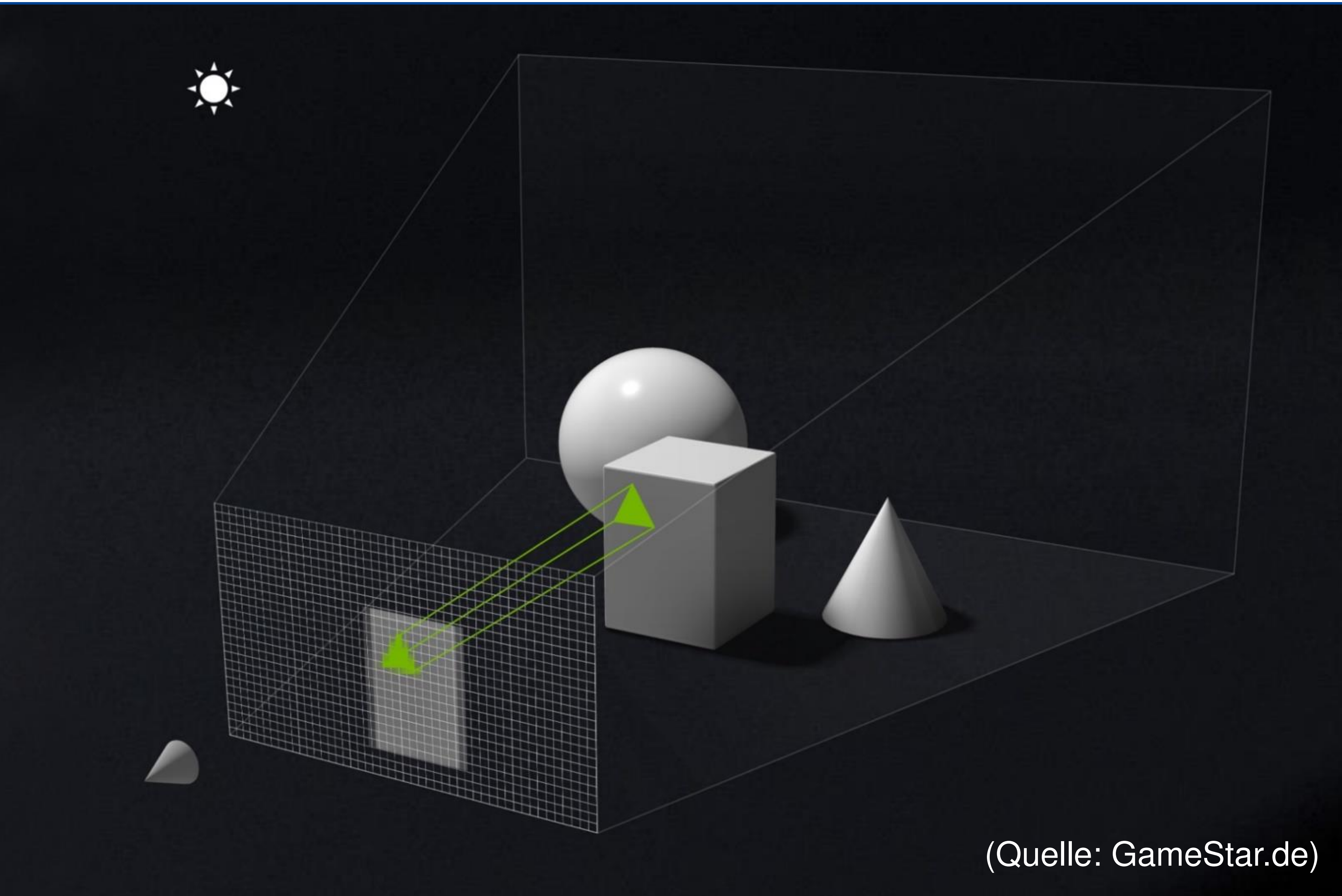
Bildsynthese (Rendering)

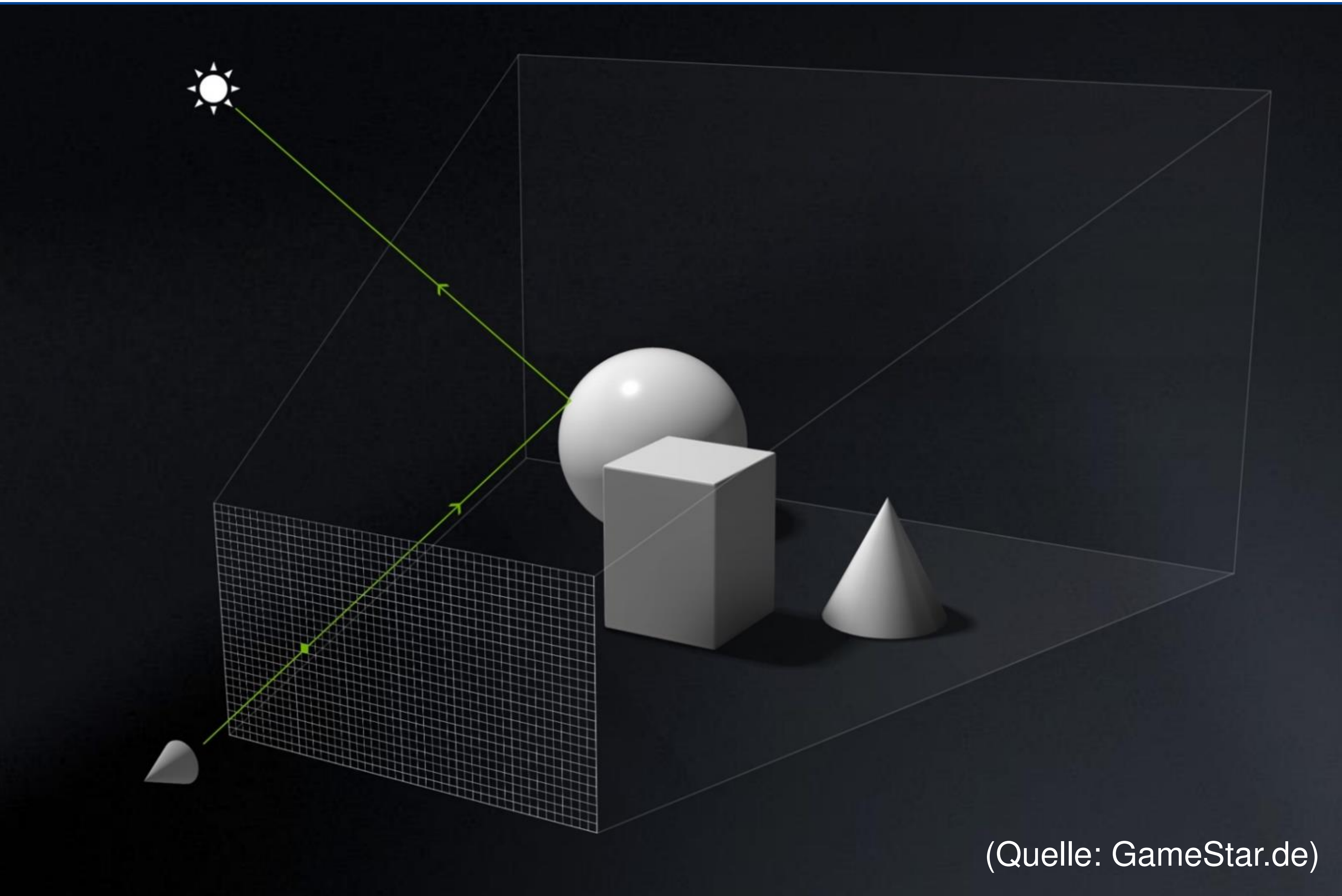
- Fotorealismus => Simulation von Licht

[Gkioulekas et al. 2013]

Rasterisierung (ab kommender Vorlesung)



Raytracing (heute)



Raytracing

Matthias B. Hullin

Institut für Informatik II, Universität Bonn

Licht

Strahlenmodell

(Geometrische Optik)

Strahl; Lichtenergie, die sich auf wohldefiniertem Pfad ausbreitet (im Vakuum: entlang einer Geraden)

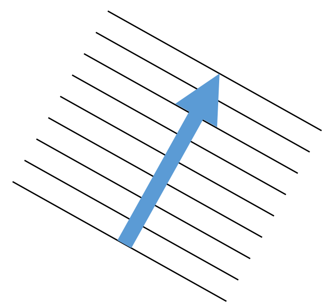
Modell gilt für inkohärente, makroskopische Lichtausbreitung

(>99% der Computer-grafik)

Wellenmodell

(Physikalische Optik)

Welle; räumlich und zeitlich veränderliches elektromagnetisches Feld

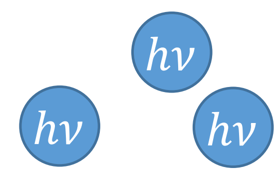


Gilt insbes. für Streuung/ Beugung an kleinen Strukturen, kohärentes Licht (Interferenz)

Teilchenmodell

(Quantenoptik)

Photon; Teilchen mit Energie $E = h\nu = {}^{hc}/_{\lambda}$; fliegt mit Licht-geschwindigkeit; keine Ruhemasse



Gilt auch für Wechsel-wirkung mit Elektronen sowie für sehr geringe Intensitäten

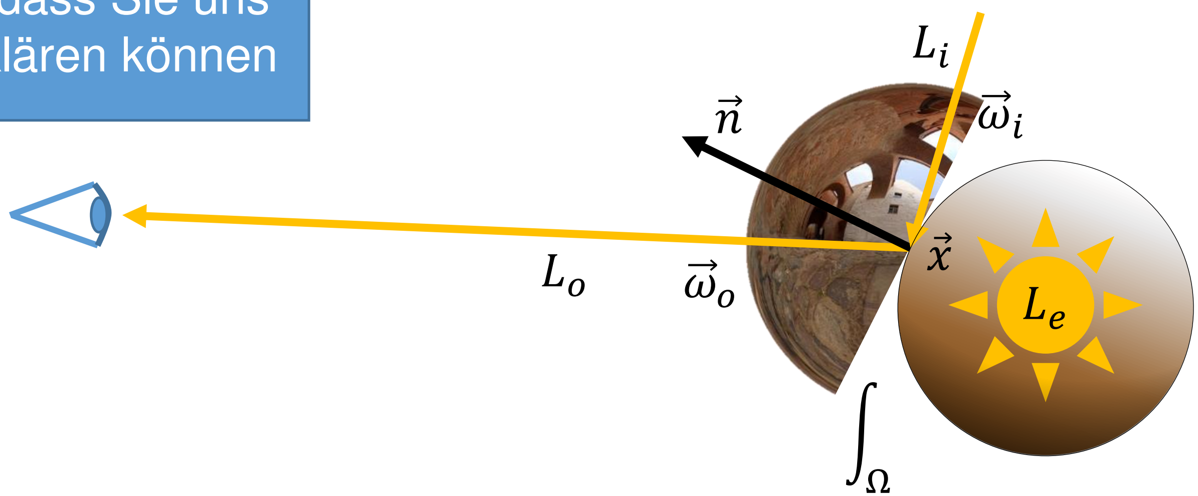
Ray tracing = Strahlen verfolgen

Die Renderinggleichung für Oberflächen

$$L_o(\vec{x},\vec{\omega}_o) = \underbrace{L_e(\vec{x},\vec{\omega}_o)}_{\text{Emission}} + \underbrace{\int_\Omega f(\vec{\omega}_i,\vec{\omega}_o) L_i(\vec{x},\vec{\omega}_i) \langle \vec{\omega}_i,\vec{n} \rangle d\vec{\omega}_i}_{\text{Reflexion}}$$

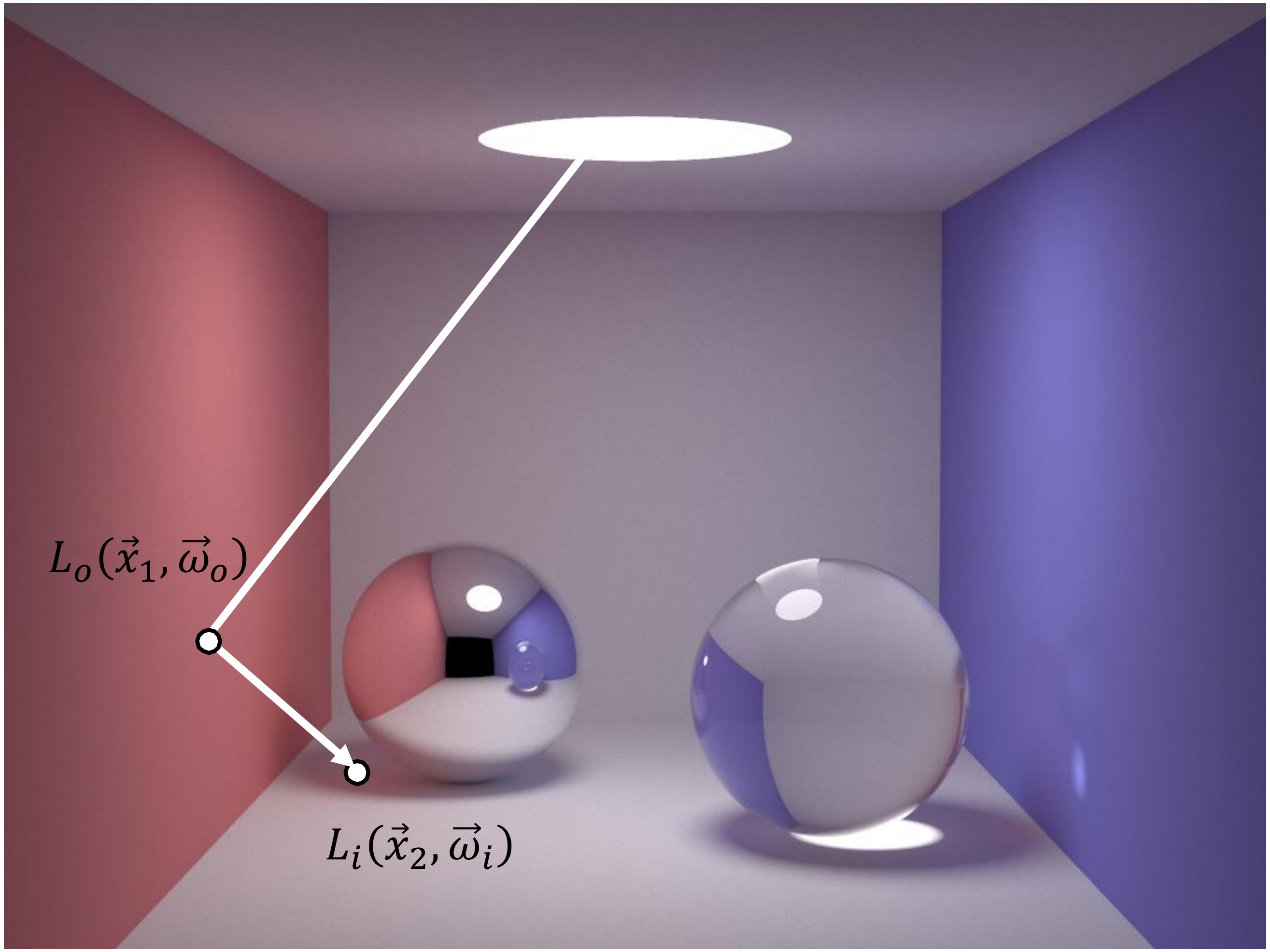
Streufunktion (BRDF) einfallende Radianz “Cosinus-term”

Wir erwarten, dass Sie uns diese Folie erklären können

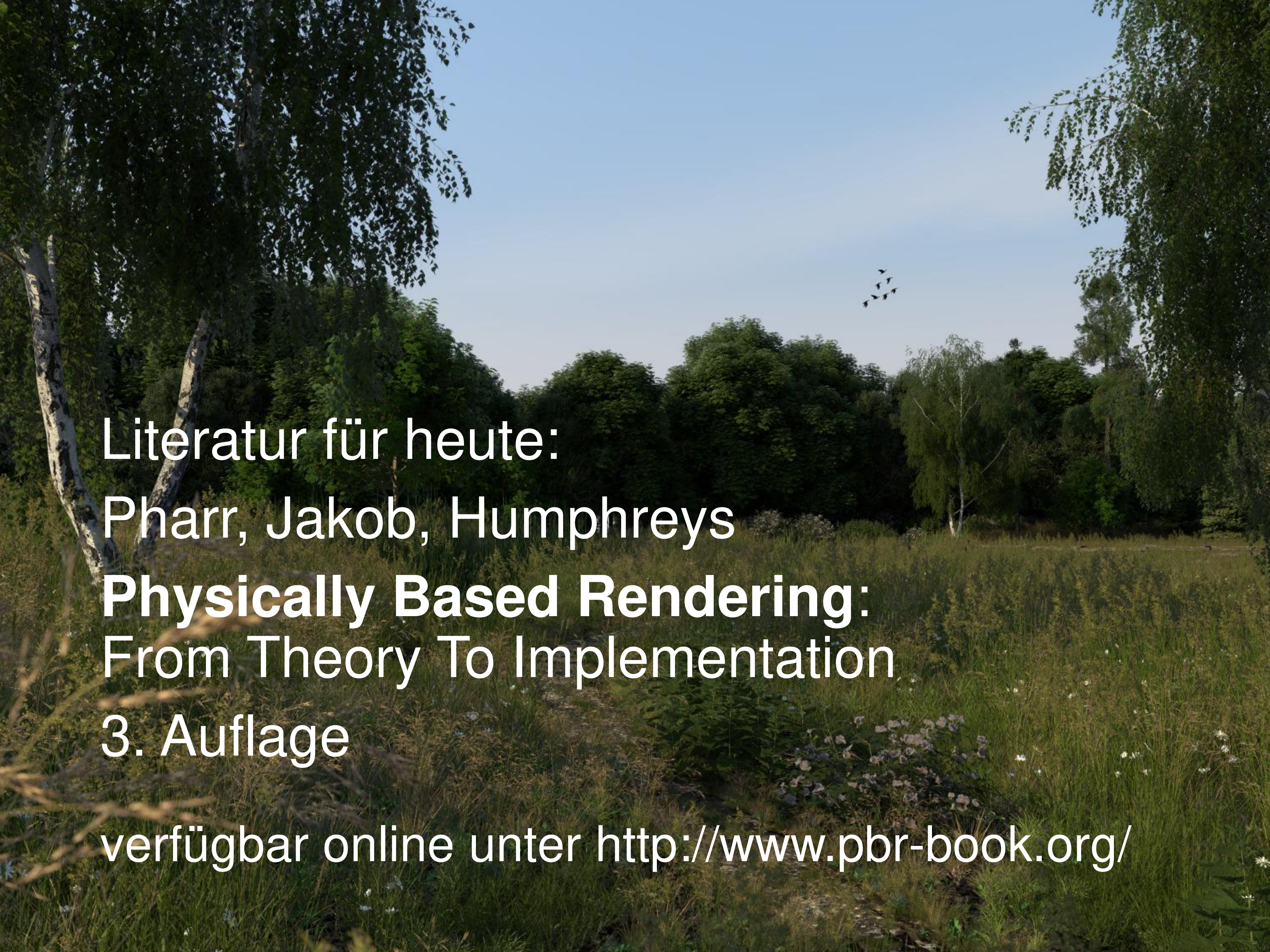


[Kajiya 1986] [Immel et al. 1986]

