Line Segment Intersection

Lecture 2, CS 4235
15 January 2004

Antoine Vigneron

antoine@comp.nus.edu.sg

National University of Singapore

News

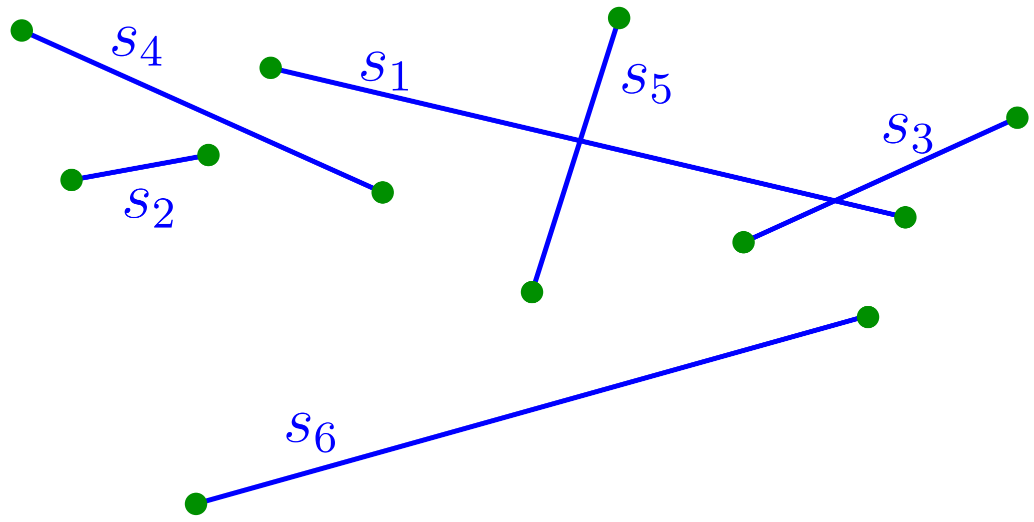
- linear programming lecture is pushed later and randomized
- midterm 1
  - on week 6 (the day after lecture 5)
  - covers lectures 1–4
- midterm 2
  - on week 10 (the day after lecture 9)
  - covers lectures 1–8
  - emphasis on lectures 5–8
- see syllabus on IVLE for the details

Outline

- reference: Dave Mount notes, lecture 5
- we study two line segments intersection problems
  - intersection detection
  - intersection reporting
- we introduce a computing paradigm: plane sweep
- we introduce two notions of algorithm analysis
  - output–sensitive algorithm
  - space complexity

Problems

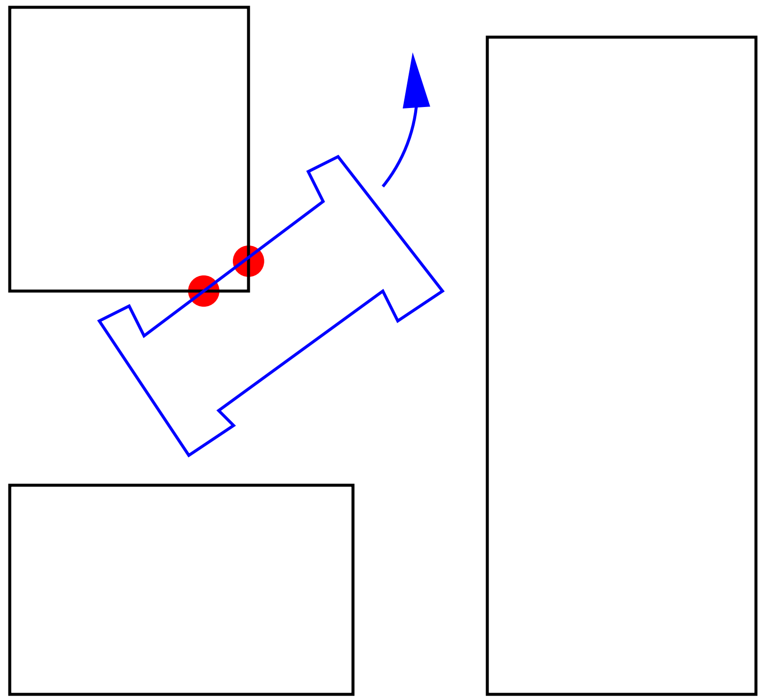
Input: a set $S = \{s_1, s_2, \ldots s_n\}$ of n line segments in $\mathbb{R}^2$ given by the coordinates of their endpoints.



