Basic concepts
The main idea

Recursively cut plane into cells along cut lines
Each cell is a convex polygon with two children

Like kD-tree, but lines might not be axis-parallel
and input might not be a set of points
When input = non-crossing line segments

Goal: separate segments by lines that do not cross them

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
What is a BSP?

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
Depth ordering

Goal: Sort objects in scene from back to front

Enables “painter’s algorithm”: draw the objects in that order, with each one covering up parts of the objects behind it

Necessary for some vector graphics formats

Not always possible!
Depth ordering from BSP

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

⇒ Depth ordering of all pieces

Time $= O$(BSP size)
Constructive solid geometry

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

Convert boundary of a shape into solid volume that it bounds

Any shape 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 $= \text{BSP size}$
2 1/2-dimensional graphics

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

“2 1/2-dimensional”: 2d floor plan of rooms, displayed as a three-dimensional scene
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

Root: bounding box
Cut diagonally through one segment

Each child gets an input segment and a piece of a crossed segment
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 $t$

$= \frac{1}{\#}$ 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 into slabs between horizontal lines
If any segment extends across slab, pick the one closest to median
Otherwise, split by horizontal line at 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 participates 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


