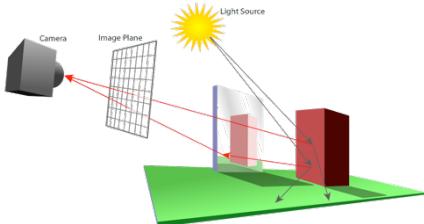


 BEUTH HOCHSCHULE FÜR TECHNIK BERLIN  
University of Applied Sciences

# Computergrafik II

## Raytracing Teil I



Bachelor Medieninformatik  
Wintersemester 2011

Prof. Dr.-Ing. Hartmut Schirmacher  
<http://schirmacher.beuth-hochschule.de>  
hschirmacher@beuth-hochschule.de



### Gliederung



BEUTH HOCHSCHULE  
FÜR TECHNIK  
BERLIN  
University of Applied Sciences

- Raytracing – Motivation (Wdh.)
- Lichtausbreitung in einer Szene
  - Photonen auf Ihrer Reise
  - Die Rendering-Gleichung
- Der Raytracing-Algorithmus
- Die „Zutaten“
  - Lochkamera (Pinhole Camera)
  - Geometrie und lineare Algebra
  - Ebenen, Kugeln, ...
  - Strahlen und Schnittpunktberechnung
- Hinweise zur Übung und Implementierung



## Motivation Raytracing



- Bildsynthese (Rendering)
  - Wie wird aus einer 3D-Szenenbeschreibung ein realistisches Bild?
- Auswahl des richtigen Verfahrens ist anforderungsgetrieben
  - Szenenkomplexität (Anzahl Objekte, wieviele Größen-Skalen)
  - Materialkomplexität (diffus, spekular, volumetrisch, ...)
  - Gewünschte globale Beleuchtungseffekte
  - Zeitbudget für Rendering (1/60s vs. 10 min)
- Raytracing ist der Rendering-„Klassiker“
  - Gut zu verstehen, guter Einstieg in die Grundlagen
  - Eigenständig implementierbar – ohne Grafikkarten-Unterstützung
  - Effizient für bestimmte Szenarien
  - Immer noch Standard im High-End Offline-Bereich



## Lichtausbreitung (Light Transport)





## Betrachtung eines einzelnen Photons



- Die Reise eines Photons
  - Lichtquelle emittiert kontinuierlich Licht / Photonen
  - Ein Photon reist geradlinig, bis es
    - auf eine Oberfläche (2D Fläche im 3D Raum) stößt oder
    - in ein 3D-Volumen („partizipierendes Medium“) eintritt
- Interaktion Photon – Oberfläche
  - Absorption: Photon wird in eine andere Energieform (Wärme) umgewandelt
  - Reflexion: Photon wird in neue Richtung von der Oberfläche weg reflektiert
  - Refraktion/Brechung: Photon dringt durch die Oberfläche, Strahl ändert Richtung
- Interaktion Photon – Volumen
  - Komplexe Interaktion (mehrfache Streuung/Scattering) im Inneren
  - Strahl durch das Volumen (viele Photonen), „von außen“ betrachtet:
    - Abschwächung des Strahls durch Absorption + (Aus-)Streuung weg von der Strahlrichtung
    - Vermehrung des Lichts entlang des Strahls durch (Ein-)Streuung in Strahlrichtung



## Betrachtung vieler Photonen entlang eines Strahls



- Perfekt spiegelnde Oberflächen
  - alle Photonen werden in die gleiche Richtung reflektiert
- Perfekt diffuse Oberflächen
  - Photonen werden in alle Richtungen gleichverteilt
- Real existierende Oberflächen
  - Es existiert eine Vielzahl von Material- und Reflexionsmodellen
  - Meist beschrieben durch eine *Verteilungsfunktion*
    - **BRDF:** Bidirectional Reflectance Distribution Function
    - **BTDF:** Bidirectional Transmittance Distribution Function
    - Noch komplexer: **Subsurface Scattering** / Volumeneffekte



**Rendering-Gleichung (Rendering Equation)**

▪ **Energie-Gleichgewicht** zwischen einfallendem und ausfallendem Licht

$$L(\mathbf{x}, \omega_o) = L_e(\mathbf{x}, \omega_o) + \int_{\Omega} f_r(\mathbf{x}, \omega_i, \omega_o) L(\mathbf{x}, \omega_i) (\omega_i \cdot \mathbf{n}) d\omega_i$$

ausfallendes Licht      Emission      Integral über die (Halb-) Kugel aller möglichen Einfallsrichtungen

Abschwächung durch Einfallswinkel      einfallendes Licht

**BRDF (Verteilungsfunktion)**

**Rendering-Gleichung (Rendering Equation)**

▪ **Energie-Gleichgewicht** zwischen einfallendem und ausfallendem Licht

$$L(\mathbf{x}, \omega_o) = L_e(\mathbf{x}, \omega_o) + \int_{\Omega} f_r(\mathbf{x}, \omega_i, \omega_o) L(\mathbf{x}, \omega_i) (\omega_i \cdot \mathbf{n}) d\omega_i$$

- $L$  = Strahldichte (Radiance)
- $[L] = W / (m^2 \cdot sr)$  „Leistung pro Fläche pro Raumwinkel“ (sr = Steradian)
- Wird gewöhnlich für jede Wellenlänge getrennt betrachtet

## Rendering-Gleichung → Beleuchtungsberechnung



BEUTH HOCHSCHULE  
FÜR TECHNIK  
BERLIN  
University of Applied Sciences

- Was sagt uns die Rendering-Gleichung?
  - Verfolge das Licht entlang von Strahlen
  - An einem Oberflächenpunkt:
    - Sammle das einfallende Licht aus allen relevanten Richtungen
    - Gewichte jeden Beitrag korrekt
      - Abschwächung je nach Winkel
      - Integralterm  $d\omega_i$
      - Verteilungsfunktion (BRDF)
    - Akkumuliere die gewichteten Beiträge

„Shading“  
(Beleuchtungsberechnung)

  - Was sagt uns die Gleichung nicht?
    - Welche BRDF soll ich nehmen?
    - Wieviele Strahlen soll ich verwenden?
    - Wo soll ich anfangen?

Beleuchtungsmodell

Algorithmus + Samplingrate





BEUTH HOCHSCHULE  
FÜR TECHNIK  
BERLIN  
University of Applied Sciences

## Der Raytracing-Algorithmus



## Vorwärtssimulation



- Vorwärts-Simulation

- Emittiere Licht von den Lichtquellen in alle Richtungen
- Oberflächen reflektieren / brechen / absorbieren
- Registriere für jedes Pixel auf der Bildebene, wieviel Licht in Richtung Auge dort ankommt

