

# CS 372: Computational Geometry

## Lecture 7

### Segment Trees

Antoine Vigneron

King Abdullah University of Science and Technology

October 3, 2012

- 1 Introduction
- 2 Segment trees
- 3 Rectangle intersection
- 4 Stabbing queries in higher dimension

# Outline

- A new data structure: the segment tree.
- Applications:
  - ▶ Stabbing queries.
  - ▶ Rectangle intersection.
- Generalization to higher dimension.

## Reference:

- [Textbook](#) Chapter 10.
- Dave Mount's [lecture notes](#), Lecture 13.

# Stabbing Queries

- Orthogonal range searching: data points, query rectangle.
- Stabbing problem: data rectangles, query point.

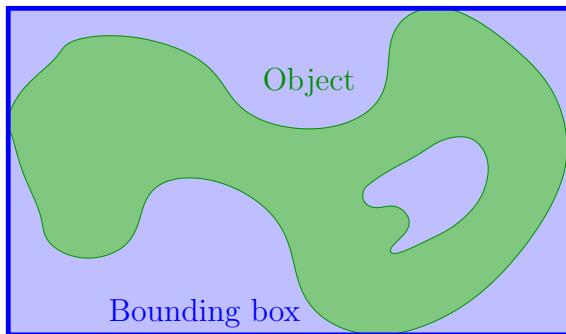
## Problem (One-dimensional stabbing problem)

*Preprocess a set of  $n$  intervals so as to be able to report quickly the  $k$  intervals that contain a query number  $q$ .*

- In  $\mathbb{R}^d$ :
  - ▶ Input: a set of  $n$  boxes, a query point  $q$ .
  - ▶ Output: the  $k$  boxes that contain  $q$ .

# Motivation

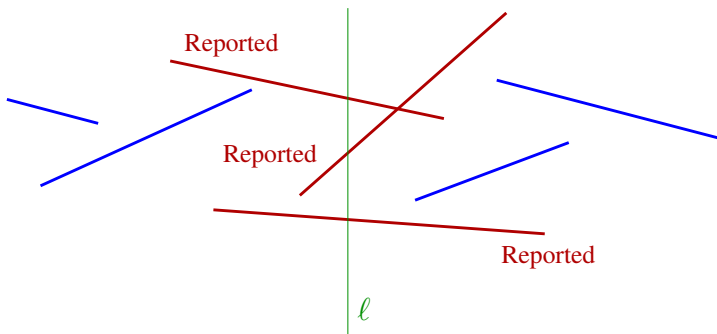
- In graphics and databases, geometric objects are often approximated by their bounding box.



- Query: Which objects does point  $x$  belong to?
- First find objects whose bounding boxes intersect  $x$ .

# Segment Trees

- A data structure to store intervals of  $\mathbb{R}$ , or segments in  $\mathbb{R}^2$ .
- Allows to answer stabbing queries.
  - ▶ In  $\mathbb{R}^2$ : Report the segments that intersect a query vertical line  $\ell$ .



- ▶ Query time:  $O(k + \log n)$ .
- ▶ Space usage:  $O(n \log n)$ .
- ▶ Preprocessing time:  $O(n \log n)$ .

# Notation

- Let  $S = (s_1, s_2, \dots, s_n)$  be a set of segments in  $\mathbb{R}^2$ .
- Let  $E$  be the set of the  $x$ -coordinates of the endpoints of the segments of  $S$ .
- First sort  $E$ :

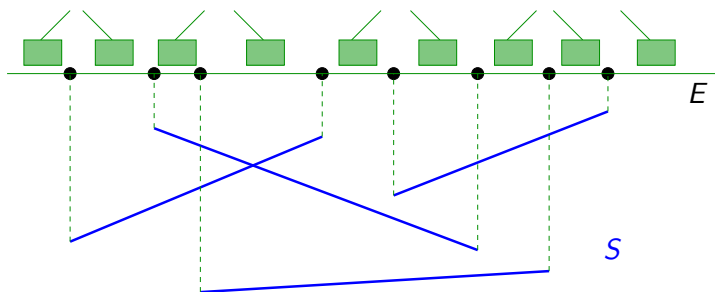
$$E = \{e_1, e_2, \dots, e_m\},$$

$$e_1 < e_2 < \dots < e_m.$$

- ▶  $m \leq 2n$ , with equality in general position.

