Spatial Splits in Bounding Volume Hierarchies

Martin Stich
Heiko Friedrich
Andreas Dietrich

NVIDIA Research

BVHs for Ray Tracing

- Perform well on GPUs
- Low memory footprint
- Simple+fast construction
- Fast refitting in animations

BVHs for Ray Tracing

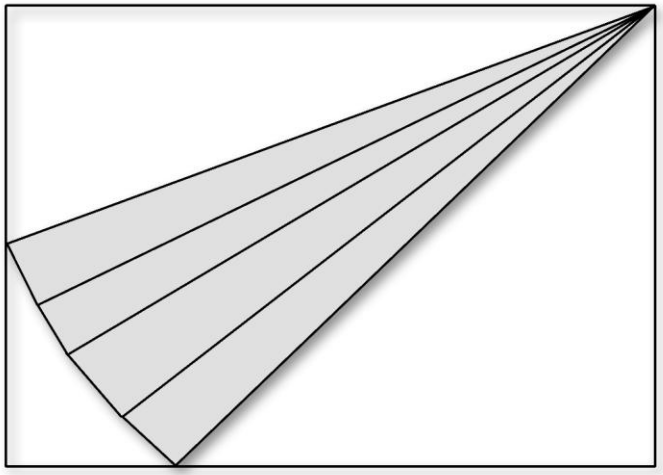
- Perform well on GPUs
- Low memory footprint
- Simple+fast construction
- Fast refitting in animations
- Don’t adapt well to non-uniformly tessellated scenes (no problem for kd-trees or other spatial hierarchies)

The Problem: Node Overlap

- Minimum leaf bounding volume is that of primitives
- Nodes overlap, particularly badly with large primitives
- Overlap means hierarchy doesn't save us any work

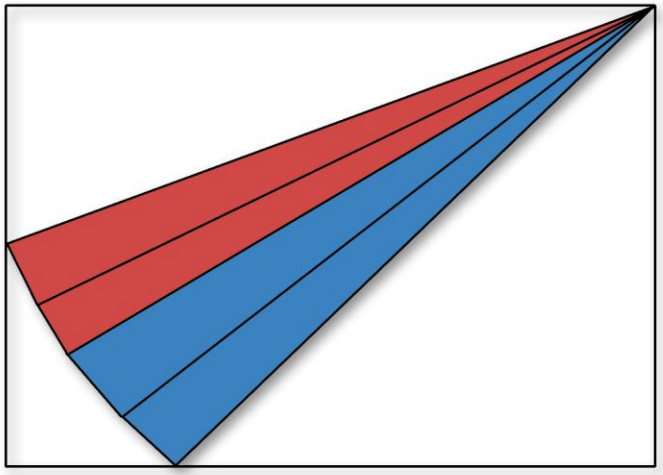
The Problem: Node Overlap

- Minimum leaf bounding volume is that of primitives
- Nodes overlap, particularly badly with large primitives
- Overlap means hierarchy doesn't save us any work



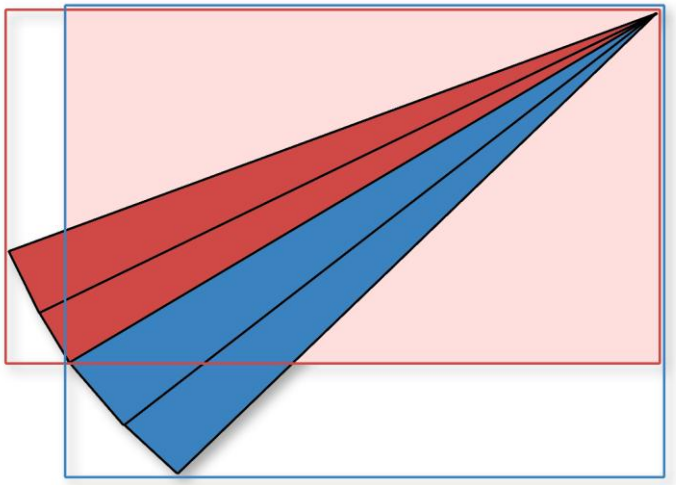
The Problem: Node Overlap

- Minimum leaf bounding volume is that of primitives
- Nodes overlap, particularly badly with large primitives
- Overlap means hierarchy doesn't save us any work



