Subspace Culling for Ray-Box Intersection

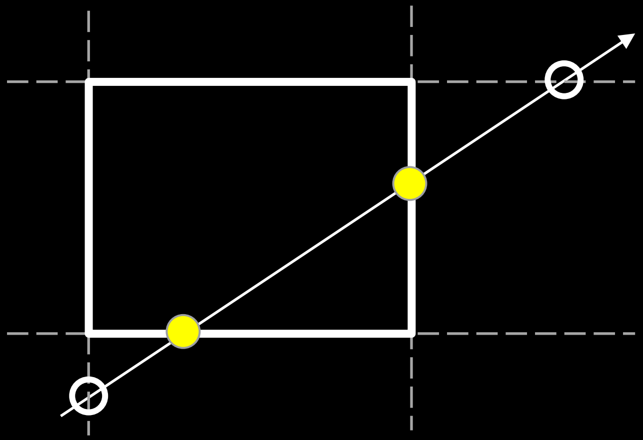
Atsushi Yoshimura
Takahiro Harada

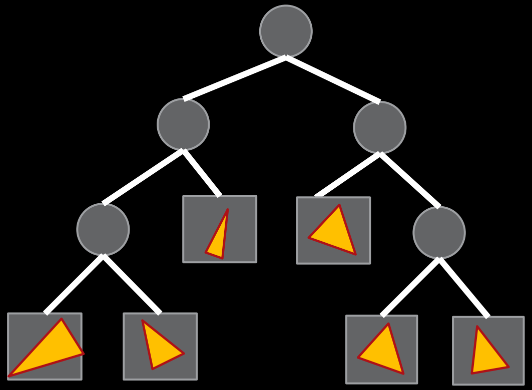
Advanced Rendering Research Group

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 ☺

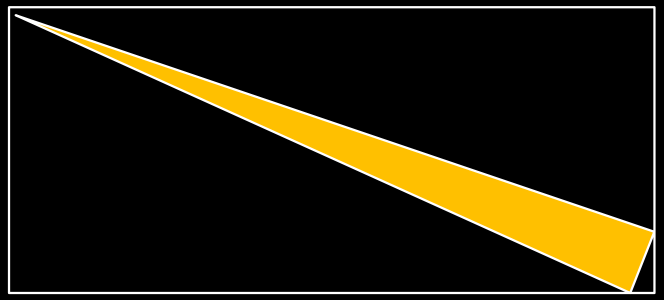
```
struct AABB
{
    vec3 min;
    vec3 max;
};
```



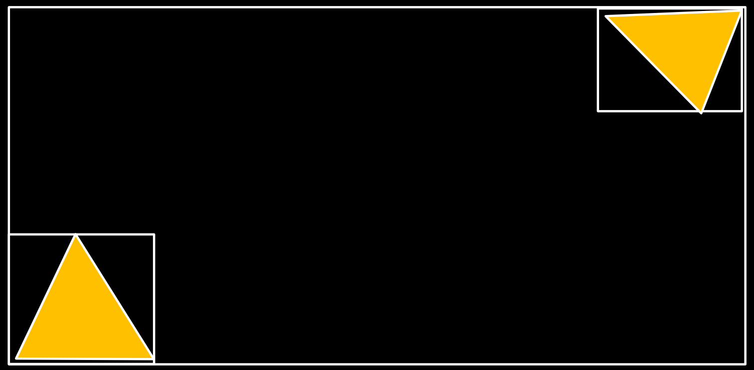


Problem - what we want to solve

- 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



Thin, tilted triangle

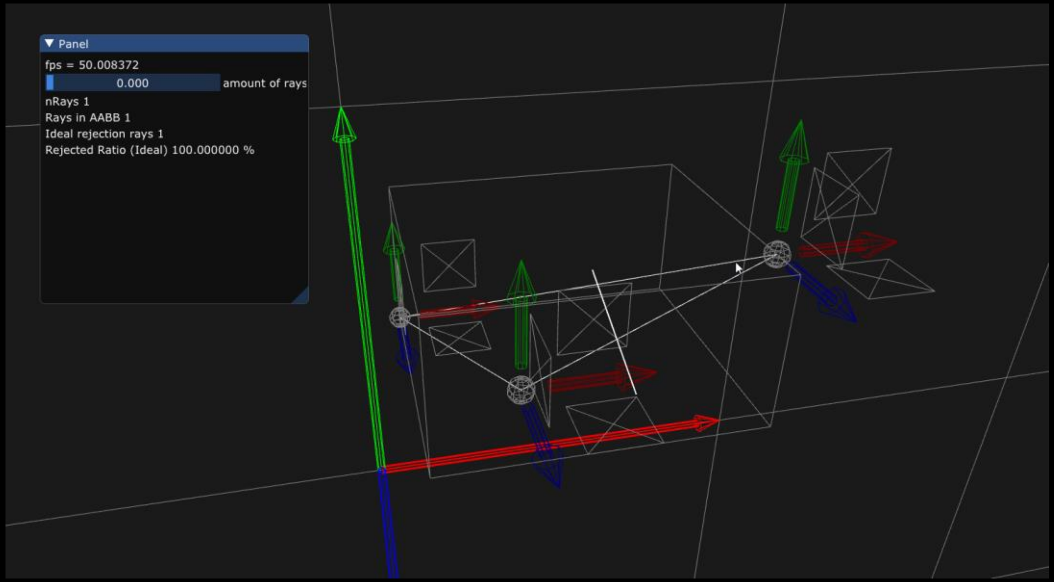


Sparse triangles

ARR
Advanced Rendering Research Group

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



$$\mathit{False\ Positive\ Ratio} = \frac{\mathit{White\ Rays}}{\mathit{Total\ Rays}}$$

