

Parallelisierung

- Einfache (triviale) Parallelisierung:
 - "Grobkörnige" Parallelisierung = Verteilung auf mehrere CPU / Cores
 - → daher auch "*thread-level parallelism*" (TLP)
 - Implementierung:
 - mehrere Threads (\approx Prozesse), shared memory
 - mehrere Prozesse, auf mehrere Rechner verteilt, kopiere Szene auf alle Rechner
 - jeder Prozeß / Thread bekommt eine Kachel des Bildes
 - Vorteil: (fast) keine Synchronisation notwendig (nur ganz zum Schluss)
- *Dynamic Load Balancing*:
 - Teile Bild auf in $k \cdot n$ Kacheln, $n = \# \text{ Procs}$, $k = 10 \dots 100$
 - Jeder Prozessor (Worker) holt sich das nächste Work-Paket (eine Bild-Kachel) aus dem Pool, sobald er mit der alten fertig ist
 - Spruch: "ray tracing is embarrassingly parallel"
- Mehr dazu in VL über Verteilte Systeme o.ä.

G. Zachmann Computer-Graphik 2 - SS 08 Ray-Tracing Acceleration 109

Parallelisierung

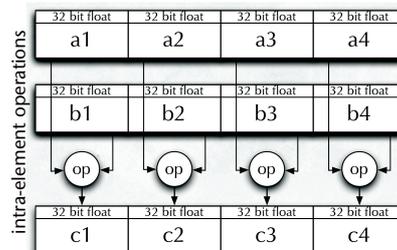
- Weitere Parallelisierungsart: *Instruction-Level Parallelism* (ILP)
- Beispiel:


```
int a = x + y;           // process 1
int b = u + v;           // process 2
int c = a + b;           // wait for proc 1 & 2
```
- Bemerkung:
 - das machen CPU & Compiler heutzutage von alleine
- Bringt für kd-Tree (z.B.) gar nichts:
 - Arbeit pro Knoten beim Traversal =
 - Float laden
 - Branch (für Splitting-Achse x, y, z)
 - Div. & Add.
 - Branch (welches Kind zuerst)
 - Branches machen ILP zunichte

G. Zachmann Computer-Graphik 2 - SS 08 Ray-Tracing Acceleration 110

- Weitere Parallelisierung: *data parallelism*
 - SIMD (*single instruction multiple data*)
 - Alle Register (Float/Int) einer CPU sind **4-fach** vorhanden → Vektor
 - Eine Operation kann auf alle 4 Komponenten gleichzeitig angewandt werden
 - M.a.W.: alle Rechenoperationen sind **gleich zeitaufwendig**, egal ob auf einzelnen Float, oder 4-fach Vektor

- Typischer SIMD-Befehlssatz (AltiVec, SSE):
 - Alle Float/Int-Operationen (Add., Mult., Comp., Round., Load/Store, ...) komponentenweise auf ein Paar von Vektoren ("*intra-element op.*")
 - **Inter-element-Operationen** (permute, pack/unpack, merge, splat, ...)
 - "**Horizontale**" Operationen = "**reduce**" (horizontal subtract, add, ...)
 - Komplexere Op.: Dot product (SSE4)



■ **Permute / Shuffle:**

$T = \text{vec_perm}(A, B, C);$

■ **Compare and Select:**

$\text{vec_cmpeq}()$

$\text{vec_sel}()$

G. Zachmann Computer-Graphik 2 - SS 08 Ray-Tracing Acceleration 113

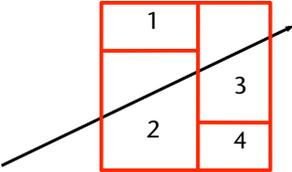
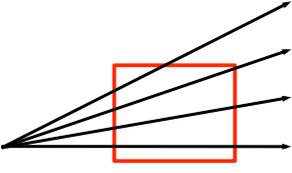
■ **Beispiel 3D-Skalarprodukt**

1 Skalarprodukt

4 Skalarprodukte

G. Zachmann Computer-Graphik 2 - SS 08 Ray-Tracing Acceleration 114

Anwendung auf kd-Tree-Traversal

- Variante: 1 Ray, 4 Objekte**
 - Problem: Daten-"Objekte" müssen von der gleichen Art sein
 - Kontrollfluß muß gleich sein
- Variante: 4 Rays (Ray Packet), 1 Objekt**
 - Daten-"Objekte" sind alle gleich
 - Genug Strahlen sind vorhanden
 - Damit Kontrollfluß gleich ist, müssen Strahlen möglichst dicht beieinander liegen

