Game Physics

Collision Detection

CS 415: Game Development

Professor Eric Shaffer

Source: Millington, Ian. Game Physics Engine Development, Second Edition.

ILLINOIS

Collision Detection

Surprisingly complex topic

- Even a high-quality engine like Unreal can have issues

Some terminology to know:

- Intersection detection

  Do two geometric entities intersect? Typically a static problem

- Collision detection

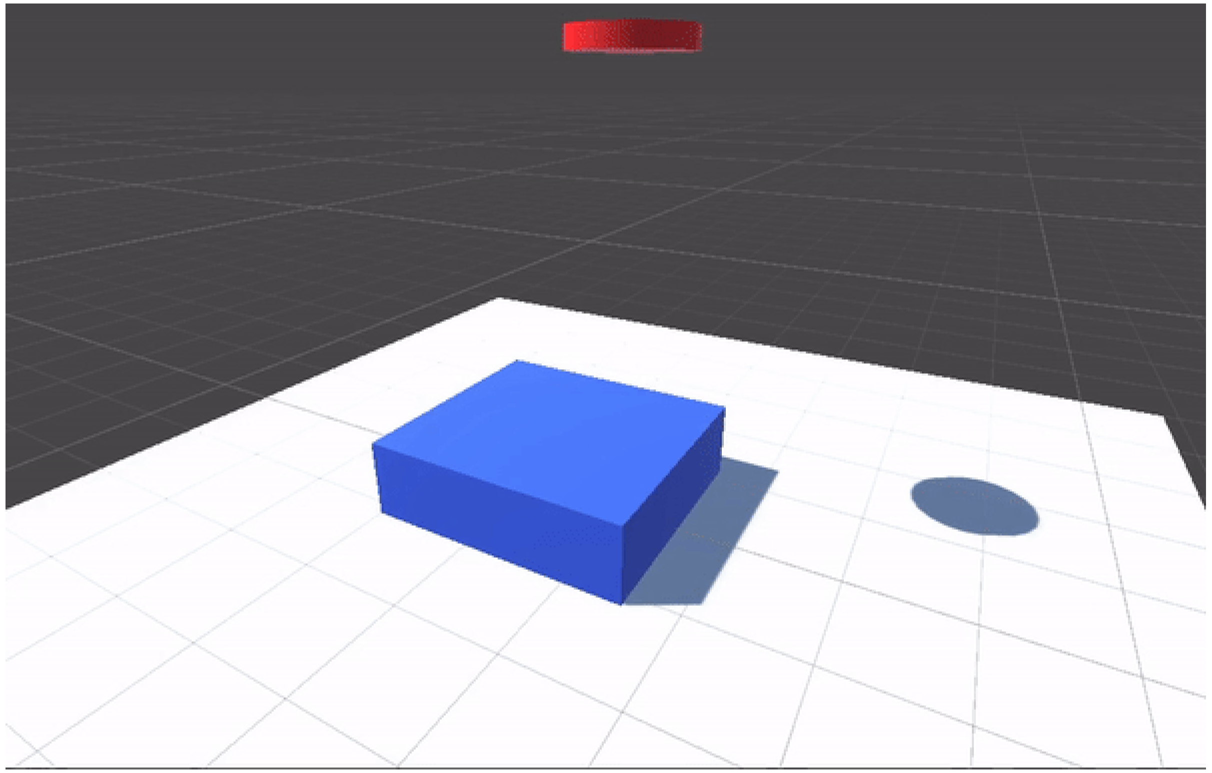
  Determine if and when moving objects collide. Dynamic problem

  Involves both time and space

Collision Detection

The need for collision detection is games is pretty obvious.
It's useful for detecting places and time when a behavior of an actor should change

- Arrow hits a target
- Car crashes
- Ball bounces off a wall



Collision Detection

Different algorithms are appropriate for different situations

- Fast moving objects
- Different kinds of geometric objects
- Large number of objects

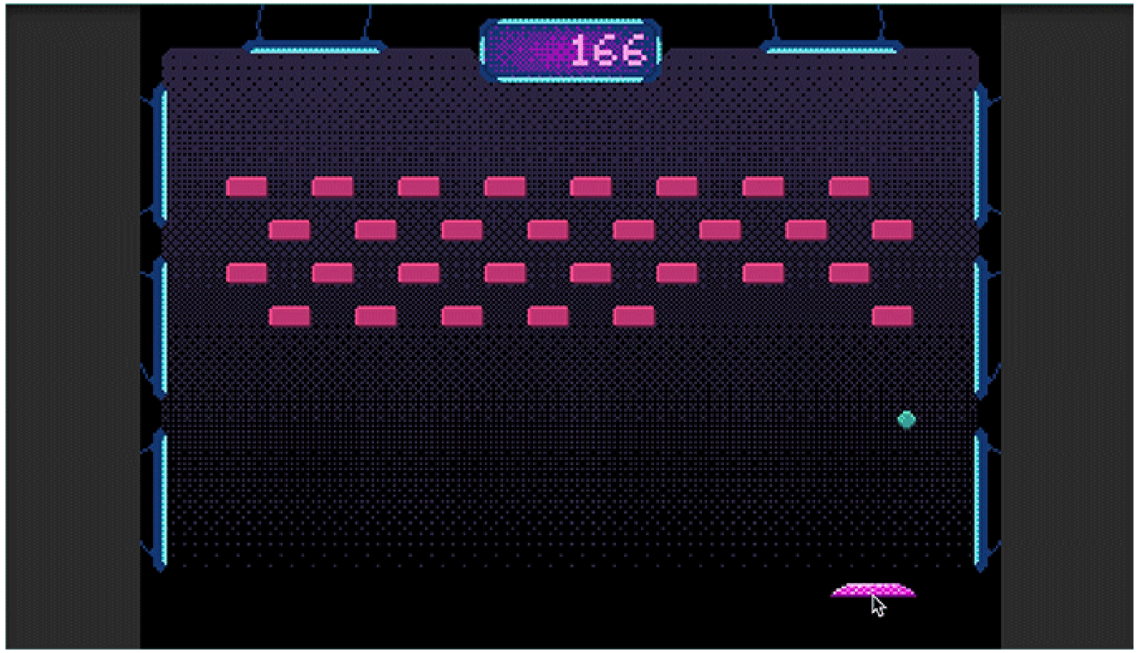
Most games will use a two-phase approach

- Broad phase identifies possible collisions quickly using a spatial data structure
- Narrow phase checks the possible collisions carefully

Spatial data structure is typically updated frame-to-frame, not rebuilt

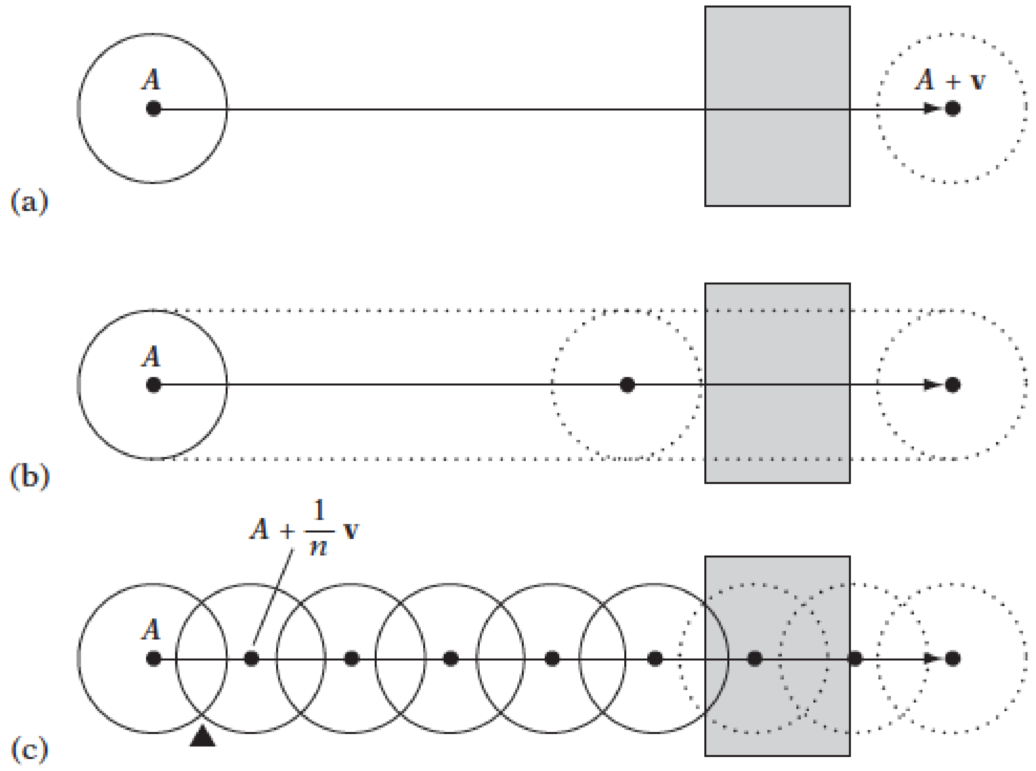
Collision Detection

Before getting to collisions using meshes…let’s look at simpler(?) problems



What happened?

Dynamic Collision Detection



Discrete collision tests can exhibit *tunneling*

- if only the final positions of the objects are tested (a)
- or even if the paths of the objects are sampled (b and c)

A **continuous collision detection method** assures detection....may not be computationally feasible.

Continuous Sphere-Plane Collision

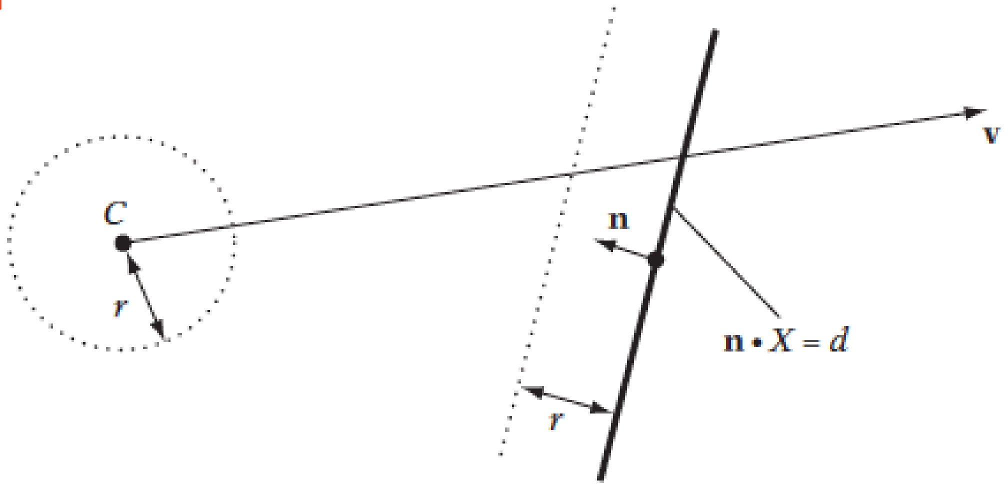
$(\mathbf{n} \cdot X) = d \pm r \Leftrightarrow$ *(plane equation for plane displaced either way)*

$\mathbf{n} \cdot (C + t\mathbf{v}) = d \pm r \Leftrightarrow$ *(substituting $S(t) = C + t\mathbf{v}$ for X)*

$(\mathbf{n} \cdot C) + t(\mathbf{n} \cdot \mathbf{v}) = d \pm r \Leftrightarrow$ *(expanding dot product)*

$t = (\pm r - ((\mathbf{n} \cdot C) - d))/(\mathbf{n} \cdot \mathbf{v})$ *(solving for t)*

Why is it $\pm r$?



