# Architektur und Programmierung von Grafik- und Koprozessoren

## Rendering Algorithmen

Stefan Zellmann

Lehrstuhl für Informatik, Universität zu Köln

SS2019

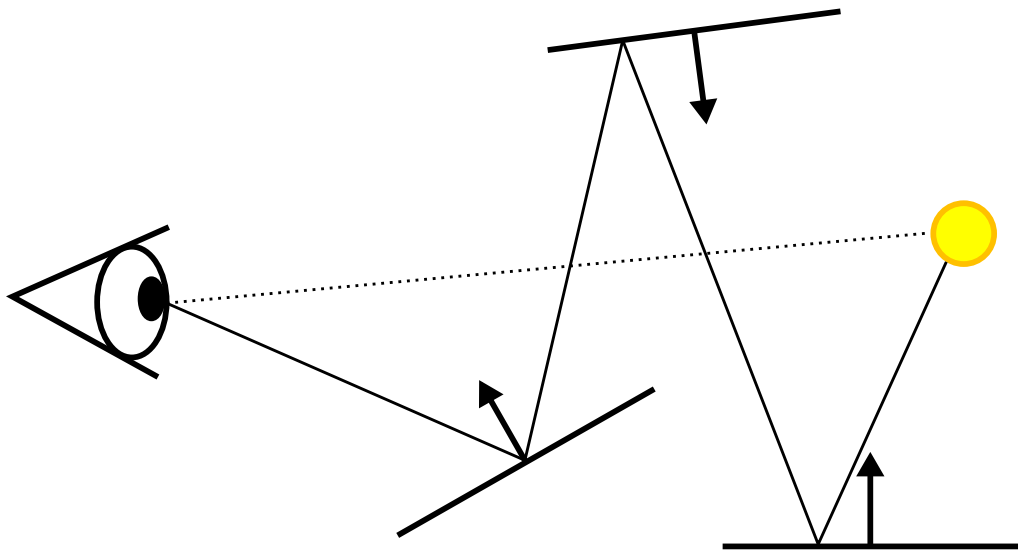
# Strahlverfolgung

# Strahlverfolgung

- Ansatz basiert auf Grundprinzip, dass sich Lichtpartikel (*Photonen*), die von einer Lichtquelle aus *emittiert* werden, entlang gerader Linien durch den Raum bewegen, und dass Lichtpartikel nicht mit anderen Lichtpartikeln interagieren.
- Jedoch Interaktion mit *massebehafteten* Partikeln $\Rightarrow$ Streuung und Absorption.
- Es ergeben sich *Pfade* - i. Allg. von Interesse sind diejenigen vom Licht zum Betrachter.
- Häufig ist das zu aufwendig - daher paart man Strahlverfolgung oft mit lokaler Beleuchtung.

# Symmetrie

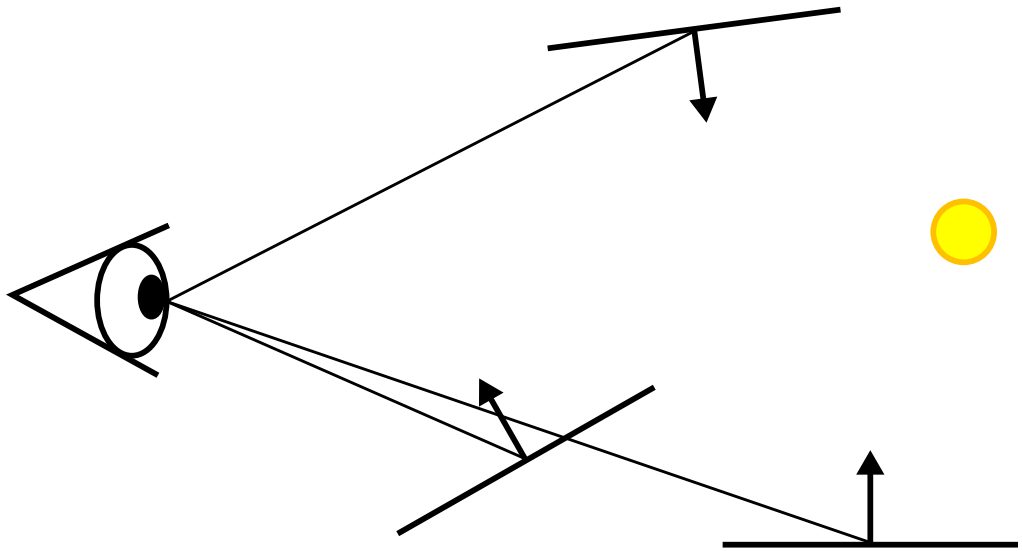
Aufgrund von Symmetrieeigenschaften darf man das Problem umformulieren, sodass Pfade vom Betrachter zum Licht gesucht werden (beginne Strahlverfolgung vom Betrachter aus).



In der Abbildung: ein *direkter Lichtpfad*, und ein Pfad, der aufgrund mehrerer Licht/Oberflächeninteraktionen entsteht.

# Primärsichtbarkeit

Für unsere Betrachtungen wollen wir uns zuerst nur für Pfade mit Länge eins hin zu einer Oberfläche interessieren (*Primärstrahlen*).



An den Interaktionspunkten wird lokales Beleuchtungsmodell ausgewertet. Eingabedaten genau wie bei Rasterisierung. Das optische Ergebnis sollte äquivalent zu Rasterisierung sein.

# Strahlen

Strahlen definiert man bzgl. *Ursprung* und *Richtung*. Den Ursprung drückt man durch eine 3D Position aus, die Richtung durch einen Einheitsvektor:

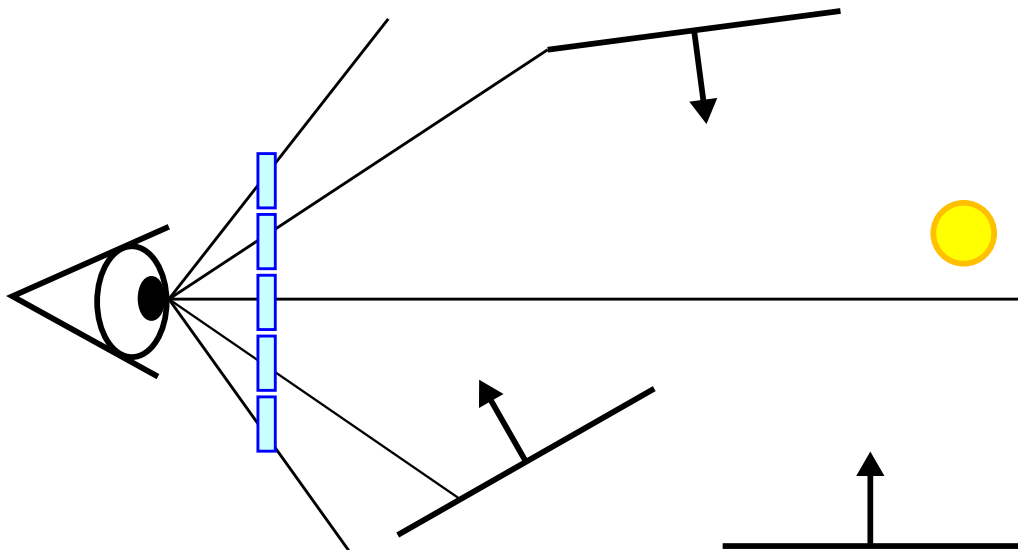
$$R = \{\boldsymbol{o}, \vec{d}\} \tag{27}$$

Die Distanz, die entlang eines Strahls zurückgelegt wurde, parametrisiert man mittels eines Skalars $t$. Darüber lassen sich z. B. Schnittpunkte mit Geometrie entlang des Strahls berechnen.

$$I = \boldsymbol{o} + \vec{d}t. \tag{28}$$

# Primärstrahlen

Primärstrahlen generiert man so, dass ihr Ursprung beim Betrachter liegt und dass sie die einzelnen Pixel des Rasters *samplen*.



# Schnitttests

Die Basisoperation bei der Strahlverfolgung ist es, zu testen, ob ein Strahl eine Oberfläche schneidet. Dazu setzt man die Strahlgleichung in der parametrischen Form (28) in die Oberflächengleichung ein und löst nach $t$ auf.

# Strahl/Ebenen Schnitttest

Für den Schnitt zwischen Strahl und Ebene wollen wir dies kurz exemplarisch erläutern. Wir betrachten die Ebenengleichung in Parameterform

$$(\boldsymbol{x} - \boldsymbol{p_0}) \cdot \vec{n} = 0, \tag{29}$$

wobei $\boldsymbol{p_0}$ ein beliebiger Punkt in der Ebene und $\vec{n}$ ein normalisierter Richtungsvektor senkrecht zur Ebene (geometrische Normale) ist. Wir setzen die Strahlgleichung ein und erhalten

$$(\boldsymbol{o} + \vec{d}t - \boldsymbol{p_0}) \cdot \vec{n} = 0. \tag{30}$$

# Strahl/Ebenen Schnitttest

$$(\boldsymbol{o} + \vec{d}t - \boldsymbol{p_0}) \cdot \vec{n} = 0$$

Wir lösen nach $t$ auf und erhalten

$$t = \frac{(\boldsymbol{p_0} - \boldsymbol{o}) \cdot \vec{n}}{\vec{d} \cdot \vec{n}}. \tag{31}$$

Ist der Nenner 0, wissen wir, dass der Strahl parallel zur Ebene verläuft. Diesen Fall behandeln wir speziell, andernfalls gibt $t$ die Distanz zwischen Strahlursprung und Ebene an.

Schnitttests mit einfachen, polynomiellen Oberflächen mit wenig Koeffizienten (z. B. Dreiecke, Quadriken etc.) führt man analog durch.