G. Zachmann Computer-Graphik 2 - SS 08 Ray-Tracing Acceleration 115

SIMD-Algo für Schnitttest Ray-Packet / Box

- Erinnerung: schneide Strahl sukzessive gegen Slabs

```

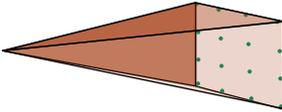
// A/B = linke/rechte Seite der Bbox
// d = Richtungsvektor, O = Aufpunkt des Strahls
// d'_a = 1 / d_a
// alle Operationen, auch min/max, sind komponentenweise!
t_min = -∞
t_max = ∞
loop a = x, y, z:
    t_1 = (A_a ⊖ O_a) ⊙ d'_a
    t_2 = (B_a ⊖ O_a) ⊙ d'_a
    t_min = max( min(t_1, t_2), t_min )
    t_max = min( max(t_1, t_2), t_max )
return ! all_ge(t_min, t_max) && all_le(t_max, 0)

```

liefert 1, wenn alle 4 Komponenten von t_{\min} größer der jew. Komponente in t_{\max} ist

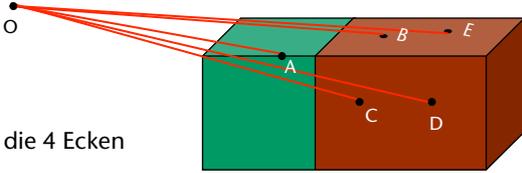
G. Zachmann Computer-Graphik 2 - SS 08 Ray-Tracing Acceleration 116

Frustum-Tracing im kd-Tree [2005]

- Ziel: mehr als nur 4 Strahlen auf einmal
- Verfolge also ganzes Strahlbündel durch kd-Baum
- Idee: repräsentiere Strahlenbündel als Frustum
 
- Bisher: beim Traversieren wurde Entscheidung immer für 1 Strahl getroffen
 - Z.B.: "nur linker Teilbaum" / "nur rechter Teilbaum"
- Beim Packet / Frustum Tracing: treffe "Oder"-Entscheidung für **alle** Strahlen
 - Z.B.: falls 1 Strahl den linken Teilbaum trifft → trace das ganze Paket durch den linken Teilbaum ; dito für rechten Teilbaum

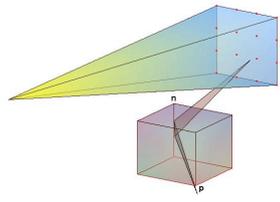
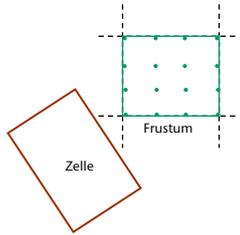
G. Zachmann Computer-Graphik 2 - SS 08 Ray-Tracing Acceleration 117

Gegenbeispiel:

- Erste (problematische) Idee:, checke nur die Eckstrahlen
- Gegenbeispiel:
 
 - Strahlen B, C, D, E sind die 4 Ecken des Strahlbündels
 - Strahl A liegt in der Ebene von B und C
 - Alle 4 Eckstrahlen schneiden nur die rechte Zelle; aber Strahl A schneidet die linke!

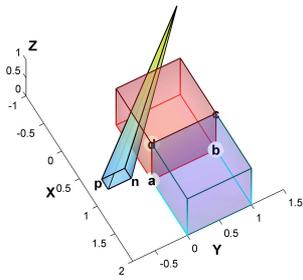
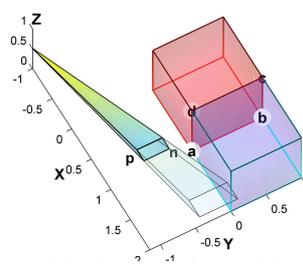
G. Zachmann Computer-Graphik 2 - SS 08 Ray-Tracing Acceleration 118

- Bessere Idee:
 - Verwende die Technik vom View-Frustum-Culling
 - Test: Box (= kd-Tree-Zelle) schneidet Frustum (= BV des Strahlbündels)?
 - Möglicher Algorithmus: wie beim View-Frustum-Culling [Möller]
- Probleme:
 - Frustum hier ist lang & schmal → viele "false positives"
 - Wir machen zu viel Arbeit:
 - Wir wissen schon, daß das Frustum die Vater-Zelle schneidet!

