

Rendering Large Scenes Using Parallel Ray Tracing

Erik Reinhard and Frederik W. Jansen
von September 1996

Vortrag von
Daniel Hienert

Motivation

- das Rendern von komplexen und großen Szenen übersteigt schnell (vor allem damals) die Möglichkeiten eines einzelnen Computers
- heute kann Echtzeitfähigkeit eine Rolle spielen
- das Rendering auf Netzwerken oder Mehrprozessorsystemen kann ermöglicht werden, wenn der Algorithmus effizient und gleichmäßig über die Prozessoren verteilt werden kann
- dieses Paper stellt eine hybride Methode vor, die demand-driven und data-driven-Techniken kombinieren
- die Entscheidung welche der Techniken für den aktuellen Task benutzt wird, entscheidet die Datenintensität und die Datenkohärenz
- damit wird eine gute Datenverteilung und eine gleichmäßige Datenkommunikation über das Netzwerk erreicht

Data parallel ray tracing vs. Demand driven ray tracing

- **Data parallel:**
 - der Ray Tracing-Algorithmus wird in mehrere Prozesse unterteilt
 - die Objektdaten werden auf diese Prozesse verteilt
- **Demand driven:**
 - Jeder Rechner hat die kompletten Objektdaten
 - die zu tracenden Strahlen werden auf die Rechner verteilt

Data parallel ray tracing

- stellt die Basis für den hybriden Algorithmus
- die Objektdaten werden auf diese Prozesse verteilt
- der Ray Tracing-Algorithmus wird in mehrere Prozesse unterteilt

Data parallel ray tracing

- der Objektraum wird in (gleichmäßige) Voxel unterteilt, die lokale Objekt-Kohärenz ausnutzen
- jedem Prozessor wird ein Prozess und ein Voxel mit seinen Objekten zugeteilt
- Ray-Tracing wird dann in einem Master- Slave Setup ausgeführt
- nach der Initialisierung ermittelt der Host-Prozess für jeden Primär-Strahl aus welchem Voxel er stammt (oder in welches er eintritt) und sendet diesen Strahl als einen Task zu dem verbundenen Tracing-Prozess
- dieser Tracing-Process liest den Ray-Task aus dem Buffer und verfolgt den Strahl

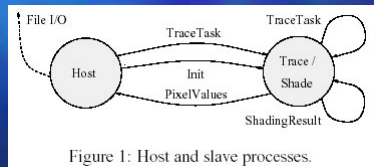


Figure 1: Host and slave processes.

Data parallel ray tracing

1. falls der Strahl das Voxel ohne Schnittpunkt eines Objekts verlässt wird er an das Nachbarvoxel weitergeleitet
 2. falls der Strahl lokale Objekte schneidet werden Sekundärstrahlen aufgespannt und an den Prozess zurückgegeben
- danach erfolgt das Shading und ein Farbwert wird zurückgegeben, der Host sammelt die Information für jedes Pixel und schreibt das Bild

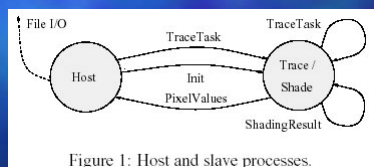


Figure 1: Host and slave processes.

Data parallel ray tracing

Vorteile & Nachteile

- + sehr große Modelle können gerendert werden
- + die Objektdaten müssen nicht für jeden Prozess komplett übergeben werden, sondern können geteilt werden
- + Kommunikation für Ray Tasks ist gering, da Nachbarvoxel meist lokal vorhanden
- - schlechte Verteilung aufgrund von „Hot Spots“ (Lichtquelle, nahe Viewpoint) in der Szene
- - steigender Kommunikationsoverhead mit großer Anzahl von Prozessoren

Demand driven pyramid tracing

- wird dem hybriden Algorithmus zur gleichmäßigeren Lastverteilung hinzugefügt
- Datenkohärenz wird ausgenutzt (dieselbe Quelle und dieselbe Richtung)
- Primärstrahlen vom Viewpoint aus und Schattenstrahlen die zur Lichtquelle geschickt werden können beispielsweise sehr gut als Strahlenbündel in Pyramiden zusammengefasst werden

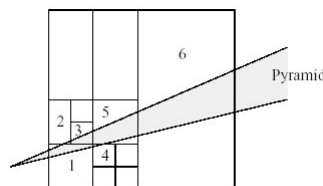


Figure 2: Pyramid traversal generates clip list (cells 1-6).

