Parallel Locally-Ordered Clustering for Bounding Volume Hierarchy Construction

Daniel Meister and Jiří Bittner

Department of Computer Graphics and Interaction
Faculty of Electrical Engineering
Czech Technical University in Prague
Motivation: Interactive Ray Tracing

Fast BVH construction for geometry that is not known a priori

- Dynamic geometry changes in every frame
- Scene is assembled on the fly

[Benthin et al. 2017]
Bounding Volume Hierarchy (BVH)

- Ray tracing, collision detection, visibility culling
- Rooted tree of arbitrary branching factor
  - References to geometric primitives in leaves
  - Bounding volumes in interior nodes

[Clark 1976]
BVH Construction Methods

Top-down
- Surface Area Heuristic [Hunt et al. 2007]
- Binning [Ize et al. 2007, Wald 2007]
- $k$-means clustering [Meister and Bittner 2016]

Bottom-up
- Agglomerative clustering [Walter et al. 2008]
- Approximate aggl. clustering [Gu et al. 2013]
BVH Construction Methods

Insertion
- Heuristic greedy search [Goldsmith and Salmon 1987]
- Online construction [Bittner et al. 2015]

Optimization
- Insertion-based optimization [Bittner 2013 et al.]
- Treelet restructuring [Karras and Aila 2013, Domingues and Pedrini 2015]
Surface Area Heuristic (SAH)

\[ c(N) = \begin{cases} 
  c_T + \frac{SA(N_L)}{SA(N)} c(N_L) + \frac{SA(N_R)}{SA(N)} c(N_R) & \text{if } N \text{ is interior node} \\
  c_I |N| & \text{otherwise} 
\end{cases} \]
Surface Area Heuristic (SAH)

\[
c(N) = \begin{cases} 
  c_T + \frac{SA(N_L)}{SA(N)} c(N_L) + \frac{SA(N_R)}{SA(N)} c(N_R) & \text{if } N \text{ is interior node} \\
  c_I |N| & \text{otherwise}
\end{cases}
\]

[MacDonald and Booth 1990]
Surface Area Heuristic (SAH)

\[ c(N) = \begin{cases} 
    c_I + \frac{SA(N_L)}{SA(N)} c(N_L) + \frac{SA(N_R)}{SA(N)} c(N_R) & \text{if } N \text{ is interior node} \\
    c_I |N| & \text{otherwise} 
\end{cases} \]

[MacDonald and Booth 1990]
Surface Area Heuristic (SAH)

\[
c(N) = \begin{cases} 
  c_T + \frac{SA(N_L)}{SA(N)} c(N_L) + \frac{SA(N_R)}{SA(N)} c(N_R) & \text{if } N \text{ is interior node} \\
  c_I |N| & \text{otherwise}
\end{cases}
\]

\[
c(N_{\text{root}}) = \frac{1}{SA(N_{\text{root}})} \left[ c_T \sum_{N_i} SA(N_i) + c_I \sum_{N_l} SA(N_l) |N_l| \right]
\]

[MacDonald and Booth 1990]
Surface Area Heuristic (SAH)

\[ c(N) = \begin{cases} 
  c_T + \frac{SA(N_L)}{SA(N)} c(N_L) + \frac{SA(N_R)}{SA(N)} c(N_R) 
  & \text{if } N \text{ is interior node} \\
  c_I |N| 
  & \text{otherwise} 
\]  

\[ c(N_{\text{root}}) = \frac{1}{SA(N_{\text{root}})} \left[ c_T \sum_{N_i} SA(N_i) + c_I \sum_{N_i} SA(N_i) |N_i| \right] \propto \sum_{N_i} SA(N_i) \quad \text{if } |N_l| = 1 \]  

[MacDonald and Booth 1990]
Agglomerative Clustering

Search for nearest neighbors for each cluster
Merge the closest cluster pair

Distance between clusters $C_1$ and $C_2$

$\text{d}(C_1, C_2) = \text{SA}(C_1 \cup C_2)$

[Walter et al. 2008]
Agglomerative Clustering

Repeat until only one cluster remains

Distance between clusters $C_1$ and $C_2$:

$$d(C_1, C_2) = SA(C_1 \cup C_2)$$

[Walter et al. 2008]
Agglomerative Clustering

Repeat until only one cluster remains

- Search for nearest neighbors for each cluster

\[ d(C_1, C_2) = SA(C_1 \cup C_2) \]

[Walter et al. 2008]
Agglomerative Clustering

Repeat until only one cluster remains

- Search for nearest neighbors for each cluster
- Merge the closest cluster pair
Agglomerative Clustering

