CSC317 Computer Graphics
Tutorial 4

October 2, 2024

About Assignment 1: Raster Images Grading

- Will try to finalize the grading by the end of this week
- If not, by next Wednesday
- Sorry for the delay!

Assignment 4: Bounding Volume Hierarchy

- Due Date: October 8 @ 11:59 pm
- This slides follows my personal recommendation order of implementation

Assignment 4: BVH

- Task1: ray_intersect_triangle.cpp
  - Intersect a ray with a triangle
  - Feel free to use your code from A3 Ray Casting (is the same function) :)

Assignment 4: BVH

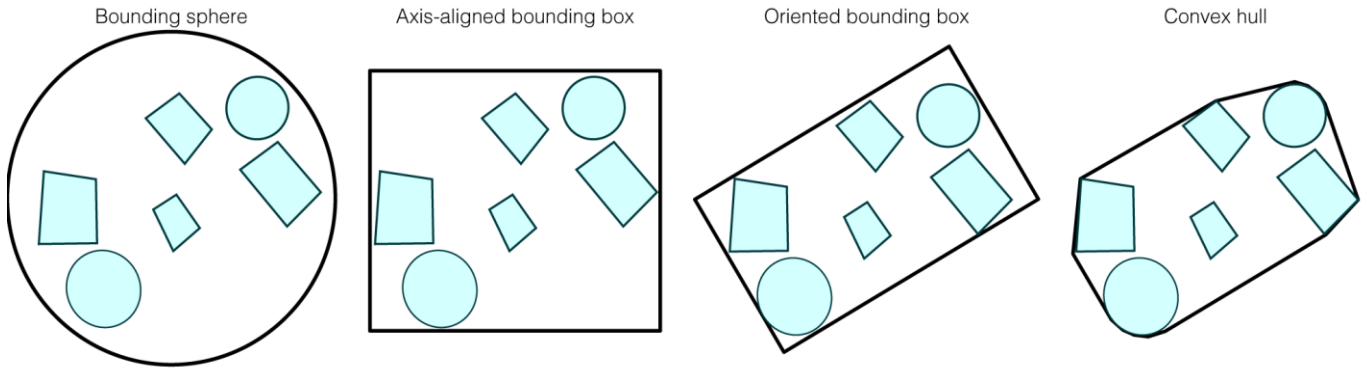
- Task2: ray_intersect_triangle_mesh_brute_force.cpp
  - Shoot a ray at a triangle mesh with n faces and record the closest hit. Use a brute force loop over all triangles, aim for O(n) complexity but focus on correctness. This will be your reference solution.
  - Hint: brute force... for loop... there are n faces...
    - for i in range(n):
      - Do something
      - Should only take 5 seconds to implement

Assignment 4: BVH

- Task3: ray_intersect_box.cpp
- Intersect a ray with a **solid** box
  - if the ray or min_t lands inside the box this could still hit something stored inside the box, so this counts as a hit.
  - Implication: **Any** intersection of the ray with the box, or if the ray's starting point (or the point at parameter min_t) is **inside** the box, should be considered a hit. This means that even if the ray does not intersect the box's surface directly but originates from within the box, it should still be regarded as an intersection event.

Assignment 4: BVH

- By "box", we meant AABB (**A**xis **A**ligned **B**ounding **B**ox)
  - What's axis aligned?
  - What's bounding box?
- Task12: box_box_intersect.cpp
- Task4: insert_box_into_box.cpp
- Task5: insert_triangle_into_box.cpp

