HLBVH: Hierarchical LBVH Construction for Real Time Ray Tracing of Dynamic Geometry

Jacopo Pantaleoni and David Luebke
NVIDIA Research

hpg 2010

Some Background

- **Real Time Ray Tracing is almost there***

  [Garanzha and Loop 2010, Aila and Laine 2009, Wald et al 2007, ...]

  160-200 M rays/s on GF480

Some Background

- **Real Time Ray Tracing is almost there***

  [Garanzha and Loop 2010, Aila and Laine 2009, Wald et al 2007, ...]

160-200 M rays/s on GF480

*** but only for static scenes**

Some Background

- **Real Time Ray Tracing is almost there***

  [Garanzha and Loop 2010, Aila and Laine 2009, Wald et al 2007, ...]

  160-200 M rays/s on GF480

  *** but only for static scenes**

- **Spatial Index construction real-time only for 100K tris!**

Some Background

- **Real Time Ray Tracing is almost there***

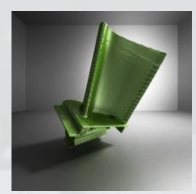
  [Garanzha and Loop 2010, Aila and Laine 2009, Wald et al 2007, ...]

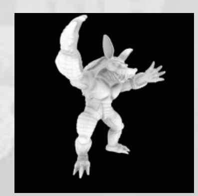
  160-200 M rays/s on GF480

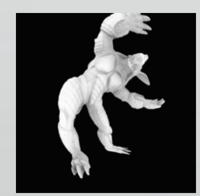
  *** but only for static scenes**

- **Spatial Index construction real-time only for 100K tris!**

- **Our target is 1M dynamic tris**







Some Background

- Many approaches: refitting, partial rebuilds...
  but LBVH [Lauterbach et al] probably fastest
  available GPU builder

Some Background

- Many approaches: refitting, partial rebuilds...
  but LBVH [Lauterbach et al] probably fastest
  available GPU builder

- still not fast enough...   1M tris   => ~150ms

Some Background

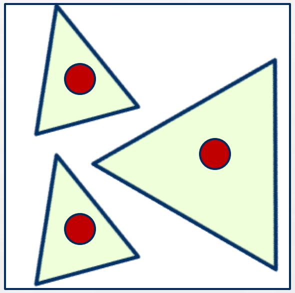
- Many approaches: refitting, partial rebuilds...
  but LBVH [Lauterbach et al] probably fastest
  available GPU builder

- still not fast enough...   1M tris   => ~150ms

- But could be made faster! ☺

LBVH

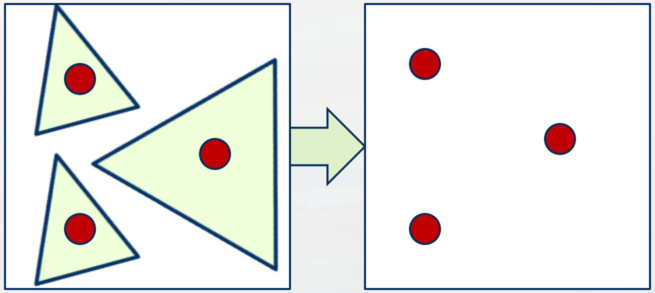
- **Consider barycenters of each primitive**



LBVH

- **Consider barycenters of each primitive**

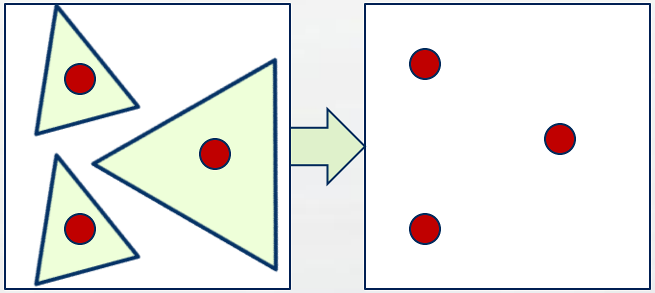
  so that it works with point sets



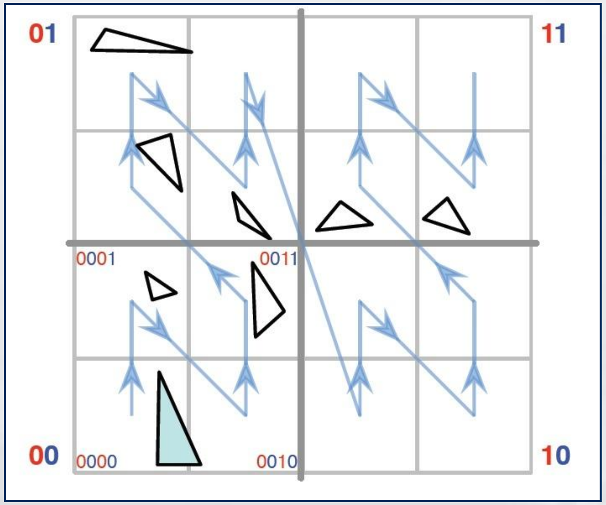
LBVH

- **Consider barycenters of each primitive**

  so that it works with point sets



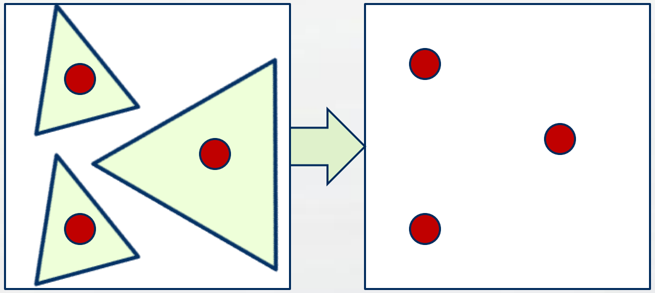
- **sort them along a 1D Morton curve through a grid...**



