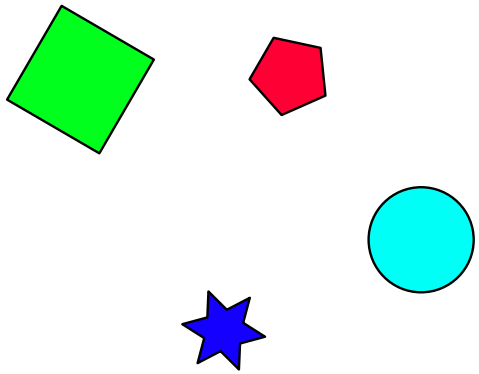
Karlsruher Institut für Technologie

Parallele BVH-Konstruktion auf der GPU

Daniel Opitz

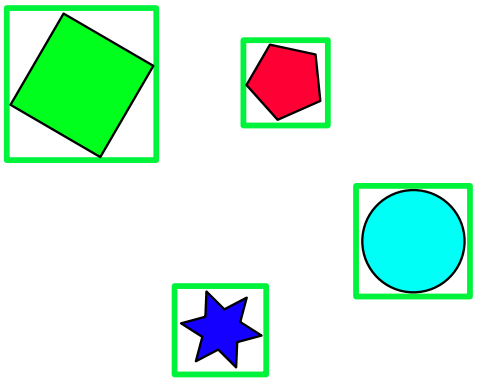
Institut für Visualisierung und Datenstrukturen

BVH Konzeptionell



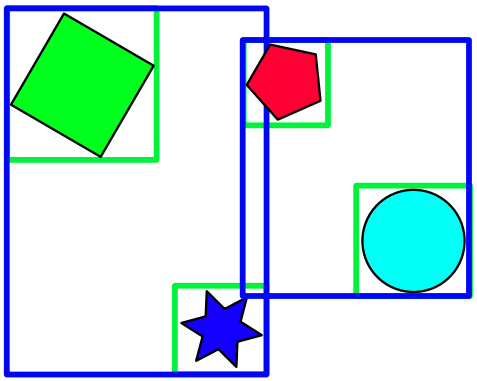
- Objekte im Raum
- Bounding Boxen für Objekte
- Konservativ Bounding Boxen zusammenfassen

BVH Konzeptionell



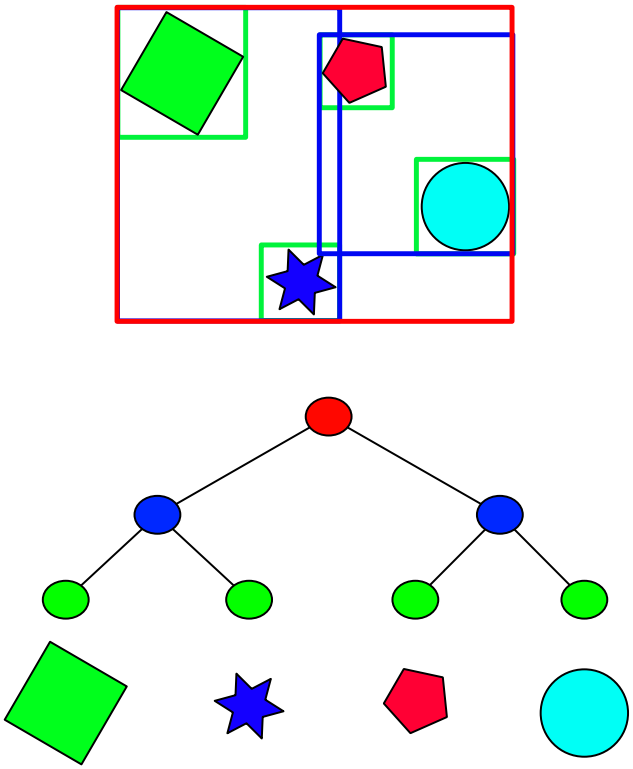
- Objekte im Raum
- Bounding Boxen für Objekte
- Konservativ Bounding Boxen zusammenfassen

BVH Konzeptionell



- Objekte im Raum
- Bounding Boxen für Objekte
- Konservativ Bounding Boxen zusammenfassen

