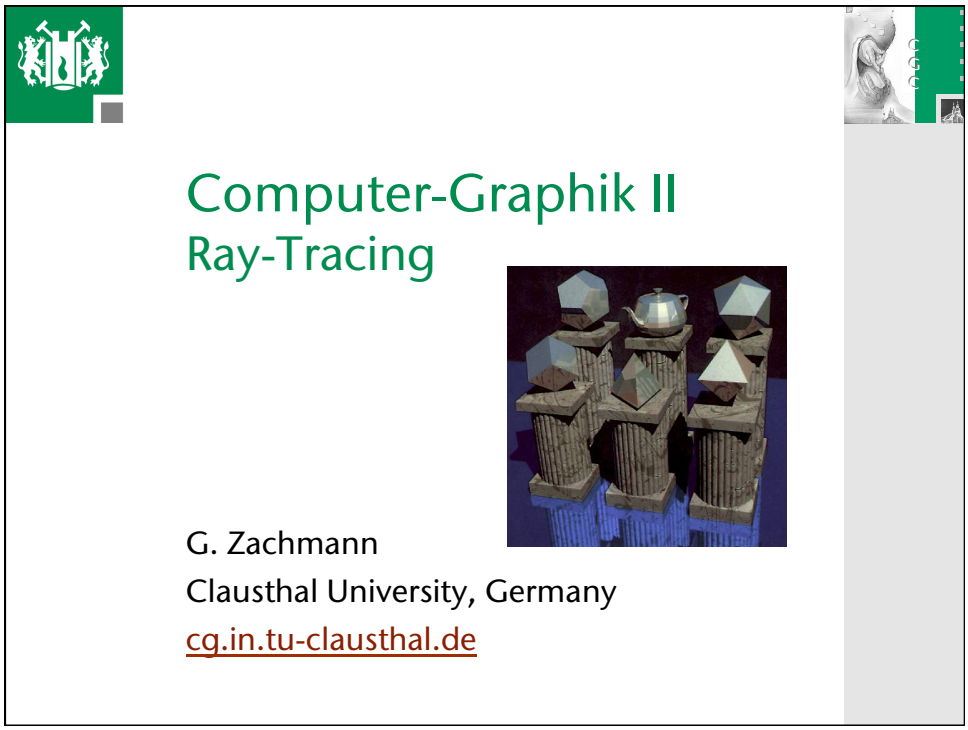
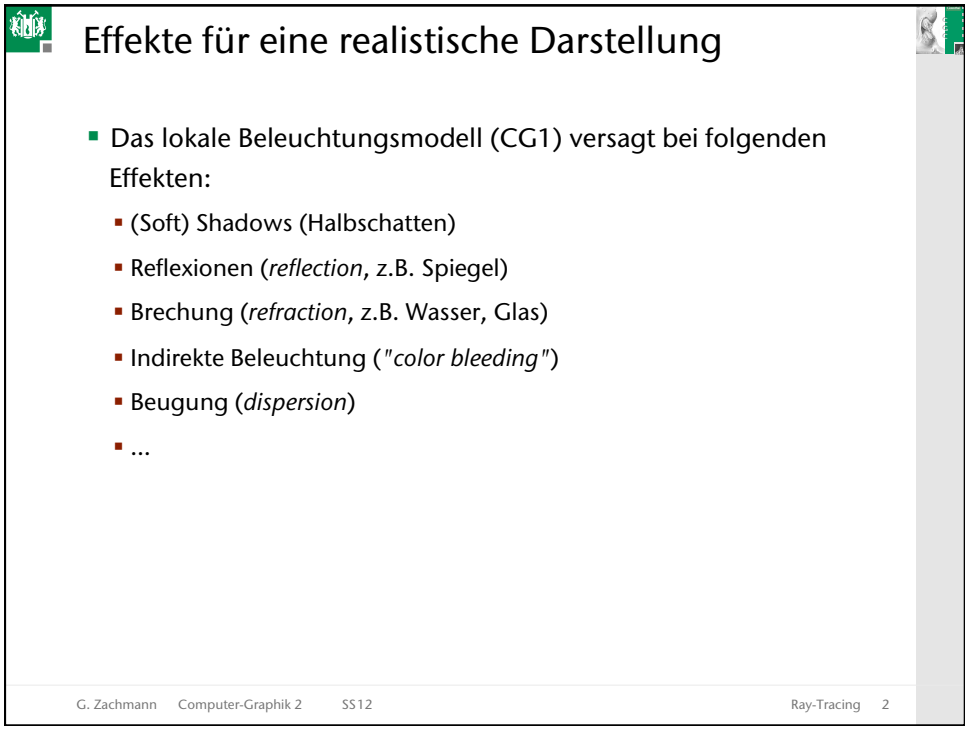


Computer-Graphik II
Ray-Tracing

G. Zachmann
Clausthal University, Germany
cg.in.tu-clausthal.de



Effekte für eine realistische Darstellung

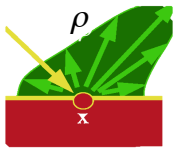
- Das lokale Beleuchtungsmodell (CG1) versagt bei folgenden Effekten:
  - (Soft) Shadows (Halbschatten)
  - Reflexionen (*reflection*, z.B. Spiegel)
  - Brechung (*refraction*, z.B. Wasser, Glas)
  - Indirekte Beleuchtung (*"color bleeding"*)
  - Beugung (*dispersion*)
  - ...

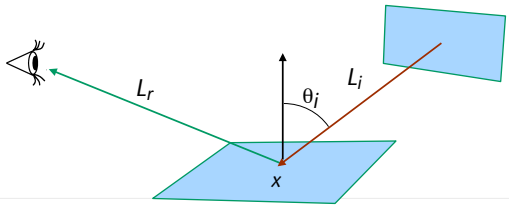
Globale Beleuchtungsrechnung

- Ziel: Photorealistisches Rendering
- Die "Lösung": die Rendering-Gleichung [Kajiya, Siggraph 1986]

$$L_r(x,\omega_r) = L_e(x,\omega_r) + \int_\Omega \rho(x,\omega_r,\omega_i) L_i(x,\omega_i) \cos(\theta_i) d\omega_i$$

L_i = aus Richtung ω_i inzidente "Lichtmenge"
L_e = emittierte Lichtmenge
L_r = in Richtung ω_r reflektiertes Licht
ρ = Reflexionskoeffizientenfunktion (BRDF)
Ω = Halbkugel um Normale



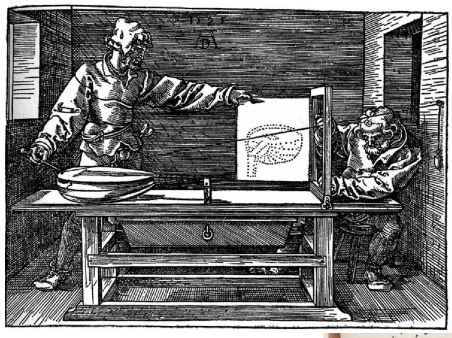


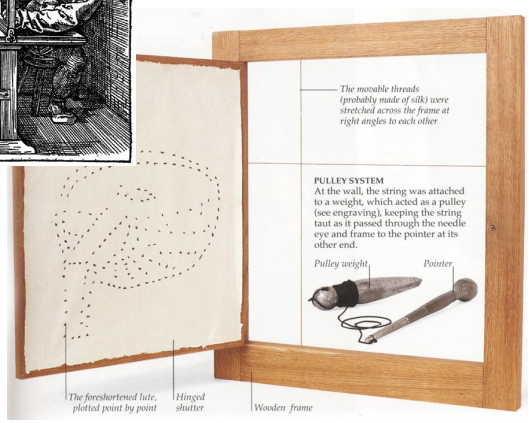
Die Erfindung des Ray-Tracings

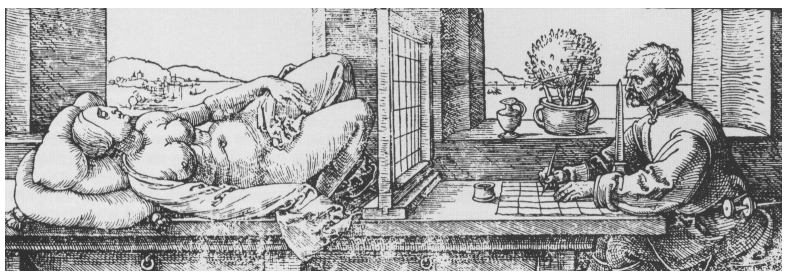
- Analytische Lösung ist unmöglich!
- Die Rendering Gleichung kann als rekursive Funktion aufgefaßt werden
- Daraus folgen praktische Approximations-Verfahren, die auf der Verfolgung des Lichts entlang Strahlen beruhen
  - Ray tracing [Whitted, Siggraph 1980, "An Improved Illumination Model for Shaded Display"]
  - Radiosity [Goral et. al, Siggraph 1984, "Modeling the Interaction of Light between diffuse Surface"]
  - Monte Carlo Verfahren

Turner Whitted, Microsoft Research

Albrecht Dürers "Ray-Casting-Maschinen" [16. Jhrdt.]







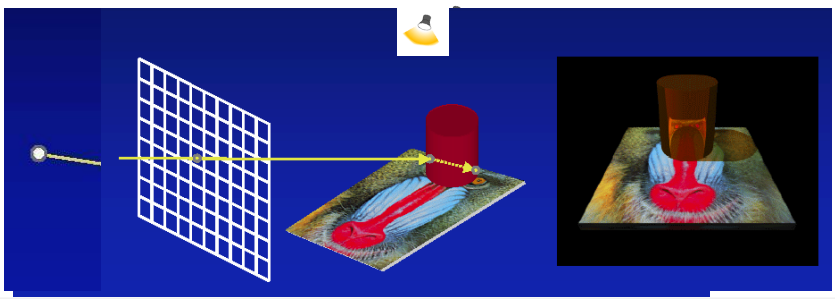
Das einfache Whitted-style Ray-Tracing

- Algorithmus zur Approximation der Rendering-Gleichung
- Modelliert werden nur:
  - Reflexion
  - Brechung
  - Verdeckungsrechnung
  - Schatten

- Strahlen werden nur in Richtung des reflektierten bzw. gebrochenen Strahls verfolgt
- Annahmen:
  - Punktlichtquellen
  - Phong-Modell
  - keine Halbschatten

