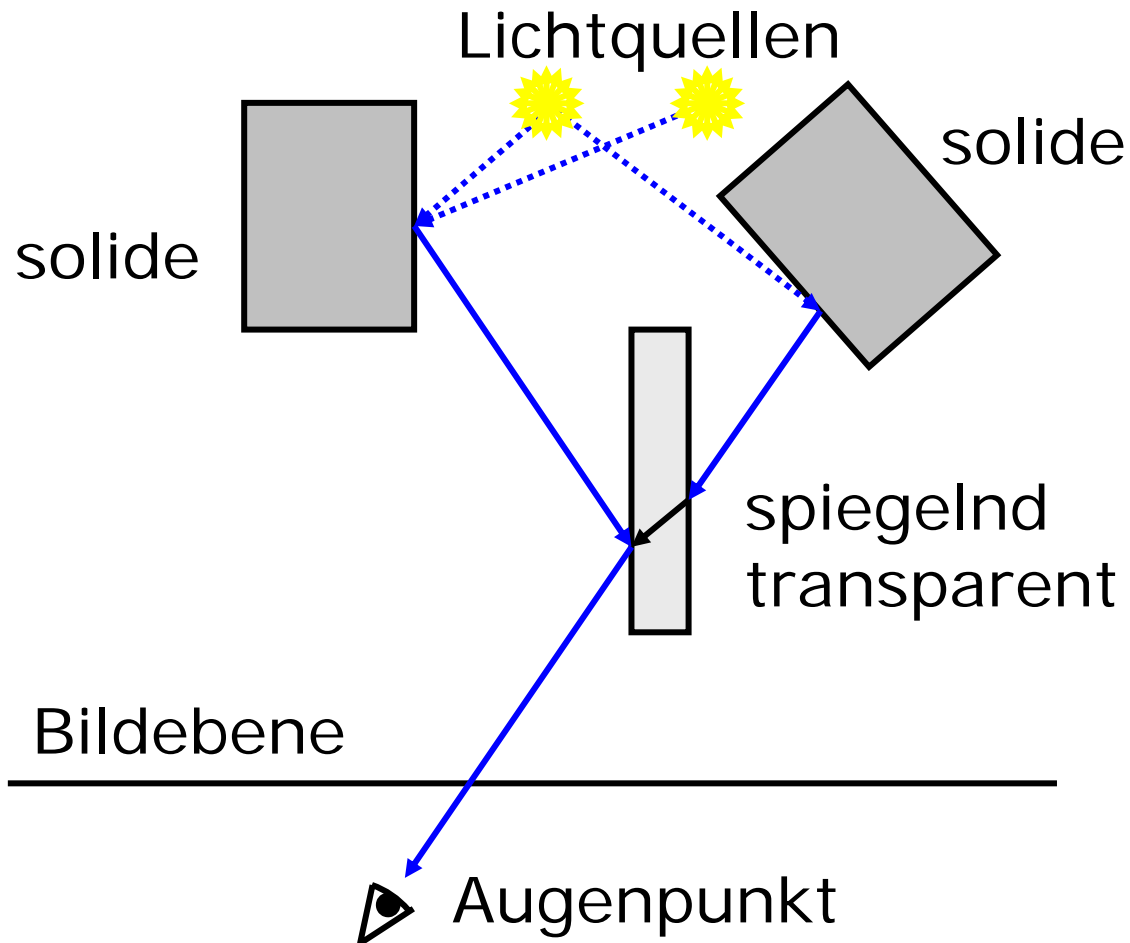