Continuous Sphere-Sphere Collision Detection

The vector **d** between the sphere centers at time t is given by

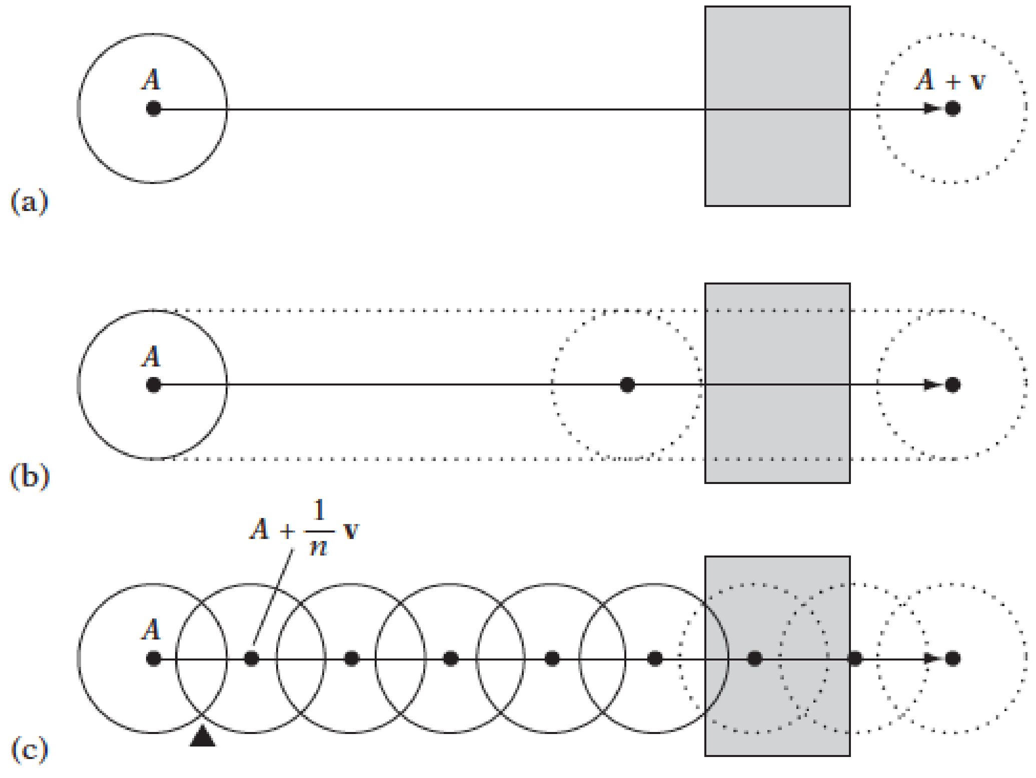
$$\mathbf{d}(t) = (C_0 + t\mathbf{v}_0) - (C_1 + t\mathbf{v}_1) = (C_0 - C_1) + t(\mathbf{v}_0 - \mathbf{v}_1)$$

$$
\begin{aligned}
&\mathbf{d}(t)\cdot\mathbf{d}(t) = (r_0 + r_1)^2 \Leftrightarrow && \textit{(original expression)}\\
&(\mathbf{s} + t\,\mathbf{v})\cdot(\mathbf{s} + t\,\mathbf{v}) = r^2 \Leftrightarrow && \textit{(substituting } \mathbf{d}(t) = \mathbf{s} + t\,\mathbf{v}\textit{)}\\
&(\mathbf{s}\cdot\mathbf{s}) + 2(\mathbf{v}\cdot\mathbf{s})t + (\mathbf{v}\cdot\mathbf{v})t^2 = r^2 \Leftrightarrow && \textit{(expanding dot product)}\\
&(\mathbf{v}\cdot\mathbf{v})t^2 + 2(\mathbf{v}\cdot\mathbf{s})t + (\mathbf{s}\cdot\mathbf{s} - r^2) = 0 && \textit{(canonic form for quadratic equation)}
\end{aligned}
$$

This is a quadratic equation in t. Writing the quadratic in the form $at^2 + 2bt + c = 0$, with $a = \mathbf{v}\cdot\mathbf{v}$, $b = \mathbf{v}\cdot\mathbf{s}$, and $c = \mathbf{s}\cdot\mathbf{s} - r^2$ gives the solutions for t as

$$t = \frac{-b \pm \sqrt{b^2 - ac}}{a}.$$

Dynamic Collision Detection



If objects are not moving fast

Little change in position from frame to frame

Discrete collision can be used

Can use continuous for specific fast moving objects

Both Unreal and Unity support this approach

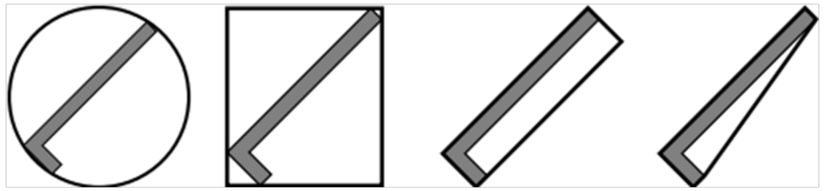
Developers manually flag fast moving objects

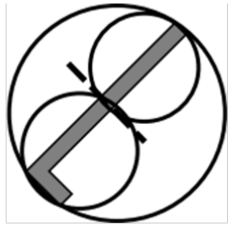
Using Bounding Volumes for Acceleration

Most objects in a scene will not be colliding

Quick rejection tests can determine if two objects are too distant to collide

Use bounding volume intersection as a quick test e.g. test if two bounding spheres intersect

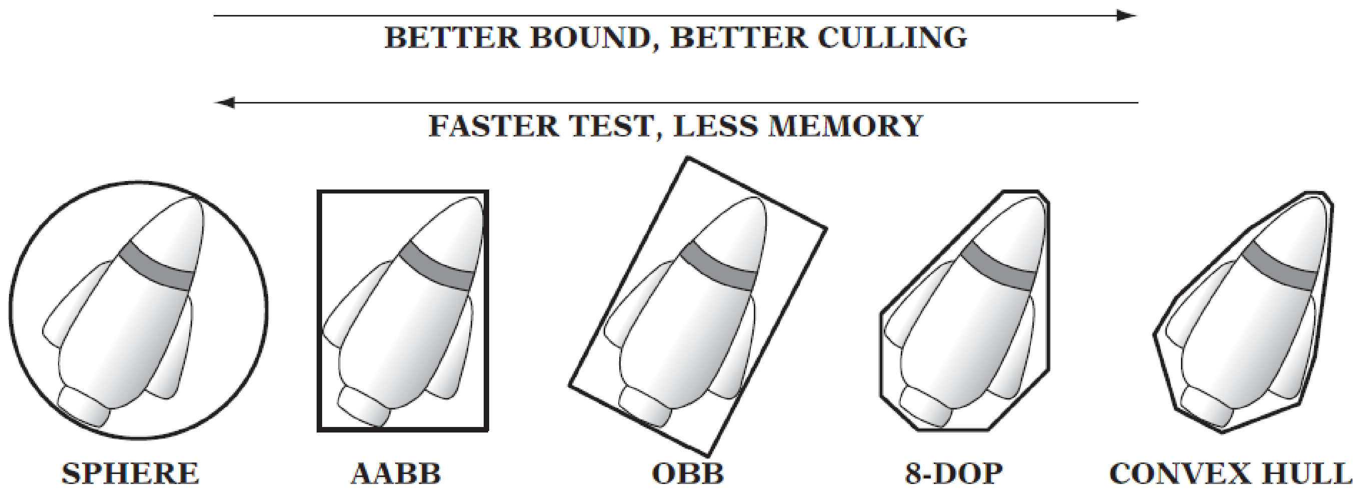




Sphere AABB OOBB Convex Hull Hierarchical (BVH)

AABB=Axis Aligned Bounding Box
OOBB = Object-Oriented Bounding Box
BVH = Bounding Volume Hierarchy

Bounding Volumes: Comparison



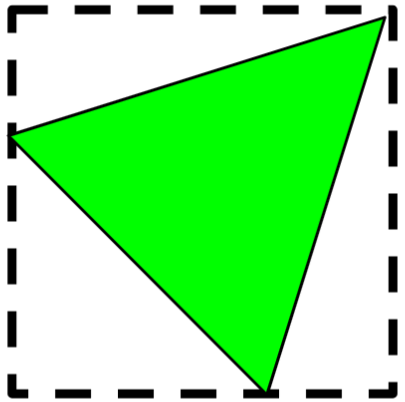
Which to use depends on several factors:

- How much computation is needed to build the BV?
- How much computation is needed to check for BV intersections
- How many false positive collisions do the BVs generate?

AABBs are popular

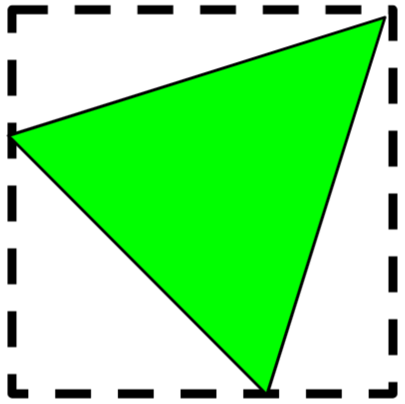
How Can We Build an AABB for a Mesh?

```
// region R = { (x, y, z) | min.x<=x<=max.x, min.y<=y<=max.y, min.z<=z<=max.z }
struct AABB {
   Point min;
   Point max;
};
```



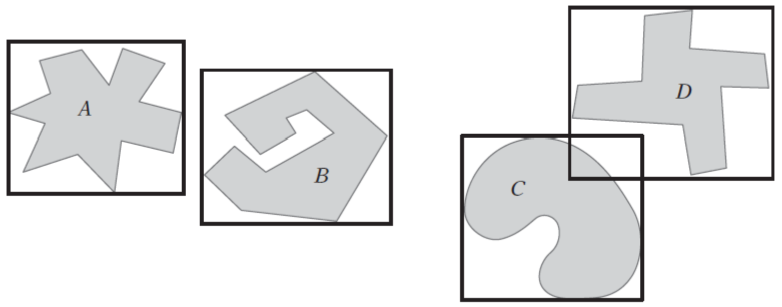
How Can We Build an AABB for a Mesh?

