Recap: 2-Dimensional Rectangular Range Searching

Important

- Assume now two points have the same x- or y-coordinate.

- Generalize 1-dimensional idea.
  - 1-dimensional:
    - Split number of points in half at each node.
    - Points consist of one value.
  - 2-dimensional:
    - Points consist of two values.
    - Split number of points in half w.r.t. one value.
    - Switch between values depending on depth.
Motivation

- hidden surface removal
- which pixel is visible
- important for rendering
z-Buffer Algorithm

- Transform scene such that viewing direction is positive $z$-direction
- Consider objects in scene in arbitrary order
- Maintain two buffers
  - Frame buffer $\mathbf{1}$ currently shown pixel
  - $z$-buffer $\mathbf{2}$ $z$-coordinate of object shown
- Compare $z$-coordinate of $z$-buffer and object

- First sort object in depth-order
- Depth-order may not always exist 🎨
- How to efficiently sort objects?
partition space using hyperplanes
- binary partition similar to kd-tree
- hyperplanes create half-spaces and cut objects into fragments

\[ h^+ = \{(x_1, \ldots, x_d): a_1 x_1 + \cdots + a_d x_d > 0\} \]
\[ h^- = \{(x_1, \ldots, x_d): a_1 x_1 + \cdots + a_d x_d < 0\} \]

- each split creates two nodes in a tree
- if number of objects in space is one: leaf
- otherwise: inner node
BSP Trees (2/2)

- for leaf: store object/fragment
- for inner node \( v \): store hyperplane \( h_v \) and the objects contained in \( h_v \)
- left child represents objects in upper half-space \( h^+ \)
- right child represents objects in lower half-space \( h^- \)

- space of BSP tree is number of objects stored at all nodes
- what about fragments?
- too many fragments can make the tree big
Auto-Partitioning

- sorting points for kd-trees worked well
- BSP-tree is used to sort objects in dept-order
- auto-partitioning uses splitters through objects
  - 2-dimensional: line through line segments
  - 3-dimensional: half-plane through polygons
Painter’s Algorithm

- consider view point $p_{\text{view}}$
- traverse through tree and always recurse on half-space that does not contain $p_{\text{view}}$ first
- then scan-convert object contained in node
- then recurse on half-space that contains $p_{\text{view}}$
Constructing Planar BSP Trees (1/3)

- use auto-partitioning
- construction similar to construction of kd-tree
- store all necessary information
  - hyperplane
  - objects in hyperplane
- how to determine next hyperplane?
- creating fragments increases size of BSP tree

- let $s$ be object and $\ell(s)$ line through object
- order matters
Constructing Planar BSP Trees (2/3)

Lemma: Number Line Fragments

The expected number of fragments generated when iterating through the line segments using a random permutation is $O(n \log n)$

Proof (Sketch)

- distance of lines $dist_{s_i}(s_j) =$
  \[
  \begin{cases}
    \# \text{ segments inters. } \ell(s_i) & \text{between } s_i \text{ and } s_j \\
    \ell(s_i) \text{ inters. } s_j & \infty \quad \text{otherwise}
  \end{cases}
  \]

- example on the board

Proof (Sketch, cnt.)

- let $dist_{s_i}(s_j) = k$ and $s_{j_1}, \ldots, s_{j_k}$ be segments between $s_i$ and $s_j$
- what is the probability that $\ell(s_i)$ cuts $s_j$?
- this happens if no $s_{j_x}$ is processed before $s_i$
- since order is random

$$
\mathbb{P}[\ell(s_i) \text{ cuts } s_j] \leq \frac{1}{dist_{s_i}(s_j) + 2}
$$
Constructing Planar BSP Trees (3/3)

Proof (Sketch, cnt.)

- expected number of cuts

\[ \mathbb{E}[\# \text{ cuts generated by } s_i] \leq \sum_{j \neq i} \frac{1}{\text{dist}_{s_i}(s_j) + 2} \leq 2 \sum_{k=0}^{n-2} \frac{1}{k + 2} \leq 2 \ln n \]

- all lines generate at most $2n \ln n$ fragments

Lemma: BSP Construction

A BSP tree of size $O(n \log n)$ can be computed in expected time $O(n^2 \log n)$

Proof (Sketch)

- computing permutation in linear time
- construction is linear in number of fragments to be considered
- number of fragments in subtree is bounded by $n$
- number of recursions is $n \log n$
Conclusion and Outlook

This Lecture
- BSP trees

Next Lecture
- your presentations

Advanced Data Structures
- retroactive PQ
- String B-tree
- SA & LCP
- Kd- & Range Tree
- Successor
- CSA
- RMQ
- static/dynamic BV
- static/dynamic succ. trees
- range min-max tree
- succ. graphs
Recap

- bit vectors
- succinct trees
- dynamic bit vectors and trees
- predecessor and RMQ queries
- suffix array and string B-tree
- compressed suffix array
- persistent data structures
- retroactive data structures
- orthogonal range search
- binary space partitions