# Strahlstärke

- Uns interessiert die *Strahlstärke* (engl.: Radiance), die entlang der Pfade aufgesammelt wird. Diese wird auf der Bildfläche gemessen ($\Rightarrow$ *Irradiance*) und in RGB umgewandelt.
- Physikalisch basierte Ray Tracer führen Berechnungen in radiometrischen Einheiten (Radiance, Irradiance, Flux u.s.w.) durch und wandeln am Ende der Pipeline in RGB um.
- Einfache Ray Tracer führen alle Berechnungen in RGB durch (Performanzgründe).

# Primärstrahlverfolgung

```
function PRIMAERSTRAHLVERFOLGUNG(V,M,L,f,MV,PR,VP)
    for all xy ∈ Pixel do
        r ←GENERIEREPRIMAERSTRAHL(xy)
        t ← ∞
        for all v1, v2, v3 ∈ V do
            tt ← INTERSECT(r,v1,v2,v3)
            if tt < t then
                t ← tt
            end if
        end for
        P ← r.o + t * r.d
        for all Lj ∈ Lichtquellen do
            PixelFarbe += BELEUCHTE(P,M,Lj,f)
        end for
    end for
end function
```

# Primärstrahlverfolgung

Wir iterieren über jedes Bildschirmpixel und generieren Primärstrahl.

**function** PRIMAERSTRAHLVERFOLGUNG(V,M,L,f,MV,PR,VP)
    **for all** xy $\in$ Pixel **do**
        $r \leftarrow$GENERIERE PRIMAERSTRAHL(xy)
        ...
    **end for**
**end function**

# Primärstrahlverfolgung

Wir suchen dann das dem Strahlursprung am nächsten gelegene Dreieck.

```
function PRIMAERSTRAHLVERFOLGUNG(V,M,L,f,MV,PR,VP)
    for all xy ∈ Pixel do
        ...
        t ← ∞
        for all v1, v2, v3 ∈ V do
            tt ← INTERSECT(r,v1,v2,v3)
            if tt < t then
                t ← tt
            end if
        end for
        ...
    end for
end function
```

# Primärstrahlverfolgung

Am Schnittpunkt werten wir ein lokales Beleuchtungsmodell aus. Später lernen wir Strahlverfolgungsalgorithmen kennen, die hier globale Beleuchtung berechnen.

**function** PRIMAERSTRAHLVERFOLGUNG(V,M,L,f,MV,PR,VP)
  **for all** xy $\in$ Pixel **do**
    ...
    $P \leftarrow r.o + t * r.d$
    **for all** $L_j \in$ Lichtquellen **do**
      PixelFarbe $+=$ BELEUCHTE(P,M,$L_j$,f)
    **end for**
  **end for**
**end function**

# Primärstrahlverfolgung - Laufzeitkomplexität

- Der Algorithmus PRIMAERSTRAHLVERFOLGUNG hat eine asymptotische worst-case Laufzeit von $O(VP) \times (O(V) + O(L))$. Dies entspricht der asymptotischen Komplexität von Deferred Shading.
  - Dies entspricht der Intuition: Beleuchtungsberechnungen werden erst durchgeführt, nachdem Sichtbarkeitstest entschieden.
  - Der Algorithmus PRIMAERSTRAHLVERFOLGUNG unterstützt keine teiltransparente Geometrie.
- Basisalgorithmus setzt sich aus den Phasen *intersect* und *shade* zusammen. Je nach Geometriekomplexität sowie Materialeigenschaften und Beleuchtungsverhältnissen ist die eine oder andere Phase die aufwendigere.

# Primärstrahlverfolgung - Paralleler Algorithmus

Die intuitivste Möglichkeit, diesen Algorithmus zu parallelisieren, ist über die äußere Schleife.

**function** PRIMAERSTRAHLVERFOLGUNG(V,M,L,f,MV,PR,VP)
  **for all** xy $\in$ Pixel **do in parallel**
    $r \leftarrow$GENERIERESTRAHL(xy)
    INTERSECT(r)
    SHADE(r)
  **end for**
**end function**

# Primärstrahlverfolgung - Paralleler Algorithmus

- Die Arbeitskomplexität beträgt $W(n) = O(VP) \times (O(V) + O(L))$. Die Zeitkomplexität beträgt $S(n) = O(V) + O(L)$.
  - Beleuchtung ist "deferred": finde erst nächsten Schnitt mit Geometrie. Erst dann wird beleuchtet.
- Der parallele Algorithmus PRIMAERSTRAHLVERFOLGUNG ist auf der EREW PRAM lauffähig.
  - Alle Pixel unabhängig.
  - Anders als bei RASTERISIERUNG wird kein geteilter Speicher verwendet, in den Prozessoren *gleichzeitig* schreiben.

# Primärstrahlverfolgung

Rendering, nur Primärsichtbarkeit. Diffuse Materialien sowie eine Punktlichtquelle.

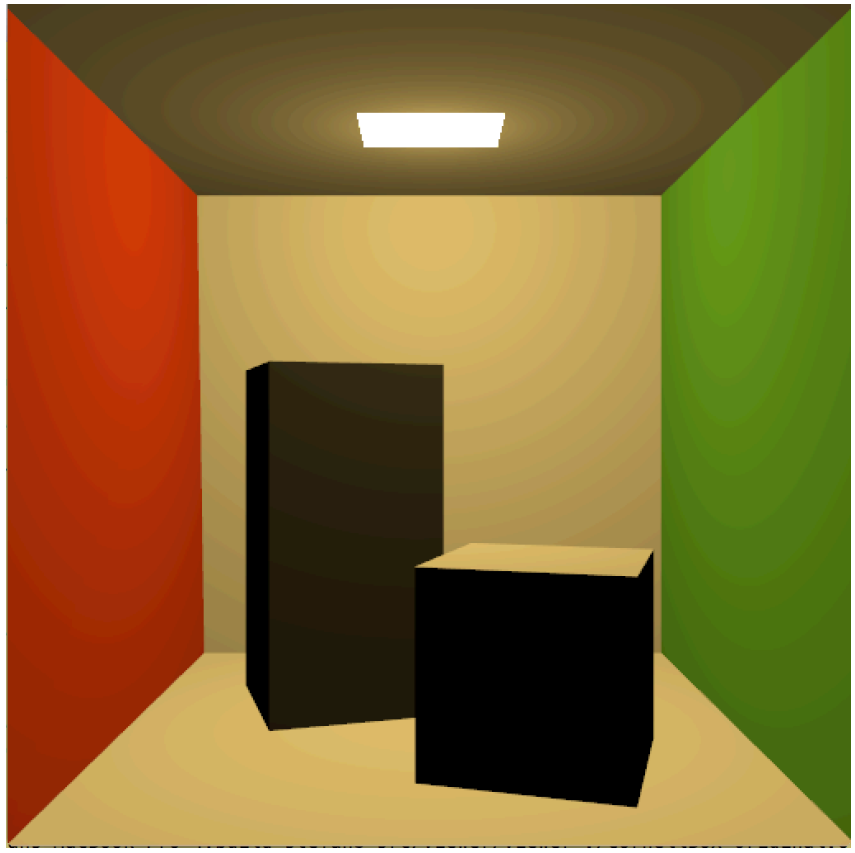


Abbildung: Modell: Cornell Box (https://www.graphics.cornell.edu/online/box/), Rendering: Stefan Zellmann

# Rendering nur mit Primärsichtbarkeit

Die drei bisher vorgestellten Algorithmen würden für die Cornell Box Szene qualitativ das gleiche Resultat erzeugen.

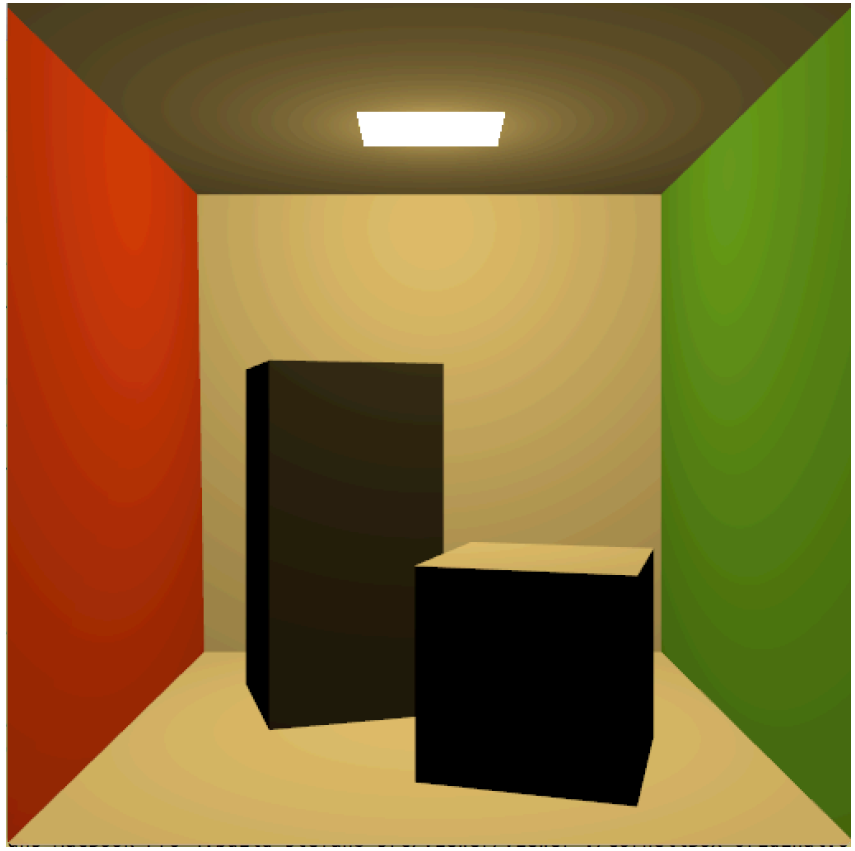


Abbildung: Modell: Cornell Box (https://www.graphics.cornell.edu/online/box/), Rendering: Stefan Zellmann

# Primärstrahlverfolgung - Paralleler Algorithmus

## Diskussion

Anscheinend ist der Algorithmus PRIMAERSTRAHLVERFOLGUNG ggü. dem Algorithmus RASTERISIERUNG überlegen (deferred, EREW). Was sind / waren Gründe, dass RASTERISIERUNG in GPUs implementiert ist und PRIMAERSTRAHLVERFOLGUNG nicht?