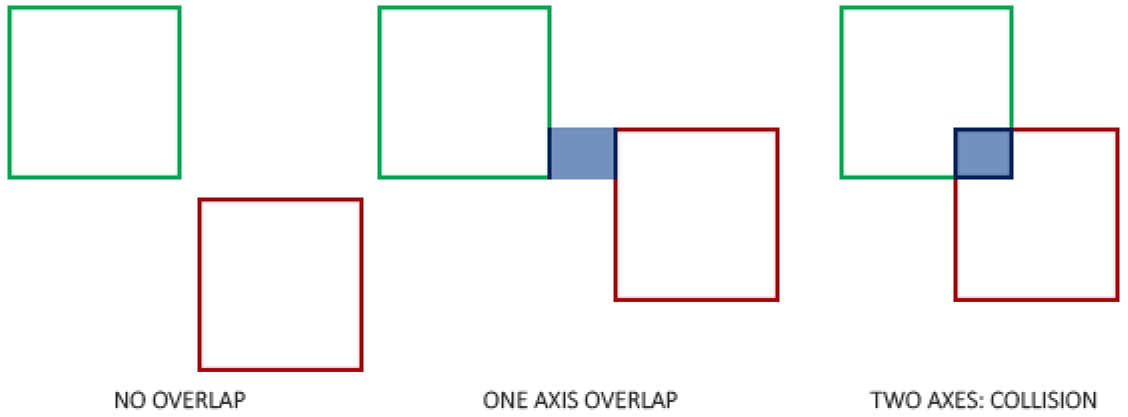
```
// region R = { (x, y, z) | min.x<=x<=max.x, min.y<=y<=max.y, min.z<=z<=max.z }
struct AABB {
   Point min;
   Point max;
};
```



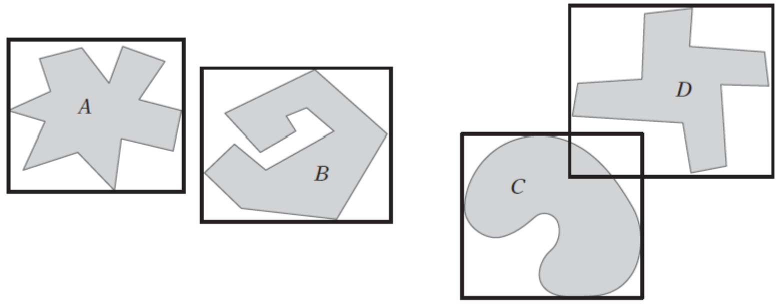
```
// Returns indices imin and imax into pt[] array of the least and
// most, respectively, distant points along the direction dir
void ExtremePointsAlongDirection(Vector dir, Point pt[], int n, int *imin, int *imax)
{
    float minproj = FLT_MAX, maxproj = -FLT_MAX;
    for (int i = 0; i < n; i++) {
        // Project vector from origin to point onto direction vector
        float proj = Dot(pt[i], dir);
        // Keep track of least distant point along direction vector
        if (proj < minproj) {
            minproj = proj;
            *imin = i;
        }
        // Keep track of most distant point along direction vector
        if (proj > maxproj) {
            maxproj = proj;
            *imax = i;
        }
    }
}
```

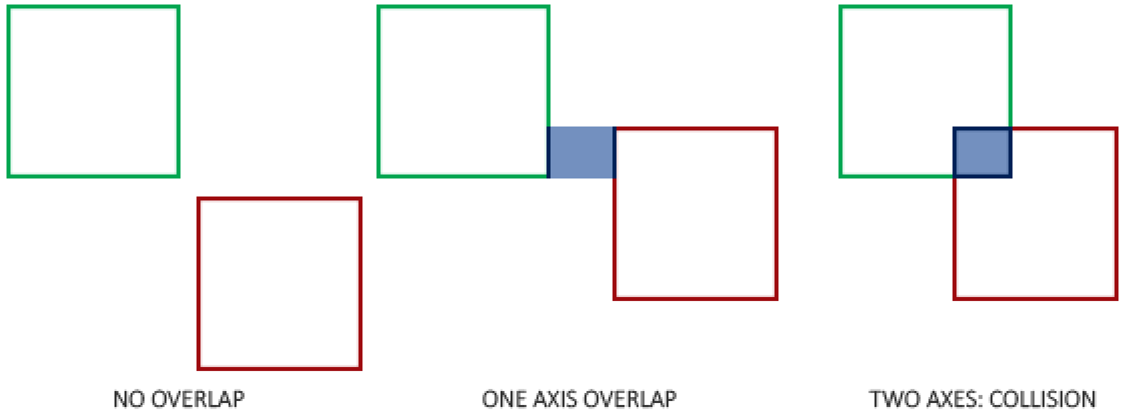
Static AABB Intersection





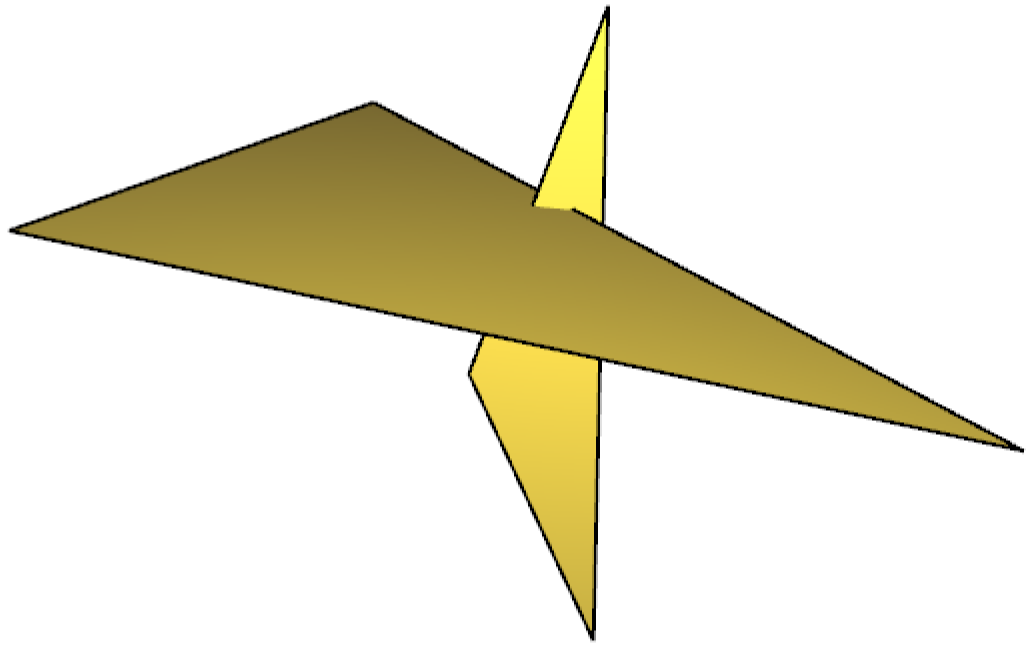
Static AABB Intersection





```
int TestAABBAABB(AABB a, AABB b)
{
    // Exit with no intersection if separated along an axis
    if (a.max[0] < b.min[0] || a.min[0] > b.max[0]) return 0;
    if (a.max[1] < b.min[1] || a.min[1] > b.max[1]) return 0;
    if (a.max[2] < b.min[2] || a.min[2] > b.max[2]) return 0;
    // Overlapping on all axes means AABBs are intersecting
    return 1;
}
```

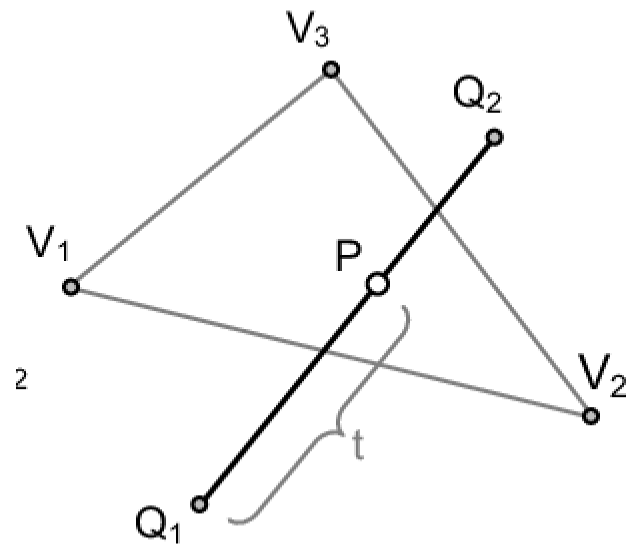
Static Triangle-Triangle Intersection

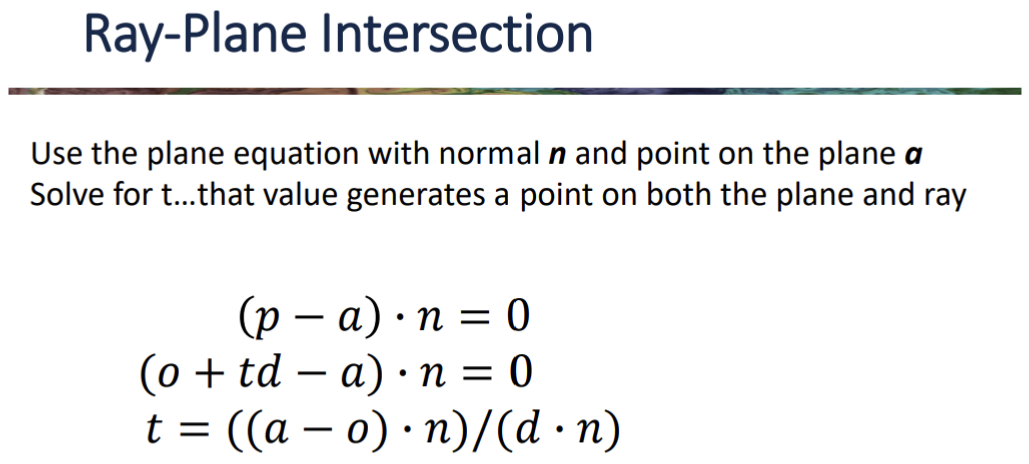


ILLINOIS

Static Triangle-Triangle Intersection

Test to see if each edge of each triangle intersects the other triangle

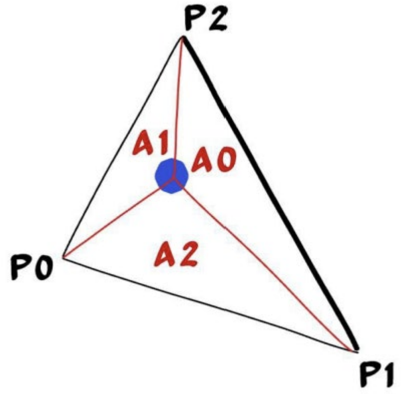




Ray-Plane Intersection

Use the plane equation with normal ***n*** and point on the plane ***a***
Solve for t…that value generates a point on both the plane and ray

$$(p-a)\cdot n = 0$$
$$(o+td-a)\cdot n = 0$$
$$t = ((a-o)\cdot n)/(d\cdot n)$$

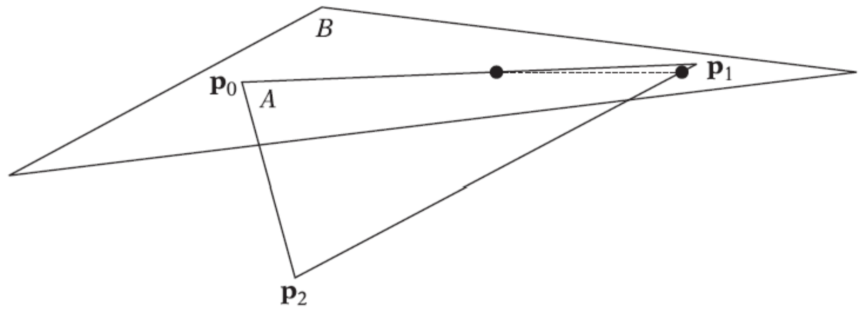


Static Triangle-Triangle Intersection

Actual test would be optimized…code would be written specifically for this test can not use ray-triangle

e.g. might test endpoints of the edge to see if they are on different sides of the triangle

