Lecture 8: Orthogonal Range Searching
Outline for Today

- Homework 3 Posted
- Last Time: Bounding Spheres & Randomized Incremental Construction
- Motivating Application: Database Queries
- Motivating Application: Graphics & Photon Mapping
- Data Structure Choices
  - Cost to construct, memory to construct, cost to query
- Review: (1D) Binary Search Trees
- 2D kD Trees & Higher dimension kD Trees
- 2D Range Trees & Higher Dimension Range Trees
Homework 3 - CGAL Programming Task

- Compute triangulation of input polygon & triangulation of “pockets” outside input polygon but inside convex hull
- Compute areas
- Compute changes to boundary edges
- Leverage CGAL libraries for convex hull & triangulation

*Computational Geometry Algorithms and Applications*, de Berg, Cheong, van Kreveld and Overmars, Chapter 3
Outline for Today

- Homework 3 Posted
- Last Time: Bounding Spheres & Randomized Incremental Construction
- Motivating Application: Database Queries
- Motivating Application: Graphics & Photon Mapping
- Data Structure Choices
  - Cost to construct, memory to construct, cost to query
- Review: (1D) Binary Search Trees
- 2D kD Trees & Higher dimension kD Trees
- 2D Range Trees & Higher Dimension Range Trees
Problem: Minimal Bounding Sphere Circle

- **Input:** \( n \) vertices in 3D 2D
- **Assume (for convenience):** “General Position”
  - No 3 points are collinear
  - No 4 points lie on the same circle
- **Output:** 3 of those vertices uniquely define a circle such that all other points lie inside of that circle

*Note: In 3D, we would output 4 vertices (4 vertices uniquely define a sphere)*
Randomized Incremental Construction

- We start with all $n$ points and the optimal minimal bounding circle, which is defined by 3 of those points.
- Each step, we randomly choose one of $n$ points to remove.
Outline for Today

● Homework 3 Posted
● Last Time: Bounding Spheres & Randomized Incremental Construction
● Motivating Application: Database Queries
● Motivating Application: Graphics & Photon Mapping
● Data Structure Choices
  ● Cost to construct, memory to construct, cost to query
● Review: (1D) Binary Search Trees
● 2D kD Trees & Higher dimension kD Trees
● 2D Range Trees & Higher Dimension Range Trees
Motivating Application: 2D Database Queries

- Return all data points with $x$ value in range $[x_0, x_1]$ and $y$ value in range $[y_0, y_1]$.

Find all values in an axis parallel box: a "rectangular range query" a.k.a. "orthogonal range query".

*Computational Geometry Algorithms and Applications, de Berg, Cheong, van Kreveld and Overmars, Chapter 5*
Higher Dimensional Database Queries

- Return all data points with $x$ value in range $[x_0, x_1]$ and $y$ value in range $[y_0, y_1]$ and $z$ value in range $[z_0, z_1]$ and …

Find all values in an axis parallel box: a “rectangular range query” a.k.a. “orthogonal range query”

*Computational Geometry Algorithms and Applications, de Berg, Cheong, van Kreveld and Overmars, Chapter 5*
Outline for Today

● Homework 3 Posted
● Last Time: Bounding Spheres & Randomized Incremental Construction
● Motivating Application: Database Queries
● Motivating Application: Graphics & Photon Mapping
● Data Structure Choices
  ● Cost to construct, memory to construct, cost to query
● Review: (1D) Binary Search Trees
● 2D kD Trees & Higher dimension kD Trees
● 2D Range Trees & Higher Dimension Range Trees
Motivating Application: Photon Mapping

- Photons bounce around room and stored on each surface they hit
Using Photon Map for Rendering

- Find the tightest sphere capturing $k$ photons
- Divide the energy from those photons by the surface area covered by that sphere
- What is the best data structure to store millions of photons?
Data Structures Homework 8: Quad Tree

input points

after adding the 1\textsuperscript{st} point

after adding the 2\textsuperscript{nd} point

after adding the 3\textsuperscript{rd} point

after adding 9 points

after adding all 21 points
Collecting Photons from a $kd$ tree

- Query point, and initial guess for radius (red)
- Make a rectangular/orthogonal query to the $kd$ tree (yellow)
- $kd$ tree returns all cells that overlap with query box (blue)
- Further processing necessary to filter points inside red circle and find smallest circle capturing exactly $k$ photons
Outline for Today

- Homework 3 Posted
- Last Time: Bounding Spheres & Randomized Incremental Construction
- Motivating Application: Database Queries
- Motivating Application: Graphics & Photon Mapping
- Data Structure Choices
  - Cost to construct, memory to construct, cost to query
- Review: (1D) Binary Search Trees
- 2D kD Trees & Higher dimension kD Trees
- 2D Range Trees & Higher Dimension Range Trees
Outline for Today

● Homework 3 Posted
● Last Time: Bounding Spheres & Randomized Incremental Construction
● Motivating Application: Database Queries
● Motivating Application: Graphics & Photon Mapping
● Data Structure Choices
  ● Cost to construct, memory to construct, cost to query
● Review: (1D) Binary Search Trees
● 2D kD Trees & Higher dimension kD Trees
● 2D Range Trees & Higher Dimension Range Trees
Review: 1 Dimensional Binary Search Trees