# Mehr Strahlen

Strahlverfolgungsalgorithmen intuitiv um globale Effekte erweiterbar. Dafür *Sekundärstrahlen* nötig - z. B. Schatten:

**function** PUNKTLICHTSCHATTEN(V,M,L,f,MV,PR,VP)
  **for all** xy $\in$ Pixel **do in parallel**
    $r \leftarrow$GENERIEREPRIMAERSTRAHL(xy)
    INTERSECT(r)
    $P \leftarrow r.o + t * r.d$
    **for all** $L_j \in$ Lichtquellen **do**
      $\vec{l} \leftarrow$ NORMALISIERE$(L_{posj} - P)$
      $sr \leftarrow \{P, \vec{l}\}$ ▷ Schattenstrahl
      **if** *not* INTERSECT(sr) **then**
        PixelFarbe $+=$ BELEUCHTE(P,M,$L_j$,f)
      **end if**
    **end for**
  **end for**
**end function**

# Mehr Strahlen

## Links: Primärstrahlen, rechts: Cornell Box mit Schatten

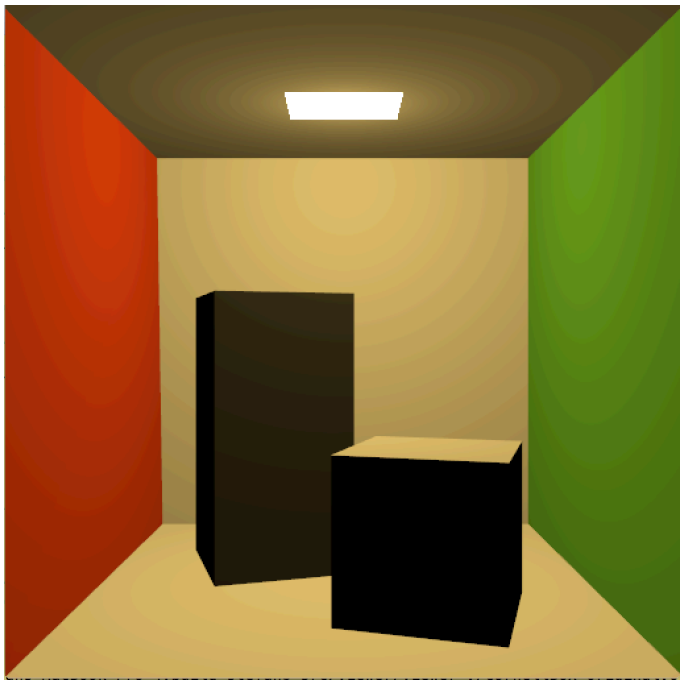


Abbildung: Modell: Cornell Box (https://www.graphics.cornell.edu/online/box/), Rendering: Stefan Zellmann

# Mehr Strahlen

Noch mehr Sekundärstrahlen: perfekt spekulare Reflektion.

```
function Verfolge(r)
    Intersect(r)
    Shade(r)
    if Spiegelt(m ∈ M) then
        r ← Reflektiere(r)
        Verfolge(r)
    end if
end function
function Strahlverfolgung(V,M,L,f,MV,PR,VP)
    for all xy ∈ Pixel do in parallel
        r ←GenerierePrimaerstrahl(xy)
        Verfolge(r)
    end for
end function
```

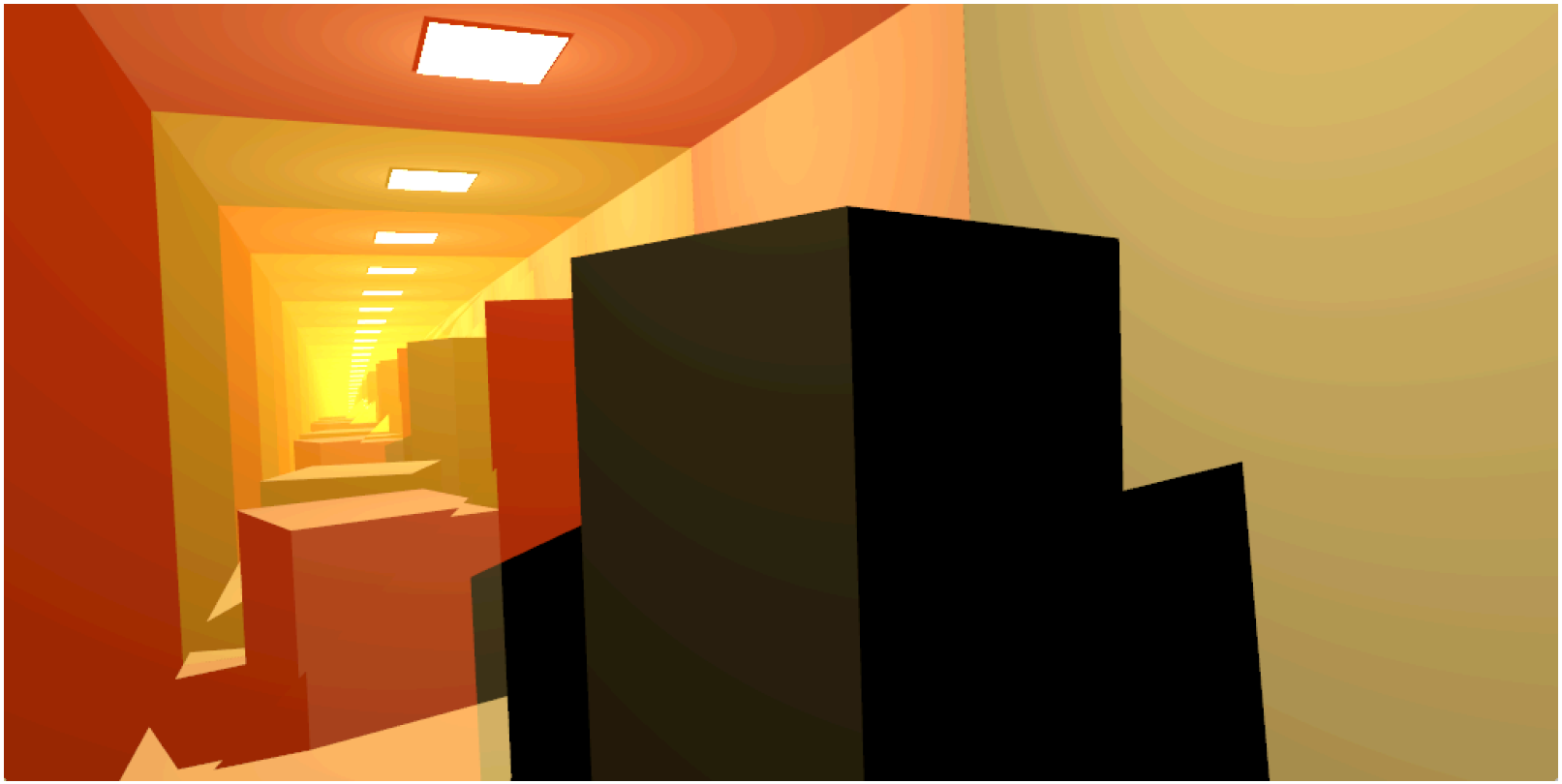
# Reflektion

## Rechts: Cornell Box mit Schatten und Reflektion

Abbildung: Modell: Cornell Box (https://www.graphics.cornell.edu/online/box/), Rendering: Stefan Zellmann

# Reflektion

## “Hall of Mirrors” Effekt



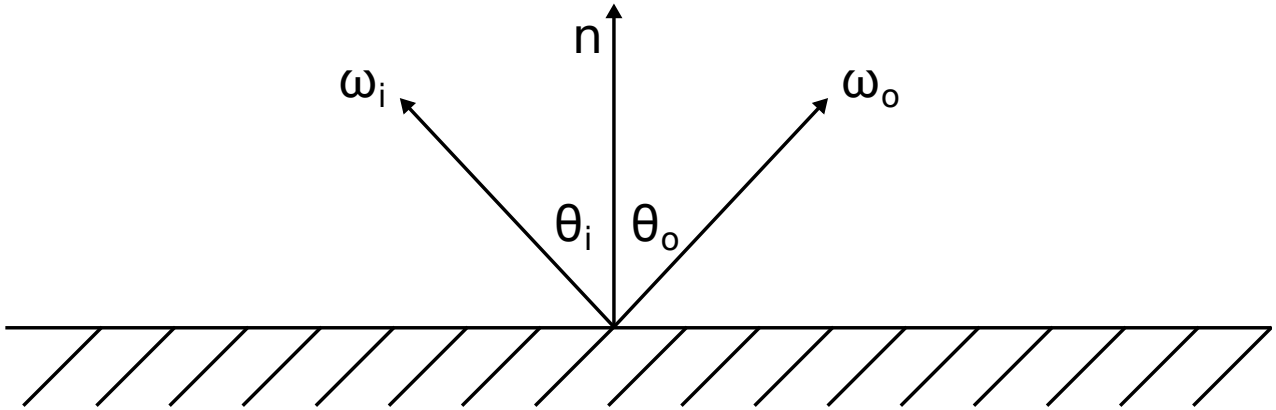
Vermeide unendlich viele “Bounces”: limitiere Rekursionstiefe durch Konstante; akkumuliere Reflektionskoeffizient + Abbruchkriterium (Koeffizient $< \epsilon$).

