

Interactive Distributed Ray Tracing of Highly Complex Models



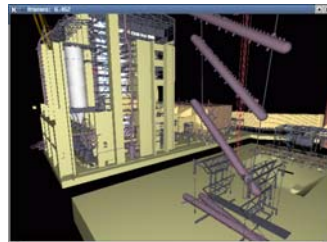
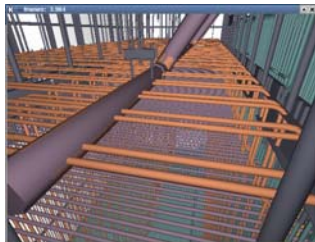
Ingo Wald, Philipp Slusallek, Carsten Benthin
<http://graphics.cs.uni-sb.de>

1



Einführung

- Komplexes Rendering vs. heutige 3D Graphik Hardware
- Schiffe, Gebäude, Fabrikanlagen
- incl. smooth-shading und Reflexionen



2



Interaktives Ray Tracing auf Standard-PC's

- hohe Rendering Geschwindigkeiten durch neue Raytracing-Algorithmen und Optimierungen auf niedrigem Level möglich
- z.B. optimierte Datenstrukturen für den Cache, SIMD-Erweiterungen nutzen
- ausschließlich Rendern von Dreiecken
- Schattenberechnung durch ladbare Module
- Vertauschen der Reihenfolge bei der Berechnung der Strahlen
- Raytracing in Paketen zu mehr als 4 Strahlen

3



Vergleichbare Arbeiten

- UNC „Framework for Realtime Walkthrough of Massive Models“
- nutzt high-end Graphik-Hardware und einen shared-memory Multiprozessor-Rechner
- erreicht konsistente Frameraten von 5-15 Bildern pro Sekunde
- kombiniert verschiedene Beschleunigungstechniken:
- ersetzt entfernte Geometrie durch texturierte Tiefenmeshes:
- -96% weniger gerenderte Polygone
- nutzt View Frustum und LOD: -50% Ersparnis
- nutzt occlusion culling : -10 %
- braucht lange Zeit für das Preprocessing: 17 h – 3 Wochen

4



Vergleichbare Arbeiten

- Parker et al.
 - full-featured Raytracer auf einem shared-memory Supercomputer
 - optimiert für Cache Performance und parallele Ausführungen
 - komplexe Szenen mit einigen hunderttausend Primitiven interaktiv darstellbar
- Muuss
 - CSG-Objekte als Primitive
 - High-end Supercomputer mit bis zu 96 CPU's
 - teurer shared-memory Rechner

5



Vergleichbare Arbeiten

- Pharr et al.
 - verwendet Kohärenz zwischen den Strahlen
 - incl. globaler Beleuchtungseffekte durch Pathtracing
 - bis zu 50 Mio Dreiecke
 - kein Real-Time Rendering
 - Unterteilung der Szene in Voxel

6



Interactive distributed Raytracing

- Einsatz mehrerer Prozessoren
- Ziel: ein Cluster von normalen PC's
- Bandbreite des Netzwerkes schränkt Performance ein
- ein Master - Rechner für die Darstellung
- nur eine Szenendatenbank

7



Anforderungen

- Unterteilung der Szene
- Scene Cache Management
- vermeiden von Wartezeit
- Lastverteilung

8



Distributed Data Management

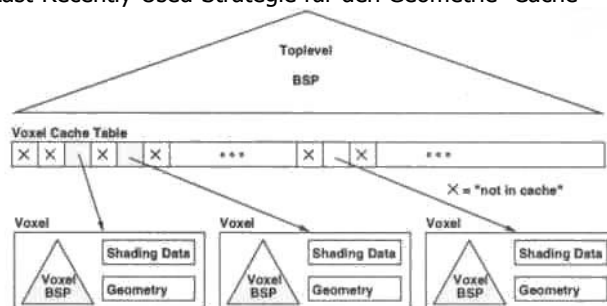
- Probleme bei hoch komplexen Szenen:
 - Beschränkte Dateigröße
 - Limitierter Adressraum
 - Verteilung der Modelldaten im Netzwerk
 - Unterbrechen der Bearbeitung des Strahls, während fehlende Daten gelesen werden
- Lösung:
 - asynchrones Laden der fehlenden Daten
 - Client arbeitet an einem anderen Strahl weiter
 - geht nicht mit dem automatischen loading on demand des Betriebssystems, da zu viele Threads nötig