- Diese Methode wird i.a. als ***Path Tracing*** bezeichnet



## Rückwärts-Simulation

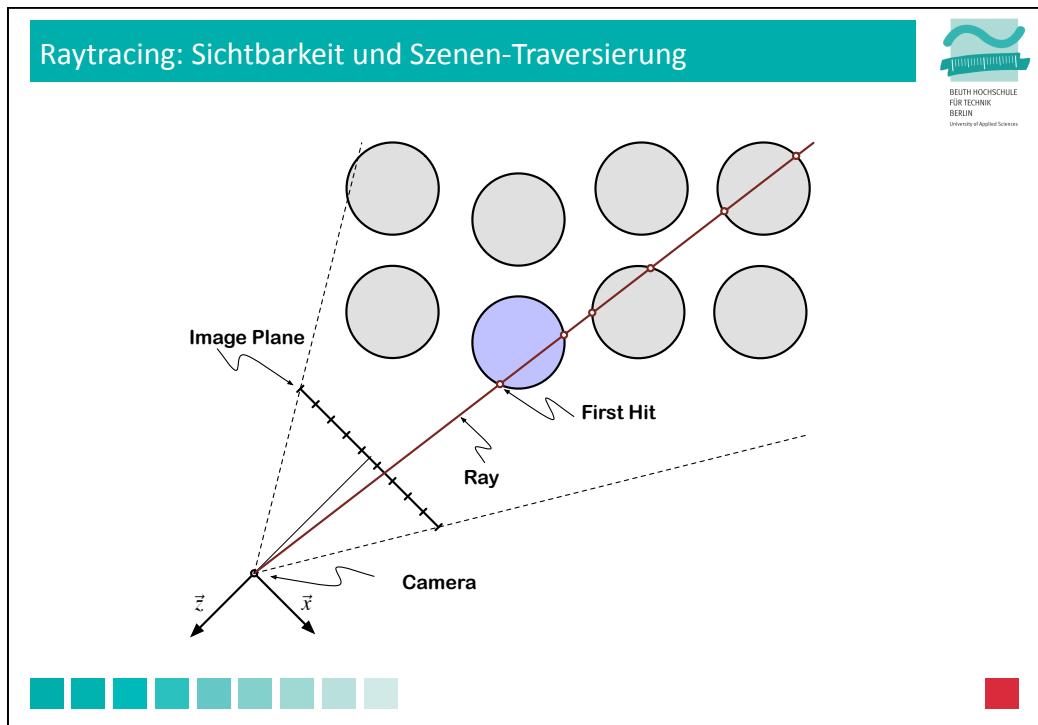
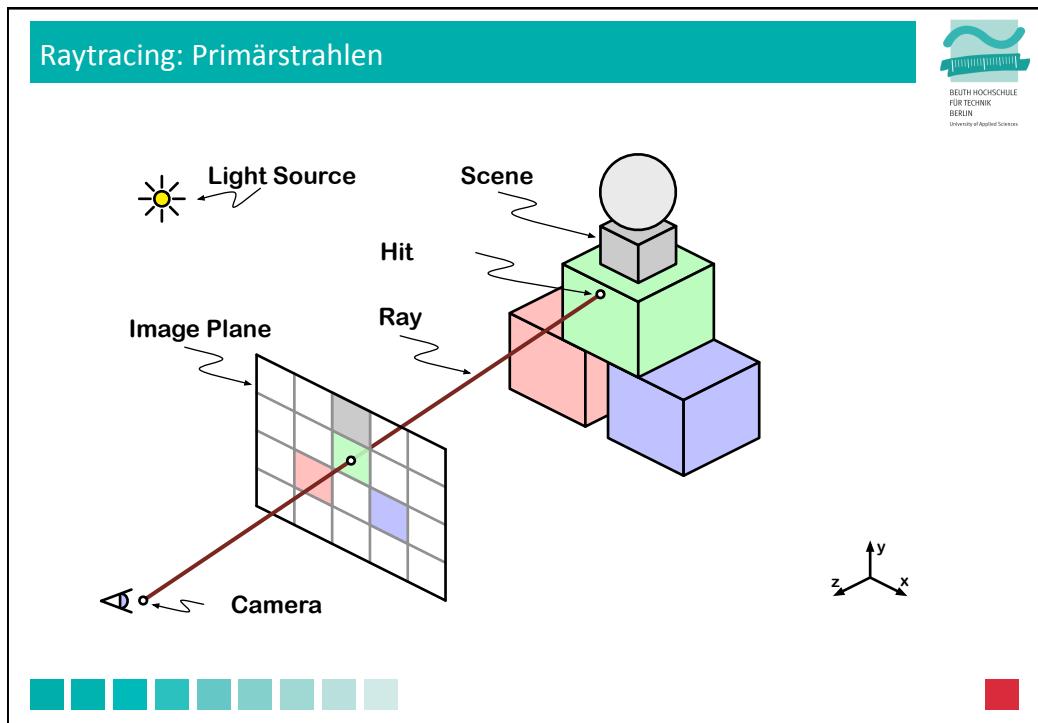


- Rückwärts-Simulation

- Berechne Strahl vom Auge durch jedes Pixel
- Ermittle, welche Objekte Licht entlang dieses Strahls senden
- Berechne, die Lichtmenge (Farbe) von jedem Objekt in Richtung des Strahls zum Pixel/Auge

- Diese Methode wird i.a. als ***Ray Tracing*** bezeichnet





## Pseudocode des Raytracing Main Loops



```
// calculate color by tracing rays from the camera into the scene
raytrace(scene, camera, image)
{
    // loop over all pixels in image
    for all (i,j) from (0,0) to (image.width-1, image.height-1) do
    {
        // generate ray from eye point through pixel center
        ray ← generate_ray(i, j, image.width, image.height, camera);

        // find first intersection point with scene objects
        hit ← intersect(ray, scene);

        // calculate light intensity/color at hit point
        color ← shade(hit, ray, scene);

        // set corresponding pixel color in image
        image.pixel(i,j) ← color;
    }
}
```



