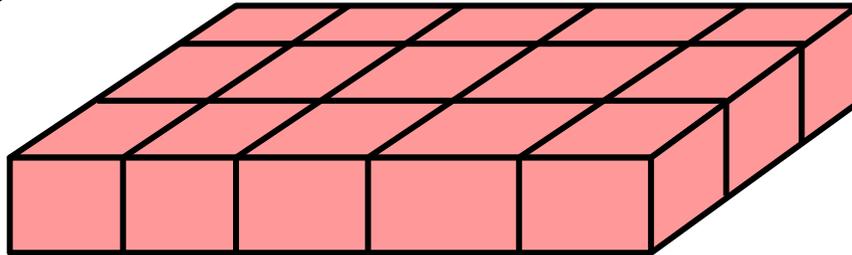
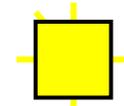
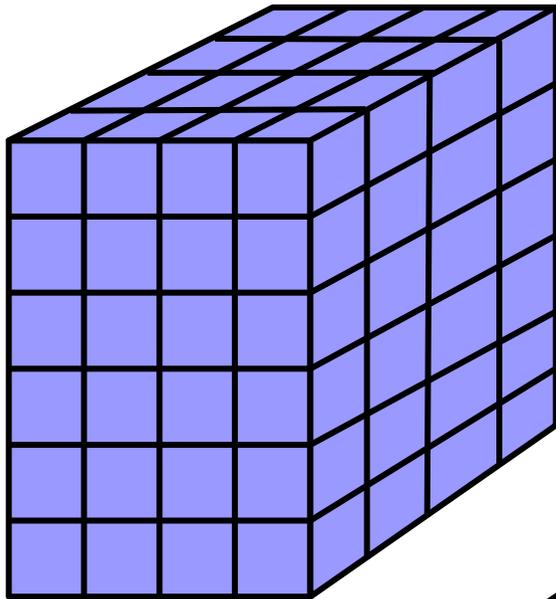


Computergrafik SS 2014

Oliver Vornberger

Kapitel 20:  
Globale Beleuchtung

# Radiosity



# Radiosity-Gleichung

$$B_i \cdot A_i = E_i \cdot A_i + \rho_i \sum_{j=1}^n B_j \cdot F_{ji} \cdot A_j$$

$B_i$  Radiosity (Energie pro Fläche, pro Zeit)

$A_i$  Größe von Fläche i

$E_i$  Eigenstrahlung (pro Fläche, pro Zeit)

$\rho_i$  Reflexionsvermögen von Fläche i

$F_{ji}$  Anteil der von j nach i abgegebenen Rate  
= Formfaktor

# Formfaktoren

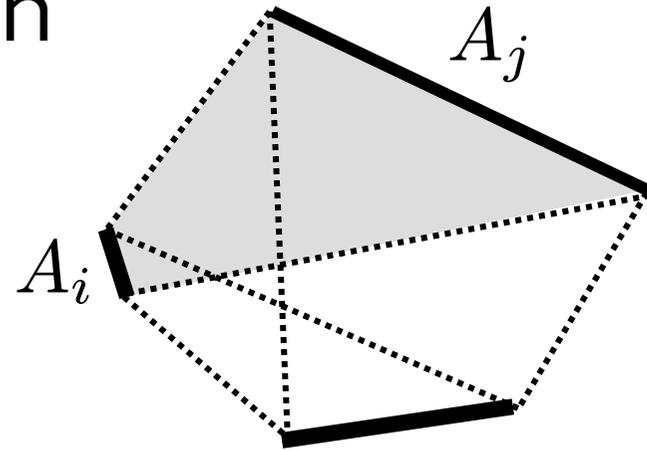
$$A_i \cdot F_{ij} = A_j \cdot F_{ji}$$

$$F_{ij} = \frac{A_j}{A_i} \cdot F_{ji}$$

$$B_i \cdot A_i = E_i \cdot A_i + \rho_i \sum_{j=1}^n B_j \cdot F_{ji} \cdot A_j$$

$$B_i = E_i + \rho_i \sum_{j=1}^n B_j \cdot F_{ij}$$

$$B_i - \rho_i \sum_{j=1}^n B_j \cdot F_{ij} = E_i$$



# Gleichungssystem

$$B_i - \rho_i \sum_{1 \leq j \leq n} B_j \cdot F_{ij} = E_i$$

$$B_1 - \rho_1 B_1 F_{11} - \rho_1 B_2 F_{12} - \rho_1 B_3 F_{13} \dots = E_1$$

$$B_1(1 - \rho_1 F_{11}) - B_2 \rho_1 F_{12} - B_3 \rho_1 F_{13} \dots = E_1$$

$$\begin{pmatrix} 1 - \rho_1 F_{11} & -\rho_1 F_{12} & \dots & -\rho_1 F_{1n} \\ -\rho_2 F_{21} & 1 - \rho_2 F_{22} & \dots & -\rho_2 F_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ -\rho_n F_{n1} & -\rho_n F_{n2} & \dots & 1 - \rho_n F_{nn} \end{pmatrix} \cdot \begin{pmatrix} B_1 \\ B_2 \\ \vdots \\ \vdots \\ \vdots \\ B_n \end{pmatrix} = \begin{pmatrix} E_1 \\ E_2 \\ \vdots \\ \vdots \\ \vdots \\ E_n \end{pmatrix}$$

$n^2$  Formfaktoren !