# Atomic Intervals

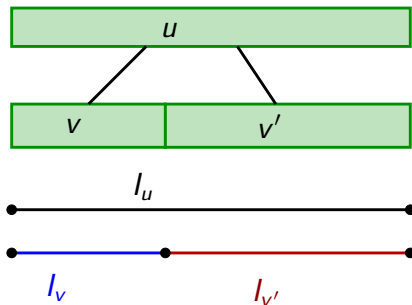
- $E$  splits  $\mathbb{R}$  into  $m + 1$  *atomic intervals*:
  - ▶  $(-\infty, e_1]$
  - ▶  $[e_i, e_{i+1}]$  for  $i \in \{1, 2, \dots, m - 1\}$
  - ▶  $[e_m, \infty)$
- These intervals are stored at the leaves of the segment tree.





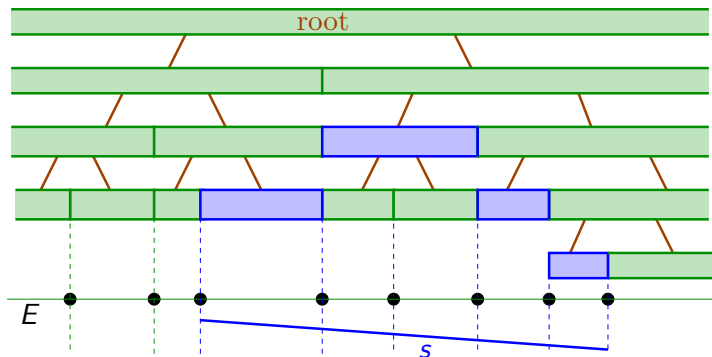
## Internal Nodes

- The segment tree  $\mathcal{T}$  is a balanced binary search tree.
- Each internal node  $u$  with children  $v$  and  $v'$  is associated with an interval  $I_u = I_v \cup I_{v'}$ .
- An *elementary interval* is an interval associated with a node of  $\mathcal{T}$ . (It can be an atomic interval).



## Partitioning a Segment

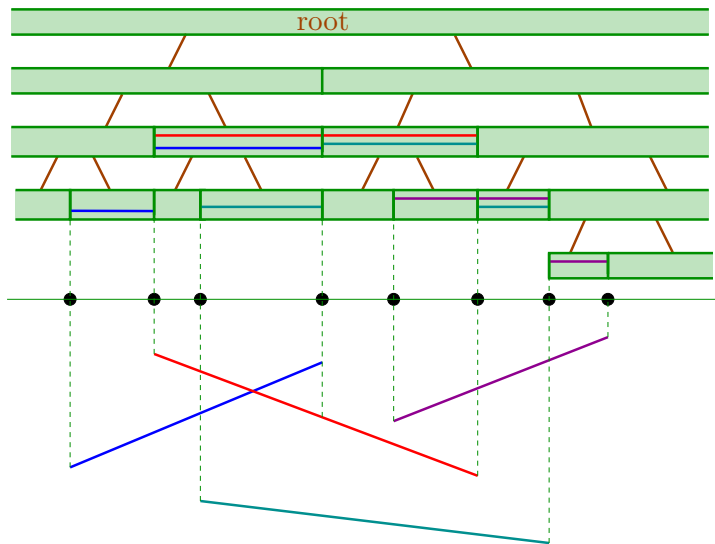
- Let  $s \in S$  be a segment whose endpoints have  $x$ -coordinates  $e_i, e_j$ .
- $[e_i, e_j]$  is split into several elementary intervals.
- These intervals are chosen as close as possible to the root.
- Segment  $s$  is stored at each node associated with these elementary intervals.