⇨ **False-positive ratio can be over 70%!**

AMD

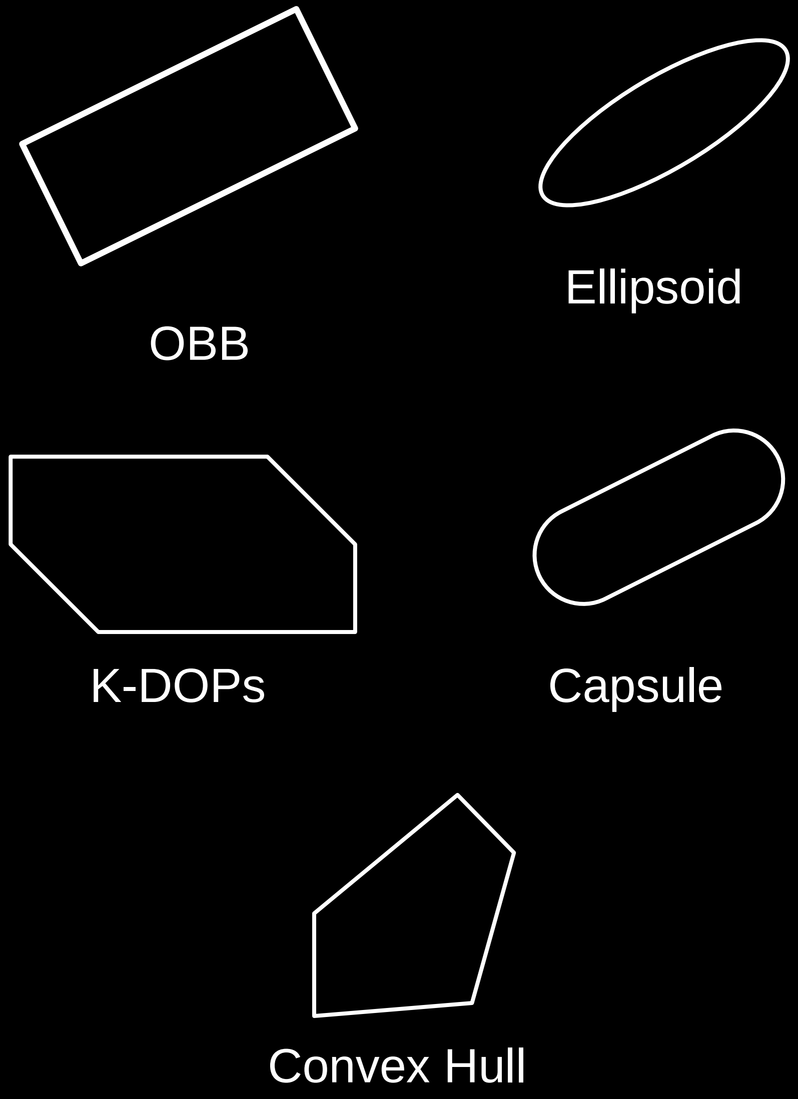
Alternative Bounding volumes?

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”

