EFFICIENT BVH CONSTRUCTION VIA APPROXIMATE AGGLOMERATIVE CLUSTERING

Yan Gu, Yong He, Kayvon Fatahalian, Guy Blelloch
Carnegie Mellon University
BVH CONSTRUCTION GOALS

- High quality: produce BVHs of comparable (or better) quality to full-sweep SAH algorithms.

- High performance: faster construction than widely used SAH-based algorithms that use binning.

OUR APPROACH

- An agglomerative clustering (bottom-up) based construction algorithm.
  - Motivated by [Walter et al. 2008].
Hierarchical clustering example

Source data points:
**Hierarchical Clustering Example**

Source data points:

\[ d(i, j) = \text{distance from cluster } i \text{ to cluster } j \]
HIERARCHICAL CLUSTERING EXAMPLE

Source data points:
Hierarchical Clustering Example

Source data points:

Resulting cluster hierarchy:
Hierarchical clustering example

Source data points:

Resulting cluster hierarchy:
Hierarchical clustering example

Source data points:

Resulting cluster hierarchy:
Hierarchical Clustering Example

Source data points:

Resulting cluster hierarchy:
HIERARCHICAL CLUSTERING IS A GENERAL TECHNIQUE FOR ORGANIZING DATA.

<table>
<thead>
<tr>
<th>Domain</th>
<th>Clustered primitives</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linguistic</td>
<td>Languages</td>
</tr>
<tr>
<td>Image retrieval</td>
<td>Images</td>
</tr>
<tr>
<td>Anthropology</td>
<td>Surnames / races</td>
</tr>
<tr>
<td>Biology</td>
<td>Genes / species</td>
</tr>
<tr>
<td>Social network</td>
<td>People / behaviors</td>
</tr>
</tbody>
</table>
BVH BUILD USING AGGLOMERATIVE CLUSTERING
[WALTER ET AL. 2008]

- Elements to cluster = scene primitives
- Distance = surface area of aggregate bounding box
BVH BUILD USING AGGLOMERATIVE CLUSTERING
[WALTER ET AL. 2008]

- Compute the nearest neighbor to each primitive.
BVH BUILD USING AGGLOMERATIVE CLUSTERING
[WALTER ET AL. 2008]

- Find the “closest” pair of primitives and combine them into a cluster.
BVH BUILD USING AGGLOMERATIVE CLUSTERING [WALTER ET AL. 2008]

- Update nearest neighbor links.
BVH BUILD USING AGGLOMERATIVE CLUSTERING
[WALTER ET AL. 2008]

- Repeat: combine closest remaining clusters.
BVH BUILD USING AGGLOMERATIVE CLUSTERING [WALTER ET AL. 2008]

- Repeat: combine closest remaining clusters.
BVH BUILD USING AGGLOMERATIVE CLUSTERING [WALTER ET AL. 2008]

- Repeat: combine closest remaining clusters.
**BVH BUILD USING AGGLOMERATIVE CLUSTERING**

[WALTER ET AL. 2008]

- Repeat: combine closest remaining clusters.
BVH BUILD USING AGGLOMERATIVE CLUSTERING [WALTER ET AL. 2008]

- Repeat: combine closest remaining clusters.
BVH BUILD USING AGGLOMERATIVE CLUSTERING [WALTER ET AL. 2008]

- Continue until one cluster remains (BVH root).
**BVH BUILD USING AGGLOMERATIVE CLUSTERING**

[WALTER ET AL. 2008]

- Good: often higher quality BVH than sweep builds

- Bad: lower performance than binned builds
  - KD-tree search/update in each clustering step.
  - Data-dependent parallel execution.
Most computation occurs at the lowest levels of the BVH of the construction process when the number of clusters is large (near leaves).

Top $\frac{h}{2}$ levels:
Number of nodes $\approx \sqrt{n}$

Bottom $\frac{h}{2}$ levels:
Number of nodes $\approx n$
CONTRIBUTION

Approximate Agglomerative Clustering (AAC)

- New algorithm for BVH construction that is work efficient, parallelizable, and produces high-quality trees.
OUR MAIN IDEA

- Restrict nearest neighbor search to a small subset of neighboring scene elements.
OUR MAIN IDEA

- Restrict nearest neighbor search to a small subset of neighboring scene elements.
OUR MAIN IDEA

- Restrict nearest neighbor search to a small subset of neighboring scene elements.
Phase 1: Primitive Partitioning ("Downward/Divide Phase")

Computation graph:

Primitive partitioning:

\[ \delta = 4 \]
PHASE 2: AGGLOMERATIVE CLUSTERING ("UPWARD PHASE")

Computation graph:

Primitive partitioning:
**Phase 2: Agglomerative Clustering ("Upward Phase")**

Computation graph:
Each node = combine input into $f(n)$ clusters

Primitive partitioning:
PHASE 2: AGGLOMERATIVE CLUSTERING ("UPWARD PHASE")

Computation graph:
Each node = combine input into $f(n)$ clusters

Primitive partitioning:
PHASE 2: AGGLOMERATIVE CLUSTERING (“UPWARD PHASE”)