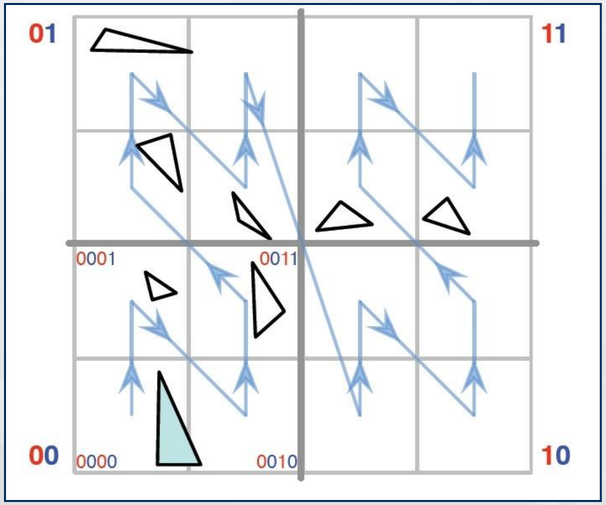
LBVH

- **Consider barycenters of each primitive**

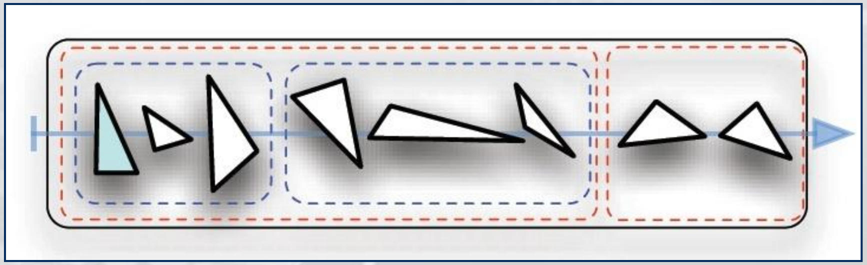
  so that it works with point sets



- **sort them along a 1D Morton curve through a grid...**



- **and group them by cell**



LBVH

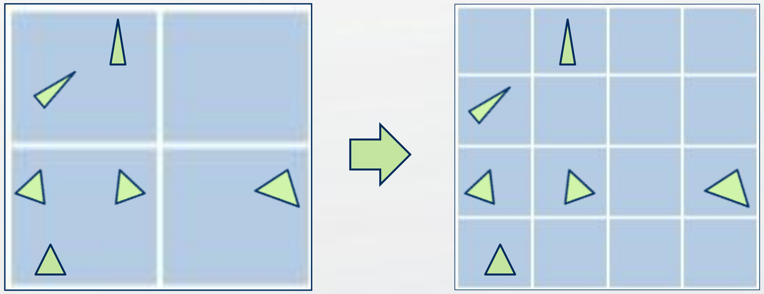
- Morton codes computed using 10 bits per component
- primitives sorted with a single 30bit global sort
- parallel hierarchy emission required 2 additional sorting operations on $\Omega(N * 30)$ split planes

HLBVH = Hierarchical LBVH

H(ierarchical)LBVH

HLBVH: at a glance

- hierarchical process



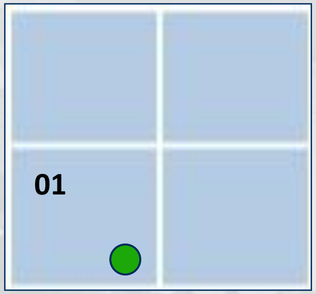
- exploit spatial and temporal coherence in the input mesh
- novel hierarchy emission algorithm
- novel SAH hybrid

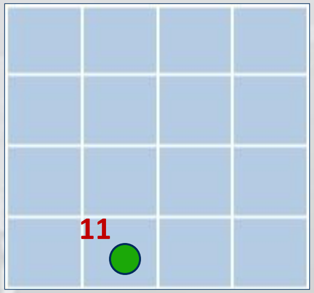
- Given a point its *Morton code* is obtained interleaving the bits of its coordinates:

e.g. (0100, 1001, 0111) => 010101001011

- Each triplet of bits => next octant in a grid hierarchy:

2D example: 0111

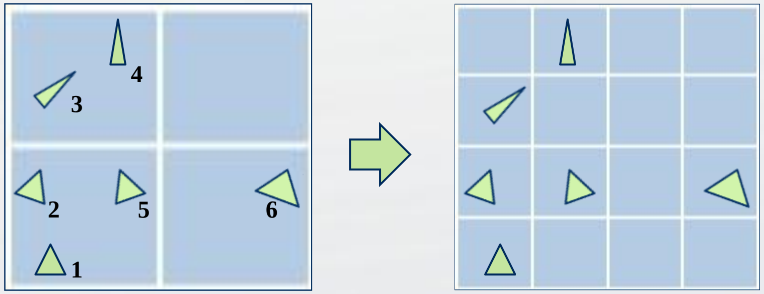




HLBVH: primitive sorting

- **Consider a 2 level hierarchy:**

coarse:	**3m**	**bits**
fine:	**3n**	**bits**

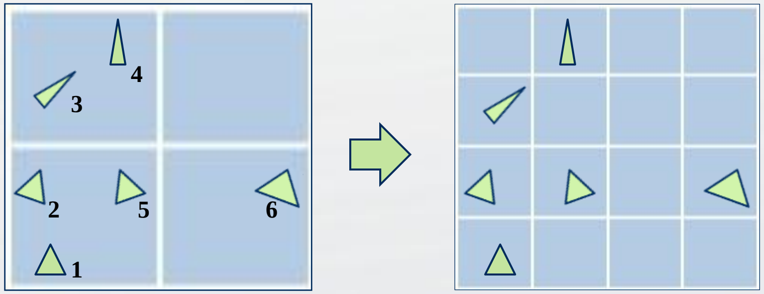


HLBVH: primitive sorting

- Consider a 2 level hierarchy:

  coarse: 3m bits

  fine: 3n bits



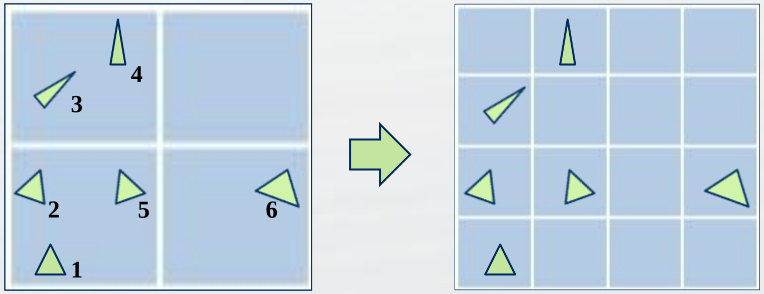
- smaller m => higher chances consecutive prims fall in the same voxel (e.g. {1,2}, {3,4})

HLBVH: primitive sorting

- Consider a 2 level hierarchy:

coarse: 3m bits

fine: 3n bits



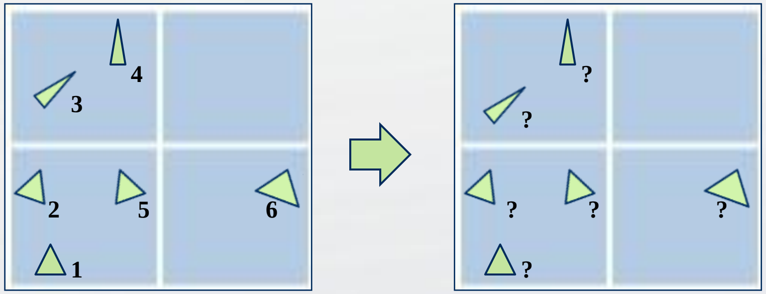
- smaller m => higher chances consecutive prims fall in the same voxel (e.g. {1,2}, {3,4})

- Exploit coherence:

Compress-Sort-Decompress [Garanzha and Loop 2010] within coarse grid

HLBVH: primitive sorting (part 1)

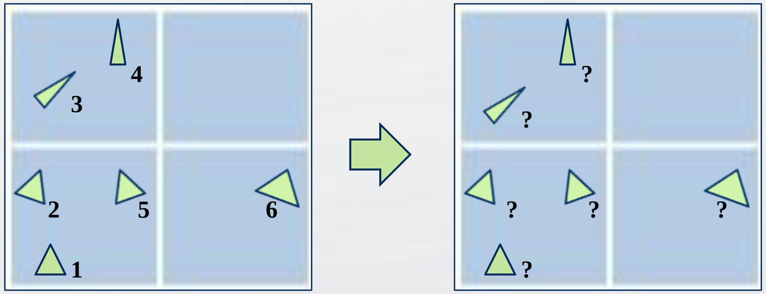
- Compute n-bit Morton codes



- Compress: run-length encode based on first 3m bits

HLBVH: primitive sorting (part 1)

- Compute **n**-bit Morton codes

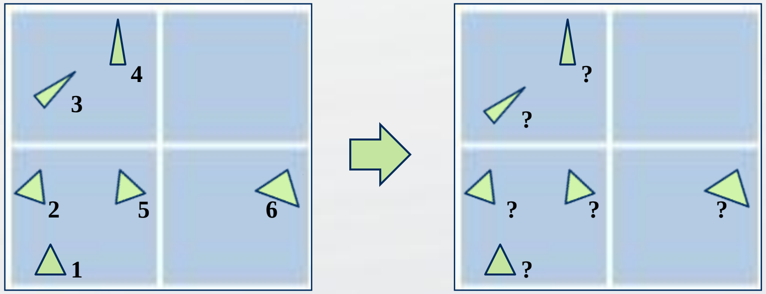


- Compress: run-length encode based on first 3**m** bits

- Sort : do a 3**m**-bit radixsort of the rle key blocks

HLBVH: primitive sorting (part 1)

- Compute n-bit Morton codes



- Compress: run-length encode based on first 3m bits

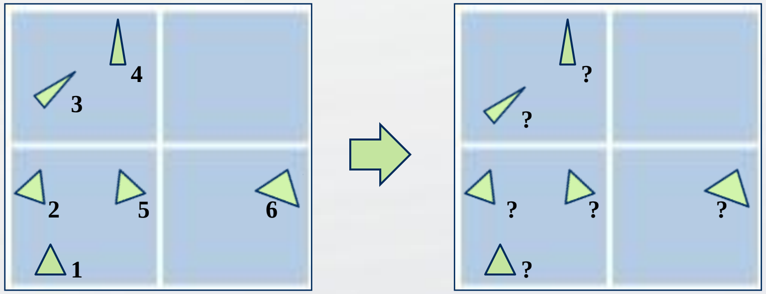
- Sort : do a 3m-bit radixsort of the rle key blocks

- Decompress: run-length decode sorted keys

HLBVH: primitive sorting (part 1)

- **CSD at work:**

  { 7, 7, 1, 1, 1, 3, 3, 4, 5, 5 }



HLBVH: primitive sorting (part 1)

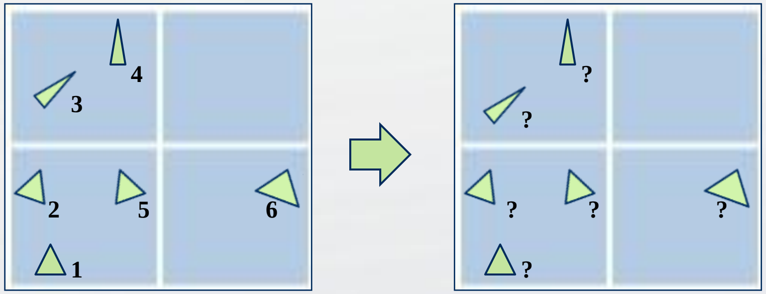
- **CSD at work:**

  { 7, 7, 1, 1, 1, 3, 3, 4, 5, 5 }

- **Compress:**

  { 7, 1, 3, 4, 5 } run values

  { 2, 3, 2, 1, 2 } run lengths



HLBVH: primitive sorting (part 1)

