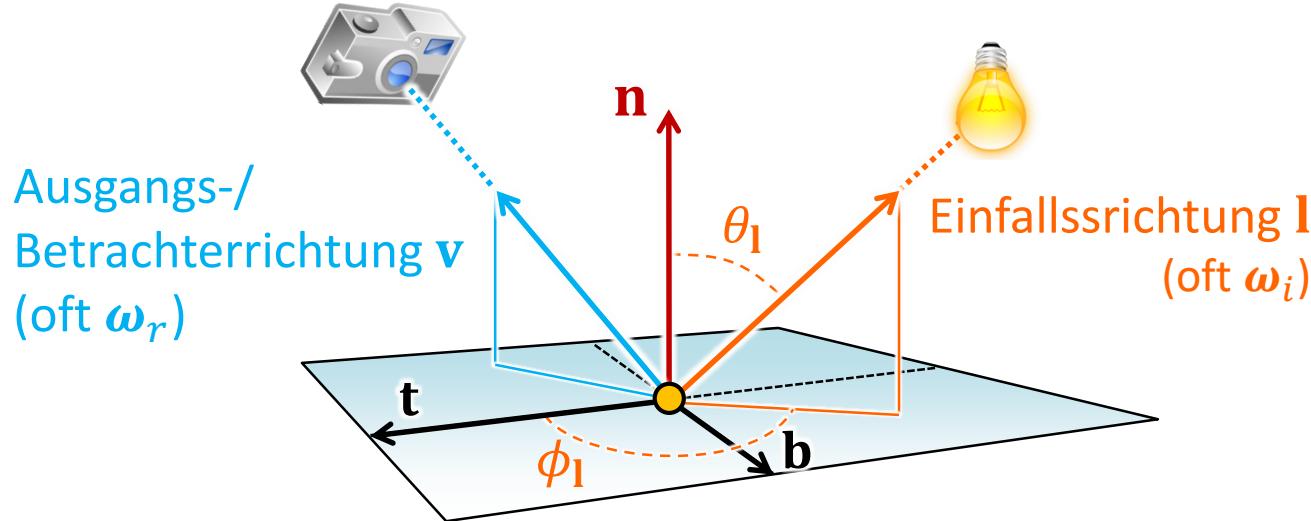
directional diffuse
(selten verwendeter Begriff)



Einführung: Lokales Beleuchtungsmodell

Bidirektionale Reflektanzverteilungsfunktion

- ▶ ... beschreibt Reflexionsverhalten von Oberflächen
(engl. Bidirectional Reflectance Distribution Function, BRDF)
- ▶ um BRDFs unabhängig von einem bestimmten Oberflächenpunkt
(und seiner Orientierung) anzugeben, verwendet man ein
Referenzkoordinatensystem



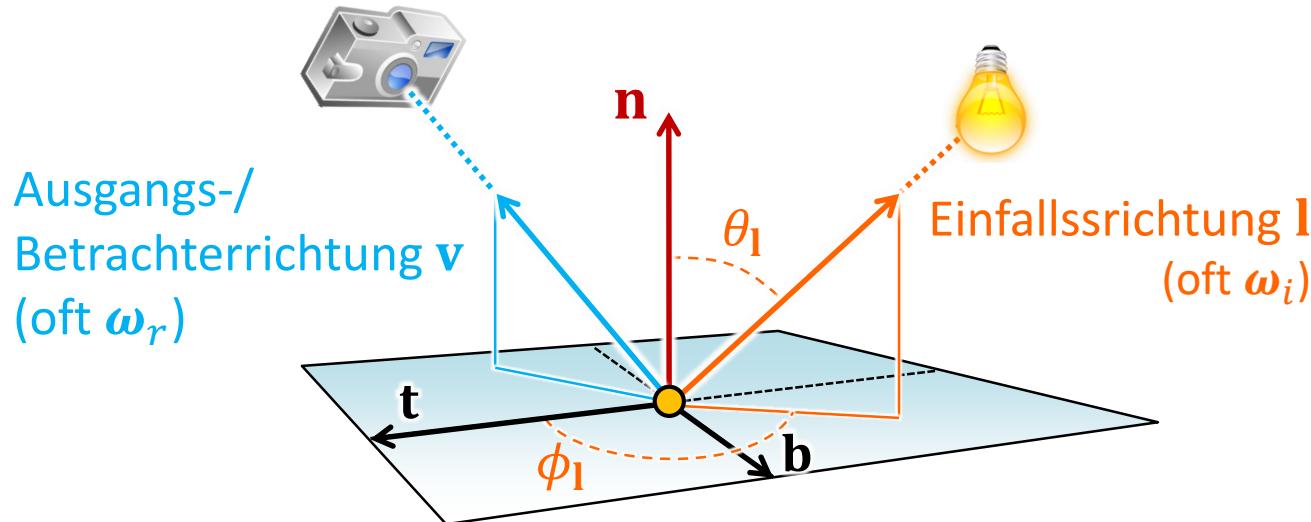
Einführung: Lokales Beleuchtungsmodell

Bidirektionale Reflektanzverteilungsfunktion

- ▶ ... beschreibt Reflexionsverhalten von Oberflächen
(engl. Bidirectional Reflectance Distribution Function, BRDF)
- ▶ (Referenz-)Koordinatensystem mit den aufspannenden Vektoren Tangente **t**, Bitangente **b** und Normale **n** am Oberflächenpunkt **x**

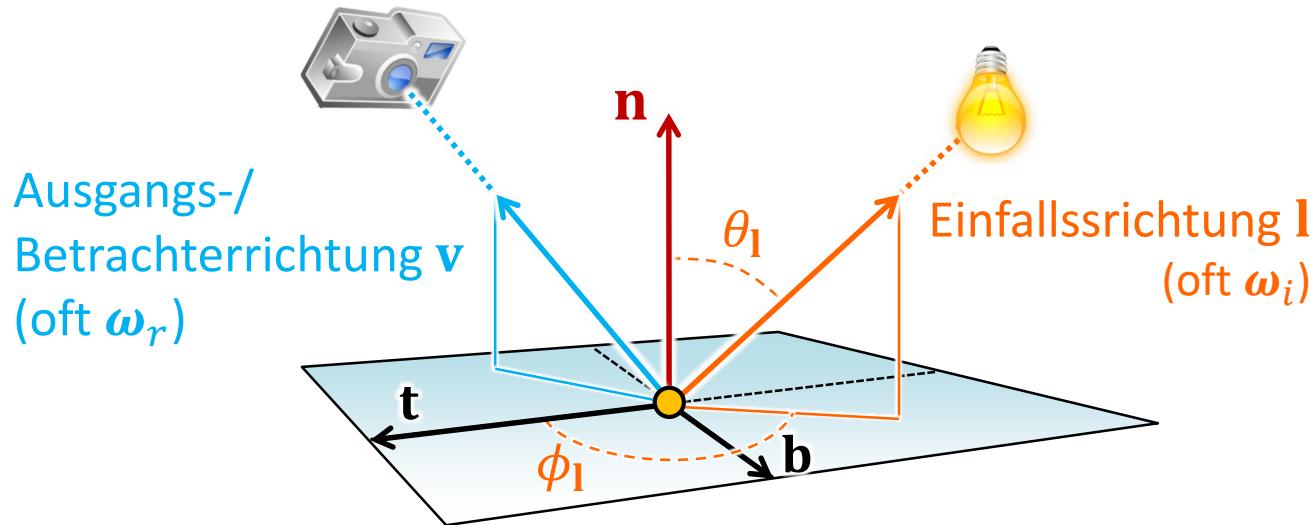
$$f_r(\mathbf{l}, \mathbf{x}, \mathbf{v}) = f_r(\theta_l, \phi_l, \mathbf{x}, \theta_v, \phi_v) \quad [\text{sr}^{-1}]$$

mit Polarwinkel $0 \leq \theta_l, \theta_v \leq \pi/2$ und Azimutalwinkel $0 \leq \phi_l, \phi_v < 2\pi$



Bidirektionale Reflektanzverteilungsfunktion

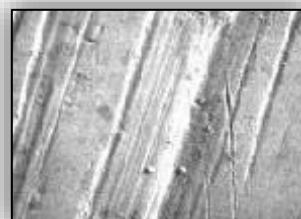
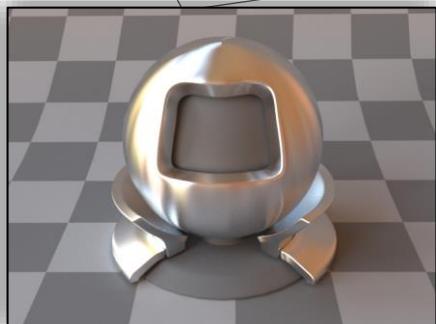
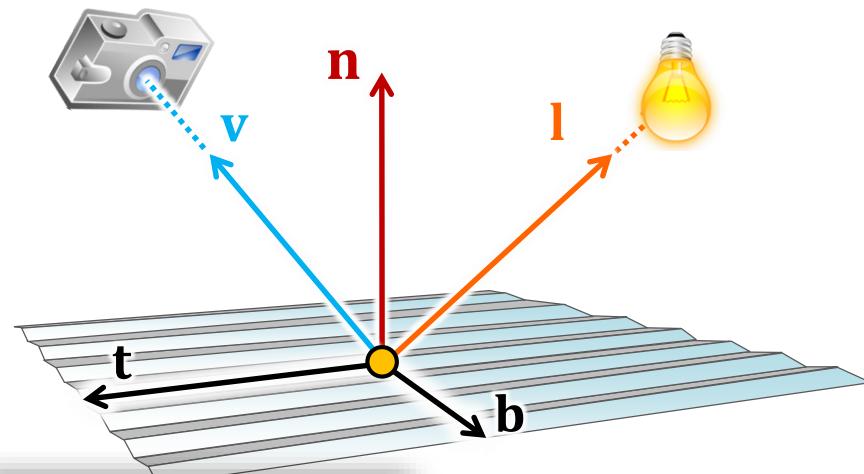
- ▶ ... beschreibt Reflexionsverhalten von Oberflächen
(engl. Bidirectional Reflectance Distribution Function, BRDF)
- ⚠ nur oberflächliche Erklärung (keine Radiometrie)
 - ▶ Radiance (Strahldichte) die Oberfläche in Richtung **v** verlässt, zu
 - ▶ einfallender Irradiance (Flussdichte [W/m^2]) aus Richtung **I**
 - ▶ merken Sie sich vorläufig einfach nur:
„Verhältnis von ausgehendem zu einfallendem Licht“



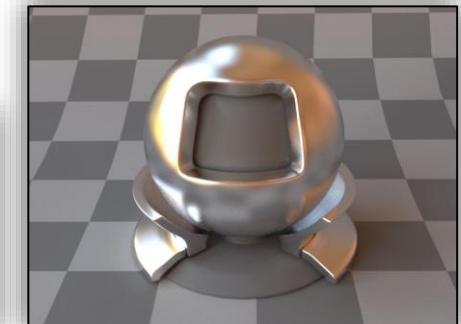
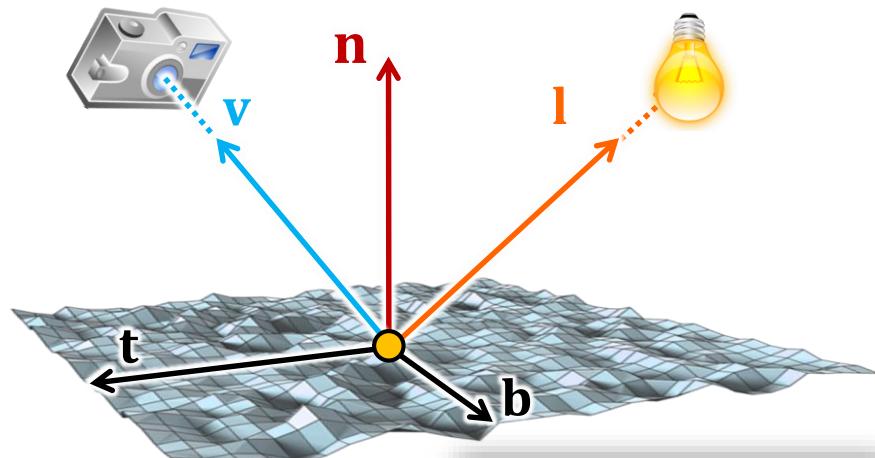
Einführung: Lokales Beleuchtungsmodell

Bidirektionale Reflektanzverteilungsfunktion

- ▶ **anisotrope BRDF (4D ohne Ort, links):** Reflexion abhängig von der Rotation um die Normale aufgrund orientierter Mikrostrukturen
- ▶ **isotrope BRDF (3D ohne Ort, rechts):** Rotationsinvarianz um die Normale
$$f_r(\theta_l, \phi_l, \mathbf{x}, \theta_v, \phi_v) = f_r(\theta_l, \phi_l + \Delta\phi, \mathbf{x}, \theta_v, \phi_v + \Delta\phi)$$
 - ▶ Achtung: die Differenz $\phi_l - \phi_v$ ist deswegen nicht irrelevant!



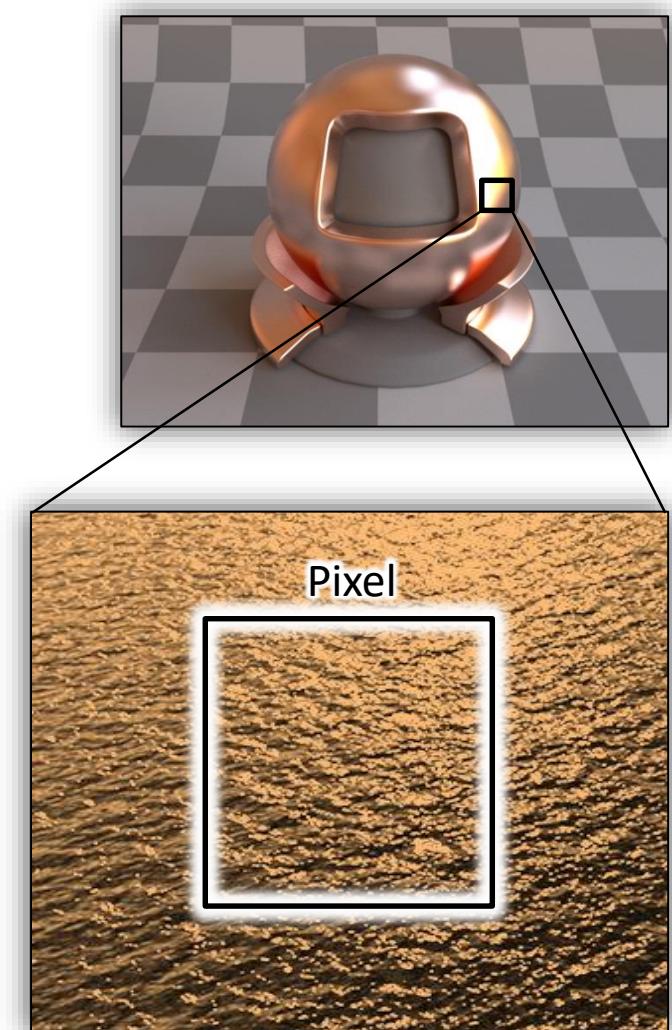
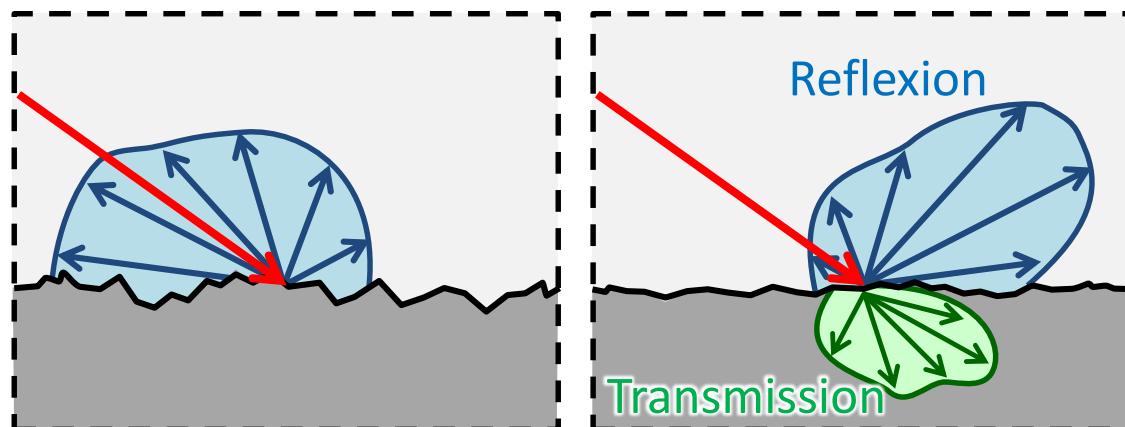
mikroskop.
Aufnahmen



Einführung: Lokales Beleuchtungsmodell

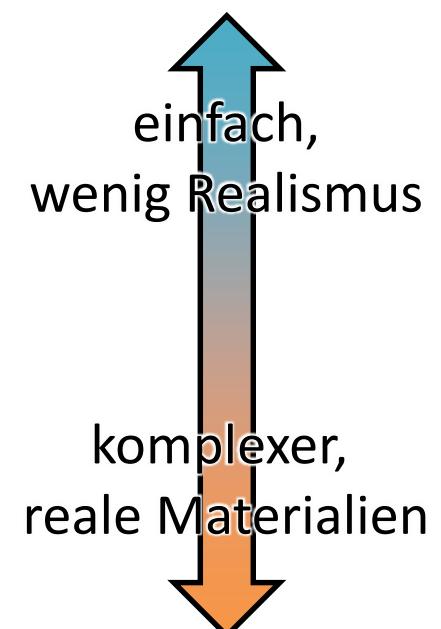
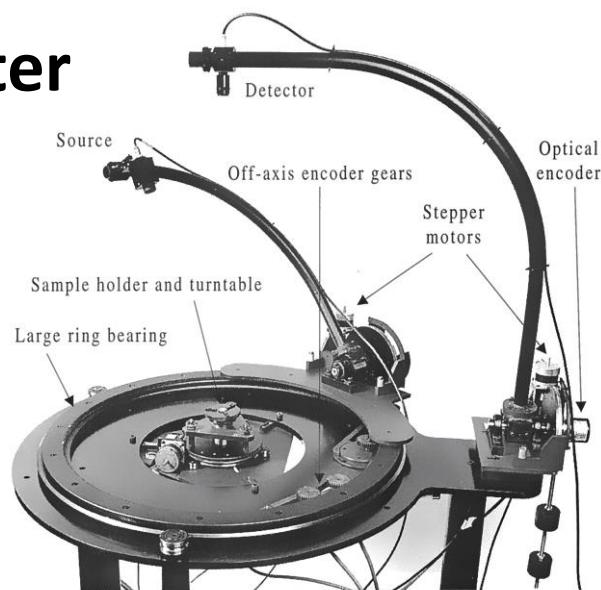
BRDFs und Skalen

- mit einer BRDF beschreiben wir immer den Einfluss von Oberflächenstrukturen auf die Reflexion, die so klein sind, dass wir sie nicht geometrisch/anderweitig modellieren
- analog für Transmission: BTDF = Bidirectional Transmission Distribution Function
- Bidirectional Scattering Distribution Function bezeichnet BRDF und BTDF zusammen



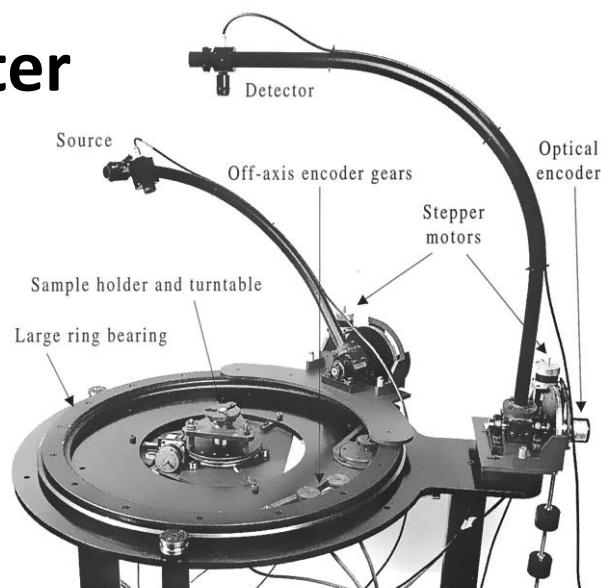
Messung von BRDFs – Gonioreflektometer

- ▶ Messung realer Materialproben ist aufwändig
- ▶ Modelle dagegen sind kompakt, rauschfrei und können an Messdaten angepasst werden
- ▶ phänomenologische/empirische Modelle
 - ▶ wenige, intuitiv verständliche Parameter
 - ▶ keine strengen physikalischen Gesetzmäßigkeiten
 - ▶ oft ungeeignet Messdaten zu repräsentieren
- ▶ physikalisch-basierte Modelle
 - ▶ geeignet reale Materialien nachzubilden
 - ▶ Parameterwahl erfordert Verständnis des Modells
 - ▶ z.B. Mikrofacetten Modelle



Messung von BRDFs – Gonioreflektometer

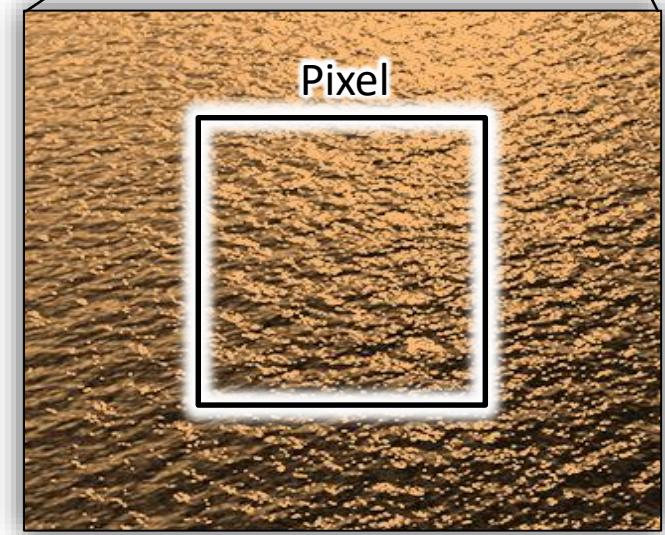
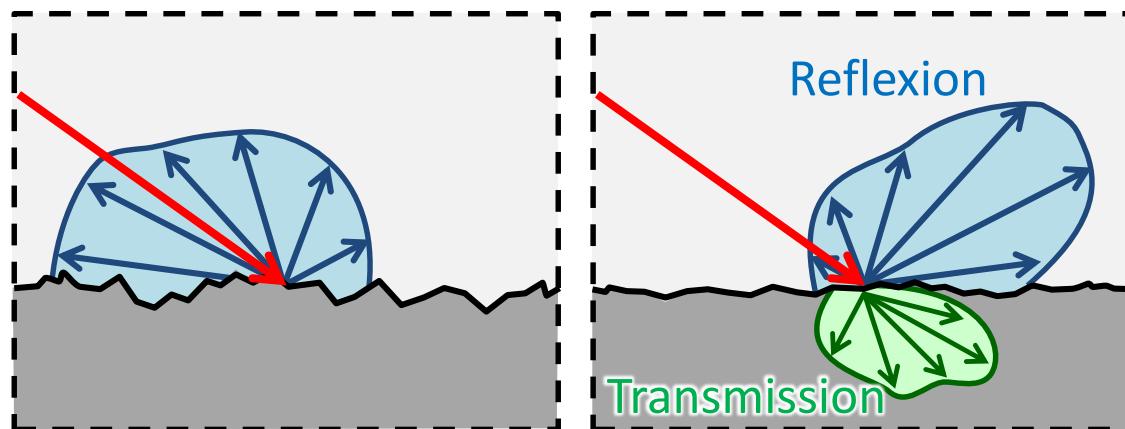
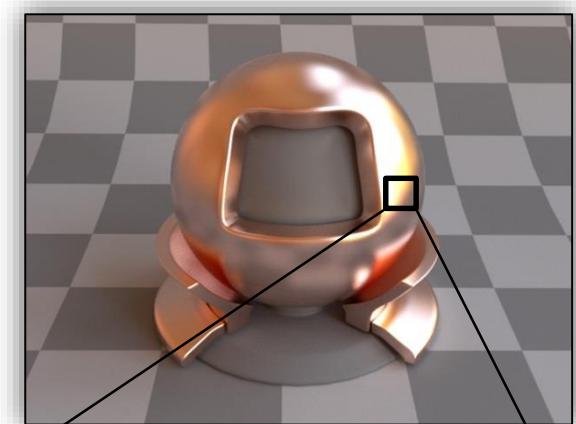
- ▶ Messung realer Materialproben ist aufwändig
- ▶ Modelle dagegen sind kompakt, rauschfrei und können an Messdaten angepasst werden
- ▶ phänomenologische/empirische Modelle
Phong [75], Blinn-Phong [77],
Ward [92], Lafourture et al. [97],
Ashikhmin et al. [00], ...
- ▶ physikalisch-basierte Modelle
Mikro-Facetten Modelle (Cook-Torrance [81], Walter et al. [07])
Multiple-Scattering Microfacet BSDFs..., Heitz et al. 2016
A Practical Extension to ... Iridescence, Belcour et al. 2017
und viele mehr!



Exkurs: Modellierung realistischer Materialien

Mikrofacetten-Streumodelle für Oberflächen

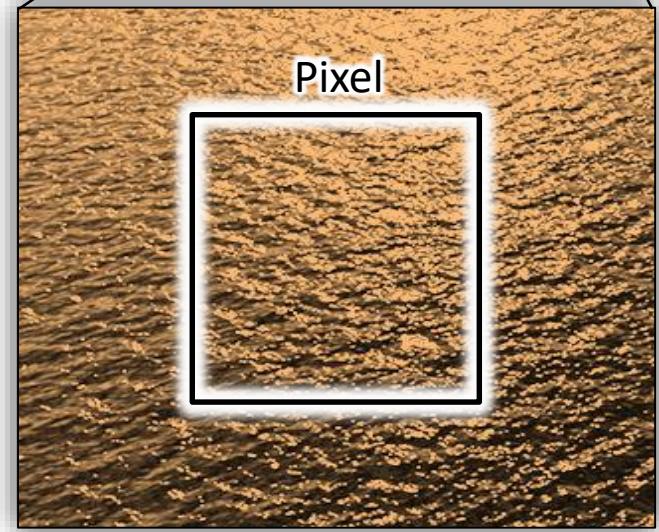
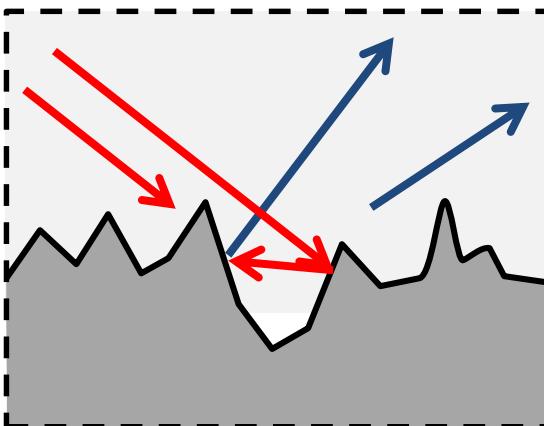
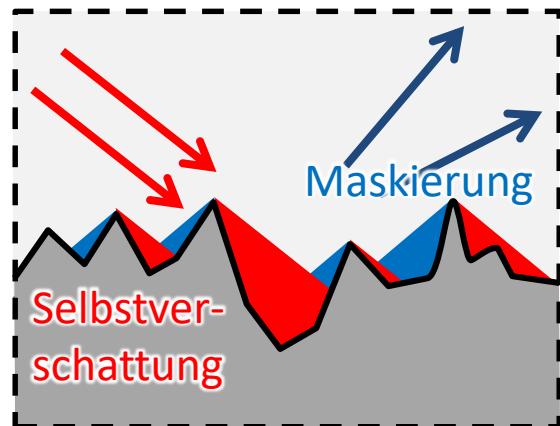
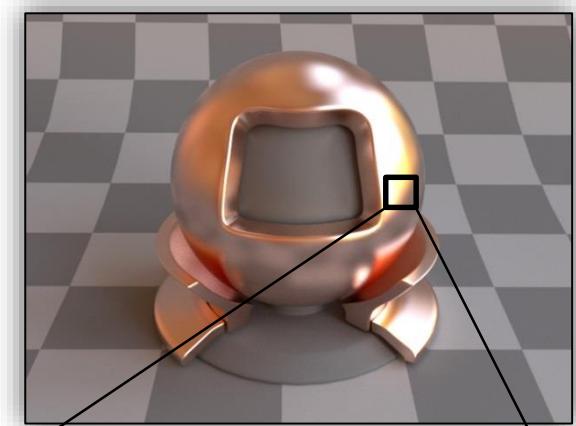
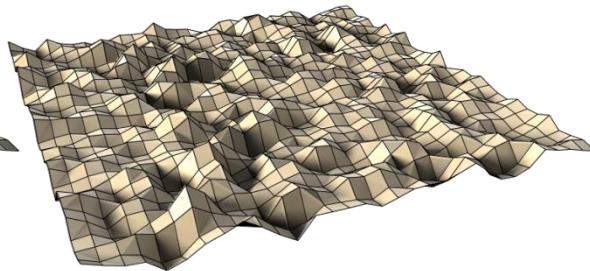
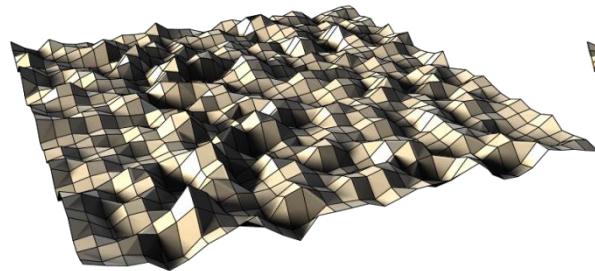
- Vorstellung: Oberfläche zusammengesetzt aus vielen kleinen planaren, spiegelnden oder reflektierenden Flächen



Exkurs: Modellierung realistischer Materialien

Mikrofacetten-Streumodelle für Oberflächen

- Reflexionseigenschaften maßgeblich durch Rauheit bestimmt: spechere Verteilung der Orientierungen der Mikrofacetten



Multiple-Scattering Microfacet BSDFs with the Smith Model

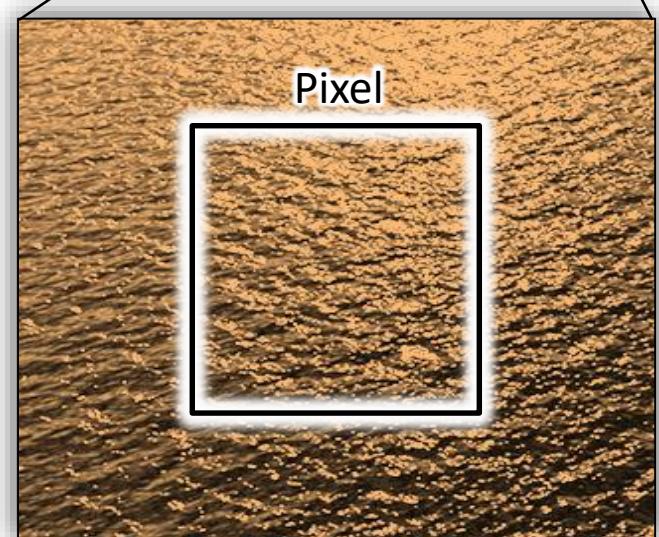
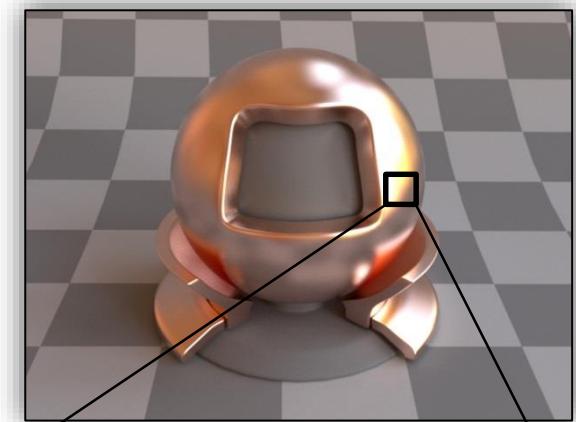
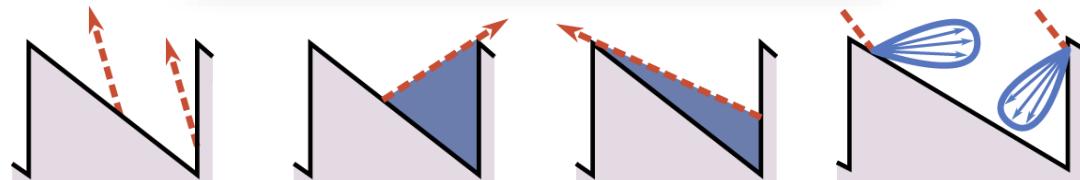
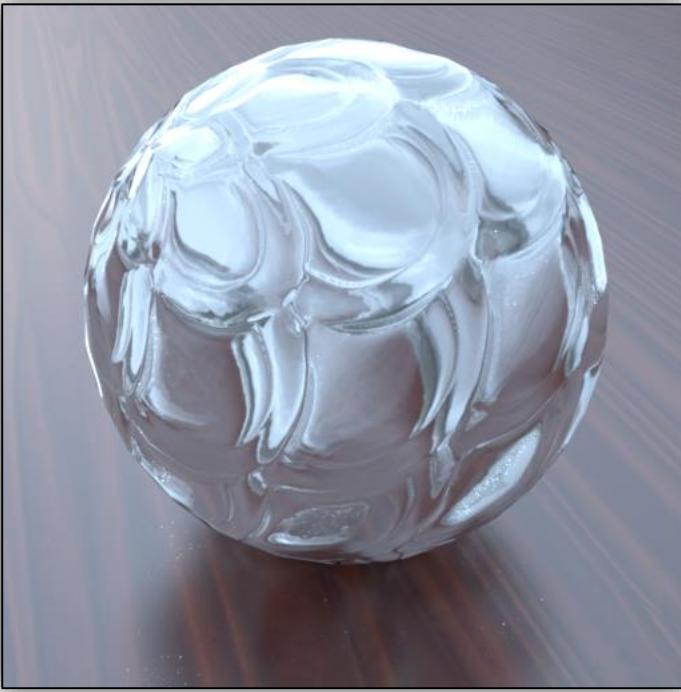
Heitz, Hanika, d'Eon, Dachsbacher

ACM Transactions on Graphics (Proc. of SIGGRAPH), 2016

Exkurs: Modellierung realistischer Materialien

Mikrofacetten-Streumodelle für Oberflächen

Mesoskopisches Detail



Pixel

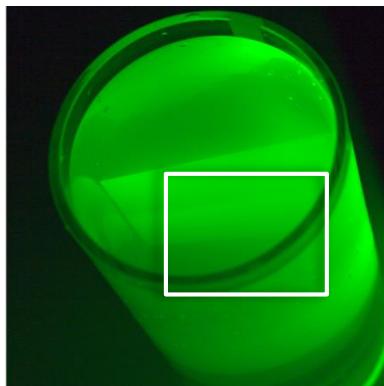
Microfacet-based normal mapping for robust Monte Carlo path tracing

Schüssler, Heitz, Hanika, Dachsbacher

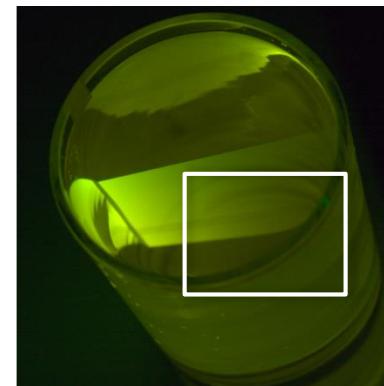
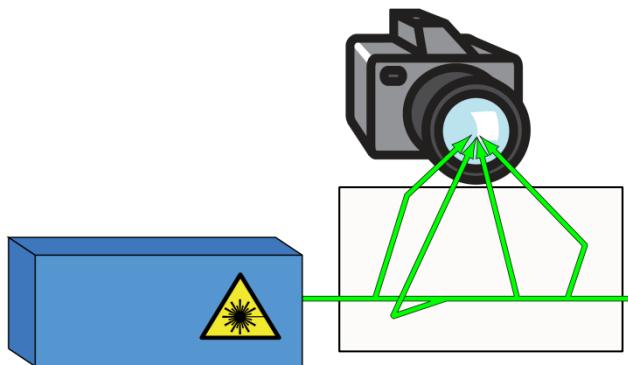
ACM Transactions on Graphics (Proc. of SIGGRAPH Asia), 2017

Exkurs: Motivation phänomenologische Modelle

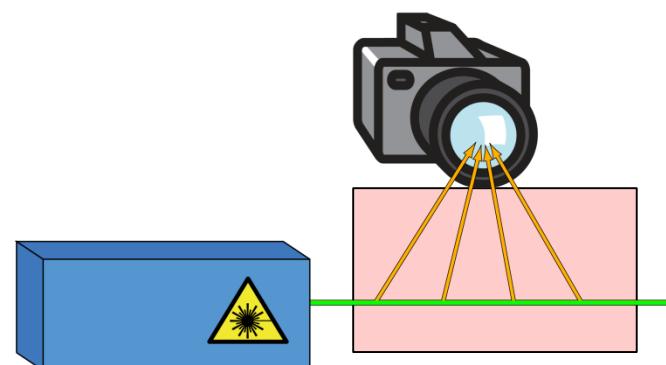
- wie kann man Lichtstrahlen sichtbar machen?



verdünnte Milch:
mehrfahe Streuung

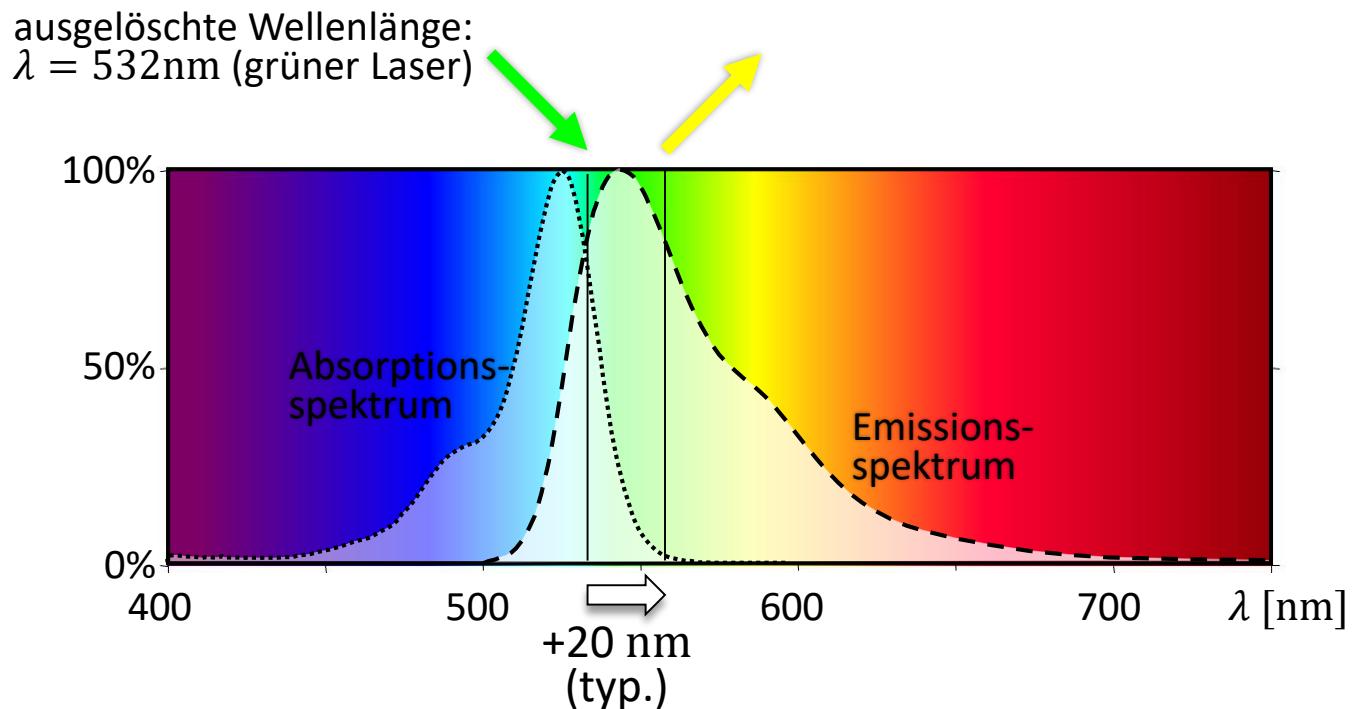


fluoreszierende Lösung:
einfache Streuung



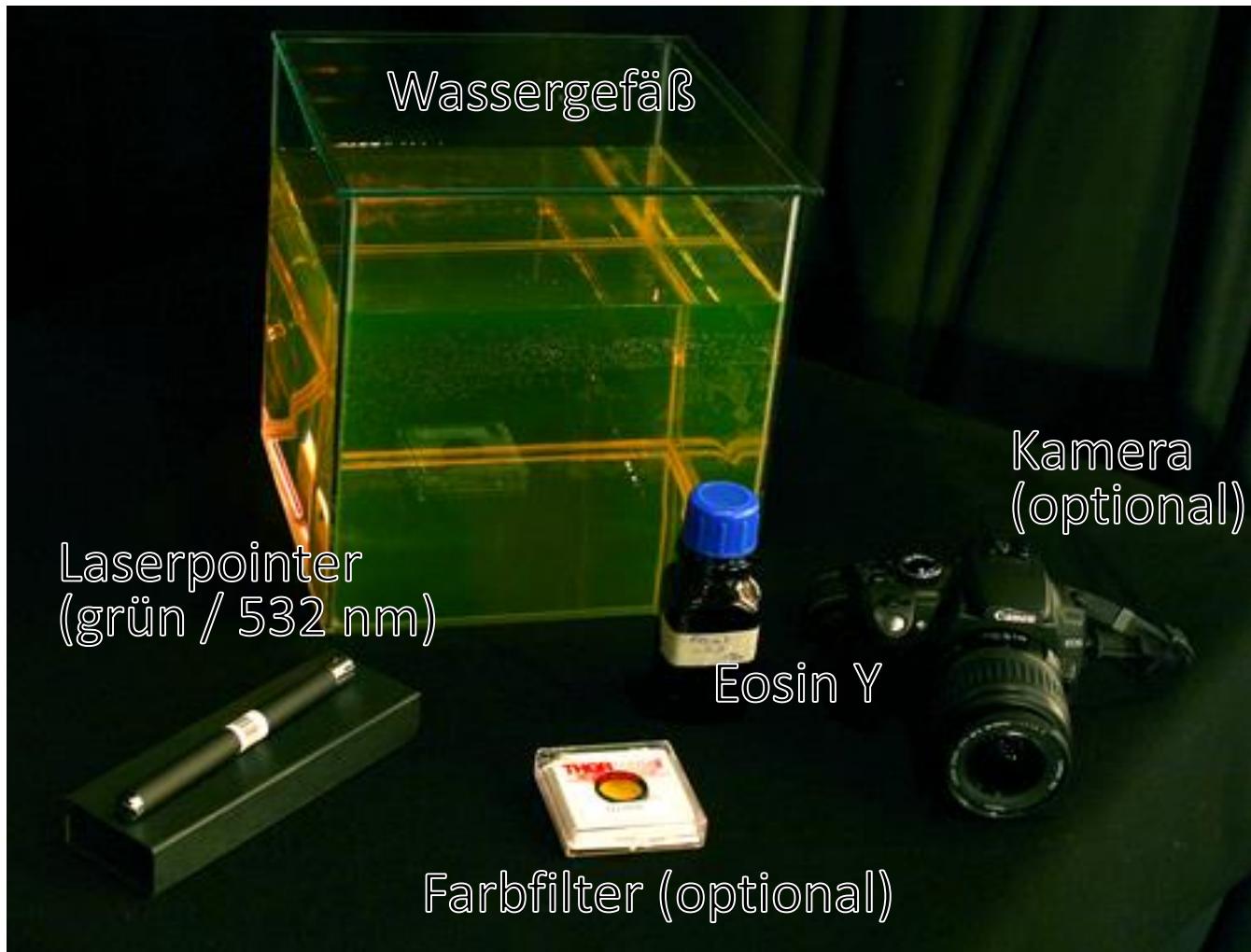
Exkurs: Fluoreszenz

- Absorptions- und Emissionsspektrum eines fluoreszierenden Farbstoffs (Eosin Y, Y=„yellowish“)



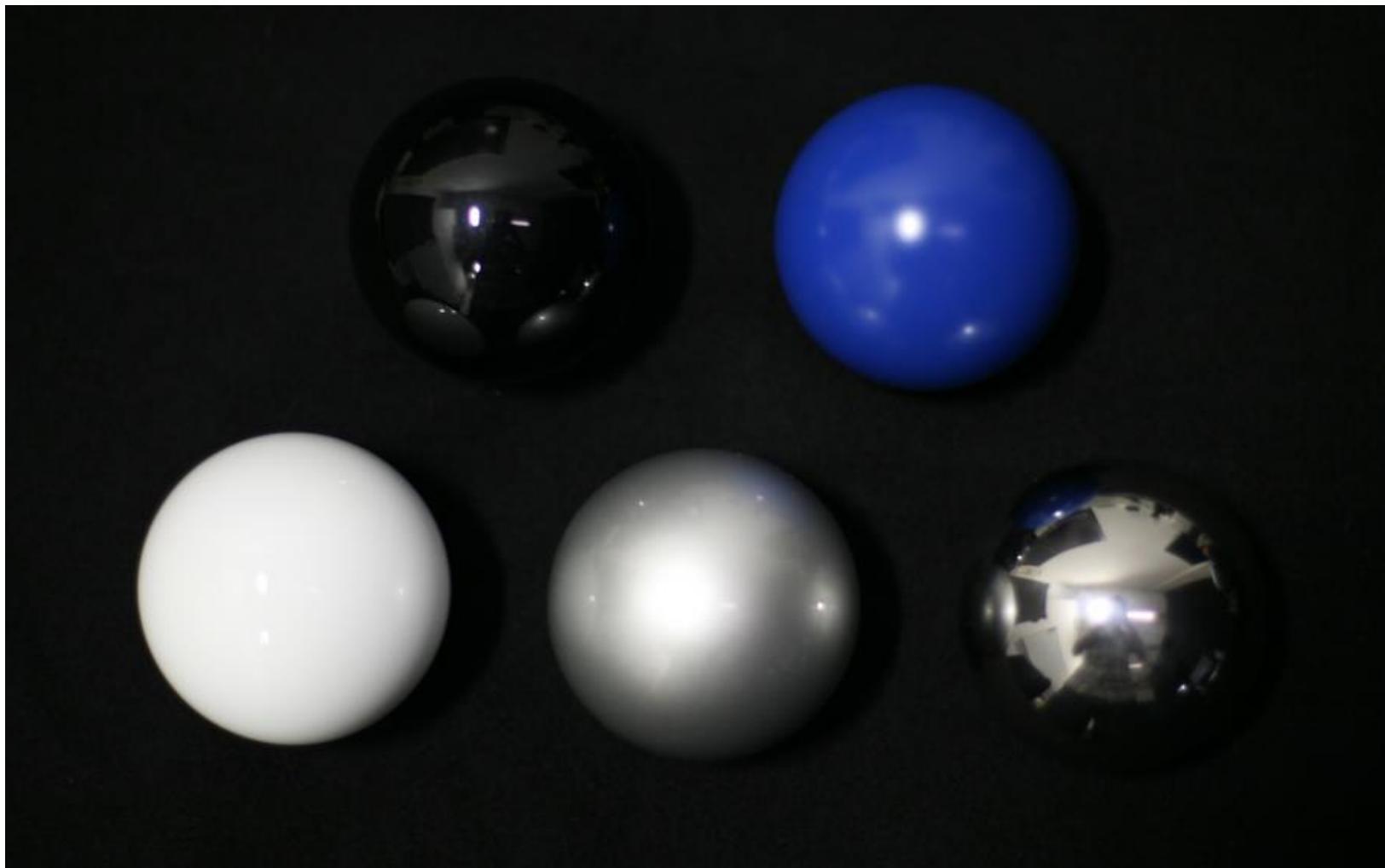
- ein Streu-/Reemissionsvorgang ändert die Wellenlänge des Photons: (praktisch) nur einfache Streuung möglich

