14. Raytracing

"globales" Verfahren:
bei Beleuchtungsrechnung für einen Punkt wird (potenziell) die gesamte Szene mit einbezogen

- Einsatz bei angestrebtem hohen Grad an Fotorealismus
- angezeigt bei Szenen mit nicht vernachlässigbarem Anteil an Mehrfachreflexionen
- Versuch, die physikalischen Vorgänge zu modellieren

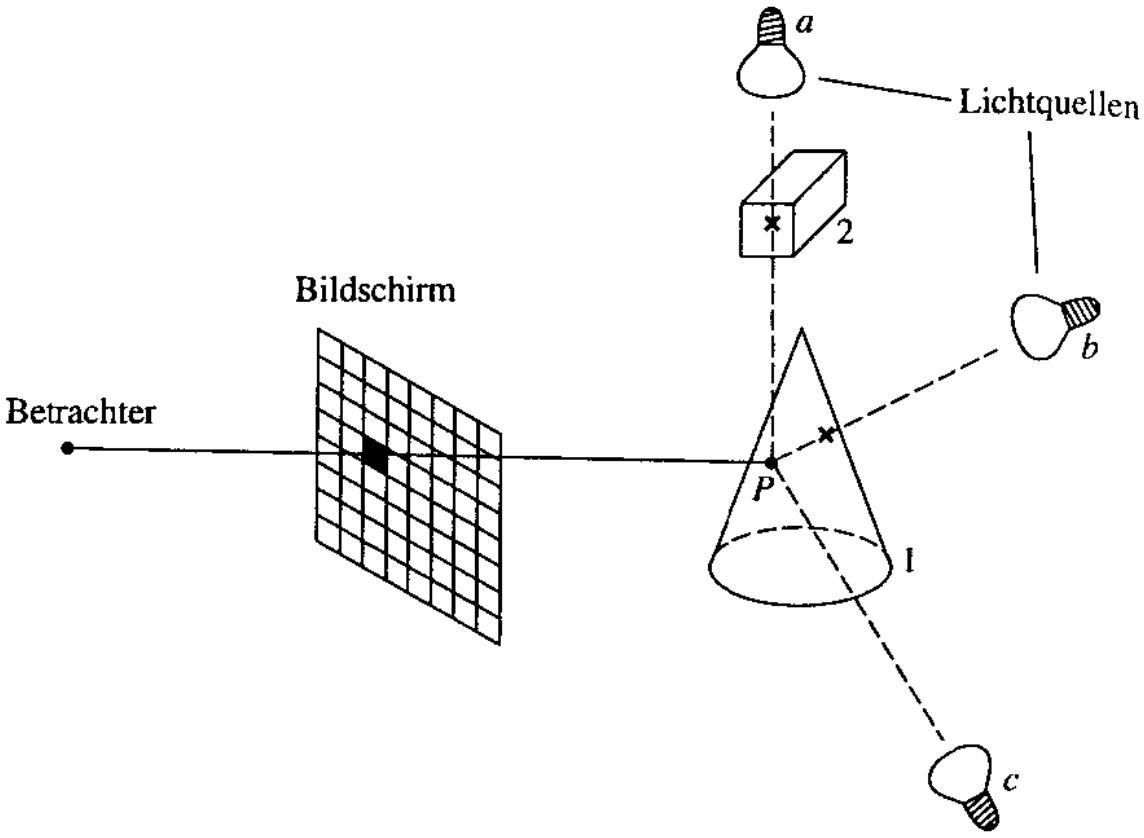
Raytracing:
"Strahlverfolgung"; genauer eigentlich: *visibility tracing*

- simuliert den Prozess der Lichtausbreitung und arbeitet dabei nach den strahlenoptischen Gesetzen für ideale Spiegelung und Lichtbrechung
- vor allem für Szenen mit hohem Anteil spiegelnder und transparenter Flächen
- *Grundidee:* Lichtstrahlen auf Weg von der Quelle bis zum Auge / zur virtuellen Kamera verfolgen
- Da nur wenige Strahlen das Auge erreichen, *kehrt man das Verfahren um* (Ausnutzung der Reziprozität der Reflexion) und sendet durch jedes Pixel des Bildschirms einen vom Augpunkt ausgehenden Strahl in die Szene
- Vereinfachung: beim konventionellen Raytracing (nach Whitted) werden *nur ideal reflektierte und ideal gebrochene* Strahlen weiterverfolgt.
- Trifft der Sehstrahl auf ein Objekt, so wird dort *das lokale Beleuchtungsmodell berechnet*. Anschließend werden zwei neue Strahlen erzeugt, nämlich der reflektierte und der gebrochene (transmittierte) Sehstrahl. Der Leuchtdichte-beitrag dieser beiden Strahlen wird rekursiv berechnet.

- Dieser Prozess bricht ab, wenn

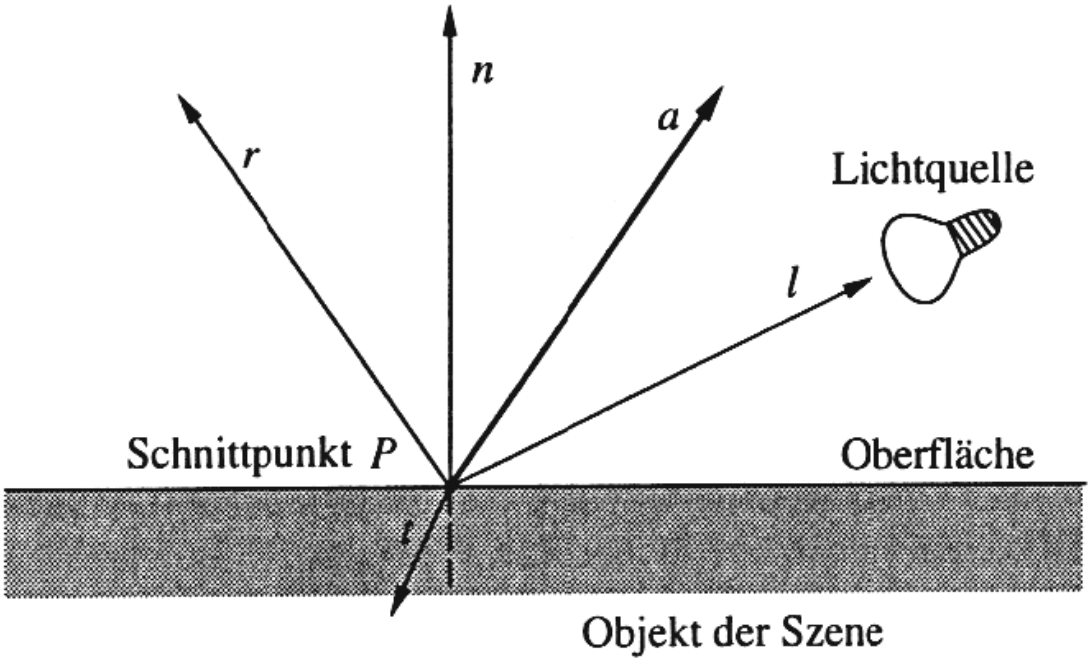
  - eine Lichtquelle getroffen wird,
  - die auf dem Strahl transportierte Energie zu gering wird,
  - der Sehstrahl die Szene verlässt.

- Aus praktischen Gründen setzt man a priori eine Obergrenze für die Rekursionstiefe.
- Das Verfahren wird zugleich auch für die *Schatten-berechnung* benutzt: Dazu sendet man von den Auftreff-punkten des Sehstrahls sogenannte *Schattenstrahlen* zu den Lichtquellen der Szene. Nur wenn kein undurchsichtiges Objekt zwischen einer Lichtquelle und dem Auftreffpunkt liegt, trägt sie zur (direkten) Beleuchtung des betreffenden Punktes bei.
- Das Verfahren erledigt auch die Sichtbarkeitsrechnung!



Schattenberechnung beim Raytracing (aus Bungartz et al. 1996). Der Schattenstrahl von *P* zu *a* schneidet Objekt 2, der von *P* zu *b* schneidet Objekt 1 ein weiteres Mal (Austritt), der Schattenstrahl von *P* zu *c* erreicht diese Lichtquelle dagegen ungehindert.

Wir haben also 3 (oder mehr) *sekundäre Strahlen*, die in jedem Auftreffpunkt erzeugt werden:



Primärstrahl *a* vom Betrachter (bzw. hier – als Vektor – zum Betrachter hin gerichtet)
Sekundärstrahl l zur Lichtquelle (Schattenstrahl)
Sekundärstrahl *r* (perfekt reflektierter Strahl)
Sekundärstrahl *t* (gebrochener Strahl)

Entsprechende Erweiterung des Phong-Beleuchtungsmodells durch Whitted (1980):
zum lokalen Beleuchtungsanteil nach Phong kommen noch der ideal spiegelnde und der ideal transmittierte Anteil hinzu.

nun:

$$I = k_a I_a + \sum_i f(d_i)\, I_{Lq_i} \left[k_d \left(\vec{N} \cdot \vec{L}_i \right) + k_s \left(\vec{V} \cdot \vec{R}_i \right)^n \right] + I_t + I_r$$

wobei:

I_t Intensität durch Transmission

I_r Intensität durch Reflektion

Berechnung der Richtungen der Sekundärstrahlen:

- für reflektierten Strahl nach idealem Reflexionsgesetz
- für transmittierten Strahl nach Brechungsgesetz von Snellius

Welche Strahlen gehen wo hin?

