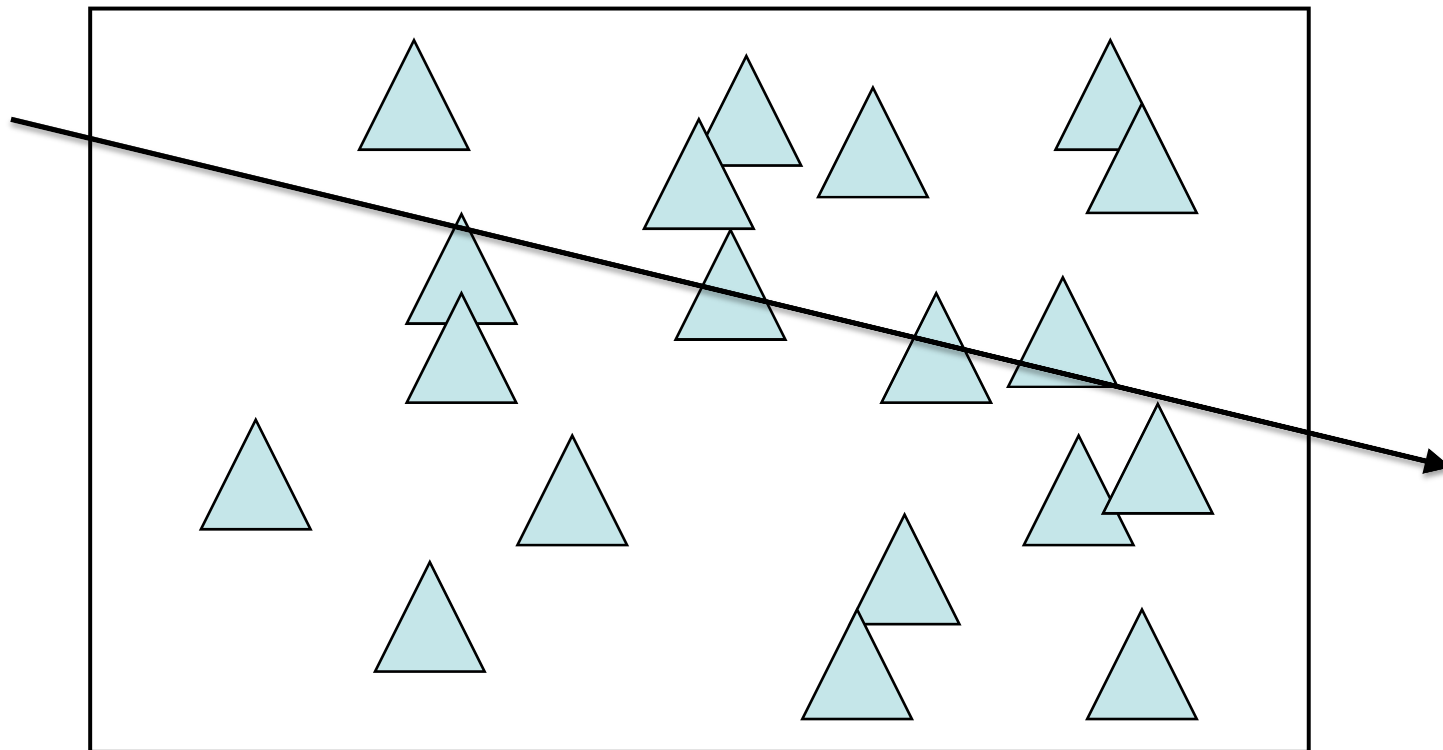
Acceleration Structure for Ray Tracing

K-D Tree Traversal

Copyright © 2010 by Yong Cao

Motivation

- **Locate the primitives that intersect with a ray, FAST!** **(Avoid brute-force approach)**



Copyright © 2010 by Yong Cao

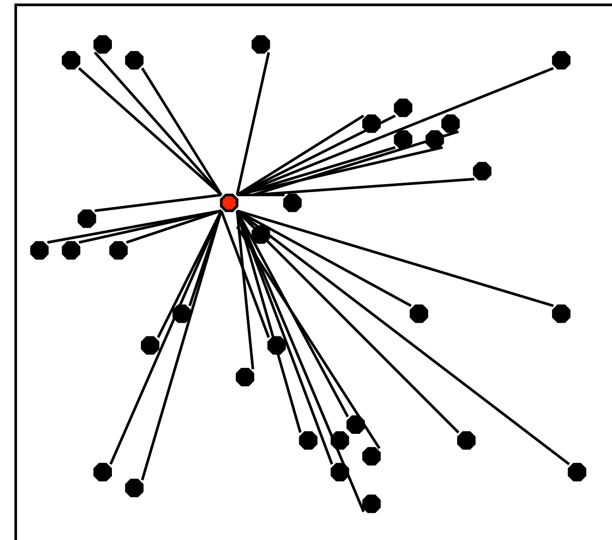
Background

- **Similarity search or nearest neighbor search**
- **Applications:**
  - Statistics
  - Content-based retrieval
  - Data-mining and Pattern recognition
  - Geographic information system (GIS)
    - where is the closest post office
  - Graphics

Copyright © 2010 by Yong Cao

Naïve Approach

- Given a query point X.
- Scan through each point Y
- Takes O(N) time for each query!



33 Distance Computations

Copyright © 2010 by Yong Cao

Acceleration: Spatial Index Structure

- **We can speed up the search for the nearest neighbor:**
  - Examine nearby points first.
  - Ignore any points that are further then the nearest point found so far.

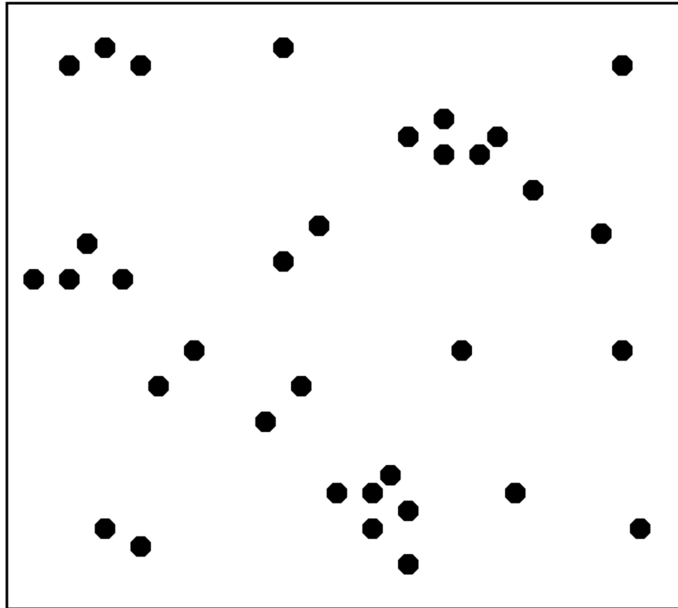
Copyright © 2010 by Yong Cao

K-D Tree

- k-d tree is a multidimensional binary search tree.
- Recursively partitions points into axis aligned boxes. One axis at a time.

Copyright © 2010 by Yong Cao

KD-Tree Construction

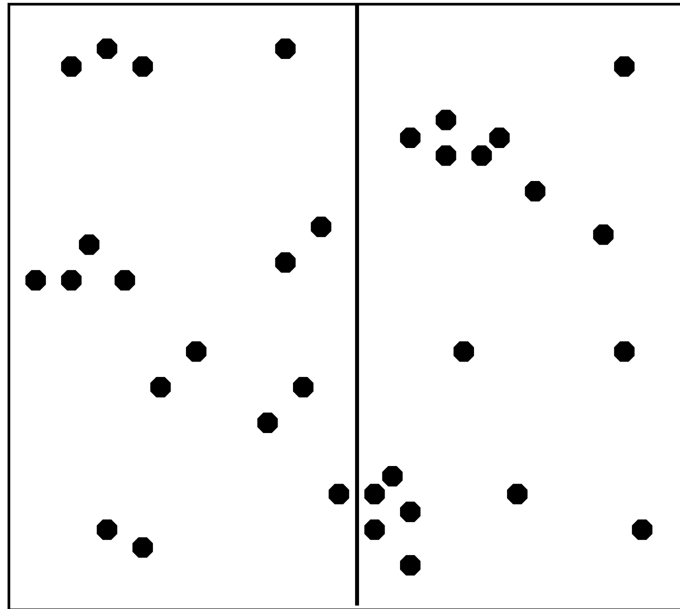


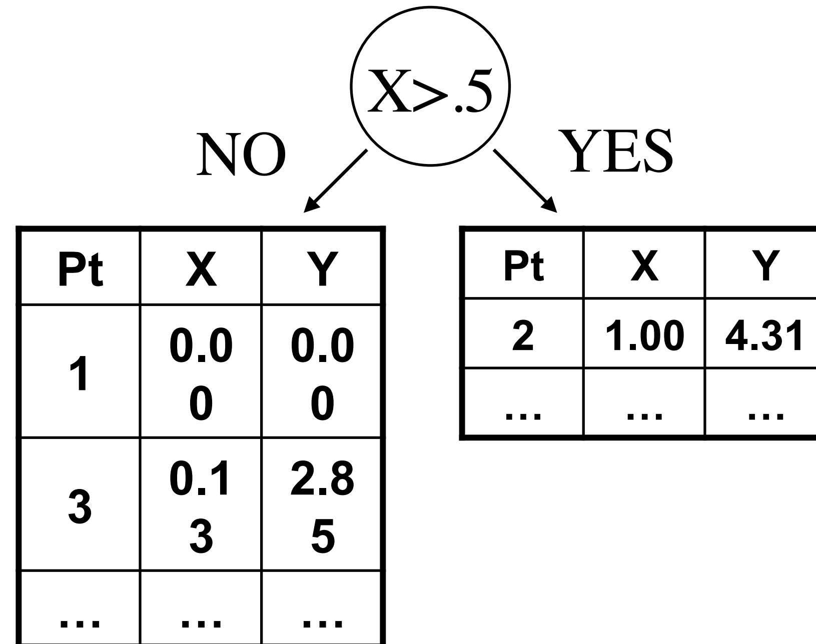
Pt	X	Y
1	0.00	0.00
2	1.00	4.31
3	0.13	2.85
…	…	…

We start with a list of n-dimensional points.

Copyright © 2010 by Yong Cao

KD-Tree Construction





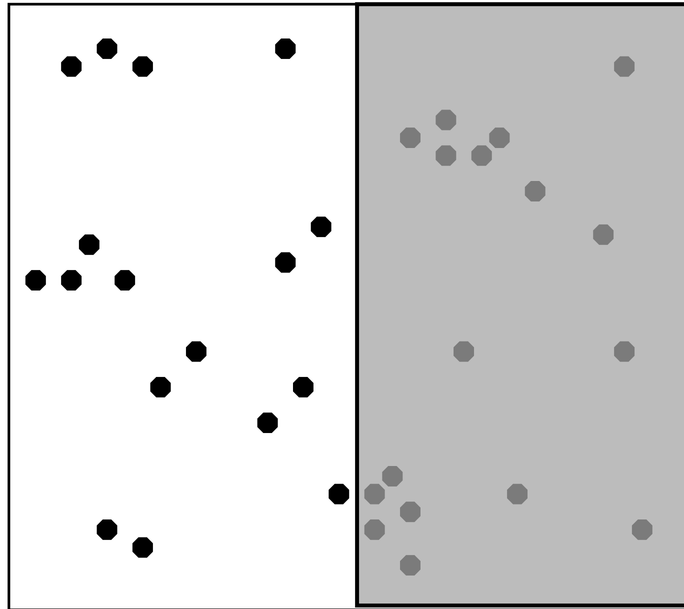
Pt	X	Y
1	0.00	0.00
3	0.13	2.85
…	…	…

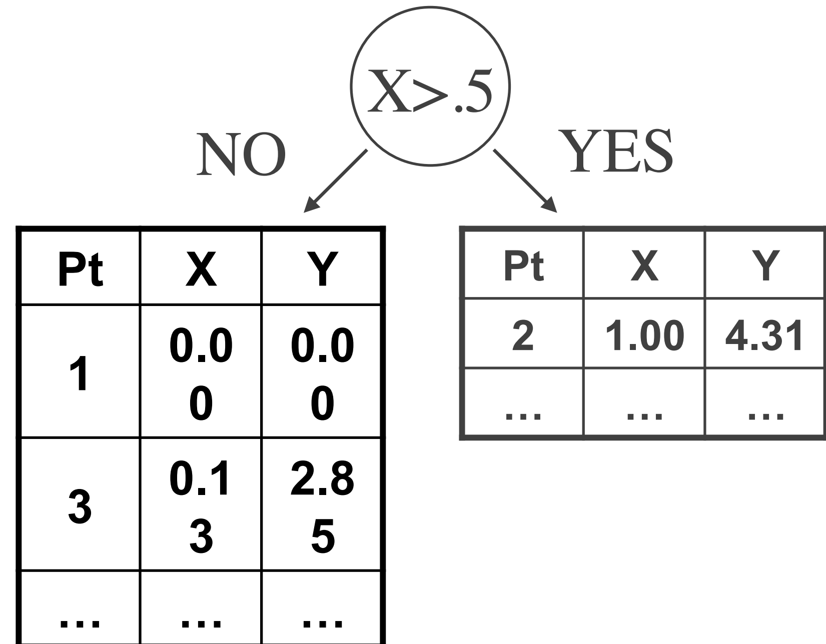
Pt	X	Y
2	1.00	4.31
…	…	…

We can split the points into 2 groups by choosing a dimension X and value V and separating the points into X > V and X <= V.

