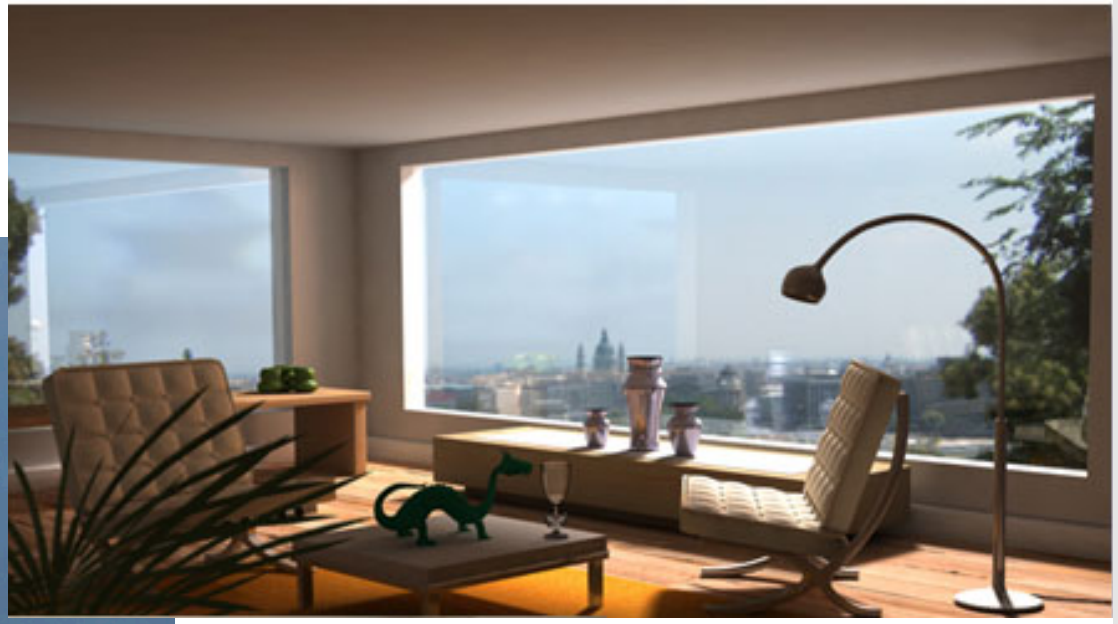
Welches ist die "richtige" Normale?

- Klappt die korrekte Berechnung des reflektierten und des gebrochenen Strahls auch, wenn die Normale in die "falsche" Richtung zeigt?

Was können wir hier noch nicht simulieren?



Fresnel-Terme

- Beim Wechsel von einer Materie in eine andere wird immer ein Anteil Licht reflektiert, der restliche Anteil gebrochen
- Der Reflexionskoeffizient ρ hängt ab vom Brechungsindex der beiden Materialien und vom Einfallswinkel:

$$\rho_{\parallel} = \frac{n_2 \cos\theta_1 - n_1 \cos\theta_2}{n_2 \cos\theta_1 + n_1 \cos\theta_2}$$

$$\rho_{\perp} = \frac{n_1 \cos\theta_1 - n_2 \cos\theta_2}{n_2 \cos\theta_1 + n_1 \cos\theta_2}$$

$$\rho = \frac{1}{2} \cdot \left(\rho_{\parallel}^2 + \rho_{\perp}^2 \right)$$

- Beispiel:
  - Luft (n = 1.0) nach Glas (n = 1.5), senkrechter Lichteinfall:

$$\rho_{\parallel} = \frac{1.5-1}{1.5+1} = \frac{1}{5} \quad \rho_{\perp} = \frac{1-1.5}{1.5+1} = \frac{1}{5} \quad \rho = \frac{1}{2} \cdot \frac{2}{25} = 4\%$$

  - D.h., beim Übergang von Luft nach Glas wird 4% des Lichtes reflektiert, der Rest gebrochen
- Approximation der Fresnel-Terme [Schlick 1994]:

$$\rho(\theta) \approx \rho_0 + (1-\rho_0)\,(1-\cos\theta)^5$$

$$\rho_0 = \left(\frac{n_2-1}{n_2+1}\right)^2$$

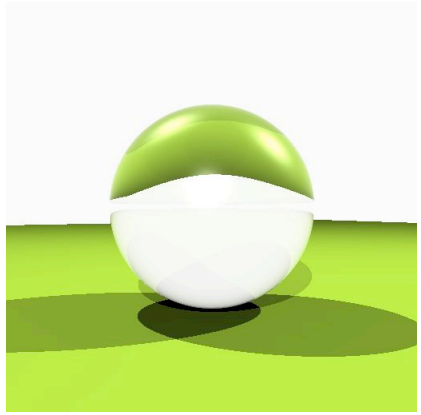
wobei ρ_0 der Fresnel-Term des senkrechten Lichteinfalls ist und θ der Winkel im dünneren Medium (also der größere).

  - 1-ρ ergibt dann den transmittierten Anteil

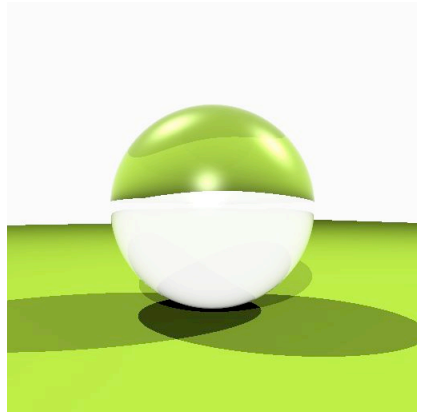
Beispiel für Brechung unter Berücksichtigung der Fresnel-Terme