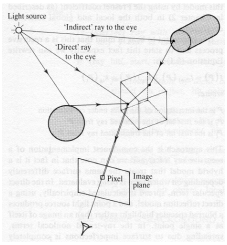
Funktionsweise

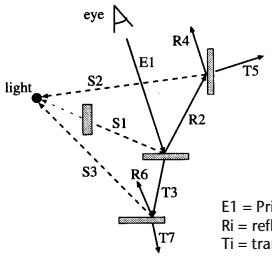
1. Synthetische Kamera = Augpunkt + Bildebene in Weltkoordinaten
2. Schieße Strahlen vom Augpunkt aus durch die Pixel in die Szene
3. Falls der Strahl mehr als ein Objekt schneidet, betrachte nur den ersten Schnittpunkt
4. Schieße weitere Strahlen von dort zu allen Lichtquellen (Schattenstrahlen; *"shadow feelers"*)
5. Treffen diese Schattenstrahlen auf ein Objekt, so liegt der betrachtete Flächenpunkt im Schatten. Ansonsten wird das Phong-Beleuchtungsmodell ausgewertet
6. Ist das sichtbare Objekt spiegelnd, dann schieße einen reflektierten Strahl in die Szene
7. Ist das Objekt transparent, so wird zusätzlich ein gebrochener Strahl weiterverfolgt

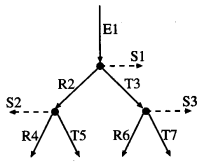


Der Strahlbaum

- Grundidee des Raytracing: Strahlengänge von den Lichtquellen bis zum Auge konstruieren, aber dabei beim Auge starten und diese Strahlengänge rückwärts "suchen"
- Ergibt (konzeptionell!) einen Strahlenbaum:



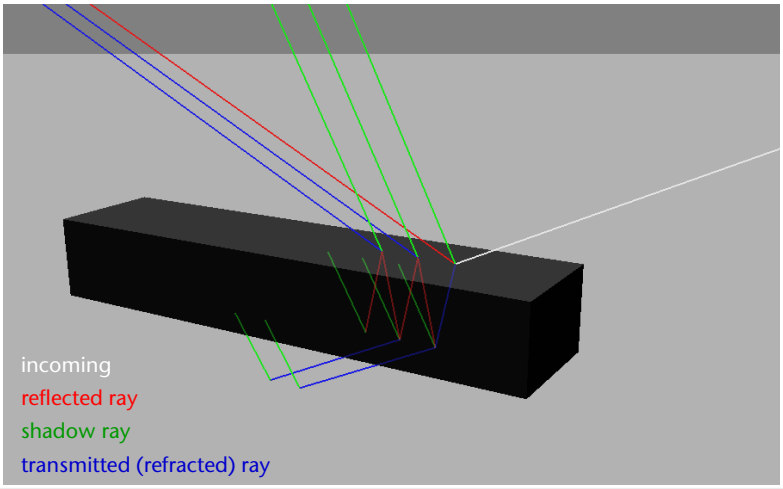




E1 = Primärstrahl
Si = Schattenstrahl
Ri = reflektierter Strahl
Ti = transmittierter Strahl

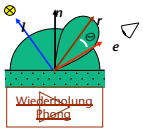
Exkurs: die antike Erklärung des Sehens: Sehstrahlen

- Visualisierung eines Strahlbaumes (eignet sich hervorragend zum Debugging)



Das Beleuchtungsmodell

- Beleuchtung auf der Fläche

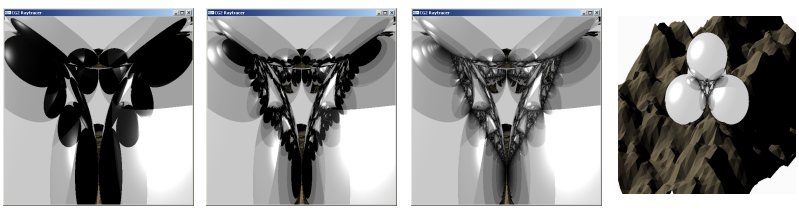


$$L_{\text{ges}} = L_{\text{Phong}} + r_s L_s + r_t L_t$$

r_s = Reflexionskoeffizient für das reflektierte Licht L_s

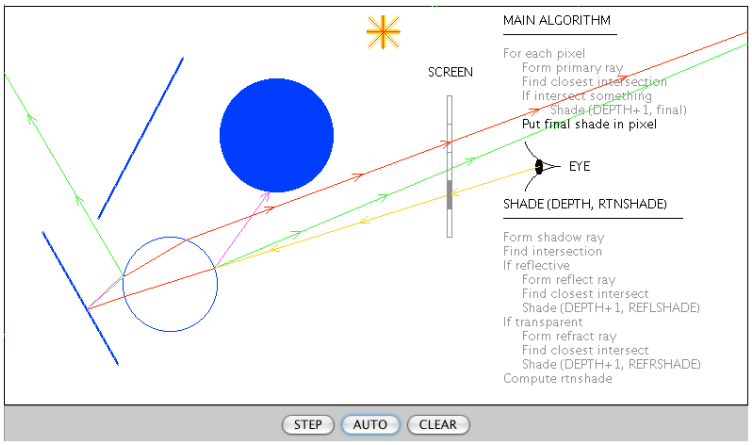
r_t = Transmissionskoeffizient für das transmittierte Licht L_t

- Abbruch der Rekursion:
  - Falls maximale Rekursionstiefe erreicht; oder/und,
  - falls Beitrag zur Beleuchtung zu klein (schrumpft wie r^n)



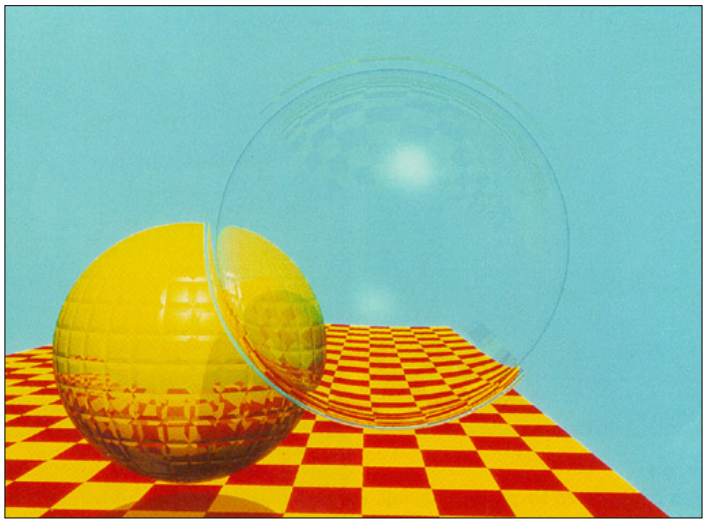
Rek. Tiefe: 3 Rek. Tiefe: 5 Rek. Tiefe: 100

Demo



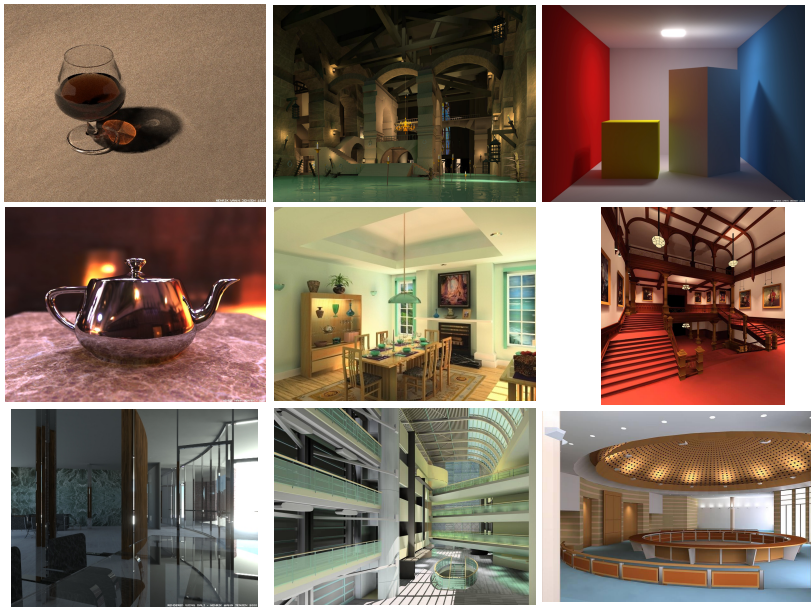
http://www.siggraph.org/education/materials/HyperGraph/raytrace/rt_java/raytrace.html