Copyright © 2010 by Yong Cao

KD-Tree Construction





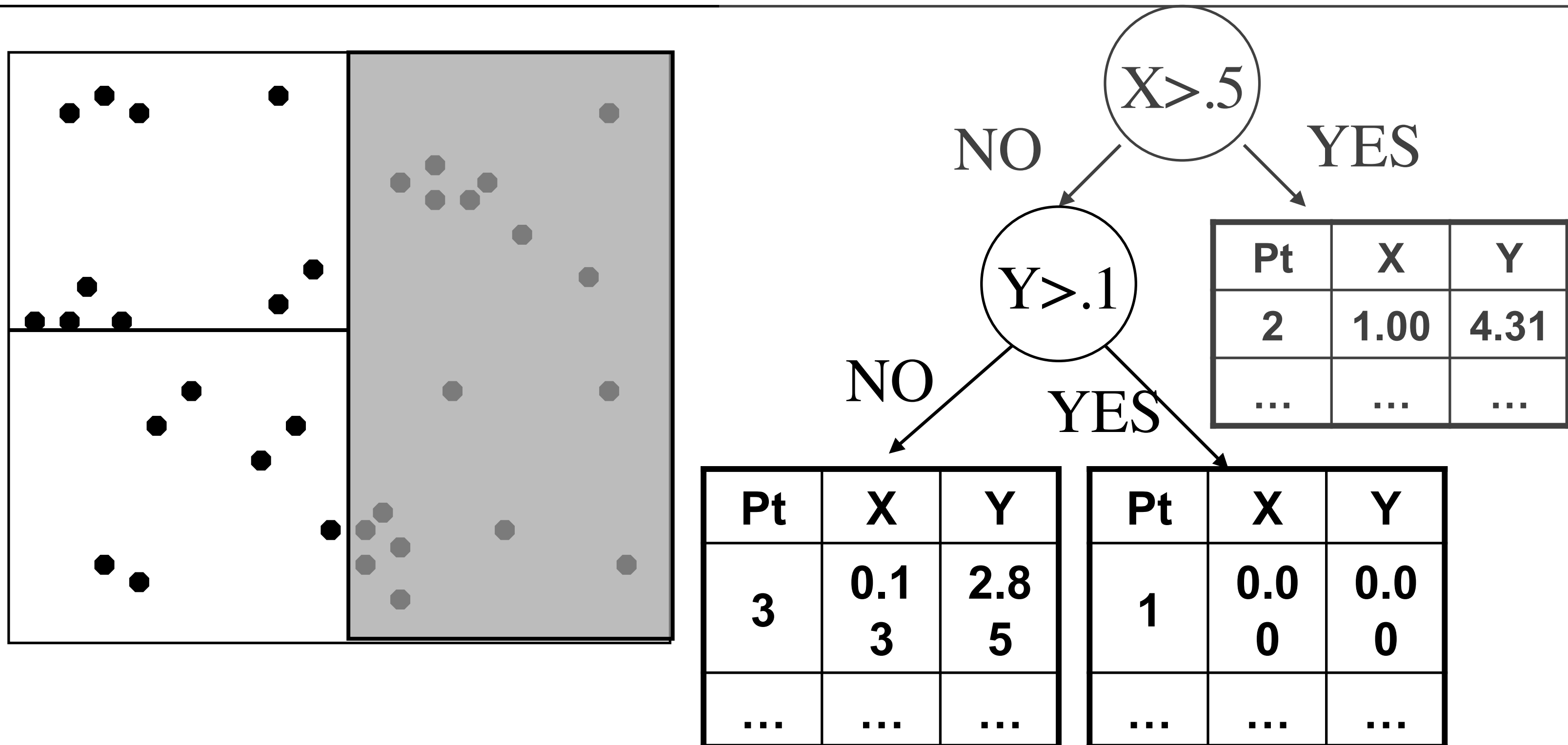
Pt	X	Y
1	0.00	0.00
3	0.13	2.85
…	…	…

Pt	X	Y
2	1.00	4.31
…	…	…

We can then consider each group separately and possibly split again (along same/different dimension).

Copyright © 2010 by Yong Cao

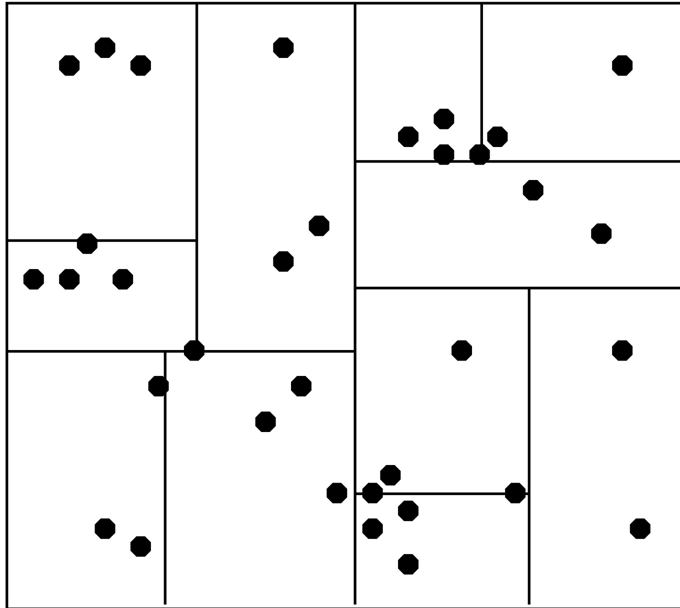
KD-Tree Construction

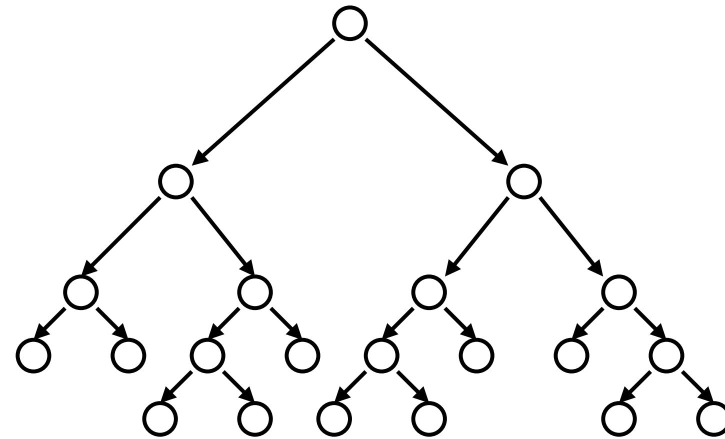


We can then consider each group separately and possibly split again (along same/different dimension).

Copyright © 2010 by Yong Cao

KD-Tree Construction

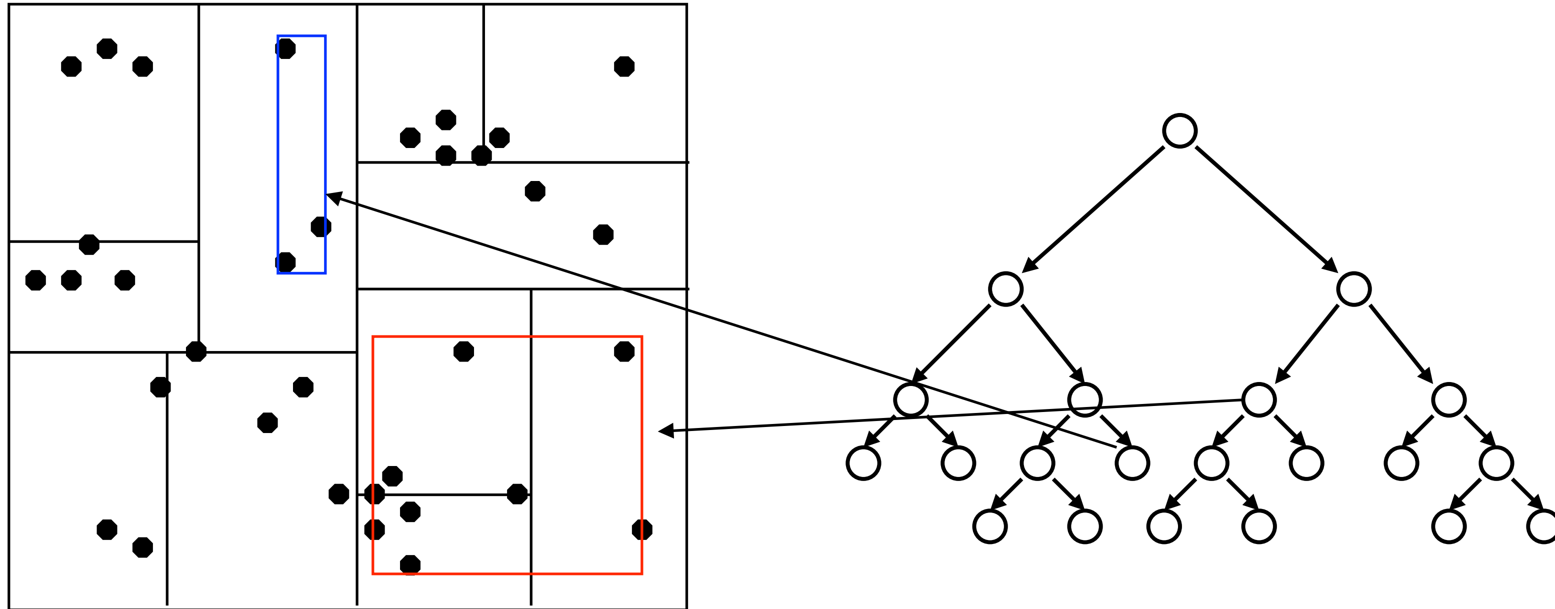




We can keep splitting the points in each set to create a tree structure. Each node with no children (leaf node) contains a list of points.

Copyright © 2010 by Yong Cao

KD-Tree Construction



We will keep around one additional piece of information at each node. The (tight) bounds of the points at or below this node.

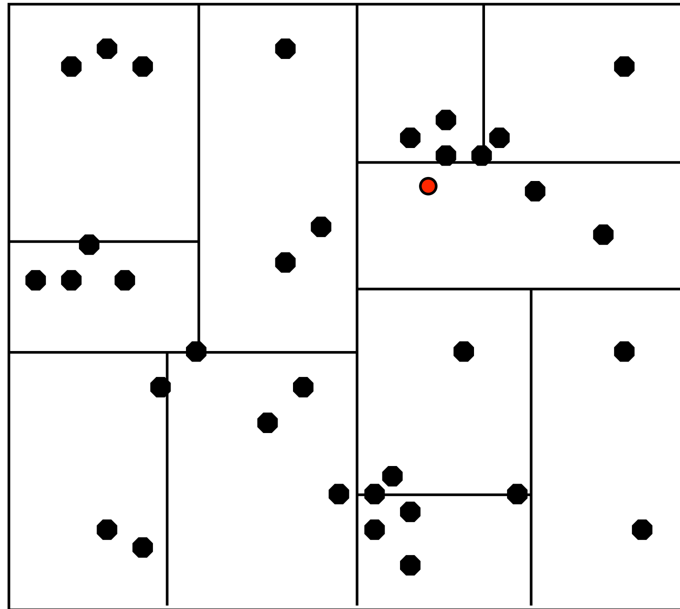
Copyright © 2010 by Yong Cao

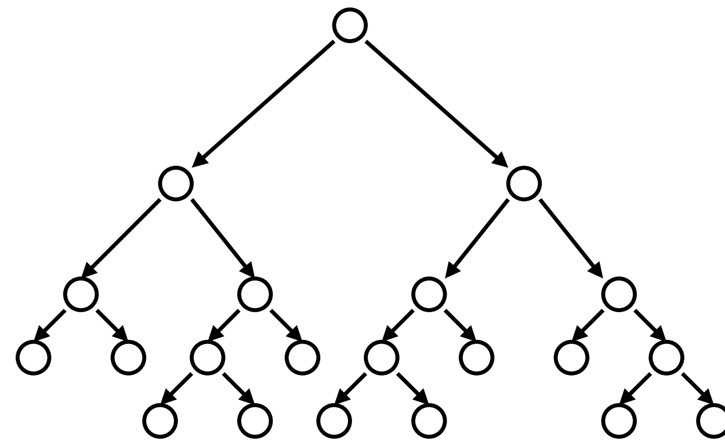
KD-Tree Construction

- Use heuristics to make splitting decisions:
- Which dimension do we split along? Widest
- Which value do we split at? Median of value of that split dimension for the points.
- When do we stop? When there are fewer then m points left OR the box has hit some minimum width.