→ welcher Anteil wird transmittiert und reflektiert?

Lösung: Fresnel-Term (vereinfachte Form): Anteil reflektierter Energie

$$F = \frac{(\eta_i{}^2 - \eta_t{}^2)c_{it}{}^2 + (\cos\Theta_i{}^2 - \cos\Theta_t{}^2)n_{it}{}^2}{(c_{it}(\eta_i{}^2 + \eta_t{}^2) + n_{it}(\cos\Theta_i{}^2 + \cos\Theta_t{}^2))^2}$$

mit

$$c_{it} = cos\Theta_i \cos\Theta_t,\ n_{it} = \eta_i\eta_t.$$

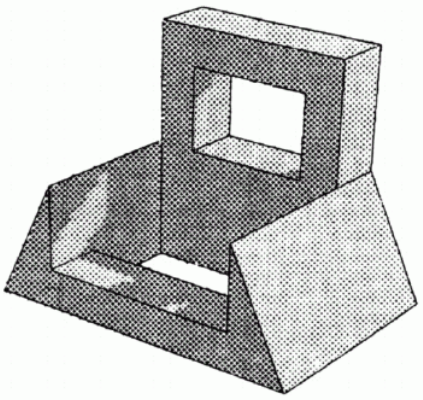
→ berechne F für jede Grenzschicht ($F \in [0..1]$).

→ schicke Strahlen zufaellig weiter je nach F

→ oder addiere Werte beider Strahlen gewichtet nach F

(θ_i : Winkel des einfallenden Strahls, θ_t : Winkel des transmittierten Strahls, jeweils zur Flächennormale,
η: Brechungsindices der beiden Medien)

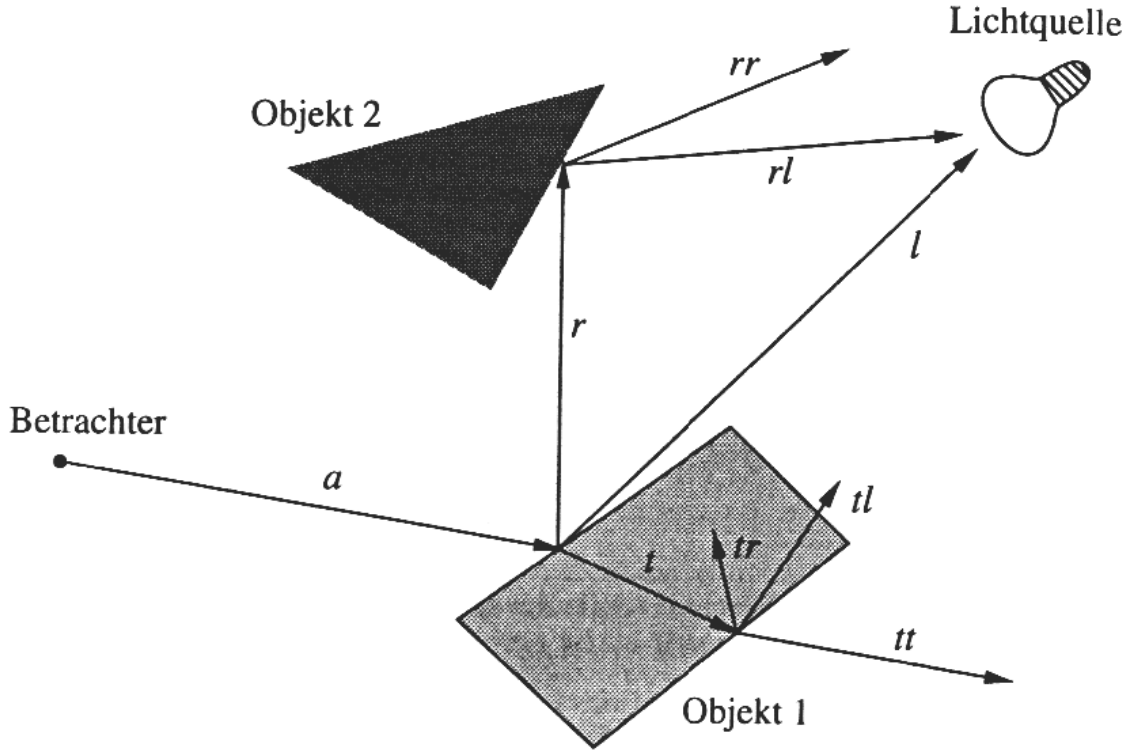
Historisch: Anwendung des Raytracing zunächst nur für die Schattenberechnung



Erstes Raytracingbild von Appel 1968

aus A Glassner: Principles of Digital Image Synthesis

Beispiel der rekursiven Strahlverfolgung, Rekursionstiefe 2: Objekt 1 wird als transparent angenommen, Objekt 2 ist lichtundurchlässig



(*a*: Primärstrahl, *r*: reflektierter Strahl, *t*: transmittierter Strahl, *l*: Schattenstrahl)

grobe Struktur des rekursiven Algorithmus:

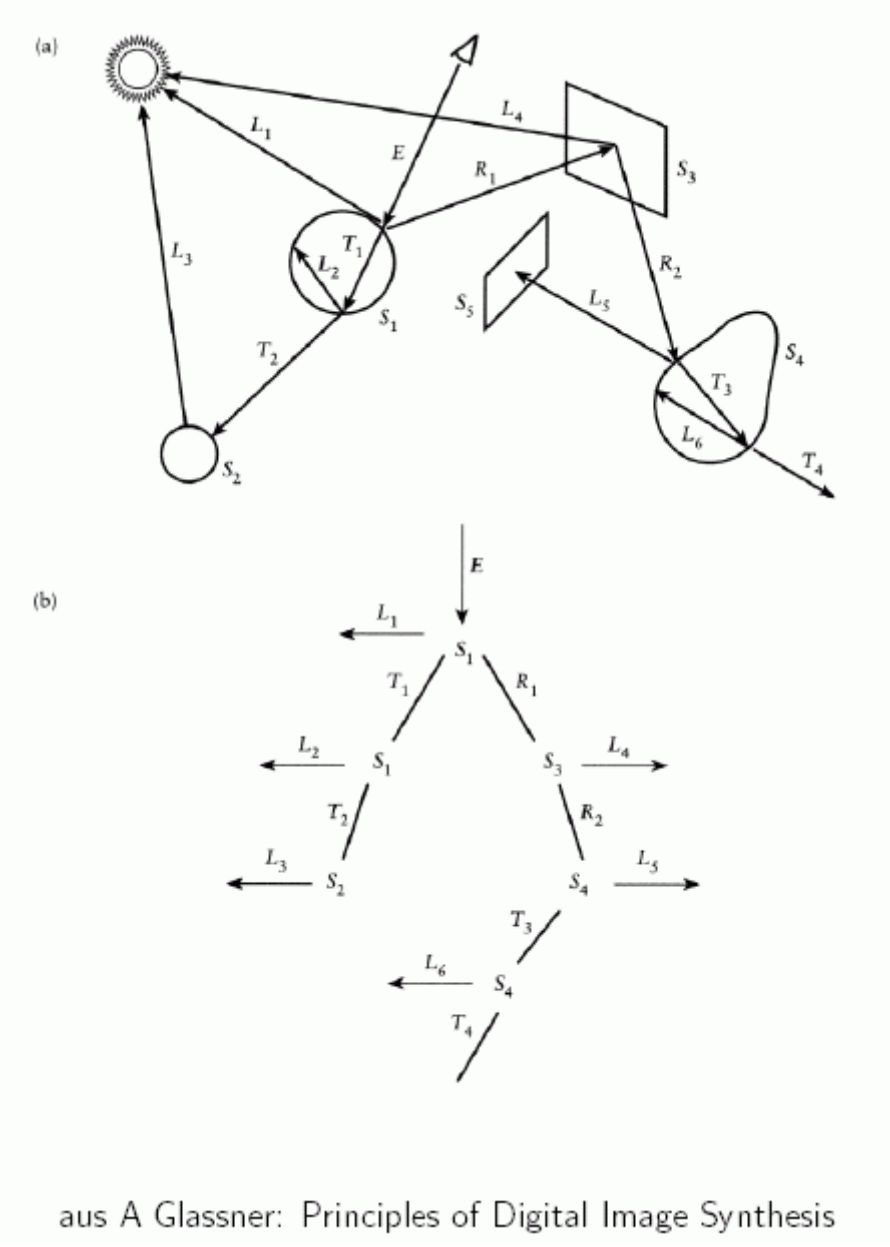
für jedes Pixel {

1. bestimme nächstliegenden Schnittpunkt des entspr. Sehstrahls mit einem Objekt der Szene
2. berechne ideal reflektierten Lichtstrahl
3. berechne die Leuchtdichte aus dieser Richtung
4. berechne ideal gebrochenen Lichtstrahl
5. berechne die Leuchtdichte aus dieser Richtung
6. berechne Schattenstrahl(en) zu den Lichtquellen
7. werte das Phong-Beleuchtungsmodell an dieser Stelle aus und addiere die gewichteten Leuchtdichten des reflektierten und des gebrochenen Sekundärstrahls

}

Beachte: Bei den Schattenstrahlen interessieren nicht die Positionen eventueller Schnittpunkte mit Szenenobjekten, nur, ob solche existieren.