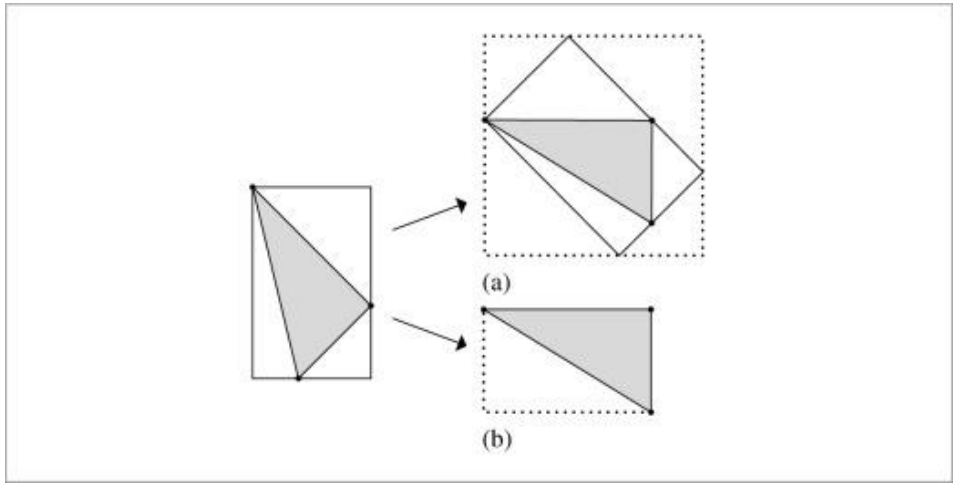


Assignment 4: BVH

- Task4: insert_box_into_box.cpp
- There are two boxes, if you insert Box A into Box B, how do you relocate Box B's corner vertices so that B bounds A?
- Hint: every box is represented with two corner vertices, the **min** corner and the **max** corner. Why?

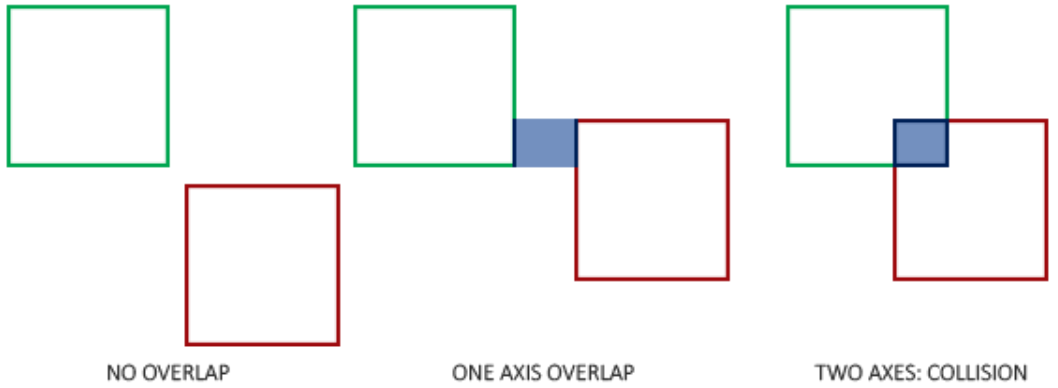
Assignment 4: BVH

- Task5: insert_triangle_into_box.cpp
- Given a triangle (corner vertex A, B, C)
- Find the box that tightly bound this triangle (AABB box is parametrized by the **min** corner and **max** corner)
  - Hint: **Axis-aligned**, so check the max/min along every dimension



Assignment 4: BVH

- Task12: box_box_intersect.cpp
- There are two boxes, when do they overlap/intersect?
- Hint: all you need to check are the 4 corners (2 each box)



Assignment 4: BVH

- Task6: src/AABBTree.cpp (Hard)
- Left tree… Right tree…
- What does it sound like to you?