Copyright © 2010 by Yong Cao

Nearest Neighbor with KD Trees

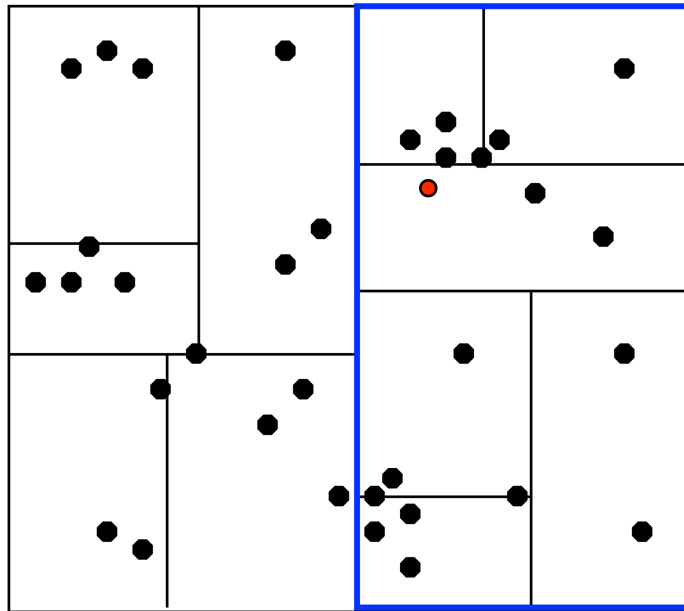


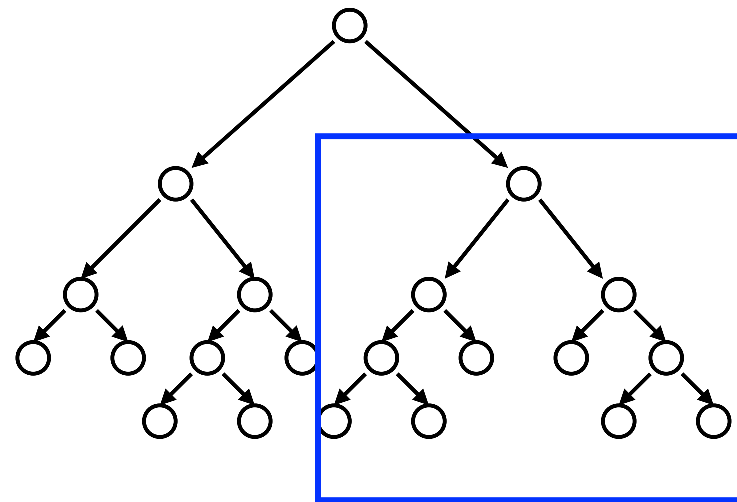


We traverse the tree looking for the nearest neighbor of the query point.

Copyright © 2010 by Yong Cao

Nearest Neighbor with KD Trees

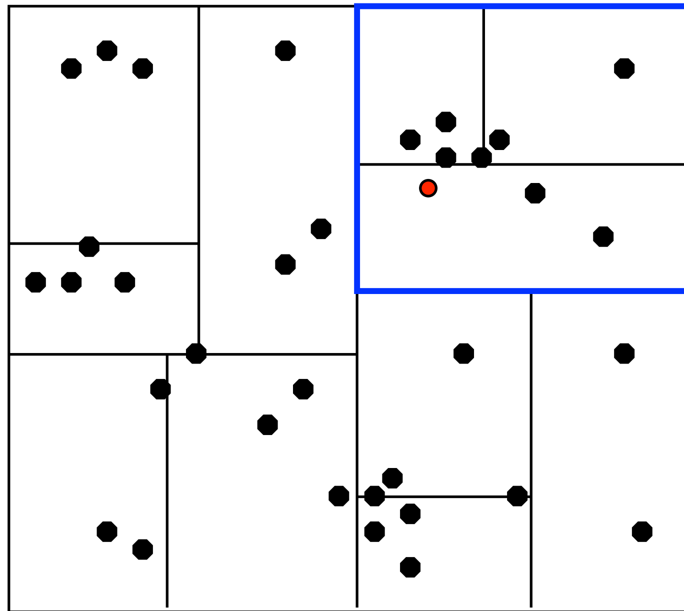


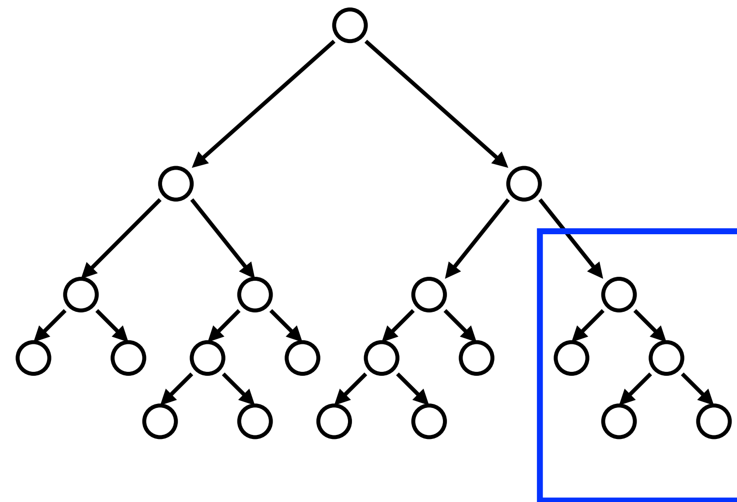


Examine nearby points first: Explore the branch of the tree that is closest to the query point first.

Copyright © 2010 by Yong Cao

Nearest Neighbor with KD Trees

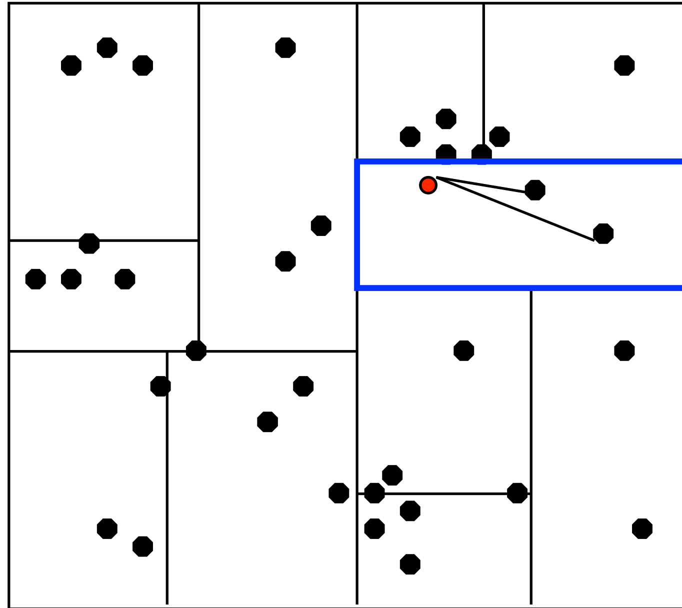


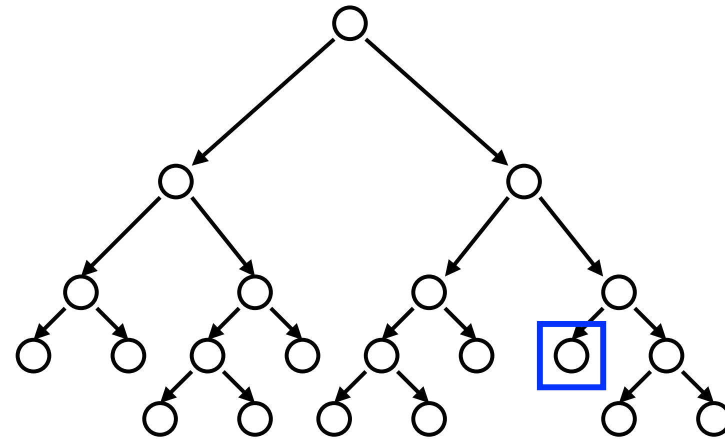


Examine nearby points first: Explore the branch of the tree that is closest to the query point first.

Copyright © 2010 by Yong Cao

Nearest Neighbor with KD Trees

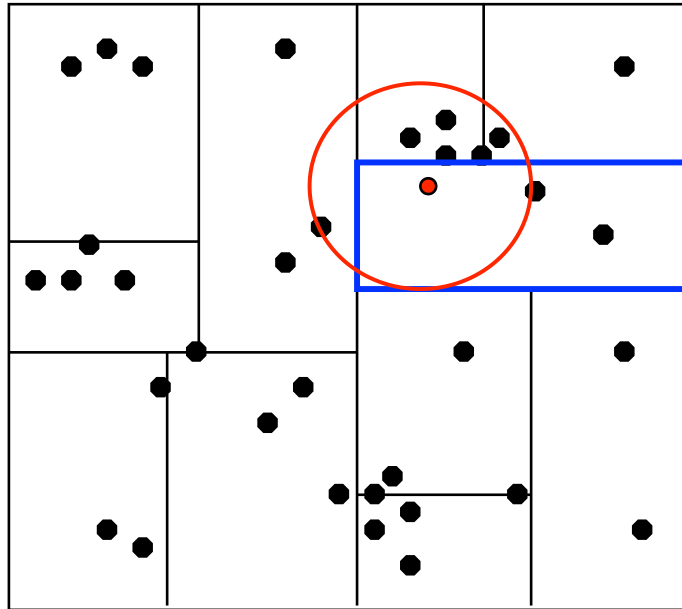


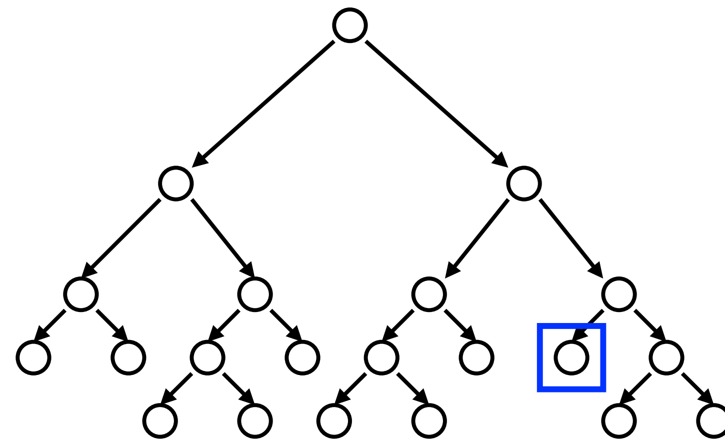


When we reach a leaf node: compute the distance to each point in the node.

Copyright © 2010 by Yong Cao

Nearest Neighbor with KD Trees

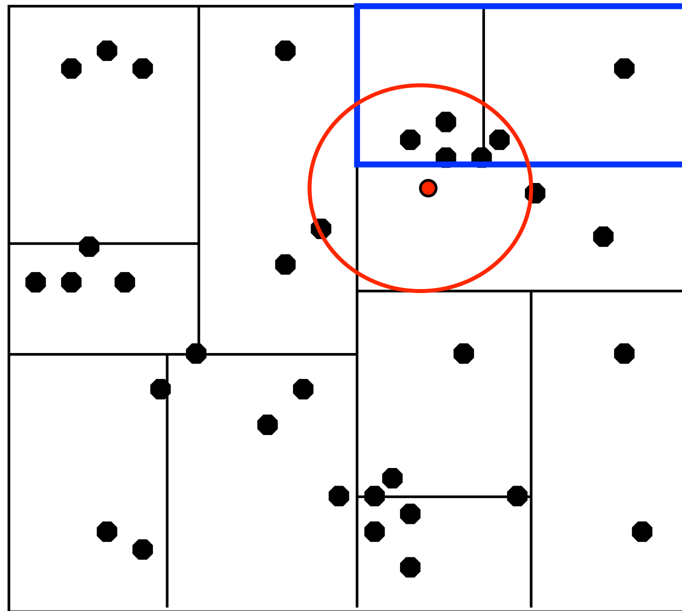


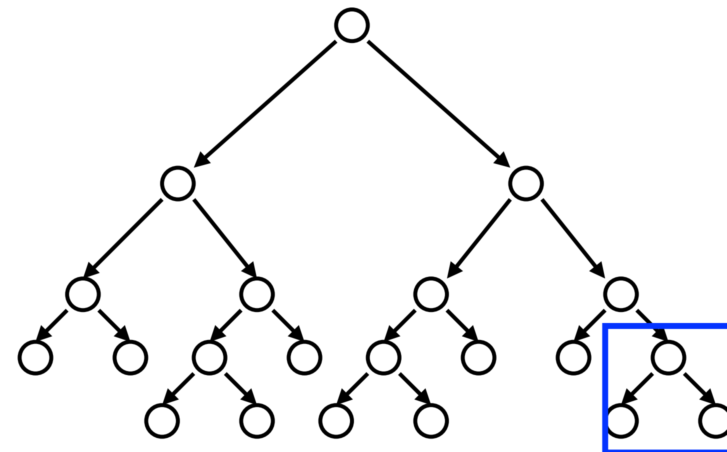


When we reach a leaf node: compute the distance to each point in the node.

