Building fast kd-trees for ray-tracing

From the paper:

On building fast kd-Trees for Ray Tracing, and on doing that in O(N log N), Wald and Havran, 2006.

KD-tree modifications for ray-tracing

kd-trees to contain objects (instead of points):
Objects are -ray-tracing primitives (triangles in our case).
Objects only at leaf nodes.
Objects which cross boundaries, will be in multiple leaf nodes
Split planes can be at any value along any axis

Goals

The KD-tree will be built highly optimized for ray-tracing traversal
using the Surface Area Heuristic (SAH) algorithm
Traversal time is **utmost** importance
Build time is of lesser importance,
The paper presents three algorithms for building the SAH tree

1. $O(N^2)$
2. $O(N \log^2 N)$
3. $O(N \log N)$

Generic kd-tree building algorithm

```
BuildTree(triangles T):
    V = AABB(T)
    return BuildNode(T,V)

BuildNode(triangles T, box V):
    if Terminate(T,V):
        return LeafNode(T)
    else:
        p,axis= ChooseSplitPlane(T,V)
        V_L, V_R = SplitBoxAtPlane(V,p,axis)
        T_L = { t∈T | (t∩V_L) ≠ ∅ }
        T_R = { t∈T | (t∩V_R) ≠ ∅ }
        return Node(P,axis, BuildNode(T_L,V_L), BuildNode(T_R,V_R))
```

All the intelligence is built into **Terminate** and **ChooseSplitPlane.**

Naive algorithm:

A very simple algorithm (not one of the three in the paper):
ChooseSplitPlane:

```
    axis= depth mod 3
    p = (V.min[axis]+V.max[axis])/2
```

Terminate(T,V) = $|T| \le K_{MaxTriangles}$ or $Depth(V) \ge K_{MaxDepth}$

Traversal for Ray Casting

```
TraverseNode(node, ray) // returns first intersection along ray
    If node is leaf node,
        For each object in leaf-node's list of objects
            intersect ray and object
            keep track of front-most intersection (if any)
    else // is an internal node
        determine one of these four cases:
            ray intersects only left child
            ray intersects only right child
            ray intersects left child then right child
            ray intersects right child then left child
        Recurse down first child
        If an intersection is found:
            return it
        else if a second child is hit
            recurse down second child
            if an intersection is found:
                return it
            else
                return NO-INTERSECTION
```

The Surface Area Heuristic (SAH) algorithm

For arbitrary splitting plane p,axis

Define "expected cost" of traversing node a split at p:

This "cost" should account for:

cost of intersecting ray with leaf node objects

cost of traversing a node

cost of recursing down the tree

Also define a "expected cost" for not splitting a node

Algorithm for splitting a node

Calculate cost of splitting at each p along each axis

Calculate cost of not splitting

Choose the one with the minimal cost

This answers both **Terminate** and **ChooseSplitPlane**

Assumptions:

Rays are uniformly distributed and infinite in length

Costs are known and constant:

Cost of one traversal step: C_T

Cost of one intersection: C_I

Probability of intersecting a child node V_c of a node V is computed from surface areas:

$$P(V_c, V) = \frac{SA(V_c)}{SA(V)}$$

Expected cost of subdividing V at plane p:

$$C_V(p) = C_T + C_I P(V_L, V)|T_L| + C_I P(V_R, V)|T_R|$$

Terminate when $min_P(C_V(p)) > C_I|(T)|$

Build a tree which minimizes the "expected cost" of traversal:

Paper gives three ways to build the tree using this metric:

$O(N^2)$ Naive:

Calculate $|T_L|$ and $|T_R|$ for each split candidate.

$O(N \log^2 N)$ Sweep:

Calculate $|T_L|$ and $|T_R|$ incrementally across each dimension.

$O(N \log N)$ No sort sweep: 3.

One global sort is applied to all levels

Limiting calculation to *finitely many* split candidates:

The parts of $C_V(p)$ are either constant, linear, or pieces-wise linear

So $C_V(p)$ is piecewise linear with breaks at triangle vertices.

A property of piecewise linear functions:

minimum will occur at one of the piecewise breaks.

So we only need to consider triangle vertices for split candidates.

Naive algorithm $O(N^2)$:

Calculate cost of not splitting
For each axis s:
    For each triangle's min and max point p on axis s:
        Calculate cost of splitting at p
            (Note: requires a loop through all triangles to calc $|T_L|, |T_R|$)
Choose minimal cost, and split or not accordingly.

Sweep algorithm $O(N \log^2 N)$:

Calculate cost of not splitting
For each axis s:
    Gather points along s-axis
        points = list of each triangle's min and max points on s-axis
    Sort points on s-axis
    Group points by common position
        For each group of points with a unique position p_k along s-axis:
            calculate:
                p_k^+ = number triangles starting at p_k
                p_k^0 = number triangles in plane of p_k
                p_k^- = number triangles ending at p_k
    Sweep left-to-right along s-axis
        Start counters at p_0
            $N_L = 0$
            $N_R = N - p_0^0$
            $N_P = p_0^0$
        For each group p_k along s-axis:
            Inc/Dec the counters from p_{k-1} to p_k
                $N_L \mathrel{+}= p_{k-1}^0 + p_{k-1}^+$
                $N_R \mathrel{-}= p_k^0 + p_k^-$
                $N_P = p_k^0$
            Calculate $C_V(p)$ using $|T_L| = N_L + N_P$ and $|T_R| = N_R + N_P$

Choose minimal cost, and split or not accordingly.

Sweep-no-sort algorithm $O(N \log N)$:

Same as previous algorithm except remove the need to sort at each node by doing three global sorts (one per axis) and maintaining the three sorts into the recursive calls.

Turning a list of triangles (maintained according to the three sorts) into two sub-lists of triangles (also maintained according to the three sorts) seems like a difficult thing to do.