Subspace Culling for Ray-Box Intersection

Atsushi Yoshimura
Takahiro Harada
Problem - what we want to solve

- AABB is a popular bounding volume, especially, on BVH for ray tracing
  - Simple representation 😊
  - Cheap intersection 😊
  - There are well-matured methods to build a high-quality BVH with a good quality 😊

```c
struct AABB {
    vec3 min;
    vec3 max;
};
```
Problem - what we want to solve

• AABB is a popular bounding volume, especially, on BVH for ray tracing
  • Simple representation 😊
  • Cheap intersection 😊
  • There are well-matured methods to build a high-quality BVH with a good quality 😊
• AABB may not be able to tightly bound a geometry in some cases 😞
  • Ray-BVH intersection can be slower due to traversal of too many nodes
How many false Positives Are there?

• A Toy Experiment
  1. A triangle enclosed by an AABB
  2. Shoot random rays toward the AABB
  3. Count how many rays do not hit the triangle

\[ \text{False Positive Ratio} = \frac{\text{White Rays}}{\text{Total Rays}} \]

\[ \Rightarrow \text{False-positive ratio can be over 70%!} \]
Alternative Bounding volumes?

There are alternative volumes

- Better culling 😊
- More computational overhead 😃
- Larger memory consumption 😞
- Some of them are struggling to build high quality BVH efficiently 😞
  - Nick Vitsas et al. proposed a parallel OBB tree construction - “Parallel Transformation of Bounding Volume Hierarchies into Oriented Bounding Box Trees”

Our goal

- Culls false positives more than AABB
- Small computational overhead for culling
- Small memory footprint
- Simple and Fast BVH construction
Voxel data structure

- It can fit densely to the primitive 😊
  - As long as the grid resolution is fine enough

- Simple representation
  - 1 bit per cell to represent its occupancy

- Fast Intersection with Rays
  - $\text{DDA}^1$-like iterative method?
  - We use LUT-based approach
    - $O(1)$
    - Only a few arithmetic operations

1. Digital Differential Analyzer
Small voxels vs Ray

- Intersection with a bitwise AND
  - A ray vs small grid test can be conservatively replaced by “bitwise AND”
  - Ray mask can be precomputed as a look-up table 😊, so we can make it O(1)
Ray Mask Build

- In general cases, it takes linear time complexity to build a voxel pattern from a ray
  - DDA-like method. e.g. John Amanatides, Andrew Woo, "A Fast Voxel Traversal Algorithm for Ray Tracing"
  - But we can use a look-up table approach - $O(1)$ as an approximation 😊
Our Approach – in a nutshell

• Use voxels for one step more culling after the AABB intersection

Step 1. AABB Intersection  
Step 2. Voxel Culling  
Rejected 😊
- A few memory lookups
- A few bitwise ANDs
Classical Traversal

```plaintext
function traverse(root, ray, R, rayMasks)
    push(root);
    while True
        node ← pop();
        if node is empty then break;
        if node is a leaf then
            Find an intersection ray and triangles in node;
            continue;
        end
        hits ← find intersections AABBs in node and ray;
        foreach hit_i ∈ hits do
            child ← get an child node that corresponds hit_i;
            push(child);
        end
    end
end
```

Traversal Loop

Pop a node

Process a leaf node

AABB tests and pushing if it hits
Our Traversal

function traverse(root, ray, R, rayMasks)
    push(root);
    while True
        node ← pop();
        if node is empty then break;
        if node is a leaf then
            Find an intersection ray and triangles in node;
            continue;
        end
        hits ← find intersections AABBs in node and ray;
        foreach hiti ∈ hits do
            AABB, objectMask, ps ← get an AABB, an object mask, and intersections of hiti;
            rayMask ← lookupRayMask(AABBlower, AABBupper, ps0, ps1, R, rayMasks);
            if objectMask ∧ rayMask is not zero then
                child ← get an child node that corresponds hiti;
                push(child);
            end
        end
    end
Hierarchical Construction

- The best voxels
  - Traverse all of the triangles below 😞
  - It is expensive 😞

A bvh hierarchy

The voxels on the top node
Hierarchical Construction

- Approximate the occupancy by voxels

Calculated mask from the children’s voxels
Hierarchical Construction

- Approximate the occupancy by voxels
- don’t have to traverse lower level of the data structure 😊
Exact Occupancy vs Approximated Occupancy

