CSCI 4560/6560 Computational Geometry

https://www.cs.rpi.edu/~cutler/classes/computationalgeometry/F23/

Lecture 16: Windowing, Interval & Segment Trees

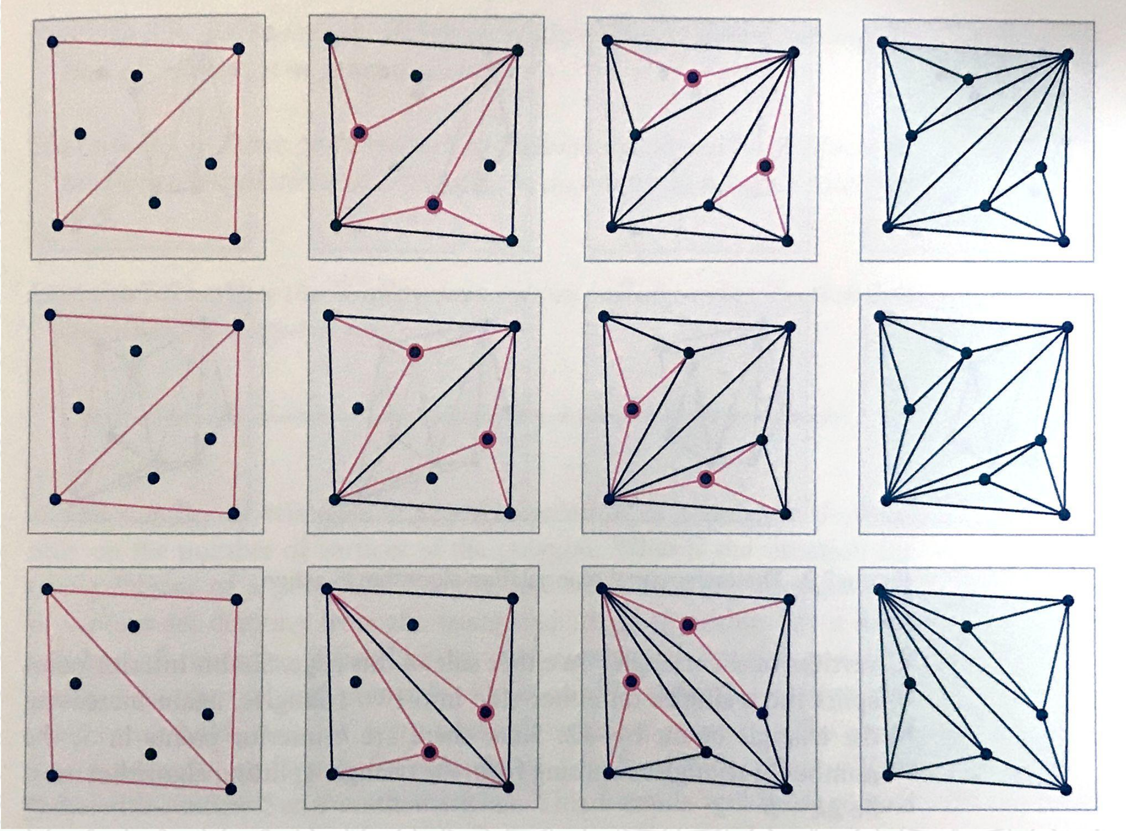
Outline for Today

- Review from Last Time: Delaunay Triangulations
- Motivation: Cartography Windowing & Data Selection
- Lecture 8 Review: Points in k-D trees and Range Trees
- 1D Interval Tree
- 1D Interval Tree Analysis
- 2D Interval Tree + Range Tree
- 2D Interval Tree + Range Tree Analysis
- 2D Axis Aligned Segment Query
- Segment Tree for general 2D Segment Query
- Segment Tree Analysis
- Next Time: Quad Trees

“Discrete and Computational Geometry”, Devadoss & O’Rourke, Princeton University Press 2011, Chapter 3

Construction by Point Insertion

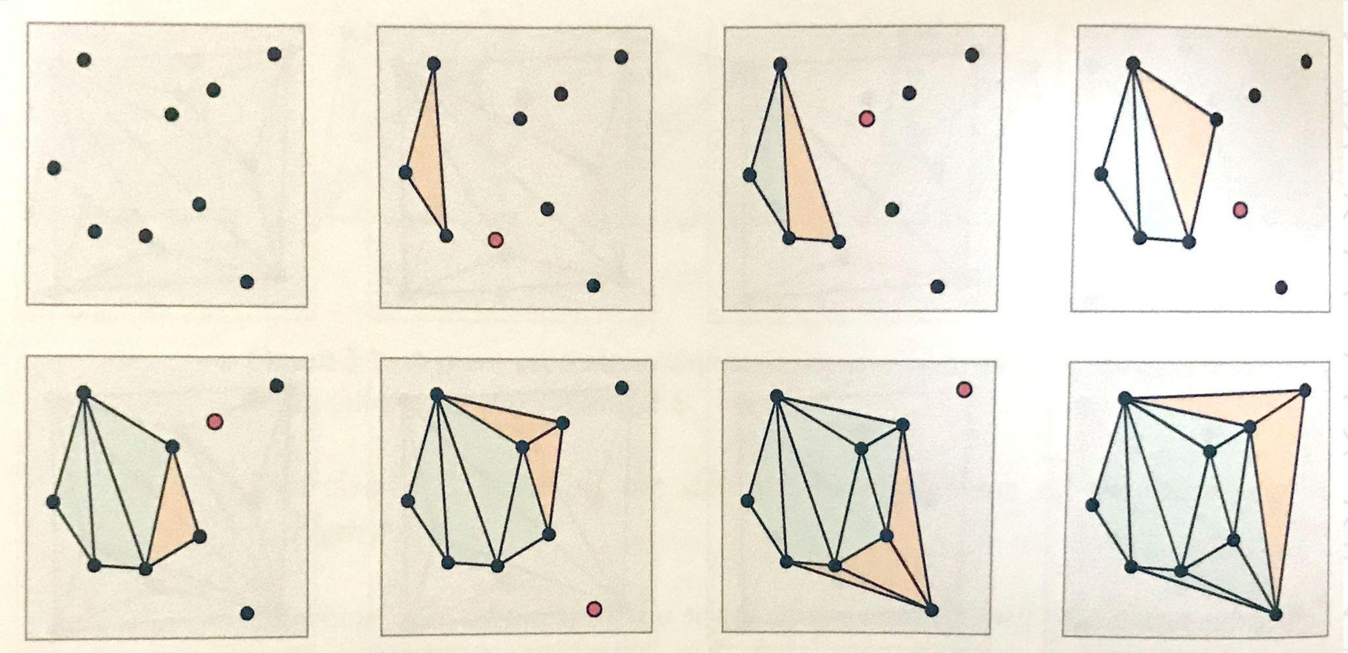
- Start with convex hull
  - Triangulate it
  - *k*-2 triangles
- For some ordering of the other points
  - Determine which triangle the point lies inside of
  - Replace that triangle with 3 triangles
  - (*n* - *k*) * 2 additional triangles
- 2**n* - *k* - 2 total triangles!



Construction by Line Sweep

“Discrete and Computational Geometry”, Devadoss & O’Rourke, Princeton University Press 2011, Chapter 3

- Sort the input points by x
- Form a triangle with the 3 leftmost points

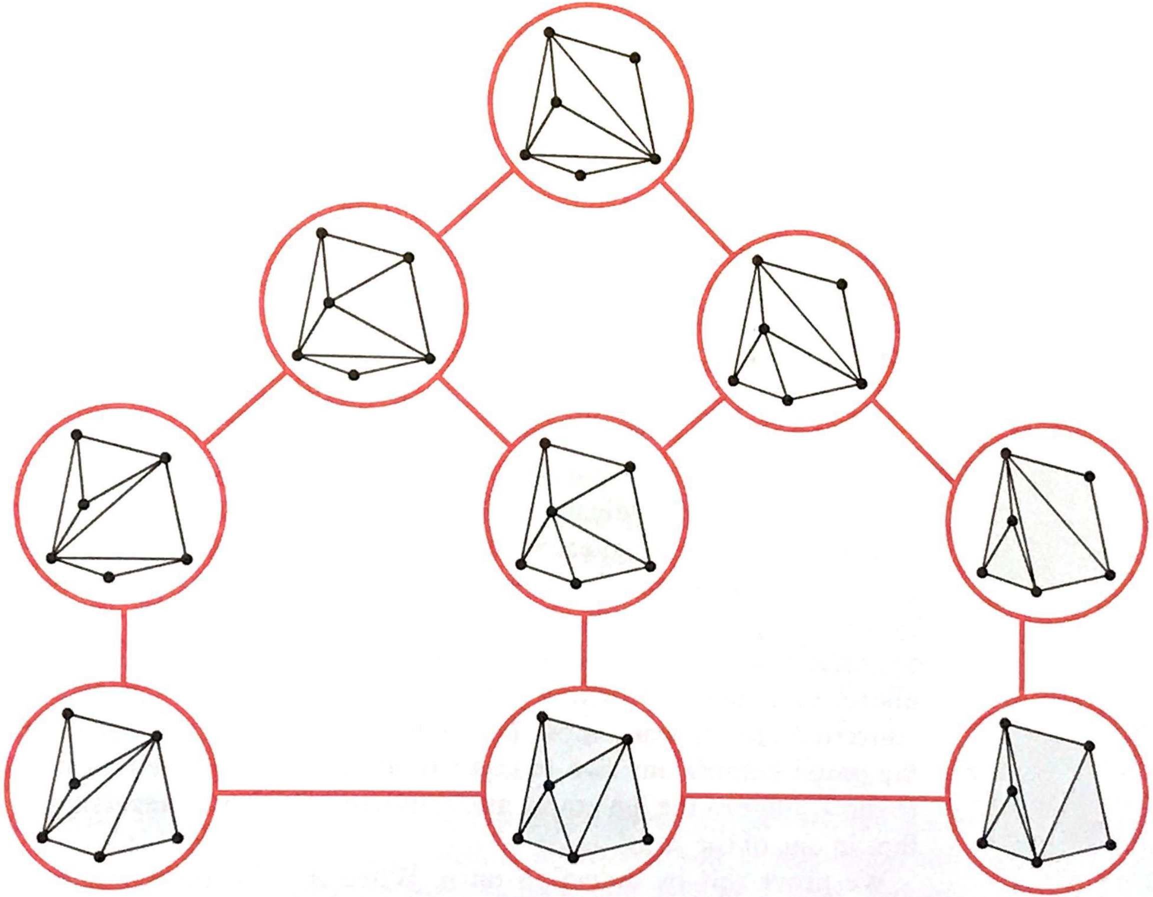


- Add every other point from left to right
  - Determine which points on the current hull are visible from the new point
  - Add a fan of triangles connecting the new point to the visible hull points

The Flip Graph

- If we did generate every triangulation…

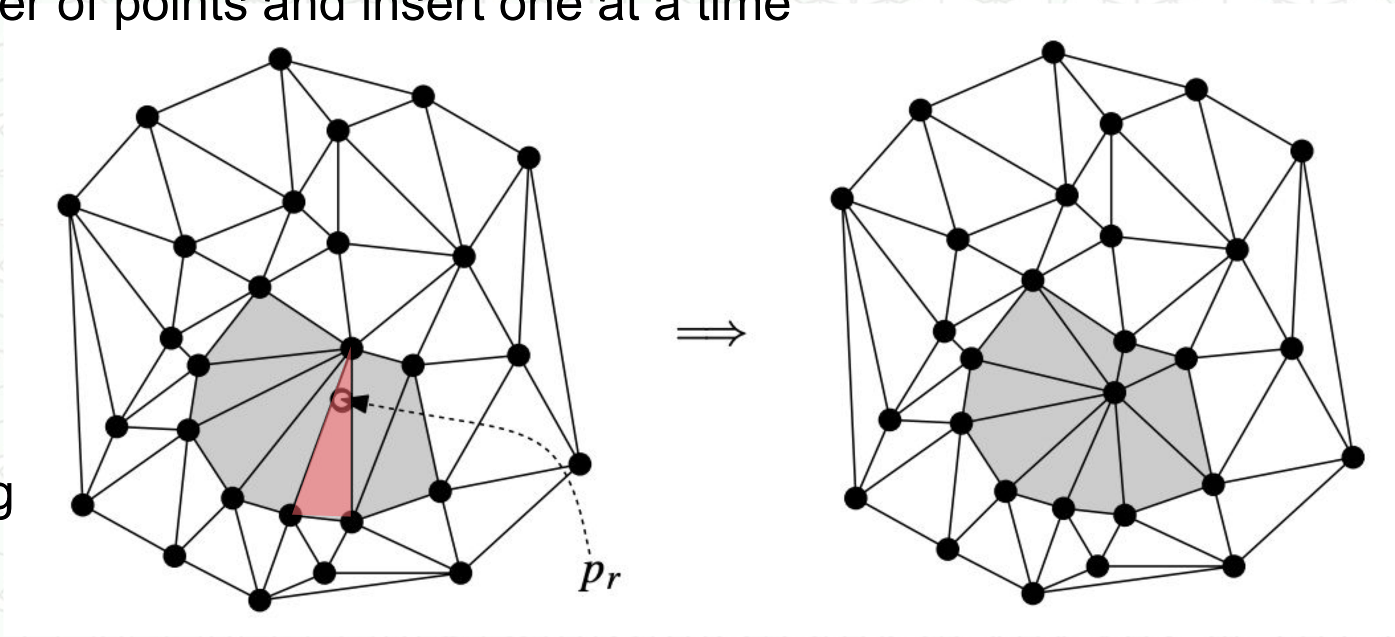
- Let’s organize the triangulations as nodes in a graph
- We’ll put an edge between two nodes if flipping a single edge converts one triangulation into the other triangulation



“Discrete and Computational Geometry”, Devadoss & O’Rourke,
Princeton University Press 2011, Chapter 3

Randomized Incremental Construction of Delaunay Triangulation

- Randomize order of points and insert one at a time
- Identify which triangle contains p_r
- Split into 3 smaller triangles
- Flip neighboring edges as necessary



Hopefully the footprint of impact is small!

Computational Geometry Algorithms and Applications, de Berg, Cheong, van Kreveld and Overmars, Chapter 9

Delaunay Construction Analysis Summary

- Brute force (enumerate all triangles, construct circles, reject…)

  → $O(n^3 * n) = O(n^4)$

- Construct any triangulation & Flip until all edges are legal

  → $O(n^2)$

- Randomized Incremental Construction

  → $O(n \log n)$

- By duality, reduce to problem of Constructing the Voronoi Diagram

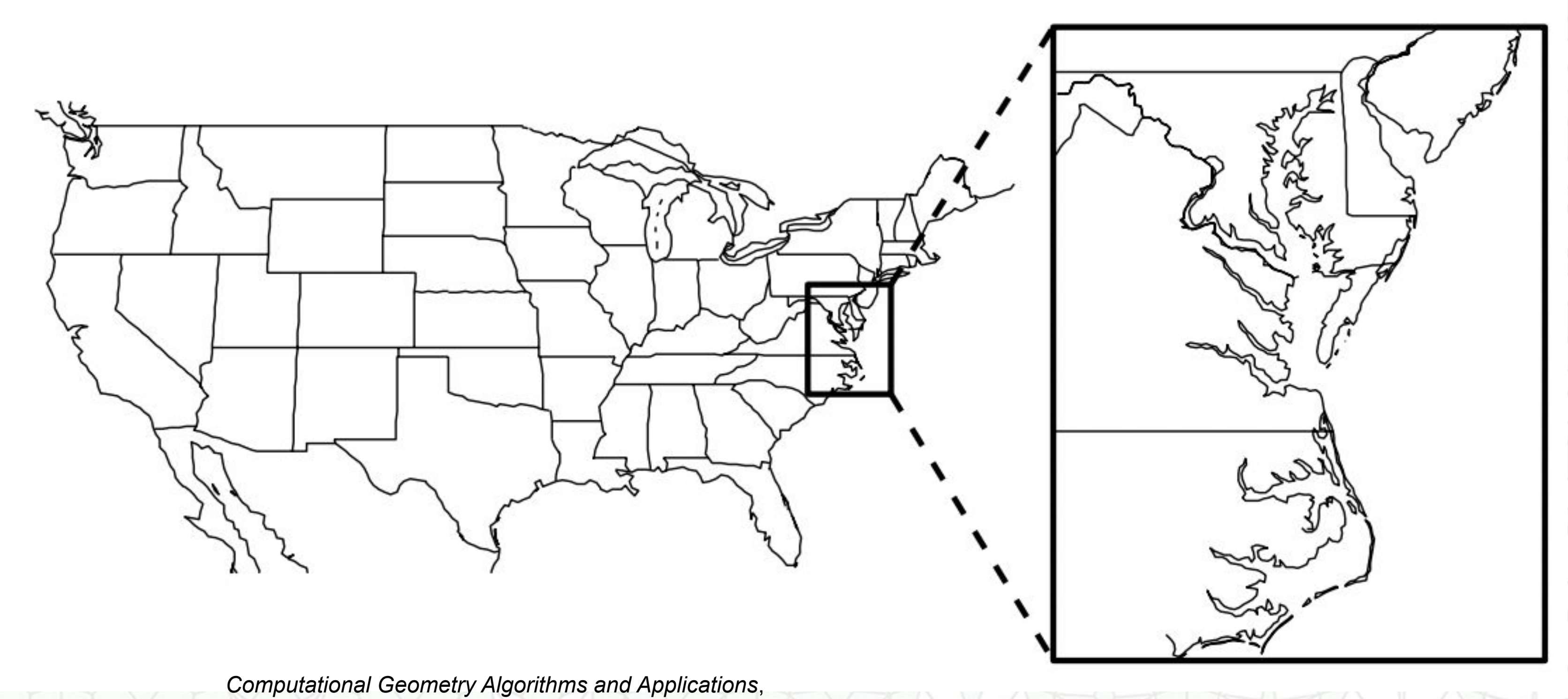
  → $O(n \log n)$

Outline for Today

- Review from Last Time: Delaunay Triangulations
- Motivation: Cartography Windowing & Data Selection
- Lecture 8 Review: Points in k-D trees and Range Trees
- 1D Interval Tree
- 1D Interval Tree Analysis
- 2D Interval Tree + Range Tree
- 2D Interval Tree + Range Tree Analysis
- 2D Axis Aligned Segment Query
- Segment Tree for general 2D Segment Query
- Segment Tree Analysis
- Next Time: Quad Trees

