

# A Simple and Practical Method for Interactive Ray Tracing of Dynamic Scenes



Ingo Wald

Carsten Benthin

Philipp Slusallek

Marco Lang

## Bisher



- Ray Tracing erstmals von Appel benutzt
- dutzende Algorithmen vor allem um die Strahl/Objekt Schnittpunkttests zu reduzieren
- viele verschiedene Beschleunigungs-/Indexstrukturen
  - reguläre und hierarchische Gitter
  - Bounding Volume Hierarchien
  - Octrees
  - BSP-Bäume (*Binary Space Partitioning*)
  - kd-Bäume
- Parallelisierung
- optimierte Schnitttests

➤ Jedoch alles auf statische Umgebungen beschränkt !

Für die Interaktivität fehlt die Unterstützung von dynamische Szenen !

## Interaktives Ray Tracing

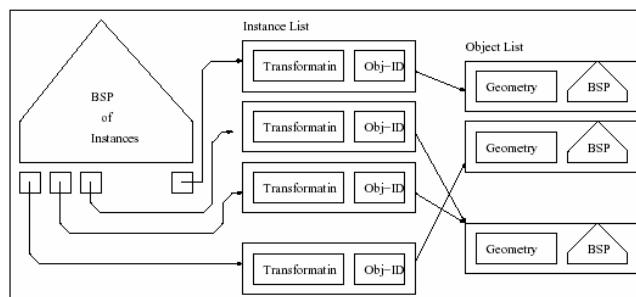


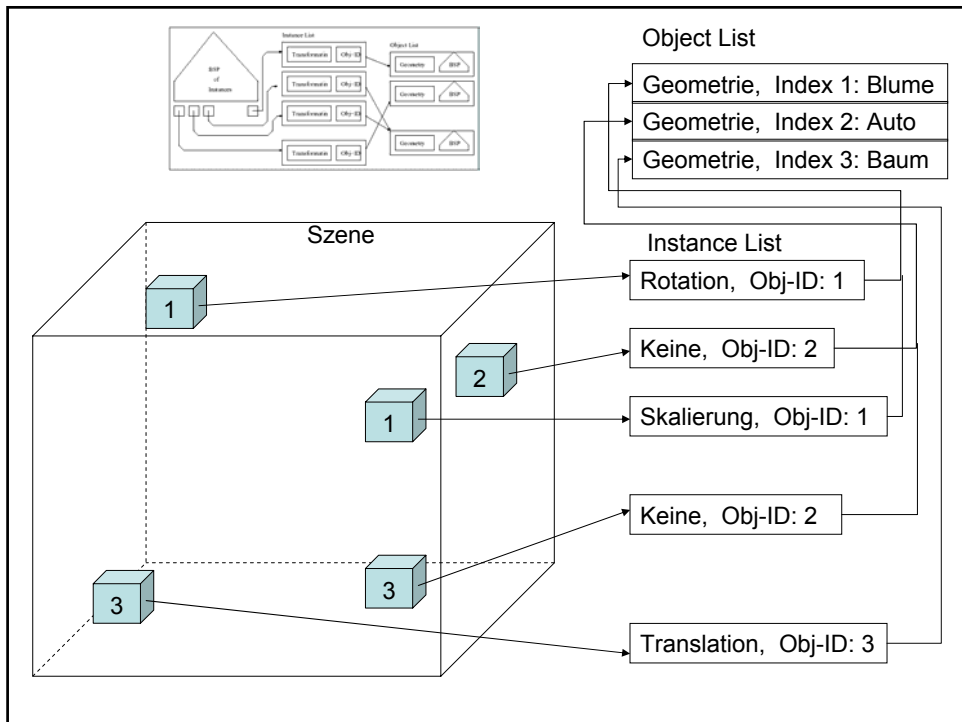
- Relativ neues Forschungsgebiet
- Bisher: Konzentration auf das Beschleunigen des Prozesses zum Erstellen eines einzelnen Bildes
  - Durch umfangreiche Vorbereitungen und
  - Komplexe Datenstrukturen (Beschleunigungsstrukturen)
- Bisherige beschleunigte Datenstrukturen werden zum Flaschenhals hinsichtlich ihres linearen Verhaltens
- Für dynamische Szenen müssen sie für jeden Frame neu erstellt werden

## Ansatz



- motiviert durch Szenengraph
- eine Zwei-Ebenen Hierarchie
- Eine Beschleunigungsstruktur (BSP Baum) auf oberster Ebene enthält Referenzen auf Instanzen (Transformation + Objekt-ID)
- Jedes Objekt besitzt eigene lokale Beschleunigungsstruktur und die Geometriedaten