## Pseudocode des Raytracing Main Loops



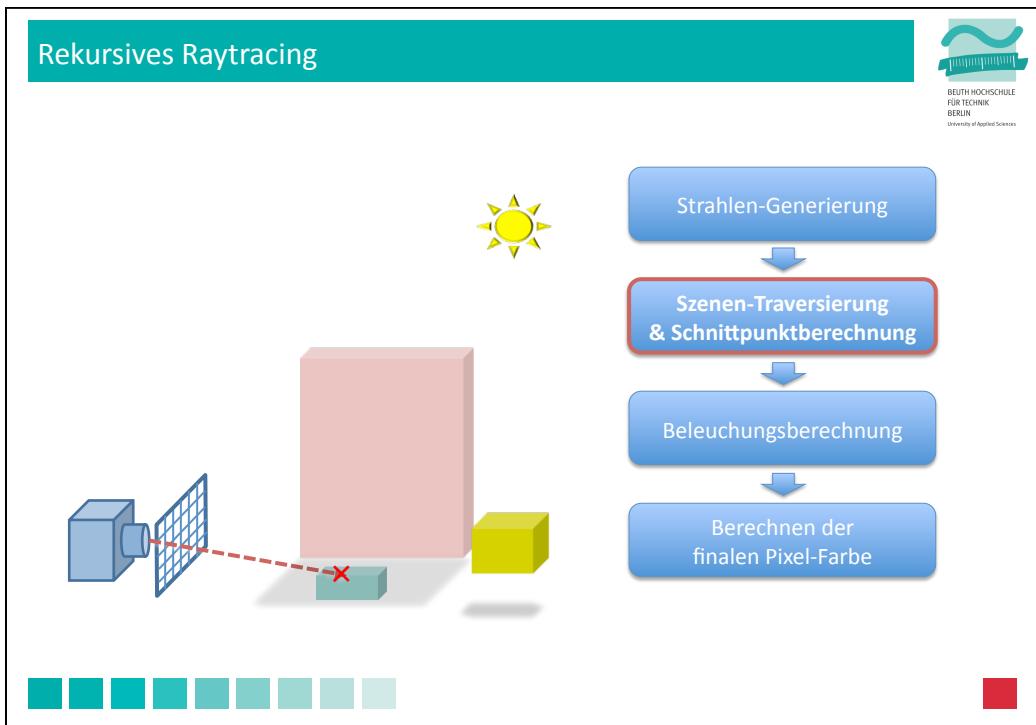
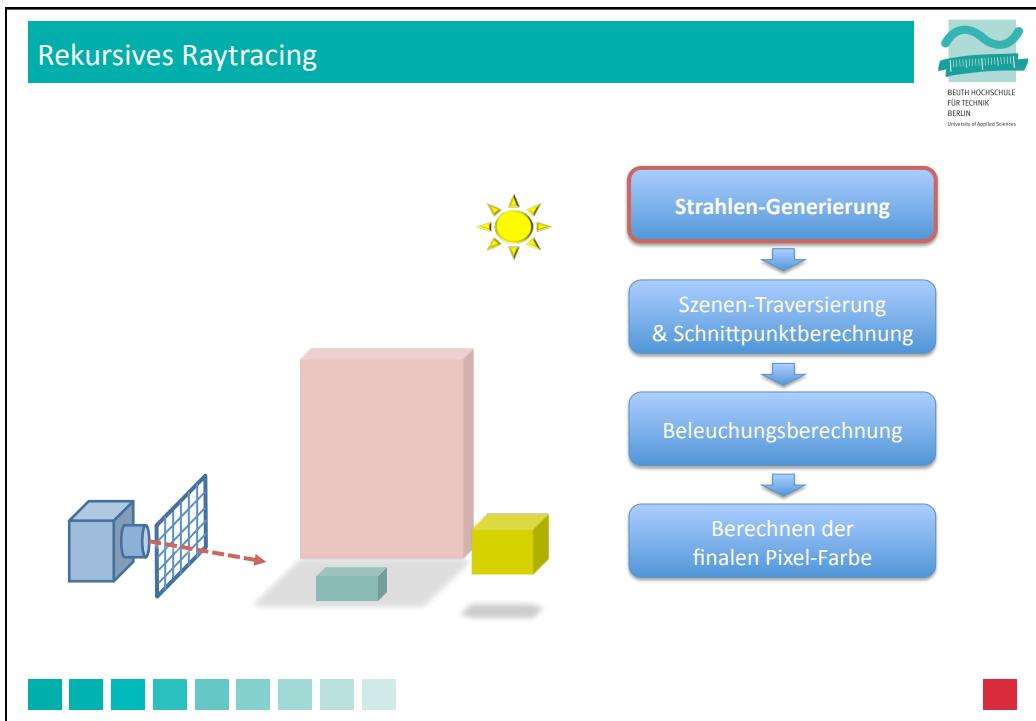
```
// calculate color by tracing rays from the camera into the scene
raytrace(scene, camera, image)
{
    // loop over all pixels in image
    for all (i,j) from (0,0) to (image.width-1, image.height-1) do
    {
        // generate ray from eye point through pixel center
        ray ← generate_ray(i, j, image.width, image.height, camera);

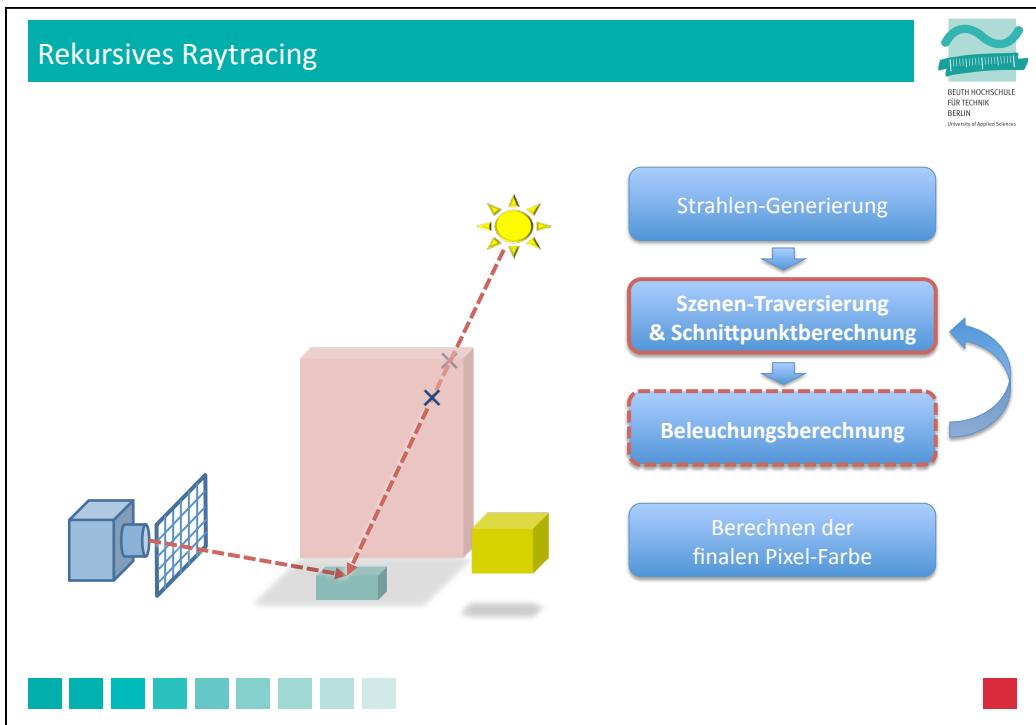
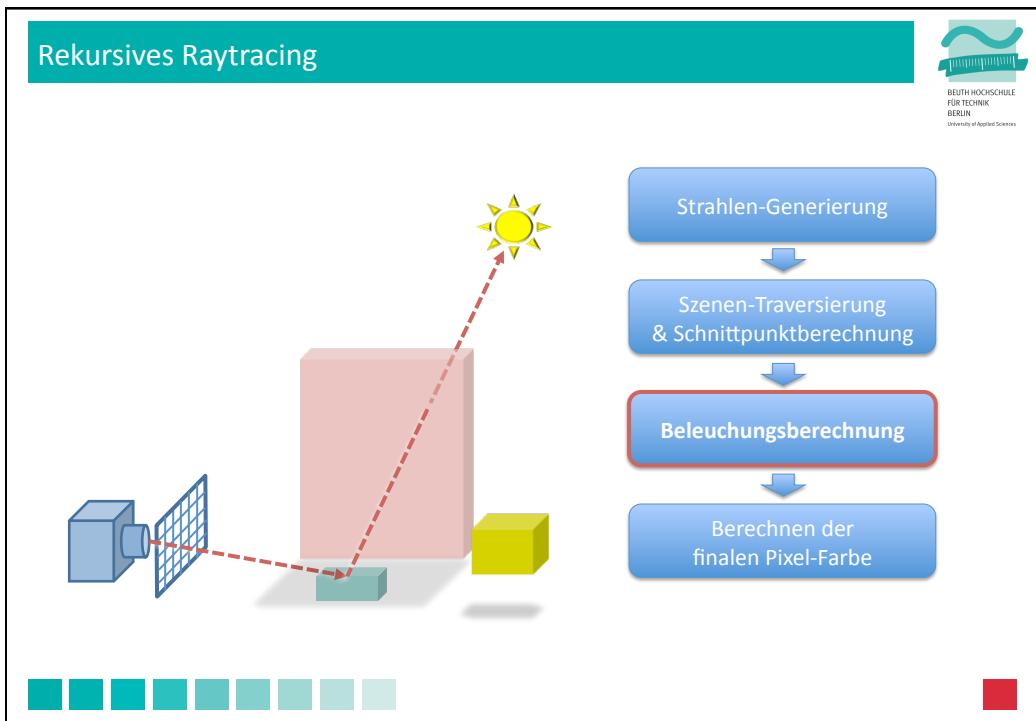
        // find first intersection point with scene objects
        hit ← intersect(ray, scene);

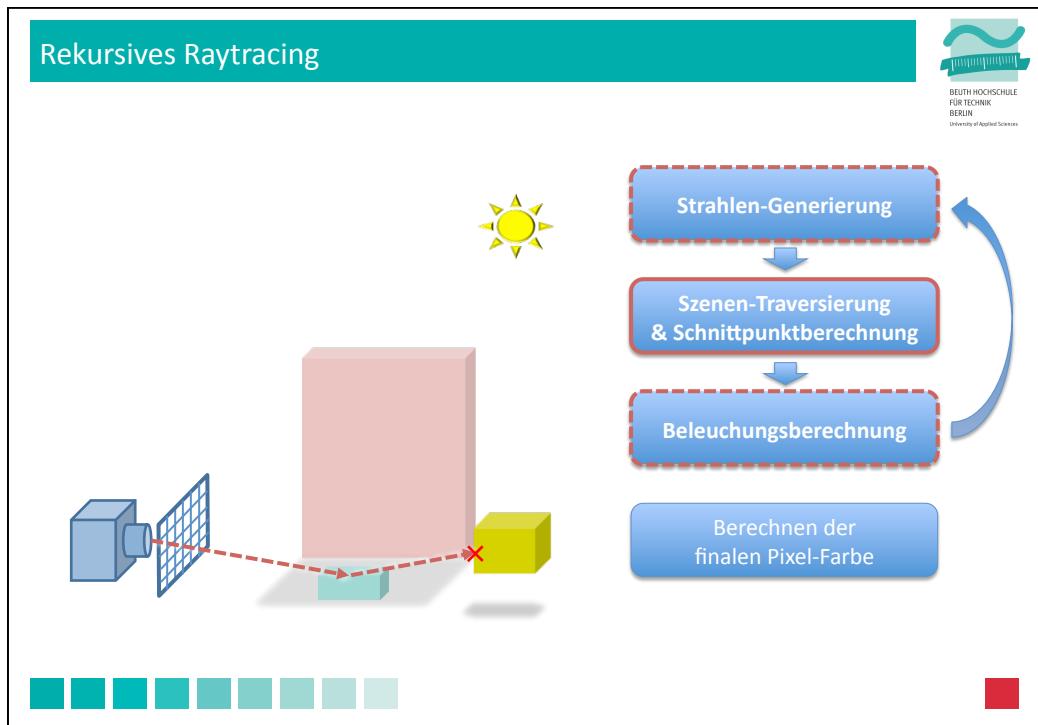
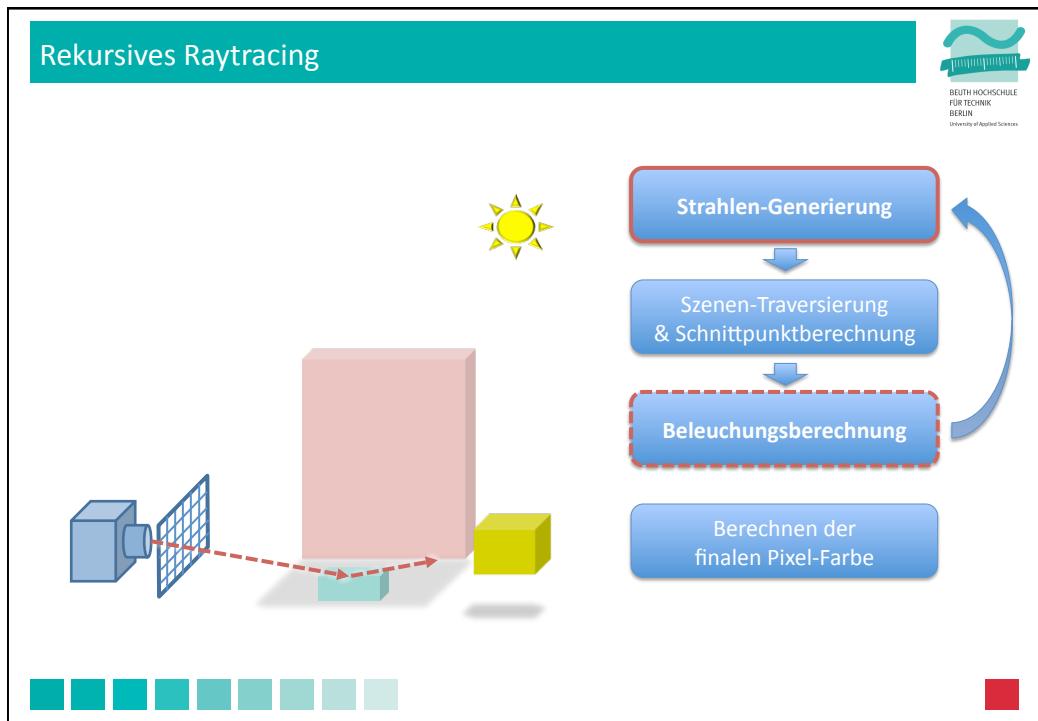
        // calculate light intensity/color
        color ← shade(hit, ray, scene); Potentiell rekursiv!!!

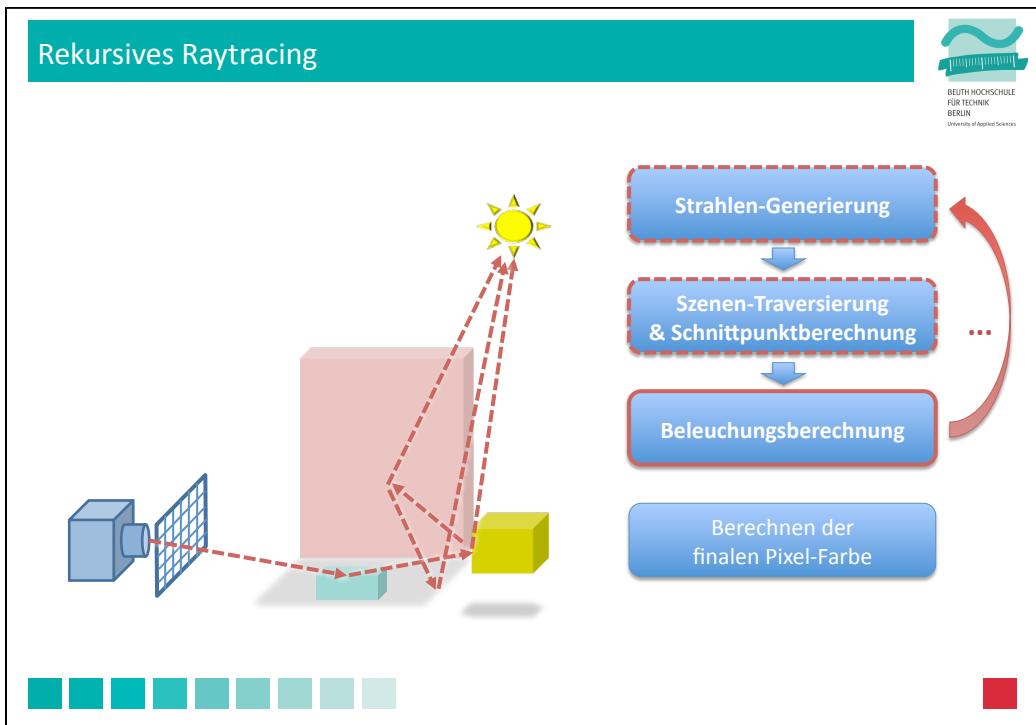
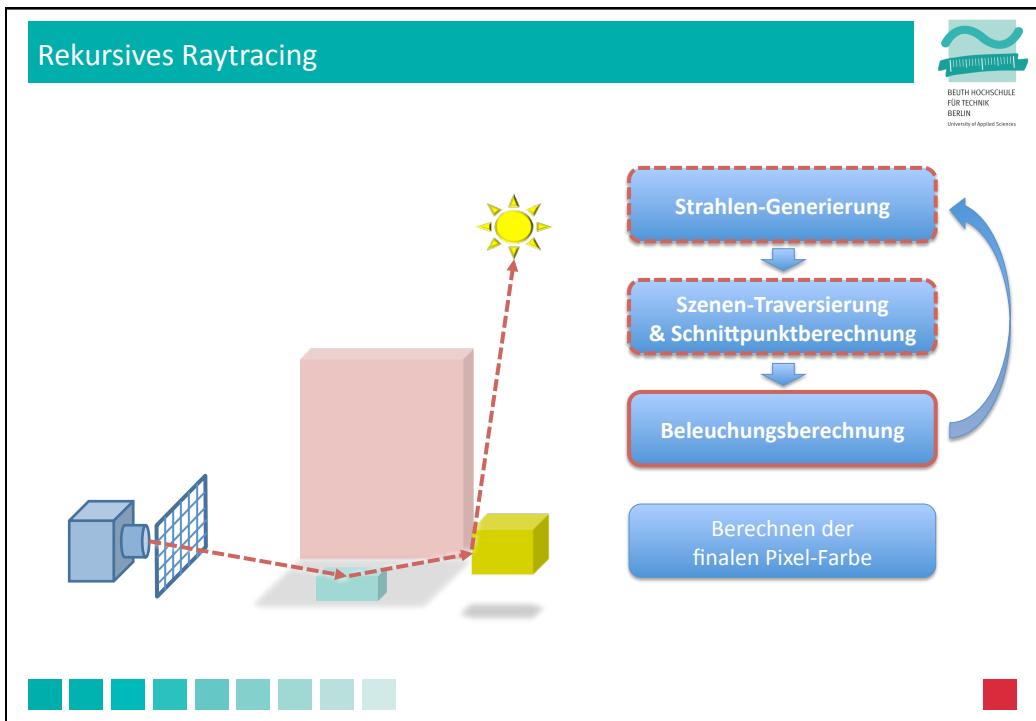
        // set corresponding pixel color in image
        image.pixel(i,j) ← color;
    }
}
```

