Metaballs

- Abgeschaut von den Molekülen
- Idee: betrachte Kugel als Menge aller Punkte im Raum, die dasselbe "Potential" haben, wobei das Maximum des Potentialfeldes im Kugelmittelpunkt herrscht → Isofläche
- Potentialfeld wird beschrieben durch Potentialfunktion, z.B.

$$p(r) = \frac{1}{r^2}$$

wobei

$$r = r_1(\mathbf{x}) = \|\mathbf{x} - \mathbf{x}_1\|$$

- Die Kugelfläche ist damit

$$K = \{\mathbf{x} \mid p(\mathbf{x}) = t\}$$

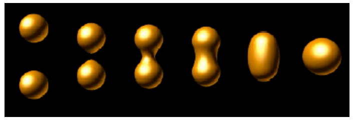
  - t heißt Schwellwert oder Isowert.

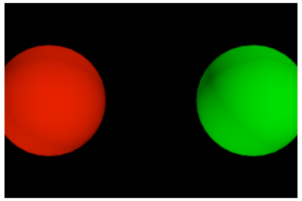
- Implizite Funktion setzt sich zusammen aus
  Distanzfunktion + Potentialfunktion
- Entsprechend gibt es viele Varianten und Namen: "metaballs", "soft objects", "blobs", "blobby modeling", "implicit modeling" ...
- Komplexere Objekte entstehen durch Überlagerung (*Blending*) der Potentialfelder mehrerer Punkte
  - Einfachstes Blending ist Addition der Felder:

$$P(\mathbf{x}) = \sum_{i=1}^{n} a_i \frac{1}{r_i^2(\mathbf{x})}$$

  - Alle Punkte zusammen heißen Skelett (*skeleton*), P ist das Gesamtpotential, a_i = "Feldstärke" bestimmen jew. Einfluß
  - Negative Feldstärken nehmen "Material" weg (z.B. für Löcher)

- Beispiel für 2 Skelett-Punkte:



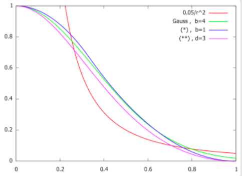


- Andere Potentialfunktionen:

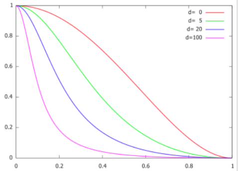
$$p_i(r) = e^{-br^2}$$

$$p(r) = \begin{cases} 1 - 3\frac{r^2}{b^2} & , r \le \frac{1}{3}b \\ \frac{3}{2}(1 - \frac{r}{b})^2 & , \frac{1}{3}b \le r \le b \\ 0 & , r > b \end{cases} \quad (*)$$

$$p(r) = \begin{cases} \frac{r^4 - 2r^2 + 1}{1 + dr^2} & , r \le 1 \\ 0 & , r > 1 \end{cases} \quad (**)$$

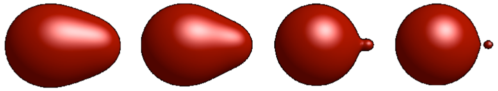


Vergleich



(**) mit verschiedenen Parametern

- Effekt der Variation des Parameters:

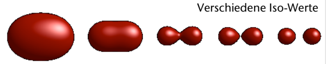


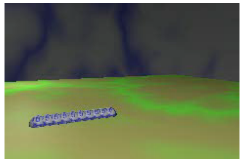
Potentialfkt (**), *d* für den linken Skelettpunkt fest, *d* = 10 ... 2000 für den rechten Punkt

Deformationen

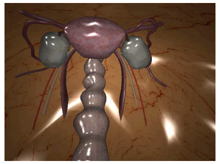
- Mit diesem Modell lassen sich Deformationen von "blob-artigen" Objekten sehr einfach modellieren:
  - Verschiebe Skelett-Punkte
  - Modifiziere Parameter *a*, *b*, ...
  - Modifiziere den Iso-Wert *t*