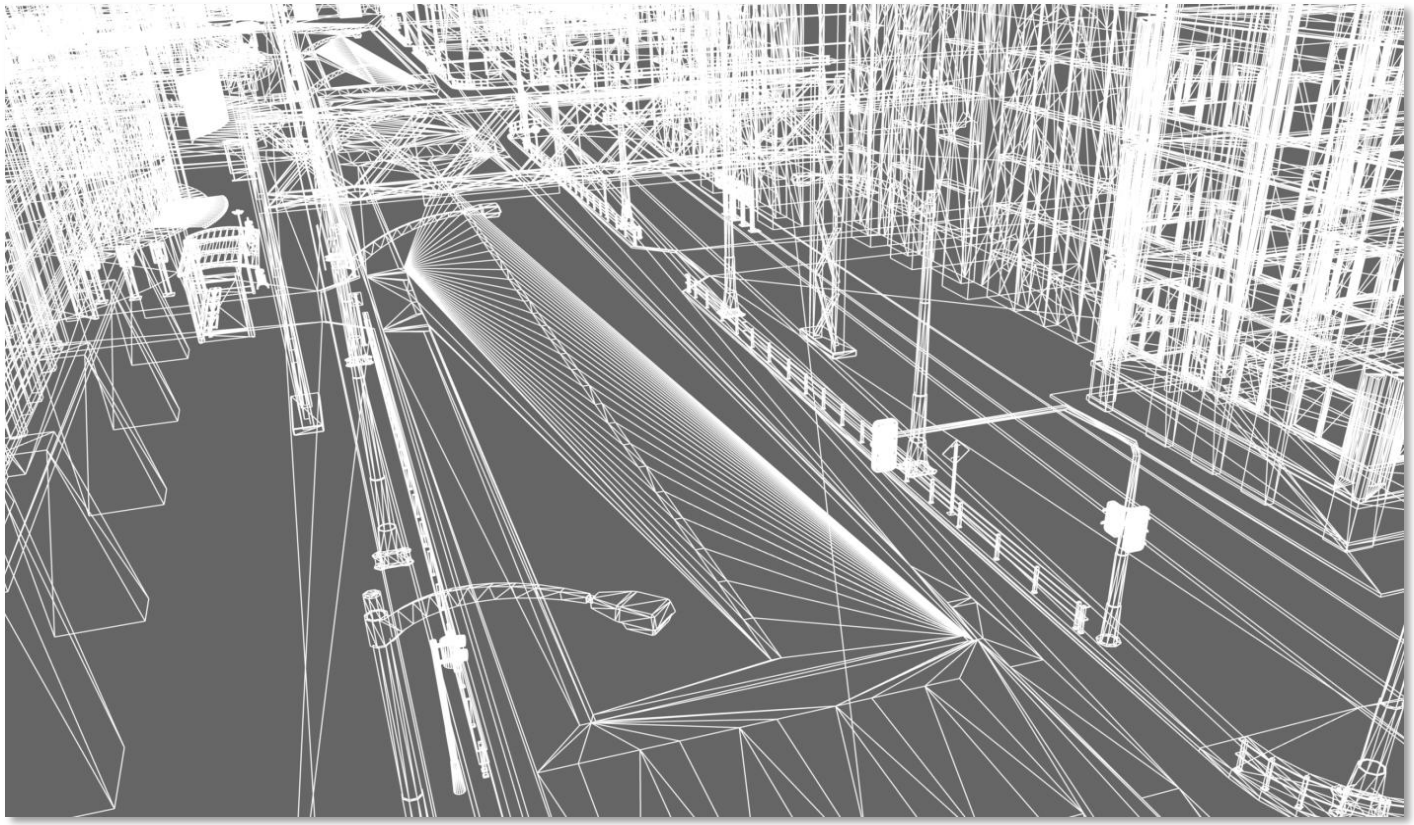
The Problem: Node Overlap

- Minimum leaf bounding volume is that of primitives
- Nodes overlap, particularly badly with large primitives
- Overlap means hierarchy doesn't save us any work



The Problem: Node Overlap

- It happens in the “real world”



