Lecture 3 - “The Perfect BVH”

Welcome!

\[ I(x, x') = g(x, x') \left[ \epsilon(x, x') + \int S(x, x', x''') I(x', x'') dx'' \right] \]
Today's Agenda:

- Building Better BVHs
- The Problem of Large Polygons
- Refitting
- Fast BVH Construction
Better BVHs
Better BVHs
Better BVHs

What Are We Trying To Solve?

A BVH is used to reduce the number of ray/primitive intersections.

But: it introduces new intersections.

The ideal BVH minimizes:

- # of ray / primitive intersections
- # of ray / node intersections.
Better BVHs

```c
// Define the BVH structure and rendering functions here.
// This code snippet is an example of how to implement BVHs for efficient ray tracing.
// The code includes decision-making processes for selecting nodes and determining
// whether to sample from the diffuse or specular branch.

// Example BVH node structure
struct Node { ...

// Ray-tracing function
void traceRay(const Ray& r, Node* node, float& t) {
    if (node == nullptr) return;
    // ... decision-making logic ...
    if (node->isLeaf) {
        // Sample from diffuse branch
        float pdf = samplePdf(node); ...
        // Update t with probability
        t *= pdf;
    } else {
        // Split into child nodes
        Node* left = ...; ...
        Node* right = ...; ...
        traceRay(r, left, t); ...
        traceRay(r, right, t); ...
    }
}
```
Better BVHs

BVH versus kD-tree

The BVH better encapsulates geometry.

➔ This reduces the chance of a ray hitting a node.

➔ This is all about probabilities!

What is the probability of a ray hitting a random triangle?

What is the probability of a ray hitting a random node?

This probability is proportional to surface area.
Better BVHs

Route 1: 10% up-time, $1000 fine

Route 2: 100% up-time, $100 fine
Optimal Split Plane Position

The ideal split minimizes the expected cost of a ray intersecting the resulting nodes.

This expected cost is based on:

- Number of primitives that will have to be intersected
- Probability of this happening

The cost of a split is thus:

\[ A_{left} \times N_{left} + A_{right} \times N_{right} \]
Optimal Split Plane Position

The ideal split minimizes the cost of a ray intersecting the resulting nodes.

This cost depends on:

- Number of primitives that will have to be intersected
- Probability of this happening

The cost of a split is thus:

\[ A_{left} \times N_{left} + A_{right} \times N_{right} \]
Better BVHs

Optimal Split Plane Position

Or, more concisely:

\[ A_{\text{left}}^0 \times (A_{\text{left}}^1 \times N_{\text{left}}^1 + A_{\text{right}}^1 \times N_{\text{right}}^1) \]

+ 

\[ A_{\text{right}}^0 \times (A_{\text{left}}^2 \times N_{\text{left}}^2 + A_{\text{right}}^2 \times N_{\text{right}}^2) \]
Better BVHs

Optimal Split Plane Position

Which positions do we consider?

Object subdivision may happen over x, y or z axis.

The cost function is constant between primitive centroids.

⇒ For N primitives: \(3(N - 1)\) possible locations

⇒ For a 2-level tree: \((3(N - 1))^2\) configurations
Better BVHs

SAH and Termination

A split is ‘not worth it’ if it doesn’t yield a cost lower than the cost of the parent node, i.e.:

\[ A_{\text{left}} \times N_{\text{left}} + A_{\text{right}} \times N_{\text{right}} \geq A \times N \]

This provides us with a natural and optimal termination criterion.

(and it solves the problem of the Bad Artist)
Better BVHs

Optimal Split Plane Position

Evaluating $(3(N - 1))^2$ configurations?

Solution: apply the *surface area heuristic* (SAH) in a greedy manner*.  

Better BVHs

Optimal Split Plane Position

Comparing naïve versus SAH:
- SAH will cut #intersections in half;
- expect ~2x better performance.

SAH & kD-trees:
- Same scheme applies.
Better BVHs

Median Split
Better BVHs

Surface Area Heuristic
Better BVHs
Better BVHs
Better BVHs
Today’s Agenda:

- Building Better BVHs
- The Problem of Large Polygons
- Refitting
- Fast BVH Construction
Splitting

Problematic Large Polygons

Large polygons lead to poor BVHs.

(far more common than you’d think)
Splitting

Problematic Large Polygons

Large polygons lead to poor BVHs.

Using the spatial splits in kD-trees, this is far less of an issue:

The triangle will simply be assigned to each subspace.
Splitting

Problematic Large Polygons

Large polygons lead to poor BVHs.

Using the spatial splits in kD-trees, this is far less of an issue:

The triangle will simply be assigned to each subspace.

Solution 1: split large polygons*.

Observations:

1. A polygon can safely reside in multiple leafs;
2. The bounds of a leaf do not have to include the entire polygon.

*: Early Split Clipping for Bounding Volume Hierarchies, Ernst & Greiner, 2007
Early Split Clipping

Observations:
1. A polygon can safely reside in multiple leafs;
2. The bounds of a leaf do not have to include the entire polygon.
3. BVH construction only uses primitive bounding boxes.