$$Ax = b$$

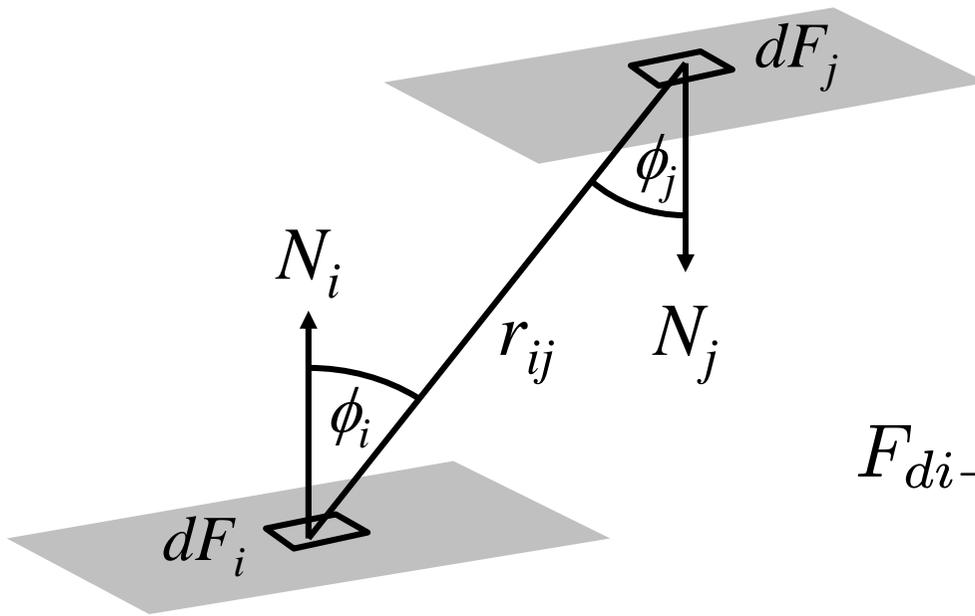
# Gauß-Seidel

$$Ax = b \quad \text{i-te Zeile: } \sum_{j=1}^n A[i, j]x[j] = b[i]$$

$$x[i] = \frac{1}{A[i, i]} \left( b[i] - \sum_{j \neq i} A[i, j]x[j] \right)$$

$$x_k[i] = \frac{1}{A[i, i]} \left( b[i] - \sum_{j=1}^{i-1} x_k[j]A[i, j] - \sum_{j=i+1}^n x_{k-1}[j]A[i, j] \right)$$

## Formfaktoren [Cornell, 1984]



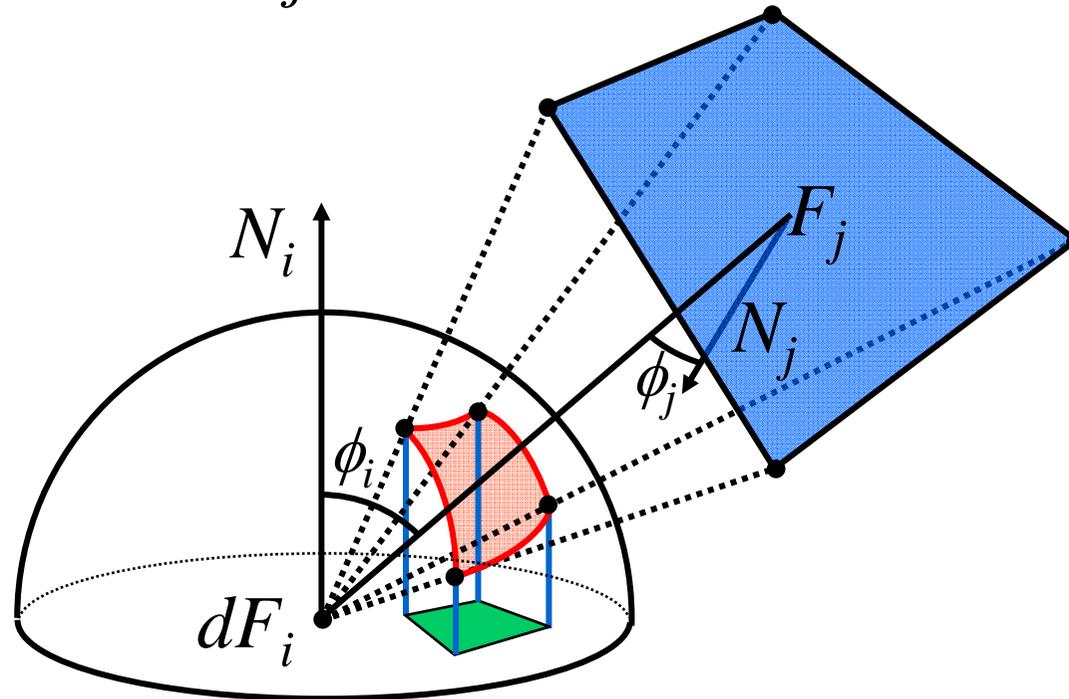
$$F_{di-dj} = \frac{\cos(\phi_i) \cos(\phi_j)}{\pi r_{ij}^2} b_{ij}$$

$$F_{di-j} = \int_{F_j} \frac{\cos(\phi_i) \cos(\phi_j)}{\pi r_{ij}^2} b_{ij} dF_j$$