Exkurs: Gerätschaften

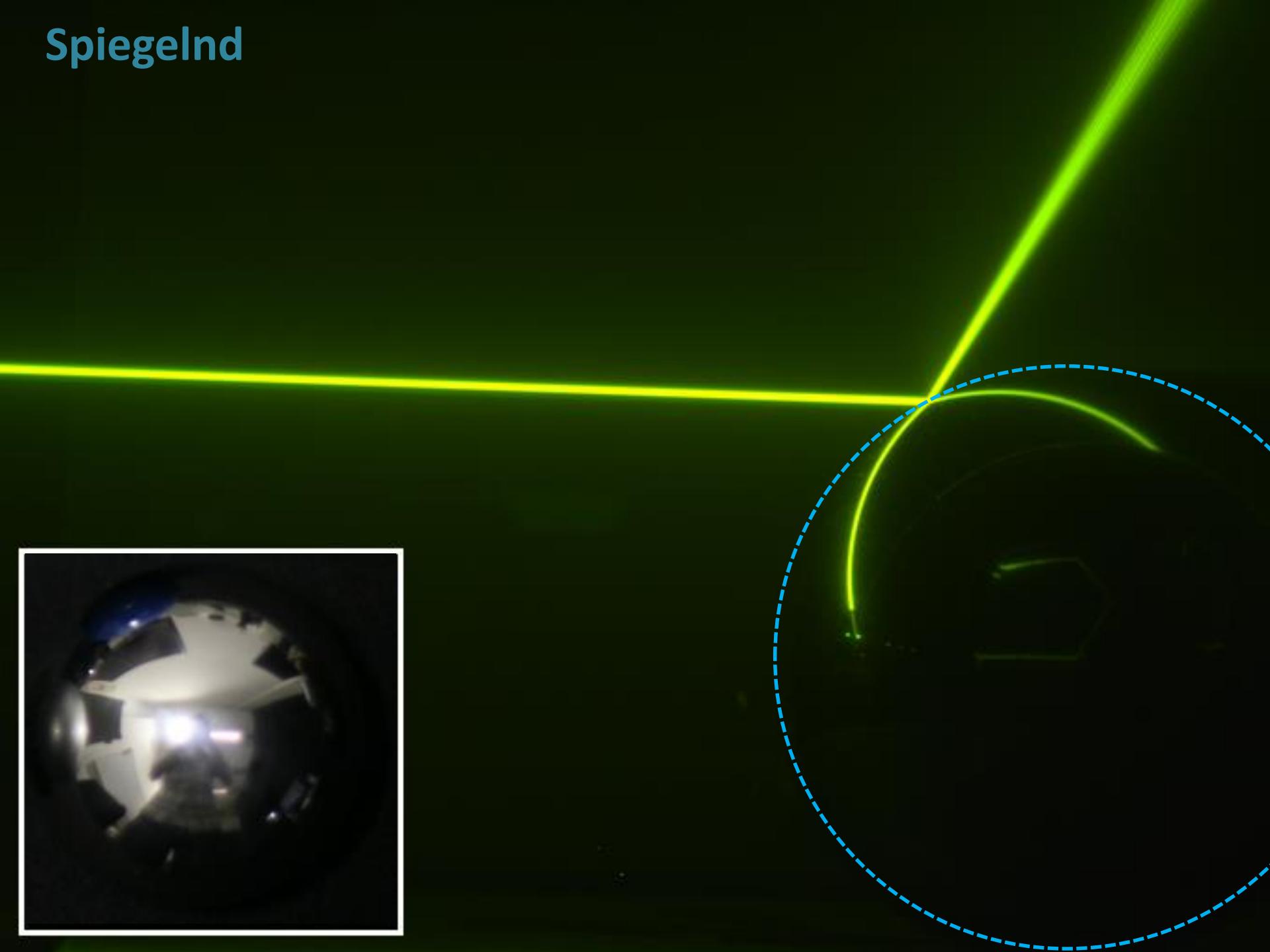


Hullin et al., **Direct Visualization of Real-World Light Transport**
(VMV 2008)

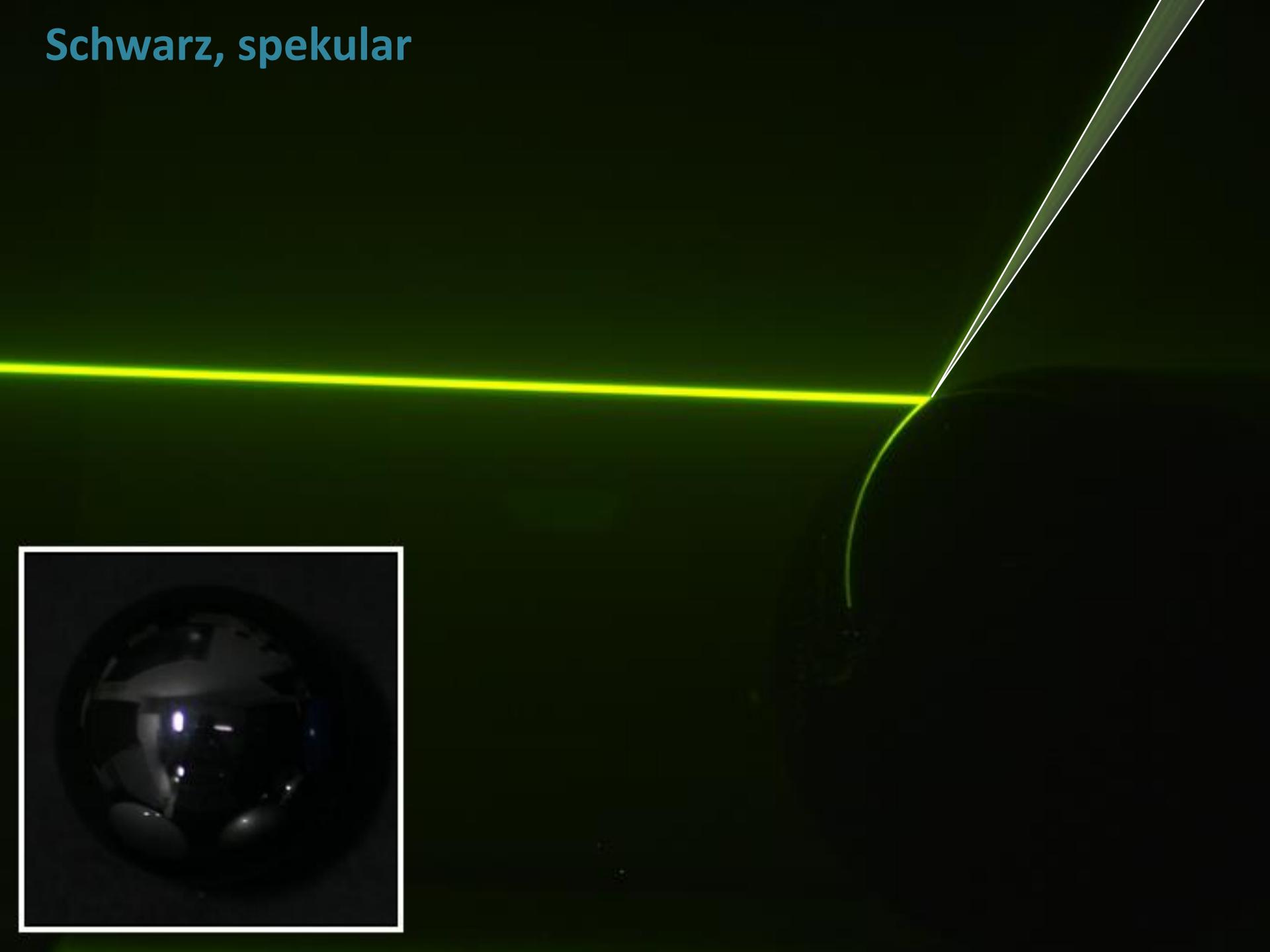
Exkurs: Oberflächenreflexion



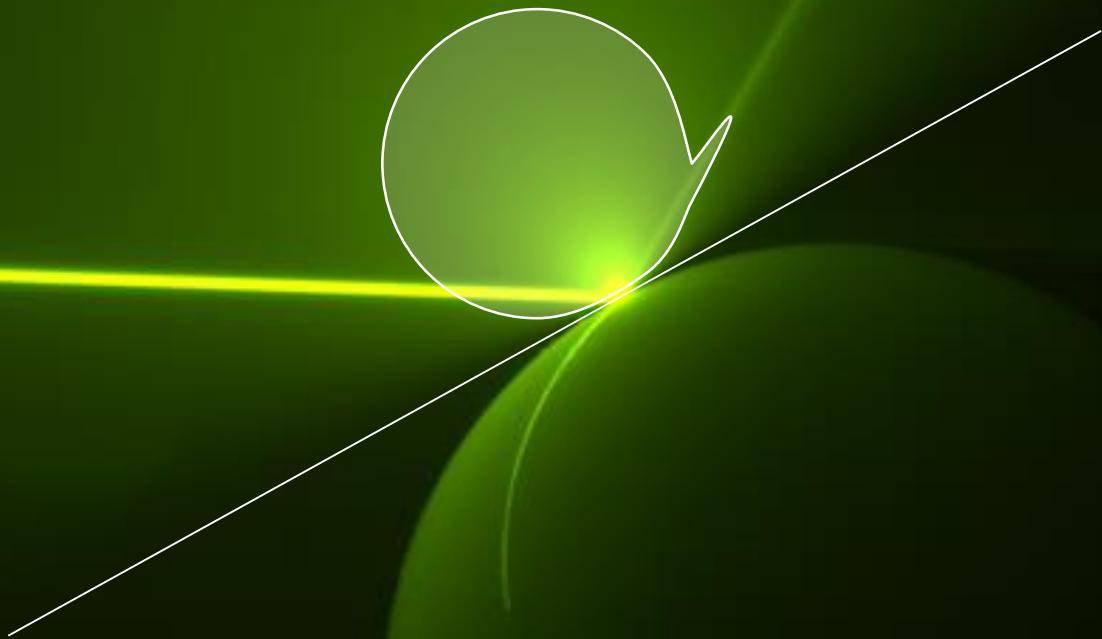
Spiegelnd



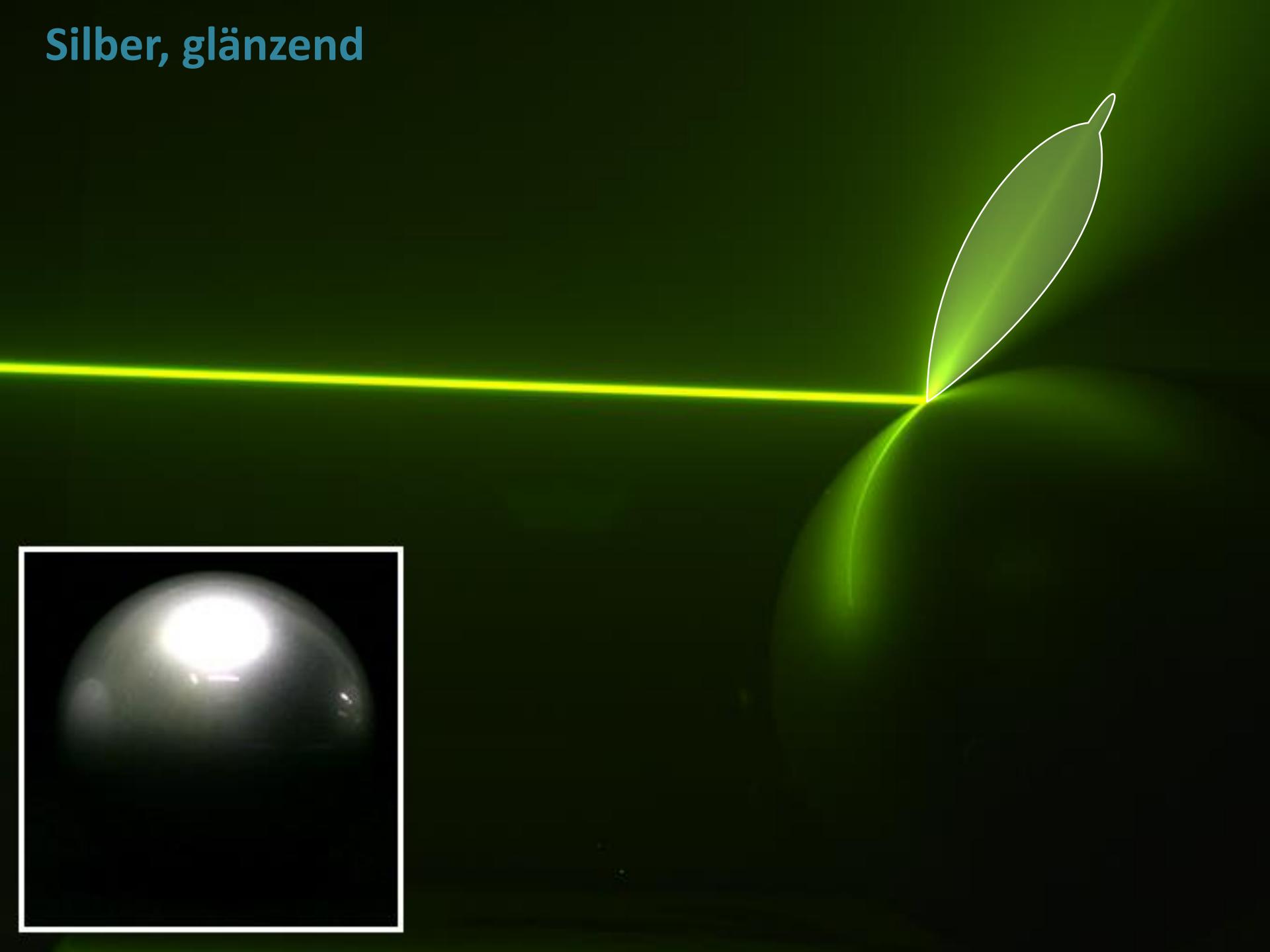
Schwarz, spekular



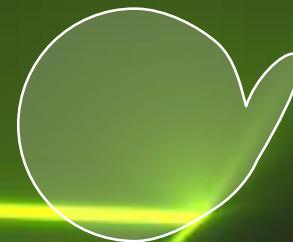
Diffus weiß, mit Lackschicht



Silber, glänzend

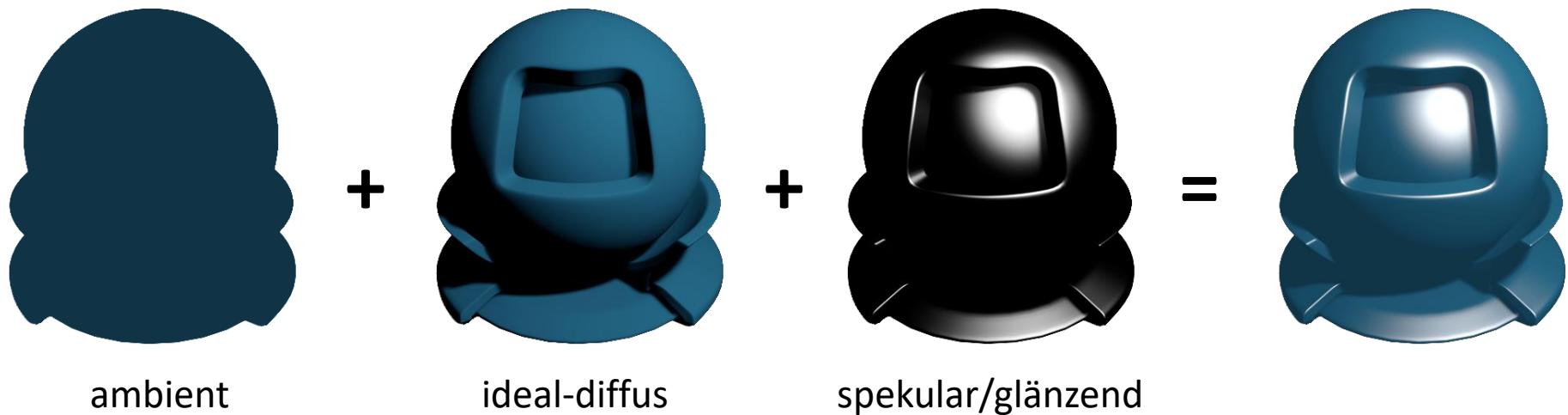


Diffus und glänzend



Phong-Beleuchtungsmodell ¹

- ... ist ein phänomenologisches Modell und modelliert die Reflexion mit drei Komponenten
 - ▶ **ambient**: indirekte Beleuchtung, Licht von anderen Oberflächen
 - ▶ **diffus**: nach dem Lambertschen Gesetz
 - ▶ **spekular**: imperfekte Spiegelung



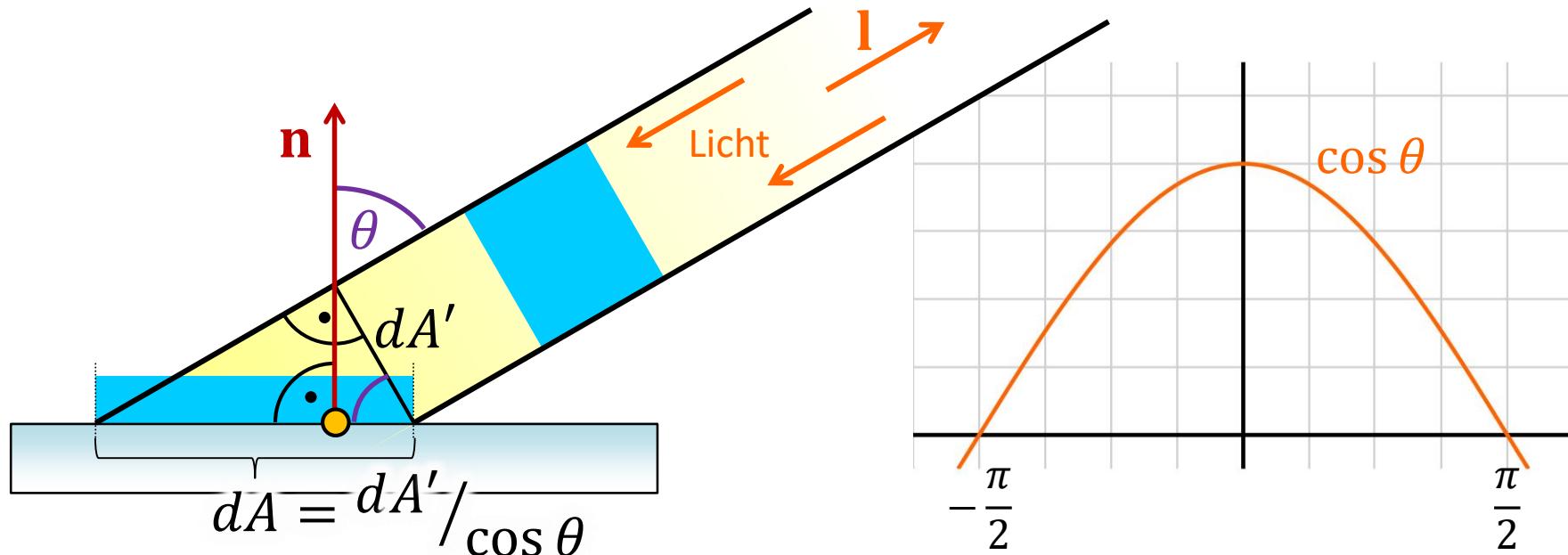
- ... lässt sich natürlich als BRDF $f_r(\mathbf{l}, \mathbf{x}, \mathbf{v})$ schreiben, wir verwenden aber zunächst eine anschaulichere Schreibweise (und gehen Folgenden davon aus, dass alle Richtungsvektoren normiert sind!)

Phong-Beleuchtungsmodell



Lambertsche (ideal-diffuse) Reflexion

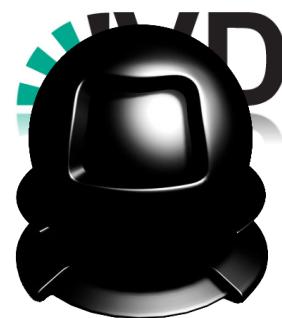
- ▶ Lichtstrahl aus Richtung \mathbf{l} zur Lichtquelle mit Intensität I_L und Materialkoeffizient k_d (geg. für Wellenlängen oder RGB)
- ▶ Bestrahlungsstärke der Fläche: $I_L \frac{dA'}{dA} \sim \cos \theta$
- ▶ modelliere diffuse Reflexion mit $k_d \cdot I_L \cdot \cos \theta$, wenn $|\theta| \leq \frac{\pi}{2}$
 - ▶ mit Materialkoeffizient k_d und mit $\cos \theta = \mathbf{n} \cdot \mathbf{l}$
 - ▶ Erinnerung: Skalarprodukt definiert als $\mathbf{a} \cdot \mathbf{b} = |\mathbf{a}||\mathbf{b}| \cos \phi$



Lambertsche (ideal-diffuse) Reflexion

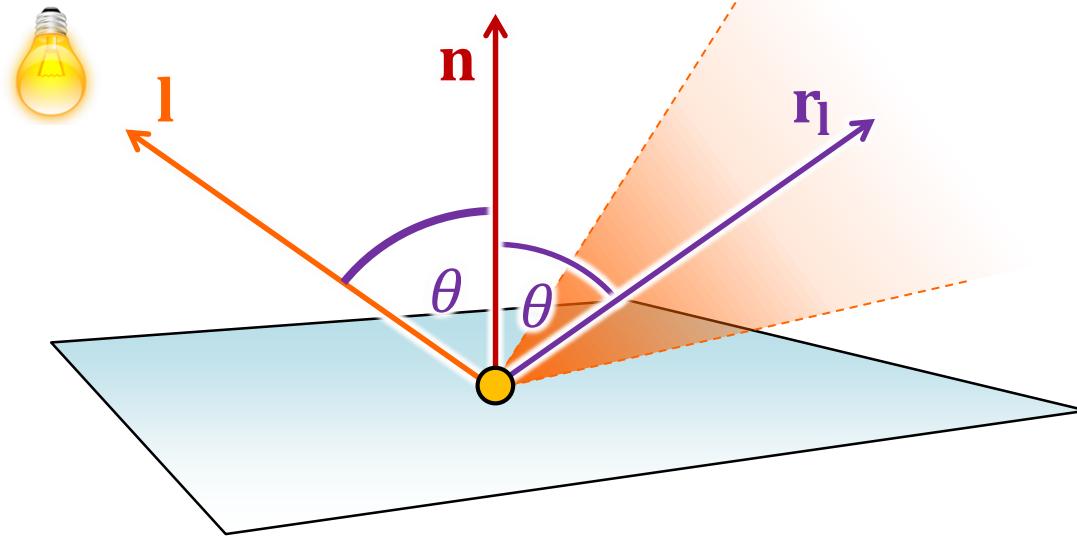


Phong-Beleuchtungsmodell



Spekulare Reflexion: Herleitung perfekte Spiegelung

- ▶ durch gerichtete Reflexion entstehen Glanzlichter
- ▶ zentriert um die Richtung der *perfekten Spiegelung* r_l

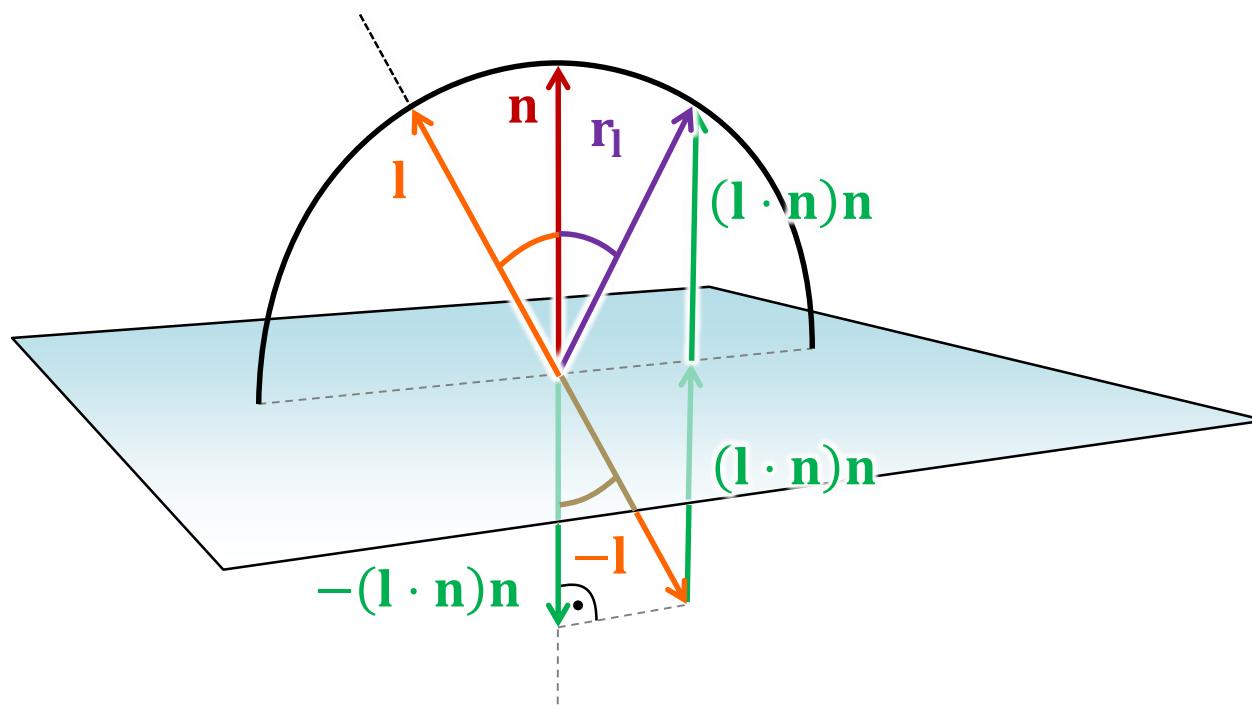


Phong-Beleuchtungsmodell

Spekulare Reflexion: Herleitung perfekte Spiegelung

- ▶ Spiegelung des Vektors \mathbf{l} an der Oberfläche mit der Normale \mathbf{n}
- ▶ die Vektoren \mathbf{l} , \mathbf{n} , und \mathbf{r}_l liegen in einer Ebene

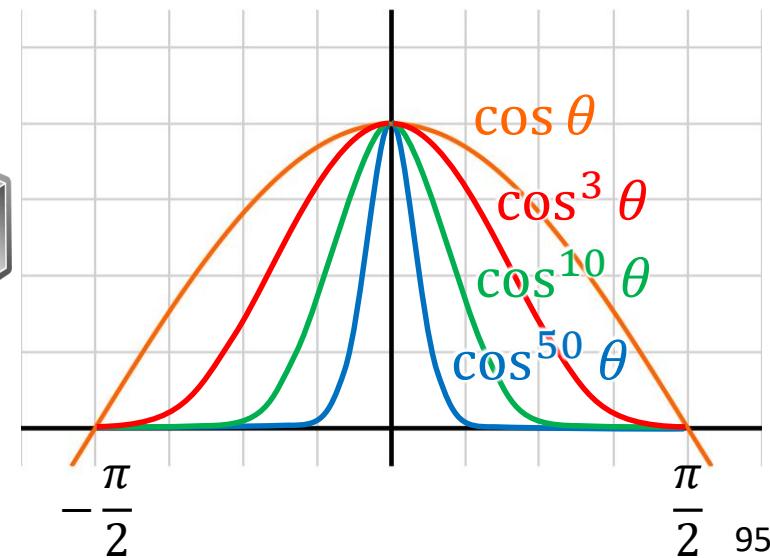
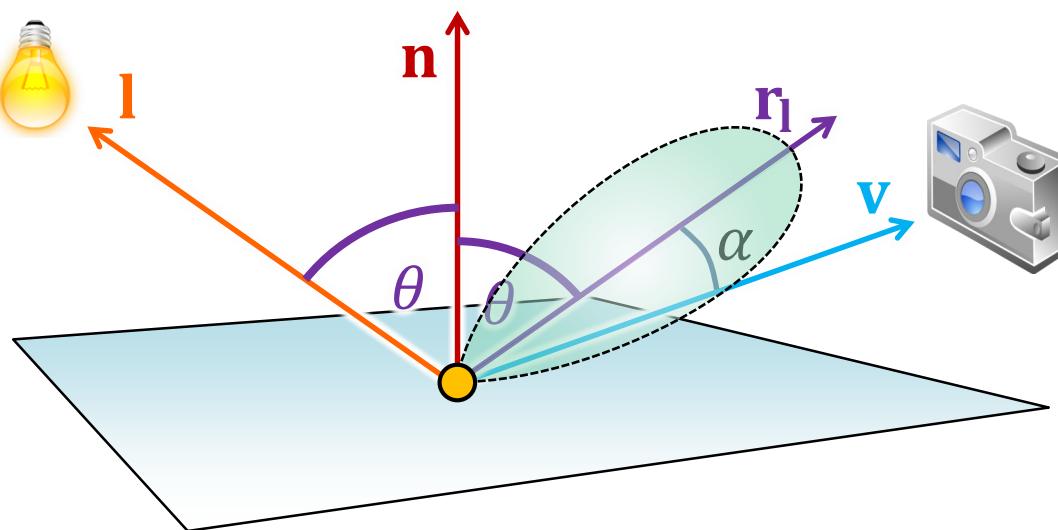
$$\mathbf{r}_l = 2(\mathbf{l} \cdot \mathbf{n})\mathbf{n} - \mathbf{l}$$



Phong-Beleuchtungsmodell

Spekulare Reflexion

- *perfekte Spiegelung* in Richtung $\mathbf{r}_l = 2(\mathbf{l} \cdot \mathbf{n})\mathbf{n} - \mathbf{l}$
- Stärke der *imperfekten Spiegelung* fällt für von \mathbf{r}_l verschiedene Richtungen ab: modelliere Abfall durch $\cos^n \alpha$
- spekulare Reflexion: $I_s = k_s \cdot I_L \cdot \cos^n \alpha = k_s \cdot I_L \cdot (\mathbf{r}_l \cdot \mathbf{v})^n$
 - spekularer Reflexionskoeffizient k_s und sog. Phong-Exponent n
 - n groß → kleine Glanzlichter, n klein → große Glanzlichter
 - Hinweis: $(\mathbf{r}_l \cdot \mathbf{v}) = (\mathbf{r}_v \cdot \mathbf{l})$



Spekulare Reflexion und Glanzlichter

► Auswirkungen des Phong-Exponenten

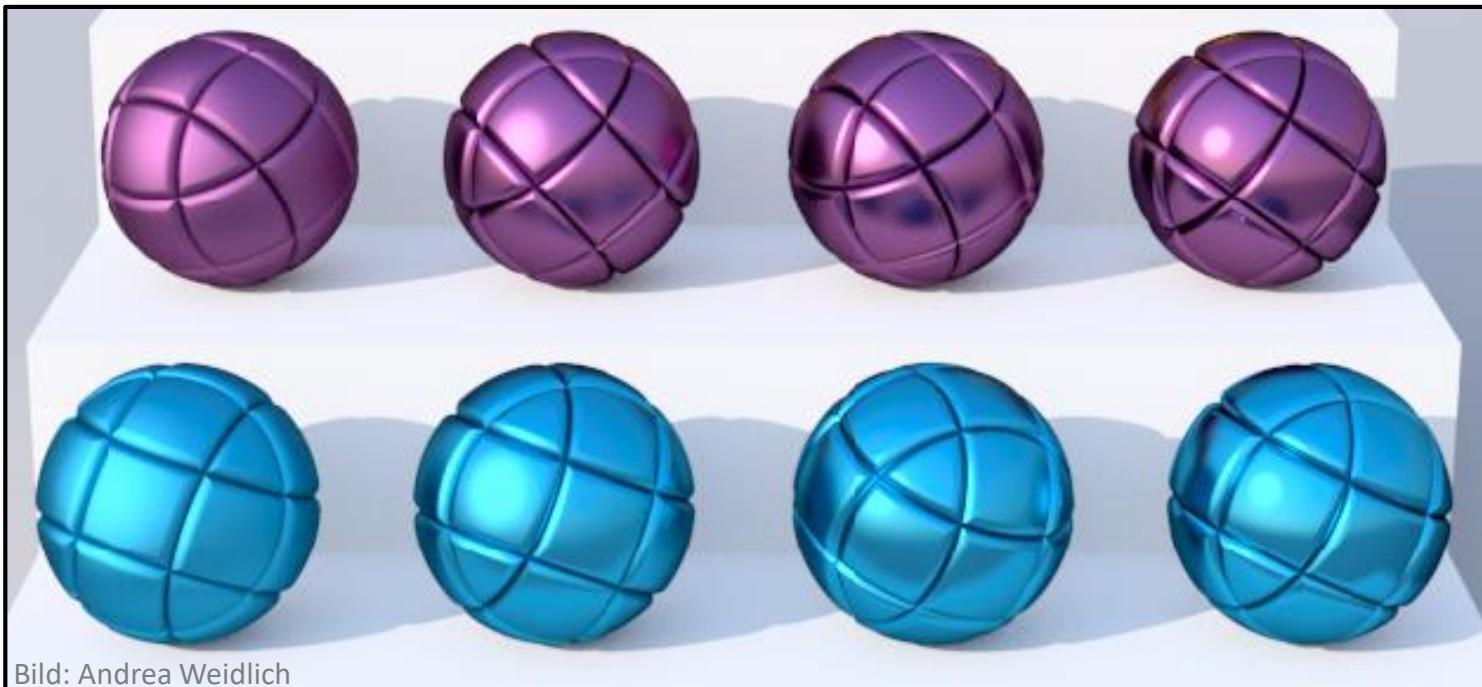
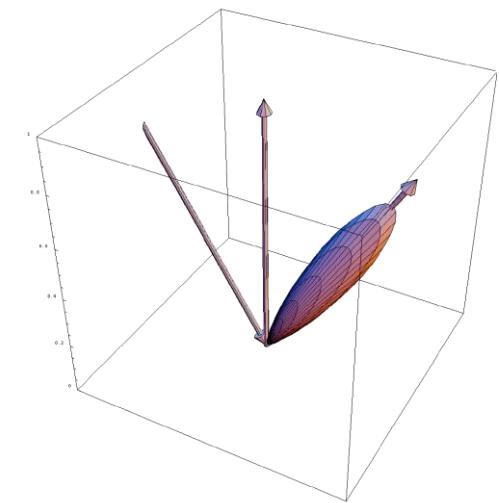
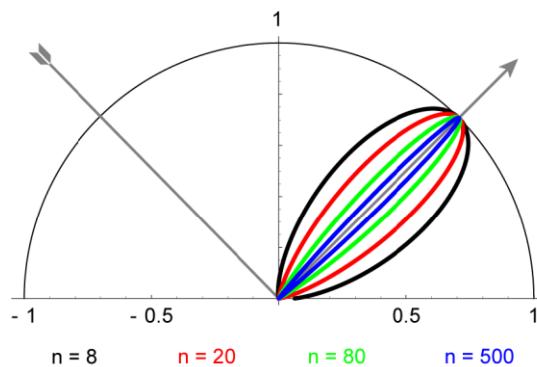
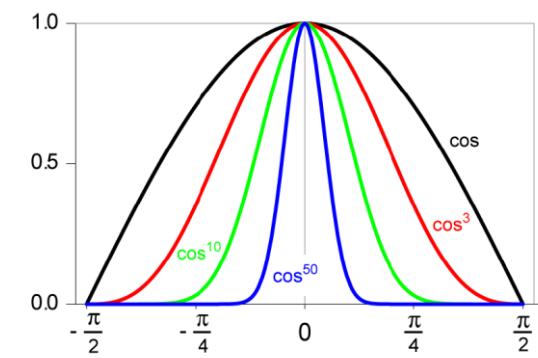
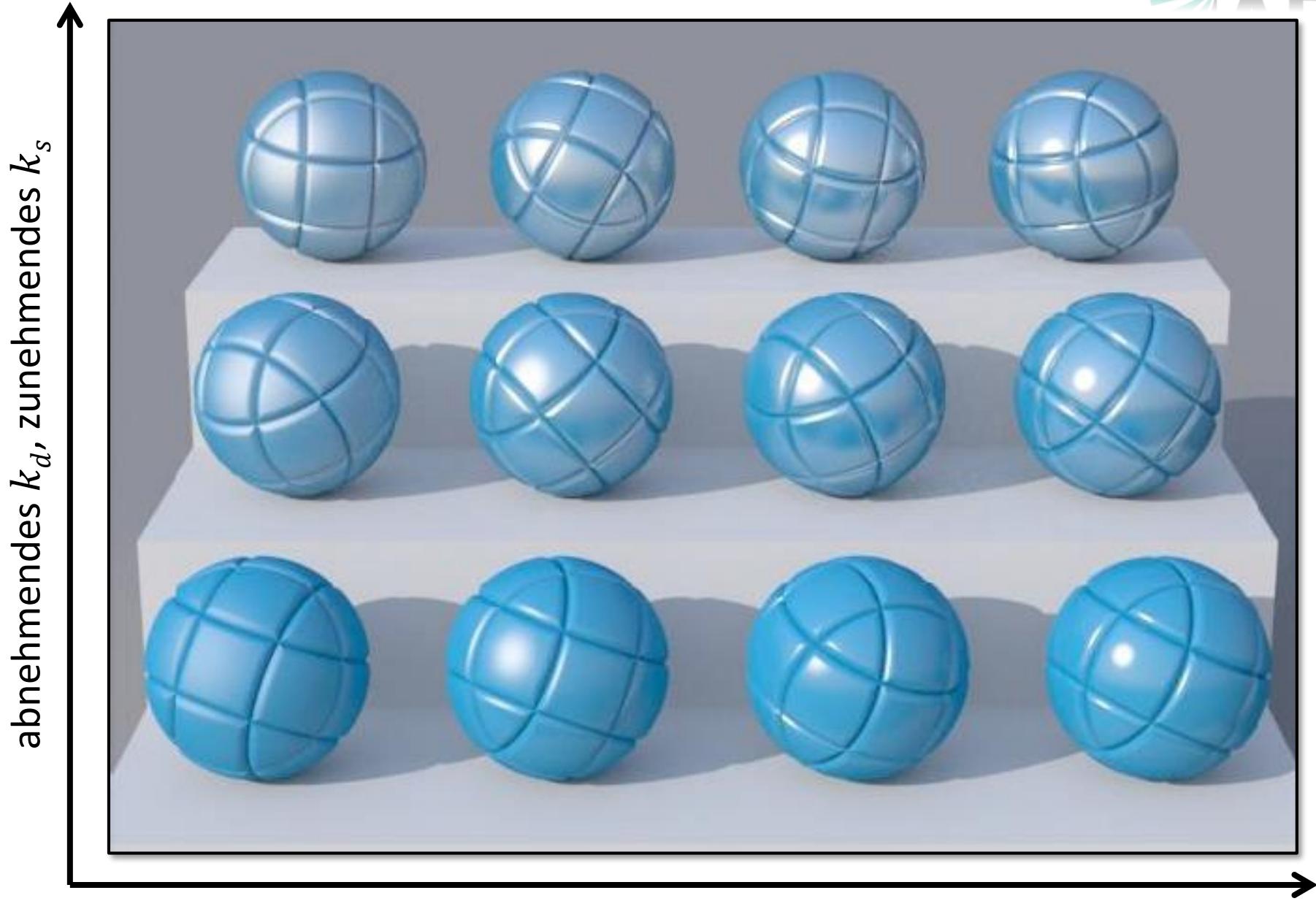


Bild: Andrea Weidlich

Spekulare Reflexion und Glanzlichter

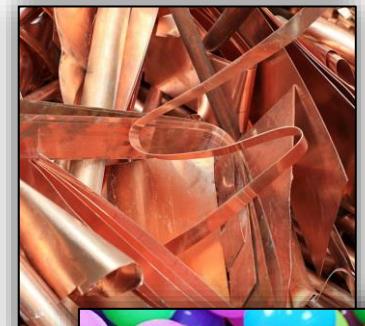


Zusammenfassung

- das Phong-Beleuchtungsmodell ist ein phänomenologisches Modell mit drei Komponenten **ambient**, **diffus** und **spekular**:

$$I = k_a \cdot I_L + k_d \cdot I_L \cdot (\mathbf{n} \cdot \mathbf{l}) + k_s \cdot I_L \cdot (\mathbf{r}_l \cdot \mathbf{v})^n$$

- Bemerkungen:
 - k_a, k_d, k_s und Lichtintensität I_L sind wellenlängenabhängig
 - k_a, k_d repräsentieren die Eigenfarbe der Oberfläche
 - spekulare Reflexion k_s hat bei Metallen die Farbe der Oberfläche, sonst meist die Farbe der Lichtquelle
 - wir sind nur an Richtungen interessiert, für die die **Skalarprodukte positiv** sind; daher verwendet man oft z.B. $(\mathbf{n} \cdot \mathbf{l})^+ := \max(0, (\mathbf{n} \cdot \mathbf{l}))$
- physikalisch-korrekt?
 - welche Werte dürfen k_a, k_d, k_s annehmen?



Raytracing Pseudocode

```
for ( y = 0; y < height; y++ ) {
    for ( x = 0; x < width; x++ ) {
        u = l + (r-l) * (x+0.5) / width;
        v = t + (b-t) * (y+0.5) / height;
        s = ...;
        d = normalize( s );

        // finde nächsten Schnittpunkt
        intersection = NULL;
        float t = FLOAT_MAX;

        for ( each object ) {
            t' = intersect( object, e, d );
            if ( t' > 0 && t' < t ) {
                intersection = object;
                t = t';
            }
        }

        // Beleuchtungsberechnung
        if ( intersection != NULL ) { ... }
    }
}
```

Beleuchtungsberechnung im Raytracer



```
struct vec3 {           // wir verwenden in der Übung die
    float x, y, z;     // OpenGL Mathematics (GLM) Library
    ...
};

class Material {
    vec3 ka, kd, ks;   // Reflexionsparameter (hier: RGB, oder Spektrum)
    float n;            // Phong-Exponent
    ...
};

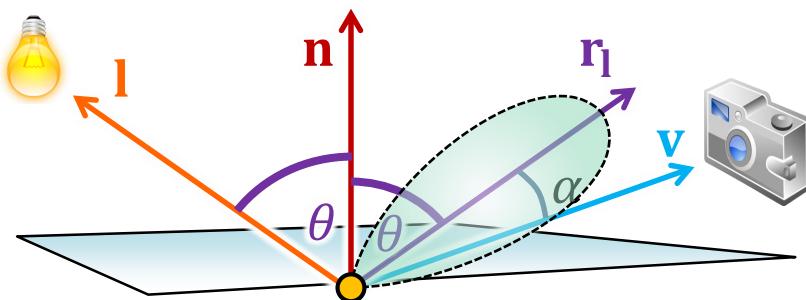
struct Intersection {
    vec3 p, n, v;      // Position, Normale,
                        // Vektor zur Kamera/zum Startpunkt des Strahls
    Material *material; // Reflexionsparameter
};

class PointLight : public LightSource {
    vec3 pos,           // Position
        I_L;            // Intensität der LQ (hier: RGB, oder Spektrum)
    ...
};
```

Beleuchtungsberechnung im Raytracer

```
vec3 PointLight::computeDirectLight( const Intersection &i, ... )  
{  
    // ambienter Term  
    vec3 I = i.material->ka * I_L;  
  
    // hier: Punktlichtquelle  
    vec3 L = normalize( pos - i.p );  
    float NdotL = dot( i.n, L );  
  
    if ( NdotL > 0 ) {  
        // diffuser Term  
        I += i.material->kd * I_L * NdotL;  
  
        // spekularer Term  
        vec3 R = 2.0f * i.n * NdotL - L;  
        float RdotV = dot( R, V );  
        if ( RdotV > 0 ) {  
            I += i.material->ks * I_L * powf( RdotV, i.material->n );  
        }  
    }  
};
```

$$I = k_a \cdot I_L + k_d \cdot I_L \cdot (\mathbf{n} \cdot \mathbf{l}) + k_s \cdot I_L \cdot (\mathbf{r}_l \cdot \mathbf{v})^n$$



Beleuchtungsberechnung im Raytracer

```
vec3 PointLight::computeDirectLight( const Intersection &i, ... )
{
    // ambienter Term
    vec3 I = i.material->ka * I_L;

    // hier: Punktlichtquelle
    vec3 L = normalize( pos - i.p );
    float NdotL = dot( i.n, L );
```

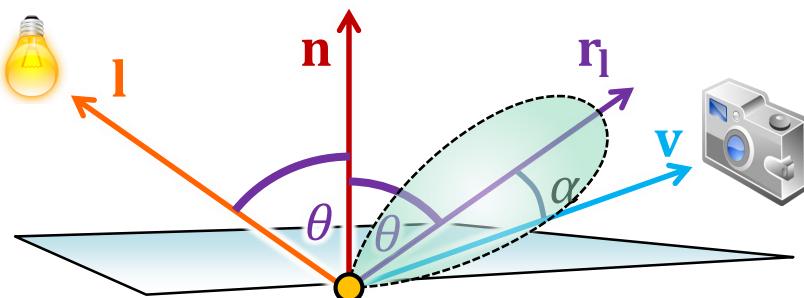


die Intensität einer „realistischen“
Punktlichtquelle würde mit dem
 Abstandsquadrat abfallen:

```
float dist = length( pos - i.p );
...
I += i.material->kd * I_L * NdotL /
    ( dist * dist );
```

```
float RdotV = dot( R, V );
if ( RdotV > 0 ) {
    I += i.material->ks * I_L * powf( RdotV, i.material->n );
}
};
```

$$I = k_a \cdot I_L + k_d \cdot I_L \cdot (\mathbf{n} \cdot \mathbf{l}) + k_s \cdot I_L \cdot (\mathbf{r}_l \cdot \mathbf{v})^n$$

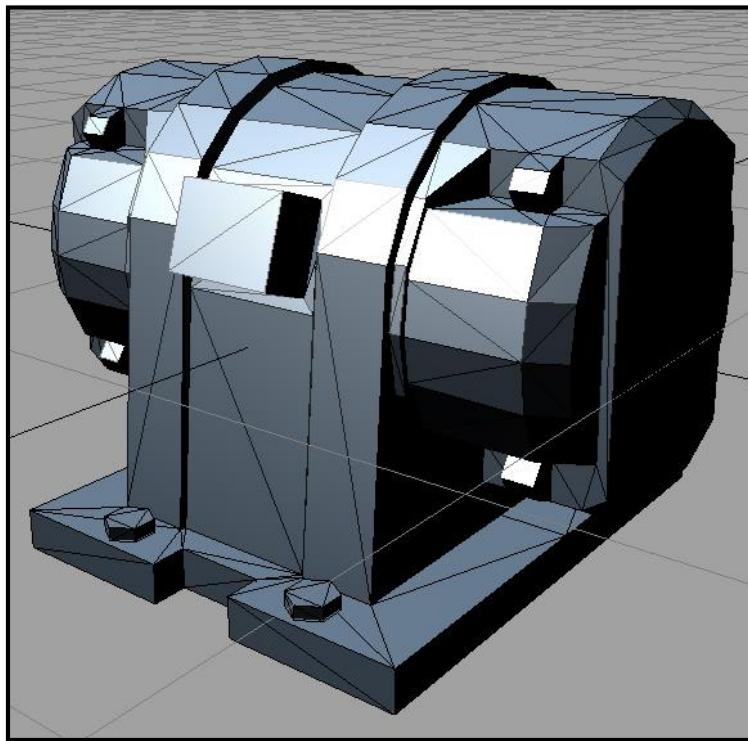


Die Beiträge mehrerer Lichtquellen werden aufsummiert (Superpositionsprinzip)

Schattierung von Dreiecksnetzen

Beleuchtungsberechnung mit welcher Normale?