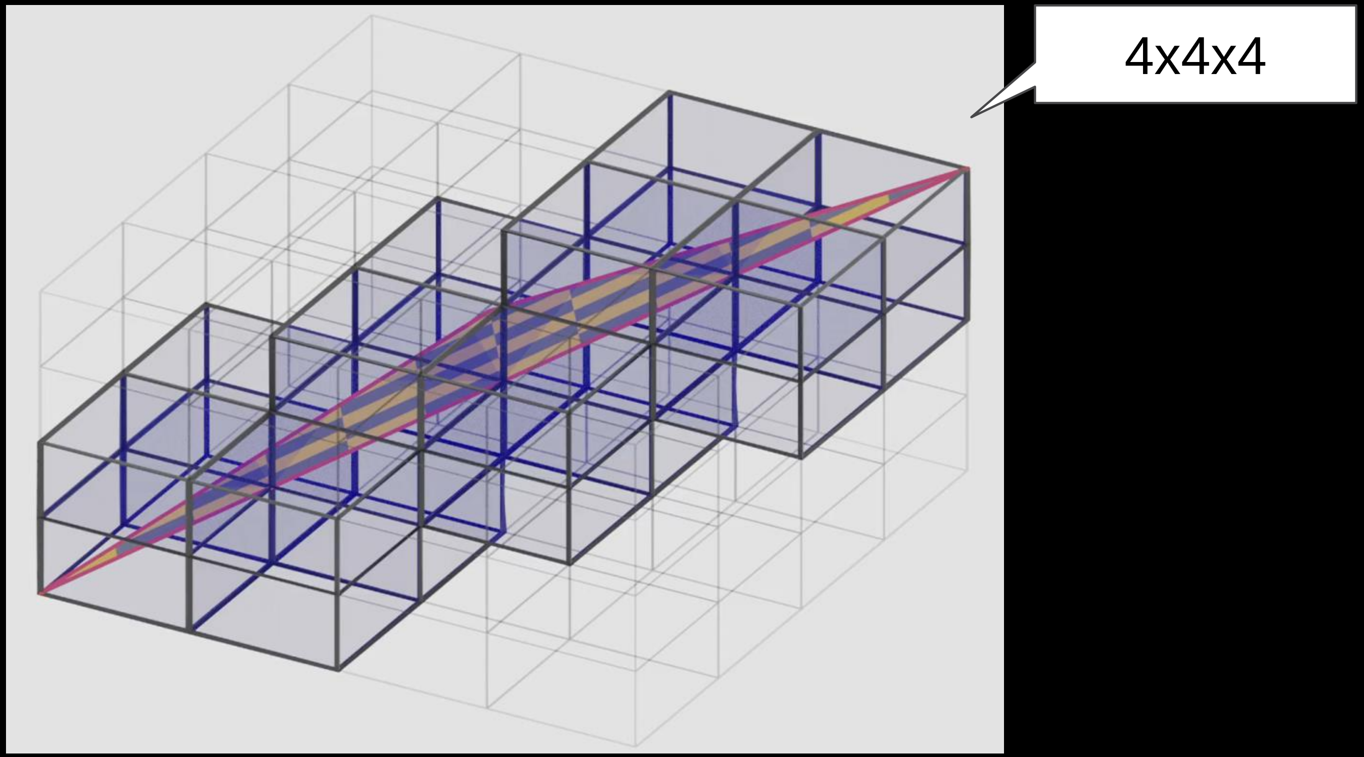
Our goal

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



Voxel data structure

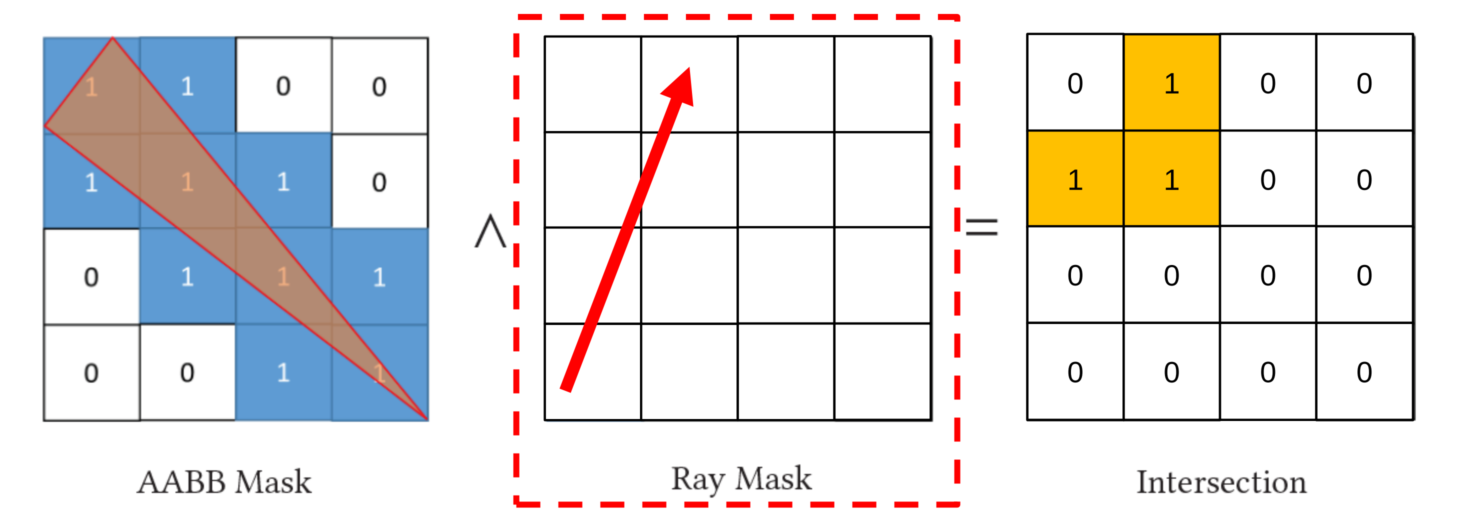
- It can fit densely to the primitive ☺
  - As long as the grid resolution is fine enough
- Simple representation
  - 1 bit per cell to represent its occupancy
- Fast Intersection with Rays
  - **DDA[1]-like iterative method?**
  - **We use LUT-based approach**
    - **O(1)**
    - **Only a few arithmetic operations**