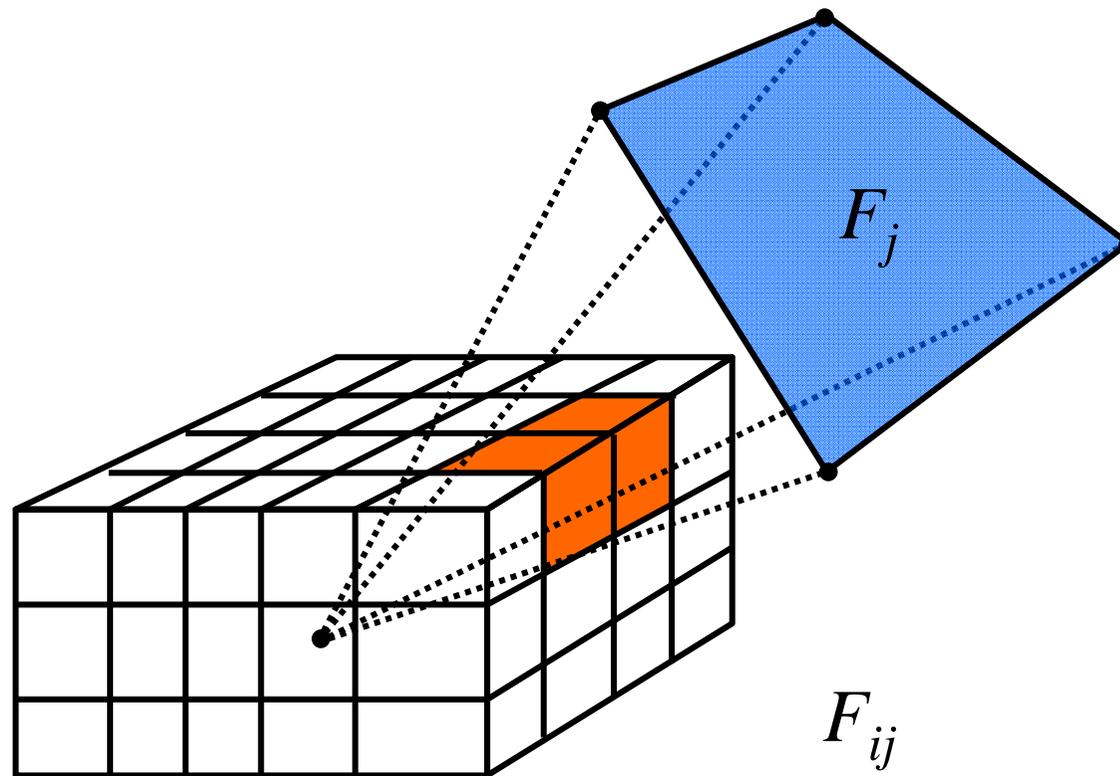
$$F_{ij} = \frac{1}{A_i} \int_{F_i} \int_{F_j} \frac{\cos(\phi_i) \cos(\phi_j)}{\pi r_{ij}^2} b_{ij} dF_j dF_i$$

# Geometrische Interpretation

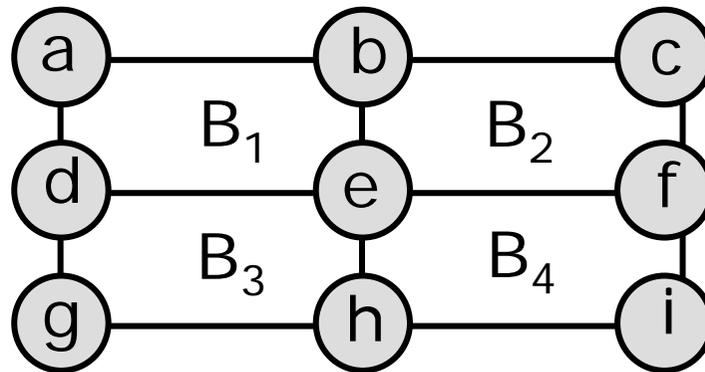
$$F_{di-j} = \int_{F_j} \frac{\cos(\phi_i) \cos(\phi_j)}{\pi r_{ij}^2} b_{ij} dF_j$$



# Näherungslösung [1985]



## von der Fläche zum Eckpunkt



$$B(e) = (B_1 + B_2 + B_3 + B_4)/4$$

$$B(a) = 2 \cdot B_1 - B(e)$$

$$B(c) = 2 \cdot B_2 - B(e)$$

$$B(g) = 2 \cdot B_3 - B(e)$$

$$B(i) = 2 \cdot B_4 - B(e)$$

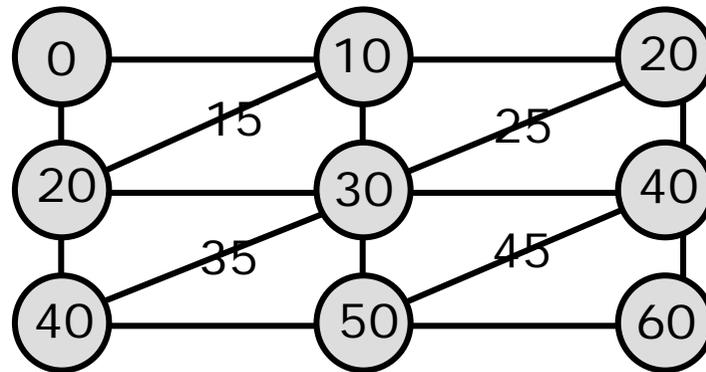
$$B(b) = (B(a) + B(c))/2$$

$$B(d) = (B(a) + B(g))/2$$

$$B(f) = (B(c) + B(i))/2$$

$$B(h) = (B(g) + B(i))/2$$

# Rendern mit Radiosity



Radiosity für Patches berechnen

Radiosity für Eckpunkte interpolieren

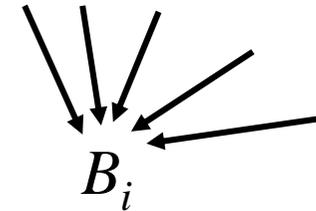
Triangulieren

Einfärben mit Interpolation

# Sammeln und Verteilen

bisher: für Patch  $i$  die Strahlung einsammeln:

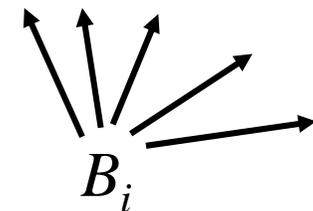
$B_i$  updaten durch  $\sum \rho_j B_j F_{ij}$  für alle  $j$



jetzt: von Patch  $i$  die Strahlung verteilen:

Jedes  $B_j$  updaten durch  $\rho_j B_i F_{ji}$

Jedes  $B_j$  updaten durch  $\rho_j B_i F_{ij} A_i/A_j$



# Progressive Refinement: Initialisierung

$B_i$  momentane Strahlung für Patch  $i$

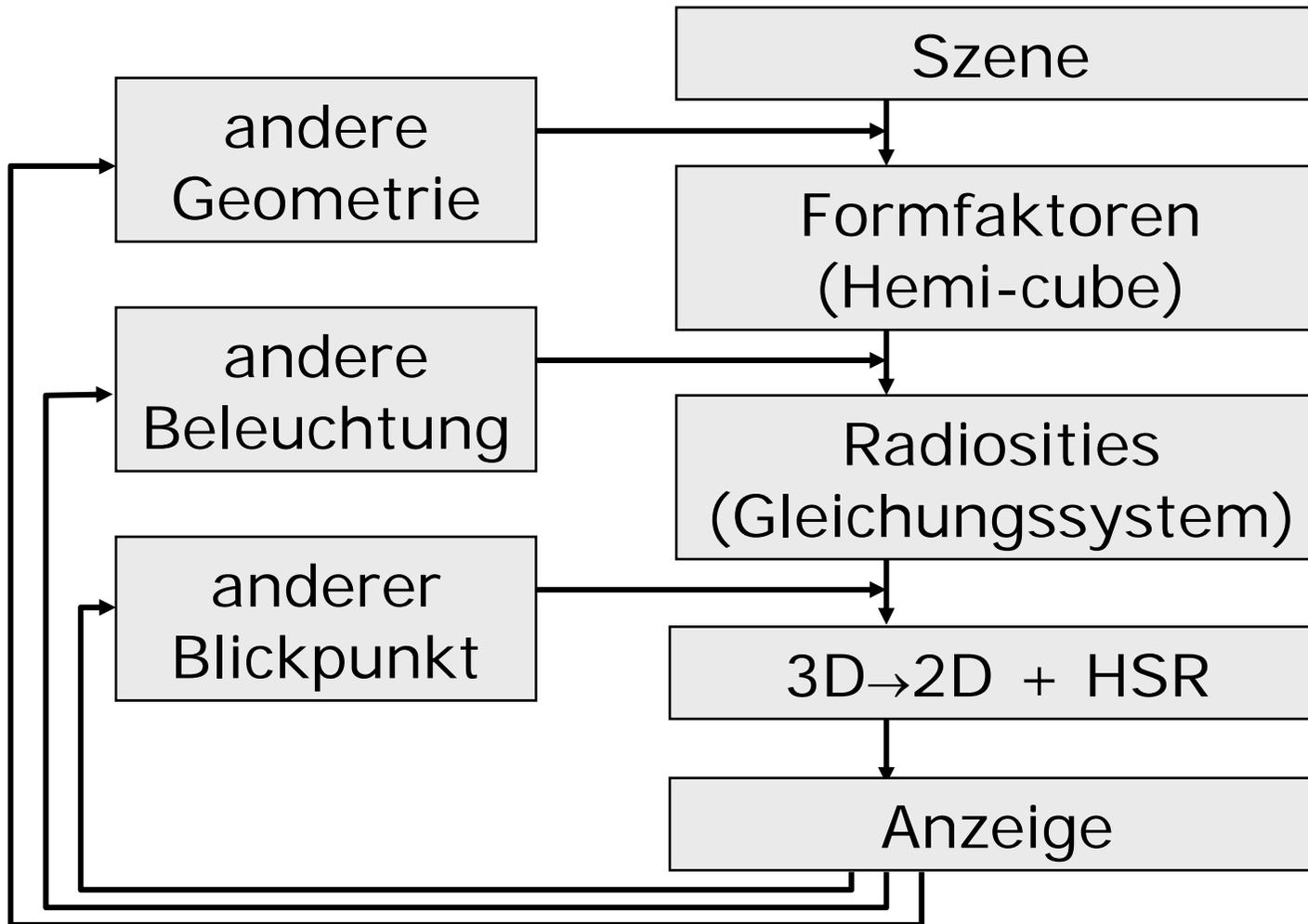
$\Delta B_i$  von Patch  $i$  noch nicht verteilte Strahlung

```
for i=1 to n do {  
  if patch i ist Lichtquelle  
    then  $B_i := \Delta B_i := \text{Emissionswert}$   
    else  $B_i := \Delta B_i := 0$   
}
```