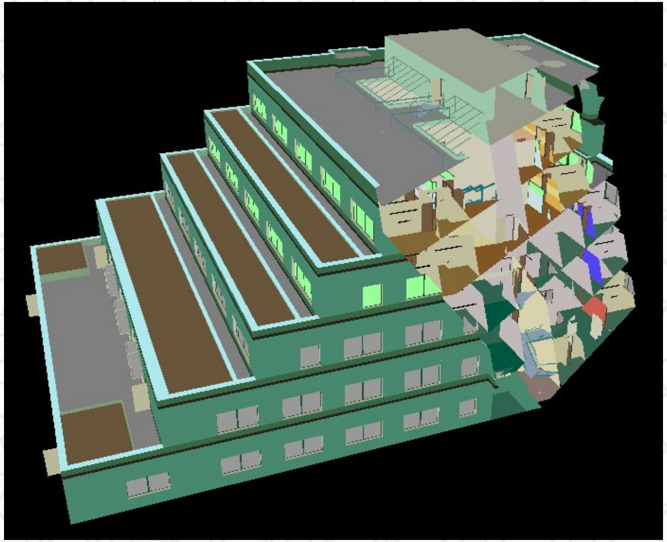
Motivation: Cartography (Map-Making)

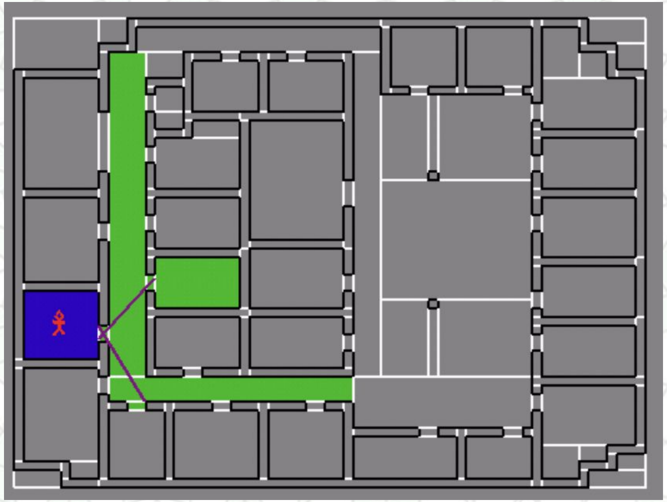
- Select a small rectangular region to display in a window at larger scale



Computational Geometry Algorithms and Applications,
de Berg, Cheong, van Kreveld and Overmars, Chapter 10

Motivation: Visibility

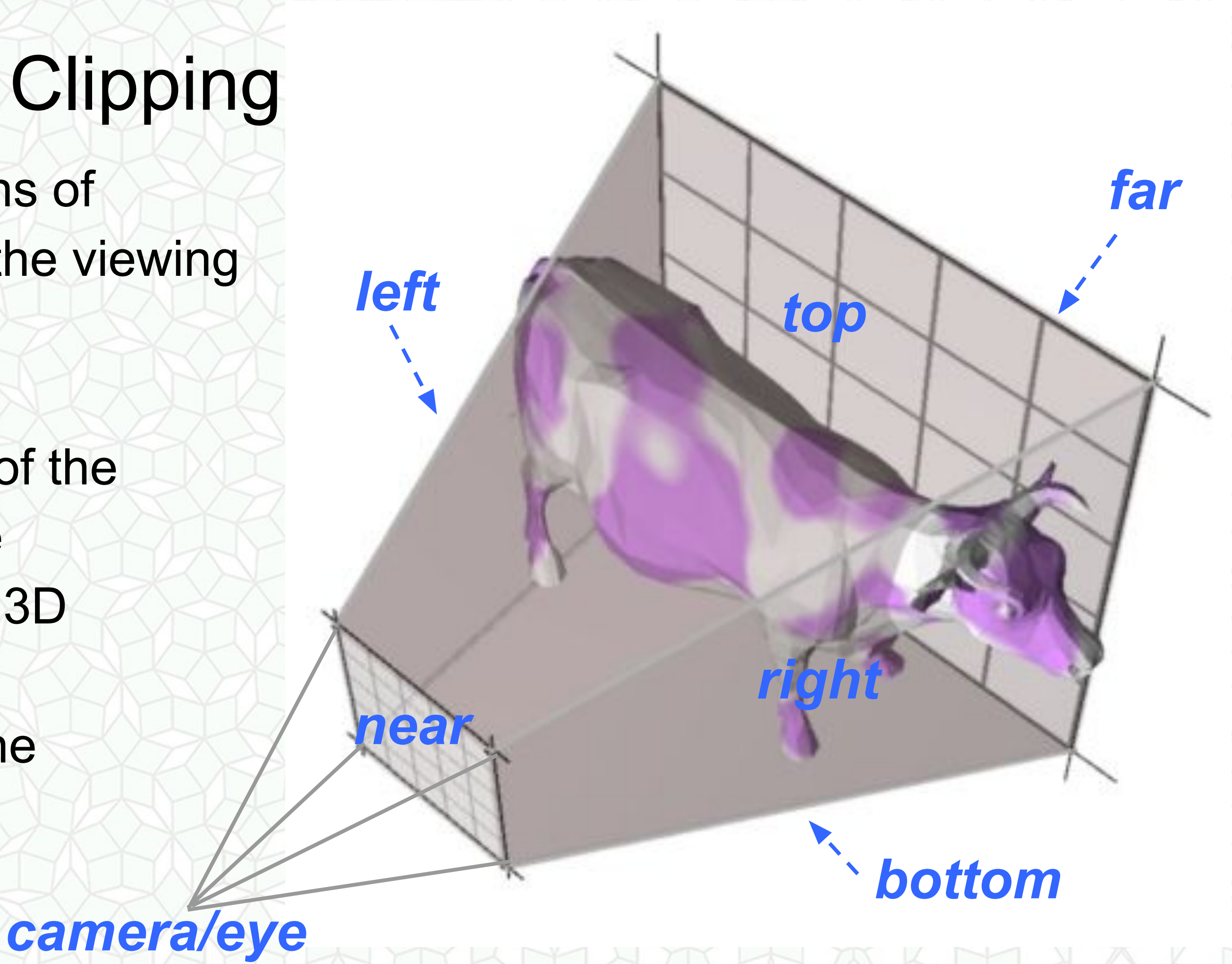




Seth Teller, PhD thesis, 1992,
Berkeley Soda Hall walkthrough

Graphics: 3D Clipping

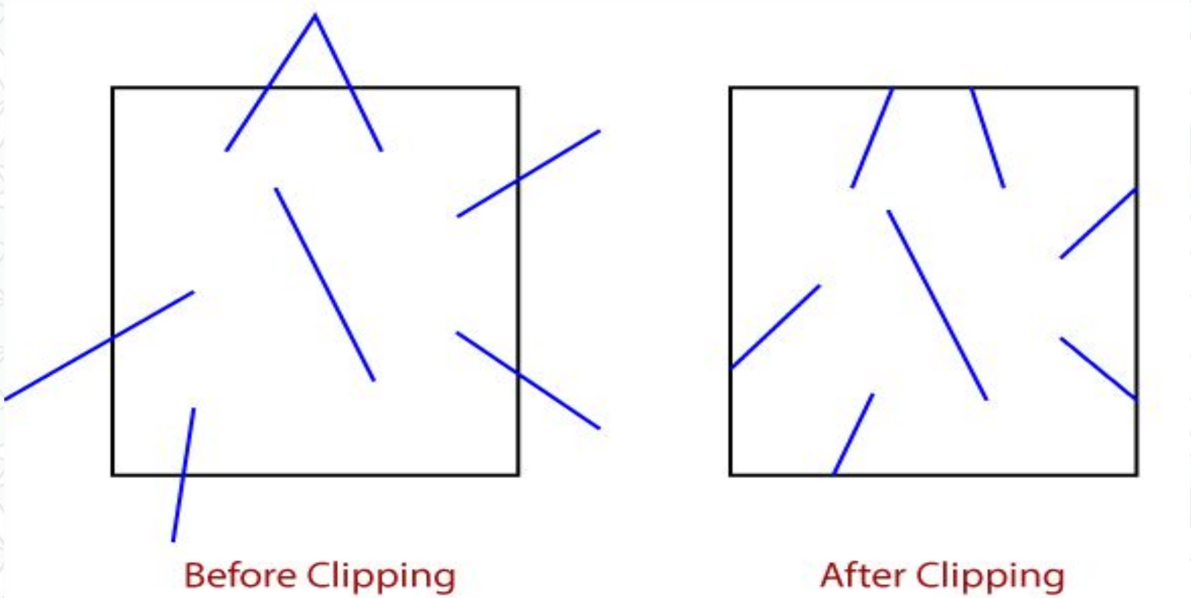
- Eliminate portions of objects outside the viewing frustum
- View Frustum
  - boundaries of the image plane projected in 3D
  - a near & far clipping plane



Graphics: 2D Clipping

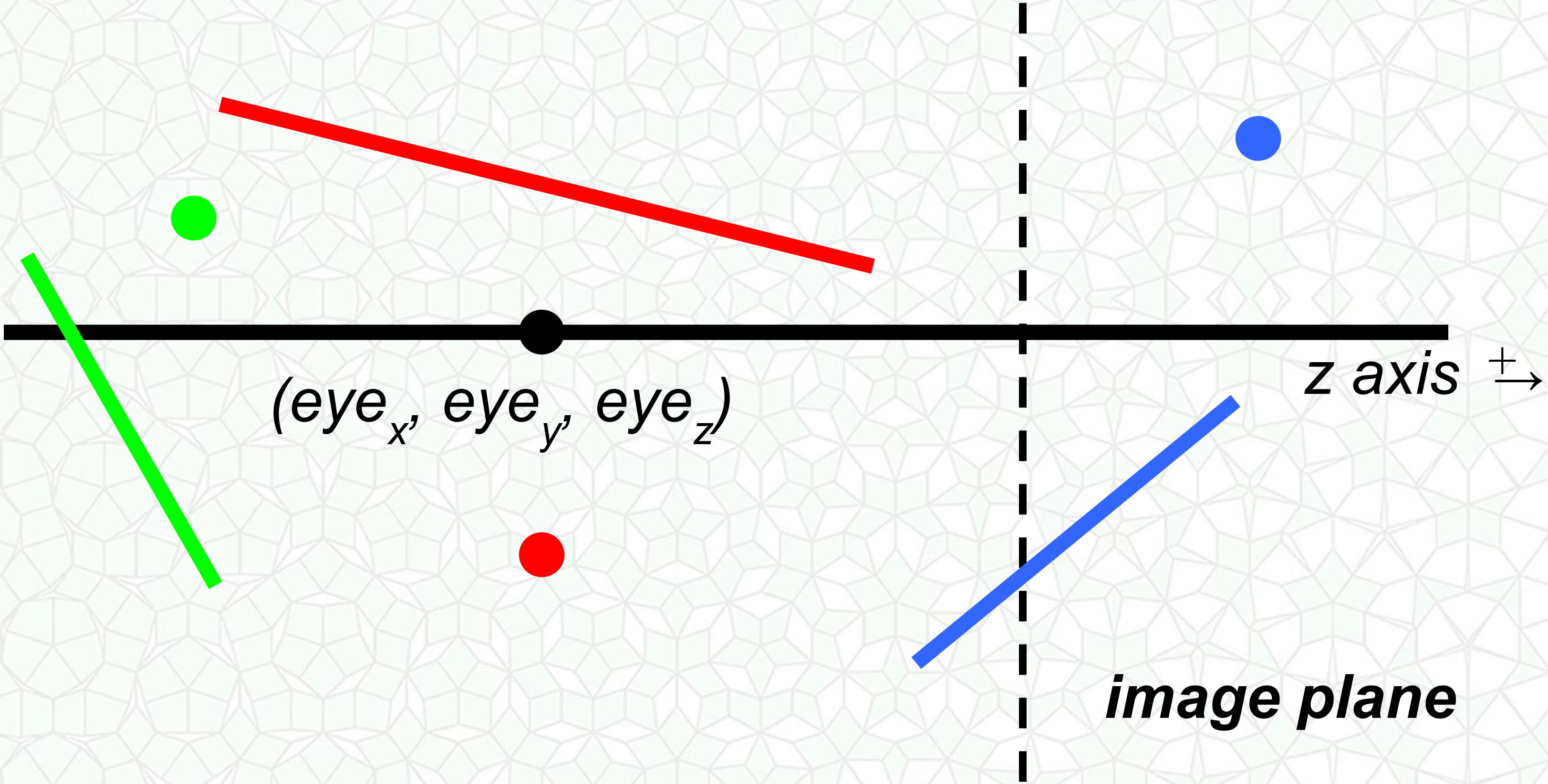
Why do it?

- Reduce amount of geometry going through graphics pipeline
- Prevent rendering bugs from overflow, wraparound, things behind the camera, etc.