Verschiedene Iso-Werte

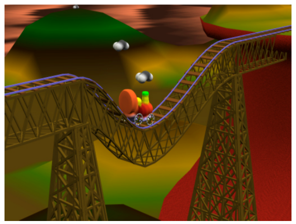




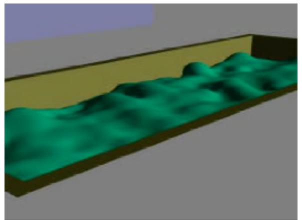
Brian Wyvill
http://pages.cpsc.ucalgary.ca/~blob/animations.html



Frédéric Triquet
http://www2.lifl.fr/~triquet/implicit/video/

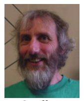


"The Great Train Rubbery" — Siggraph 1986



"Soft"

"The Wyvill Brothers"

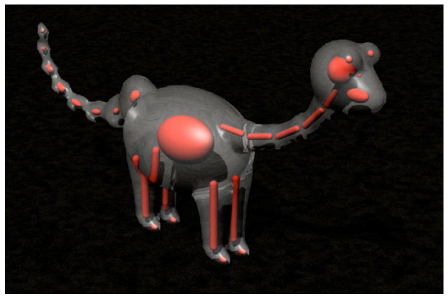


Geoff

Brian

Verallgemeinerungen / Varianten

- Punkte sind das einfachste Primitiv zur Konstruktion eines Skeletts; analog kann man Linien, Polygone, Ellipsoide, etc., verwenden:



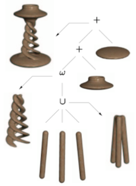
Beispiele weiterer Skelett-Primitve:

- Andere Blending-Funktionen:

$$P_{\cup}(\mathbf{x}) = \max\{p_1(\mathbf{x}), p_2(\mathbf{x})\}$$

$$P_{\cap}(\mathbf{x}) = \min\{p_1(\mathbf{x}), p_2(\mathbf{x})\}$$

- Ein Baum von Blending-Operationen — der BlobTree:



Bemerkungen zum *"implicit modeling"*

- Man kann nette Effekte recht einfach erzielen
- Als professionelles Tool in der Animationsindustrie oder im CAD hat es sich nicht durchgesetzt, weil einfach zu viel "Magie" im Spiel ist [sagt auch Geoff Wyvill]
- Brian Wyvill arbeitet immer noch an diesen Methoden [2004]

Normale an impliziten Flächen

- Normale in Punkt ***x*** auf impliziter Fläche *f*(***x***)

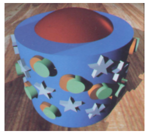
$$\mathbf{n}(\mathbf{x}) = \nabla f(\mathbf{x}) = \begin{pmatrix} \frac{\partial f}{\partial x}(\mathbf{x}) \\ \frac{\partial f}{\partial y}(\mathbf{x}) \\ \frac{\partial f}{\partial z}(\mathbf{x}) \end{pmatrix}$$

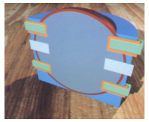
$$\approx \begin{pmatrix} f(x+\varepsilon, y, z) - f(\mathbf{x}) \\ f(x, y+\varepsilon, z) - f(\mathbf{x}) \\ f(x, y, z+\varepsilon) - f(\mathbf{x}) \end{pmatrix}$$

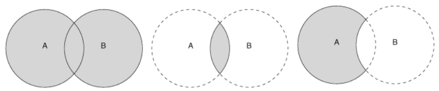
$$\approx \begin{pmatrix} f(x+\varepsilon, y, z) - f(x-\varepsilon, y, z) \\ f(x, y+\varepsilon, z) - f(x, y-\varepsilon, z) \\ f(x, y, z+\varepsilon) - f(x, y, z-\varepsilon) \end{pmatrix}$$