1. Digital Differential Analyzer

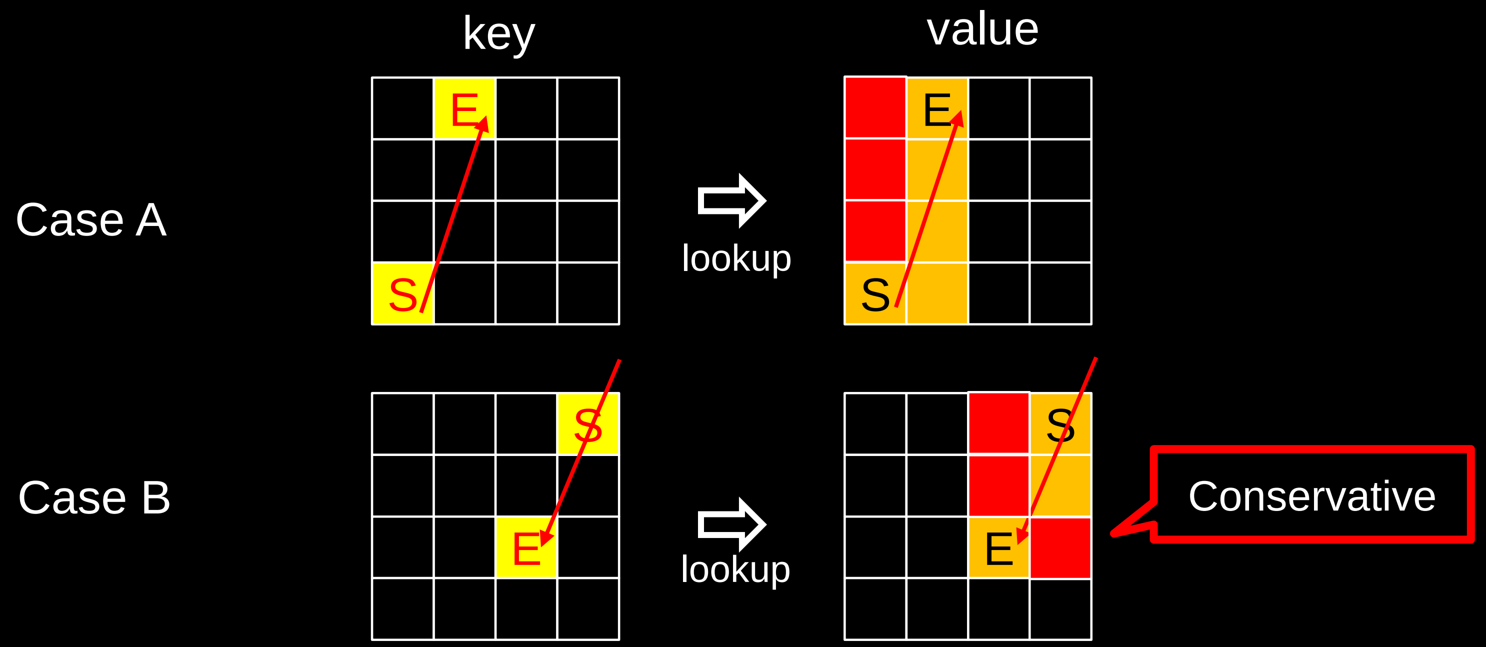
Small voxels vs Ray

- Intersection with a bitwise AND
  - Holger Gruen, “Block-Wise Linear Binary Grids for Fast Ray-Casting Operations” in GPU Pro 360
  - 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)



