CS 164 & CS 266: Computational Geometry

Lecture 13

Binary space partitions

David Eppstein
University of California, Irvine

Fall Quarter, 2025

Basic concepts

Depth ordering

Goal: Sort objects in scene from back to front

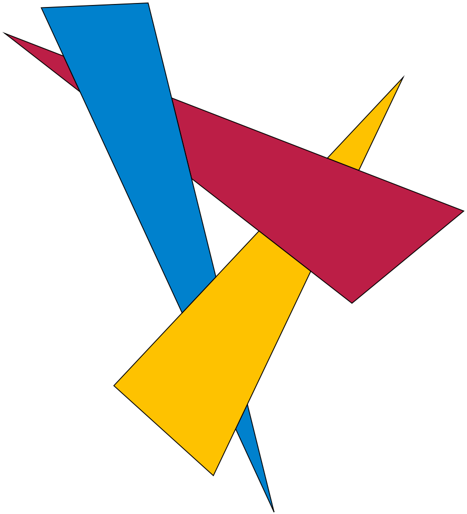
Enables “painter’s algorithm”:

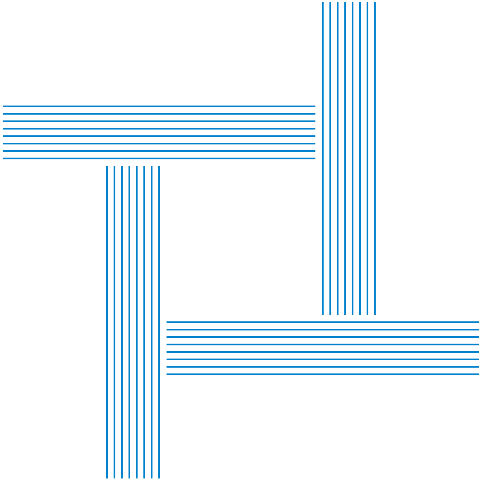
Draw the objects in back-to-front order,
with each one covering up parts of the objects behind it

Necessary for some vector graphics formats

Depth orderings do not always exist!

When there is a cycle in the visibility ordering,
we will need to cut the objects into smaller pieces





The main idea

Recursively cut plane (or 3d) into cells along cut lines (or planes)

Each cell is convex and has two children

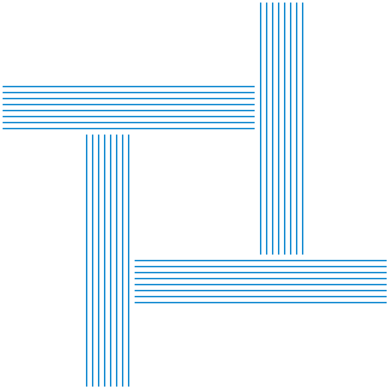
We will use this in a recursive depth ordering:

Scenery in the farther top-level halfplane first

Scenery in the nearer top-level halfplane last

When input = non-crossing line segments

Goal: choose recursive split lines that do not cross any of the segments



Not always possible!

For this input, the first cut must cross a linear # of input segments

So some segments will be subdivided into pieces that participate in multiple tree nodes

⇒ nonlinear complexity

Binary Space Partition

Given non-crossing line segments, recursively cut into convex cells

When a line segment lies on the cut line, it stays at that node

When a line segment is cut, put its pieces into both children

Keep recursing until all pieces of line segments lie on cuts

Applications

Point location?

Simpler than trapezoidal decomposition + history DAG:
follow a path in a tree, checking at each step which side of the cut line to recurse into

Needs a partition for which these paths are short

(true for some BSP constructions but not all)

Uses more space than trapezoidal decomposition: $O(n \log n)$ instead of $O(n)$

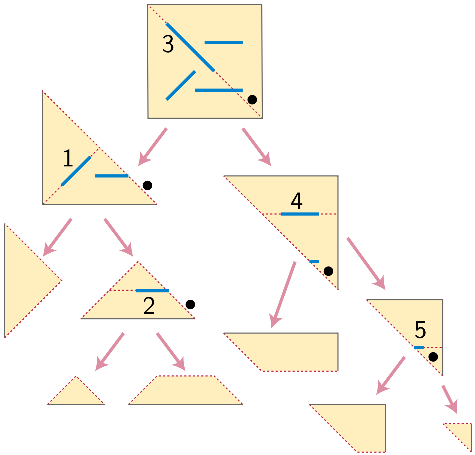
Conclusion: Ok to re-use BSP for point location if you're going to build the BSP for some other reason, but if you only want point location, choose something else