Eines der ersten Ray-Tracing-Bilder

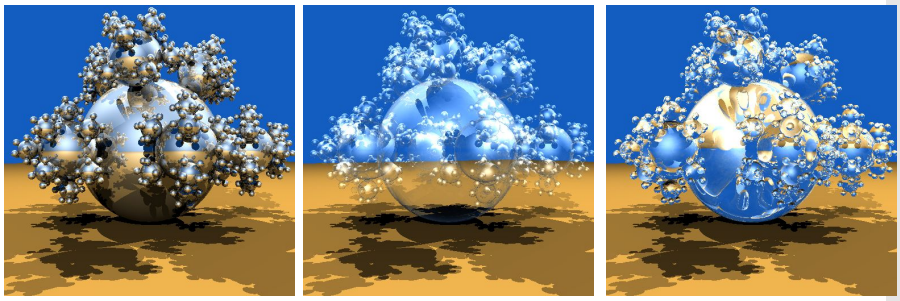


Turner Whitted 1980

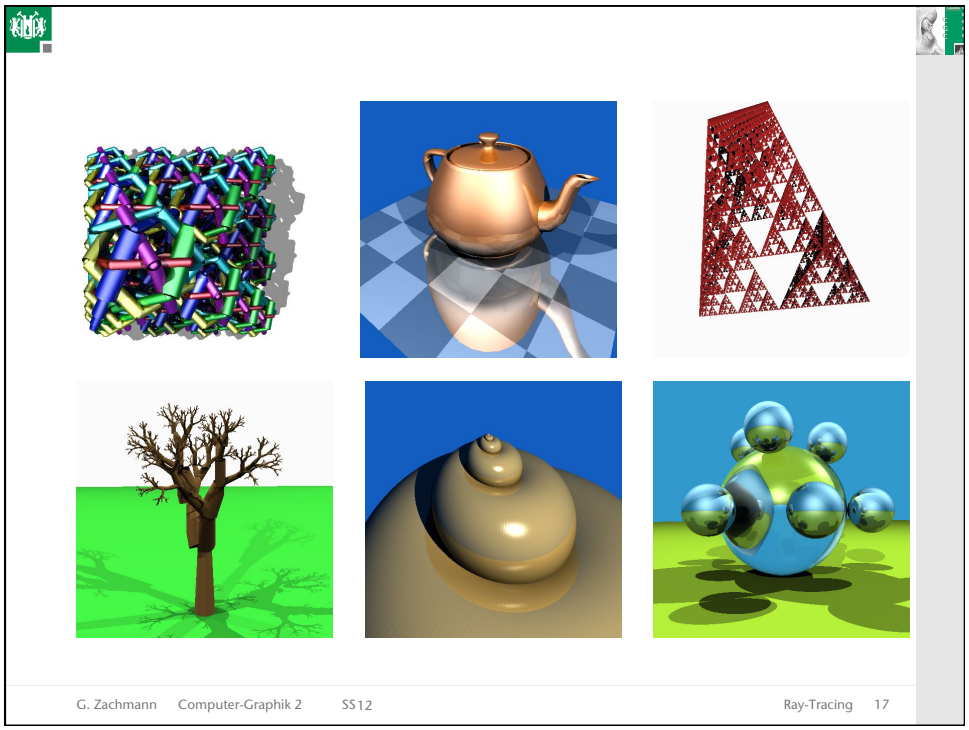
Beispiele

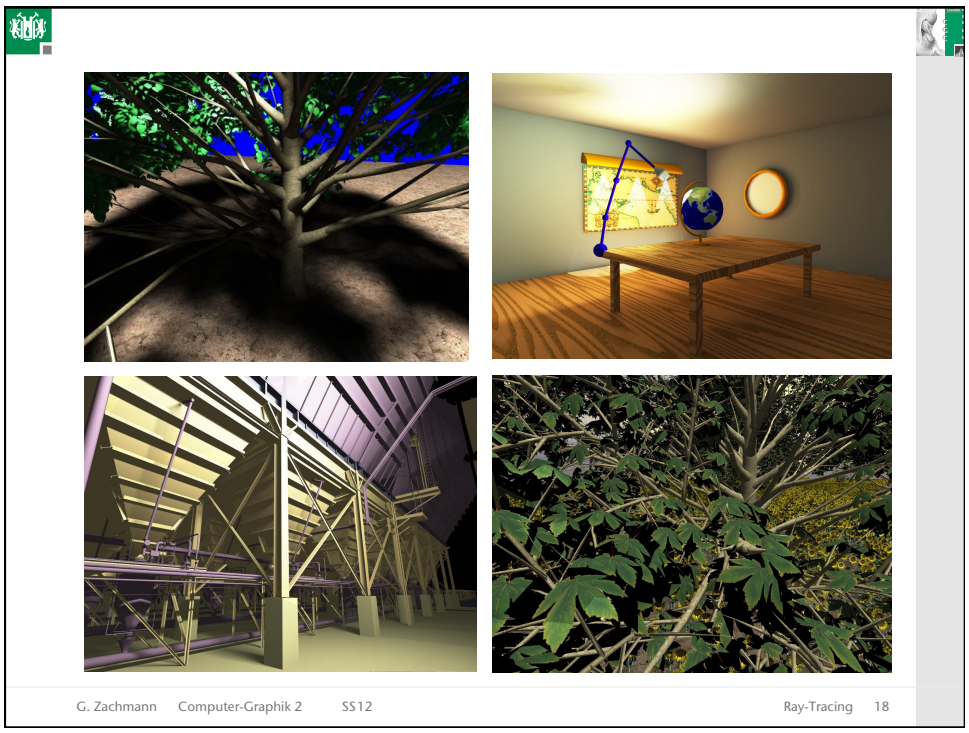


Jensen, Lightscape

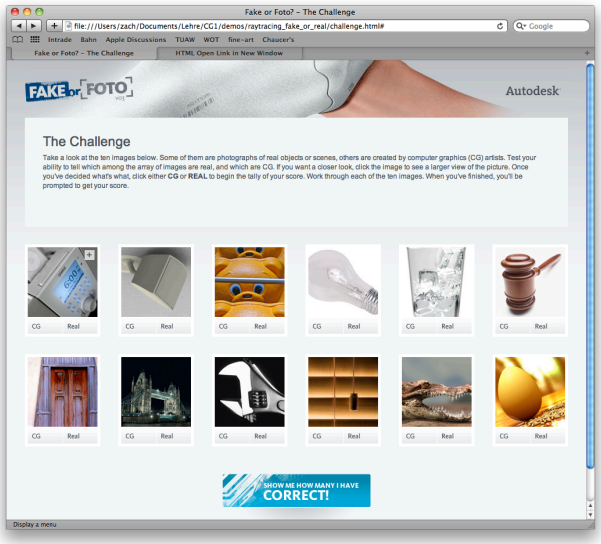


Objekt "sphere flake" aus der "Standard Procedural Databases" (SPD) von Eric Haines [http://www.acm.org/tog/resources/SPD/].

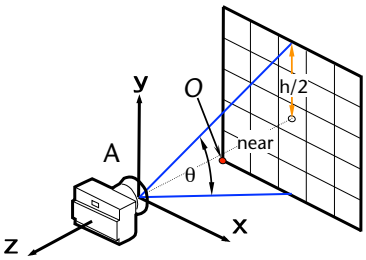




Fake or Real?



Die Kamera (ideale Lochkamera)



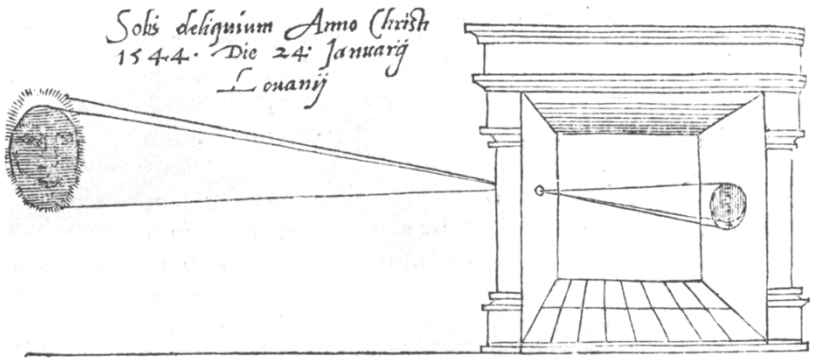
$$\frac{h}{2} = \text{near} \cdot \tan\frac{\theta}{2}$$

$$O = A - \text{near} \cdot \mathbf{z} - \frac{w}{2}\mathbf{x} - \frac{h}{2}\mathbf{y}$$

Die Main-Loop eines Ray-Tracers

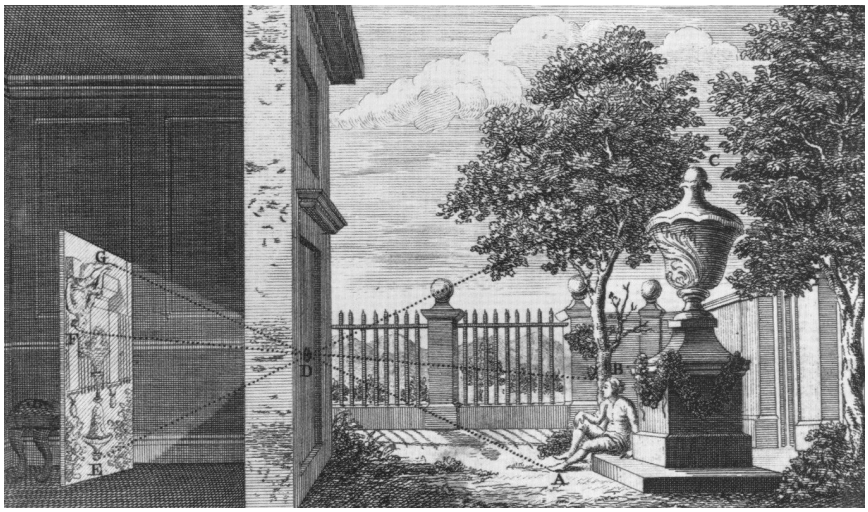
```
for ( i = 0; i < height; i ++ )
  for ( j = 0; j < width; j ++ )
    ray.from = A
    t = (i/height - 0.5) * h
    s = (j/width - 0.5)  * w
    ray.at = O + s·x + t·y
    trace( 0, ray, &color );
    putPixel( x, y, color );
```

Älteste Abbildung einer Lochkamera



Von R. Gemma Frisius, 1545

Camera Obscura

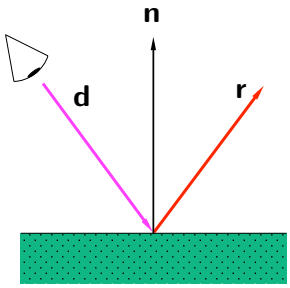


Andere sonderbare Kameras

- Mit Ray-Tracing sind nicht-standard Projektionen sehr einfach
- Z.B. Fischauge, Omnimax, Panorama

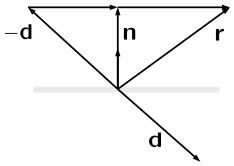
Sekundärstrahlen

- Annahme: Hit zwischen Primärstrahl und Szene gefunden
- Reflektierter Strahl:



$$\begin{aligned}\mathbf{r} &= \big((-\mathbf{d}\cdot\mathbf{n})\cdot\mathbf{n} - (-\mathbf{d})\big)\cdot 2 + (-\mathbf{d}) \\ &= \mathbf{d} - 2(\mathbf{d}\cdot\mathbf{n})\cdot\mathbf{n}\end{aligned}$$

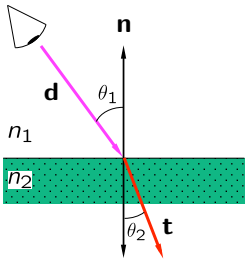
wobei $\|\mathbf{n}\| = 1$



Gebrochener Strahl

- Brechungsgesetz [Snell ~1600] :

$$n_1 \sin\theta_1 = n_2 \sin\theta_2$$



- Der transmittierte Strahl:

$$\mathbf{t} = \frac{n_1}{n_2}(\mathbf{d} + \mathbf{n}\cos\theta_1) - \mathbf{n}\cos\theta_2$$

$$\cos\theta_1 = -\mathbf{dn}$$

$$\cos^2\theta_2 = 1 - \frac{n_1^2}{n_2^2}\left(1 - (\mathbf{dn})^2\right)$$

- Brechungsindizes:

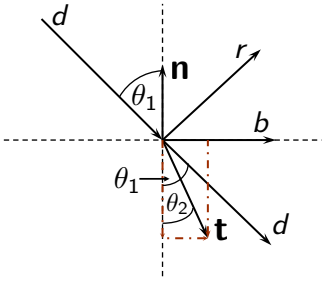
Luft	Wasser	Glas	Diamant
1.0	1.33	1.5 - 1.7	2.4

$$|\mathbf{n}| = |\mathbf{b}| = 1$$

$$\mathbf{t} = \cos\theta_2 \cdot (-\mathbf{n}) + \sin\theta_2 \cdot \mathbf{b}$$
$$\mathbf{d} = \cos\theta_1 \cdot (-\mathbf{n}) + \sin\theta_1 \cdot \mathbf{b}$$

$$\mathbf{b} = \frac{\mathbf{d} + \mathbf{n} \cdot \cos\theta_1}{\sin\theta_1}$$

$$\mathbf{t} = -\mathbf{n} \cdot \cos\theta_2 + \frac{\sin\theta_2}{\sin\theta_1}(\mathbf{d} + \mathbf{n} \cdot \cos\theta_1)$$



$$\frac{\sin\theta_2}{\sin\theta_1} = \frac{n_1}{n_2}$$
$$\cos\theta_1 = \mathbf{n} \cdot (-\mathbf{d})$$