Demand driven pyramid tracing

- PyraClip (der Zwaan, Rheinhard & Jansen 1995) gibt eine Liste von Zellen geordnet nach Tiefe in einer Bintreestruktur zurück
- 1. nach dem PyraClip PreProcessing kann das Verfolgen einzelner Pyramid-Strahlen nur mit dem Objekten, die in den Zellen liegen demand-driven ausgeführt werden

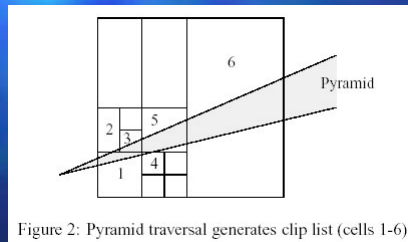


Figure 2: Pyramid traversal generates clip list (cells 1-6).

Demand driven pyramid tracing

- 2. Lichtpyramiden werden vom initialisierenden Prozess (hier linkes Voxel) und nicht von dem Voxel das die Lichtquelle enthält ausgeführt (hier rechts)

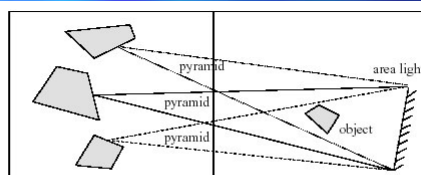


Figure 3: The object in the right voxel is needed for light pyramid tracing by the process managing the left voxel.

Hybrid system

Verteilung des PyraClip Preprocessing & PyraClip RayTracing

- auch das PyraClip PreProcessing wird verteilt
- der Hostprozess vergibt Strahlenbündel auf Nachfrage an den Traceprozess
- die Traceprozesse fertigen gleich nach den data parallel Raytracing das demand-driven PyraClip-Preprocessing und demand-driven PyraClip Ray Tracing ab
- nur die räumlich Teilstruktur wird an jeden Prozess weitergegeben

Hybrid system

- PyraClip erstellt zuerst eine geordnete Liste der benutzten Zellen
- Danach wird der Inhalt jeder Zelle geordnet
- in der Praxis Cache mit residenten Objekten, die dem Viewpoint am nächsten liegen
- falls Schittpunkt mit Objekt -> Transferierung zum Prozess des Objekts (data parallel) -> Sekundärstrahlen (data parallel) -> Schattenstrahlen (data parallel)
- Shading, zurück zum Hostprozess

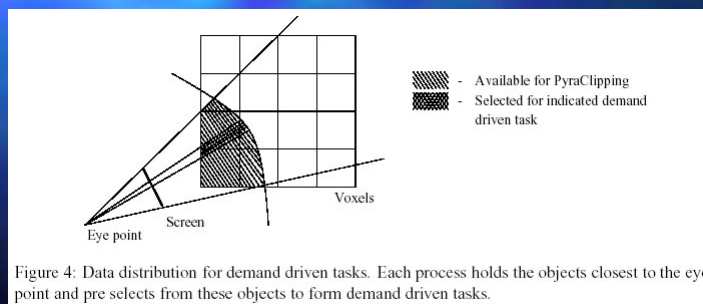


Figure 4: Data distribution for demand driven tasks. Each process holds the objects closest to the eye point and pre selects from these objects to form demand driven tasks.

Hybrid system

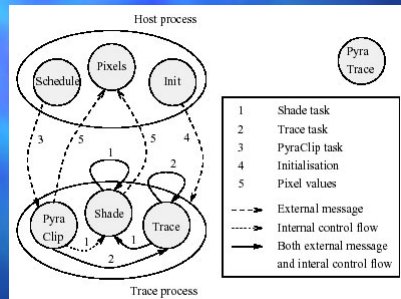


Figure 5: Control and message flow between processes.