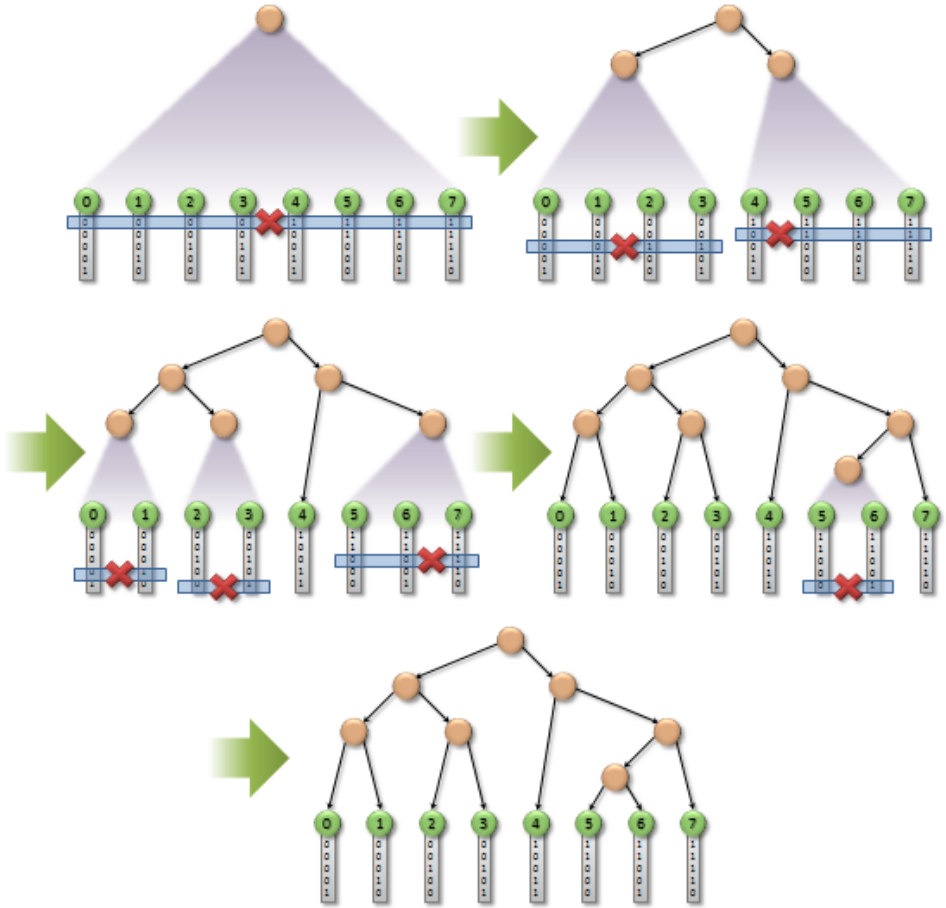
BVH Konzeptionell



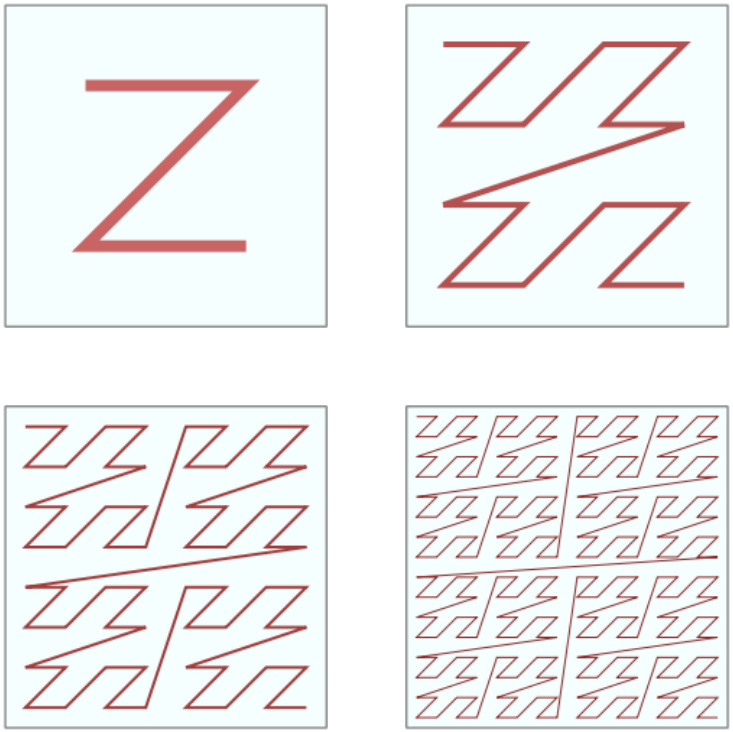
- Objekte im Raum
- Bounding Boxen für Objekte
- Konservativ Bounding Boxen zusammenfassen

BVH Konstruktion

Welche Knoten Zusammenfassen?