Globale Beleuchtung



- Angeleuchtete Flächen werden selbst zu Lichtquellen



Literatur für heute:

Pharr, Jakob, Humphreys

Physically Based Rendering:
From Theory To Implementation

3. Auflage

verfügbar online unter http://www.pbr-book.org/

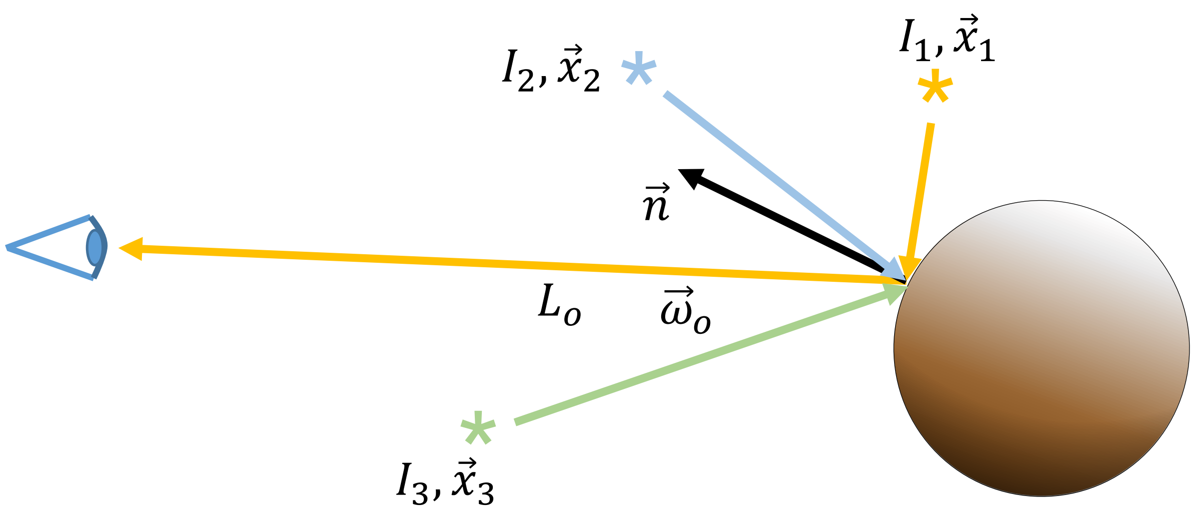
Einfachster Fall: direkte Beleuchtung (Punktquellen)

$$L_o(\vec{x},\vec{\omega}_o) = \cancel{L_e(\vec{x},\vec{\omega}_o)} + \int_\Omega f(\vec{\omega}_i,\vec{\omega}_o)\underbrace{L_i(\vec{x},\vec{\omega}_i)}_{\text{einfallende Radianz}}\langle\vec{\omega}_i,\vec{n}\rangle d\vec{\omega}_i$$

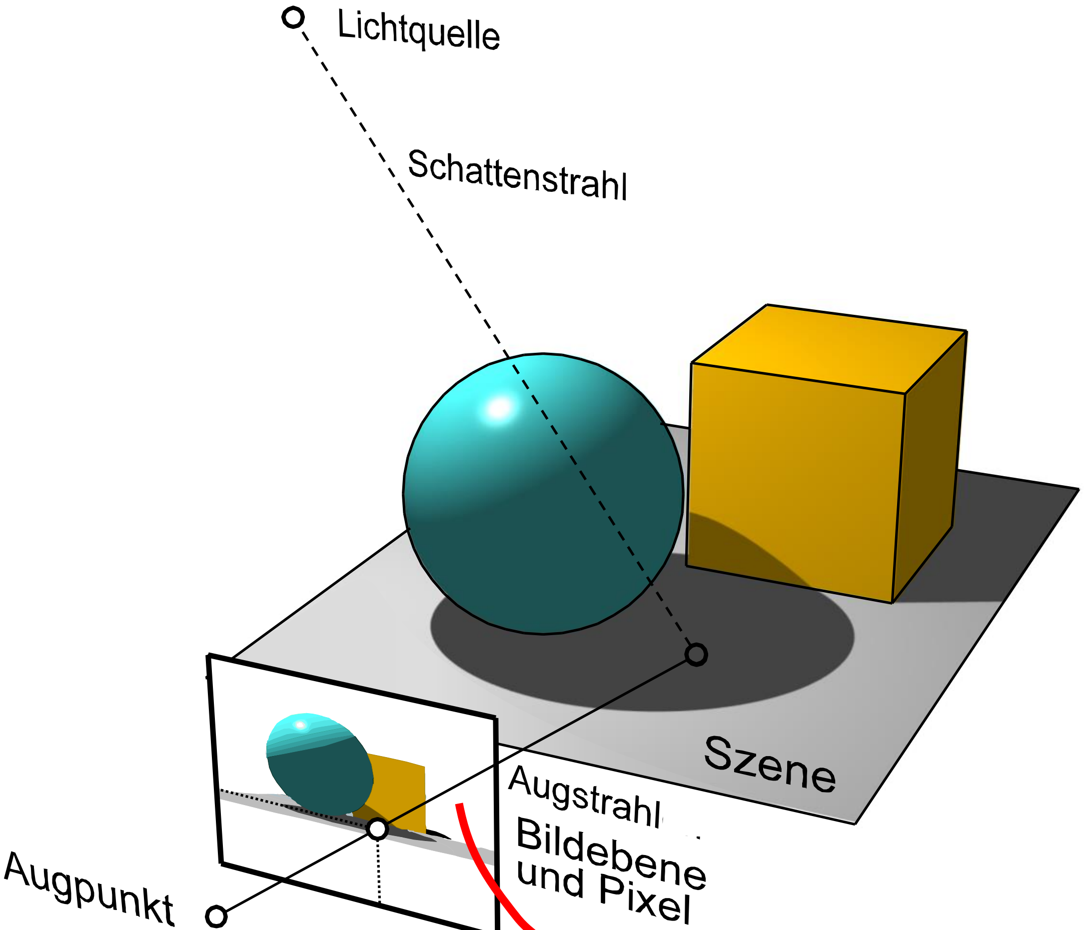
Intensität Sichtbarkeit

$$L_o(\vec{x},\vec{\omega}_o) = \sum_k f(\vec{\omega}_k,\vec{\omega}_o)\frac{I_k}{(\vec{x}_k - \vec{x})^2} V(\vec{x},\vec{x}_k)\langle\vec{\omega}_k,\vec{n}\rangle$$

$1/R^2$-Term



Raytracing nach [Appel 1968]



Welche Strahlen
brauche ich?

Zutaten:

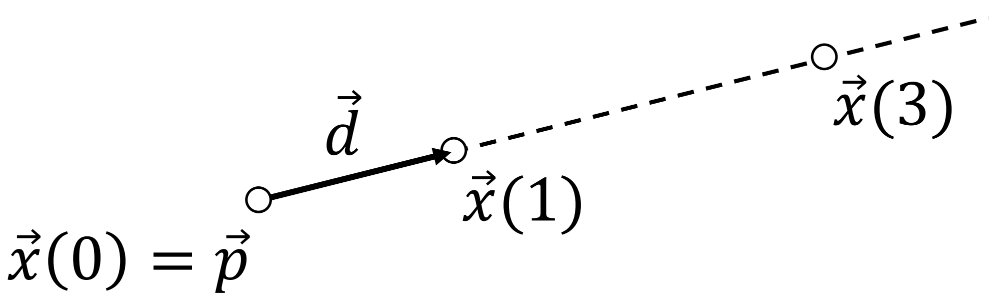
1. Strahlerzeugung
2. Schnittpunkt-
berechnung
3. Schattierung

1. Strahlerzeugung

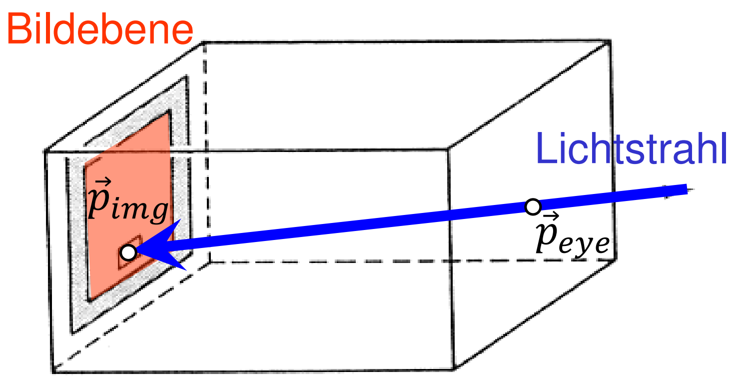
- Ein Strahl ist gegeben durch seinen Startpunkt (*position*, $\vec{p}$) und seine Richtung (*direction*, $\vec{d}$). Ein Strahlparameter t erzeugt Punkte $\vec{x}(t)$ auf dem Strahl:

$$\vec{x}(t) = \vec{p} + t\vec{d}, \qquad \vec{p}, \vec{d} \in \mathbb{R}^3; t \in \mathbb{R}_{\geq 0}$$

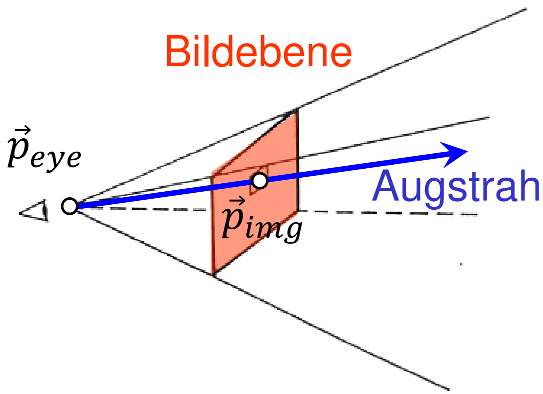
- Der Strahlparameter t beschreibt den Abstand zwischen $\vec{p}$ und $\vec{x}$ in Vielfachen von $\vec{d}$ (meist: $|\vec{d}| = 1$)



1. Strahlerzeugung



Lochkamera
Bild hinter Augpunkt, kopfstehend



Raytracing
Bild vor Augpunkt, aufrecht

- Für jeden Pixel (u, v):
  - Bestimme Punkt in der Bildebene, $\vec{p}_{img}(u, v)$
  - Für $\vec{d}$, verbinde Augpunkt $\vec{p}_{eye}$ mit Bildpunkt $\vec{p}_{img}$:

$$\vec{d} = \frac{\vec{p}_{img} - \vec{p}_{eye}}{\left|\vec{p}_{img} - \vec{p}_{eye}\right|}$$

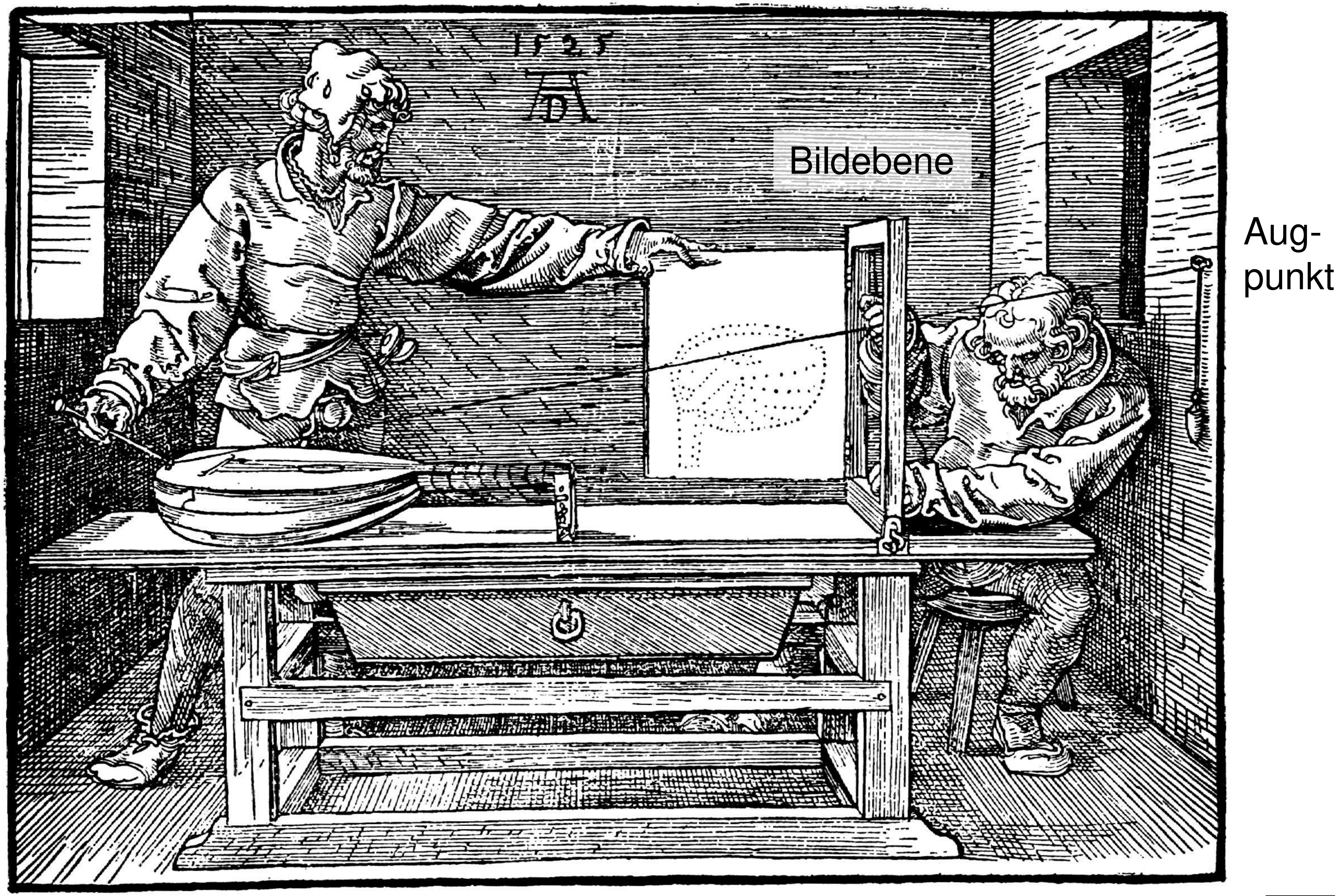
  - Strahl: $\vec{x} = \vec{p}_{eye} + t\vec{d}$