# Reflektion

## Fixe Rekursionstiefe

```
function VERFOLGE(r, D)
    INTERSECT(r)
    SHADE(r)
    if SPIEGELT(m ∈ M) and D < 5 then                    ▷
        r ← REFLEKTIERE(r)
        VERFOLGE(r, D+1)
    end if
end function
function STRAHLVERFOLGUNG(V,M,L,f,MV,PR,VP)
    for all xy ∈ Pixel do in parallel
        r ←GENERIEREPRIMAERSTRAHL(xy)
        VERFOLGE(r, 0)
    end for
end function
```

# Perfekt Spekulare Reflektion




$\vec{n}$: Oberflächennormale

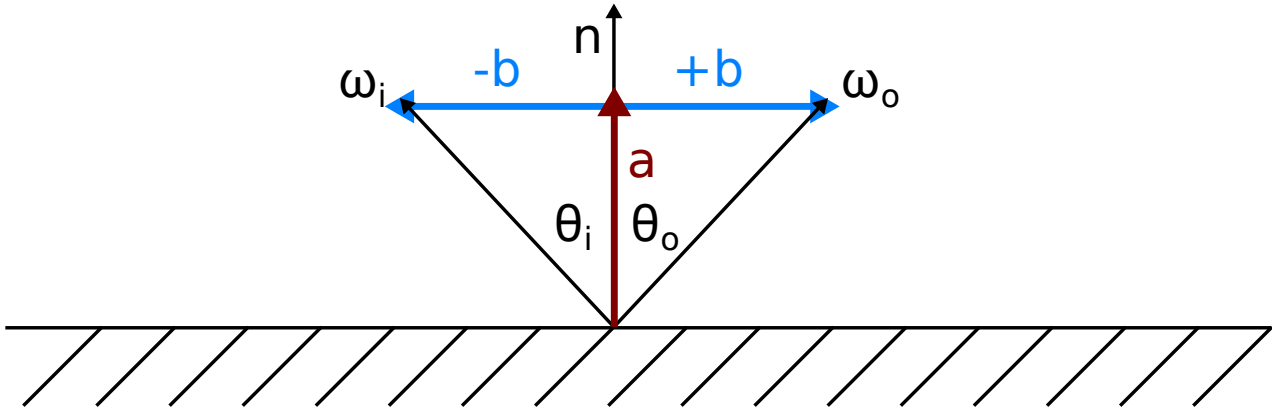
$\vec{\omega_i}$: Vektor zum Licht (direkt oder indirekt!)

$\theta_i$: Winkel zwischen Lichtrichtung und Oberflächennormale

$\vec{\omega_o}$: Perfekt spekulare Reflektionsrichtung

$\theta_i$: Winkel zwischen Reflektionsrichtung und Oberflächennormale

# Perfekt Spekulare Reflektion

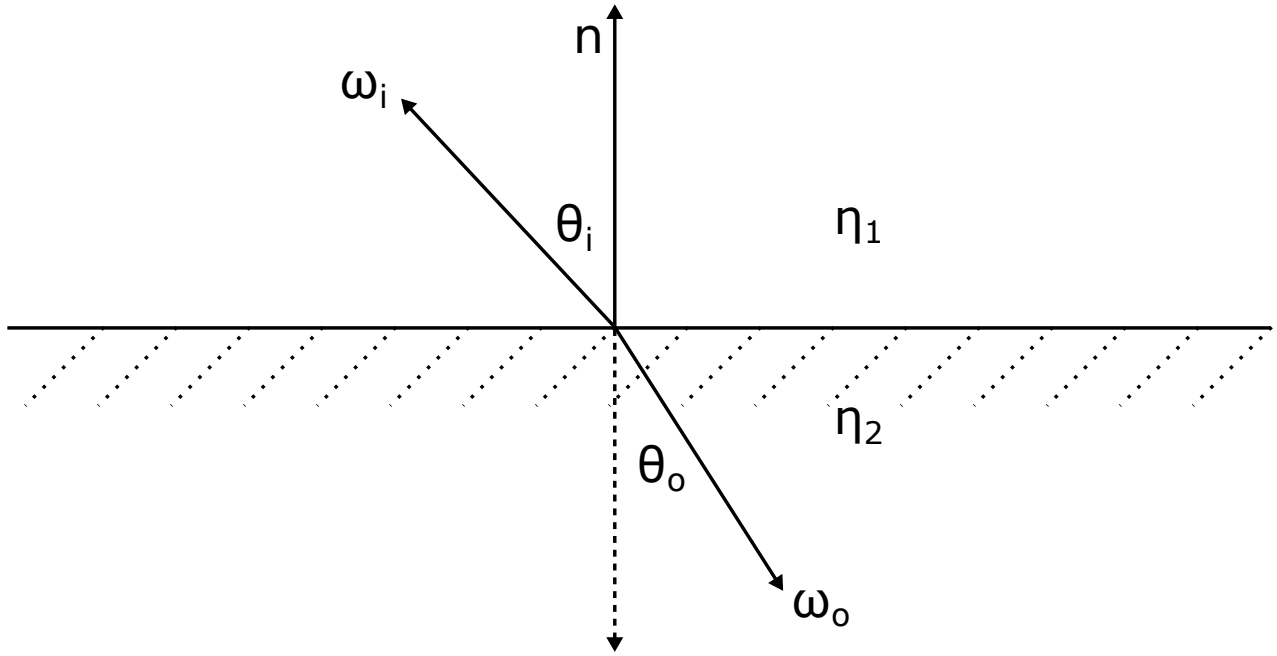



Wir suchen $\vec{\omega_o}$.
$\vec{\omega_i} = \vec{a} - \vec{b}$
$\vec{\omega_o} = \vec{a} + \vec{b}$

Da $\theta_i = \theta_o$, erhalten wir durch Einsetzen:
$\vec{a} = \vec{n}cos(\theta_i) = \vec{n}cos(\theta_o) \stackrel{!}{=} \vec{n}cos(\theta)$
$\vec{b} = \vec{n}cos(\theta) - \vec{\omega_i}$

$\Rightarrow \vec{\omega_o} = 2\vec{n}cos(\theta) - \vec{\omega_i}$

# Perfekt Spekulare Brechung

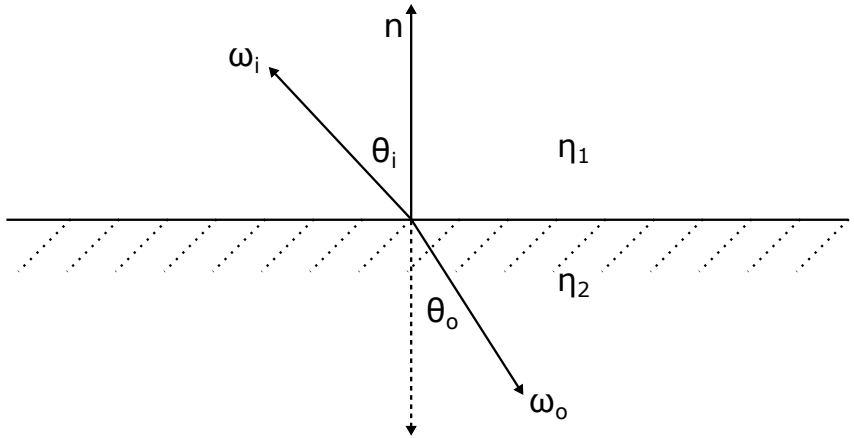



$\eta_1$: Brechungsindex von Medium 1
$\eta_2$: Brechungsindex von Medium 2

Licht bewegt sich mit Geschwindigkeit proportional zum Brechungsindex $\eta$ durch *Medium*.

# Perfekt Spekulare Brechung

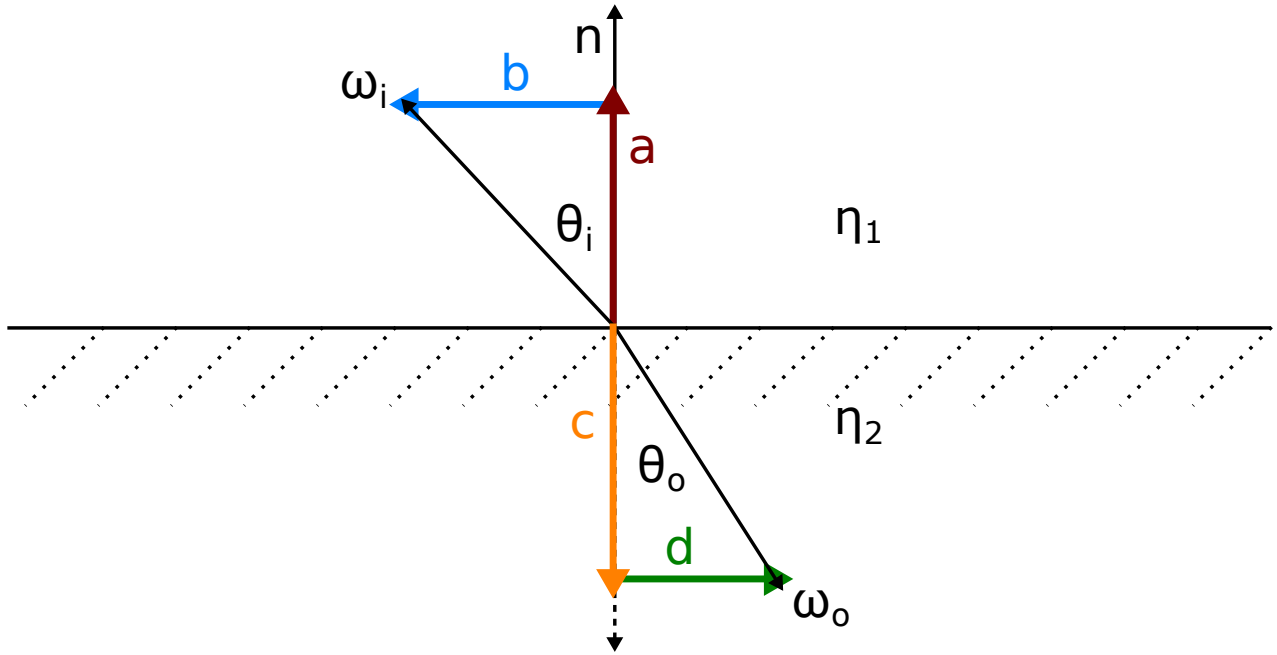
## Snell's Law




Verhältnis zwischen Sinus der Winkel zwischen eintretendem und in das Medium refraktiertem Licht ist reziprok proportional zum Verhältnis zwischen beiden Brechungsindizes:

$$\frac{sin(\theta_i)}{sin(\theta_o)} = \frac{\eta_2}{\eta_1}. \tag{32}$$

# Perfekt Spekulare Brechung

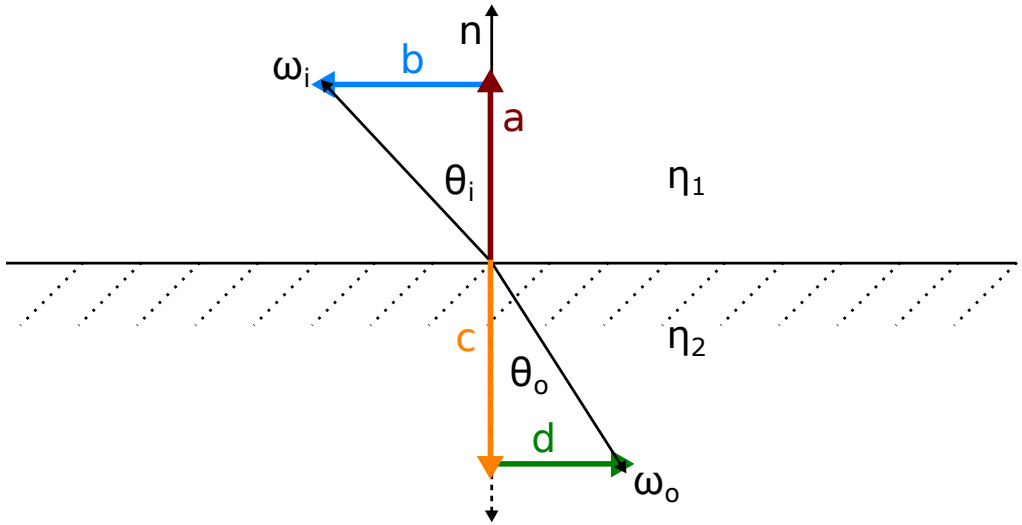



Wir suchen wieder $\vec{\omega_o}$.

$\vec{\omega_i} = \vec{a} + \vec{b}$

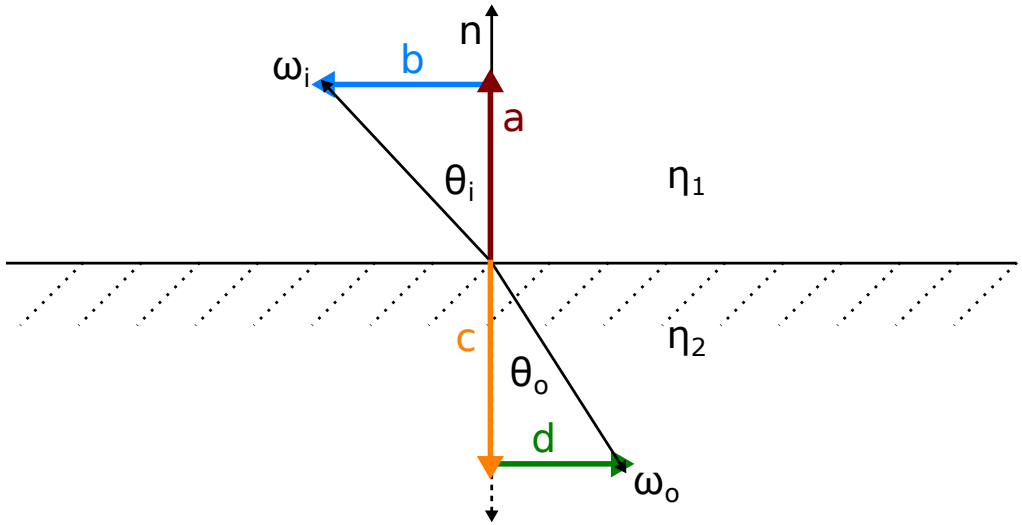
$\vec{\omega_o} = \vec{c} + \vec{d}$

# Perfekt Spekulare Brechung




Vereinfachende Annahme: $\eta_1 < \eta_2$. Wir betrachten außerdem (o. B. d. A.) nur den Fall $\vec{n} \cdot \vec{\omega_i} > 0$.

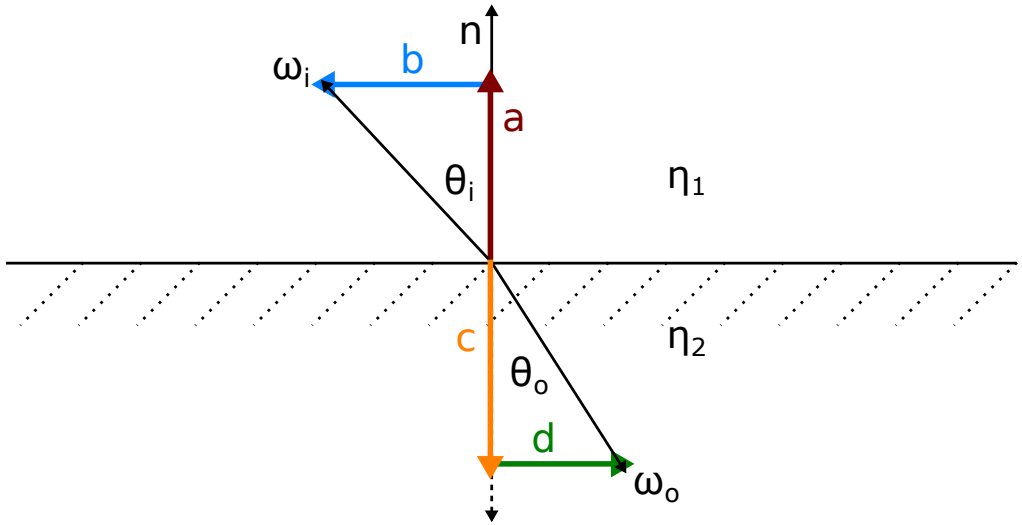
# Perfekt Spekulare Brechung




Aus Snell's Law folgt

$$|\vec{d}| = \frac{\eta_1}{\eta_2}|\vec{b}|. \tag{33}$$

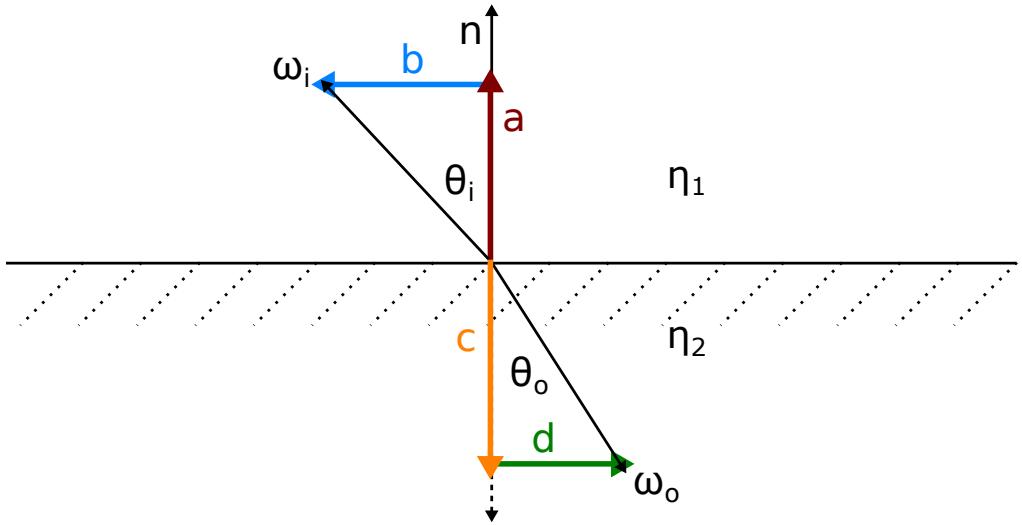
# Perfekt Spekulare Brechung




Da $\vec{b}$ und $\vec{d}$ parallel und umgekehrt orientiert, folgt

$$\vec{d} = -\frac{\eta_1}{\eta_2}\vec{b} = -\frac{\eta_1}{\eta_2}[\vec{\omega_i} - cos(\theta_i)\vec{n}]. \tag{34}$$

