BSP Trees

This is a small collection of pointers to information on BSP trees.

A binary space partition (BSP) tree is a multidimensional generalization of the binary tree in which space is recursively split by lines, planes, or hyperplanes. It is also a generalization of the k-dimensional (k-d) tree.

Collections

- Bretton Wade's BSP tree FAQ. Casual discussion of the early history, gives general tips on BSP tree construction and use, pointers to other BSP info. Last major update: 1997?
- BSP tree software, index collected by Jeff Erickson.
- BSP tree Java applet by Paton Lewis
- Spatial data structures for optimizing ray tracing. Discussion of BSP trees and other data structures, from the online newsletter *Ray Tracing News*, edited by Eric Haines (indexing help by Paul Heckbert).

Papers and Books

- *Computational Geometry*, Mark de Berg and others, 2000. (undergrad textbook, theory-oriented) Excellent general introduction to BSP trees, especially in 2-D.
- Kinetic binary space partitions for intersecting segments and disjoint triangles, Pankaj K. Agarwal, Jeff Erickson, Leonidas J. Guibas, 1998. (theory paper) Describes randomized algorithms for constructing 3-D BSP trees of O(n log^2 n) expected size. They are dynamic (work well for moving scenes).
- Bruce Naylor's notes for the course *The Computational Representation of Geometry* at SIGGRAPH '94 or '96. Available on CD ROM. You might have trouble finding this (I think I have a copy). Here's the key excerpt: A Tutorial on Binary Space Partitioning Trees (25 page PDF)
- Bruce Naylor's paper *Constructing Good Partitioning Trees*, Graphics Interface '93. Asks the question: what's a good tree? Discusses heuristics for selecting good split planes from the set of triangles, perpendicular to the axes, or in other ways that yield small or balanced trees.
- Michael Abrash has written on Quake's use of BSP trees (a game-oriented perspective). His *Graphics Programming Black Book* is online. Local mirror of that is faster. Also his *Ramblings in Realtime* and his talk for the Computer Game Developers Conference, 1996.
- BSP trees for collision detection by Stan Melax.
- Hanan Samet's books *The Design and Analysis of Spatial Data Structures* and *Applications of Spatial Data Structures* contain a bit on BSP trees and are excellent general references on octrees, quadtrees, k-d trees, and bounding volumes. See also Frantisek Brabec and Samet's very impressive Java applet, for comparing data structures!

Issues

- The amount of splitting that occurs during tree construction greatly affects its memory and time cost.
- Choosing a good split line (plane) at each step to minimize splitting and/or balance the tree is critical.

- When using a BSP tree for a painter's algorithm, typically the entire tree is traversed, so the total size of the tree should be minimized, and balance is unimportant.
- When using a BSP tree for ray tracing, typically several paths from root to leaves are followed, so minimizing the depth of the tree is more critical than minimizing its total size, and hence tree balance is important.

- **Styles of BSP Tree**
  There are several options according to how one answers the questions:
  - How are line segments / triangles / spheres / other objects that span the split line (plane) handled?
    - split and recurse. Subdivide the segment (or whatever) into pieces with the splitter and recurse on the left and right subtrees. This is the standard approach.
    - recurse without splitting. List the full object in both subtrees. Ray tracing this data structure is inefficient unless ray-intersection "mailboxes" are used.
    - store the object at the internal node. This way tree size is linear, but queries on the data structure are less efficient in some (most?) cases.
  - Which lines (planes) are used for splitting?
    - Auto-partition: splitting lines (planes) are extensions of the line segments (triangles) in the input set.
    - General: arbitrary lines (planes) can be used as splitters.
    - Axis-aligned: if the split planes are always perpendicular to the axes, the BSP tree is called a k-d tree. Sometimes people split at the geometric midpoint, or at the median of the vertices along one dimension.
  - When to stop splitting?
    - split until the leaves contain a single line segment (triangle). This is the classic approach.
    - stop sooner, when the leaves contain 10 objects or fewer, say. This might be better, in some situations.

Results

Note that theory folks (the computational geometry community) tend to study auto-partitions a lot, while computer graphics and game people usually don't limit themselves to auto-partitions. When reading "expected" results, be careful to note the statistics of the scenes being analyzed; they may differ significantly from "real" scenes, e.g. lots of possibly big polygons versus a triangulated manifold consisting of lots of small polygons. This difference probably explains the discrepancy between expected results and empirical results. Naylor '93 discusses this issue.

- **optimal tree construction**
  Building the smallest tree is NP-complete; nobody does this in practice. Instead people use heuristics to find approximately-optimal trees.
- **2-D, auto-partition of non-intersecting line segments**
  - **expected size**
    O(n log n). See de Berg's book.
  - **lower bound size**
    Just proven: there are configurations of segments whose BSP tree has superlinear size. In particular, there are sets of line segments where every BSP tree needs Omega((n log n)/log log n) cuts, disproving the long-standing conjecture that O(n) cuts suffice. See paper by Toth will be in the next (2001?) ACM SOCG proceedings.

  - **empirical size**
    O(n)?
- **2-D, auto-partition of non-intersecting "fat" line segments**
  - If the ratio of longest to shortest segment is bounded, a linear size BSP tree exists. See de Berg's papers.
- **3-D, auto-partition of non-intersecting triangles**
  - **expected size**
    Best known algorithm: O(n log^2 n). See Agarwal's paper.
    Older algorithms: O(n^2). See de Berg's book.
  - **lower bound size**
    Omega(n^2). That is, there are some configurations for which the smallest BSP tree is Theta(n^2). See de Berg's book.
  - All known algorithms will sometimes build trees of quadratic size (Theta(n^2)) when a tree of linear size exists. No one knows a fast algorithm for building a BSP tree that is within a constant factor of the optimal size. See Agarwal's paper.
  - **empirical size**
    O(n log n)? See Naylor's GI'93 paper.

I hope I've got the above results right. Send me email if not.

15-463, Computer Graphics 2
Paul Heckbert, 12 Apr. 2001