Kameraparameter

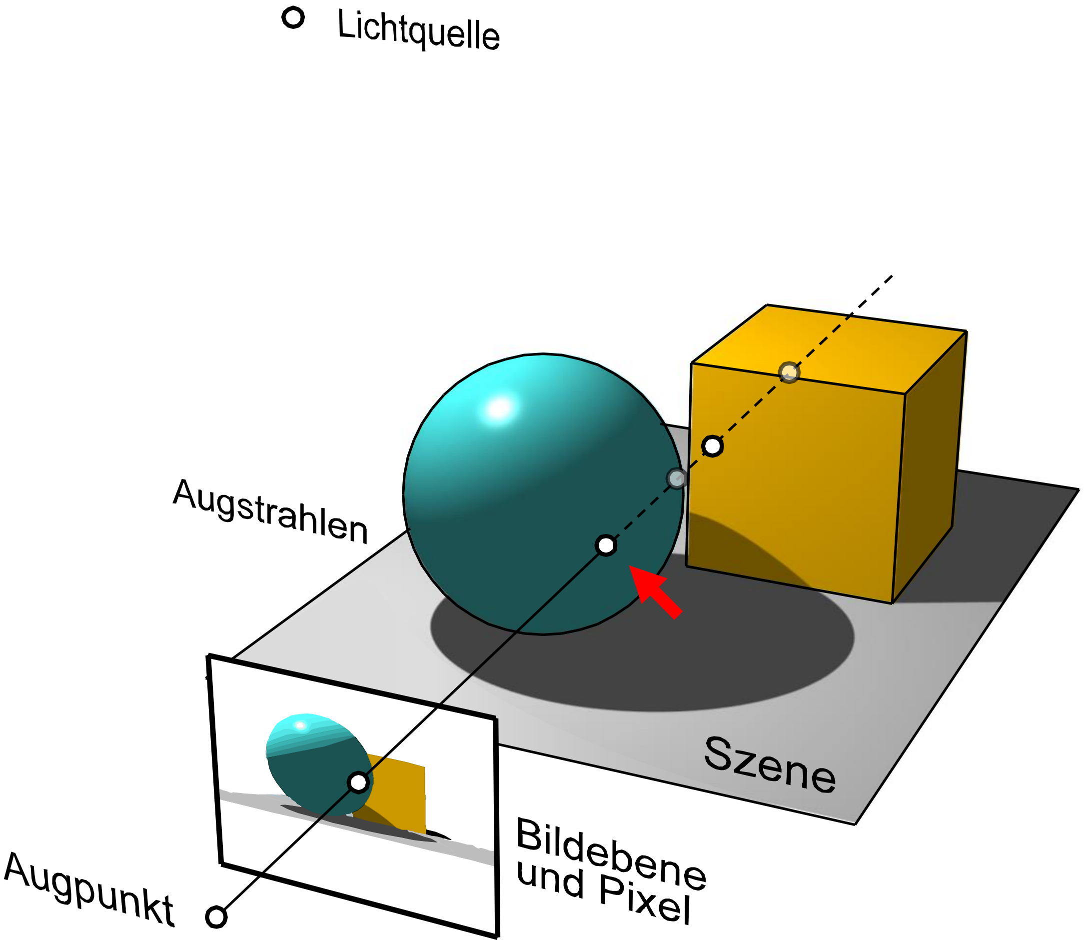
- $\vec{o}$ Kameraposition
- $\vec{f}$ „Brennweite“ (Vektor von $\vec{o}$ zur Mitte der Bildebene)
- $\overrightarrow{\mathrm{up}}$ „Up-vector“ der Kameraorientierung
- $\vec{x}$, $\vec{y}$ spannen die Bildebene auf
- xres, yres Bildauflösung

```
for (yi = 0, yi < yres, ++yi)
        for (xi = 0, xi < xres, ++xi)
        {
                d⃗ = f⃗ + 2.0 * (xi / (xres – 1.0) - 0.5) * x⃗
                      + 2.0 * (yi / (yres – 1.0) - 0.5) * y⃗;
                d⃗ = d⃗ / |d⃗|;
                trace_ray(o⃗, d⃗); // usw.
        }
```

Mechanisches Raytracing vor 500 Jahren



Albrecht Dürer: Der Zeichner der Laute (1525)



Zutaten:

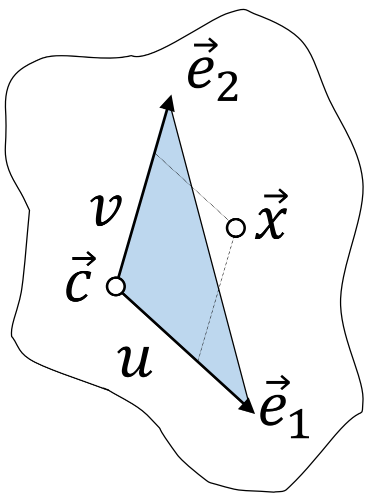
1. Strahlerzeugung
2. Schnittpunkt-berechnung
3. Schattierung

2. Schnittpunktberechnung

Gesucht: Schnittpunkt zw. Strahl $\vec{x} = \vec{p} + t\vec{d}$ und Objekt

Objektgeometrie kann auf verschiedene Weise definiert sein:

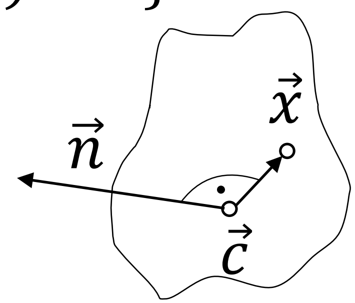
explizit: Funktion $f: \mathbb{R}^2 \to \mathbb{R}^3$ erzeugt Punkte auf der Oberfläche $\vec{x} = f(u, v)$



z.B.:
Ebene $f_1(u, v)$
$= \vec{c} + u\vec{e}_1 + v\vec{e}_2$

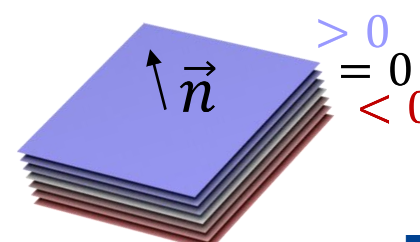
Dreieck
$u > 0; v > 0; u + v \leq 1$

implizit: Objektoberfläche als Isofläche einer Funktion $g: \mathbb{R}^3 \to \mathbb{R}$: $\{\vec{x} | g(\vec{x}) = 0\}$



z.B.:
Ebene $g_1(\vec{x}) = \vec{n} \cdot (\vec{x} - \vec{c})$
$= \vec{n} \cdot \vec{x} - d$

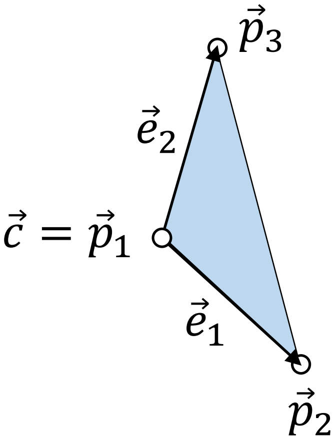
Isoflächen
für verschiedene
Werte von $g(x)$



Schnitt Strahl-Dreieck (ad-hoc Skizze)

Gegeben: Eckpunkte $\vec{p}_1, \vec{p}_2, \vec{p}_3$, Strahl $\vec{x} = \vec{p} + t\vec{d}$

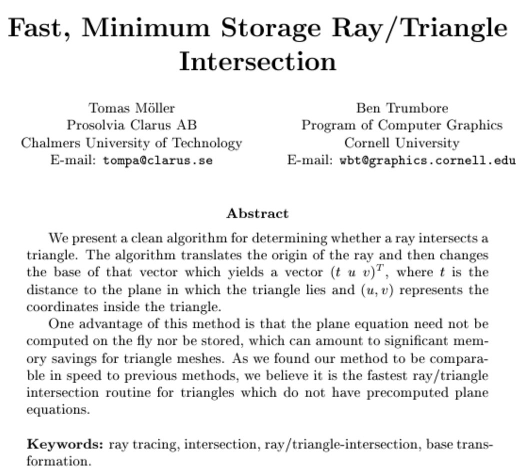
1. Berechne Aufpunkt $\vec{c}$ und Normale $\vec{n}$
   $\vec{c} = \vec{p}_1$
   $\vec{n} = \vec{e}_1 \times \vec{e}_2 = (\vec{p}_2 - \vec{p}_1) \times (\vec{p}_3 - \vec{p}_1)$



2. Schneide Strahl mit Ebene
   $\vec{n} \cdot \left(\vec{p} + t\vec{d} - \vec{c}\right) = 0$
   $\Leftrightarrow t = \frac{\vec{n} \cdot (\vec{c} - \vec{p})}{\vec{n} \cdot \vec{d}}$

3. Löse Gleichungsystem für u, v
   $\vec{c} + u\vec{e}_1 + v\vec{e}_2 = \vec{p} + t\vec{d}$

4. Teste ob $u > 0, v > 0, u + v < 1$



Fast, Minimum Storage Ray/Triangle Intersection

Tomas Möller
Prosolvia Clarus AB
Chalmers University of Technology
E-mail: tompa@clarus.se

Ben Trumbore
Program of Computer Graphics
Cornell University
E-mail: wbt@graphics.cornell.edu

Abstract

We present a clean algorithm for determining whether a ray intersects a triangle. The algorithm translates the origin of the ray and then changes the base of that vector which yields a vector $(t\ u\ v)^T$, where t is the distance to the plane in which the triangle lies and (u, v) represents the coordinates inside the triangle.

One advantage of this method is that the plane equation need not be computed on the fly nor be stored, which can amount to significant memory savings for triangle meshes. As we found our method to be comparable in speed to previous methods, we believe it is the fastest ray/triangle intersection routine for triangles which do not have precomputed plane equations.

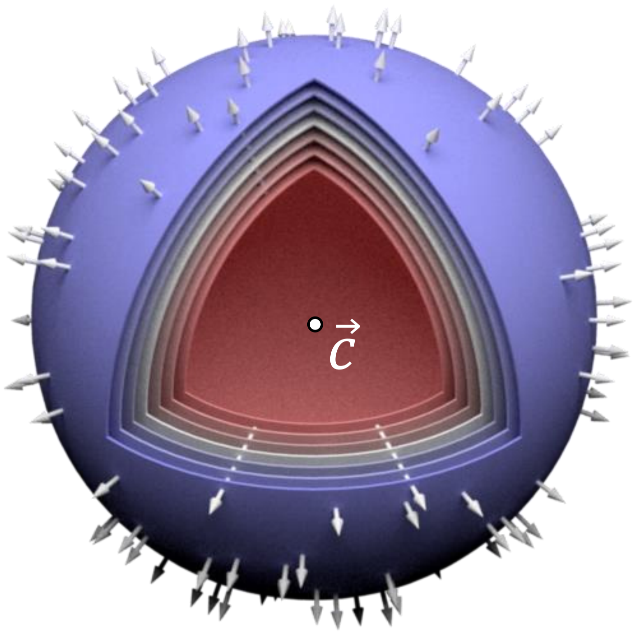
Keywords: ray tracing, intersection, ray/triangle-intersection, base transformation.

Besser: Möller und Trumbore, 1997.

2. Schnittpunktberechnung - Nachlese

- Echter Schnittpunkt nur für $t > 0$, besser $t > \epsilon$
- Von allen Schnittpunkten, wähle den mit kleinstem t (nächstgelegener Punkt entlang Strahl)
- Berechne Normalenvektor $\vec{n}$ im Schnittpunkt $\vec{x}$
  - Ebene/Dreieck: bereits vorhanden
  - bei impliziter Geometrie allgemein: Richtung des Gradienten

$$\vec{n} = \vec{\nabla} g(\vec{x}) = \begin{pmatrix} \partial/\partial x \\ \partial/\partial y \\ \partial/\partial z \end{pmatrix} g(\vec{x})$$

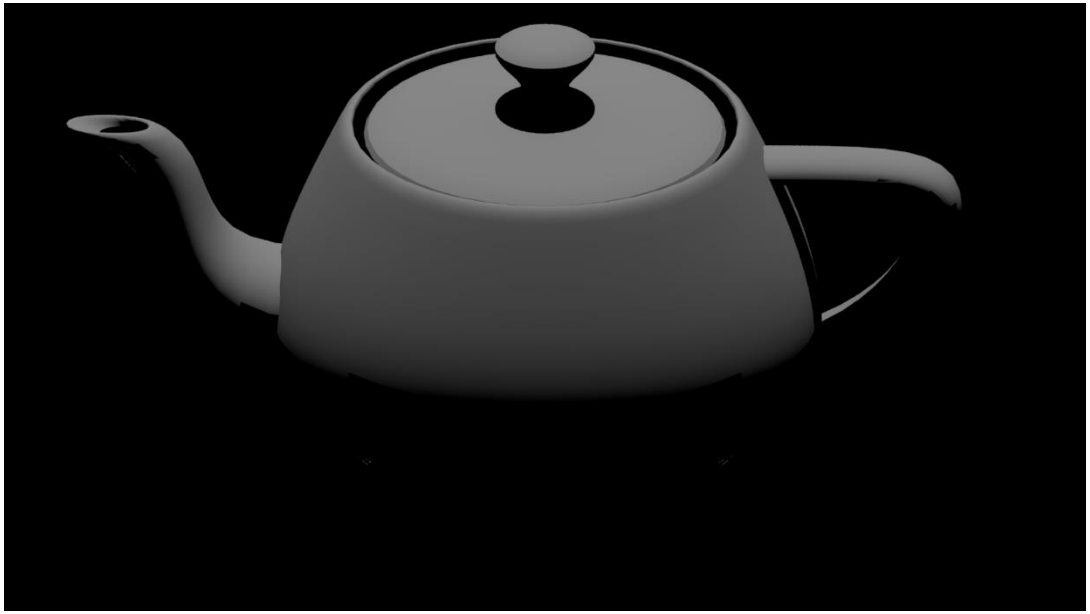


Kugel:

$$g(\vec{x}) = |\vec{x} - \vec{c}| - R$$
$$\vec{\nabla} g(\vec{x}) = \vec{x} - \vec{c}$$

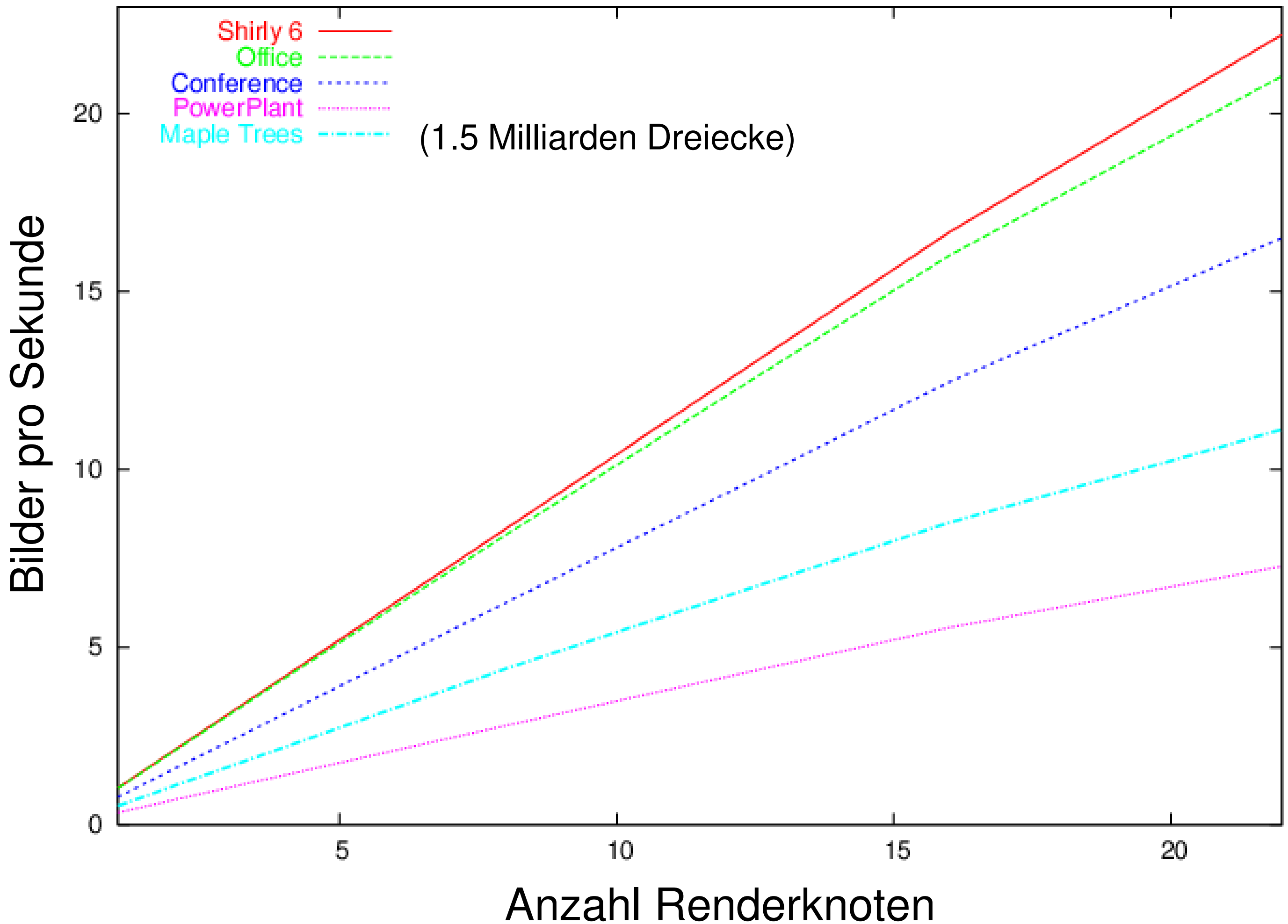
Kosten von naivem Raytracing

- (Mindestens) lineare Komplexität
- $1920 \times 1080 \times 6320$ Dreiecke! (plus Schattentests usw.)



Trotzdem kann man extrem komplexe Szenen rendern.
(Diese Szene mit 1.5 Milliarden Dreiecken war bereits 2007 in Echtzeit darstellbar). Wie geht das?