- **CSD at work:**

  { 7, 7, 1, 1, 1, 3, 3, 4, 5, 5 }

- **Compress:**

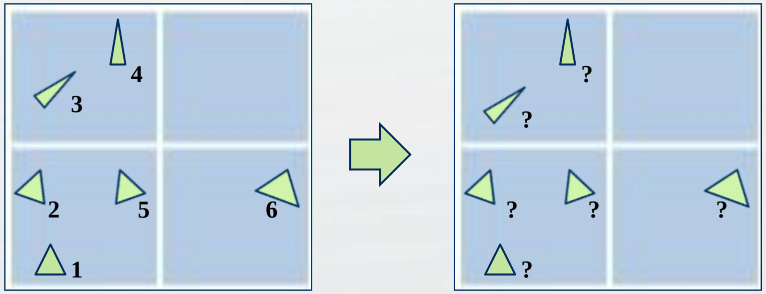
  { 7, 1, 3, 4, 5 } run values

  { 2, 3, 2, 1, 2 } run lengths

- **Sort:**

  { 1, 3, 4, 5, 7 } run values

  { 3, 2, 1, 2, 2 } run lengths



HLBVH: primitive sorting (part 1)

- **CSD at work:**

  { 7, 7, 1, 1, 1, 3, 3, 4, 5, 5 }

- **Compress:**

  { 7, 1, 3, 4, 5 } run values

  { 2, 3, 2, 1, 2 } run lengths

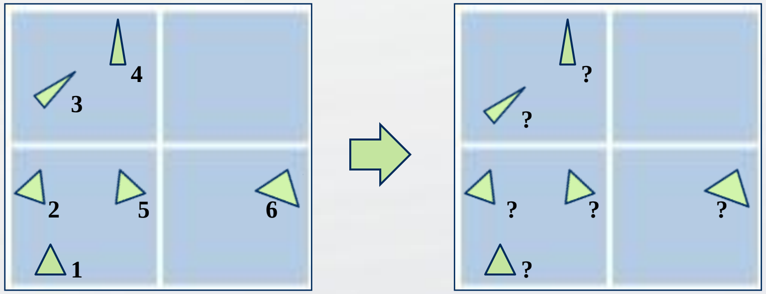
- **Sort:**

  { 1, 3, 4, 5, 7 } run values

  { 3, 2, 1, 2, 2 } run lengths

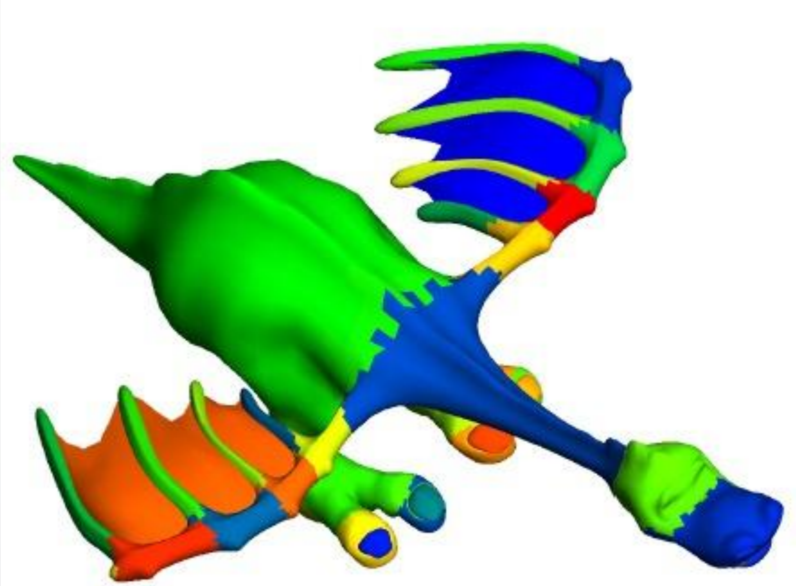
- **Decompress:**

  { 1, 1, 1, 3, 3, 4, 5, 5, 7, 7 }



HLBVH: primitive sorting (part 1)

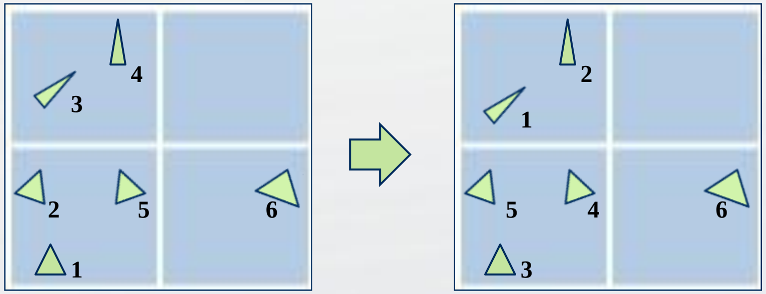
- **Meshes often show such coherence**



Levy et al

HLBVH: primitive sorting (part 2)