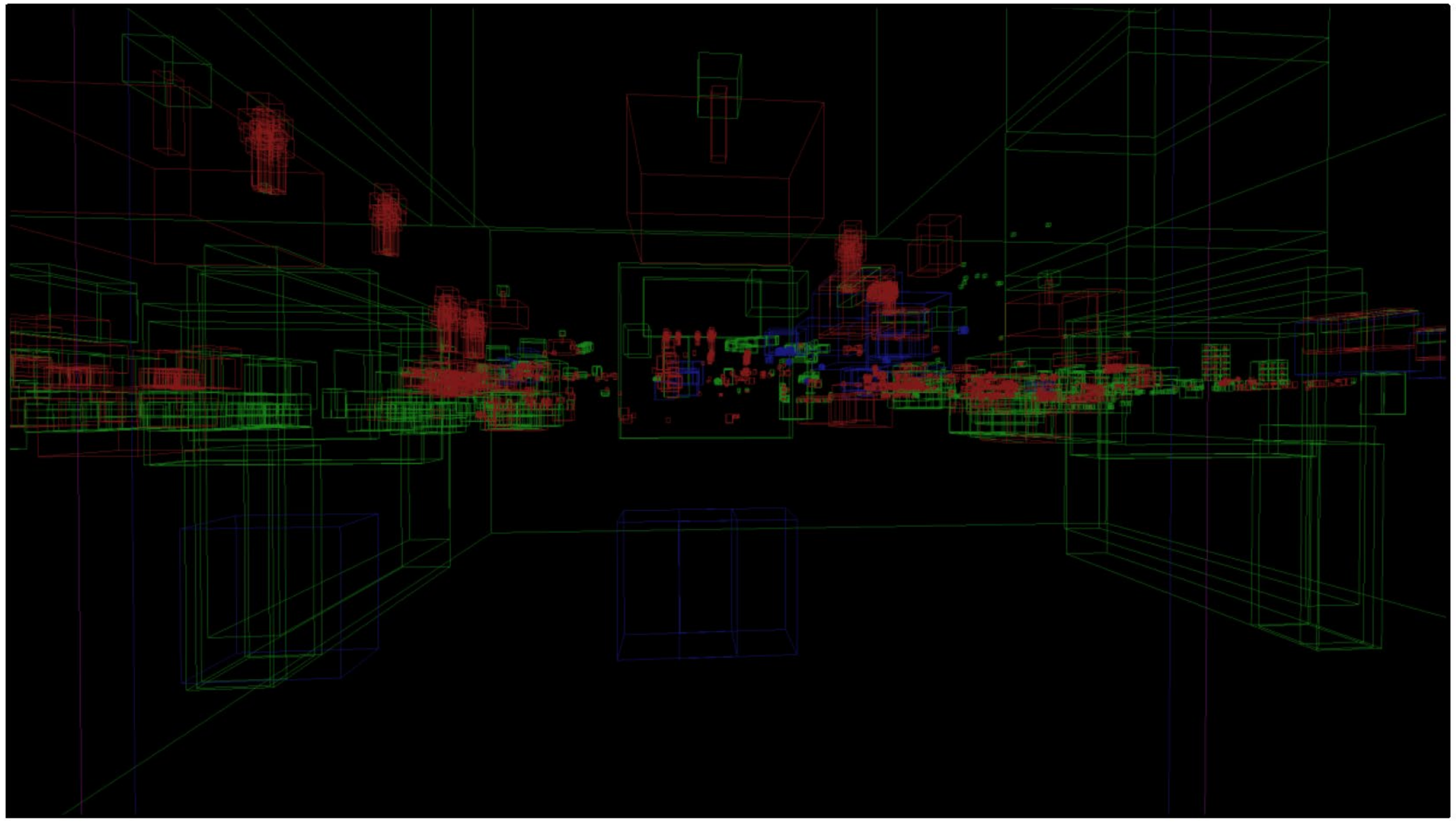
Also need to be careful about numerical precision and special cases like co-planar triangles



The edge $\overline{\mathbf{p}_0\mathbf{p}_1}$ is almost parallel to triangle B's plane, and thus is regarded as nonintersecting by the finite-precision ray-triangle test. The only intersecting edge we find is $\overline{\mathbf{p}_1\mathbf{p}_2}$.

Bounding Volumes Are Not Enough

Still don't want to check all pair-wise possible bounding volume intersections



There is almost 9000 separate collision objects in the editor. Green boxes are static objects, blue kinematic, and red dynamic.

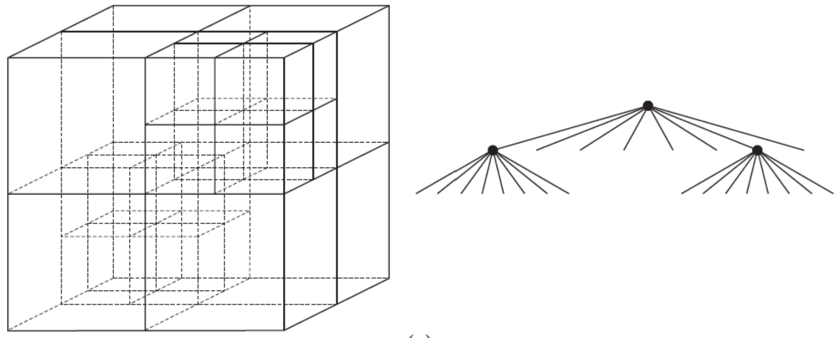
Scene from Overwatch
…a Blizzard game

ILLINOIS

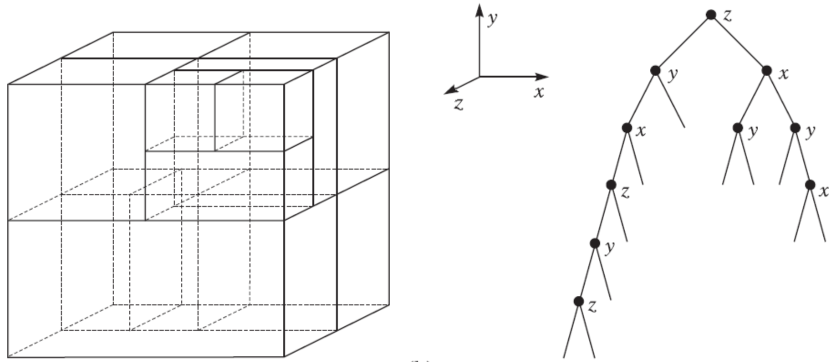
More Acceleration With Spatial Data Structures

Space Partitioning

A space partitioning is a subdivision of space into convex regions, called *cells*. Each cell in the partitioning maintains a list of references to objects that are (partially) contained in the cell. Using such a structure, a lot of object pairs can be quickly rejected from intersection testing, since we only need to test the pairs of objects that share a cell



(a)



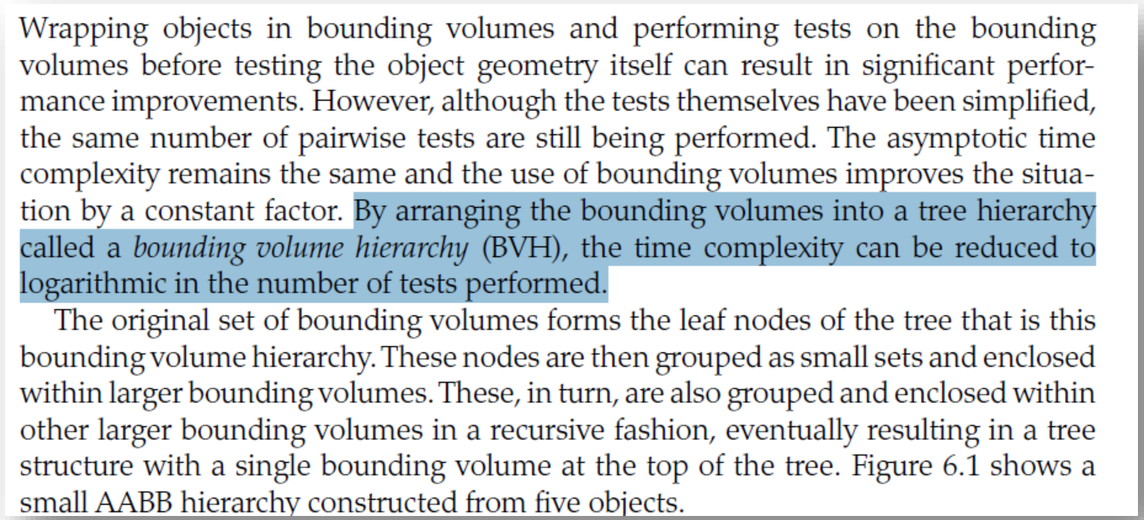
(b)

BSP Trees are popular

Can anyone name some others?

…we won’t be discussing these…instead we will talk ab out Bounding Volume Hierarchies

Bounding Volume Hierarchies



Wrapping objects in bounding volumes and performing tests on the bounding volumes before testing the object geometry itself can result in significant performance improvements. However, although the tests themselves have been simplified, the same number of pairwise tests are still being performed. The asymptotic time complexity remains the same and the use of bounding volumes improves the situation by a constant factor. By arranging the bounding volumes into a tree hierarchy called a *bounding volume hierarchy* (BVH), the time complexity can be reduced to logarithmic in the number of tests performed.