Die Schnittpunkte der übrigen Strahlen bilden einen Baum:



aus A Glassner: Principles of Digital Image Synthesis

Der Algorithmus wird manchmal zweistufig durchgeführt:
in der 1. Phase wird der Baum mit Hilfe der Schnittpunkt-berechnungen aufgebaut, in der zweiten Phase wird der Baum traversiert, wobei die Intensitäten berechnet werden.

Als Faustregel für die maximale Rekursionstiefe gilt 5.

Pseudocode für die einphasige Version des Algorithmus (ohne Schattenstrahlen):

Raytracing-Code

→ zwei Prozeduren:

- Strahlanfrage (Ray_hit), Bestimmung Schnittpunkt
- Strahlverfolgung (TraceRay), rekursive Bestimmung Helligkeit

```
void TraceRay(start, direction, depth, colour);
point start, direction;
int depth;
colours *colour;
{
   int ray_hit();
   point hit_point, reflected_direction, transmitted_direction;
   colours local_colour, reflected_colour, transmitted_colour;
   object hit_object;
   if (depth > MAXDEPTH) *colour = BLACK;
   else {
     /* Intersect ray with all objects and find intersection point
     ( if any ) that is closest to start of ray */
     if (ray_hit( start, direction, &hit_object, &hit_point )) {
       /* contribution of local colour at intersection point */
       shade( hit_object, hit_point, &local_colour );
       /* calculate direction of reflected and refracted rays */
       calculate_reflection( hit_object, hit_point, &reflected_direction );
       calculate_transmission( hit_object, hit_point, &transmitted_direction );
       TraceRay( hit_point, reflected_direction, depth+1, &reflected_colour );
       TraceRay( hit_point, transmitted_direction, depth+1, &transmitted_colour );
       /* combine colours according to surface properties of hit_object */
       Combine( hit_object, local_colour, reflected_colour, transmitted_colour, colour );
     }
     else *colour = BACKGROUND_COLOUR;
   }
}
```

kritischer Teil des Algorithmus: **Schnittpunktberechnung**
Strahl mit Objekt

Whitted stellte nach Laufzeitmessungen fest: 75 % der Zeit werden für die Schnittpunktberechnungen verwendet und 12 % für die Berechnung des Beleuchtungsmodells

typische Schnittpunktprobleme:

Strahl – Kugel:
Strahl in (Zweipunkte-) Parameterform (Parameter t) gegeben
Kugel durch Mittelpunkt $M = (x_M, y_M, z_M)$ und Radius r:

$$(x - x_M)^2 + (y - y_M)^2 + (z - z_M)^2 = r^2$$

Einsetzen von x, y, z aus Parametergleichung des Strahls liefert quadratische Gleichung für t, Lösung ergibt 2 Werte
wähle kleineren der beiden Werte, dieser muss > 0 sein
Einsetzen in Parametergleichung des Strahls liefert Schnittpunktposition P
Normalenvektor am Schnittpunkt: $N = (P - M)/r$

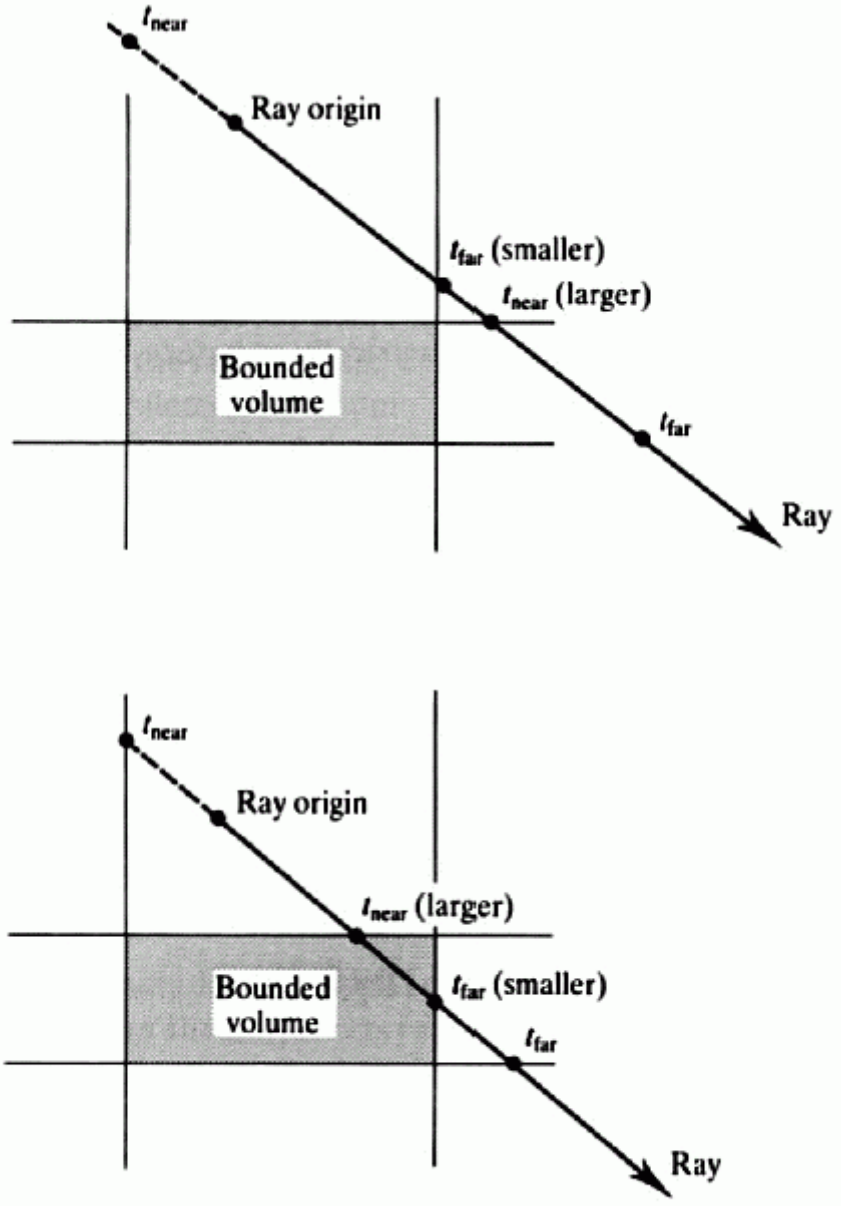
Strahl – Dreieck
- bilde Ebene durch Dreieck
- schneide Strahl mit Ebene
- teste, ob Schnittpunkt innerhalb des Dreiecks liegt (bei Verwendung zweier Seitenvektoren als Richtungsvektoren für 2-Parameterform der Ebenengleichung muss für die Parameter s, t gelten: $s, t, s+t \in [0; 1]$)

Objekte aus mehreren Dreiecken:
mit allen Dreiecken schneiden (level-of-detail-Ansatz sinnvoll)

Polygonnetze: Triangulation

Strahl – Quader
(wichtig, weil effizient berechenbar)
insbes.: achsenparallele Quader (Boxen)
einfacher Schnittalgorithmus:

- Bearbeite die jeweils parallelen Ebenenpaare und bestimme t_{near} und t_{far} (Abstand zu Ebene auf Strahl)
- ist der größere Wert von t_{near} größer als der kleinere von t_{far} $\Rightarrow$ kein Schnitt



aus: Watt, Watt: Advanced animation and rendering techniques

Strahl – Quadrik
allgemeine Bestimmungsgleichung für Quadriken:

$$A\,x^2+E\,y^2+H\,z^2+2\,Bxy+2\,Fyz+2\,Cxz+2\,Dx+2\,Gy+2\,Iz+J=0$$

$$(x,y,z,1)\begin{pmatrix} A & B & C & D \\ B & E & F & G \\ C & F & H & I \\ D & G & I & J \end{pmatrix}\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}=0$$

Spezialfälle:
Kugel, Kegel, Zylinder, Ellipsoid, Paraboloid, Hyperboloid

Schnittpunktbestimmung analog zur Kugel durch Einsetzen der parametrischen Strahl-Gleichung

andere Objekte
Schnittpunktverfahren für Fraktale; Prismen; Rotationskörper (vgl. Kajiya 1983)
bikubische Freiformflächen: schwierig, daher Zerlegung in Dreiecke

Anzahl der Schnittpunktberechnungen:
- proportional zur Anzahl der Strahlen und zur Anzahl der (Primitiv-) Objekte
- wächst exponentiell mit der Anzahl der Rekursionsstufen

zwei Verbesserungsansätze:
- effizientes Bestimmen, ob ein Objekt überhaupt geschnitten wird
- effiziente Schnittpunktbestimmung

Lösungsansätze für die Effizienzverbesserung:
Szene mit geeigneter Datenstruktur verwalten
("Beschleunigungsstruktur")

- bounding-Objekte (Quader, Kugeln etc.)
- Raumaufteilung (Zellen, Octrees, BSP-Trees etc.)

