Computergrafik I
Kapitel 2: Ray Tracing

Wintersemester 2014/2015

Prof. Dr. Timo Ropinski
Forschungsgruppe Visual Computing

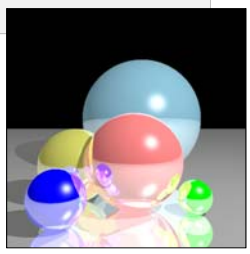
Beispielbilder

Aus der Galerie der Embree Ray Tracing Website
Quelle: http://embree.github.io/

Rendering

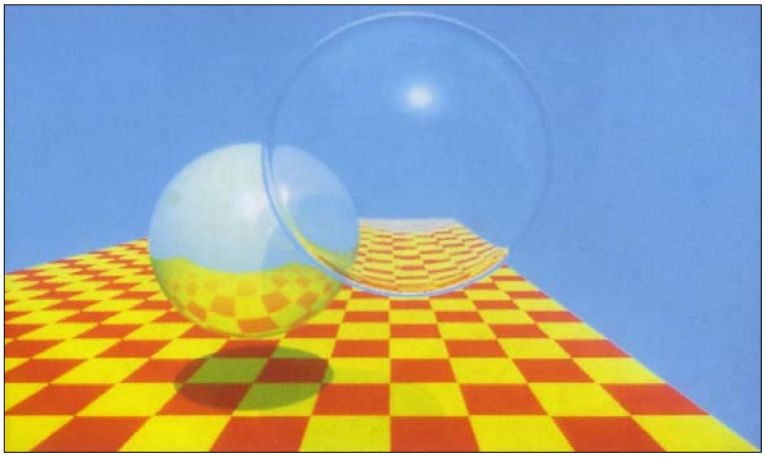
- Überführung einer 3D Szenebeschreibung in ein 2D Bild
  - Erzeuge Abbild der Geometrie
  - Simuliere Lichtinteraktion zwischen Szenenobjekten

```
Scene* scene = new Scene();
scene->addSceneObject(new Plane(Vec3(0.0, 1.0, 0.0), 0.0, Vec3(0.3, 0.3, 0.3)));
scene->addSceneObject(new Sphere(Vec3(0.05, 0.3, 0.0), 0.3, Vec3(0.8, 0.2, 0.2)));
scene->addSceneObject(new Sphere(Vec3(0.4, 0.1, 0.1), 0.1, Vec3(0.0, 1.0, 0.0)));
scene->addSceneObject(new Sphere(Vec3(-0.35, 0.25, -0.15), 0.25, Vec3(0.5, 0.5, 0.1)));
scene->addSceneObject(new Sphere(Vec3(-0.35, 0.15, 0.3), 0.15, Vec3(0.0, 0.0, 1.0)));
scene->addSceneObject(new Sphere(Vec3(0.15, 0.6, -1.0), 0.6, Vec3(0.2, 0.4, 0.5)));
scene->addLightSource(Vec3(1.0, 1.5, 1.0));
```



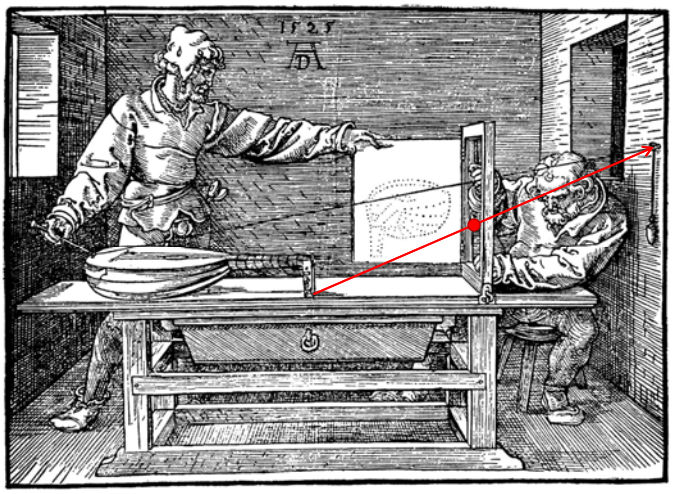
Beleuchtungseffekte

- Welche Beleuchtungseffekte sind sichtbar?