Raytracing skaliert gut auf parallelen Systemen



universitätbonn

Theoretischer Hintergrund

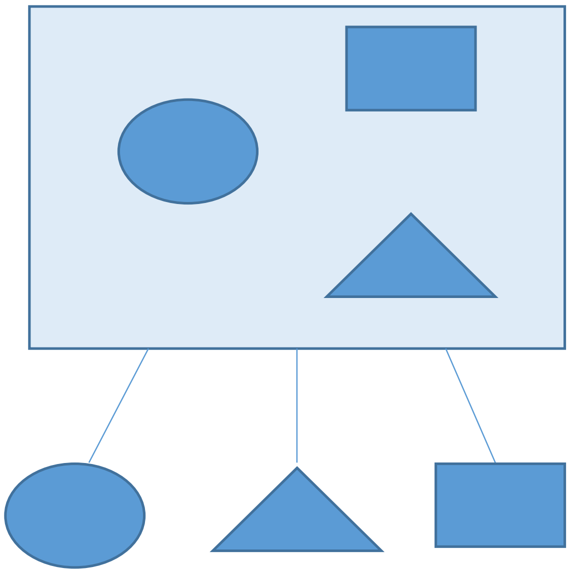
- Unsortierte Daten: mindestens lineare Laufzeit
  - Jedes Primitive könnte den ersten Schnittpunkt enthalten
  - Jedes Primitive muss einzeln getestet werden
  - Speicherkohärenz hilft nicht
- Komplexität nur durch Vorsortieren der Daten erreichbar
  - Räumliches Sortieren der Geometrie (Indizierung wie in einer Datenbank) ermöglicht effiziente Suchstrategien
  - Hierarchische Strukturen ermöglichen Suchkomplexität $O(\log n)$
  - Bei dynamischen Szenen: Kompromiss zwischen Laufzeit und Zeit für Aufbau der Suchstruktur
  - Worst case immer noch linear
- Allgemeines Problem in der Grafik
  - Räumliche Indices für Raytracing
  - Räumliche Indices für Occlusion und Frustum Culling
  - Sortieren für Transparenz

Raytracing beschleunigen

- Schneide Strahl mit allen Objekten
  - Viel zu teuer
- Schnellere Schnittberechnung
  - Immer noch $O(n)$
- Weniger Schnittberechnungen
  - Raumpartitionierung (space partitioning), oft hierarchisch
    - Gitter, Hierarchien von Gittern
    - Octrees
    - Binary Space Partition oder kd-Baum
    - Bounding Volume Hierarchy (BVH)
  - Andere Ansätze denkbar, aber wenig verbreitet
- Strahlenbündel verfolgen
  - Benachbarte Strahlen schneiden wahrscheinlich dasselbe Objekt

Sammelobjekte

- Objekt, das Gruppe von Objekten enthält
- Speichert Hüllkörper (Bounding Box) und Pointer auf Kinder
- Nützlich für Instantiierung und Hüllkörperhierarchien (Bounding Volume Hierarchies)

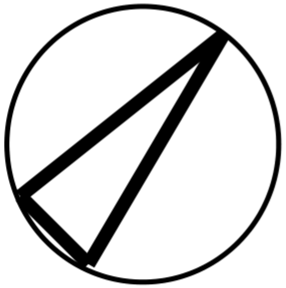


Bounding Volumes (BV)

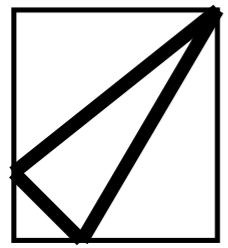
- Objektgeometrie vollständig von BV umschlossen
  - Genaue Schnittberechnung nur nötig, wenn Strahl BV trifft

Bounding Volumes (BV)

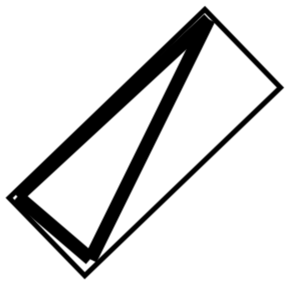
- Objektgeometrie vollständig von BV umschlossen
  - Genaue Schnittberechnung nur nötig, wenn Strahl BV trifft
- Kugel
  - Schnelle Schnittberechnung, aber meist zu groß



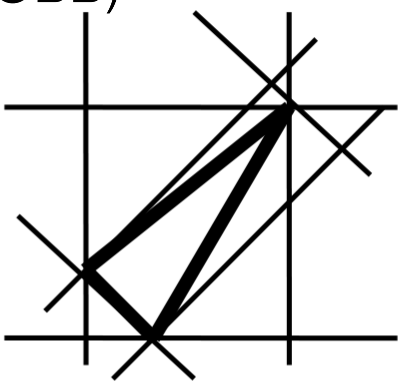
- Axis-aligned bounding box (AABB)
  - Sehr einfache Schnittberechnung
  - Manchmal zu groß



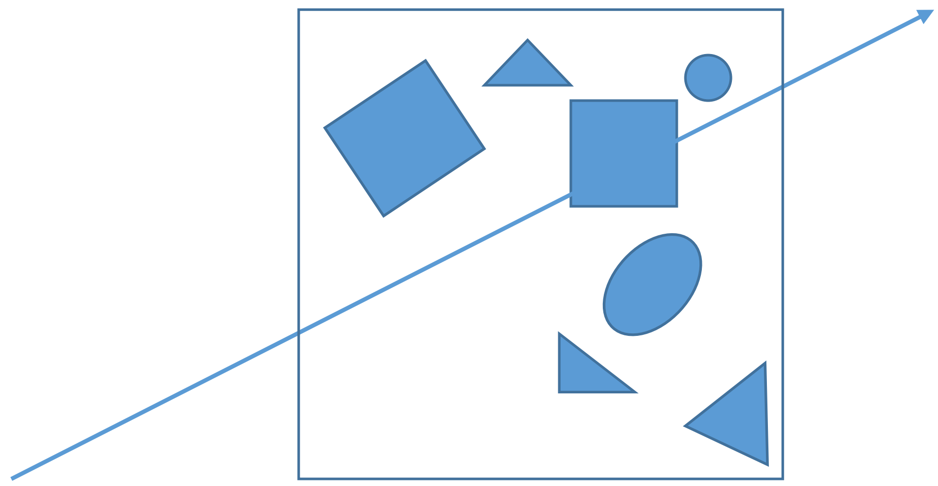
- Object-aligned / oriented bounding box (OBB)
  - Passt oft besser
  - Einigermaßen schwer zu berechnen



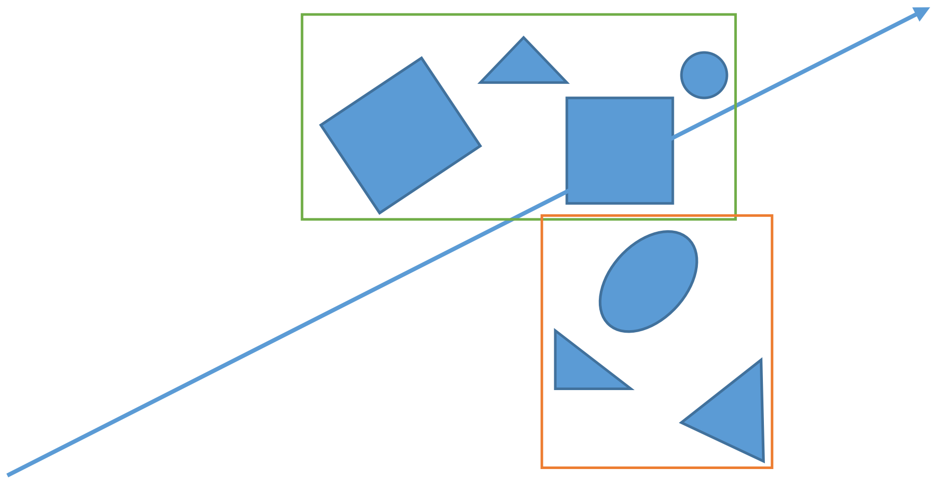
- Slabs („Scheiben“)
  - Paare von Halbräumen
  - Feste Zahl von Orientierungen
  - Einigermaßen effiziente Berechnung



Bounding Volume Hierarchy (BVH)

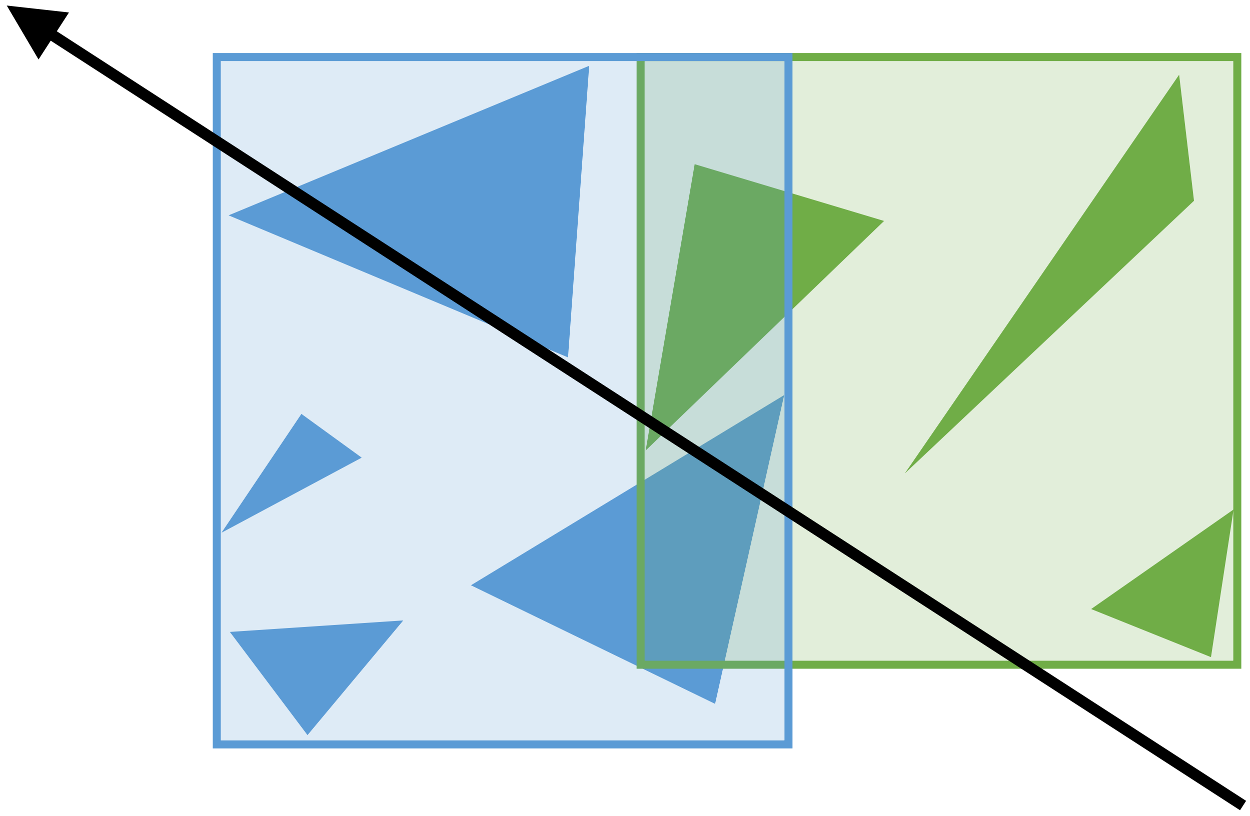


Bounding Volume Hierarchy (BVH)



BVH Traversierung

- Schnittpunkt nicht sofort zurückgeben – eine andere Bounding Box könnte noch einen näheren enthalten!



Bounding Volume Hierarchy (BVH)

Baumstruktur

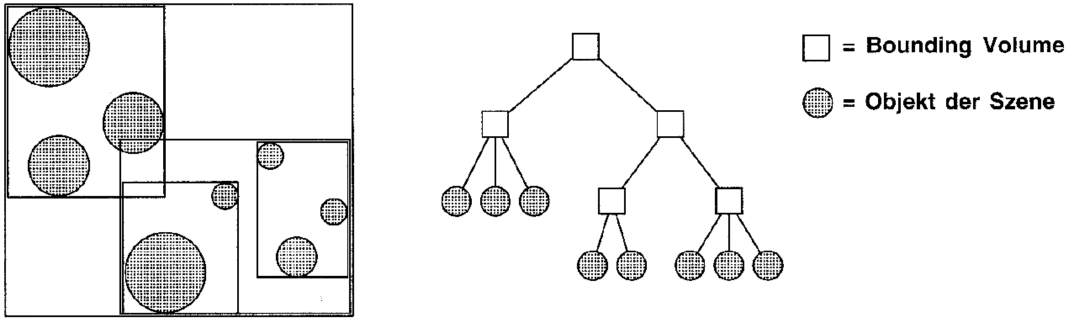
- Innere Knoten sind Sammelobjekte
- Blätter sind geometrische Objekte
- Schnitttest durch Traversierung

Merkmale

- Sehr gute Adaptivität
- Effiziente Suche ($O(\log n)$)
- Oft in Raytracern verwendet

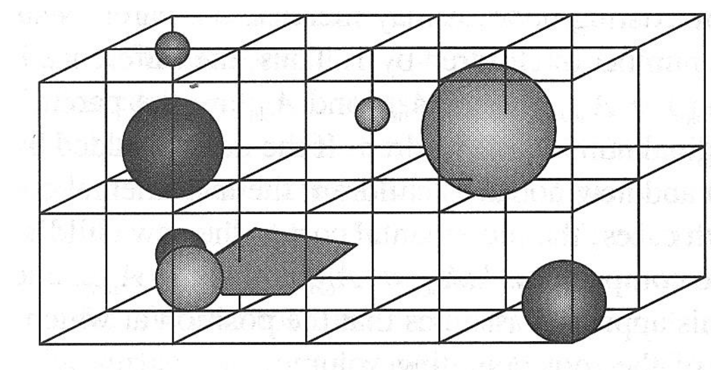
Problem

- Wie aufbauen?



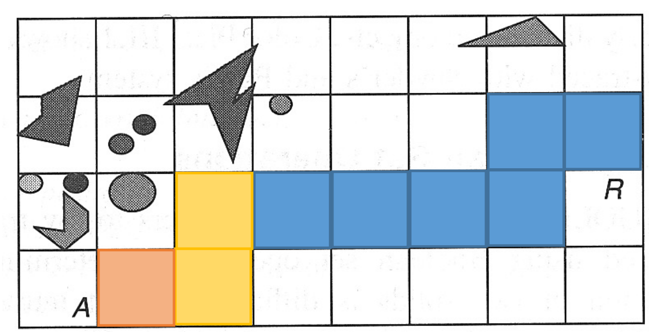
Gitter

Einteilung in „Voxel“ konstanter Größe



Gitterstruktur aufbauen

- Bounding Box (BB) unterteilen
- Auflösung: häufig $\sqrt[3]{n}$
- Objekte einfügen
  - Objekte, die mit mehreren BBs überlappen, werden dupliziert
  - Einfach zu optimieren



Traversierung

- Über Voxel iterieren, die vom Strahl geschnitten werden (in Reihenfolge der Durchdringung)
- Schnitt mit Objekten im Voxel berechnen
- Sobald Schnittpunkt gefunden: Abbruch

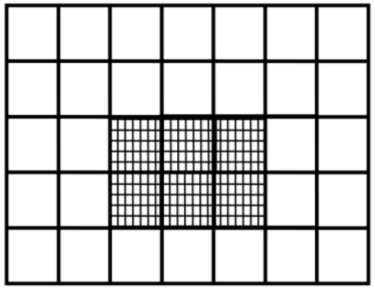
Gitter

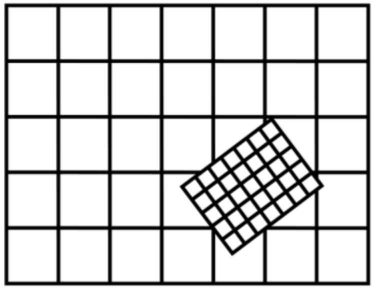
- Gittertraversierung
  - Erfordert Aufzählung der Voxel entlang Strahl
    - 3D-DDA, modifizierter Bresenham (später mehr)
  - Einfach und hardwarefreundlich
- Gitterauflösung
  - Stark szenenabhängig
  - Kann sich nicht an lokale Objektdichte anpassen
    - „Teapot in a stadium"
    - Lösungsansatz: „Gitter in Gittern" – hierarchische Gitter
- Objekte über mehrere Voxel
  - Speichere nur Referenzen
  - Verwende „mailboxing", um mehrere Schnittprüfungen zu vermeiden
    - Speichere Objekthashes in kleinem Cache pro Strahl
    - Falls im Cache gefunden, nicht erneut schneiden

Hierarchische Gitter

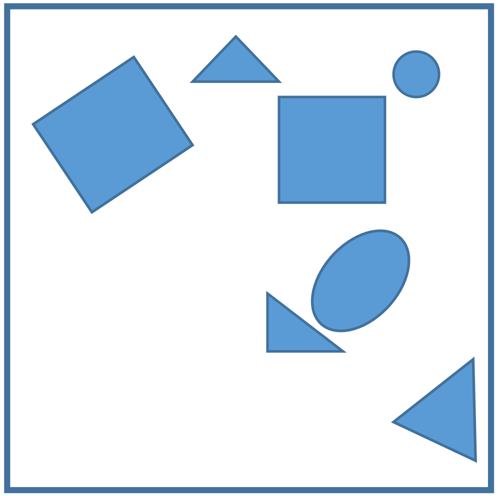
- Einfacher Konstruktionsalgorithmus
  - Grobes Gitter für gesamte Szene
  - Erzeuge rekursiv feinere Gitter in Voxeln hoher Dichte
  - Problem: welche Auflösung für welche Stufe?

- Fortgeschrittener Algorithmus
  - Platziere Gruppe von Objekten in eigenem Gitter
  - Füge Gitter in Elterngitter ein
  - Problem: was sind geeignete Gruppen/Cluster?