9



Explizites Datenmanagement

- unterteilen des Modells in kleine Voxel
- jedes Voxel besitzt seinen eigenen BSP-Tree
- alle Voxel sind in einem high-level BSP tree organisiert
- Last-Recently-Used-Strategie für den Geometrie- Cache



10



Komprimierte Daten

- Dateigröße der Voxel vs. Overhead durch Verdopplung der Dreiecksdaten
- durchschnittliche Dateigröße: 250KB
- weitere Komprimierung der Daten, ca. 3:1 ->ca. 75 KB
- höhere Geschwindigkeit
- Kosten für Dekompression sind unbedeutend gegenüber der Übertragungszeit

11



Shared Voxel Cache

- dual Prozessor PC's
- 2 Raytracing Thread's laufen parallel
- Halbieren der Bandbreite des Netzwerkes erhöht Preis-/Leistungsverhältnis
- geladene Daten werden beiden threads zur Verfügung gestellt
- beide Threads teilen sich denselben Voxel-Cache
- ein dritter Cache Management Thread zusammen mit einem Thread, der die Voxel holt, bündelt die Cache Funktionalität
- Overhead so klein wie möglich

12



Preprocessing

- Algorithmus, bei dem 2 Parameter gesetzt werden:
 - Anzahl der Dreiecke in den Voxeln
 - Max. Tiefe des BSP-Baums
- manuell, da gute Werte noch nicht automatisch generierbar sind
- gesamte Datenmenge abhängig von Voxelgröße und Tiefe des BSP-Baums
- Kosten: $O(n \log n)$, abhängig von der Modellgröße
- serielle Implementation
- resultierende Dateien auf Modell Server
- Beschleunigungen möglich

13



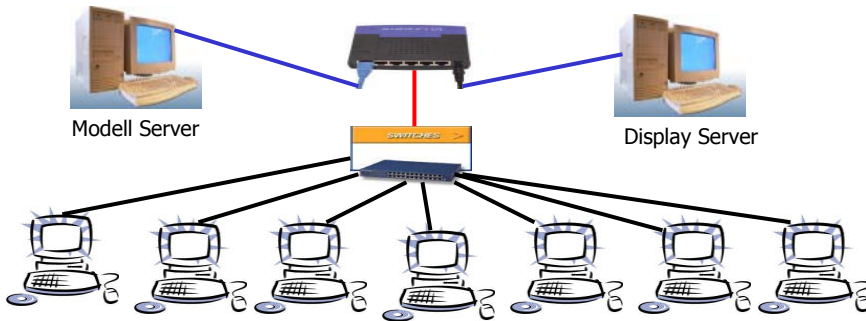
Lastverteilung

- Bild in Ausschnitte von 32x32 Pixeln zerlegt
- Display Server verteilt die Ausschnitte on demand an die Clients
- Manager gibt einem Client den Ausschnitt, den er vorher gerendert hat, um den Cache effizient zu nutzen
- Render Cache Algorithmus:
 - für jeden Strahl: merke 3-D-Schnittpunkte und ihre Rendering-Kosten, sowie den Client, der diese berechnet hat
 - für jeden Ausschnitt werden die Kosten von Schnittpunkten und Rendering gemittelt -> client affinity value
- teure Ausschnitte werden zuerst verteilt
- jeder Client hat eine weitere Aufgabe gepuffert
- sind alle Ausschnitte zugeordnet, verteilt der Server die des nächsten Bildes