Previous Methods

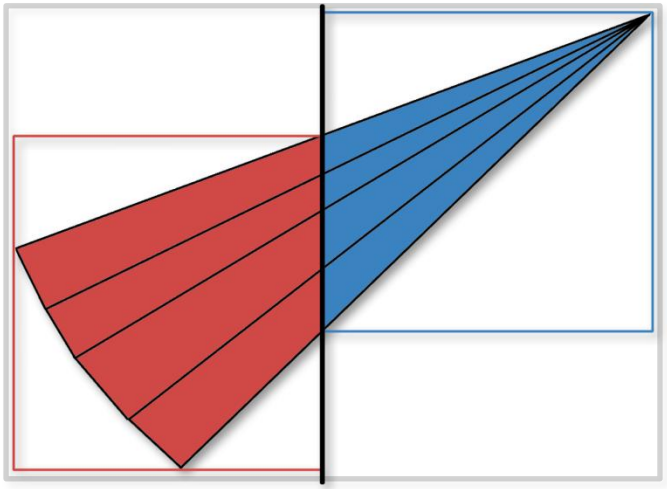
- Straightforward pre-tessellation
  - Waste of memory
  - Numerically difficult

- Presplit bounding boxes
  - *Early Split Clipping* [Ernst & Greiner 07]
  - *Edge Volume Heuristic* [Dammertz & Keller 08]
  - Basic idea: pre-pass subdivides primitive AABBs
    - BVH built on top of those
  - Primitives may be referenced more than once (which is ok)

- Hard to get good !/$ with all above methods
  - ! is often small, if anything

Spatial Splits in BVHs

- Need for better splitting strategy
- No reason we can't spatially split a BVH node:



- Object splitting "on demand" during build
  - No longer a pre-pass
  - Can splits objects only where we really need it

The Split-BVH (SBVH)

- Use spatial splits during BVH build
  - But only where we expect benefit
- Very simple algorithm:

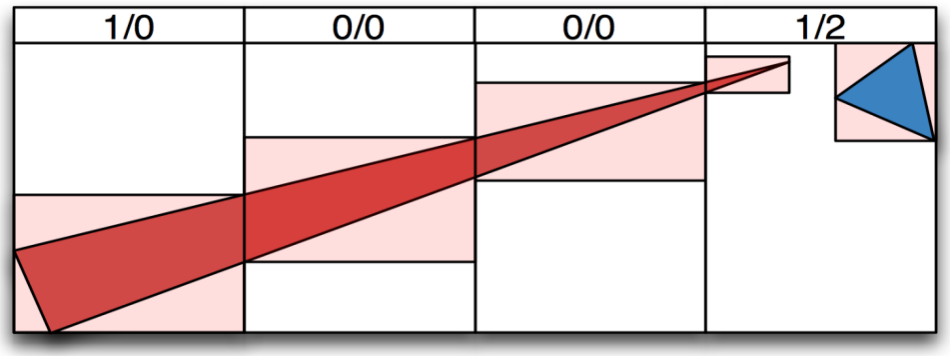
```
SplitNode()
{
    split1 = findBestObjectSplit()    // BVH split as usual
    split2 = findBestSpatialSplit()   // like kd (almost)

    if( split1.SAHcost <= split2.SAHcost )
        PerformSplit( split1 )
    else
        PerformSplit( split2 )
}
```

Finding Spatial Splits

- Very similar to split search for kd-trees
- A few subtle differences exist, though
  - Because we store full AABBs, not just a plane
  - More details in the paper
- Our implementation uses a binning approach
  - Good tradeoff between speed and quality

Finding Spatial Splits

- Binning: consider a set of equidistant split planes
  - Finds split in linear time
- Adapt conventional binning to fit our needs
  - Store an AABB in each bin (like BVH binning)
  - Store entry/exit counters in each bin (like kd-tree binning)
  - Fill the bins with clipped primitive AABBs (*"chopped binning"*)