Implementation and Experiments

- implementiert in Public Domain Ray Tracer „Rayshade“ 1992
- Parsytec PowerXplorer mit 32 Motorola MPC 601 mit 80 Mhz mit jeweils 32 Mbyte Ram
- SGI Onyx mit einem R8000 Prozessor und 256 Mbyte Ram
- Bildgröße 256x256 Pixel

Implementation and Experiments

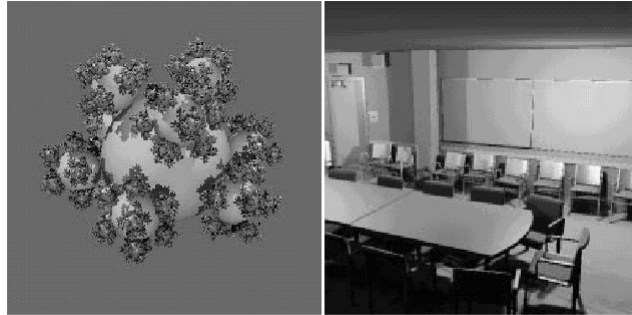


Figure 6: Balls model containing 66.430 spheres (left) and conference room with 23.966 polygons (right).

Comparison between scheduling techniques

- data-parallel Rendering eher schwach
- hybrid Algorithmus besser, aufgrund der besseren Lastverteilung auf die Prozessoren

Table 1: Timings in seconds for balls5 (1 light source) and conference room (8 resp. 30 light sources). o.m = out of memory. Optimal timings for (sequential) standard Rayshade: 34.5 s, 100.0 s. and 264.2 s. respectively.

| Procs | Balls 5 | | Conference room 8 | | Conference room 30 | |
|-------|---------------|--------|-------------------|--------|--------------------|--------|
| | Data parallel | Hybrid | Data parallel | Hybrid | Data parallel | Hybrid |
| 1 | o.m | o.m | o.m | o.m | o.m | o.m |
| 2 | o.m | o.m | 226.7 | 368.9 | 675.0 | 1266.0 |
| 4 | o.m | o.m | 137.0 | 167.7 | 430.7 | 562.0 |
| 8 | o.m | 38.6 | 91.5 | 81.2 | 262.8 | 237.0 |
| 16 | 42.9 | 26.4 | 88.7 | 66.3 | 272.2 | 236.1 |
| 24 | 47.1 | 22.8 | 82.4 | 49.8 | 274.3 | 242.5 |
| 32 | 51.3 | 26.8 | 89.3 | 62.9 | 193.0 | 173.5 |

Comparison between scheduling techniques

- die Prozesse die wenig data-parallel Tasks haben erhalten viele demand-driven Tasks und umgekehrt

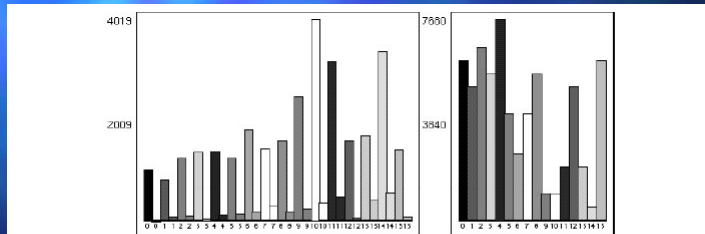


Figure 7: Demand driven rays complement the workload of the data parallel cluster (balls model; 16 processors). The left bars in the left figure represent the data parallel rays per processor. The right bars in the left figure represent the fraction of data parallel rays that was transferred to another processor. The right figure depicts the demand driven rays per processor.

Comparison between scheduling techniques

- nach der Hälfte der Zeit sind die demand-driven Tasks alle beendet und nur Sekundärstrahlen in data-driven Technik werden berechnet
- Lösung: mehr demand driven Tasks (shadow tracing)

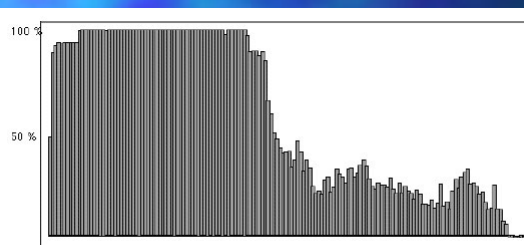


Figure 8: Efficiency over time (balls model; 16 processors). Each bar represents a 0,1s time interval.