Ausnutzung der Objektkohärenz durch Raumaufteilung:

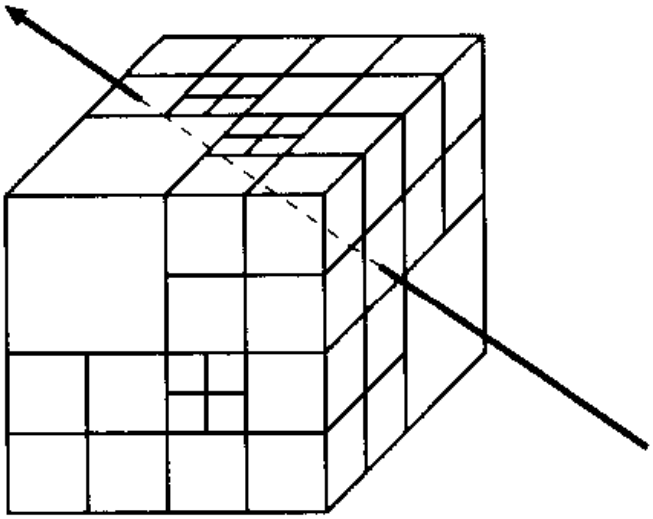
(a) *Voxel-Space*
Vorverarbeitungsschritt: Unterteile den 3D-Objektraum in Voxel ($\rightarrow$ reguläres kubisches Gitter)
bei der Strahlverfolgung werden dann der Reihe nach die vom Strahl durchlaufenen Voxel bestimmt (vgl. Schnitt Strahl – Quader, s.o.)
$\Rightarrow$ Schnittpunkte müssen nur noch mit denjenigen Objekten bestimmt werden, die (teilweise) in diesen Voxeln liegen

- inkrementeller Strahltraversierungsalgorithmus von Amanatides & Woo zum Raytracing im Voxel-Space (ähnlich Bresenham); s. Encarnação et al. II, S. 144f.
- durch Speicherung der Objekt-Primitive in Listen, die den sie enthaltenden Voxeln zugeordnet sind, kann Zeit gespart werden

reguläre Gitter gut geeignet für kleine und mittelgroße Szenen mit nicht allzu ungleichmäßig verteilten Objekt-Primitiven

wenn die Objekt-Primitive sehr "geklumpt" auftreten: besser:

(b) *Octree-Zerlegung*
- Startvoxel: achsenparallele Bounding-Box aller Szenen-objekte
- wird nach Bedarf rekursiv in jeweils 8 gleichgroße Sub-Voxel unterteilt



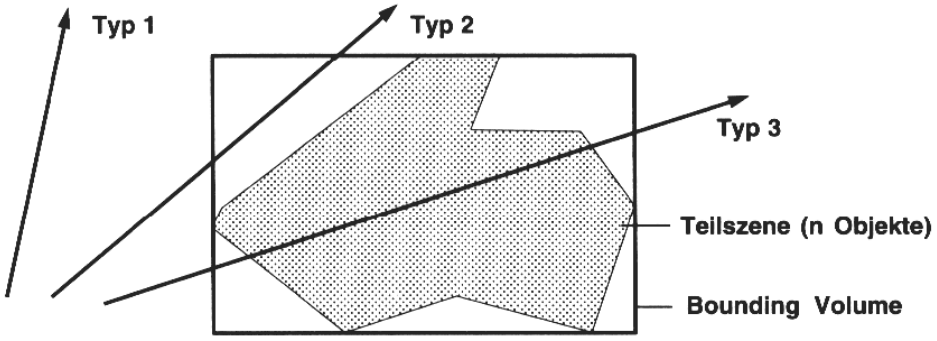
effizienter Strahltraversierungsalgorithmus für Octrees:

SMART-Algorithmus ("Spatial Measure for Accelerated RayTracing"; Spackman & Willis 1991), s. Encarnação et al. II, S. 145ff. – numerisch stabiler Algorithmus; Integerarithmetik.

Weitere Alternative: kd-Baum (hierarchische, binäre Raumaufteilung mit Schnittebenen parallel zu den Koordinatenebenen)

(c) Verwendung von hierarchischen *bounding volumes*

Prinzip der Effizienzverbesserung durch bounding volume beim Raytracing:



alle 3 Strahlen werden zunächst mit dem bounding volume geschnitten (einfache Schnittabfrage)

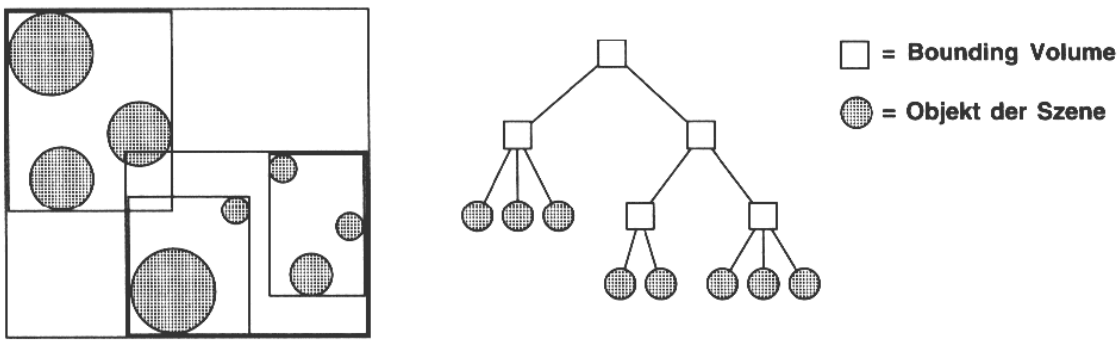
Typ 1: verfehlt das bounding volume, damit auch die darin enthaltene Teilszene ⇒ keine komplizierteren Schnitte notwendig
wenn bounding volume getroffen wird: noch keine Aussage, ob Typ 2 oder 3 vorliegt ⇒ Strahl mit der Teilszene schneiden

Kosten der Schnittpunktberechnung eines Strahls mit einer Teilszene nach diesem Verfahren:

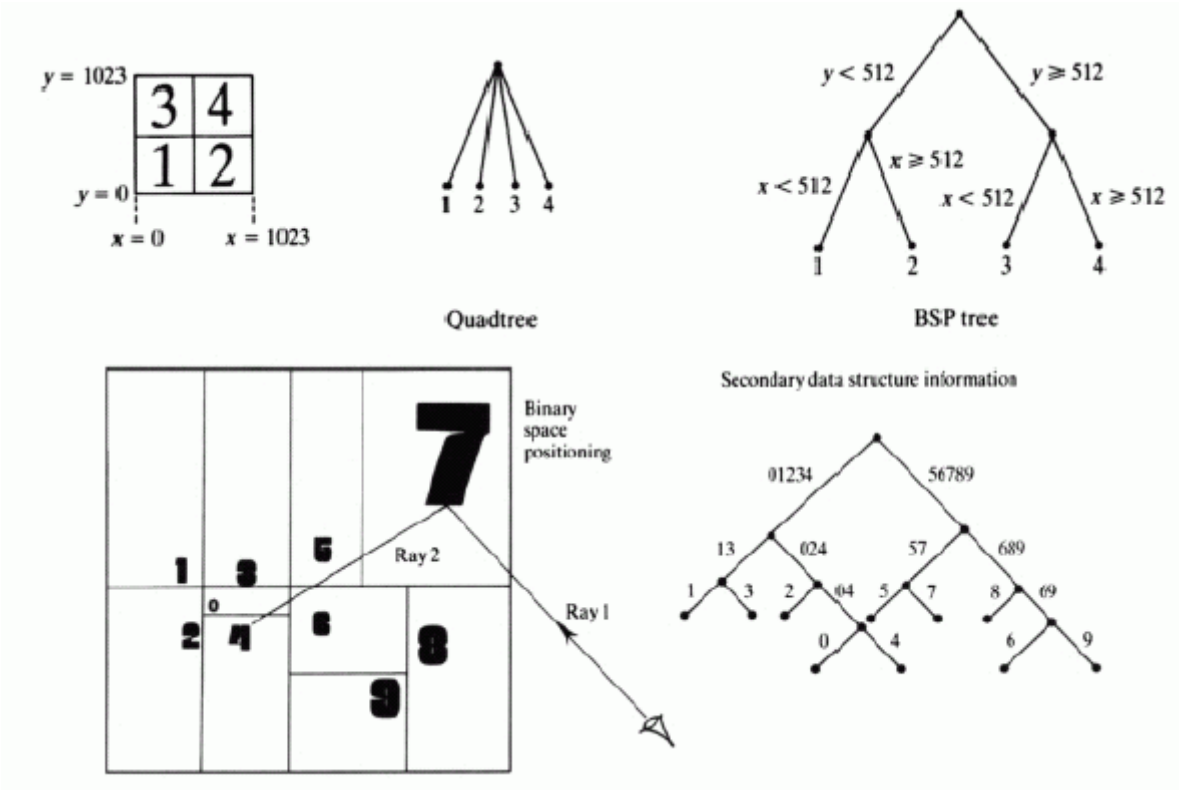
$$T = n \cdot B + m \cdot J$$

wobei
T = gesamte Schnittpunktberechnungskosten
n = Anzahl der Strahlen, die gegen das bounding volume auf einen Schnitt getestet werden (konstant)
B = Kosten des Schnittpunkttests mit dem bounding volume
m = Anzahl der Strahlen, die das bounding volume tatsächlich treffen ($m \leq n$)
J = Kosten der Schnittpunkttests mit den Objekten der Teilszene (konstant)
– durch Wahl des bounding volumes kann man versuchen, B und m so auszutarieren, dass T minimiert wird
(z.B. Quader, Ellipsoid, ..., konvexe Hülle)