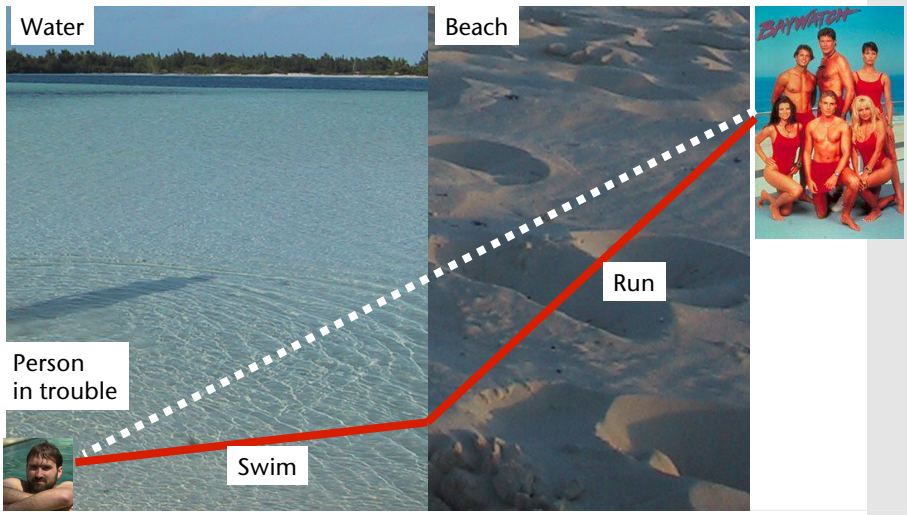
$\cos\theta_2$ ausrechnen:

$$\sin\theta_2 = \frac{n_1}{n_2}\sin\theta_1$$
$$\sin^2 + \cos^2 = 1$$
$$\cos^2\theta_2 = 1 - (\frac{u_1}{u_2}\sin\theta_1)^2$$

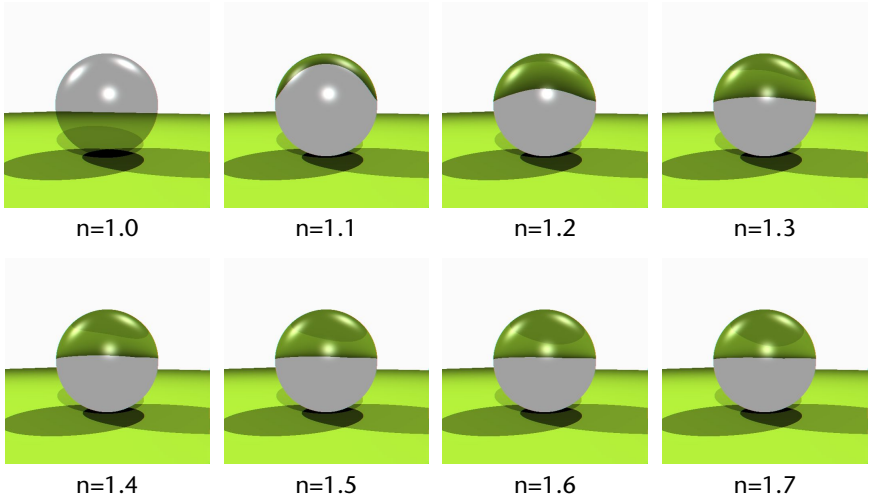
- Totalreflexion:
  wenn Radikand < 0 $\Leftrightarrow$ $\cos^2\theta_1 \leq 1 - \frac{n_2^2}{n_1^2}$

Refraction and the Lifeguard Problem

- Running is faster than swimming



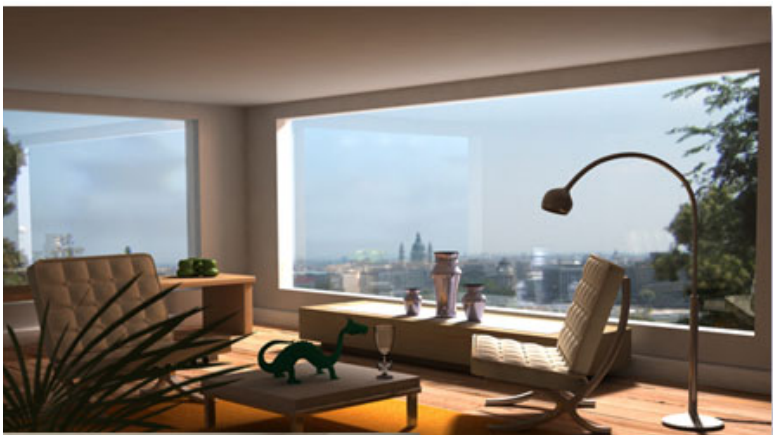
Wirkung des Brechungsindex



Welches ist die "richtige" Normale?

- Klappt die korrekte Berechnung des reflektierten und des gebrochenen Strahls auch, wenn die Normale in die "falsche" Richtung zeigt?

Was können wir hier noch nicht (richtig) simulieren?



Fresnel-Terme

- Beim Wechsel von einer Materie in eine andere wird immer ein Anteil Licht reflektiert, der restliche Anteil gebrochen
- Der Reflexionskoeffizient ρ hängt ab vom Brechungsindex der beiden Materialien und vom Einfallswinkel:

$$\rho_{\parallel} = \frac{n_2 \cos\theta_1 - n_1 \cos\theta_2}{n_2 \cos\theta_1 + n_1 \cos\theta_2}$$

$$\rho_{\perp} = \frac{n_1 \cos\theta_1 - n_2 \cos\theta_2}{n_2 \cos\theta_1 + n_1 \cos\theta_2}$$

$$\rho = \frac{1}{2} \cdot \left(\rho_{\parallel}^2 + \rho_{\perp}^2 \right)$$

- Beispiel:
  - Luft (n = 1.0) nach Glas (n = 1.5), senkrechter Lichteinfall:

$$\rho_{\parallel} = \frac{1.5-1}{1.5+1} = \frac{1}{5} \quad \rho_{\perp} = \frac{1-1.5}{1.5+1} = \frac{1}{5} \quad \rho = \frac{1}{2} \cdot \frac{2}{25} = 4\%$$

  - D.h., beim Übergang von Luft nach Glas wird 4% des Lichtes reflektiert, der Rest gebrochen
- Approximation der Fresnel-Terme [Schlick 1994]:

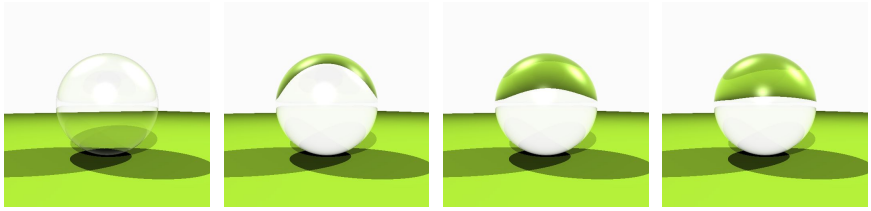
$$\rho(\theta) \approx \rho_0 + (1-\rho_0)(1-\cos\theta)^5$$

$$\rho_0 = \left(\frac{n_2 - 1}{n_2 + 1} \right)^2$$

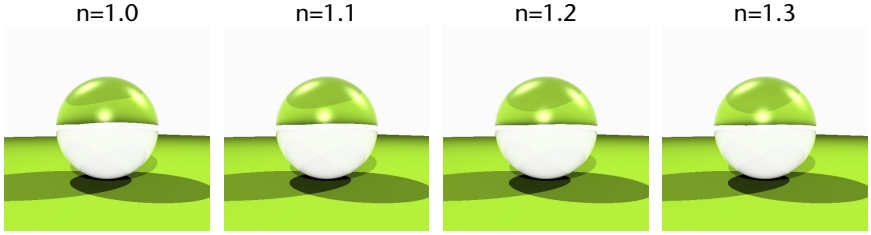
wobei ρ_0 der Fresnel-Term des senkrechten Lichteinfalls ist und θ der Winkel im dünneren Medium (also der größere).

  - 1-ρ ergibt dann den transmittierten Anteil

Beispiel für Brechung unter Berücksichtigung der Fresnel-Terme



n=1.0 n=1.1 n=1.2 n=1.3



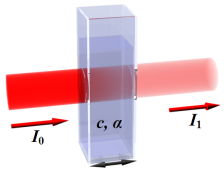
n=1.4 n=1.5 n=1.6 n=1.7

Dämpfung im Medium (*participating media*)

- Die durch ein Medium transportierte Lichtintensität schwächt sich mit zunehmender Entfernung gemäß dem Lambert-Beer'schen Gesetz ab:

$$I(s) = I_0 e^{-\alpha s}$$

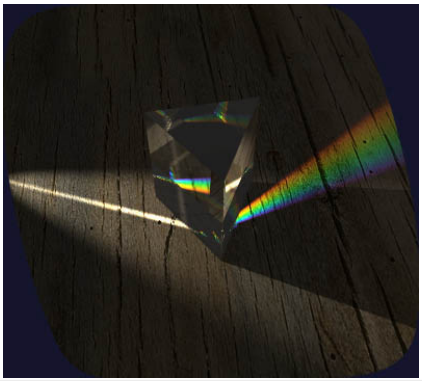
wobei α eine Materialkonstante ist
und s der im Medium zurückgelegte Weg.

