Multi-Level Ray Tracing Algorithm (MLRTA)

Kathrin Becker

26. Januar 2010

Zur Erinnerung: Ray Tracing

- Algorithmus (ursprünglich) zum Rastern von Bildern
- Beruht auf Verdeckungsberechnung (Ermittlung der Sichtbarkeit von 3D-Objekten)
- Szenenbeschreibung enthält Position der *Primitive*, lokaler Beleuchtungsmodelle und Lichtquelle(n)
- Kamera- und Bildebenenposition sind gegeben
- Für alle Pixel der Bildebene wird Strahl betrachtet, der durch Kamera und Pixel verläuft
- Mittels Schnittpunkttest und kürzesten Abstände wird das auf der Bildebene sichtbare Primitiv ermittelt

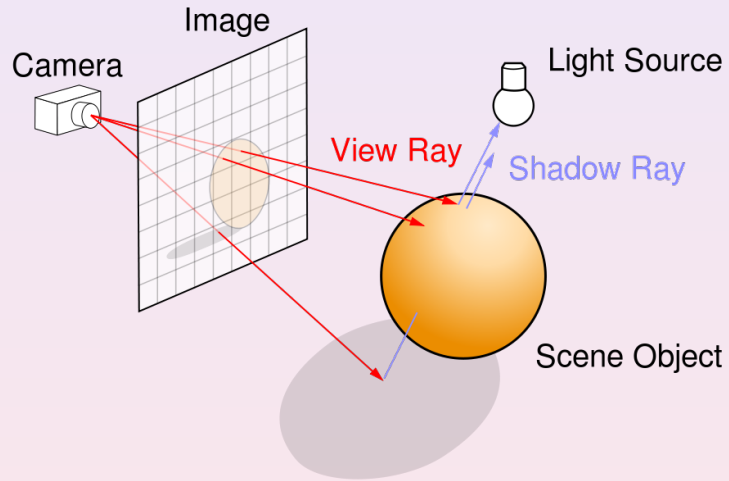


Abbildung: Aus Wikimedia Commons, Autor: Henrik, Link:

`http://de.wikipedia.org/wiki/Raytracing`

Stand: 25.01.2010

Die Idee des Multilevel Ray Tracing Algorithmus (MLRTA)

- Organisation der Szene mittels Suchbaum
- Statt für jeden Strahl den gesamten Suchbaum zu traversieren:
  - Gruppierung von Strahlen
  - Berechnung eines gemeinsamen *Entry Points* (EP) im Suchbaum
  - Schnittpunktsuche für die einzelnen Rays einer Gruppe braucht nur noch unterhalb des EPs statt finden
- MLRTA erlaubt Einsatz von *Multi Processing* und *Multi Threading*

KD-Tree

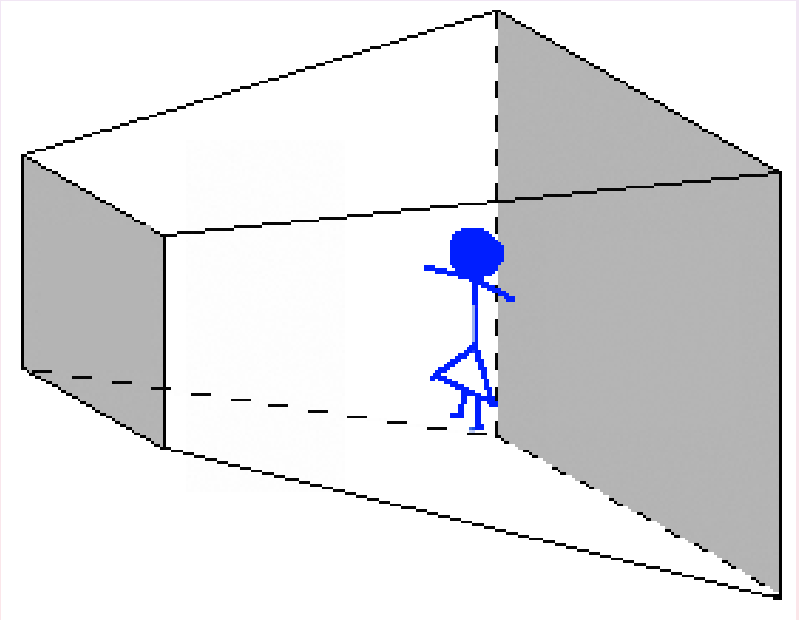
- Haupt-Operation: Splitten des aktuellen Raums in zwei Teilräume mittels Ebene, die *orthogonal* zu einer der Achsen ist
- Rekursives Wiederholen dieser Operation bis Terminationsbedingung erfüllt ist (Kosten fürs Splitten größer als fürs nicht Splitten)
- Splitposition wird folgendermaßen bestimmt:
  - Iteration über Splitkandidaten
  - Für beide Teilräume: Multiplikation der Oberfläche des entstehenden Teilraums mit der Anzahl der Objekte, die diese schneiden
  - Split erfolgt dort, wo dieser Wert minimal ist. (Ausnahmen: 2D Zellen, kleine Zellen und große leere Unterräume)