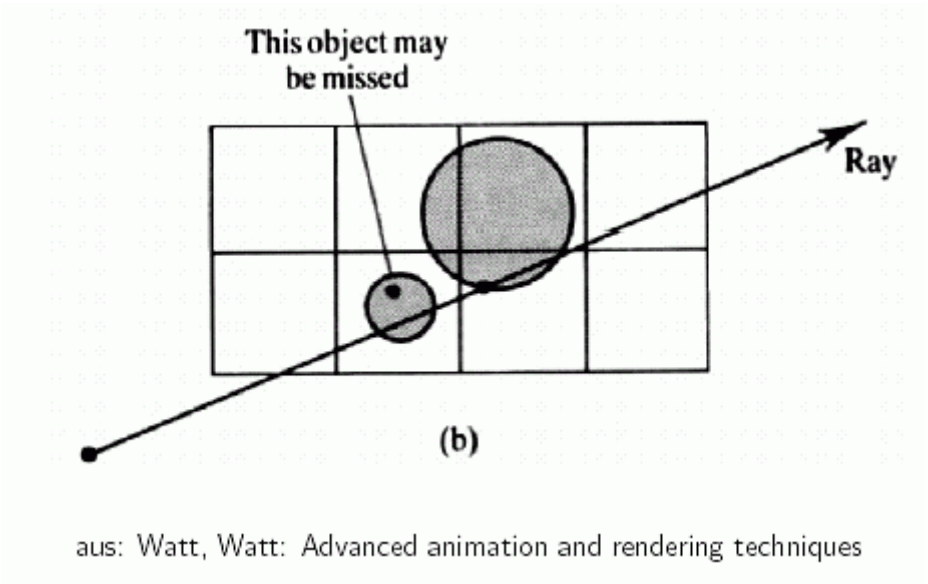
weitere Effizienzsteigerung:
Hierarchie von bounding volumes aufbauen



→ BSP-Baum
→ zum Aufbau: Median-Schnitt-Algorithmus (vgl. Kap. 5.3, optimierte Belegung der Farbtabelle)



Vorsicht bei allen Raumaufteilungsverfahren zum Raytracing:
es sind Problemfälle zu berücksichtigen
z.B. bei Voxel-Aufteilung:



aus: Watt, Watt: Advanced animation and rendering techniques

Weitere Beschleunigungstechniken:

Adaptive Tiefenkontrolle

Veränderung der Rekursionstiefe in Abhängigkeit vom Material – wenn zu erwartender Beitrag eines Sekundärstrahls zur Gesamthelligkeit eines Pixels zu klein wird, wird abgebrochen

Mischung von Standard- (Phong-) Shading mit Raytracing

- Raytracing nur bei spiegelnden und / oder durchsichtigen Flächen
- Spiegelungen werden als Texturen auf die Objekte projiziert, dadurch "Zwischenspeicherung" der Ergebnisse des (unvollständigen) Raytracing

Schatten-Caches

Ausnutzung der Objektkohärenz beim Schattenwurf:
jede Lichtquelle verwaltet einen Schatten-Cache, der das Resultat der Strahlverfolgung des letzten Schattenstrahls speichert. War der Schattenstrahl auf ein undurchsichtiges Objekt gestoßen, wird ein Pointer auf dieses schattenwerfende Objekt im Schatten-Cache der Lichtquelle gespeichert (sonst ein Null-Pointer). Bevor der nächste Schattenstrahl verfolgt wird, führt man einen Schnittpunkttest mit dem durch den Schatten-Cache referenzierten Objekt durch – nur wenn dieser negativ ist, wird der Schattenstrahl explizit durch die Szene verfolgt
$\Rightarrow$ ggf. Einsparung von Schnittpunkttests mit anderen, irrelevanten Objekten

Pixel-selected raytracing

(für schnelle previewing-Bilder, die nicht alle Details enthalten müssen)

- Detektieren homogener Regionen in der Bildebene; diese werden interpoliert
- Divide-and-Conquer-Ansatz

1. Unterteile die Bildebene in Regionen von je $Dx \times Dy$ Pixeln
2. Bestimme die Intensitäten der 4 Eck-Pixel durch Raytracing
3. Sind die Pixel direkt benachbart, gib die Pixelfarben aus und stoppe.
4. Ist die Intensitätsdifferenz zwischen den Eckpunkten kleiner als ein Schwellenwert, so fülle die Region durch bilineare Interpolation der 4 Eck-Intensitäten und stoppe.
5. Sonst unterteile die Region in 2 oder 4 Unterregionen, berechne alle noch nicht bekannten Ecken-Intensitäten durch Raytracing und wende die Schritte 3 bis 5 rekursiv an.

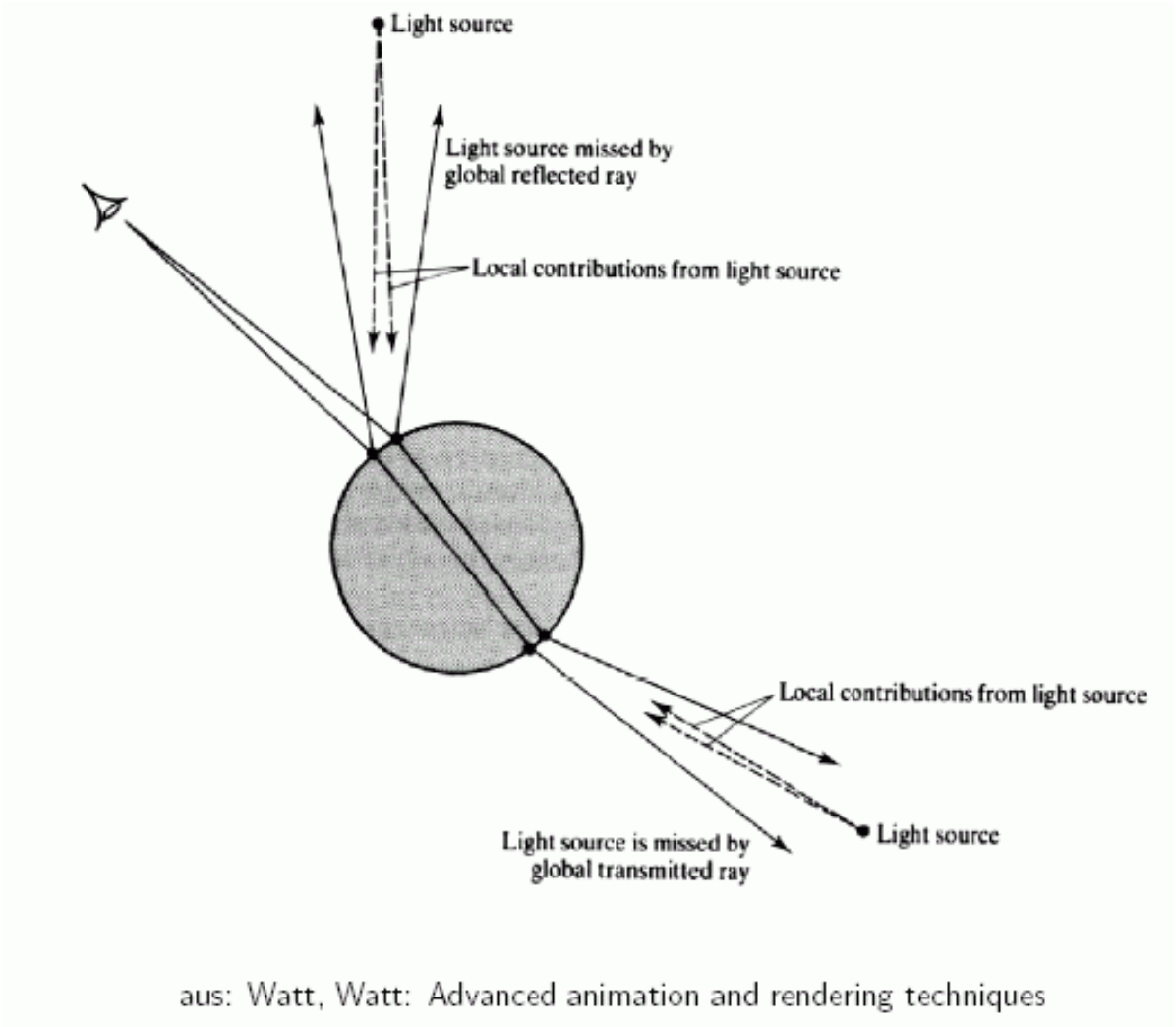
Algorithmus produziert Fehler, wenn kleine Objekte nicht durch Strahlen getroffen werden.
Weitere Verbesserungen dieser Technik möglich (Akimoto et al.).