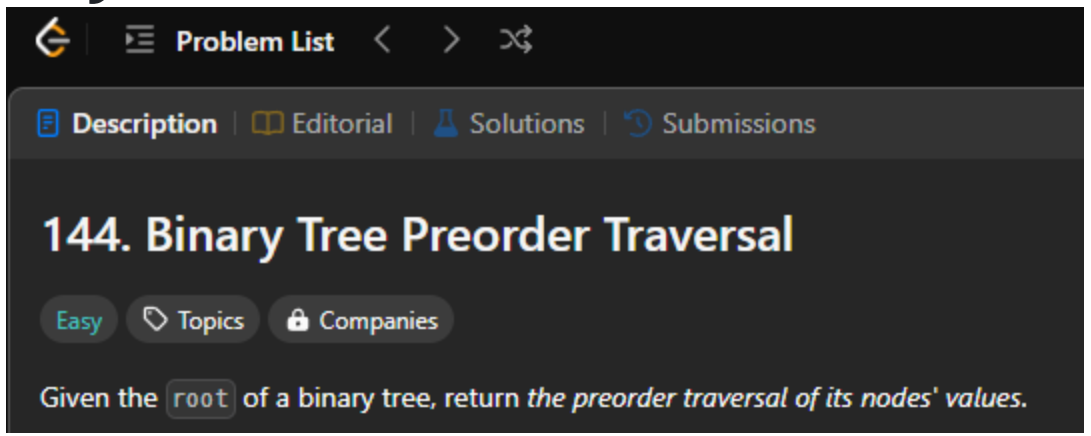
```cpp
struct AABBTree : public Object, public std::enable_shared_from_this<AABBTree>
{
  // Pointers to left and right subtree branches. These could be another
  // AABBTree (internal node) or a leaf (primitive Object like MeshTriangle, or
  // CloudPoint)
  std::shared_ptr<Object> left;
  std::shared_ptr<Object> right;
  // For debugging, keep track of the depth (root has depth == 0)
  int depth;
  // For debugging, keep track of the number leaf, descendants
  int num_leaves;
  // Construct a axis-aligned bounding box tree given a list of objects. Use the
  // midpoint along the longest axis of the box containing the given objects to
  // determine the left-right split.
  //
  // Inputs:
  //   objects  list of objects to store in this AABBTree
  //   Optional inputs:
  //     depth  depth of this tree (usually set by constructor of parent as
  //       their depth+1)
  // Side effects: num_leaves is set to objects.size() and left/right pointers
  // set to subtrees or leaf Objects accordingly.
  AABBTree(
    const std::vector<std::shared_ptr<Object> > & objects,
    int depth=0);
  // Object implementations (see Object.h for API)
  bool ray_intersect(
    const Ray& ray,
    const double min_t,
    const double max_t,
    double & t,
    std::shared_ptr<Object> & descendant) const override;
  bool point_squared_distance(
    const Eigen::RowVector3d & query,
    const double min_sqrd,
    const double max_sqrd,
    double & sqrd,
    std::shared_ptr<Object> & descendant) const override
  {
    assert(false && "Do not use recursive DFS for AABBTree distance");
    return false;
  }
};
```

Assignment 4: BVH

- **Recursion!**
- **Initialization**: The constructor takes in a list of objects and a depth value, which initializes the depth and num_leaves attributes.
- **Bounding Box Calculation**: The code iterates through each object and updates the bounding box (box) to encompass all objects. (How do we do that?)
- **Splitting Logic**:
  - If there are more than two objects, the code determines the **longest** axis along which to split the bounding box. It computes the midpoint on this axis and divides objects into left tree objects and right tree objects based on whether their centers fall to the left or right of this split.
  - It ensures that both left and right object trees contain at least **one** object to avoid an imbalance.
- **Recursive Construction**: The constructor recursively builds the left and right subtrees by creating new AABBTree instances with the divided object lists.
- **Base Cases**: If there are only one or two objects, they are directly assigned to the left and right children of the tree without further splitting.
  - If only one object, just leave it in the left tree.

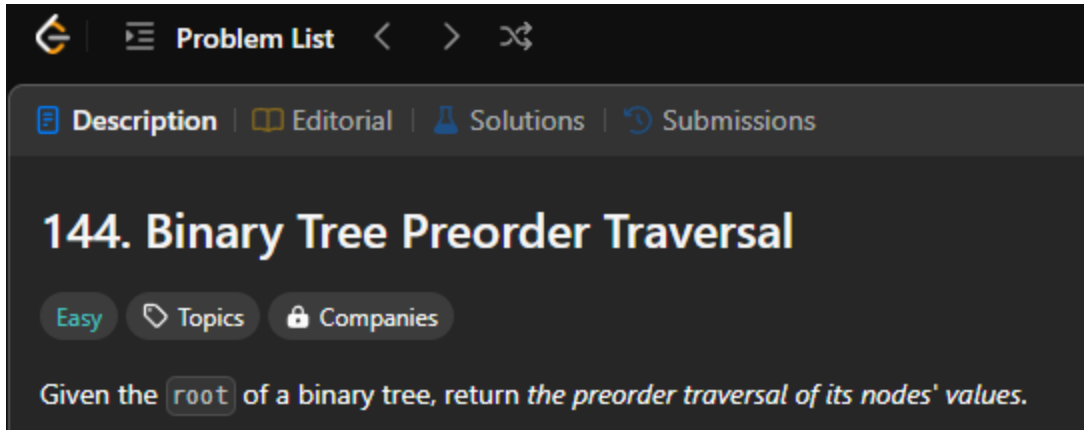
Assignment 4: BVH

- Task7: `src/AABBTree_ray_intersect.cpp`
- AABB Tree is a tree of nested boxes
- Tree traversal to check if ray hit AABB Tree
  - Check box in the box
    - Check box in the box
      - Check box in the box
        - Check box in the box
          - Check box in the box
            - Check box in the box
              - Check...you know