Zusätzliche Regeln

1. Leerer Teilraum, der mehr als 10 Prozent der Elternzelle einnimmt, hat Vorrang
2. Vollständig von koplanaren Dreiecken bedeckte Splitebenen haben Vorrang, die Primitive werden der kleineren Teilzelle zugeordnet
3. Keine Erzeugung von Zellen, die ein minimales Volumen unterschreiten

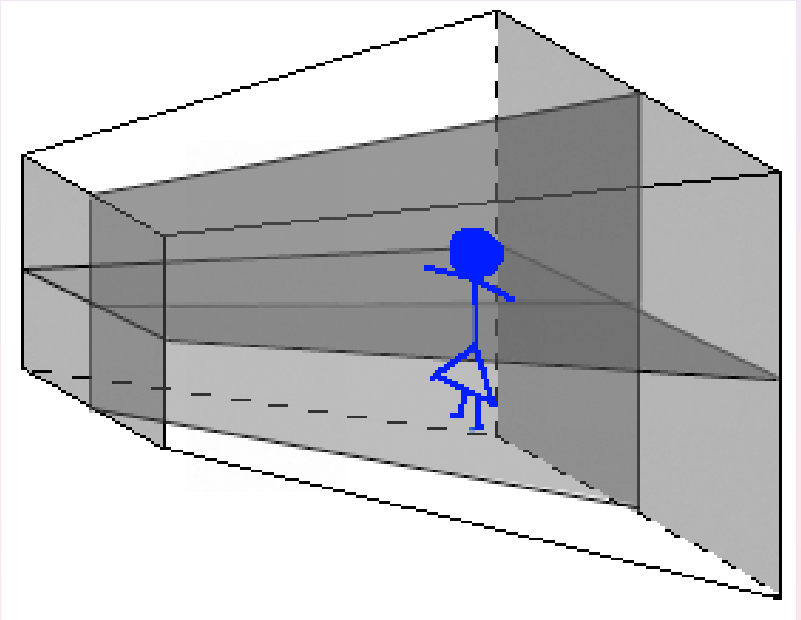
Beispiel zur Gruppierung von Strahlen

- Raytracing der Frau im Sichtfeld
- Durch Kamera und jedes Pixel der Bildebene verlaufe ein Strahl (nicht dargestellt)