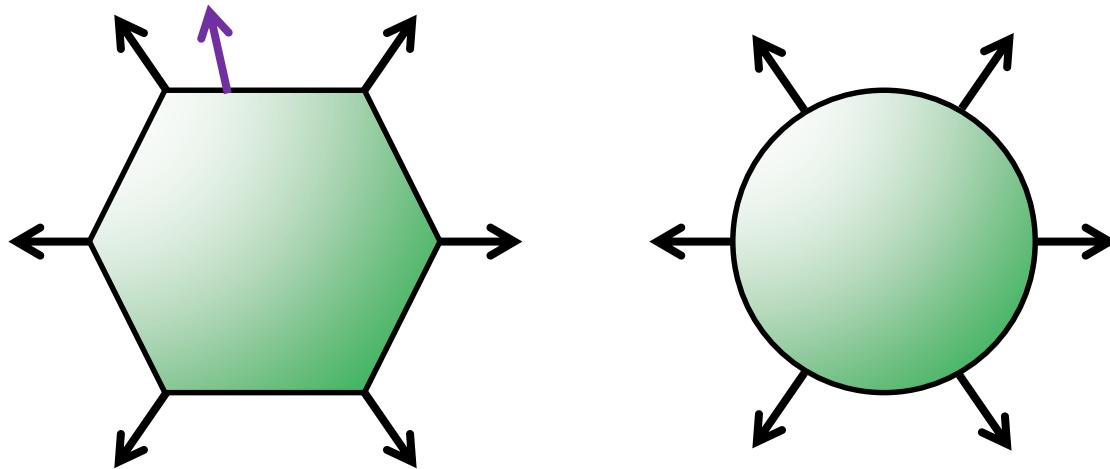
- Dreiecke: stückweise lineare Approximation einer (möglicherweise gekrümmten) Oberfläche
- **Flat Shading:** Verwendung der Dreiecksnormale für Beleuchtung



Schattierung von Dreiecksnetzen

Beleuchtungsberechnung mit welcher Normale?

- ▶ Illusion einer glatten, gekrümmten Fläche durch
 - ▶ Normalenvektoren an den Vertices/Eckpunkten („Vertex-Normalen“)
 - ▶ Interpolation der Normalen über den Dreiecken



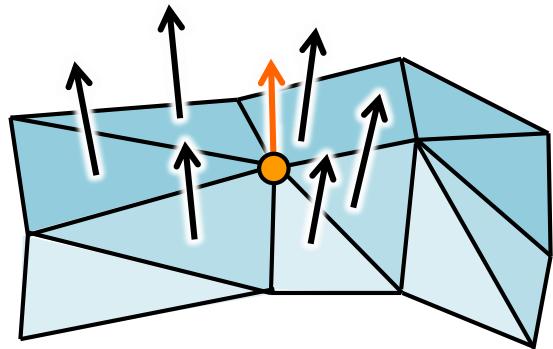
- ▶ Vertex-Normalen können entweder
 - ▶ berechnet werden (z.B. bei der Triangulierung parametrischer Flächen)
 - ▶ direkt aus einem Dreiecksnetz bestimmt werden

Schattierung von Dreiecksnetzen

Normalen für Dreiecksnetze

- ▶ Vertex-Normalen bestimmen über
 - ▶ (gewichtete) Summe der Normalen angrenzender Flächen
 - ▶ Normalen für ein Dreieck über Kreuzprodukt

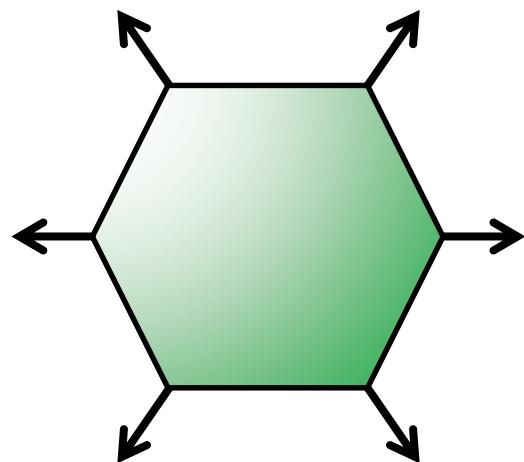
- ▶ Vorgehen
 - ▶ berechne Normale für jedes Dreieck
 - ▶ für jeden Vertex: summiere die Normalen aller angrenzenden Dreiecke
 - ▶ optional: skaliere Normalen mit der Dreiecksfläche (ohne Zutun im Kreuzprodukt)
 - ▶ **normalisiere** die Vertex-Normalen
 - ▶ NICHT durch die Anzahl der Normalen teilen!



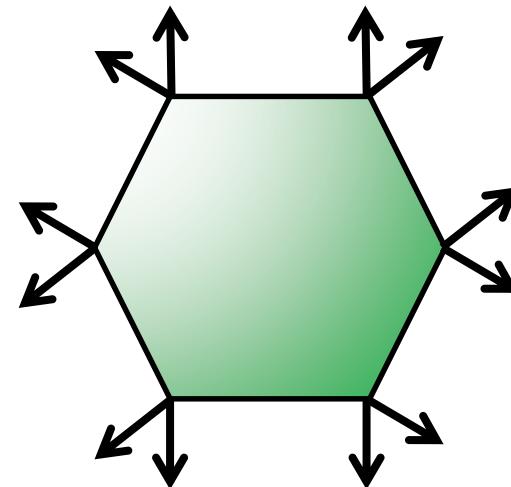
Schattierung von Dreiecksnetzen

Normalen für Dreiecksnetze

- ▶ scharfe Kanten brauchen mehrere Normalen
 - ▶ Kante, wenn der Winkel zweier benachbarter Dreiecke zu groß ist
- ▶ bei Dreiecksnetzen werden pro Dreieck typischerweise 3 Eckpunkte und 3 Normalen (und zusätzliche Attribute, z.B. Farben) gespeichert
 - ▶ bei Mittelung pro Vertex: berücksichtige nur Dreiecksnormalen, deren Winkel zur Normale des „eigenen“ Dreiecks nicht zu groß ist



glatte Schattierung

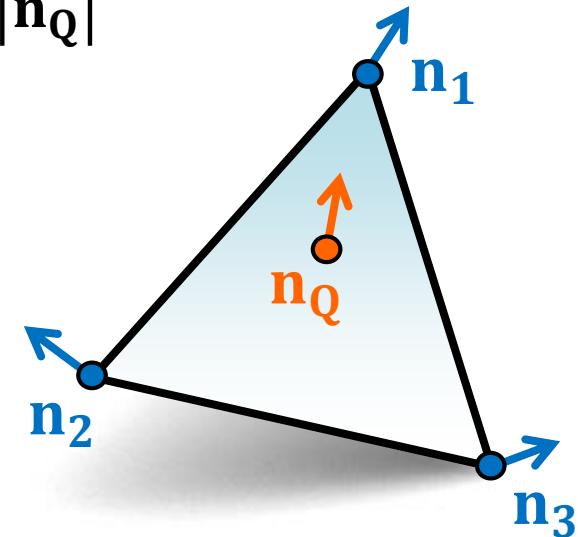


scharfe Kanten

Schattierung von Dreiecksnetzen

Interpolation von Normalen (z.B. für Schnittpkt. für Schattierung)

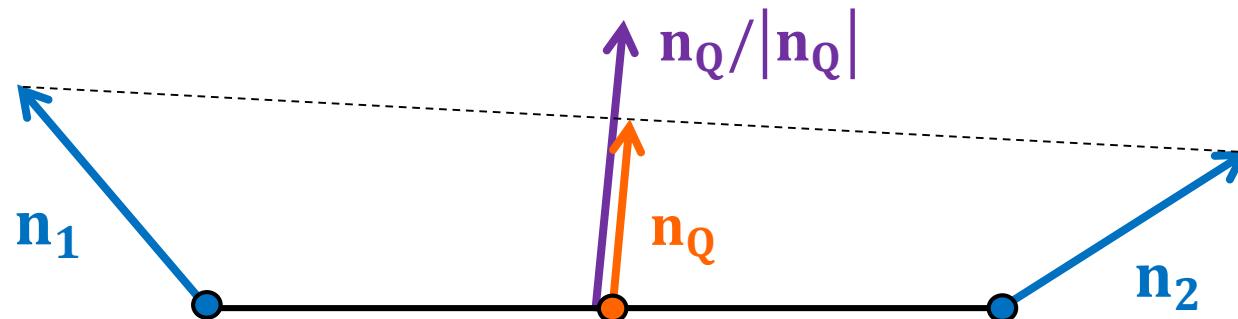
- Berechnung durch lineare komponentenweise Interpolation der Vertex-Normalen anhand der baryzentrischen Koordinaten
- Beispiel:
 - geg.: Normalen $\mathbf{n}_1, \mathbf{n}_2, \mathbf{n}_3$ eines Dreiecks $\triangle (P_1, P_2, P_3)$
 - ges.: interpolierte Normale \mathbf{n}_Q an einem Punkt Q auf dem Dreieck
 - Lösung: berechne $\lambda_1, \lambda_2, \lambda_3$, dann ist $\mathbf{n}_Q = \lambda_1 \mathbf{n}_1 + \lambda_2 \mathbf{n}_2 + \lambda_3 \mathbf{n}_3$
- Beleuchtungsberechnung mit der Normale $\mathbf{n}_Q / |\mathbf{n}_Q|$
 - ⚠️ ► lineare komponentenweise Interpolation ist nicht längenerhaltend!
 - Beleuchtungsberechnung mit interpolierten Normalen nennt man **Phong Shading**
(!= Phong Beleuchtungsmodell)



Schattierung von Dreiecksnetzen

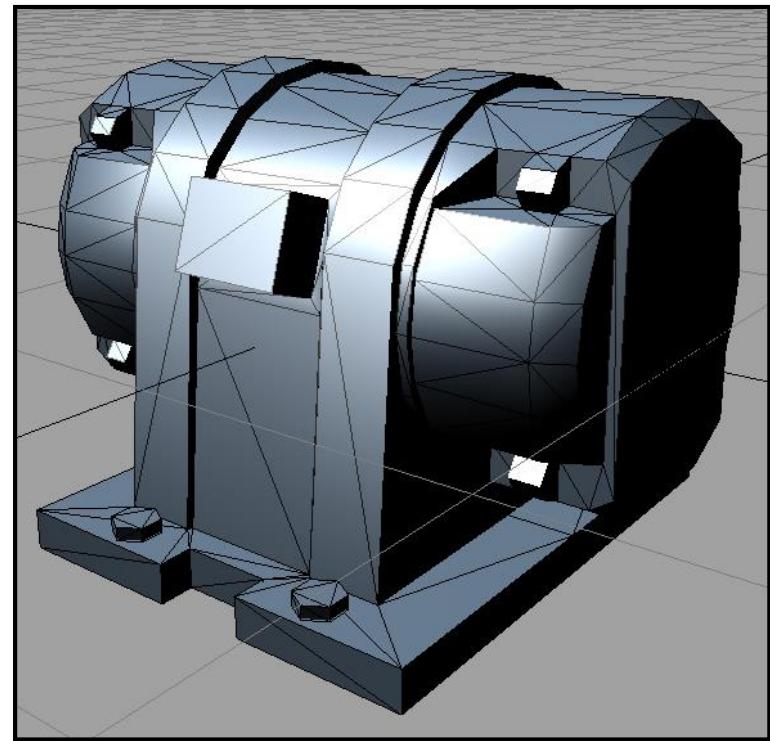
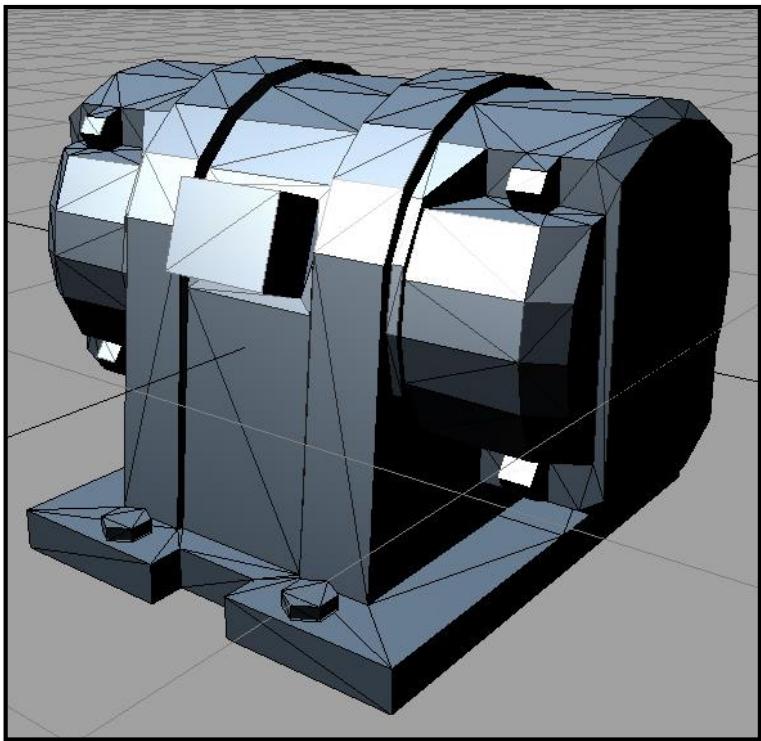
Interpolation von Normalen (z.B. für Schnittpkt. für Schattierung)

- ▶ Berechnung durch lineare komponentenweise Interpolation der Vertex-Normalen anhand der baryzentrischen Koordinaten
- ▶ Beleuchtungsberechnung mit der Normale $n_Q / |n_Q|$
 - ▶ lineare komponentenweise Interpolation ist nicht längenerhaltend!
- ▶ Darstellung in 2D: Interpolation entlang einer Kante



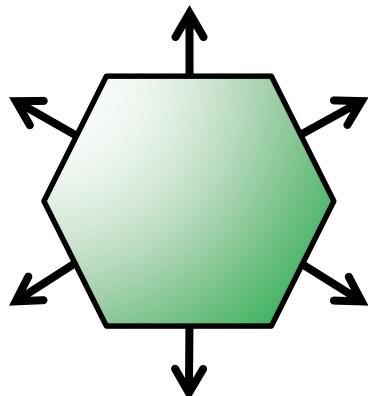
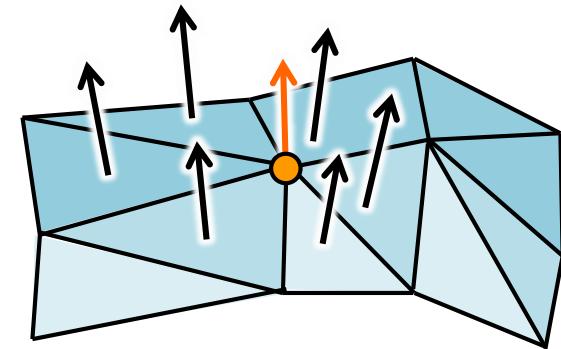
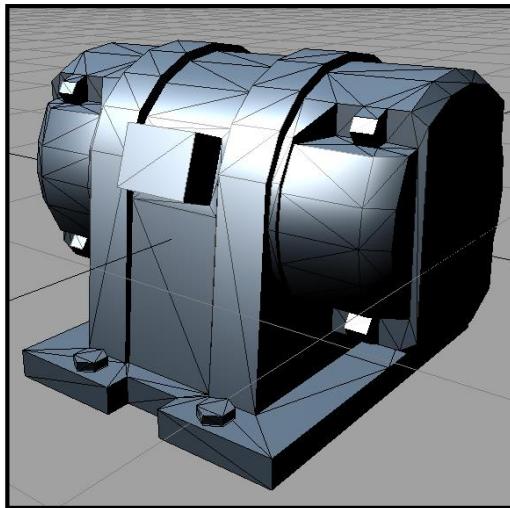
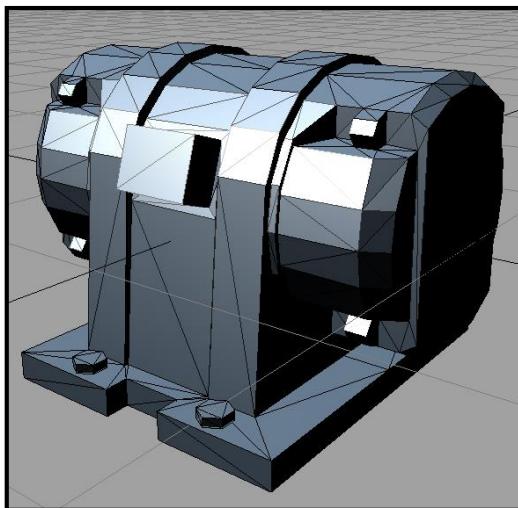
Schattierung von Dreiecksnetzen

- Flat Shading (links) und Phong Shading (rechts, mit Erhalten scharfer Kanten, also 3 Normalen pro Dreieck gespeichert)

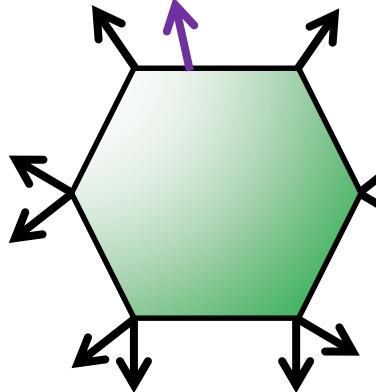


Zusammenfassung: Schattierung von Dreiecksnetzen

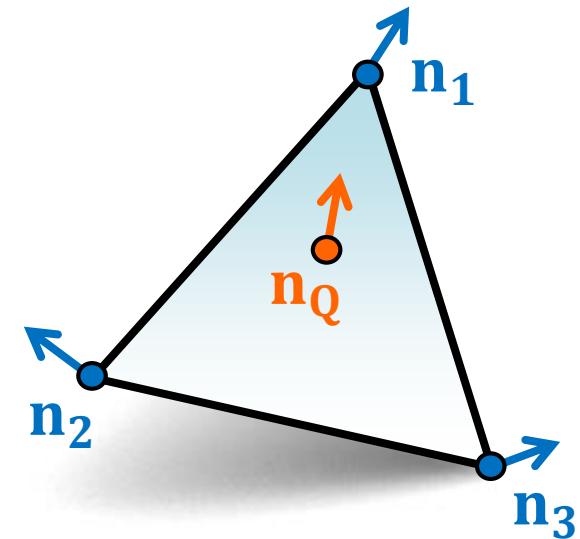
Normalen für Shading



Flat Shading
(Dreiecksnormale)

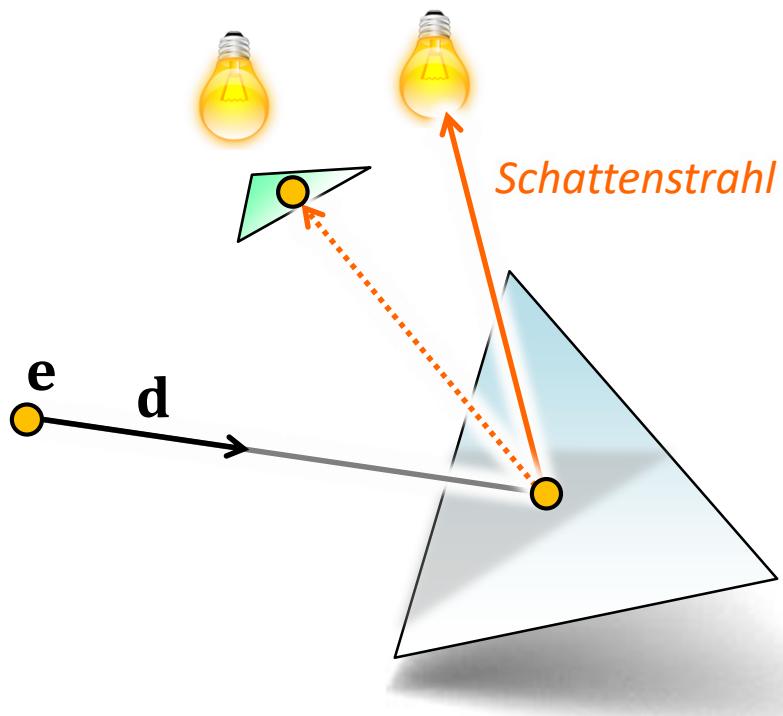


Phong Shading
(interpolierte Normale)



Licht und Schatten

- ▶ bisher: eine Oberfläche wird immer beleuchtet, wenn sie nicht von der Lichtquelle abgewandt ist
- ▶ aber: andere Objekte der Szene können Schatten werfen
- ▶ Lösung: sende einen „Schattenstrahl“ zur Lichtquelle hin
 - ▶ teste, ob dieser Strahl auf dem Weg **von der Oberfläche zur Lichtquelle** ein anderes Objekt schneidet



Schattenberechnung im Raytracer

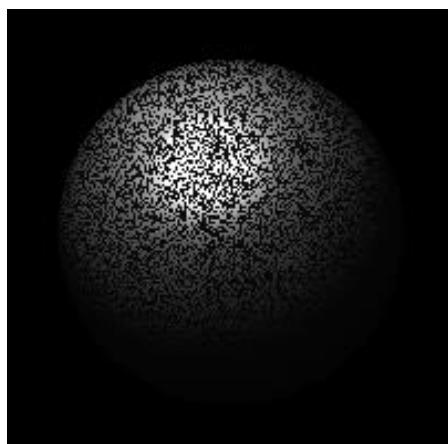


```
vec3 PointLight::computeDirectLight( const Intersection &i, ... ) {  
    // ambienter Term  
    vec3 I = i.material->ka * I_L;  
  
    vec3 L = pos - i.p;  
  
    // Schattenstrahl  
    ! float dist2Light = length( L );  
  
    ! for ( each object ) {  
        !     t' = intersect( object, i.p, L );  
        !     if ( t' > 0 && t' < dist2Light )  
            return I;  
    }  
    ...  
};
```

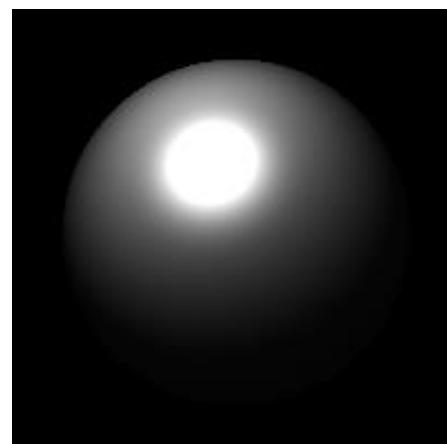
- i.d.R. zwei unterschiedliche Routinen in einem Raytracer
 - ▶ teste ob irgendein Schnittpunkt für $t \in (0; t_{max})$
 - ▶ suche den nächsten Schnittpunkt für $t > 0$
- semi-transparente Flächen reduzieren I_L

Schattenberechnung im Raytracer

- Problem: endliche Genauigkeit bei der Gleitkommadarstellung
 - wird der Schattentest so durchgeführt, wie auf der vorherigen Folie beschreiben, treten Artefakte auf:
der Strahl schneidet u.U. die Oberfläche von der er startet nochmal!
- prinzipiell zwei Möglichkeiten
 - teste explizit, ob nochmals dasselbe Objekt geschnitten wurde:
funktioniert nur bei konvexen Objekten (z.B. Dreiecke, Kugeln)
 - üblicher: starte etwas entfernt von der Oberfläche, z.B. mit $t' = \text{intersect(object, i.p + epsilon * i.n, I);}$



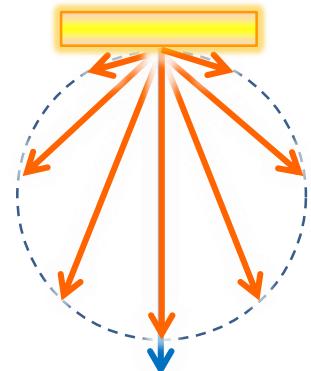
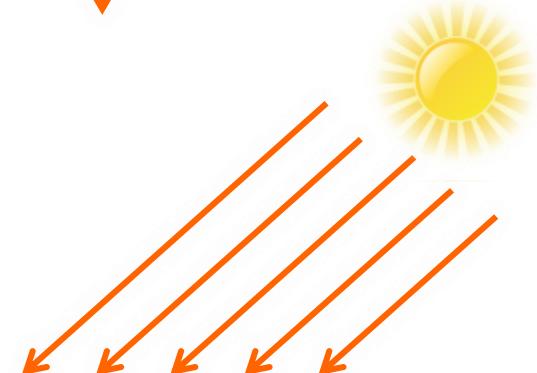
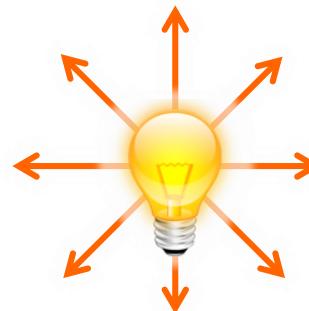
$$\epsilon = 0$$



$$\epsilon = 10^{-7}$$

Arten von Lichtquellen

- ▶ Punktlichtquelle
 - ▶ definiert durch Position \mathbf{p} und Intensität I_L (W/sr)
 - ▶ reale Lichtquellen: Abfall der Intensität mit dem Abstandsquadrat
- ▶ paralleles Licht/direktionale Lichtquelle
 - ▶ definiert durch Richtung \mathbf{d} und Flussdichte E (W/m^2)
 - ▶ Bsp. Sonnenlicht ist annähernd parallel
- ▶ weitere Lichtquellentypen
 - ▶ Strahler (Spot-Lights): Lichtkegel, Abstrahlungscharakteristik mit $\cos^n \theta$
 - ▶ Punktlichtquellen mit Richtungscharakteristik
 - ▶ Flächenlichtquellen



Raytracing Pseudocode

```
class Ray {
    vec3 origin, direction; // Startpunkt und Richtung des Strahls
    float t;                // Strahlparameter
    void * object;          // Zeiger auf evtl. getroffenes Objekt
    ...
};

for ( y = 0; y < height; y++ ) {
    for ( x = 0; x < width; x++ ) {
        u = l + (r-l) * (x+0.5) / width;
        v = t + (b-t) * (y+0.5) / height;
        s = ...;
        d = normalize( s );
    }
    // finde nächsten Schnittpunkt
    intersection = NULL;
    float t = FLOAT_MAX;

    for ( each object ) {
        t' = intersect( object, e, d );
        if ( t' > 0 && t' < t ) {
            intersection = object;
            t = t';
        }
    }
    // Beleuchtungsberechnung
    if ( intersection != NULL ) {
        computeDirectLight ...
    }
}
```

Strahlerzeugung (abh. vom Kameramodell)
generateRay(Ray *ray, int x, int y);

Strahlverfolgung und Beleuchtung
vec3 color =
raytrace(Ray *ray, ...);

Raytracing Pseudocode

```
class Ray {
    vec3 origin, direction; // Startpunkt und Richtung des Strahls
    float t;                // Strahlparameter
    void * object;          // Zeiger auf evtl. getroffenes Objekt
    ...
};

for ( y = 0; y < height; y++ ) {
    for ( x = 0; x < width; x++ ) {
        u = 1 + (r-1) * (x+0.5) / width;
        v = t + (b-t) * (y+0.5) / height;
        s = ...;
        d = normalize( s );
    }
    // finde nächsten Schnittpunkt
    intersection = NULL;
    float t = FLOAT_MAX;

    for ( each object ) {
        t' = intersect( object, e, d );
        if ( t' > 0 && t' < t ) {
            intersection = object;
            t = t';
        }
    }
}

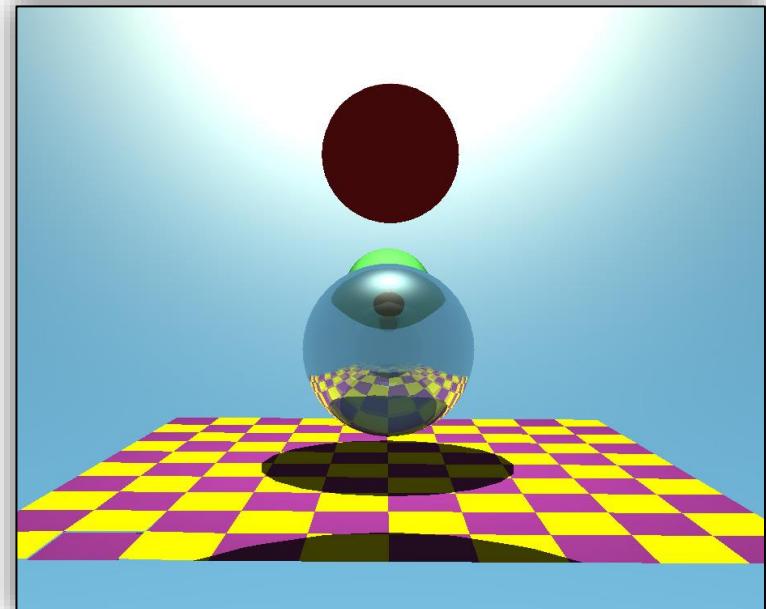
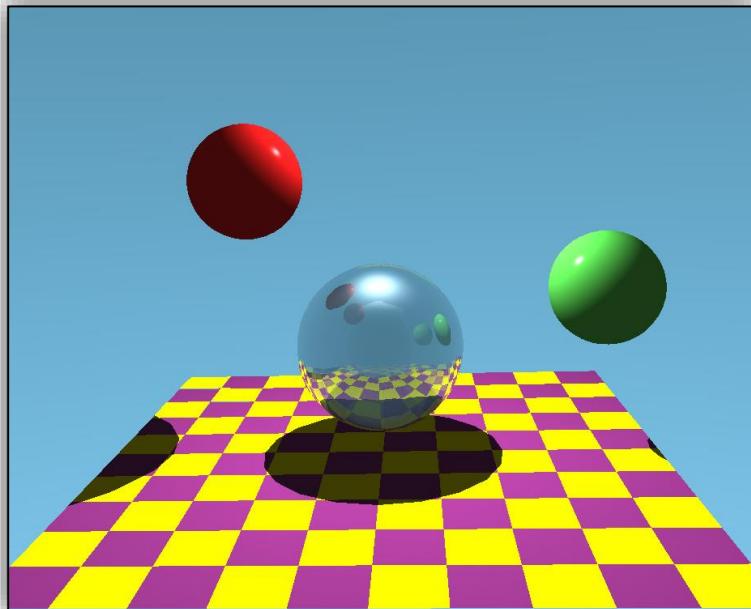
// Beleuchtungsberechnung
if ( intersection != NULL ) {
    computeDirectLight ...
} } }
```

Strahlerzeugung (abh. vom Kameramodell)
generateRay(Ray *ray, int x, int y);

Schnittpunktberechnung
bool cast(Ray *ray, float maxDist);

Schritte

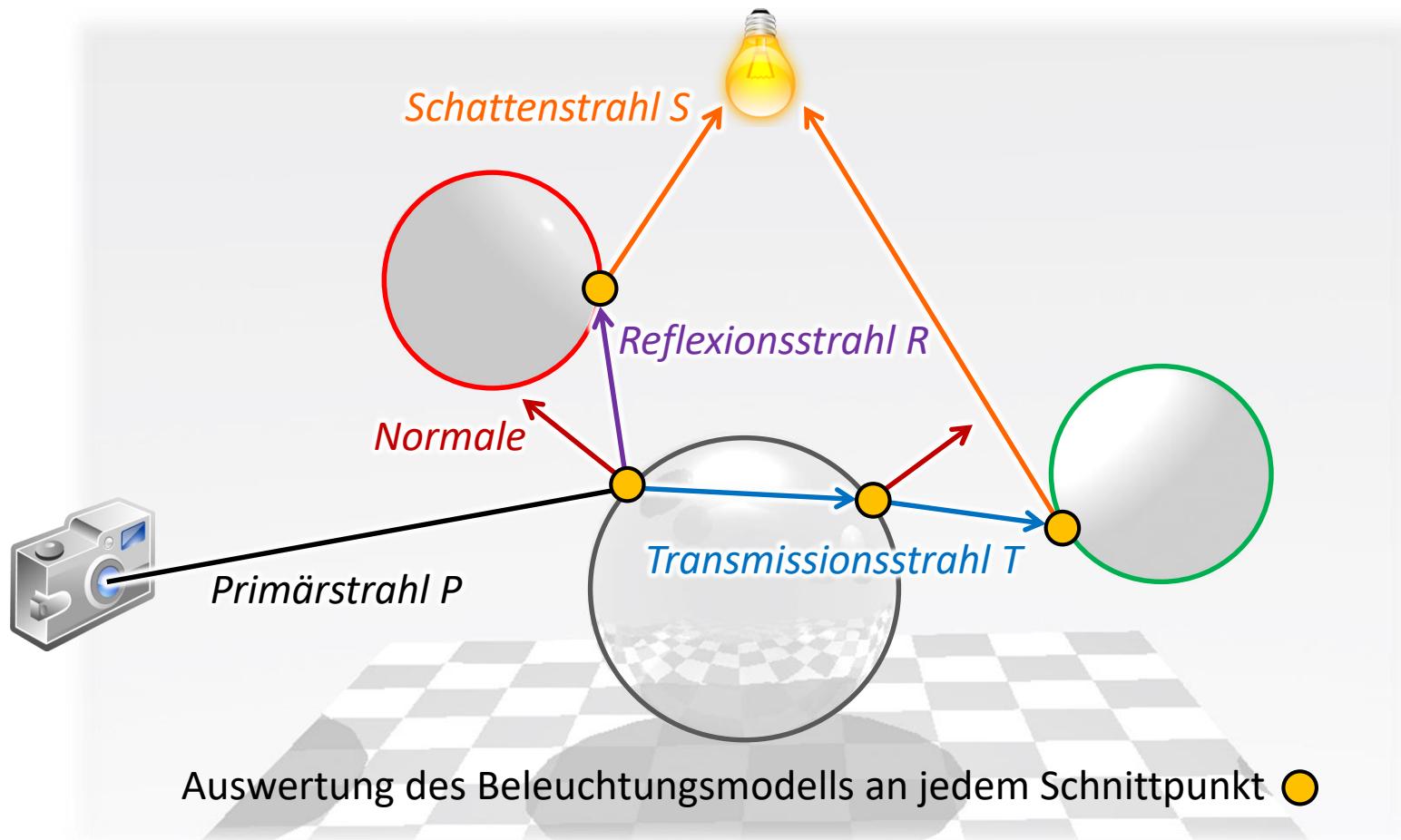
- ▶ Erzeugung der Sichtstrahlen durch jeden Pixel (ray generation)
- ▶ Schnittberechnung (ray casting):
finde geometrisches Primitiv/Objekt, das den Sichtstrahl am nächsten zur Kamera schneidet
- ▶ Schattierung und Beleuchtungsberechnung (shading)
- ▶ Sekundärstrahlen für Spiegelung und Transmission



Sekundärstrahlen

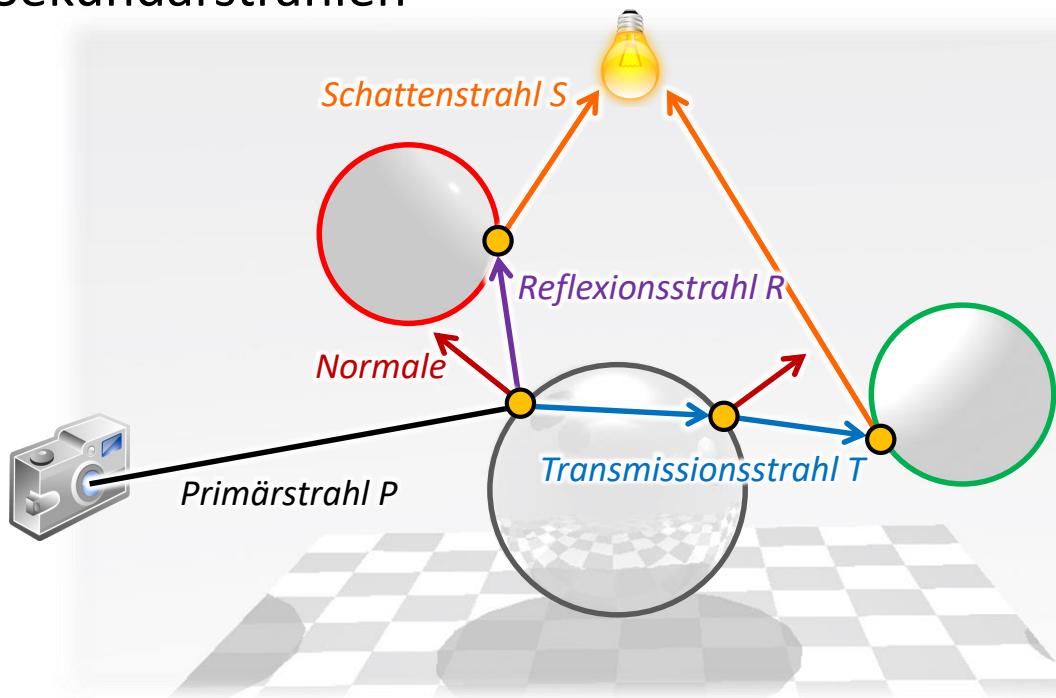
Berechnung von Spiegelungen und Transmission: Rekursives Raytracing

- bestimme Richtungen der Reflexions- und Transmissionsstrahlen (sog. Sekundärstrahlen)
- berechne Beitrag zur Farbe einer Oberfläche mit `raytrace(. . .)`



Spiegelungen, Reflexionsstrahlen

- ▶ Spiegelungen der Szene auf reflektierenden Oberflächen
 - ▶ Sichtstrahl P wird an der Oberfläche reflektiert → Reflexionsstrahl R
- ▶ behandle Reflexionsstrahl (fast so) wie einen Sichtstrahl:
berechne Schnittpunkt und gebe Farbe zurück
 - ▶ Beleuchtungsberechnung für die „Blickrichtung“ R
 - ▶ rekursive Verfolgung von (Mehrfach-)Reflexionen
 - ▶ weitere Schatten- und Sekundärstrahlen
- ▶ addiere Farbe des
Reflexions- bzw.
Transmissionsstrahls zur
Farbe am Ausgangspunkt
 - ▶ gewichtet mit
weiteren Koeffizienten
 k_r bzw. k_t



Raytracing Pseudocode

```
class Ray {  
    vec3 origin, direction; // Startpunkt und Richtung des Strahls  
    float t;                // Strahlparameter  
    void * object;          // Zeiger auf evtl. getroffenes Objekt  
    ...  
};  
  
for ( y = 0; y < height; y++ ) {  
    for ( x = 0; x < width; x++ ) {  
        u = l + (r-l) * (x+0.5) / width;  
        v = t + (b-t) * (y+0.5) / height;  
        s = ...;  
        d = normalize( s );  
  
        // finde nächsten Schnittpunkt  
        intersection = NULL;  
        float t = FLOAT_MAX;  
  
        for ( each object ) {  
            t' = intersect( object, e, d );  
            if ( t' > 0 && t' < t ) {  
                intersection = object;  
                t = t';  
            } }  
  
        // Beleuchtungsberechnung  
        if ( intersection != NULL ) {  
            computeDirectLight ...  
        } } }
```

Strahlverfolgung und Beleuchtung

vec3 color =
 raytrace(Ray *ray, ...);

um rekursive Strahlen zu verfolgen

Raytracing Pseudocode

```
vec3 raytrace( Ray *ray, ... ) {
    // Farbe
    vec3 color = 0.0f;

    // Schnittberechnung
    Intersection i;
    if ( !cast( ray, FLOAT_MAX, &i ) )
        return color;

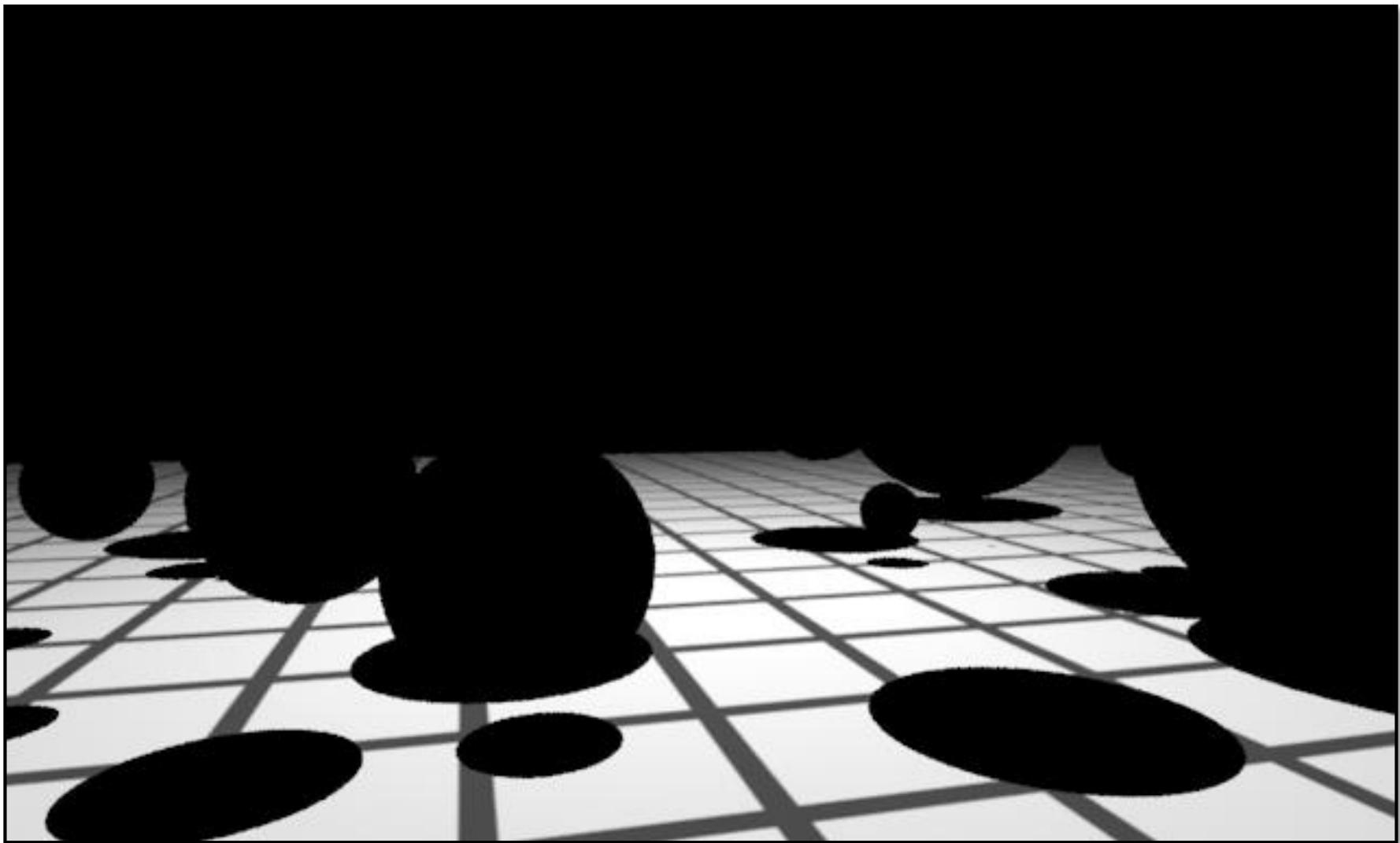
    for ( jede Lichtquelle )
        color += computeDirectLight( ... );

    if ( Fläche ist spiegelnd ) {
        // berechne Reflexionsstrahl
        Ray reflect = ...;
        color += i.kr * raytrace( reflect, ... );
    }

    return color;
}
```

Metallkugeln – Rekursionstiefe 0

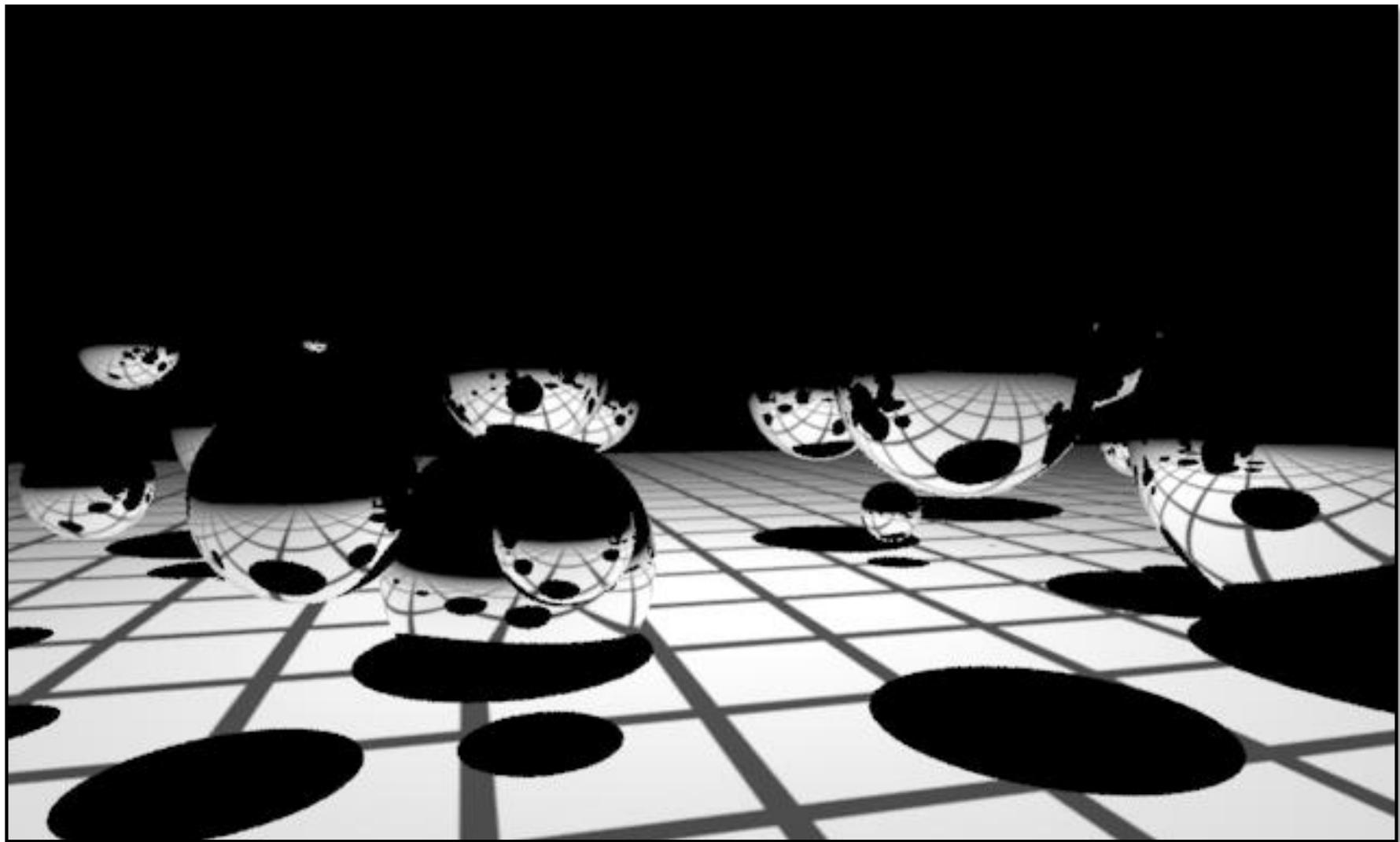
► Bild: Pat Hanrahan



Materialkoeffizienten der Metallkugeln: $k_a = k_d = k_s = k_t = 0$ und $k_r > 0$