n=1.0

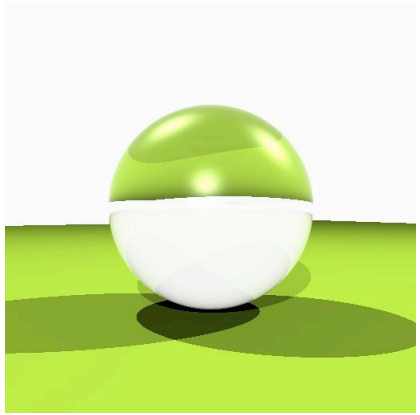
n=1.1



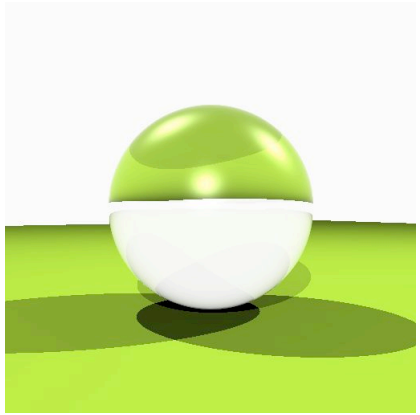
n=1.2



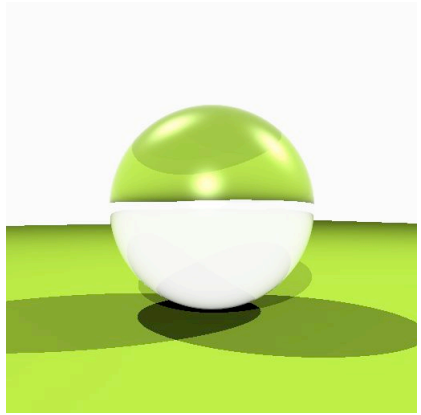
n=1.3



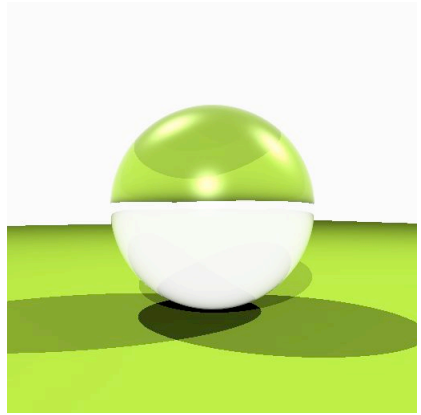
n=1.4



n=1.5



n=1.6

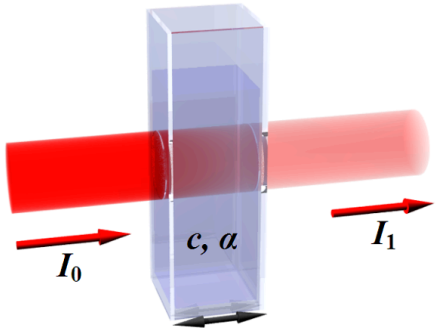


n=1.7

Dämpfung im Medium

- Die durch ein Medium transportierte Lichtintensität schwächt sich mit zunehmender Entfernung gemäß dem Lambert-Beer'schen Gesetz ab:

$$I(s) = I_0 e^{-\alpha s}$$

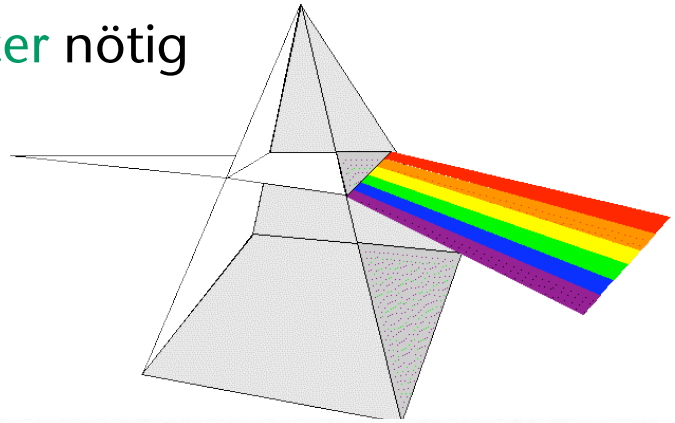


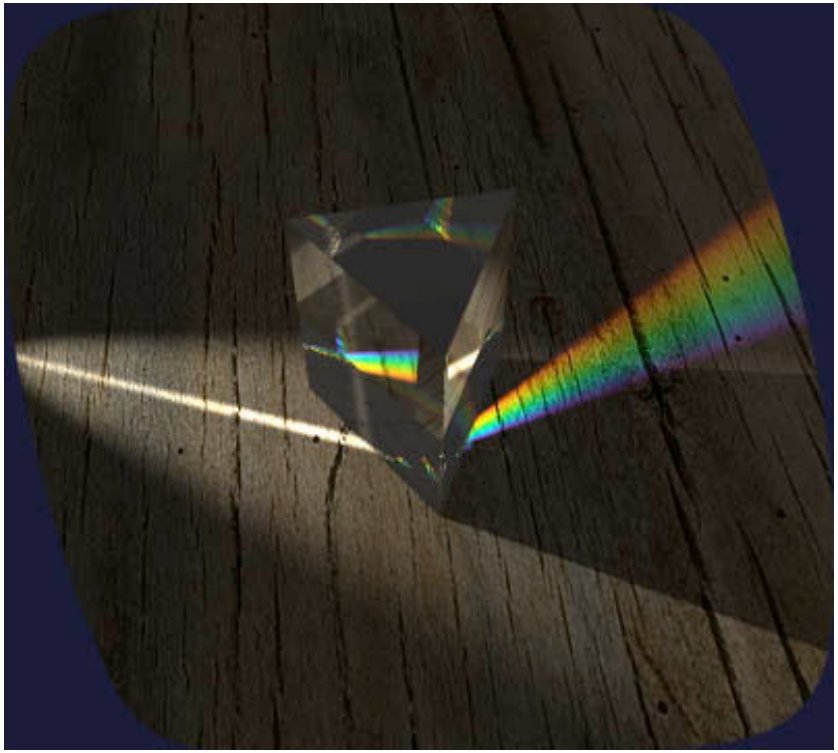
wobei α eine Materialkonstante ist
und s der im Medium zurückgelegte Weg.