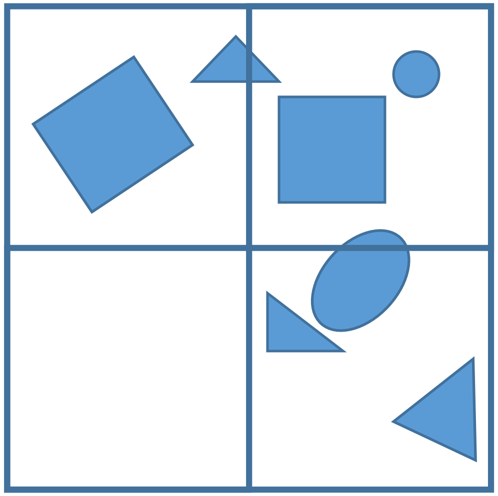




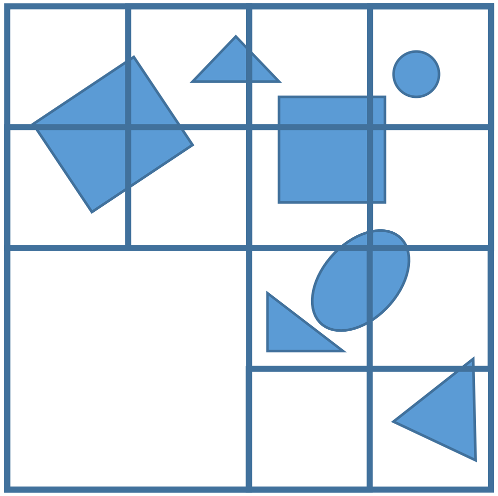
Quadtree – 2D Beispiel



Quadtree – 2D Beispiel

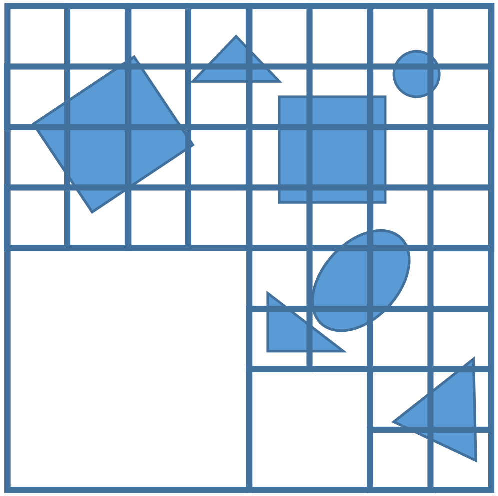


Quadtree – 2D Beispiel



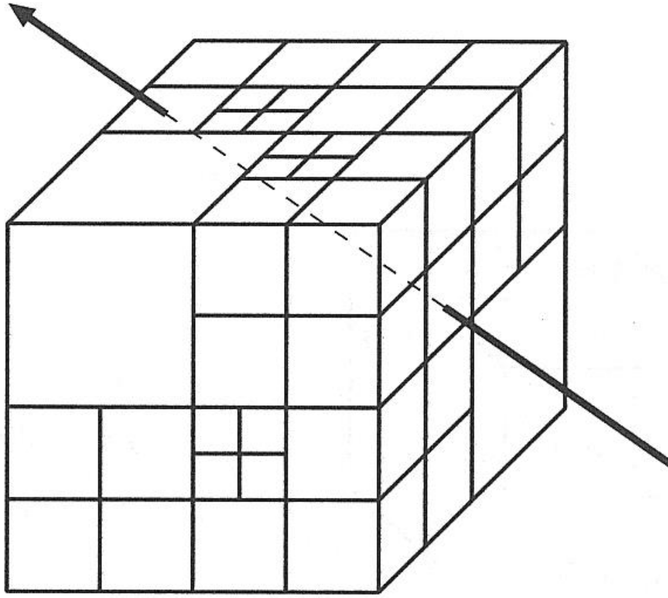
Quadtree – 2D Beispiel

- Hierarchische Unterteilung
- Teile, wenn Zelle nicht leer (oder wenn sie mehr als N Primitives enthält)



Octree

- Hierarchische Raumunterteilung
  - Starte mit BB der ganzen Szene
  - Unterteile Voxel rekursiv in 8 Sub-Voxel
  - Kriterien
    - Anzahl verbleibender Primitives und maximale Tiefe
  - Ergibt adaptive Unterteilung
    - Grobe Traversierungsschritte in leeren Regionen



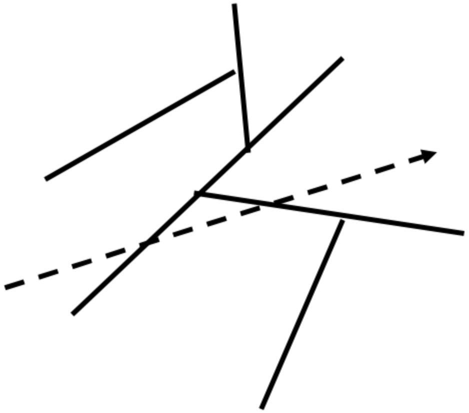
- Probleme
  - Traversierungsalgorithmen recht komplex
  - Verfeinerung komplexer Regionen langsam
- Traversierungsalgorithmen
  - HERO, SMART, ...
  - Oder kd-Baum-Algorithmus verwenden

BSP- und kd-Bäume

Rekursive Raumunterteilung mit Halbräumen

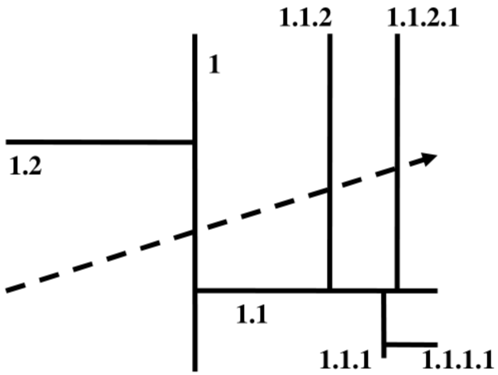
Binary Space Partition (BSP)

- Halbiere Raum rekursiv mit Ebenen in beliebiger Position
  - Oft durch vorhandene Polygone definiert
- Oft für Sichtbarkeitsabfragen in Spielen verwendet (Doom...)
  - Durchquere binären Baum von vorne nach hinten



kd-Baum

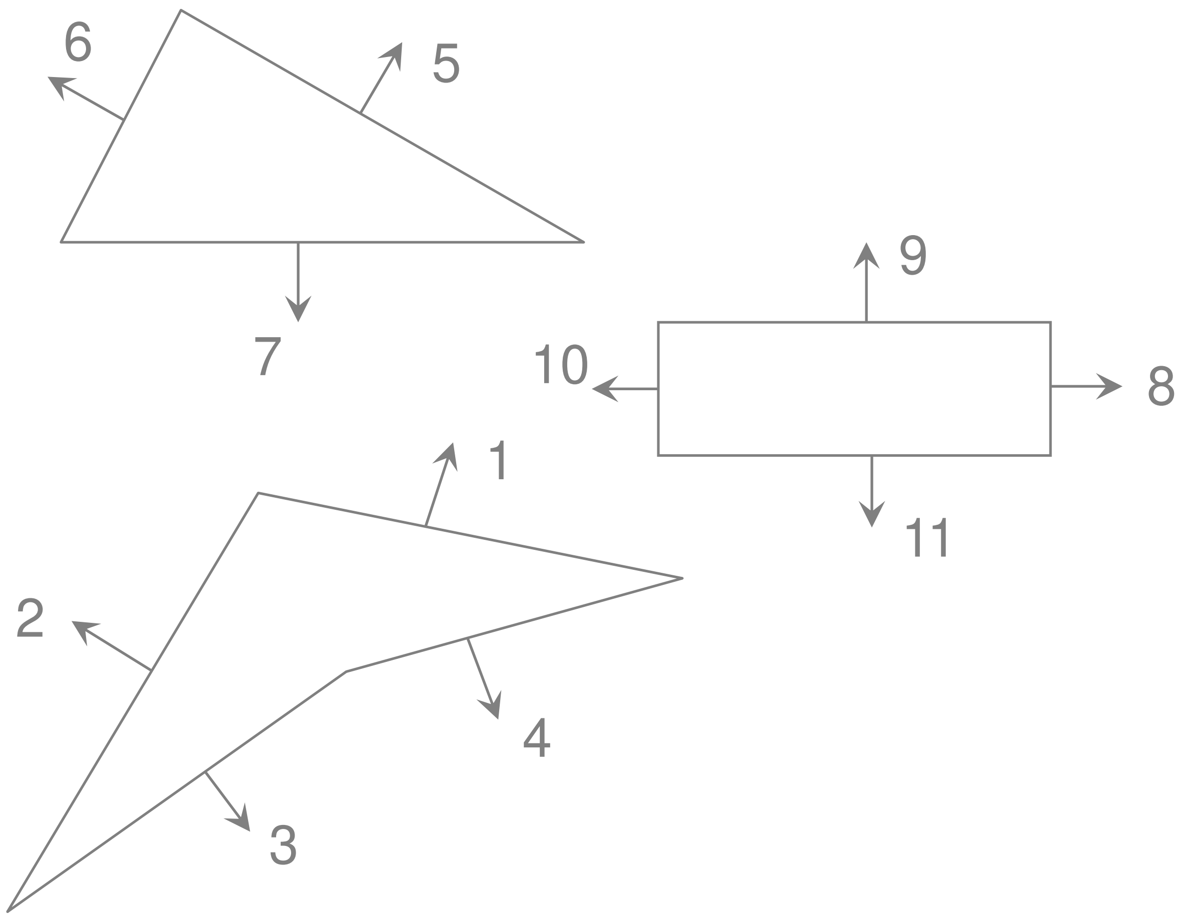
- Spezialfall von BSP
  - Teile mit achsenorientierten Ebenen
- Rekursiv definiert durch Knoten mit
  - Achsen-Flag (oft alternierend)
  - Split-Koordinate
  - Zeiger auf Kinder



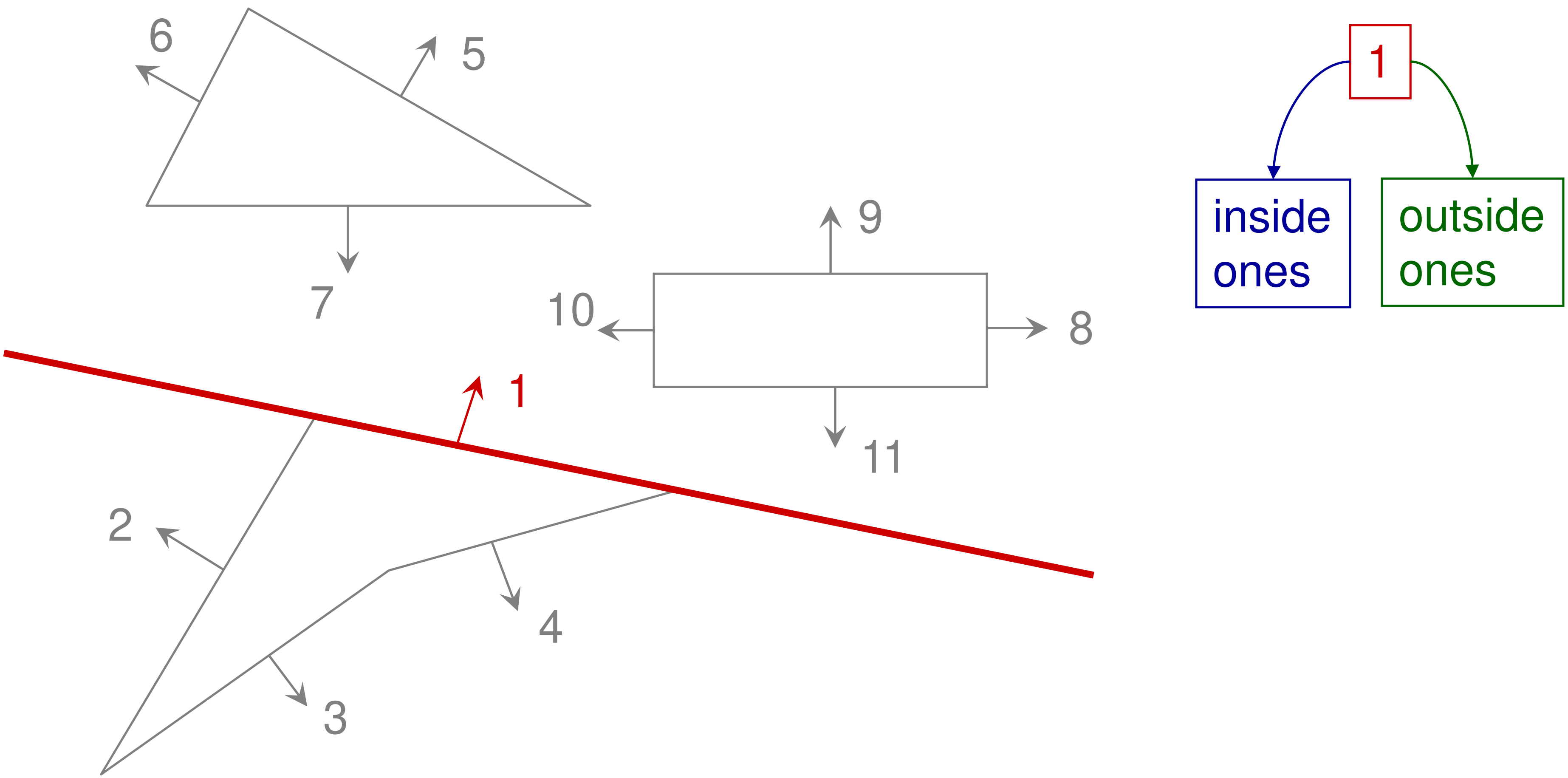
Traversierung von BSP- und kd-Bäumen

- „Front-to-back“ – traversiere Kinder gemäß Reihenfolge entlang Strahl
- Abbruch, sobald Schnittpunkt gefunden
- Verwende Stack von Unterbäumen
  - effizienter als Rekursion