Copyright © 2010 by Yong Cao

Nearest Neighbor with KD Trees

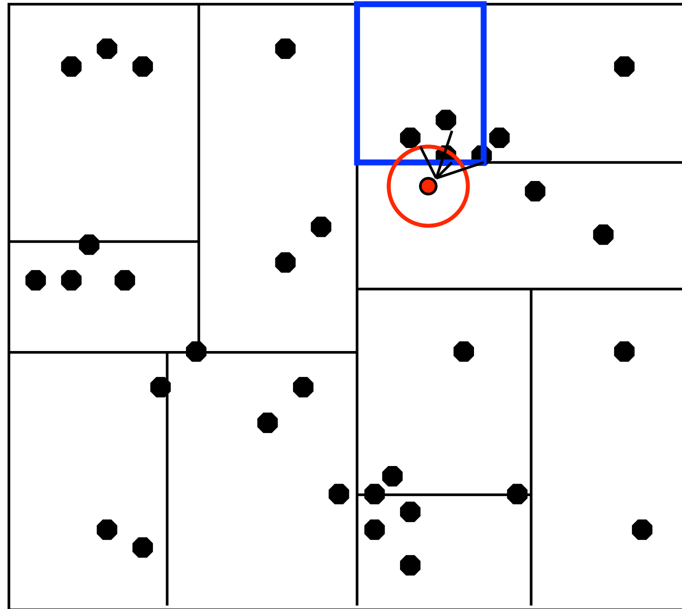


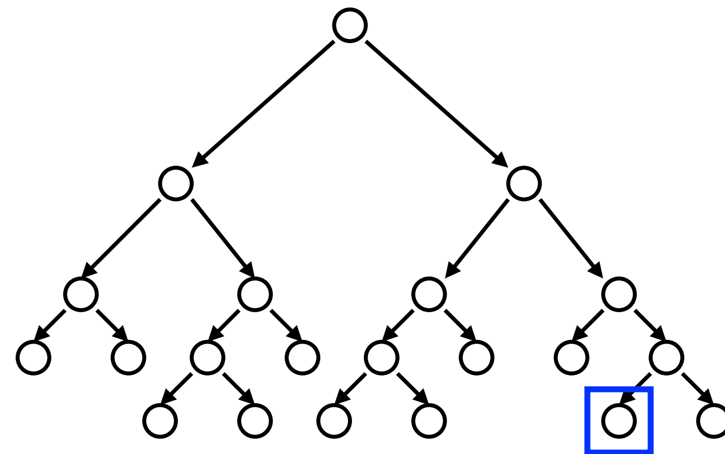


Then we can backtrack and try the other branch at each node visited.

Copyright © 2010 by Yong Cao

Nearest Neighbor with KD Trees

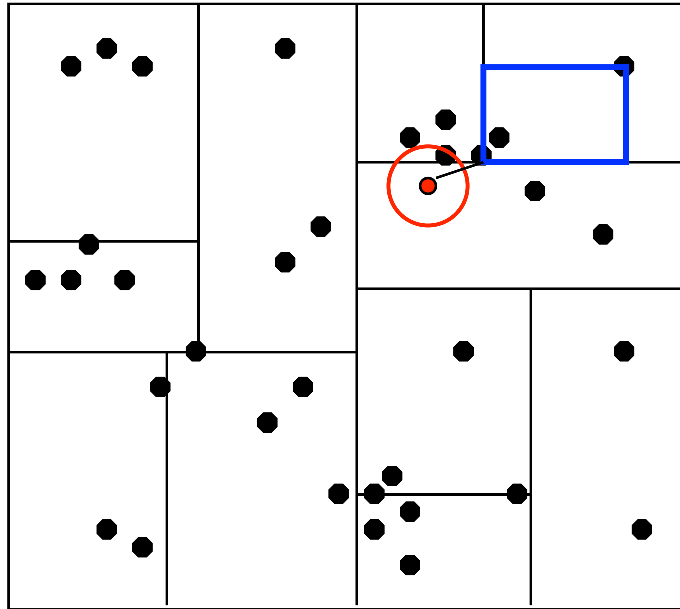


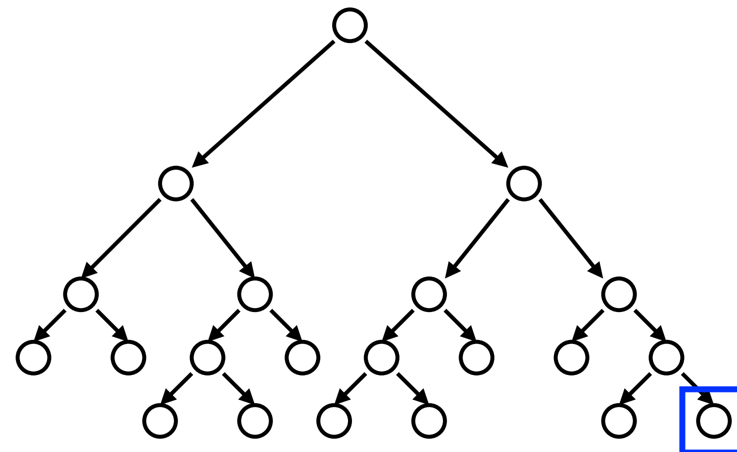


Each time a new closest node is found, we can update the distance bounds.

Copyright © 2010 by Yong Cao

Nearest Neighbor with KD Trees

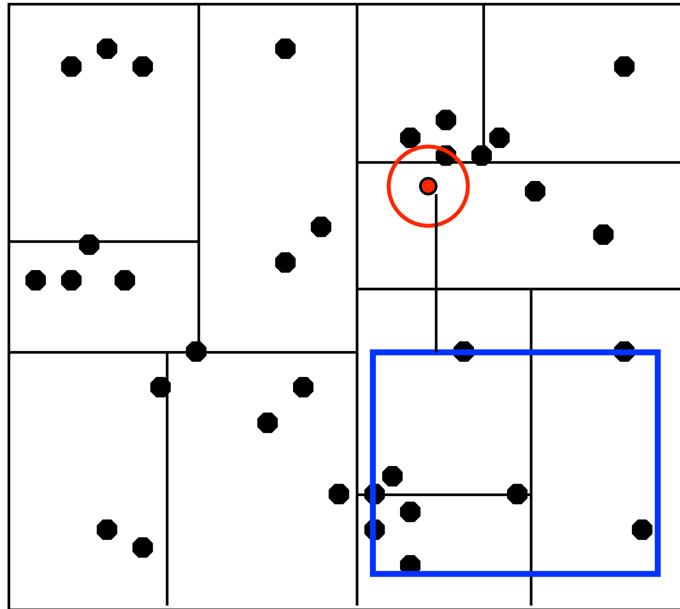


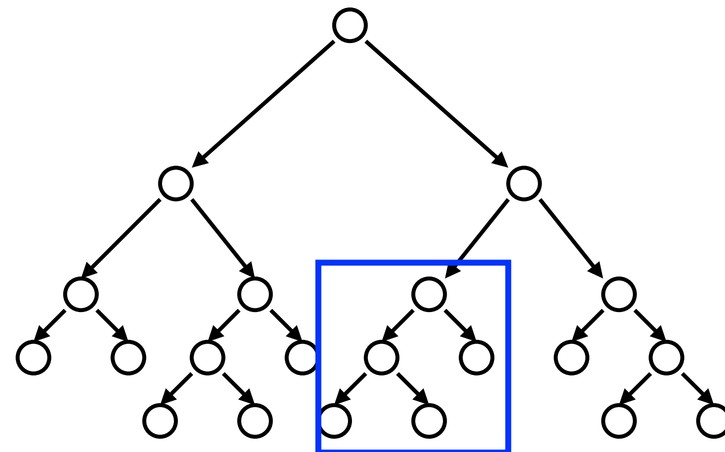


Using the distance bounds and the bounds of the data below each node, we can prune parts of the tree that could NOT include the nearest neighbor.

Copyright © 2010 by Yong Cao

Nearest Neighbor with KD Trees

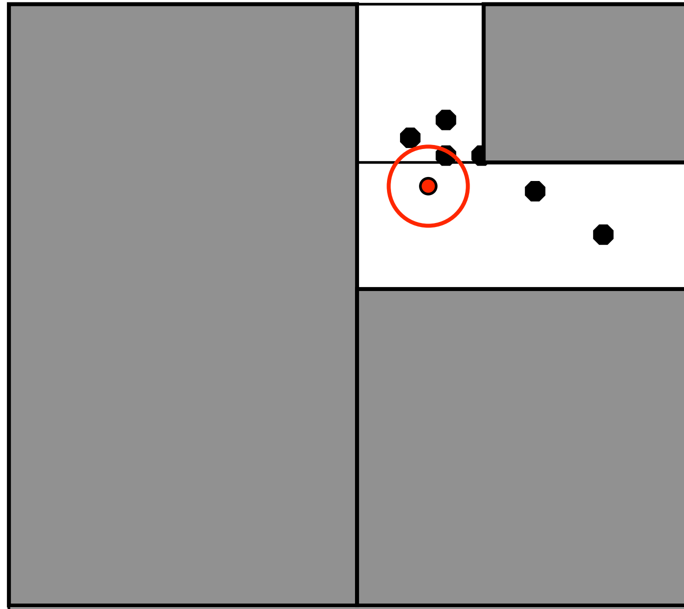


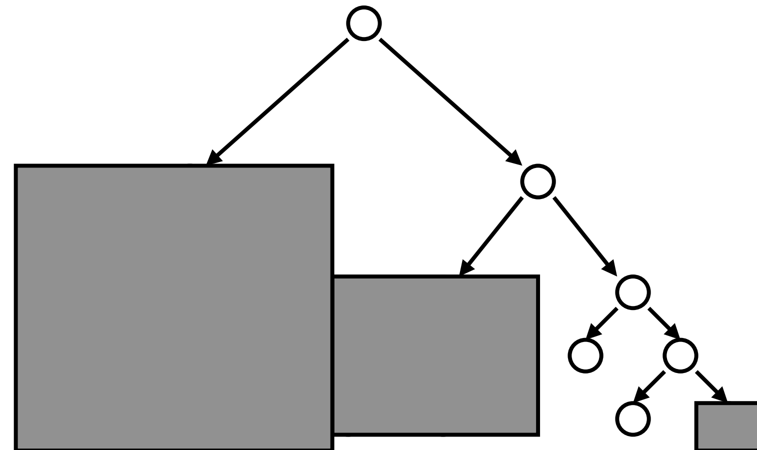


Using the distance bounds and the bounds of the data below each node, we can prune parts of the tree that could NOT include the nearest neighbor.

Copyright © 2010 by Yong Cao

Nearest Neighbor with KD Trees





Using the distance bounds and the bounds of the data below each node, we can prune parts of the tree that could NOT include the nearest neighbor.