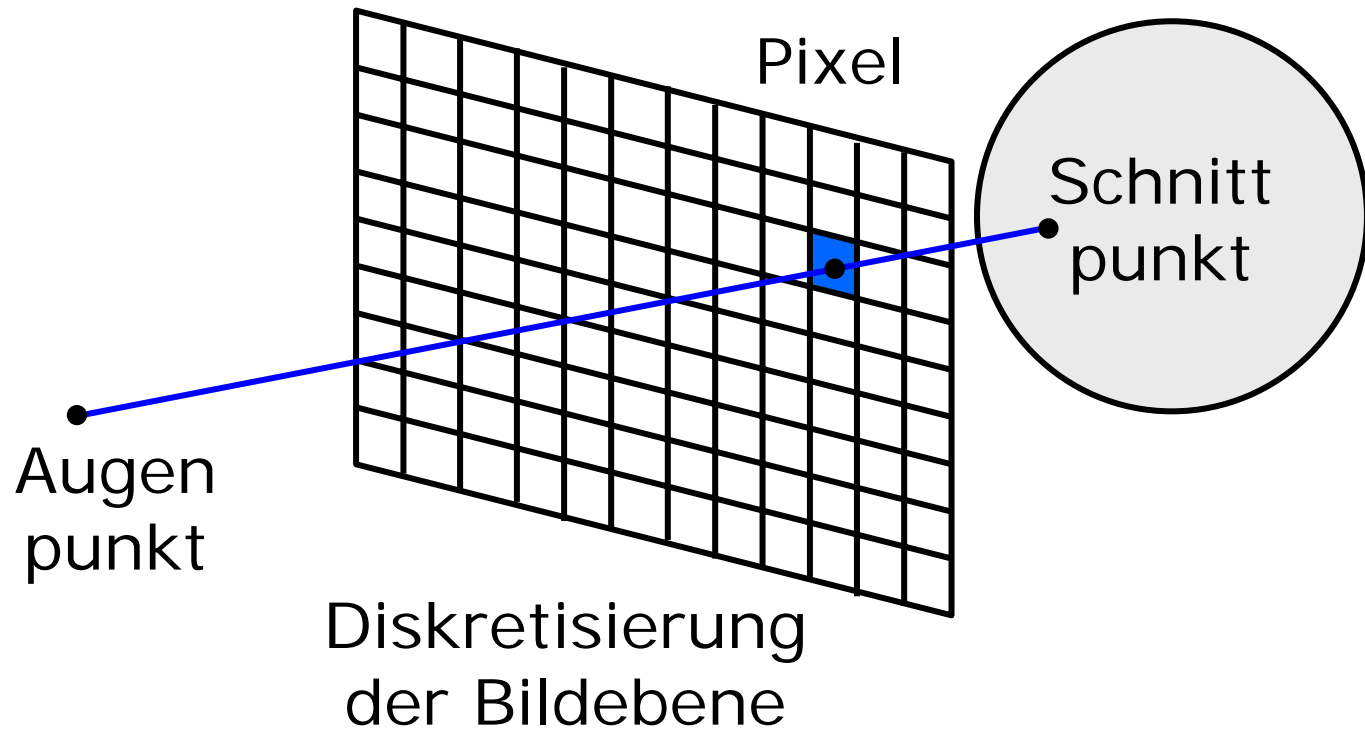