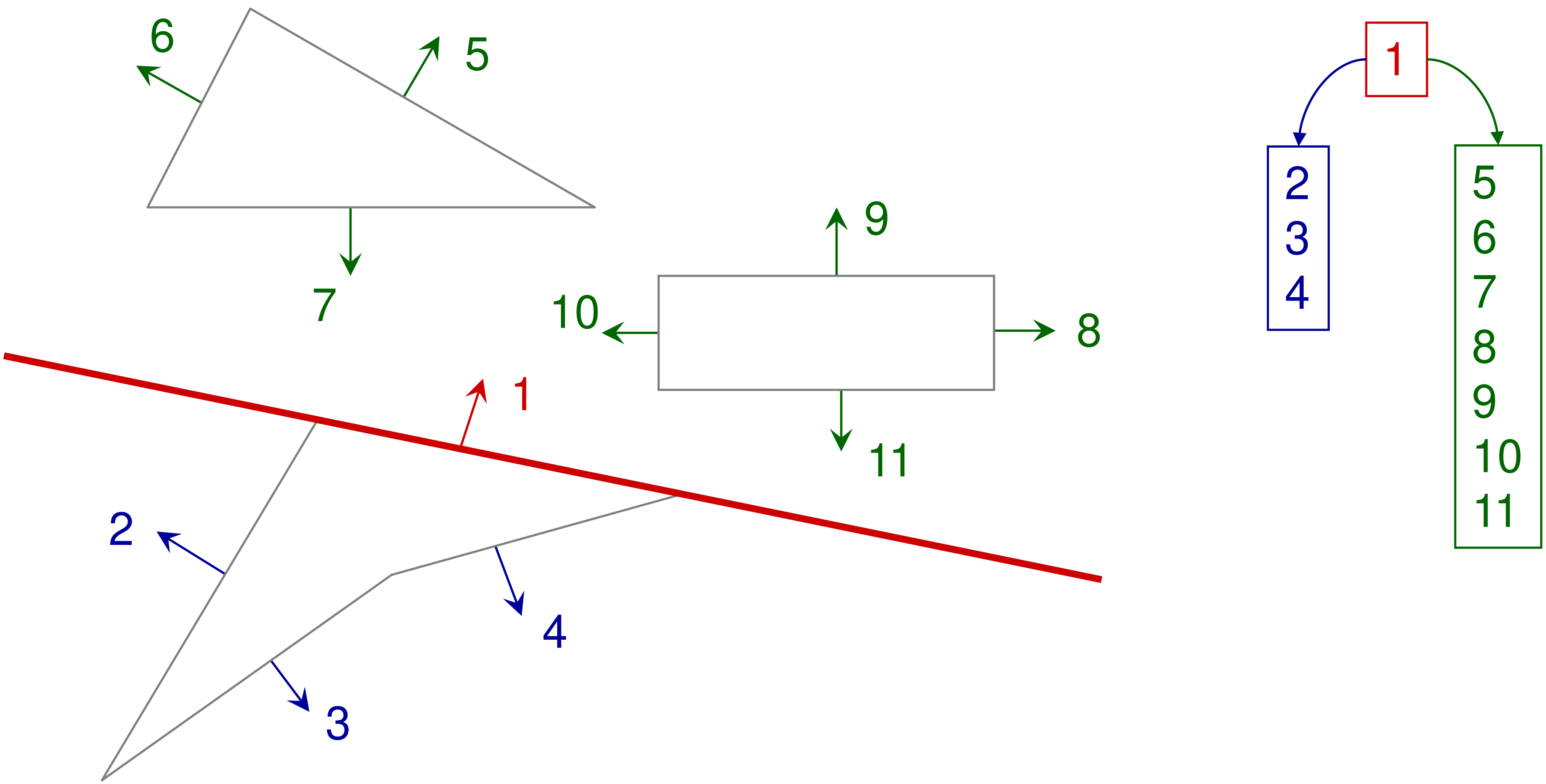
BSP tree



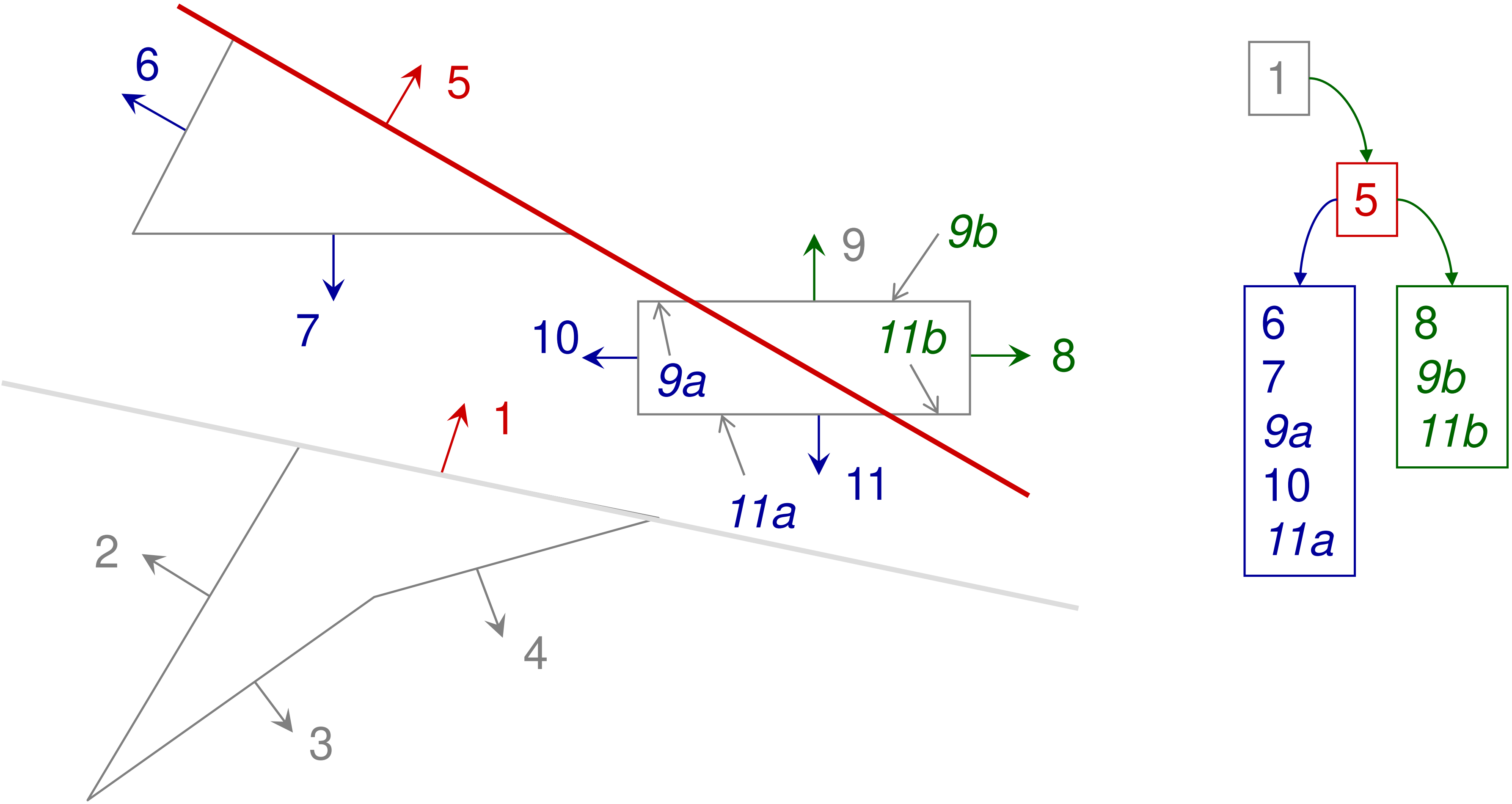
BSP tree



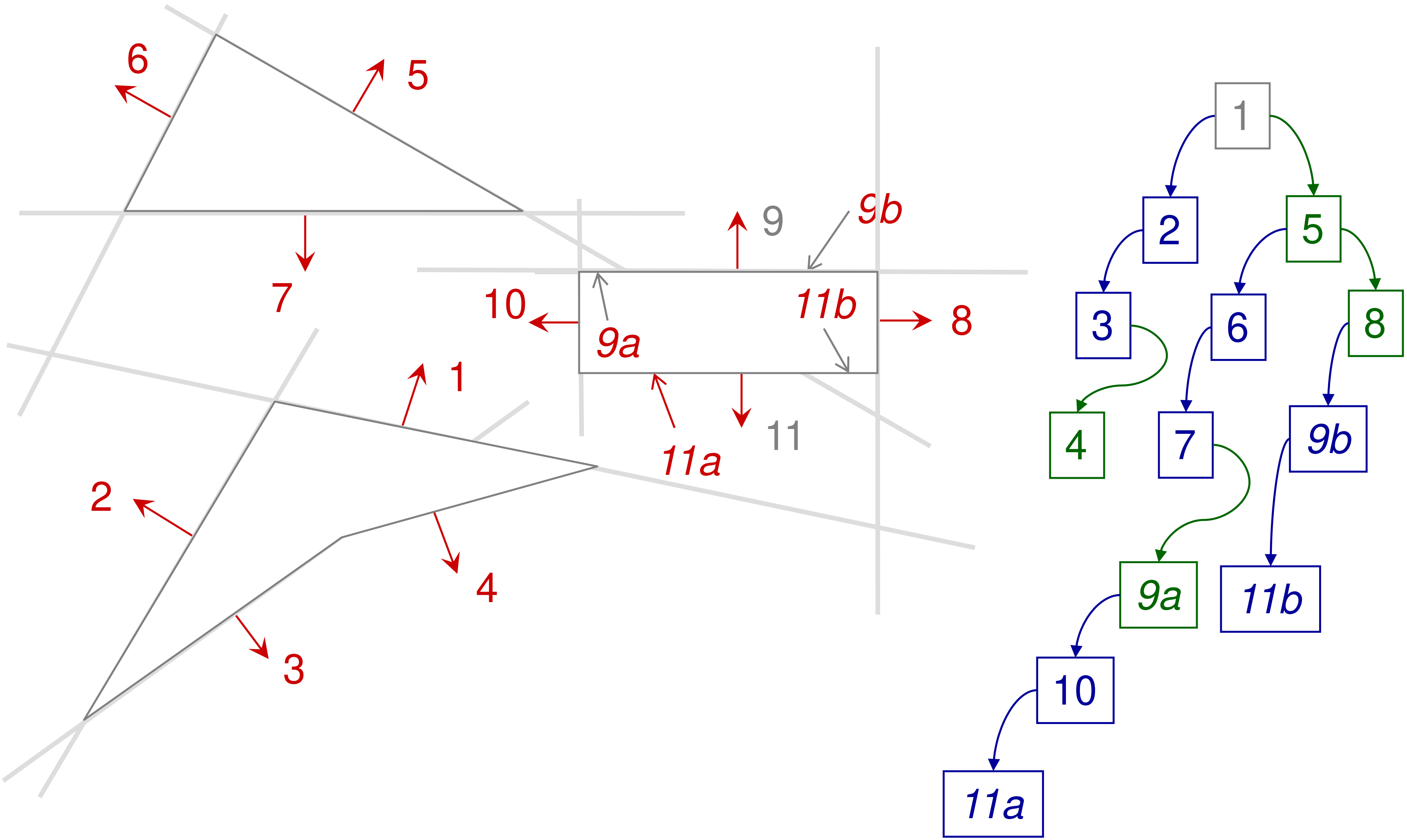
BSP tree



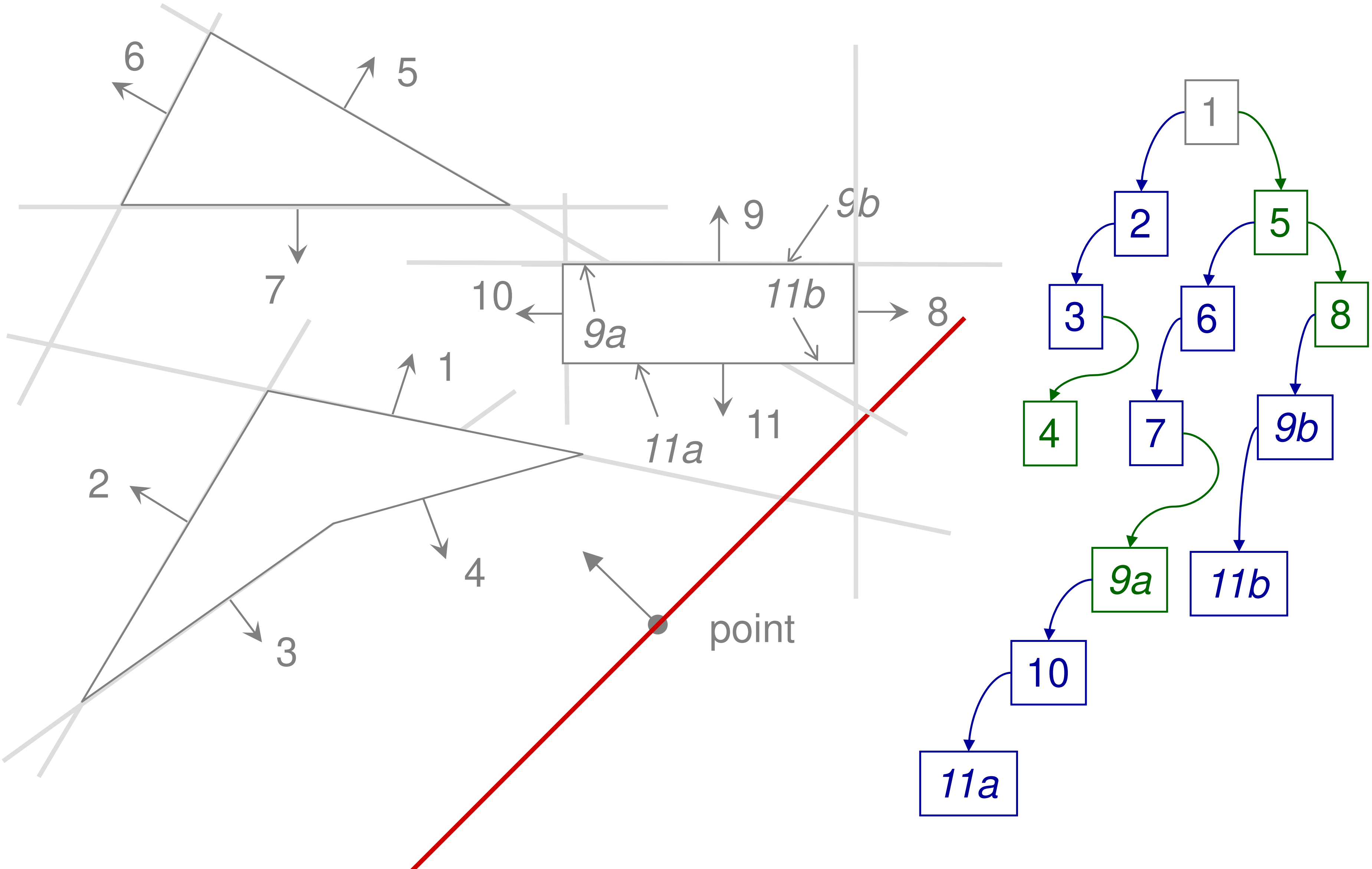
BSP tree



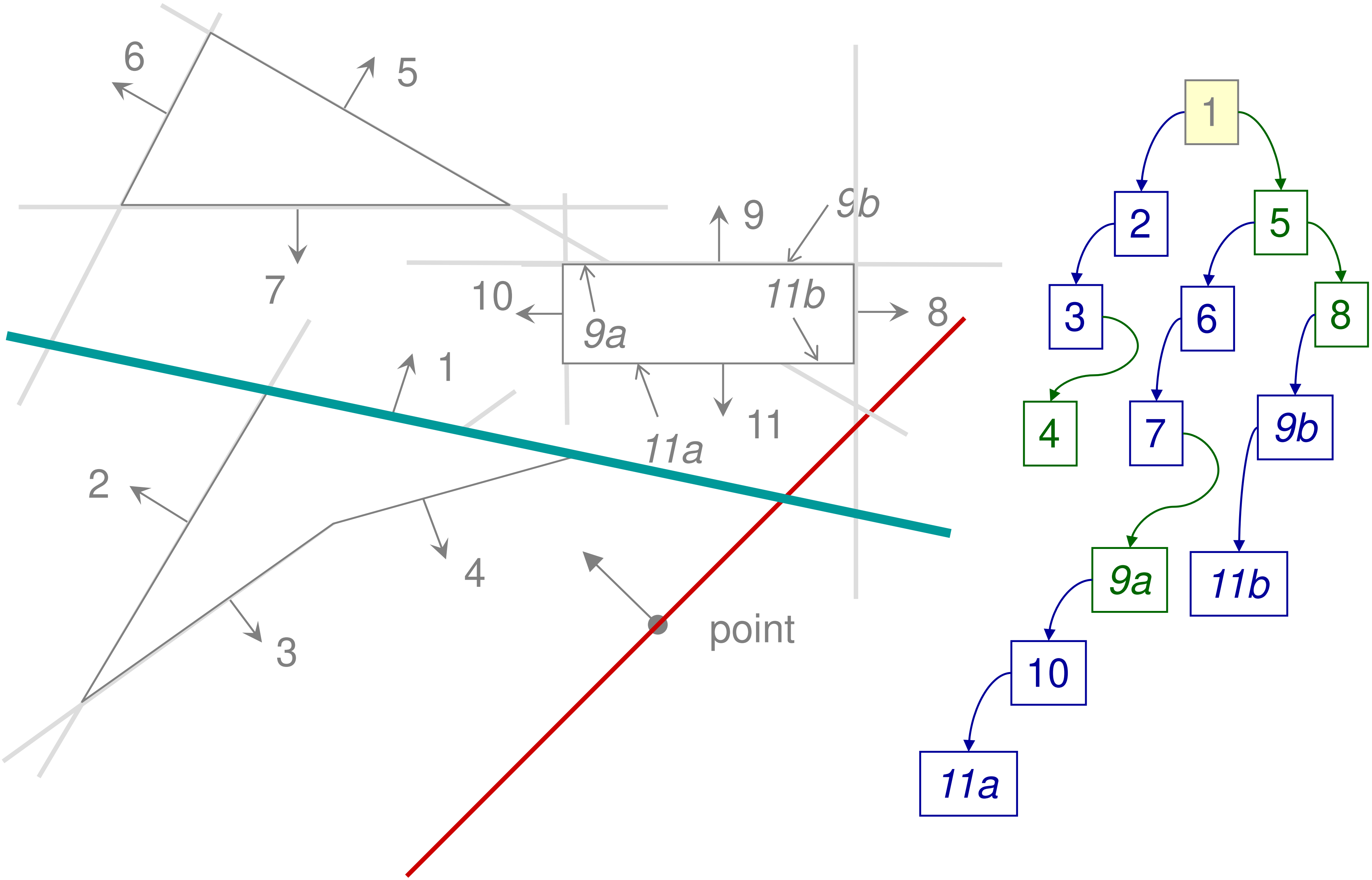
BSP tree



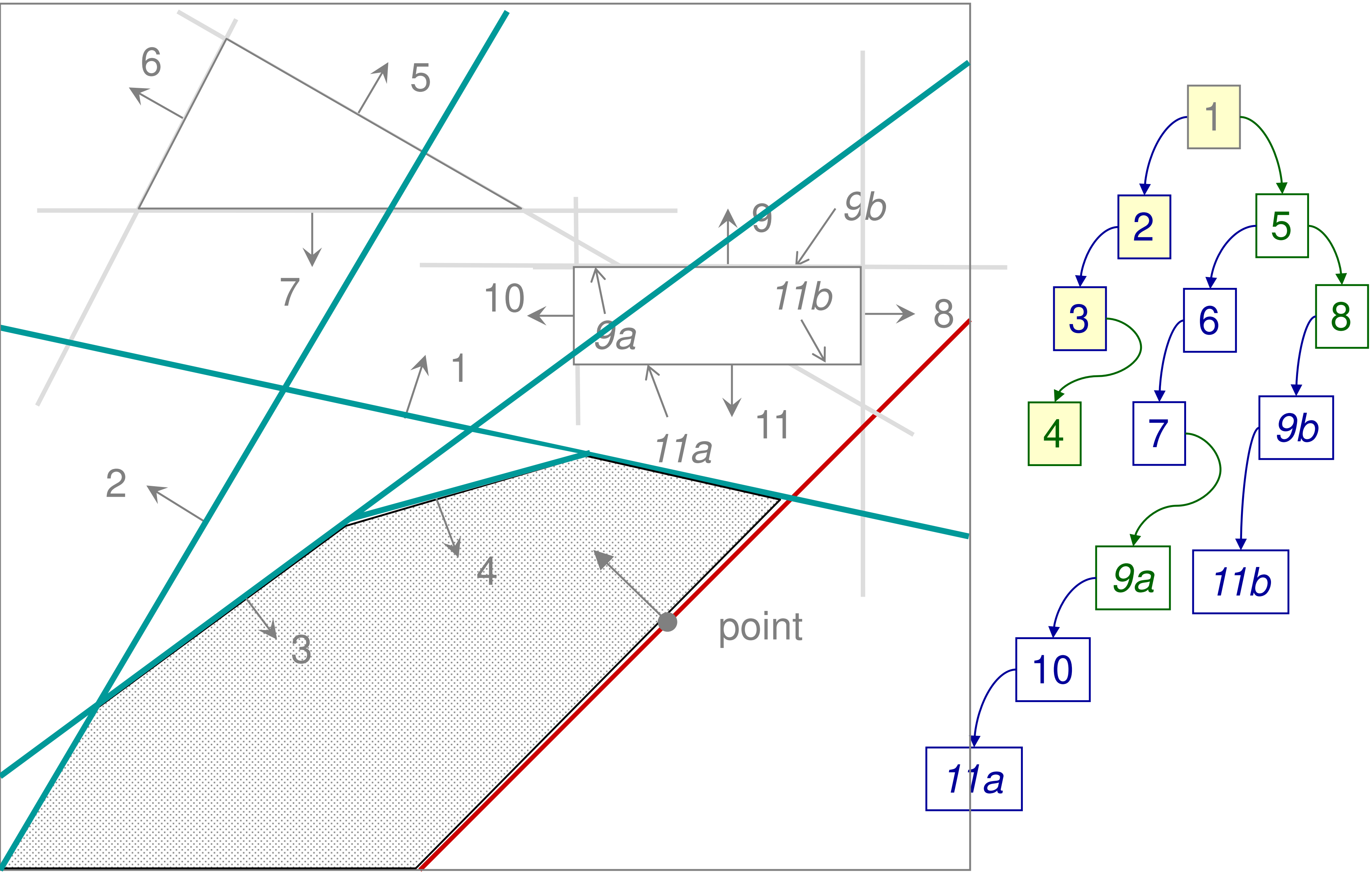
BSP tree traversal



BSP tree traversal

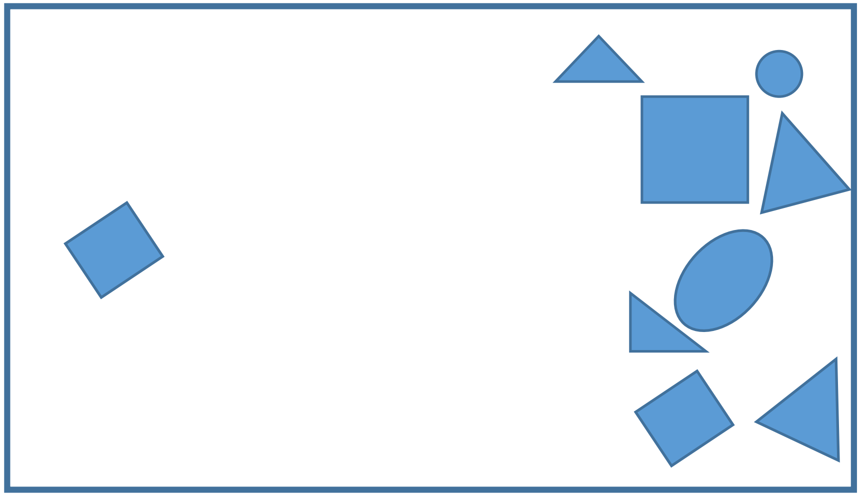


BSP tree traversal



kd-Baum aufbauen

Bewährt: Teile entlang X-Achse, dann Y-Achse, dann Z-Achse, und so weiter.

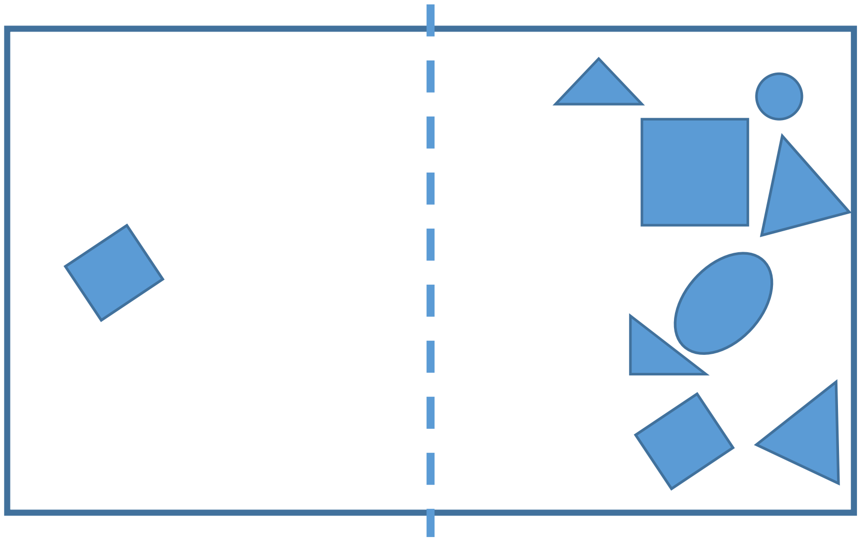


Aber wo den Schnitt platzieren?

Optimiere die zu erwartenden Kosten:

$$C(\text{Cell}) = C(\text{Trav}) + P(\text{hit}\,L) \cdot C(L) + P(\text{hit}\,R) \cdot C(R)$$

kd-Baum aufbauen

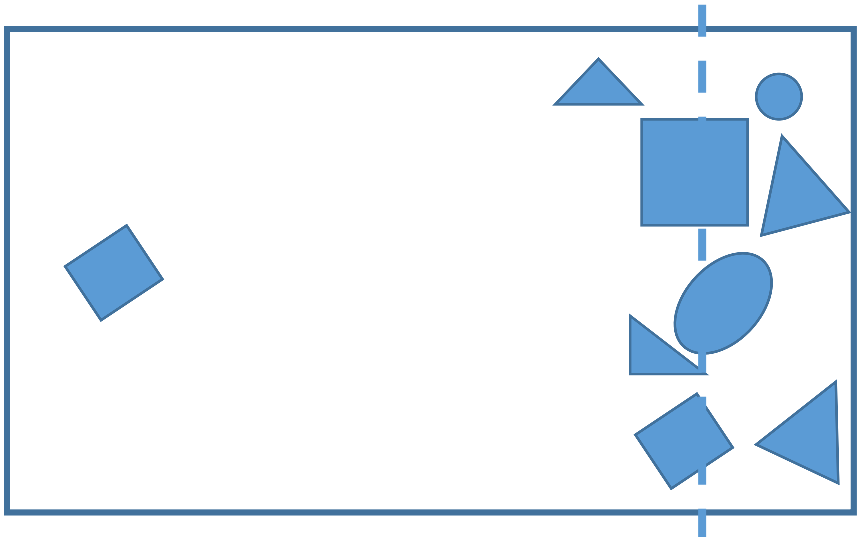


Teile in der **Mitte**:

Gleiche Wahrscheinlichkeiten für L und R,

aber Kosten für L und R nicht mit einbezogen

kd-Baum aufbauen

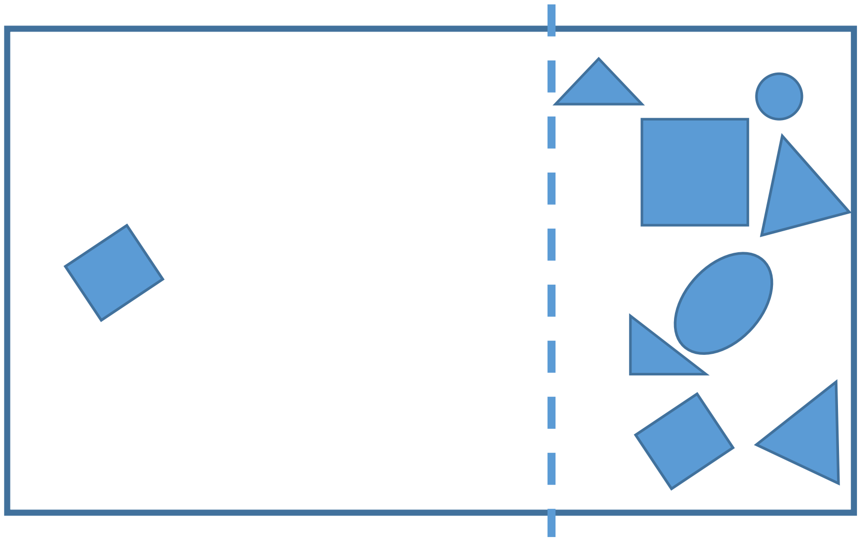


Teile am **Median**:

Gleiche Kosten für L und R,

aber Wahrscheinlichkeiten für L und R nicht einbezogen

kd-Baum aufbauen



Teile mit **optimierten Kosten**:

Isoliert Komplexität automatisch und schnell

Erzeugt große leere Bereiche

kd-Baum aufbauen

Benötige Schnitt**wahrscheinlichkeiten**

=> proportional zur **Oberfläche** der Bounding Box

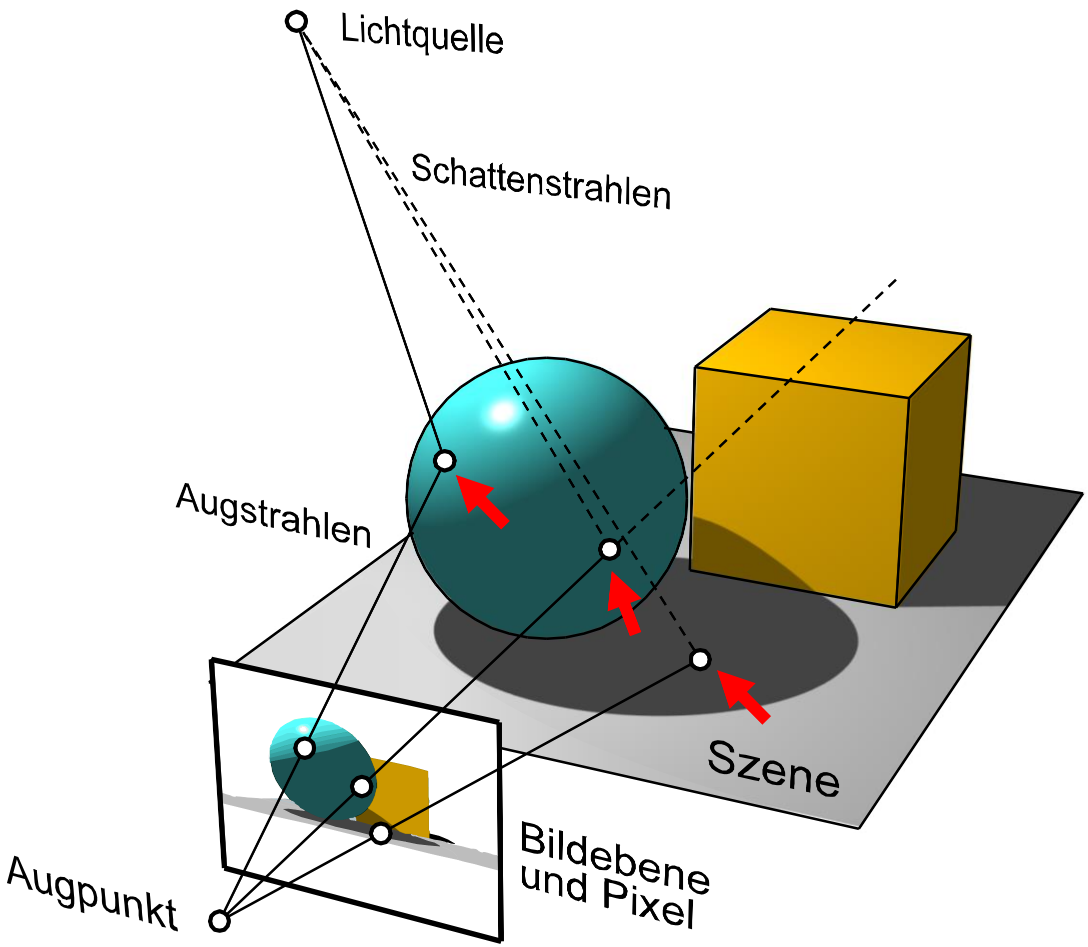
(„**Surface Area Heuristic**“ / SAH) [MacDonald und Booth 1990]

Benötige **Kosten** der Kinder

=> proportional zur **Zahl** der Dreiecke