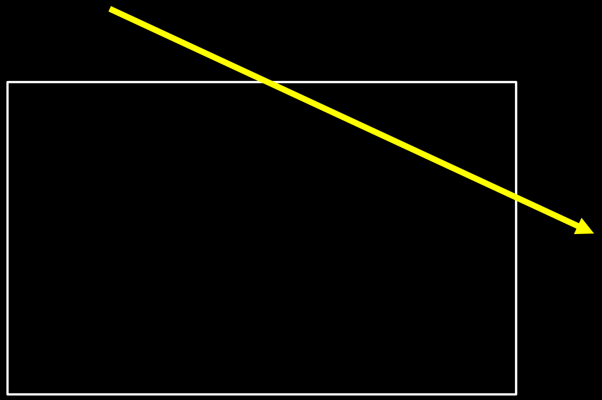
Ray Mask Build

- 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 ☺

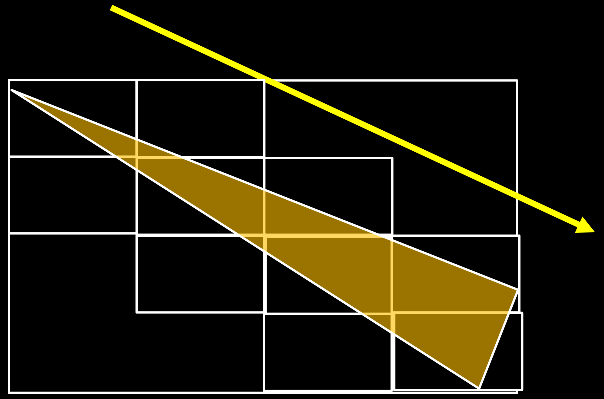


Our Approach – in a nutshell

- 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

```
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

ARR
Advanced Rendering Research Group

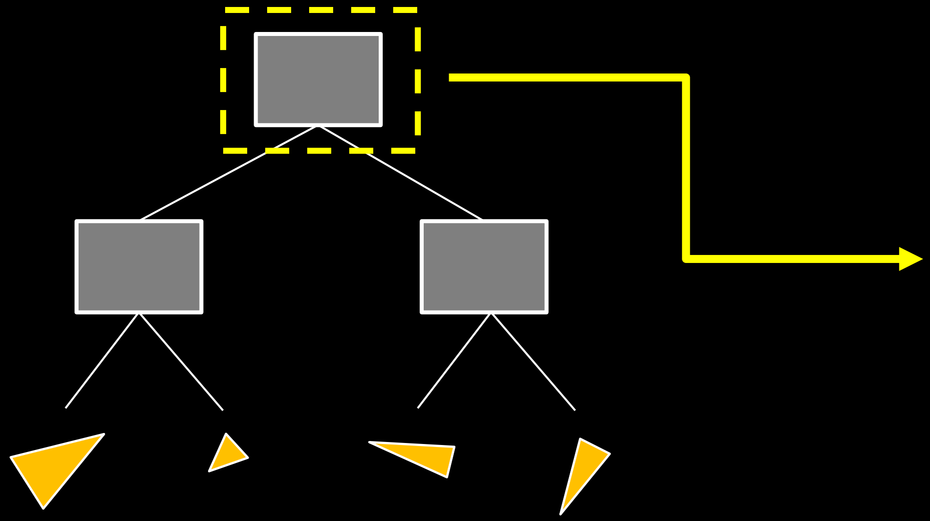
Our Traversal