The original set of bounding volumes forms the leaf nodes of the tree that is this bounding volume hierarchy. These nodes are then grouped as small sets and enclosed within larger bounding volumes. These, in turn, are also grouped and enclosed within other larger bounding volumes in a recursive fashion, eventually resulting in a tree structure with a single bounding volume at the top of the tree. Figure 6.1 shows a small AABB hierarchy constructed from five objects.

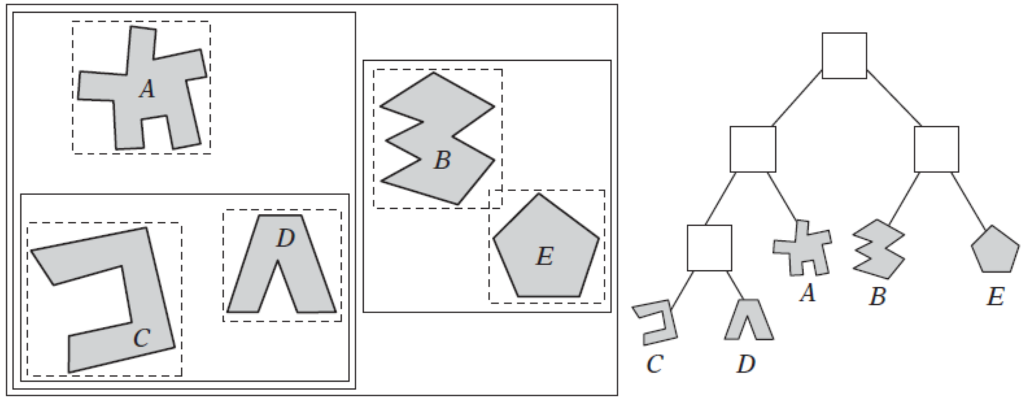


Figure 6.1 A bounding volume hierarchy of five simple objects. Here the bounding volumes used are AABBs.

A BVH does not partition space

Cells can overlap

Construction algorithms for BVHs try to limit the number of overlaps

Desired BVH Characteristics

- The nodes contained in any given subtree should be near each other.
  The lower down the tree the nodes are the nearer they should be to each other

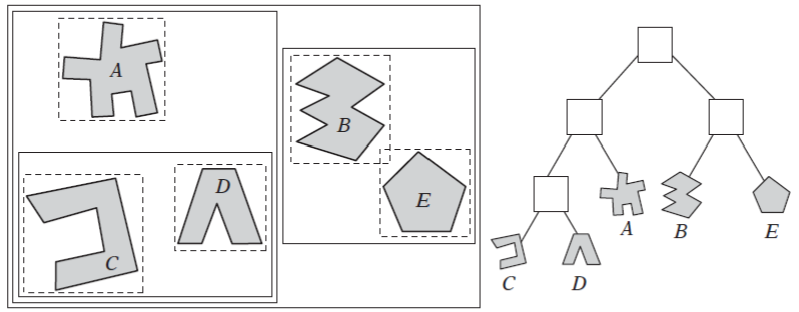
- Each node in the hierarchy should be of minimal volume

- Greater attention should be paid to nodes near the root of the hierarchy.
  Pruning a node near the root of the tree removes more objects from further consideration than removal of a deeper node would.

- The volume of overlap of sibling nodes should be minimal

- The hierarchy should be balanced with respect to both its node structure and its content

BVH Construction

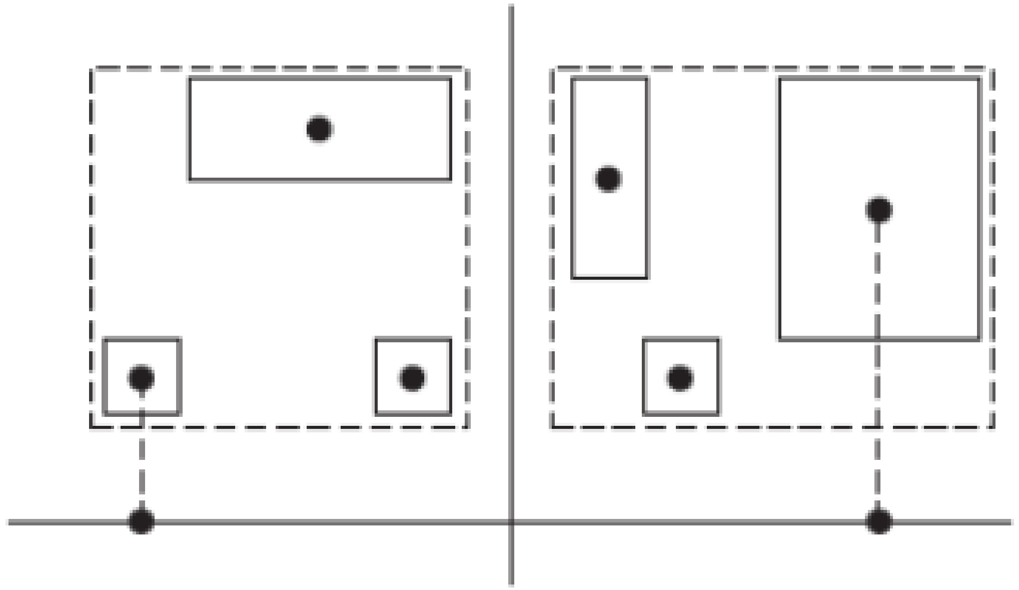


Can be constructed top down:

1. Compute bounding volume enclosing all of the geometry
2. Sort the geometry into two or more groups
3. Compute bounding volume for each group
4. Recurse
5. Leaf nodes will enclose only one(?) geometric primitive

Can also be built by bottom up merging which offers better parallelism

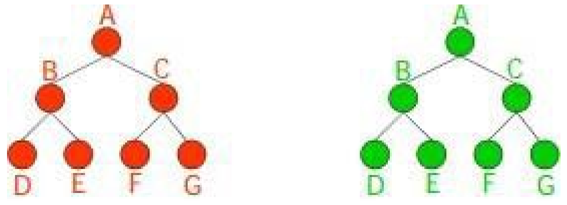
BVH: How to Sort Objectx



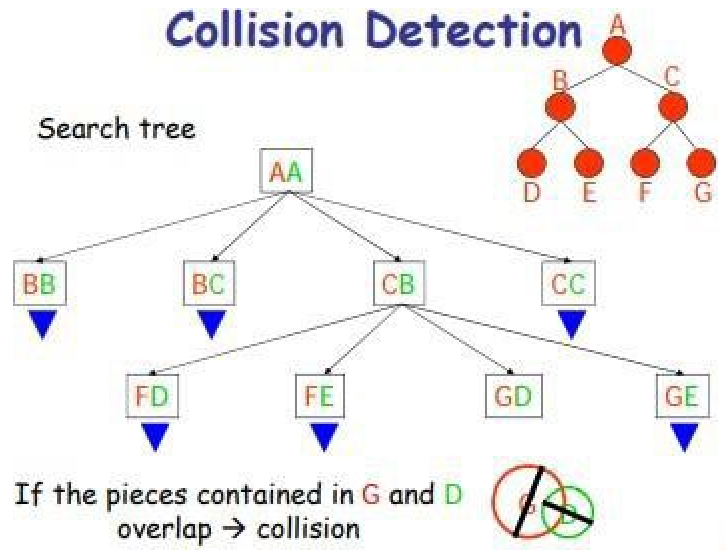
- Can compute a centroid for each geometric primitive
  - Split on median centroid, along longest axis
  - Split on average centroid, along longest axis
- More sophisticated splitting criteria can be used
  - E.g. Surface Area Heuristic used on BVHs for ray-tracing

BVH: How to Collide

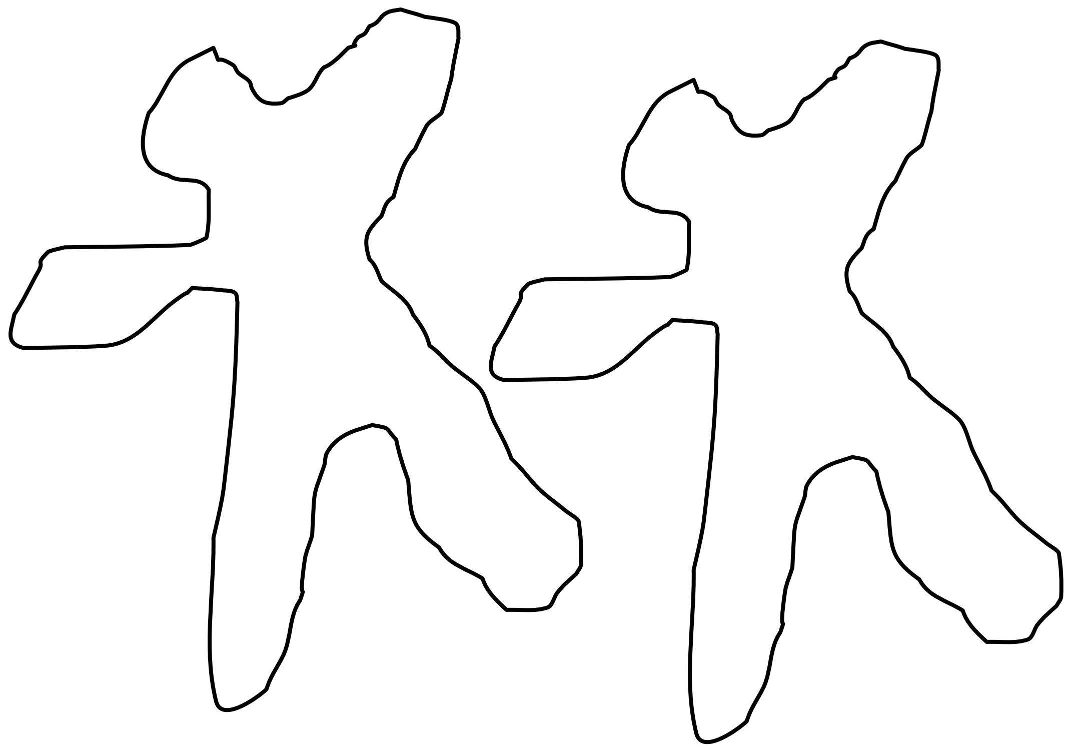
- In a single BVH for scene
  - Two geometric primitives can overlap only if their volumes overlap
- Or, BVHs can be used for each composite object (e.g. mesh)
  - A search tree is constructed that records descent into each BVH
  - Determines if any cells overlap