Depth ordering from BSP

2d version of the problem: Given line segments in the plane, and a viewpoint in the plane, find a depth order of pieces of the line segments

For any viewpoint p:

- Compare p to root cut
- Recursively order subtree on far side of cut
- Output pieces that lie on the cut
- Recursively order subtree on near side of cut



$\Rightarrow$ Depth ordering of all pieces

Time $= O(\text{BSP size})$

Constructive solid geometry

Goal: Represent complicated area as intersection or union of simpler shapes (half-spaces)

Any shape whose boundary has been represented by a BSP is union of:

- Shape on one side of cut, intersected with half-space bounded by cut
- Shape on other side of cut, intersected with half-space bounded by cut

Produces union-intersection formula with complexity $=$ BSP size

$2\frac{1}{2}$-dimensional graphics

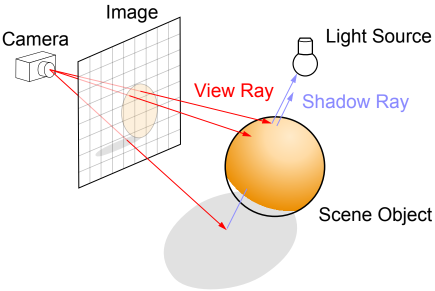
Problem: Render background scenery quickly on slow hardware in first-person games like Wolfenstein 3D (1992), Doom (1993)

"$2\frac{1}{2}$-dimensional": 2d floor plan of rooms, displayed as a three-dimensional scene

Ray tracing

Ray tracing (fully 3D rendering technique):

- Viewpoint = a point in 3d scene
- Each pixel = a ray from viewpoint into scene
- Find first object hit by ray
- Use object color and illumination to determine pixel color

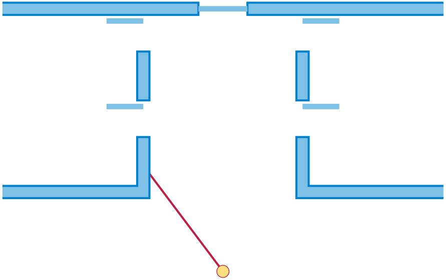


[Henrik 2008]

Ray casting

Ray casting ($2\,\frac{1}{2}$-dimensional):

- Viewpoint = a point in 2D floor plan
- Each column of scene = 2D ray
- Find first object hit by ray
- Look up appearance of entire column of pixels



Ray casting from BSP

To find first object hit by ray in 2d scene:

Recursively search BSP starting at root

At each node, search the same side of the cut as the viewpoint first, then the other side

Stop whenever we find something blocking the ray

Search order automatically prioritizes closer obstacles

No theoretical analysis but works well for mostly-enclosed scenes

Randomized incremental autopartition

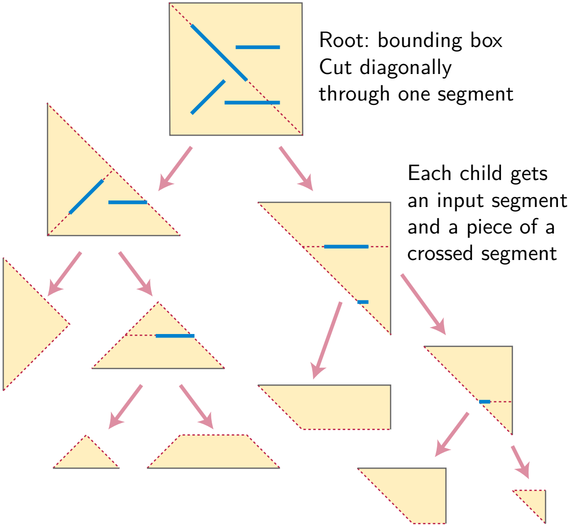
Autopartition

Special kind of BSP

Cut only on lines through input segments

Segments used for cut are removed from children

Recurse until each polygon is empty



Randomized incremental autopartition

Form binary space partition of a subset of segments, adding segments one by one in random order

Initial state: no segments, one node representing bounding box of all segments

For each new segment, recurse down the tree into all cells that contain a piece of the segment

When recursion reaches a leaf of the tree, split it into two along the segment

How big is the tree?