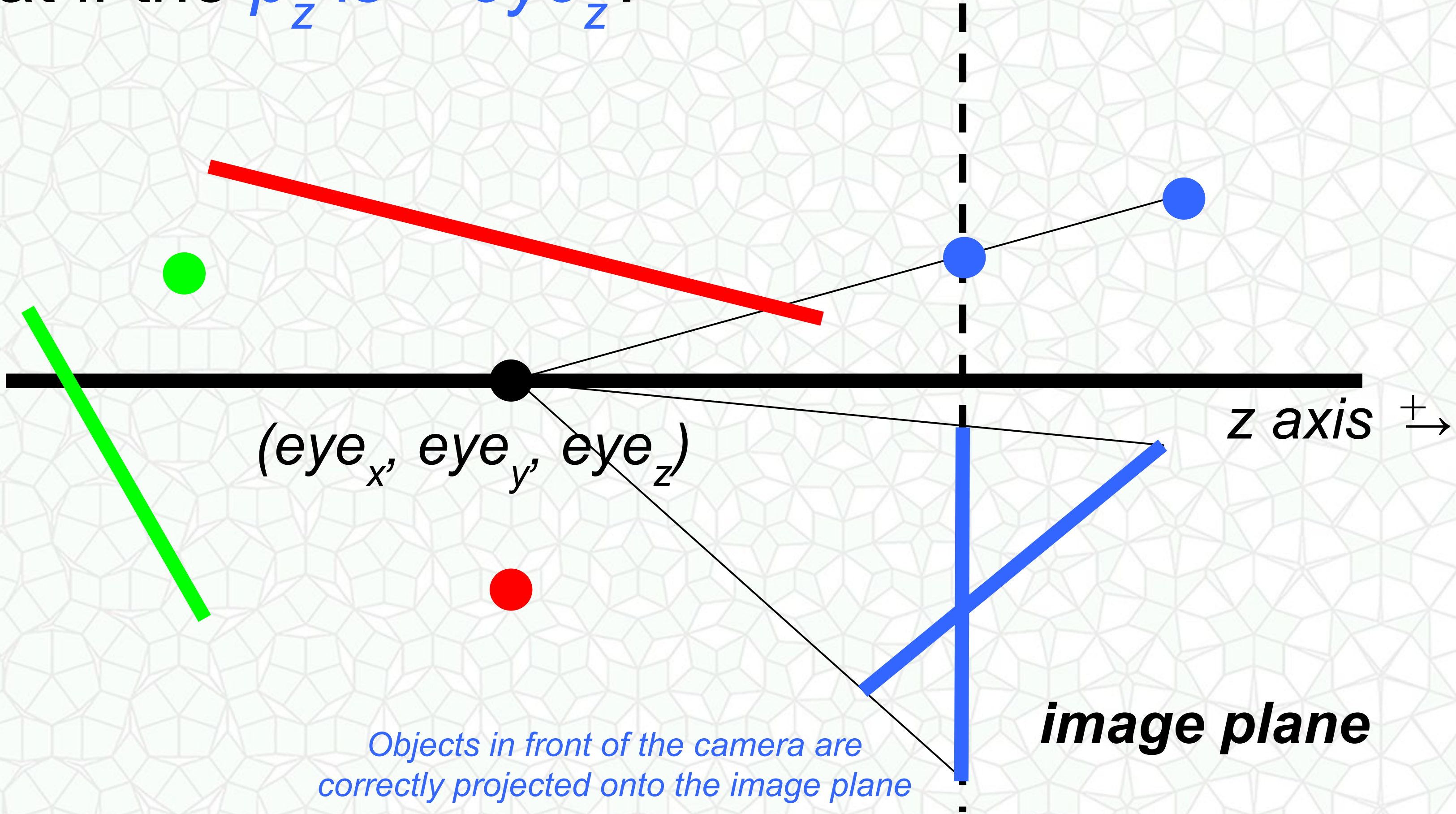


https://www.tutorialandexample.com/clipping-in-computer-graphics

Perspective Projection onto Image Plane

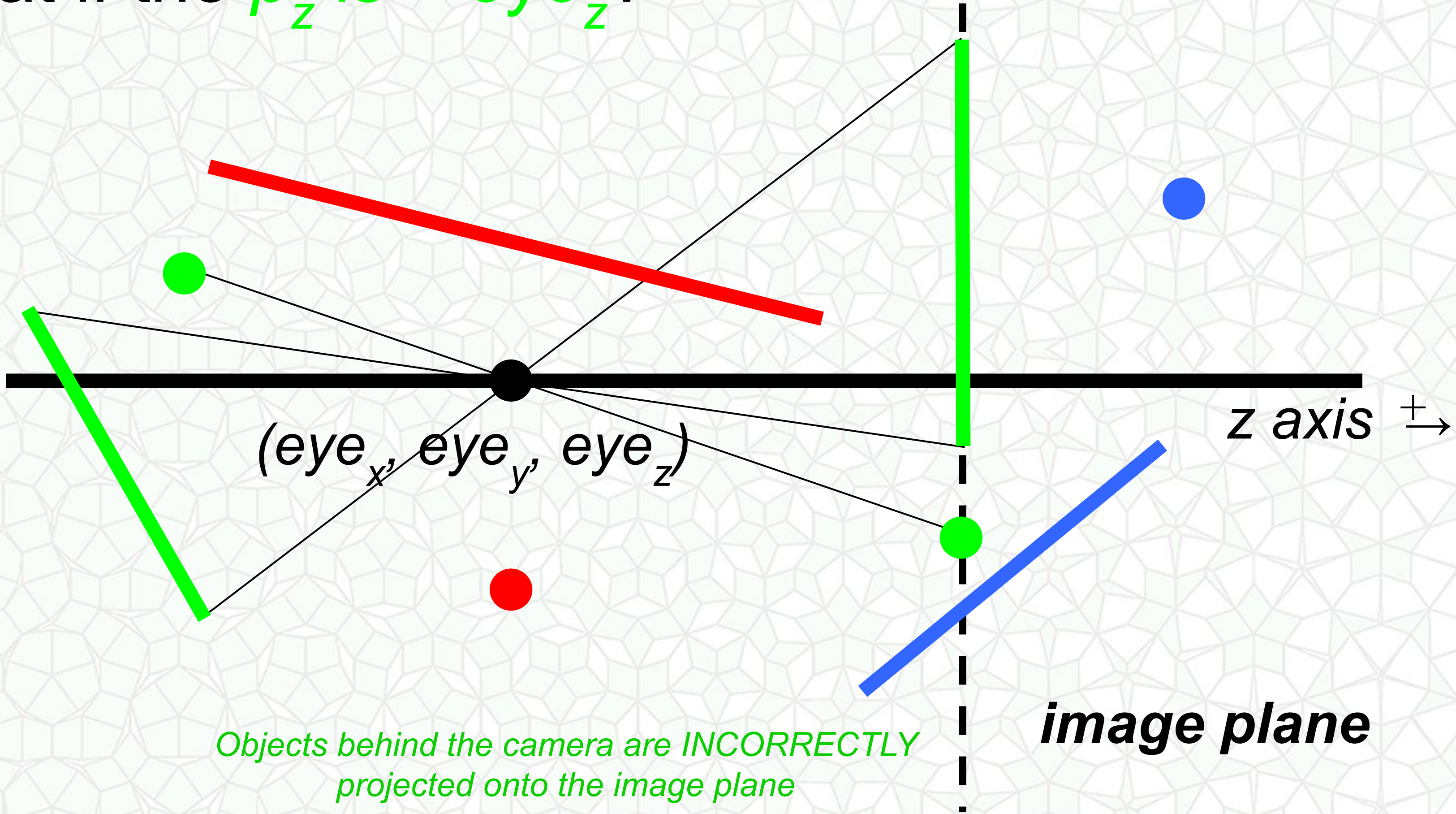


What if the p_z is > eye_z?



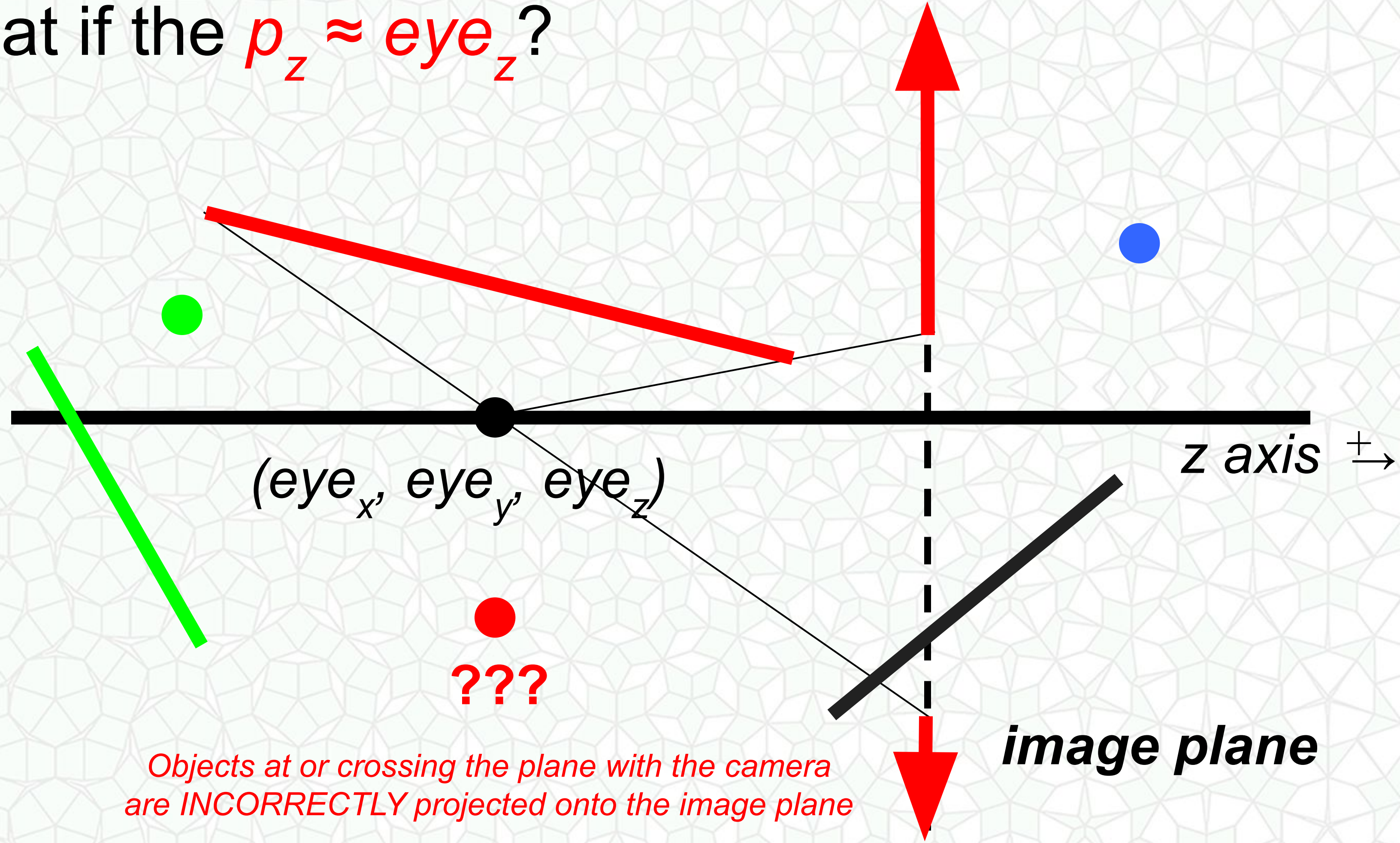
Objects in front of the camera are correctly projected onto the image plane

What if the p_z is < eye_z?



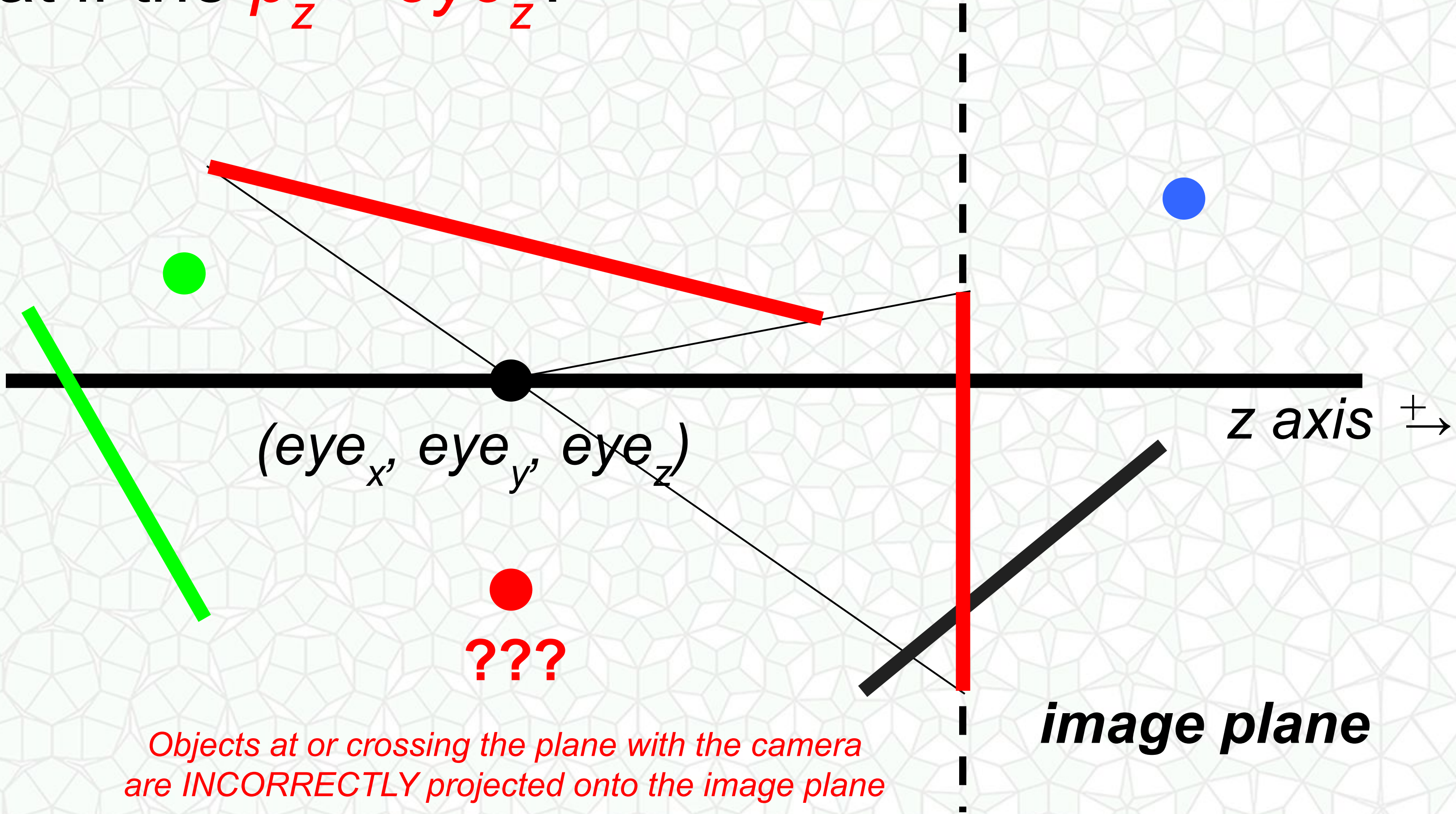
Objects behind the camera are INCORRECTLY projected onto the image plane

What if the $p_z \approx eye_z$?



Objects at or crossing the plane with the camera are INCORRECTLY projected onto the image plane

What if the $p_z \approx eye_z$?



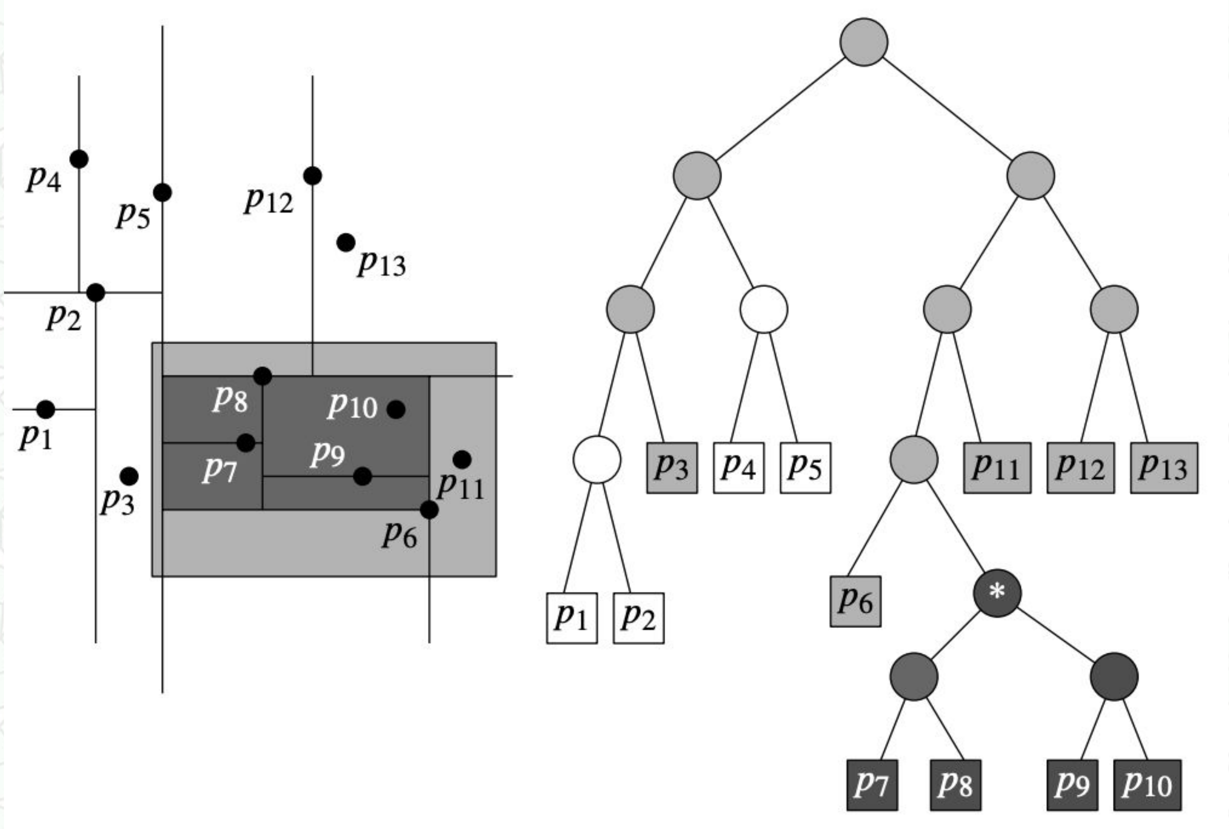
Objects at or crossing the plane with the camera are INCORRECTLY projected onto the image plane

Outline for Today

- Review from Last Time: Delaunay Triangulations
- Motivation: Cartography Windowing & Data Selection
- Lecture 8 Review: Points in k-D trees and Range Trees
- 1D Interval Tree
- 1D Interval Tree Analysis
- 2D Interval Tree + Range Tree
- 2D Interval Tree + Range Tree Analysis
- 2D Axis Aligned Segment Query
- Segment Tree for general 2D Segment Query
- Segment Tree Analysis
- Next Time: Quad Trees