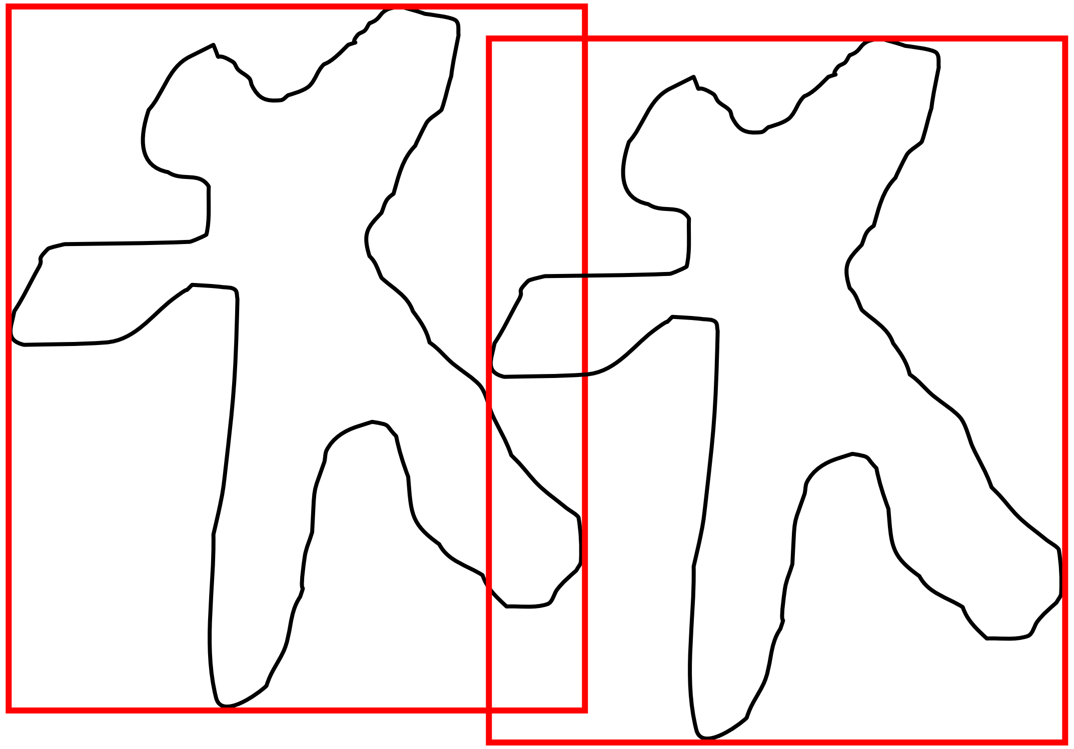
Two objects described by their precomputed BVHs