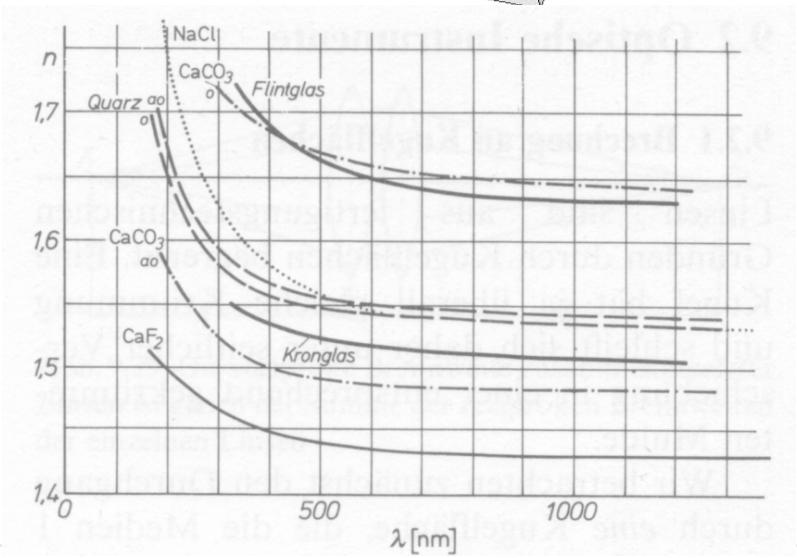
  - α kann auch von der Wellenlänge abhängen

Dispersion

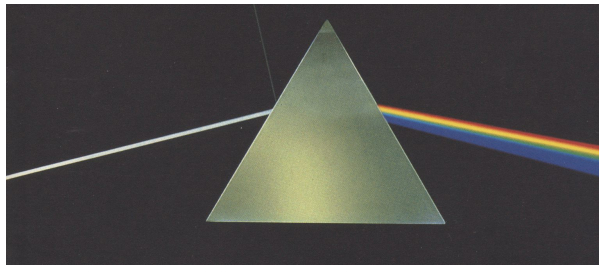
- Brechungsindex ist abhängig von der Wellenlänge
- Diese Effekte lassen sich allerdings in RGB nicht mehr abbilden; hierzu wäre ein „spektraler“ Ray-Tracer nötig







Giovanni Battista Pittoni, 1725,
"An Allegorical Monument to Sir Isaac Newton"



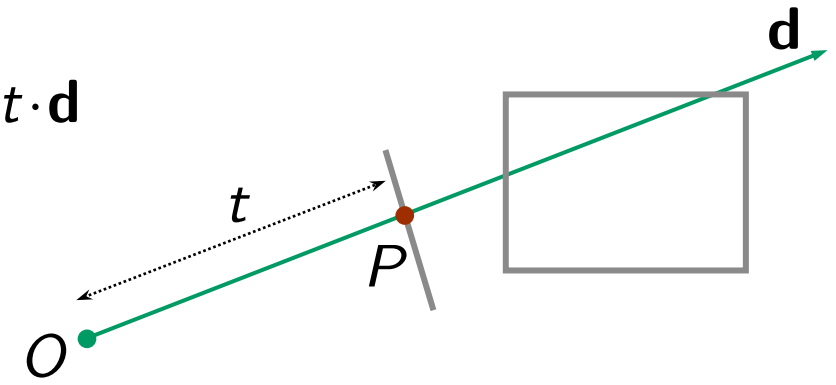
Pink Floyd, *The Dark Side of the Moon*

Beispiel mit Fresnel-Term und Dispersion

Schnittberechnungen

- Der wesentliche Bestandteil der Rechenzeit
- Gegeben: Menge Objekte (Polygone, Kugeln, ...)
  und Strahl

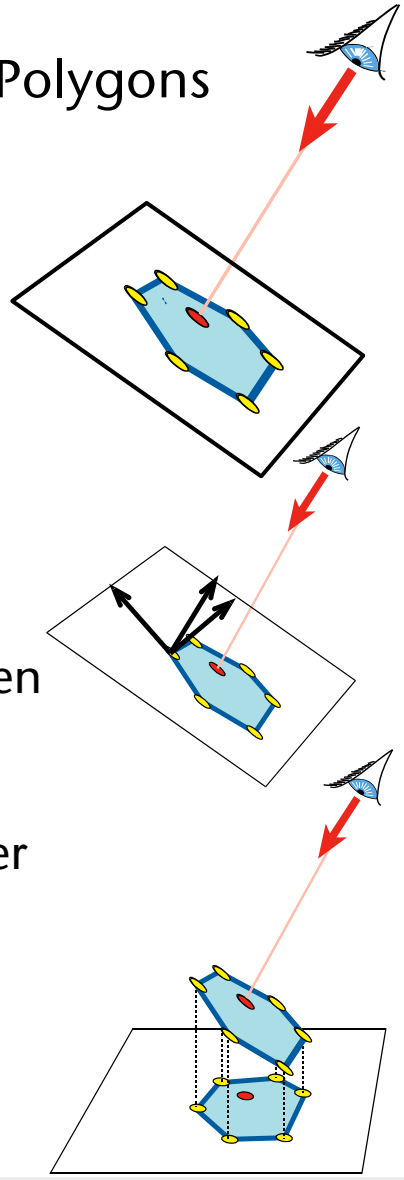
$$P(t) = O + t \cdot \mathbf{d}$$

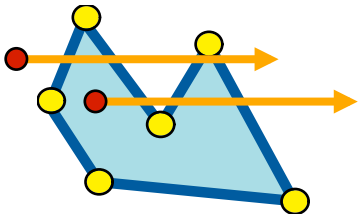


- Gesucht: Linien-
  parameter t
  des ersten Schnitt-
  punktes $P = P(t)$ mit der Szene