Optimiere die zu erwartenden Kosten:

$$
\begin{aligned}
C(\mathrm{Cell}) &= C(\mathrm{Trav}) + P(\mathrm{hit}\,L) \cdot C(L) + P(\mathrm{hit}\,R) \cdot C(R) \\
&= C(\mathrm{Trav}) + A(L) \cdot \#\mathrm{Tri}(L) + A(R)\ \#\mathrm{Tri}(R)
\end{aligned}
$$



Zutaten:

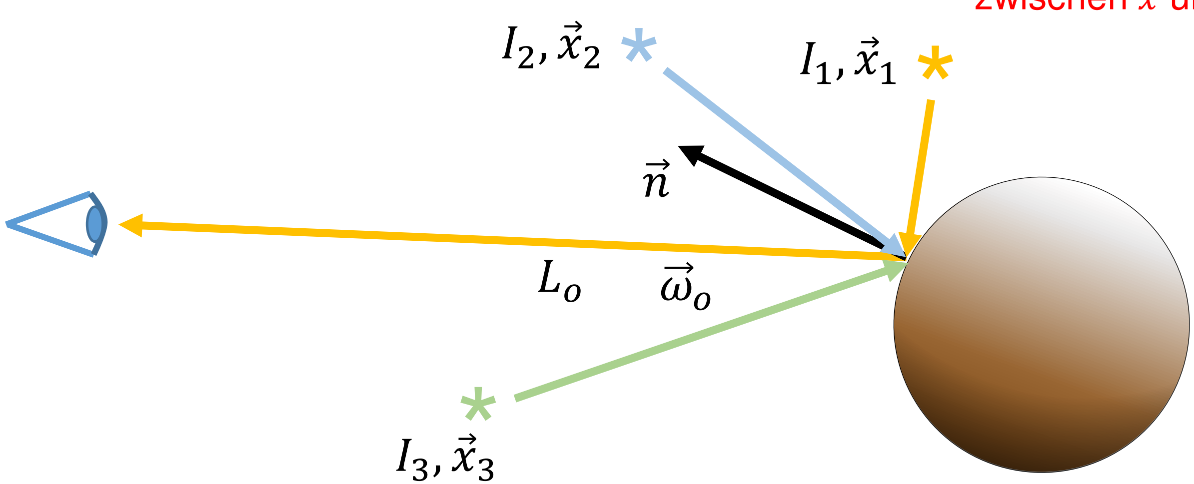
1. Strahlerzeugung
2. Schnittpunkt-
berechnung
3. Schattierung

Direkte Beleuchtung (Punktquellen)

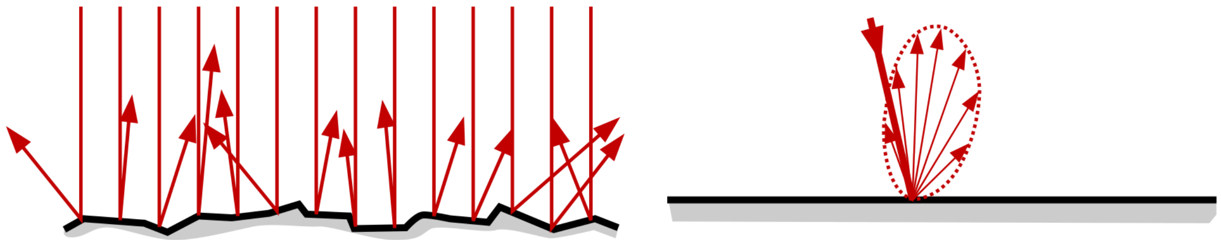
$$L_o(\vec{x},\vec{\omega}_o) = \cancel{L_e(\vec{x},\vec{\omega}_o)} + \int_\Omega f(\vec{\omega}_i,\vec{\omega}_o)\underbrace{L_i(\vec{x},\vec{\omega}_i)}_{\text{einfallende Radianz}}\langle\vec{\omega}_i,\vec{n}\rangle d\vec{\omega}_i$$

$$L_o(\vec{x},\vec{\omega}_o) = \sum_k f(\vec{\omega}_k,\vec{\omega}_o)\overbrace{\overset{\text{Intensität}}{\underset{1/R^2\text{-Term}}{\frac{I_k}{(\vec{x}_k-\vec{x})^2}}}\;\overset{\text{Sichtbarkeit}}{V(\vec{x},\vec{x}_k)}}\langle\vec{\omega}_k,\vec{n}\rangle$$

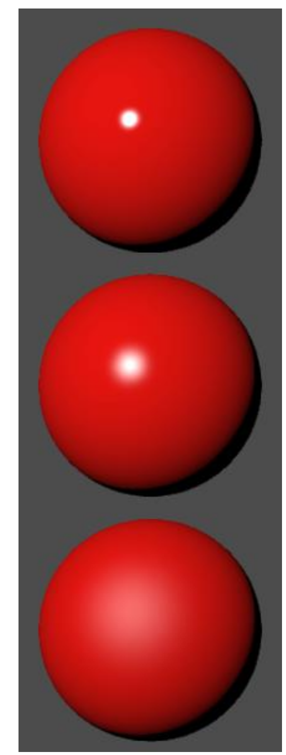
Ist Lichtquelle k vom Punkt $\vec{x}$ aus sichtbar? (Befinden sich keine Objekte auf Schattenstrahlen zwischen $\vec{x}$ und $\vec{x}_k$?)



3. Schattierung

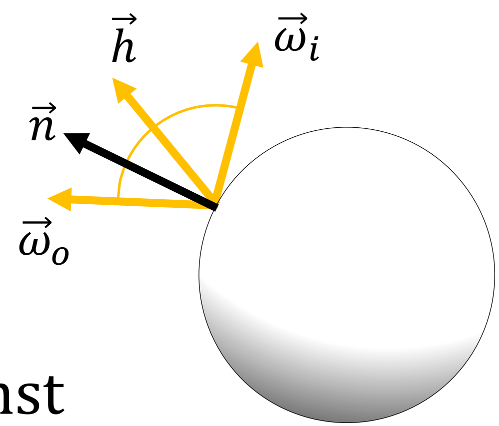


mikroskopische Oberflächenrauigkeit ⇒ makroskopische Reflektanzverteilung



- Erscheinungsbild von Oberflächen oft beschrieben durch Bidirektionale Reflektanz-Verteilungs-Funktion (BRDF)

$$f(\vec{\omega}_i, \vec{\omega}_o) = \frac{\mathrm{d}L_o(\vec{\omega}_o)}{\mathrm{d}E_i(\vec{\omega}_i)}$$



z.B.

