

### COMP5045 Computational Geometry

#### COMMONWEALTH OF AUSTRALIA Copyright Regulations 1969 WARNING

This material has been reproduced and communicated to you by or on behalf of the University of Sydney pursuant to Part VB of the Copyright Act 1968 (**the Act**).

The material in this communication may be subject to copyright under the Act. Any further copying or communication of this material by you may be the subject of copyright protection under the Act.

**Do not remove this notice.**

---

---

---

---

---

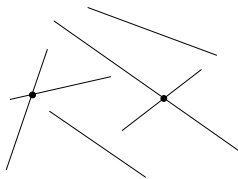
---

---

---

### Segment intersection

## Segment intersection



---

---

---

---

---

---

---

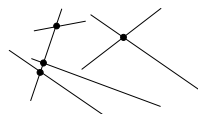
---

### Segment intersection

**Input:** A set of  $n$  line segments  $S = \{s_1, s_2, \dots, s_n\}$  in the plane, represented as pairs of coordinates.

**Intersection detection:**  
Is there a pair of lines in  $S$  that intersect?

**Intersection reporting:**  
Find all pairs of segments that intersect.



---

---

---

---

---

---

---

---

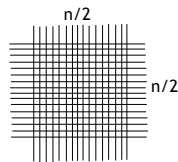
### Brute force algorithm

Check every possible pair of segments if they intersect  
⇒  $O(n^2)$  time

Can we do better?

Detection? Maybe!

Reporting? Nope!



---

---

---

---

---

---

---

---

### Plane sweep algorithm

Plane sweep (general method):

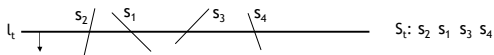
1. Sweep the input from top to bottom and stop at event points
2. Maintain invariant
3. At each event point restore invariant

Event points?

event points = end points

Invariant:

we know the order of the segments intersecting the sweep line



---

---

---

---

---

---

---

---

### Plane sweep algorithm

$l_t$  : the horizontal line at  $y=t$

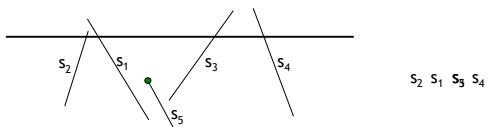
$S_t$  : the sequence of the segments that intersects  $l_t$  in order from left to right.

Idea:

Maintain  $S_t$  while  $l_t$  moves from top to bottom

Invariant:

We know  $S_t$



---

---

---

---

---

---

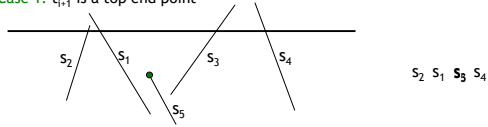
---

---

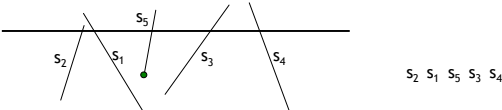
## Event handling

Initially: Let  $t_1, t_2, \dots, t_{2n}$  be the y-coordinates of the endpoints ordered from top to bottom

Case 1:  $t_{i+1}$  is a top endpoint



Case 2:  $t_{i+1}$  is a bottom endpoint



7

---

---

---

---

---

---

---

---

## Data structure

We need to store  $S_t$  in a data structure that supports fast insertions and deletions.

**Structure:** Balanced binary tree  
Each update can be done in  $O(\log n)$  time

**Problem:** How do we check intersections!

8

---

---

---

---

---

---

---

---

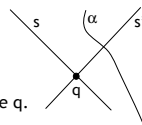
## Intersection points

**Observation:** Let  $q$  be the topmost intersection point, where  $q$  is an intersection point between the segments  $s$  and  $s'$  with y-coordinate  $t$ , then  $s$  and  $s'$  are adjacent in  $S_t$ .

**Proof:**

Assume  $S_t = (\dots s \dots \alpha \dots s' \dots)$

1. Bottom endpoint of  $\alpha$  must be below  $q$ .
2. If  $q$  is to the left of  $\alpha$  then  $\alpha$  intersects  $s'$  above  $q$ .  
 $\Rightarrow$  contradicts that  $q$  is topmost intersection
3. Similarly,  $q$  cannot lie to the right of  $\alpha$   
 $\Rightarrow$  contradiction!