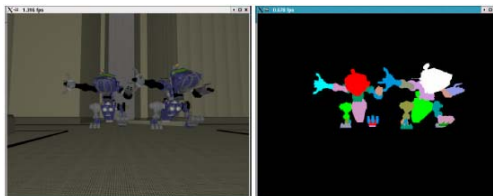




## Ansatz



- Szene in unabhängige Objekte unterteilen
- 3 Klassen :
  - Statische Objekte
  - Objekte mit affinen Transformationen (hierarchische Bewegung):  
alle Dreiecke, die der selben Transformation unterzogen werden, zusammenfassen
  - Objekte mit unstrukturierten Bewegungen:  
alle Dreiecke werden entsprechend ähnlich unstrukturierten Bewegungen gruppiert



## OpenRT



- Primitive werden in Objekten gruppiert, ähnlich display lists in OpenGL
- Dreiecke innerhalb solcher Objekte werden durch Transformation der display list transformiert
- Anstatt unstrukturierte Bewegungen im immediate mode zu rendern, werden spezielle Objekte benutzt, die neu definiert werden
- Hauptunterschied zwischen OpenRT und OpenGL: Objekte erlauben keine Seiteneffekte

## Traversieren durch oberste Ebenenstruktur



- Fallunterscheidung bei Treffen einer Bounding-Box
- statische Objekte: Strahl fährt in Bounding Box unverändert weiter  
→ lokale BSP Baum muss nur einmal erstellt werden
- affine Transformation: Anwenden der gespeicherten Transformation auf den Strahl bevor Traversieren weiter geht  
→ lokale Beschleunigungsstruktur bleibt erhalten  
→ oberste Ebenenstruktur entsprechend neuer Position anpassen
- Unstrukturierte Bewegung: lokale Beschleunigungsstruktur und die der obersten Ebene müssen für jedes Bild neu erstellt werden

## Implementation in OpenRT



- Strahl wird durch die Datenstruktur traversiert
- Innerhalb jedes Objektes statisches Ray Tracing (wie vorher)
- Für statische Objekte / hierarchische Bewegungen BSP Baum nur einmal bei Objektdefinition erstellen
- Speziell optimierte BSP Bäume für unstrukturierte Bewegungen:
  - Weniger teure Heuristiken
  - Typische Unterteilungstiefe: 2-3 Dreiecke pro Blatt → tiefere Bäume
  - Modifizierte Unterteilung: je tiefer desto mehr Dreiecke sind erlaubt
  - Konstanter Faktor für jede Tiefe
  - Flachere Bäume und größere Blattzellen
  - Schnellere Erstellung des Baumes, aber langsames Traversieren

## Beispiel: Strahlenverfolgung



1. Strahl erst an Szenenbegrenzungsbox clippen
  2. Iterativ durch obersten Ebenenbaum traversieren
  3. Wenn Strahl auf eine Blattzelle trifft
  4. Strahl sequentiell mit allen Instanzen in dieser Zelle schneiden
  5. Für jede Instanz Strahl erst in das lokale Koordinatensystem des Objektes transformieren
  6. Strahl mit dem Objekt unter Verwendung des original Algorithmus für statische Szenen schneiden
  7. Stopp sobald ein gültiger Schnittpunkt
- Mit „Mail-Box“ Verfahren wird mehrfaches schneiden des selben Objektes vermieden

## Konstruktion des BSP auf oberster Ebene



- Unterschied zwischen der Konstruktion des BSP für Objekte und für die oberste Ebene
- Objekt BSPs benötigen teure Dreieck-in-Zelle Berechnungen, sorgfältiges auswählen der SP und Unterstützung von degenerierten Fällen
- BSP auf oberster Ebene enthält nur Instanzen die durch eine Achsen-parallele Bounding-Box (AABB) des transformierten Objektes repräsentiert werden
- Das setzen der SP wird einfacher und es treten keine degenerierten Fälle auf, da nur AABB verwendet werden