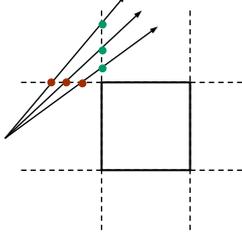
G. Zachmann Computer-Graphik 2 - SS 08 Ray-Tracing Acceleration 119

- Idee: teste Frustum gegen Splitting-Plane ("*inverse frustum culling*")
- Beispiel:
 - \mathbf{d}^i = Richtung der Strahlen
 - $\forall i : \mathbf{d}_x^i > 0$
 - Frustum schneidet Vater
 - Splitting-Plane sei $x=1$
 - Seien die y-Koord. aller Schnittpunkte aller Strahlen $<$ y-Koord. der Zelle (*)
 - Fallunterscheidung:
 - $\forall i : \mathbf{d}_y^i < 0 \rightarrow$ nur die rote Kind-Zelle
 - $\forall i : \mathbf{d}_y^i > 0 \rightarrow$ nur die blaue Kind-Zelle
- Bemerkung: hier genügen wirklich nur die 4 Eckstrahlen!

G. Zachmann Computer-Graphik 2 - SS 08 Ray-Tracing Acceleration 120

- Problem: gibt immer noch "false positives"
- Ziel: genauere Box-Frustum-Test, der für SIMD geeignet ist
- Erste Idee: erweitere Test Box-Strahl auf 4 Strahlen
 - Erinnerung: teste Strahl gegen Folge von Slabs
 - Pro Strahl erhält man ein "t entry" und ein "t exit"
- Problem: könnte zu "false negatives" führen!!
 - Beispiel: siehe 3 Folien früher
 - Hier "false negative" = Test sagt "Frustum schneidet nicht", aber in Wahrheit schneidet es doch!

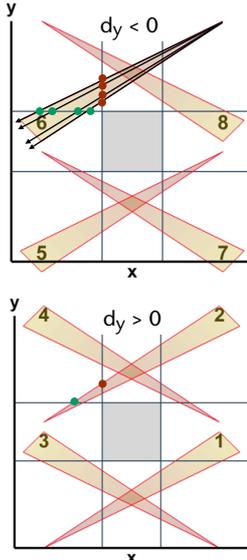


G. Zachmann Computer-Graphik 2 - SS 08 Ray-Tracing Acceleration 121

- Idee: projiziere Frustum auf xy-Ebene und teste dort
 - Man muß nicht die "Randstrahlen" im 2D identifizieren; führe Berechnungen einfach mit allen 4 (projizierten) Eckstrahlen durch (ist gleich teuer, da SIMD)
 - Seien y_i^{entry} die y-Koord. der "Enter"-Schnittpunkte der Strahlen (im 2D) mit den Ebenen der Begrenzungsseiten $y=\text{const}$ der AABB
 - Dito y_i^{exit}
 - Dito für $x \rightarrow x_i^{\text{entry}}, x_i^{\text{exit}}$
 - Es gibt 8 Fälle, 2 Tests genügen:

$$\min\{y_i^{\text{entry}}\} > \max\{x_i^{\text{exit}}\} \vee (1,3,6,8)$$

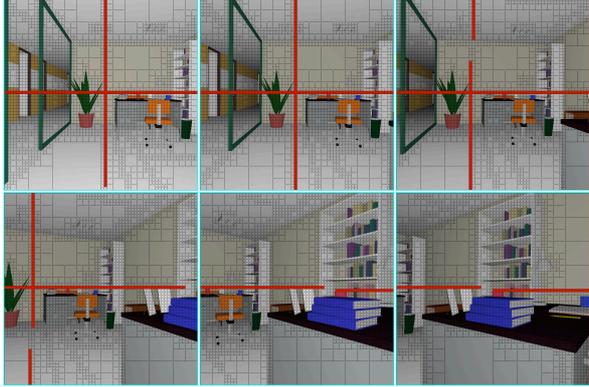
$$\min\{x_i^{\text{entry}}\} > \max\{y_i^{\text{exit}}\} \quad (2,4,5,7)$$



G. Zachmann Computer-Graphik 2 - SS 08 Ray-Tracing Acceleration 122

Adaptive Tile / Frustum Splitting

- Starte mit "großen" Strahlenbündeln (= Frusta) als "Primärstrahlen"
- Versuche, damit den kd-Tree zu traversieren
- Spalte Frustum auf, wenn die Bedingungen (*) für den Frustum-Zellen-Test nicht (mehr) gegeben sind

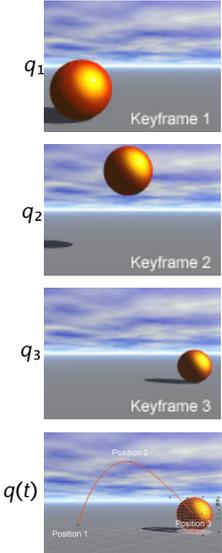


(Courtesy Reinhard et al.)