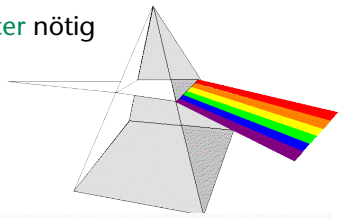


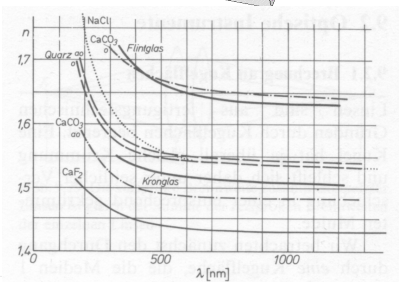
  - α kann auch von der Wellenlänge abhängen

Dispersion

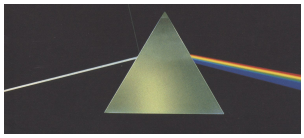
- Brechungsindex ist abhängig von der Wellenlänge
- Diese Effekte lassen sich allerdings in RGB nicht mehr abbilden; hierzu wäre ein „spektraler“ Ray-Tracer nötig







Giovanni Battista Pittoni, 1725,
"An Allegorical Monument to Sir Isaac Newton"



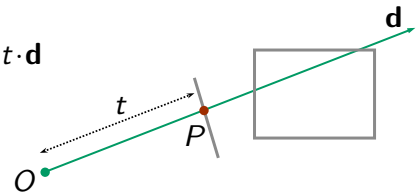
Pink Floyd, *The Dark Side of the Moon*

Beispiel mit Fresnel-Term und Dispersion

Schnittberechnungen

- Der wesentliche Bestandteil der Rechenzeit
- Gegeben: Menge von Objekten (Polygone, Kugeln, ...) und ein Strahl

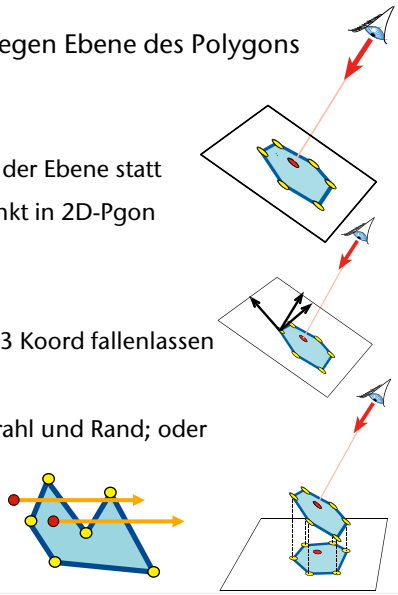
$$P(t) = O + t \cdot \mathbf{d}$$



- Gesucht: Linienparameter t des ersten Schnittpunktes $P = P(t)$ mit der Szene

Schnitt Strahl—Polygon

- Schneide Strahl (parametrisch) gegen Ebene des Polygons (implizit) → Punkt
- Teste "Punkt in Polygon"
  - Dieser Test findet ausschließlich in der Ebene statt
  - 3D-Punkt in 3D-Polygon ⇔ 2D-Punkt in 2D-Pgon
- Projiziere Punkt & Polygon
  - Entlang der Normale: zu teuer
  - Auf Koord.ebene: einfach eine der 3 Koord fallenlassen
- Test "Punkt in Polygon":
  - Zähle Anzahl Schnitte zwischen Strahl und Rand; oder
  - Bestimme "Winding Number"



Interludium: die vollständige Ray-Tracing-Routine

```
traceRay( ray ):
  hit = intersect( ray )
  if no hit:
    return no color
  reflected_ray = reflect( ray, hit )
  reflected_color = traceRay( reflected_ray )
  refracted_ray = refract( ray, hit )
  refracted_color = traceRay( refracted_ray )
  for each lightsource[i]:
    shadow_ray = compShadowRay( hit, lightsource[i] )
    if intersect(shadow_ray):
      light_color[i] = 0
  overall_color = shade( hit,
                         reflected_color,
                         refracted_color,
                         light_color )
  return overall_color
```

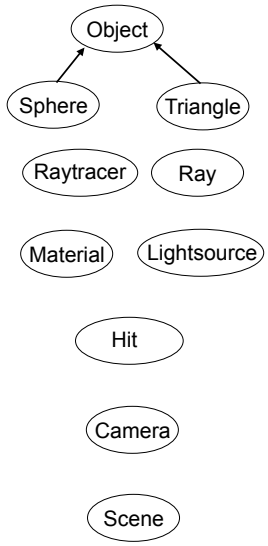
hit ist eine Datenstruktur, die alle Infos über einen Schnitt zwischen Strahl und Szene enthält, u.a. Schnittpunkt, Objekt, Normale, ...

Diese **intersect**-Funktion kann deutlich optimiert werden gegenüber der obigen; außerdem interessiert nur ein Schnitt **vor** der Lichtquelle.

Wertet die Beleuchtungsgleichung für das getroffene Obj aus.

Software-Architektur eines einfachen Raytracers

- Szene: einfache Liste von Kugeln, Dreiecken, etc.
- Schnittpunkt Strahl—Objekt
- Diffuse und spekulare Reflexion (Phong-Modell)
- Sekundärstrahlen
- Mögliche Erweiterungen
  - Lichtbrechung, Transparenzen
  - Besseres Oberflächenmodell (Fresnel)
  - Andere Objekte (Kegel, Zylinder, Polygon, ...)
  - Szene einladen



Typische Raytracer-Klassen

- Lightsource (hier nur Einfache Punktlichtquelle)

```
Vector m_location;      // Position
Vector m_color;         // Farbe
```

- Material

```
Vector m_color;         // Farbe der Oberfläche
float m_diffuse;        // Diffuser / Spekularer
float m_specular;       // Reflexionskoeff. [0..1]
float m_phong;          // Phong-Exponent
```

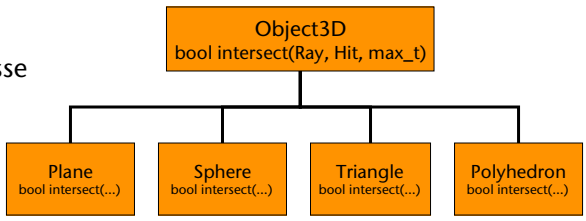
- Ray

```
Vector m_origin;        // Aufpunkt des Strahls
Vector m_direction;     // Strahlrichtung
```

- Hit: Speichert Informationen über den Schnittpunkt

```
Ray m_ray;          // Strahl
float m_t;          // Geradenparameter t
Object* m_object;   // Geschnittenes Objekt
Vector m_location;  // Schnittpunkt
Vector m_normal;    // Normale am Schnittpunkt
```


- Object:
  Abstrakte Basisklasse für alle Geometrie-objekte

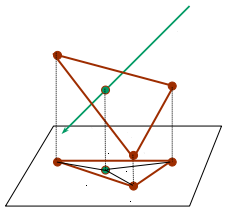



```
// Schnittpunkt von Strahl mit Objekt
virtual bool closestIntersection( Intersection * hit ) = 0;
virtual bool anyIntersection( const Ray & ray, float max_t,
                              Intersection * hit ) = 0;

// Normale am Schnittpunkt
virtual void calcNormal( Intersection * hit ) = 0;

// Material des Objekts
int getMaterialIndex() const;
```

- Camera:
  - Alle Eigenschaften der Kamera, z.B. from, at, up, angle
  - Generiert Primärstrahlen durch alle Pixel
- Scene:
  - Speichert alle Daten der Szene
    - Liste aller Objekte
    - Liste aller Materialien
    - Liste aller Lichtquellen
    - Kamera

Schnitt Strahl—Dreieck [Badouel 1990]

- Verwende Methode Strahl—Polygon; oder
- Cleverer sein: baryzentrische Koordinaten + Projektion
- Schneide Strahl mit Ebene (Normalenform) → t → Punkt
- Projiziere Punkt & Dreieck in Koord.ebene
- Berechne baryzentrische Koord. des 2D-Punktes
- Baryzentrische Koord. des 2D-Punktes (α,β,γ) = baryzentrische Koord. des 3D-Punktes!
- 3D-Punkt in Dreieck $\leftrightarrow$ $\alpha,\beta,\gamma > 0$, $\alpha+\beta+\gamma = 1$
- Alternative Methode: siehe Möller & Haines "Real-time Rendering"
- Code: http://jgt.akpeters.com/papers/MollerTrumbore97/
- Geht noch schneller, falls Schnittpunkt nicht nötig [Segura & Feito]

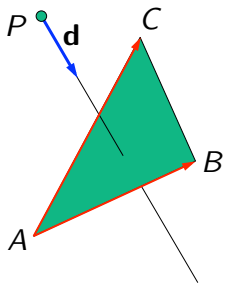


Alternative Schnittberechnung Strahl—Dreieck [Möller]

- Geradengleichung $X = P + t \cdot \mathbf{d}$
- Ebenengleichung $X = A + r \cdot (B - A) + s \cdot (C - A)$
- Gleichsetzen $-t \cdot \mathbf{d} + r \cdot (B - A) + s \cdot (C - A) = P - A$

- In Matrixschreibweise:

$$\begin{pmatrix} \vdots & \vdots & \vdots \\ -\mathbf{d} & \mathbf{u} & \mathbf{v} \\ \vdots & \vdots & \vdots \end{pmatrix} \begin{pmatrix} t \\ r \\ s \end{pmatrix} = \mathbf{w}$$



wobei

$$\mathbf{u} = B - A$$
$$\mathbf{v} = C - A$$
$$\mathbf{w} = P - A$$

$$\begin{pmatrix} t \\ r \\ s \end{pmatrix} = \frac{1}{\det(-\mathbf{d}, \mathbf{u}, \mathbf{v})} \cdot \begin{pmatrix} \det(\mathbf{w}, \mathbf{u}, \mathbf{v}) \\ \det(-\mathbf{d}, \mathbf{w}, \mathbf{v}) \\ \det(-\mathbf{d}, \mathbf{u}, \mathbf{w}) \end{pmatrix}$$

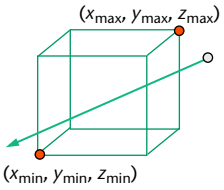
$$\det(\mathbf{a}, \mathbf{b}, \mathbf{c}) = \mathbf{a} \cdot (\mathbf{b} \times \mathbf{c}) = (\mathbf{a} \times \mathbf{b}) \cdot \mathbf{c}$$

$$\begin{pmatrix} t \\ r \\ s \end{pmatrix} = \frac{1}{(\mathbf{d} \times \mathbf{v}) \cdot \mathbf{u}} \cdot \begin{pmatrix} (\mathbf{w} \times \mathbf{u}) \cdot \mathbf{v} \\ (\mathbf{d} \times \mathbf{v}) \cdot \mathbf{w} \\ (\mathbf{w} \times \mathbf{u}) \cdot \mathbf{d} \end{pmatrix}$$

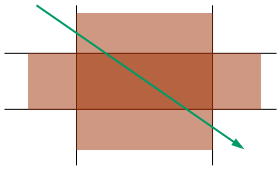
- Kosten: 2 Kreuzprodukte + 4 Skalarprodukte
- Liefert: Geradenparameter + baryzentrische Koordinaten bzgl. Dreieck
- Test ob s,t im Bereich (0,1) und s+t <= 1

Schnitt Strahl — Box

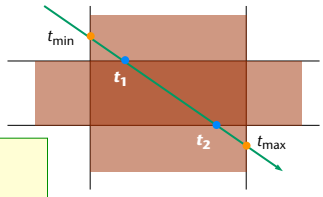
- Box (Quader) wird später noch wichtig als Bounding Box
- Hier: nur achsenparallele Boxes (AABB = *axis-aligned bounding box*)
- Definition einer AABB: durch die zwei extremen Eckpunkte
  $(x_{min}, y_{min}, z_{min})$ und $(x_{max}, y_{max}, z_{max})$