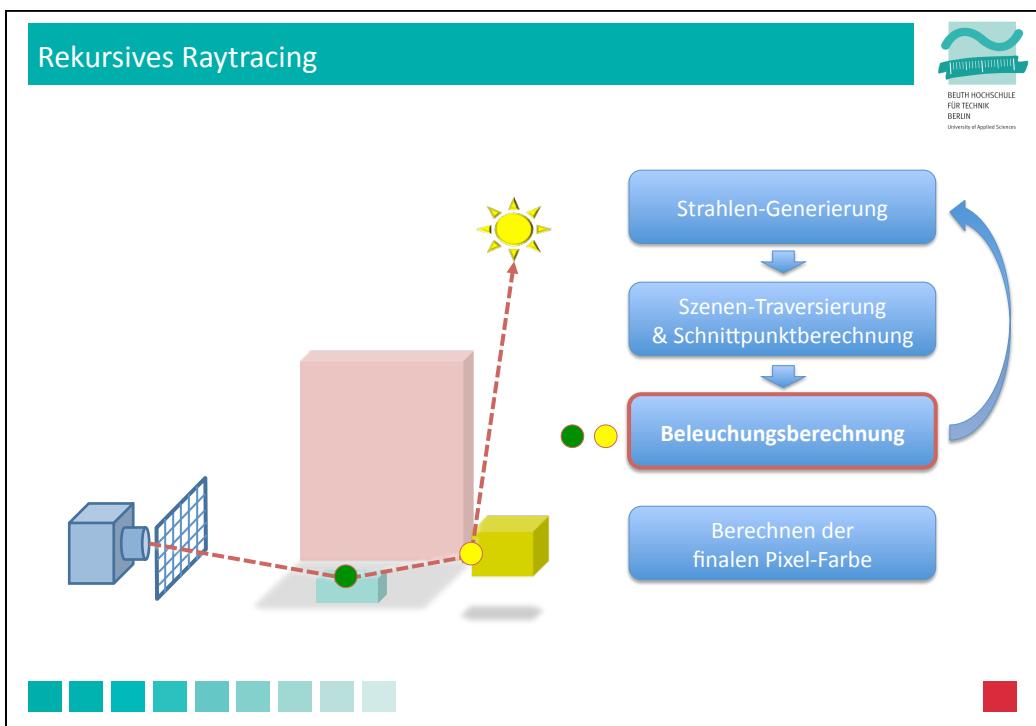
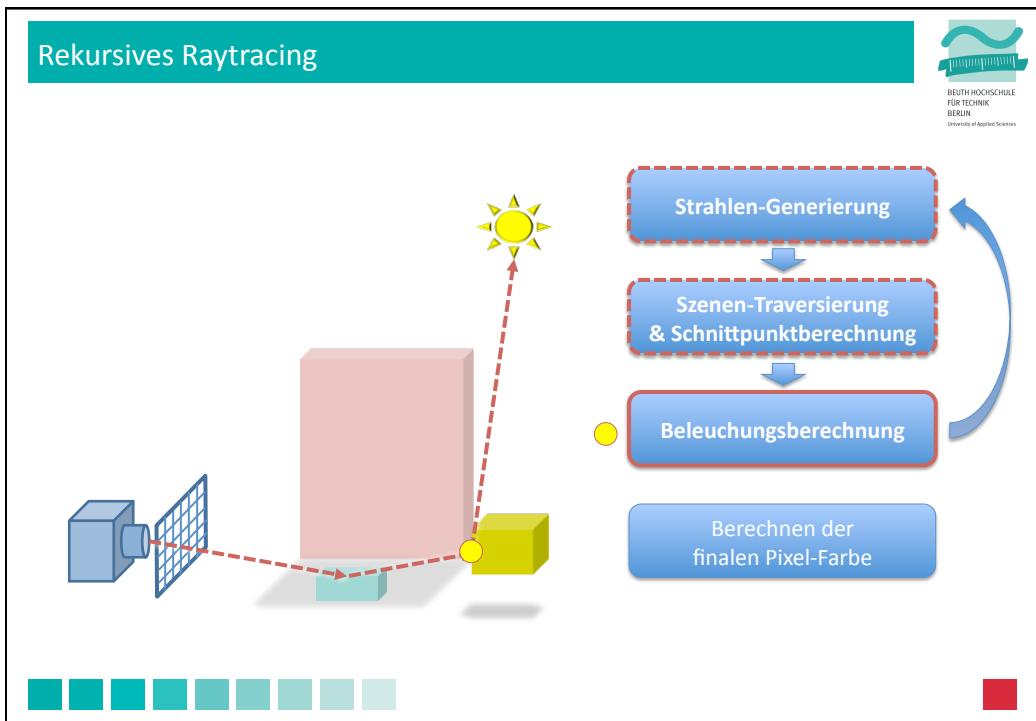