G. Zachmann Computer-Graphik 2 - SS 08 Ray-Tracing Acceleration 123

Keyframe Animationen

- Wie beschreibt man eine stetigen Pfad eines Objektes?
- Wie beschreibt man eine stetige Deformation?
- Prinzipielle Idee:
 - Spezifiziere die Position des Objektes zu verschiedenen Zeitpunkten → **Keyframes**
 - Das System interpoliert alle Frames dazwischen:
 - Interpolation der definierenden Parameter, z.B. Translation / Rotation, Gelenkwinkel, Vertex-Positionen
 - Interpolation mittels Splines



q_1 Keyframe 1

q_2 Keyframe 2

q_3 Keyframe 3

$q(t)$

Position 1 Position 2

G. Zachmann Computer-Graphik 2 - SS 08 Ray-Tracing Acceleration 124

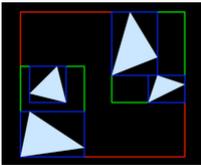
Dynamische Szenen

- Problem:
 - Alle Vertices bewegen sich (Animation / Simulation)
 - kd-Tree / Gitter / BVH wird ungültig (und viele andere DS ebenso)
- Naïve Idee:
 - In jedem Frame Beschleunigungsdatenstruktur neu aufbauen (nachdem neue Position der Vertices berechnet ist)
 - Kann man beim Gitter machen, aber zu teuer für alle anderen DSen

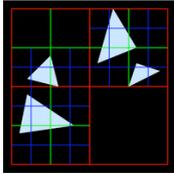
G. Zachmann Computer-Graphik 2 - SS 08 Ray-Tracing Acceleration 125

Was ist an Gittern so speziell?

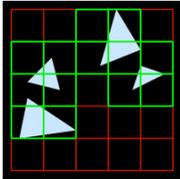
- Seit den 70-ern: viele *acceleration data structures*



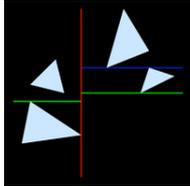
BVH



Octree



Grid



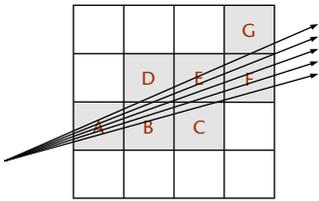
Kd-tree

- Von allen ist nur das Gitter nicht-hierarchisch!

G. Zachmann Computer-Graphik 2 - SS 08 Ray-Tracing Acceleration 126

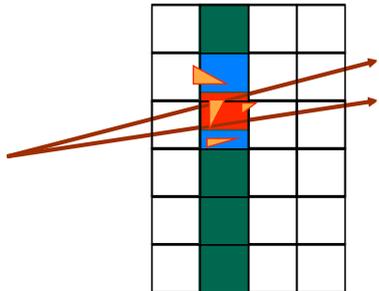
Coherent Grid Traversal [2006]

- Ziel: Strahlen-Pakete durchs Gitter beschleunigen (mit SIMD)
- Problem: Traversal ist inkompatibel mit Packet Tracing
 - In welcher Folge besucht man die Zellen? ABCD oder ABDC?
 - Inkrementelle Traversal-Algos (Midpoint, 3DDDA) sind nicht mehr SIMD-fähig, sobald Strahlen auseinanderlaufen
 - Entscheidungsvariable für verschiedene Rays im selben Paket verschieden!
 - Pakete aufteilen degeneriert schnell zum Einzelstrahl-Traversal
- Idee:
 - Pakete funktionieren **nicht** mit einem Gitter...
 - ... aber **Frusta** schon.