Assignment 4: BVH

- Task7: `src/AABBTree_ray_intersect.cpp`
- So, **recursion** to traverse a tree
- Preorder traversal

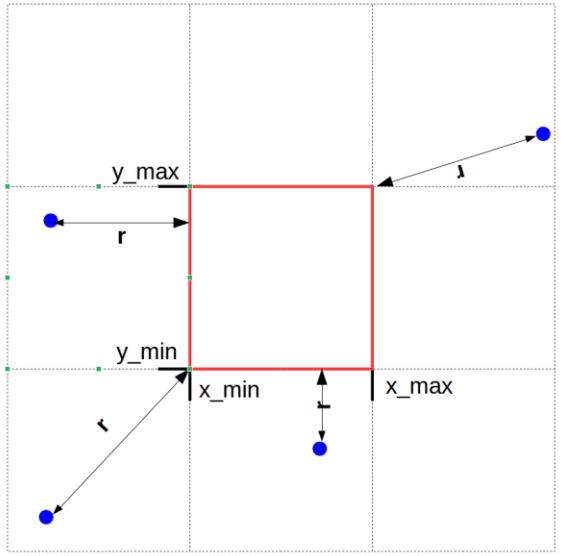


Assignment 4: BVH

- Task8: nearest_neighbor_brute_force.cpp
- Brute force approach to compute point wise distance and find nearest neighbor of query point u
  - For every point v
    - Distance(u, v)
    - You know what's next

Assignment 4: BVH

- Task9: point_box_squared_distance.cpp
- This one is easy



Assignment 4: BVH

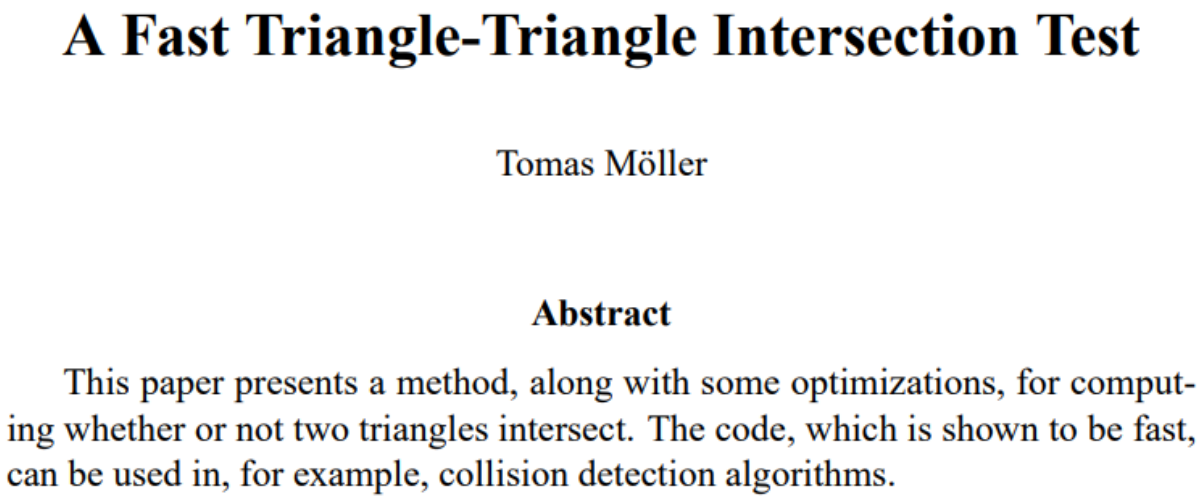
- Task10: point_AABBTree_squared_distance.cpp (Hard)
- In README
- Iterative
- **Best-first search**
- Use a priority queue for **minimum distance** book keeping

Breadth-first search is a much better structure for distance queries on a spatial acceleration data-structure. Pseudo-code for a closest distance algorithm might look like:

```
// initialize a queue prioritized by minimum distance
d_r ← distance to root's box
Q.insert(d_r, root)
// initialize minimum distance seen so far
d ← ∞
while Q not empty
  // d_s: distance from query to subtree's bounding box
  (d_s, subtree) ← Q.pop
  if d_s < d
    if subtree is a leaf
      d ← min[ d , distance from query to leaf object ]
    else
      d_l ← distance to left's box
      Q.insert(d_l ,subtree.left)
      d_r ← distance to right's box
      Q.insert(d_r ,subtree.right)
```