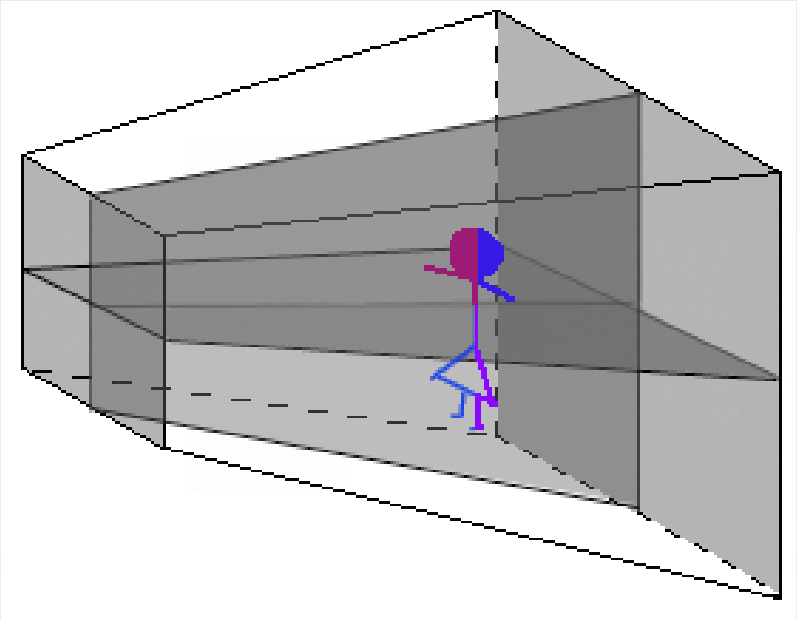
Beispiel zur Gruppierung von Strahlen

- Aufteilung der Strahlen in 4 Gruppen



Beispiel zur Gruppierung von Strahlen

- Jede Strahlengruppe schneidet nur eine Teilmenge der Objekte, überflüssige Knoten könnten aus KD-Baum gestrichen werden (Frustum Culling)
- Für jede Strahlengruppe kann eigener Thread gestartet werden



Funktionsweise von Frustum Culling

- *Frustum Culling* ist Verfahren zum Identifizieren von Objekten, die nicht im *view frustum* liegen.
- Adaptation für Raytracing:
  - Aktueller Strahl/ Gruppe von Strahlen spanne Kegelstumpf auf
  - Boxen im KD-Tree enthalten achsenparallele/-orthogonale Splittebenen
  - Mittels Splitebenen und *frustum*: Test, ob beide oder nur eine Teilzelle getroffen wird.
  - Falls Schnittpunkt in der Box: Knoten bleibt Kandidat für Schnittpunkt
  - Sonst: Mittels Richtung des *frustums* wird entschieden, Teilzelle getrichen werden kann.
  - Blatt ist eine Empty Cell: Kein Schnittpunkt

Frustum Culling - Wrong Decisions

- Nur Klassifikation *no intersection* ist immer korrekt
- In (d) wird Splitebene vom *frustum* außerhalb der Zelle geschnitten (wrong decission)
- Beim MLRTA erfolgt deshalb noch zusätzliche Überprüfung, ob Schnittpunkt außerhalb der Box ist (verringert Zahl der false positives)

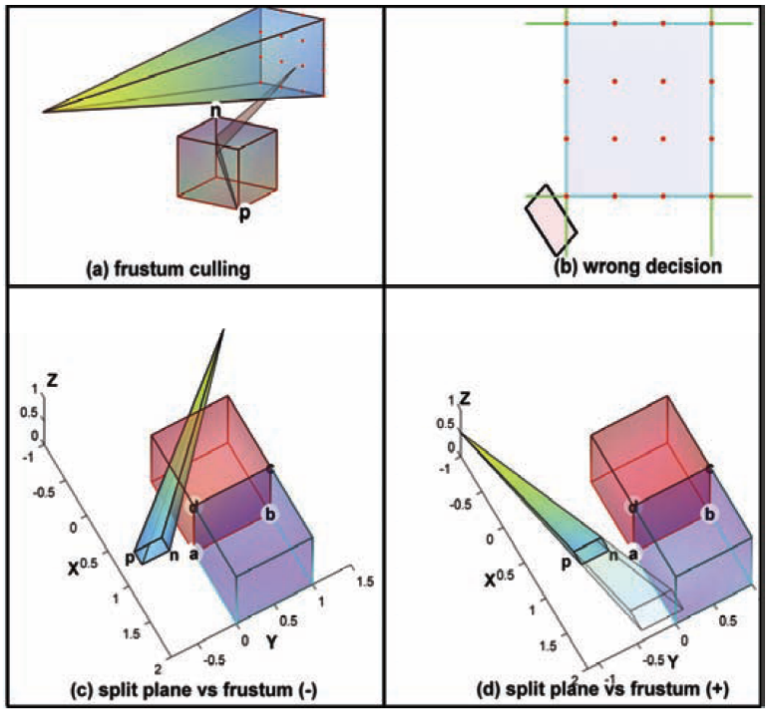


Figure 3: Direct (a,b) and inverse (c,d) frustum culling algorithms.