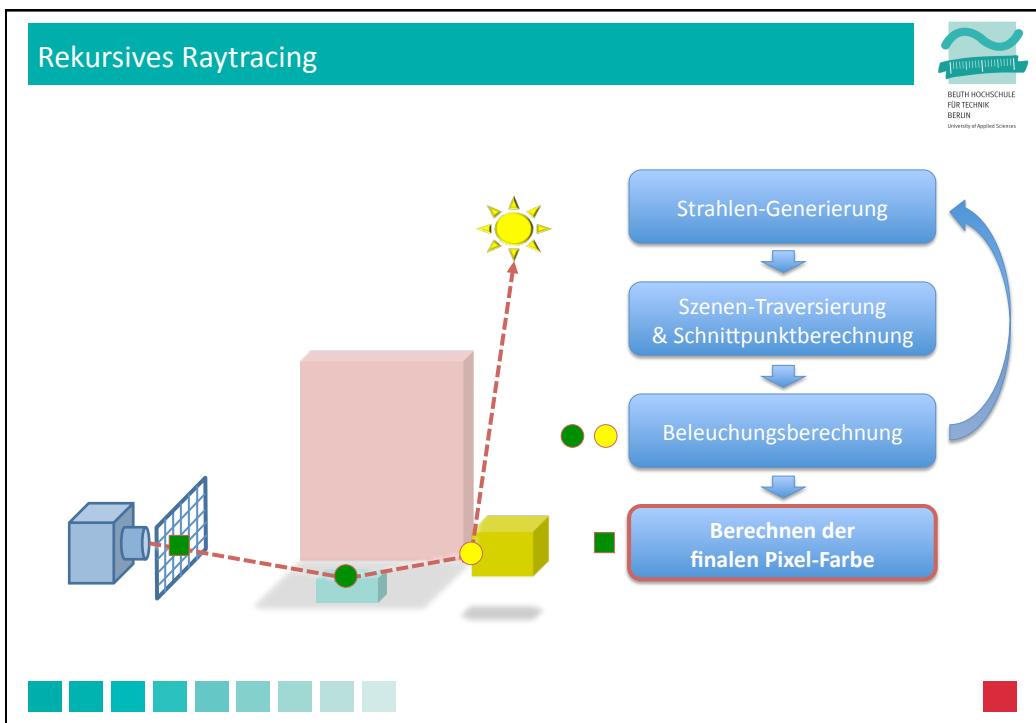












## Notation & Literatur



Typ	Darstellung	Beispiel
Winkel	klein, griechisch	$\alpha, \beta, \gamma$
Skalarwert	klein, lateinisch, kursiv	$a, b, c$
Vektor (Richtung oder Punkt)	klein, lateinisch, <b>fett</b>	<b>a, b, c, sv, n</b>
Matrix	groß, lateinisch, <b>fett</b>	<b>M, T</b>
Skalarprodukt (dot product)	Punkt	$\mathbf{a} \bullet \mathbf{b}$
Kreuzprodukt (cross product)	Kreuz	$\mathbf{a} \times \mathbf{b}$
Matrix-Determinante	senkrechte Linien	$ M $
Vektornorm / Vektorlänge	senkrechte Linien	$ n $

Empfohlene Literatur:

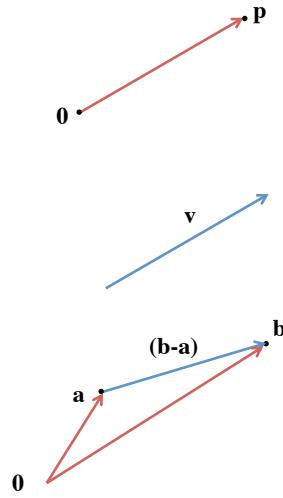
- John Vince, *Mathematics for Computer Graphics*, Springer Verlag.
- Peter Shirley, *Fundamentals of Computer Graphics*, AK Peters.