- *Intersection detection*: is there a pair $(s_i, s_j) \in S^2$ such that $i \neq j$ and $s_i \cap s_j \neq \emptyset$?
- *Intersection reporting*: find all pairs $(s_i, s_j) \in S^2$ such that $i \neq j$ and $s_i \cap s_j \neq \emptyset$

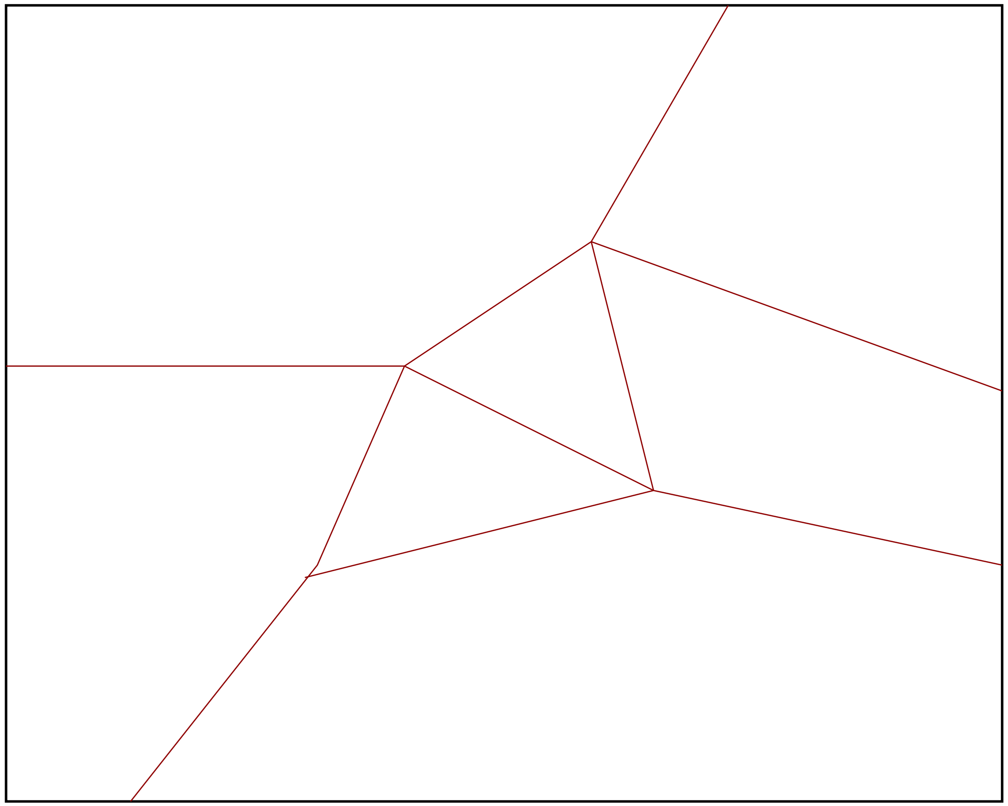
Motivation

- Motion planning $\Rightarrow$ collision detection