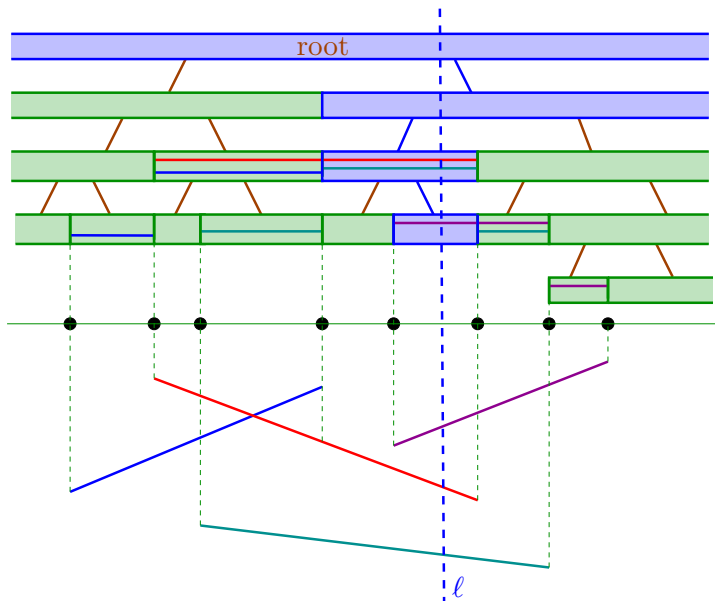
# Standard Lists

- At each node  $u$ , we store a *standard list* of segments  $L_u$ .
- Let  $e_i < e_j$  be the  $x$ -coordinates of the endpoints of  $s \in S$ .
- Then  $s$  is stored in  $L_u$  iff  $I_u \subset [e_i, e_j]$  and  $I_{\text{parent}(u)} \not\subset [e_i, e_j]$ .  
(See previous slide and next slide.)

# Example



# Answering a Stabbing Query



# Pseudocode

## Answering a stabbing query

**Algorithm** *ReportStabbing*( $u, x_\ell$ )

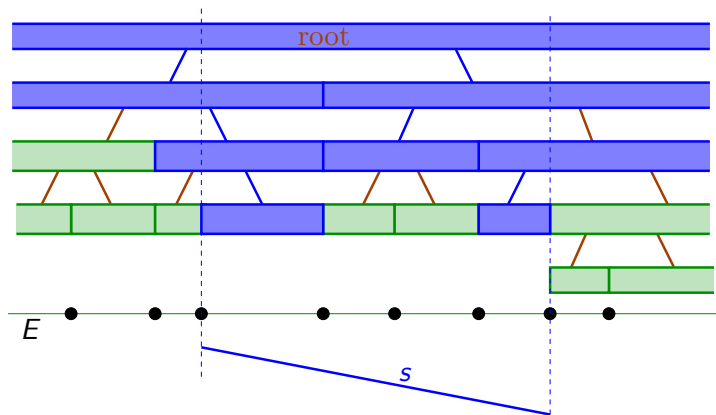
**Input:** The root  $u$  of  $\mathcal{T}$ , the  $x$ -coordinate  $x_\ell$  of  $\ell$ .

**Output:** The segments in  $S$  that cross  $\ell$ .

1. Output  $L_u$ .
2. **if**  $u$  is an internal node
3.     **then if**  $x_\ell \in I_{u.left}$
4.         **then** *ReportStabbing*( $u.left, x_\ell$ ).
5.     **if**  $x_\ell \in I_{u.right}$
6.         **then** *ReportStabbing*( $u.right, x_\ell$ ) .

- Query time:  $O(k + \log n)$ .