## Konstruktion des BSP auf oberster Ebene

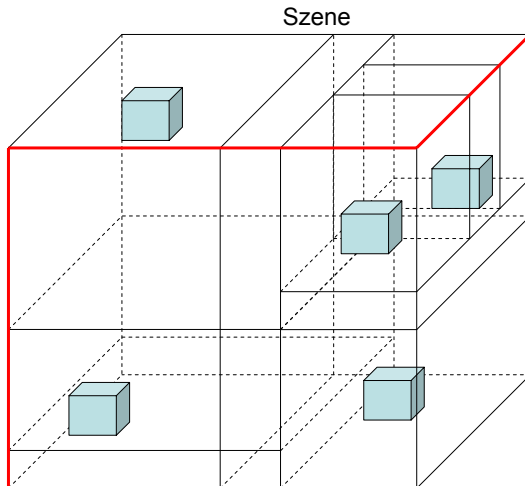


- Teilen einer Szene
  1. Teilen in der Dimension der maximalen Ausdehnung
  2. Platzieren der BSP-Ebene macht nur an den Grenzen von Objekten der aktuellen Zelle Sinn
  3. Optimale Position der SP zwischen dem geometrischen Zentrum der Zelle und dem Objekt-Median

**Algorithm 1** Algorithm for building the top-level BSP tree.

```
BuildTree(instances, cell)
  for d = x,y,z in order of maximum extent
    P = {i.min_d, i.max_d | i ∈ instances}
    c = center of cell
    if (more instances on left side of c than on right)
      p = max({p ∈ P | p < c})
    else
      p = min({p ∈ P | p ≥ c})
    if (p inside cell)
      {leftvox, leftinst, rightvox, rightinst}
      = SplitCell(instances, cell, p)
      BuildTree(leftinst, leftvox);
      BuildTree(rightinst, rightvox);
    return;
  # no valid splitting plane found
  cell = Leaf(instances)
```

## BSP Baum - Konstruktion



1. Finde Aufteilungsebene  
(Dimension der max. Ausdehnung, AABB)
2. Teile Objekten mit Ebene  
(zuerst Mitte, dann nächstgelegene Objektgrenze der Seite mit mehr Objekten)
3. Rekursive Anwendung  
(bis keine gültige Aufteilungsebene → Blattzelle)

## Testszenen

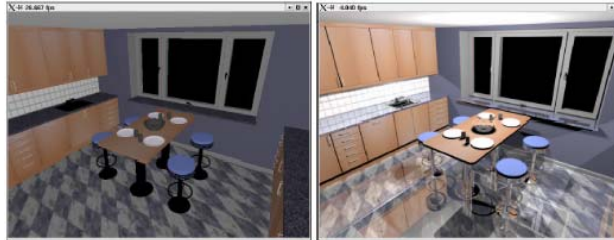


- BART Benchmark scenes
- Cluster von dual AMD AthlonMP 1800+
- FastEthernet Netzwerk
- verbunden mit Gigabit Ethernet Verbindung mit dual AthlonMP 1700+ Server
- Anwendung läuft auf Server und weiß nichts vom verteilten Rendern
- verwaltet die Geometrie in einem Szenen Graphen
- kontrolliert das Rendern durch Aufrufe der OpenRT API
- alle Beispiel sind in 640 x 840 gerendert

## BART Testszene: Küche



- Hierarchische Animation von 110.000 Dreiecken auf 5 Objekten
- Hauptkosten: viele verschiedene Strahlen, da 6 Punktlichtquellen und hohe Reflektivität
- Mit Reflektionstiefe 3 → 3.867.661 Strahlen pro Bild  
→ 526.000 verfolgte Strahlen pro Sekunde pro CPU bei 32 CPUs  
→ Framerate von 4.3 fps

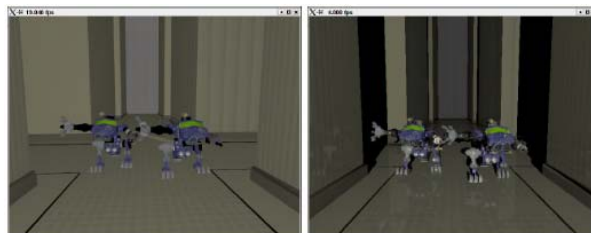


CPU	2	4	8	16	32
OpenGL-like	1.7	3.4	6.8	13.6	>26
Ray Tracing	0.25	0.5	1.05	2	4

## BART Testszene: Roboter



- 16 Roboter bewegen sich durch komplexe Stadt mit hierarchischer Animation ihrer Körperteile mit 161 verschiedenen Objekten
- Keine Unstrukturierte Bewegung
- Für Objekte muss BSP nur einmal erstellt werden
- Für oberste Ebene BSP für jedes Frame neu erstellen
- Hauptkosten: große Anzahl von Reflexions- und Schattenstrahlen