# Kapitel 23: Ray Tracing

# Strahlverfolgung



# Strahl vom Auge durch Bildebene



# Raytracing (ohne Spiegelung)

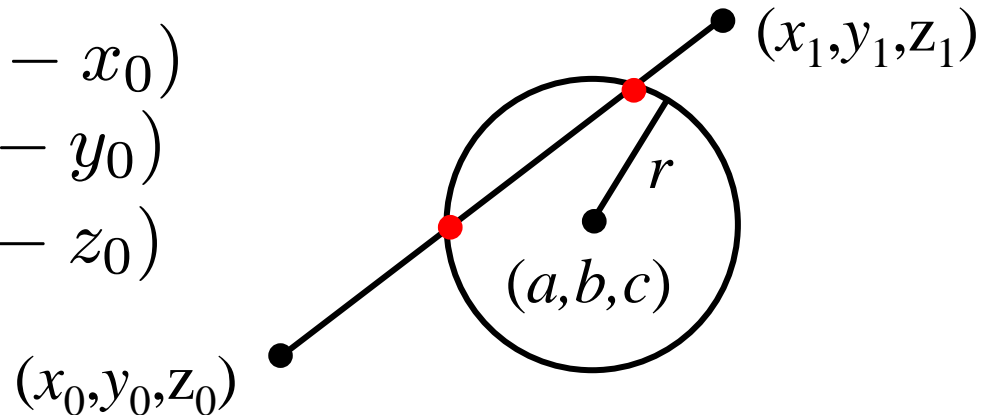
```
Wähle Augenpunkt A und Bildebene B;  
für jedes Pixel P von B tue {  
  berechne Strahl R von A durch P;  
  nächstliegender Schnittpunkt  $S_0 = \infty$ ;  
  für jedes Objekt o der Szene tue {  
    S = Schnittpunkt von R mit o;  
    falls S näher als  $S_0$  {  
       $S_0 = S$ ;  
      färbe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

# Beispiel: Kugel

$$x = x_0 + t(x_1 - x_0)$$

$$y = y_0 + t(y_1 - y_0)$$

$$z = z_0 + t(z_1 - z_0)$$



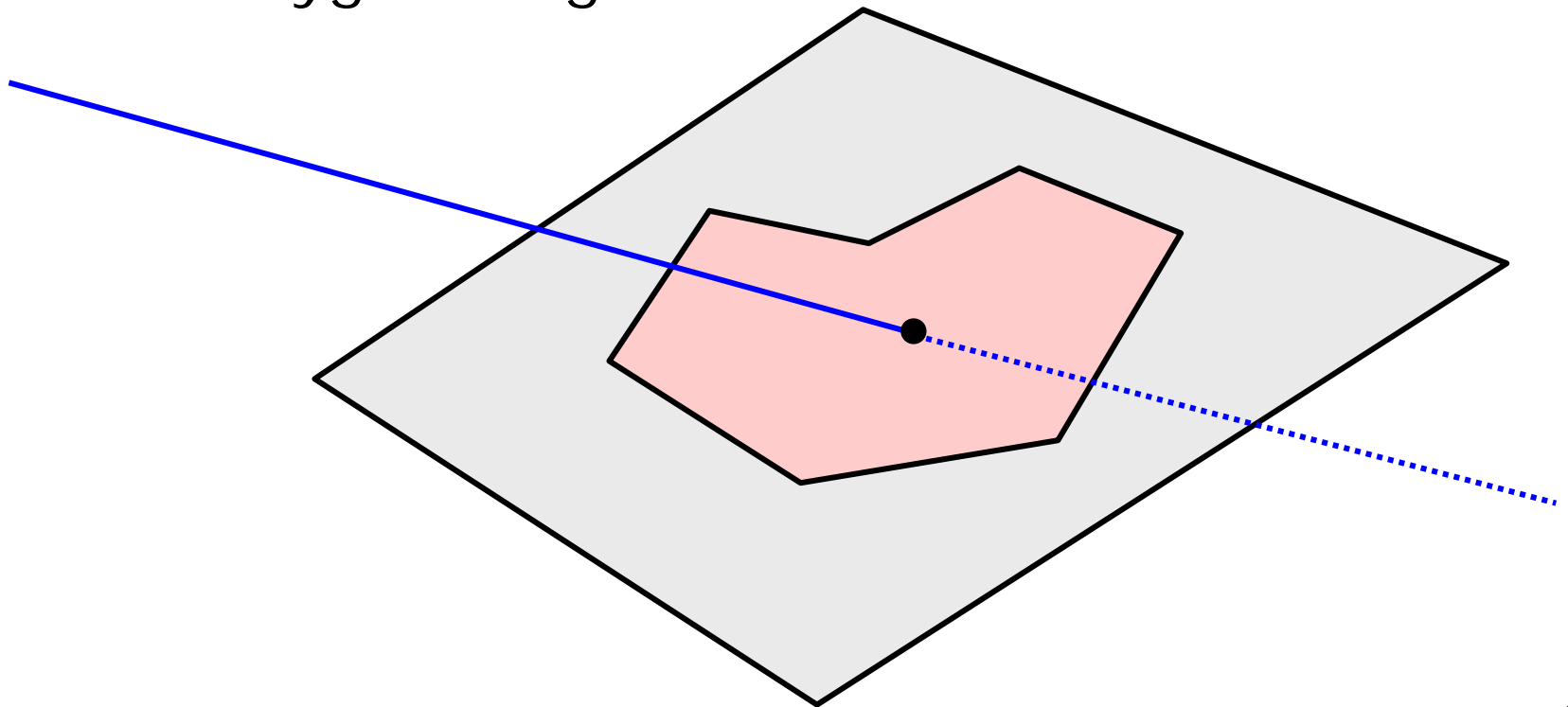
$$(x - a)^2 + (y - b)^2 + (z - c)^2 = r^2$$

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

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

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

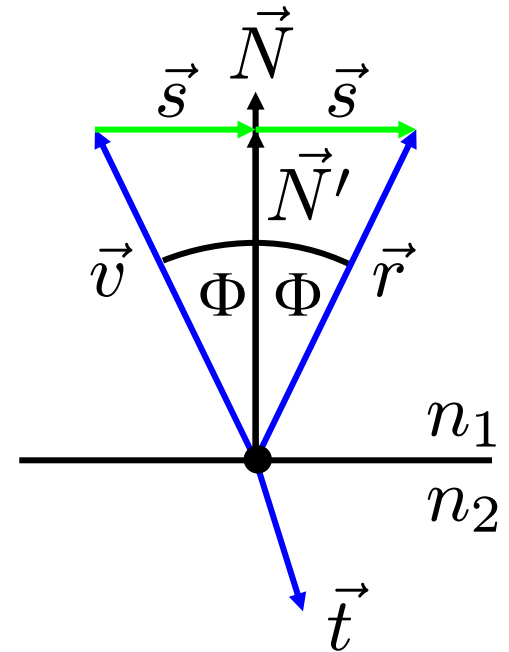
$$\vec{r} = \vec{v} + \vec{s} + \vec{s} = \vec{N}' + \vec{N}' - \vec{v}$$

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