Review from Lecture 8: 2D k-d Tree

- Used to store points
- Alternate splitting horizontally & vertically
- If data is available for preprocess, the structure is easy to balance
- Point data is only stored at the leaves



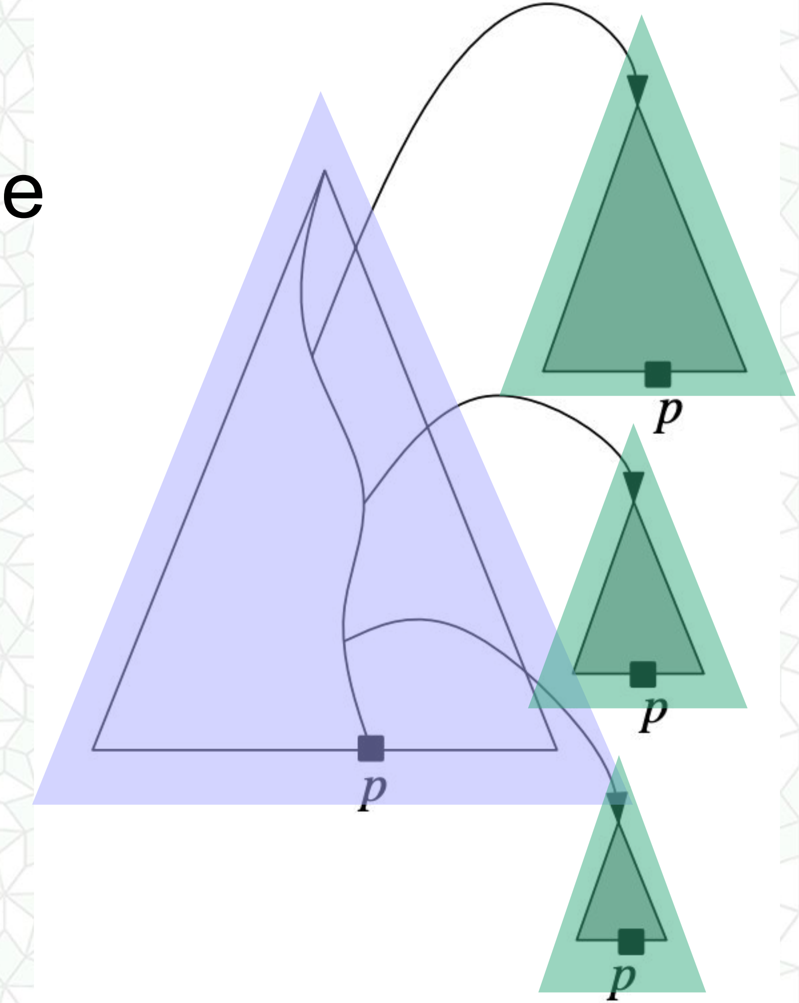
Review from Lecture 8: How to Query 2D Range Tree

- Search through level 1 (blue) tree for all intermediate nodes that fit completely inside the query's *x* range

  For each matched intermediate blue node

  - Search through the corresponding level 2 (green) trees for all nodes and leaves that fit completely inside the query's y range

    Return all matching data!

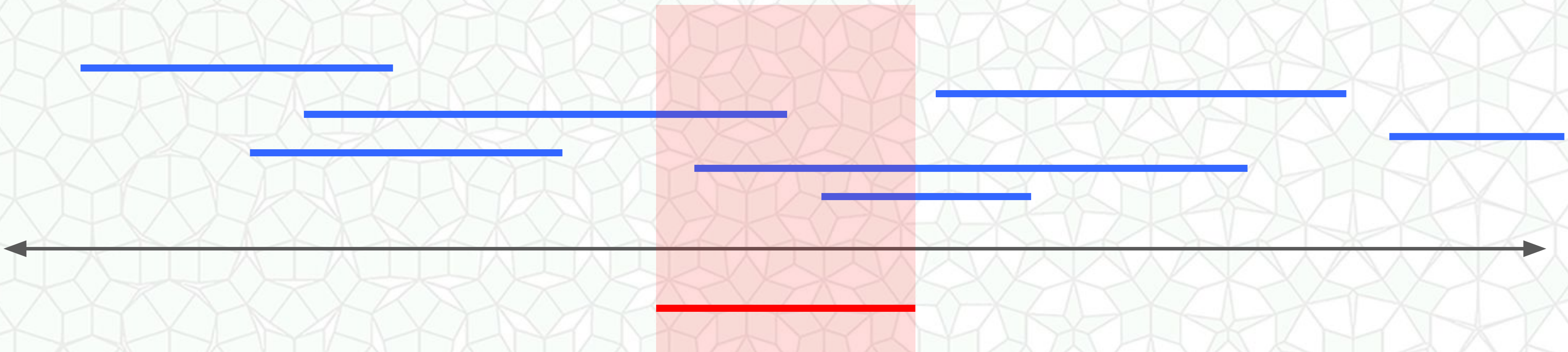


Outline for Today

- Review from Last Time: Delaunay Triangulations
- Motivation: Cartography Windowing & Data Selection
- Lecture 8 Review: Points in k-D trees and Range Trees
- 1D Interval Tree
- 1D Interval Tree Analysis
- 2D Interval Tree + Range Tree
- 2D Interval Tree + Range Tree Analysis
- 2D Axis Aligned Segment Query
- Segment Tree for general 2D Segment Query
- Segment Tree Analysis
- Next Time: Quad Trees

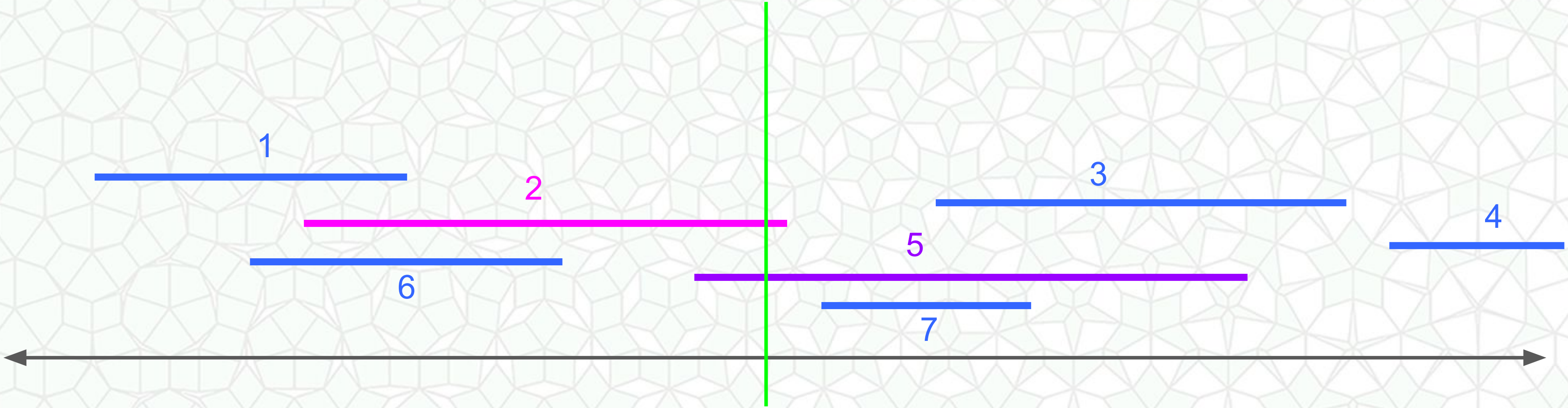
What about Segments? Let’s Tackle 1D First…

- Input: A collection of *n* line segments on the x-axis
- For a query interval, return all line segments that overlap the query interval



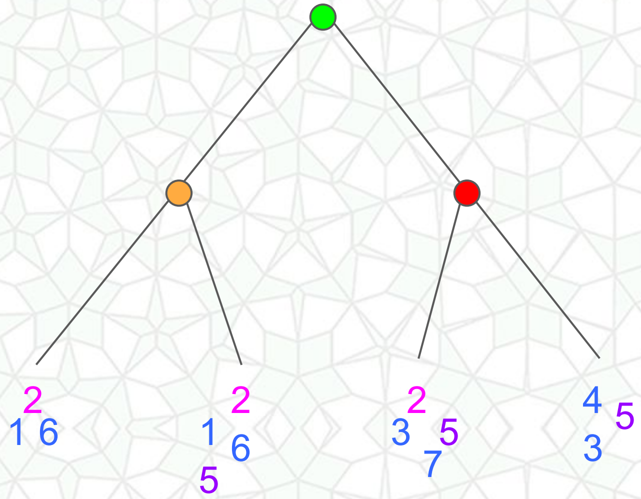
Traditional Binary Search Tree

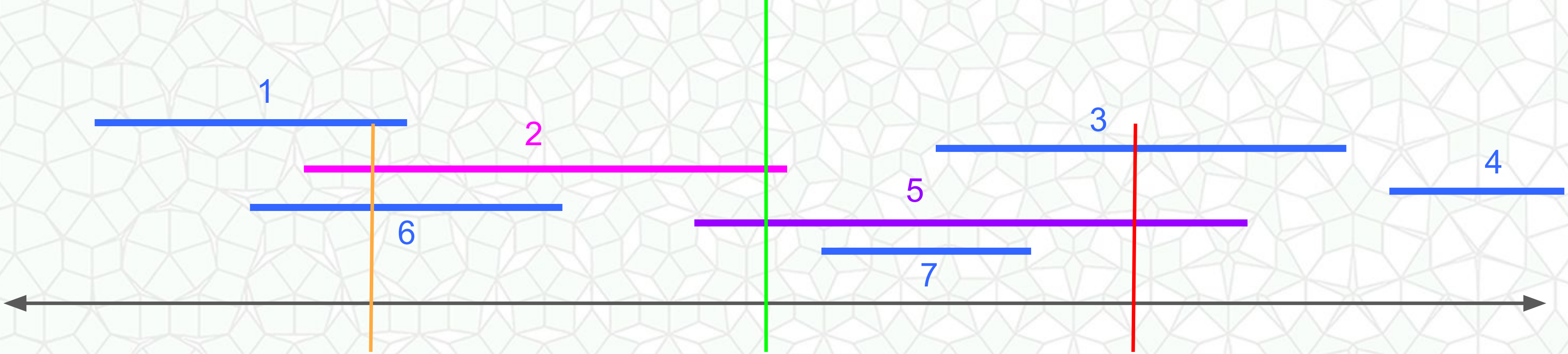
- Select split point near middle of data
- What about segments that overlap the split?



Traditional Binary Search Tree

- Select split point near middle of data
- What about segments that overlap the split?
- Should we store them on both sides?
  - Uses extra memory
  - Have to remove duplicates
  - We may lose our O(log n) performance!

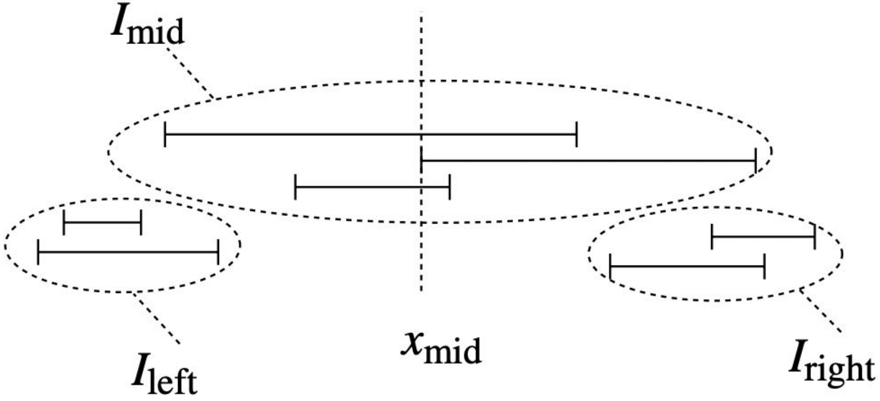


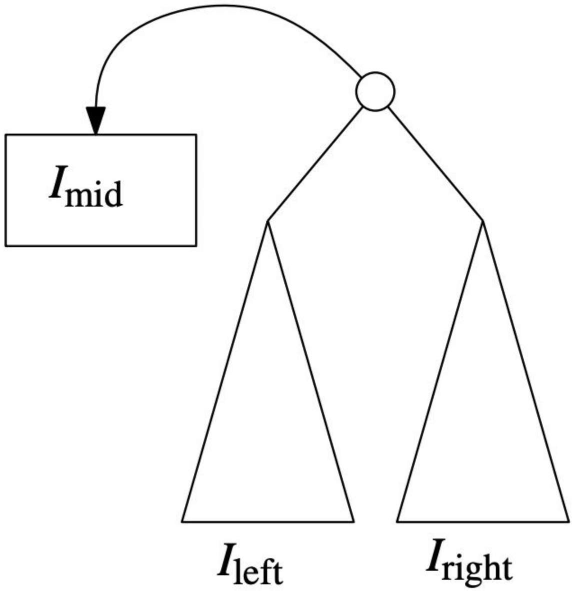


Computational Geometry Algorithms and Applications,
de Berg, Cheong, van Kreveld and Overmars, Chapter 10

Interval Tree

- Chose a split point and make 3 groups:
  - I_{mid} = Segments that overlap the split
  - I_{left} = Segments completely to the left
  - I_{right} = Segments completely to the right

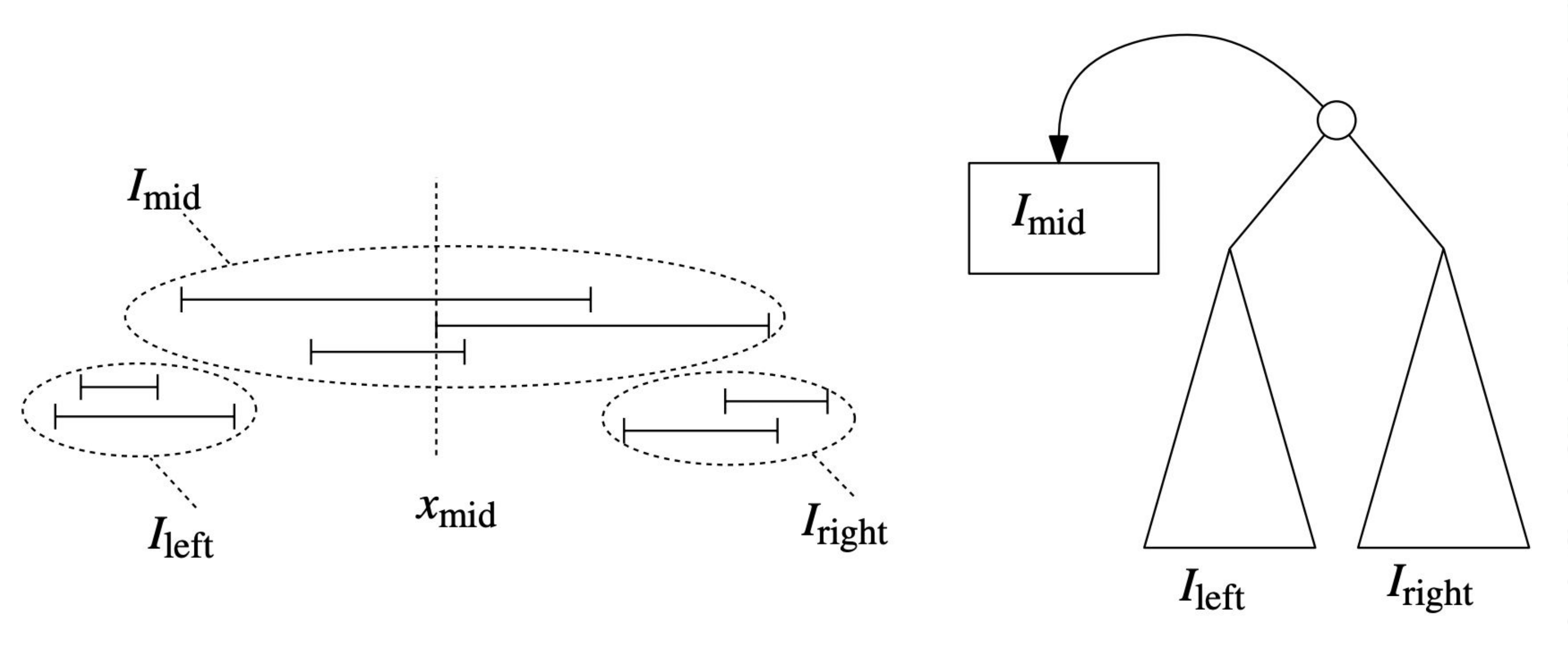




Computational Geometry Algorithms and Applications,
de Berg, Cheong, van Kreveld and Overmars, Chapter 10

Interval Tree

- Recurse down the tree only with items that DO NOT overlap the split point.