Assignment 4: BVH

- Task11: triangle_triangle_intersection.cpp
  - There are multiple way to do it, in reference solution, we implemented paper ***A Fast Triangle-Triangle Intersection Test***
  - Separation Axis Theorem (SAT)



A Fast Triangle-Triangle Intersection Test

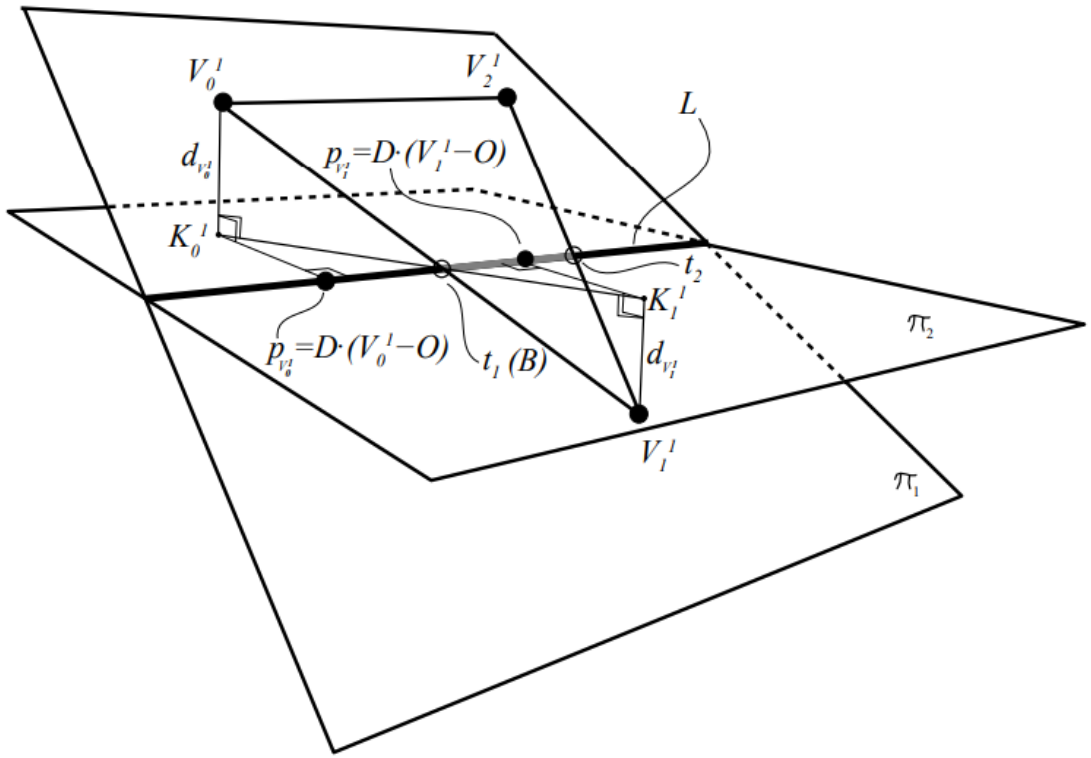
Tomas Möller

Abstract

This paper presents a method, along with some optimizations, for computing whether or not two triangles intersect. The code, which is shown to be fast, can be used in, for example, collision detection algorithms.