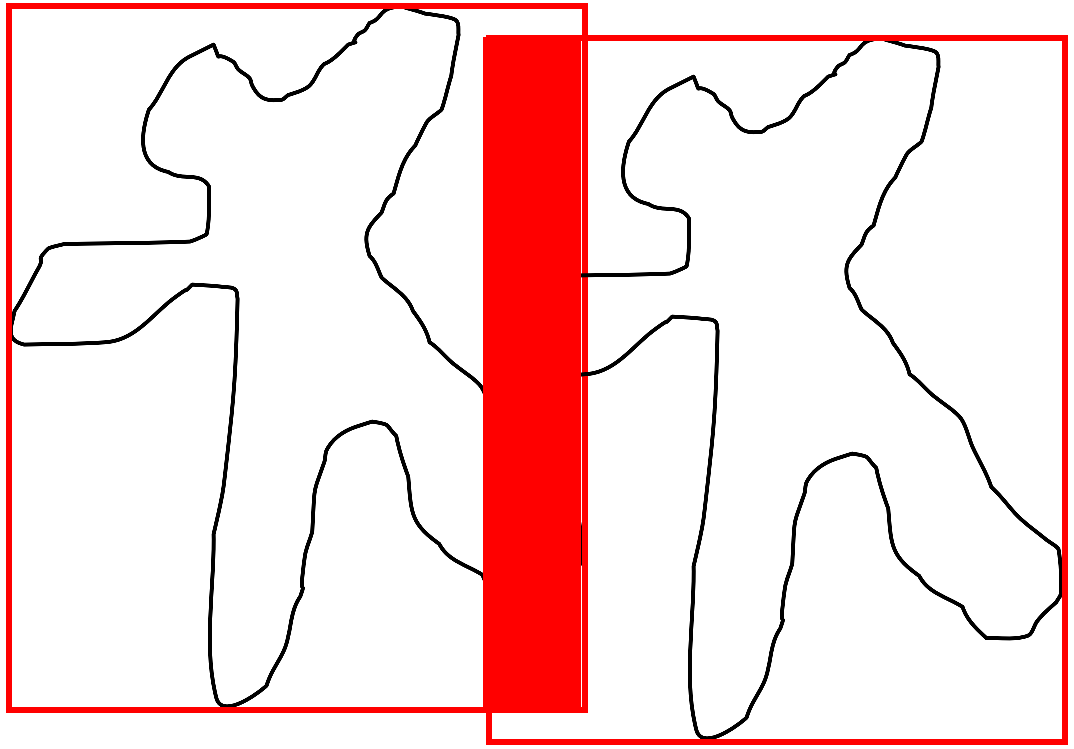
BVH Collision Test example



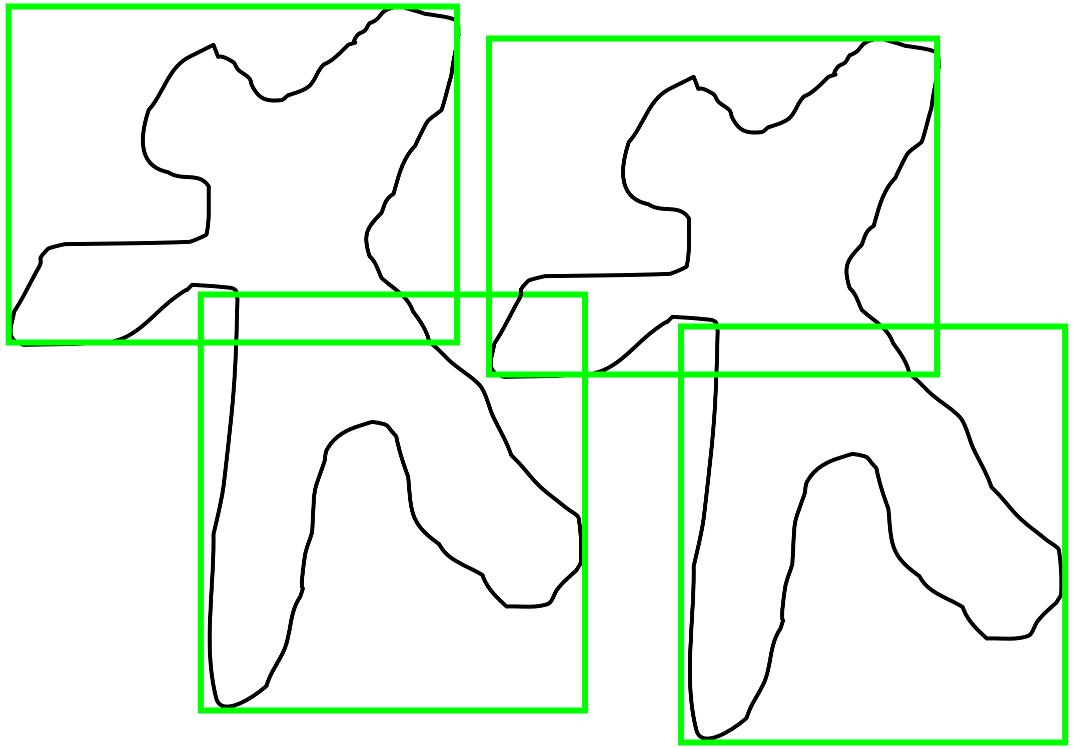
BVH Collision Test example



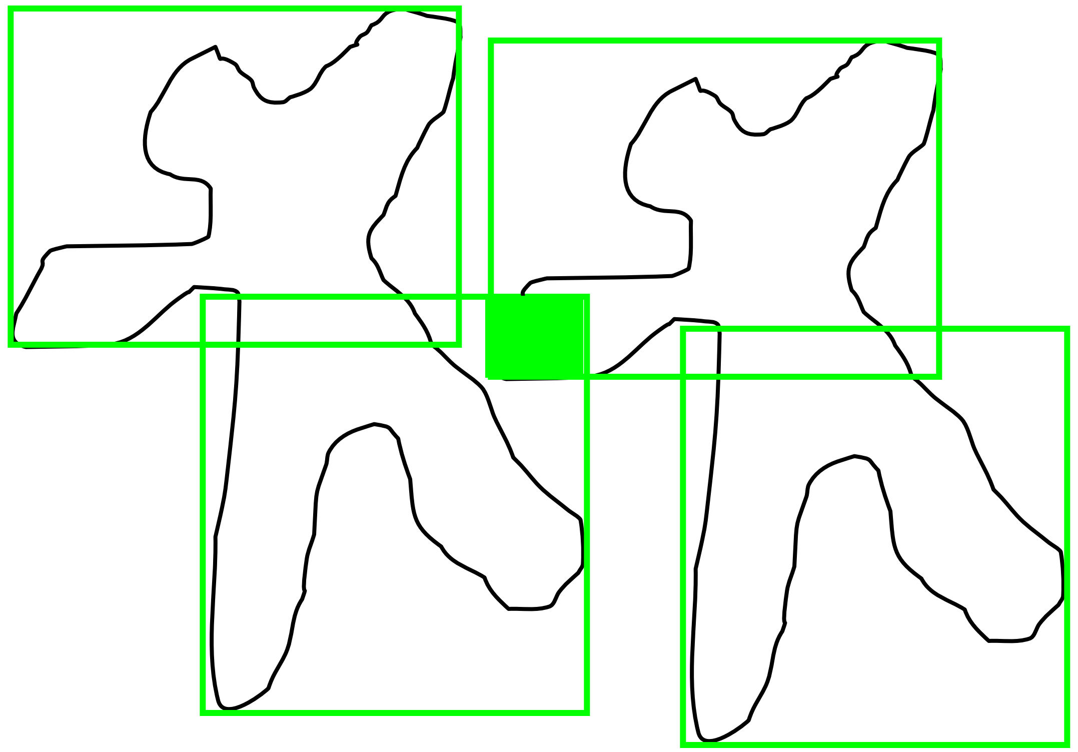
BVH Collision Test example



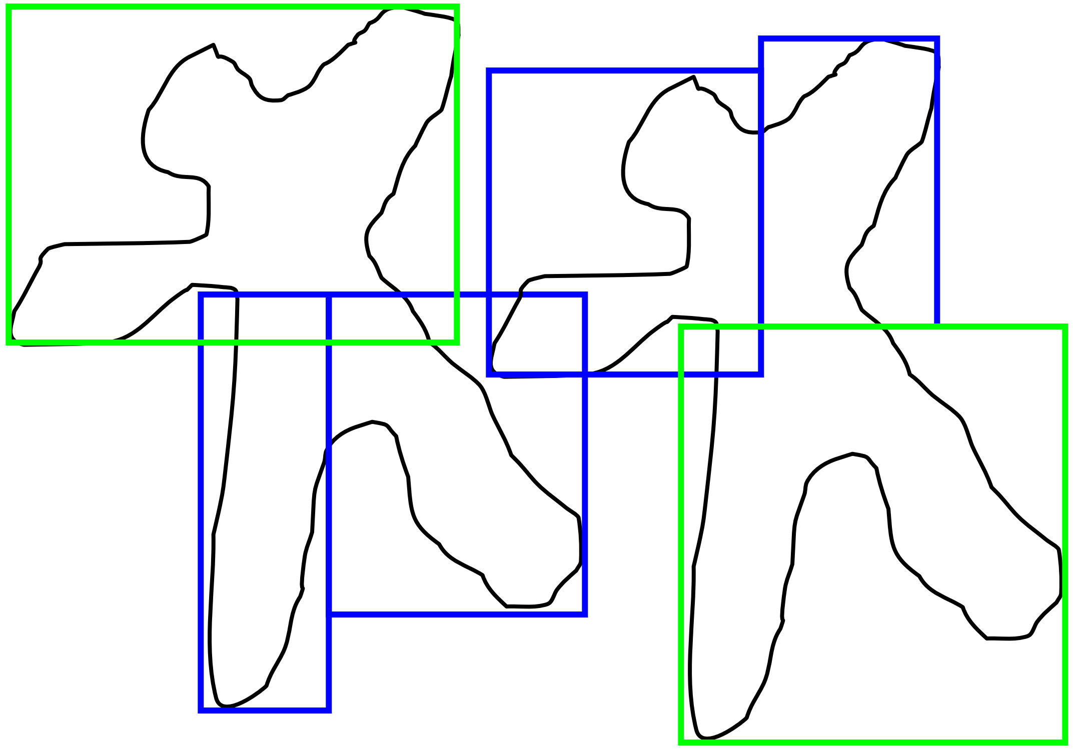
BVH Collision Test example



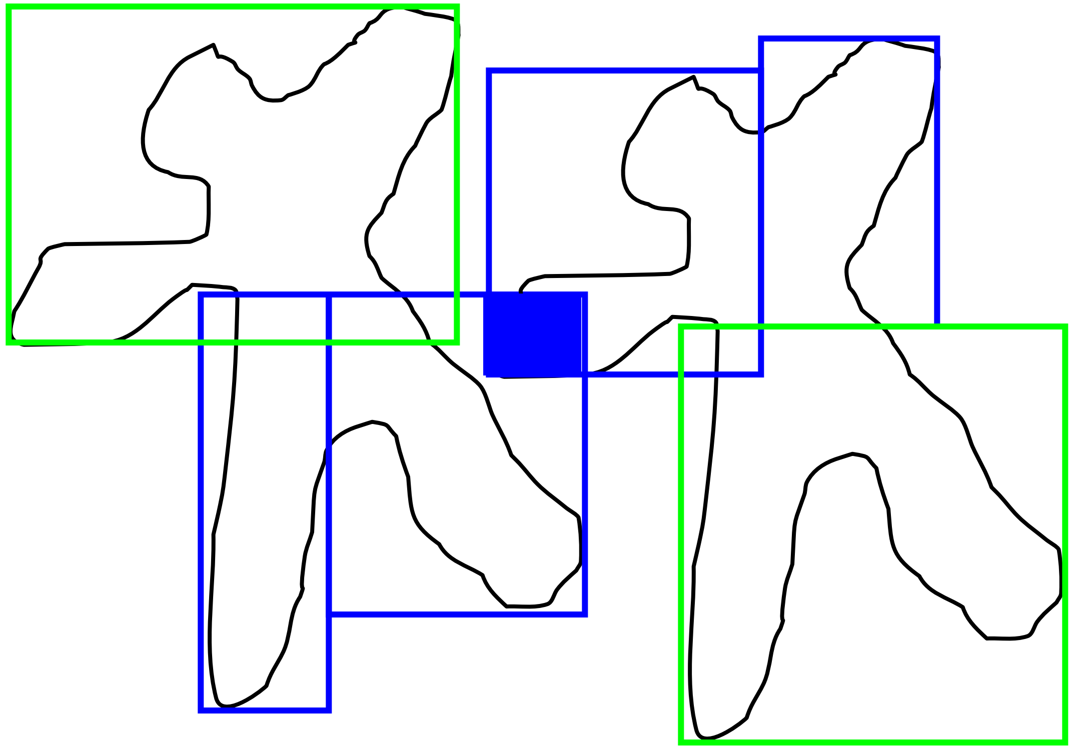
BVH Collision Test example



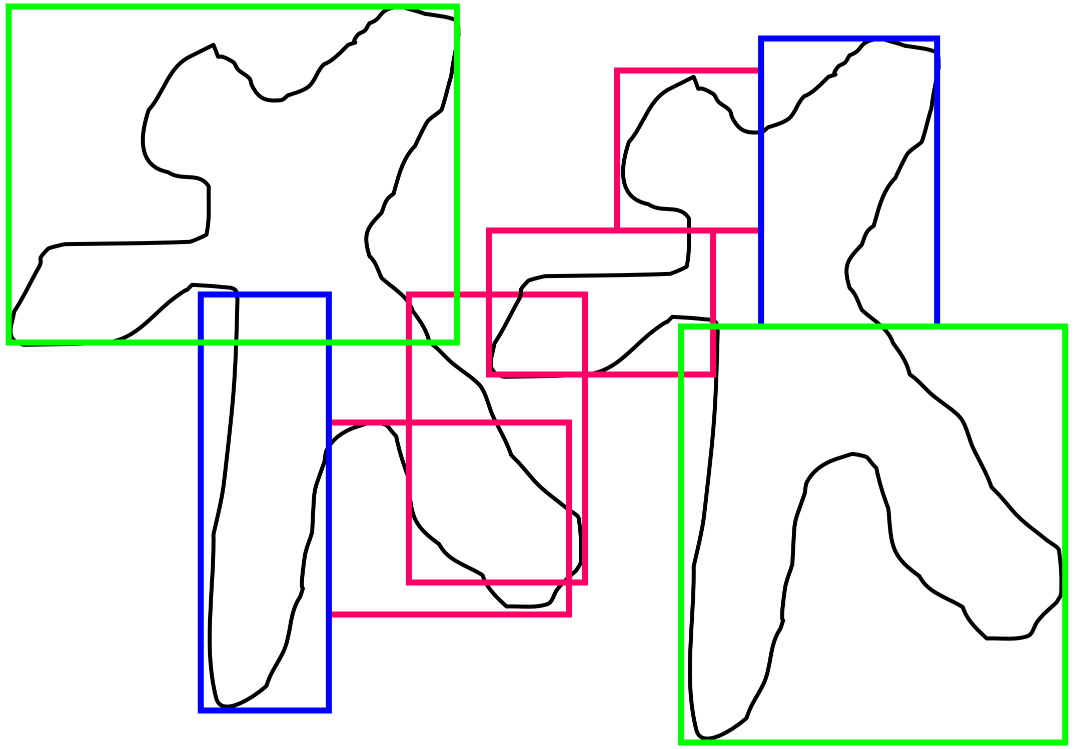
BVH Collision Test example



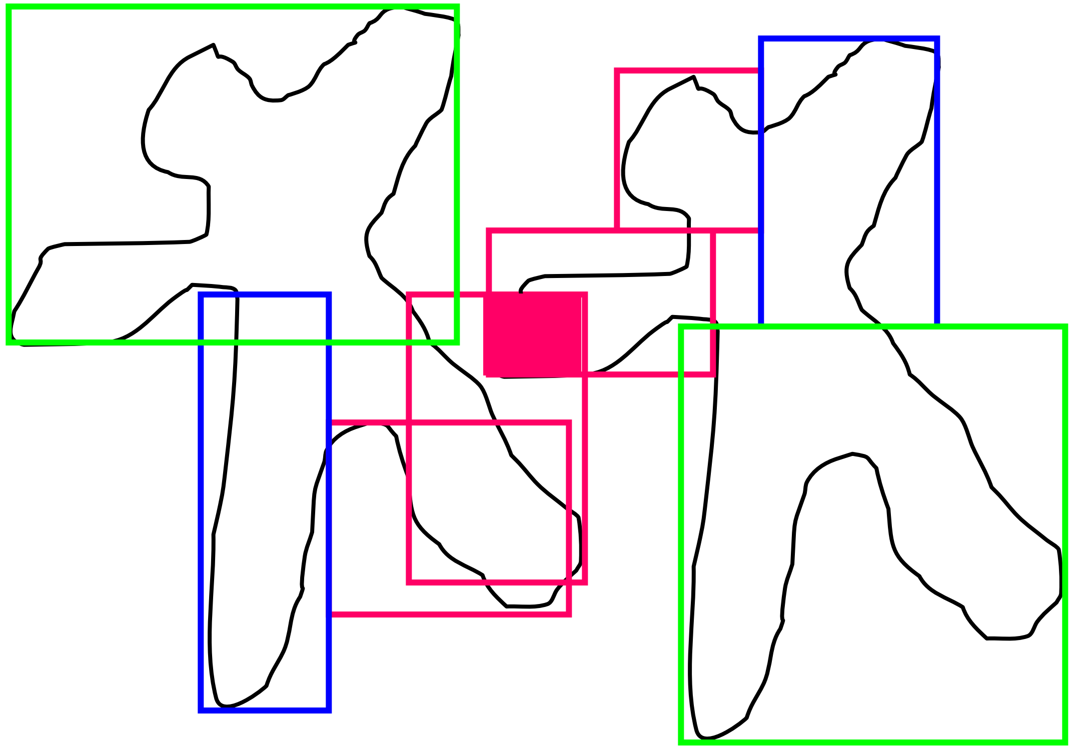
BVH Collision Test example



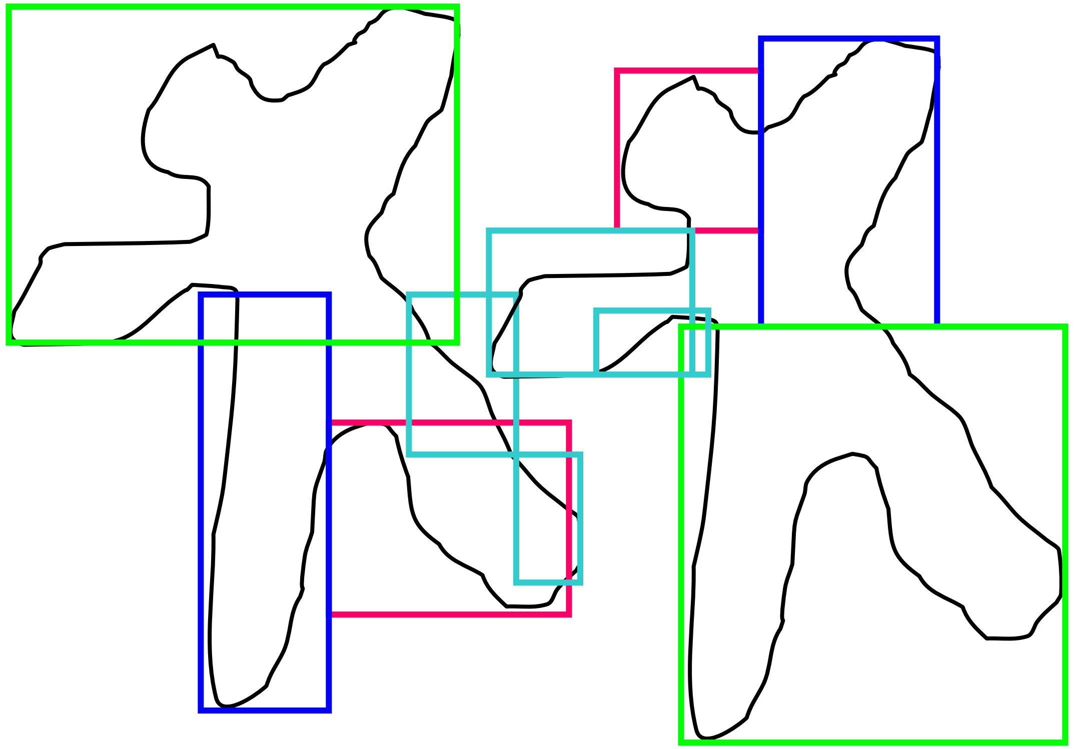