Weitere Objekt-Repräsentation: CSG

- Fügt sich genauso zwanglos ins Raytracing ein
- Zentrale Idee: konstruiere neue Objekte durch Mengen-Operationen auf einfachen Grund-Volumina (→ CSG = *constructive solid geometry*)
  - Mengen-Operationen: Schnittmenge, Vereinigung, Differenz
  - Grund-Primitive: alle Objekte, die sich leicht implizit beschreiben lassen
  - Rekursive Anwendung der Operationen → "Objekt-Arithmetik"



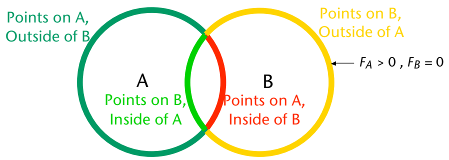




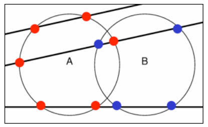
Vereinigung Schnittmenge Differenz

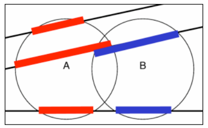
Implementierung

- Verwende implizite Form der Grund-Objekte

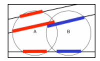


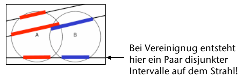
- Bestimme **alle** Schnittpunkte eines Strahls mit allen Grundobj.en
  - Falls alle Grundobj. konvex → 1 Intervall auf dem Strahl pro Grundobj.

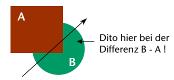




- Führe die Operation auf den Intervallen durch
- Rekursiv den CSG-Baum nach oben
- Falls an der Wurzel ein leeres Intervall entsteht → kein Schnitt
- Sonst: wähle Minimum aller Intervalle, die bis zur Wurzel übrigbleiben / entstanden sind
- Achtung:
  - Bei einer Operation können mehrere disjunkte Intervalle entstehen!





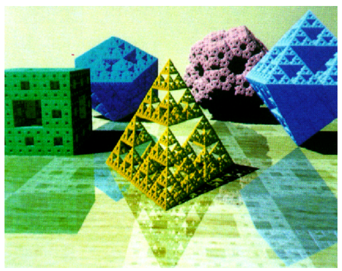


  - Achte auf numerische Stabilität (z.B.: lösche zu kleine Intervalle)

Fraktale

- Auch Fraktale kann man trivial ray-tracen
- Einfach Rekursion "on demand" bis zur gewünschten Tiefe

→ Prozedural beschriebene Objekte



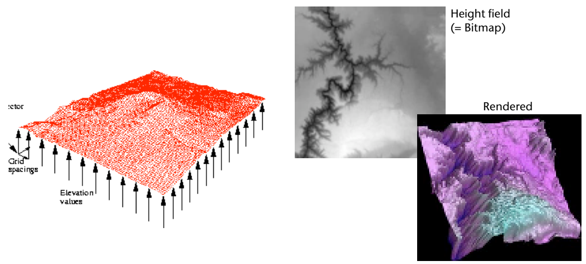
Height Fields

[Henning & Stephenson, 2004]

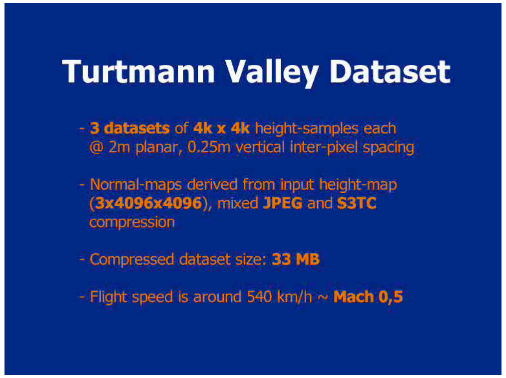
- Height Field = Alle Arten von Flächen, die sich als Funktion

$$z = f(x, y)$$

schreiben lassen
  - Z.B.: Terrain, Meßwerte über einer Ebene, 2D-Skalarfeld, ...



