Fast BVH Construction on GPUs

Published in EUROGRAGHICS, (2009)
C. Lauterbach, M. Garland, S. Sengupta, D. Luebke, D. Manocha
University of North Carolina at Chapel Hill
NVIDIA
University of California Davis

Presented by
Soojin Han, Moonhwan Jeong

Granphics and Media Lab.
Seoul National University
Contents

Part I
1. Introduction
2. Background
3. LBVH hierarchy construction

Part II
4. SAH hierarchy construction
5. Results and Analysis
6. Conclusion and Future work
Contents

Part I

1. Introduction
2. Background
3. LBVH hierarchy construction

Part II

4. SAH hierarchy construction
5. Results and Analysis
6. Conclusion and Future work
Fast BVH Construction on GPUs

• (동영상)
Fast BVH Construction on GPUs

• What is Bounding Volume Hierarchy (BVH)?
  – A tree structure on a set of geometric objects
  – Objects are wrapped in bounding volumes
Fast **BVH Construction** on GPUs

- What is Bounding Volume Hierarchy (BVH)?
  - A tree structure on a set of geometric objects
  - Objects are wrapped in bounding volumes
Fast BVH Construction on GPUs

• What is Bounding Volume Hierarchy (BVH)?
  – A tree structure on a set of geometric objects
  – Objects are wrapped in bounding volumes
Two BVH Construction Techniques

- **LBVH construction:**
  - Linear ordering derived from spatial Morton codes

- **SAH construction:**
  - Top-down approach optimized for fast ray tracing

- **Hybrid two algorithms into one**
  - Significant benefit for both construction and traversal cost
Contents

Part I

1. Introduction
2. Background
3. LBVH hierarchy construction

Part II

4. SAH hierarchy construction
5. Results and Analysis
6. Conclusion and Future work
Why we need BVH?
BVH Construction: Top-down approach

- Generate bounding volume
BVH Construction

- Find longest axis
BVH Construction

- Find split plane, and sort in longest axis direction
BVH Construction

• Make parent node
BVH Construction

- Repeat
BVH Construction Issues

• Top-down, or bottom-up?
• How to insert a node dynamically?
BVH Traversal
BVH Traversal
BVH Traversal
Parallel computing model

- Design principles:
  - Decomposing work into chunks suitable to each thread block
  - Exploiting the on-chip memory given to each block
  - Exposing enough fine-grained parallelism

- Frequent CPU <-> GPU memory copying introduce expensive latency
Contents

Part I

1. Introduction
2. Background
3. LBVH hierarchy construction

Part II

4. SAH hierarchy construction
5. Results and Analysis
6. Conclusion and Future work
LBVH hierarchy construction

- Tree construction using Morton Codes
- Data-parallel LBVH Construction
Tree Construction Using Morton Codes

- Morton Codes (z-order curve)
  - Improving cache coherence
  - Thread block decomposing
  - Equivalent to Most-significant-bit radix-2 sort
  - Time complexity $O(n)$
Tree Construction Using Morton Codes

- Example 2-D Morton code
How can we determine interval lines?

- Levels of tree can be determined by Morton codes
  - 01100000
  - 01100100
  \[ h = 6 \]

- Split list: split must exist at each level from \( h \) to \( 3k \)
- Resorting this split list by level (\( O(n) \))
- A list of all the splits that should occur on this level
Data-Parallel LBVH Construction

- Asymptotic work complexity
  - Sorting primitives by Morton code
  - Sorting split list by level
  - Both can be implemented using a radix sort algorithm: $O(n)$

- Asymptotic depth complexity
  - Same as parallel scan: $O(\log n)$

- Efficient algorithm
  - Asymptotically same amount of work as the corresponding sequential sort and has a logarithmic depth

- But, not optimized for performance in ray tracing
Part II
Contents

Part I

1. Introduction
2. Background
3. LBVH hierarchy construction

Part II

4. SAH hierarchy construction
5. Results and Analysis
6. Conclusion and Future work
What is SAH?

Surface Area Heuristic

- For building hierarchies
- To be a good indicator of expected ray intersection
**What is SAH?**

*Intersection probability function*

- A *cost function* that allows to evaluate all possible split positions at a node and then pick the one with the *lowest cost*

\[
P_L = \frac{SA(V_L)}{SA(V)}, \quad P_R = \frac{SA(V_R)}{SA(V)}
\]

*SA(X)*: Surface area of volume X

*V*: Volume of parent node.