Probleme beim Standard- (Whitted-) Raytracing

- keine Halbschatten darstellbar
  Schattenwürfe wirken unnatürlich scharf

- Reflexionen / Transmissionen sind "zu perfekt" – in der Realität gibt es in der Regel Abweichungen vom perfekten Spiegel

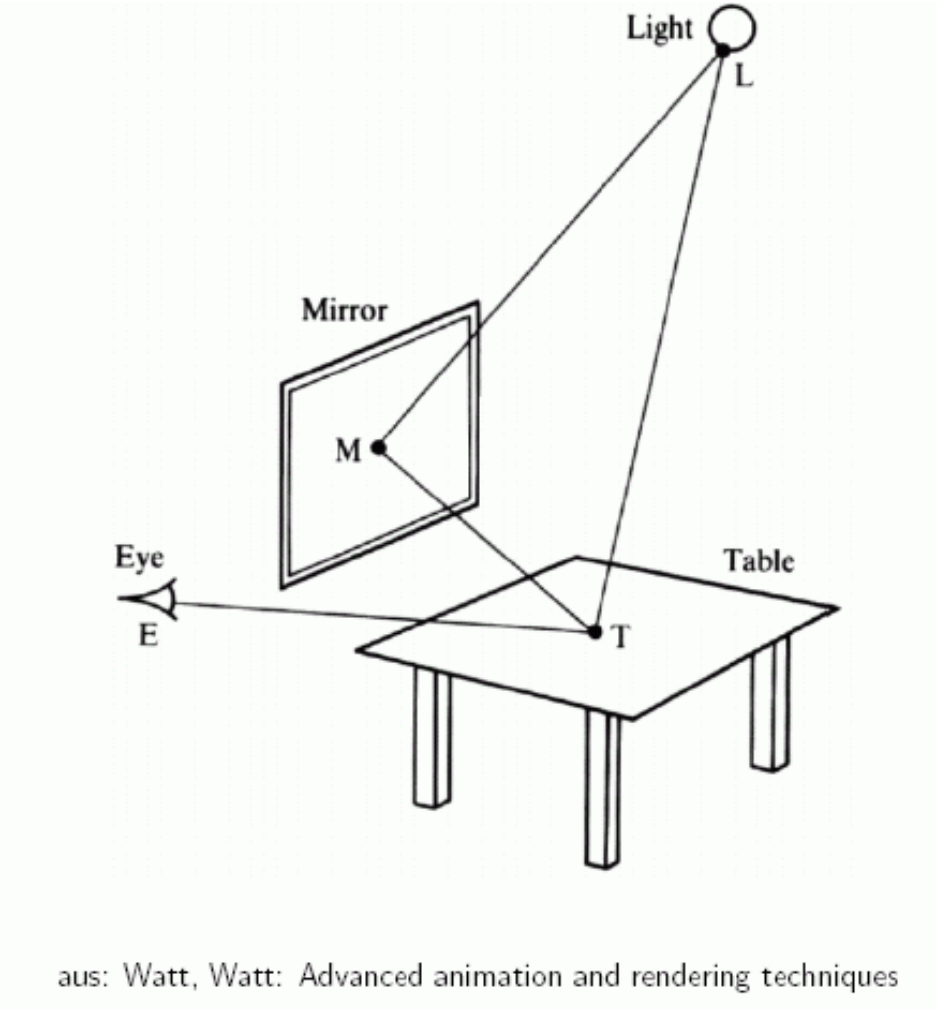
- mit Strahlen ist eine Punktlichtquelle oft nicht zu finden:



aus: Watt, Watt: Advanced animation and rendering techniques

- spiegelnd-diffuse Interaktionen werden nicht korrekt abgebildet

Beispiel: Raytracing erkennt den Lichtweg Lichtquelle-Spiegel-Tisch nicht, wenn der Tisch nur diffuses Material besitzt



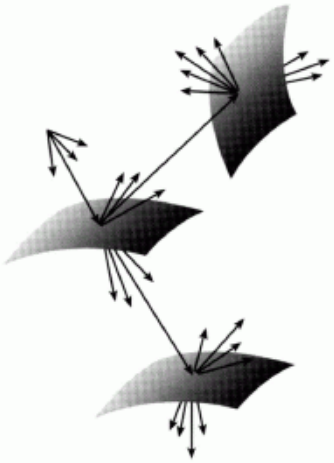
aus: Watt, Watt: Advanced animation and rendering techniques

zur Behebung dieser Mängel wurden verschiedene Erweiterungen und Modifikationen des Standard-Raytracing vorgeschlagen

Fortgeschrittene Raytracing-Verfahren

Distributed Raytracing (Stochastical Raytracing)

auch: Path Tracing



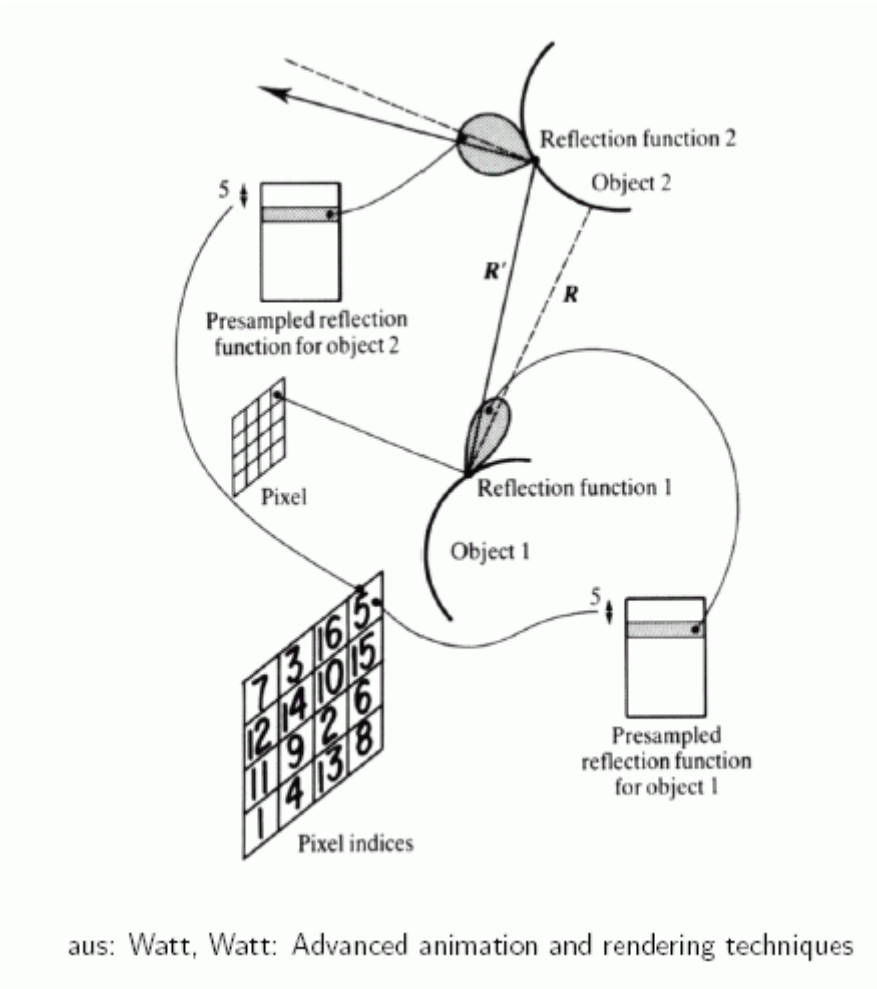
aus A Glassner: Principles of Digital Image Synthesis

einzelne Strahlen werden durch Strahlenbündel ersetzt mit zufällig (nach vorgegebener Verteilungsfunktion) ausgewählten Repräsentanten (klassische Monte-Carlo-Methode)

Vorteile:

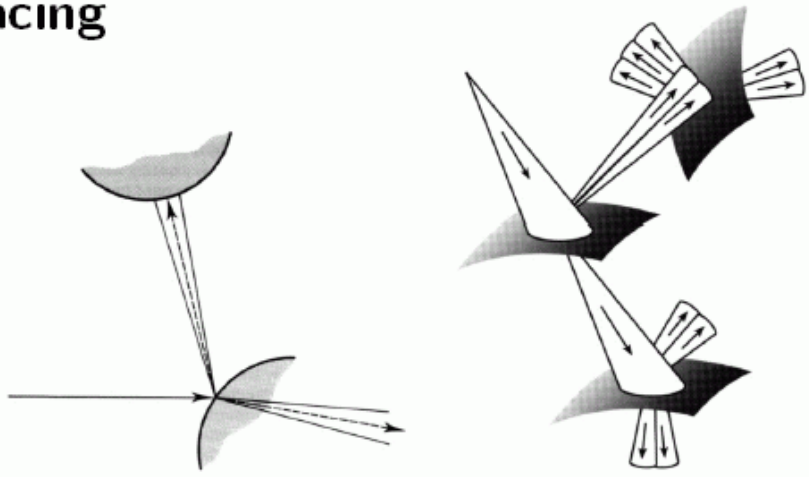
- nicht-perfekte Reflexion / Transmission wird simuliert
- weiche Schatten
- zusätzlich möglich: Simulation von Tiefenschärfe (Linsen-Effekt)

bei Kenntnis der BRDF eines Objekts können die Reflexionsstrahlen nach dieser verteilt werden