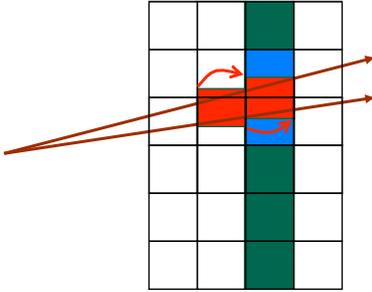
G. Zachmann Computer-Graphik 2 - SS 08 Ray-Tracing Acceleration 127

- Verwende uniformes Gitter
- Bestimme für ein Paket von Strahlen die obere/untere/linke/rechte bounding plane → "achsenparalleles" Frustum
- Traversiere mit Frustum durch das Gitter **schichtenweise**
 - Bestimme Overlap-Box zwischen Frustum und Gitter-Schicht
 - Runde auf ganzzahlige Indizes → überdeckte Zellen
 - Schneide alle Dreiecke in überdeckten Zellen



G. Zachmann Computer-Graphik 2 - SS 08 Ray-Tracing Acceleration 128

- Die Overlap-Box kann man inkrementell, von Schicht zu Schicht, aktualisieren
 - Trivial, da die Bounding Planes des Frustums bekannt sind und achsenparallel
 - Insgesamt pro Schritt 4 Additionen (= 1 SIMD-Op.)
 - Unabhängig von der Anzahl der Strahlen im Frustum
- Dazu noch SIMD-Frustum-Culling, um Dreiecke zu entfernen, die das Frustum nicht schneiden



Diplomarbeit ...

G. Zachmann Computer-Graphik 2 - SS 08 Ray-Tracing Acceleration 129

Bemerkungen

- Rel. teure Setup-Phase
 - Frustum berechnen, Setup des inkrementellen Algos
- Sehr billiger Update-Schritt von Schicht zu Schicht
- Sehr gut geeignet für dynamische Szenen:
 - Wiederaufbau = wenige Millisek für ~100.000 Dreiecke (1 Proc)
 - Einfach zu Parallelisieren: 10 MTris in ~150 ms (16 Opterons)
- Hierarchische Gitter: im Prinzip möglich
- Ähnlich wenige Schnittberechnung (Strahl-Obj) wie beim Kd-Tree
- Kleiner Nachteil: man muß Mailboxes verwenden
 - Gitter ohne FC & MB : 14 M ray-tri isecs
 - Gitter mit FC & MB : .9 M ray-tri isecs (14x less)
 - Kd-tree : .85M ray-tri isecs (5% kleiner als Gitter)
- Insgesamt: nur ~2x langsamer als BVH und Kd-Tree, aber dafür für dynamische Szenen!

G. Zachmann Computer-Graphik 2 - SS 08 Ray-Tracing Acceleration 130

Wahl der optimalen Paketgröße

- Kosten des Traversal-Schrittes ungefähr unabhängig von der Anzahl Strahlen →
 - Größere Pakete = mehr „Potential“ für Amortisation (+)
- Mehr Strahlen/Paket = größeres Frustum →
 - Mehr besuchte Zellen, mehr Dreiecke, die gegen alle Strahlen im Paket getestet werden müssen (-)
- "Sweet spot":
 - Am Besten ist 4x4 (grün) oder 8x8 (blau)

#triangles	2x2	4x4	8x8	16x16	52x32
0	0.00	0.00	0.00	0.00	0.00
50k	0.10	0.04	0.03	0.02	0.01
100k	0.12	0.05	0.04	0.03	0.02
150k	0.13	0.06	0.05	0.04	0.03
200k	0.14	0.07	0.06	0.05	0.04
250k	0.15	0.08	0.07	0.06	0.05
300k	0.16	0.09	0.08	0.07	0.06
350k	0.17	0.10	0.09	0.08	0.07

G. Zachmann Computer-Graphik 2 - SS 08 Ray-Tracing Acceleration 131

Resultate

- Dual-Xeon 3.2GHz, 1024x1024, ohne Schattierung, reine Anim.

"Hand"
16K triangles
34.5/15.3 fps

"Runner"
78K triangles
15.8/7.8 fps

"Toys"
11K triangles
29.3/10.2 fps

"Marbles"
8.8K triangles
57.1/26.2 fps

X/Y fps:
X=raycast only
Y=raycast+shade+texture+shadows

G. Zachmann Computer-Graphik 2 - SS 08 Ray-Tracing Acceleration 132

- Video (Fee)
 - 174k tris, 1024x1024 Pixels, 16-core Opteron (180 GFLOPs)
 - CELL = 256 GFLOPs
 - ATI X1900 ~ 1000 GFLOPs
 - 3.4 fps (raycast only)
 - 1.2 fps (raycast + shade + texture + shadows)



G. Zachmann Computer-Graphik 2 - SS 08 Ray-Tracing Acceleration 133