Motivation

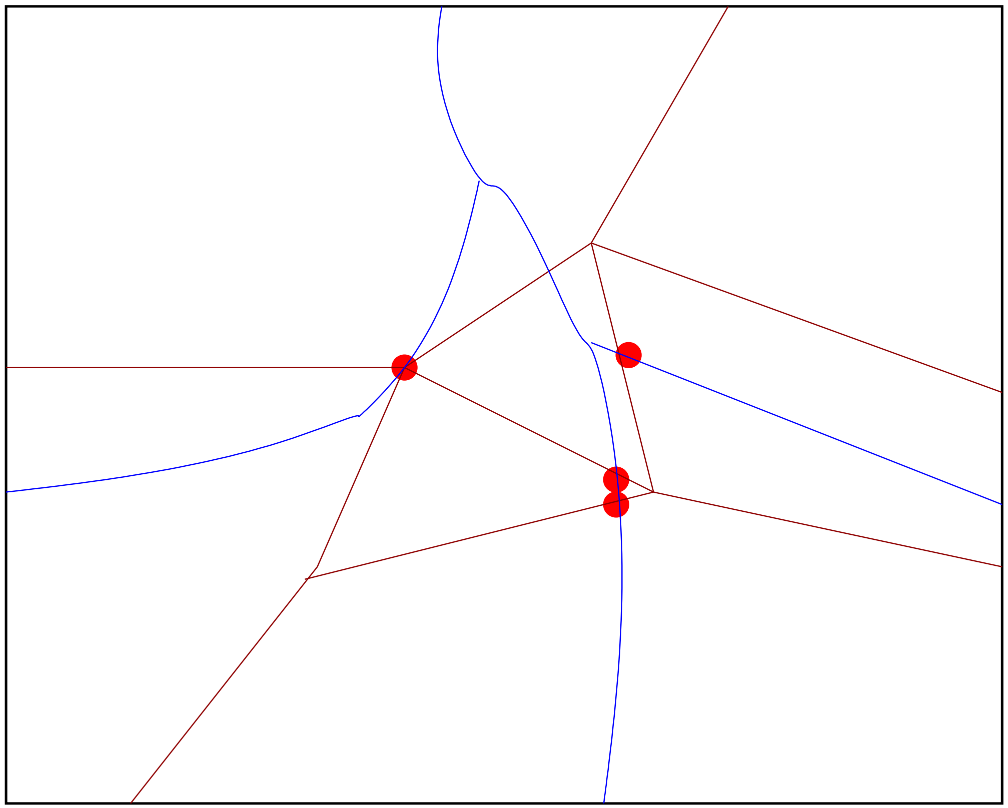
- Geographic Information Systems $\Rightarrow$ map overlay
- More generally: spatial join in databases



ROAD MAP

Motivation

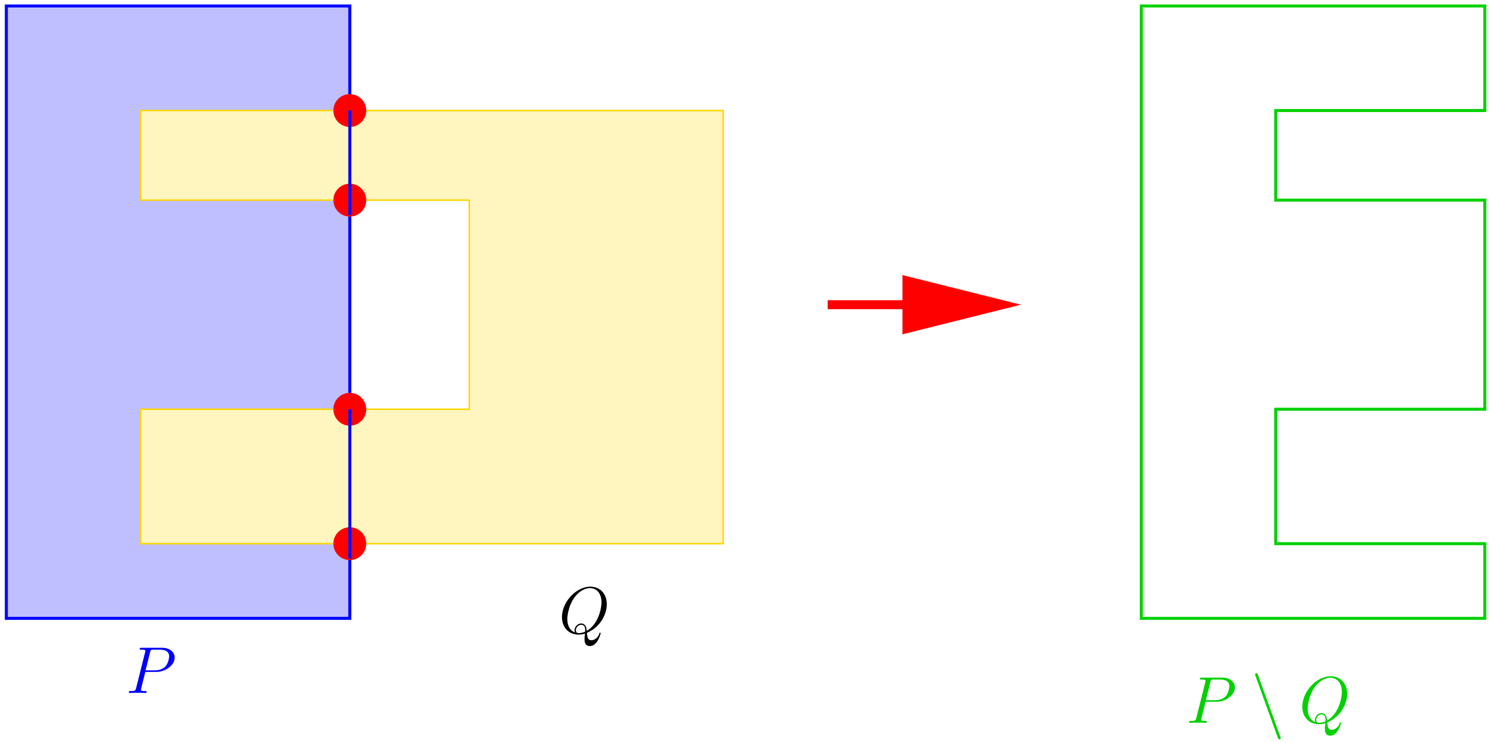
- Geographic Information Systems $\Rightarrow$ map overlay
- More generally: spatial join in databases