## Interpretation von Vektoren



- **Positionsvektor  $\mathbf{p}$** 
  - Repräsentiert Koordinaten des Punktes  $\mathbf{p}$
  - Relativ zum Ursprung  $\mathbf{0}$  des Koord.-Systems
  - Wird auch „gebundener Vektor“ (bound vector) genannt
- **Richtungsvektor  $\mathbf{v}$** 
  - Repräsentiert eine Richtung
  - Kann frei verschoben werden
  - Wird auch „freier Vektor“ (free vector) genannt
- **Differenz zweier Positionen**
  - Richtungsvektor  $(\mathbf{b}-\mathbf{a})$  zeigt von  $\mathbf{a}$  nach  $\mathbf{b}$



## Skalarprodukt (Dot Product) zweier Vektoren



- Definition in  $\mathbb{R}^n$

$$\mathbf{u} \cdot \mathbf{v} = \sum_{i=0}^{n-1} u_i v_i$$

- Definition in  $\mathbb{R}^3$

$$\mathbf{u} \cdot \mathbf{v} = \begin{pmatrix} u_x \\ u_y \\ u_z \end{pmatrix} \cdot \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} = u_x v_x + u_y v_y + u_z v_z$$



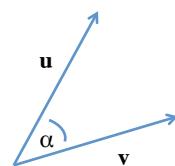
## Geometrische Interpretation des Skalarprodukts



- Skalarprodukt: Winkel zwischen zwei Vektoren

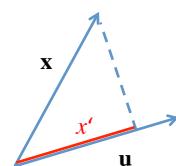
$$\mathbf{u} \cdot \mathbf{v} = |\mathbf{u}| \cdot |\mathbf{v}| \cos \alpha$$

- Länge der beiden Vektoren • Kosinus des eingeschlossenen Winkels
- Vektoren normalisiert → Skalarprodukt gleich Kosinus!



- Skalarprodukt: Projektion auf normierten Vektor

- wenn  $|\mathbf{u}| = 1$ ,
- dann ist  $\mathbf{x}' = \mathbf{x} \bullet \mathbf{u}$  die Länge der Projektion eines Vektors  $\mathbf{x}$  auf  $\mathbf{u}$
- $\mathbf{x}' < 0 \rightarrow \mathbf{x}$  und  $\mathbf{u}$  haben entgegengesetzte Orientierung
- Vektoren orthogonal → Skalarprodukt = 0



## Kreuzprodukt



- Kreuzprodukt zweier Vektoren  $\mathbf{u}$  und  $\mathbf{v}$

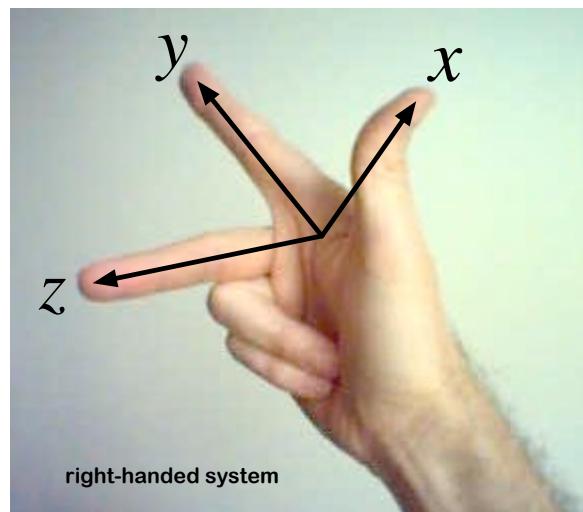
$$\mathbf{w} = \mathbf{u} \times \mathbf{v} = \begin{pmatrix} u_y v_z - u_z v_y \\ u_z v_x - u_x v_z \\ u_x v_y - u_y v_x \end{pmatrix}$$

- Eigenschaften

- $\mathbf{w} \perp \mathbf{u}$
- $\mathbf{w} \perp \mathbf{v}$
- $|\mathbf{w}| = |\mathbf{u}| \cdot |\mathbf{v}| \sin \alpha$
- $\mathbf{u}, \mathbf{v}, \mathbf{w}$  spannen ein rechtshändiges Koordinatensystem auf



## Kreuzprodukt – Rechte-Hand-Regel

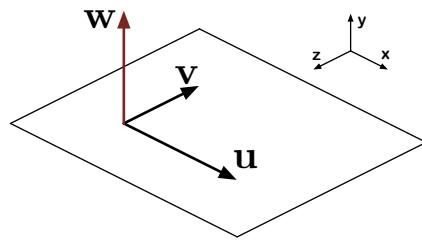


## Kreuzprodukt – geometrische Interpretation



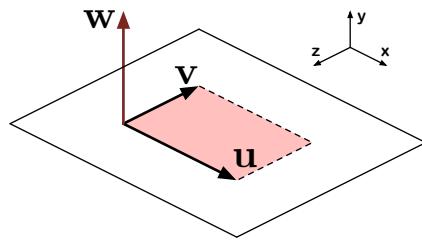
- Normalenvektor

- $w$  ist orthogonal zu  $u$  und  $v$
- Also ist  $w$  eine Normale zu der von  $u$  und  $v$  aufgespannten Ebene
- Bilden  $|u|$  und  $|v|$  ein Orthonormal-  
system (orthogonal und  $|u|=1$ )  
bilden, ist auch  $|w|=1$



- Fläche

- Wegen  $|w| = |u| \cdot |v| \sin \alpha$   
ist  $|w|$  die Fläche des von  
 $u$  und  $v$  aufgespannten  
Parallelogramms

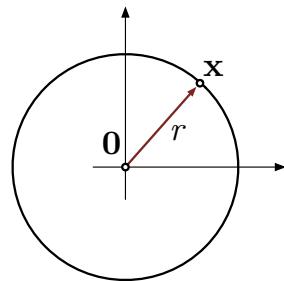


## Kugel in $\mathbb{R}^3$



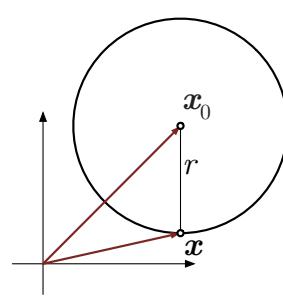
- Kugel um den Ursprung  
mit Radius  $r$

$$\mathbf{x}^2 = r^2$$



- Kugel um  $\mathbf{x}_0$   
mit Radius  $r$

$$(\mathbf{x} - \mathbf{x}_0)^2 = r^2$$



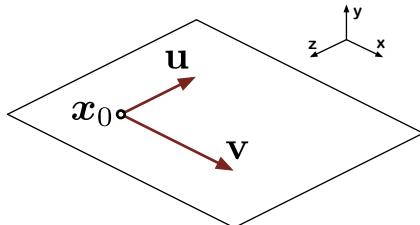
## Ebene in $\mathbb{R}^3$

  
BEUTH HOCHSCHULE  
FÜR TECHNIK  
BERLIN  
University of Applied Sciences

■ Parametrische Form

$$\mathbf{x}(\lambda, \mu) = \mathbf{x}_0 + \lambda \mathbf{u} + \mu \mathbf{v}$$

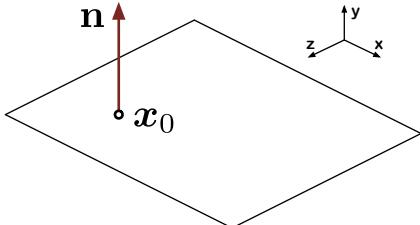
- (Anker-) Punkt  $\mathbf{x}_0$  auf der Ebene
- Zwei Richtungsvektoren  $\mathbf{u}$  und  $\mathbf{v}$
- Zwei Parameter  $\lambda, \mu \in [-\infty, \infty]$