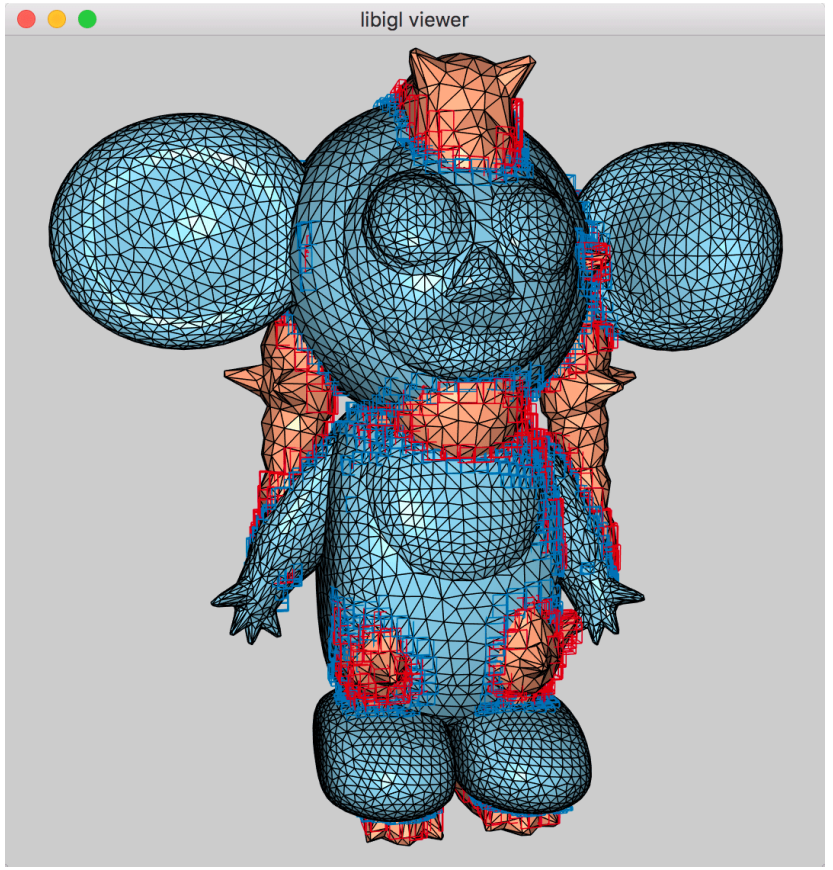
Assignment 4: BVH

- Task11: triangle_triangle_intersection.cpp
  - There are multiple way to do it, in reference solution, we implemented paper ***A Fast Triangle-Triangle Intersection Test***
  - Separation Axis Theorem (SAT)



Assignment 4: BVH

- Task13: find_all_intersecting_pairs_using_AABBTrees.cpp (Hard)
- Find all intersecting pairs of **boxes**
- So, call box-box intersection checks
- But how?



Assignment 4: BVH

- See README
- Basically iterate through all combination of boxes in tree A and tree B and check for intersection
- **iterative BFS**

Intersection queries between two trees

Suppose we want to find *all pairs* of intersecting triangles between two meshes. One approach would be to put one mesh's triangles in an AABB tree, then loop over the other mesh's triangles using the tree to accelerate intersection tests. This works well if the mesh in the tree has many more triangles than the other mesh, but can we do better if both mesh have many triangles? How about putting both meshes in a AABB trees. If the root bounding boxes don't overlap we find out *instantaneously* that there are no pairs of intersecting triangles. If they do overlap, we check their childrens' boxes against each other. Anytime two boxes don't overlap we save many expensive pairwise triangle checks. A rough sketch of this algorithm using a simple (i.e., non prioritized) queue is like this:

```
// initialize list of candidate leaf pairs
leaf_pairs ← {}
if root_A.box intersects root_B.box
  Q.insert( root_A, root_B )
while Q not empty
  {nodeA,nodeB} ← Q.pop
  if nodeA and nodeB are leaves
    leaf_pairs.insert( node_A, node_B )
  else if node_A is a leaf
    if node_A.box intersects node_B.left.box
      Q.insert( node_A, node_B.left )
    if node_A.box intersects node_B.right.box
      Q.insert( node_A, node_B.right )
  else if node_B is a leaf
    if node_A.left.box intersects node_B.box
      Q.insert( node_A.left, node_B)
    if node_A.right.box intersects node_B.box
      Q.insert( node_A.right, node_B)
  else
    if node_A.left.box intersects node_B.left.box
      Q.insert( node_A.left, node_B.left )
    if node_A.left.box intersects node_B.right.box
      Q.insert( node_A.left, node_B.right )
    if node_A.right.box intersects node_B.right.box
      Q.insert( node_A.right, node_B.right )
    if node_A.right.box intersects node_B.left.box
      Q.insert( node_A.right, node_B.left )
```

Assignment 4: BVH

- What to submit

A4: Bounding Volume Hierarchy	Tuesday, October 08, 2024, 11:59:00 PM EDT

ray_intersect_triangle.cpp
ray_intersect_triangle_me
ray_intersect_box.cpp
insert_box_into_box.cpp
insert_triangle_into_box.c
AABBTree.cpp
AABBTree_ray_intersect.c
nearest_neighbor_brute_f
point_box_squared_distan
point_AABBTree_squared_
triangle_triangle_intersect
box_box_intersect.cpp
find_all_intersecting_pairs