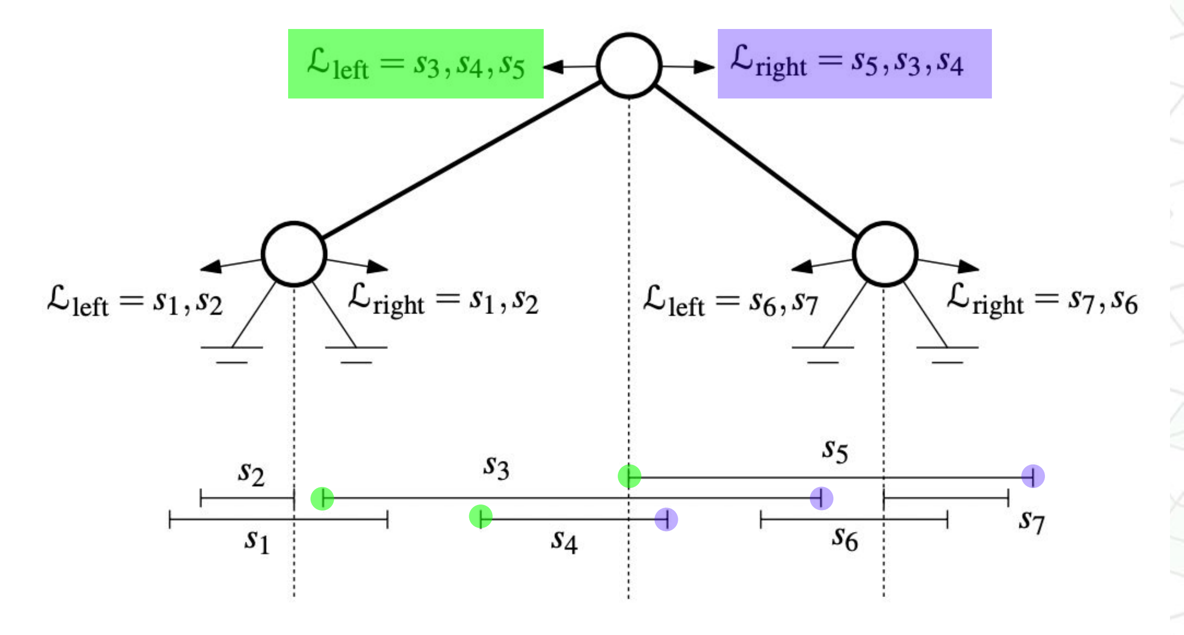


Interval Tree

- Items in I_{mid} group will stay at the current node

- Each node stores two two sorted lists:

- L_{left} = I_{mid} sorted by left endpoint (increasing)
- L_{right} = I_{mid} sorted by right endpoint (decreasing)

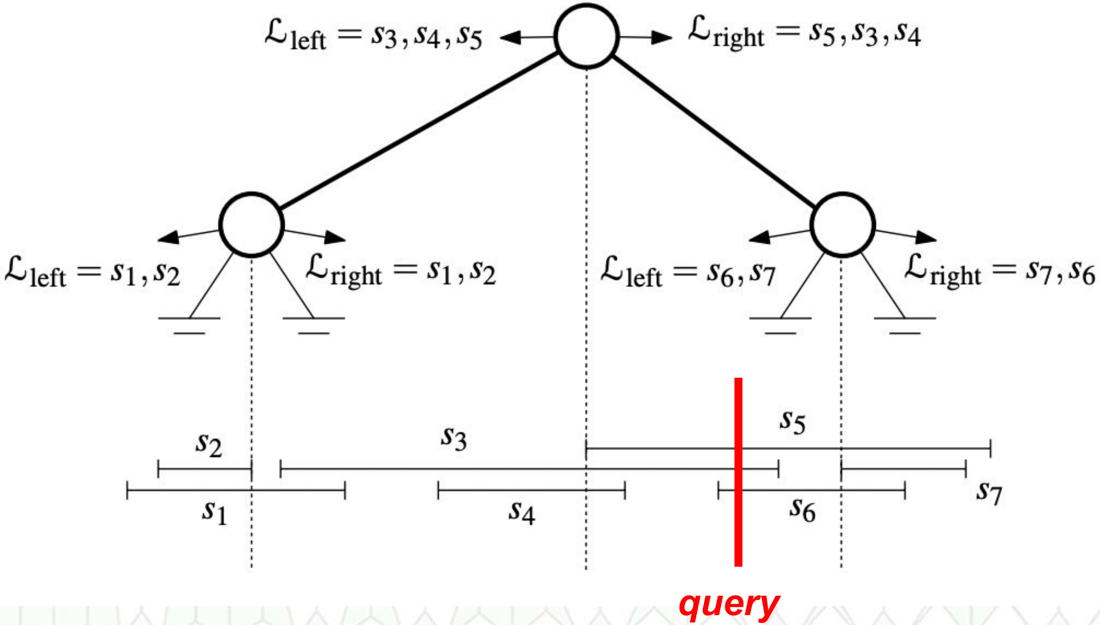


$\mathcal{L}_{\text{left}} = s_3, s_4, s_5$
$\mathcal{L}_{\text{right}} = s_5, s_3, s_4$
$\mathcal{L}_{\text{left}} = s_1, s_2$
$\mathcal{L}_{\text{right}} = s_1, s_2$
$\mathcal{L}_{\text{left}} = s_6, s_7$
$\mathcal{L}_{\text{right}} = s_7, s_6$
s_2
s_3
s_5
s_1
s_4
s_6
s_7

Computational Geometry Algorithms and Applications, de Berg, Cheong, van Kreveld and Overmars, Chapter 10

Interval Tree

For a specific *query*

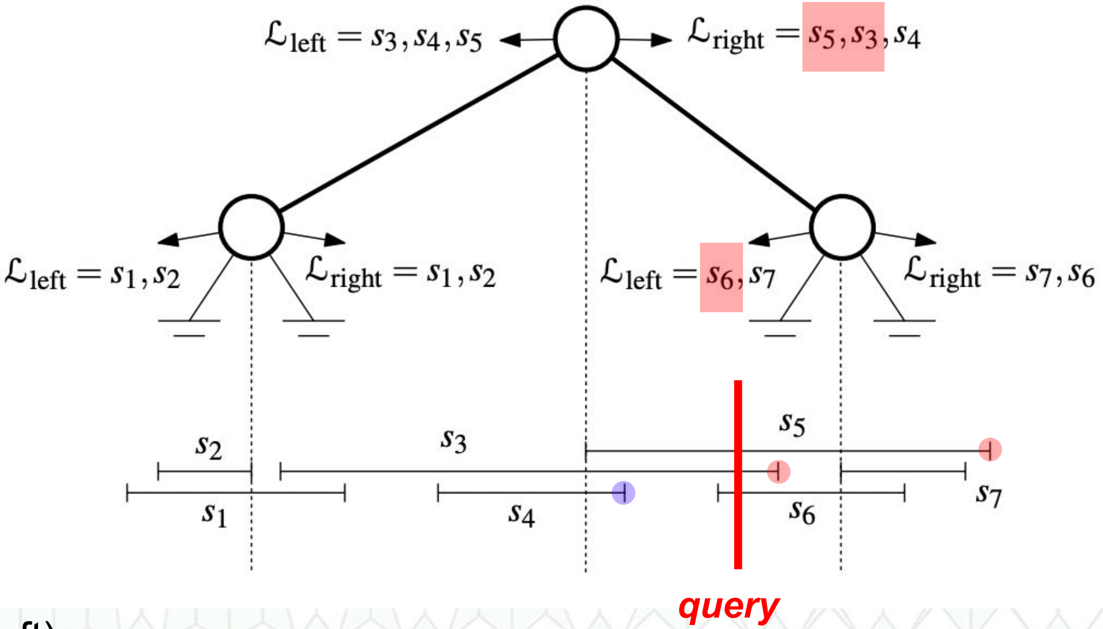

$\mathcal{L}_{\text{left}} = s_3, s_4, s_5$
$\mathcal{L}_{\text{right}} = s_5, s_3, s_4$
$\mathcal{L}_{\text{left}} = s_1, s_2$
$\mathcal{L}_{\text{right}} = s_1, s_2$
$\mathcal{L}_{\text{left}} = s_6, s_7$
$\mathcal{L}_{\text{right}} = s_7, s_6$
s_5
s_2
s_3
s_7
s_1
s_4
s_6
query

Computational Geometry Algorithms and Applications,
de Berg, Cheong, van Kreveld and Overmars, Chapter 10

Interval Tree

For a specific *query*

- Determine if the query is to the right (or left) of the current node
- Walk through the L_{right} list (or L_{left} list) which is sorted by right (or left) endpoint
  - Return segments with endpoint further away from the query
  - *Stop when you find a segment that doesn't match the query*
- And recurse down the right (or left)


$\mathcal{L}_{\text{left}} = s_3, s_4, s_5$
$\mathcal{L}_{\text{right}} = s_5, s_3, s_4$
$\mathcal{L}_{\text{left}} = s_1, s_2$
$\mathcal{L}_{\text{right}} = s_1, s_2$
$\mathcal{L}_{\text{left}} = s_6, s_7$
$\mathcal{L}_{\text{right}} = s_7, s_6$
s_2
s_3
s_5
s_1
s_4
s_6
s_7
query

Computational Geometry Algorithms and Applications, de Berg, Cheong, van Kreveld and Overmars, Chapter 10

Outline for Today

- Review from Last Time: Delaunay Triangulations
- Motivation: Cartography Windowing & Data Selection
- Lecture 8 Review: Points in k-D trees and Range Trees
- 1D Interval Tree
- 1D Interval Tree Analysis
- 2D Interval Tree + Range Tree
- 2D Interval Tree + Range Tree Analysis
- 2D Axis Aligned Segment Query
- Segment Tree for general 2D Segment Query
- Segment Tree Analysis
- Next Time: Quad Trees

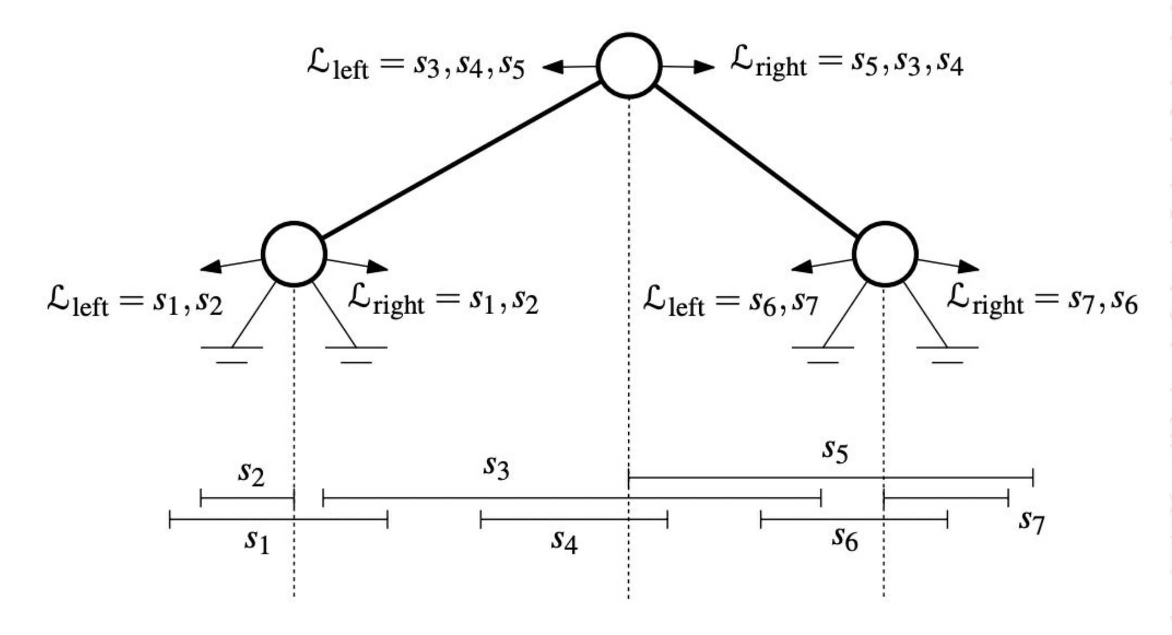
1D Interval Tree Analysis

- For *n* input segments and a query that will return *k* items

- Memory Usage:

- Construction Time:

- Query Time:



$\mathcal{L}_{\text{left}} = s_3, s_4, s_5$
$\mathcal{L}_{\text{right}} = s_5, s_3, s_4$
$\mathcal{L}_{\text{left}} = s_1, s_2$
$\mathcal{L}_{\text{right}} = s_1, s_2$
$\mathcal{L}_{\text{left}} = s_6, s_7$
$\mathcal{L}_{\text{right}} = s_7, s_6$
s_5
s_2
s_3
s_7
s_1
s_4
s_6

Computational Geometry Algorithms and Applications,
de Berg, Cheong, van Kreveld and Overmars, Chapter 10

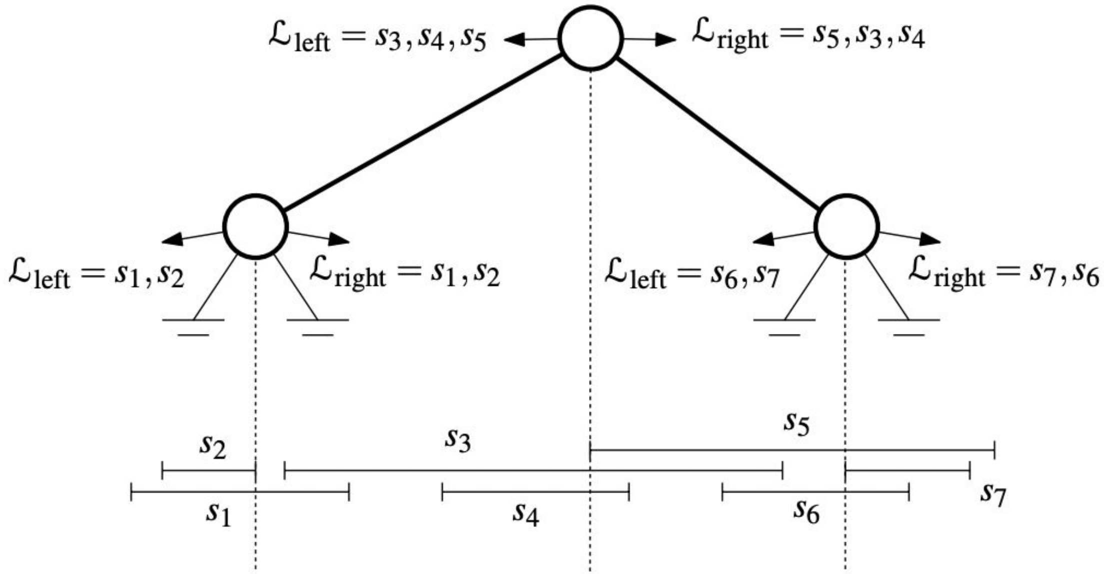
1D Interval Tree Analysis

- For *n* input segments and a query that will return *k* items

- Memory Usage:
  → *O(n)*

- Construction Time:
  → *O(n log n)*

- Query Time:
  → *O(log n + k)*



$\mathcal{L}_{\text{left}} = s_3, s_4, s_5$
$\mathcal{L}_{\text{right}} = s_5, s_3, s_4$
$\mathcal{L}_{\text{left}} = s_1, s_2$
$\mathcal{L}_{\text{right}} = s_1, s_2$
$\mathcal{L}_{\text{left}} = s_6, s_7$
$\mathcal{L}_{\text{right}} = s_7, s_6$
s_2
s_3
s_5
s_1
s_4
s_6
s_7

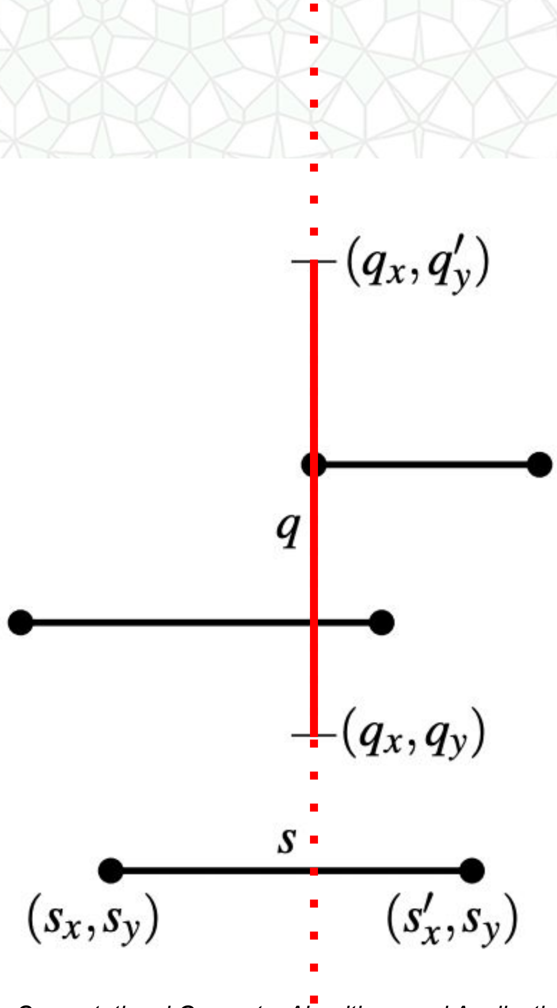
Computational Geometry Algorithms and Applications,
de Berg, Cheong, van Kreveld and Overmars, Chapter 10

Outline for Today

- Review from Last Time: Delaunay Triangulations
- Motivation: Cartography Windowing & Data Selection
- Lecture 8 Review: Points in k-D trees and Range Trees
- 1D Interval Tree
- 1D Interval Tree Analysis
- 2D Interval Tree + Range Tree
- 2D Interval Tree + Range Tree Analysis
- 2D Axis Aligned Segment Query
- Segment Tree for general 2D Segment Query
- Segment Tree Analysis
- Next Time: Quad Trees

How do we Extend to 2D?