aus: Watt, Watt: Advanced animation and rendering techniques

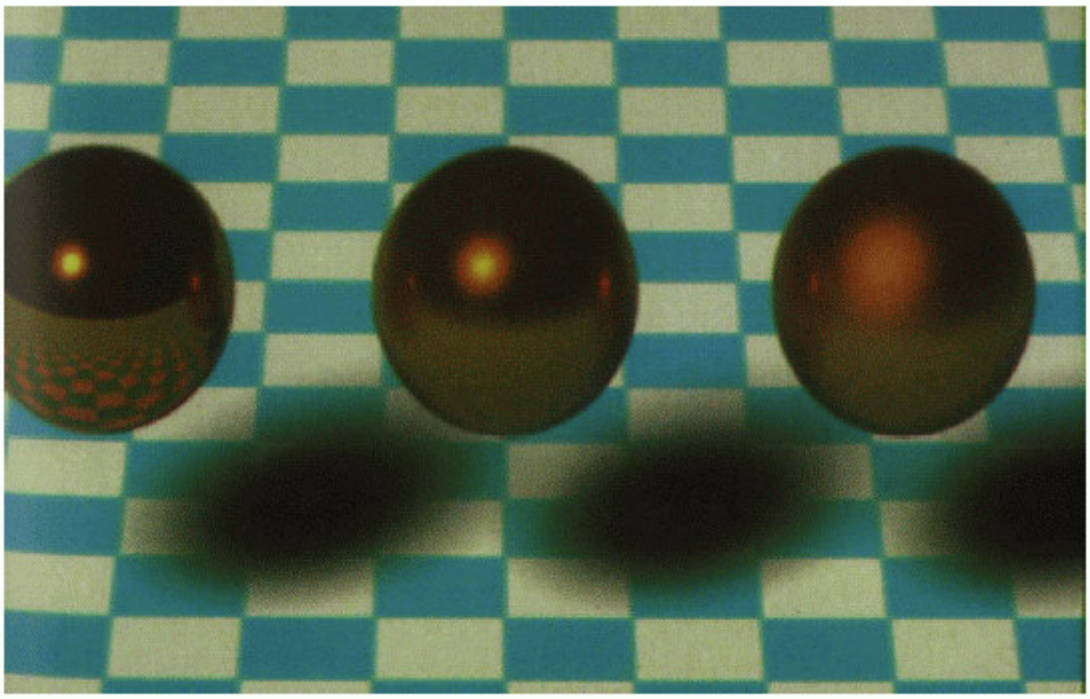
Cone-Tracing



aus: Watt, Watt bzw. Glassner

- verwendet Kegel statt Strahlen
- Vermeidung der Zufallskomponente
- erlaubt ebenfalls nicht-perfekte Spiegel und weiche Schatten
- Nachteil: aufwändige Schnittoperationen

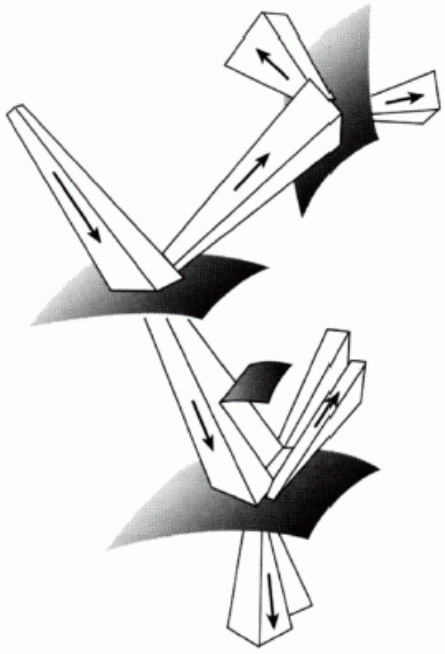
Beispiel für die Anwendung:



Conetracing

Vereinfachung:

Beam-Tracing



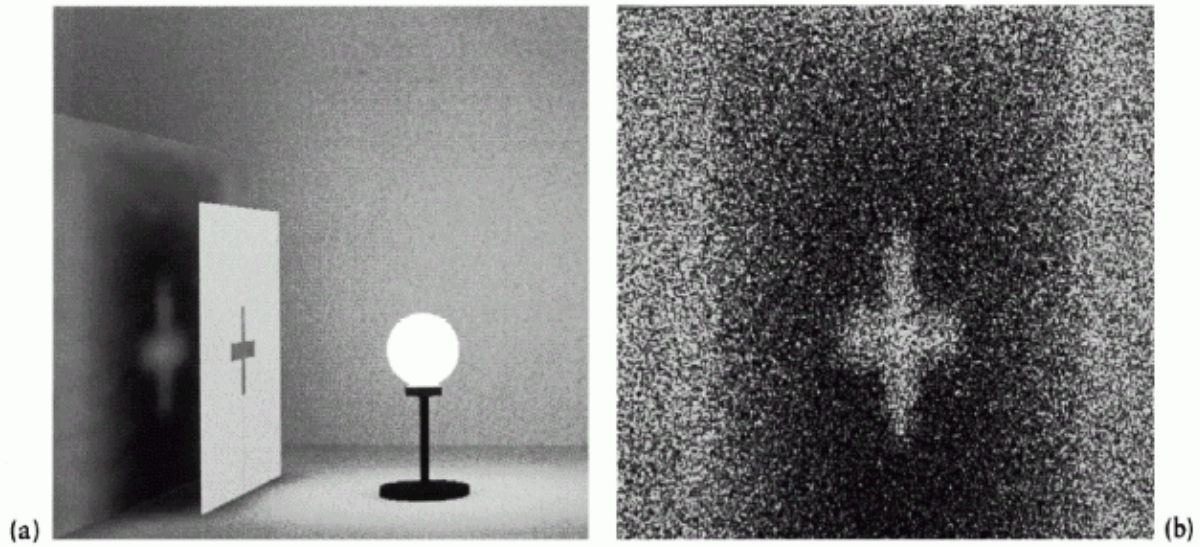
→ statt Kegel werden Pyramidenstümpfe verwendet

zur Vermeidung einiger Probleme des Standard-Raytracing, insbes. der fehlenden spiegelnd-diffusen Interaktion:

Photon-Tracing, d.h. Rückwärts-Raytracing

Umkehrung der Richtung des Standard-Raytracing: Photonen werden von der Lichtquelle aus verfolgt

- Problem: nur wenige Strahlen treffen das Auge
  $\Rightarrow$ Optimierung notwendig
- Rauschen tritt auf, wenn Photonen zufällig losgeschickt werden
  $\Rightarrow$ sehr viele Photonen notwendig



Bildgenerierung über Photon-Tracing

aus A Glassner: Principles of Digital Image Synthesis

Verbesserung des Ansatzes:

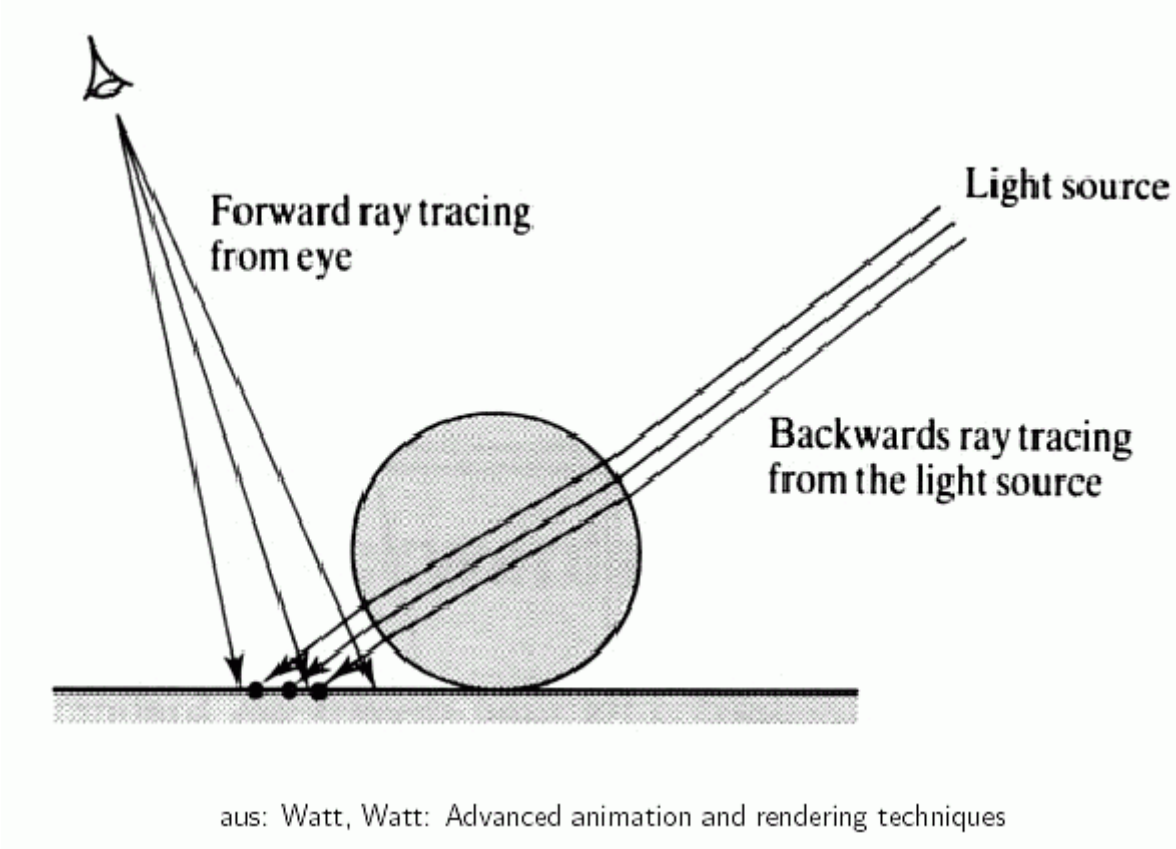
Two-Pass Raytracing

Kombination Visibility Tracing (Standard) und Photon-Tracing; Idee: Vorteile beider Verfahren kombinieren