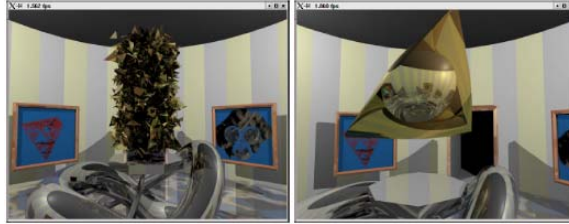


CPU	2	4	8	16	32
OpenGL-like	1.25	2.49	5.1	10	20
Ray Tracing	0.24	0.48	1.01	2.01	4

## BART Testszene: Museum



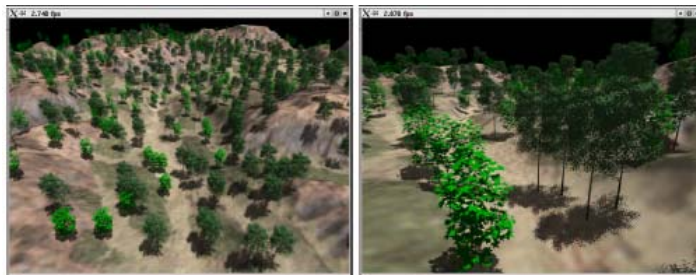
- Testen unstrukturierter Bewegungen
- Keine hierarchischen Bewegungen
- In der Mitte mehrere Dreiecke auf vordefinierten Animationspfaden
- 2 Punktlichtquellen
- Hauptkosten: Schatten und Reflexionen
- Ergebnisse vom Anfang der Animation → schlimmster Fall, Dreiecke willkürlich im Raum verteilt
- 4,8 fps für 1024 Dreiecke auf 8 CPUs
- 4,2 fps für 4096 Dreiecke auf 8 CPUs
- 19 fps mit OpenGL auf 8 CPUs



## BART Testszene: Terrain



- Testen der Skalierbarkeit mit größerer Anzahl von Instanzen und Dreiecken
  - 661 Instanzen von 2 verschiedenen Bäumen = 10 M Dreiecke
  - 1 Punktlichtquelle
  - Interaktives Bewegen der Bäume
- Kosten beschränken die Skalierbarkeit



## Ergebnisse



- Transformationskosten
  - Kernidee: vermeiden die komplette Datenstruktur immer wieder neu zu erstellen
  - stattdessen: transformieren der Strahlen in das Koordinatensystem des Objekts durch Matrix-Vektor-Multiplikationen
  - halbe Million Strahlen pro Sekunde → 100000 M-V-Multiplikationen
  - M-V-M kostet nur 23 Zyklen wenig im Vergleich zu Traversierung eines Strahls, von mehreren hundert bis tausend Zyklen
  - weitere Verbesserung durch Verwendung von SSE

## Ergebnisse



- unstrukturierte Bewegungen
  - Museums Szene → teuer zu ray tracen
  - noch nicht passend für Szenen mit starken unstrukturierten Bewegungen
- Überschneidung von Bounding Volumes
  - passiert in allen Testszenen
  - ist keine großes Performanz-Problem

## Ergebnisse



- Überabschätzung von Objektgrenzen
  - Abschätzen der Grenzen anhand des Originalobjekts
  - Problem ist auf Transformation und Clippen des Strahls beschränkt
  - nach Transformation in Objektkoordinatensystem, clippen an den richtigen Grenzen
- Teapot-in-a-Stadium Problem
  - Automatisches Einstellen auf variierende Dichte der Objekte

## Ergebnisse



- Skalierungsfähigkeit mit der Anzahl der Instanzen
  - wenige große Objekte anstelle von vielen kleinen Objekten
  - statische Dreiecke in einem einzigen Objekt speichern
  - effizientes Rendern durch Instanziierung
- Skalierungsfähigkeit in verteilten Umgebungen
  - gut Ergebnisse für mehrer Clients, außer wenn viele Informationen auf allen Clients upgedatet werden müssen

## BART Testszenen: Vergleich

OpenGL-like	1	2	4	8	16
Robots	1.25	2.49	5.1	10	20
Kitchen	1.7	3.4	6.8	13.6	26(-)
Terrain	0.68	1.34	2.55	4.76	8.33
Museum/1k	2.7	5.4	10.2	19.5	26(-)
Museum/4k	2.5	4.5	7.5	4.5	2.5
Museum/16k	1.6	2.4	1.7	1	0.5

Ray Tracing	1	2	4	8	16
Robots	0.24	0.48	1.01	2.01	4
Terrain	0.3	0.6	1.19	2.29	4.26
Kitchen	0.25	0.5	1.05	2	4
Museum/1k	0.6	1.2	2.4	4.8	9.3
Museum/4k	0.55	1.1	2.2	4.2	2.5
Museum/16k	0.45	0.9	1.65	0.98	0.53

## BART Testszenen: Video