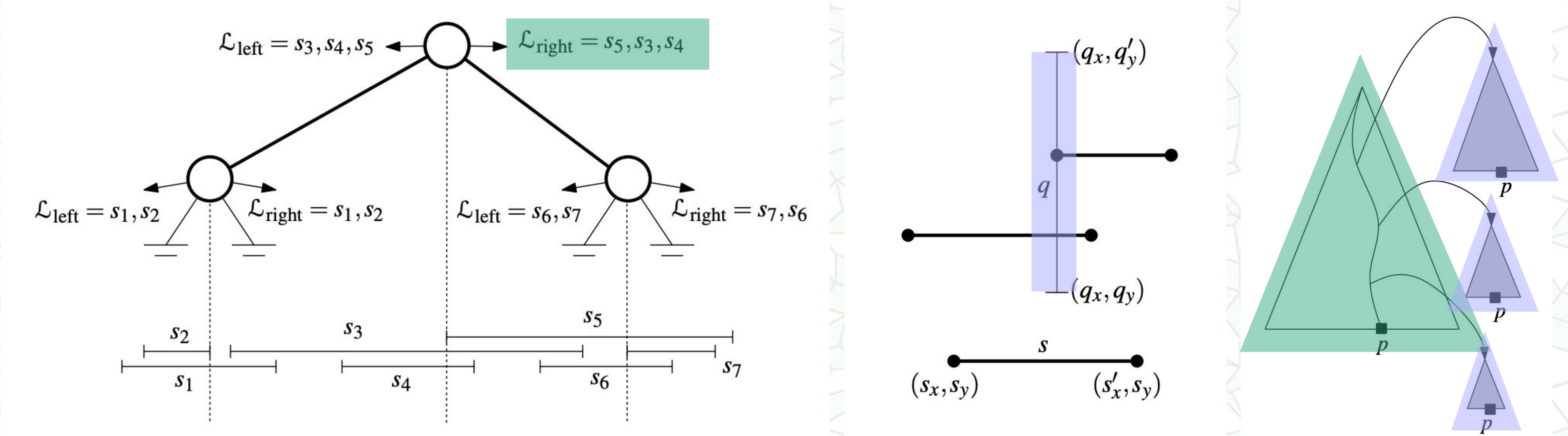
- First consider only horizontal input line segments
- And instead of a query line, we'll have a ***query line segment***



Computational Geometry Algorithms and Applications, de Berg, Cheong, van Kreveld and Overmars, Chapter 10

How do we Extend to 2D?

- We’ll replace the sorted lists of the interval tree with a 2D range query (first by x then by *y*) of the 2D endpoints (Lecture 8)
- This will now require additional memory…



$\mathcal{L}_{\text{left}} = s_3, s_4, s_5$
$\mathcal{L}_{\text{right}} = s_5, s_3, s_4$
$\mathcal{L}_{\text{left}} = s_1, s_2$
$\mathcal{L}_{\text{right}} = s_1, s_2$
$\mathcal{L}_{\text{left}} = s_6, s_7$
$\mathcal{L}_{\text{right}} = s_7, s_6$
s_1
s_2
s_3
s_4
s_5
s_6
s_7
(q_x, q'_y)
q
(q_x, q_y)
s
(s_x, s_y)
(s'_x, s_y)
p
p
p
p

Computational Geometry Algorithms and Applications,
de Berg, Cheong, van Kreveld and Overmars, Chapters 8 & 10

Outline for Today

- Review from Last Time: Delaunay Triangulations
- Motivation: Cartography Windowing & Data Selection
- Lecture 8 Review: Points in k-D trees and Range Trees
- 1D Interval Tree
- 1D Interval Tree Analysis
- 2D Interval Tree + Range Tree
- 2D Interval Tree + Range Tree Analysis
- 2D Axis Aligned Segment Query
- Segment Tree for general 2D Segment Query
- Segment Tree Analysis
- Next Time: Quad Trees

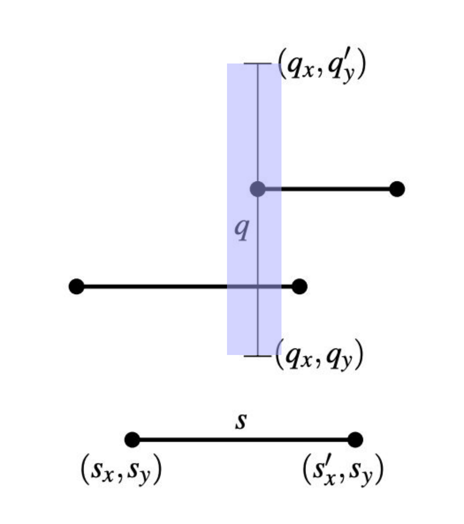
2D Interval Tree + Range Tree Analysis

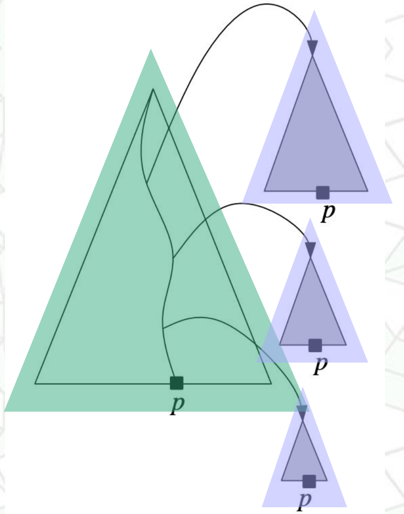
- For *n* horizontal input segments and a query segment that will return *k* items

- Memory Usage:

- Construction Time:

- Query Time:





Computational Geometry Algorithms and Applications,
de Berg, Cheong, van Kreveld and Overmars, Chapters 8 & 10

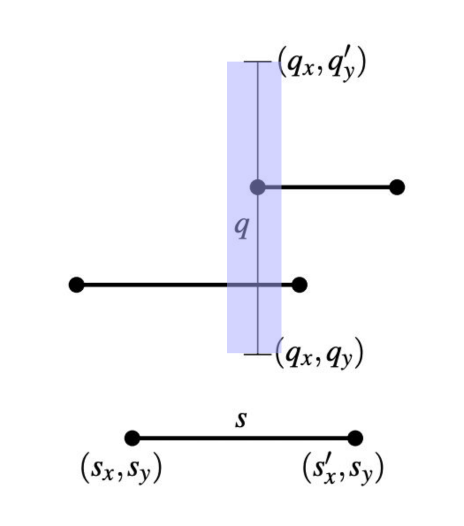
2D Interval Tree + Range Tree Analysis

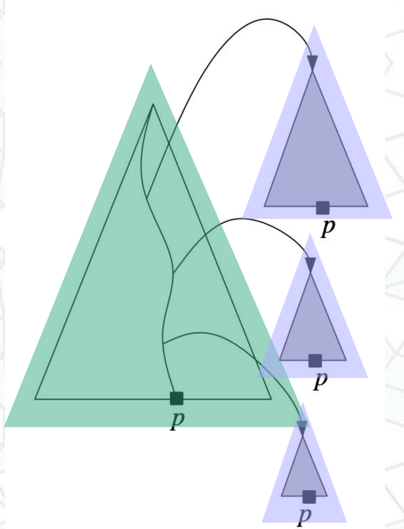
- For *n* horizontal input segments and a query segment that will return *k* items

- Memory Usage:
  → *O(n log n)*

- Construction Time:
  → *O(n log n)*

- Query Time:
  → *O(log n + k)*





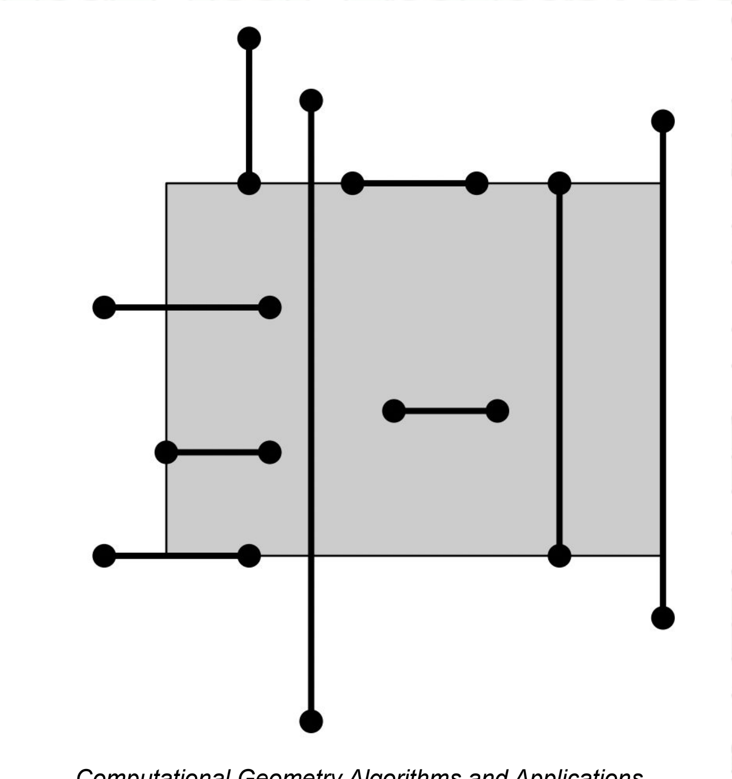
Computational Geometry Algorithms and Applications,
de Berg, Cheong, van Kreveld and Overmars, Chapters 8 & 10

Outline for Today

- Review from Last Time: Delaunay Triangulations
- Motivation: Cartography Windowing & Data Selection
- Lecture 8 Review: Points in k-D trees and Range Trees
- 1D Interval Tree
- 1D Interval Tree Analysis
- 2D Interval Tree + Range Tree
- 2D Interval Tree + Range Tree Analysis
- 2D Axis Aligned Segment Query
- Segment Tree for general 2D Segment Query
- Segment Tree Analysis
- Next Time: Quad Trees

How to handle a 2D Axis-Aligned Query Box?

- Initially, let’s restrict to horizontal & vertical segments

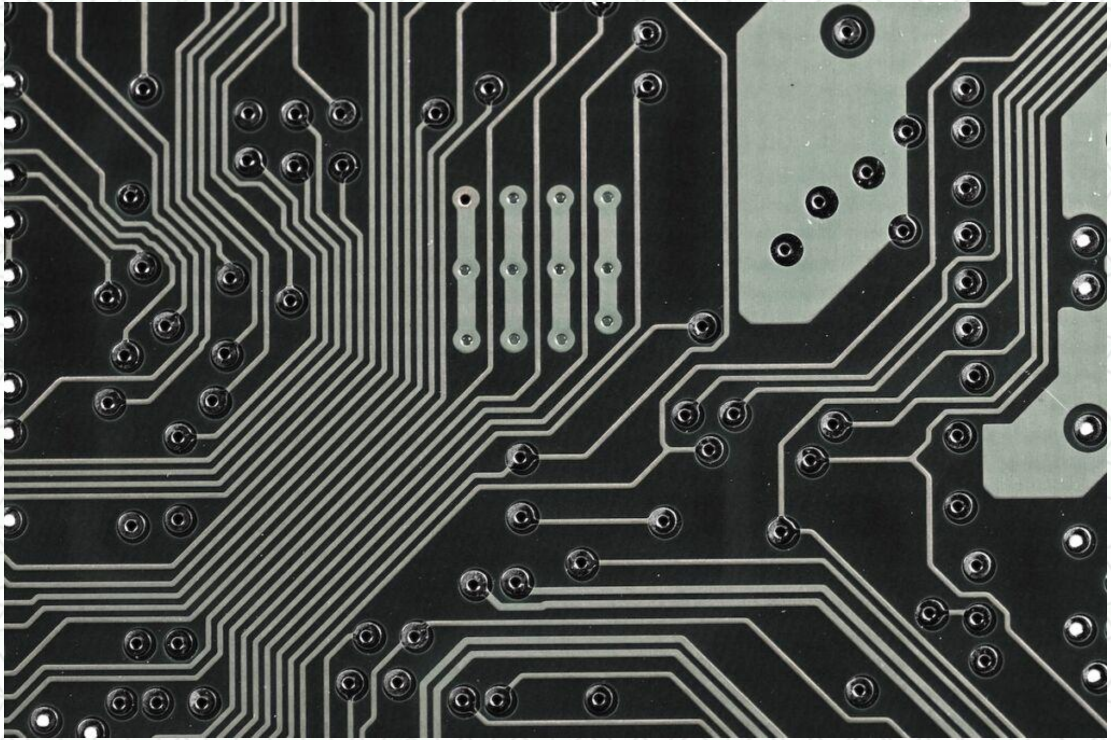


Computational Geometry Algorithms and Applications,
de Berg, Cheong, van Kreveld and Overmars, Chapter 10

How to handle a 2D Axis-Aligned Query Box?

- Why is Axis-Aligned Interesting?
- A bit later we will also restrict ourselves to non-overlapping segments…
  Why is that interesting?

Motivation:
Circuit board layouts →

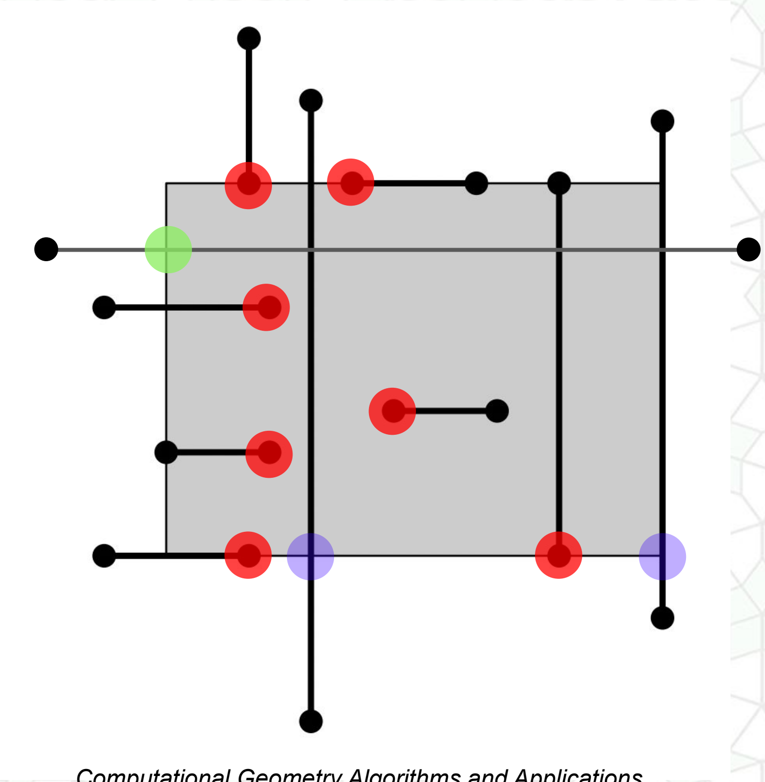


https://www.findlight.net/blog/evolution-of-laser-marking-and-engraving-technologies/

How to handle a 2D Axis-Aligned Query Box?