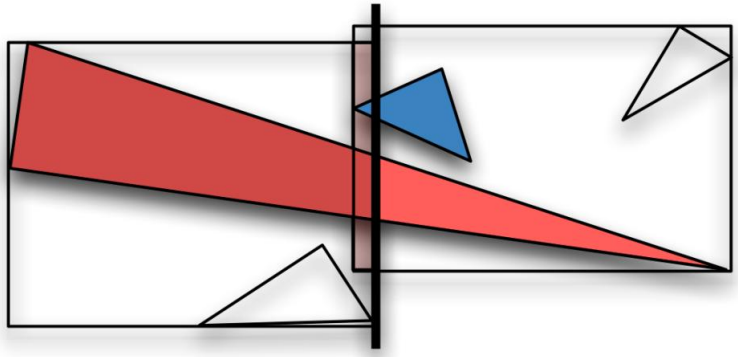


Finding Spatial Splits

- Bin in all 3 dimensions
- Use bins to find cheapest split
- Done!
  - That's all we needed to implement **`findBestSpatialSplit()`**

Improving Spatial Splits

- Building a BVH, so no reason a split plane must be definite
  - Allow some overlap even for spatial splits
  - Results in “hybrid” between object and spatial split



- *Unsplitting:* Check if putting a split object only in one of the children improves SAH cost
- Reduces overall SAH cost (but not dramatically)

Restricting Spatial Splits

- SBVH is good at finding splits
  - Hierarchies may become deeper than we want
  - But low memory footprint was what we liked about BVHs!
- Spatial Splits are most effective at the top levels
  - That's where node overlap is biggest and hurts the most
- Solution: use only object splits in lower levels
  - Need heuristic that decides whether attempting a spatial split is worth it

Restricting Spatial Splits

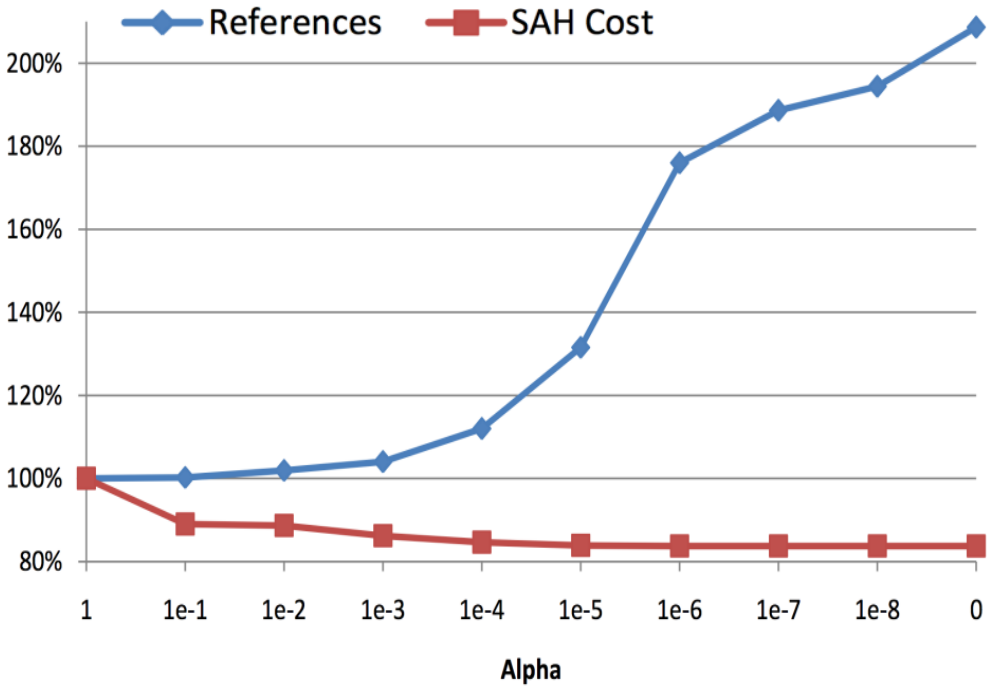
- Make spatial split attempts dependent on amount of overlap from best object split:

$$\text{attemptSpatial} = \frac{SA(\text{AABB}_1 \cap \text{AABB}_2)}{SA(\text{AABB}_{\text{root}})} > \alpha$$

- α is user-parameter in [0,1]
  - 1 = regular BVH, 0 = always try spatial split
  - Parameter is bounded and does *not* control splitting directly
  - Well-behaved: we just use $\alpha = 10^{-5}$ all the time