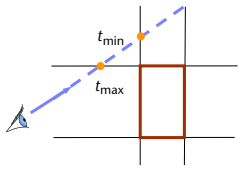
- Idee des Algo:
  - Eine Box ist der Schnitt von 3 *Slabs* (ein *Slab* = Schicht des Raumes, wird von 2 parallelen Ebenen begrenzt)
  - Jeder Slab schneidet vom Strahl ein Intervall heraus
  - Betrachte also sukzessive jeweils Paare von Box-Seiten

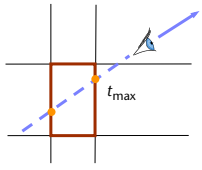


- Der Algorithmus:



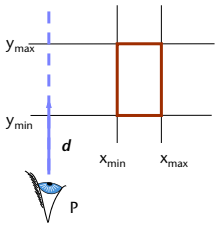

```
setze t_min = -∞ , t_max = +∞
loop über alle Paare von Ebenen:
  schneide Strahl mit den
           beiden Ebenen → t1 , t2
  if t2 < t1:
    vertausche t1 , t2
  // jetzt gilt: t1 < t2
  t_min ← max(t_min, t1 )
  t_max ← min(t_max, t2 )
// now: [t_min,t_max] = interval inside box
if t_min > t_max → kein Schnitt
if t_max < 0    → kein Schnitt
```





Bemerkungen

- Optimierung: beide Ebenen eines Slabs haben dieselbe Normale
  - Spart ein Skalarprodukt
- Bemerkung: der Algo funktioniert genauso für "schiefe" Boxes (sog. *OBBs = oriented bounding boxes*)
- Weitere Optimierung: falls AABB, nutze aus, daß die Normalen nur 1 Komponente ≠ 0 haben!
- Achtung: "shit happens"
  - Hier: teste auf Parallelität!
  - Im Fall der AABB:

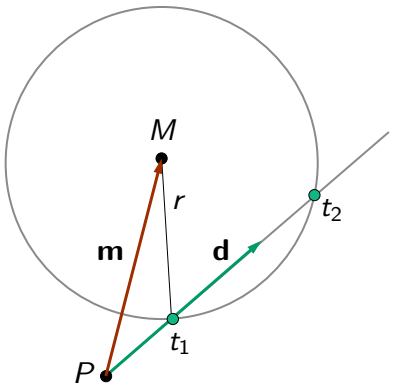



```
if |d_x| < ε:
  if P_x < x_min || P_x > x_max:
    Strahl geht an Box vorbei
  else:
    t_1, t_2 = y_min, y_max     // evtl noch swappen!
```


Schnitt Strahl—Kugel

- Annahme: **d** ist normiert
- Die geometrische Methode:

$$|t\cdot\mathbf{d}-\mathbf{m}| = r$$
$$(t\cdot\mathbf{d}-\mathbf{m})^2 = r^2$$
$$t^2 - 2t\cdot\mathbf{md} + \mathbf{m}^2 - r^2 = 0$$

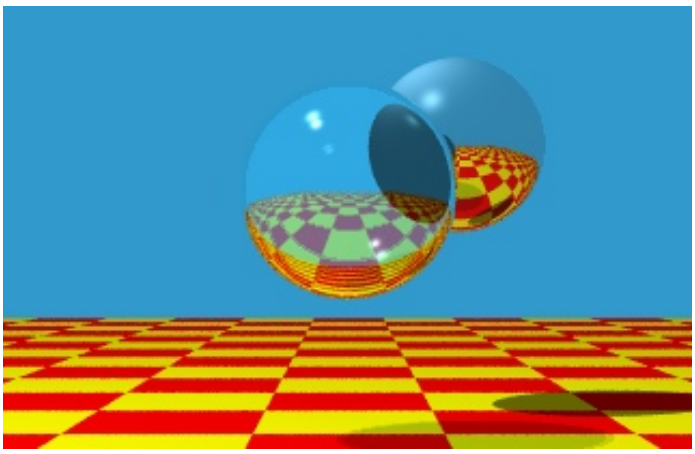


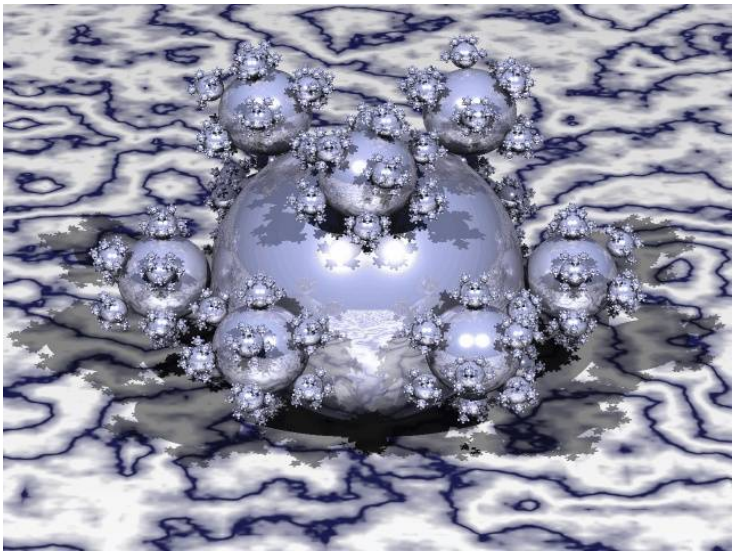
- Die algebraische Methode:
  Punkt auf Strahl in implizite Kugelgleichung einsetzen
- Es gibt noch andere Ansätze ...

- Der Algorithmus, mit kleinen Optimierungen:

```
berechne m²−r²
berechne b = m·d
if  m²−r² >= 0        // Blickpunkt ausserhalb Kugel
    and b <= 0 :      // sieht von Kugel weg
then
     return "kein Schnittpunkt"
setze d = b² − m² + r²
if d < 0:
    return "kein Schnittpunkt"
if  m²−r² > ε :
    return t1 = b − √d  // enter; t1 is > 0
else:
    return t2 = b + √d  // leave; t2 is > 0   (t1<0)
```


- Es ist so einfach, daß alle Ray-Tracer Kugeln als Primitiv haben!





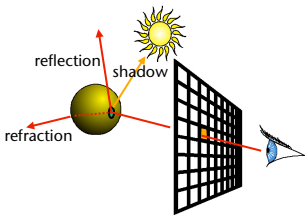
Die sog. "sphere flake"

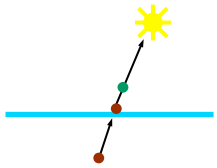
Geometrisch vs. Algebraisch

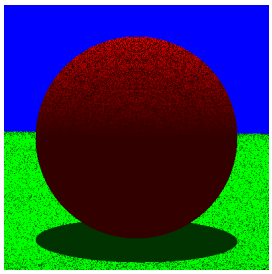
- Die algebraische Methode ist einfach und generisch
- Die geometrische Methode ist schneller
  - Durch geometrische Einsicht
  - Frühe Tests
  - Insbesondere für die Strahlen, die weg zeigen

The evil ε

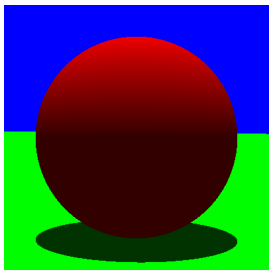
- Was passiert, wenn der Ursprung des Strahles "exakt" **auf** der Oberfläche eines Objektes sitzt?
- Floating-Point ist immer unexakt!
  - Folge: bei den folgenden Strahltests erscheint dieser Ursprung evtl. innerhalb des Objektes!
  - Folge: als nächsten Schnittpunkt erhalten wir denselben Punkt wieder!
- "Lösung":
  verschiebe Aufpunkt des Strahls immer zuerst um ein ε in Richtung des (neuen) Strahls



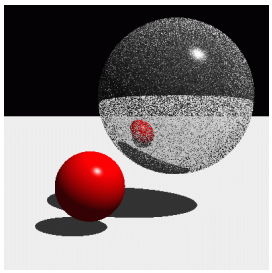


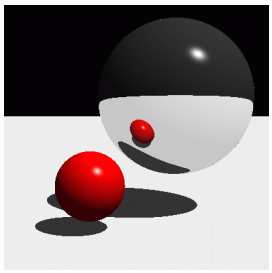


Without epsilon



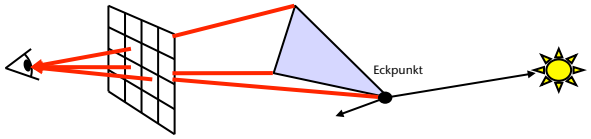
With epsilon



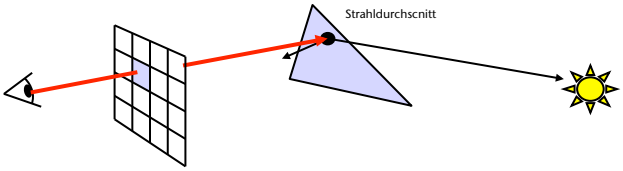


Scankonvertierung vs. Raytracing

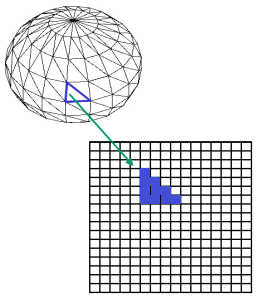
- Scan-Konvertierung: Auswerten eines Strahls, der durch jeden Eckpunkt eines Objektes gesendet wird



- Raytracing: Auswerten eines Strahls, der durch einen Bildschirmpixel gesendet wird

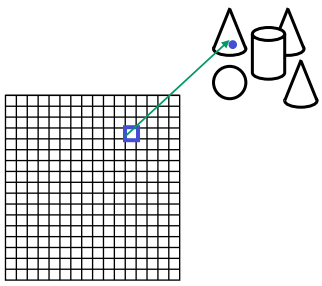


- Zum Umwandeln einer Szene mittels Scan-Konvertierung ...



... scan-konvertiere jedes Dreieck

- Zum Umwandeln einer Szene mittels Raytracing ...