Ray Tracing Ursprünge

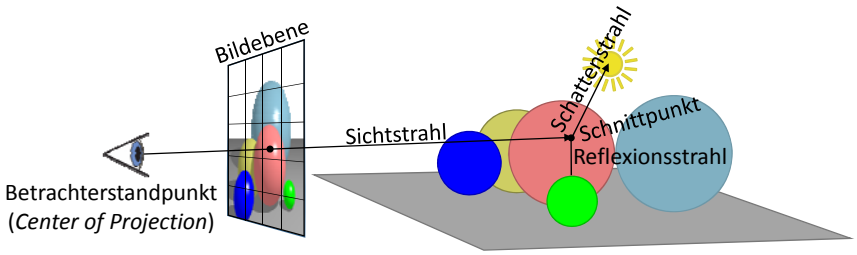
- Konzept zuerst in der Kunst aufgetaucht



Albrecht Dürer: Der Zeichner der Laute, 1525.

Übersicht

- Ray Tracing ist ein Bild-basiertes Rendering Verfahren, welches mit Strahlverfolgung arbeitet
  - Erzeuge einen Sichtstrahl durch jedes Pixel
  - Berechne Schnittpunkt mit Szenenobjekt
  - Berechne Farbe am Schnittpunkt



Kapitelstruktur

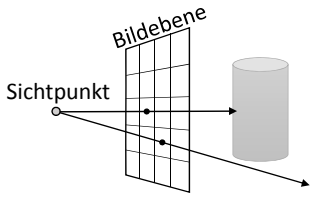
2.1 Ray Casting
2.2 Lokale Beleuchtung
2.3 Schatten
2.4 Reflexionen
2.5 Refraktion
2.6 Erweiterungen
2.7 Weiterführende Literatur

1.1 Ray Casting

Ausschließliche Berücksichtigung erster Schnittpunkte

Ray Casting Prinzip

- Konzept wurde 1968 von Arthur Appel vorgestellt
- Bild-basiertes Verfahren zur Strahlverfolgung
  - **Strahl Initialisierung -** Für jedes Pixel initialisiere ein Strahl ausgehend vom Sichtpunkt (Ray Generation)
  - **Schnittpunktberechnung** - Finde für den Strahl den der Kamera nächsten Schnittpunkt mit einem Objekt (Ray Intersection)
  - **Schattierung** - Bestimme Pixelfarbe aufgrund der Farbe des Objektes im Schnittpunkt (Shading)



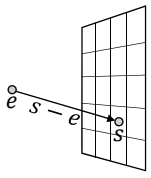

```
// for-Schleife über alle Pixel
for (each pixel) {
   ray = initializeViewingRay();
   point = findFirstIntersectionPoint(ray);
   pixel = computeShading(point);
}
```


Strahl Initialisierung 1/3

- Ein Strahl wird durch Ursprung (Origin) und Richtung definiert
- Mathematisch als 3D Linie parametrisiert über t:

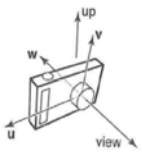
$$p(t) = e + t(s - e)$$

  - e: Ursprung des Strahls
  - $s - e$: Richtung des Strahls
- Eigenschaften der Darstellung
  - $p(0) = e$ und $p(1) = s$
  - $0 < t_1 < t_2 \Rightarrow p(t_1)$ näher an e als $p(t_2)$
  - $t < 0 \Rightarrow p(t)$ liegt *hinter* dem Ursprung

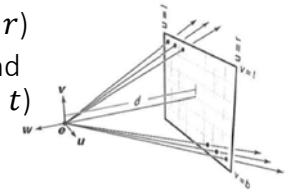


Strahl Initialisierung 2/3

- Wie können wir s finden?
- Annahme: Wir haben ein vereinfachtes Kameramodell
  - Positioniert in e
  - Orthonormale Basis $\{u, v, w\}$
    - Orthonormal: orthogonal + normalisiert
    - $\{u, v, w\}$ definiert rechtshändiges Koordinatensystem