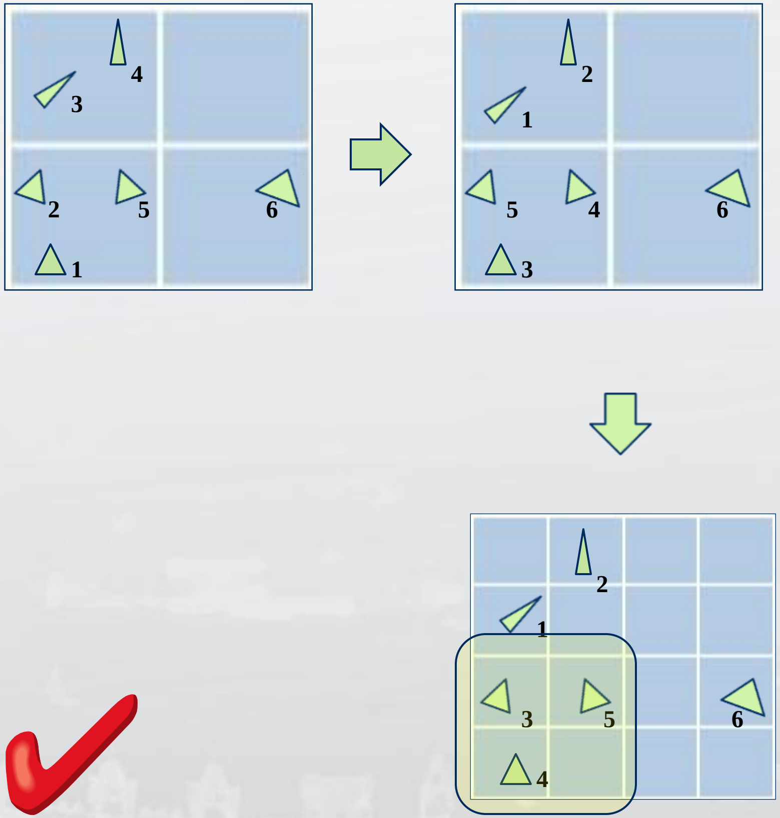
- **Prims are now sorted in coarse voxels**



HLBVH: primitive sorting (part 2)

- **Prims are now sorted in coarse voxels**

- **Sort within each voxel using intra-cta (shared-mem) sort**



HLBVH: primitive sorting (results)

- By CSD we have substantially reduced **BW** taking advantage of **spatial coherence**

- And if we reuse the final ordering across frames, we can take advantage of **temporal coherence** too

HLBVH: hierarchy emission

- This is all good, but we are still left with hierarchy emission, which is the hard part:

hierarchy emission		prim sorting
2 * Ω (N*30) sorts	vs	1 * O(N) sort

in LBVH

HLBVH: hierarchy emission

- **Input: array of sorted prims**

- **Output: array of nodes forming a tree**

HLBVH: hierarchy emission

- **Input: array of sorted prims (sequence of Morton codes)**

0	1	2	3	4	...	N-4	N-3	N-2	N-1
0	0	0	0	0	...	1	1	1	1
0	0	0	0	0	...	0	0	0	1
0	0	0	1	0	...	0	0	1	1
0	0	0	1	1	...	1	1	1	0
1	1	1	0	0	...	1	1	1	0
0	0	1	0	0	...	0	0	0	0
1	1	0	0	0	...	0	1	0	0
0	1	0	1	1	...	0	0	0	0

HLBVH: hierarchy emission

- **Input: array of sorted prims (sequence of Morton codes)**
- **Output: sequence of nested *segments***

[0	1	2	3	4	...]	[N-4	N-3	N-2	N-1]
0	0	0	0	0	...	1	1	1	1
0	0	0	0	0	...	0	0	0	1
0	0	0	1	0	...	0	0	1	1
0	0	0	1	1	...	1	1	1	0
1	1	1	0	0	...	1	1	1	0
0	0	1	0	0	...	0	0	0	0
1	1	0	0	0	...	0	1	0	0
0	1	0	1	1	...	0	0	0	0

HLBVH: hierarchy emission

- **Input: array of sorted prims (sequence of Morton codes)**
- **Output: sequence of nested *segments***

[0	1	2]	[3	4	...]	[N-4	N-3]	[N-2	N-1]
0	0	0	0	0	...	1	1	1	1
0	0	0	0	0	...	0	0	0	1
0	0	0	1	0	...	0	0	1	1
0	0	0	1	1	...	1	1	1	0
1	1	1	0	0	...	1	1	1	0
0	0	1	0	0	...	0	0	0	0
1	1	0	0	0	...	0	1	0	0
0	1	0	1	1	...	0	0	0	0

HLBVH: hierarchy emission

- **Partial Breadth First Traversal**

- **Consider p-bit planes at a time**

0	1	2	3	4	…	N-4	N-3	N-2	N-1
0	0	0	0	0	…	1	1	1	1
0	0	0	0	0	…	0	0	0	1
0	0	0	1	0	…	0	0	1	1
0	0	0	1	1	…	1	1	1	0
1	1	1	0	0	…	1	1	1	0
0	0	1	0	0	…	0	0	0	0
1	1	0	0	0	…	0	1	0	0
0	1	0	1	1	…	0	0	0	0

HLBVH: hierarchy emission

- **Partial Breadth First Traversal**

- **Consider p-bit planes at a time**

0	1	2	3	4	…	N-4	N-3	N-2	N-1
0	0	0	0	0	…	1	1	1	1
0	0	0	0	0	…	0	0	0	1
0	0	0	1	0	…	0	0	1	1
0	0	0	1	1	…	1	1	1	0
1	1	1	0	0	…	1	1	1	0
0	0	1	0	0	…	0	0	0	0
1	1	0	0	0	…	0	1	0	0
0	1	0	1	1	…	0	0	0	0