... verfolge für jedes Pixel einen Strahl

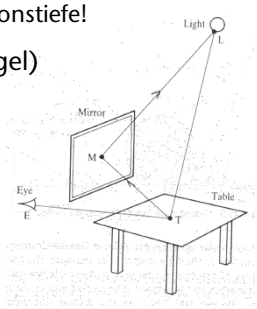
Vor- und Nachteile

- Scan-Konvertierung:
  - schnell (da nur Eckpunkte)
  - wird unterstützt von aktuellen Grafikkarten
  - geeignet für Echtzeitanwendungen
  - ad-hoc Lösung für Schatten, Transparenz
  - Keine Interreflexion
- Raytracing:
  - noch rel. langsam (Suche nach Schnittpunkten zwischen Strahlen und Objektprimitiven)
  - bisher von keiner kommerziellen Hardware unterstützt
  - Offline-Rendering-Verfahren
  - Allgemeine Lösung für Schatten, Transparenz und Interreflexion, Clipping und Culling

- Weitere Vorteile des Raytracings:
  - Eignet sich besonders für Szenen mit hohem spiegelndem und transparentem Flächenanteil
  - Kann beliebige Objektrepräsentationen verarbeiten (z.B. CSG, Rauch, ...)
    - Einzige Anforderung: man muß Schnitt zwischen Strahl und Objekt und die Normale in diesem Schnittpunkt berechnen können
  - Keine explizite perspektivische Transformation oder Clipping nötig

Nachteile des (einfachen) Raytracings

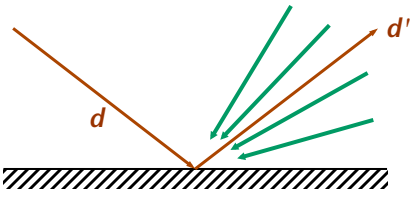
- Sehr viele Strahlen
  - Naives Ray-Casting (nur Primärstrahlen): $O(p \cdot (n+l))$, p = # Pixel, n = # Polygone, l = # Lichtquellen
  - Anzahl Strahlen wächst exponentiell mit Rekursionstiefe!
- Keine indirekte Beleuchtung (z.B. durch Spiegel)
- Keine weichen Halbschatten
- Shading muß bei jeder Änderung der Kamera neu berechnet werden, obwohl dieses nur von den Lichtquellen und den Objekten abhängen

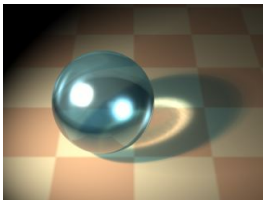


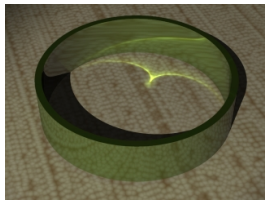
- Für alle diese Nachteile wurden natürlich viele verschiedene Abhilfen vorgeschlagen ...

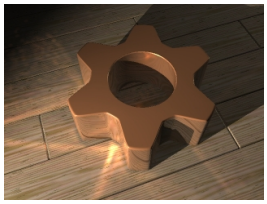
Beispiel für das Problem der indirekten Beleuchtung: Kaustiken

- Konzentration von Licht
- Lichtstrahlen treffen sich in einem Punkt
- Das normale Raytracing betrachtet nur 1 reflektierten Strahl







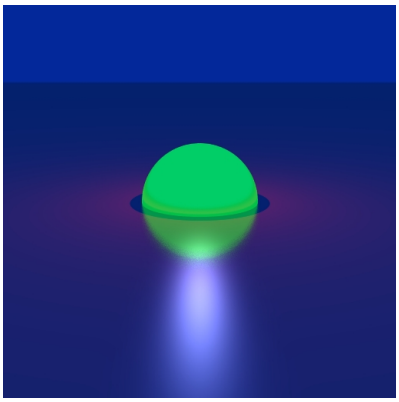


Aliasing

- Ein Strahl pro Pixel → typ. Aliasing-Artefakte:
  - Treppeneffekte
  - Moiré- Effekt

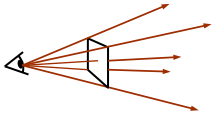
Distribution Ray Tracing

- Einfache Modifikationen des rekursiven Ray Tracings für
  - Antialising
  - Weiche Schatten
  - Tiefenschärfe
  - Spekulare Reflexion
- Bewegungsunschärfe
- Anderer Name früher:
  - „Distributed Ray Tracing“
  - ist aber sehr unglücklich ("distributed" = verteilt)

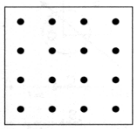


Anti-Aliasing beim Ray-Tracing

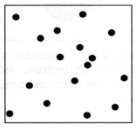
- Anstatt pro Pixel nur einen Strahl zu verfolgen werden mehrere Strahlen verfolgt und die resultierende Farbe gemittelt



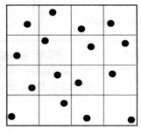
- Methoden zur Auswahl der Punkte:
  - Regelmäßige Abtastung (Problem der Moire Muster)



  - Zufällige Abtastung (Problem des Rauschens)

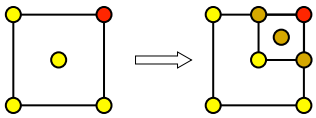


  - Stratifikation, d.h. eine Kombination von regelmäßiger und zufälliger Abtastung, indem ein reglmäßiges Gitter zufällig gestört wird.

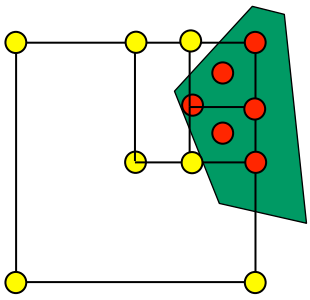


Adaptives Supersampling

- Idee: verschieße Lichtstrahlen nur bei großen Farbunterschieden

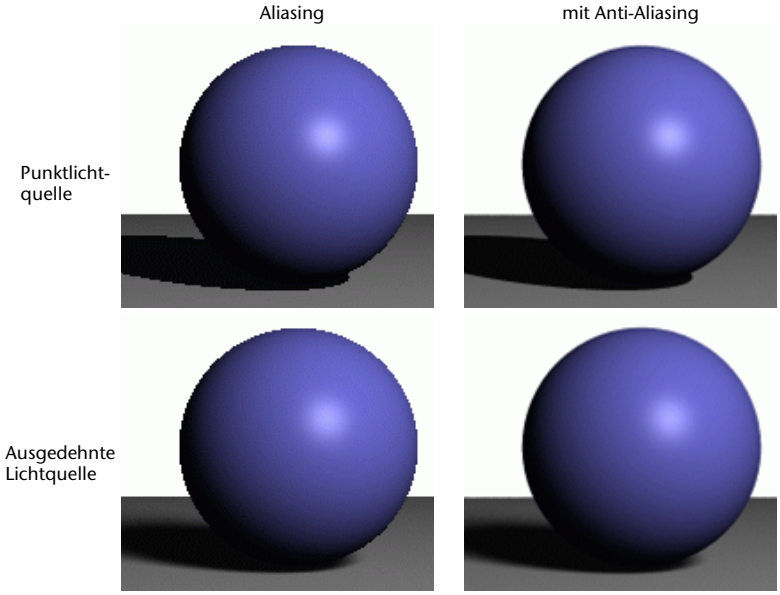


- Beispiel:



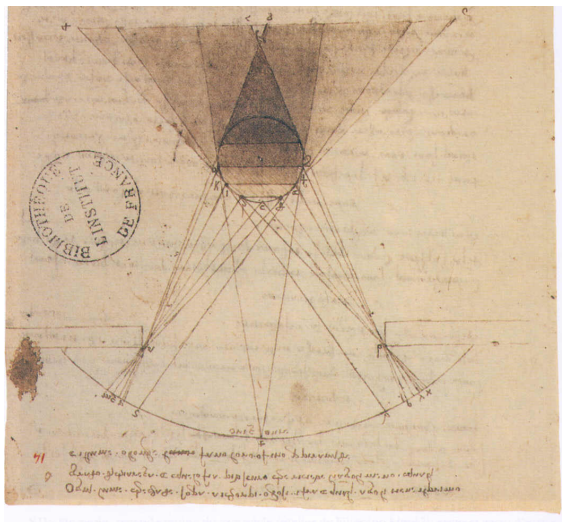
- Resultierende Farbe = Durchschnittsfarbe aller Samples, gewichtet mit dem Flächenanteil des Pixels, den das Sample "überdeckt"

Beispiel



Weiche Schatten, Halbschatten

- Regionen:
  - "Vollschatten" (*umbra*)
  - Halbschatten (*penumbra*)
  - voll beleuchtet

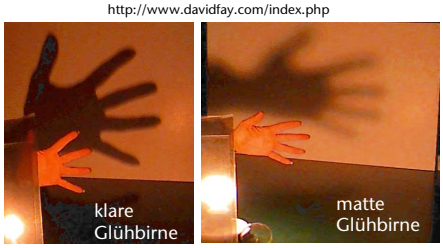


XVI. Léonard de Vinci (1452-1519). Lumière d'une fenêtre sur une sphère ombreuse avec (en partant du haut) ombre intermédiaire, primitive, dérivée et (sur la surface, en bas) portée. Plume et lavis sur pointe de métal sur papier, 24 x 38 cm. Paris, Bibliothèque de l'Institut de France (ms. 2185 ; B.N. 2038. f° 14 r°).

In der Realität ...

http://3media.initialized.org/photos/2000-10-18/index_gall.htm

http://www.davidfay.com/index.php

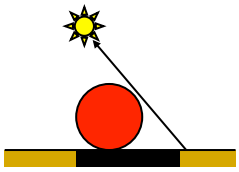


http://www.pa.uky.edu/~sciworks/light/preview/bulb2.htm

... und im Ray-Tracing

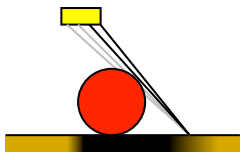
- Bisher: 1 Schattenstrahl

$$s_i = \begin{cases} 1\,, & \text{Lichtquelle sichtbar} \\ 0\,, & \text{nicht sichtbar} \end{cases}$$

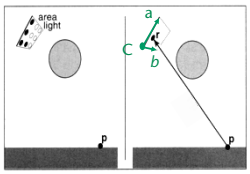


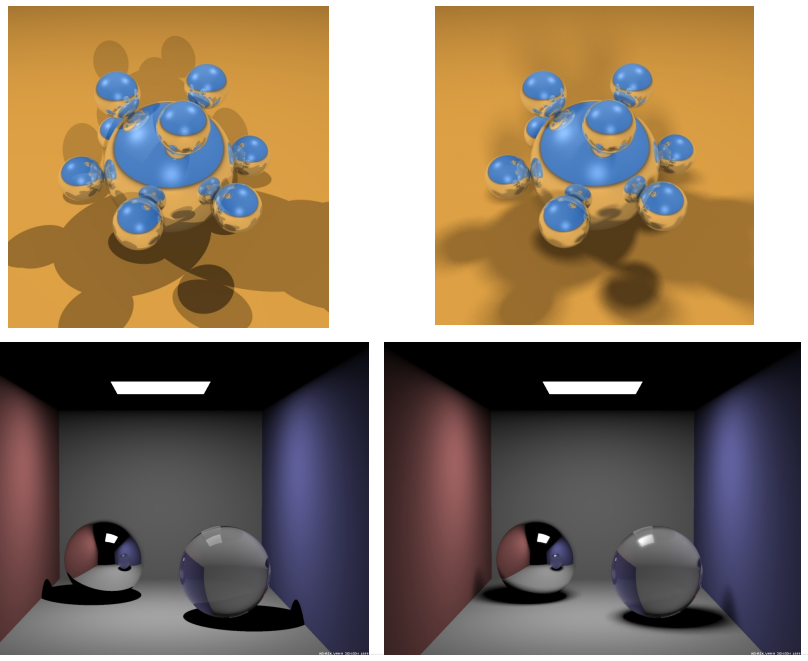
- Jetzt: mehrere Schattenstrahlen

$$s_i = \frac{\text{Anzahl sichtbarer Samples}}{\text{Anzahl Schattenstrahlen}}$$