1. Photon-Tracing von Lichtquelle über Spiegelungen und Brechungen zu diffuser Oberfläche
Lichtwerte dort als Quasi-Textur für diffuse Fläche speichern

2. Raytracing (Standard) vom Auge bis zur diffusen Fläche

"auf diffusen Flächen wird Leuchtkraft gesammelt und zwischengespeichert"



aus: Watt, Watt: Advanced animation and rendering techniques

Probleme:

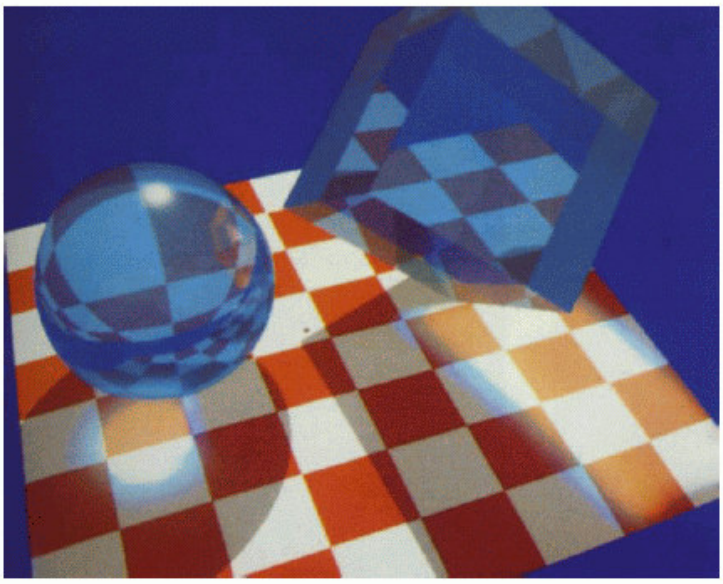
- Aliasing kann auftreten
- immer noch keine Interaktion zwischen mehreren diffusen Oberflächen
  aber: praktisch alle anderen Effekte

weitere Varianten:

- Bidirectional path tracing – Pfade (des Lichts) zwischen Lichtquelle und virtueller Kamera werden generiert
- Metropolis light transport – diese Pfade werden durch kleine Änderungen modifiziert

Einige Spezialeffekte, die mit advanced raytracing erzielt werden können:

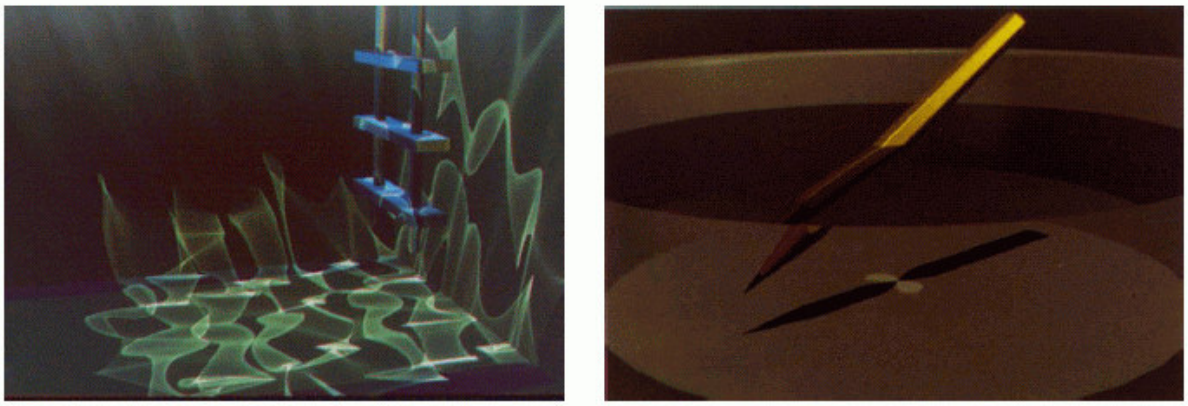
Kaustiken
= Lichtverstärkung (reflexiv oder transmissiv) durch Oberflächen



Lichtverstärkung durch Glas

aus: Watt, Watt: Advanced animation and rendering techniques

auch hierfür werden Photonen auf diffusen Flächen aufgesammelt und in Texturdaten gespeichert



Lichtverstärkung und -brechung durch Wasser

aus: Watt, Watt: Advanced animation and rendering techniques

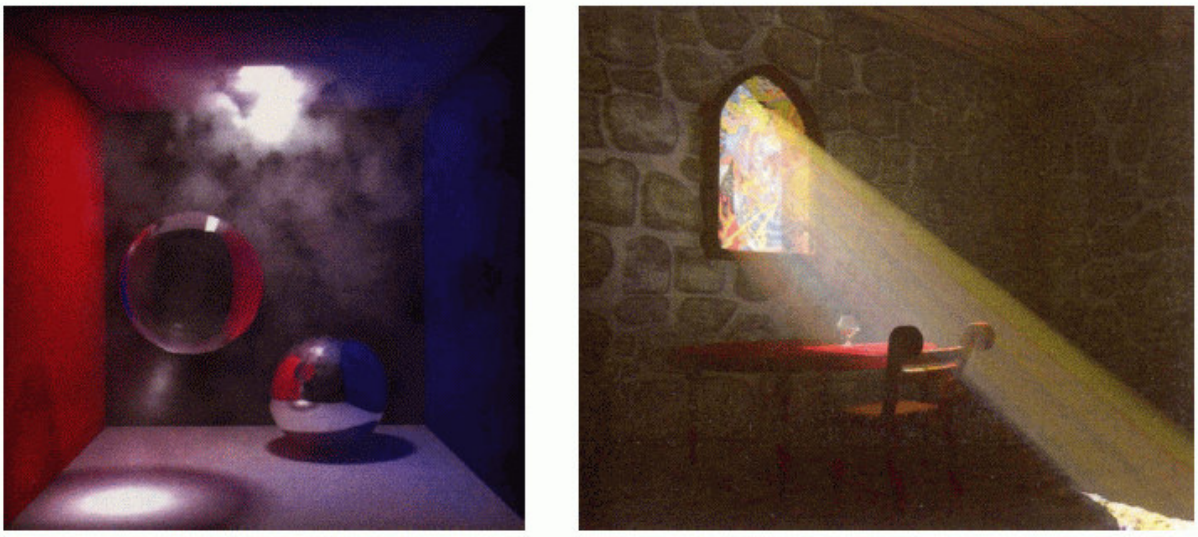
Dispersion

Staubteilchen streuen Licht $\Rightarrow$ sichtbare Lichtstrahlen

hierfür:
- Speicherung von "Photonen" in Volumina
- Raytracing-Modifikation: kommt Strahl in der Nähe eines gespeicherten Photons vorbei, wird er aufgehellt

Probleme:
- Photoneneffizienz (wieviele Photonen sind nötig)
- Speichereffizienz (wie speichere ich Photonen effizient)
- Rendering-Effizienz (wie finde ich schnell heraus, ob und wo ein Photon in der Nähe eines Strahls ist)

Lösungsansatz von Jensen & Christensen (SIGGRAPH 1998)



Mit Photonmaps erzeugte Bilder (Rauch, farbige Lichtstrahlen)

siehe auch
Volumen-Rendering (nächstes Kapitel)

Bewertung des Raytracing (Fazit)

Vorteile:

- Szene kann beliebig komplexe Objekte enthalten – einzige Bedingung: Objektnormalen und Schnittpunkte mit Strahlen müssen berechnet werden können. Keine Notwendigkeit, alle Objekte durch Polygone zu approximieren
- Hidden Surface Removal, Schattenberechnung, Berechnung von Reflexionen und Transparenzen werden alle mit erledigt
- Explizite perspektivische Transformationen der Objekte und Clipping-Berechnungen sind nicht notwending
- Objekte dürfen sich gegenseitig durchdringen, Schnitte zwischen Objekten brauchen nicht berechnet zu werden
- das Beleuchtungsmodell muss nur in sichtbaren Punkten ausgewertet werden

Nachteile:

- Abtastung der Szene mit einem Strahl pro Pixel erzeugt in der Regel Aliasing (kann mit Supersampling oder stochastischem Abtasten gemildert werden)
- großer **Rechenaufwand**, da für die (zahlreichen) Schnitt-punktberechnungen im Objektraum in der Regel Floating-Point-Arithmetik erforderlich ist; u.U. müssen viele Millionen Strahlen verfolgt werden. Beschleunigungs-Datenstrukturen erforderlich!
- Schatten haben stets scharfe Grenzen – weiche Halbschatten nur durch Erweiterungen des Standardverfahrens
- Schatten müssen bei jeder Änderung der Kameraparameter neu berechnet werden, obwohl sie nur von den Lichtquellen und den Objekten der Szene abhängen
- keine Berücksichtigung von Mehrfachreflexionen an diffusen Oberflächen (diese bildet aber signifikanten Anteil der Beleuchtung in Innenräumen)
- Szenen werden bzgl. eines Blickpunktes berechnet – ein Durchwandern von Räumen in Echtzeit erfordert daher viel Rechenaufwand und ist erst in jüngster Zeit möglich