Computation graph:
Each node = combine input into $f(n)$ clusters

Primitive partitioning:
PHASE 2: AGGLOMERATIVE CLUSTERING ("UPWARD PHASE")

Computation graph:
Each node = combine input into $f(n)$ clusters

Primitive partitioning:
**Phase 2: Agglomerative Clustering ("Upward Phase")**

**Computation graph:**
Each node = combine input into $f(n)$ clusters

**Primitive partitioning:**

[Diagram showing computation graph and primitive partitioning]
PHASE 2: AGGLOMERATIVE CLUSTERING ("UPWARD PHASE")

Computation graph:
Each node = combine input into $f(n)$ clusters

Primitive partitioning:
PHASE 2: AGGLOMERATIVE CLUSTERING ("UPWARD PHASE")

Computation graph:
Each node = combine input into $f(n)$ clusters

Primitive partitioning:
PHASE 2: AGGLOMERATIVE CLUSTERING ("UPWARD PHASE")

Computation graph:

Primitive partitioning:
PHASE 2: AGGLOMERATIVE CLUSTERING ("UPWARD PHASE")

Computation graph:

Primitive partitioning:
AAC IS AN APPROXIMATION TO THE TRUE AGGLOMERATIVE CLUSTERING SOLUTION.

Computation graph:

Primitive partitioning:
AAC is an approximation to the true agglomerative clustering solution.

Computation graph:

Primitive partitioning:
AAC HAS TWO PARAMETERS

- $\delta$: stopping criterion for stop partitioning (maximum of primitives in leaf regions).

- $f(n)$: function that determines the number of clusters to generate in each graph node ($n$ is the number of primitives in the corresponding region.)
DETERMINING HOW MUCH TO CLUSTER

- \( f(n) = 1 \): close to spatial bisection BVH.
**Determining how much to cluster**

- $f(n) = n$: all primitives pushed to top of computation graph, AAC solution is same as true agglomerative clustering.
We use $f(n) = cn^\alpha$, where $0 < \alpha < 0.5$. 
AAC has linear time complexity

- Downward phase is linear.

- Upward clustering phase:
  
  - Let $f(n) = cn^\alpha$, where $0 < \alpha < 0.5$.
  
  - Assumptions:
    - $\delta$ is a small constant.
    - Time complexity on each graph node is $O(n^2)$ [Olson 1995], where $n$ is the number of input primitives in this node.
COMPLEXITY ANALYSIS

Let $f(n) = cn^\alpha$, where $0 < \alpha < 0.5$.

Work done at leaves: \( \frac{N}{\delta} \) nodes, \( O(f(\delta)^2) = O(\delta^{2\alpha}) \) computation each.

\( O(N\delta^{2\alpha-1}) \) work total.
Let $f(n) = cn^\alpha$, where $0 < \alpha < 0.5$.

Let $C = N\delta^{2\alpha-1}$

Work done at leaves: $\frac{N}{\delta}$ nodes, $O(f(\delta)^2) = O(\delta^{2\alpha})$ computation each.

$O(C)$ work total.
LINEAR TIME COMPLEXITY

- Let $f(n) = cn^\alpha$, where $0 < \alpha < 0.5$.

Let $C = N\delta^{2\alpha-1}$

$\frac{N}{2\delta}$ nodes,

$O(f(2\delta)^2) = O((2\delta)^{2\alpha})$

computation each.

$\rightarrow O(N(2\delta)^{2\alpha-1})$ work total.

$\rightarrow O(C)$ work total.
**LINEAR TIME COMPLEXITY**

- Let $f(n) = cn^\alpha$, where $0 < \alpha < 0.5$.

Let $C = N\delta^{2\alpha-1}$

$r = 2^{2\alpha-1} < 1$

$O(C \cdot r)$ work total.

$O(C)$ work total.
**Linear Time Complexity**

- Let $f(n) = cn^\alpha$, where $0 < \alpha < 0.5$.

Geometrically decreasing.

Let

- $C = N\delta^{2\alpha-1}$
- $r = 2^{2\alpha-1} < 1$

$O(C \cdot r^2)$ work total.

$O(C \cdot r)$ work total.

$O(C)$ work total.
THEORY MEETS PRACTICE: WE OBSERVE LINEAR SCALING WITH SCENE SIZE

AAC-HQ BVH Construction Time (Single Core)

Scene Triangle Count

BVH Construction Time (ms)

San Miguel

Sponza/Fairy

Conference

Half-Life

Buddha
Parameters

- Trade off BVH quality and construction speed by changing $\delta$ and $f$ in the algorithm.

- We proposed 2 sets of parameters:
  - AAC-HQ (high quality): $\delta = 20$, $f(n) = \frac{\delta^{0.6}}{2} \cdot n^{0.4}$;
  - AAC-Fast: $\delta = 4$, $f(n) = \frac{\delta^{0.7}}{2} \cdot n^{0.3}$. 
IMPLEMENTATION DETAILS

- Parallelization:
  - Algorithm is divide-and-conquer, so very easy to parallelize.