It's a binary tree, so
Tree size $=$ 2 ($\#$ leaf cells) $-$ 1

Splitting a cell on a piece of a segment adds one more leaf, so
Leaf cells $=$ $\#$ pieces of segments $+$ 1

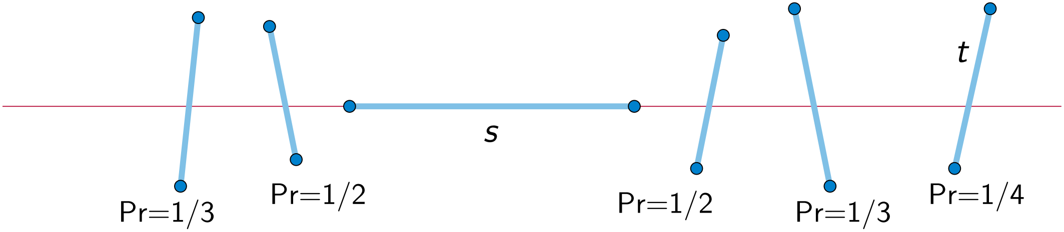
$\#$ pieces in a segment $=$ $\#$ crossings with cut lines $+$ 1, so
total $\#$ pieces $=$ n $+$ total $\#$ crossings

Putting it together:
Tree size $= O$(crossings)

Counting crossings

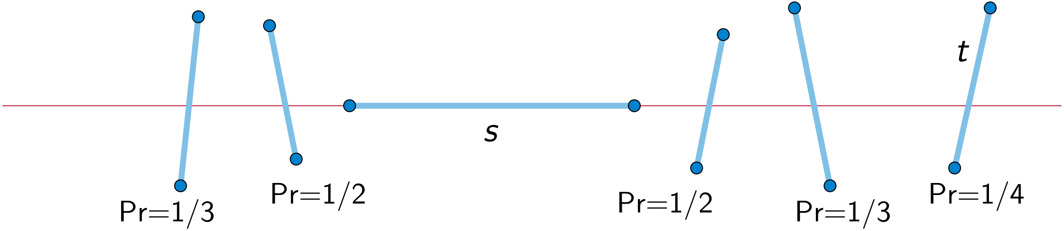
For the cut line L through segment s, another segment t is crossed by L exactly when:

- t extends across line L
- Among s, t, and all segments between them, s is added first



Counting crossings

Probability that s is chosen first, cutting t
$= 1/\#$ segments from s to t

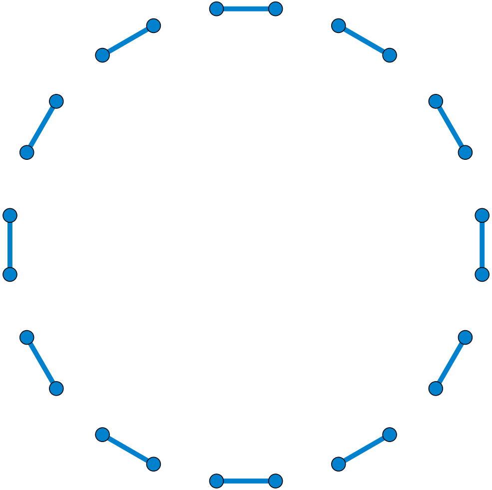


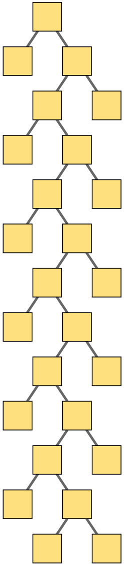
Summing over all segments that extend across the line, expected number of cuts by s
$\leq 2 \sum \frac{1}{i} = O(\log n)$

Total expected size of binary space partition $= O(n \log n)$

Other constructions

Autopartition tree can have linear depth



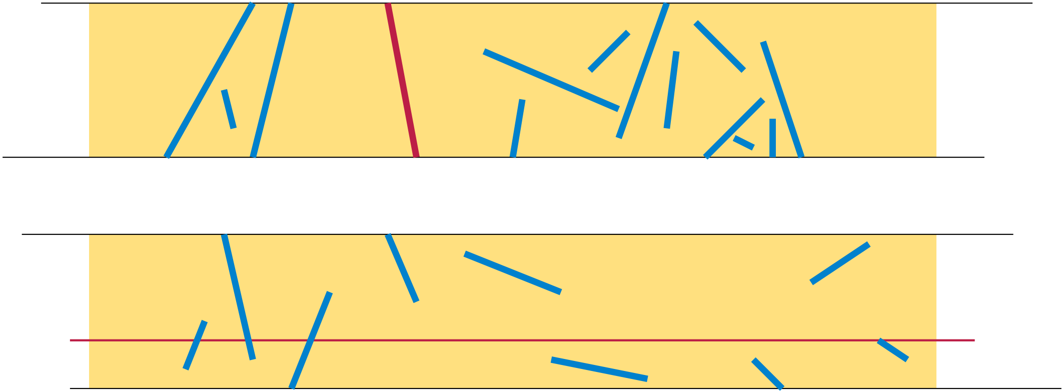


Shallow partition strategy

Recursively subdivide so that each recursive subproblem is a “slab” between two horizontal lines, possibly bounded on the left and right by pieces of input segments