Schnitt Strahl—Polygon

- Schneide Strahl (parametrisch) gegen Ebene des Polygons (implizit) → Punkt
- Teste "Punkt in Polygon"
  - Dieser Test findet ausschließlich in der Ebene statt
  - 3D-Punkt in 3D-Polygon ↔ 2D-Punkt in 2D-Pgon
- Projiziere Punkt & Polygon
  - Entlang der Normale: zu teuer
  - Auf Koord.ebene: einfach eine der 3 Koord fallenlassen
- Test "Punkt in Polygon":
  - Zähle Anzahl Schnitte zwischen Strahl und Rand; oder
  - Bestimme "Winding Number"





Interludium: Die vollständige Ray-Tracing-Routine

```
traceRay( ray ):
  hit = intersect( ray )
  if no hit:
    return no color
  reflected_ray = reflect( ray, hit )
  reflected_color = traceRay( reflected_ray )
  refracted_ray = refract( ray, hit )
  refracted_color = traceRay( refracted_ray )
  for each lightsource[i]:
    shadow_ray = calcLightFeeler( hit, lightsource[i] )
    if intersect(ray):
      light_color[i] = 1
  overall_color = shade( hit,
                         reflected_color,
                         refracted_color,
                         light_color )
  return overall_color
```

hit ist eine Datenstruktur, die alle Infos über einen Schnitt zwischen Strahl und Szene enthält, u.a. Schnittpunkt, Objekt, Normale, ...

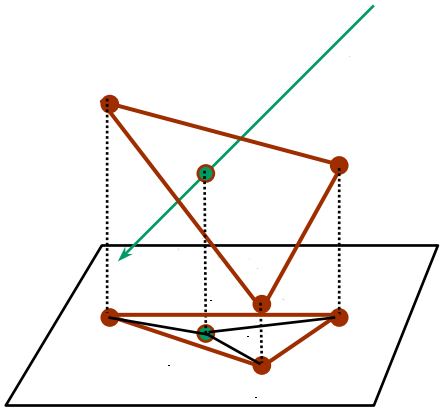
Diese **intersect**-Funktion kann deutlich optimiert werden gegenüber der obigen; außerdem interessiert nur ein Schnitt **vor** der Lichtquelle.

Wertet die Beleuchtungsgleichung für das getroffene Obj aus.

Schnitt Strahl—Dreieck

[Badouel 1990]

- Verwende Methode Strahl—Polygon; oder
- Cleverer sein: baryzentrische Koordinaten + Projektion
- Schneide Strahl mit Ebene (Normalenform) → t → Punkt
- Projiziere Punkt & Dreieck in Koord.ebene
- Berechne baryzentrische Koord. des 2D-Punktes
- Baryzentrische Koord. des 2D-Punktes = baryzentrische Koord. des 3D-Punktes!



- 3D-Punkt in Dreieck ↔ $\alpha,\beta,\gamma > 0$, $\alpha+\beta+\gamma < 1$
- Alternative Methode: siehe Möller & Haines "Real-time Rendering"
- Code: http://jgt.akpeters.com/papers/MollerTrumbore97/
- Geht noch schneller, falls Schnittpunkt nicht nötig [Segura & Feito]

Alternative Schnittberechnung Strahl—Dreieck

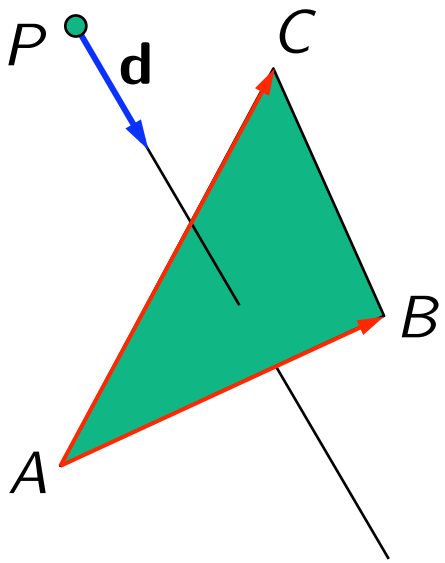
[Möller]

- Geradengleichung $X = P + t \cdot \mathbf{d}$
- Ebenengleichung $X = A + r \cdot (B - A) + s \cdot (C - A)$
- Gleichsetzen $-t \cdot \mathbf{d} + r \cdot (B - A) + s \cdot (C - A) = P - A$
- In Matrixschreibweise $\begin{pmatrix} -\mathbf{d} & B - A & C - A \end{pmatrix} \cdot \begin{pmatrix} t \\ r \\ s \end{pmatrix} = P - A$

$\mathbf{u} = B - A$

$\mathbf{v} = C - A$

$\mathbf{w} = P - A$



$$\begin{pmatrix} t \\ r \\ s \end{pmatrix} = \frac{1}{\det(-\mathbf{d}, \mathbf{u}, \mathbf{v})} \cdot \begin{pmatrix} \det(\mathbf{w}, \mathbf{u}, \mathbf{v}) \\ \det(-\mathbf{d}, \mathbf{w}, \mathbf{v}) \\ \det(-\mathbf{d}, \mathbf{u}, \mathbf{w}) \end{pmatrix}$$

$$\det(\mathbf{a}, \mathbf{b}, \mathbf{c}) = \mathbf{a} \cdot (\mathbf{b} \times \mathbf{c})$$