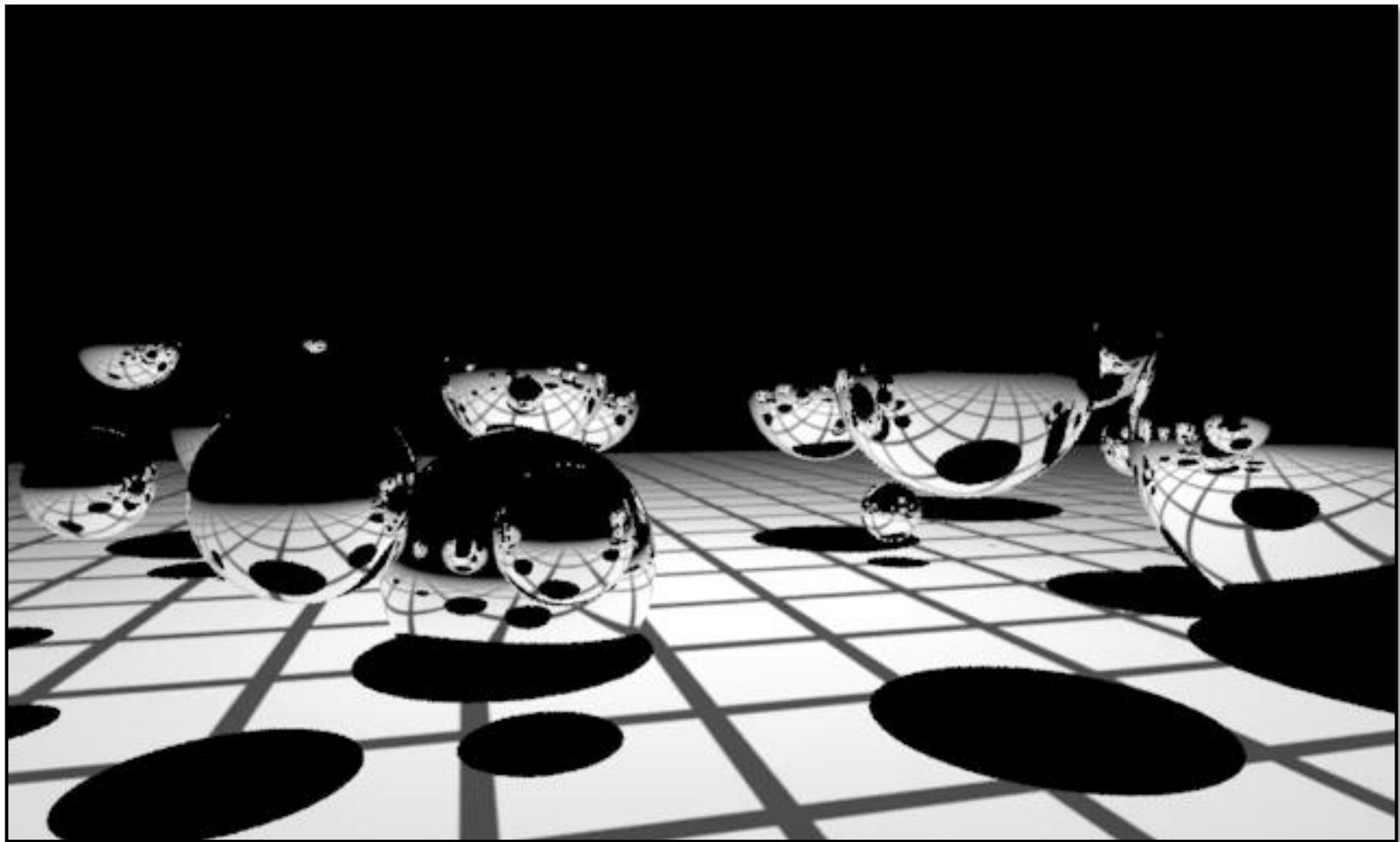
Metallkugeln – Rekursionstiefe 1

► Bild: Pat Hanrahan



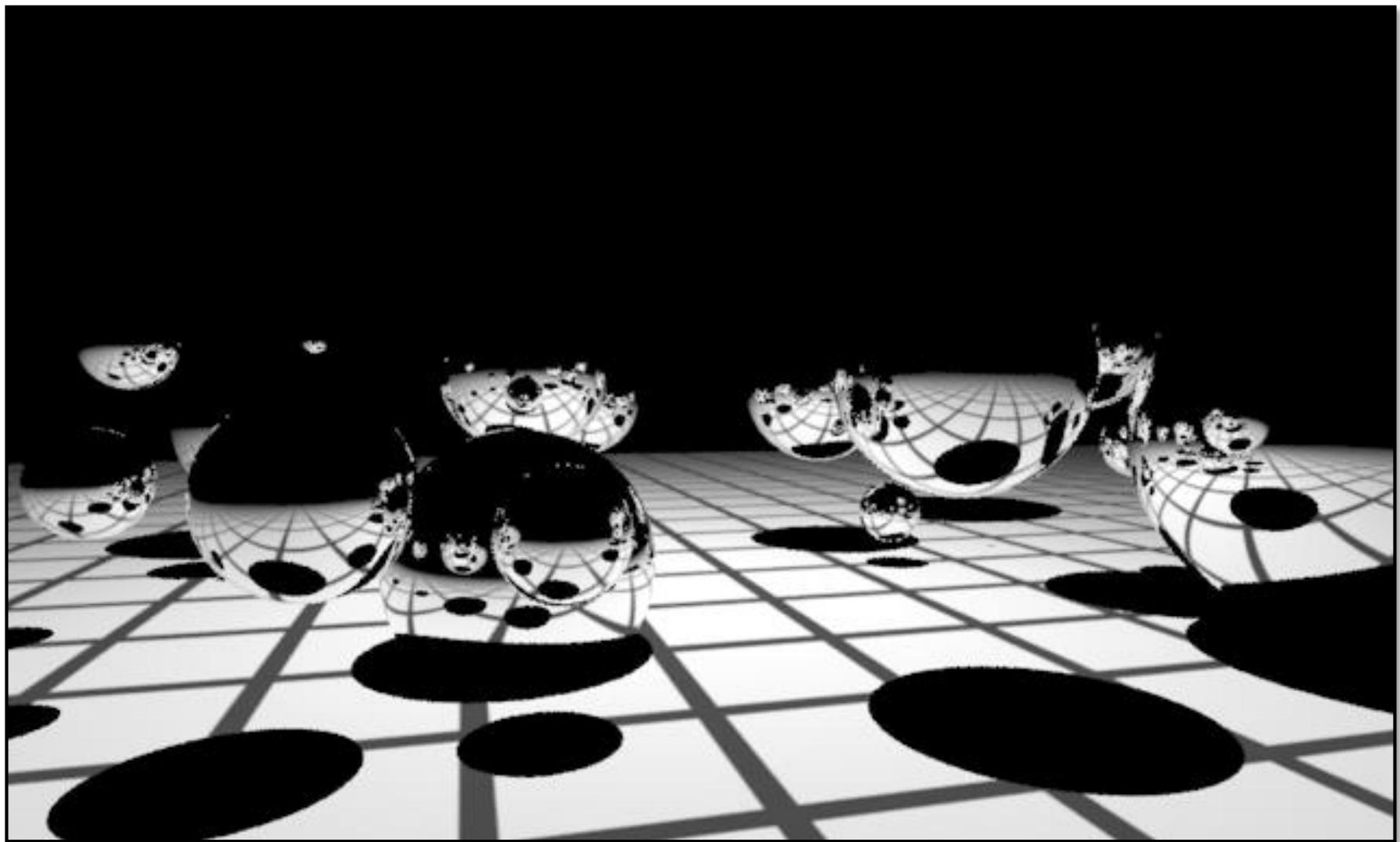
Metallkugeln – Rekursionstiefe 2

► Bild: Pat Hanrahan



Rekursionstiefe 5

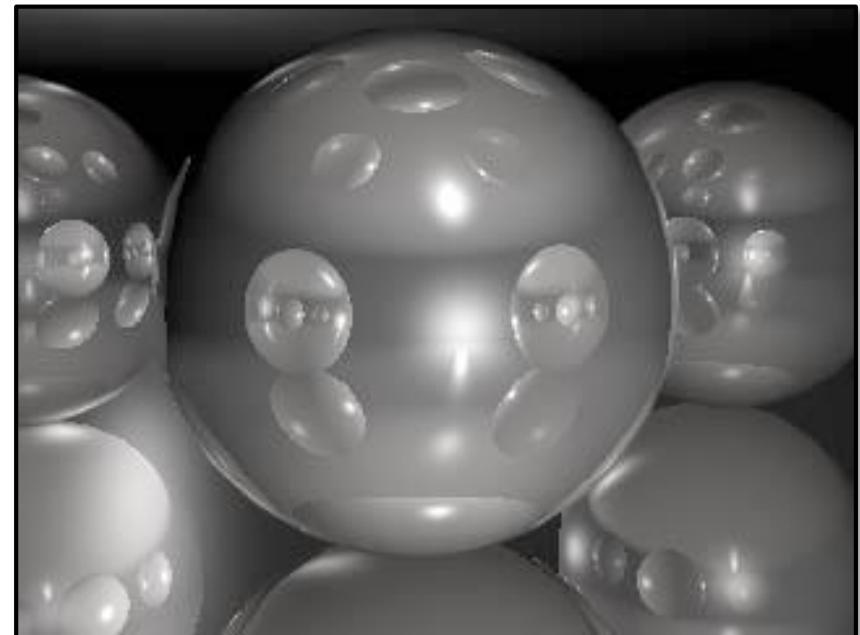
► Bild: Pat Hanrahan



Spiegelungen, Reflexionsstrahlen

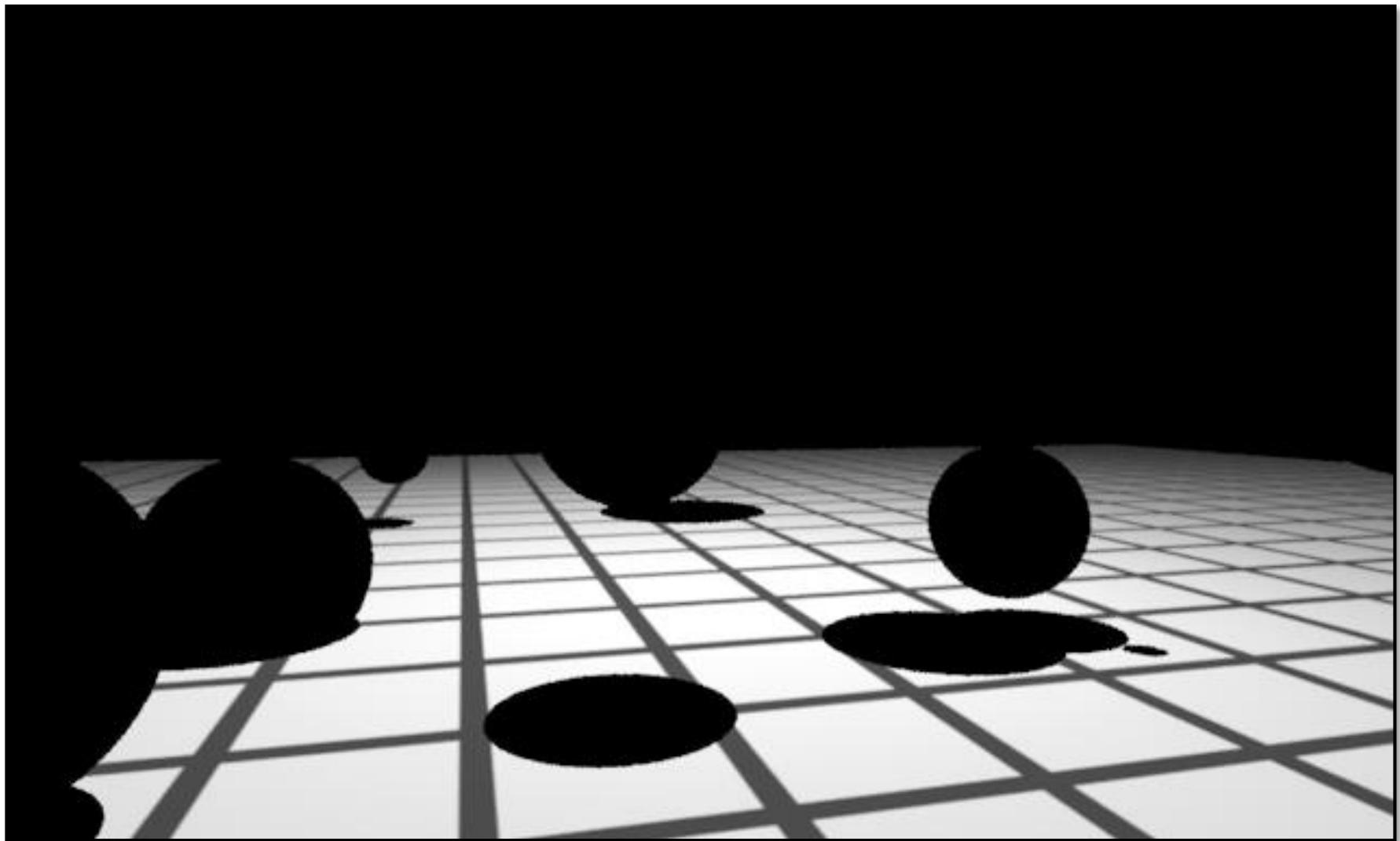
Wie bestimmt man die Rekursionstiefe?

- ▶ vorgegebene maximale Anzahl von Reflexionen, oder
- ▶ Rekursion bis der Beitrag zur Farbe vernachlässigbar wird
 - ▶ Beitrag der 1. Reflexion: $k_{r,1}$
 - ▶ Beitrag der 2. Reflexion: $k_{r,1} \cdot k_{r,2}$
 - ▶ verfolge bis $\prod_i k_{r,i} < \epsilon_r$
 - ▶ Achtung: der tatsächliche Beitrag hängt auch von Intensität der LQ ab



Glaskugeln – Rekursionstiefe 0

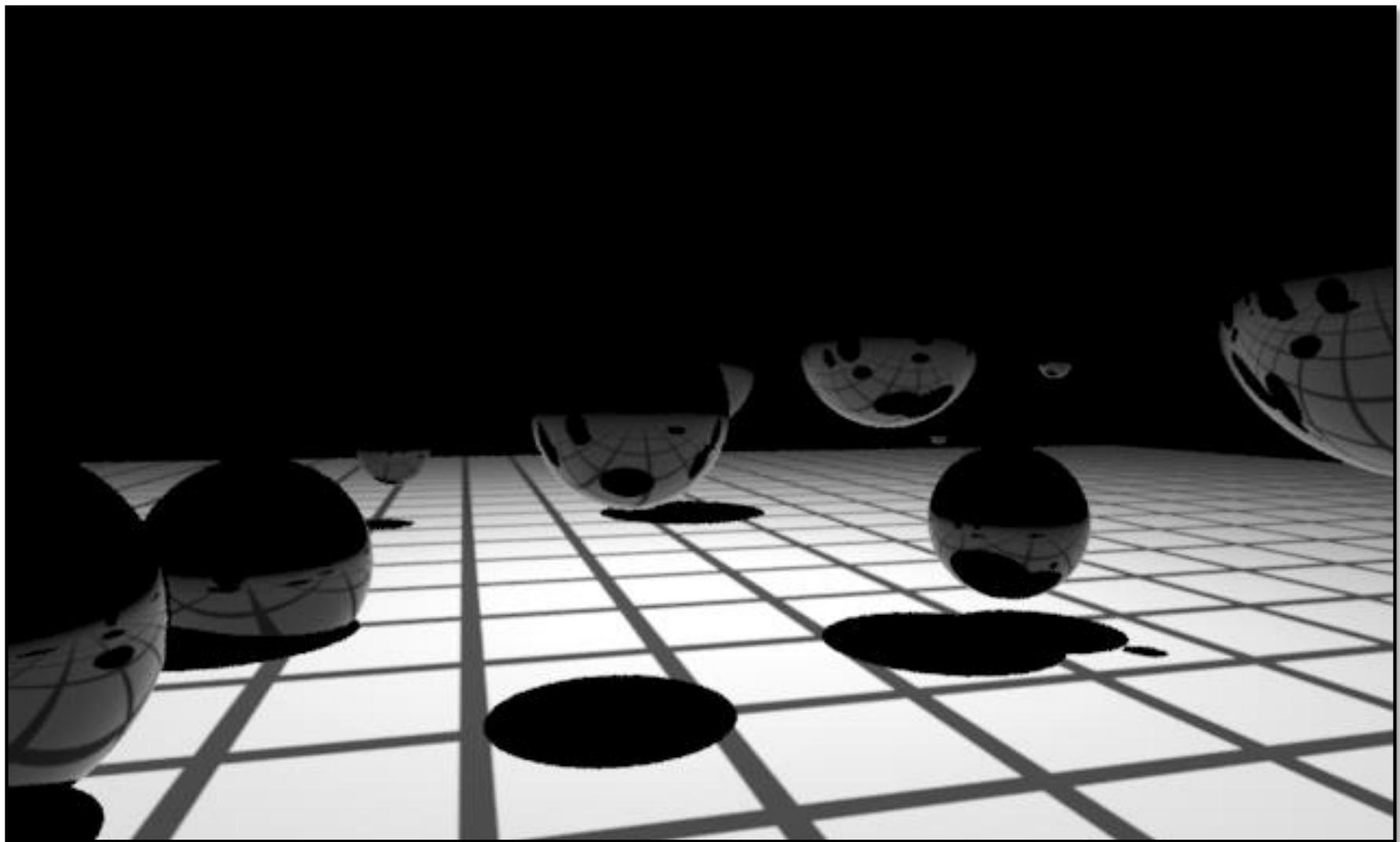
► Bild: Pat Hanrahan



Materialkoeffizienten der Glaskugeln: $k_a = k_d = k_s = 0$ und $k_r, k_t > 0$

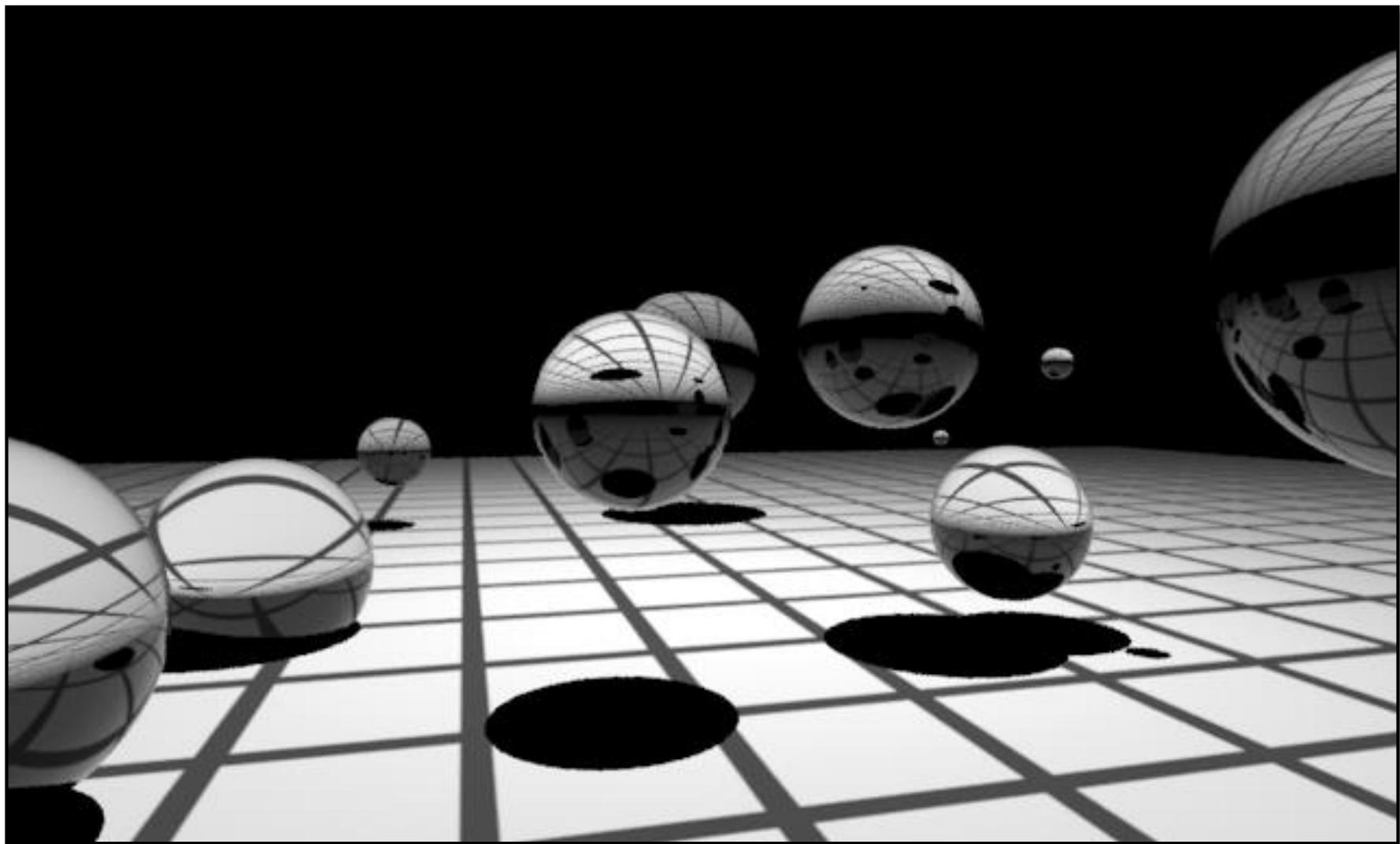
Glaskugeln – Rekursionstiefe 1

► Bild: Pat Hanrahan



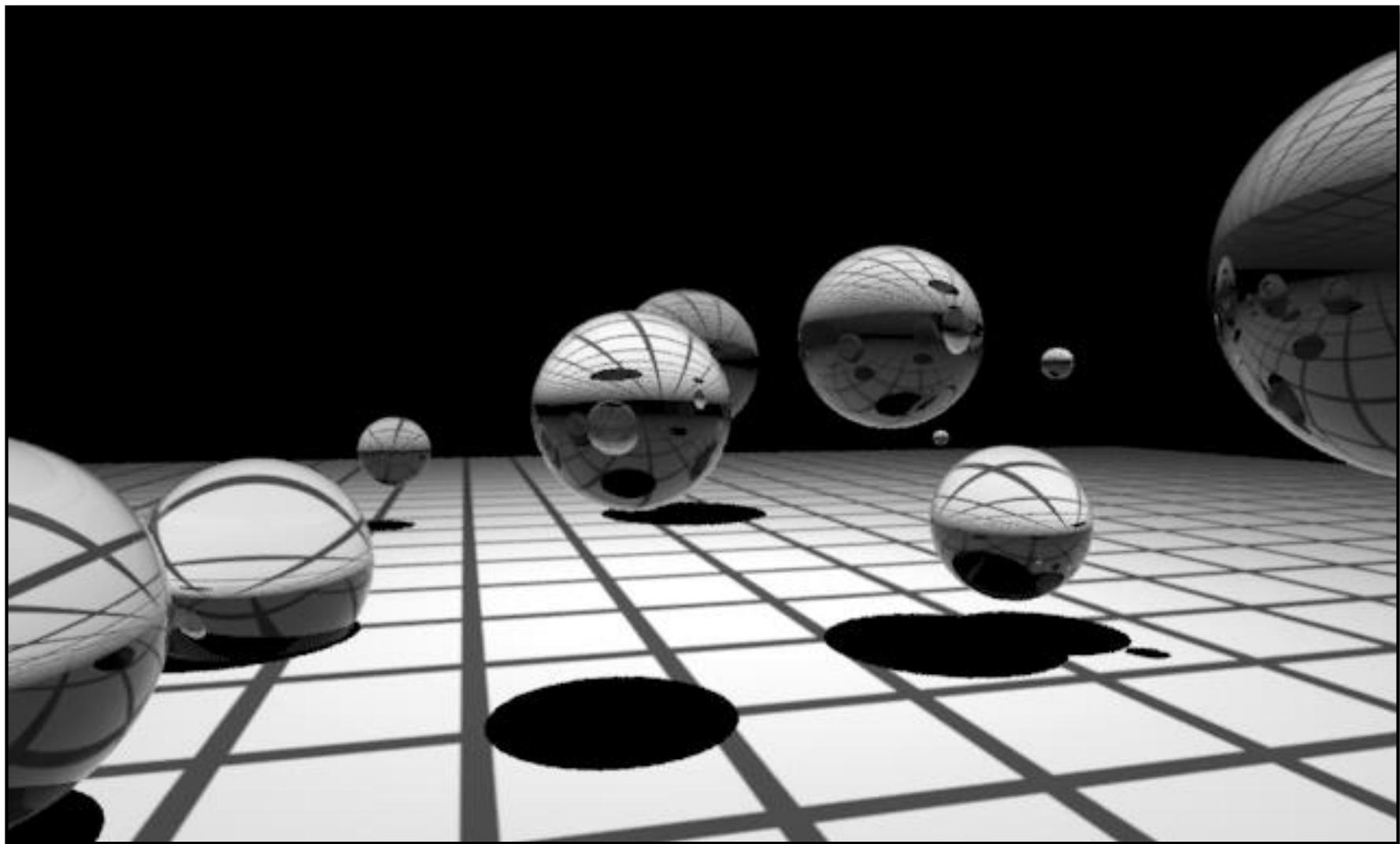
Glaskugeln – Rekursionstiefe 2

► Bild: Pat Hanrahan



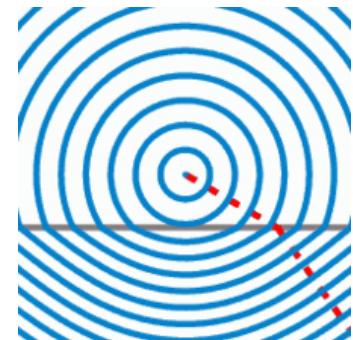
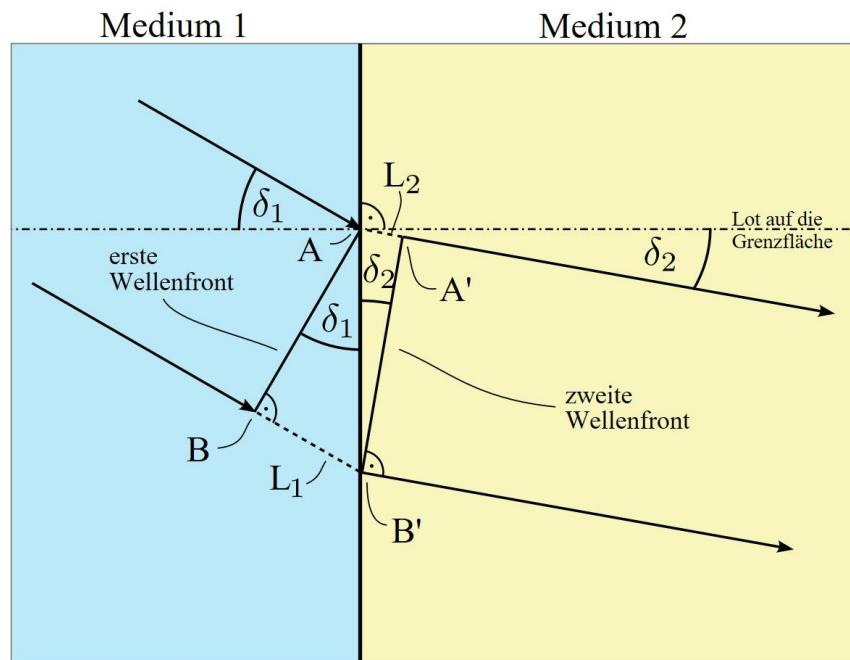
Glaskugeln – Rekursionstiefe 5

► Bild: Pat Hanrahan



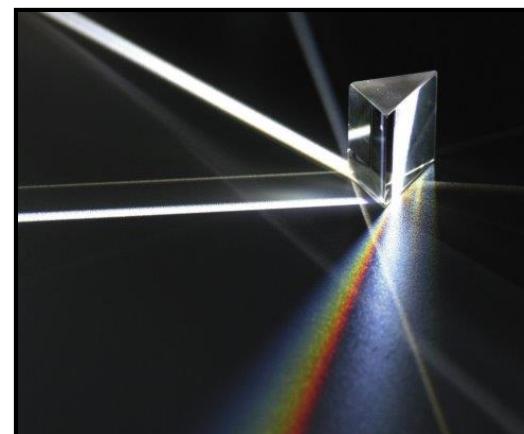
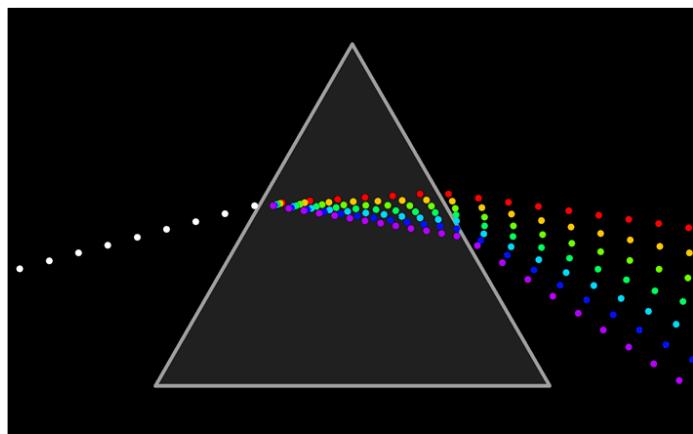
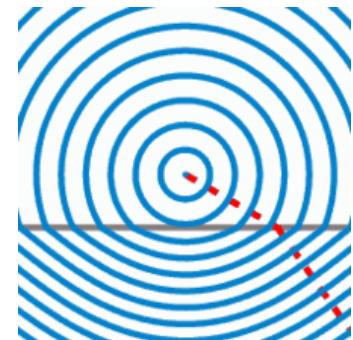
Snellsches Brechungsgesetz

- ▶ ... beschreibt die Richtungsänderung einer Welle (Licht) beim Übergang von einem Medium in eines mit anderer Brechzahl
- ▶ Licht bewegt sich in unterschiedlichen Medien unterschiedlich schnell
 - ▶ (reelle) Brechzahl $\eta = c_0/c_\eta$
 - ▶ c_0, c_η Phasengeschwindigkeit im Vakuum/Medium
 - ▶ Ausbreitungsrichtung senkrecht zur Wellenfront



Snellsches Brechungsgesetz

- ▶ ... beschreibt die Richtungsänderung einer Welle (Licht) beim Übergang von einem Medium in eines mit anderer Brechzahl
- ▶ Licht bewegt sich in unterschiedlichen Medien unterschiedlich schnell
 - ▶ (reelle) Brechzahl $\eta = c_0/c_\eta$
 - ▶ c_0, c_η Phasengeschwindigkeit im Vakuum/Medium
 - ▶ Ausbreitungsrichtung senkrecht zur Wellenfront
- ▶ Brechzahl ist i.d.R. wellenlängenabhängig $\eta(\lambda)$
 - ▶ wellenlängenabhängige Ausbreitungsgeschwindigkeit von Wellen (Dispersion) → Aufspaltung in unterschiedliche Wellenlängen

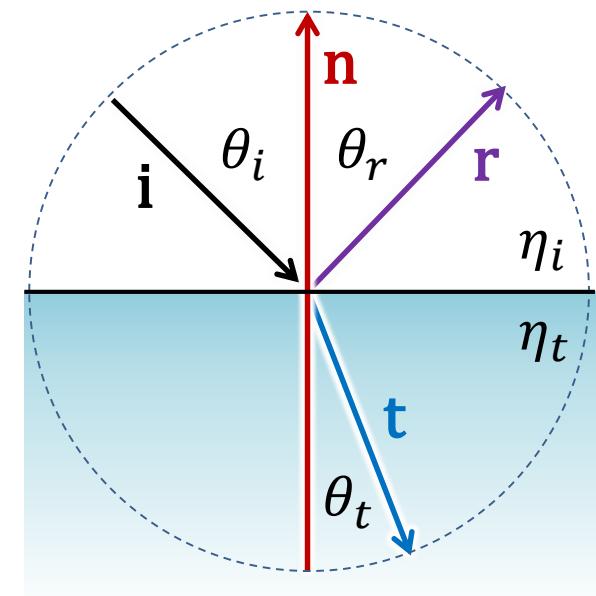


Snellsches Brechungsgesetz

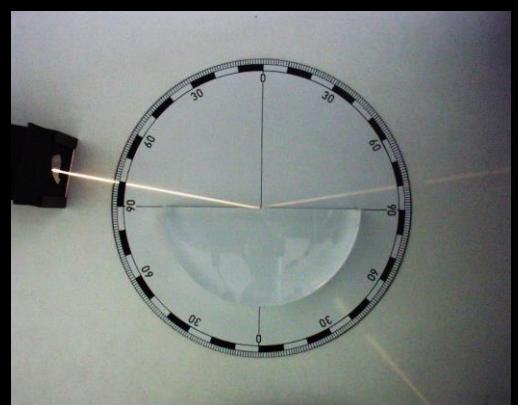
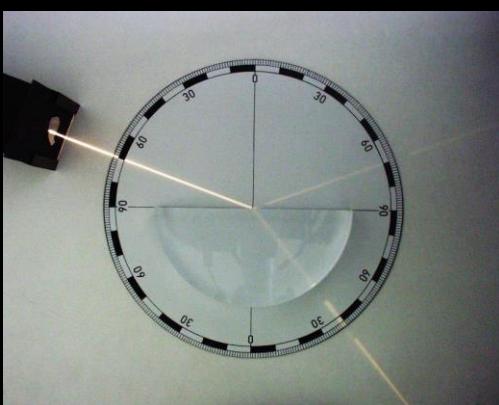
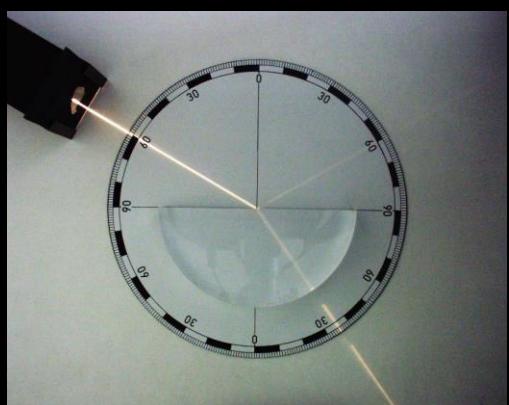
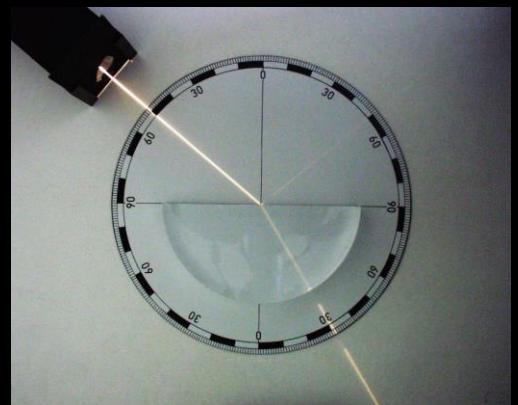
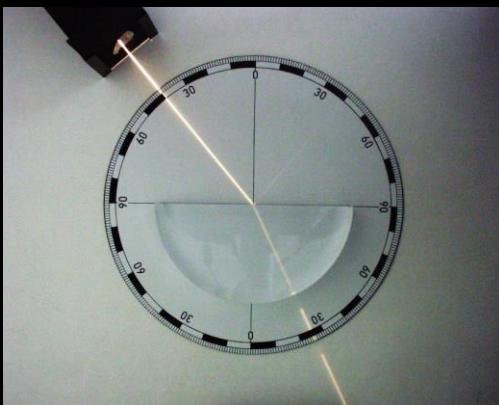
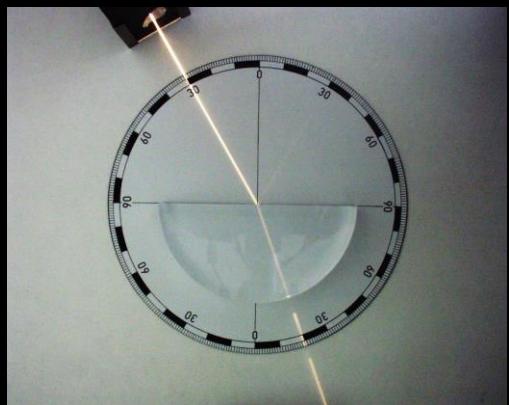
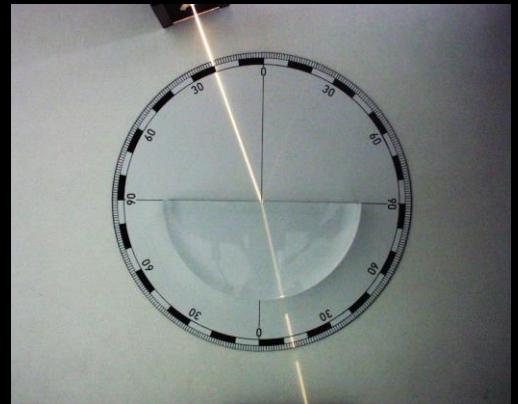
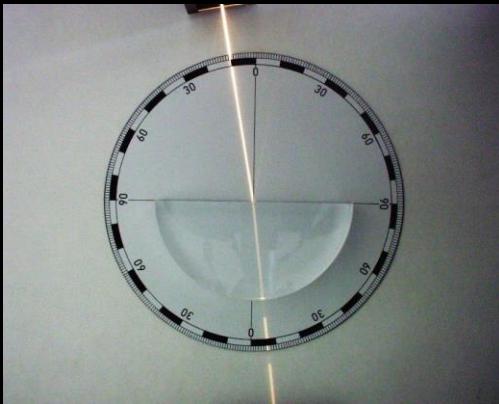
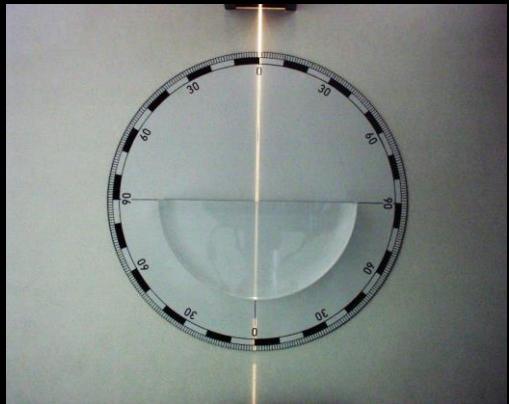
- ▶ Brechungsgesetz: $\eta_i \sin \theta_i = \eta_t \sin \theta_t$
- ▶ Brechung bei Übergang ins optisch dichtere Medium ($\eta_t > \eta_i$) zum Lot hin
- ▶ beim Übergang ins optisch dünnere Medium vom Lot weg
- ▶ Fresnel-Effekt: Verteilung der Strahldichte
- ▶ Grenzwinkel der Totalreflexion
 - ▶ Licht wird an der Grenzfläche von dichterem zu dünnerem Medium vollständig reflektiert: $\sin \theta_c = \eta_i / \eta_t$
- ▶ Transmissionsvektor **t** (Herleitung siehe Fund. of Computer Graphics)

$$\begin{aligned}
 \mathbf{t} &= -\frac{\sin \theta_t}{\sin \theta_i} (\mathbf{i} - \mathbf{n} \cos \theta_i) - \mathbf{n} \cos \theta_t \\
 &= -\frac{\eta_i}{\eta_t} \mathbf{i} + \left(\frac{\eta_i}{\eta_t} \cos \theta_i - \sqrt{1 - \left(\frac{\eta_i}{\eta_t}\right)^2 (1 - \cos^2 \theta_i)} \right) \mathbf{n}
 \end{aligned}$$

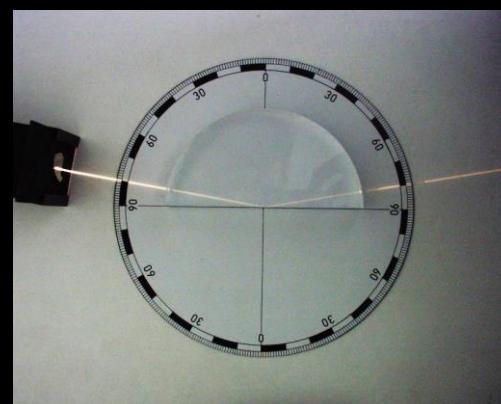
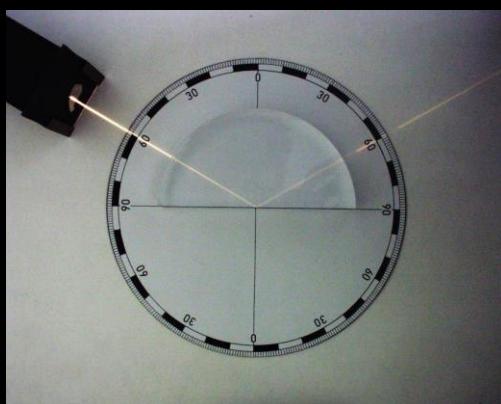
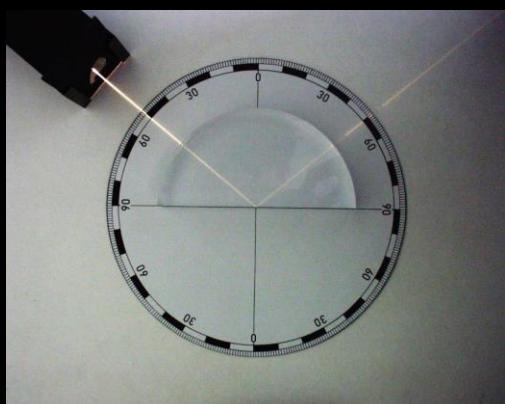
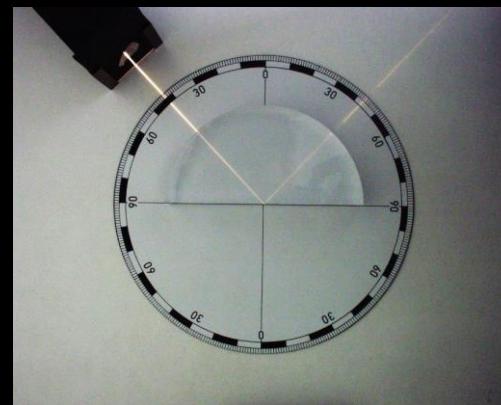
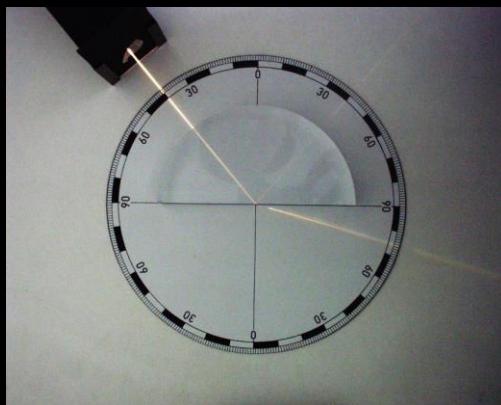
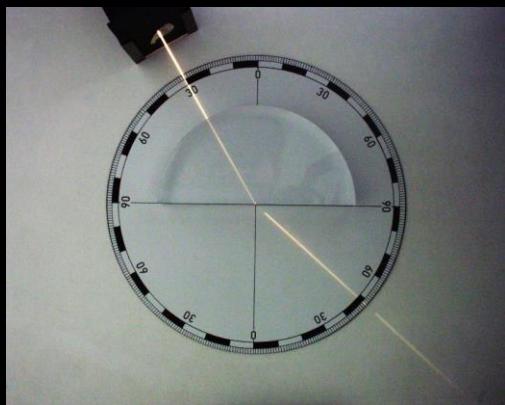
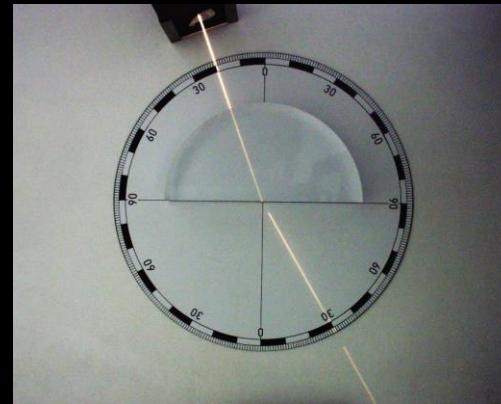
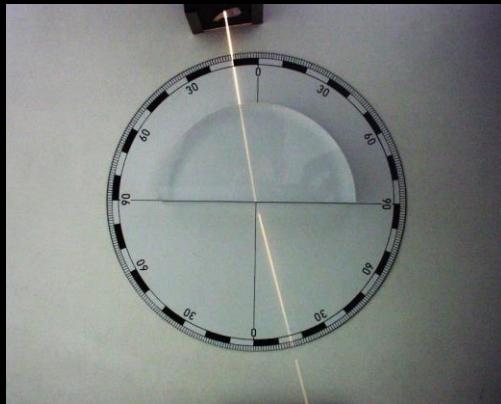
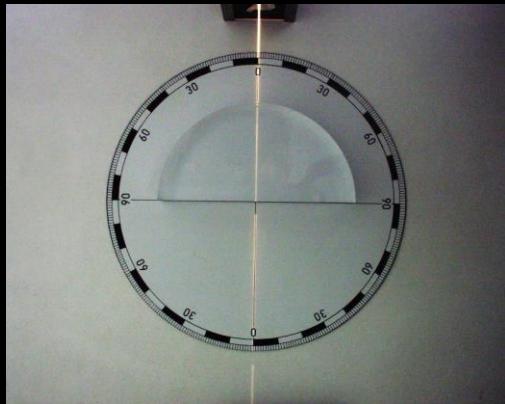
- ▶ **i, n, r** und **t** liegen in einer Ebene



Fresnel-Effekt, Lichtbrechung und Reflexion



Totalreflexion



Totalreflexion und Fresneleffekt

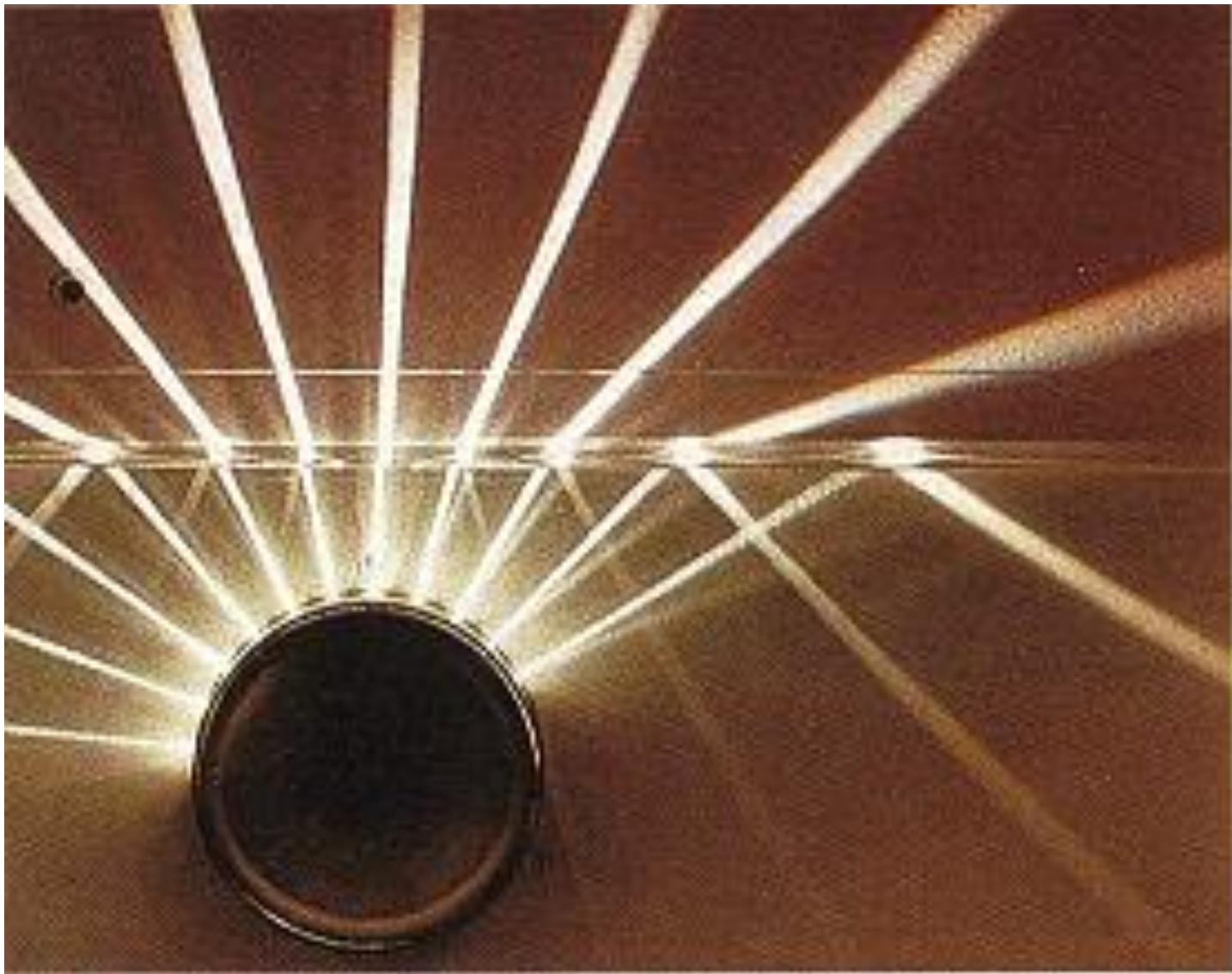


Bild: http://www.dieter-heidorn.de/Physik/VS/Optik/K04_Brechung/K04_Brechung.html

Refraktion: Snell's Window, Optical Manhole

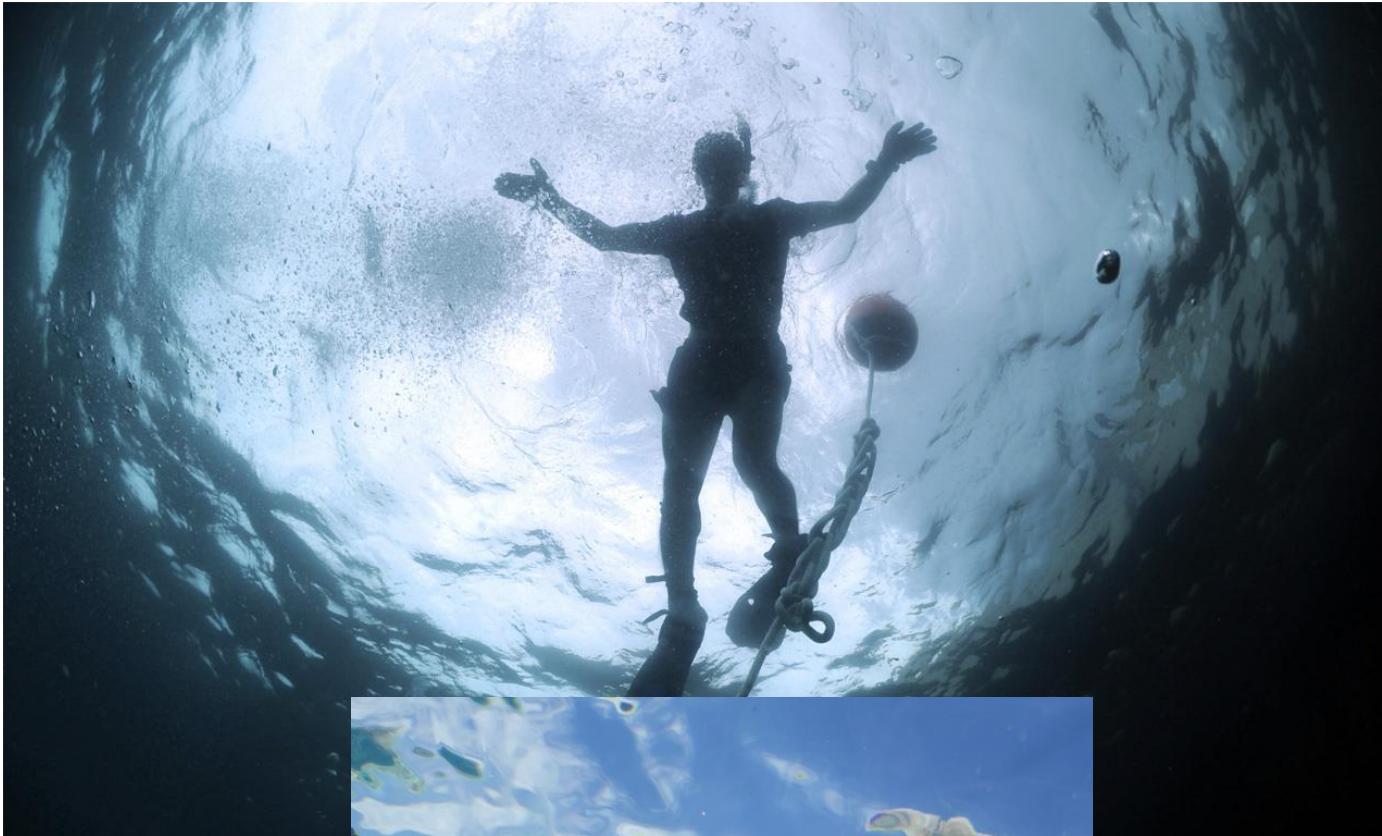


Bild: Wikipedia

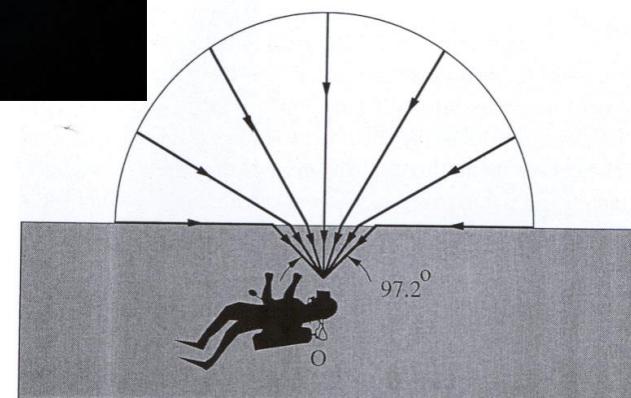
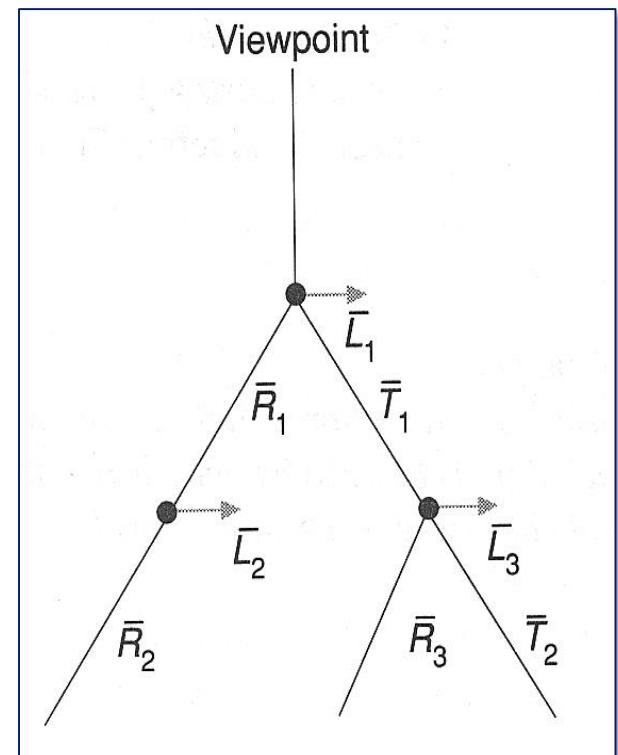


Fig. 3.7B The optical manhole. Light from the horizon (angle of incidence = 90°) is refracted downward at an angle of 48.6° . This compresses the sky into a circle with a diameter of 97.2° instead of its usual 180° .