- Spezifikation der Bildebene in Relation zu $\{u, v, w\}$
  - l und r definieren linken und rechten Rand relativ zu e in Richtung u (typisch: $l < 0 < r$)
  - b und t definieren unteren und oberen Rand relativ zu e in Richtung v (typisch: $\mathrm{b} < 0 < t$)
  - Distanz zwischen Bildebene und e ist d



Strahl Initialisierung 3/3

- Definition des Pixel Rasters mit $n_x \times n_y$ Pixeln
  - Größe des Pixel Rasters ist $(r - l) \times (t - b)$
  - Pixelabstand beträgt
    - X-Koordinate: $(r - l)/n_x$
    - Y-Koordinate: $(t - b)/n_y$
  - Um das Raster in $(r - l) \times (t - b)$ zu zentrieren wird ein 0,5 Pixel breiter Außen Rahmen angenommen
- Berechne Pixelposition (x, y) in Kamerakoordinaten

$$s_u = l + (r - l) \cdot (x + 0{,}5)/n_x$$
$$s_v = b + (t - b) \cdot (y + 0{,}5)/n_y$$
$$s_w = -n$$

- Berechnung der Strahlrichtung

$$d = s - e = s_u \cdot u + s_v \cdot v + s_w \cdot w$$

Schnittpunktberechnung

- Finden das kleinste positive t für das $p(t)$ ein Schnittpunkt
- Einfaches Vorgehen
  - Iteriere über alle Szenenobjekte und Berechne Schnittpunkte
  - Sortiere Schnittpunkte
  - Wähle Schnittpunkt mit kleinstem t
- Schnittpunktberechnung wird $n_x \cdot n_y \cdot n_o$ mal ausgeführt
- Optimierte Schnittpunktberechnung für Objektklassen notwendig
  - Kugel
  - Dreieck
  - …

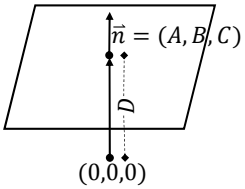
Iterative Schnittpunktberechnung (Ray Marching)

- Gehe vom Ausgangspunkt in ε Schritten und Prüfe alle Szenenobjekte auf Intersektion

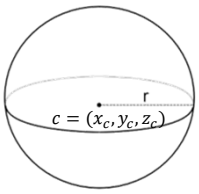
```
Intersection intersect(const Ray& ray) const {
    // ray marching
    Intersection result = Intersection(Vec3(0.0), Vec3(0.0), 0.0, Vec3(0.0));
    double t=0.0;
    double epsilon=0.01;
    for (int i=0; i<BIG_NUMBER; i++) {
      Vec3 rayPos = ray.origin + ray.direction*t;
      if (sceneObject.intersects(rayPos)) {
        result.pos = ray.origin + ray.direction*t;
        result.color = sceneObject.color;
        result.normal = sceneObject.normal;
        return result;
      }
      t+=epsilon;
    }
}
```

Schnittpunktberechnung - Implizite Fläche

- Finde Menge aller Punkte mit $f(x, y, z) = 0$
- Beispiele: Ebene, Kugel, Quadriken, …



$f(x, y, z) = \mathrm{Ax} + \mathrm{By} + \mathrm{Cz} + \mathrm{D}$



$f(x, y, z) = (x - x_c)^2 + (y - y_c)^2 + (z - z_c)^2 - r^2$

- Schnittpunktberechnung: finde $f\big(e + t(s - e)\big) = 0$

Schnittpunktberechnung – Kugel 1/3

- Idee: Finde Punkte die auf Strahl **und** Kugel liegen
  - Strahl-Punkte: $f(e + t \cdot d) = 0$
  - Kugel-Punkte: $f((x - x_c)^2 + (y - y_c)^2 + (z - z_c)^2 - R^2) = 0$
    wobei $c = (x_c, y_c, z_c)$ Zentrum und R Radius der Kugel sind
- Einsetzen ergibt quadratische Gleichung mit 0 oder 2 Lösungen (=Schnittpunkt-Parametern)