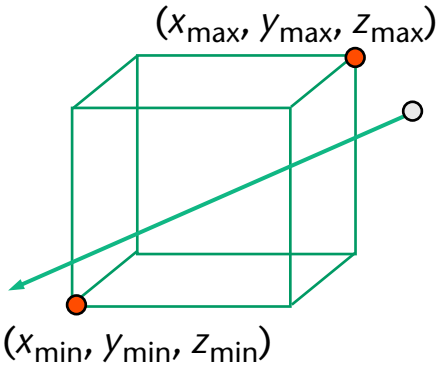
$$\begin{pmatrix} r \\ r \\ s \end{pmatrix} = \frac{1}{(\mathbf{d} \times \mathbf{v}) \cdot \mathbf{u}} \cdot \begin{pmatrix} (\mathbf{w} \times \mathbf{u}) \cdot \mathbf{v} \\ (\mathbf{d} \times \mathbf{v}) \cdot \mathbf{w} \\ (\mathbf{w} \times \mathbf{u}) \cdot \mathbf{d} \end{pmatrix}$$

- Kosten: 2 Kreuzprodukte + 4 Skalarprodukte
- Liefert: Geradenparameter + baryzentrische Koordinaten bzgl. Dreieck
- Test ob s,t im Bereich (0,1) und s+t <= 1

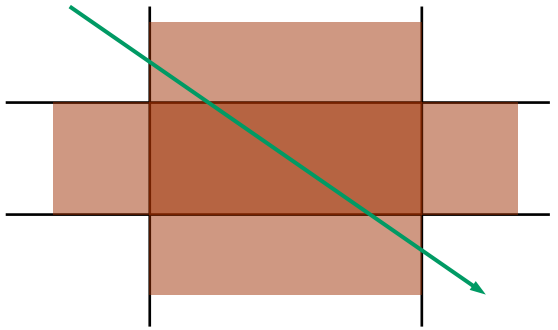
Schnitt Strahl — Box

- Box (Quader) wird später noch wichtig als Bounding Box
- Hier: nur achsenparallele Boxes (AABB = *axis-aligned bounding box*)
- Definition einer AABB: durch die zwei extremen Eckpunkte

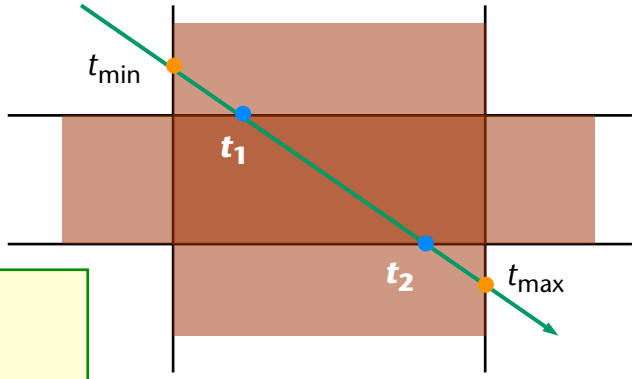
  $(x_{min}, y_{min}, z_{min})$ und $(x_{max}, y_{max}, z_{max})$



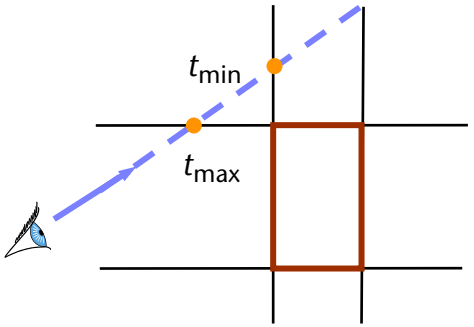
- Idee des Algo:
  - Eine Box ist der Schnitt von 3 *Slabs* (ein *Slab* = Schicht des Raumes, wird von 2 parallelen Ebenen begrenzt)
  - Jeder Slab schneidet vom Strahl ein Intervall heraus
  - Betrachte also sukzessive jeweils Paare von Box-Seiten

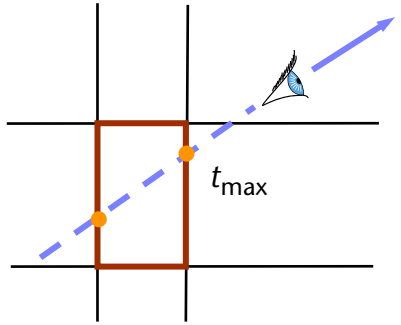


- Der Algorithmus:



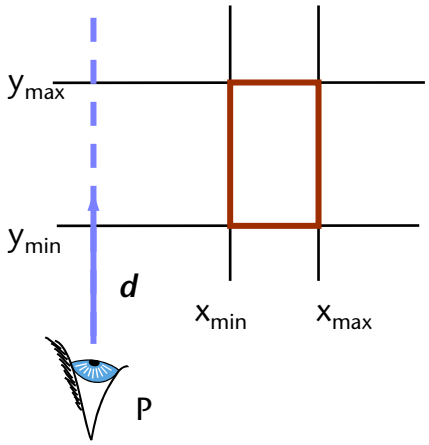

```
setze t_min = -∞ , t_max = +∞
loop über alle Paare von Ebenen:
  schneide Strahl mit den
           beiden Ebenen → t1 , t2
  if t2 < t1:
    vertausche t1 , t2
  // jetzt gilt: t1 < t2
  t_min ← max(t_min, t1 )
  t_max ← min(t_max, t2 )
// now: [t_min,t_max] = interval inside box
if t_min > t_max → kein Schnitt
if t_max < 0    → kein Schnitt
```