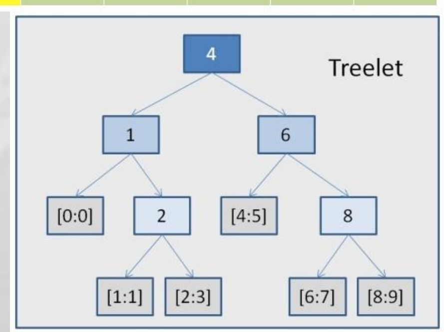
HLBVH: hierarchy emission

- **Partial Breadth First Traversal**

- **Consider p-bit planes at a time**

0	1	2	3	4	...	N-4	N-3	N-2	N-1
0	0	0	0	0	...	1	1	1	1
0	0	0	0	0	...	0	0	0	1
0	0	0	1	0	...	0	0	1	1
0	0	0	1	1	...	1	1	1	0
1	1	1	0	0	...	1	1	1	0
0	0	1	0	0	...	0	0	0	0
1	1	0	0	0	...	0	1	0	0
0	1	0	1	1	...	0	0	0	0

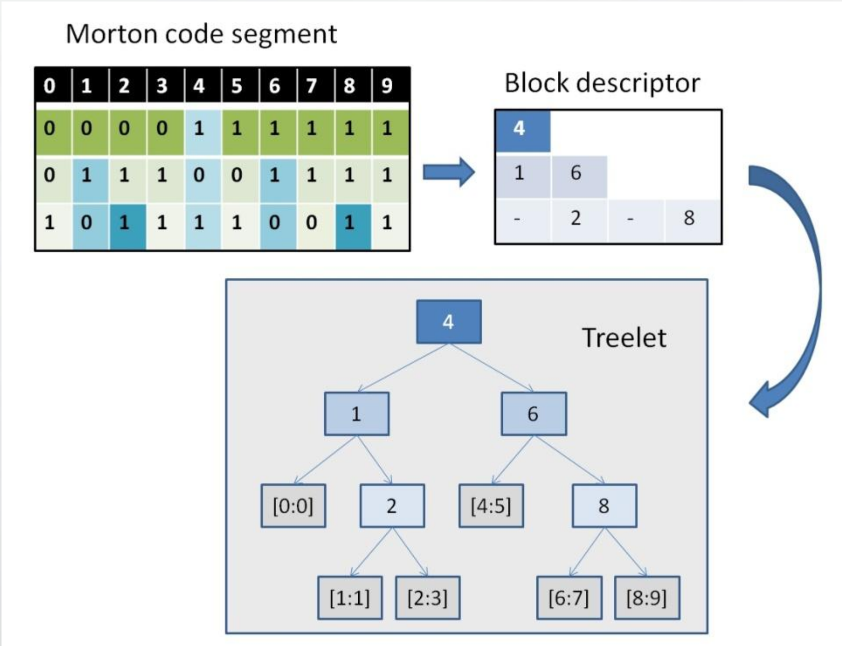
- **For each segment, emit a treelet**



HLBVH: hierarchy emission

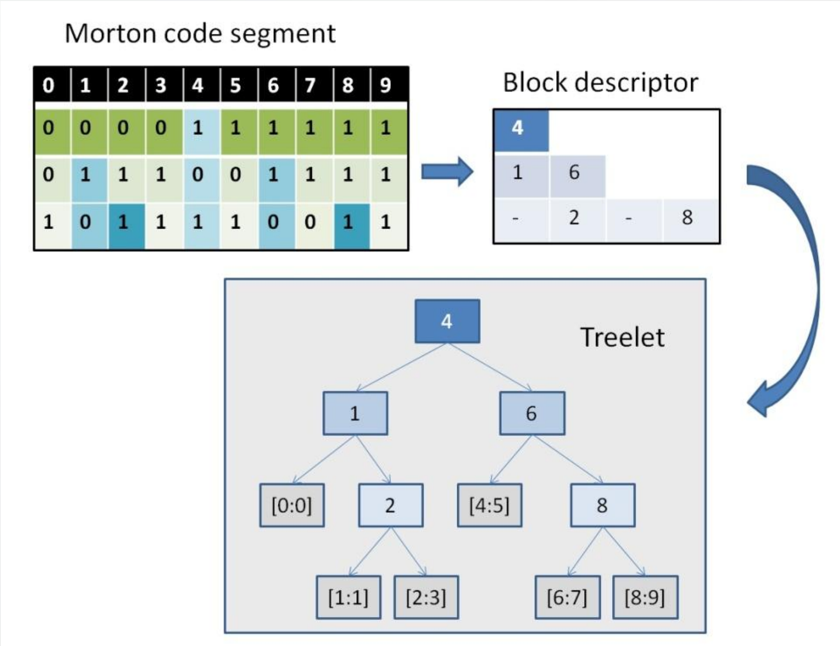
- **Partial Breadth First Traversal**

- **Details in the paper**



HLBVH: hierarchy emission

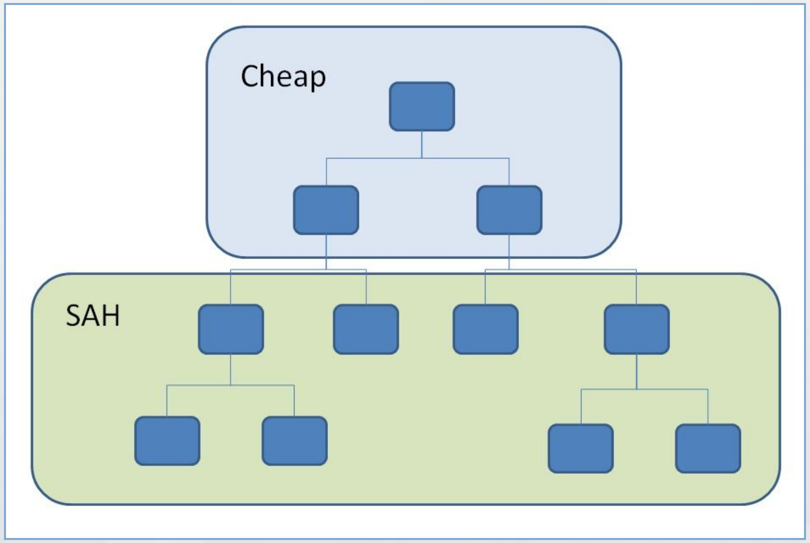
- **Partial Breadth First Traversal**