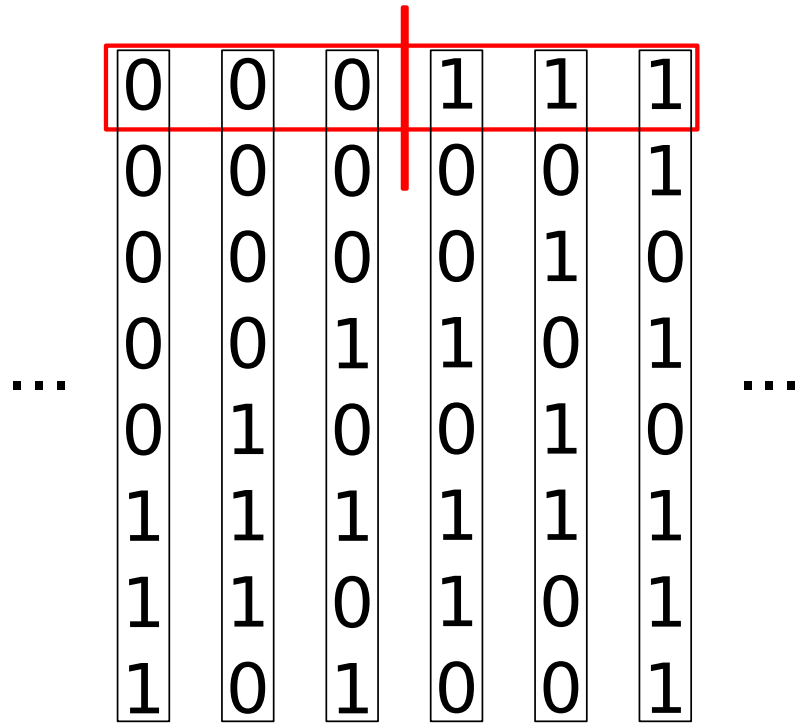


BVH Konstruktion



- Nutze Z-Order-Curve um gute Split-Ebenen zu finden
- Berechne für alle Objekte (Blätter) den zugeörigen Morton-Code
- Sortiere Objekte anhand ihres Morton-Codes
- Finde Split-Ebenen anhand gleicher Präfixe

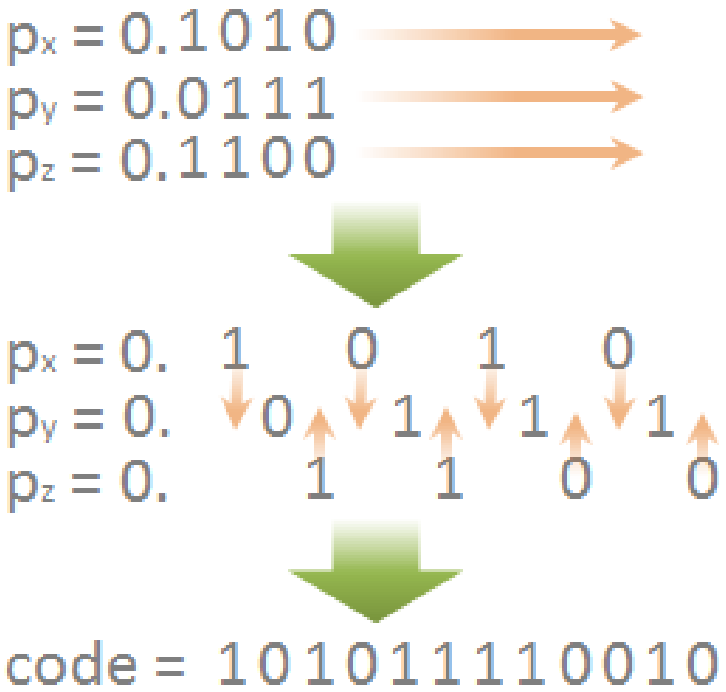
BVH Konstruktion



- Nutze Z-Order-Curve um gute Split-Ebenen zu finden
- Berechne für alle Objekte (Blätter) den zugeörigen Morton-Code
- Sortiere Objekte anhand ihres Morton-Codes
- Finde Split-Ebenen anhand gleicher Präfixe