■ Implizite Form

$$\mathbf{n}(\mathbf{x} - \mathbf{x}_0) = 0$$

- (Anker-) Punkt  $\mathbf{x}_0$  auf der Ebene
- Normalenvektor  $\mathbf{n}$




■

## Ebene in $\mathbb{R}^3$

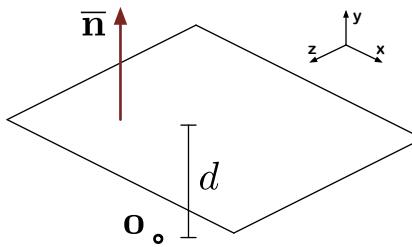
  
BEUTH HOCHSCHULE  
FÜR TECHNIK  
BERLIN  
University of Applied Sciences

■ Hessische Normalform

$$\mathbf{n} \cdot \mathbf{x} - d = 0$$

- Einheits-Normalenvektor  $\mathbf{n}$  mit  $|\mathbf{n}| = 1$
- Abstand (mit Vorzeichen!)  $d$  zwischen Ursprung  $\mathbf{0}$  und Ebene
  - $d > 0$ : Normale zeigt in die Richtung, in der die Ebene relativ zum Ursprung liegt
  - $d < 0$ : Ebene liegt auf anderer Seite des Ursprungs
- Abstand (mit Vorzeichen)  $a$  zwischen beliebigem Punkt  $\mathbf{x}$  und der Ebene

$$a = \mathbf{n} \cdot \mathbf{x} - d$$

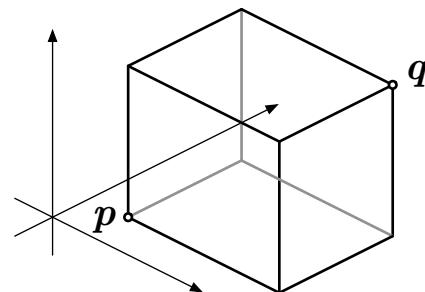



■

## Achsenparalleler Quader (Axis Aligned Box)



- Zwei Punkte  $p$  und  $q$  mit:
  - $p_x < q_x$
  - $p_y < q_y$
  - $p_z < q_z$
  - Repräsentieren die minimalen und maximalen Koordinatenwerte aller Punkte innerhalb der Box  
→ „Axis Aligned Bounding Box“ AABB
- Sechs Ebenen
  - Quader besteht aus sechs Seitenflächen
  - Normalen zeigen nach außen



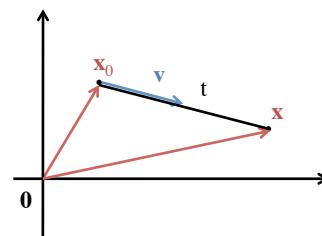
## Strahl / Halbline (Ray / Half Line)



- Parametrische Liniengleichung

$$\mathbf{x}(t) = \mathbf{x}_0 + t\mathbf{v}$$

- Ausgangspunkt  $\mathbf{x}_0$
- Richtung  $\mathbf{v}$ 
  - Wird oft normalisiert:  $|\mathbf{v}| = 1$
- Parameter  $t \in [-\infty, \infty]$



- Halblinie
  - $t \in [0, \infty]$





## Virtuelle Kamera und Primärstrahlen



## Im Algorithmus: Generieren der Primärstrahlen



```
// calculate color by tracing rays from the camera into the scene
raytrace(scene, camera, image)
{
    // loop over all pixels in image
    for all (i,j) from (0,0) to (image.width-1, image.height-1) do
    {
        // generate ray from eye point through pixel center
        ray ← generate_ray(i, j, image.width, image.height, camera);

        // find first intersection with scene objects
        hit ← intersect(ray, scene);

        // calculate light intensity/color at hit point
        color ← shade(hit, ray, scene);

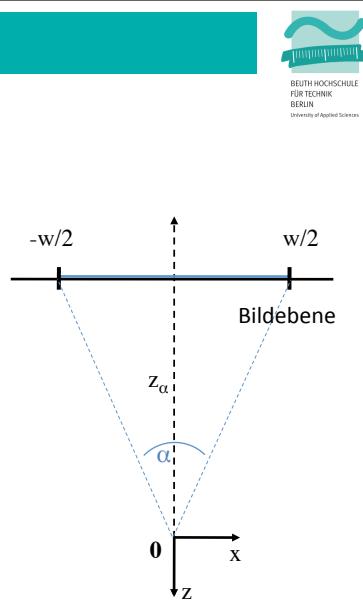
        // set corresponding pixel color in image
        image.pixel(i,j) ← color;
    }
}
```



## Eine einfache virtuelle Kamera (1)

- Kameramodell („Pinhole Camera“)
  - Auge im Ursprung  $\mathbf{0}$
  - Blick entlang  $-z$
  - $y$  zeigt nach oben („up vector“)
  - Öffnungswinkel (in  $x$ ) ist  $\alpha$
  - Größe des Bildes in der Bildebene ist  $(w, h)$  (nicht Pixel, sondern Koordinaten)
  - Bild zentriert in  $(x = 0, y = 0)$
  - Die virtuelle Bildebene liegt im Abstand  $z_\alpha$  vor dem Auge:

$$z_\alpha = \frac{w}{2 \tan \frac{\alpha}{2}}$$



## Eine einfache virtuelle Kamera (2)

- Pixelkoordinaten in der Bildebene
  - Bild umfaßt  $(N_x, N_y)$  Pixel
  - Achtung: quadratische Pixel gewünscht  $\rightarrow N_x : N_y = w : h$
  - $(x,y)$ -Koordinaten des Mittelpunkts von Pixel  $(i,j)$ :

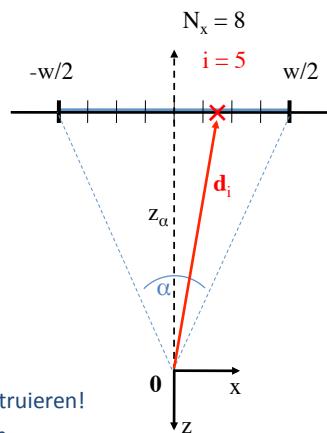
$$x(i) = -\frac{w}{2} + (i + 0.5) \frac{w}{N_x}$$

$$y(j) = -\frac{h}{2} + (j + 0.5) \frac{h}{N_y}$$

- Vektor Auge  $\rightarrow$  Pixel  $(i,j)$

$$\mathbf{d}_{i,j} = \begin{pmatrix} x(i) \\ y(i) \\ z_\alpha \end{pmatrix}$$

Damit können wir den Strahl konstruieren!  
Es ist oft praktisch,  $\mathbf{d}$  zu normieren.





## Schnittberechnung (Ray-Object Intersection)



### Im Algorithmus: Schnittberechnung



```
// calculate color by tracing rays from the camera into the scene
raytrace(scene, camera, image)
{
    // loop over all pixels in image
    for all (i,j) from (0,0) to (image.width-1, image.height-1) do
    {
        // generate ray from eye point through pixel center
        ray ← generate_ray(i, j, image.width, image.height, camera);

        // find first intersection point with scene objects
        hit ← intersect(ray, scene);

        // calculate light intensity/color at hit point
        color ← shade(hit, ray, scene);

        // set corresponding pixel color in image
        image.pixel(i,j) ← color;
    }
}
```



### Schnittberechnung Strahl – Ebene



- Strahl gegeben als:

$$\mathbf{r}(t) = \mathbf{x}_0 + t\mathbf{d}$$

- Ebene in Normalform:

$$\mathbf{n} \cdot \mathbf{x} - d = 0$$

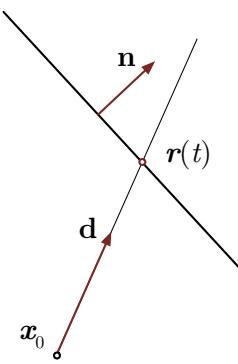
- $\mathbf{x}$  mit  $\mathbf{r}(t)$  substituieren:

$$\mathbf{n} \cdot (\mathbf{x}_0 + t\mathbf{d}) - d = 0$$

- Vereinfacht:

$$t = \frac{d - \mathbf{n} \cdot \mathbf{x}_0}{\mathbf{n} \cdot \mathbf{d}}$$

Nenner = 0  $\rightarrow$   $\mathbf{n}$  orthogonal zu  $\mathbf{d}$   $\rightarrow$  Strahl parallel zu Ebene



