Geometric query problem

- Complex geometry or large environment
- How can we perform geometry query efficiently? (e.g. for ray tracing)
In ray tracing

Given a ray

For each triangle in the scene
Intersect( , )
Keep the closest hit

EndFor

- Complexity: $O(N)$
- Can we do better?
In ray tracing

- Ray only occupies a narrow beam
- We don’t need to loop over all triangles
- Need good data structure to organize triangles
Idea: Simpler bounding geometry

- Bound geometries with simpler shapes such as box, sphere etc.
- Test if ray intersects with bounding box (cheap)
- If the ray does hit the bounding box, query triangles in the box
- Skip all triangles in the box if the ray misses the box
- Worst case scenario is still \( O(N) \)...
How do we speed it up?

\[ O(N) \quad \longrightarrow \quad O(\log N) \]
A simpler but similar problem

- Suppose we have set of integers $S$

  10 123 2 100 6 25 64 11 200 30 950 111 20 8 1 80

- Given an integer, say $k = 18$, find the element in $S$ closest to $k$

- First, sort the set into a binary search tree $O(N \log N)$

- Then any query would take $O(\log N)$
Fast geometry query

• Can we also reorganize scene triangles to enable fast ray-scene intersection queries?

• Inspired by hierarchal search trees…
  ▶ Bounding volume hierarchy (BVH)
  ▶ k-D tree
  ▶ Octree
Fast geometry query

- Bounding volume hierarchy (BVH)
- k-D tree
- Octree
Bounding Volume Hierarchy (BVH)

- Leaf nodes: small number of primitives
- Interior nodes: store sub-bounding boxes
bounding volume hierarchy (bvh)

- subtrees can overlap
Bounding Volume Hierarchy (BVH)
Bounding Volume Hierarchy (BVH)

- To build BVH
- Repeatedly partition each group into two sub-groups
  - To fairly partition group, set up some score based on position
  - Based on histogram, evenly partition elements into two groups
Fast geometry query

- Bounding volume hierarchy (BVH)
- k-D tree
- Octree
Fast geometry query

- Bounding volume hierarchy (BVH)
- k-D tree
- Octree
Spatial partitioning

- Primitive-based partitioning (e.g. Bounding volume hierarchy)
- Space-based partitioning

- Partition space into disjoint regions
Spatial partitioning

• For example, partition space into uniform grid
• We can march the ray cell-by-cell and stop whenever we hit a triangle
  ▶ For BVH there is no obvious ordering among bounding volumes
  ▶ But each cell may contain much more nodes than BVH
k-D tree

• k-D tree partition space non-uniformly
  ▶ Repeatedly separate space by axis-aligned planes
  ▶ E.g. when partitioning normal to y axis, score elements by y axis and use the medium value for the position of the partition plane

• Search intersection similar to BVH

• Unlike BVH, we can terminate early when a hit is found (easy to figure out the order of boxes along the ray)
Fast geometry query

- Bounding volume hierarchy (BVH)
- k-D tree
- Octree
Fast geometry query

- Bounding volume hierarchy (BVH)
- k-D tree
- Octree
Quad-tree or Octree

- If needed, repeatedly bisect box into 4 (2D) or 8 (3D) sub-boxes
- Like k-D tree, it is adaptive to scene complexity
- Much easier to build than k-D tree (no need to choose separating planes)
- But lower ray traversal performance than k-D tree
**Summary**

- **BVH**
  - Adaptive
  - No ordering

- **Uniform grid**
  - Non-adaptive
  - Ordered
  - Easy to build

- **k-D tree**
  - Adaptive
  - Ordered, fast marching
  - More effort to build

- **Octree**
  - Adaptive
  - Ordered, slower marching
  - Easy to build

Quad-tree nodes have 4 children (partitions 2D space)
Octree nodes have 8 children (partitions 3D space)
More hierarchical accelerations

• **Geometry**
  - Closest-point query
  - Distance query

• **Particle simulation**
  - Fast multipole method (Distant clusters can be simplified as one particle)
  - Fluid simulation, N-body problem