- Key optimizations possible:
  - Reduce redundant computation of cluster distances;
  - Reducing data movement;
  - Sub-tree flatting for improved tree quality.
EVALUATION
**SETUP**

- We compared 5 CPU implementations

<table>
<thead>
<tr>
<th>CPU Implementation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAH</td>
<td>A standard top-down full-sweep SAH build [MacDonald and Booth 1990]</td>
</tr>
<tr>
<td>SAH-BIN</td>
<td>A top-down “binned” SAH build using at most 16 bins along the longest axis [Wald 2007]</td>
</tr>
<tr>
<td>Local-Ord</td>
<td>Locally-ordered agglomerative clustering [Walter et al. 2008]</td>
</tr>
<tr>
<td>AAC-HQ</td>
<td>AAC with high quality settings: $\delta = 20$, $f(n) = 3n^{0.4}$</td>
</tr>
<tr>
<td>AAC-Fast</td>
<td>AAC configured for performance: $\delta = 4$, $f(n) = 1.3n^{0.3}$</td>
</tr>
</tbody>
</table>
SCENES

Sponza  
Half-Life  
Conference  

Fairy  
San Miguel  
Buddha
**Tree cost comparison**

- Cost = number of traversal steps + intersection tests during ray tracing.

![Chart showing tree cost comparison for different scenes and methods](image-url)
AAC-HQ produces BVHs that have similar cost as those produced by true agglomerative clustering builds.
AAC-Fast produces BVHs with equal or lower cost than the **full sweep build** in all cases except Buddha.
AAC-Fast produces BVHs with equal or lower cost than the **full sweep build** in all cases except Buddha.
AAC IS ABLE TO MAKE PARTITIONS THAT ARE NOT DETERMINED BY PARTITION PLANES.
BVH CONSTRUCTION TIME (SINGLE CORE)

- AAC-HQ build times are five to six times lower than Local-Ord (while maintaining comparable BVH quality)

![Normalized BVH Build Time (Single Core)](image-url)
AAC-HQ build times are comparable to SAH-BIN
AAC-Fast build times up to four times faster than SAH-BIN
AAC PARALLEL EXECUTION SPEEDUP

AAC-HQ achieves nearly linear speedup out to 16 cores, and a 34× speedup on 40 cores.
## AAC 32-Core Speedup

AAC Build Execution Times (milliseconds) and Parallel Speedup

<table>
<thead>
<tr>
<th>Tri Count</th>
<th>AAC-HQ</th>
<th></th>
<th></th>
<th>AAC-Fast</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 core</td>
<td>32 cores</td>
<td></td>
<td>1 core</td>
<td>32 cores</td>
</tr>
<tr>
<td>Sponza</td>
<td>67 K</td>
<td>52</td>
<td>2</td>
<td>(24.0)</td>
<td>20</td>
</tr>
<tr>
<td>Fairy</td>
<td>174 K</td>
<td>117</td>
<td>5</td>
<td>(24.5)</td>
<td>44</td>
</tr>
<tr>
<td>Conference</td>
<td>283 K</td>
<td>225</td>
<td>10</td>
<td>(23.6)</td>
<td>70</td>
</tr>
<tr>
<td>Buddha</td>
<td>1.1 M</td>
<td>1,101</td>
<td>43</td>
<td>(25.8)</td>
<td>397</td>
</tr>
<tr>
<td>Half-Life</td>
<td>1.2 M</td>
<td>1,080</td>
<td>42</td>
<td>(25.7)</td>
<td>359</td>
</tr>
<tr>
<td>San Miguel</td>
<td>7.9 M</td>
<td>7,350</td>
<td>298</td>
<td>(24.6)</td>
<td>2,140</td>
</tr>
</tbody>
</table>
SUMMARY

- AAC algorithm: BVH construction via an approximation to agglomerative clustering of scene primitives
  - Comparable quality BVH to full sweep SAH build
  - Up to four-times faster than binned SAH build
  - Amenable to parallelism on many-core CPUs
SIMILARITY TO KARRAS13 (NEXT TALK)

- Fast initial organization of scene primitives via Morton codes
  - AAC: to define constraints on clustering
  - Karras13: to define initial BVH

- “Brute-force” optimization of local sub-structures
  - AAC: brute-force local clustering in each node
  - Karras13: brute-force enumeration of treelet structures
  - In both: more flexible partitions than defined by spatial partition plane

- AAC does not address triangle splitting
Looking forward

- Have not yet explored parallelization of AAC on GPUs

- Post-process BVH optimizations can be applied on a smaller set of clusters generated by AAC

- Clustering in low dimensional space has many other applications in computer graphics including:
  - Lighting (e.g., Light Cuts)
  - N-body simulation
  - Collision detection
Thank you

We acknowledge the support of:
The National Science Foundation (CCF-1018188)
Intel Labs Academic Research Office
NVIDIA corporation
BVHs produced by AAC methods realize greater benefit for shadow rays than diffuse bounce rays.

AAC-HQ BVH cost (normalized to full sweep SAH)
WHY AAC PERFORMS WORSE FOR BUDDHA.