Restricting Spatial Splits

- Example: Conference room



Results

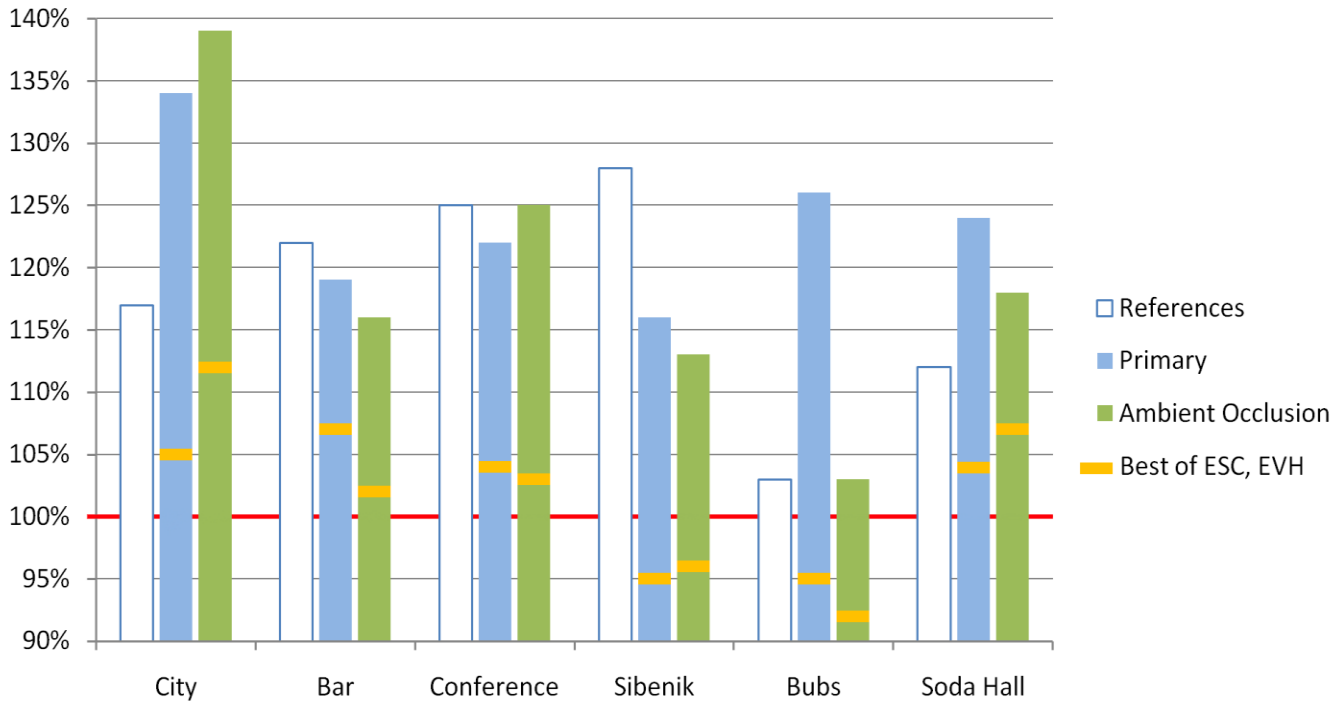
- Measured number of datapoints across variety of scenes
  - Total SAH cost
  - Memory (references and nodes)
  - Primitive intersections
  - Traversal steps
  - Ray casting performance

  }
    - Multiple viewpoints per scene
    - Primary and AO rays

- Compared SBVH to regular BVH, ESC, and EVH
  - ESC and EVH had same object duplication budget as SBVH
  - SBVH consistently better
  - See the paper for details

Results

- Perf measured with our fastest kernels [Aila & Laine 09]



- Semi-artificial case (rotated Sponza): about 200%

Conclusion

- SBVH: new BVH scheme for “difficult” scenes
  - Doesn’t hurt if scene is *not* difficult
- Consistent improvement over previous methods
- Simple to implement
  - Just add spatial split search to your BVH builder
- Practical
  - We use it in OptiX

Conclusion

- Limitations/Downsides
  - Builds are slower than pure BVH
  - Sacrificed some simplicity
  - No more simple refitting
- Future work
  - Parallel GPU builds
  - In-place build in bounded memory
  - Applications other than ray tracing

Thank You!

mstich@nvidia.com