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

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

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





# Gesamtberechnung

$$I = k_a I_a$$

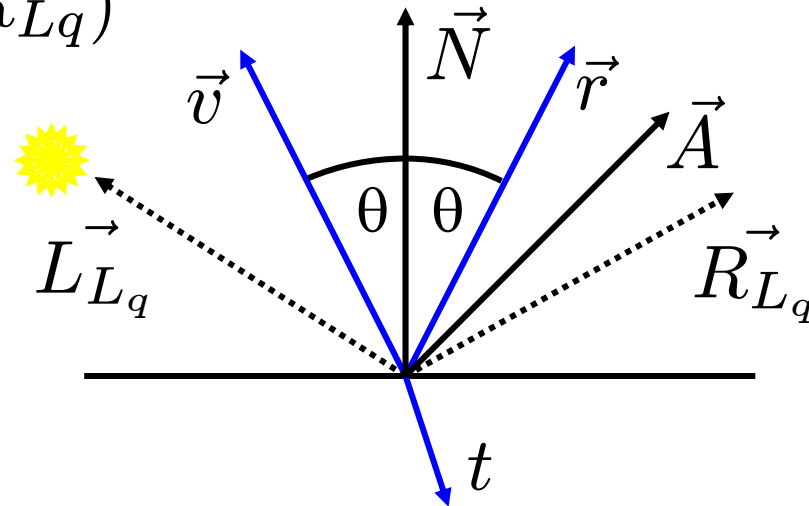
$$+ k_d \sum_{Lq} I_{Lq} (\vec{N} \cdot \vec{L}_{Lq})$$

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

$$+ k_s I_r$$

$$+ k_t I_t$$

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 s von Strahl v
    zum nächstliegenden Objekt o;
    falls vorhanden {
        berechne Normale N im Schnittpunkt;
        return Shade(o,v,s,N,depth);
    } else {
        return HINTERGRUND_FARBE;
    }
}
```

# RT\_shade, Teil 1

```
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) {  
        // Teil 2: Reflektion + Brechung  
    }  
    return C;  
}
```

## RT\_Shade, Teil 2

```
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;  
}
```

# Persistence of Vision Ray Tracer



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

[scene.pov](#)

Povray 3.6