BVH - Teilprobleme

Morton Codes



- Berechne Codes für Schwerpunkte der Blätter
- Benötigt Koordinaten in $[0, 1]$ $\Rightarrow$ Reduktion zu Wurzel-AABB
- Padding der Nachkommateile von x,y und z Komponenten
- Verordern zu finalem Morton-Code

BVH - Teilprobleme

Radix Sort

keys | B | A | 3 | 8 | 5 | F | 1 | 0 |

- Input sind Morton-Codes
- Bestimme ob k-tex bit 0 oder 1
  $\texttt{flags}_0[i] = !\texttt{keys}[i]_k$
  $\texttt{flags}_1[i] = !\texttt{flags}_0[i]$
- Berechen Präfixsumme über $\texttt{flags}$
- Ordne $\texttt{keys}$ neu

BVH - Teilprobleme

Radix Sort

keys	B	A	3	8	5	F	1	0
$\texttt{flags}_0$	0	1	0	1	0	0	0	1
$\texttt{flags}_1$	1	0	1	0	1	1	1	0

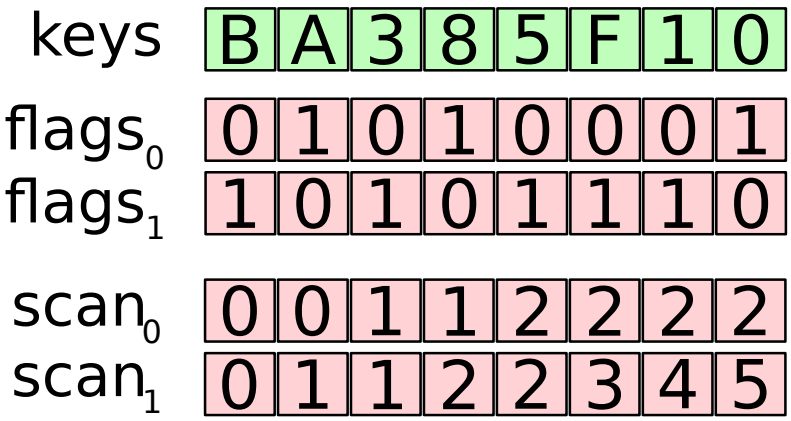
- Input sind Morton-Codes
- Bestimme ob k-tex bit 0 oder 1

  $\texttt{flags}_0[i] = !\texttt{keys}[i]_k$

  $\texttt{flags}_1[i] = !\texttt{flags}_0[i]$
- Berechen Präfixsumme über `flags`
- Ordne `keys` neu

BVH - Teilprobleme

Radix Sort



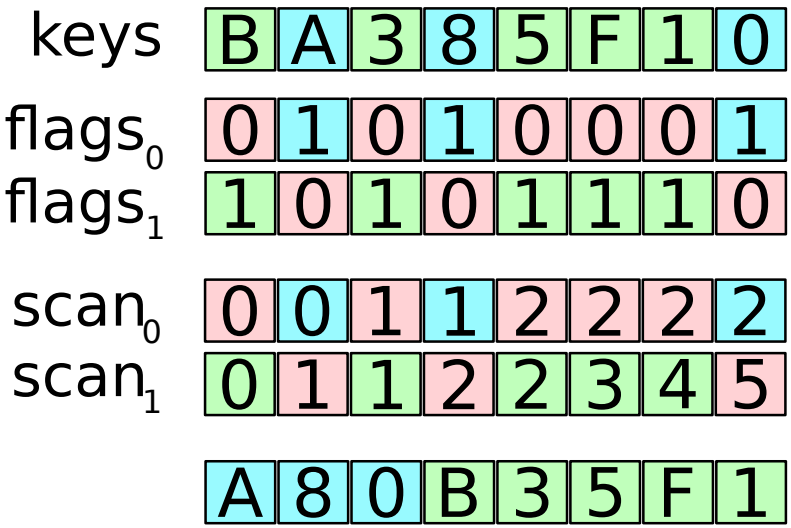
- Input sind Morton-Codes
- Bestimme ob k-tex bit 0 oder 1

  $\texttt{flags}_0[i] = !\texttt{keys}[i]_k$

  $\texttt{flags}_1[i] = !\texttt{flags}_0[i]$
- Berechen Präfixsumme über `flags`
- Ordne `keys` neu