- Similar result with the approximated occupancy with lower cost 😊
- Trade-offs
Exact Occupancy vs Approximated Occupancy

• Similar result with the approximated occupancy with lower cost 😊
• Trade-offs
  • How many levels we will go down

Rough Approximation

1 Level

2 Levels

1 step more accurate, but need more nodes to visit
Voxel Compression

• How much memory do we need for the voxels?

  \[4 \times 4 \times 4 = 64 \text{ bits} = 8 \text{ bytes}\]
  \[6 \times 6 \times 6 = 216 \text{ bits} = 27 \text{ bytes}\]
Voxel Compression

- Good news: The bit patterns are not just random
- There are some “frequently used mask patterns”

Look-up table approach

<table>
<thead>
<tr>
<th>Index</th>
<th>Mask</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0]</td>
<td>11111111</td>
</tr>
<tr>
<td>[1]</td>
<td>01100111</td>
</tr>
<tr>
<td>[2]</td>
<td>11000000</td>
</tr>
<tr>
<td>[3]</td>
<td>11010101</td>
</tr>
<tr>
<td>....</td>
<td>....</td>
</tr>
</tbody>
</table>

256 elements

frequently used masks

Masks by SAH-based prioritization

A BVH node

<table>
<thead>
<tr>
<th>child 0</th>
<th>child 1</th>
<th>child 2</th>
<th>child 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>AABB 0</td>
<td>AABB 1</td>
<td>AABB 2</td>
<td>AABB 3</td>
</tr>
</tbody>
</table>

1 2 8 3

Just 1 byte per box always 😊
Voxel resolution independent
Results

• We measured:
  • **Number of intersections** to see culling efficiency
  • Entire rendering performance
Results: Number of Intersection

- Voxel resolutions
  - 4x4x4, 6x6x6
- Object Mask
  - Only 1 byte per AABB by the LUT-based compression
- Culling
  - DDA-based exact culling for measuring the voxel culling capability
- From 9 to 38% of the intersections are reduced with R=4 😊
- From 12 to 46% of the intersections are reduced with R=6 😊

- Good reduction for thin and tilted geometries 😊
- LUT-based compression for the voxel mask works very well 😊

<table>
<thead>
<tr>
<th>Scene</th>
<th>Ratio of Intersection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bedroom</td>
<td>86.9%</td>
</tr>
<tr>
<td>San Miguel</td>
<td>76.5%</td>
</tr>
<tr>
<td>Ninja</td>
<td>87.7%</td>
</tr>
<tr>
<td>Bistro</td>
<td>71.5%</td>
</tr>
<tr>
<td>Classroom</td>
<td>85.1%</td>
</tr>
<tr>
<td>Hairball</td>
<td>74.3%</td>
</tr>
<tr>
<td>Curly Hair</td>
<td>60.4%</td>
</tr>
<tr>
<td>Straight Hair</td>
<td>53.6%</td>
</tr>
<tr>
<td>VictorianTrains</td>
<td>81.1%</td>
</tr>
</tbody>
</table>

The same Memory Size
Results: Entire performance

- Voxel resolutions
  - 4x4x4, 6x6x6
- Object Mask
  - Naïvely keep 8 bytes, and 27 bytes respectively without compression
- Culling
  - LUT-based approach
- Unfortunately, we observed some scenes get worse 😞
- Hair scenes got performance improvements 😊
  - 12%, 13% faster than the baseline with R=4
  - 13%, 14% faster than the baseline with R=6
Summary

• Culls false positives more than AABB
  ✔ Voxel-based culling reduces the number of intersections very well
• Small computational overhead for culling
  ✔ LUT-based fast intersection
• Small memory footprint
  ✔ LUT-based compression significantly reduces memory size but keep culling efficiency
• Simple and Fast BVH construction
  ✔ A simple occupancy approximation for efficient voxel builds in BVH
  ✔ Trade-off control
Limitations and Future work

• It’s still hard to see improvement with all scenes
• Dynamic scenes
  • Animated geometries requires the voxel data update
• GPU measurement & optimization
• The voxel compression algorithm is still sub-optimal