- Everything to the left is $\leq$ root
- Everything to the right is $> root

![Binary Search Tree Diagram]

*Computational Geometry Algorithms and Applications*, de Berg, Cheong, van Kreveld and Overmars, Chapter 5
Assumptions

- No 2 data points have the same value in any dimension
  … for convenience, there are workarounds

- We are given all of the data points at the start, allowing us to sort the data and construct well-balanced trees
1D BST Construction Algorithm

Computational Geometry Algorithms and Applications,
de Berg, Cheong, van Kreveld and Overmars, Chapter 5
1D BST Construction Algorithm

- Sort the data by x value
- Put the median (middle) value at the root
- Create 2 sublists for left & right
- Recurse
1D BST Query Algorithm

- Given a desired range $[\mu, \mu']$
1D BST Query Algorithm

- Given a desired range $[\mu, \mu']$
- Locate the leaf storing $\mu \rightarrow O(\log n)$
- Locate the leaf storing $\mu' \rightarrow O(\log n)$
- Increment from $\mu \rightarrow \mu'$
  - $\text{Operator++} \rightarrow O(1) \text{ expected time}$
  - $\text{Operator++ from } \mu \rightarrow \mu' \rightarrow k \cdot O(1) = O(k) \text{ expected}$
1D BST Query Algorithm

- Given a desired range \([\mu, \mu']\)
- Locate the leaf storing \(\mu\) \(\rightarrow O(\log n)\)
- Locate the leaf storing \(\mu'\) \(\rightarrow O(\log n)\)
- Increment from \(\mu \rightarrow \mu'\)
  - \textit{Operator++} \(\rightarrow O(1)\) expected time
  - \textit{Operator++} from \(\mu \rightarrow \mu'\)
    \(\rightarrow k \cdot O(1) = O(k)\) expected
- \textit{Equivalently}: Find all subtrees between the leaves, return all values in those subtrees

\[\text{Computational Geometry Algorithms and Applications, de Berg, Cheong, van Kreveld and Overmars, Chapter 5}\]
Analysis: 1D Binary Search Tree

Starting with $n$ values...

- Memory to store:
  - # of leaf nodes:
  - # of intermediate nodes:
  - Height of tree:
- Time to construct:
  - Sort the data:
  - Place middle value at root, recurse on left & right sublists:
- Time to query:
  - For search target / output returning $k$ values
Analysis: 1D Binary Search Tree

Starting with $n$ values...

- Memory to store: $O(n)$
  - # of leaf nodes: $n$
  - # of intermediate nodes: $n-1$
  - Height of tree: $\log n$
- Time to construct: $O(n \log n)$
  - Sort the data: $O(n \log n)$
  - Place middle value at root, recurse on left & right sublists: $O(n)$
- Time to query: $O(\log n + k)$
  - For search target / output returning $k$ values
Outline for Today

- Homework 3 Posted
- Last Time: Bounding Spheres & Randomized Incremental Construction
- Motivating Application: Database Queries
- Motivating Application: Graphics & Photon Mapping
- Data Structure Choices
  - Cost to construct, memory to construct, cost to query
- Review: (1D) Binary Search Trees
  - 2D kD Trees & Higher dimension kD Trees
  - 2D Range Trees & Higher Dimension Range Trees
What is a $k$-d Tree?

"Multidimensional Binary Search Trees Used for Associative Searching", Communications of the ACM, Bentley 1975
2D kd Tree Construction

The diagram illustrates the construction of a 2D kd tree with points labeled A(50,50), B(10,70), C(80,85), D(25,20), E(40,85), F(70,85), and G(10,60). The axis labels are $k_0$ and $k_1$. The tree is constructed by recursively dividing the space into smaller regions based on the split axis at each level.
2D kd Tree Construction

- Make 2 sorted lists, by x value and by y value
- Alternate dimensions (first split by x then by y)
- Find the median value
- Make a copy of the sorted lists, removing values from the other side
- Recurse
2D kd Tree Query Algorithm
2D kd Tree Query Algorithm

- At each split point
- Determine if the query box overlaps the split line
- Recurse down one or both branches
- If a subtree lies complete inside the box, return all items in that subtree
- Perform filtering in the leaves as necessary
2D kd Tree Query Analysis

- 1 item is stored per leaf node
- For a query that will collect $k$ items
2D kd Tree Query Analysis

- 1 item is stored per leaf node
- For a query that will collect $k$ items
- Best/Average(?) Case:
  - An approximately square query (equal width & height)
  - touches/overlaps $O(k)$ leaves
  - gathering leaves $O(\log n + k)$
  - Overall $\rightarrow O(\log n + k)$

Ignore that this is quadtree, pretend it is kdtree
2D kd Tree Query Analysis

- 1 item is stored per leaf node
- For a query that will collect \(k\) items
- **Best/Average(?) Case:** An approximately square query (equal width & height)
  - touches/overlaps \(O(k)\) leaves
  - gathering leaves \(O(\log n + k)\)
  - Overall \(\rightarrow O(\log n + k)\)
- **Worst Case Query:** For a skinny / lopsided query box
  - touches/overlaps \(-\sqrt{n} + k\) leaves
  - gathering leaves \(O(\sqrt{n} + k)\)
  - Overall \(\rightarrow O(\sqrt{n} + k)\)
Analysis: 2D kd Tree

Starting with $n$ values..