Repeat until only one cluster remains

- Search for nearest neighbors for each cluster
- Merge the closest cluster pair

Distance between clusters $C_1$ and $C_2$ [Walter et al. 2008]

$$d(C_1, C_2) = SA(C_1 \cup C_2)$$
Agglomerative Clustering
Agglomerative Clustering
Agglomerative Clustering
Agglomerative Clustering
Agglomerative Clustering
Agglomerative Clustering
Agglomerative Clustering
Agglomerative Clustering
Agglomerative Clustering
Agglomerative Clustering
Agglomerative Clustering
Agglomerative Clustering
Agglomerative Clustering
Locally-Ordered Clustering

Non-decreasing property [Walter et al. 2008]

\[ d(C_1, C_2) \leq d(C_1 \cup C_3, C_2) : \forall C_1, C_2, C_3 \]

We can merge mutually corresponding clusters!
Locally-Ordered Clustering

Non-decreasing property [Walter et al. 2008]

\[ d(C_1, C_2) \leq d(C_1 \cup C_3, C_2) : \forall C_1, C_2, C_3 \]

We can merge mutually corresponding clusters!
Parallel Locally-Ordered Clustering

- Diagram of various shapes and bounding boxes.
Parallel Locally-Ordered Clustering
Parallel Locally-Ordered Clustering
Parallel Locally-Ordered Clustering
Parallel Locally-Ordered Clustering
Parallel Locally-Ordered Clustering
Parallel Locally-Ordered Clustering
Parallel Locally-Ordered Clustering
Parallel Locally-Ordered Clustering
Nearest Neighbor Search

Naïve approach
Time complexity $O(n^2)$
Prohibitive for practical use

kD-tree [Walter et al. 2008]
Difficult to implement
Not suitable for parallel processing

Morton curve (our approach)
Sort clusters along the Morton curve
Search in the sorted array around a given cluster
Neighborhood around the cluster determined by radius parameter $r$
Nearest Neighbor Search

Naïve approach

- Time complexity $\mathcal{O}(n^2)$
- Prohibitive for practical use
Nearest Neighbor Search

Naïve approach
- Time complexity $O(n^2)$
- Prohibitive for practical use

kD-tree [Walter et al. 2008]
- Difficult to implement
- Not suitable for parallel processing
Nearest Neighbor Search

Naïve approach
- Time complexity $O(n^2)$
- Prohibitive for practical use

kD-tree [Walter et al. 2008]
- Difficult to implement
- Not suitable for parallel processing

Morton curve (our approach)
- Sort clusters along the Morton curve
- Search in the sorted array around a given cluster
- Neighborhood around the cluster determined by radius parameter $r$
Approx. Nearest Neighbors along Morton Curve

Neighborhood determined by radius $r = 2$
Approx. Nearest Neighbors along Morton Curve

Neighborhood determined by radius $r = 2$
Approx. Nearest Neighbors along Morton Curve

Neighborhood determined by radius $r = 2$
Algorithm Overview

Repeat until one cluster remains
- Search nearest neighbor (in parallel)
- Merge (in parallel)
- Compact via prefix scan (in parallel)
Algorithm Overview

Repeat until one cluster remains
Algorithm Overview

Repeat until one cluster remains

- Search nearest neighbor (in parallel)
Algorithm Overview

Repeat until one cluster remains

- Search nearest neighbor (in parallel)

- Merge (in parallel)
Algorithm Overview

Repeat until one cluster remains
- Search nearest neighbor (in parallel)
- Merge (in parallel)
- Compact via prefix scan (in parallel)
Parallel Subtree Collapsing

1. Decide whether collapsing pays off
2. Identify leaf nodes (i.e. roots of collapsed subtrees)
3. Mark nodes as valid or invalid
Parallel Subtree Collapsing

1. Decide whether collapsing pays off
Parallel Subtree Collapsing

1. Decide whether collapsing pays off
Parallel Subtree Collapsing

1. Decide whether collapsing pays off
Parallel Subtree Collapsing

1. Decide whether collapsing pays off
Parallel Subtree Collapsing

1. Decide whether collapsing pays off
Parallel Subtree Collapsing

1. Decide whether collapsing pays off
2. Identify leaf nodes (i.e. roots of collapsed subtrees)
Parallel Subtree Collapsing

1. Decide whether collapsing pays off
2. Identify leaf nodes (i.e. roots of collapsed subtrees)
Parallel Subtree Collapsing

1. Decide whether collapsing pays off
2. Identify leaf nodes (i.e. roots of collapsed subtrees)
Parallel Subtree Collapsing

1. Decide whether collapsing pays off
2. Identify leaf nodes (i.e. roots of collapsed subtrees)
Parallel Subtree Collapsing

1. Decide whether collapsing pays off
2. Identify leaf nodes (i.e. roots of collapsed subtrees)
Parallel Subtree Collapsing