*V_L, V_R*: Volume of left, right children node

*P_L, P_R*: Intersection probability between ray and children node
What is SAH?

Intersection cost

- a cost function that allows to evaluate all possible split positions at a node and then pick the one with the lowest cost

\[ C_N = C_I \times N \]

\( C_N \) : Intersection cost of leaf
\( C_I \) : average Intersection cost between ray and triangle
\( N \) : Number of triangle in the leaf node

\[ C(p) = C_T + C_I \times (P_L N_L + P_R N_R) \]

\( C_T \) : average search cost
What is SAH?

Split position

SAH probability to position.

Split position : X=2.3107

SAH expectation cost graph.
GPU SAH construction

Parallel algorithm

- sufficiently parallel so as to exploit the computational power of the processors

Top-down construction algorithm,

a) process multiple splits in parallel
b) parallelize an actual split operation
**Algorithm for parallelizing**

1. result in either 0, 1 or 2
2. reads in its split item
3. writes out new splits
4. null split to the output queue at positions 2i,2i+1
5. The output work queue may have several null splits that need to be eliminated
Data-Parallel SAH split

Spatial partitioning

1. Determine the best split position by evaluating the SAH.
2. Reorder the primitives (or the indices to the primitives) such that the order in the global list corresponds to the new split.
**Data-Parallel SAH split**

**Method in this paper**

1. Determine the best split position by evaluating the SAH
2. Reorder the primitives (or the indices to the primitives) such that the order in the global list corresponds to the new split

be referenced on one side of the split (commonly determined by the location of the centroid of the primitive),

the reordering can be performed in-place and no additional memory is needed.
Data-Parallel SAH split (Reduction)

What is Reduction?

A class of operations involves:
- A ordered set $S=\{a_0, a_1, a_2, \ldots, a_{n-1}\}$ of n numbers
- A binary associative operator

Examples of reduction operations:
- Sum = $\text{Reduce}(+, S) = a_0 + a_1 + a_2 + \ldots + a_{n-1}$
- Product = $\text{Reduce}(\times, S) = a_0 \times a_1 \times a_2 \times \ldots \times a_{n-1}$
- Min = $\text{Reduce}(\text{min}, S) = \min(a_0, a_1, a_2, \ldots, a_{n-1})$

The output is a single number.
Require $O(N)$ time to compute on a sequential computer.
Data-Parallel SAH split

What is Reduction?

- Technique for performing reduction on parallel computers
- Compute the min of 2 numbers in each step
- Reduce the numbers in the set by half
- Require $\log_2 N$ steps on a $\text{N}$-processor computer
Data-Parallel SAH split
Prefix SUM

- Given a list of n numbers, compute the partial sums using only numbers on the left sides
- Input: a0, a1, a2, ... , an-1
- Output: a0, a0+a1, a0+a1+a2, ..., a0+a1+a2+...+an-1
- Require O(N) on a sequential compute
Data-Parallel SAH split
Prefix SUM( radix sort )

1. Set a 1 for all false sort keys (b = 0) and a 0 for all true sort keys.

2. We then scan this buffer.

3. The scan's output total False.

4. Now we compute the destination.

5. We scatter the original sort.
Small split optimizations

Construction timings per level

- Can efficiently use more samples due to high data parallelism and the SAH quality

![Graphs showing construction timings per level](image)

Left: normal construction algorithm. Right: construction with small split kernel as described in the text.

- The initial splits at the top levels of the hierarchy are slow due to lack of processor parallelism
- Large numbers of very small splits at the end
- Improve on the performance at the lower levels of the hierarchy
  → Using a different split kernel for all splits with sizes below a specified threshold and modify the compaction kernel
**Hybrid GPU construction algorithm**

**Combine LBVH with SAH**

- Combine both algorithms described in this paper into a hybrid builder.

- The main bottleneck in the SAH construction algorithm is lack of parallelism in the initial splits.

- The LBVH algorithm builds a very shallow hierarchy with large numbers of primitives at the leafs.