Raytracing Pseudocode

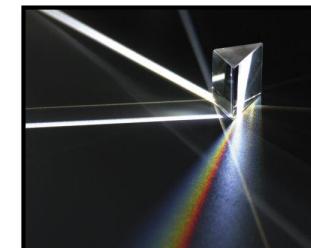
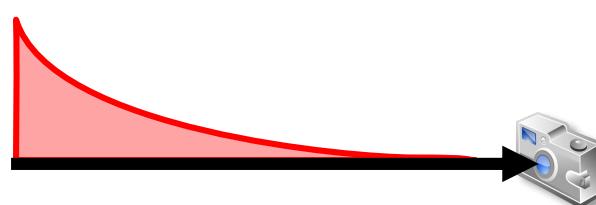
```
vec3 raytrace( Ray *ray, ... ) {  
    vec3 color = 0.0f;  
  
    if ( !cast( ray, FLOAT_MAX ) )  
        return color;  
  
    for ( jede Lichtquelle )  
        color += computeDirectLight( ... );  
  
    if ( Fläche ist spiegelnd ) {  
        // berechne Reflexionsstrahl  
        Ray reflect = ...;  
        color += i.kr * raytrace( reflect, ... );  
    }  
  
    if ( Fläche ist (semi-)transparent ) {  
        // berechne Transmissionsstrahl  
        Ray refract = ...;  
        if ( Totalreflexion )  
            color += i.kr * raytrace( reflect, ... );  
        else  
            color += i.kt * raytrace( refract, ... );  
    }  
  
    return color;  
}
```

Diese rekursive Formulierung heißt
Whitted-Style Raytracing
(nach Turner Whitted)



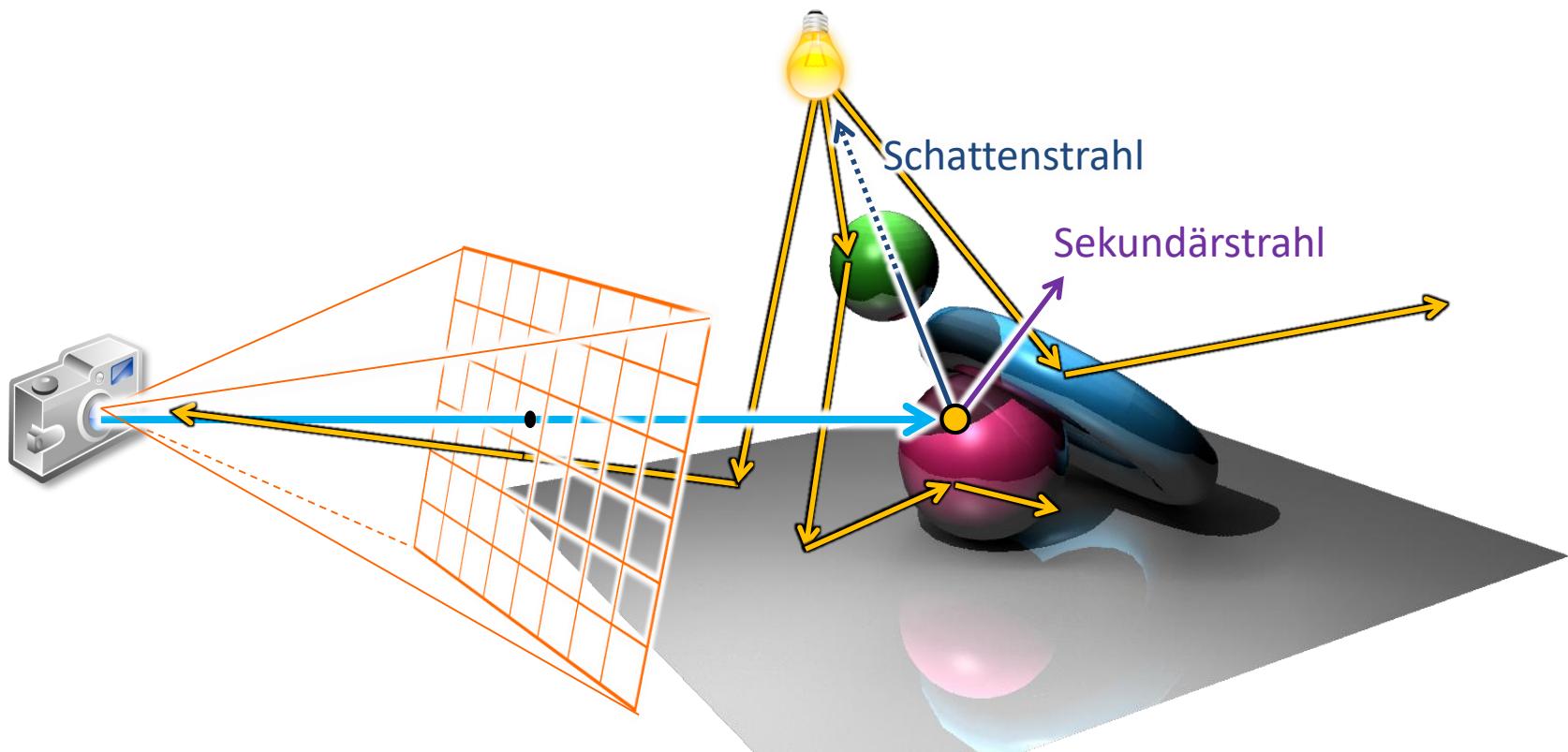
Transmission

- ▶ Transparentes Material mit keiner bzw. konstanter Abschwächung (links)
- ▶ exponentieller Abfall durch Absorption $e^{-\sigma t \cdot s}$ (Beer's Law, Mitte)
- ▶ zus. Dispersion mit einem Transmissionsstrahl pro Wellenlänge (rechts)



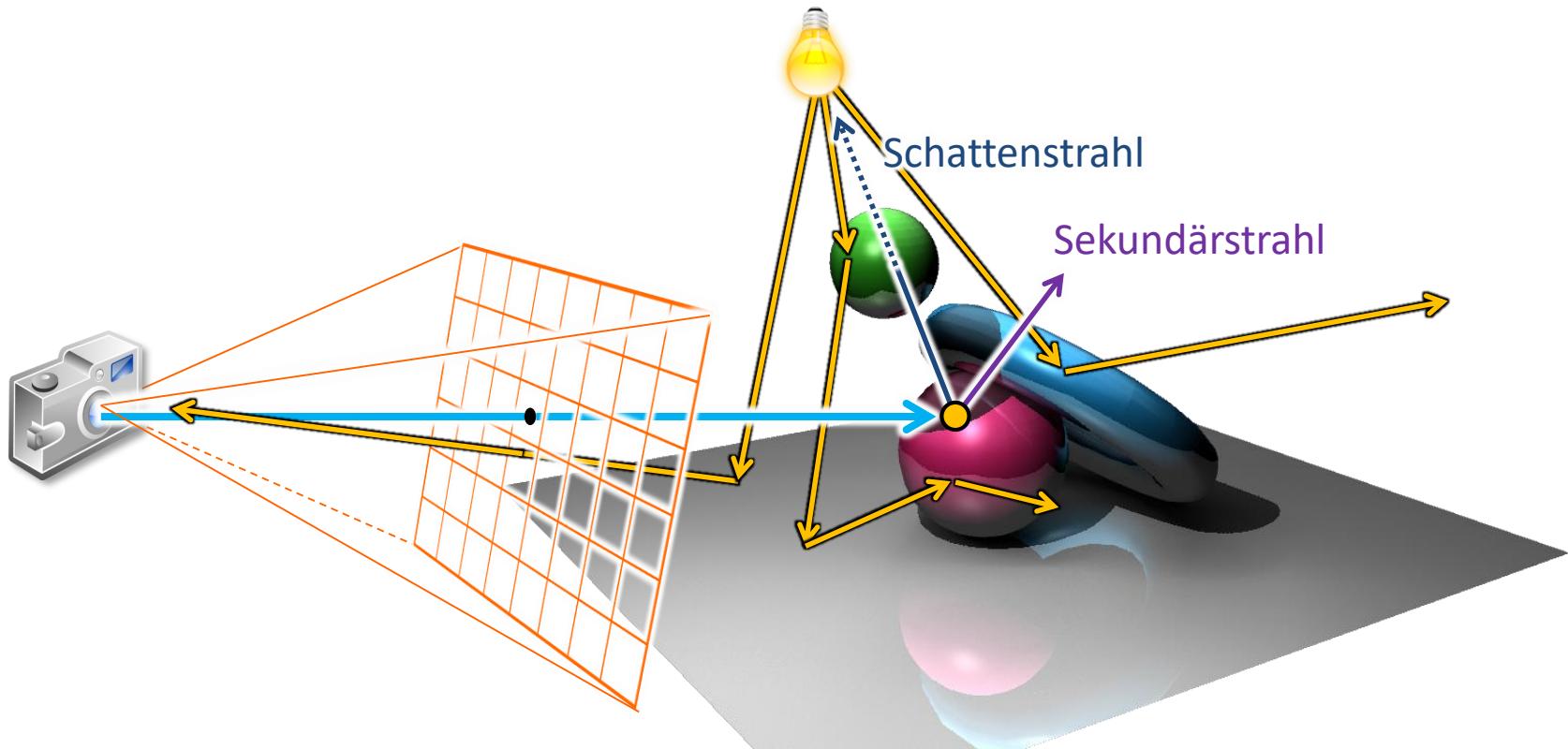
Zusammenfassung

- ▶ Raytracing ist ein Verfahren zur Bildsynthese (engl. Rendering)
- ▶ geometrische Optik: Licht breitet sich entlang von Strahlen aus
- ▶ die Ausbreitung des Lichts wird – von der Kamera aus – zurückverfolgt



Zusammenfassung: Schritte

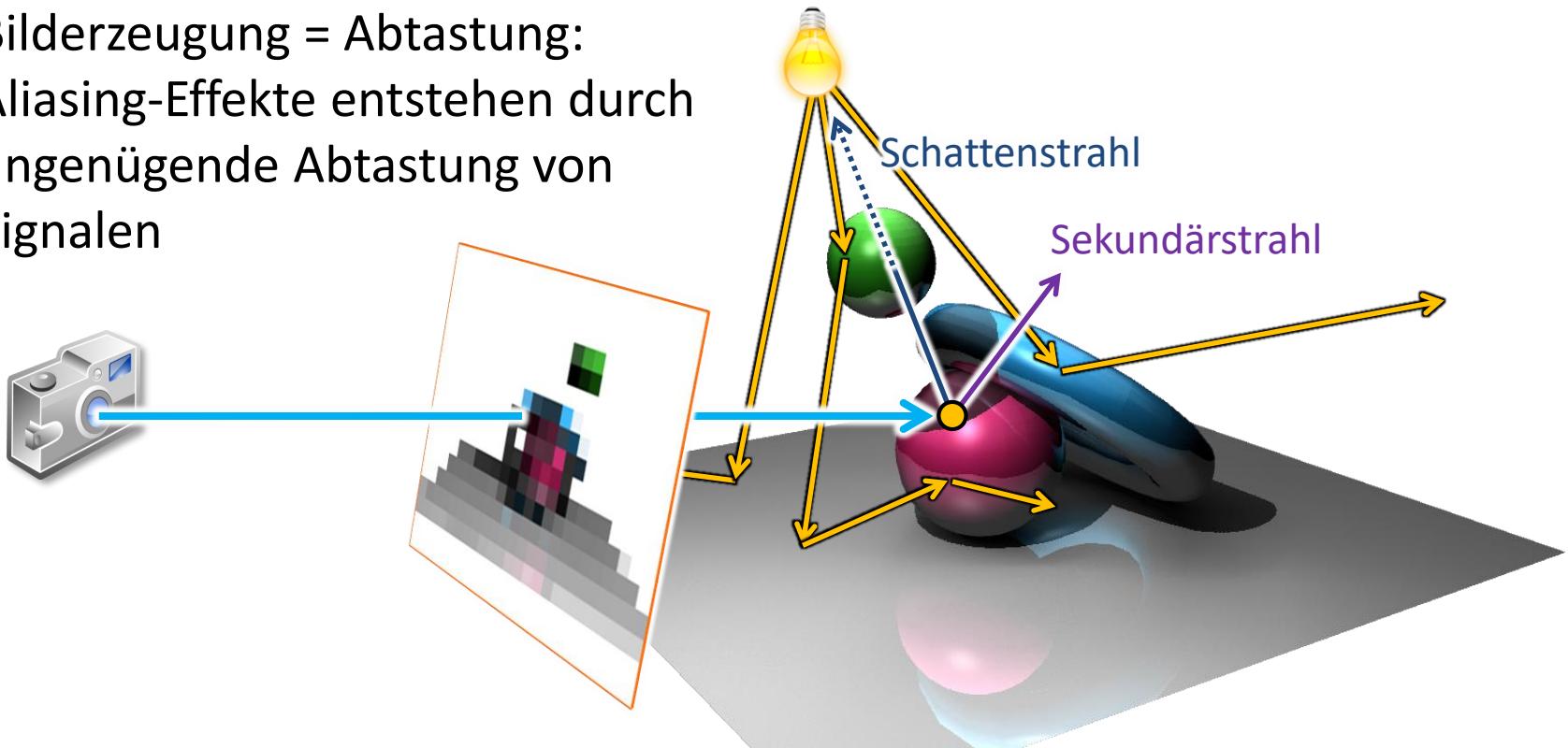
- ▶ Erzeugung der Sichtstrahlen durch jeden Pixel (ray generation)
- ▶ Schnittberechnung (ray casting)
- ▶ Schattierung und Beleuchtungsberechnung (shading):
lokale Beleuchtungsberechnung und **globale** Schattenstrahlen
- ▶ rekursive Sekundärstrahlen für Spiegelung und Transmission



Raytracing und Aliasing

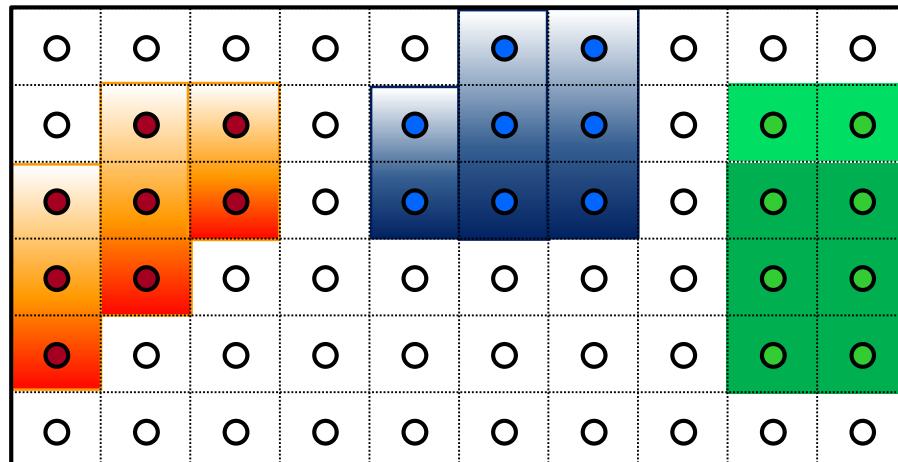
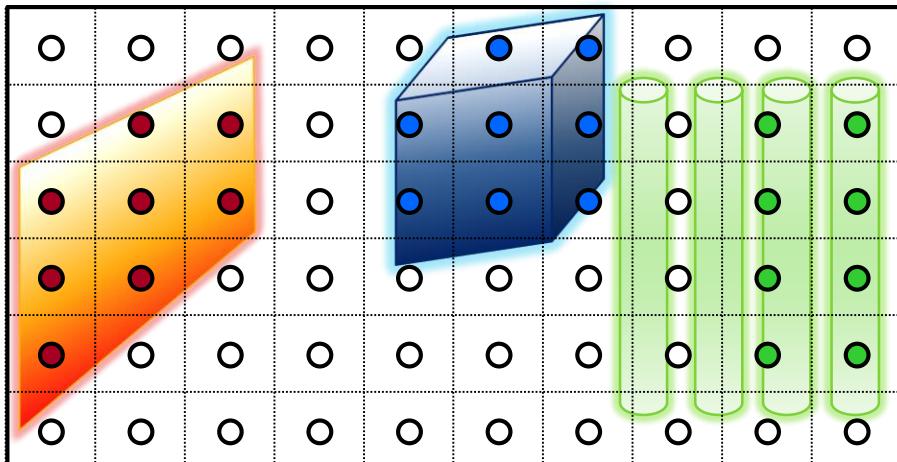
Zusammenfassung: Schritte

- ▶ Erzeugung der Sichtstrahlen durch jeden Pixel (ray generation)
- ▶ Schnittberechnung (ray casting)
- ▶ Schattierung und Beleuchtungsberechnung (shading):
lokale Beleuchtungsberechnung und **globale** Schattenstrahlen
- ▶ rekursive Sekundärstrahlen für Spiegelung und Transmission
- ▶ Bilderzeugung = Abtastung:
Aliasing-Effekte entstehen durch
ungenügende Abtastung von
Signalen



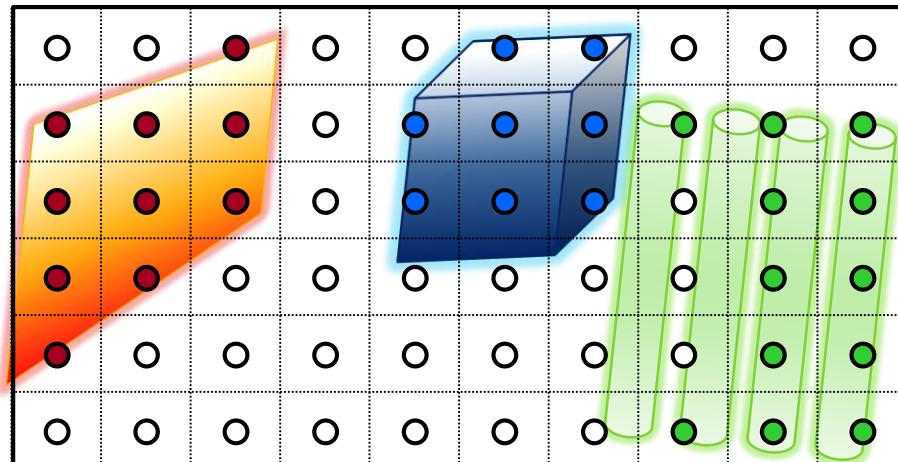
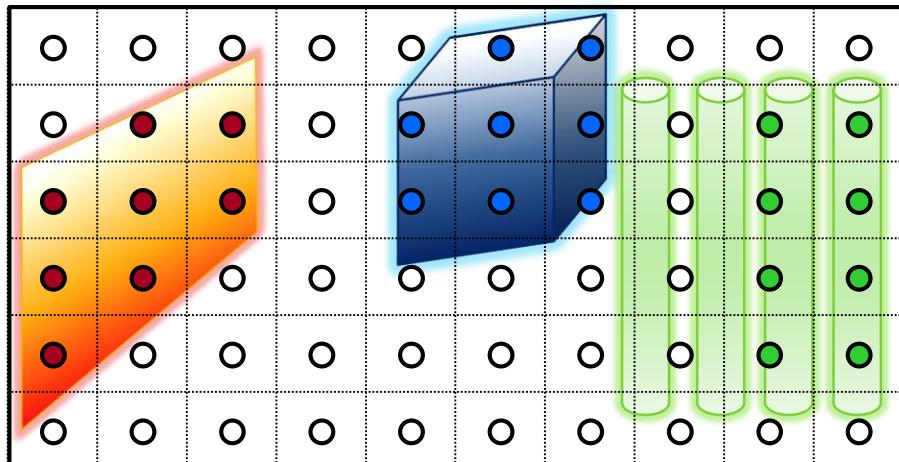
Raytracing und Aliasing

- das Bild einer 3D-Szene ist ein zweidimensionales Signal
 - ▶ äquidistante, diskrete Abtastung eines Bildsignals
 - ▶ Farbe eines Pixels bestimmt durch Strahl durch seinen Mittelpunkt, d.h. an den Pixelmittelpunkten tasten wir das Bildsignal ab
(Vorstellung: stückweise konstante Funktion)
 - ▶ das Bildsignal ist nicht bandbegrenzt (gilt insb. für die Objektkanten)
 - ▶ durch quadratische Form der Pixel sehen wir zusätzlich Treppenstufen („Jaggies“) bei der Rekonstruktion des Signals



Raytracing und Aliasing

- das Bild einer 3D-Szene ist ein zweidimensionales Signal
 - ▶ äquidistante, diskrete Abtastung eines Bildsignals
 - ▶ Farbe eines Pixels bestimmt durch Strahl durch seinen Mittelpunkt, d.h. an den Pixelmittelpunkten tasten wir das Bildsignal ab
(Vorstellung: stückweise konstante Funktion)
 - ▶ das Bildsignal ist nicht bandbegrenzt (gilt insb. für die Objektkanten)
 - ▶ durch quadratische Form der Pixel sehen wir zusätzlich Treppenstufen („Jaggies“) bei der Rekonstruktion des Signals



Bild, Signal und Abtastung

- Graustufenbild einer Funktion $f(d) = \cos ad^2$ mit $d \in [-1,1]^2$, $a \in \mathbb{R}$

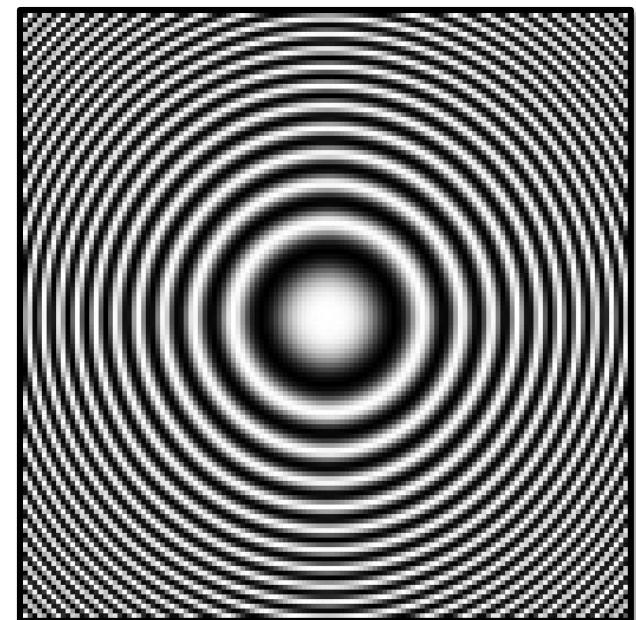
```
#define WIDTH 128
#define HEIGHT 128

unsigned char img[ WIDTH * HEIGHT ];

for ( y = 0; y < HEIGHT; y++ ) {
    for ( x = 0; x < WIDTH; x++ ) {
        float dx = (float)( 2*x - WIDTH ) / (float)WIDTH;
        float dy = (float)( 2*y - HEIGHT ) / (float)HEIGHT;
        float d2 = dx * dx + dy * dy;

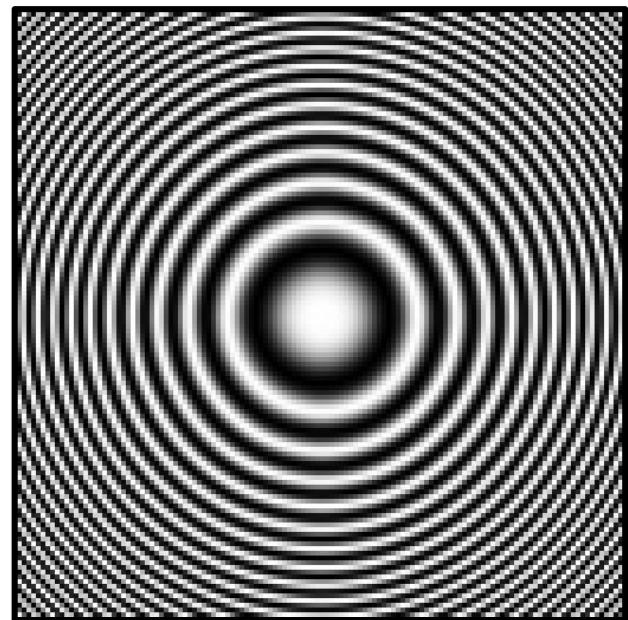
        // f(d) mit a = 64
        float f = cosf( 64.0f * d2 );

        // abbilden von f(d) auf [0..255]
        img[ x + y * WIDTH ] =
            ( f + 1.0f ) * 127.5f;
    }
}
```

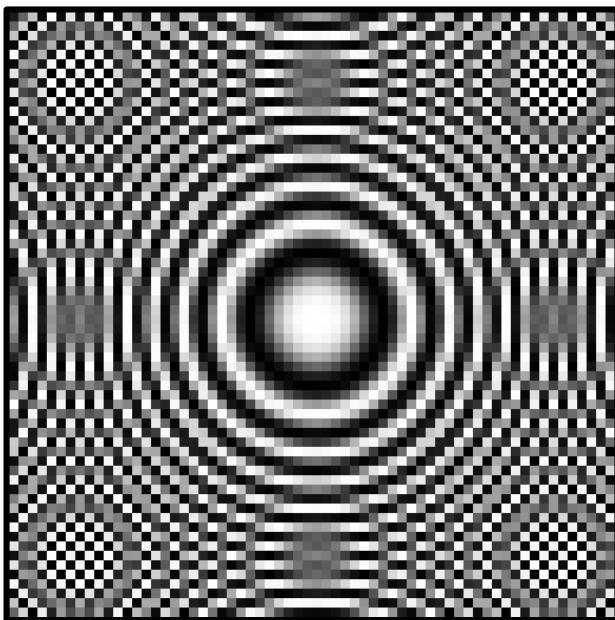


Bild, Signal und Abtastung

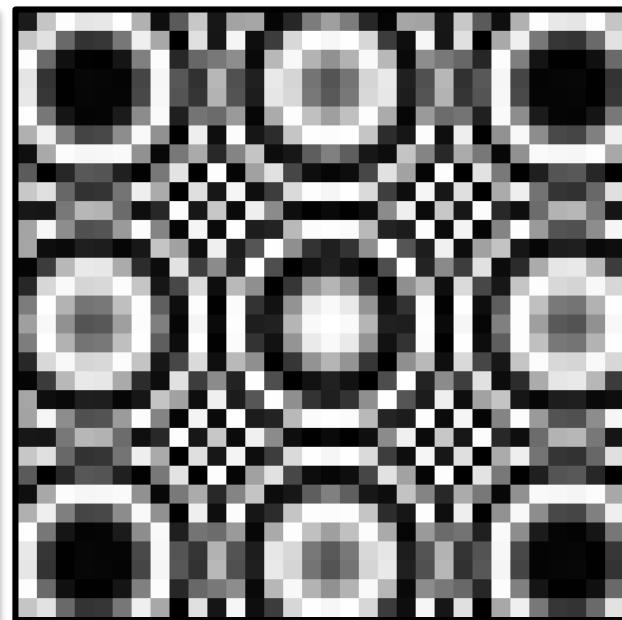
- ▶ Graustufenbild einer Funktion $f(d) = \cos ad^2$ mit $d \in [-1,1]^2$, $a \in \mathbb{R}$
- ▶ Aliasing-Effekte entstehen durch ungenügende Abtastung von Signalen



WIDTH=HEIGHT=128

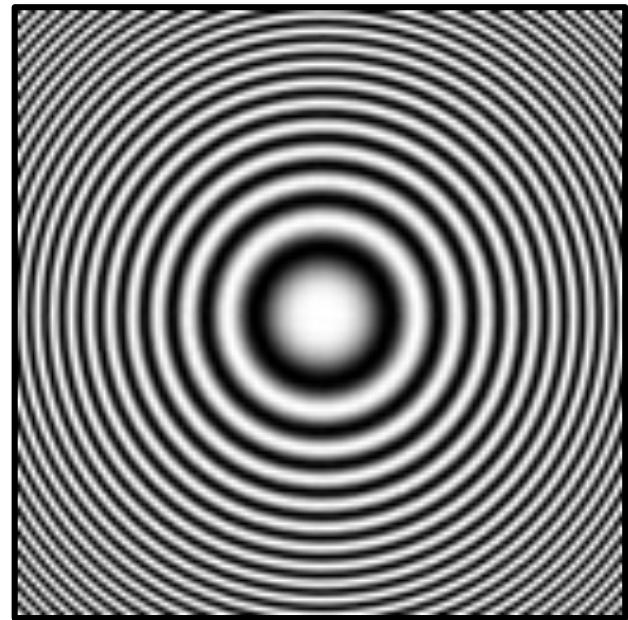


WIDTH=HEIGHT=64

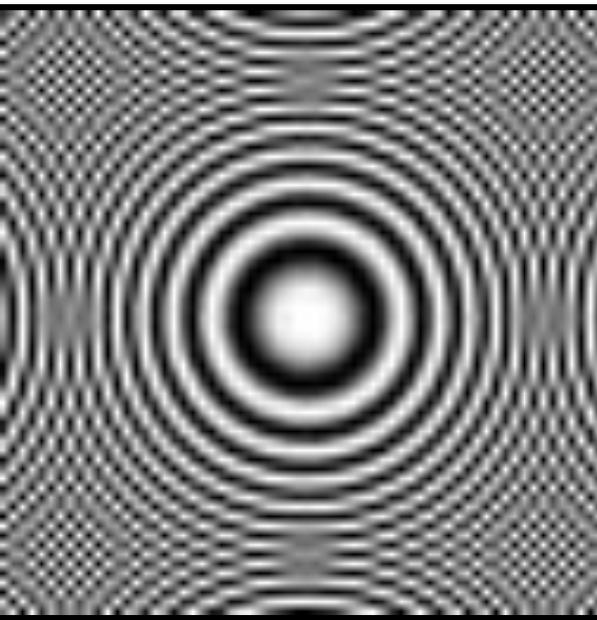


WIDTH=HEIGHT=32

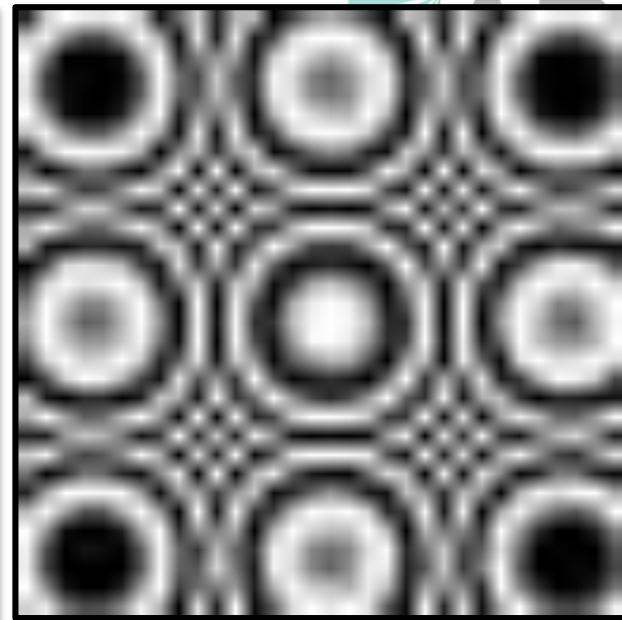
Unterschiedliche Rekonstruktionsfilter



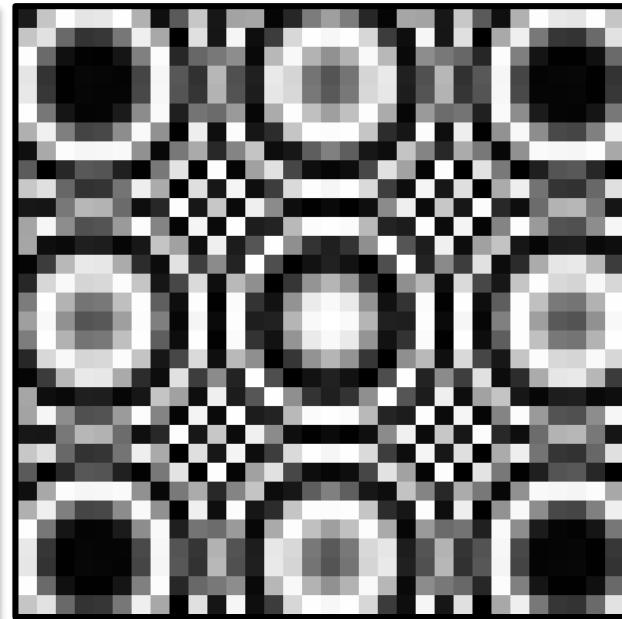
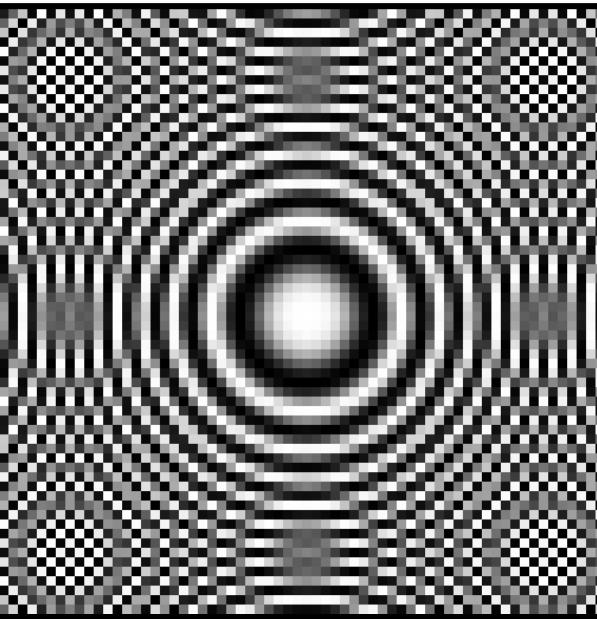
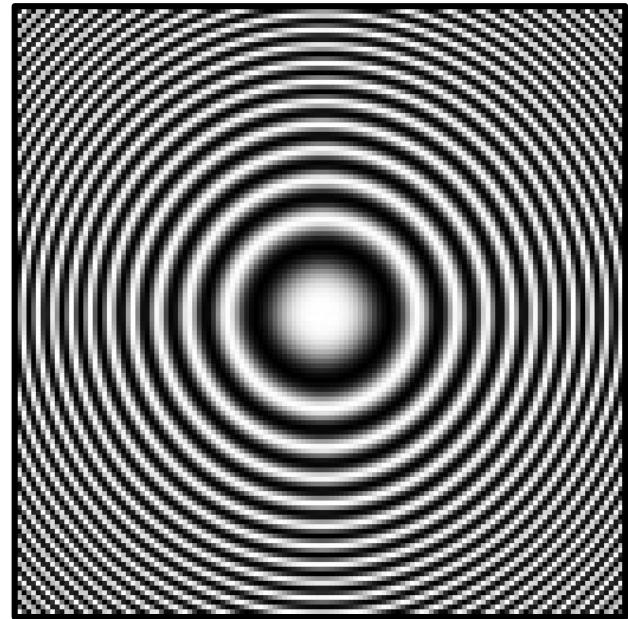
WIDTH=HEIGHT=128



WIDTH=HEIGHT=64

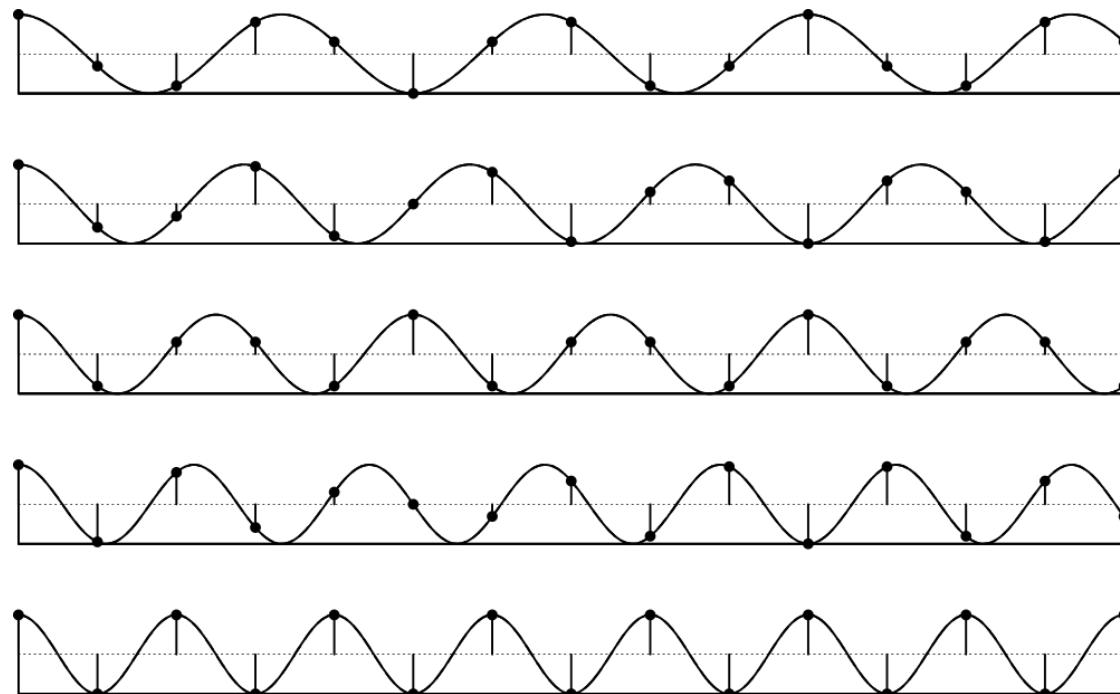


WIDTH=HEIGHT=32



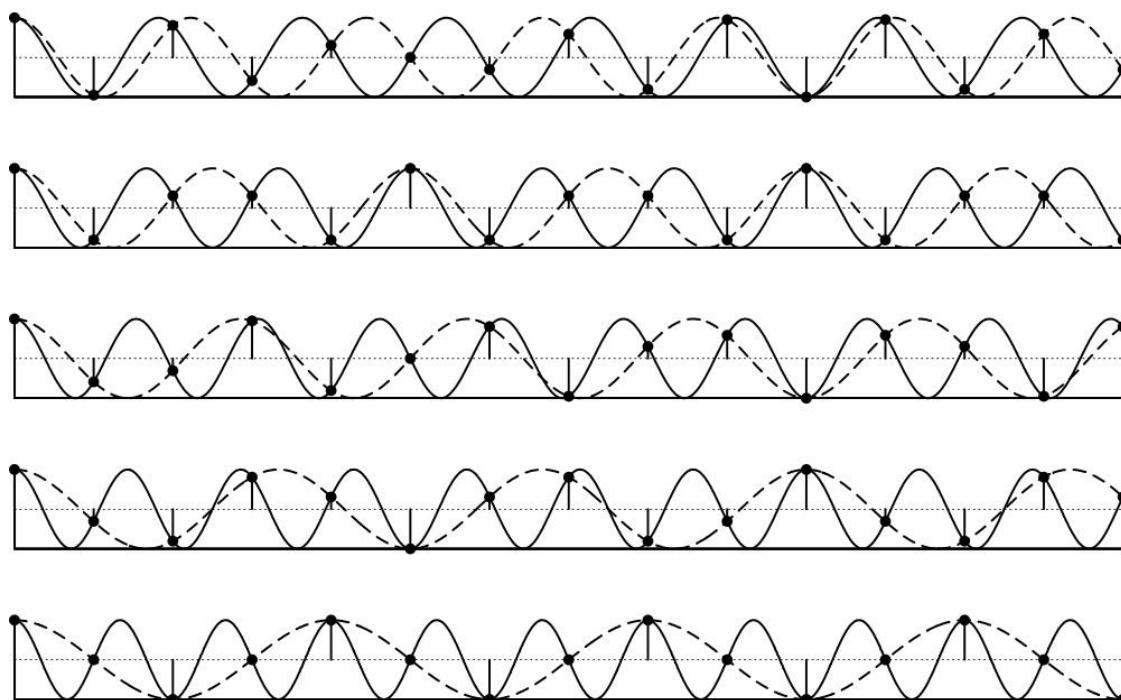
Bild, Signal und Abtastung

- ▶ Nyquist-Shannon-Abtasttheorem:
ein kontinuierliches, bandbegrenztes Signal mit einer max. Frequenz f_{max} muss mit einer Frequenz größer als $2f_{max}$ abgetastet werden, damit aus dem diskreten Signal das Ursprungssignal exakt rekonstruiert werden kann
- ▶ Beispiel: $f_{abtast} > 2f_{max}$ (letzte Zeile: Grenzfall $f_{abtast} = 2f_{max}$)



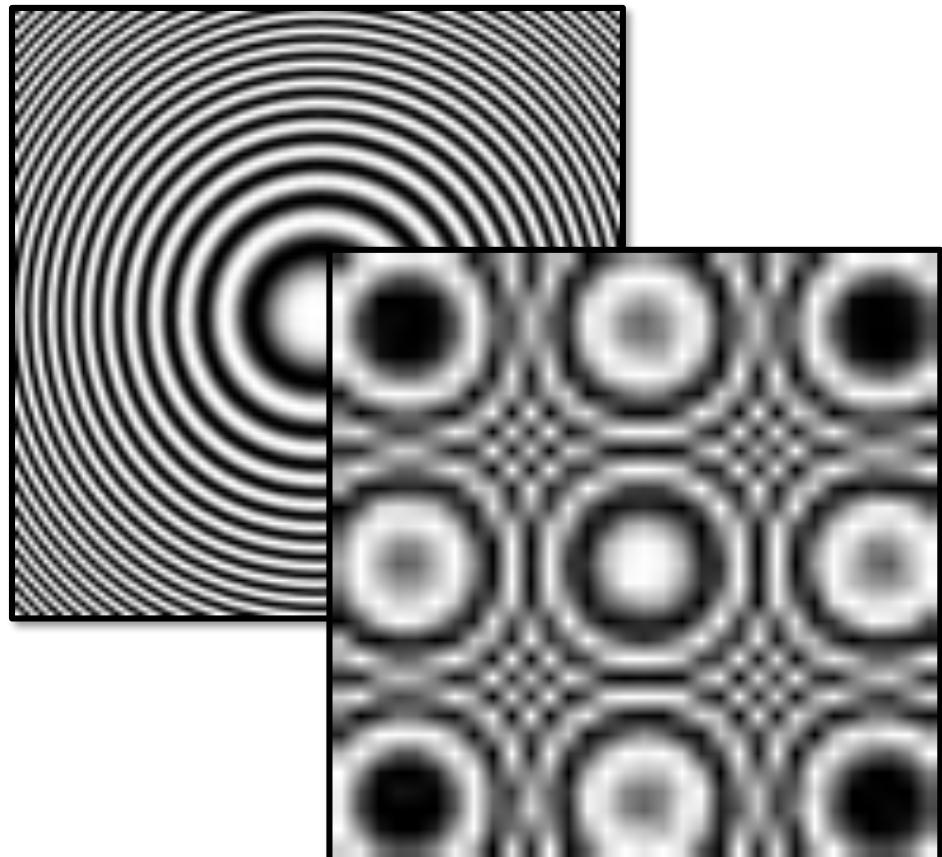
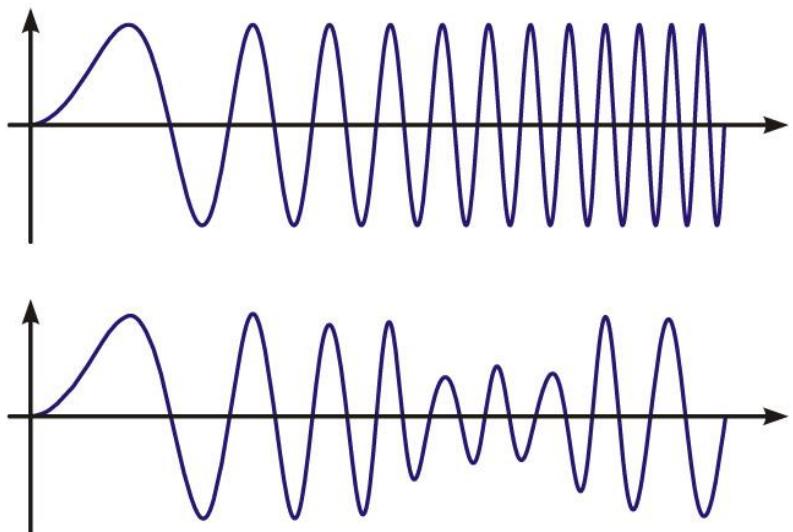
Bild, Signal und Abtastung