1. Decide whether collapsing pays off
2. Identify leaf nodes (i.e. roots of collapsed subtrees)
Parallel Subtree Collapsing

1. Decide whether collapsing pays off
2. Identify leaf nodes (i.e. roots of collapsed subtrees)
Parallel Subtree Collapsing

1. Decide whether collapsing pays off
2. Identify leaf nodes (i.e. roots of collapsed subtrees)
3. Mark nodes as valid or invalid
Parallel Subtree Collapsing

1. Decide whether collapsing pays off
2. Identify leaf nodes (i.e. roots of collapsed subtrees)
3. Mark nodes as valid or invalid
Parallel Subtree Collapsing

1. Decide whether collapsing pays off
2. Identify leaf nodes (i.e. roots of collapsed subtrees)
3. Mark nodes as valid or invalid
Parallel Subtree Collapsing

1. Decide whether collapsing pays off
2. Identify leaf nodes (i.e. roots of collapsed subtrees)
3. Mark nodes as valid or invalid
Parallel Subtree Collapsing

1. Decide whether collapsing pays off
2. Identify leaf nodes (i.e. roots of collapsed subtrees)
3. Mark nodes as valid or invalid
Parallel Subtree Collapsing

1. Decide whether collapsing pays off
2. Identify leaf nodes (i.e. roots of collapsed subtrees)
3. Mark nodes as valid or invalid
Implementation in CUDA

Shared memory cache of size $B + 2r$
- Block with $B$ threads
- Radius $r$
Results

- 9 scenes (331k - 12759k tris)
- Path tracing (GPU ray tracing kernel [Aila and Laine 2009])
  - Low quality rendering (8 spp)
  - High quality rendering (128 spp)
- Intel Core i7-3770 3.4 GHz CPU (4 cores), 16 GB RAM
- NVIDIA GeForce GTX TITAN X (Maxwell), 12 GB RAM
Tested Methods

- LBVH [Karras 2012]
  - Spatial median splits

- HLBVH [Garanzha et al. 2011]
  - Spatial median and SAH splits

- ATRBVH [Domingues and Pedrini 2015]
  - Treelet restructuring by agglomerative clustering

- PLOC
  - Parallel locally-ordered clustering (our algorithm)

Adaptive leaf sizes, SAH cost constants $c_T = 3$, $c_I = 2$
Pompeii

SAH cost (5632k tris, $r = 25$)
Pompeii

build time (5632k tris, $r = 25$)
Pompeii

time-to-image LQ (5632k tris, $r = 25$)

![Graph showing time-to-image LQ for LBVH, HLBVH, ATRBVH, PLOC]
Pompeii

time-to-image HQ (5632k tris, $r = 25$)

<table>
<thead>
<tr>
<th>Method</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LBVH</td>
<td>3980</td>
</tr>
<tr>
<td>HLBVH</td>
<td>3289</td>
</tr>
<tr>
<td>ATRBVH</td>
<td>2804</td>
</tr>
<tr>
<td>PLOC</td>
<td>2452</td>
</tr>
</tbody>
</table>
Powerplant

SAH cost (12759k tris, $r = 10$)
Powerplant

build time (12759k tris, $r = 10$)
Powerplant

time-to-image LQ (12759k tris, $r = 10$)

![Bar chart showing time-to-image LQ for different methods: LBVH, HLBVH, ATRBVH, and PLOC. The values are 746, 685, 775, and 588 respectively.]
Powerplant

time-to-image HQ (2759k tris, $r = 10$)

![Bar chart showing time-to-image HQ for different methods: LBVH, HLBVH, ATRBVH, PLOC. The values are 10551, 8958, 7909, and 6965 respectively.](chart.png)
Powerplant

![Graph showing time in milliseconds for different power plant operations]

- other computation
- collapse
- compaction
- merging
- nearest neighbor search
- sort

- \( PLOC_{r=10} \)
- \( PLOC_{r=25} \)
- \( PLOC_{r=100} \)
Conclusion and Future Work

GPU-based BVH construction using appr. agglomerative clustering
- Efficient and extremely simple
- Parallel subtree collapsing
- Implementation in CUDA with released source codes

Future work
- Varying radius across different iterations
- Extended Morton codes [Vinkler et al. 2017]
Thank you for your attention!

The project website with source codes
http://dcgi.fel.cvut.cz/projects/ploc/
Iterations

![Graph showing iterations vs. radius for various categories: Conference, Happy Buddha, Soda Hall, Hairball, Manuscript, Pompeii, San Miguel, Vienna, Power Plant.]
Comparison with AAC

Approximate agglomerative clustering [Gu et al. 2013]