If any piece of a segment extends all the way from the top line of a slab to the bottom line, pick the one that splits the segment endpoints as evenly as possible

Otherwise, split by a horizontal line at the median y-coordinate



Shallow partition analysis

After two levels of subdivision, number of segment endpoints goes down by a factor of two $\Rightarrow$ depth is $\leq 2\log_2 n$

Pieces of segments only split near the endpoints of the segment

Each endpoint partipates in $O(\log n)$ splits
(at most one per level in the tree)

$\Rightarrow$ Total size is $O(n \log n)$

[Paterson and Yao 1990]

Some more results

Optimal size for 2D BSP is $\Theta\left(n\frac{\log n}{\log\log n}\right)$

[Tóth 2011]

Perfect BSP (autopartition that makes no splits),
if it exists, can be found in polynomial time

[de Berg et al. 1997]

Axis-parallel segments $\Rightarrow$ linear-size BSP

3D axis-parallel rectangles $\Rightarrow$ size $O(n^{3/2})$

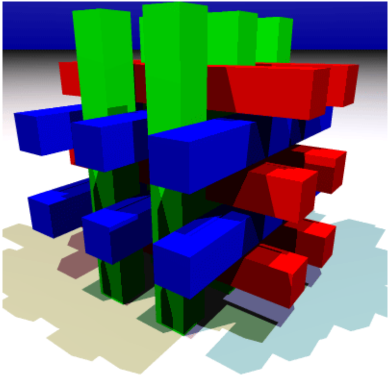
[Paterson and Yao 1992]

3D triangles random autopartition $O(n^2)$ (in textbook)

Well-behaved 3d scenes (no long thin objects) $\Rightarrow$ near-linear

[de Berg 2000; Agarwal et al. 2000; Tóth 2008]

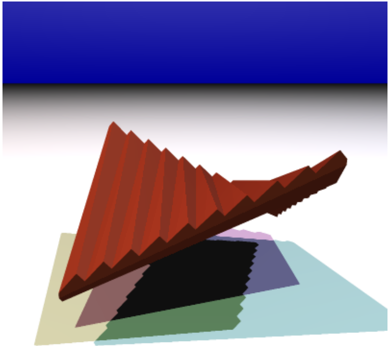
3D lower bounds



Axis-parallel $\Rightarrow \Omega(n^{3/2})$

"Tetrastix"

Each cubical hollow requires a separate piece



Arbitrary $\Rightarrow \Omega(n^2)$

Top and bottom are perpendicular spiral staircases

Interior = grid of solid spaces

References I

Pankaj K. Agarwal, Edward F. Grove, T. M. Murali, and Jeffrey Scott Vitter. Binary space partitions for fat rectangles. *SIAM Journal on Computing*, 29(5):1422–1448, 2000. doi: 10.1137/S0097539797320578.

M. de Berg. Linear size binary space partitions for uncluttered scenes. *Algorithmica*, 28 (3):353–366, 2000. doi: 10.1007/s004530010047.

Mark de Berg, Marko M. de Groot, and Mark H. Overmars. Perfect binary space partitions. *Computational Geometry: Theory and Applications*, 7(1-2):81–91, 1997. doi: 10.1016/0925-7721(95)00045-3.

Henrik. Ray trace diagram. Licensed under the Creative Commons Attribution-Share Alike 4.0 International license, April 12 2008. URL `https://commons.wikimedia.org/wiki/File:Ray_trace_diagram.svg`.

Michael S. Paterson and F. Frances Yao. Efficient binary space partitions for hidden-surface removal and solid modeling. *Discrete & Computational Geometry*, 5 (5):485–503, 1990. doi: 10.1007/BF02187806.

References II

Michael S. Paterson and F. Frances Yao. Optimal binary space partitions for orthogonal objects. *Journal of Algorithms*, 13(1):99–113, 1992. doi: 10.1016/0196-6774(92)90007-Y.

Csaba D. Tóth. Binary space partitions for axis-aligned fat rectangles. *SIAM Journal on Computing*, 38(1):429–447, 2008. doi: 10.1137/06065934X.

Csaba D. Tóth. Binary plane partitions for disjoint line segments. *Discrete & Computational Geometry*, 45(4):617–646, 2011. doi: 10.1007/s00454-011-9341-0.