```
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 hit_i ∈ hits do
            AABB, objectMask, ps ← get an AABB, an object mask, and intersections of hit_i;
            rayMask ← lookupRayMask ( AABB_lower, AABB_upper, ps_0, ps_1, R, rayMasks );
            if objectMask ∧ rayMask is not zero then
                child ← get an child node that corresponds hit_i;
                push (child);
            end
        end
    end
end
```

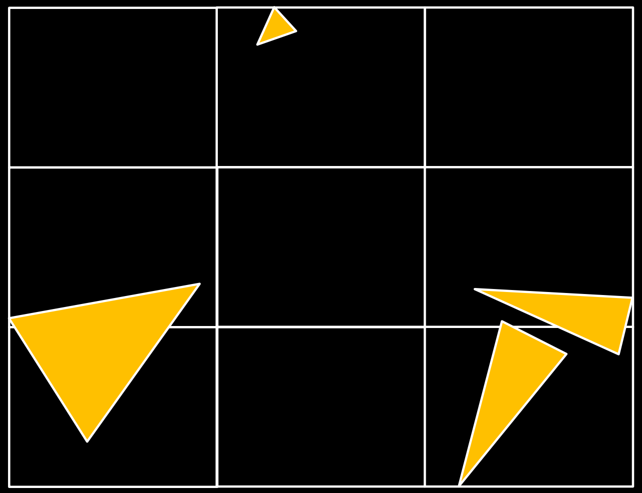

AMD

Hierarchical Construction

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



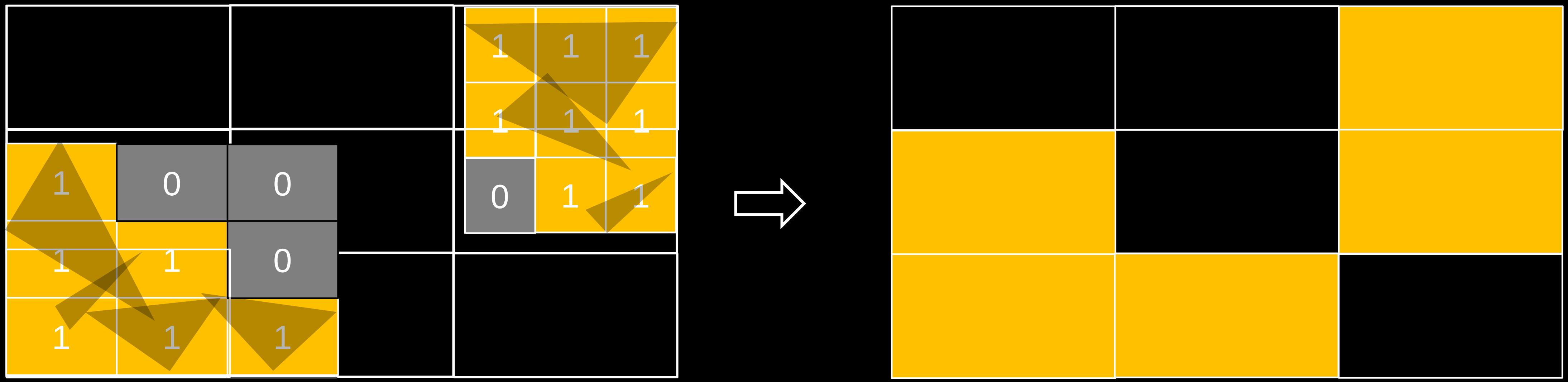
A bvh hierarchy



The voxels on the top node

Hierarchical Construction

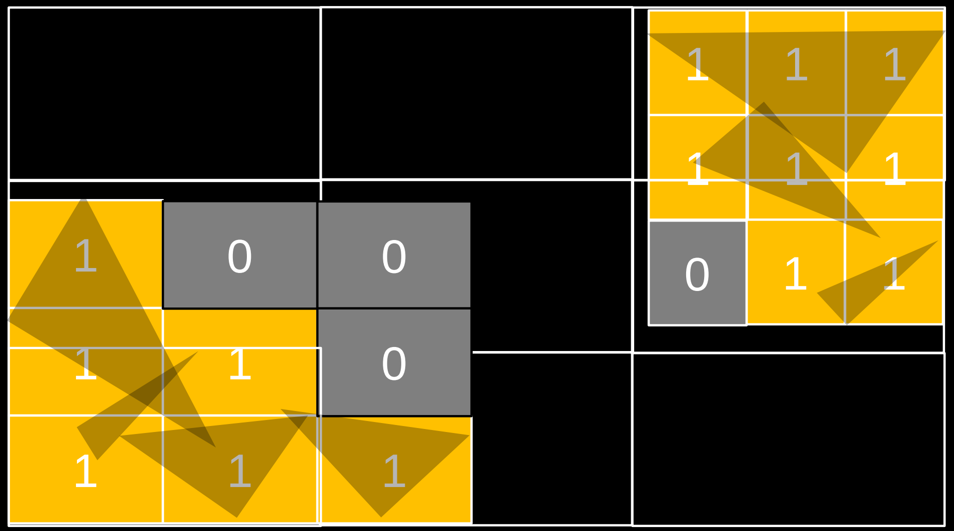
- Approximate the occupancy by voxels

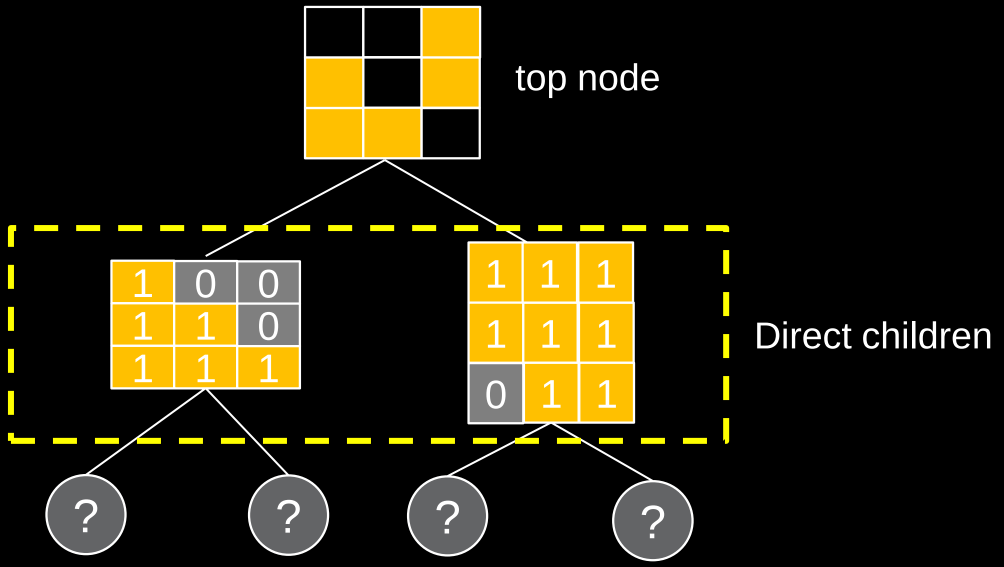


Calculated mask from the children's voxels

Hierarchical Construction

- Approximate the occupancy by voxels
  - don't have to traverse lower level of the data structure ☺





Exact Occupancy vs Approximated Occupancy

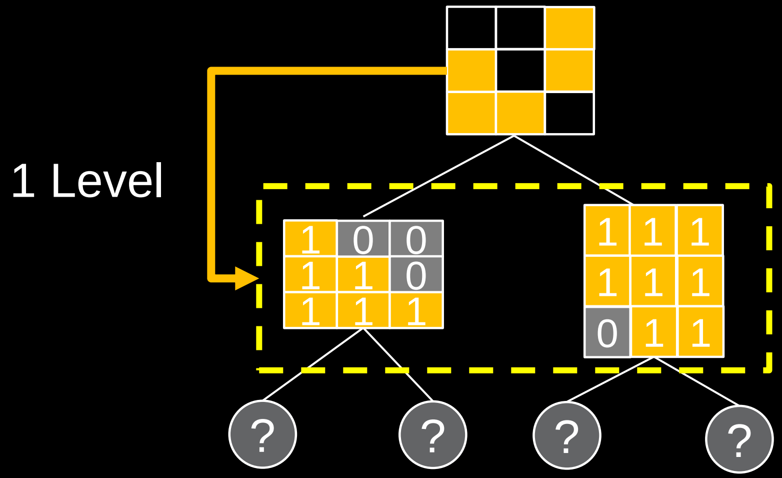
- Similar result with the approximated occupancy with lower cost ☺