# Perfekt Spekulare Brechung

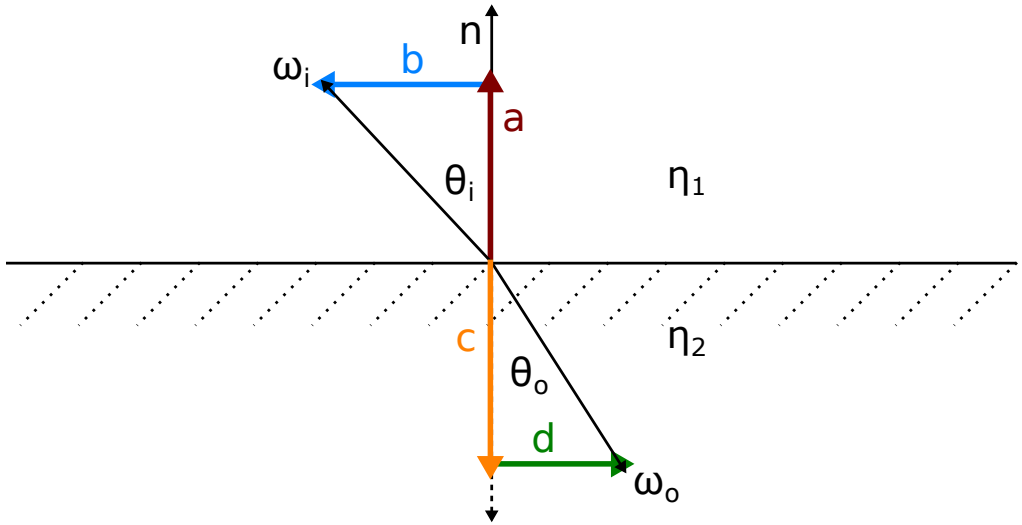



Über einfache geometrische Zusammenhänge (insb. $cos^2\theta + sin^2\theta = 1$) erhalten wir auch $\vec{c}$:

$$\vec{c} = -cos(\theta_o)\vec{n} = -\sqrt{1 - sin^2(\theta_o)}\vec{n}. \tag{35}$$

Wir erhalten mittels Snell's Law: $sin^2(\theta_o) = \left(\frac{\eta_1}{\eta_2}\right)^2(1 - cos^2(\theta_i))$ und damit alle Informationen, um $\vec{\omega_o}$ zu berechnen.

# Perfekt Spekulare Brechung




$$\vec{\omega_o} = \vec{c} + \vec{d} = -\sqrt{1 - \left(\frac{\eta_1}{\eta_2}\right)^2 (1 - cos^2(\theta_i))}\vec{n} - \frac{\eta_1}{\eta_2}[\vec{\omega_i} - cos(\theta_i)\vec{n}] \tag{36}$$

# Perfekt Spekulare Brechung

## Totale Interne Reflektion (TIR)

Nur möglich, wenn $\eta_1 > \eta_2$ (z. B. Licht von Wasser ($\eta_1 = 1.3$) nach Vakuum ($\eta_2 = 1$)). Tritt auf, wenn Term unter der Wurzel aus Gleichung 36 kleiner 0 (kritischer Winkel).

$$\vec{\omega_o} = \vec{c} + \vec{d} = -\sqrt{1 - \left(\frac{\eta_1}{\eta_2}\right)^2 (1 - cos^2(\theta_i))}\vec{n} - \frac{\eta_1}{\eta_2}[\vec{\omega_i} - cos(\theta_i)\vec{n}]$$

Vgl. Gleichung 36, jetzt jedoch $\eta_2 > \eta_1$. Im Fall von TIR wird alles Licht *reflektiert*.

# Perfekt Spekulare Brechung

## Totale Interne Reflektion (TIR)

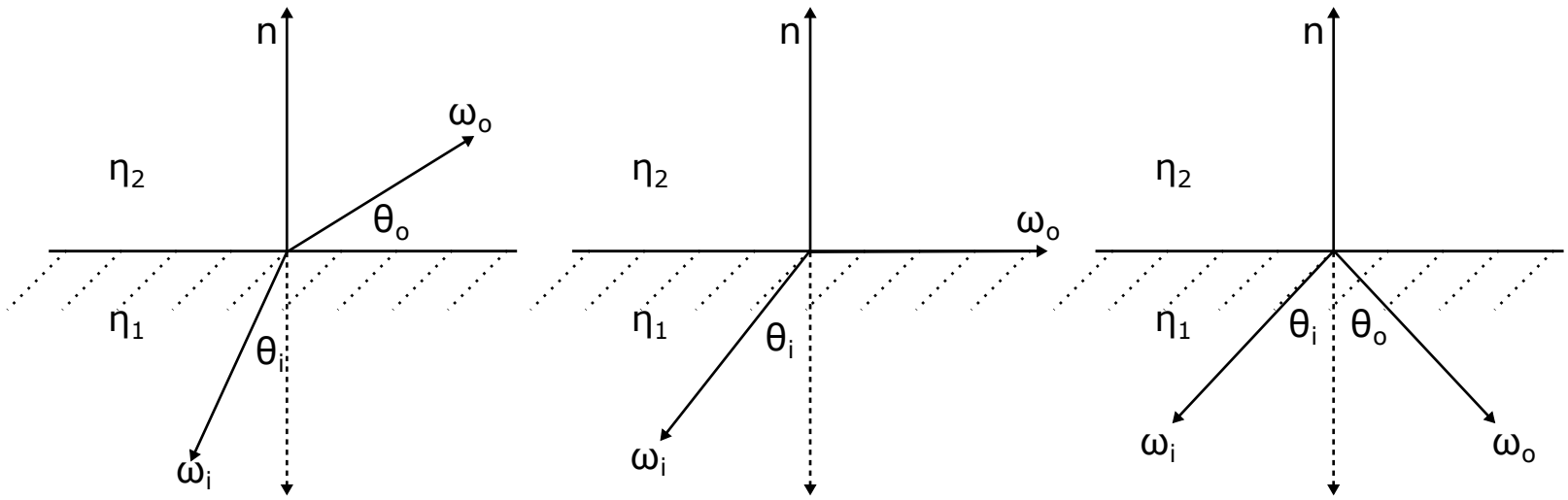
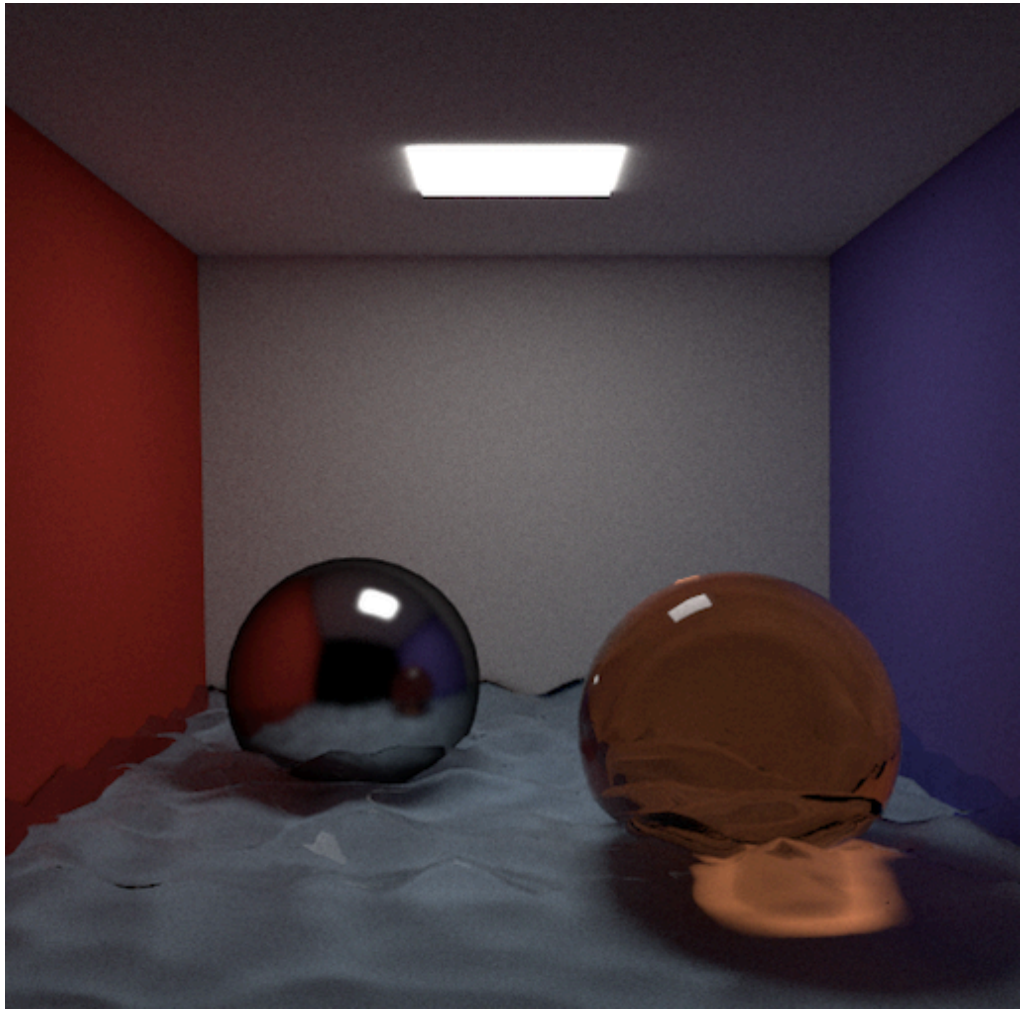



Abbildung: Links: Licht tritt aus Medium mit höherem Brechungsindex aus. Mitte: Kritischer Winkel. Rechts: Kritischer Winkel überschritten $\Rightarrow$ Totale interne Reflektion.