# Inserting a Segment



# Pseudocode

## Inserting a segment into a segment tree

**Algorithm**  $Insert(u, s)$

**Input:** The root  $u$  of  $\mathcal{T}$ , a segment  $s = ((x_1, y_1), (x_2, y_2))$ .

1. **if**  $I_u \subset [x_1, x_2]$
2.     **then**  $L_u \leftarrow L_u \cup \{s\}$
3.     **else**
4.         **if**  $(x_1, x_2] \cap I_{u.left} \neq \emptyset$
5.             **then**  $Insert(u.left, s)$
6.         **if**  $[x_1, x_2) \cap I_{u.right} \neq \emptyset$
7.             **then**  $Insert(u.right, s)$



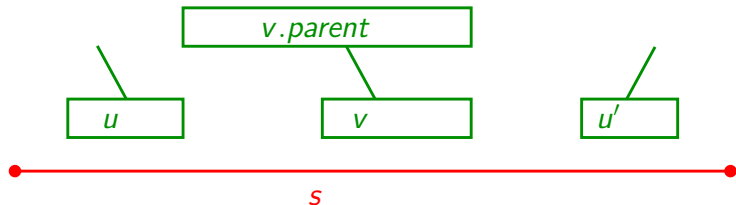
# Property

## Property

*Any segment  $s$  is stored at most twice at each level of the segment tree  $\mathcal{T}$ .*

Proof (by contradiction):

- Suppose that  $s$  is stored at more than 2 nodes at level  $i$ .
- Let  $u$  be the leftmost such node,  $u'$  be the rightmost.
- Let  $v$  be another node at level  $i$  containing  $s$ .



- Then  $I_{v.parent} \subset [x_1, x_2]$ .
- So  $s$  cannot be stored at  $v$ .

# Analysis

The property in previous slide implies:

- Space usage  $O(n \log n)$ .
  - ▶ Actually space usage is  $\Theta(n \log n)$ . (Example?)
- Insertion in  $O(\log n)$  time.  
(Similar proof: four nodes at most are visited at each level).
- Query time  $O(k + \log n)$ .
- Preprocessing:
  - ▶ Sort endpoints in  $\Theta(n \log n)$  time.
  - ▶ Build empty segment tree over these endpoints in  $O(n)$  time.
  - ▶ Insert  $n$  segments into  $\mathcal{T}$  in  $O(n \log n)$  time.
  - ▶ Overall  $\Theta(n \log n)$  preprocessing time.

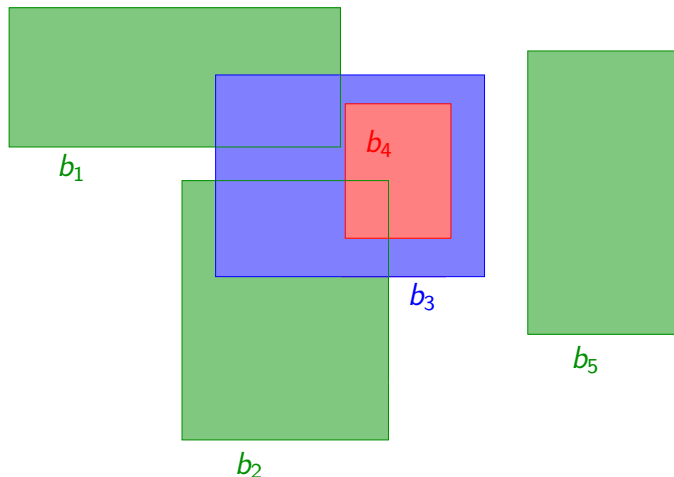
# Rectangle Intersection

## Problem (rectangle intersection reporting)

*Given a set a set  $B$  of  $n$  boxes in  $\mathbb{R}^2$ , report all the pairs of boxes  $b, b' \in B$  such that  $b \cap b' \neq \emptyset$ .*

- Using segment trees, we give an  $O(k + n \log n)$  time algorithm when  $k$  is the number of intersecting pairs.
  - ▶ This is optimal.
  - ▶ Faster than our line segment intersection algorithm.
- Space usage:  $\Theta(n \log n)$  due to segment trees.
  - ▶ Space usage is not optimal.  
(Space  $O(n)$  is possible with the same running time.)