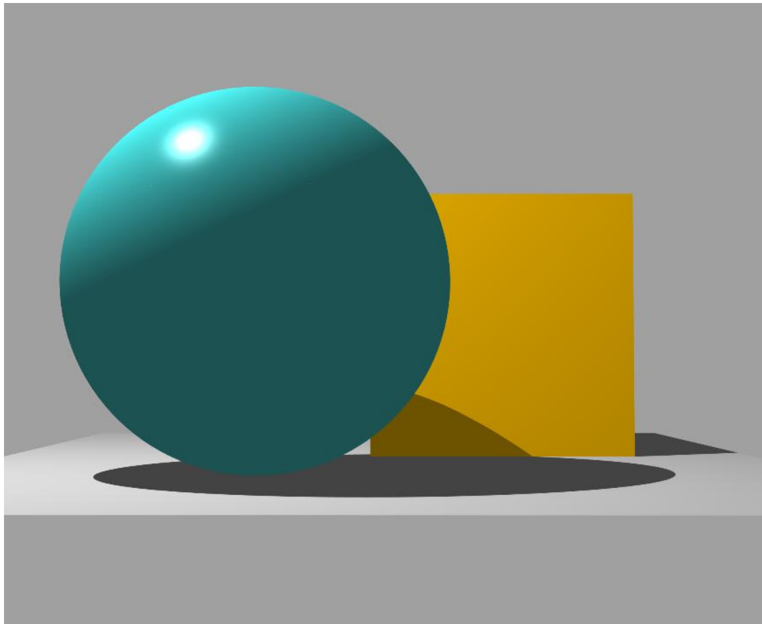
- diffus: $f(\vec{\omega}_i, \vec{\omega}_o) = {}^{k_d}/_{\pi} \;= \mathrm{const}$
- Blinn-Phong: $f(\vec{\omega}_i, \vec{\omega}_o) = k_d/\pi + k_s \dfrac{(\vec{n} \cdot \vec{h})^p}{\vec{n} \cdot \vec{\omega}_i}$

[Das machen wir alles noch ausführlich und physikalisch fundiert!]

Raytracing Grundschema auf einen Blick

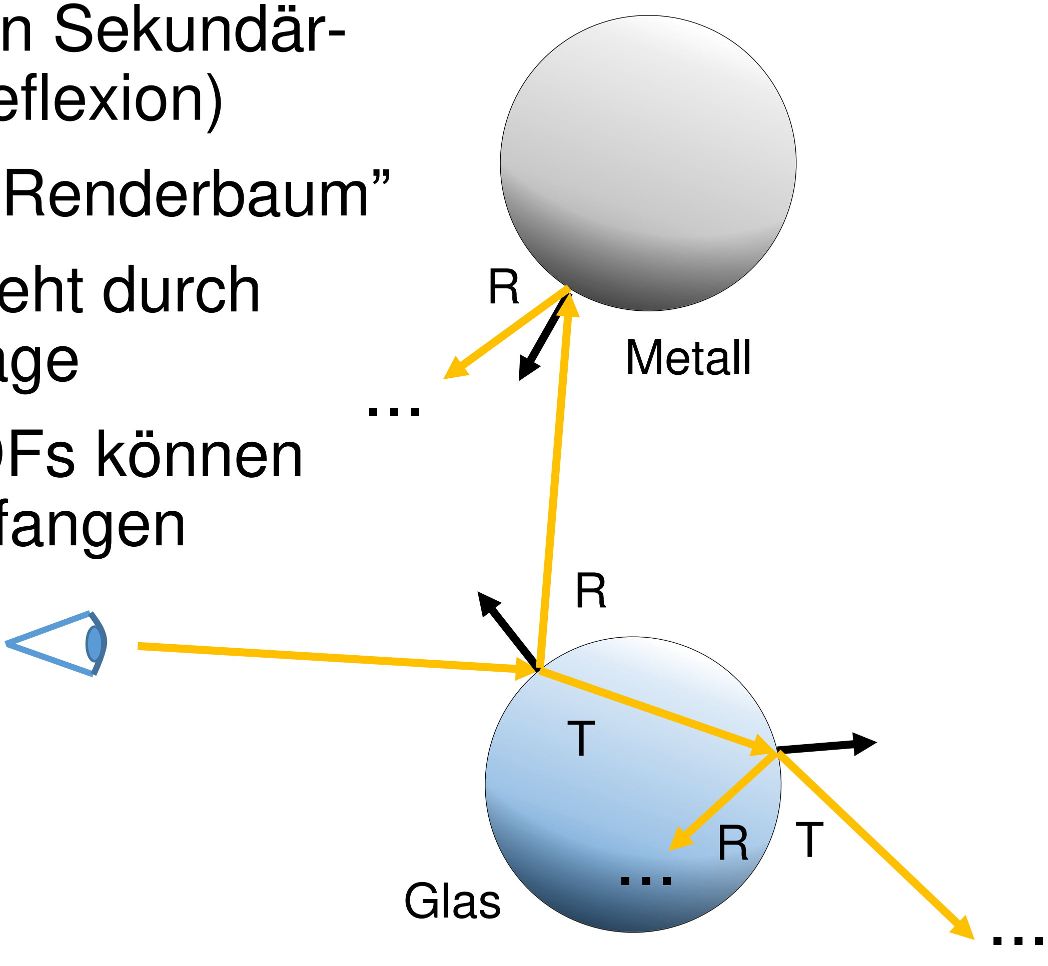
für alle Pixel ***mit*** Koordinate (u, v):

- Erzeuge Augstrahl $(\vec{p}, \vec{d})(u, v)$
- $(\vec{x}, \vec{n}, \mathrm{Objekt}) \leftarrow \mathrm{Schneide}(\vec{p}, \vec{d}, \mathrm{Szene})$
- Helligkeit $E \leftarrow 0$
- ***für alle*** Lichtquellen ***mit*** Position $\vec{x}_k$:
  - ***wenn*** keine Objekte zwischen $\vec{x}$ und $\vec{x}_k$:
    - $E \leftarrow E +$ Beitrag der k-ten Lichtquelle
- ***return*** E



Rekursives Raytracing [Whitted1980]

- Ideal glatte Oberflächen
- Schnittpunkte erzeugen Sekundär-strahlen (Brechung, Reflexion)
- und zwar **rekursiv** ⇒ “Renderbaum”
- Farbe des Pixels entsteht durch Summation aller Beiträge
- Diffuse Objekte + BRDFs können nur direktes Licht empfangen



Direkte Beleuchtung + Spiegelung/Brechung

$$L_o(\vec{x},\vec{\omega}_o) = \cancel{L_e(\vec{x},\vec{\omega}_o)} + \int_\Omega f(\vec{\omega}_i,\vec{\omega}_o)\underbrace{L_i(\vec{x},\vec{\omega}_i)}_{\text{einfallende Radianz}}\langle\vec{\omega}_i,\vec{n}\rangle d\vec{\omega}_i$$

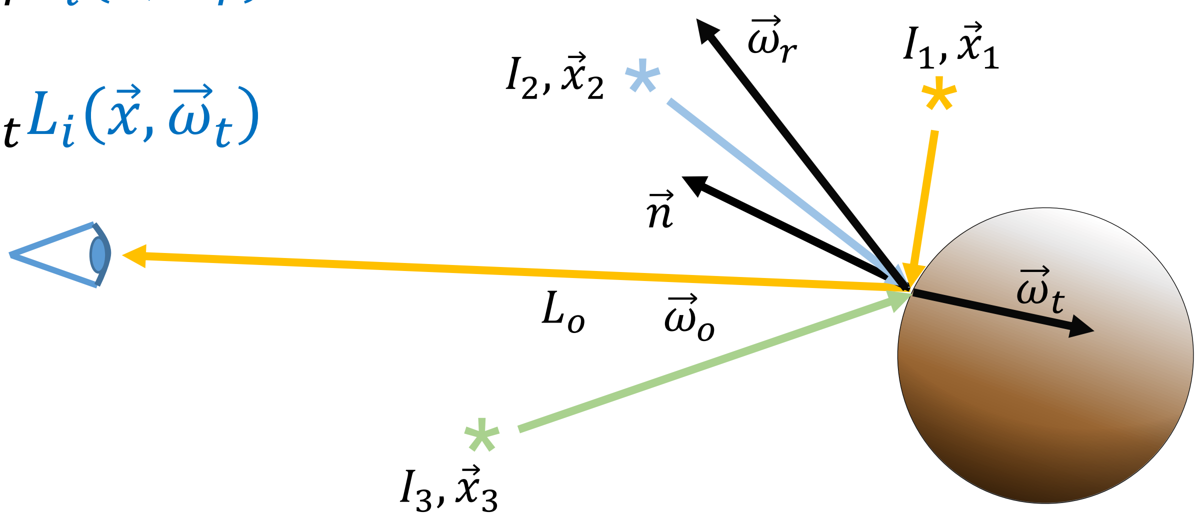
Intensität Sichtbarkeit

$$L_o(\vec{x},\vec{\omega}_o) = \sum_k f(\vec{\omega}_k,\vec{\omega}_o)\frac{I_k}{(\vec{x}_k - \vec{x}\,)^2}V(\vec{x},\vec{x}_k)\langle\vec{\omega}_k,\vec{n}\rangle$$

$1/R^2$-Term

$$+k_r L_i(\vec{x},\vec{\omega}_r)$$

$$+k_t L_i(\vec{x},\vec{\omega}_t)$$



Reflexion

Reflexionsgesetz: „Einfallswinkel gleich Ausfallswinkel“:

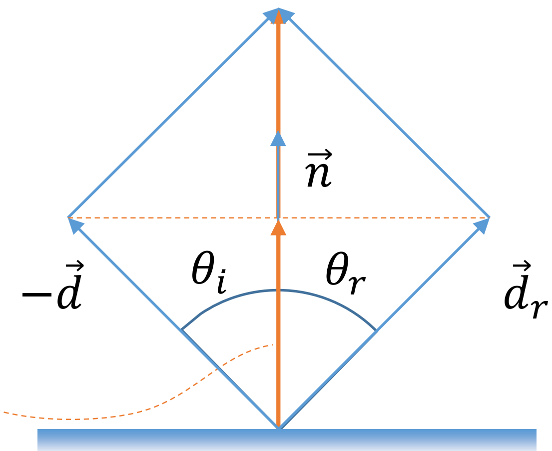
$$\theta_i = \theta_r$$

Normale $\vec{n}$, Einfallsrichtung $\vec{d}$ und Reflexionsrichtung $\vec{d}_r$ sind komplanar

Angenommen, alle Richtungsvektoren sind normiert:

$$\vec{d}_r + (-\vec{d}) = 2\overbrace{\left(\vec{n}\cdot(-\vec{d})\right)\vec{n}}^{\text{Projektion von } \vec{d} \text{ auf } \vec{n}}$$

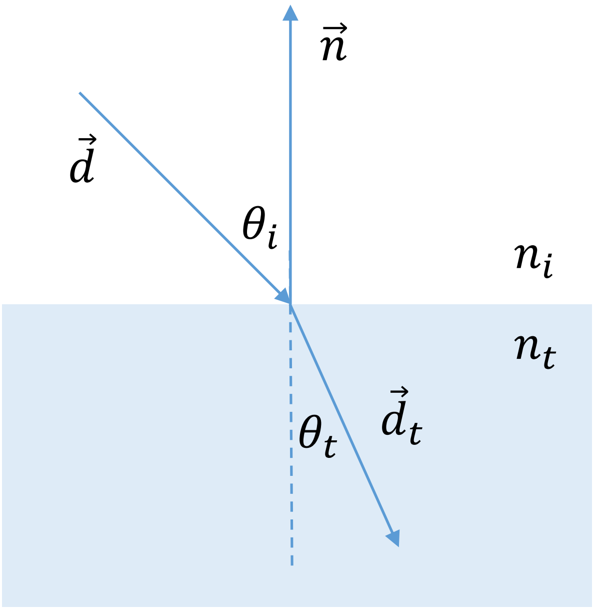
$$\Rightarrow \vec{d}_r = \vec{d} - 2(\vec{n}\cdot\vec{d})\vec{n}$$



Brechung (Refraktion)

Brechungsgesetz (Snellius): $\frac{\sin(\theta_t)}{\sin(\theta_i)} = \frac{n_i}{n_t} = \eta$

Normale $\vec{n}$, Einfallsrichtung $\vec{d}$
und Transmissionsrichtung $\vec{d}_t$
sind komplanar



$$\vec{d}_t = \eta\vec{d} + \left(\eta c - \sqrt{1-\eta^2(1-c^2)}\right)\vec{n}$$

mit $\quad c = -\vec{n}\cdot\vec{d}$

Rekursives Raytracing (Whitted)

für alle Pixel ***mit*** Koordinate (u, v):

> Erzeuge Augstrahl $(\vec{p}, \vec{d})(u, v)$
>
> $E \leftarrow \text{Helligkeit}(\vec{p}, \vec{d})$
>
> $\text{Pixel}[u, v] \leftarrow E$

Funktion $\text{Helligkeit}(\vec{p}, \vec{d})$:

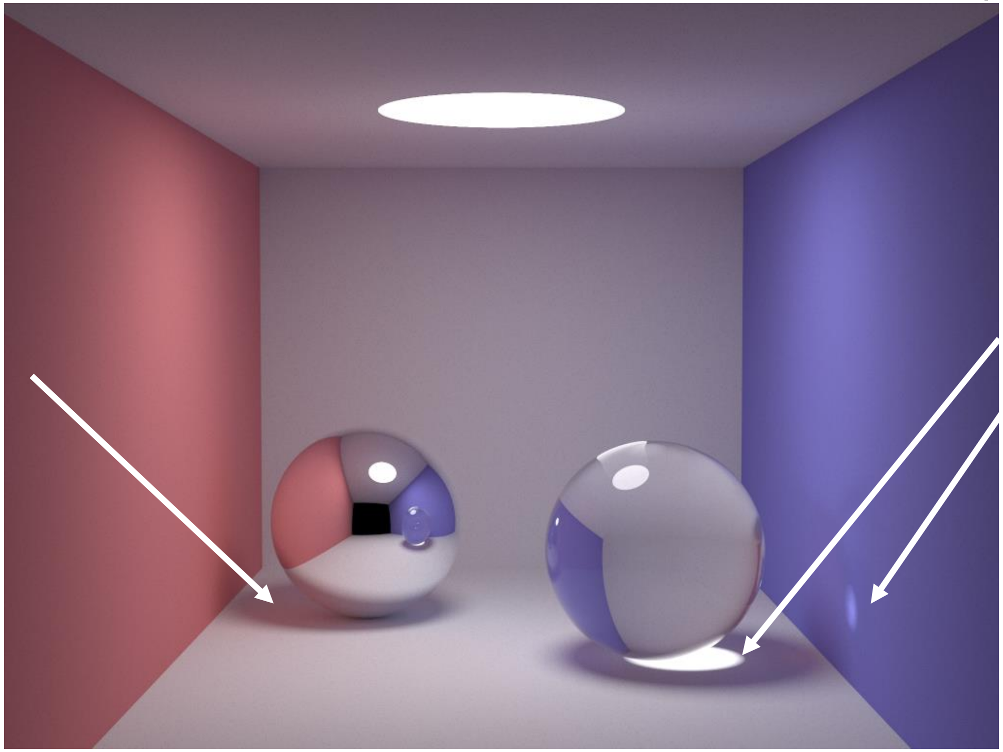
> $(\vec{x}, \vec{n}, \text{Objekt}) \leftarrow \text{Schneide}(\vec{p}, \vec{d}, \text{Szene})$
>
> $(\vec{d}_r, R) \leftarrow \text{Reflexion}(\vec{d}, \vec{n}, \text{Objekt})$
>
> $E \leftarrow R \cdot \text{Helligkeit}(\vec{x}, \vec{d}_r)$;
>
> $(\vec{d}_t, T) \leftarrow \text{Brechung}(\vec{d}, \vec{n}, \text{Objekt})$
>
> $E \leftarrow E + T \cdot \text{Helligkeit}(\vec{x}, \vec{d}_t)$;
>
> ***für alle*** Lichtquellen ***mit*** Position $\vec{x}_k$:
>
> > ***wenn*** $\text{Schneide}(\vec{x}, \vec{x}_k - \vec{x}, \text{Szene}) = \text{NULL}$
> >
> > > $E \leftarrow \text{E} +$ Beitrag der k-ten Lichtquelle;
>
> ***return*** E;

Erweiterungen

Pathtracing [Kajiya1986]

- Numerische Lösung der Renderinggleichung
- Wähle an jedem Schnittpunkt zufällig neue Strahlrichtung (“Sampling”)
- Verfolge Pfade bis zur Lichtquelle
- Berechne Integral über einfallendes Licht als gewichtete Summe der Samples (Monte-Carlo-Integration)
- Anzahl der Samples bestimmt Bildqualität (Rauschen)

“Color bleeding”



Kaustiken (Muster aus gebrochenem Licht)

universitätbonn

“Material Appearance”

Datengetriebene Modelle aus Reflektanzmessungen

MERL-Datenbank

UBonn BTFs/SVBRDFs