Beispiele für Terrain

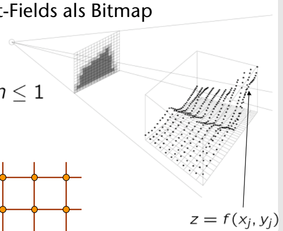


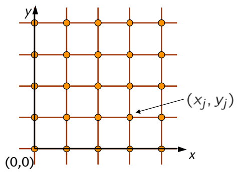
Bonn University

Valles Marineris, Mars - http://mars.jpl.nasa.gov

Situation

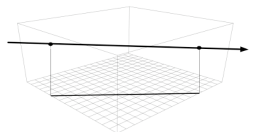
- Naïve Methode, ein Height-Field zu Raytracen:
  - konvertiere in $2n^2$ Dreiecke, teste Strahl dagegen
  - Probleme: langsam, benötigt viel Speicher
- Ziel: direktes Ray-Tracing des Height-Fields als Bitmap
- Gegeben:
  - Strahl:
    $y = mx + b$, oBdA $0 \le m \le 1$
  - Feld [0…n]x[0…n] als Bitmap
  - Höhenwerte liegen auf den Gitterknoten vor
- Terminologie:
  - Strahl = unendlich in eine Richtung
  - Linie = unendlich in beide Richtungen



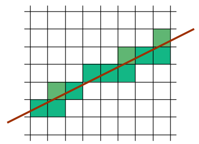


Algorithmus

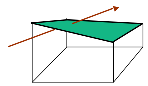
1. Dimensionsreduktion
   - Projiziere Strahl in xy-Ebene



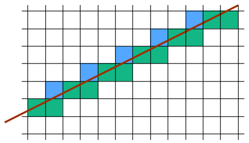
2. Alle Zellen der Reihe nach besuchen, die vom Strahl geschnitten werden (und nur diese)
   § Ähnlich zu Scan-Conversion, aber mit zusätzlichen Zellen



3. Strahl testen gegen das Flächenstück, das von den 4 Höhenwerten an den Ecken aufgespannt wird

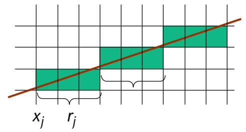


- Vereinfachungen zunächst:
  - Betrachte (unendliche) Linien mit $y = mx\ , \quad 0 \le m \le 1$
  - Betrachte nur die Folge der grünen Zellen = Zellen, die an ihrer linken Kante von der Linie geschnitten werden
- Terminologie:
  - Zelle wird identifiziert durch deren linken unteren Eckpunkt (x_j, y_j)
  - Span := Folge von Zellen mit gleicher y-Koord.
  - Länge des j-ten Spans = r_j



- Beobachtung: die diskrete Linie ist vollständig durch die Folge der Span-Längen definiert, denn

$$(x_{j+1}, y_{j+1}) = (x_j + r_j, y_j + 1)$$



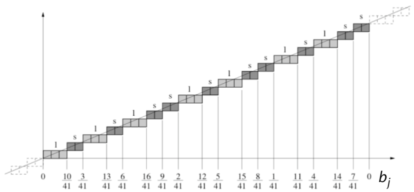
- Satz (o. Bew.):
  Alle Spans der diskretisierten Linie haben nur eine von genau zwei verschiedenen Längen, nämlich

$$\forall j:\ r_j = r \ \vee\ r_j = r + 1$$

- Klar ist:

$$\tfrac{1}{2} \le m \le 1 \ \Rightarrow\ r = 1$$

- Beispiel:



- Beobachtung: wenn wir ein seeehr langes Segment der Linie betrachten, dann gilt

$$\frac{\#\ \text{Spans}}{\#\ \text{Zellen}} = \frac{\Delta y}{\Delta x} \approx m$$