Algorithm:
Prior to BVH construction, we recursively subdivide any polygon with a surface area that exceeds a certain threshold.

Issues:
- Threshold parameter
- Individual polygons are split, regardless of surrounding geometry
- Primitives may end up multiple times in the same leaf

(some of these issues are resolved in: The Edge Volume Heuristic - Robust Triangle Subdivision for Improved BVH Performance, Dammertz & Keller, 2008)
Splitting

Spatial Splits for BVHs

Observation: spatial splits are not limited to kD-trees.

But: spatial splits tend to increase the cost of a split.

Idea:

1. Determine cost of optimal object partition;
2. Determine cost of optimal spatial split;
3. Apply spatial split if cost is lower than object partition*.

\[ C_{\text{split}} = A_{\text{left}} \times N_{\text{left}} + A_{\text{right}} \times N_{\text{right}} < A \times N \]

*: Spatial Splits in Bounding Volume Hierarchies, Stich et al., 2009
Splitting

State of the Art: SBVH

Summary: high quality bounding volume hierarchies can be obtained by combining the surface area heuristic and spatial splits.

Compared to a regular SAH BVH, spatial splits improve the BVH by $\sim25\%$ (see paper for scenes and figures).
Better BVHs
Better BVHs
Today’s Agenda:

- Building Better BVHs
- The Problem of Large Polygons
- Refitting
- Fast BVH Construction
Refitting

Summary of BVH Characteristics

A BVH provides significant freedom compared to e.g. a kD-tree:

- No need for a 1-to-1 relation between bounding boxes and primitives
- Primitives may reside in multiple leafs
- Bounding boxes may overlap
- Bounding boxes can be altered, as long as they fit in their parent box
- A BVH can be very bad but still valid

Some consequences / opportunities:

- We can rebuild part of a BVH
- We can combine two BVHs into one
- We can refit a BVH
Refitting

Q: What happens to the BVH of a tree model, if we make it bend in the wind?

A: Likely, only bounds will change; the topology of the BVH will be the same (or at least similar) in each frame.

Refitting:

*Updating the bounding boxes stored in a BVH to match changed primitive coordinates.*
Refitting

**Refitting**

*Updating the bounding boxes stored in a BVH to match changed primitive coordinates.*

**Algorithm:**

1. For each leaf, calculate the bounds over the primitives it represents.
2. Update parent bounds.
Refitting

Refitting - Suitability
Refitting – Practical

Order of nodes in the node array:

We will never find the parent of node \( X \) at a position greater than \( X \).

Therefore:

```c
for( int i = N-1; i >= 0; i-- )
  nodeArray[i].AdjustBounds();
```
Today's Agenda:

- Building Better BVHs
- The Problem of Large Polygons
- Refitting
- Fast BVH Construction
Binning

Rapid BVH Construction

Refitting allows us to update hundreds of thousands of primitives in real-time. But what if topology changes significantly?

Rebuilding a BVH requires $3N\log N$ split plane evaluations.

Options:

1. Do not use SAH (significantly lower quality BVH)
2. Do not evaluate all 3 axes (minor degradation of BVH quality)
3. Make split plane selection independent of $N$
Binned BVH Construction*

Binned construction:

*Evaluate SAH at N discrete intervals.*

Binning

Binned BVH Construction

Detailed algorithm:

1. Calculate spatial bounds
2. Calculate object centroid bounds
3. Calculate intervals (efficiently and accurately!)
4. Populate bins
5. Sweep: evaluate cost, keep track of counts
6. Use best position
Binning

Binned BVH Construction

Performance evaluation:

1.9s for 10M triangles (8 cores @ 2.6Ghz).
Binning

```
# (depth < threshold)
if (inside) { 1.0 } else {
  S = nt - n, H = nt - n;
  R = 0.5 * W + H;
  if (R > 0.75) R = 0.5 + 0.5 * (R - 0.75);
  if (R < 0.25) R = 0.5 + 0.5 * (R - 0.25);
  E = diffuse;
  E = true;
  if (refr) && (depth < threshold)
    E = true;
  } // diffuse
```

olve = SurviveProbability + diffuse; estimation - doing it properly, this
do not to efficiency;
Radiance = SampleLight: brand, I, A, N, N.x + radiance.y + radiance.z) > 0)
E = true;
int Breitbart = EvaluateDiffuse( L, N.)
att factor = diffuse = I002;
att weight = dist (direct?, brander);
att convolution = dot (0, 1.1);
E = (weight * costTheta400) * direct;
random walk - done properly, closely follows
```
Today’s Agenda:

- Building Better BVHs
- The Problem of Large Polygons
- Refitting
- Fast BVH Construction
INFOMAGR – Advanced Graphics

Jacco Bikker  -  November 2018 - February 2019

END of “The Perfect BVH”

next lecture: “Real-time Ray Tracing”