# Perfekt Spekulare Brechung

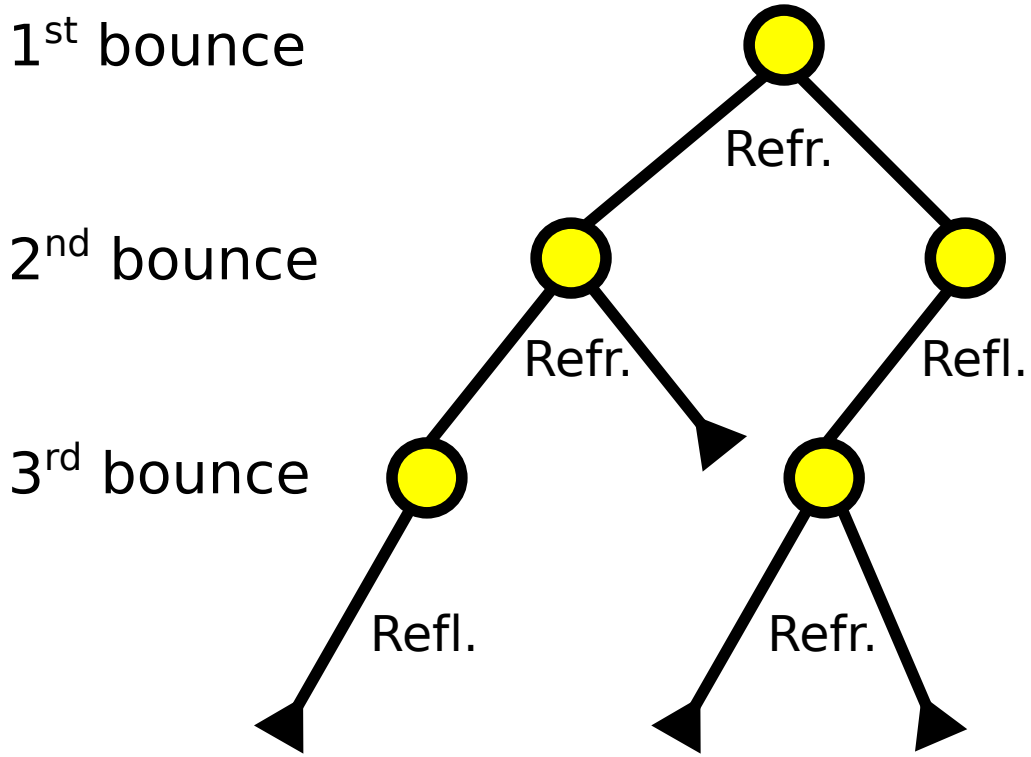


# Strahlverfolgung nach Whitted

```
function VERFOLGE(r)
    INTERSECT(r)
    SHADE(r)
    if SPIEGELT(m ∈ M) then
        r ← REFLEKTIERE(r)
        VERFOLGE(r)
    else if BRICHT(m ∈ M) then
        r ← REFLEKTIERE(r)
        VERFOLGE(r)
        r ← BRECHE(r)
        VERFOLGE(r)
    end if
end function
```

Vgl. Turner Whitted: An improved illumination model for shaded display (1980).

# Whitted Strahlbaum



# Whitted Algorithmus - Komplexität

Arbeits- und Zeitkomplexität des Whitted Algorithmus sind proportional zur maximalen Höhe des Strahlbaums:

Arbeitskomplexität:
$W(n) = (O(VP) \times (O(V) + O(L))) \times O(logh)$.

Zeitkomplexität:
$S(n) = (O(V) + O(L)) \times O(logh)$.

# Whitted Algorithmus - Komplexität

## Warum Worst-Case Betrachtung?

Wir haben im Laufe der Vorlesung Algorithmen vornehmlich bzgl. Ihres Worst-Case Laufzeitverhaltens analysiert. Beim Strahlverfolgunsalgorithmus nach Whitted wird klar, warum: wird nur ein Strahl aus $1M$ Strahlen sehr häufig reflektiert, kann das gesamte Bild nicht angezeigt werden.

I. Allg. Worst-Case Betrachtung wichtig, um z. B. konstante Frameraten zu gewährleisten.

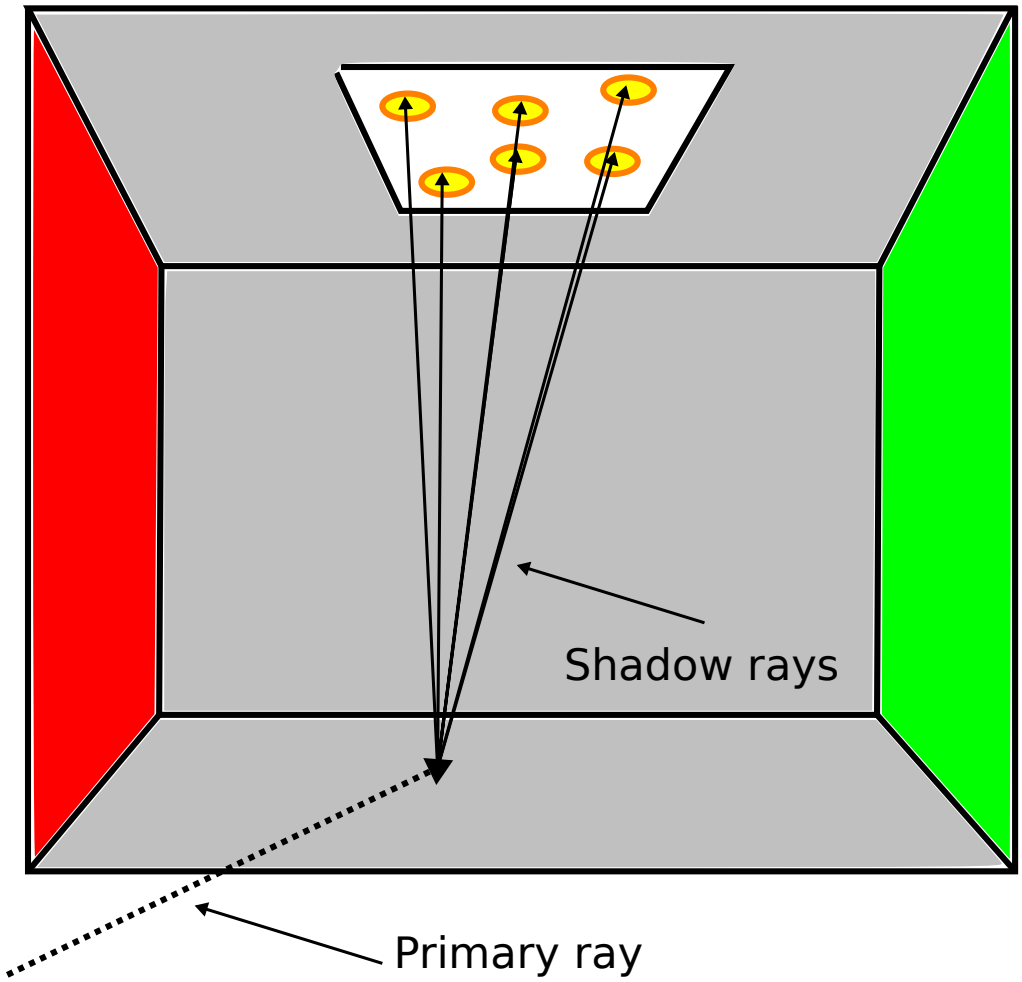
Pragmatische Lösungen bei Strahlverfolgung (vgl. “Hall of Mirrors”): maximale Anzahl *Bounces*; breche Äste im Strahlbaum vorzeitig ab, wenn Strahlstärke unter vordefiniertes $\epsilon$ fällt; etc.

# Whitted Algorithmus - Bemerkungen

- Beim Algorithmus PRIMAERSTRAHLVERFOLGUNG haben wir *nur lokale Beleuchtung* an den Schnittpunkten berechnet.
- Beim Algorithmus nach Whitted teilen wir das Problem auf:
  - Für *perfekt spekulare* Licht / Oberflächeninteraktion berücksichtigen wir echte *globale Beleuchtung*.
  - Für alle anderen Oberflächen (z. B. diffus, rau, anisotrop etc.): *lokale Beleuchtung*.

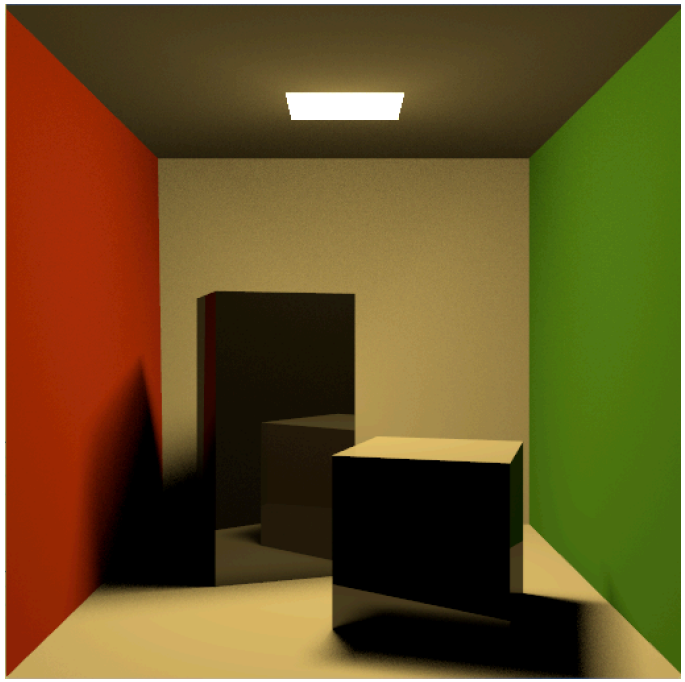
# Ausblick: Noch Mehr Realismus

Weiche Schatten: sample *Fläche* der Lichtquelle und mittle Sichtbarkeitsinformation.