- Trade-offs

Exact Occupancy

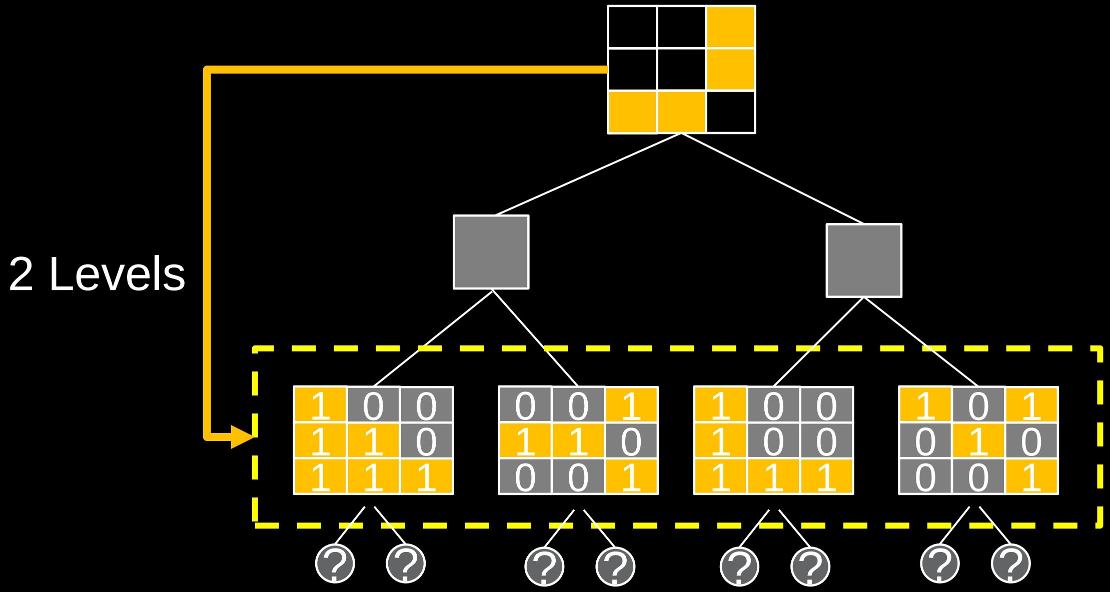
Approximated Occupancy

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

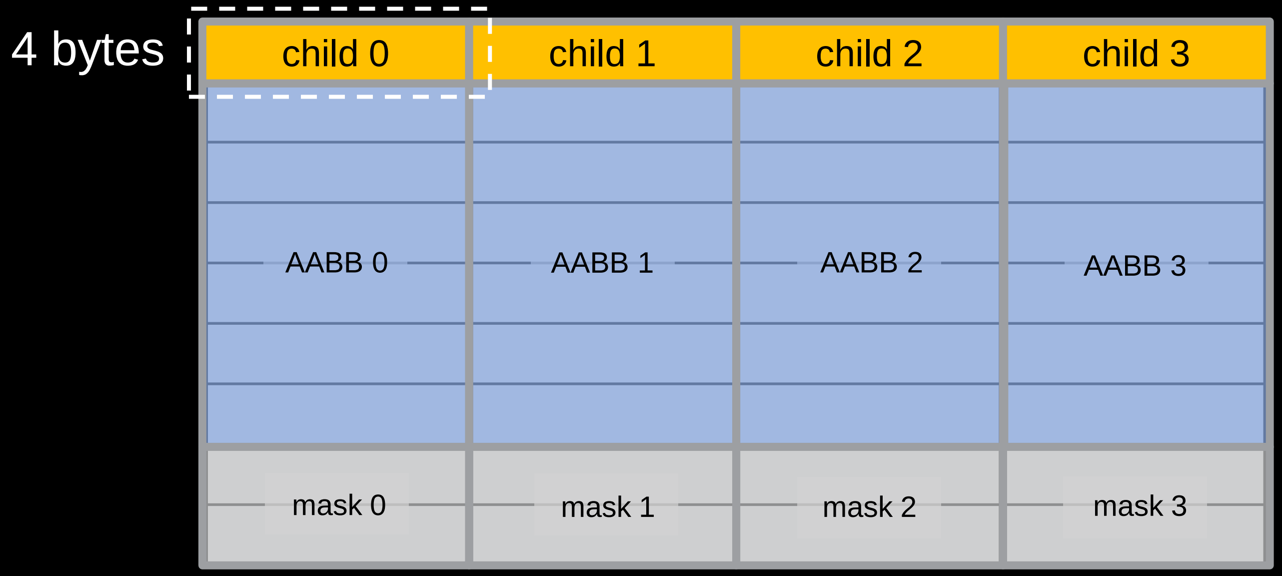


1 step more accurate, but need more nodes to visit

Voxel Compression

- How much memory do we need for the voxels?

$$4 \times 4 \times 4 = 64\ bits = 8\ bytes$$
$$6 \times 6 \times 6 = 216\ bits = 27\ bytes$$



Voxel Compression

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

⇨ Look-up table approach

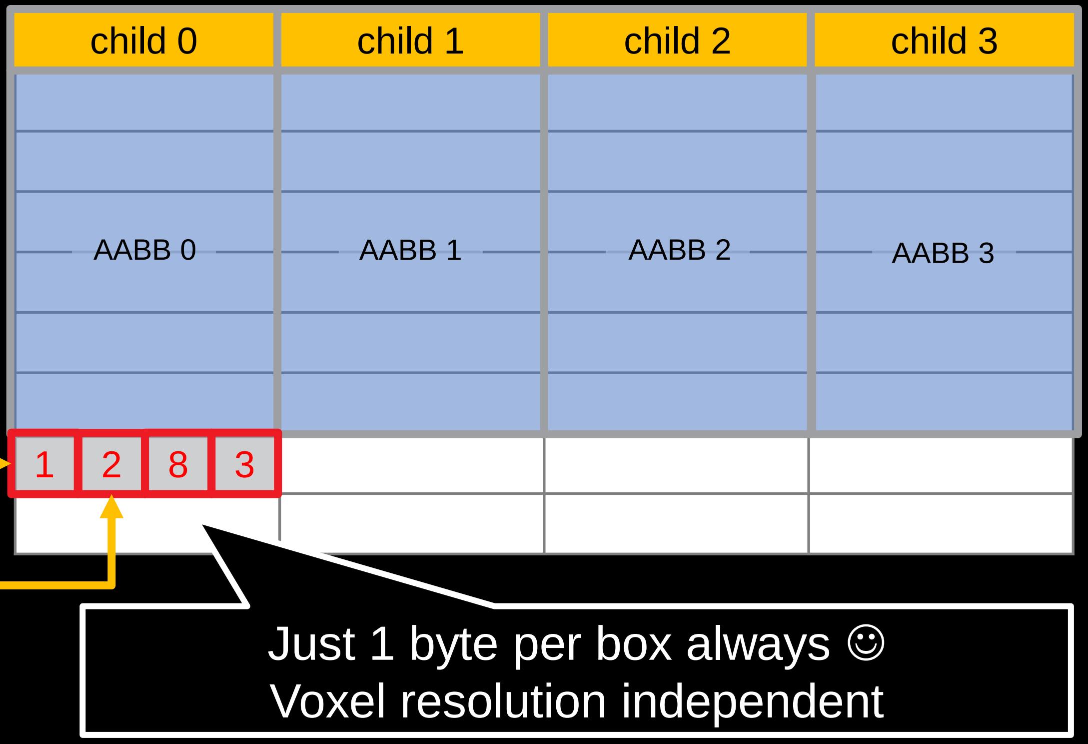
256 elements

Index	Mask
[0]	11111111
[1]	01100111
[2]	11000000
[3]	11010101
….	….

~~frequently used masks~~

Masks by SAH-based prioritization

A BVH node



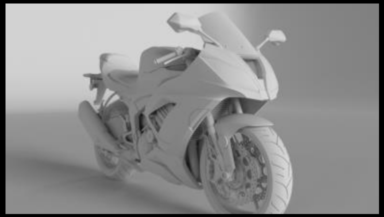
AMD

Results

- We measured:
  - Number of intersections to see culling efficiency
  - Entire rendering performance

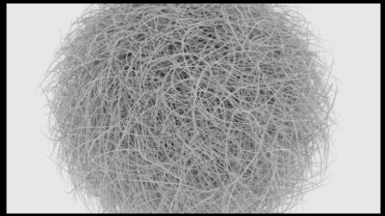

Bedroom (462.8 K tris)


San Miguel (9.9 M tris)


Ninja (1.3 M tris)


Bistro (2.8 K tris)


Classroom (606.1 K tris)


Hairball (2.8 M tris)


Curly Hair (12.1 M tris)


Classroom (7.3 M tris)


Victorian Trains (884.1 K tris)