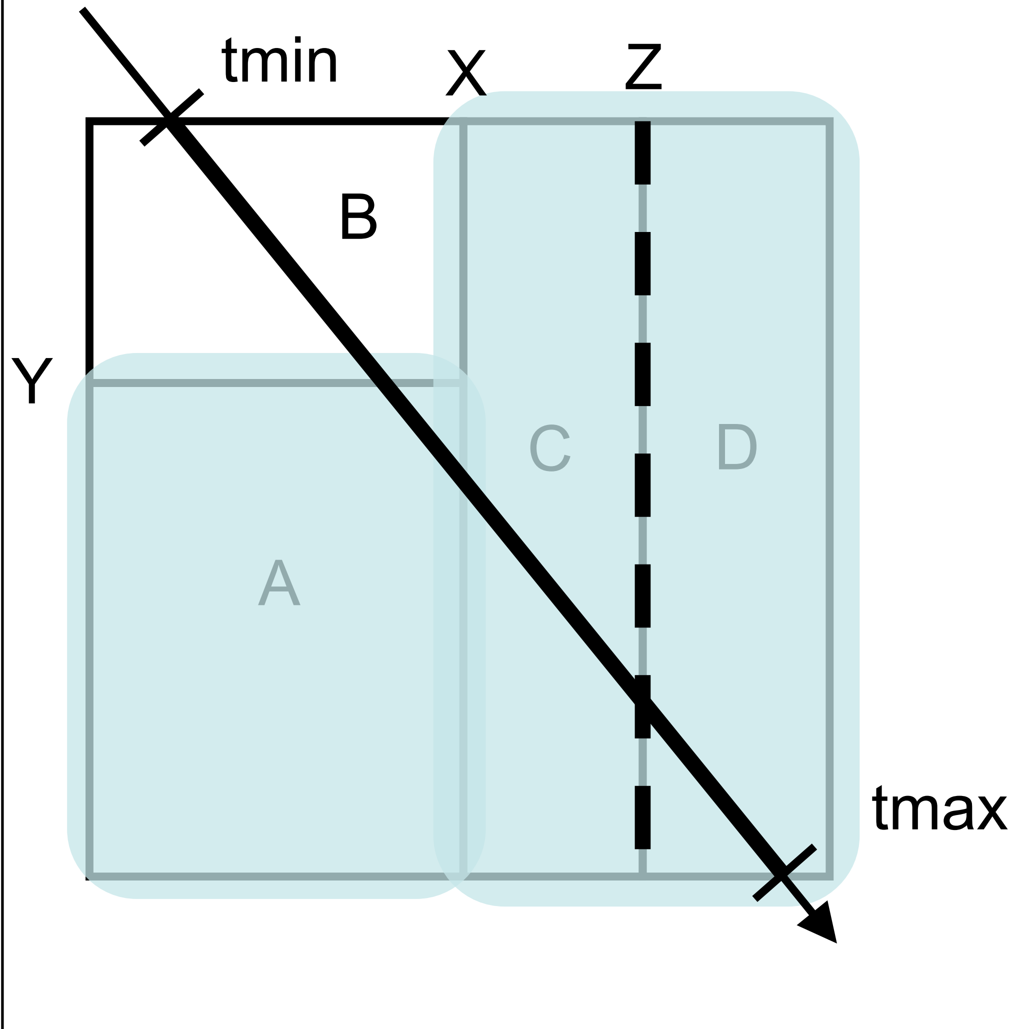
Copyright © 2010 by Yong Cao

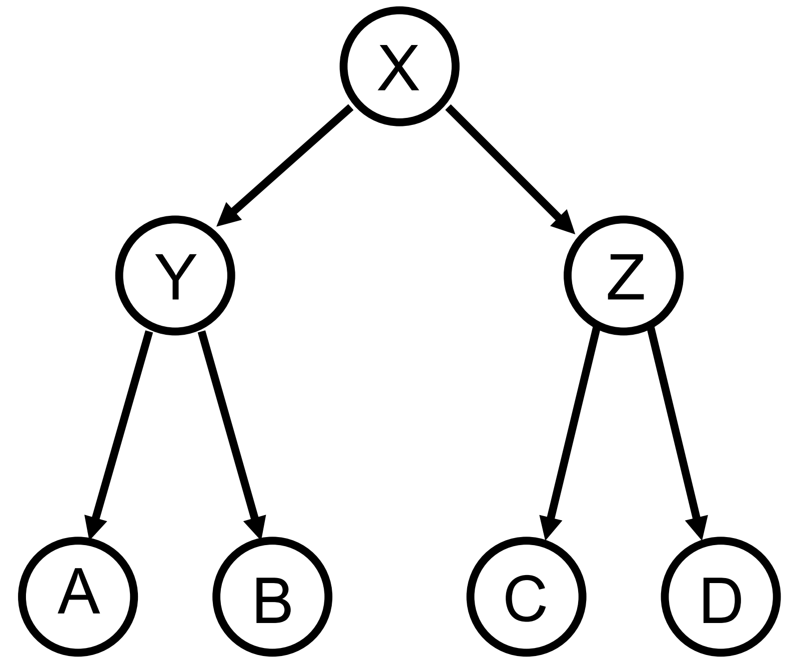
KD-Tree in Ray Tracing

- Query: given a ray r(t), find the primitives that intersected with the ray

Copyright © 2010 by Yong Cao

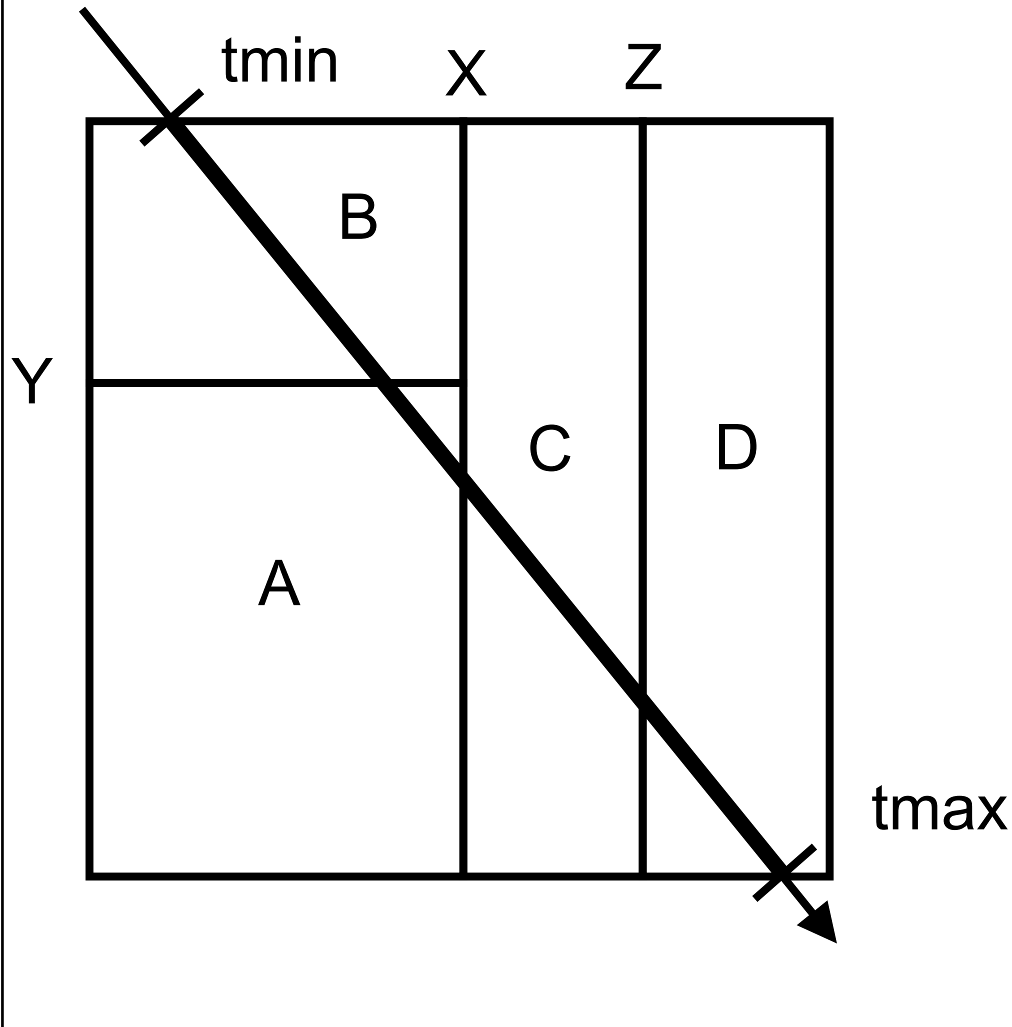
KD-Tree in Ray Tracing

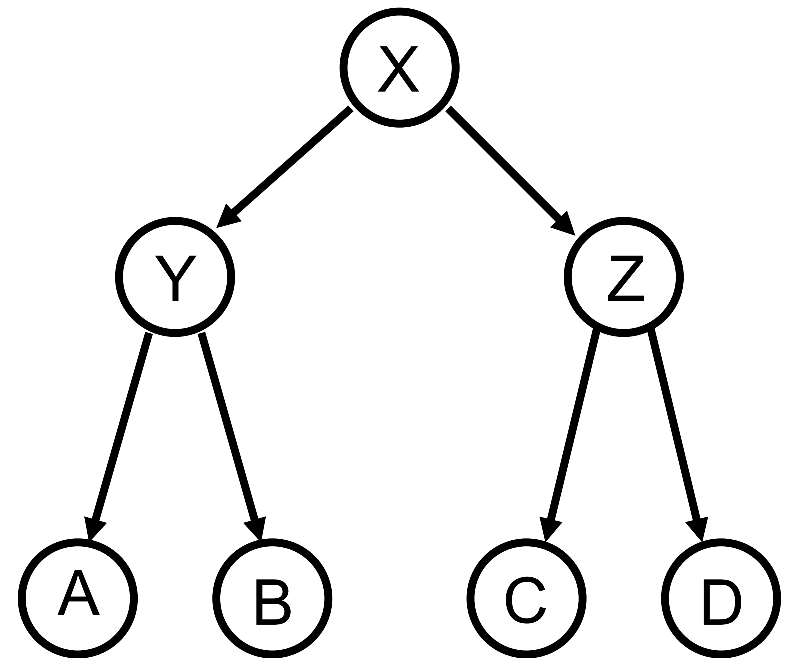




Copyright © 2010 by Yong Cao

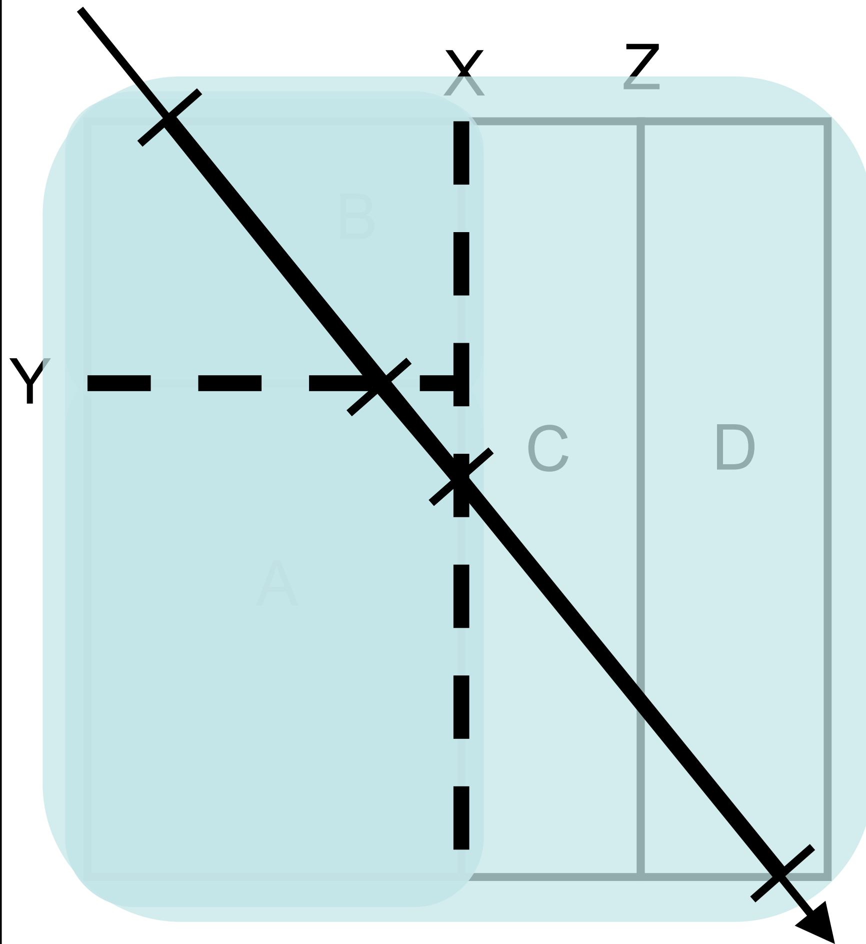
KD-Tree in Ray Tracing

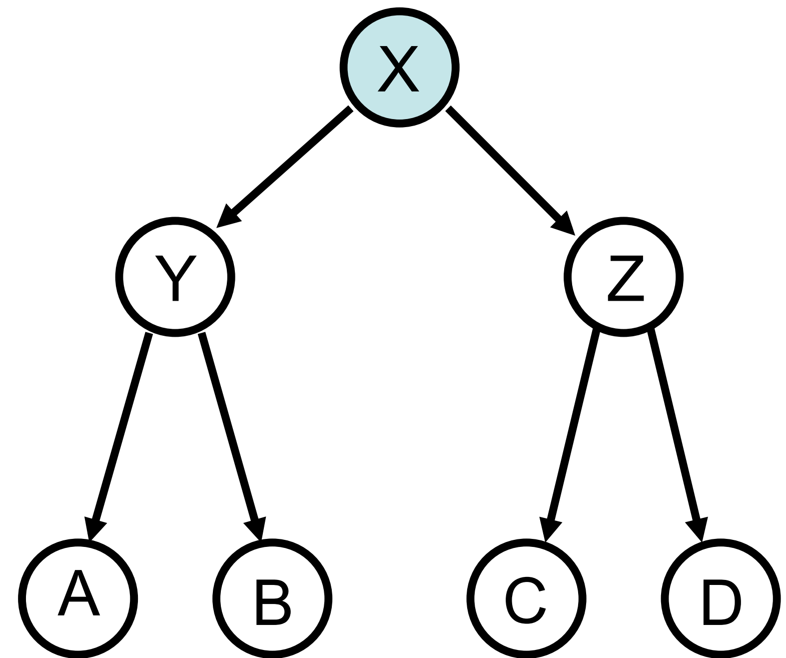




Copyright © 2010 by Yong Cao

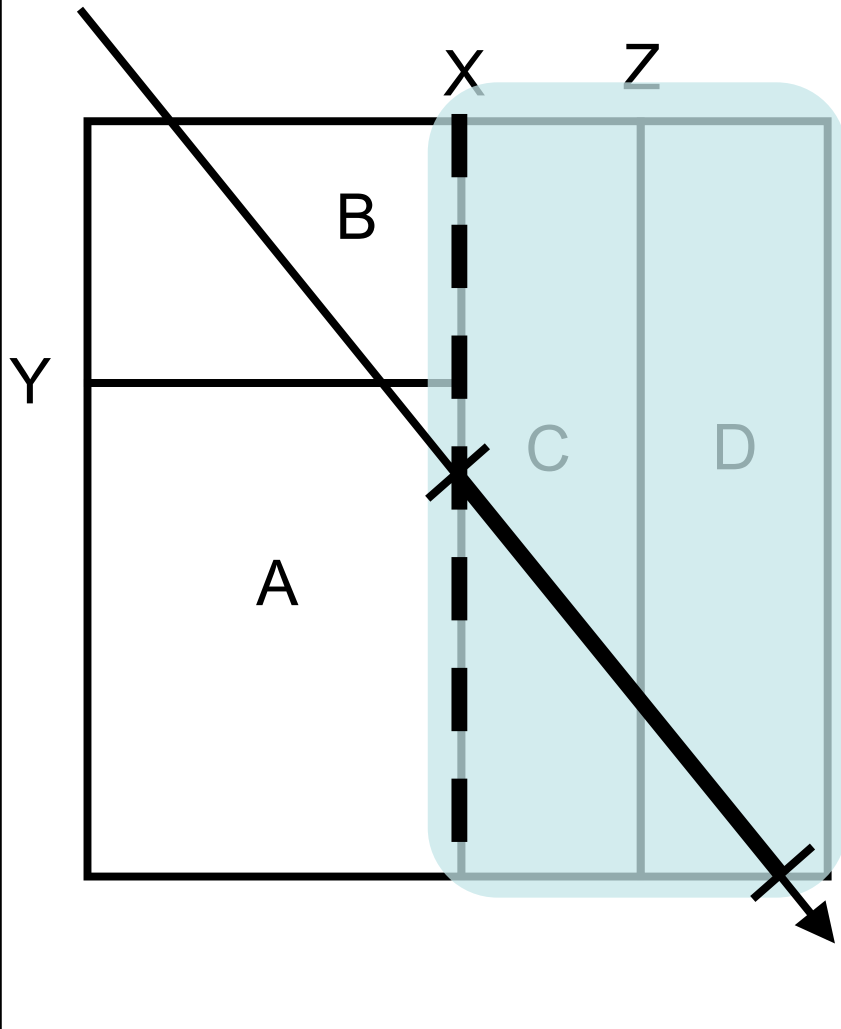
KD-Tree Traversal

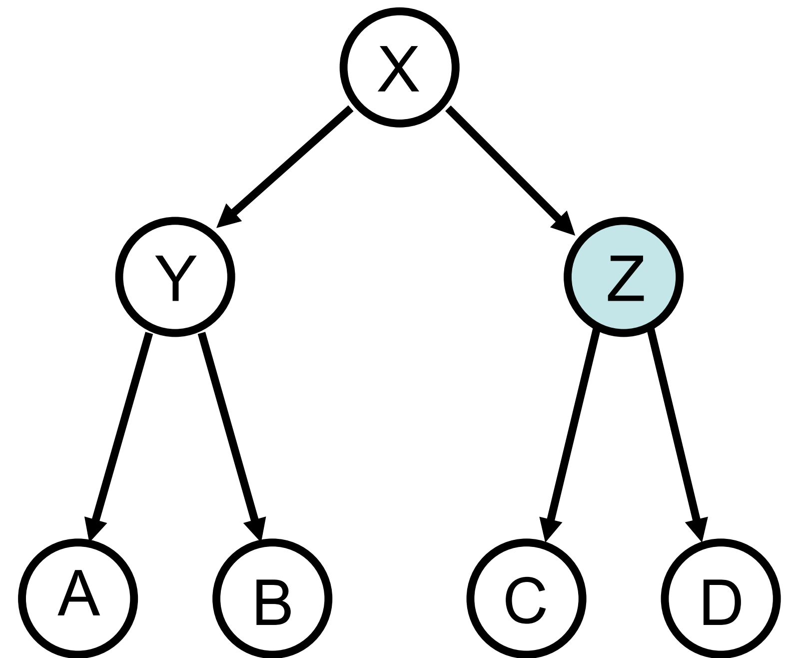




Copyright © 2010 by Yong Cao

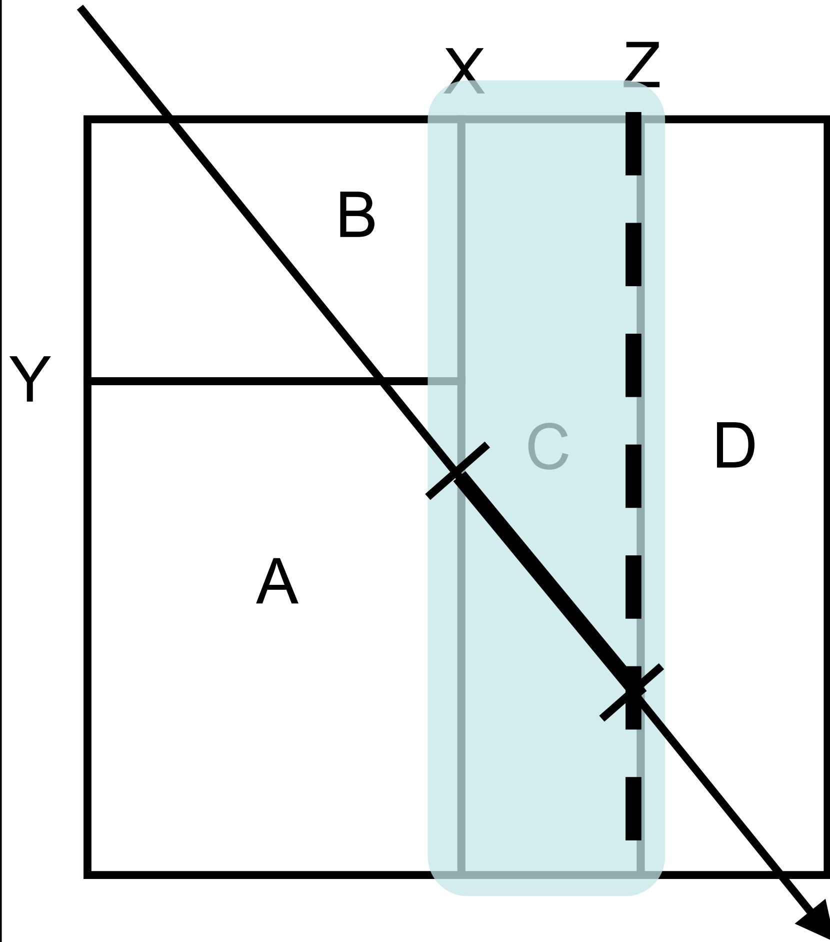
KD-Tree Traversal

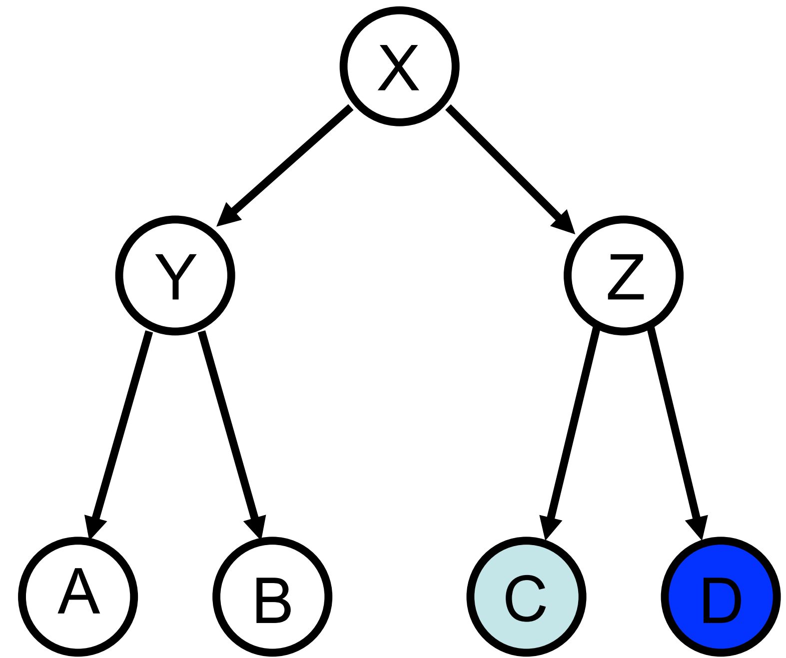




Copyright © 2010 by Yong Cao

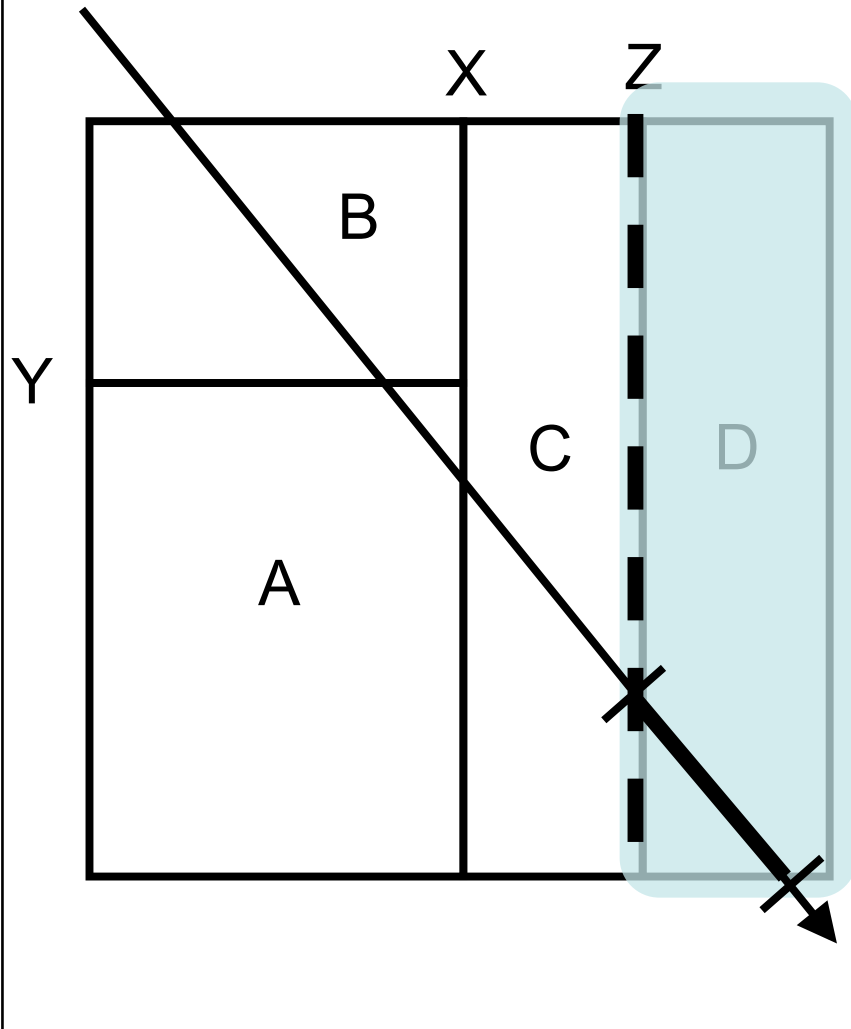
KD-Tree Traversal