14

Implementation

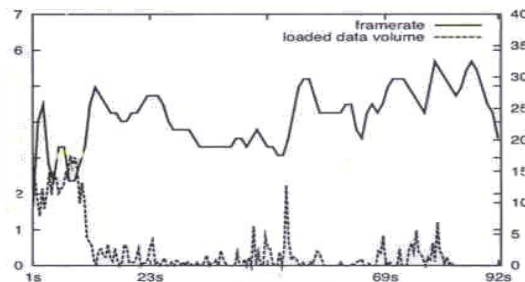
- 2 Server per Gigabit Ethernet mit einem Gigabit Ethernet Switch verbunden, per Gigabit Uplink mit Fast Ethernet Switch verbunden



15

Implementation

- Kraftwerksmodell: 12,5 Mio Dreiecke
- incl. Smooth Shading und Reflexion
- ohne Optimierung des Preprocessings: 2,5h incl. Konvertierung der Originaldaten
- mit Optimierung des Preprocessings: <0,5h

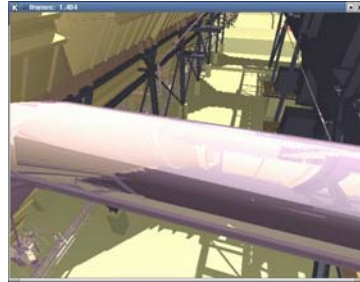
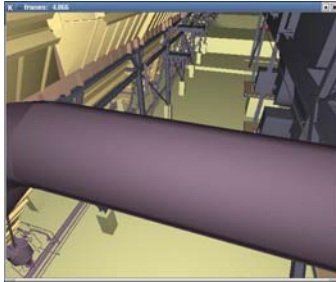


16



Implementation

- Erweiterungen:
- Lichtquelle hinzugefügt und Reflexionen berechnet
- Abfall der Performance durch Schattenföhler und Reflexionen ist proportional zur Anzahl der verfolgten Strahlen

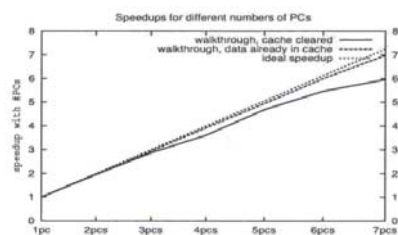


17



Zusammenfassung

- explizites Datenmanagement
- hoher Parallelisierungsgrad: Lastverteilung, Bandbreite, geringe Wartezeiten
- schnell
- gutes Preis-/Leistungsverhältnis
- interaktive Frameraten: 3-5fps
- nur statische Szenen möglich
- Skalierbarkeit:



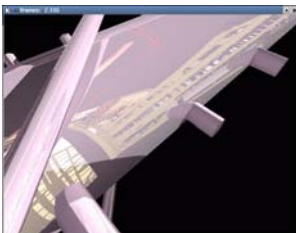
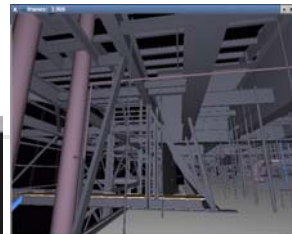
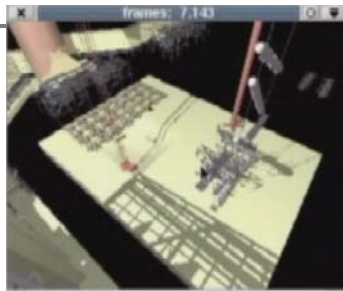
18



Verbesserungen

- schnellere Rechner und Netzwerke
- Aktivieren der SIMD-Erweiterungen: 6-12 fps
- bessere preprocessing Methoden
- verteilte Szenendatenbank
- Bandbreite reduzieren durch Teilen von BSP, Geometrie und Schattenobjekte in einzeln ladbare Objekte

19



20