Abbildung: Aus [1]

Finden eines optimalen EPs im KD-Tree

1. Für jede Strahlengruppe: Durchsuche Baum mit DFS
   - An Gabelungen: Speichere entsprechenden Knoten in LIFO Stack bis erstes Blatt mit möglichen Intersektionen gefunden wurde
   - Markiere diesen Blatt als Kandidaten für Einstiegsknoten
2. Fahre mit der DFS jeweils mit dem obersten Knoten des Stacks fort
   - Falls ein weiteres Blatt mit möglichen Schnittpunkten gefunden wird, so wird der vom Stack entnommene Knoten zum Kandidaten für Einstiegsknoten
3. Terminiere, wenn Stack leer ist und DFS gestoppt hat, gebe aktuellen Kandidaten als optimalen EP zurück

Der Algorithmus - Teil 2

- Sobald der *optimale EP* (entry point) gefunden wurde, wird für jede Strahlengruppe *XP-search* (crossing point search) durchgeführt: Ermittlung der tatsächlichen Schnittpunkte zwischen Strahlengruppe und Zellen des KD-Baums
- Problem: Bei schlechter Gruppierung (z.B. Strahlengruppen mit großem Range) liegt der EP nicht besonders tief im Baum, so dass teure XP search bereits in oberen Suchbaumebenen durchgeführt werden muss

Beispiel

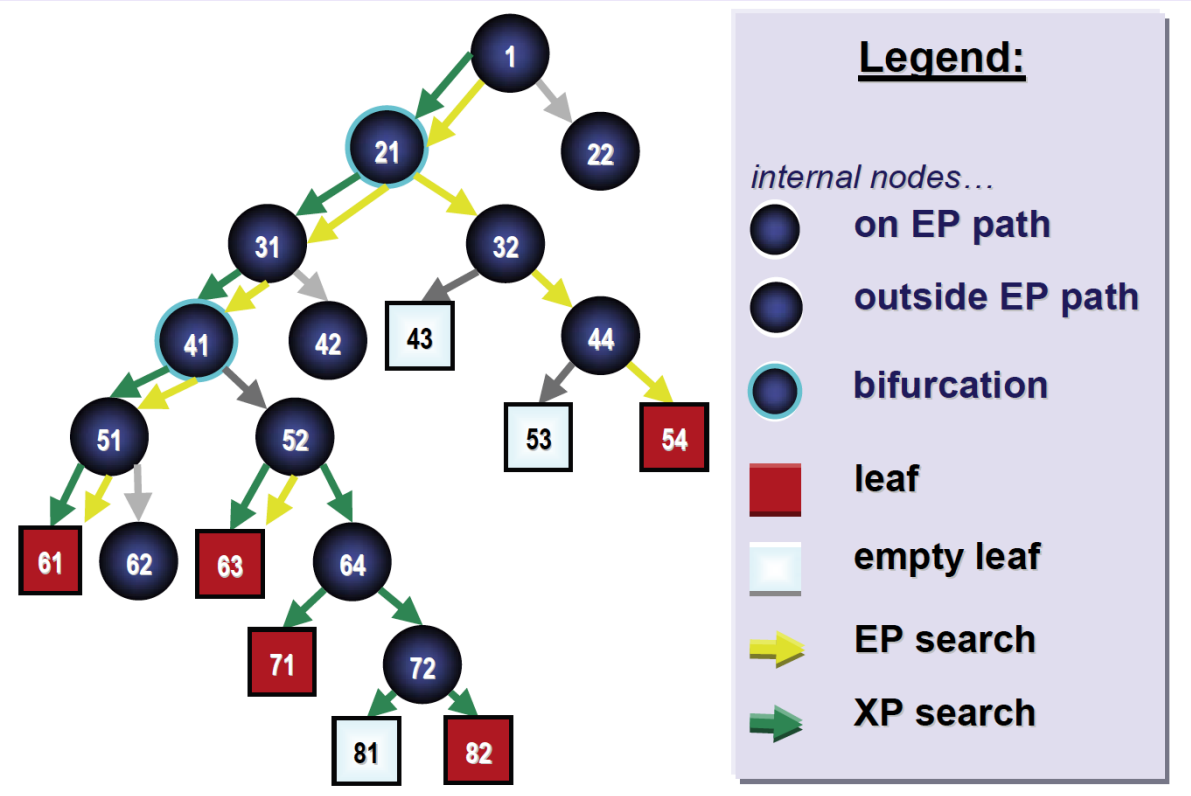


Abbildung: Aus [1]

Tile Splitting

- Wenn Strahlengruppe großen Bereich abdeckt, kann ggf. kein guter Einstiegspunkt bestimmt werden
  - Aufteilung der Bildebene in Kacheln
  - Gruppierung von Strahlen, die die gleiche Richtung haben und die gleiche Kachel schneiden
- Pro Kachel kann z.B. ein Thread gestartet werden