$$At^2 + Bt + C = 0$$

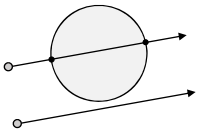
$$t_0 = \frac{-B + \sqrt{B^2 - 4AC}}{2A}$$

$$t_1 = \frac{-B - \sqrt{B^2 - 4AC}}{2A}$$

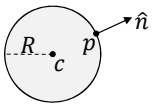
Schnittpunktberechnung – Kugel 2/3

- Prüfe ob Diskriminante größer Null

$$t = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}$$



  - Falls ja: 2 Schnittpunkte p_0 und p_1 existieren
    - Wähle kleineres t für nächsten Schnittpunkt
  - Falls nein: Es existiert kein Schnittpunkt
- Normalenberechnung
  - Normale gegeben durch: $\vec{n} = 2(p - c)$
  - Normalisierung durch Division durch R: $\vec{n} = (p - c)/R$

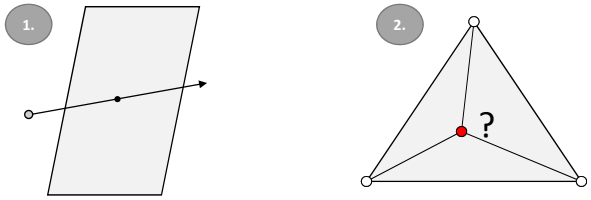


Schnittpunktberechnung - Kugel 3/3

```
Intersection intersect(const Ray& ray) const {
    // analytic solution
    Intersection result = Intersection(Vec3(0.0), Vec3(0.0), 0.0, Vec3(0.0));
    double A = ray.direction.x*ray.direction.x + ray.direction.y*ray.direction.y + ray.direction.z*ray.direction.z;
    double B = 2.0 * (ray.direction.x*(ray.origin.x-center.x) + ray.direction.y*(ray.origin.y-center.y) +
                      ray.direction.z*(ray.origin.z-center.z));
    double C = (ray.origin.x-center.x)*(ray.origin.x-center.x) + (ray.origin.y-center.y)*(ray.origin.y-center.y) +
               (ray.origin.z-center.z)*(ray.origin.z-center.z) - radius*radius;
    double discriminant = B*B - 4*A*C;
    if (discriminant > 0.0) {
      // two intersection points exist, find the closest one
      double t0 = (-B + sqrt(discriminant))/2*A;
      if (t0 > 0.0) {
        result.pos = ray.origin + ray.direction*t0;
      } else {
        double t1 = (-B - sqrt(discriminant))/2*A;
        result.pos = ray.origin + ray.direction*t1;
      }
      result.colorDiffuse = color;
      result.colorAmbient = color*0.5;
      result.colorSpecular = Vec3(1.0,1.0,1.0);
      result.distance = (ray.origin-result.pos).length();
      result.normal = (result.pos-center).normalize();
    }
    return result;
}
```

Schnittpunktberechnung - Dreieck

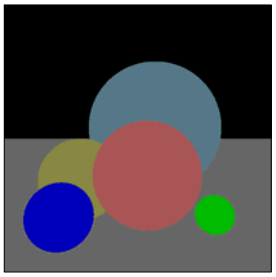
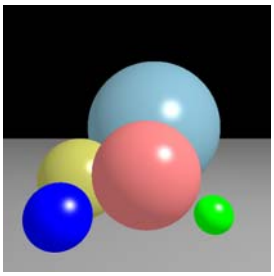
- Idee: Finde Schnittpunkt mit Ebene in der Dreieck liegt, und teste ob Schnittpunkt innerhalb Dreieck
  1. Schnittpunktberechnung mit Ebene über Nullstellensuche
  2. Baryzentrische Koordinaten