Bemerkungen

- Optimierung: beide Ebenen eines Slabs haben dieselbe Normale
  - Spart ein Skalarprodukt
- Bemerkung: der Algo funktioniert genauso für "schiefe" Boxes (sog. *OBBs = oriented bounding boxes*)
- Weitere Optimierung: falls AABB, nutze aus, daß die Normalen nur 1 Komponente ≠ 0 haben!
- Achtung: teste auf Parallelität!
  - "shit happens"
  - Im Fall der AABB:




```
if |d_x| < ε:
  if P_x < x_min || P_x > x_max:
    Strahl geht an Box vorbei
  else:
    t_1, t_2 = y_min, y_max     // evtl noch swappen!
```

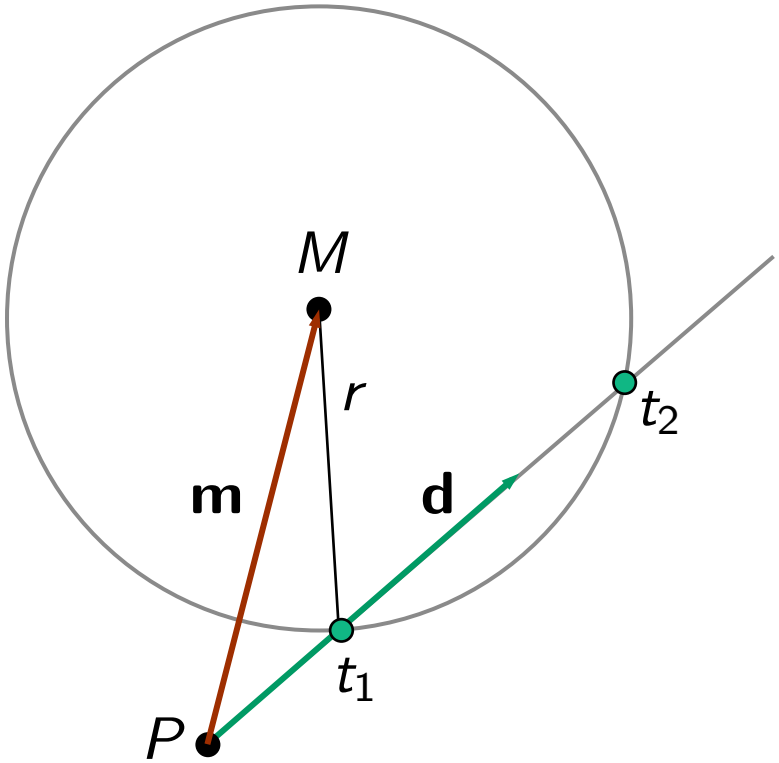
Schnitt Strahl—Kugel

- Annahme: **d** normiert
- Die geometrische Methode:

$$|t\cdot\mathbf{d}-\mathbf{m}| = r$$

$$(t\cdot\mathbf{d}-\mathbf{m})^2 = r^2$$

$$t^2 - 2t\cdot\mathbf{md} + \mathbf{m}^2 - r^2 = 0$$

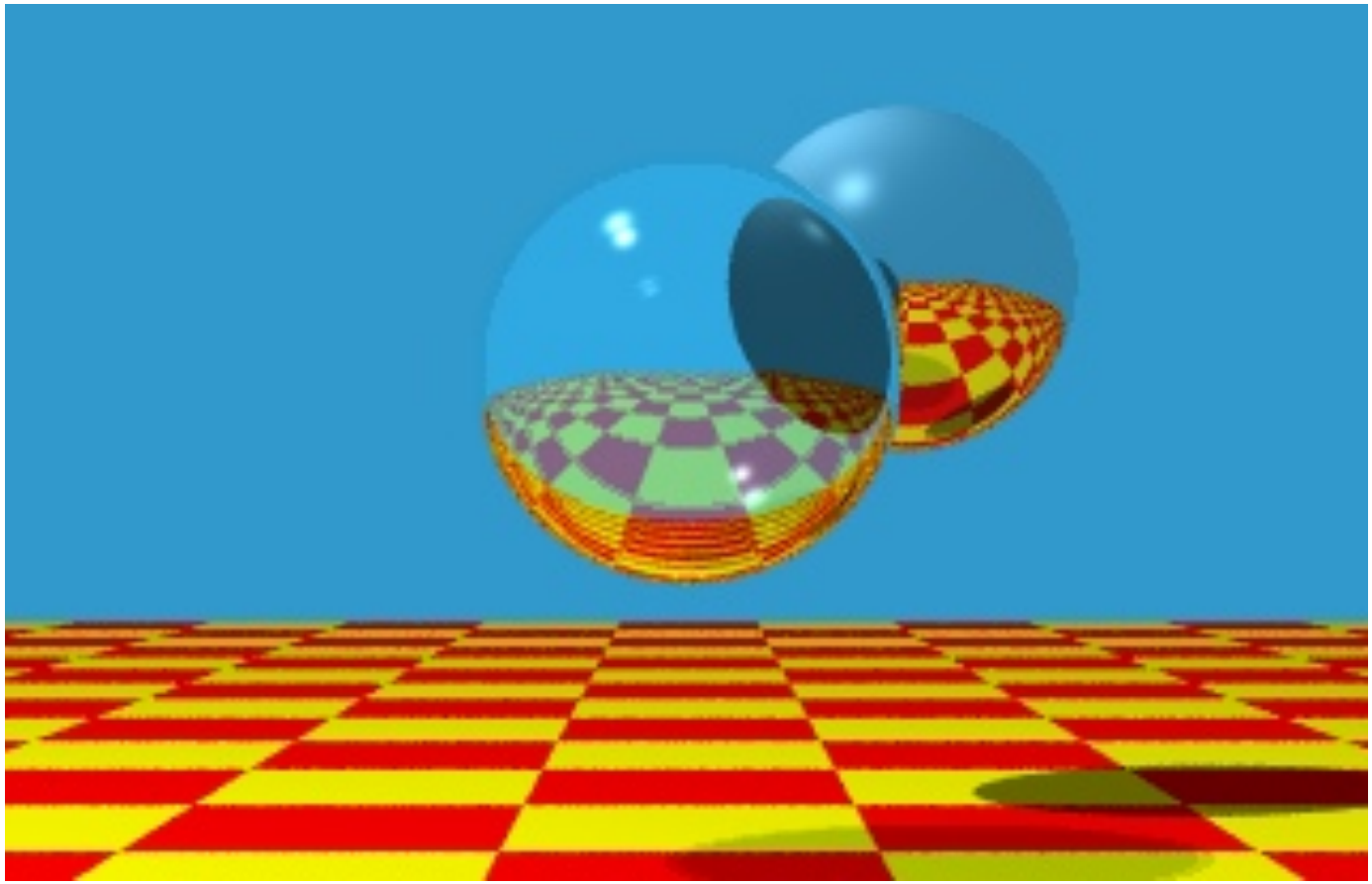


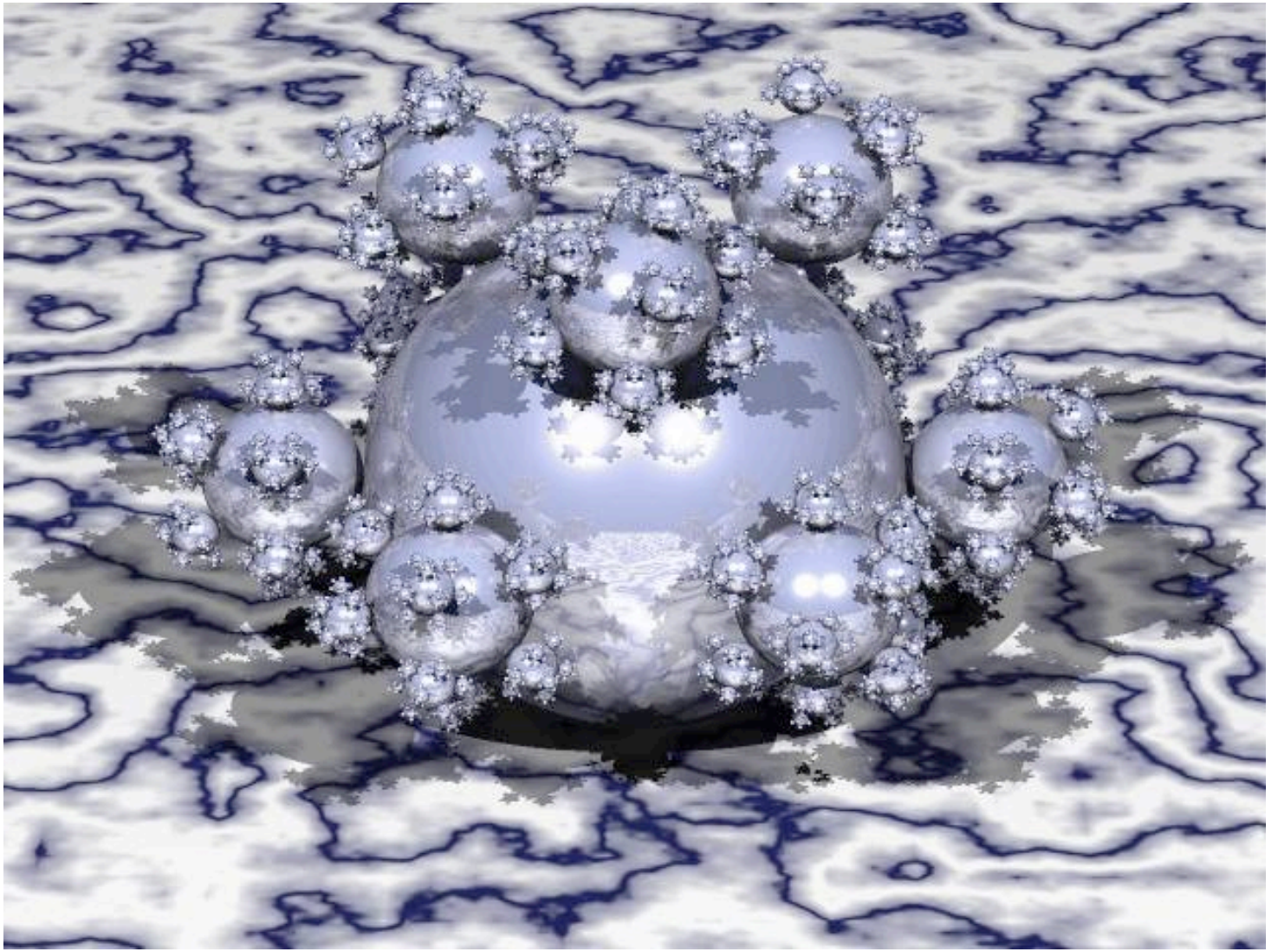
- Die algebraische Methode:
  Punkt auf Strahl in implizite Kugelgleichung einsetzen
- Es gibt noch andere Ansätze ...

- Der Algorithmus, mit kleinen Optimierungen:

```
berechne m^2−r^2
berechne b = m·d
if  m^2−r^2 >= 0        // Blickpunkt ausserhalb Kugel
    and b <= 0 :        // sieht von Kugel weg
then
     return "kein Schnittpunkt"
setze d = b^2 − m^2 + r^2
if d < 0:
    return "kein Schnittpunkt"
if  m^2−r^2 > ε :
    return t_1 = b − √d  // enter; l_1 > 0
else:
    return t_2 = b + √d  // leave; l_2 > 0
```

- Es ist so einfach, daß alle Ray-Tracer Kugeln haben!