# Progressive-Refinement: Schleife

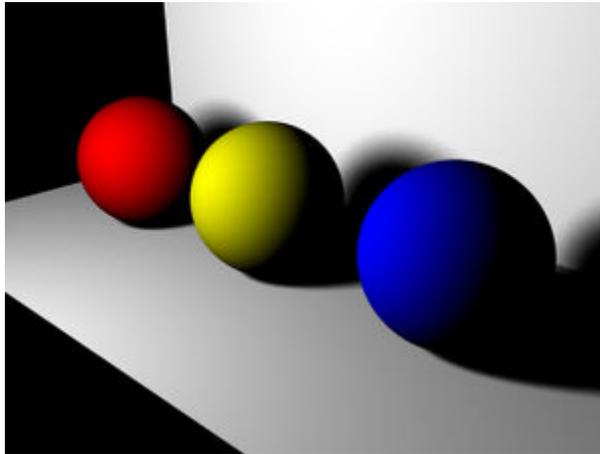
```
while noch nicht zufrieden do {  
  wähle Patch  $i$  mit maximalem  $\Delta B_i A_i$   
  berechne  $F_{ij}$  für alle  $j$   
  for  $j = 1$  to  $n$  do {  
     $\Delta R := \rho_j \cdot \Delta B_i \cdot F_{ij} \cdot A_i / A_j$   
     $\Delta B_j := \Delta B_j + \Delta R$   
     $B_j := B_j + \Delta R$   
  }  
   $\Delta B_i := 0$   
}
```