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 ☺

Curly Hair (12.1 M tris)

Straight Hair (7.3 M tris)

Resolution = 4, Compressed Voxels

Scene	Relative Intersections
Bedroom	90.7%
San Miguel	83.2%
Ninja	90.0%
Bistro	78.2%
Classroom	88.3%
Hairball	73.2%
Curly Hair	66.3%
Straight Hair	**62.1%**
VictorianTrains	86.9%

Resolution = 6, Compressed Voxels

Scene	Ratio of Intersection
Bedroom	86.9%
San Miguel	76.5%
Ninja	87.7%
Bistro	71.5%
Classroom	85.1%
Hairball	74.3%
Curly Hair	60.4%
Straight Hair	**53.6%**
VictorianTrains	81.1%

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

Resolution = 4, Ray Mask LUT

Scene	**Relative Rendering Time ($R_{ray} = 4$)**
Bedroom	103.6%
San Miguel	97.6%
Ninja	101.0%
Bistro	94.9%
Classroom	106.0%
Hairball	97.9%
Curly Hair	86.9%
Straight Hair	88.0%
VictorianTrains	104.3%

Resolution = 6, Ray Mask LUT

Scene	**Relative Rendering Time ($R_{ray} = 6$)**
Bedroom	106.5%
San Miguel	98.2%
Ninja	104.5%
Bistro	97.3%
Classroom	107.1%
Hairball	102.0%
Curly Hair	**87.1%**
Straight Hair	**85.6%**
VictorianTrains	110.0%

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

ARR

Advanced Rendering Research Group