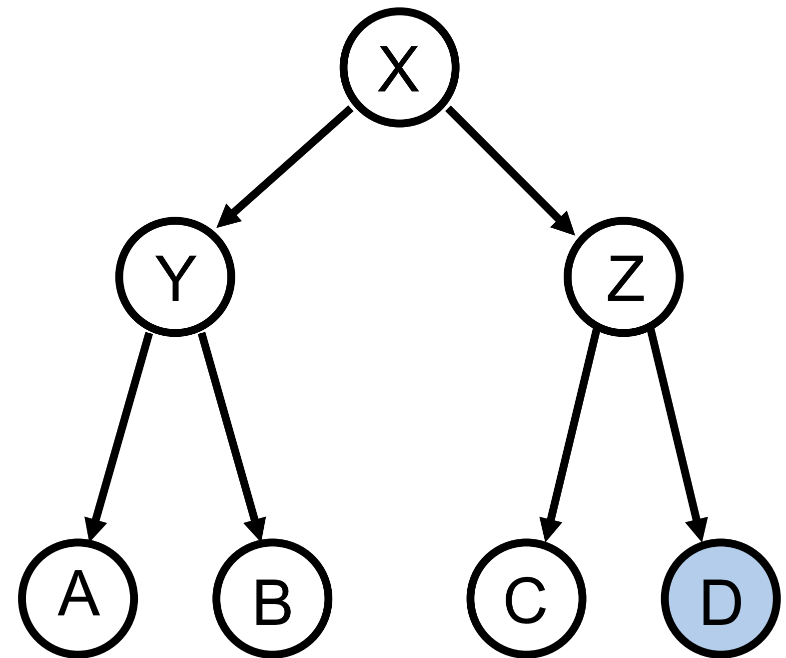




Copyright © 2010 by Yong Cao

KD-Tree Traversal





Copyright © 2010 by Yong Cao

Advantages of kD-Trees

- **Adaptive**
  - Can handle the “Teapot in a Stadium”
    - Octree is not.
- **Compact**
  - Relatively little memory overhead
  - 8 bytes only (function as BSP-tree, not more compact)
- **Cheap Traversal**
  - One FP subtract, one FP multiply

Copyright © 2010 by Yong Cao

Fast Ray Tracing w/ kD-Trees

- Adaptive
- Compact
- Cheap traversal

Copyright © 2010 by Yong Cao

Building kD-trees

- **Given:**
  - axis-aligned bounding box ("cell")
  - list of geometric primitives (triangles?) touching cell
- **Core operation:**
  - pick an axis-aligned plane to split the cell into two parts
  - sift geometry into two batches (some redundancy)
  - recurse
  - termination criteria!