- Memory to store: $O(n)$
  - # of leaf nodes: $n$
  - # of intermediate nodes: $n-1$
  - Height of tree: $\log n$
- Time to construct: $O(n \log n)$
  - pre-sort the data, separately in $x$ and in $y$: $O(n \log n)$
  - Alternate axes - place middle value at root, recurse on the two sublists: $O(n \log n)$
- Time to query: $O(n^{1/2} + k) = O(\sqrt{n} + k)$
  - For search target / output returning $k$ values
Is Query Time = $O(\sqrt{n} + k)$ a problem?
Is Query Time $= O(\sqrt{n} + k)$ a problem?

- $O(1) \ < \ O(\log n) \ < \ O(\log^2 n) \ < \ O(\sqrt{n}) \ < \ O(n)$
Analysis: 3D kd Tree and higher dimensions

Starting with \( n \) values..

- Memory to store: \( \rightarrow O(n) \)
  - # of leaf nodes: \( n \)
  - # of intermediate nodes: \( n-1 \)
  - Height of tree: \( \log n \)
- Time to construct: \( \rightarrow O(n \log n) \)
  - pre-sort the data, separately in \( x \) and in \( y \) and in \( z \): \( O(n \log n) \)
  - Rotate through axes (\( x, y, z, x, \ldots \)) – place middle value at root, recurse on two sublists: \( O(n \log n) \)
- Time to query: \( \rightarrow O(n^{2/3} + k) \rightarrow O(n^{(1-1/d)} + k) \)
  - For search target / output returning \( k \) values
Is Query Time = $O(n^{(1-1/d)} + k)$ a problem?

- Yeah, this is a problem as dimensions increase
- Common for complex databases
Outline for Today

- Homework 3 Posted
- Last Time: Bounding Spheres & Randomized Incremental Construction
- Motivating Application: Database Queries
- Motivating Application: Graphics & Photon Mapping
- Data Structure Choices
  - Cost to construct, memory to construct, cost to query
- Review: (1D) Binary Search Trees
- 2D kD Trees & Higher dimension kD Trees
- 2D Range Trees & Higher Dimension Range Trees
What is a Range Tree?

- **Idea:** If we use more memory, can we reduce worst case query time of $kD$ tree?
  - First we organize the data in a BST by $x$ value
  - At every node in the tree, we store a pointer to a BST with the same data, but organized by $y$ value
What is a Range Tree?

- Idea: If we use more memory, can we reduce worst case query time of $kD$ tree?

- First we organize the data in a BST by $x$ value
- At every node in the tree, we store a pointer to a BST with the same data, but organized by $y$ value
How to Construct the 2D Range Tree?

How much memory does it use?
How to Construct the 2D Range Tree?

How much memory does it use?

- Each point p is stored once in the level 1 (organized by x) tree
- And many times in level 2 (organized by y) trees
- How many level 2 trees? And how big are they?
  - 1 tree with n values
  - 2 trees with n/2 values
  - 4 trees with n/4 values
  - ...
  - n trees with 1 values

→ $O(n \log n)$ memory
How to Query
2D Range Tree?
Analysis: 2D Range Tree

Starting with $n$ values..

- Memory to store: $\rightarrow O(n \log n)$

- Time to construct: $\rightarrow O(n \log n)$

- Time to query: $\rightarrow O(\log^2 n + k)$
Analysis: 2D Range Tree

Starting with $n$ values..

- Memory to store:
- Time to construct:
- Time to query:
Higher Dimensional Range Tree

- ... and can be extended to arbitrarily higher dimensions
Analysis: 3D \( kd \) Tree and higher dimensions

Starting with \( n \) values..

- Memory to store:

- Time to construct:

- Time to query:
Analysis: 3D kd Tree and higher dimensions

Starting with $n$ values..

- Memory to store: $\rightarrow O(n \log^{d-1} n)$
- Time to construct: $\rightarrow O(n \log^{d-1} n)$
- Time to query: $\rightarrow O(\log^d n + k)$
Summary Comparison

- **kD tree**
  - Construction time
  - Memory
  - Query time

- **Range tree**
  - Construction time
  - Memory
  - Query time
Outline for Today

- Homework 3 Posted
- Last Time: Bounding Spheres & Randomized Incremental Construction
- Motivating Application: Database Queries
- Motivating Application: Graphics & Photon Mapping
- Data Structure Choices
  - Cost to construct, memory to construct, cost to query
- Review: (1D) Binary Search Trees
- 2D kD Trees & Higher dimension kD Trees
- 2D Range Trees & Higher Dimension Range Trees