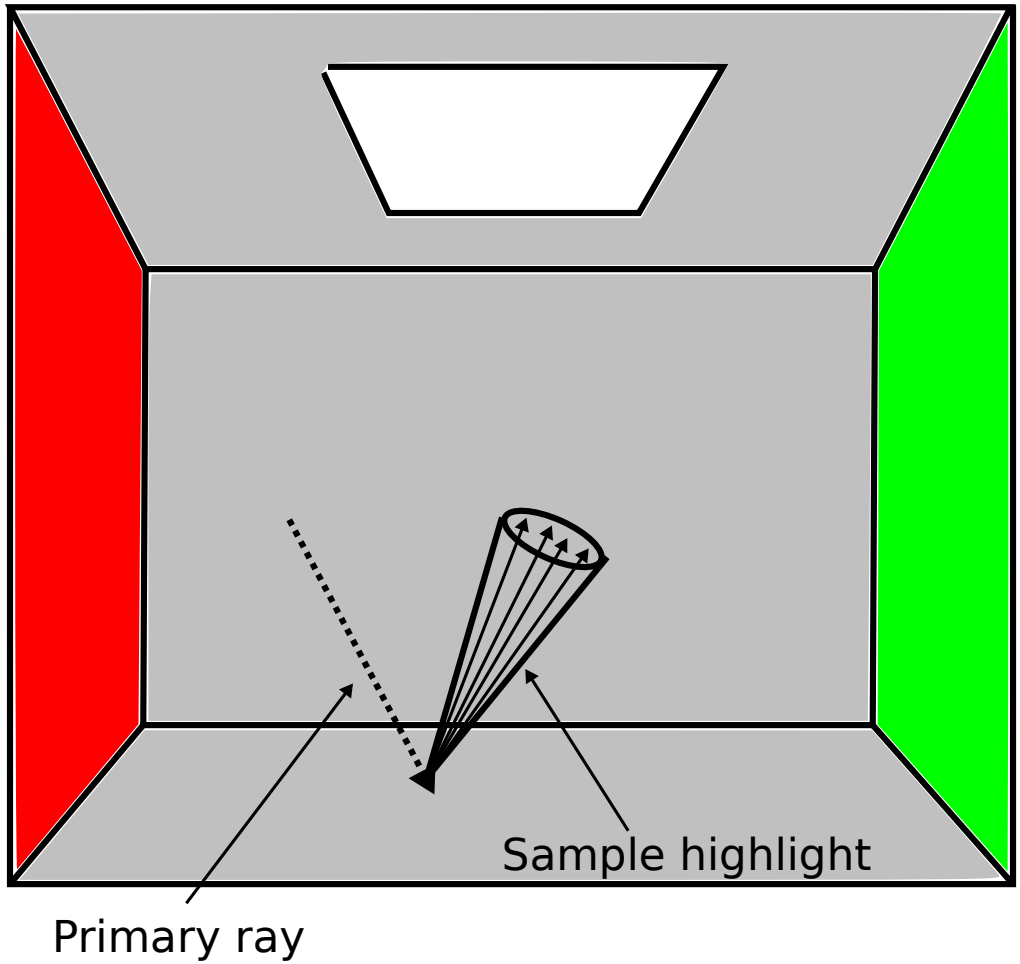
# Ausblick: Noch Mehr Realismus

Links: harte Punktlichtschatten, rechts: weiche Schatten



# Ausblick: Noch Mehr Realismus

Raue, spiegelnde Oberflächen: sample die *bidirectional reflectance distribution function* (BRDF) (z. B. Blinn Phong Cone).



# Ausblick: Noch Mehr Realismus

## Spiegelnde Oberfläche mit abnehmender Rauheit

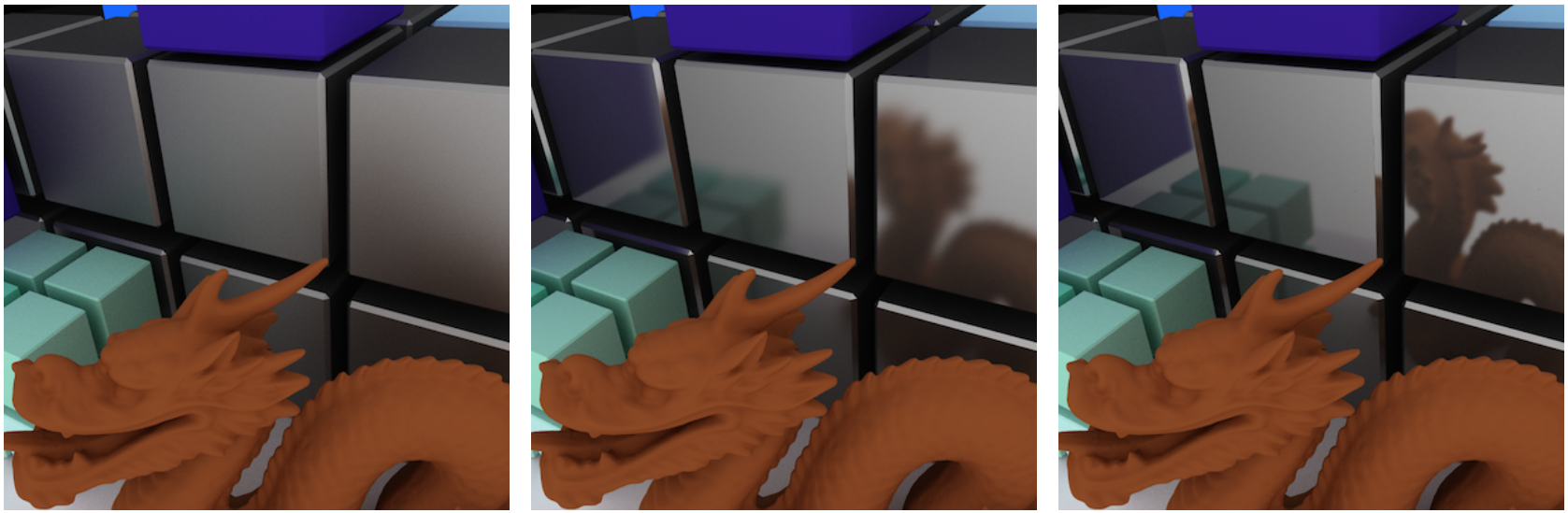
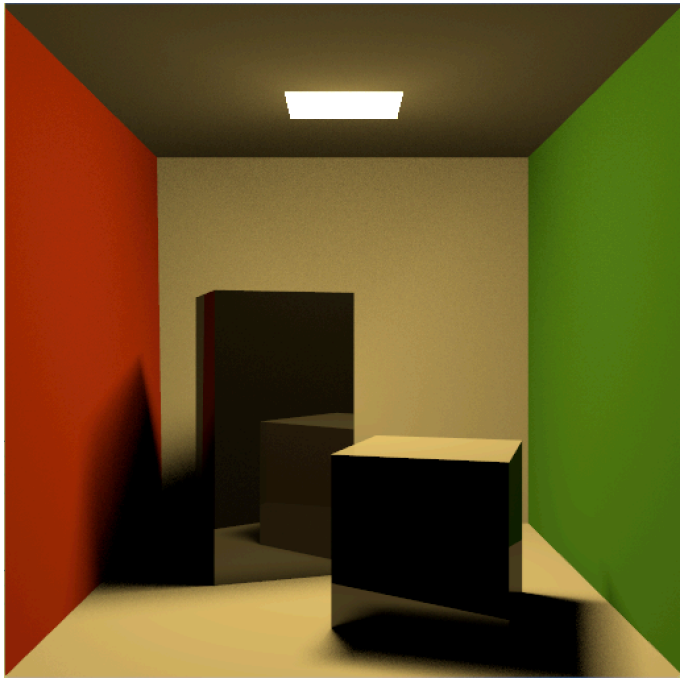
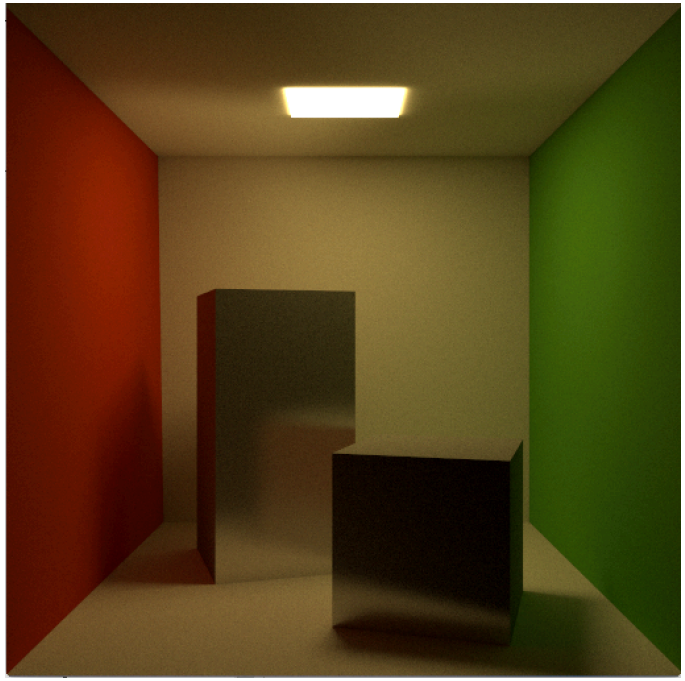


Abbildung: Modell: Stanford Dragon (Stanford 3D Scanning Repository), Rendering: Stefan Zellmann

# Ausblick: Path Tracing

## Links: Lokale Beleuchtung, rechts: Globale Beleuchtung





Weiterführend: *path tracing* Algorithmus, mit dem höchster Realismus durch Kombination verschiedenster Effekte erzielt werden kann.