Copyright © 2010 by Yong Cao

"Intuitive" kD-Tree Building

- **Split Axis**
  - Round-robin; largest extent
- **Split Location**
  - Middle of extent; median of geometry (balanced tree)
- **Termination**
  - Target # of primitives, limited tree depth

Copyright © 2010 by Yong Cao

"Hack" kD-Tree Building

- **Split Axis**
  - Round-robin; largest extent
- **Split Location**
  - Middle of extent; median of geometry (balanced tree)
- **Termination**
  - Target # of primitives, limited tree depth
- **All of these techniques stink.**

Copyright © 2010 by Yong Cao

"Hack" kD-Tree Building

- **Split Axis**
  - Round-robin; largest extent
- **Split Location**
  - Middle of extent; median of geometry (balanced tree)
- **Termination**
  - Target # of primitives, limited tree depth
- **All of these techniques stink. Don't use them.**

Copyright © 2010 by Yong Cao

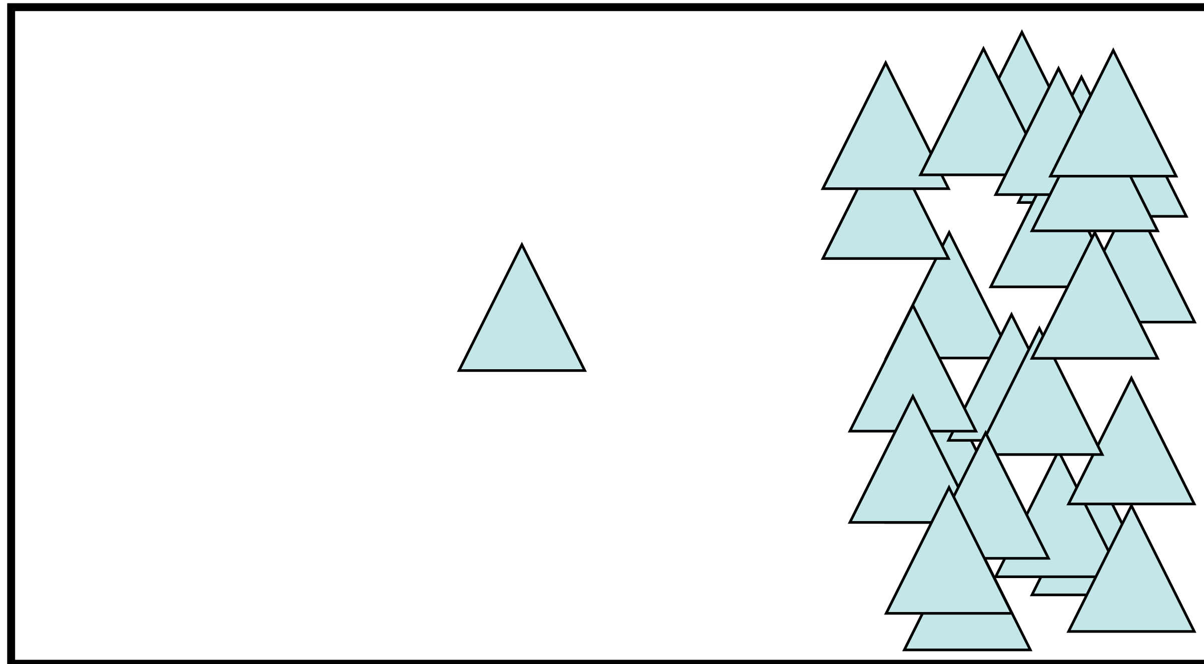
Building good kD-trees

- **What split do we really want?**
  - Clever Idea: The one that makes ray tracing cheap
  - Write down an expression of cost and minimize it
  - *Cost Optimization*
- **What is the cost of tracing a ray through a cell?**

Cost(cell) = C_trav + Prob(hit L) * Cost(L) + Prob(hit R) * Cost(R)

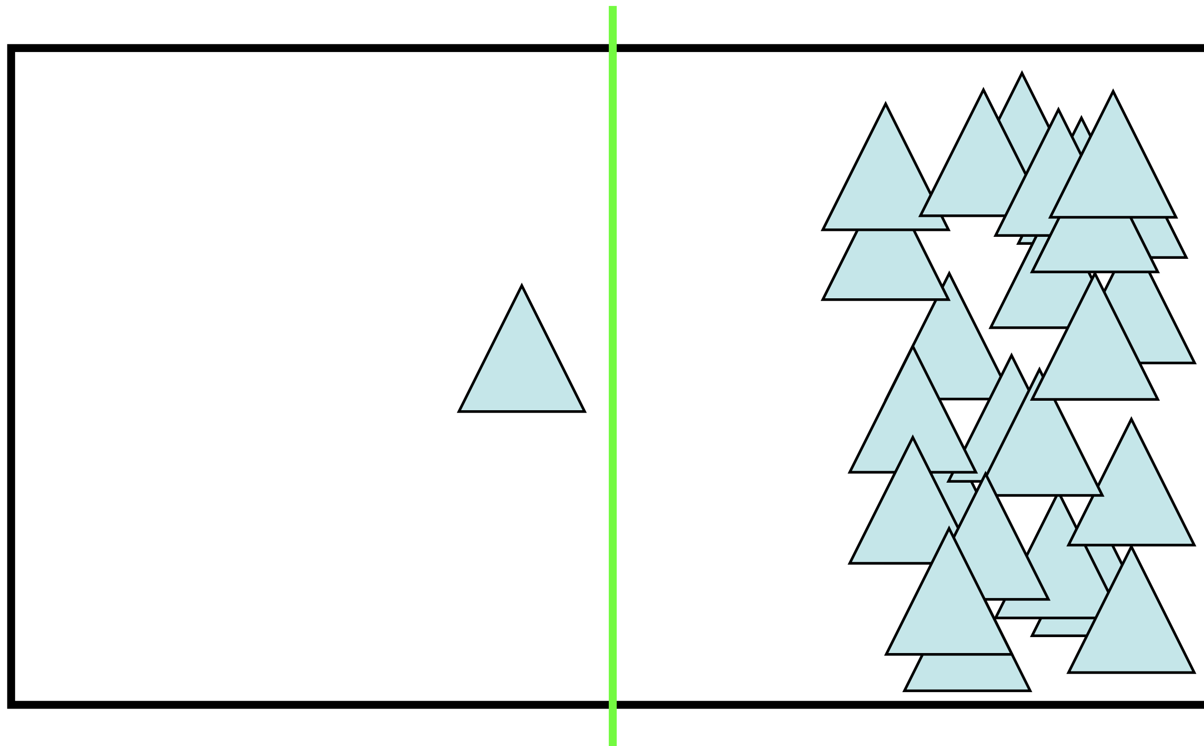
Copyright © 2010 by Yong Cao

Splitting with Cost in Mind



Copyright © 2010 by Yong Cao

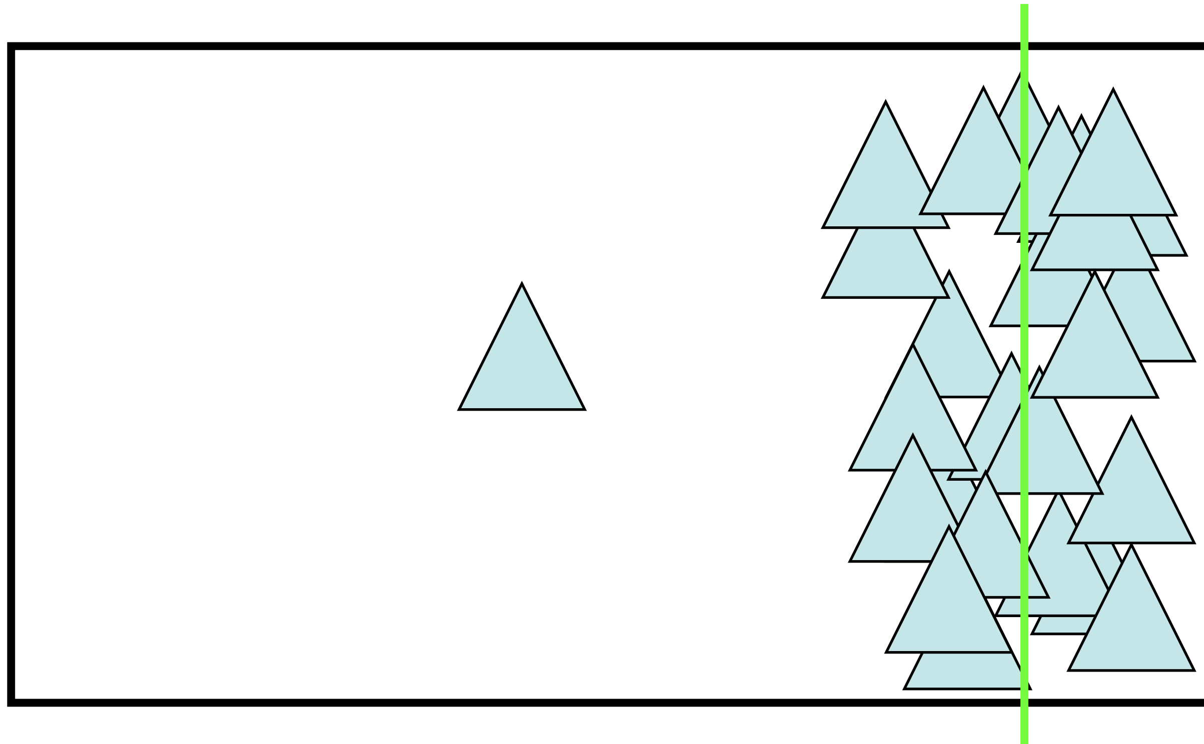
Split in the middle



- Makes the L & R probabilities equal
- Pays no attention to the L & R costs

Copyright © 2010 by Yong Cao

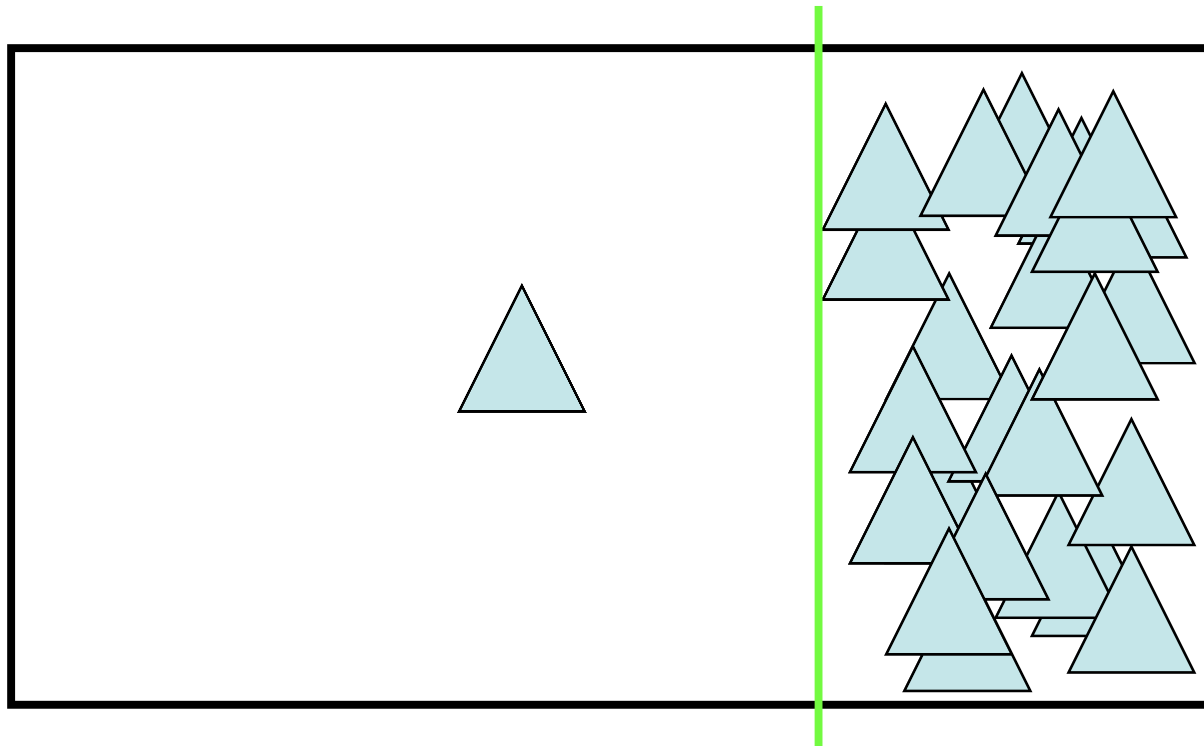
Split at the Median



- Makes the L & R costs equal
- Pays no attention to the L & R probabilities

Copyright © 2010 by Yong Cao

Cost-Optimized Split



- Automatically and rapidly isolates complexity
- Produces large chunks of empty space

Copyright © 2010 by Yong Cao

Building good kD-trees

- **Need the probabilities**
  - Turns out to be proportional to surface area
- **Need the child cell costs**
  - Simple triangle count works great (very rough approx.)