1.2 Lokale Beleuchtung

Hinzufügen von Lichteffekten auf Oberflächen

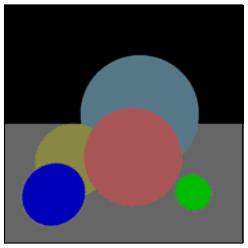
Lokale Beleuchtung

- Annahme: Funktion gegeben, die Farbe abhängig von Lichtquellen, Oberflächenmaterial und Oberflächenorientierung bestimmt
- Beobachtungen:
  - Lichtreflexion ist abhängig vom Eintrittswinkel
  - Lichtreflexion ist teilweise Betrachter-abhängig

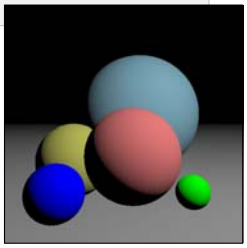
Ambientes Licht

```
Vec3 shadingAmbient(const Intersection& intersection) {
   Vec3 resultColor = Vec3();
   resultColor = intersection.colorAmbient;
   return resultColor;
}
```



Diffuse Reflexion

```
Vec3 shadingLambert(const Intersection& intersection) {
   Vec3 resultColor = Vec3();
   for (int i=0; i<myScene->getNumLightSources(); i++) {
      Vec3 lightVector = (myScene->getLightSource(i)-intersection.pos).normalize();
      double lambert = lightVector.dot(intersection.normal);
      if (lambert > 0.0)
         resultColor = resultColor + intersection.colorDiffuse*lambert;
   }
   return resultColor;
}
```



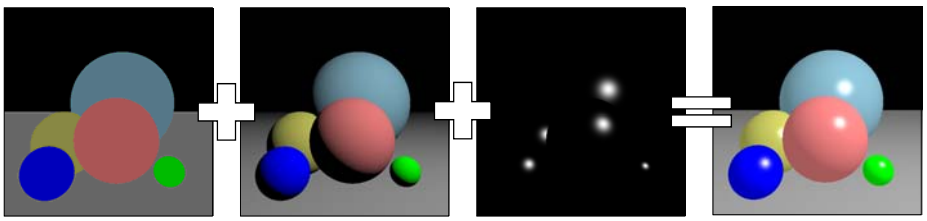
Spekulare Reflexion

- Halfway-Vektor nach Blinn-Phong

```
Vec3 shadingBlinnPhong(const Intersection& intersection, const Ray& viewRay) {
   Vec3 resultColor = Vec3();
   for (int i=0; i<myScene->getNumLightSources(); i++) {
      Vec3 viewVector = (viewRay.origin-intersection.pos).normalize();
      Vec3 lightVector = (myScene->getLightSource(i)-intersection.pos).normalize();
      Vec3 halfWayVector = (viewVector + lightVector).normalize();
      double specular = intersection.normal.dot(halfWayVector);
      if (specular > 0.0)
         resultColor = resultColor + intersection.colorSpecular*pow(specular, 120.0);
   }
   return resultColor;
}
```

Beleuchtungsmodell

- Lichtintensitäten sind additiv



Ambient Diffus Spekular

1.3 Schatten

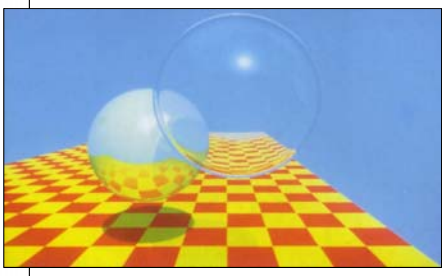
Teste Sichtbarkeit der Lichtquelle(n)

Whitted-Style Ray Tracing

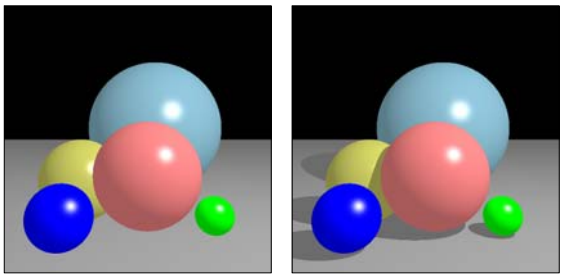
- Turner Whitted: *An Improved Illumination Model for Shaded Display*, SIGGRAPH 1979.