- Initially, let’s restrict to horizontal & vertical segments

- Case Analysis:
  Segments that touch the query box will:
  - Have one endpoint inside the box,
    OR
  - Will have both endpoints outside the box
    AND
    - Will be a horizontal segment that overlaps the left edge of the box
      OR
    - Will be a vertical segment that overlaps the bottom edge of the box



Computational Geometry Algorithms and Applications, de Berg, Cheong, van Kreveld and Overmars, Chapter 10

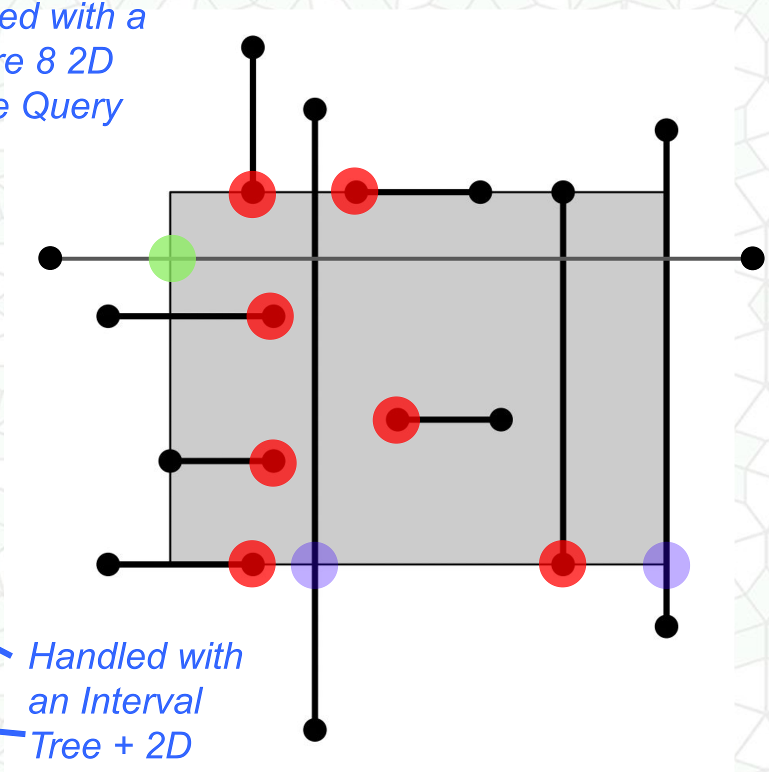
How to handle a 2D Axis-Aligned Query Box?

- Initially, let's restrict to horizontal & vertical segments

- Case Analysis:
  Segments that touch the query box will:
  - Have one endpoint inside the box,
    OR
  - Will have both endpoints outside the box
    AND
    - Will be a horizontal segment that overlaps the left edge of the box
      OR
    - Will be a vertical segment that overlaps the bottom edge of the box

Handled with a Lecture 8 2D Range Query

Handled with an Interval Tree + 2D Range Query

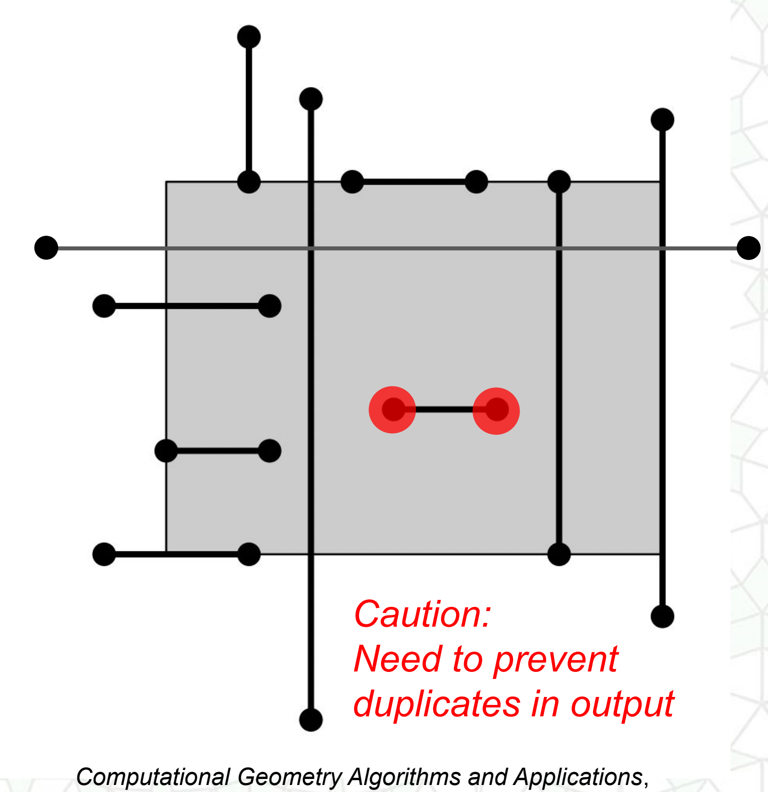


Computational Geometry Algorithms and Applications, de Berg, Cheong, van Kreveld and Overmars, Chapter 10

How to handle a 2D Axis-Aligned Query Box?

- Initially, let's restrict to horizontal & vertical segments

- Case Analysis:
  Segments that touch the query box will:
  - Have one endpoint inside the box, OR
  - Will have both endpoints outside the box AND
    - Will be a horizontal segment that overlaps the left edge of the box OR
    - Will be a vertical segment that overlaps the bottom edge of the box



Caution: Need to prevent duplicates in output

Computational Geometry Algorithms and Applications, de Berg, Cheong, van Kreveld and Overmars, Chapter 10

Outline for Today

- Review from Last Time: Delaunay Triangulations
- Motivation: Cartography Windowing & Data Selection
- Lecture 8 Review: Points in k-D trees and Range Trees
- 1D Interval Tree
- 1D Interval Tree Analysis
- 2D Interval Tree + Range Tree
- 2D Interval Tree + Range Tree Analysis
- 2D Axis Aligned Segment Query
- Segment Tree for General 2D Segment Query
- Segment Tree Analysis
- Next Time: Quad Trees

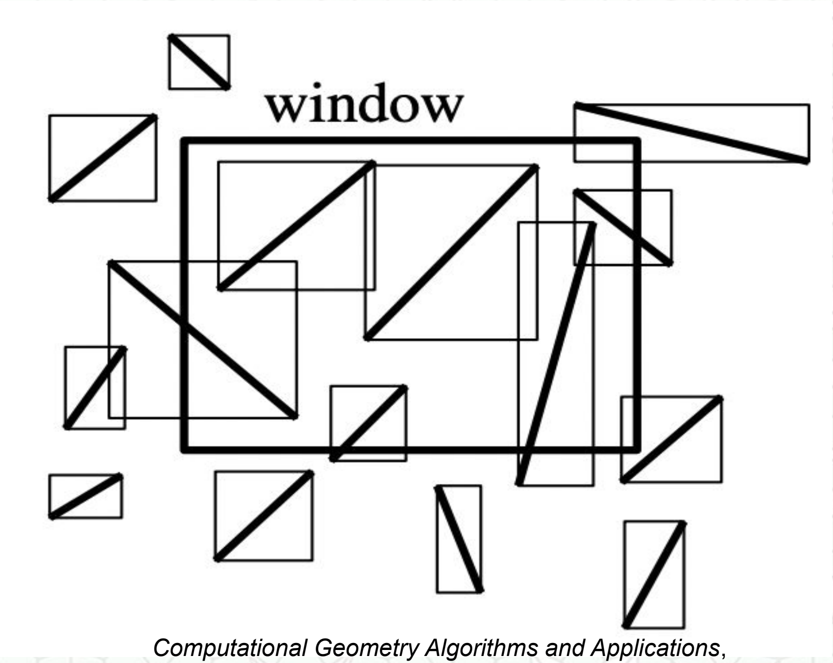
How do we handle General 2D Segments?

- Not restricted to horizontal & vertical segments!
- (Note: We will later insist that the segments do not cross…)

How do we handle General 2D Segments?

- *Do the (sloppy?) Computer Graphics thing…*

  Output the segment if its bounding overlaps the axis-aligned query box

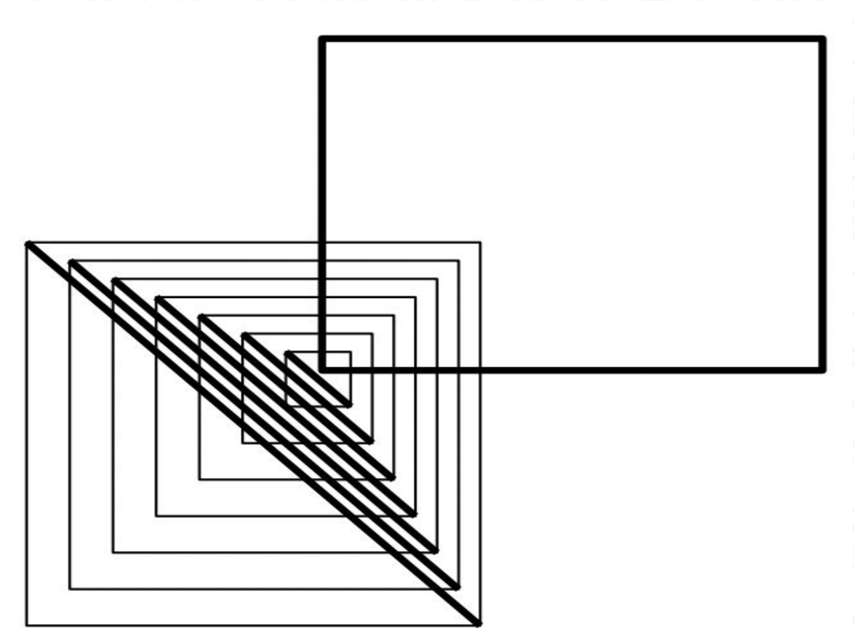


Computational Geometry Algorithms and Applications,
de Berg, Cheong, van Kreveld and Overmars, Chapter 10

How do we handle General 2D Segments?

- *Do the (sloppy?) Computer Graphics thing…*
  Output the segment if its bounding overlaps the axis-aligned query box

- ***We might have LOTS of false positives!***



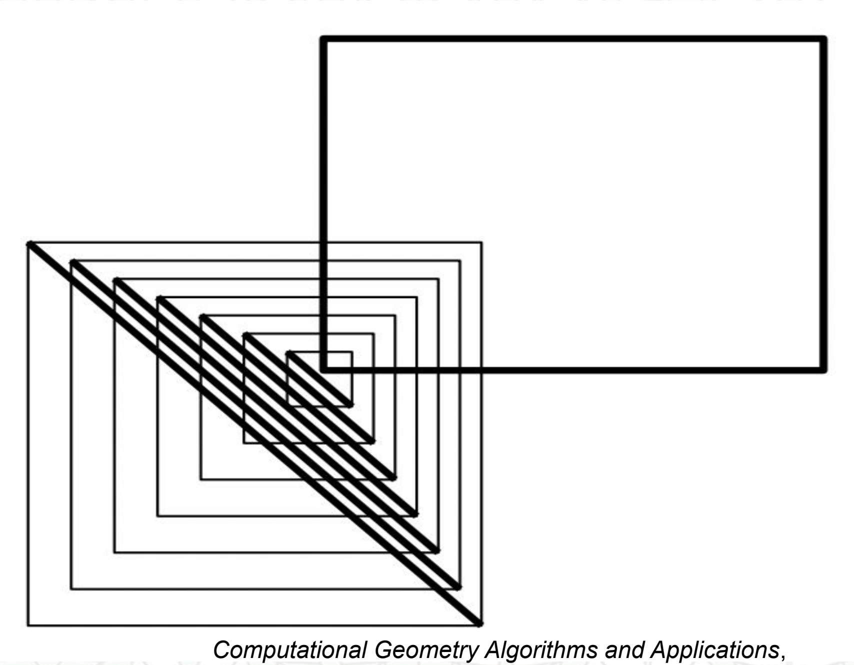
Computational Geometry Algorithms and Applications,
de Berg, Cheong, van Kreveld and Overmars, Chapter 10

How do we handle General 2D Segments?

- *Do the (sloppy?) Computer Graphics thing…*
  Output the segment if its bounding overlaps the axis-aligned query box

- ***We might have LOTS of false positives!***

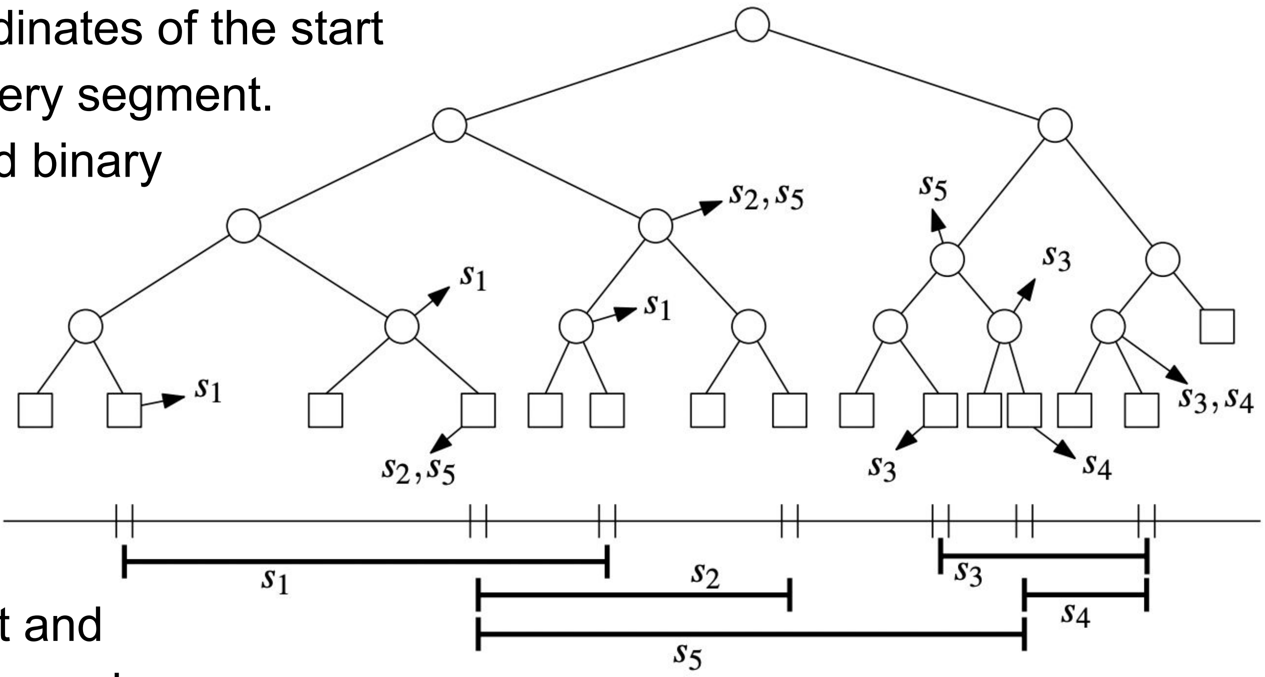
- Can we do better?
  - Ensure good (output sensitive) Performance
    AND
  - Avoid false positives?