Cost(cell) = C_trav + Prob(hit L) * Cost(L) + Prob(hit R) * Cost(R)

= C_trav + SA(L) * TriCount(L) + SA(R) * TriCount(R)

Copyright © 2010 by Yong Cao

Termination Criteria

- When should we stop splitting?
  - Use the cost estimates in your termination criteria
- Threshold of cost improvement
  - Stretch over multiple levels
- Threshold of cell size
  - Absolute probability so small there’s no point

Copyright © 2010 by Yong Cao

Building good kD-trees

- **Basic build algorithm**
  - Pick an axis, or optimize across all three
  - Build a set of “candidates” (split locations)
    - BBox edges or exact triangle intersections
  - Sort them or bin them
  - Walk through candidates or bins to find minimum cost split
- **Characteristics you’re looking for**
  - “stringy”, depth 50-100, ~2 triangle leaves, big empty cells

Copyright © 2010 by Yong Cao

Fast Ray Tracing w/ kD-Trees

- **adaptive**
  - build a cost-optimized kD-tree w/ the surface area heuristic
- **compact**
- **cheap traversal**

Copyright © 2010 by Yong Cao

What's in a node?

- **A kD-tree internal node needs:**
  - Am I a leaf?
  - Split axis
  - Split location
  - Pointers to children

Copyright © 2010 by Yong Cao

Compact (8-byte) nodes

- **kD-Tree node can be packed into 8 bytes**
  - Leaf flag + Split axis
    - 2 bits
  - Split location
    - 32 bit float
  - Always two children, put them side-by-side
    - One 32-bit pointer

Copyright © 2010 by Yong Cao

Compact (8-byte) nodes

- **kD-Tree node can be packed into 8 bytes**
  - Leaf flag + Split axis
    - 2 bits
  - Split location
    - 32 bit float
  - Always two children, put them side-by-side
    - One 32-bit pointer
- **So close! Sweep those 2 bits under the rug…**

Copyright © 2010 by Yong Cao

No Bounding Box!

- **kD-Tree node corresponds to an AABB**
- **Doesn't mean it has to *contain* one**
  - 24 bytes
  - 4X explosion (!)

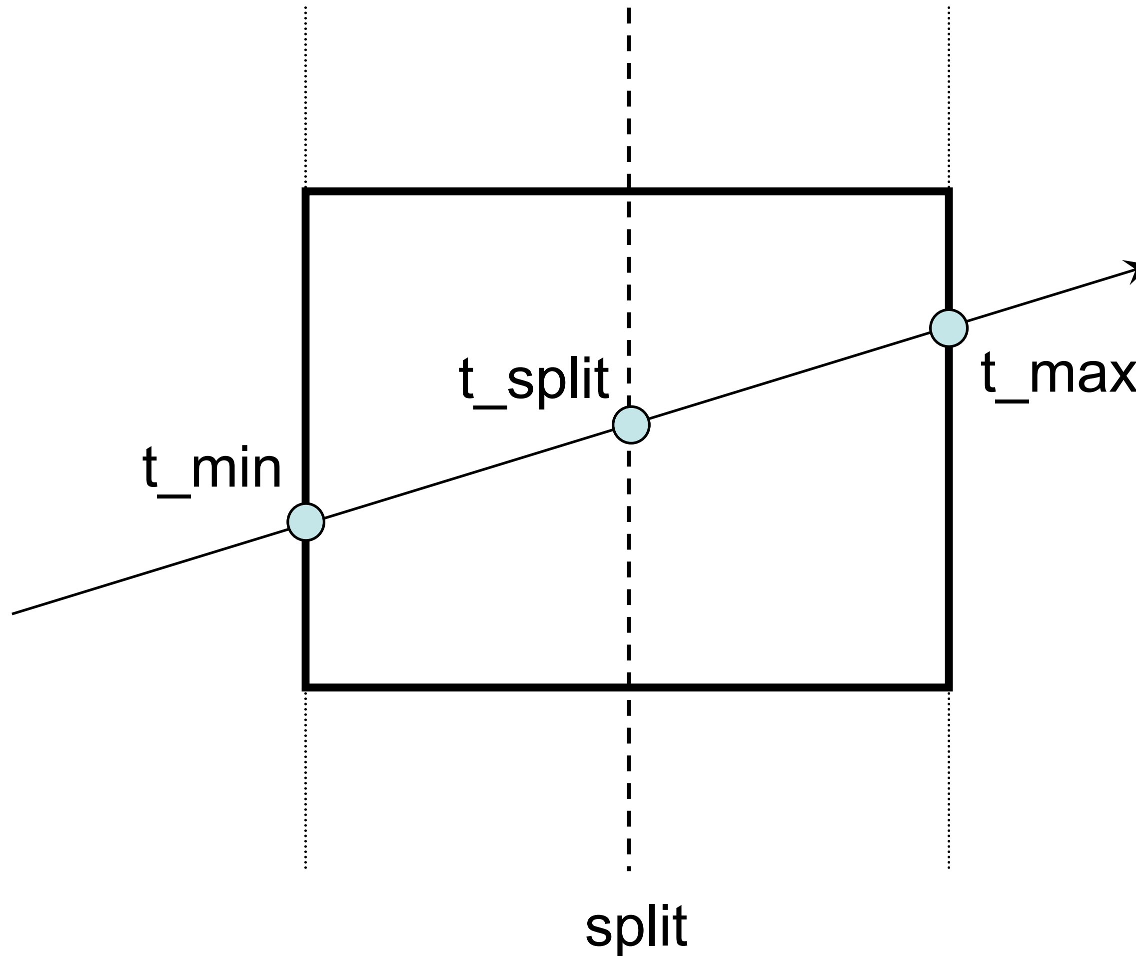
Copyright © 2010 by Yong Cao

Fast Ray Tracing w/ kD-Trees

- **adaptive**
  - build a cost-optimized kD-tree w/ the surface area heuristic
- **compact**
  - use an 8-byte node
  - lay out your memory in a cache-friendly way
- **cheap traversal**

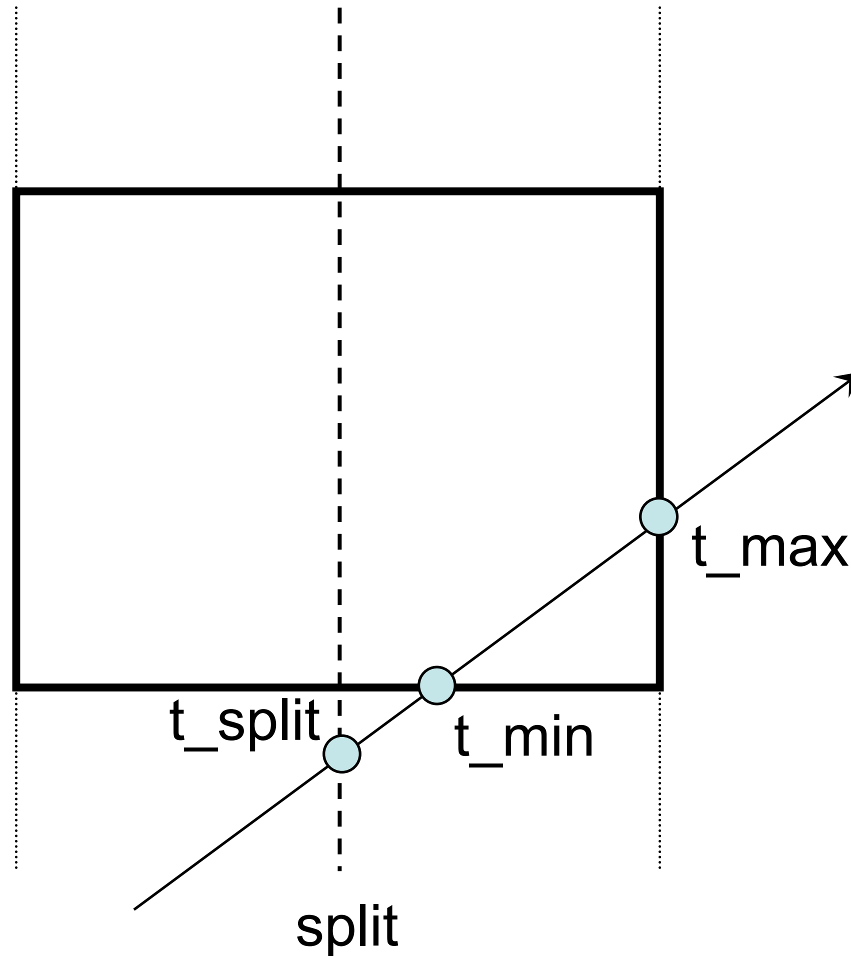
Copyright © 2010 by Yong Cao

kD-Tree Traversal Step



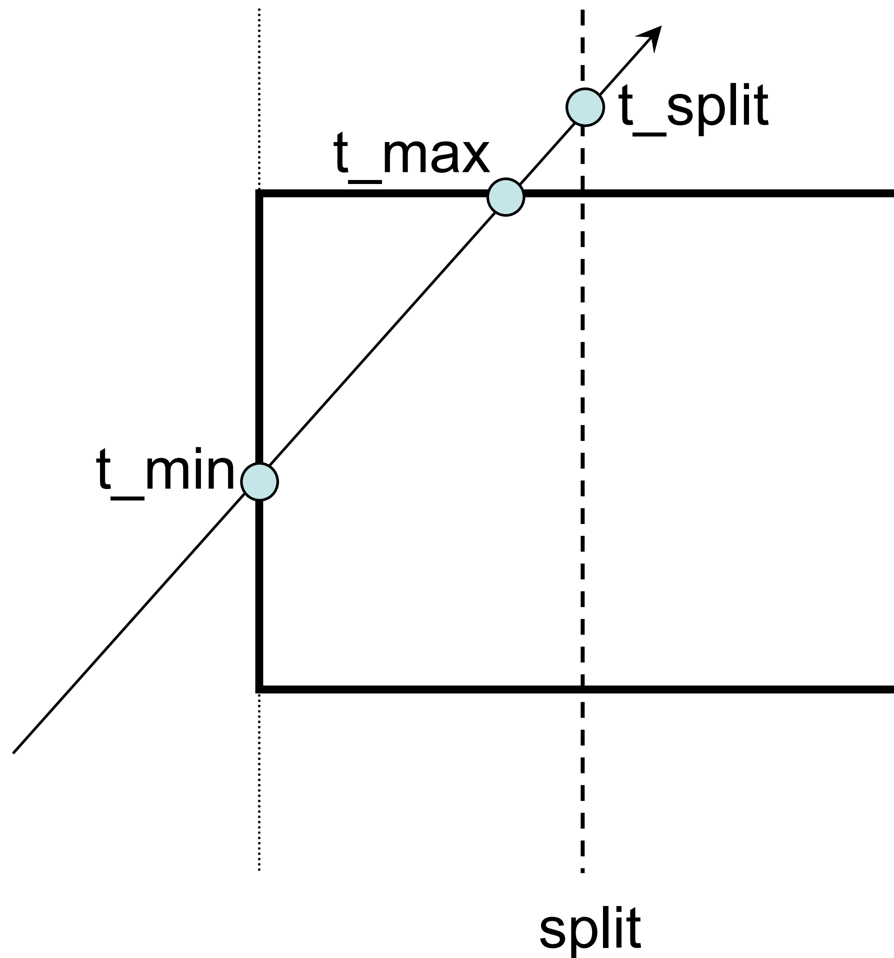
Copyright © 2010 by Yong Cao

kD-Tree Traversal Step



Copyright © 2010 by Yong Cao

kD-Tree Traversal Step



Copyright © 2010 by Yong Cao

kD-Tree Traversal Step

```
Given:  ray P & iV (1/V), t_min, t_max,
   split_location, split_axis

t_at_split = ( split_location - ray->
  P[split_axis] ) * ray_iV[split_axis]

if t_at_split > t_min
  need to test against near child
If t_at_split < t_max
  need to test against far child
```

Copyright © 2010 by Yong Cao

Optimize Your Inner Loop

- **kD-Tree traversal is the most critical kernel**
  - It happens about a zillion times
  - It's tiny
  - Sloppy coding *will* show up
- **Optimize, Optimize, Optimize**
  - Remove recursion and minimize stack operations
  - Other standard tuning & tweaking

Copyright © 2010 by Yong Cao

kD-Tree Traversal

while (not a leaf)

  t_{split} = (split_location - ray->P[split_axis]) * ray_iV[split_axis]

  if t_{split} <= t_{min}

   continue with far child // hit either far child or none

  if t_{split} >= t_{max}

   continue with near child // hit near child only

  // hit both children

  push (far child, t_{split}, t_{max}) onto stack

  continue with (near child, t_{min}, t_{split})

Copyright © 2010 by Yong Cao

Reference

- **Slides from Siggraph 2005 course on "Ray Tracing Performance" by Gordon Stoll**
- **Slides from "KD-Tree Acceleration Structures for a GPU Raytracer" by Tim Foley and Jeremy Sugerman**

Copyright © 2010 by Yong Cao