An Improved Illumination Model for Shaded Display

Turner Whitted
Bell Laboratories
Holmdel, New Jersey



Schattenberechnung



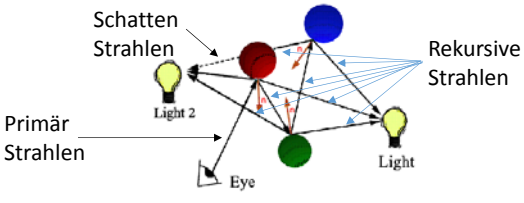
- Schatten für die räumliche Wahrnehmung bedeutend
- Beobachtung: Schatten entstehen an den Schnitt-punkten, von denen aus keine Lichtquelle sichtbar ist
- Test durch Generierung von Schattenstrahlen (Annahme: Punktlichtquellen)
- Diffuse & spekulare Reflexion nur außerhalb der Schatten

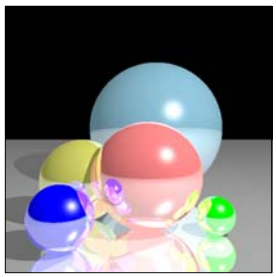
1.4 Reflexionen

Simulation von Spiegeln mit rekursiver Strahlverfolgung

Spiegelung

- Ausnutzung von Rekursion um Spiegelung zu simulieren
- Wunsch: Schicke für jeden Schnittpunkt Strahlen in alle Richtungen
- Realität: Beschränke Strahlen auf Richtungen mit höchstem zu erwartendem Einfluss
  - Hauptspiegelrichtung (Reflexion)

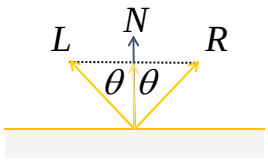




Rekursive Strahlverfolgung

- Berechnung der Hauptspiegelrichtung analog zur spekularen Beleuchtung

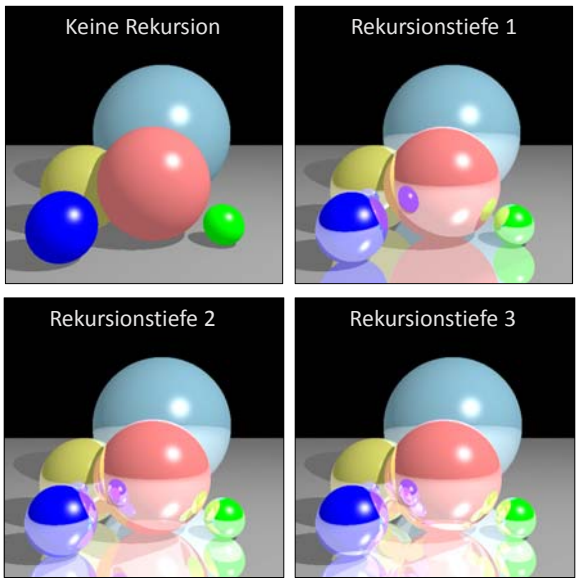
$$R = 2N\cos(\theta) - L$$



- Für welche Schnittpunkte wird Spiegelung berechnet?
  - Für alle nicht-schwarzen Punkte
  - Spiegelung wird als zusätzlicher Lichteingang betrachtet

Rekursionsabbruch

- Wann kann/muss Rekursion abgebrochen werden?

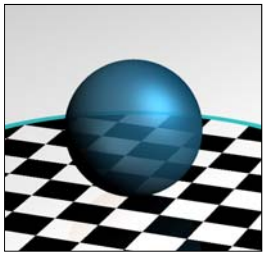


1.5 Brechung

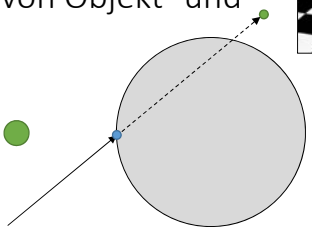
Lichtbrechung an semi-transparenten Szenenobjekten