9

---

---

---

---

---

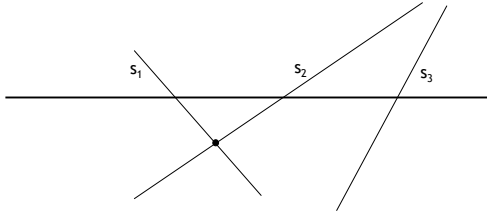
---

---

---

## Intersection point

**Conclusion:** To detect an intersection we only need to check adjacent segments in  $S_t$ .




---

---

---

---

---

---

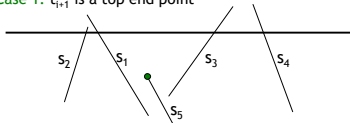
---

---

## Event handling

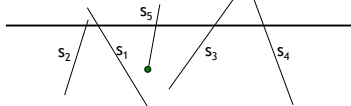
Initially: Let  $t_1, t_2, \dots, t_{2n}$  be the y-coordinates of the endpoints ordered from top to bottom

Case 1:  $t_{i+1}$  is a top endpoint



$S_2 S_1 S_3 S_4$   
Check if  $S_1$  and  $S_5$ , or  $S_3$  and  $S_5$  intersect.

Case 2:  $t_{i+1}$  is a bottom endpoint



$S_2 S_1 S_3 S_4$   
Check if  $S_1$  and  $S_3$  intersect.

---

---

---

---

---

---

---

---

## Algorithm

### Algorithm DetectIntersection( $S$ )

- Store the segments in  $S_t$  in a balanced binary search tree  $T$  w.r.t. the order along  $l_t$ .
- Given a segment in  $T$  the previous and next segment along  $l_t$  can be found in  $O(\log n)$  time.
- When deleting a segment in  $T$  two segments become adjacent. We can find them in  $O(\log n)$  time and check if they intersect.
- When inserting a segment  $s_i$  in  $T$  it becomes adjacent to two segments. We can find them in  $O(\log n)$  time and check if they intersect  $s_i$ .
- If we find an intersection we're done!

Time complexity?

---

---

---

---

---

---

---

---

### Algorithm - detection

Every endpoint is an event point  $\Rightarrow 2n$  event points

Insert segment  $s$

Add  $s$  to  $T$ :  $O(\log n)$   
Check neighbours:  $2 \times O(\log n)$

Delete segment  $s$

Remove  $s$  from  $T$ :  $O(\log n)$   
Check new neighbours:  $2 \times O(\log n)$

Total:  $O(n \log n)$

---

---

---

---

---

---

---

---

### Lower bound

Can we do better than  $O(n \log n)$ ?

Element Uniqueness problem  $\Omega(n \log n)$   
Given a set of real numbers, are they distinct?

Element Uniqueness is a simpler version than our problem  
(1D compared to 2D)

---

---

---

---

---

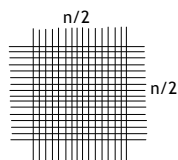
---

---

---

### Intersection Reporting

Brute force: Try all pairs of segments  
Time:  $\Theta(n^2)$  - optimal in worst case



What if the number of intersections is small?

Can we use the same approach to report all intersections?

---

---

---

---

---

---

---

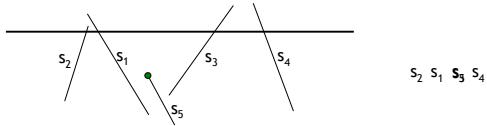
---

### Plane sweep algorithm

$l_t$  : the horizontal line at  $y=t$   
 $S_t$  : the sequence of the segments that intersects  $l_t$  in order from left to right.