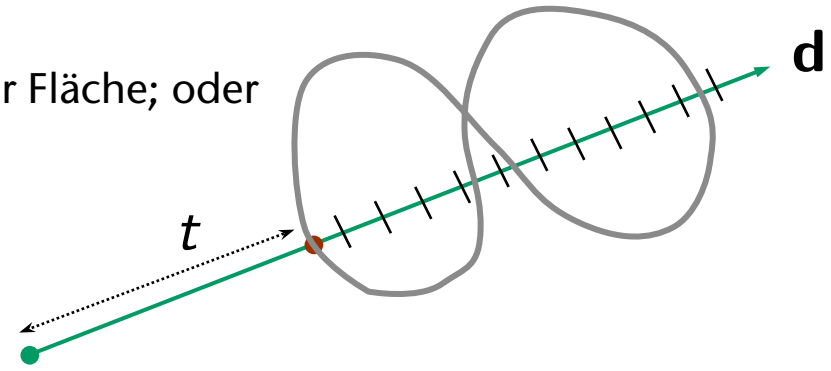
Die sog. "sphere flake"

Geometrisch vs. Algebraisch

- Die algebraische Methode ist einfach und generisch
- Die geometrische Methode ist schneller
  - Durch geometrische Einsicht
  - Frühe Tests
  - Insbesondere für die Strahlen, die weg zeigen

Schnitt Strahl – implizite Fläche

- Implizite Fläche vom Grad n: $F(x, y, z) = 0$
- Strahl: $P(t) = O + t \cdot \mathbf{d}$
- Einsetzen: $F(\,P(t)\,) = 0$
  liefert Polynom in t vom Grad n
- Nullstellensuche:
  - Falls Grad < 5: nach t auflösen
  - Sonst: Intervallschachtelung, Newton-Verfahren, ...
  - Startwerte:
    - Schnitt zwischen Strahl und BBox der Fläche; oder
    - Strahl innerhalb der BBox abtasten
- Z.B.: Kugel ...



Nullstellensuche mit Laguerre's Methode

- Eine von wenigen "*sure-fire*"-Methoden
- Algorithmus erfordert Arithmetik mit komplexen Zahlen, auch wenn alle Wurzeln reell sind (und damit auch alle Koeffizienten)
- Sehr wenig theoretische Erkenntnisse zum Konvergenzverhalten
- Sehr viel empirische Hinweise, daß Algo (fast) **immer** zu einer Wurzel konvergiert, und zwar von (fast) **jedem** Startwert aus!
- Konvergenz-Ordnung ist 3, falls die Wurzel einfach ist

Motivation für den Algorithmus

- Gegeben:

$$P(x) = (x - x_1)(x - x_2)\dots(x - x_n) \tag{0}$$

- Beziehungen:

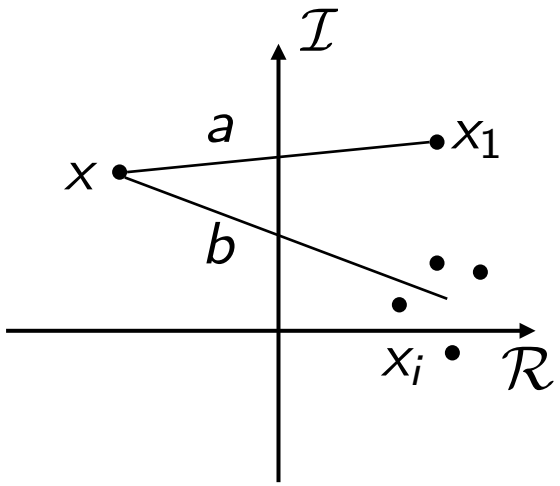
$$\ln|P(x)| = \ln|x - x_1| + \ln|x - x_2| + \cdots + \ln|x - x_n|$$

$$\frac{d}{dx}\ln|P(x)| = \frac{1}{x - x_1} + \cdots + \frac{1}{x - x_n} = \frac{P'(x)}{P(x)} =: G \tag{1}$$

$$\frac{d^2}{dx^2}\ln|P(x)| = -\frac{1}{(x - x_1)^2} - \cdots - \frac{1}{(x - x_n)^2}$$

$$= \frac{P''(x)}{P(x)} - \left(\frac{P'(x)}{P(x)}\right)^2 =: -H \tag{2}$$

- Sei x unsere aktuelle Näherung an Wurzel x_1



- "Drastische" Annahme:
  - Abstand $|x - x_1| = a$
  - Abstand zu allen anderen Wurzeln ist

$$|x - x_i| \approx b, \quad i = 2, 3, \ldots, n$$

- Dann kann man (1) & (2) so darstellen

$$G \approx \frac{1}{a} + \frac{n-1}{b} \tag{3}$$

$$H \approx \frac{1}{a^2} + \frac{n-1}{b^2} \tag{4}$$

- Einsetzen ergibt Lösung für a:

$$a \approx \frac{n}{G \pm \sqrt{(n-1)\,(nH - G^2)}} \tag{5}$$

  - VZ so wählen, daß $|a|$ minimal wird
  - Wurzel kann negativ werden

    $\longrightarrow$ a kann komplex werden
  - Neue Näherung für x_1 ist

$$x_1 = x - a$$

Algorithmus

wähle x_0

loop:

berechne $G = \frac{P'(x_l)}{P(x_l)}$

$H = G^2 - \frac{P''(x_l)}{P(x_l)}$

berechne $a = \frac{n}{G \pm \sqrt{(n-1)(nH - G^2)}}$

setze $x_{l+1} = x_l - a$

until **a "klein genug" oder** $k \geq \max$

- Achtung: möglichst Code aus *Numerical Recipes* verwenden
  - Selbst implementieren ist fehlerträchtig
  - NR-Code hat elegantes Abbruchkriterium
- Für Ray-Tracing: alle Nullstellen berechnen
  - faktorisiere gefunde Nullstelle aus, wiederhole Laguerre n Mal