Transparenz

- Erweiterung der Materialeigenschaften um Transparenz (Alpha Wert)
- Objektfarbe wird um α Komponente erweitert
  - 0,0: voll transparent (unsichtbar)
  - 0,5: semi-transparent
  - 1,0: voll opak (undurchsichtig)
- Ray Tracing berücksichtigt Transparenz durch Interpolation von Objekt- und Hintergrundfarbe

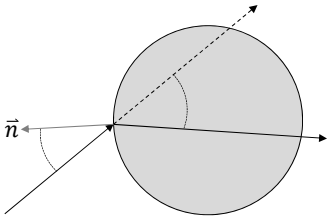


$$Farbe = \alpha \cdot \bigcirc + (1-\alpha) \cdot \bigcirc$$



Brechung

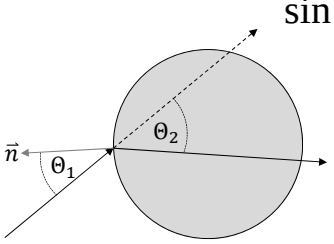
- Brechung entsteht wenn Licht auf ein semi-transparentes Material mit anderem Brechungsindex trifft
- Wird im Ray Tracing durch zusätzlichen Brechungsstrahl erzeugt
- Brechungsgesetz (=Snelliussches Gesetz) sagt den Austrittswinkel des Brechungsstrahls voraus

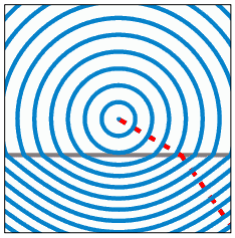


Brechungsgesetz

- *„Das Verhältnis der Sinusse der Einfallswinkel eines Lichtstrahls an der Grenzfläche zwischen zwei Materialien ist gleich dem umgekehrten Verhältnis der Brechungsindizes der Materialien ist gleich dem Verhältnis der Geschwindigkeit des Lichts in den Materialien.“*

$$\frac{\sin\theta_1}{\sin\theta_2} = \frac{n_2}{n_1} = \frac{v_1}{v_2}$$





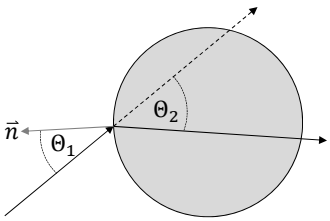
[Wikimedia]

Brechungs-Berechnung

- Brechungsindex: Verlangsamung der Lichtgeschwindigkeit innerhalb eines Mediums

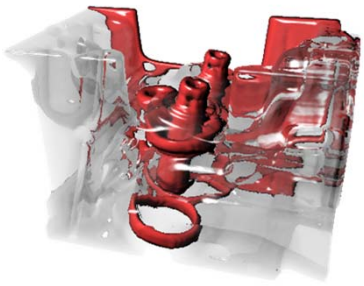
$$\theta_1 = \cos^{-1}(N \bullet D)$$

$$\frac{\sin\theta_1}{\sin\theta_2} = \frac{n_2}{n_1} \quad \Rightarrow \quad \theta_2 = \sin^{-1}\left(\frac{n_1}{n_2}\sin\theta_1\right)$$



Medium	Brechungsindex
Vakuum	1
Luft	1.0003
Diamant	2.4190
Eis	1.3100
Acryl Glas	1.4910