# Radiosity-Verfahren

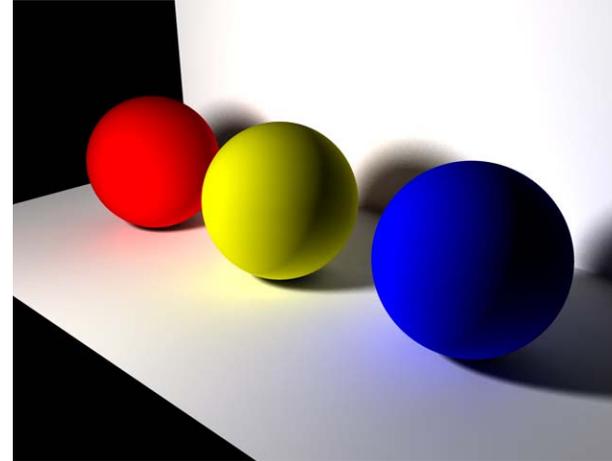


# Vergleich

©Jakob Richter



ohne



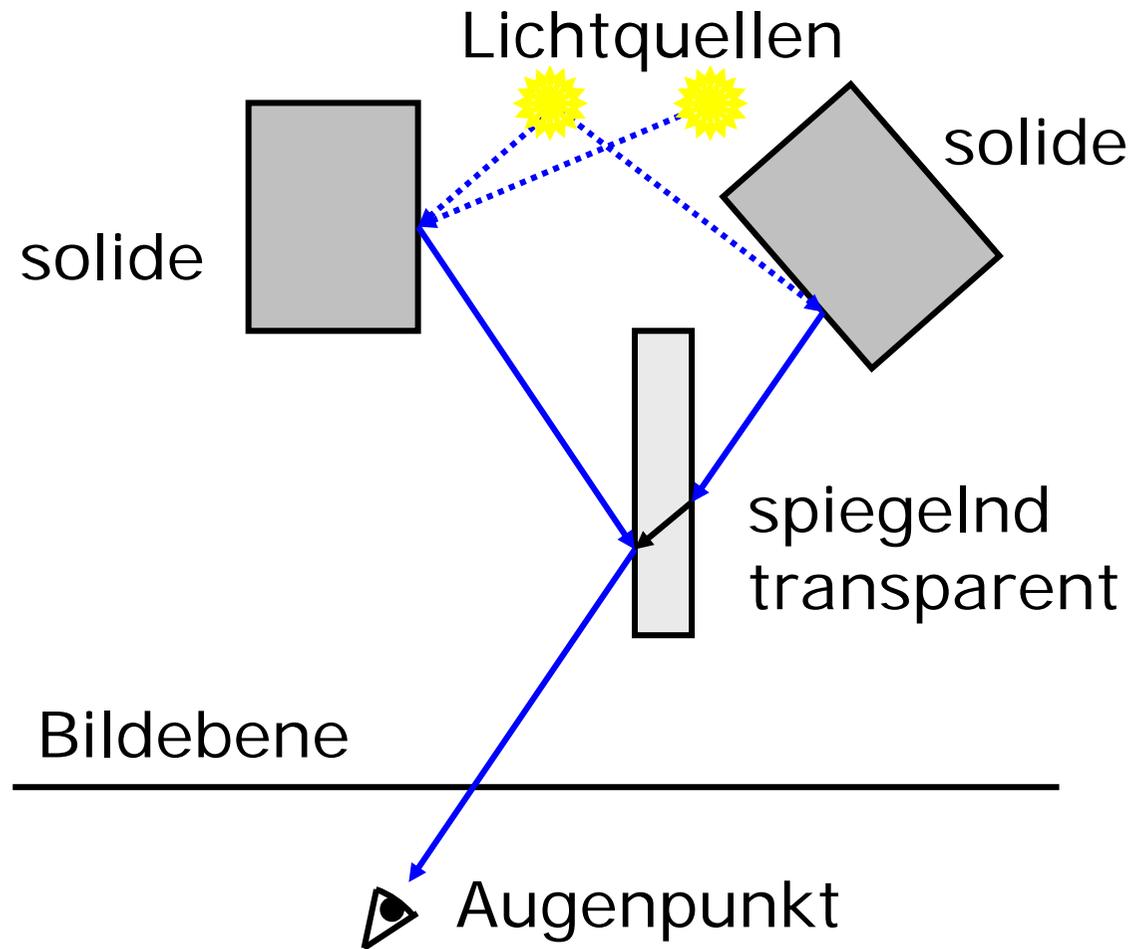
mit

# Radiosity-Beispiel

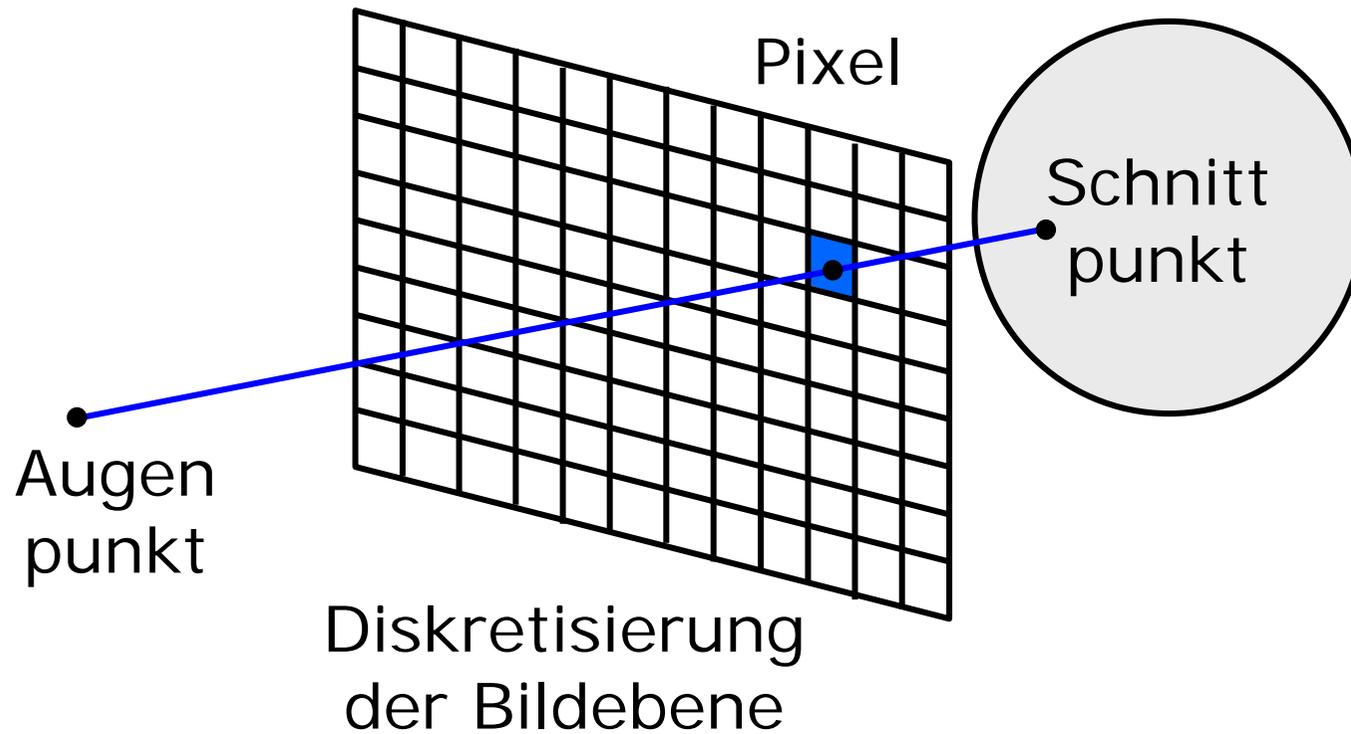
© Cornell University SIGGRAPH 1988



# Raytracing



# Strahl vom Auge durch Bildebene



# Raytracing (ohne Spiegelung)

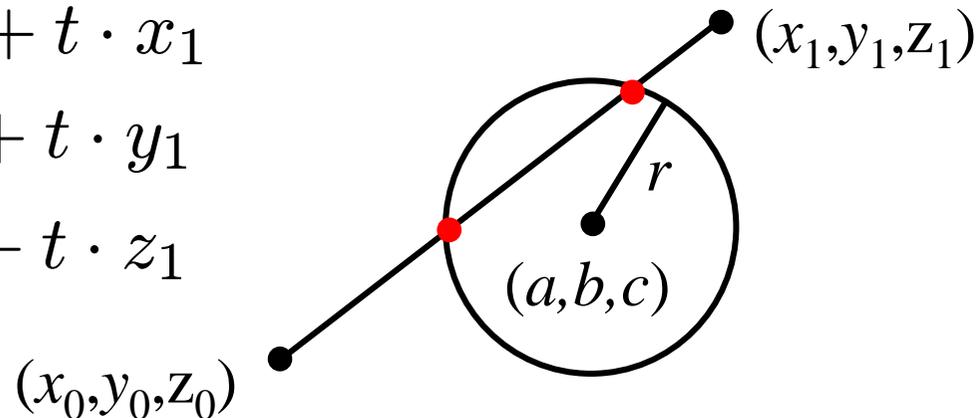
```
Wähle Augenpunkt A und Bildebene B;  
für jedes Pixel P von B tue {  
  berechne Strahl R von A durch P;  
  nächstliegender Schnittpunkt  $S_0 = \infty$ ;  
  für jedes Objekt o der Szene tue {  
    S = Schnittpunkt von R mit o;  
    falls S näher als  $S_0$  {  
       $S_0 = S$ ;  
      färbe P ein unter Verwendung  
        der Normalen von o;  
    }  
  }  
}
```