- **Details in the paper**

HLBVH: SAH Hybrid

- **Lauterbach and Wald suggested to perform SAH at the bottom of the tree**



HLBVH: SAH Hybrid

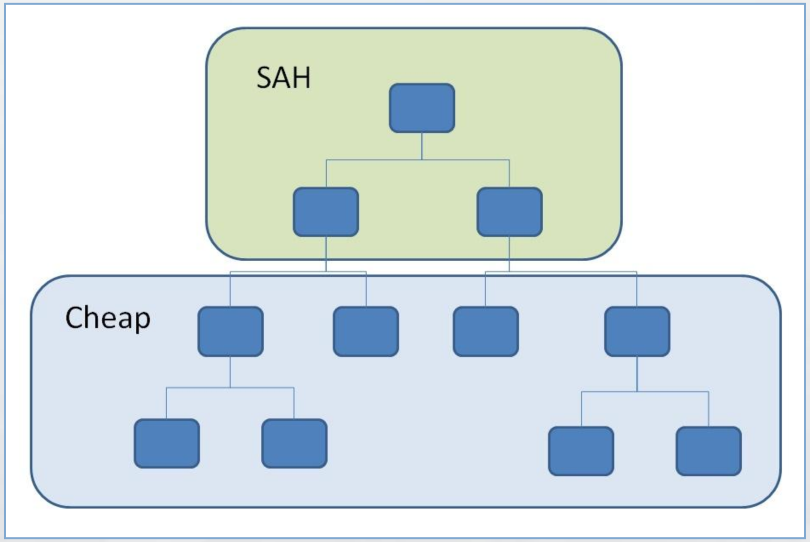
- **Lauterbach and Wald suggested to perform SAH at the bottom of the tree**

- **But with CSD we can do better!**

  Our coarse clusters can be used

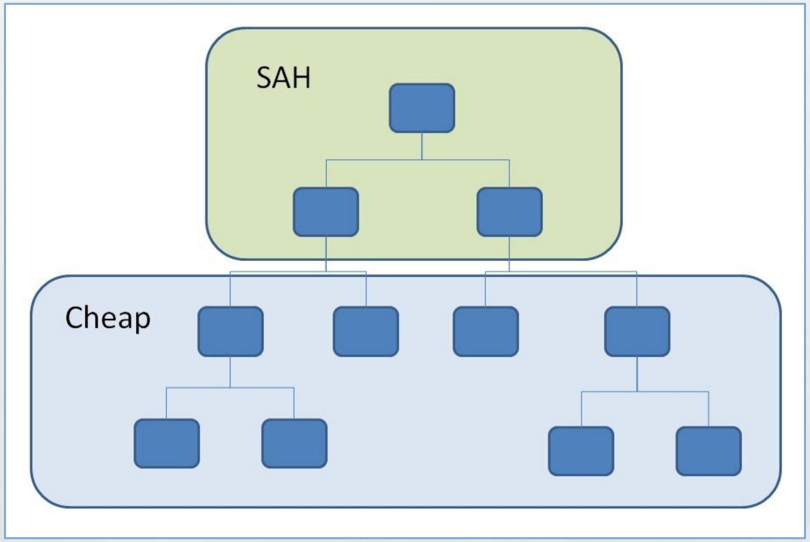
  to build a SAH-based top-level

  tree



HLBVH: SAH Hybrid

- **Lauterbach and Wald suggested to perform SAH at the bottom of the tree**



- **But with CSD we can do better!**

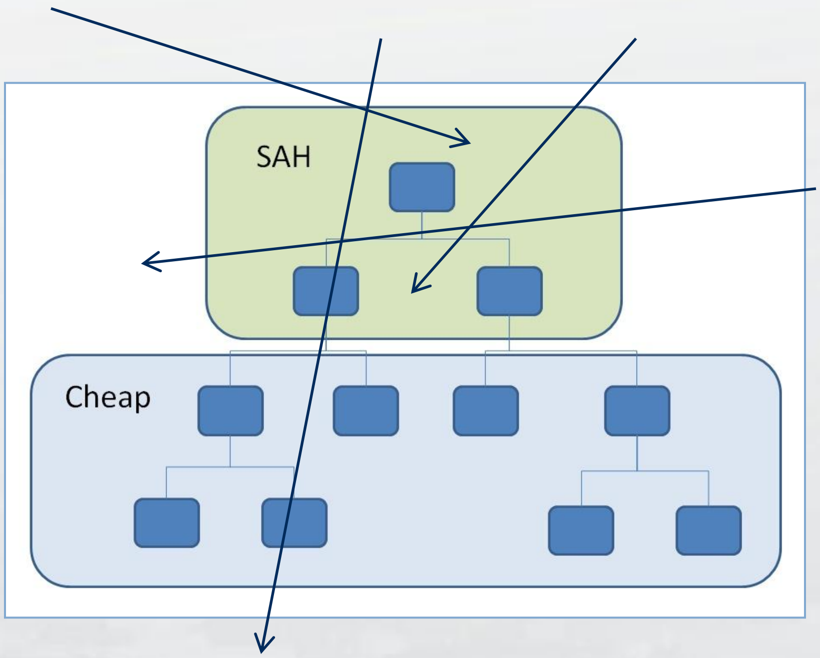
  Our coarse clusters can be used to build a SAH-based top-level tree