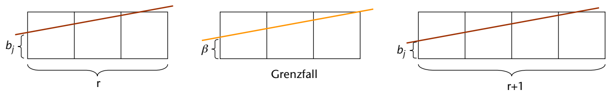
- Folge: aus der Steigung kann man die Span-Länge r (bzw. r+1) berechnen:

$$\begin{aligned}\frac{1}{m} &= \text{mittlere Span-Länge}\\ &= \text{Mittelwert von } r \text{ und } r+1 \ \Rightarrow\end{aligned}$$

$$r = \left\lfloor \frac{1}{m} \right\rfloor \ ,\ r + 1 = \left\lceil \frac{1}{m} \right\rceil$$

- Im Folgenden: Berechnung von r_j, m.a.W., Methode zur Entscheidung, ob man einen "langen Span" oder einen "kurzen Span" hat

- Wovon hängt es ab, ob man einen langen/kurzen Span hat?



- Also: falls $b_j \geq \beta$, dann kurzer Span, sonst langer Span.
- Bestimmung von β :

$$b_j = mx_j - y_j$$

$$b_{j+1} - b_j = mr_j - 1$$

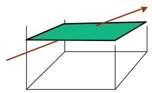
Im Grenzfall ist b_j+1=0 und $b_j = \beta$, also

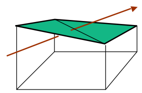
$$\beta = 1 - mr = 1 - m \left\lfloor \frac{1}{m} \right\rfloor$$

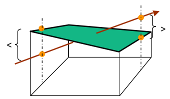
- Das nächste b_{j+1} ist also:
  falls kurzer Span → $b_{j+1} = b_j - \beta$
  falls langer Span → $b_{j+1} = b_j + m - \beta$
- Damit hat man einen iterativen, sehr effizienten Algo zur Aufzählung aller Zellen, die von einer Linie getroffen werden.
- Weiteres (lästiges) Detail:
  - Bei einem Strahl ist der erste Span i.A. gekürzt
  - Soll hier nicht weiter vertieft werden

Schnitttest Strahl — Flächenstück in der Zelle

- Naïve Methoden:
  - "Nearest-Neighbor":
    - Bestimme mittlere Höhe aus den 4 Höhenwerten an den Ecken
    - Schneide Strahl gegen horizontales Quadrat mit dieser mittleren Höhe
    - Sehr ungenau
  - "2 Dreiecke":
    - Konstruiere 2 Dreiecke aus den 4 Punkten über den Ecken
    - Knick innerhalb der Zelle, Aufteilung in Dreiecke nicht klar
- Besser: "bilineare Interpolation"
  - Betrachte Fläche als parabolisches Hyberboloid
  - Bestimme Höhe am Rand über/unter dem Strahl durch lineare Interpolation
  - Vergleiche Vorzeichen
  - Bestimmt ggf. Schnittpunkt & Normale





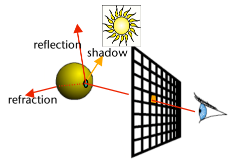


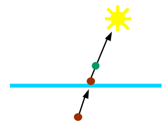
Speedup gegenüber einfachem DDA

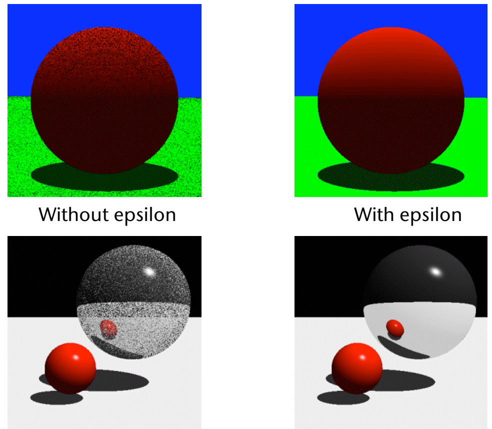
- O(n) bei DDA (z.B. Midpoint),
  O(n/r) mit der Span-basierten Methode hier,
  n = Anzahl Zellen auf dem Strahl, r = mittlere Span-Länge
- In Zahlen:
  - Ca. Faktor 2 schneller über alle Orientierungen des Strahls