- Used the LBVH algorithm for performing the first 6 levels of splits before switching to SAH construction.

- Combines the speed of the LBVH algorithm with the ray tracing performance optimizations in the SAH algorithm.
Contents

Part I

1. Introduction
2. Background
3. LBVH hierarchy construction

Part II

4. SAH hierarchy construction
5. Results and Analysis
6. Conclusion and Future work
Results

Running system & benchmark

Intel Xeon X5355 system at 2.66GHz
Microsoft Windows XP with a NVIDIA GeForce 280 GTX graphics card with 1 GB
using the NVIDIA CUDA programming

Sibenik cathedral (80K tris), Bunny/Dragon animation (252K tris), Conference room (284K tris), Soda Hall (2M tris).
Results

Compare rendering speed

<table>
<thead>
<tr>
<th>Model</th>
<th>Tris</th>
<th>CPU SAH</th>
<th>GPU SAH</th>
<th>LBVH</th>
<th>Hybrid</th>
<th>Parallel SAH [Wal07]</th>
<th>Full SAH [Wal07]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flamenco</td>
<td>49K</td>
<td>144ms 30fps/99%</td>
<td>85ms 30.3fps/100%</td>
<td>9.8ms 12.4fps/41%</td>
<td>17ms 29.9fps/99%</td>
<td>n/a</td>
<td>n/a 100%</td>
</tr>
<tr>
<td>Sibenik</td>
<td>82K</td>
<td>231ms 21.4fps/97%</td>
<td>144ms 21.7fps/98%</td>
<td>10ms 3.5fps/16%</td>
<td>30ms 21.4fps/97%</td>
<td>n/a</td>
<td>n/a 100%</td>
</tr>
<tr>
<td>Fairy</td>
<td>174K</td>
<td>661ms 11.5fps/98%</td>
<td>488ms 21.7fps/100%</td>
<td>10.3ms 1.8fps/15%</td>
<td>124ms 11.6fps/99%</td>
<td>21ms 93%</td>
<td>860ms 100%</td>
</tr>
<tr>
<td>Bunny/Dragon</td>
<td>252K</td>
<td>842ms 7.8fps/100%</td>
<td>403ms 7.75fps/100%</td>
<td>17ms 7.3fps/94%</td>
<td>66ms 7.6fps/98%</td>
<td>20ms 98%</td>
<td>1160ms 100%</td>
</tr>
<tr>
<td>Conference</td>
<td>284K</td>
<td>819ms 24.4fps/91%</td>
<td>477ms 24.5fps/91%</td>
<td>19ms 6.7fps/25%</td>
<td>105ms 22.9fps/85%</td>
<td>26ms 86%</td>
<td>1320ms 100%</td>
</tr>
<tr>
<td>Soda Hall</td>
<td>1.5M</td>
<td>6176ms 20.8fps/98%</td>
<td>2390ms 21.4fps/101%</td>
<td>66ms 3fps/14%</td>
<td>445ms 20.7fps/98%</td>
<td>n/a</td>
<td>n/a 100%</td>
</tr>
</tbody>
</table>


- Approximate SAH & hybrid build are very close to the reference results. (Full SAH)
- The efficiency of the LBVH construction is highly scene dependent.
Analysis

Current bottlenecks

- Hybrid and SAH algorithms are not limited by memory bandwidth
- LBVH algorithm shows the highest dependence on memory speed
Analysis

Time spent in the construction

- SAH split : the time for finding the optimal subdivision
- Reorder : reordering the triangles according to the split
- Compaction : compacting and maintaining queues between splits
- LBVH : the initial split time in the hybrid construction (for 6 levels).
- Rest : reading in and writing back BVH node information,

→ The hybrid build is more balanced
Contents

Part I

1. Introduction
2. Background
3. LBVH hierarchy construction

Part II

4. SAH hierarchy construction
5. Results and Analysis
6. Conclusion and Future work
Conclusion

- Algorithms for performing fast construction of optimized object hierarchies on current GPUs

- SAH techniques can be implemented while exploiting both processor as well as data parallelism with a reorganization of the construction process.

- Introduced a novel construction scheme based on fast radix sort

- Introduced the Hybrid GPU construction algorithm
Questions?
Thank You!