Acceleration Structures for Ray Tracing – Utrecht, March 4th

by

Jacco Bikker
Introduction
Path Tracing * : 40 spp

- 512x512 pixels
- 1221 minutes (>20 hours)
- 20 primitives
- 3 area light sources

143 samples/s ≈ 1k rays/s

* The Rendering Equation, Kajiya, 1986
Acceleration Structures for Ray Tracing – Utrecht, March 4th
Basic optimization theory: *

\[
\text{Total time} = \sum_{i=0}^{\# \text{tasks}} \text{time of task}_i
\]

\[
\text{time of task}_i = \frac{\text{work of task}_i}{\text{rate of work of task}_i}
\]

Optimizing ray tracing:

- Reduce cost of single ray  \(\rightarrow\) intersection cost * number of primitives
- Reduce number of rays

*Code Optimization, Paul Hsieh, 2004
Reduce cost of single ray \(\Rightarrow\) *Reduce number of intersection tests*

- Grid: step through grid, intersect only primitives that overlap grid cells
Reduce cost of single ray $\Rightarrow$ *Reduce number of intersection tests*

- Binary space subdivision
Reduce cost of single ray ➔ *Reduce number of intersection tests*

- Object subdivision
Object partitioning

BVHs: A 3-Dimensional Representation for Fast Rendering of Complex Scenes, Rubin & Whitted, ’80

BIHs: Instant Ray Tracing: The Bounding Interval Hierarchy, Wächter & Keller, ‘06

Spatial partitioning

Octrees: *Space Subdivision for Fast Ray Tracing*, Glassner, '84

Grids: *ARTS: Accelerated Ray-Tracing System*, Fujimoto et al., '86

BSPs: *Data structures for Ray Tracing*, Jansen, '86

kD-trees: Automatic Creation of Object Hierarchies for Ray Tracing, Goldsmith & Salmon, ’87

Other: Accelerating Ray Tracing using Constrained Tetrahedralizations, Lagae & Dutré, ‘08
Object or spatial partitioning?

Factors:
- Construction / update time
- Traversal performance
- Memory use
- Code complexity?

Main contenders:
- kD-tree (see e.g. Havran**)
- BVH

* Quasi-Monte Carlo Light Transport Simulation by Efficient Ray Tracing, Carsten Wächter, 2008
** Heuristic Ray Shooting Algorithms, Vlastimil Havran, 2000
BVH
1. Scene bounds
2. Triangle barycenters
3. Pick split plane
4. Classify triangles
5. Determine bounds
6. Emit nodes
Two types of node:

1. Interior node (including root):
   - Extends
   - Pointer to left and right child

2. Leaf node:
   - Extends
   - List of triangles

```cpp
class BVHNode {
public:
    float3 bb_min;
    union { int leftChild; int firstTri; };
    float3 bb_max;
    int triCount; // leaf if not 0
}; // total 32 bytes
```
class BVHNode
{
public:
    float3 bb_min;
    union { int leftChild; int firstTri; };
    float3 bb_max;
    int triCount;
};

Traversing the BVH

*start with root node*

if ray intersects the node
    if node is a leaf
        intersect triangles
    else
        traverse left child
        traverse right child

Traversing the BVH with a ray packet *

*start with root node*

if any ray intersects the node
    if node is a leaf
        intersect triangles
    else
        traverse left child
        traverse right child

* Ray Tracing Deformable Scenes using Dynamic Bounding Volume Hierarchies, Wald et al., '07
The BVH potentially halves the number of triangles to be intersected at each node, at the cost of a ray/box intersection: algorithmic complexity becomes $O(\log N)$. Actual BVH quality depends on choice of split plane.

SAH *: minimize the total surface area of the nodes in the tree.

\[
\text{cost} = \text{CTRAV} + \text{CINT} \times (A_{\text{left}} \times N_{\text{left}} + A_{\text{right}} \times N_{\text{right}}) \\
\approx A_{\text{left}} \times N_{\text{left}} + A_{\text{right}} \times N_{\text{right}}
\]

* Heuristics for Ray Tracing using Space Subdivision, MacDonald & Booth, '90
cost = \( A_{\text{left}} \times N_{\text{left}} + A_{\text{right}} \times N_{\text{right}} \)
Reality Check
Acceleration Structures for Ray Tracing – Utrecht, March 4th
Typical (game) scenery:

- Static geometry 80%
- Rigid animation 8%
- Deforming geometry 8%
- Arbitrary animation 4%
**Observation:** a BVH is valid, no matter how much nodes overlap.

Therefore, two BVHs can be combined, by creating a new root that points to the two BVHs.

**Top-level BVH concept:**

1. Build a BVH *per scene graph node*.
2. Combine the nodes using a top-level BVH.

The ‘leafs’ of the top-level BVH are the roots of the scene graph node BVHs.
Top-level BVH approach:

- Static geometry: build once, high quality
- Rigid animation: build once, \textit{refit}
- Deforming geometry: build once, \textit{refit}
- Arbitrary animation: rebuild per frame.

\textbf{Instancing}: refer multiple times to the same BVH.
BVH refitting
BVH refitting

Refit cost: ~2ms per 100k triangles
Works for animation that maintains topology
BVH quick build
(binned building*)

Rebuild cost: ~20ms per 250k triangles

* On fast Construction of SAH-based Bounding Volume Hierarchies, Wald, ‘07
Advanced
Handling scenes with large polygons:

- Early split clipping*
- SAH-based spatial splits**

* Early Split Clipping for Bounding Volume Hierarchies, Ernst & Greiner, ‘07
** Spatial Splits in Bounding Volume Hierarchies, Stich et al., ‘09
Procedure:

1. Find best object split
   \[ \text{cost} = A_{\text{left}} \times N_{\text{left}} + A_{\text{right}} \times N_{\text{right}}, \]
   \[ N_{\text{left}} + N_{\text{right}} = N_{\text{parent}} \]

2. Find best spatial split
   \[ \text{cost} = A_{\text{left}} \times N_{\text{left}} + A_{\text{right}} \times N_{\text{right}}, \]
   \[ N_{\text{left}} + N_{\text{right}} \geq N_{\text{parent}} \]

3. Apply best split
Results
THANK YOU.

Questions?

“Ray tracing performance is measured in millions of rays, not thousands”
Other useful material:

Fast, Effective BVH Updates for Dynamic Ray-Traced Scenes Using Tree Rotations
Kopta et al., ’11

HLBVH Hierarchical LBVH Construction for Real-Time Ray Tracing of Dynamic Geometry
Pantaleoni & Leubke, ’10

Ray Tracing Animated Scenes using Coherent Grid Traversal
Wald et al., ’06

Ray Tracing Dynamic Scenes using Selective Restructuring
Yoon et al., ’07

Real-Time KD-Tree Construction on Graphics Hardware
Zhou et al., ’08