The evil ε

- Was passiert, wenn der Ursprung des Strahles auf der Oberfläche eines Objektes sitzt?
- Floating-Point ist immer unexakt!
  - Folge: bei den folgenden Strahltests erscheint dieser Ursprung innerhalb des Objektes!
  - Folge: als nächsten Schnittpunkt erhalten wir denselben Punkt wieder!
- "Lösung": verschiebe Aufpunkt des Strahls immer zuerst um ein ε in Richtung des Strahls

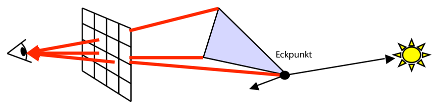




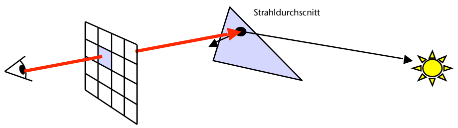


Scankonvertierung vs. Raytracing

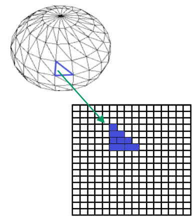
- Scan-Konvertierung: Auswerten eines Strahls, der durch jeden Eckpunkt eines Objektes gesendet wird



- Raytracing: Auswerten eines Strahls, der durch einen Bildschirmpixel gesendet wird

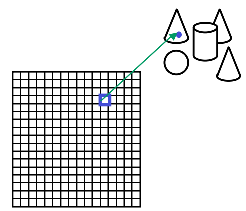


- Zum Umwandeln einer Szene mittels Scan-Konvertierung ...



... scan-konvertiere jedes Dreieck

- Zum Umwandeln einer Szene mittels Raytracing ...



... verfolge für jedes Pixel einen Strahl

Vor- und Nachteile

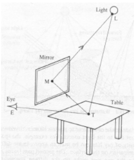
- Scan-Konvertierung:
  - schnell (da nur Eckpunkte)
  - wird unterstüzt von aktuellen Grafikkarten
  - geeignet für Echtzeitanwendungen
  - ad-hoc Lösung für Schatten, Transparenz
  - Keine Interreflexion
- Raytracing:
  - noch rel. langsam (Suche nach Schnittpunkten zwischen Strahlen und Objektprimitiven)
  - bisher von keiner kommerziellen Hardware unterstützt
  - Offline-Rendering-Verfahren
  - Allgemeine Lösung für Schatten, Transparenz und Interreflektion, Clipping und Culling

Bewertung des (einfachen) Raytracings

- Vorteile:
  - Eignet sich besonders für Szenen mit hohem spiegelndem und transparentem Flächenanteil
  - Kann beliebige Objektrepräsentationen verarbeiten (z.B. CSG, Rauch, ...)
    - Einzige Anforderung: man muß Schnitt zwischen Strahl und Objekt und die Normale in diesem Schnittpunkt berechnen können
  - Berechnung von Schatten, Reflexionen und Transparenzen sind ein inhärenter Teil des Raytracing-Algorithmus
  - Keine explizite perspektivische Transformation oder Clipping nötig

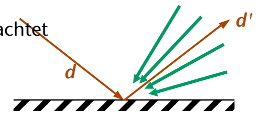
Nachteile

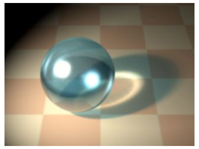
- Sehr viele Strahlen
  - Naives Ray-Casting: $O(p \cdot n \cdot l)$, p = # Pixel, n = # Polygone, l = # Lichtquellen
  - Anzahl Strahlen wächst exponentiell mit Rekursionstiefe!
- Keine indirekte Beleuchtung (Spiegel, *"color bleeding"* = diffuse indir. Bel.)
- Keine weichen Halbschatten
- Shading muß bei jeder Änderung der Kamera neu berechnet werden, obwohl diese nur von den Lichtquellen und den Objekten abhängen
- Für alle diese Nachteile wurden natürliche verschiedene Abhilfen vorgeschlagen