Implementierung von Tile Splitting im MLRTA

- Aufsplittung des Raums nach folgenden drei Parametern:
  1. Initial Tile Size (ITS)
  2. Eine Minimum Tile Size (MTS), bei Unterschreitung wird automatisch die XP-Search gestartet
  3. Ein Split-Factor (SF), der definiert, in wieviele Teile eine Kachel gesplittet werden soll
- Algorithmus arbeitet also wie folgt:
  - Zu Beginn: Aufsplittung gemäß ITS
  - Anschließend: Rekursiv Splits der Kacheln, bis MTS erreicht ist (startet XP-Sarch)

Interval Traversal Algorithmus

- Wenn Informationen über Strahlen vorhanden sind, können Schnittpunkttests parallel durchgeführt werden
- ITA optimiert XP Search, Ablauf:
  - Für alle Strahlen der aktuellen Gruppe Berechnung der minimalen und maximalen Distanz zum aktuellen Unterraum und zu dessen Splitebene; dadurch Ermittlung, ob beide Teilräume von Strahlengruppe getroffen werden
  - Falls beide getroffen werden: Ferner Teilraum wird auf LIFO Stack gelegt, Algorithmus fährt mit nahem Teilraum fort
  - Falls nur ein Teilraum getroffen wird: Algorithmus wird mit diesem fortgeführt
  - Wenn Blatt erreicht ist, wird nächster Unterraum vom Stack entnommen
  - Terminierung, wenn Stack leer und Blatt erreicht

Ergebnisse ohne Shadow Rays

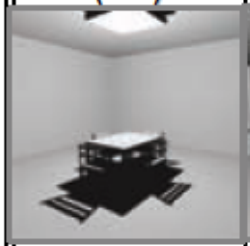
Scene / # of triangles and view / Average measurements per 4x4 packet at 1024×1024 resolution		Erw6 (804)	Conference (274K)	Soda Hall (2195K)	Asian Dragon (7M)
					
number of traversed cells	1. MLRT	3.98	20.87	32.52	32.65
	2. no MLRT	13.00	49.98	71.37	42.18
3. EP search only		0.51	2.30	4.44	2.72
non-masked intersections	4. MLRT	1.09	2.48	1.59	19.97
	5. no MLRT	1.09	2.55	1.52	19.94

Abbildung: Aus [1]

Ergebnisse mit Shadow Rays (analog)

number of traversed cells	6. MLRT	10.07	53.73	69.07	45.01
	7. no MLRT	24.83	101.06	117.22	58.41
non-masked intersections	8. MLRT	1.25	3.71	2.17	23.51
	9. no MLRT	1.22	3.75	2.09	23.48

Abbildung: Aus [1]

Frameraten bei MLRTA und OpenRT

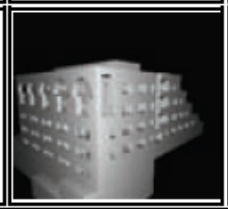
scene / # of triangles / and shader (+/-) — Framerate (FPS) @ 1024x1024 resolution			OpenRT @ 2.5 GHz P4 1 thread	MLRTA @ 2.4 GHz P4 1 thread	MLRTA @ 3.2 GHz P4 with HT 2 threads
Erw6 804		– shader	7.1	70.2	109.8
		+ shader	2.3	37.8	50.7
Confe-rence 274K		– shader	4.55	11.2	19.5
		+ shader	1.93	9.5	15.6
Soda Hall 2195K		– shader	4.12	21.1	35.5
		+ shader	1.8	15.3	24.1

Abbildung: Aus [1]

Ergebnisse

- MLRTA spart insbesondere Traversierungsschritte im KD-Baum
- Bei MLRTA mit 2 Threads: Ohne Shader 9× schneller als OpenRT, mit Shader 12× schneller
- Besonders gute Performance in Szenen mit wenig Komplexität und geringer Occlusion
- MLRTA macht Ray Tracing für Echtzeitberechnungen nutzbar

Referenzen

Alexander Reshetov, Alexei Soupikov, and Jim Hurley.

Multi-level ray tracing algorithm.

In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, pages 1176–1185, New York, NY, USA, 2005. ACM.