- Drei Arten von Sampling der Lichtquelle:
  - Regelmäßige Abtastung der Lichtquelle
  - Zufällige Abtastung der Lichtquellen
  - Stratifizierte Abtastug





- Konstruktion des Schatten- und Halbschattenbereiches:

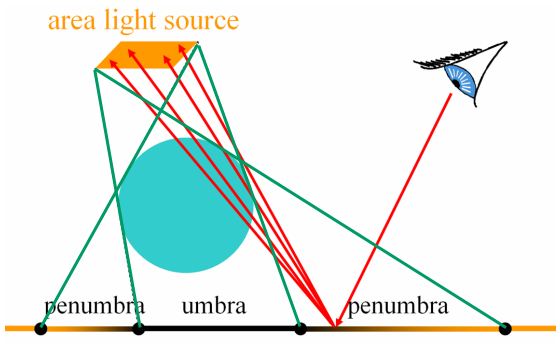
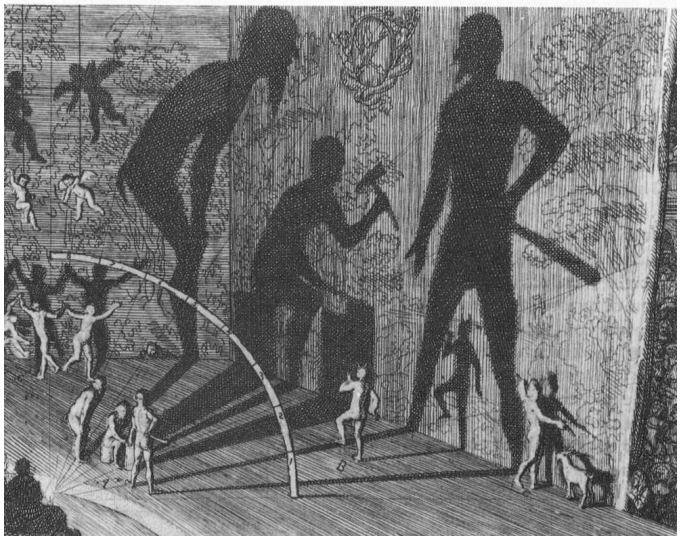
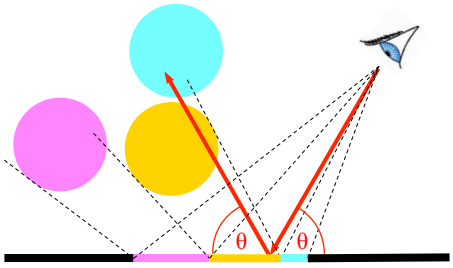


Plate 50 Samuel van Hoogstraten, *Shadow Theatre.* From *Inleyding tot de hooghe schoole der schilderkonst* 1678. (Einführung in die hohe Schule der Gemäldemalerei)

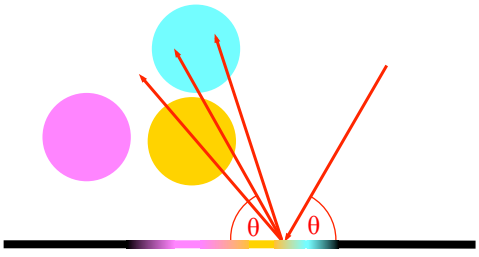


Spekulare Reflexion

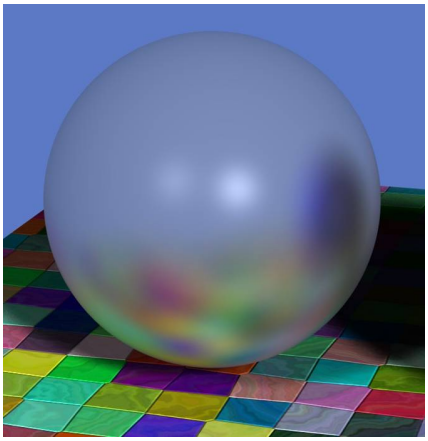
- Bisher: genau 1 reflektierter Strahl
  - Problem, falls die horizontale Fläche matt glänzend sein soll ...



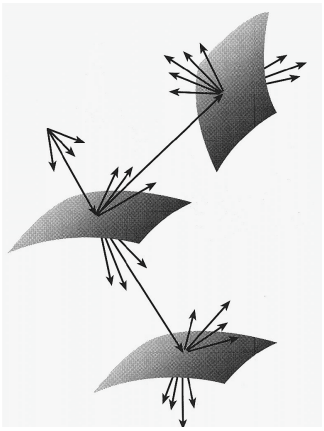
- Lösung (brute-force):
  - Viele reflektierte Strahlen
  - Beiträge gemäß Kosinus-Hoch-n-Gesetz (Phong) aufaddieren



- Beispiel:

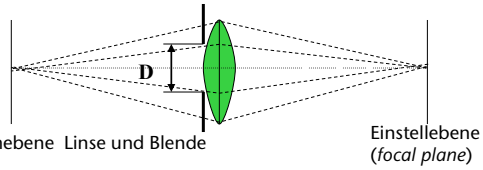


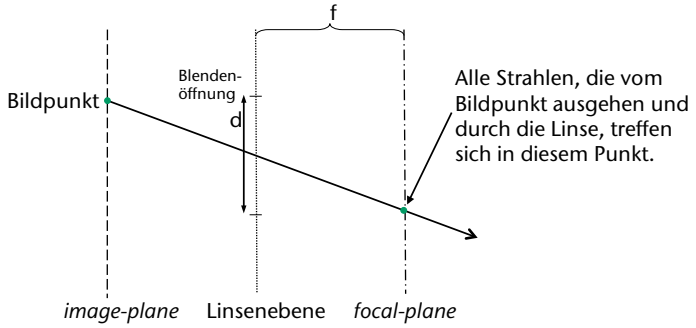
- Strahlbaum:



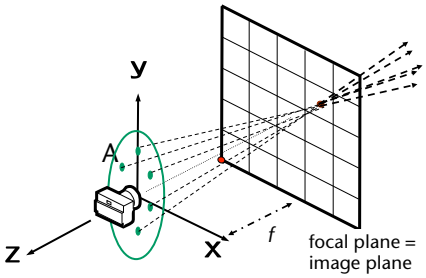
Tiefen(un-)schärfe

- Bisher: ideales Lochkameramodell
- Für Tiefenunschärfe muß man reale Kameras modellieren



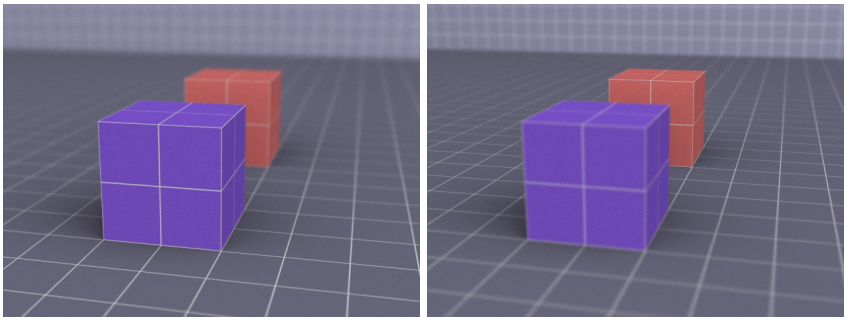


- Eine Klasse **`LensCamera`** würde die Strahlen also ungefähr so erzeugen:
  - Verteile Strahlen auf die gesamte Blendenöffnung und mittele



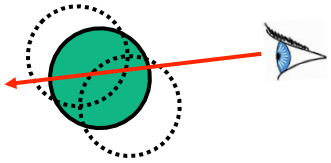
- Bemerkungen:
  - Sample die Scheibe (=Linse) stratifiziert
  - Achtung bei Kombination mit Anti-Aliasing

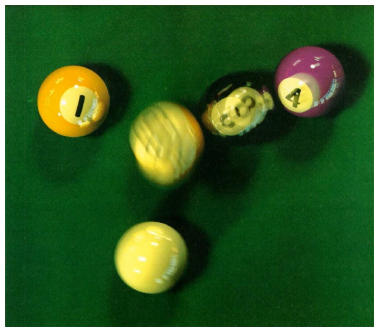
Beispiele



Motion Blur (Bewegungsunschärfe)

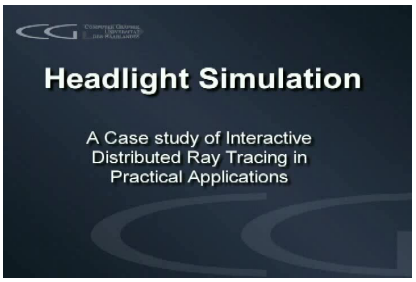
- Schieße viele Strahlen pro Pixel
- Wähle für jeden Strahl einen Zeitpunkt $t \in [t_0, t_1]$
- Betrachte während des Schnittests mit diesem Strahl alle Objekte an ihren Positionen P = P(t) zu diesem Zeitpunkt t
- Mittle Pixelwerte



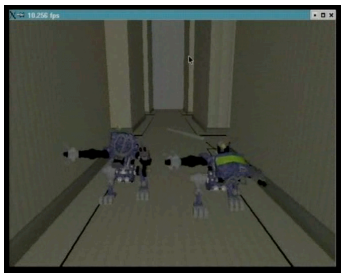


"But is it real-time?"

- Ray Tracing in der Vergangenheit war sehr langsam
- Inzwischen Echtzeit-Fähigkeit für einige Szenen
- OpenRT-Projekt: Real-Time Ray Tracing
  - Siehe http://www.openrt.de
- Special-Purpose-Hardware, PC-Cluster
- Nur eine Frage der Zeit, bis Commodity-Graphics-Hardware es kann



Uni Saarbrücken



Quake 3 mit Ray-Tracing. Plattform: Cluster mit 20 AMD XP1800. 2004
http://graphics.cs.uni-sb.de/~sidapohl/egoshooter/

Eine Anmerkung zu Typos

- Typos passieren auch auf den Folien
  - Keine Angst haben zu fragen!
  - Bitte teilen Sie mir Fehler mit
- Typos passieren sogar in Lehrbüchern
  - Ich selbst habe 2 nicht-triviale Fehler im Shirley-Buch, 2-te Auflage gefunden [WS 05/06]
  - Fazit: mitdenken, nicht einfach direkt kopieren