Brechung - Beispiele

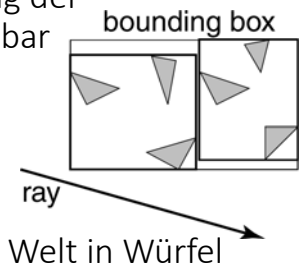


1.6 Erweiterungen

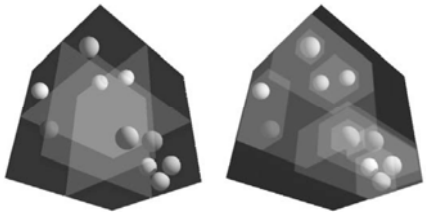
Bild- und Laufzeitverbesserungen

Datenstrukturen

- Für große Szenen ist eine Minimierung der Schnittpunktberechnungen unabdingbar

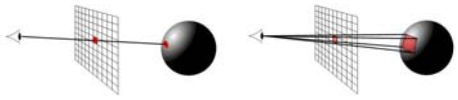


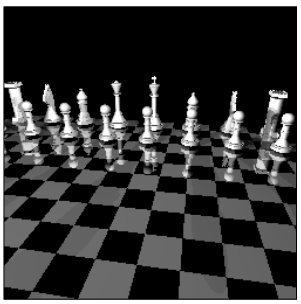
- Octrees: Hierarchische Zerlegung der Welt in Würfel
  - Kd-Bäume: Berücksichtigung des Szenen Inhalts



Supersampling 1/2

- Ein Strahl pro Pixel-Center führt zu *Aliasing* Artefakten
- Supersampling verwendet mehrere Strahlen pro Pixel, die gewichtet werden
  - Adaptives Sampling: Mehr Strahlen in heterogenen Regionen (Geometrie oder Beleuchtung)
  - Stochastisches Sampling: Generierung von mehreren Strahlen durch zufällige Pixel-Schnittpunkte
    - Ersetzt Aliasing durch Rauschen
    - Konvergiert gegen korrekte Lösung





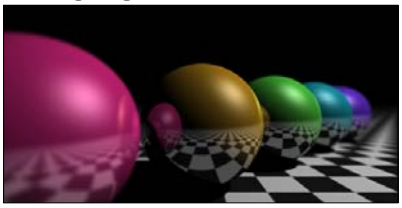
Supersampling 2/2

Ohne Supersampling

Mit Supersampling

Monte Carlo Ray Tracing

- Ergebnisbilder regulärer Ray Tracer haben häufig eine unnatürlich hohe Bildschärfe
- Monte Carlo Ray Tracing (=Stochastisches RT, Distributed RT) simuliert *Soft Effects* durch zusätzliche Strahlen
- Mögliche Effekte
  - Weiche Schatten (Flächenlichtquellen)
  - Tiefenunschärfe (Kameramodell)
  - Bewegungsunschärfe



Povray Ray Tracer

- Frei-verfügbarer Ray Tracer verfügbar unter povray.org
- Realisiert die meisten Ray Tracing Konzepte

[Quelle: hof.povray.org]

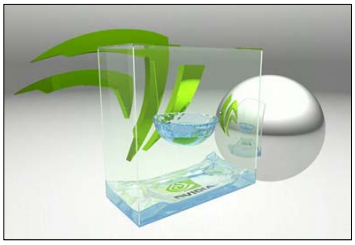
Echtzeit Ray Tracing 1/2

- Trotz Parallelisierbarkeit gilt Ray Tracing als langsam
  - Viele Gleitkomma Operationen
  - Komplexer Programmfluss
  - Komplexer Speicherzugriff
- Lösung 1: CPU Cluster
  - Skalierbar mit Multi-Core Prozessoren
  - Verteilung der Daten notwendig
  - Schwierig als Mainstream Lösung
  - Anwendungen
    - Weta Digital (Avatar)
    - Industrial Light and Magic

Echtzeit Ray Tracing mit OpenRT auf 48 CPU Cluster

Echtzeit Ray Tracing 2/2

- Lösung 2: Ausnutzung moderner GPUs
  - Ray Tracing kann in GLSL oder OpenCL implementiert werden
  - NVIDIA's Optix Ray Tracer basiert auf CUDA und unterstützt Erweiterungen um Ray Tracing Anwendungen zu entwickeln
  - Anwendungen
    - Weta Digital (Herr der Ringe, X-Men)



Echtzeit Ray Tracing mit NVIDIA's Optix GPU Ray Tracer

1.7 Weiterführende Literatur

Zugrundeliegende und ergänzende Quellen

Literatur

- P. Shirley, M. Ashikhmin, S. Marschner: Fundamentals of Computer Graphics (3. Auflage), AK Peters 2009.
  - Kapitel 4: Ray Tracing
- Turner Whitted: An Improved Illumination Model for Shaded Display, Communications of the ACM 1980.