- ▶ Nyquist-Shannon-Abtasttheorem:
ein kontinuierliches, bandbegrenztes Signal mit einer max. Frequenz f_{max} muss mit einer Frequenz größer als $2f_{max}$ abgetastet werden, damit aus dem diskreten Signal das Ursprungssignal exakt rekonstruiert werden kann
- ▶ Beispiel: $f_{abtast} < 2f_{max}$ — orig. Signal ----- rekonstr. Signal



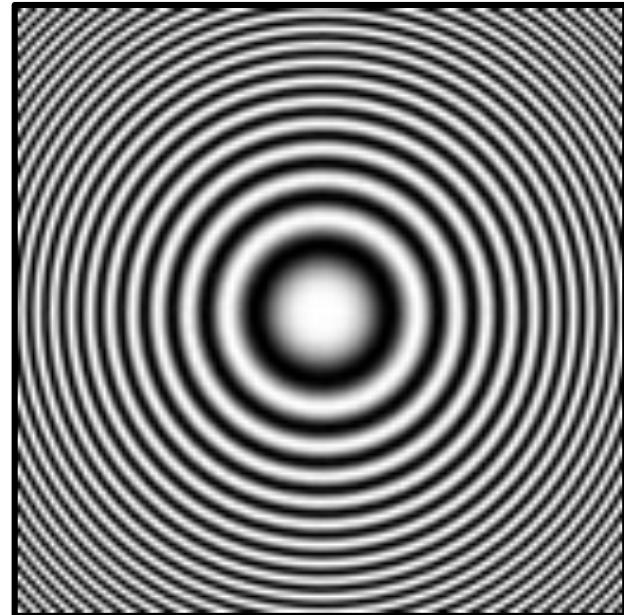
Bild, Signal und Abtastung

- Aliasing-Effekt: Fehler durch ungenügendes Abtasten von Signalen
 - im rekonstruierten Signal (z.B. Darstellung des Bildes am Monitor) treten Frequenzen auf, die im Original nicht enthalten sind
- 1D- und 2D-Version unseres Beispiels: Original und Rekonstruktion

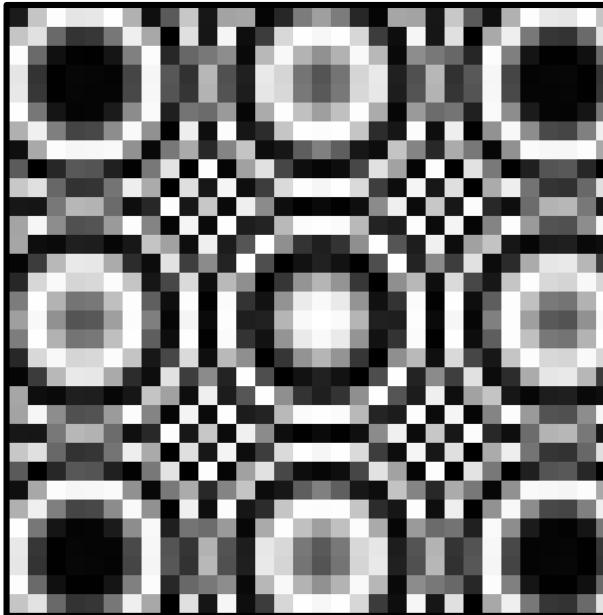


Bild, Signal und Abtastung

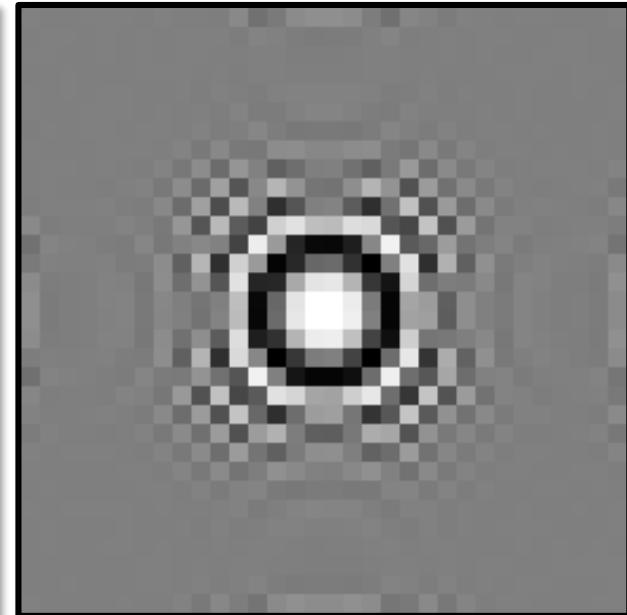
- Aliasing-Effekt: Fehler durch ungenügendes Abtasten von Signalen
 - im rekonstruierten Signal (z.B. Darstellung des Bildes am Monitor) treten Frequenzen auf, die im Original nicht enthalten sind
- Lösungsansätze
 - Filterung des Signals vor der Abtastung (hohe Frequenzen „entfernen“): im allgemeinen Fall nicht möglich
 - höhere Abtastung des Signals und anschließend Mitteln (rechtes Bild)



Abtastung ausreichend



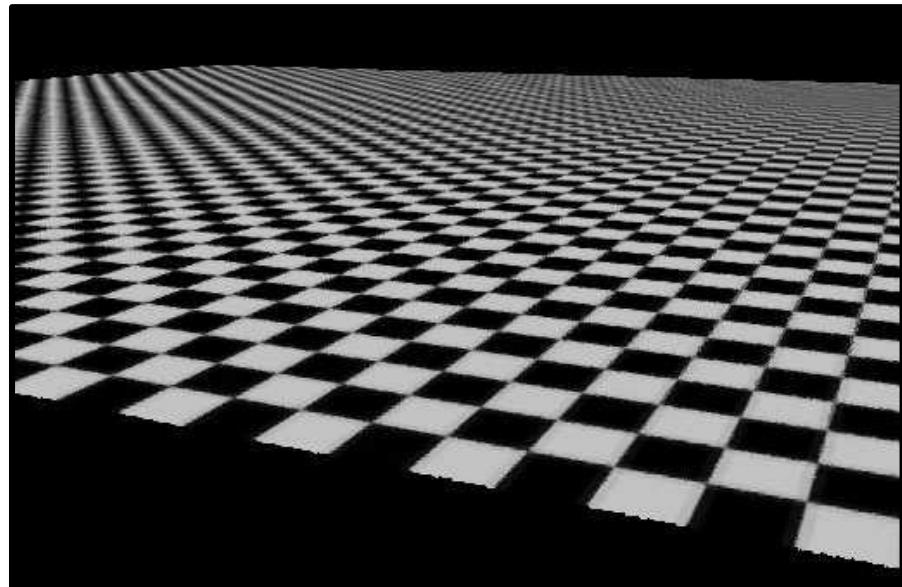
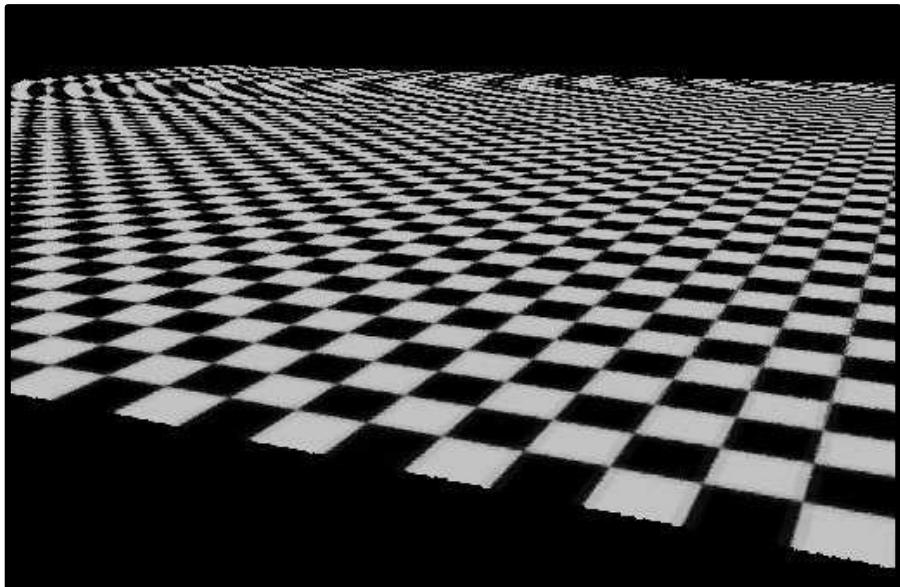
Abtastung zu niedrig → Aliasing



Überabtastung

Bsp. Aliasing

- ... das klassische Beispiel für Aliasing in der Computergrafik

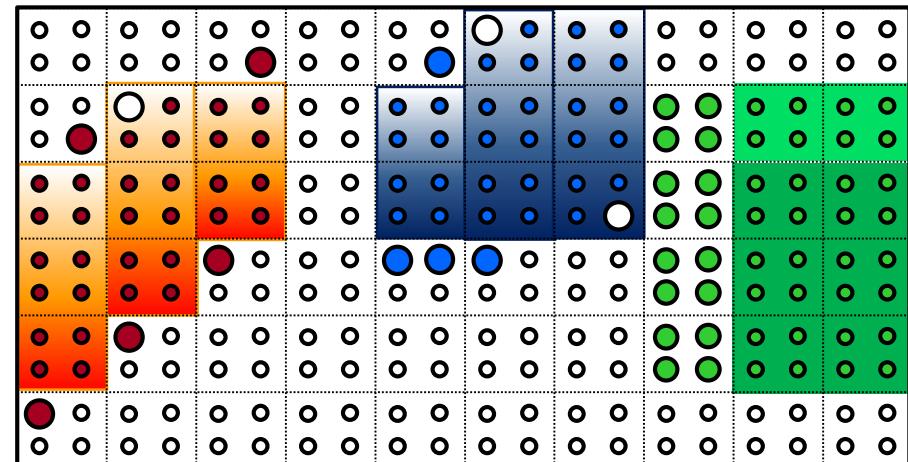
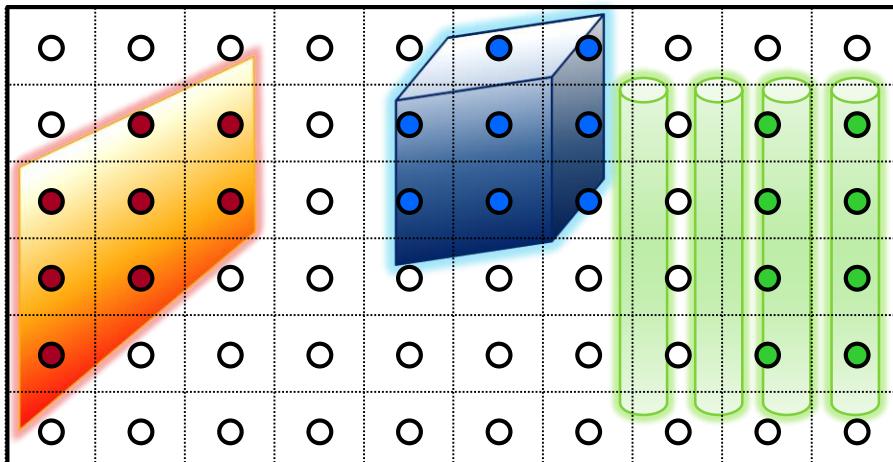


- typische Quellen von Aliasing
 - detaillierte Geometrie
 - Texturen/Textursignale (Farben, Normalen, ...)
 - Shading (z.B. Glanzlichter auf gekrümmten Flächen)
 - ...

Raytracing und Aliasing

Anti-Aliasing Strategien

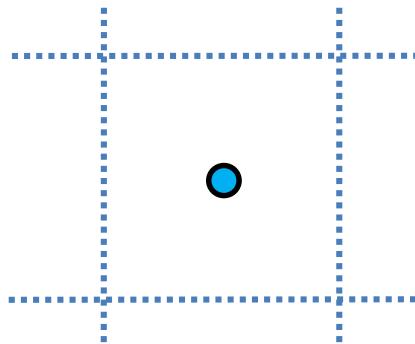
- ▶ Anti-Aliasing: Versuch Aliasing zu reduzieren oder zu vermeiden
- ▶ wir können das Bildsignal als Ganzes nicht bandbegrenzen
- ▶ es bleibt daher i.d.R. – für Aliasing aufgrund von Objektkanten und geometrischem Detail – nur Überabtastung:
 - ▶ ein Objekt bedeckt nur einen Teil eines Pixels und wir versuchen durch Überabtasten (= **Supersampling**) herauszufinden, welchen Beitrag die Objekte zu den Pixelfarben liefern



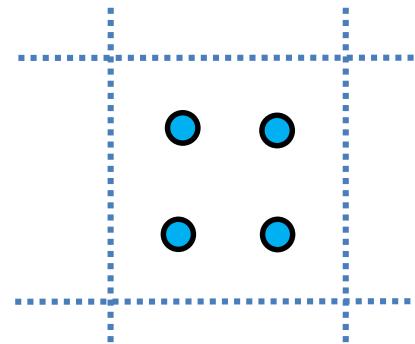
Anti-Aliasing Strategien

Uniformes Supersampling

- statt Abtastung eines Punktes innerhalb eines Pixels (ein Sample) tasten wir k^2 -mal in **äquidistanten** Intervallen (= uniform) ab

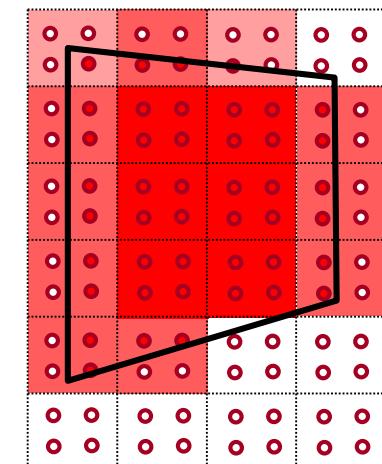
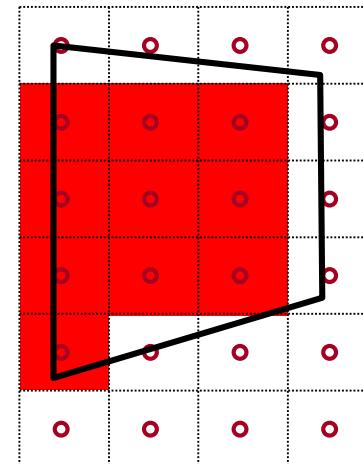


1 Pixel – 1 Sample



1 Pixel – mehrere Samples

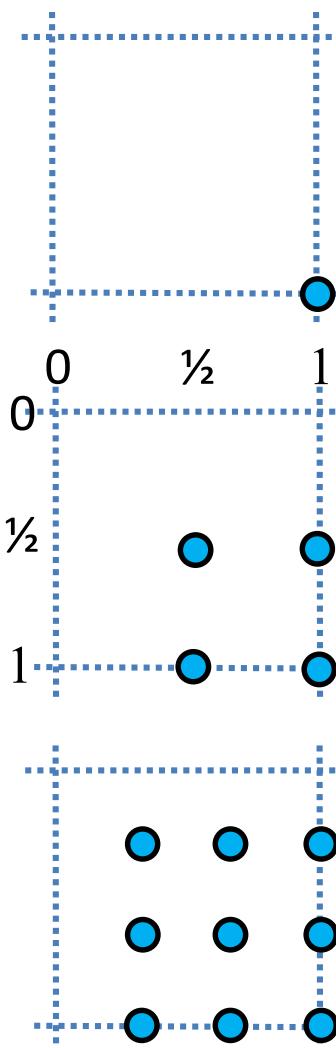
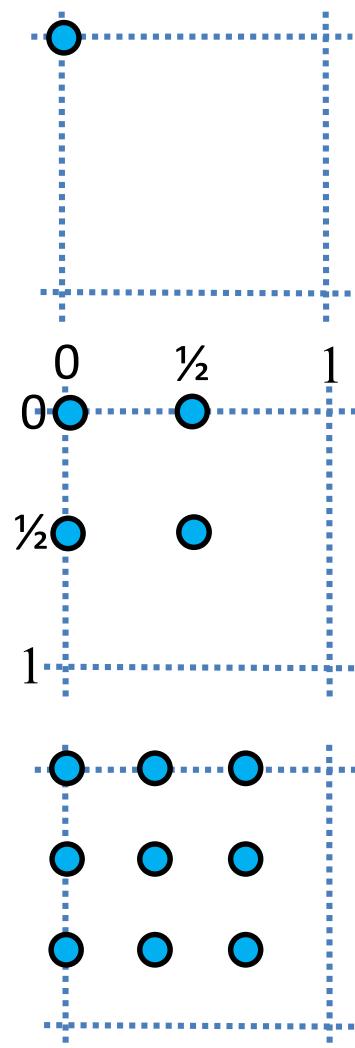
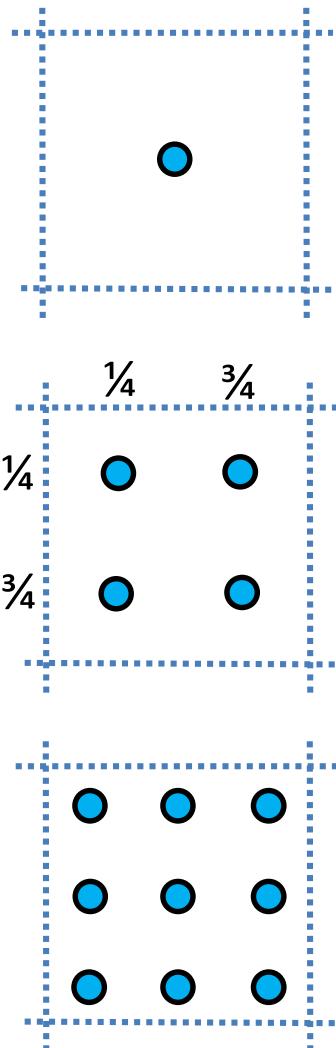
- nehme den Mittelwert der k^2 -Farben als Pixel-Farbe
- Primärstrahlen dann auch durch nicht-ganzzahlige Pixel-Positionen auf der Bildebene



Anti-Aliasing Strategien

Uniformes Supersampling

- ▶ Abtastung eines Pixels



1 Sample per Pixel

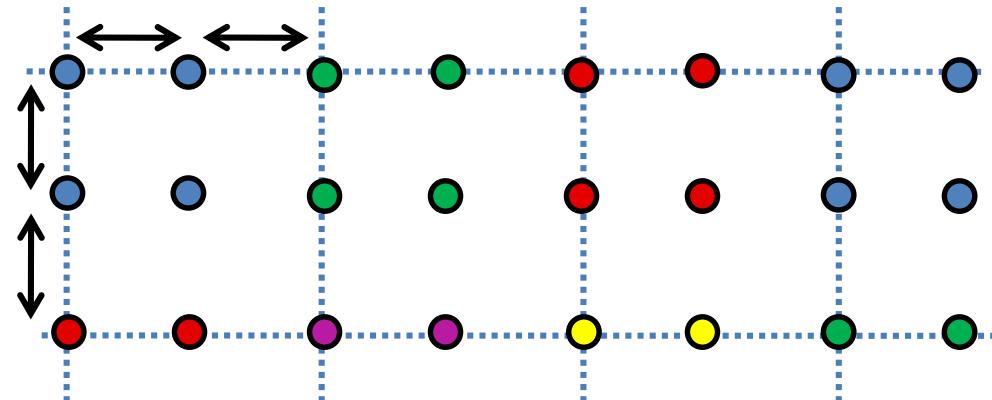
4 Samples per Pixel

9 Samples per Pixel

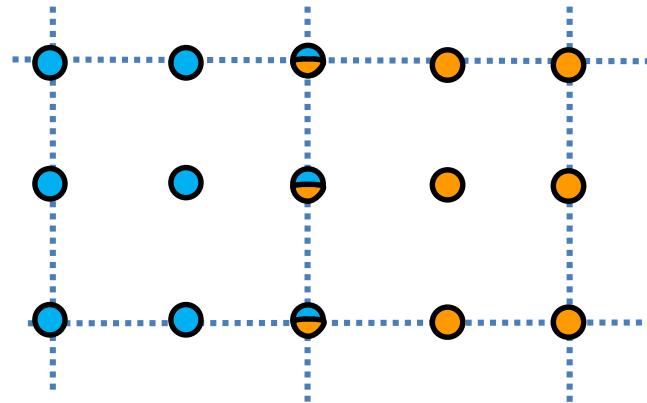
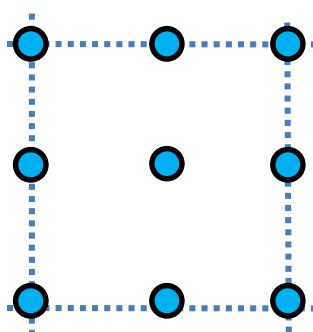
Anti-Aliasing Strategien

Uniformes Supersampling

- Abstand zwischen den Samples im Bild muss immer gleich sein



- so darf es nicht sein:

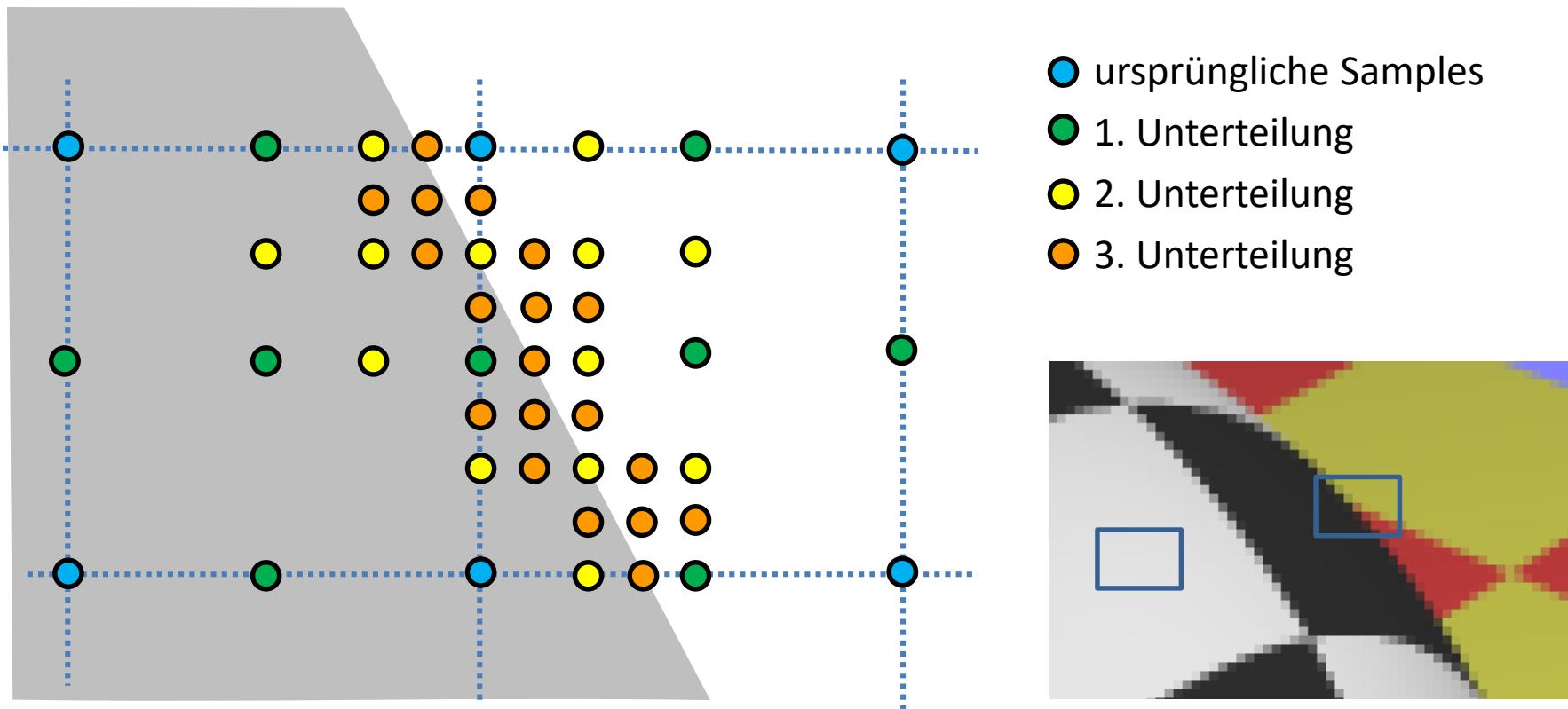


mehrere Samples an derselben Stelle

Anti-Aliasing Strategien

Adaptives Supersampling (heute für AA so nicht mehr verwendet)

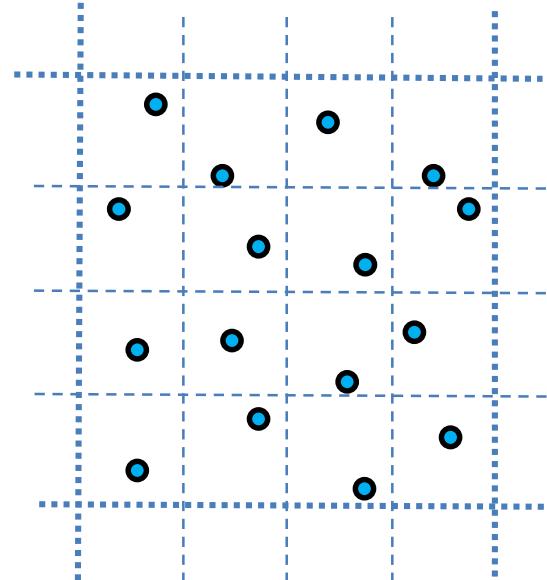
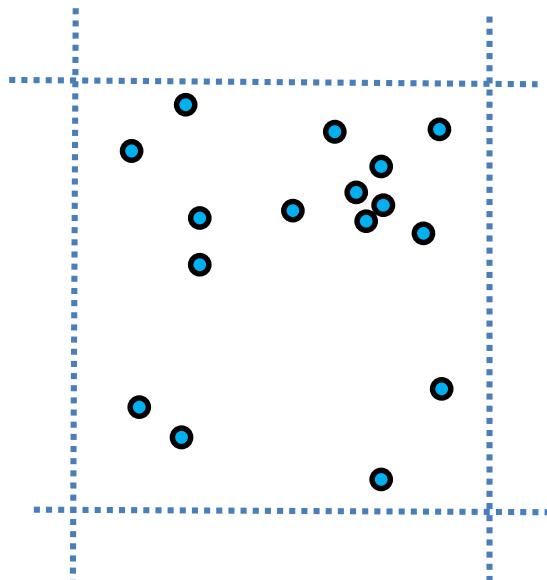
- ist die (Farb-)Differenz benachbarter Samples zu groß, dann unterteile den Pixel in 4 Bereiche und verfolge weitere Strahlen
- „einfarbige“ Pixel im Schnitt 1 Sample, Kanten werden trotzdem geglättet
- Farbdifferenzen sind nicht verlässlich (potentiell immer noch Unterabt.)



Anti-Aliasing Strategien

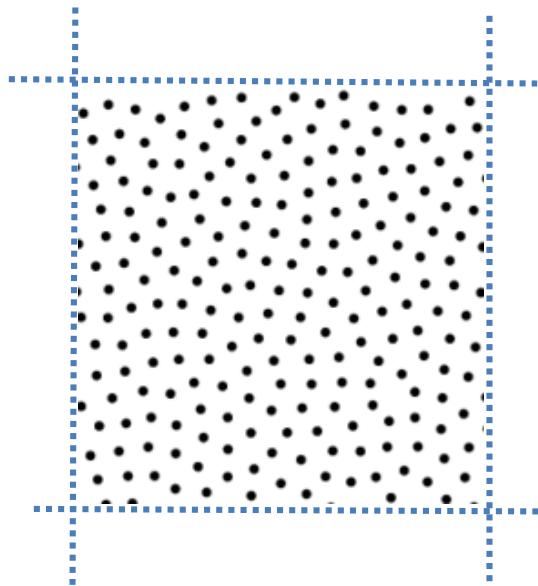
Stochastisches Sampling

- ▶ uniformes Supersampling löst hohe Frequenzen oft nicht gut auf
→ **stochastisch** = verwende zufällige Samples in einem Pixel (links)
- ▶ noch besser: zufällige Samples **mit Stratifikation** (Stratified Sampling)
 - ▶ unterteile Pixel in Gitter, wähle einen Zufallspunkt pro Stratum (Zelle)
 - ▶ sehr oft verwendeter Ansatz (rechts) auch „stratified jittered“
- ▶ nicht-uniformes Abtasten: reduziert Aliasing, aber führt zu mehr Rauschen (von unserer Wahrnehmung als weniger störend empfunden)



Blue Noise Sampling (blaues Rauschen)

- ▶ möglichst uniformer Abstand zwischen den nahsten Punkten
- ▶ weniger/keine niederfrequenten Anteile, isotropes Frequenzspektrum
- ▶ in der Praxis: vorberechnete Abtastpunkte
- ▶ unerlässlich für Echtzeit-Raytracing mit wenigen Abtastpunkten



Anti-Aliasing Strategien

Uniformes und stochastisches Überabtasten

Referenz
(256 SPP)



1 SPP
(uniform)



1 SPP
(jittered)



4 SPP
(jittered)



Raytracing in PostScript?



- ▶ PS ist eine Stacksprache (wie z.B. auch FORTH)

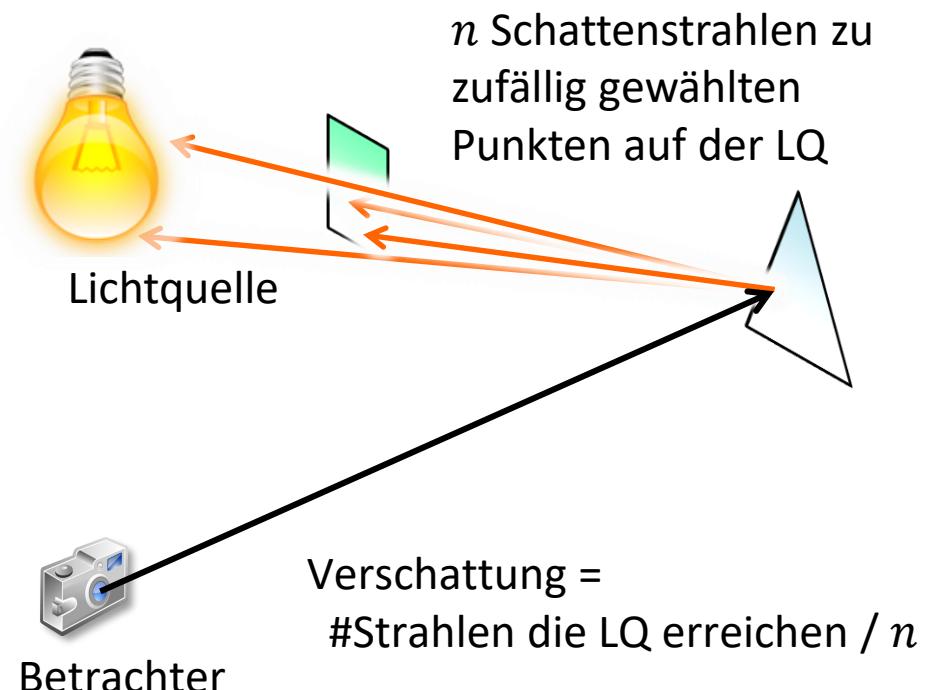
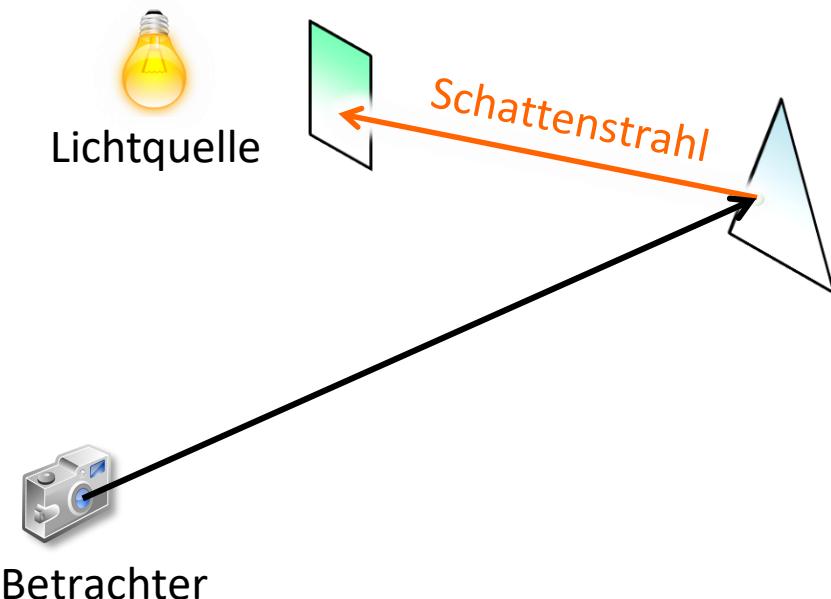
```
%!OPS-1.0 %%Creator: HAYAKAWA,Takashi<h-takasi@isea.is.titech.ac.jp>
/A/copy/p/floor/q/gt/S/add/n/exch/i/index/J/ifelse/r/roll/w/div/H{{loop}stopped
Y}def/t/and/C/neg/T/dup/h/exp/Y/pop/d/mul/s/cvi/e/sqrt/R/rlineto{load def}H 300
T translate(V2L&1i2A00053r45hNvQXz&vUX&UOvQXzFJ!FJ!J!O&Y43d9rE3IaN96r63rvx2dcaN
G&140N7!U&4C577d7!z&&93r6IQ02Z4o3AQYaNIxS2w!!f&nY9wn7wpSps1t1S!D&cjS5o32rS4oS3o
Z&bIxC1SdC9n5dh!I&3STinTinTinY!B&V0R0VRVC0R!N&3A3Axe1nwc!I&993dC99Cc96raN!a&1CD
E&YYY!F&&vGYx4oGbxD0nq&3IGbxSGY4Ixwca3AlvvUkbQkdbGYx4ofwnw!&vlx2w13wSb8Z4wS!J!
c&j1idj2id42rd!X&4I3Ax52r8Ia3A3Ax65rTdCS4iw5o5IxnwTTd32rCST0q&eCST0q&D1!&EYE0!J
&EYE0!J0q!x&jd5o32rd4odSS!K&WCVW!Q&31C85d4!k&X&E9!&1!J!v&6A!b&7o!o&1r!j&43r!W)
{( )T 0 4 3 r put T(/)q{T(9)q{cvn}{s}J}{($)q{[]}{}}J}J cvx}forall 270{def}H
K{K{L setgray moveto B fill}for Y}for showpage
```

Distributed Raytracing

Probleme des Whitted-Style Raytracing-Verfahrens

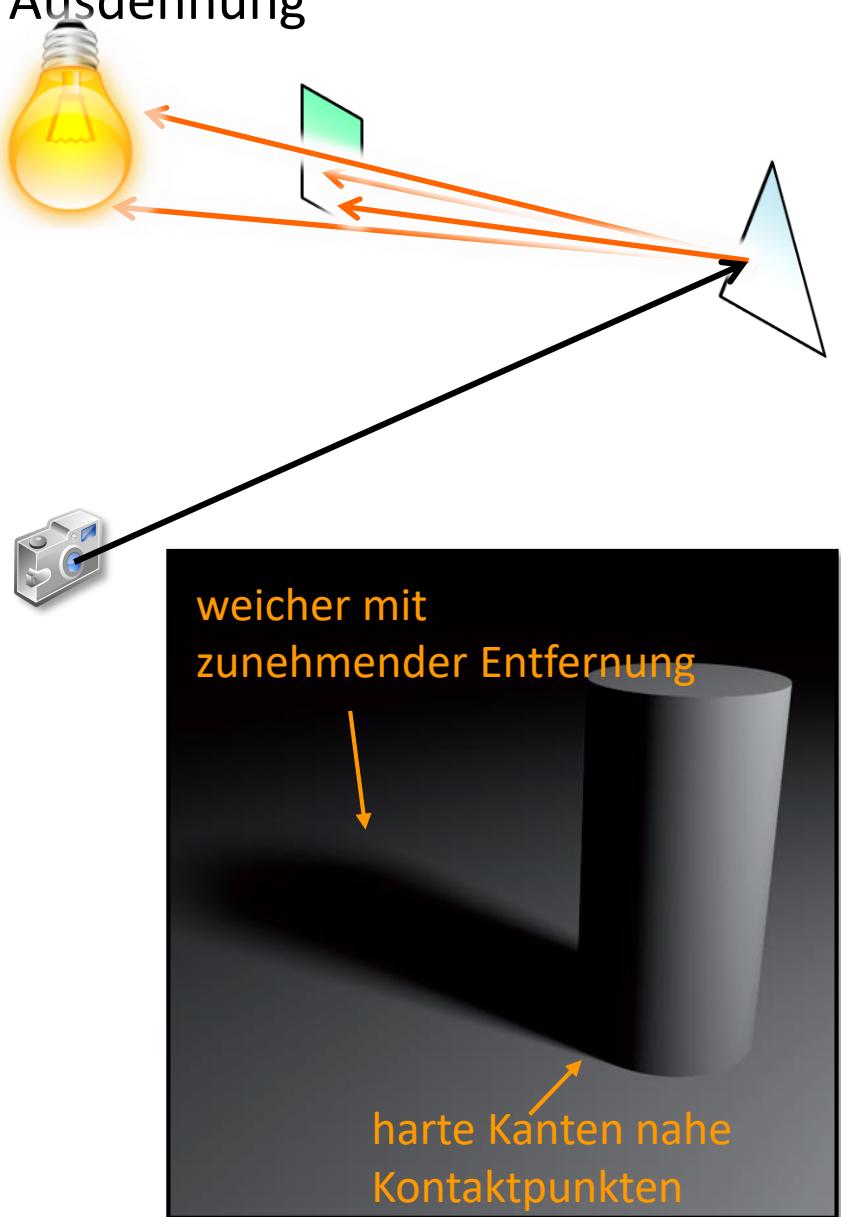
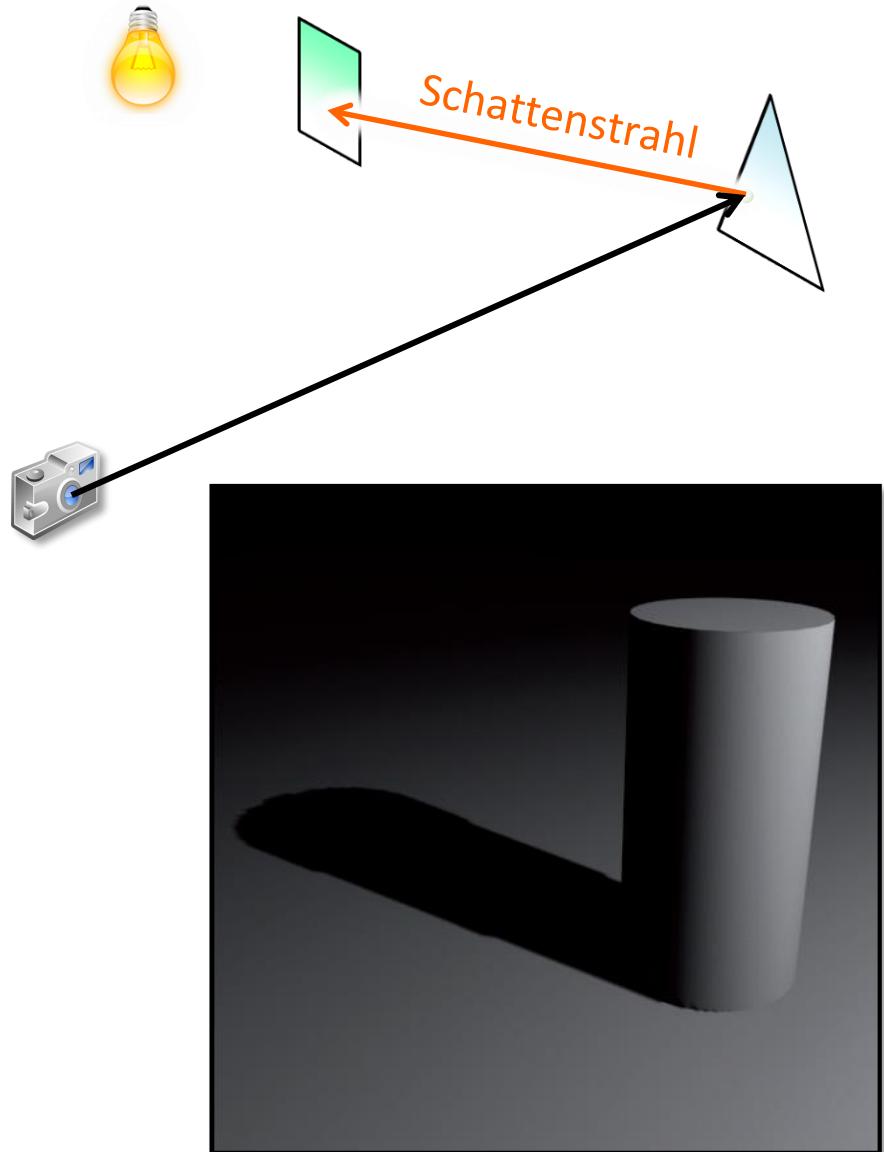
- die Bilder sehen zu makellos aus:
 - ▶ perfekte Spiegelung und Transmission
 - ▶ harte Schattenkanten, unendliche Schärfentiefe etc.
- was kann man da tun?