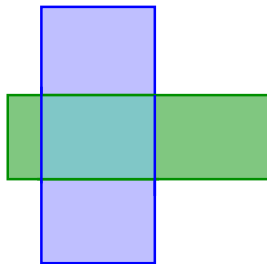
## Example



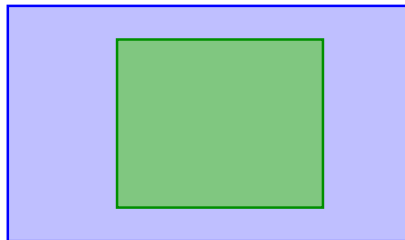
Output:  $(b_1, b_3), (b_2, b_3), (b_2, b_4), (b_3, b_4)$ .

# Two Types of Intersections

Overlap



Inclusion



- Intersecting edges.
  - ▶ Reduces to intersection reporting for axis-aligned segments.
- We can find them using stabbing queries.

## Reporting overlaps

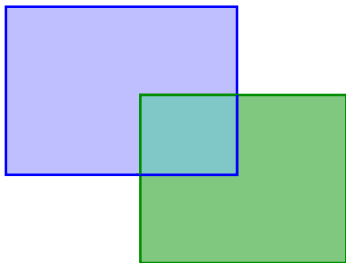
- Equivalent to reporting intersecting edges.
- Plane sweep approach.
- Sweep line status: BBST containing the horizontal line segments that intersect the sweep line, by increasing  $y$ -coordinates.
- Each time a vertical line segment is encountered, report intersection by range searching in the BBST.
- Preprocessing time:  $O(n \log n)$  for sorting endpoints.
- Running time:  $O(k + n \log n)$ .

# Reporting inclusions

- Still using plane sweep.
- Sweep line status: the boxes that intersect the sweep line  $\ell$ , in a segment tree with respect to  $y$ -coordinates.
  - ▶ The endpoints are the  $y$ -coordinates of the horizontal edges of the boxes.
  - ▶ At a given time, only rectangles that intersect  $\ell$  are in the segment tree.
  - ▶ We can perform insertion and deletions in a segment tree in  $O(\log n)$  time.
- Each time a vertex of a box is encountered, perform a stabbing query in the segment tree.

## Remarks

- At each step a box intersection can be reported several times.
- In addition there can be overlap and vertex stabbing a box at the same time.



- To obtain each intersecting pair only once, make some simple checks. (How?)



# Stabbing Queries in Higher Dimension

## Problem (Stabbing queries in $\mathbb{R}^d$ )

*Preprocess a set  $B$  of  $n$  boxes in  $\mathbb{R}^d$ , so as to be able to report quickly all the boxes that contain a query point  $q$ .*

Approach:

- We use a multi-level segment tree.
- Inductive definition, induction on  $d$ .
- First, we store  $B$  in a segment tree  $\mathcal{T}$  with respect to  $x_1$ -coordinate.
- At each node  $u$  of  $\mathcal{T}$ , store a  $(d - 1)$ -dimensional multi-level segment tree over  $L_u$ , with respect to  $(x_2, x_3 \dots x_d)$ .

# Stabbing Queries in Higher Dimension

We assume we are in fixed dimension, that is,  $d = O(1)$ .

Answering queries:

- Search for  $q$  in  $\mathcal{T}$ .
- For each node in the search path, query recursively the  $(d - 1)$ -dimensional multi-level segment tree.
- There are  $O(\log n)$  such queries.
- By induction on  $d$ , we can prove:
  - ▶ Query time:  $O(k + \log^d n)$ .
  - ▶ Space usage:  $O(n \log^d n)$ .
  - ▶ Preprocessing time :  $O(n \log^d n)$ .