### Schnittberechnung Strahl – Kugel (1)

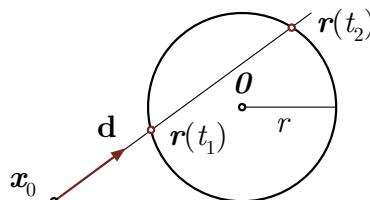


- Strahl:

$$\mathbf{r}(t) = \mathbf{x}_0 + t\mathbf{d}$$

- Kugel im Ursprung:

$$\mathbf{x}^2 = r^2$$



- $\mathbf{x}$  mit  $\mathbf{r}(t)$  substituieren:

$$(\mathbf{x}_0 + t\mathbf{d})^2 = r^2$$

$$\Leftrightarrow \mathbf{x}_0^2 + 2t\mathbf{x}_0 \cdot \mathbf{d} + t^2\mathbf{d}^2 = r^2$$

- Vereinfachung im Fall  $|\mathbf{d}| = 1$ :

$$t^2 + 2t(\mathbf{x}_0 \cdot \mathbf{d}) + (\mathbf{x}_0^2 - r^2) = 0$$

$\rightarrow$  quadratische Gleichung in  $t$   
 $x^2 + px + q = 0$



## Schnittberechnung Strahl – Kugel (2)



- Lösungen der quadratischen Gleichung:

$$x^2 + px + q = 0 \Leftrightarrow x = -\frac{p}{2} \pm \sqrt{\left(\frac{p}{2}\right)^2 - q}$$

- Angewendet auf die Schnittpunktberechnung:

$$t^2 + 2t(\mathbf{x}_0 \cdot \mathbf{d}) + (\mathbf{x}_0^2 - r^2) = 0$$

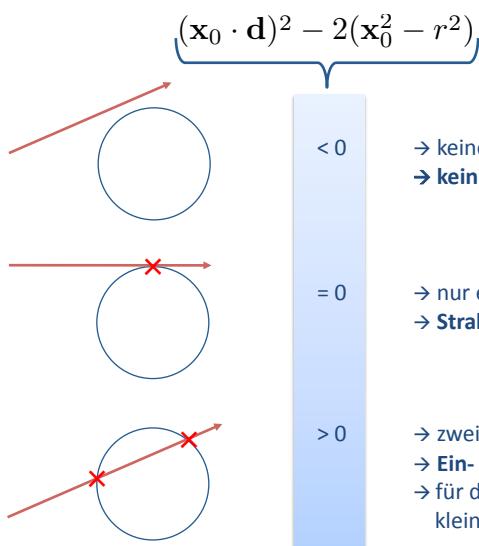
$$t = -(\mathbf{x}_0 \cdot \mathbf{d}) \pm \sqrt{\underbrace{(\mathbf{x}_0 \cdot \mathbf{d})^2 - 2(\mathbf{x}_0^2 - r^2)}_{< 0 ?}}$$

= 0 ?

> 0 ?



## Schnittberechnung Strahl – Kugel (3)



→ keine reelle Lösung  
→ **kein Schnittpunkt**

→ nur eine Lösung  
→ **Strahl streift Kugel tangential**

→ zwei Lösungen  
→ **Ein- und Austrittspunkt des Strahls**  
→ für den Primärstrahl nehmen wir das kleinere  $t$  (Eintrittspunkt)



### Schnittberechnung Strahl – AABB

- Axis Aligned (Bounding) Box
  - Sechs achsenparallele Ebenen
  - Normalen zeigen nach außen
  - in Normalform
$$P_i : \mathbf{n}_i \cdot \mathbf{x} - d_i = 0$$
- Strahl schneidet Box
  - Strahl schneidet alle sechs Ebenen
  - Finde die Ebene, auf der der Strahl tatsächlich in die Box eintritt (Eintrittspunkt)

### Schnittberechnung Strahl – AABB

- Algorithmus (Woo)
  - Finde die drei Ebenen, deren Normalen zum Ursprung des Strahls hin orientiert sind:  

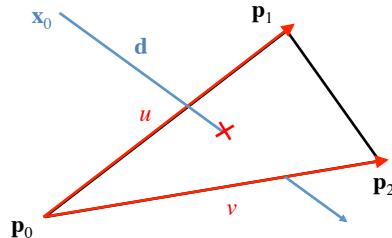
$$\mathbf{n}_i \cdot (\mathbf{x}_0 - \mathbf{p}_i) > 0$$

(dabei ist  $\mathbf{p}_i$  ein beliebiger Punkt auf der Ebene  $i$ )
  - Für diese drei Ebenen berechne den Strahlparameter  $t_i$  des jeweiligen Schnittpunkts (siehe Schnitt Strahl/Ebene)
  - Wähle das größte  $t_i$
  - Für dieses  $t_i$  teste, ob Schnittpunkt in der Box liegt  
 (simpler Koordinatenvergleich mit  $\mathbf{p}_{\min}$  und  $\mathbf{p}_{\max}$ )

## Schnittberechnung Strahl – Dreieck



- Es existieren viele Methoden (s. Literatur)...



- $(p_1-p_0)$  und  $(p_2-p_0)$  spannen eine Ebene auf
- Schnittpunkt mit (parametrischer) Ebene berechnen  
→ man erhält die beiden Parameter  $u$  und  $v$
- $u$  und  $v$  sind die baryzentrischen Koordinaten des Schnittpunkts
- Der Punkt liegt im Dreieck, wenn  $u > 0$ ,  $v > 0$  und  $u + v < 1$ .



## Kurze Geschichte Schnittpunktberechnungen



- Schnittpunkt-Algorithmen
  - Polygone: [Appel '68]
  - Quadriken, CSG: [Goldstein & Nagel '71]
  - Recursives Ray Tracing: [Whitted '79]
  - Toren: [Roth '82]
  - Bikubische Flächen: [Whitted '80, Kajiya '82]
  - Algebraische Flächen: [Hanrahan '82]
  - Swept surfaces: [Kajiya '83, van Wijk '84]
  - Fraktale: [Kajiya '83]
  - Deformationen: [Barr '86]
  - NURBS: [Stürzlinger '98]
  - Subdivision-Flächen: [Kobbelt et al '98]





## Hinweise zu Übung und Implementierung



### Übungsaufgabe (1)



- Schreiben Sie einen Raytracer:
  - Vereinfachtes Kameramodell
  - Vereinfachtes Szenenmodell (Kugeln + Ebenen)
  - Für jeden Pixel einen Primärstrahl ausrechnen
  - Für jeden Primärstrahl den ersten Treffer in der Szene ermitteln
  - Für jeden Treffer die Farbe des Objekts ermitteln (einfach nur ein „Selbstleuchten“ des Objekts)
  - Den Pixel auf die Farbe des Treffer-Punkts setzen
  - Shading und Rekursion → Nächste Vorlesung / Übung



## Übungsaufgabe (2)



### ■ Design des Systems:

- Vernünftige Klassen wie Image, Camera, Scene, Ray, Hit, Shape, Sphere, Plane, ... verwenden
- Wiederverwenden der Imager-Komponente der Warmup-Aufgabe
- Verwenden der Klassen aus cg2.vecmath (vector, color)
- Die Szene soll im Hauptprogramm definiert werden:
  - Instanziere\* Szene
  - Für jedes Szenen-Objekt (Kugel / Ebene / ...):
    - Instanziere Objekt
    - Füge Objekt zur Szene hinzu
  - Instanziere Kamera
  - Rufe Raytracer auf → erzeuge Bild in .png-Datei

\*) instanziiere, instantiere



## Übungsaufgabe (3)



Ray Traced, No Lighting