Computational Geometry Algorithms and Applications,
de Berg, Cheong, van Kreveld and Overmars, Chapter 10

Segment Tree - First Dimension (*x*)

- First, sort the *x* coordinates of the start and end points of every segment.
- Construct a balanced binary search tree with these *x* values.
- Insert every segment into the structure
- If a segment overlaps both the left and right subranges of the node store it at the node (do not recurse)



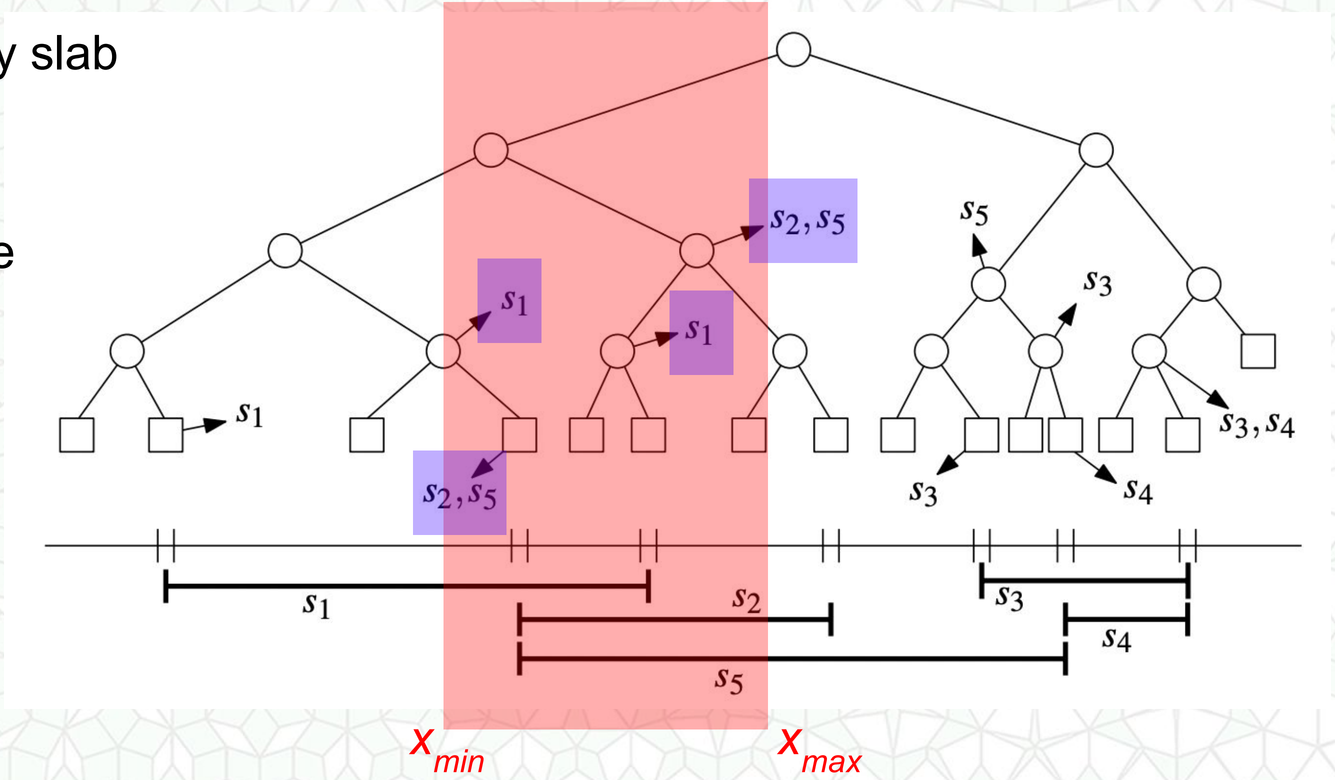
Computational Geometry Algorithms and Applications,
de Berg, Cheong, van Kreveld and Overmars, Chapter 10

Segment Tree - First Dimension (*x*)

- For a vertical query slab (x_{min}, x_{max})

- Walk down the tree
- If the node is in range, return all items at that node
- Recurse left and/or right as appropriate

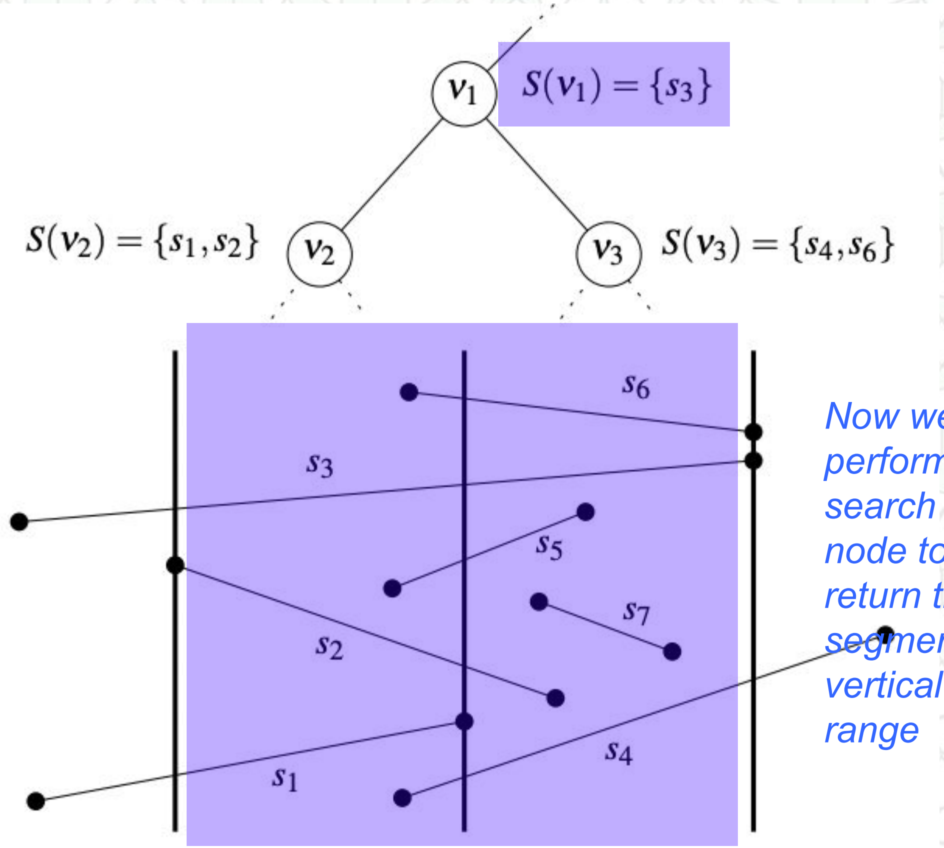
- & filter duplicates…



Computational Geometry Algorithms and Applications,
de Berg, Cheong, van Kreveld and Overmars, Chapter 10

Segment Tree - Second Dimension (*y*)

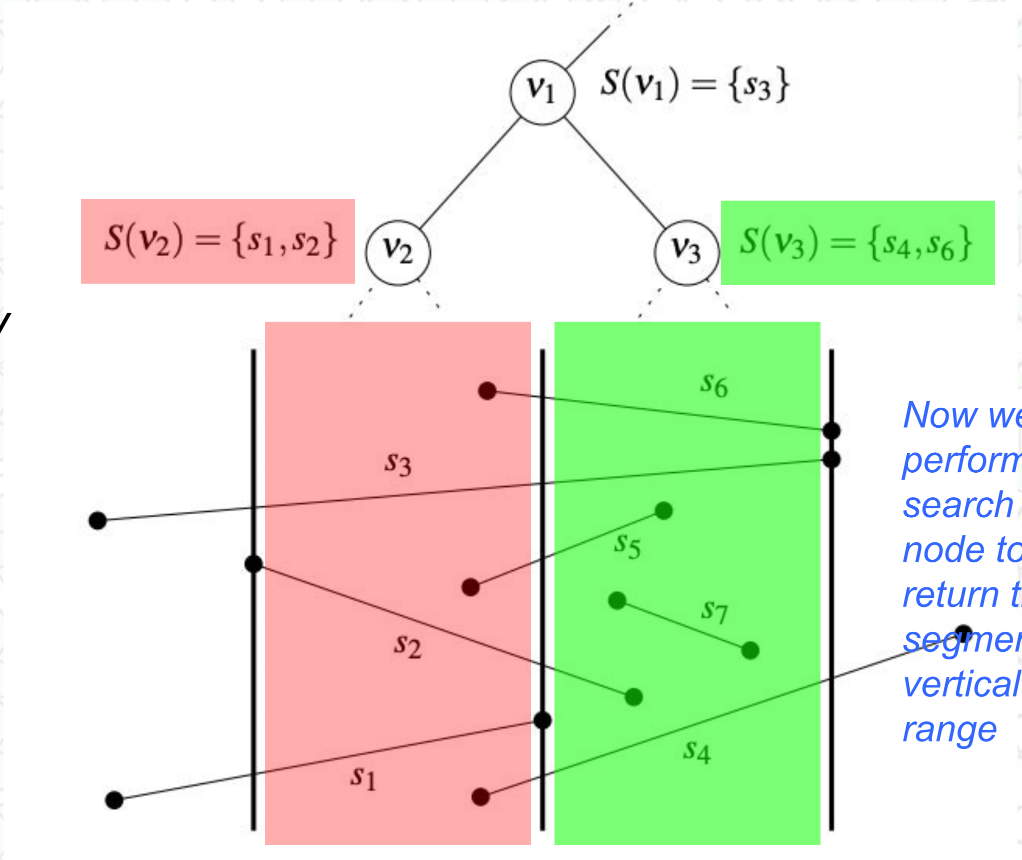
- To efficiently query a vertical range in addition to the horizontal range:
- Sort the segments stored at each node by *y*
- *Remember: this is only the segments that completely overlaps the node's range*
- Note: this is why we require no crossings in the input segments



Now we can perform binary search at each node to only return the segments in the vertical query range

Segment Tree - Second Dimension (*y*)

- To efficiently query a vertical range in addition to the horizontal range:
- Sort the segments stored at each node by *y*
- *Remember: this is only the segments that completely overlaps the node's range*
- Note: this is why we require no crossings in the input segments



Now we can perform binary search at each node to only return the segments in the vertical query range

Outline for Today

- Review from Last Time: Delaunay Triangulations
- Motivation: Cartography Windowing & Data Selection
- Lecture 8 Review: Points in k-D trees and Range Trees
- 1D Interval Tree
- 1D Interval Tree Analysis
- 2D Interval Tree + Range Tree
- 2D Interval Tree + Range Tree Analysis
- 2D Axis Aligned Segment Query
- Segment Tree for General 2D Segment Query
- Segment Tree Analysis
- Next Time: Quad Trees

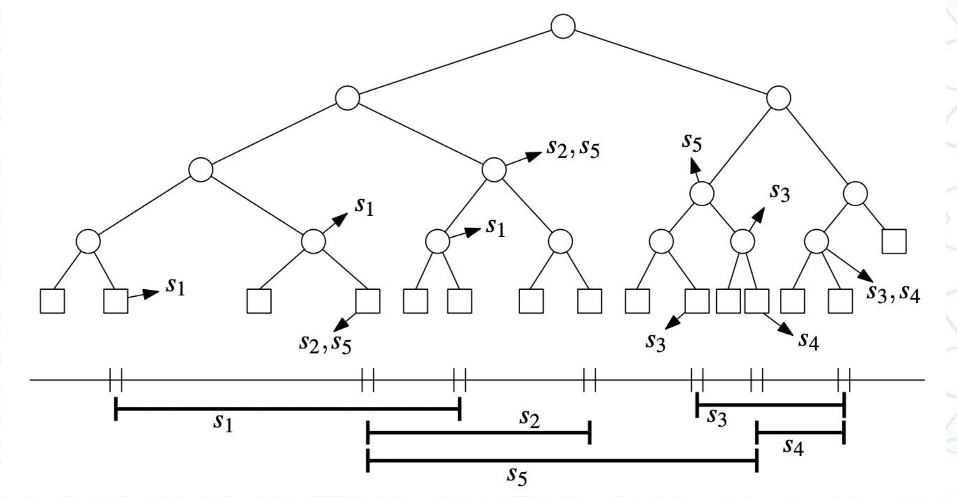
Segment Tree - Analysis

- For *n* input segments, for a query that will return *k* segments

- Memory:
  Each segment is stored in
  at most 2 nodes per level

- Construction Time:
  Presort all endpoints
  by x & y *O(n log n)*

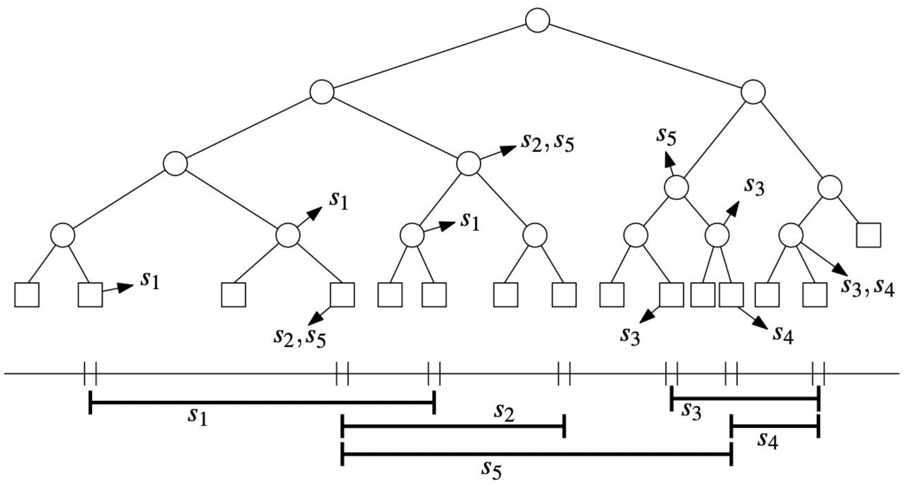
- Query Time:



Computational Geometry Algorithms and Applications,
de Berg, Cheong, van Kreveld and Overmars, Chapter 10

Segment Tree - Analysis

- For *n* input segments, for a query that will return *k* segments
- Memory:
  Each segment is stored in
  at most 2 nodes per level
  → *O(n log n)*
- Construction Time:
  Presort all endpoints
  by x & y *O(n log n)*
  → *O(n log n)*
- Query Time:
  → *O(log n * log n + k)*
  → *O(log ^{2} n + k)*

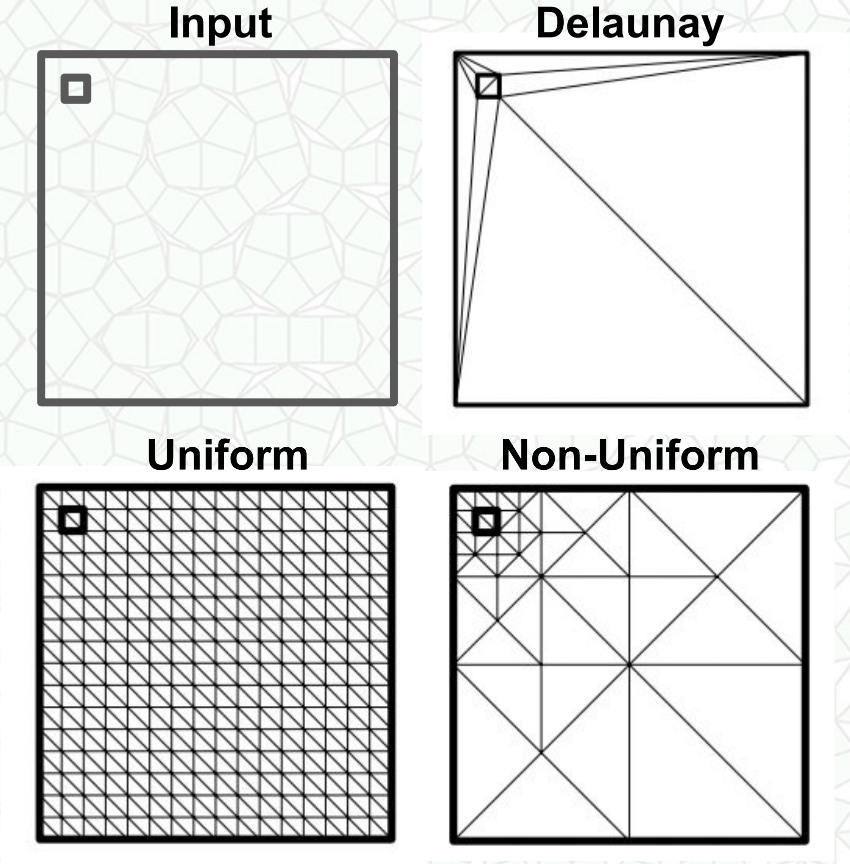


Computational Geometry Algorithms and Applications,
de Berg, Cheong, van Kreveld and Overmars, Chapter 10

Outline for Today

- Review from Last Time: Delaunay Triangulations
- Motivation: Cartography Windowing & Data Selection
- Lecture 8 Review: Points in k-D trees and Range Trees
- 1D Interval Tree
- 1D Interval Tree Analysis
- 2D Interval Tree + Range Tree
- 2D Interval Tree + Range Tree Analysis
- 2D Axis Aligned Segment Query
- Segment Tree for General 2D Segment Query
- Segment Tree Analysis
- Next Time: Quad Trees

Next Time: Quad Trees

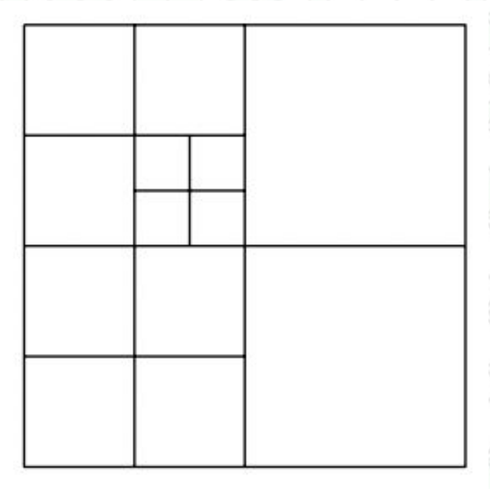


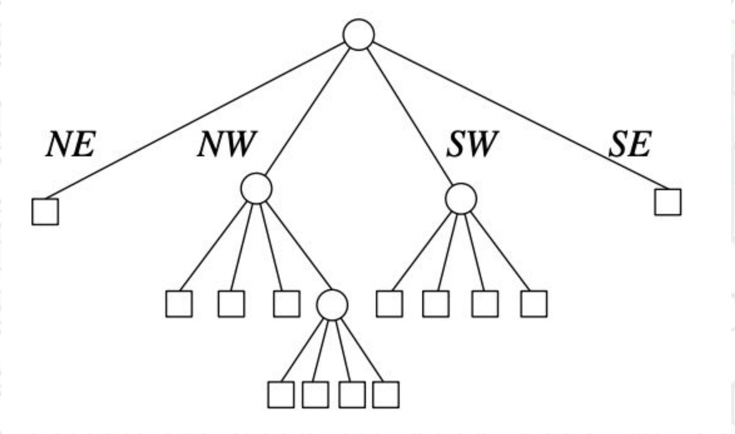
Input

Delaunay

Uniform

Non-Uniform





Computational Geometry Algorithms and Applications,
de Berg, Cheong, van Kreveld and Overmars, Chapter 14