- **As the clusters are few, the overhead is low**

HLBVH: SAH Hybrid

- **Not only this is faster...**

- **It´s also better because the top-level tree is what matters mostly**

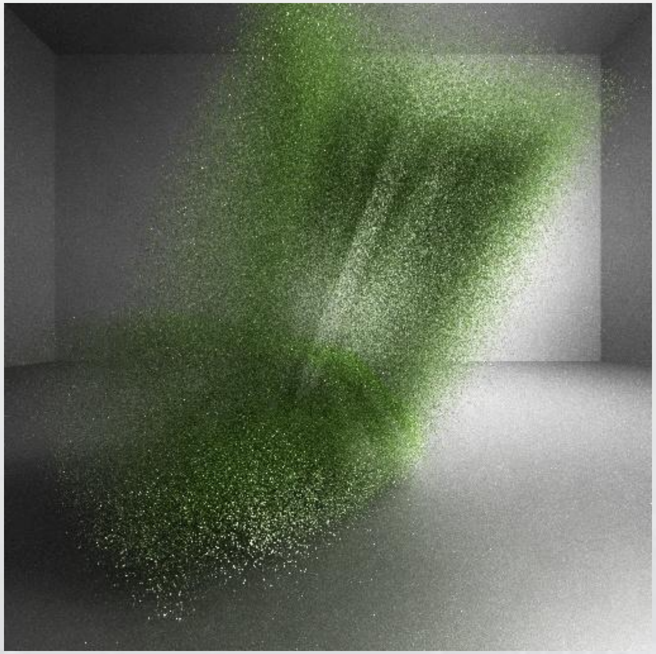


HLBVH: results

- We reduced BW by >10x
- We exploit spatial and temporal coherence
- Support fully dynamic geometry, from deformations to chaotic fracturing
- Low-overhead SAH hybrid

HLBVH: results

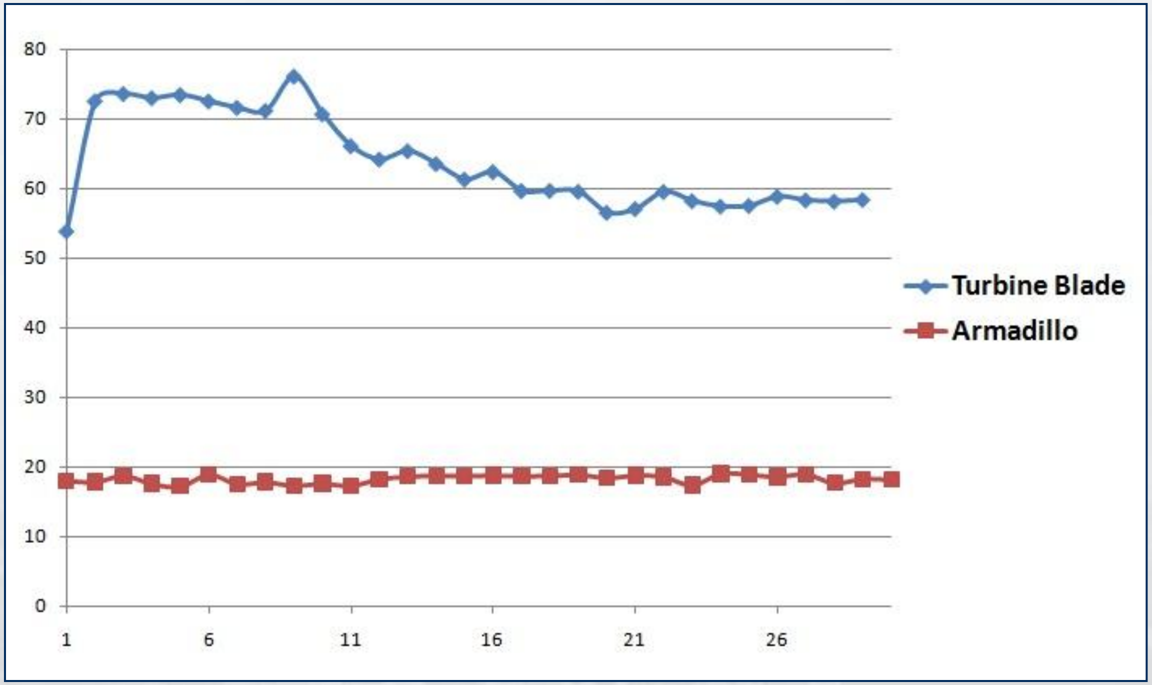
- **1M fully dynamic tris => ~35ms**



HLBVH: results

- **2M incoherent**

- **350k coherent**



HLBVH: code

- **Cleanly coded using Thrust**

- **Will be available at:**

  http://code.google.com/p/hlbvh/

```
hierarchy_emission(codes, N_prims, n_bits)
int segment_heads[]
int head_to_node[ N_prims ] = {-1}
head_to_node[0] = segment_heads[0] = 0

for (level = 0; level < n_bits; level += p)
  // compute segment ids
  segment_id[i] = scan (head_to_node[i] ≠ −1)

  // get the number of segments
  int N_segments = segment_id[N_prims-1]

   int P = (1 << p) - 1

   // compute block descriptors
   int block_splits[ N_segments * P ] = {-1}
   foreach i in [0,N_prims)
     emit_block_splits(
       i, [in] primitive index to process
       codes, [in] primitive Morton codes
       [level, level + p), [in] bit planes to process
       segment_id, [in] segment ids
       head_to_node, [in] head to node map
       segment_heads, [in] segment heads
       block_splits ) [out] block descriptors

   // compute the block offsets summing
   // the number of splits in each block
   int block_offsets[ N_segments + 1 ]
   block_offsets[s] = ex_scan (count_splits(s))
   int N_splits = block_offsets[N_segments]

   // emit treelets and update
   // segment_heads and head_to_node
   foreach segment in [0,N_segments)
     emit_treelets(
       segment, [in] block to process
       block_splits, [in] block descriptors
       block_offsets, [in] block offsets
       segment_id, [in] segment ids
       head_to_node, [in/out] head to node map
       segment_heads ) [in/out] segment heads

   node_count += N_splits * 2
```

Figure 4: *Pseudocode for our hierarchy emission loop.*

Thank You!