## Beispiel: Kugel

$$x = (1 - t) \cdot x_0 + t \cdot x_1$$

$$y = (1 - t) \cdot y_0 + t \cdot y_1$$

$$z = (1 - t) \cdot z_0 + t \cdot z_1$$



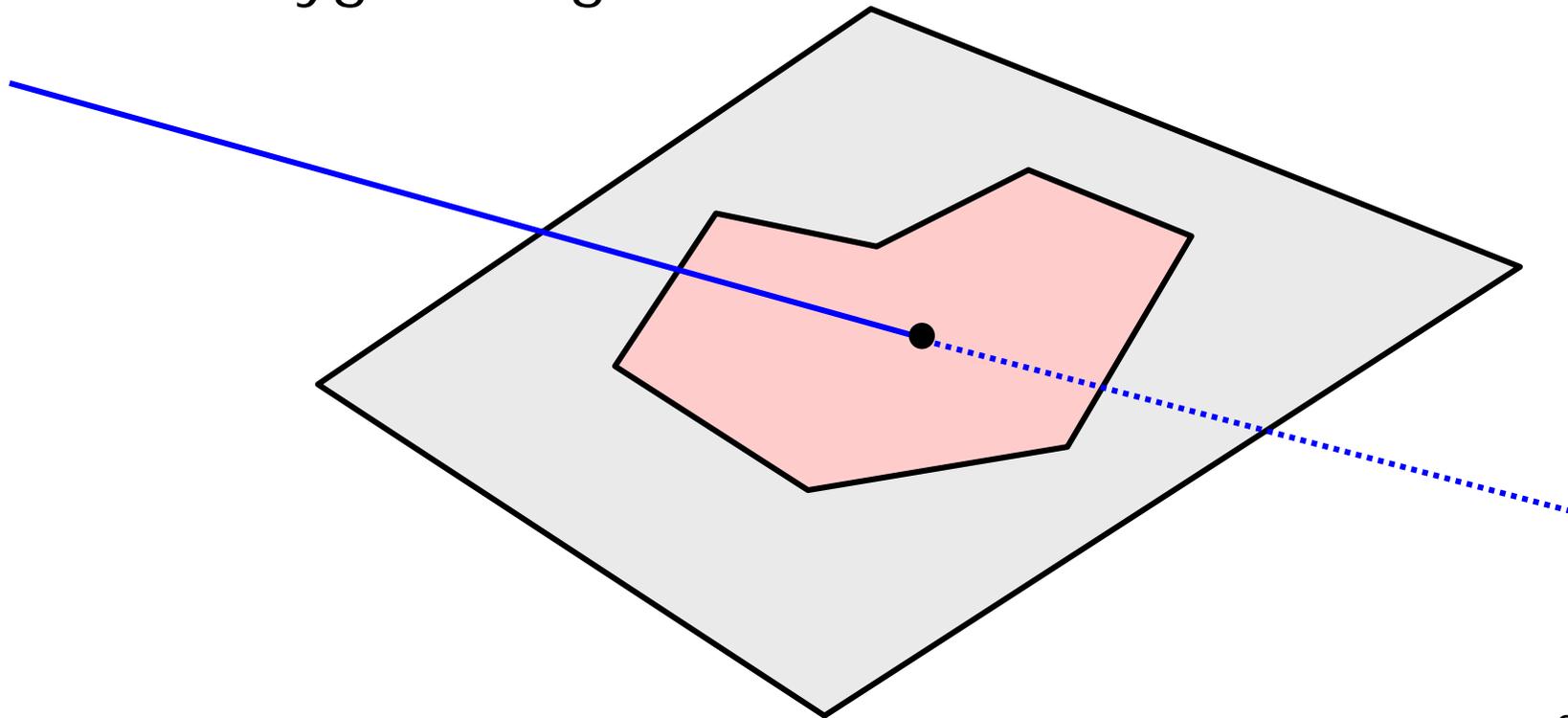
$$(x - a)^2 + (y - b)^2 + (z - c)^2 = r^2$$

Einsetzen liefert quadratische Gleichung in  $t$   
ggf. 0, 1 oder 2 Schnittpunkte  $(x, y, z)$

Normale  $((x - a)/r, (y - b)/r, (z - c)/r)$

# Beispiel: Polygon

- schneide Strahl mit Ebene des Polygons
- Prüfe, ob Schnittpunkt innerhalb des Polygons liegt



# Effizienzsteigerung

- Obacht:  
100 Objekte bei  $1024 \times 768$  verlangen  
100.000.000 Schnittpunktberechnungen
- Schnittpunkte berechnen,  
wenn Sehstrahl = z-Achse
- Begrenzungsvolumina einführen