► Beispiel: Schattenstrahlen



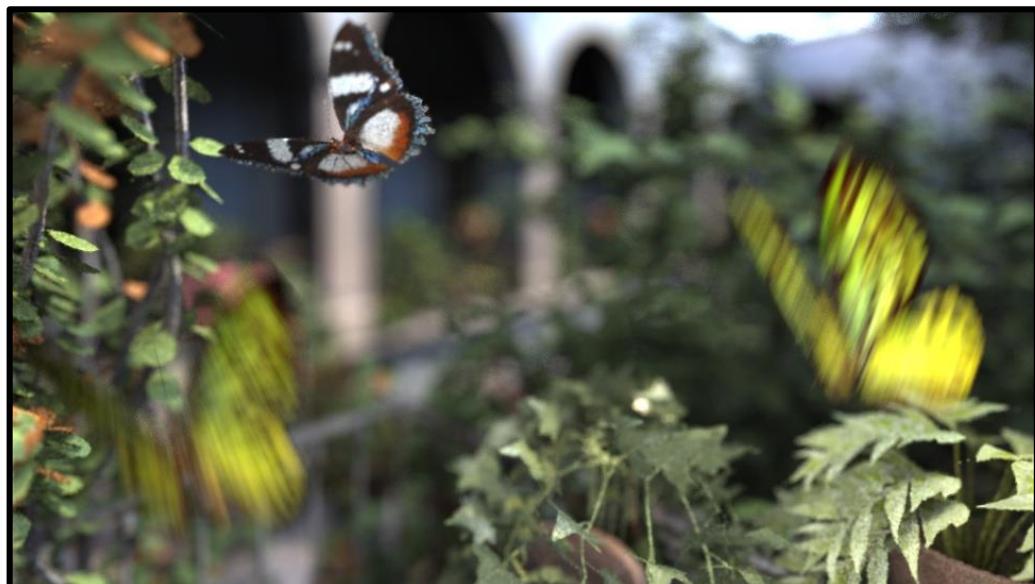
Weiche Schatten

- reale Lichtquellen besitzen endliche Ausdehnung



Bewegungsunschärfe (engl. motion blur)

- Verteilung der Strahlen in der Zeit
 - vgl. Belichtungszeit bei realen Kameras
 - Objekte bewegen sich in einem Zeitintervall $[t; t + \Delta t]$
 - erzeuge für jeden Pixel n Strahlen für einen Zeitpunkt $t' \in [t; t + \Delta t]$
 - führe Raytracing mit der Szene zum Zeitpunkt t' durch
 - mittle Farbwerte



Bewegungsunschärfe



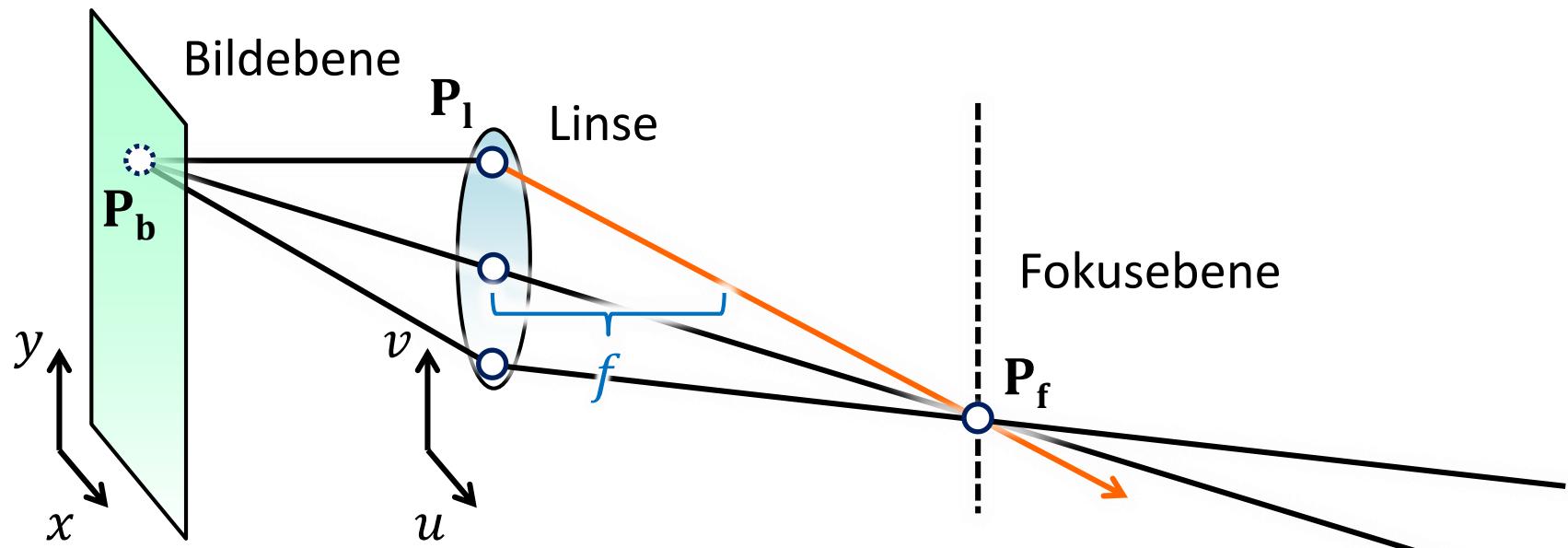
```
class Ray {
    vec3 origin, direction; // Startpunkt und Richtung des Strahls
    float t;                // Strahlparameter
    void * object;          // Zeiger auf evtl. getroffenes Objekt
    ...
};

for ( y = 0; y < height; y++ ) {
    for ( x = 0; x < width; x++ ) {
        vec3 c = 0.0f;
        for ( s = 0; s < nSuperSample; s++ ) {
            float x_ = ..., y_ = ...;           // Supersampling in Abhängigkeit von s
            for ( ts = 0; ts < nTimeSamples; ts++ )
            {
                float time = t + delta_t * random01();
                Ray ray;
                generateRay( &ray, x_, y_, time );      // time wg. Kamerabewegung
                c += raytrace( ray, time );             // time wg. Objektbewegung
            }
        }
        c /= (float)( nTimeSamples * nSuperSample );
    }
}
```

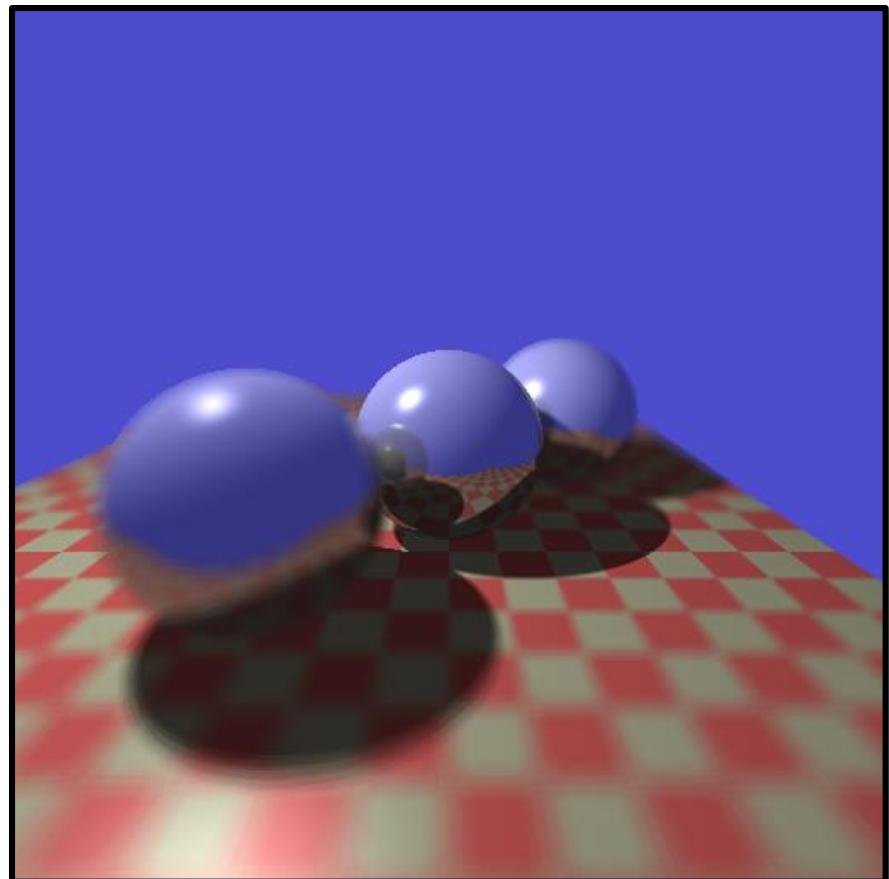
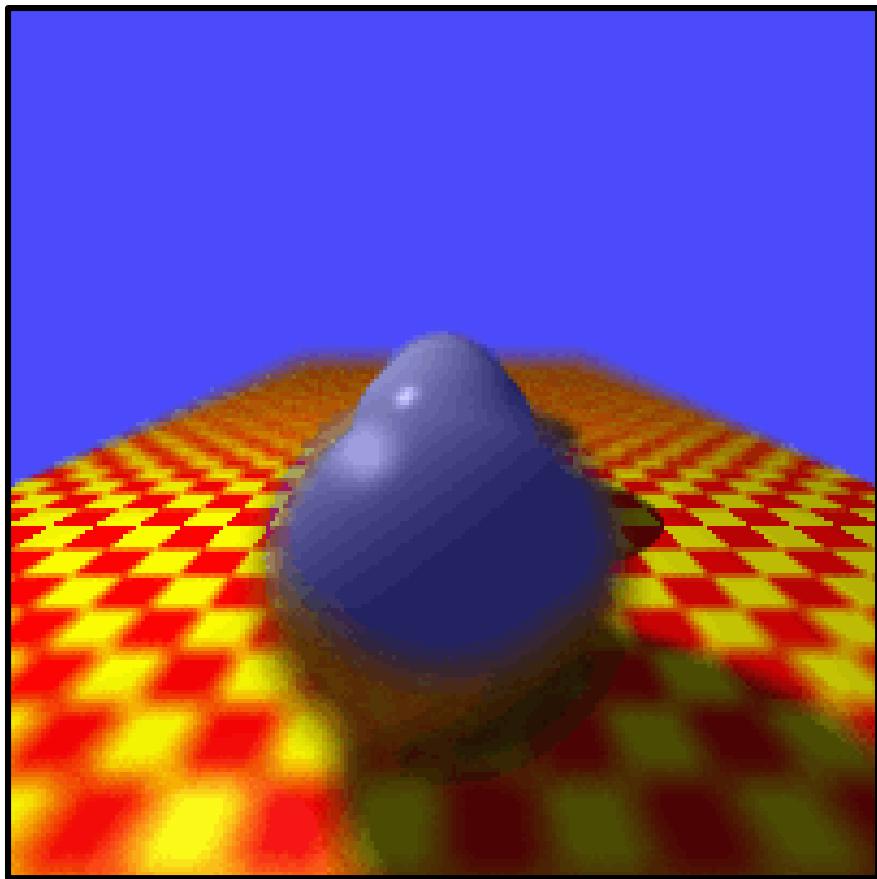
Tiefenunschärfe

Modell der „dünnen (konvexen) Linse“

- ▶ weitere Samples für das Abtasten der Linsenfläche
 - ▶ wähle Punkt auf der Bildebene P_b (und somit Pixel) und der Linse P_l
 - ▶ Strahlen durch alle Punkte auf der Linse führen zum Punkt im Fokus P_f
 - ▶ auch der Strahl durch das Zentrum, der nicht abgelenkt wird
 - ▶ berechne daraus Punkt auf der Fokusebene P_f (Abstand zur Fokusebene berechnet aus Brennweite f und Sensorabstand)
 - ▶ erzeuge **Primärstrahl** von P_l durch P_f



Tiefenunschärfe



[Cook et al. 1984]



Rob Cook is among the first people to have been awarded an Oscar for software. He is the co-architect and primary author of Pixar's RenderMan.

Bewegungs- und Tiefen(un)schärfe



```
class Ray {
    vec3 origin, direction; // Startpunkt und Richtung des Strahls
    float t;                // Strahlparameter
    void * object;          // Zeiger auf evtl. getroffenes Objekt
    ...
};

for (y = 0; y < height; y++) {
    for (x = 0; x < width; x++) {
        vec3 c = 0.0f;
        for (s = 0; s < nSuperSample * nLensSample; s++) {

            float x_ = ..., y_ = ...;           // Supersampling
            for (ts = 0; ts < nTimeSamples; ts++)
            {
                float u = random01(), v = random01(); // verwendet, um Zufallspunkt
                                                // auf der Linse zu wählen
                float time = t + delta_t * random01(); // Sampling in der Zeit
                Ray ray;
                generateRayThinLens( &ray, x_, y_, u, v, time );

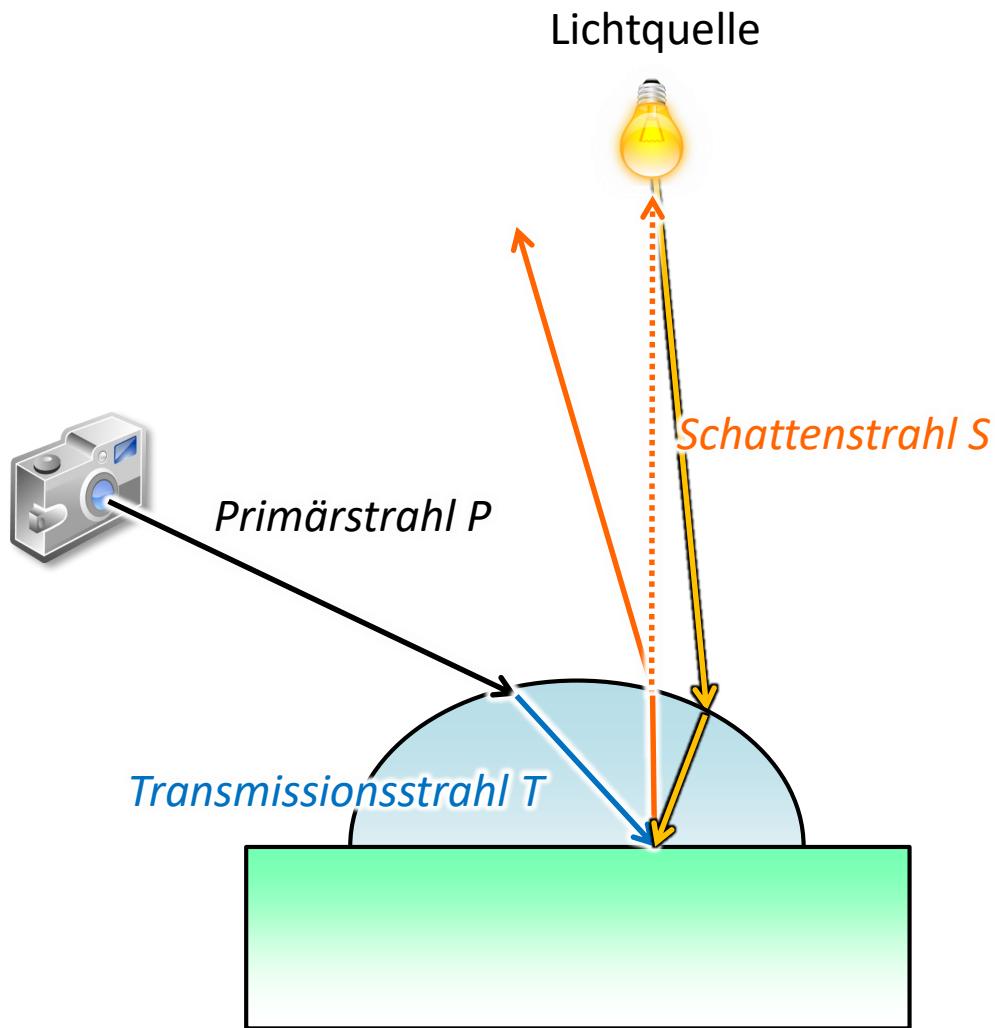
                c += raytrace( ray, time );
            }
        }
        c /= (float)( nTimeSamples * nSuperSample * nLensSample );
    }
}
```

Bewegungs- und Tiefen(un)schärfe



```
class Ray {  
    vec3 origin, direction; // Startpunkt und Richtung des Strahls  
    float t;                // Strahlparameter  
    void * object;          // Zeiger auf evtl. getroffenes Objekt  
    ...  
};  
  
  
for ( y = 0; y < height; y++ ) {  
    for ( x = 0; x < width; x++ ) {  
        vec3 c = 0.0f;  
        for ( s = 0; s < nSuperSample * nLensSample * nTimeSamples; s++ ) {  
  
            float x_ = ..., y_ = ...;                      // Supersampling  
            float u = random01(), v = random01(); // verwendet, um Zufallspunkt  
                                              // auf der Linse zu wählen  
            float time = t + delta_t * random01(); // Sampling in der Zeit  
  
            Ray ray;  
            generateRayThinLens( &ray, x_, y_, u, v, time );  
  
            c += raytrace( ray, time );  
        }  
        c /= (float)( nTimeSamples * nSuperSample * nLensSample );  
    } }  
}
```

Und was ist mit ...

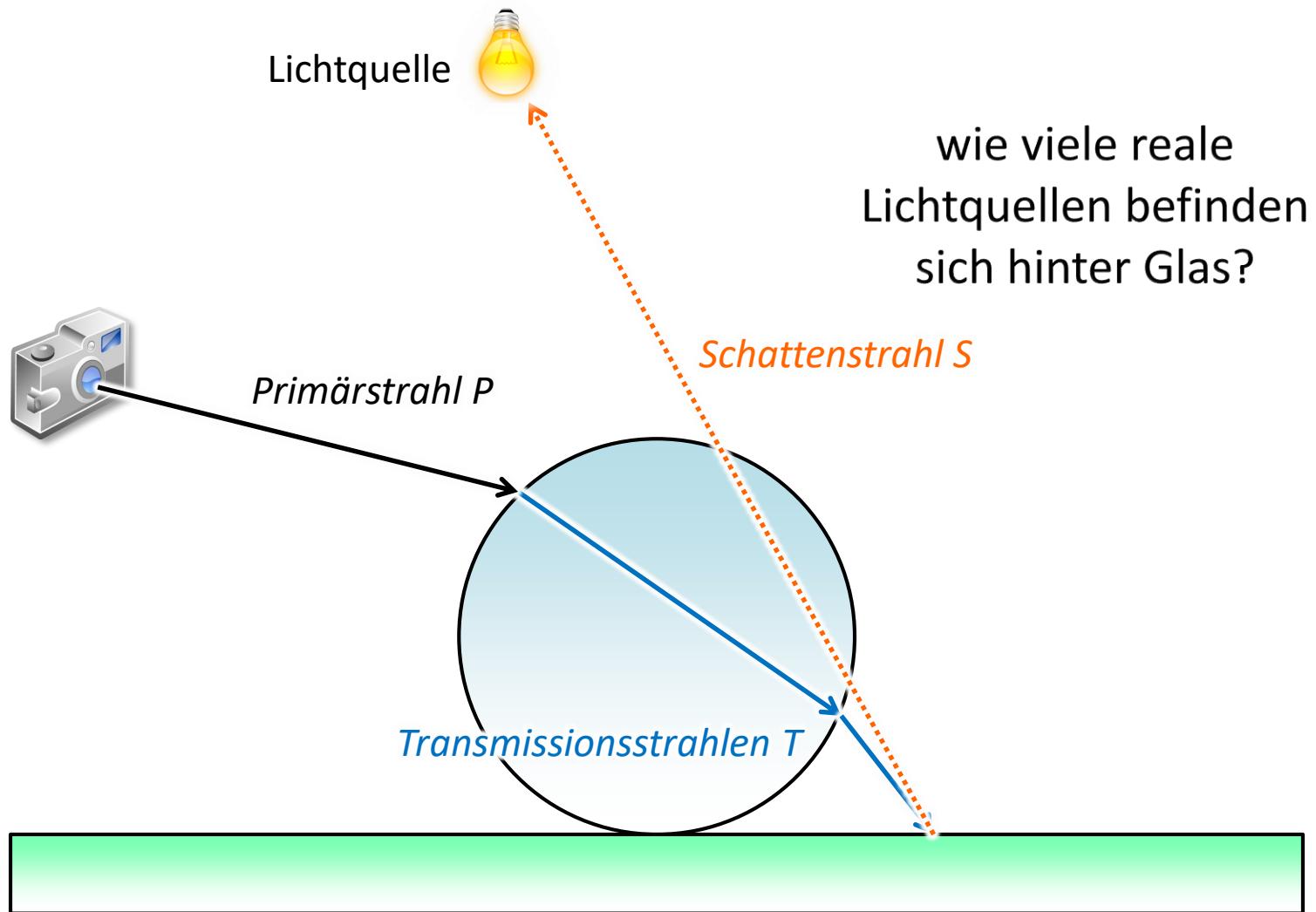


Und was ist mit ...

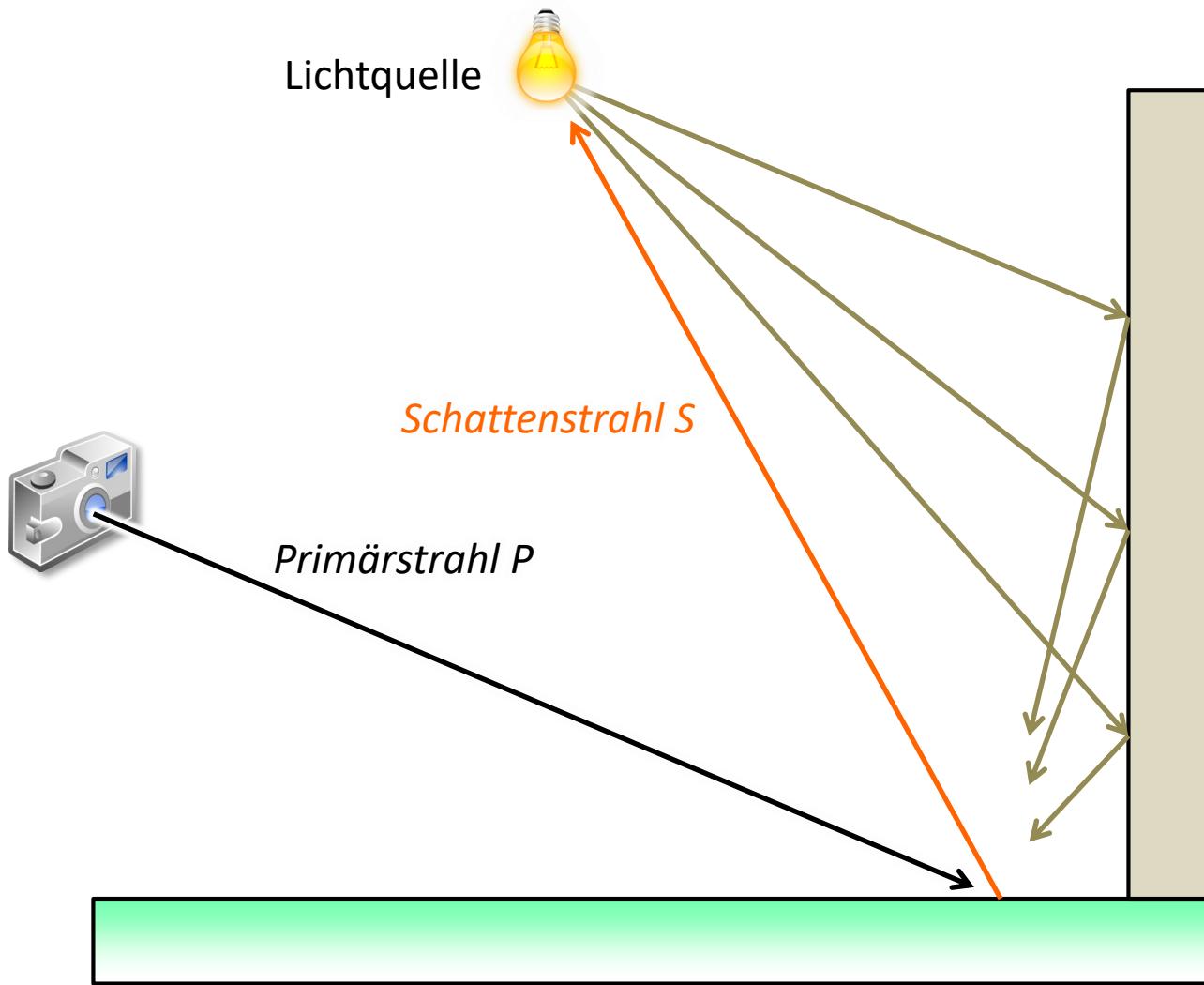
- ▶ (fast/immer noch) aktuelle Forschung: Johannes Hanika, Marc Droske, and Luca Fascione. Manifold next event estimation. *Computer Graphics Forum (Proc. of Eurographics Symposium on Rendering)*, 34(4):87--97, June 2015



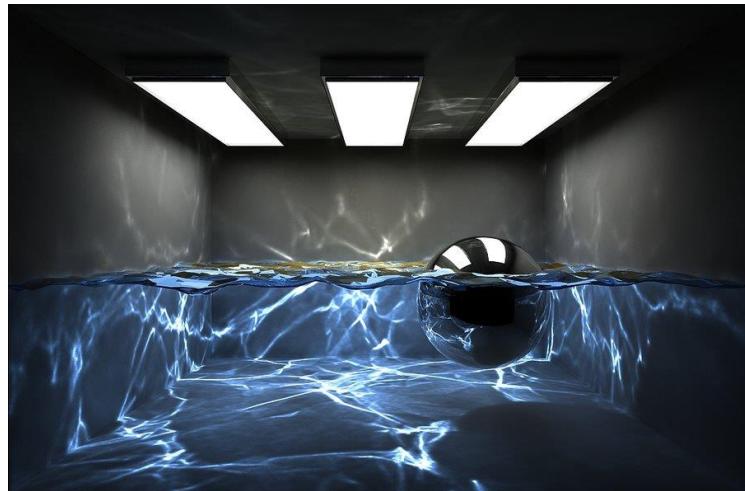
Und was ist mit ...



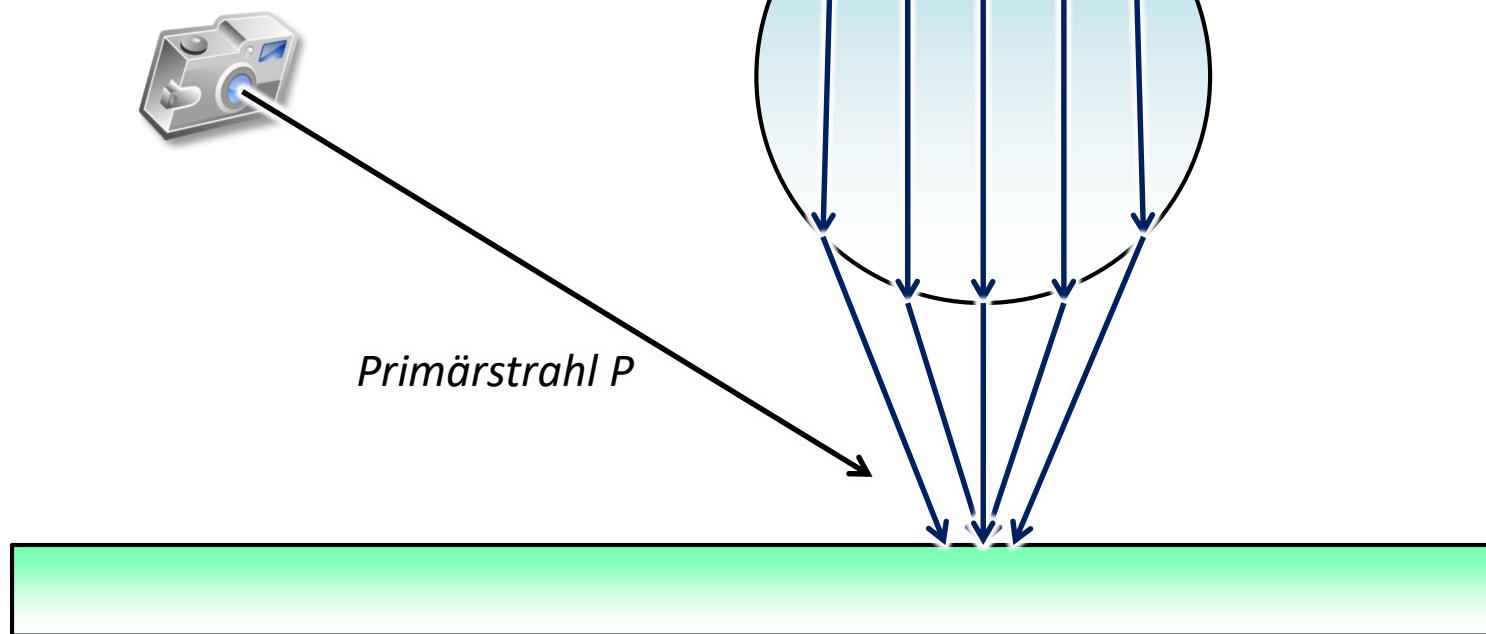
Und was ist mit ...



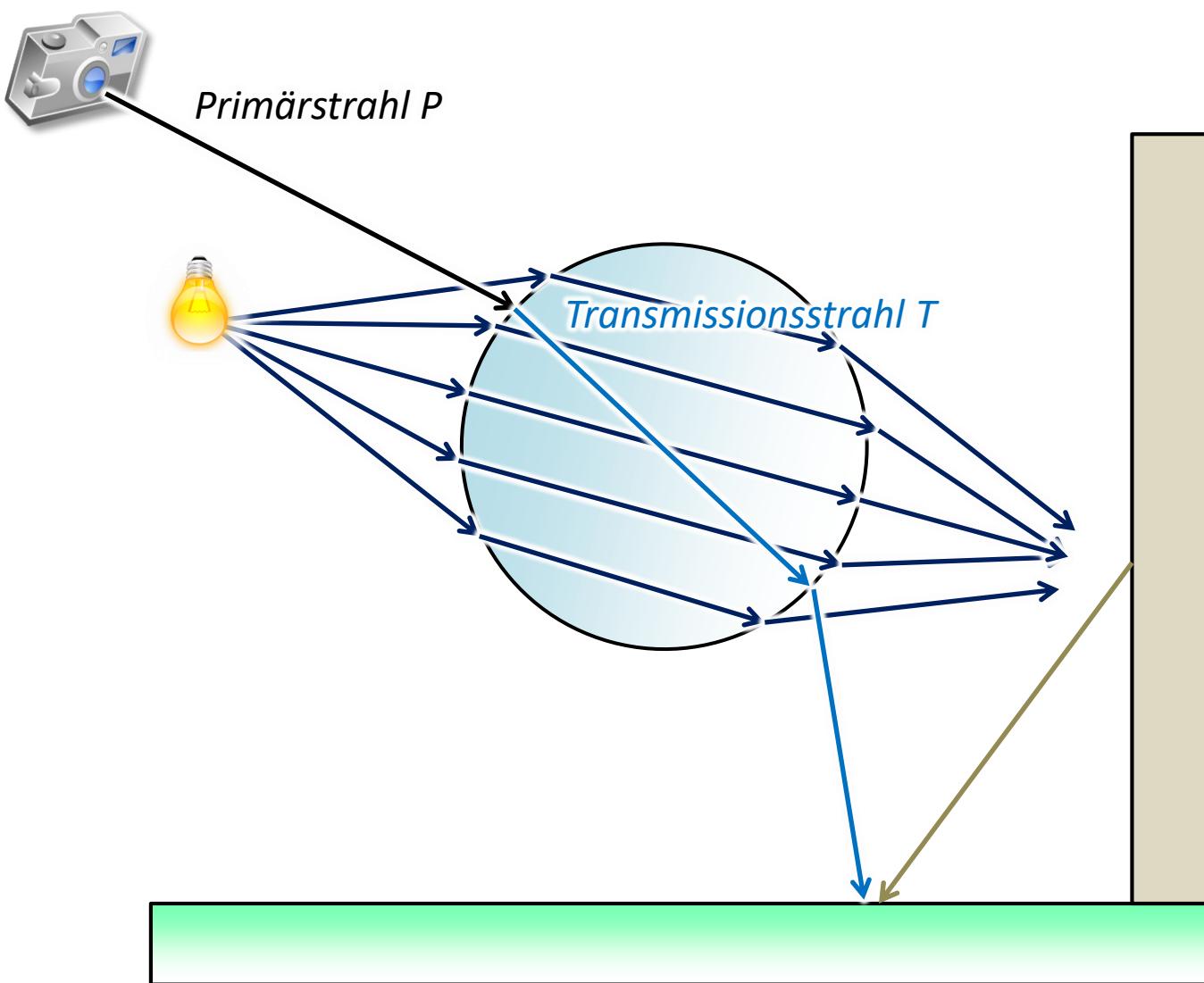
Und was ist mit ...



<http://nestohuis.deviantart.com/art/Caustic-Chrome-Tut-55223982>



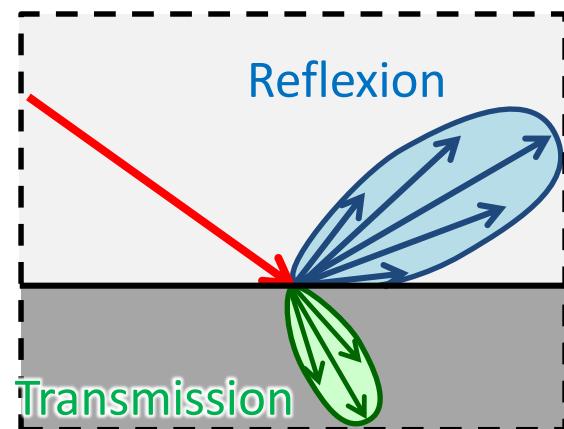
Und was ist mit ...



Imperfekte Spiegelung und Transmission

Distributed Raytracing / Distribution Raytracing

- ▶ um alle Lichtwege zu berücksichtigen, sind mehrere Strahlen für Reflexion und Transmission von jedem Schnittpunkt aus notwendig
 - ▶ es werden also viele Sekundärstrahlen weiterverfolgt
 - ▶ (quasi-)zufällige Richtungen
 - ▶ deren Beiträge zum reflektierten/ gebrochenen Licht werden durch die BSDF bestimmt

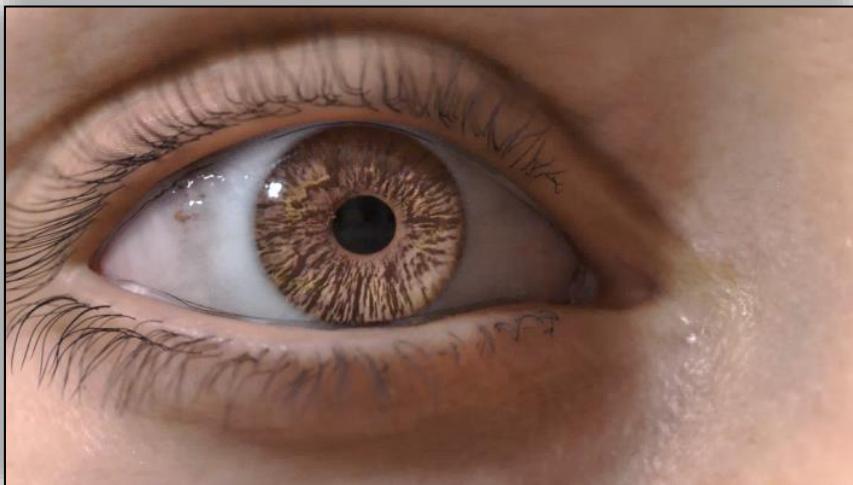


- ▶ wie viele Strahlen werden benötigt?
 - ▶ durch Rekursion verzweigt der „Strahlenbaum“ schon beim Whitted-Style Raytracing
 - ▶ Explosion bei mehreren Sekundärstrahlen an jedem Punkt

Lichttransport, Beleuchtungssimulation

Was ist die Physik dahinter?

$$L(x, \omega) = L_e(x, \omega) + \int_{\Omega} f_r(\omega_i, x, \omega) L_i(x, \omega_i) \cos \theta_i d\omega_i$$



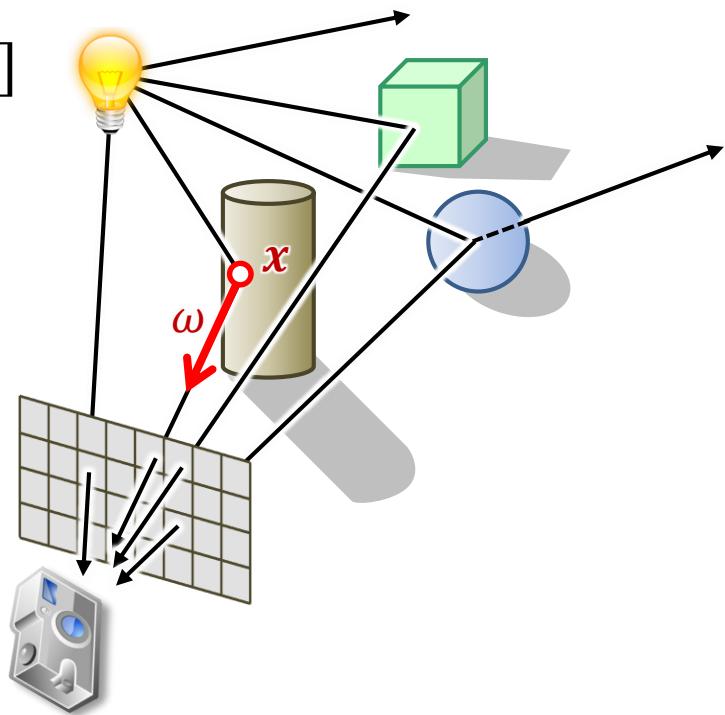
Demo: <https://wwwtyro.net/2018/02/25/caffeine.html>

Rendering Gleichung

Lichtverteilung in einer Szene [Kajiya86]

$$L(x, \omega) = L_e(x, \omega) + \int_{\Omega^+} f_r(\omega_i, x, \omega) L_i(x, \omega_i) \cos \theta_i d\omega_i$$

- ▶ L Strahldichte (engl. Radiance) [$\text{W}/\text{m}^2\text{sr}$]
- ▶ L_e Emissionsterm [$\text{W}/\text{m}^2\text{sr}$]
- ▶ Abhangigkeit von der Wellenlange
(hier nicht dargestellt)
- ▶ Integrationsbereich:
 Ω^+ positive Hemisphare (nur Reflexion)
 Ω alle Richtungen (inkl. Transmission)

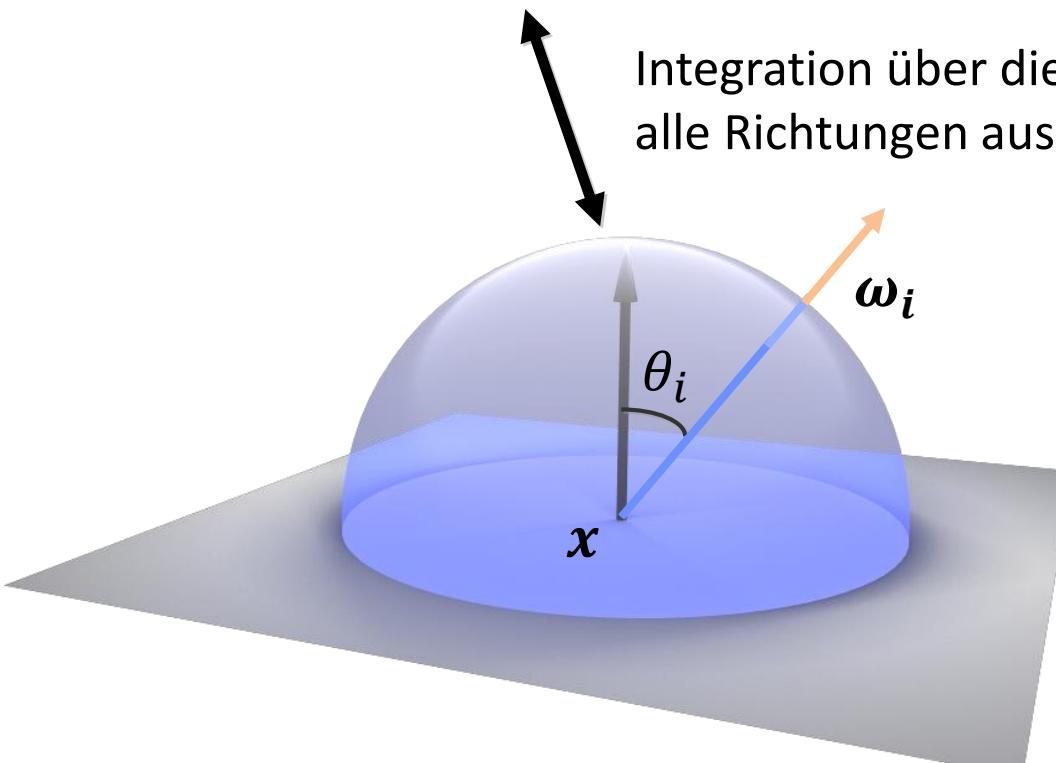


Rendering Gleichung

$$L(x, \omega) = L_e(x, \omega) + \int_{\Omega^+} f_r(\omega_i, x, \omega) L_i(x, \omega_i) \cos \theta_i d\omega_i$$

Einfallsrichtung

Integration über die positive Hemisphäre:
alle Richtungen aus denen Licht einfällt



x

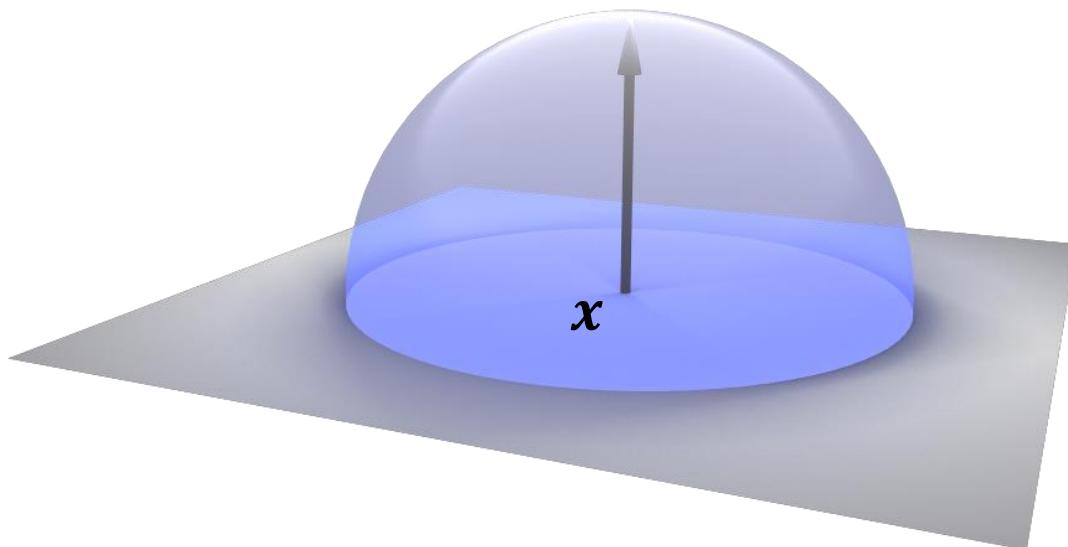
ω_i

θ_i

Rendering Gleichung

$$L(x, \omega) = L_e(x, \omega) + \int_{\Omega^+} f_r(\omega_i, x, \omega) L_i(x, \omega_i) \cos \theta_i d\omega_i$$

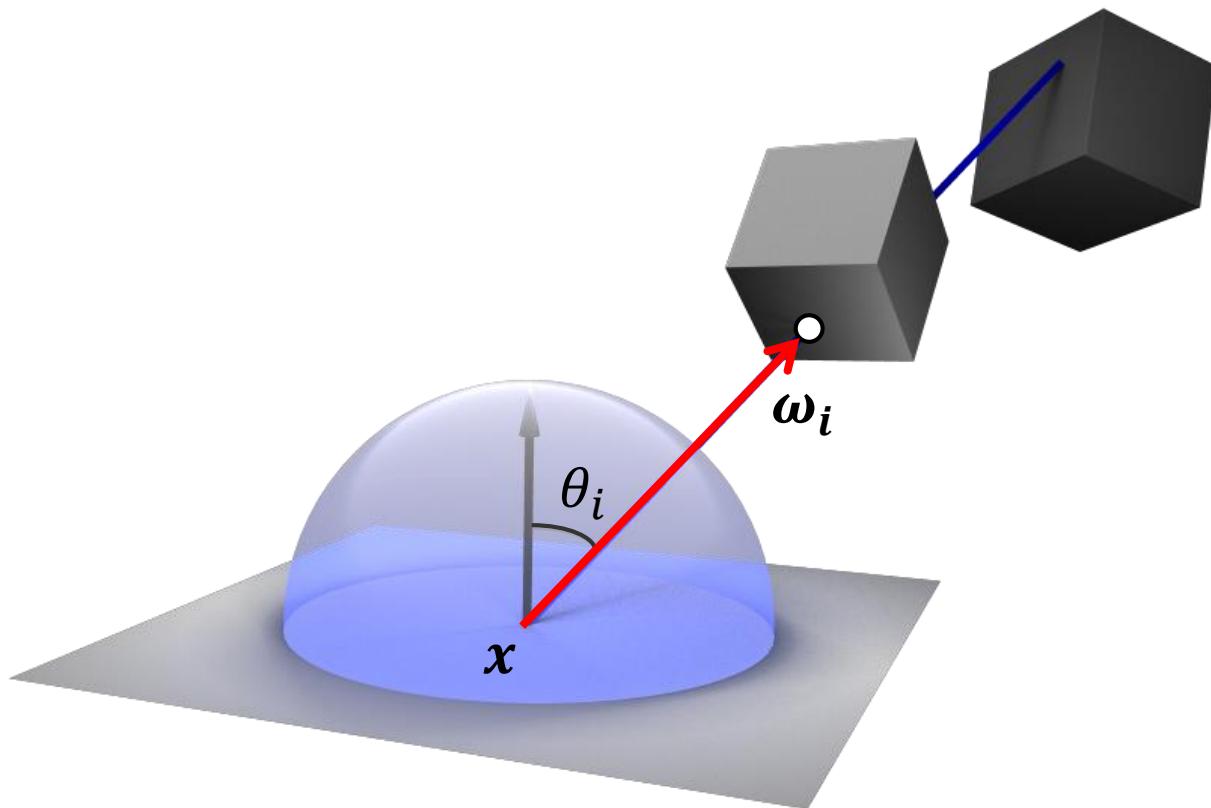
Reflektanzverteilungsfunktion:
Wie viel Licht aus Richtung ω_i
wird in Richtung ω reflektiert



Rendering Gleichung

$$L(x, \omega) = L_e(x, \omega) + \int_{\Omega^+} f_r(\omega_i, x, \omega) L_i(x, \omega_i) \cos \theta_i d\omega_i$$

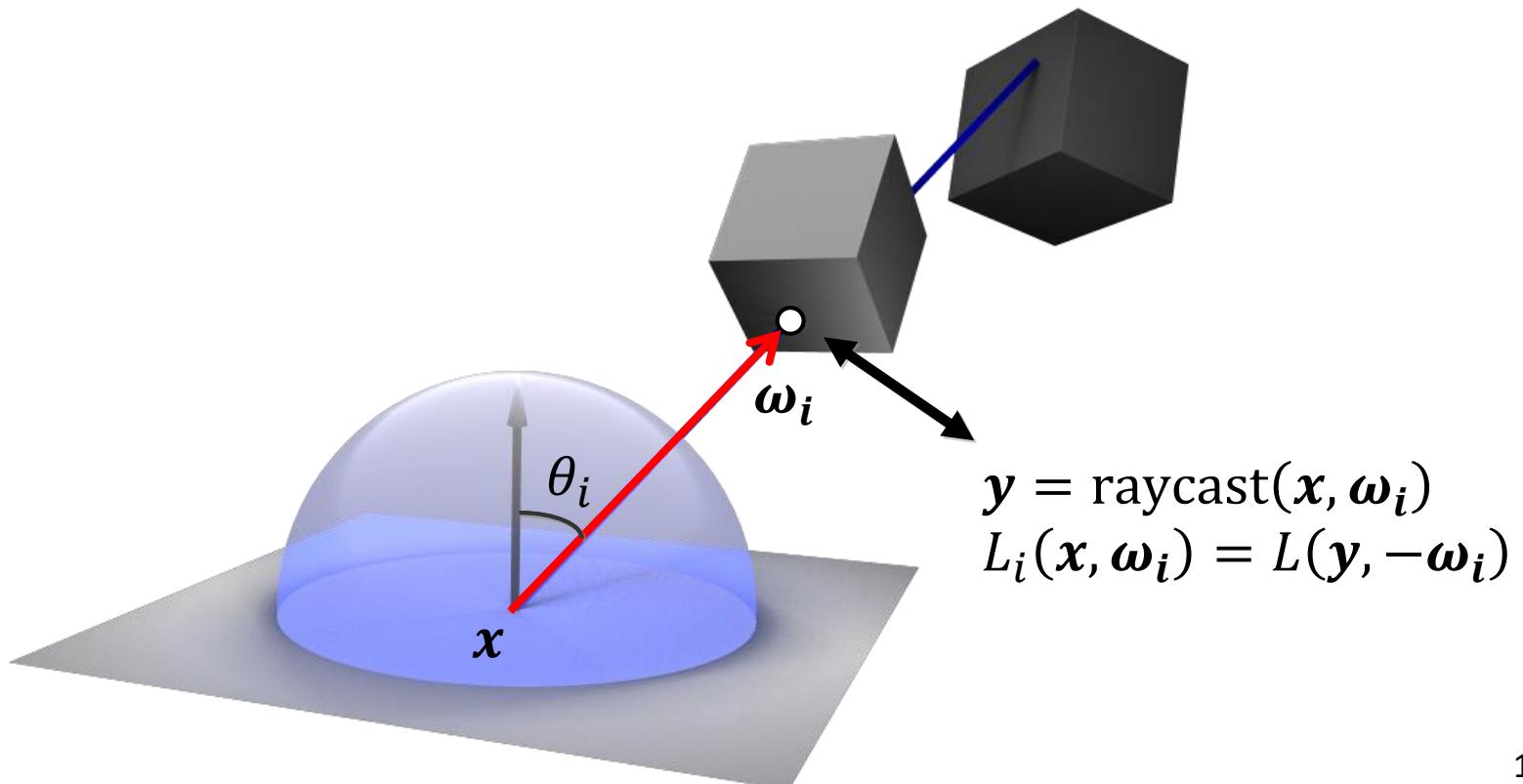
einfallendes Licht



Rendering Gleichung

$$L(x, \omega) = L_e(x, \omega) + \int_{\Omega^+} f_r(\omega_i, x, \omega) \boxed{L(y, -\omega_i)} \cos \theta_i d\omega_i$$

ausgehendes Licht

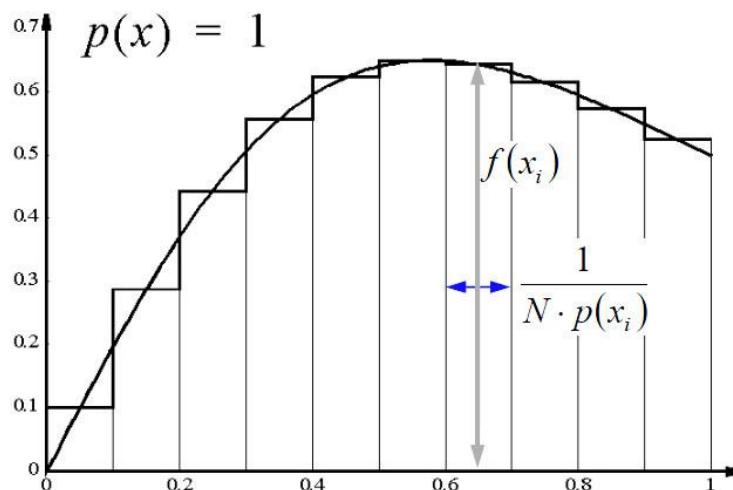


Monte Carlo-Integration

- Grundidee der Monte Carlo-Integration (vereinfacht!):
wir können den Wert eines Integrals näherungsweise berechnen indem
wir den Integranden an N zufälligen uniform-verteilten Stellen $x_i \in [a; b]$
auswerten

$$\int_a^b f(x)dx \approx \frac{b-a}{N} \sum_{i=1}^N f(x_i)$$

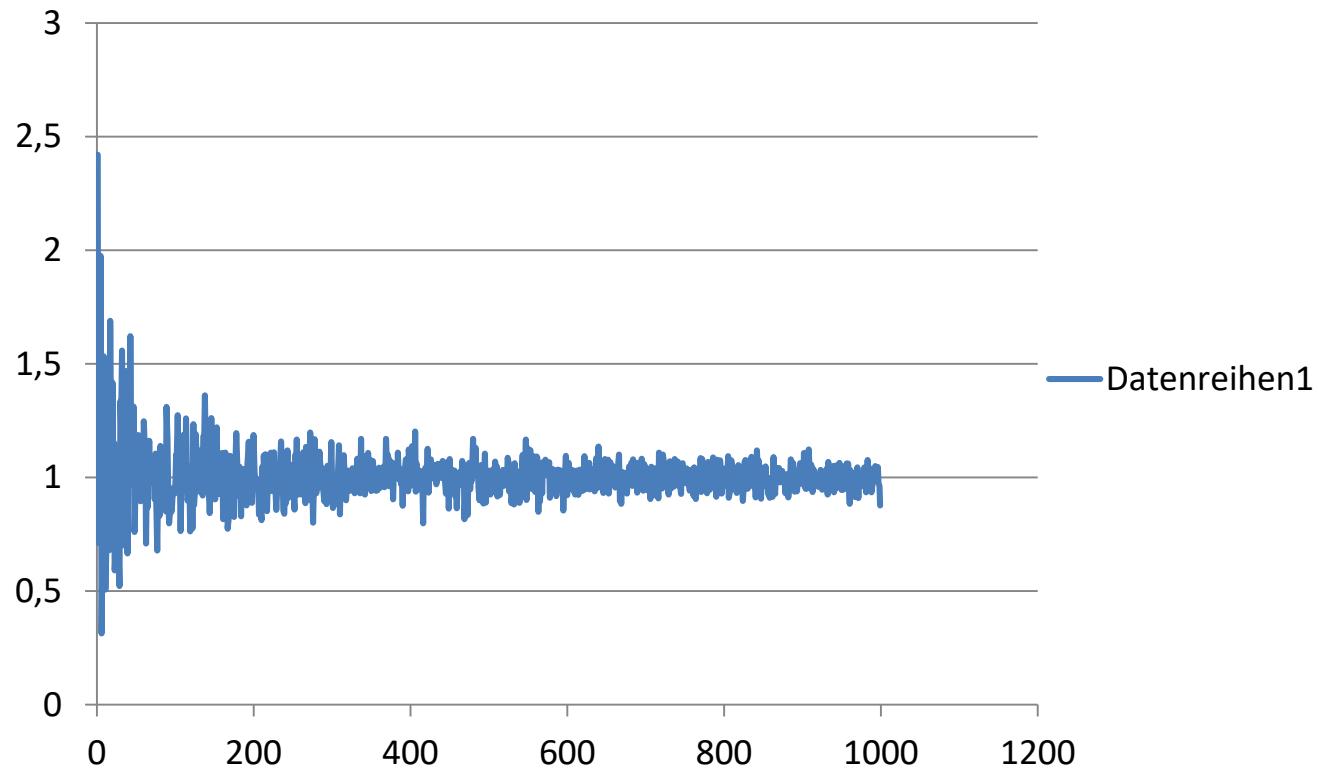
- die Näherung/Schätzung wird besser, wenn N größer wird
- vergleiche Riemann-Integral:



Monte Carlo-Integration

Beispiel

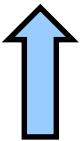
- berechne $I = \int_0^1 5x^4 dx = 1$
- naïve Monte Carlo-Integration mit $N = 1 \dots 1000$



Monte Carlo-Integration

- ▶ um den Wert des Integrals zu berechnen verwendet Distributed Raytracing (und verwandte Verfahren) die Monte Carlo-Integration
- ▶ im einfachsten Fall (wie eben gezeigt) bedeutet das: berechne den Wert des Integranden für (viele) zufällige Richtungen und mittle

$$L(\mathbf{x}, \boldsymbol{\omega}) = L_e(\mathbf{x}, \boldsymbol{\omega}) + \frac{2\pi}{N} \sum_{i=1}^N f_r(\boldsymbol{\omega}_i, \mathbf{x}, \boldsymbol{\omega}) L_i(\mathbf{x}, \boldsymbol{\omega}_i) \cos \theta_i$$



$$L(\mathbf{x}, \boldsymbol{\omega}) = L_e(\mathbf{x}, \boldsymbol{\omega}) + \int_{\Omega^+} f_r(\boldsymbol{\omega}_i, \mathbf{x}, \boldsymbol{\omega}) L_i(\mathbf{x}, \boldsymbol{\omega}_i) \cos \theta_i d\boldsymbol{\omega}_i$$

- ▶ Intuition: Faktor 2π entspricht der Oberfläche einer Halbkugel

Distributed Raytracing

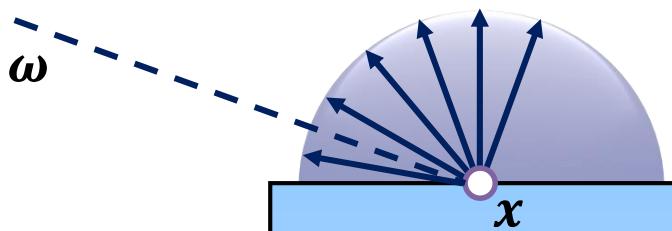
- d.h. wie Whitted-style Raytracing findet auch hier Rekursion statt, aber für jeden Schritt berechnen wir das (hemi-)sphärische Integral und nicht nur einzelne *ausgewählte* Sekundärstrahlen

$$L(x, \omega) = L_e(x, \omega) + \frac{2\pi}{N} \sum_{i=1}^N f_r(\omega_i, x, \omega) L_i(x, \omega_i) \cos \theta_i$$



kein Schnittpunkt von $\text{raycast}(x, \omega_i)$ mit anderen Oberflächen

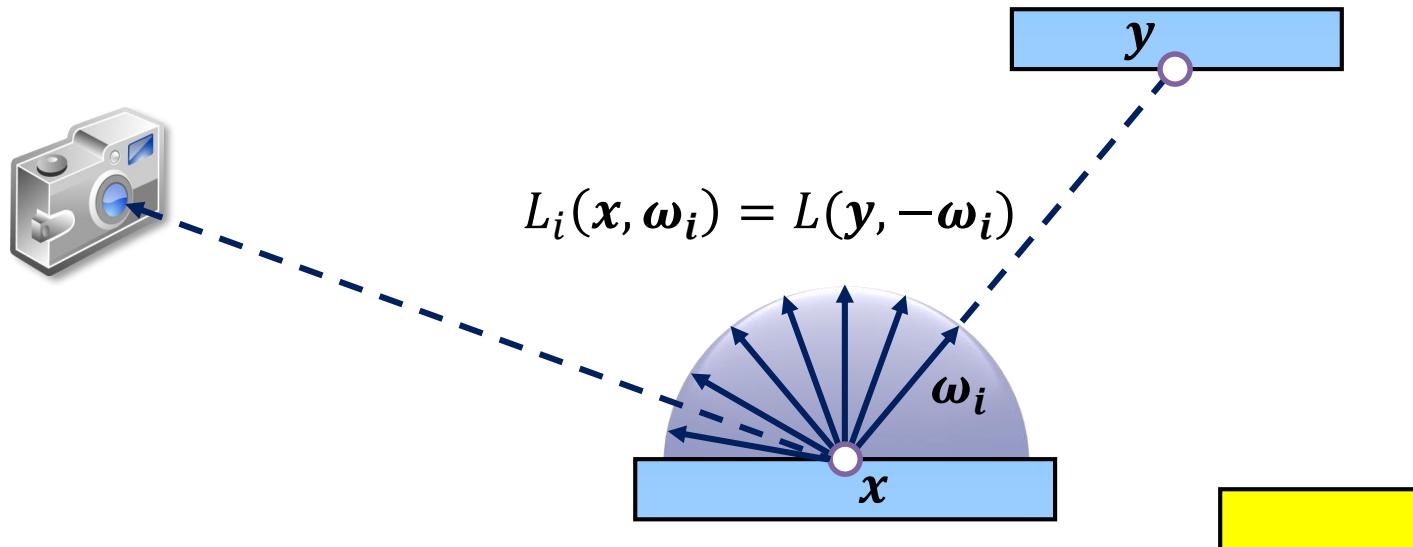
$$\Rightarrow L_i(x, \omega_i) = 0$$



Distributed Raytracing

- d.h. wie Whitted-style Raytracing findet auch hier Rekursion statt, aber für jeden Schritt berechnen wir das (hemi-)sphärische Integral und nicht nur einzelne *ausgewählte* Sekundärstrahlen

$$L(x, \omega) = L_e(x, \omega) + \frac{2\pi}{N} \sum_{i=1}^N f_r(\omega_i, x, \omega) L_i(x, \omega_i) \cos \theta_i$$

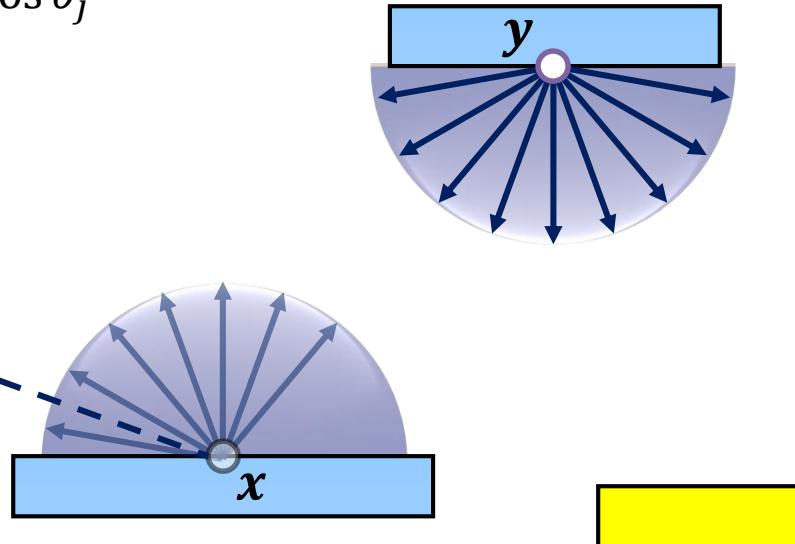


Distributed Raytracing

- d.h. wie Whitted-style Raytracing findet auch hier Rekursion statt, aber für jeden Schritt berechnen wir das (hemi-)sphärische Integral und nicht nur einzelne *ausgewählte* Sekundärstrahlen

$$L(x, \omega) = L_e(x, \omega) + \frac{2\pi}{N} \sum_{i=1}^N f_r(\omega_i, x, \omega) L_i(x, \omega_i) \cos \theta_i$$

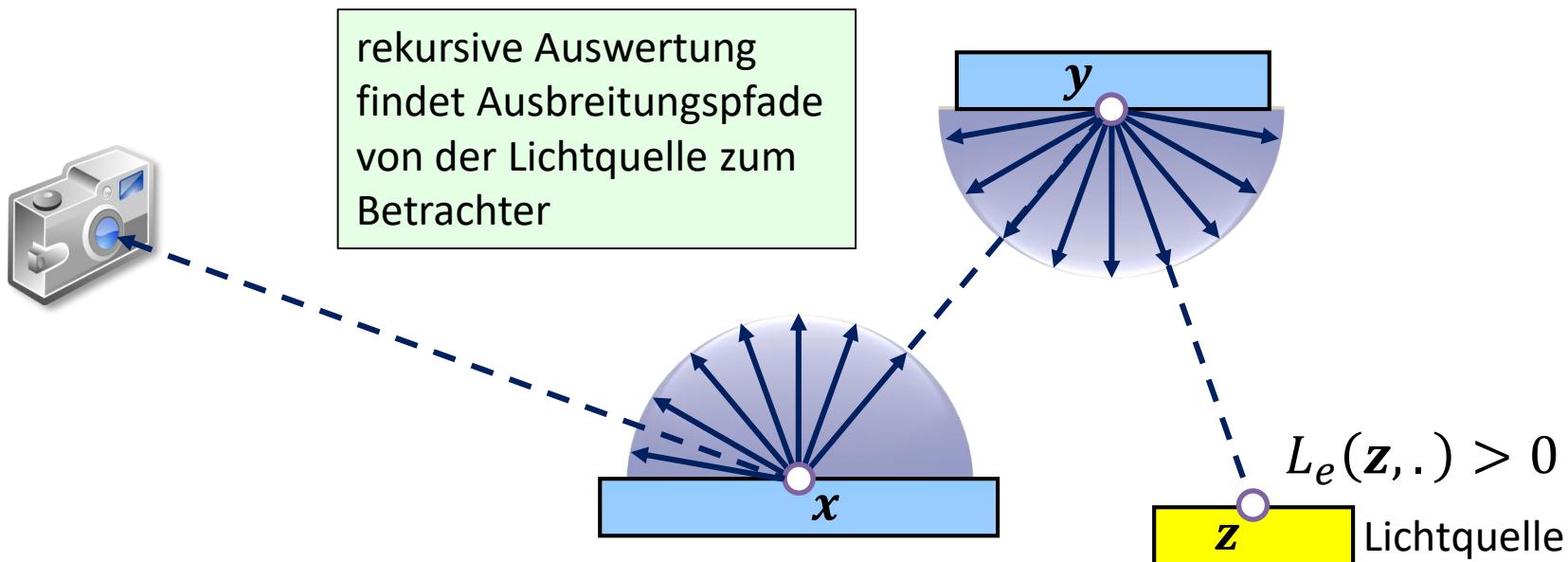
$$L(y, -\omega_i) = L_e(y, -\omega_i) + \\ + \frac{2\pi}{M} \sum_{j=1}^M f_r(\omega_j, y, -\omega_i) L_i(y, \omega_j) \cos \theta_j$$



Distributed Raytracing

- d.h. wie Whitted-style Raytracing findet auch hier Rekursion statt, aber für jeden Schritt berechnen wir das (hemi-)sphärische Integral und nicht nur einzelne *ausgewählte* Sekundärstrahlen

$$L(x, \omega) = L_e(x, \omega) + \frac{2\pi}{N} \sum_{i=1}^N f_r(\omega_i, x, \omega) L_i(x, \omega_i) \cos \theta_i$$



Weitere Schritte / Umformulierungen (Kurzfassung der Folien)

- ▶ Phong-Beleuchtungsmodell als BRDF (optional mit Energieerhaltung):

$$f_r(\omega_i, x, \omega) = k_d + k_s \cdot \frac{n+2}{2\pi} \cdot \left(((2\mathbf{N}(\mathbf{N} \cdot \omega_i) - \omega_i) \cdot \omega)^+ \right)^n / (\mathbf{N} \cdot \omega_i)^+$$

- ▶ zufällige Richtungen erzeugen (in Kugelkoordinaten)

- ▶ $\theta = \arccos(1 - 2\xi_1)$ und $\phi = 2\pi\xi_2$ mit zwei gleichverteilten Zufallszahlen $\xi_1, \xi_2 \in [0; 1]$
 - ▶ $\omega_i = (\sin \theta \cos \phi, \sin \theta \sin \phi, \cos \theta)$

- ▶ damit rekursiv berechnen, z.B. bis zu einer bestimmten Rekursionstiefe
 - ▶ z.B. $N = 64$ (für größere Rekursionstiefe: kleinere N bzw. M wählen)

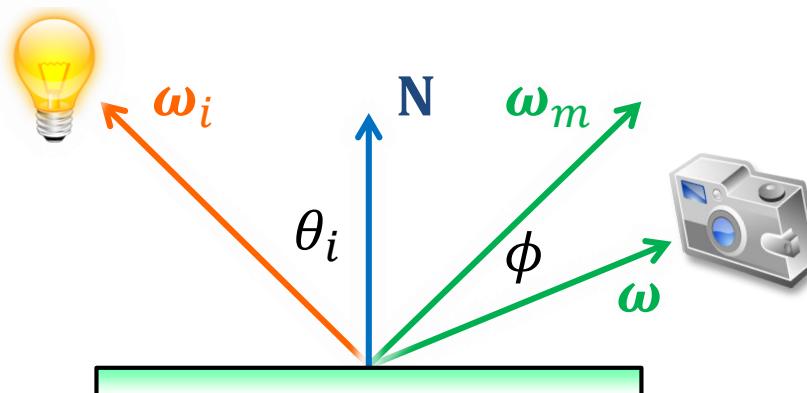
Distributed Raytracing (Details)

Wie kann man Distributed Raytracing einfach umsetzen?

- Phong-Beleuchtungsmodell als BRDF:

$$f_r(\omega_i, x, \omega) = k_d + k_s \left(((2\mathbf{N}(\mathbf{N} \cdot \omega_i) - \omega_i) \cdot \omega)^+ \right)^n / (\mathbf{N} \cdot \omega_i)^+$$

- ω_m ist die Spiegelungsrichtung von ω_i an \mathbf{N} : $\omega_m = 2\mathbf{N}(\mathbf{N} \cdot \omega_i) - \omega_i$
- ω ist die Richtung für die wir die BRDF auswerten
- Anm.: Skalarprodukt $\cos \theta_i = (\mathbf{N} \cdot \omega_i)^+$ ist Teil des Integranden, daher taucht es als Faktor bei k_d nicht auf und muß gleichermaßen für den spekularen Teil durch Division entfernt werden
- Energieerhaltung mit $k_s \cdot \frac{n+2}{2\pi} \cdot \cos^n \phi$ (und geeigneten k_d, k_s)



Distributed Raytracing (Details)

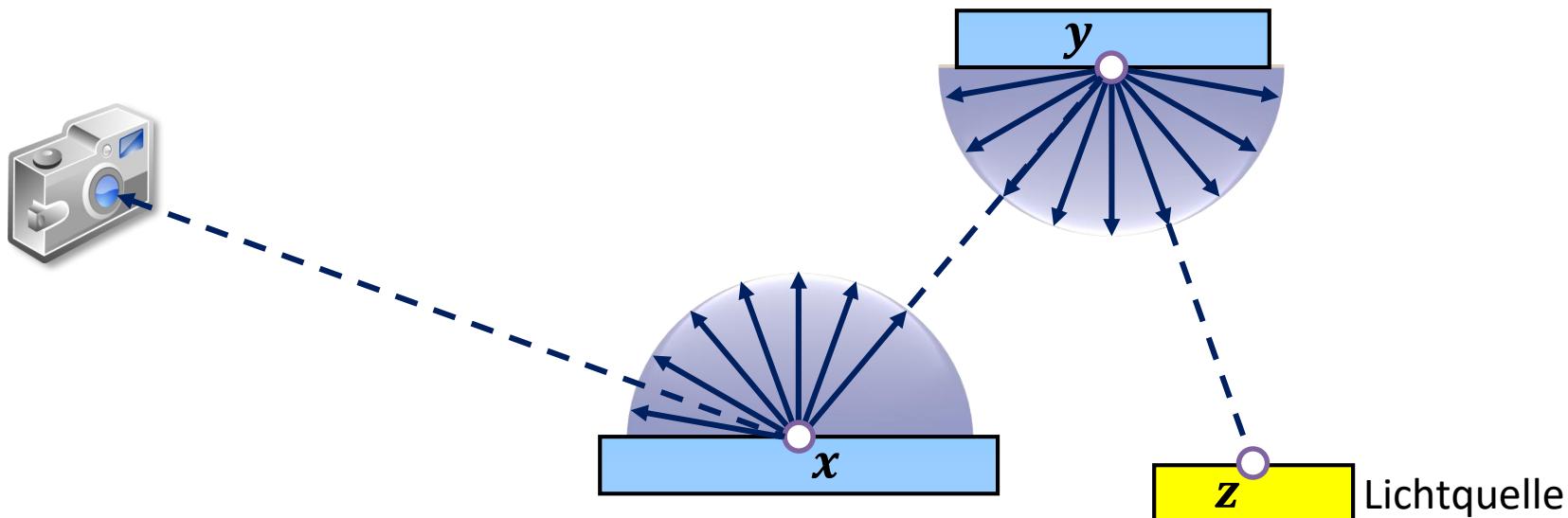
Wie kann man Distributed Raytracing einfach umsetzen?

- Phong-Beleuchtungsmodell als BRDF:

$$f_r(\omega_i, x, \omega) = k_d + k_s \left(((2\mathbf{N}(\mathbf{N} \cdot \omega_i) - \omega_i) \cdot \omega)^+ \right)^n / (\mathbf{N} \cdot \omega_i)^+$$

- Flächenlichtquellen

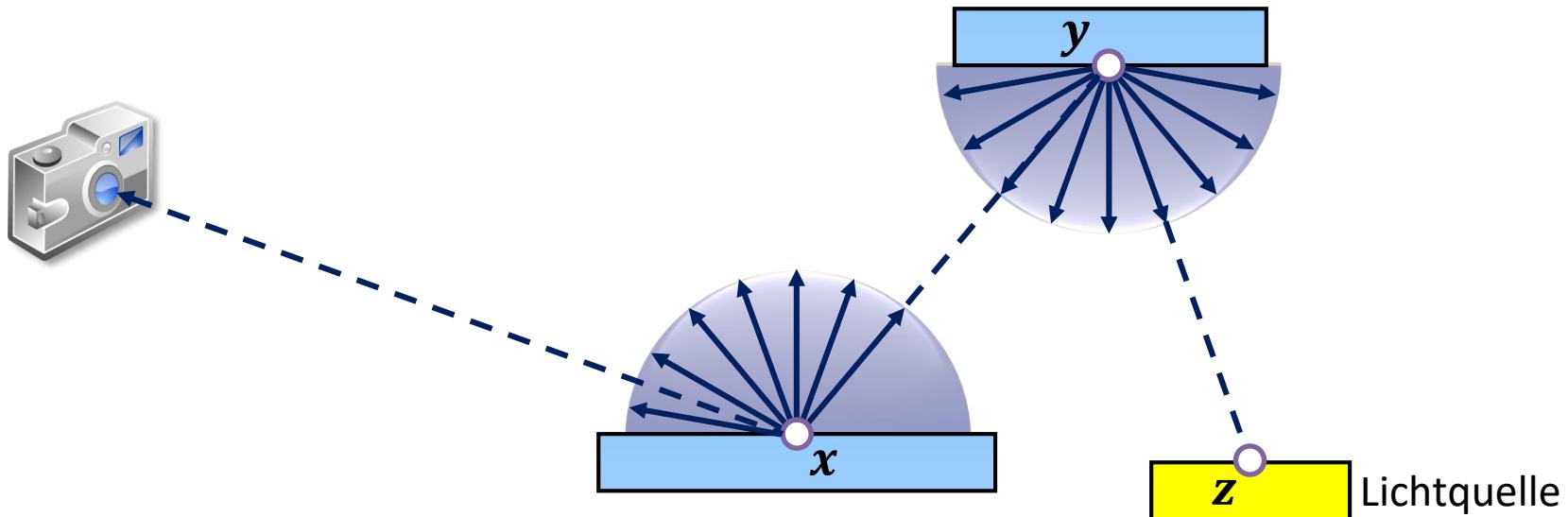
- füge zur Materialdefinition einen Emissionsterm hinzu
- $L_e(z, \omega) > 0$ für Lichtquellen (emittierende geometrische Primitive)
- wenn Sie DR probieren, dann am besten zuerst mit **großen Flächenlichtquellen**



Distributed Raytracing (Details)

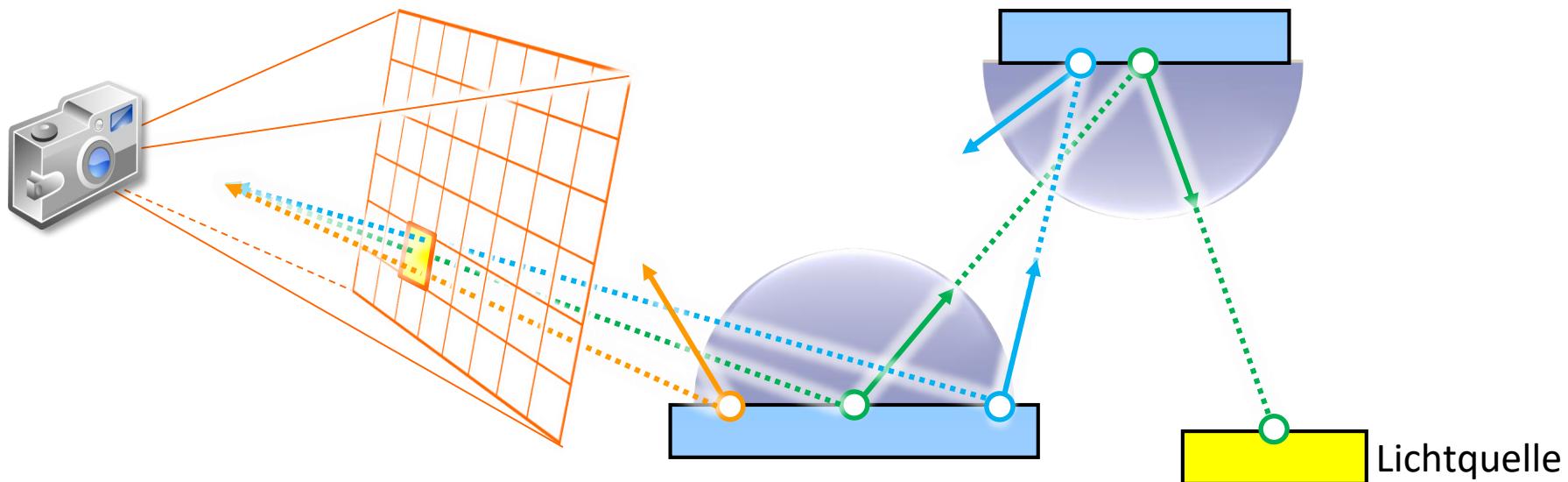
Wie kann man Distributed Raytracing einfach umsetzen?

- ▶ zufällige Richtungen erzeugen (in Kugelkoordinaten)
 - ▶ erzeuge 2 gleichverteilte Zufallszahlen $\xi_1, \xi_2 \in [0; 1]$
 - ▶ $\theta = \arccos(1 - 2\xi_1)$ und $\phi = 2\pi\xi_2$
 - ▶ $\omega_i = (\sin \theta \cos \phi, \sin \theta \sin \phi, \cos \theta)$
 - ▶ verwerfe die Richtung und erzeuge neue, wenn $(\mathbf{N} \cdot \omega_i) < 0$
- ▶ damit rekursiv berechnen, z.B. bis zu einer bestimmten Rekursionstiefe
 - ▶ z.B. $N = 64$ (für größere Rekursionstiefe: kleinere N bzw. M wählen)



Path Tracing

- ▶ verwendet mit Distributed Raytracing und praxisrelevanter:
nur jeweils ein weiterer Strahl, dafür mehrere „Pfade“ pro Pixel
- ▶ führt zu mehr abgetasteten Richtungen bei Flächen nahe der Kamera,
und geringerer Abtastung bei weiter entfernten (meist sinnvoll!)



Path Tracing

Konvergenz und schwierige Fälle



Pfade pro Pixel
 $N = 1024$

Fehler $\propto 1/\sqrt{N}$

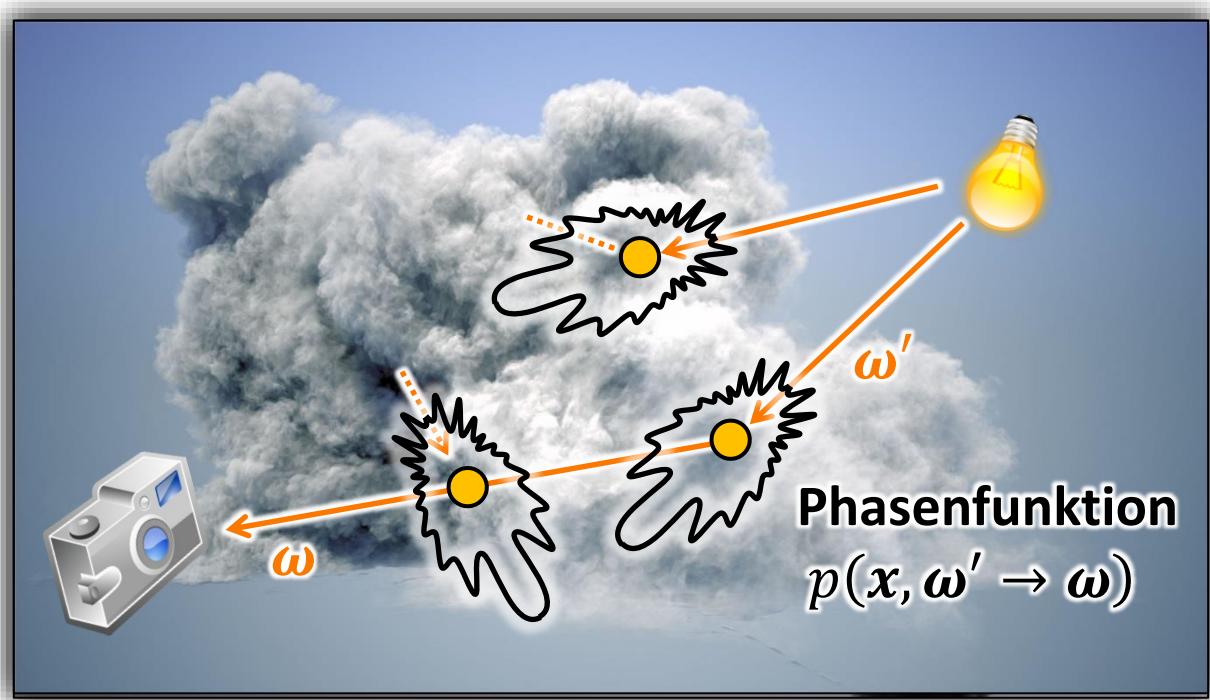
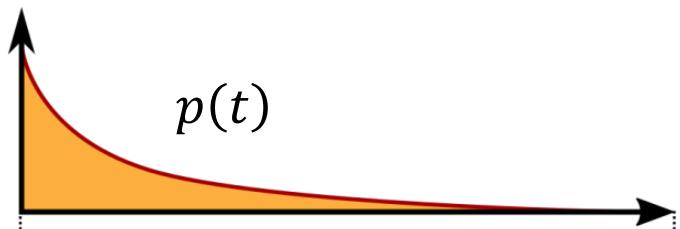
Path Tracing

- ▶ und wie lange dauert es ein Bild zu berechnen (1 Kern, 2 GHz, alte CPU)?
 - ▶ 32× stochastisches Supersampling
 - ▶ 256^2 Pixel, 480000 Dreiecke
 - ▶ Brute Force: $176462\text{sec} \approx 49\text{h}$
 - ▶ insgesamt ca. $6.3 \cdot 10^6$ Primär-, Sekundär- und Schattenstrahlen
 - ▶ ca. $3 \cdot 10^{12}$ Schnitttests
(nur Rekursionstiefe 1!)
- ▶ später in dieser Vorlesung: <<80sec
(BVH 18.3sec, Rendering 59sec)
→ 2200 mal so schnell!
(und optimierte, parallelisierte Varianten noch **DEUTLICH** schneller!)



Partizipierende Medien

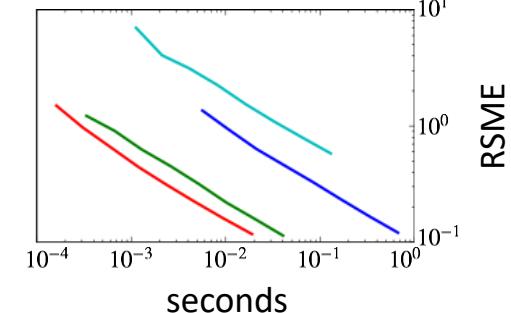
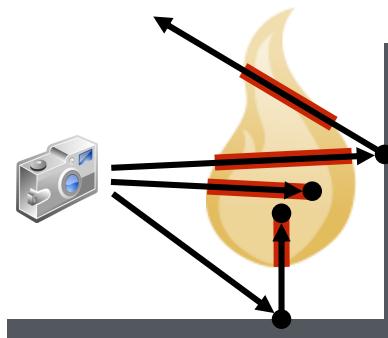
- ▶ Streu- und Absorptionskoeffizienten $\sigma_s(x), \sigma_a(x)$ [m⁻¹]



Forschung: Rendering realistischer Materialien



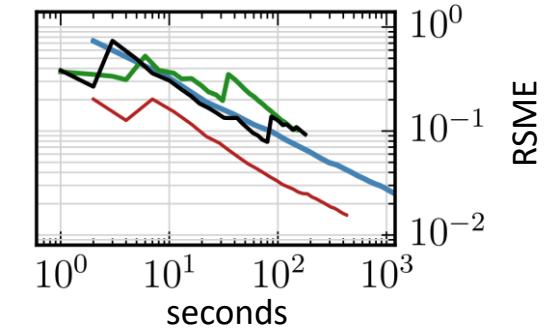
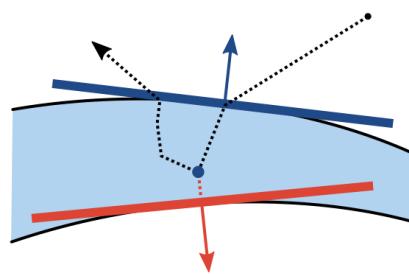
War for the Planet of the Apes



Line Integration for Rendering Heterogeneous Emissive Volumes
Simon, Hanika, Zirr, Dachsbacher,
Computer Graphics Forum (Proc. EG Symp. on Rendering), 2017

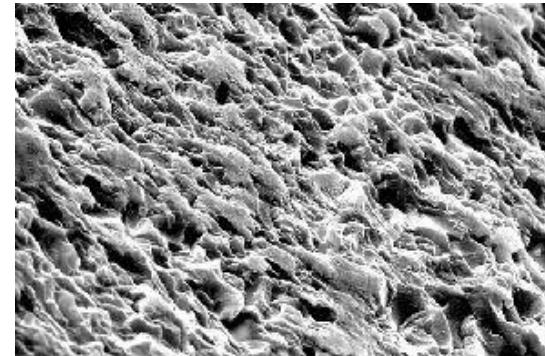
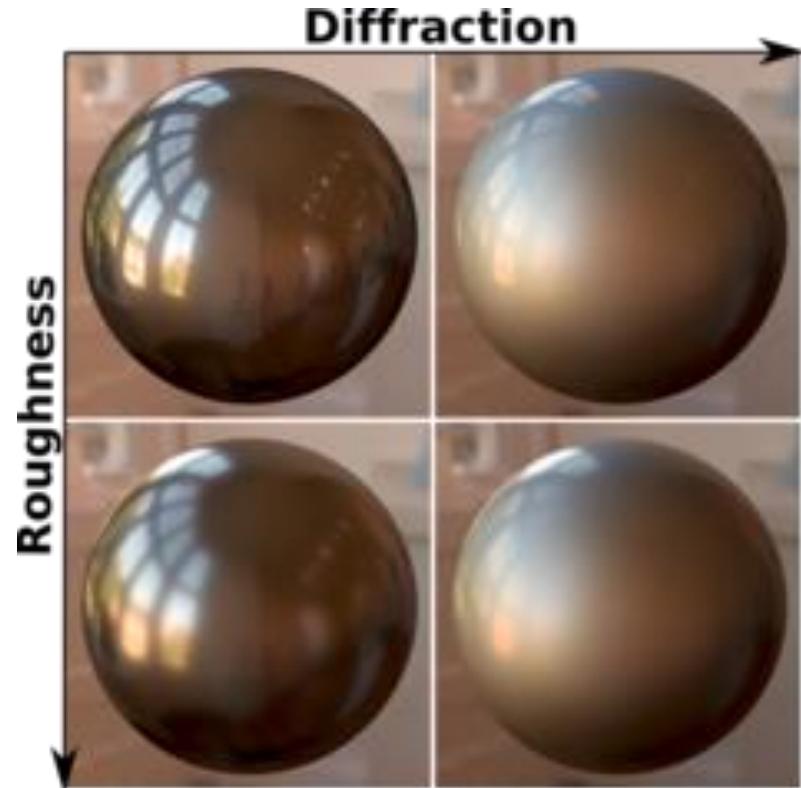


Alita: Battle Angel



Improving the Dwivedi Sampling Scheme
Meng, Hanika, Dachsbacher
Computer Graphics Forum (Proc. EG Symp. on Rendering), 2016

Beugungseffekte



A Two-Scale Microfacet Reflectance Model Combining Reflection And Diffraction

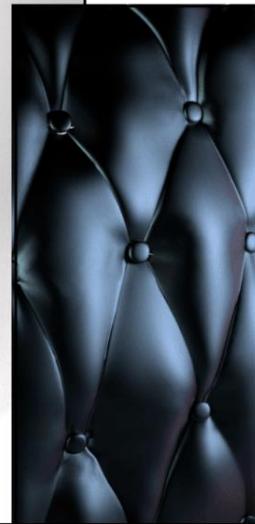
Holzschuch, Pacanowski

ACM Transactions on Graphics (Proc. of SIGGRAPH), 2017

- Interferenzen an dünnen Filmen auf der Oberfläche der Mikrofacetten



Mikrofacetten

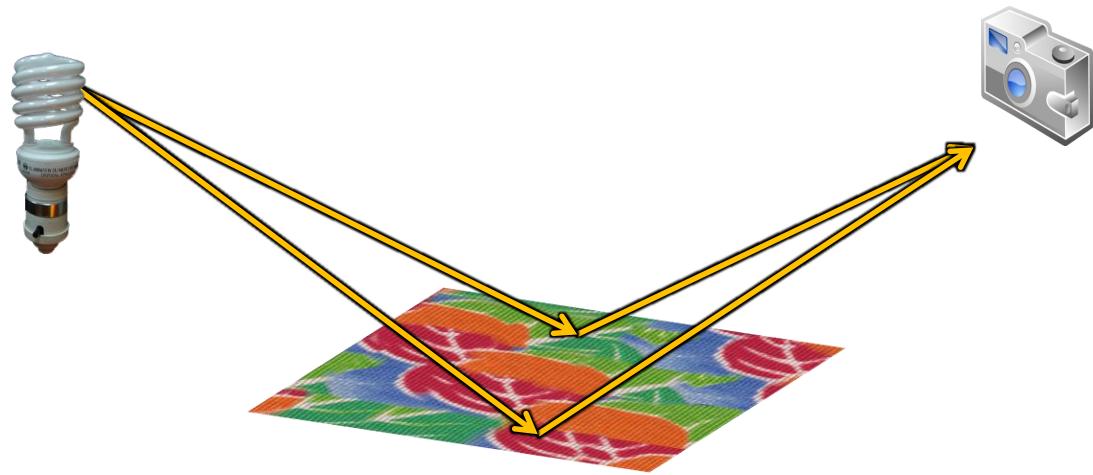


irisierende Mikrofacetten

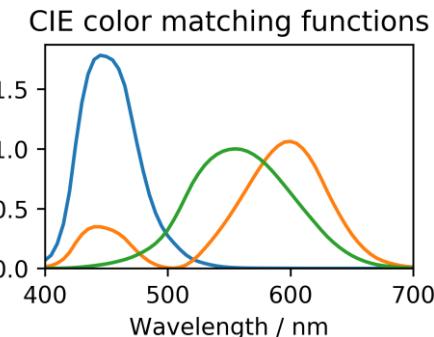
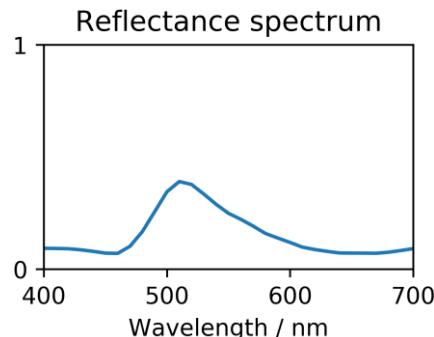
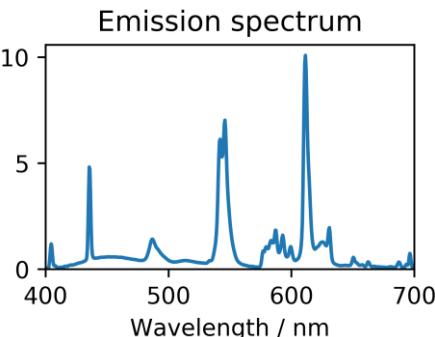
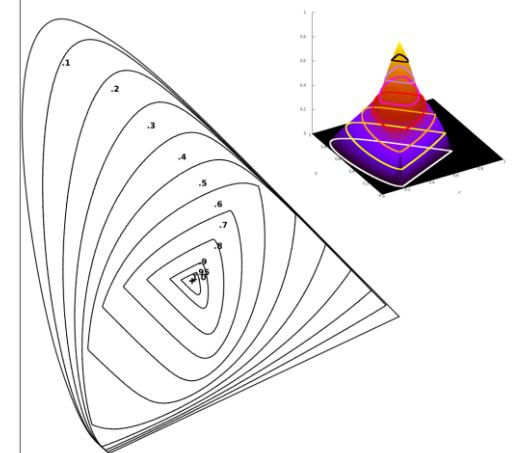


Spektrales Rendering

- ▶ verwendet aufgelöste Spektren (z.B. 5nm Schritte) statt RGB-Vektoren
- ▶ Beobachtung: Reflektanzen sind entweder sehr gesättigt oder sehr hell

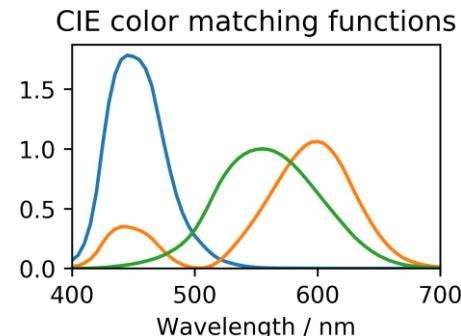
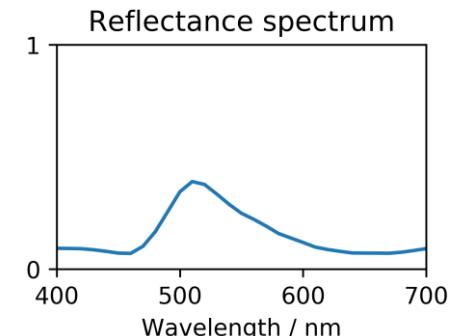
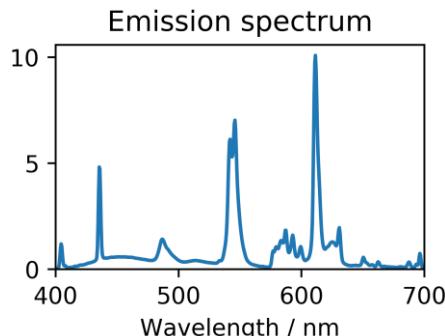


MacAdam's limit, illum. E



Spektrales Rendering

- warum spektrales Rendering?
 - ▶ exakte Reproduktion von Emission · Reflektanz
- was spricht gegen spektrales Rendering?
 - ▶ mehr Aufwand bei Licht-Material-Interaktionen
 - ▶ signifikant mehr Speicherbedarf von Texturen
- Spectral Upsampling-Methoden erzeugen ein Spektrum aus RGB-Werten
 - ▶ Vorsicht bei Reflektanzspektren: $\rho(\lambda) \leq 1$
 - ▶ natürliche Reflektanzspektren sind glatt
(noch stärkere Einschränkung im Vergleich zu $\rho(\lambda) \leq 1$)

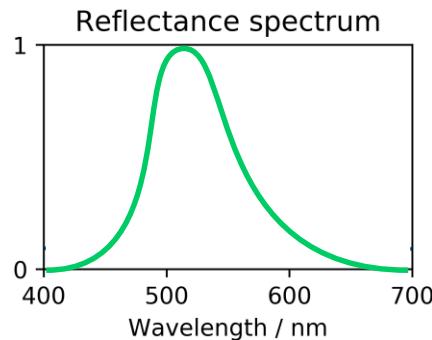


Spektrales Rendering

Beispiel: Unterschied zwischen Tristimulus- und spektralem Rendering

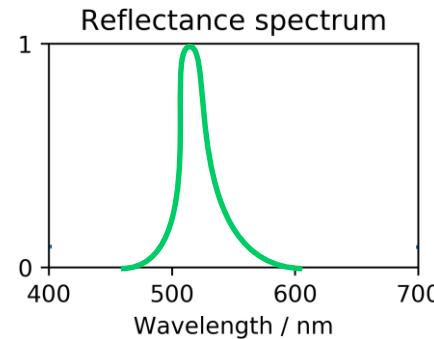
► bei n -facher Reflexion ($n > 1$) wird der Unterschied deutlich:

$$(\text{RGB})^n = (0,1,0)^T$$

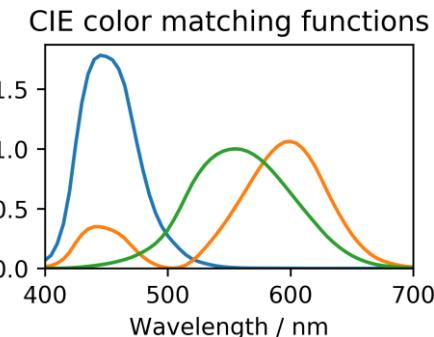
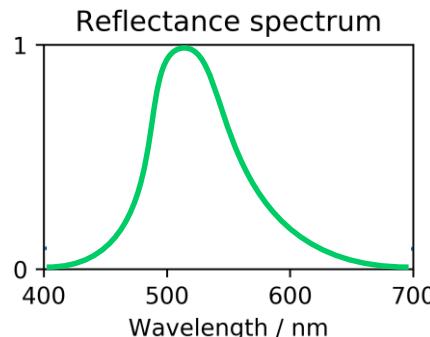
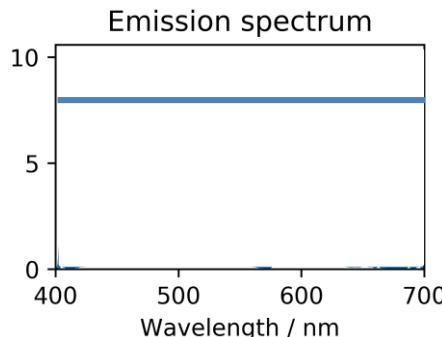


$$\rho(\lambda)^n$$

➡



$$\text{RGB} = (0,1,0)^T$$



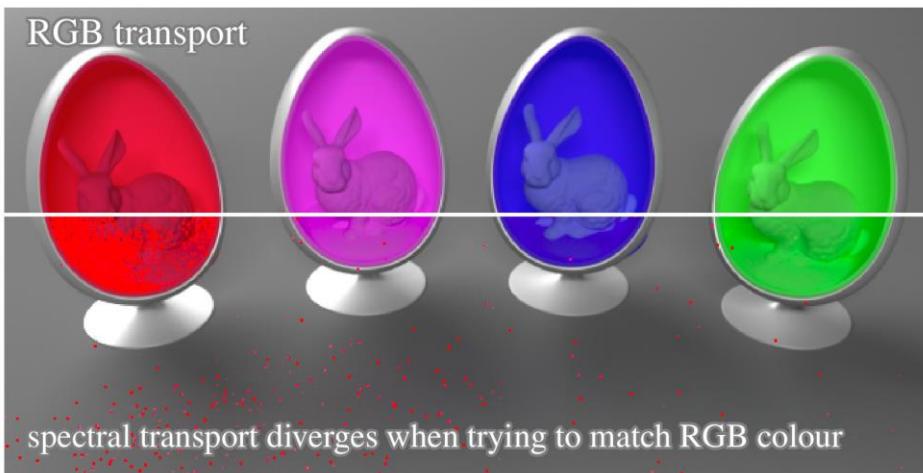
$d\lambda$

\int

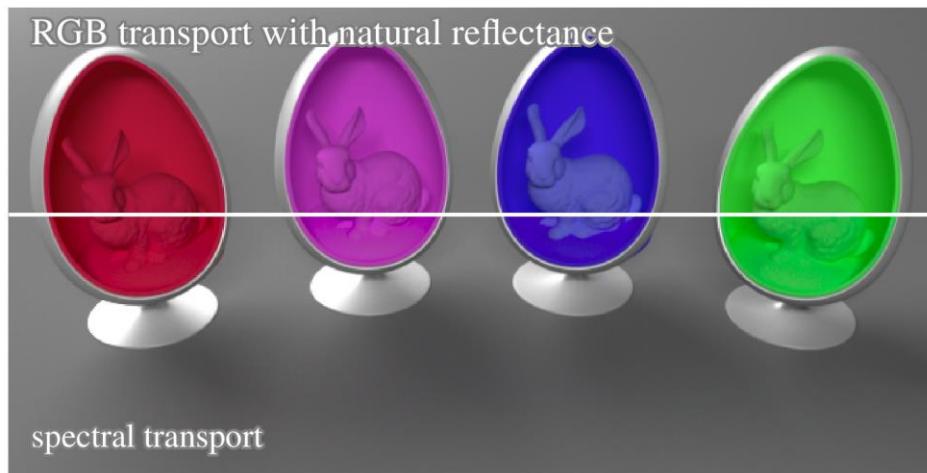
Spektrales Rendering

Beispiel: Unterschied zwischen Tristimulus- und spektralem Rendering

- ▶ bei n -facher Reflexion ($n > 1$) wird der Unterschied deutlich
- ▶ durch Einschränkungen bei der Wahl der Tristimulus-Werte für Reflektanzen wird der Unterschied geringer



Reflektanzspektren übersättigt



Reflektanzspektren: glatt, $\rho(\lambda) \leq 1$ und so gewählt, dass „RGB=spektral“

Spektrales Rendering

► wie kommen in der Realität gesättigte Farben zustande? Fluoreszenz!



Spektrales Rendering

- wie kommen in der Realität gesättigte Farben zustande? Fluoreszenz!
- Idee: erweiterte spektrales Upsampling um einen fluoreszierenden Anteil
 - verwendet, wenn Reflektanz alleine nicht genügt
- erfordert Lichttransportsimulation mit fluoreszierenden Materialien
 - größtenteils ungelöstes Problem ⇒ Forschungsthema