BVH - Teilprobleme

Radix Sort



- Input sind Morton-Codes
- Bestimme ob k-tex bit 0 oder 1
  $\texttt{flags}_0[i] = !\texttt{keys}[i]_k$
  $\texttt{flags}_1[i] = !\texttt{flags}_0[i]$
- Berechen Präfixsumme über `flags`
- Ordne `keys` neu

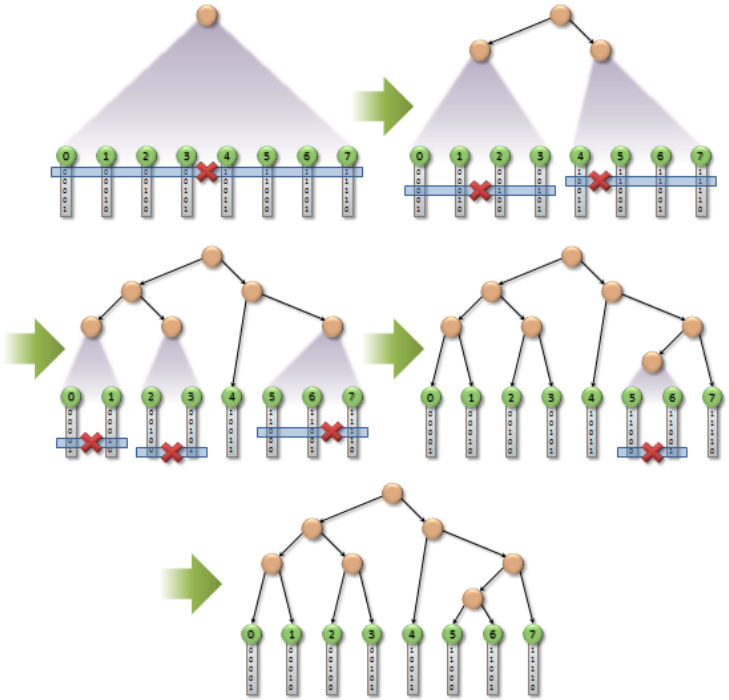
BVH - Teilprobleme

Radix Sort

- Für jedes Bit k (LSB $\rightarrow$ MSB)
  - Extrahiere k-tex Bit aus Morton-Codes
  - Berechne zwei Präfixsummen (0- und 1-Flags)
  - Ordne Morton-Codes *und Permutations-Array* neu
- $32 \cdot 2$ Präfixsummen
- Reorder-Schritt dominiert Laufzeit
- Breakeven (Laptop) bei $\sim$15K Elementen

BVH Konstruktion

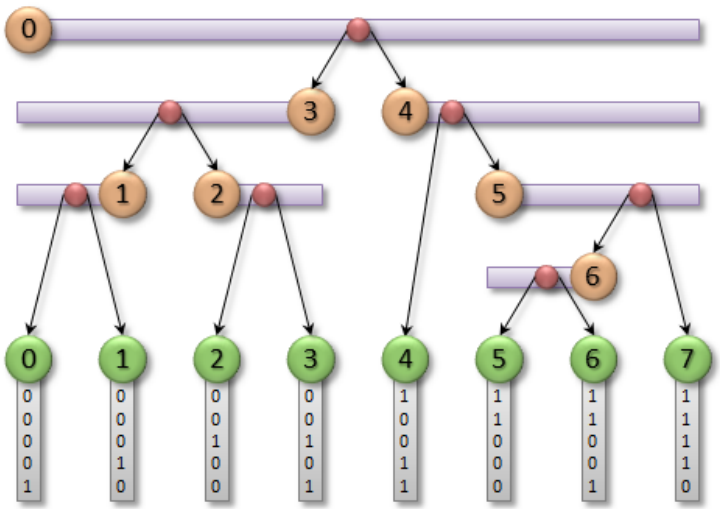
Hierarchie erstellen



- Innere Knoten repräsentiert als Index-Range
- Kindknoten unterteilen Index-Range des Elternknoten an genau einer Stelle
- Hierarchisches Vorgehen von Wurzel richtung Blätter möglich
- **Aber**: Schlechte Auslastung der GPU

BVH Konstruktion

Hierarchie erstellen



- **Besser**: Finde für jeden Knoten zu erst seine Index-Range
  - Knoten-Startposition
    + zwei binäre Suchen
- Finde Split-Index mit binärer Suche
- Mehr arbeit in jedem Thread, aber alle inneren Knoten parallel

Demo