BVH Collision Test example



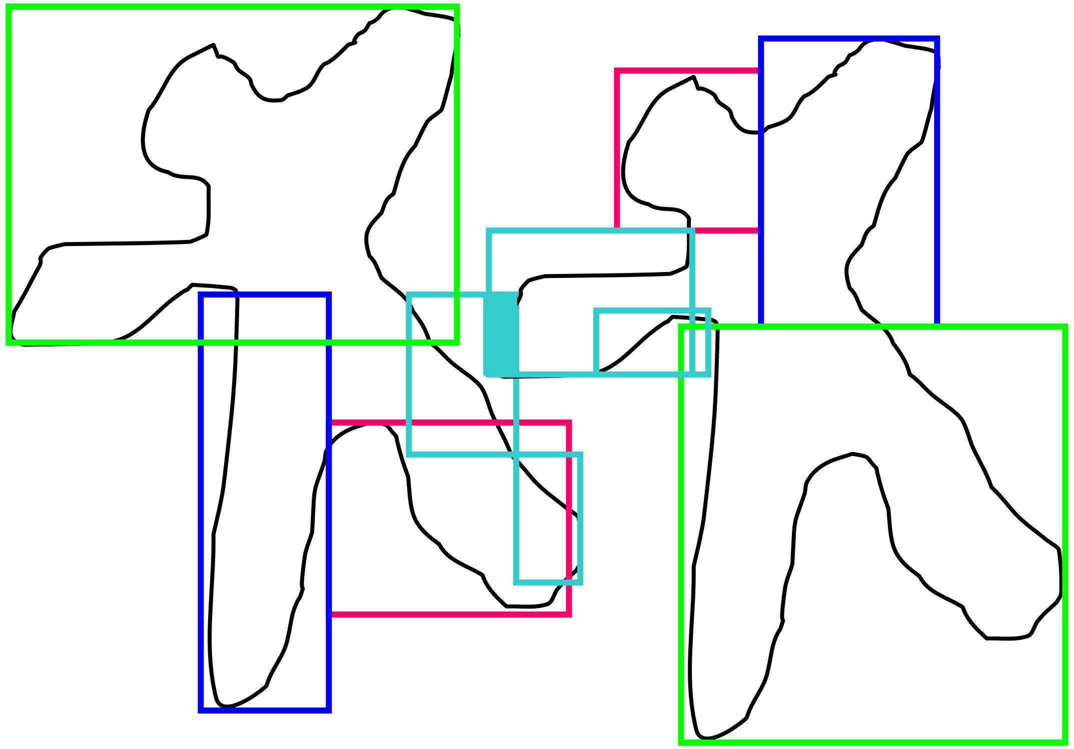
BVH Collision Test example



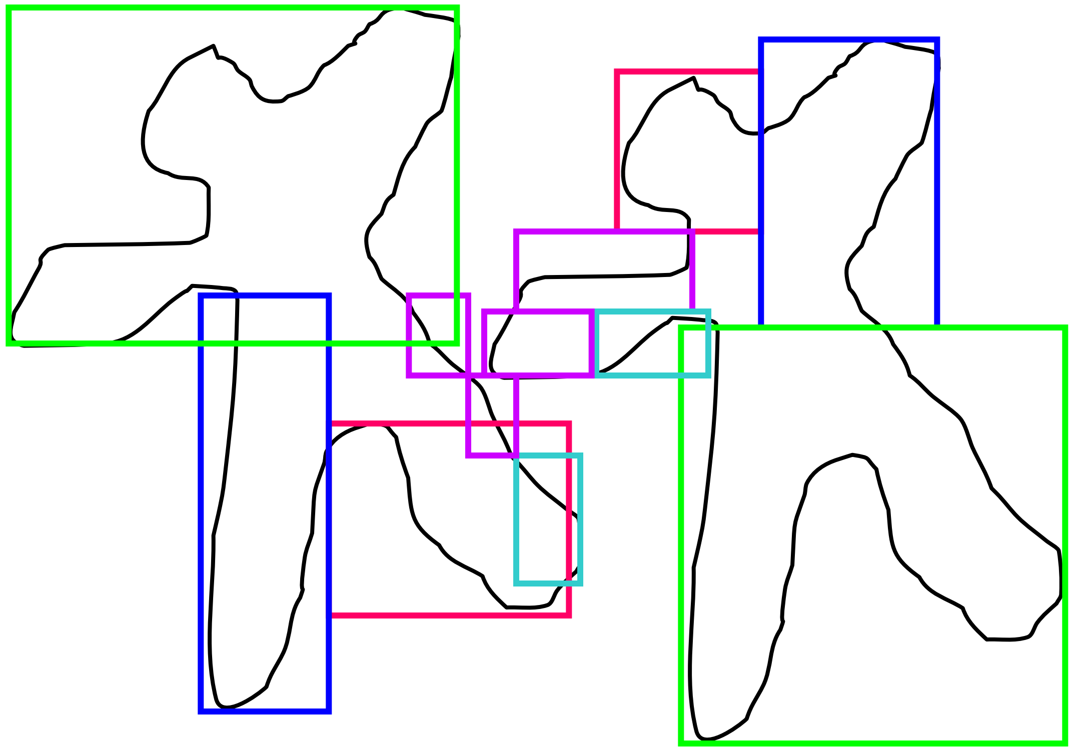
BVH Collision Test example



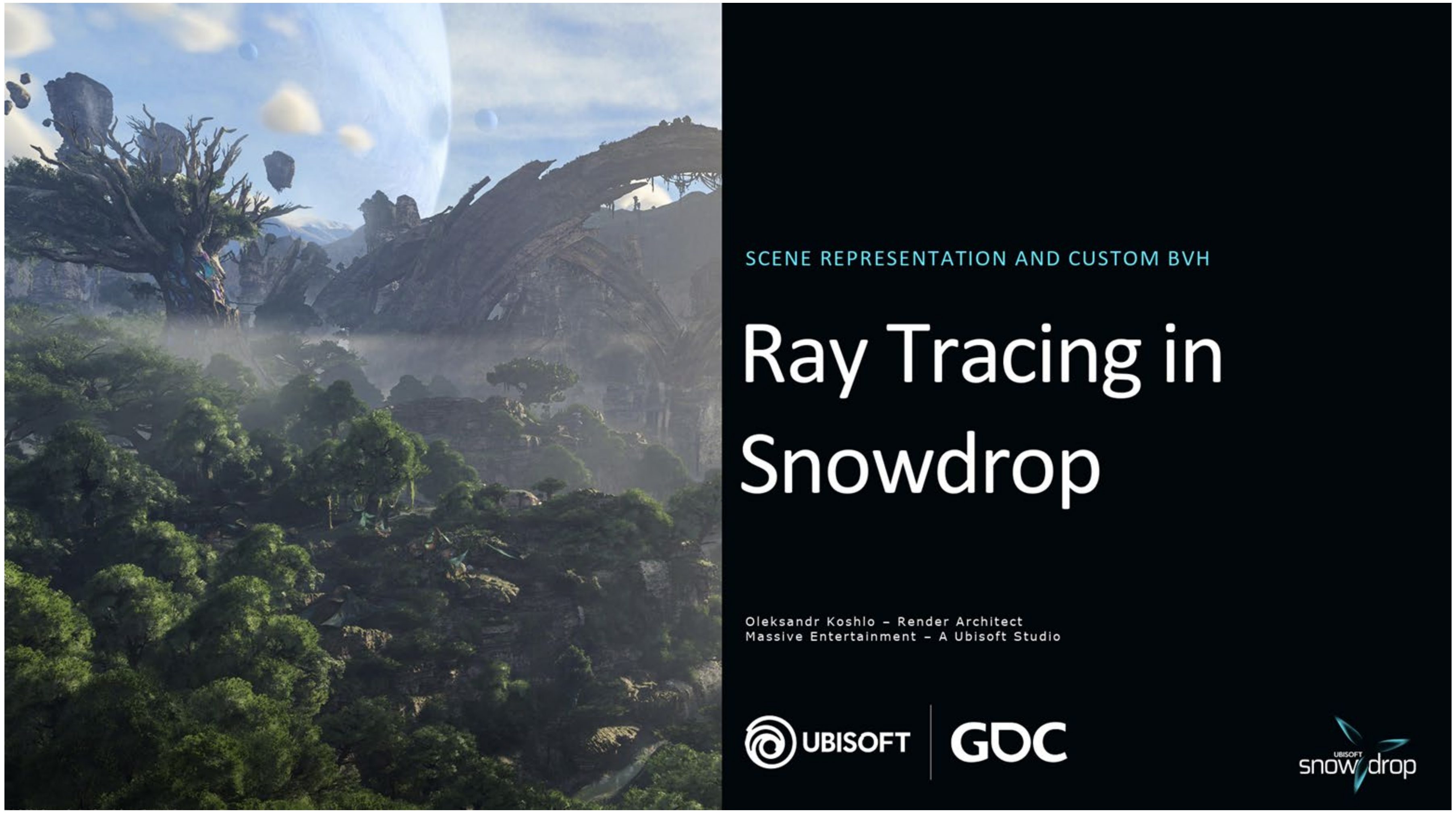
BVH Collision Test example



BVH Collision Test example



BVHs Also Can Accelerate Ray Tracing



SCENE REPRESENTATION AND CUSTOM BVH

Ray Tracing in Snowdrop

Oleksandr Koshlo – Render Architect
Massive Entertainment – A Ubisoft Studio

UBISOFT | GDC

From Game Developer’s Conference 2025