# Spiegelung und Brechung

$$\vec{N}' = \cos(\phi)\vec{N}$$

$$\vec{v} + \vec{s} = \vec{N}' \quad \vec{s} = \vec{N}' - \vec{v}$$

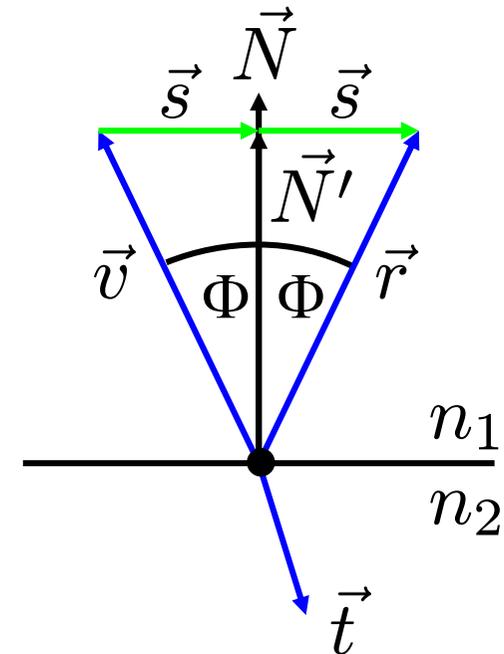
$$\vec{r} = \vec{v} + \vec{s} + \vec{s} = \vec{N}' + \vec{N}' - \vec{v}$$

$$\vec{r} = 2\cos(\phi)\vec{N} - \vec{v}$$

$$\vec{r} = 2(\vec{N} \cdot \vec{v})\vec{N} - \vec{v}$$

$$n = \frac{n_2}{n_1}$$

$$\vec{t} = [-\cos(\phi) - \sqrt{n^2 - 1 + \cos^2(\phi)}]\vec{N} + \vec{v}$$



# Gesamtberechnung

$$I = k_a I_a$$

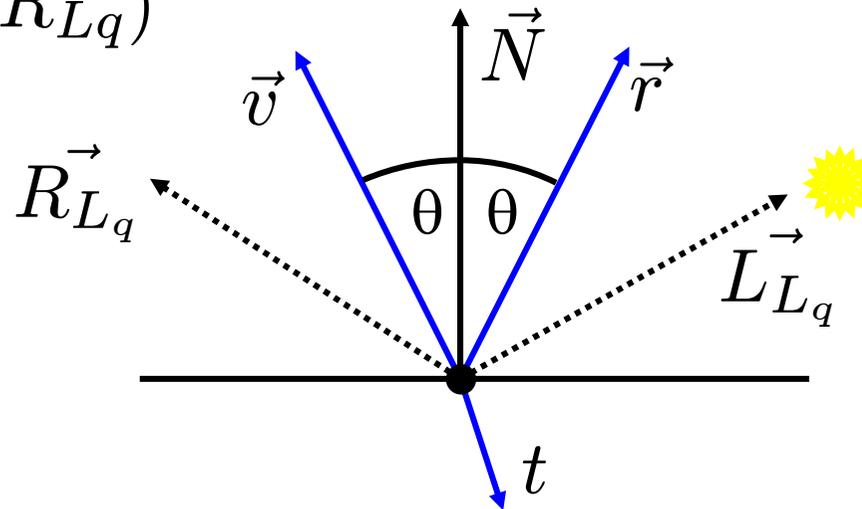
$$+ k_d \sum_{Lq} I_{Lq} (\vec{N} \cdot \vec{L}_{Lq})$$

$$+ k_s \sum_{Lq} I_{Lq} (\vec{v} \cdot \vec{R}_{Lq})^c$$

$$+ k_s I_r$$

$$+ k_t I_t$$

Hindernisse  
berücksichtigen !



# Rekursives Raytracing

```
main () {  
    Wähle Augenpunkt A und Bildebene B;  
    für jedes Pixel P von B tue {  
        berechne Strahl v von A durch P;  
        färbe P mit RT_intersect(v, 1);  
    }  
}
```

# RT\_Intersect

```
Color RT_Intersect(Ray v, int depth){
    berechne den Schnittpunkt p von Strahl v
    zum nächstliegenden Objekt o;
    falls vorhanden {
        berechne Normale N im Schnittpunkt p;
        return RT_Shade(o,v,p,N,depth);
    } else {
        return HINTERGRUND_FARBE;
    }
}
```

# RT\_shade

```
Color RT_Shade (Object o, Ray v, Point P, Normal N, int depth){

    Color C = ambientes Licht;
    für jede nicht geblockte Lichtquelle tue {
        C = C + diffuses Licht in P;
        C = C + spekulares Licht in P;
    }

    if (depth < MAX) {
        if (o reflektiert) {
            R = Strahl in Reflektionsrichtung;
            RC = RT_intersect(R, depth+1);
            C = C + RC * o.ks;
        }

        if (o ist transparent){
            T = Strahl in Brechungsrichtung;
            TC = RT_intersect(T, depth+1);
            C = C + TC * o.kt;
        }
    }
    return C;
}
```

# Persistence of Vision Ray Tracer



[www.povray.org](http://www.povray.org)

[scene.pov](http://scene.pov)

Povray 3.7

[Hall of Fame](#)