Fast Parallel Construction of High-Quality Bounding Volume Hierarchies

By Tero Karras and Timo Aila (NVIDIA, 2013)
Intro to Bounding volume hierarchies (BVH)
Building a BVH

Two important characteristics

- Build time, usually measured in seconds per build
- Efficiency, usually measured in rays per second
Surface area heuristic (SAH)

\[ C_i \sum_{n \in I} \frac{A(n)}{A(\text{root})} + C_l \sum_{l \in L} \frac{A(l)}{A(\text{root})} + C_t \sum_{l \in L} \frac{A(l)}{A(\text{root})} N(l), \] (1)

- SAH cost = expected cost of tracing a non-terminating ray through the scene
- \( C_i = 1.2, C_l = 0 \) and \( C_t = 1 \)
Other methods for BVH construction

Realtime:
- LBVH
- HLBVH

Offline
- SweepSAH
- SBVH
Linear BVH (LBVH)

- Sorts triangles along a space filling curve
- Partition them recursively so that each node represents a linear range of triangles

Very fast construction!

Not very high performance! (Around 50% of SBVH)

HLBVH: Better performance, but still not great
SBVH

- Top down approach
- Greedily selects splitting planes based on SAH cost
- Creates AABBs for each side of the plane
- Duplicates triangles that fall on both sides of the plane
- Incorporates triangle splitting

Slow build time

Best known performance!
The papers algorithm

- Builds the BVH in real time
- High performance (within 10% of SBVH)
Performance averaged over 20 test scenes
How did they do it?

- Treelet optimization
- Triangle splitting
Initial BVH construction

- Uses a method developed by Karras in 2012
- Assigns a 60-bit Morton code to each leaf node
- Creates a binary radix tree
Optimization

- Treelet optimization
- A immediate subtree of a node
- Find the best configuration of that subtree
Treelet formation

- Select root-nodes in a bottom up algorithm that avoids overlapping treelets
- Select the nodes of the root with the largest surface area
Naive implementation

- Recursively try every possible subtree with the given nodes
- 7 leaf nodes results in 1.15 million recursive function calls
Dynamic programming

- Represent a subset of leaf-nodes as a bitstring
  - 1000000 -> Just leaf node 1
  - 1100000 -> Internal node containing node 1 and node 2
- Calculate SAH cost for all subsets with 1 leaf-node, then 2, up to 7
- Final result is stored in $c_{opt}[1111111]$

```plaintext
1: // Calculate surface area for each subset
2: for $\bar{s} = 1$ to $2^n - 1$ do
3:     $a[\bar{s}] \leftarrow$ AREA(UNIONOfAABBs($L, \bar{s}$))
4: end for
5: // Initialize costs of individual leaves
6: for $i = 0$ to $n - 1$ do
7:     $c_{opt}[2^i] \leftarrow C(L_i)$
8: end for
9: // Optimize every subset of leaves
10: for $k = 2$ to $n$ do
11:     for each $\bar{s} \in [1, 2^n - 1]$ with $k$ set bits do
12:         // Try each way of partitioning the leaves
13:         $(c_{\bar{s}}, \bar{p}_{\bar{s}}) \leftarrow (\infty, 0)$
14:         for each $\bar{p} \in \{\text{partitionings of } \bar{s}\}$ do
15:             $c \leftarrow c_{opt}[\bar{p}] + c_{opt}[\bar{s} \text{ XOR } \bar{p}]$ // $S \setminus P$
16:             if $c < c_{\bar{s}}$ then $(c_{\bar{s}}, \bar{p}_{\bar{s}}) \leftarrow (c, \bar{p})$
17:         end for
18:     // Calculate final SAH cost (Equation 2)
19:     $t \leftarrow$ TOTALNUMTRIANGLES($L, \bar{s}$)
20:     $c_{opt}[\bar{s}] \leftarrow \min \left( (C_t \cdot a[\bar{s}] + c_{\bar{s}}), (C_t \cdot a[\bar{s}] \cdot t) \right)$
21:     $\bar{p}_{opt}[\bar{s}] \leftarrow \bar{p}_{\bar{s}}$
22: end for
23: end for
```
Post processing

- Every leaf node contains one triangle
- Collapse leaf nodes into nodes with multiple triangles
- Make sure there are no overlap between threads

\[
C(n) = \min \left\{ \begin{array}{ll}
C_i A(n) + C(n_l) + C(n_r) & (n \in I) \\
C_t A(n) N(n) & (n \in L)
\end{array} \right. 
\] (2)
Triangle splitting

Why?

- Reduce surface area
- Reduce overlap between subtrees

What is new?

- Experimentally found priority formula
- Cap splitting at certain percentage (usually 30%)
Results

M\text{\textperthousand}/s relative to maximum achievable ray tracing performance of SweepSAH
Results (No splitting)

<table>
<thead>
<tr>
<th>Builder</th>
<th>Perf</th>
<th>SAH</th>
<th>Build</th>
</tr>
</thead>
<tbody>
<tr>
<td>SweepSAH</td>
<td>100.0</td>
<td>100.0</td>
<td>100.00</td>
</tr>
<tr>
<td>Our</td>
<td>96.0</td>
<td>94.4</td>
<td>1.03</td>
</tr>
<tr>
<td>LBVH</td>
<td>69.4</td>
<td>131.5</td>
<td>0.28</td>
</tr>
<tr>
<td>HLBVH</td>
<td>67.4</td>
<td>129.3</td>
<td>0.57</td>
</tr>
<tr>
<td>GridSAH</td>
<td>68.1</td>
<td>129.1</td>
<td>0.70</td>
</tr>
<tr>
<td>Kensler</td>
<td>91.5</td>
<td>99.6</td>
<td>552.26</td>
</tr>
<tr>
<td>Bittner</td>
<td>103.3</td>
<td>87.7</td>
<td>1083.24</td>
</tr>
<tr>
<td>Builder</td>
<td>AVERAGE OF 20 SCENES relative to SBVH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------------------</td>
<td>--------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Perf</td>
<td>SAH</td>
<td>Build</td>
</tr>
<tr>
<td>SBVH</td>
<td>100.0</td>
<td>100.0</td>
<td>100.00</td>
</tr>
<tr>
<td>Our (no splits)</td>
<td>100.0</td>
<td>100.0</td>
<td>100.00</td>
</tr>
<tr>
<td>Our (10% splits)</td>
<td>76.5</td>
<td>111.5</td>
<td>0.14</td>
</tr>
<tr>
<td>Our (30% splits)</td>
<td>76.5</td>
<td>111.5</td>
<td>0.14</td>
</tr>
<tr>
<td>Our (50% splits)</td>
<td>87.2</td>
<td>108.8</td>
<td>0.15</td>
</tr>
<tr>
<td>Our (match SBVH)</td>
<td>91.0</td>
<td>108.0</td>
<td>0.17</td>
</tr>
<tr>
<td>ESC (match SBVH)</td>
<td>92.5</td>
<td>108.3</td>
<td>0.20</td>
</tr>
<tr>
<td>EVH (match SBVH)</td>
<td>90.7</td>
<td>107.4</td>
<td>0.16</td>
</tr>
</tbody>
</table>