Idea:  
Maintain  $S_t$  while  $l_t$  moves from top to bottom

Invariant:  
We know  $S_t$




---

---

---

---

---

---

---

---

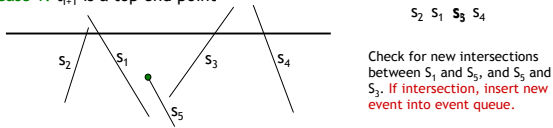
---

---

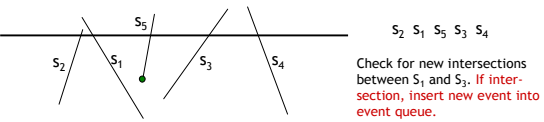
### Event handling

Initially: Let  $t_1, t_2, \dots, t_{2n}$  be the y-coordinates of the endpoints

Case 1:  $t_{i+1}$  is a top endpoint



Case 2:  $t_{i+1}$  is a bottom endpoint




---

---

---

---

---

---

---

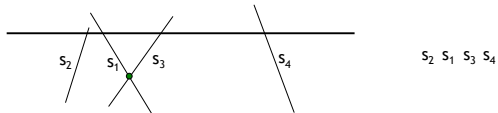
---

---

---

### Event handling: Case 3

Case 3:  $t_{i+1}$  is an intersection!



- Swap position of intersecting segments
- Report intersection
- Check for new intersections between  $S_1$  and  $S_4$ , and  $S_3$  and  $S_2$ . If intersection, insert new event into event queue.

---

---

---

---

---

---

---

---

---

---

### Events and event queue Q

- Two kinds of event points: endpoints and intersection points
- We don't know intersection points in advance
  - We can't sort them in advance
- We will use an event queue Q
  - Q contains all the event points
- Event points ordered w.r.t. y-coordinates
- Implementation:
  - Insert an event point in  $O(\log n)$  time
  - Delete (dequeue) the event point with smallest y-value in  $O(\log n)$  time

---

---

---

---

---

---

---

---

---

---

### Algorithm

- Initial event points in the event queue: all end points ( $2n$ )
- Sweep line from top to bottom - maintain  $S_t$  (use balanced binary tree)
- Stop at each event point - dequeue event point and handle it
  - Top end point
  - Bottom end point
  - Intersection point

1. Update  $S_t$
2. Check for new intersections
3. If new intersection, insert it into the event queue (in order)

---

---

---

---

---

---

---

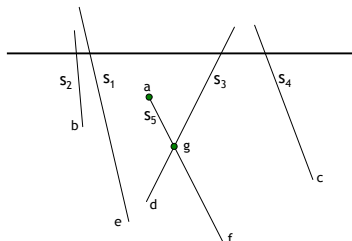
---

---

---

### Event handling - new case

Case 1:  $t_{i+1}$  is a top end point



$S_2 S_1 S_3 S_4$

Q: a **g** c d e f

---

---

---

---

---

---

---

---

---

---

### Output sensitive

#### Running time?

- $k = \#$  intersections
- $k$  is also the output size
- Each event can be processed in  $O(\log(n+k))$  time
- Number of events?  $O(n+k)$
- Total time  $O((n+k) \log(n+k))$  **output sensitive!**
- Total space  $O(n+k)$

---

---

---

---

---

---

---

---

### Summary: Intersections

**Input:** A set of  $n$  line segments  $S = \{s_1, s_2, \dots, s_n\}$  in the plane, represented as pairs of coordinates.

**Intersection detection:** Is there a pair of lines in  $S$  that intersect?

Plane sweep:  $O(n \log n)$  time  
Requires  $\Omega(n \log n)$  time in worst case

**Intersection reporting:** Find all pairs of segments that intersect.

Brute force:  $\Theta(n^2)$  time  
Plane sweep:  $O((n+k) \log n)$  time

---

---

---

---

---

---

---

---

### References

**Intersection detection:** Shamos & Hoey'76

**Intersection reporting:**  
Bentley & Ottmann'79  $O((n+k) \log n)$  time and  $O(n+k)$  space

Pach & Sharir'91  $O((n+k) \log n)$  time and  $O(n)$  space

Chazelle & Edelsbrunner  $O(n \log n+k)$  time and  $O(n+k)$  space

Balaban'95  $O(n \log n+k)$  time and  $O(n)$  space **Optimal!**

---

---

---

---

---

---

---

---