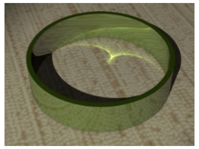


Beispiel für das Problem der indirekten Beleuchtung: Kaustiken

- Konzentration von Licht
- Lichtstrahlen treffen sich in einem Punkt
- Raytracing wird uneffektiv
- Nur 1 reflektierter Strahl wird betrachtet

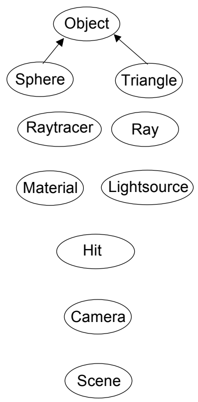






Software-Architektur eines einfachen Raytracers

- Szene: einfache Liste von Kugeln, Dreiecken, etc.
- Schnittpunkt Strahl—Objekt
- Diffuse und spekulare Reflexion (Phong-Modell)
- Sekundärstrahlen
- Mögliche Erweiterungen
  - Lichtbrechung, Transparenzen
  - Besseres Oberflächenmodell (Fresnel)
  - Andere Objekte (Kegel, Zylinder, Polygon, ...)
  - Szene einladen



Typische Raytracer-Klassen

- Lightsource (hier nur Einfache Punktlichtquelle)

```
Vector m_location;      // Position
Vector m_color;         // Farbe
```

- Material

```
Vector m_color;         // Farbe der Oberfläche
float m_diffuse;        // Diffuser / Spekularer
float m_specular;       // Reflexionskoeff. [0..1]
float m_phong;          // Phong-Exponent
```

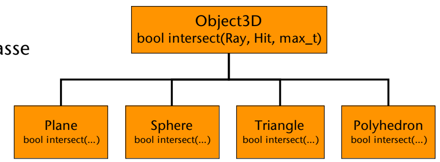
- Ray

```
Vector m_origin;        // Aufpunkt des Strahls
Vector m_direction;     // Strahlrichtung
```


- Hit: Speichert Informationen über den Schnittpunkt

```
Ray m_ray;              // Strahl
float m_t;              // Geradenparameter t
Object* m_object;       // Geschnittenes Objekt
Vector m_location;      // Schnittpunkt
Vector m_normal;        // Normale am Schnittpunkt
```


- Object: Abstrakte Basisklasse für alle Geometrie-objekte




```
// Schnittpunkt von Strahl mit Objekt
virtual bool closestIntersection( Intersection * hit ) = 0;
virtual bool anyIntersection( const Ray & ray, float max_t,
                              Intersection * hit ) = 0;
// Normale am Schnittpunkt
virtual void calcNormal( Intersection * hit ) = 0;
// Material des Objekts
int getMaterialIndex() const;
```

- Camera:
  - Alle Eigenschaften der Kamera, z.B. from, at, up, angle
  - Generiert Primärstrahlen durch alle Pixel
- Scene:
  - Speichert alle Daten der Szene
    - Liste aller Objekte
    - Liste aller Materialien
    - Liste aller Lichtquellen
    - Kamera

Das OBJ-File-Format

vertices

```
v -1 -1 -1
v  1 -1 -1
v -1  1 -1
v  1  1 -1
v -1 -1  1
v  1 -1  1
v -1  1  1
v  1  1  1
```

triangles

```
f 1 3 4
f 1 4 2
f 5 6 8
f 5 8 7
f 1 2 6
f 1 6 5
f 3 7 8
f 3 8 4
f 1 5 7
f 1 7 3
f 2 4 8
f 2 8 6
```