ROAD MAP+ RIVER MAP

Motivation

- Computer Aided Design $\Rightarrow$ boolean operations



Preliminary

Intersection of two line segments

Finding the intersection of two line segments

- find the intersection of their two support lines $\Leftarrow$ linear system, two variables, two equations
- check whether it is between the two endpoints of each line segment. If not, the intersection is empty
- degenerate case: same support line. Intersection may be a line segment

$O(1)$ time

Checking whether two line segments intersect without computing the intersection point

$\Leftarrow 4\ CCW(.)$ tests. (see next tutorial)

First approach

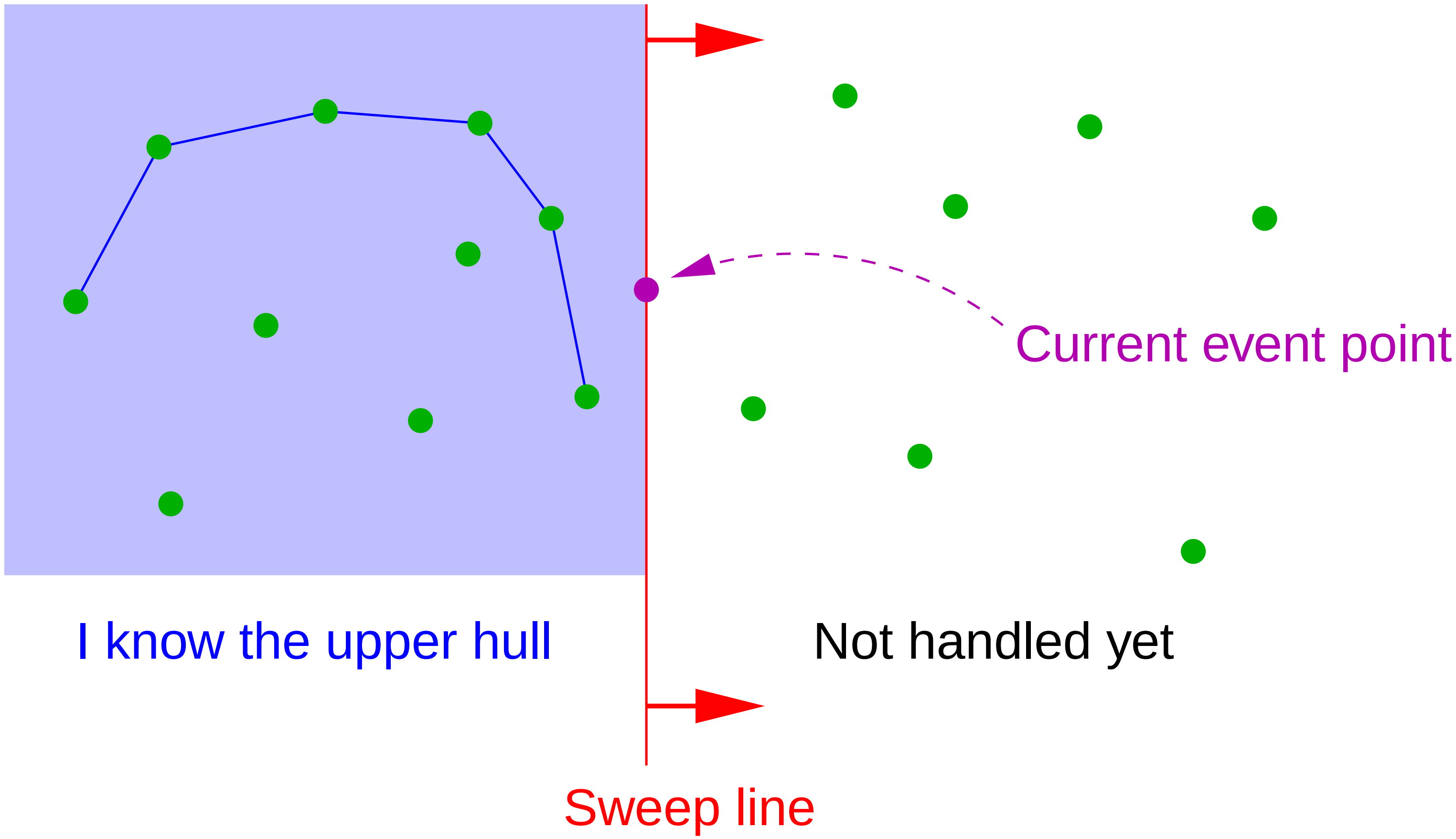
- Brute force algorithm: check all pairs of segments for intersection.
- Running time:

$$\binom{n}{2} \cdot \theta(1) = \theta(n^2)$$

- Can we do better?
  - intersection detection $\Rightarrow$ maybe
  - intersection reporting
    $\Rightarrow$ if all pairs intersect there are $\Omega(n^2)$ intersections
    $\Rightarrow$ our algorithm is optimal (see later, however ...)

Plane sweep algorithms

- intuition: a vertical line sweeps the plane from left to right and draws the picture
- example: last week's convex hull algorithm



Plane sweep algorithms

- the sweep line moves from left to right and stops at *event points*
  - convex hull: the event points are just the input points
  - sometimes they are not known from the start (see intersection reporting)
- we maintain invariants
  - convex hull: I know the upper hull of the points to the left of the sweep line
- at each event, restore the invariants

Intersection Detection

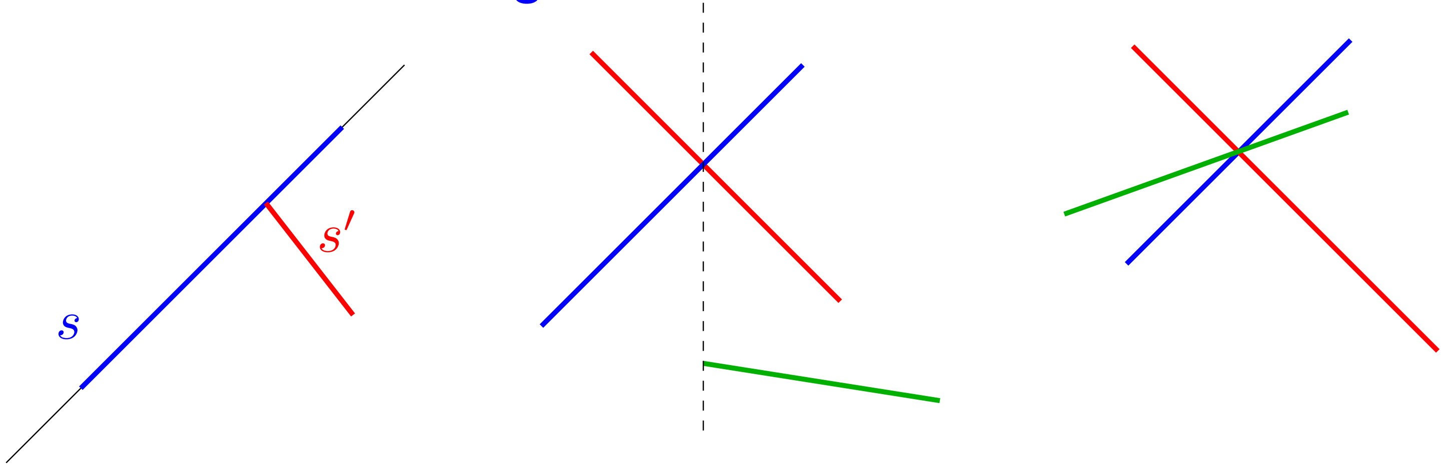
General position assumptions

- no three endpoints are collinear

Let E be the set of the endpoints. Let I be the set of intersection points.

- no two points in $E \cup I$ have same x–coordinate
- no three segments intersect at the same point

Degenerate Cases

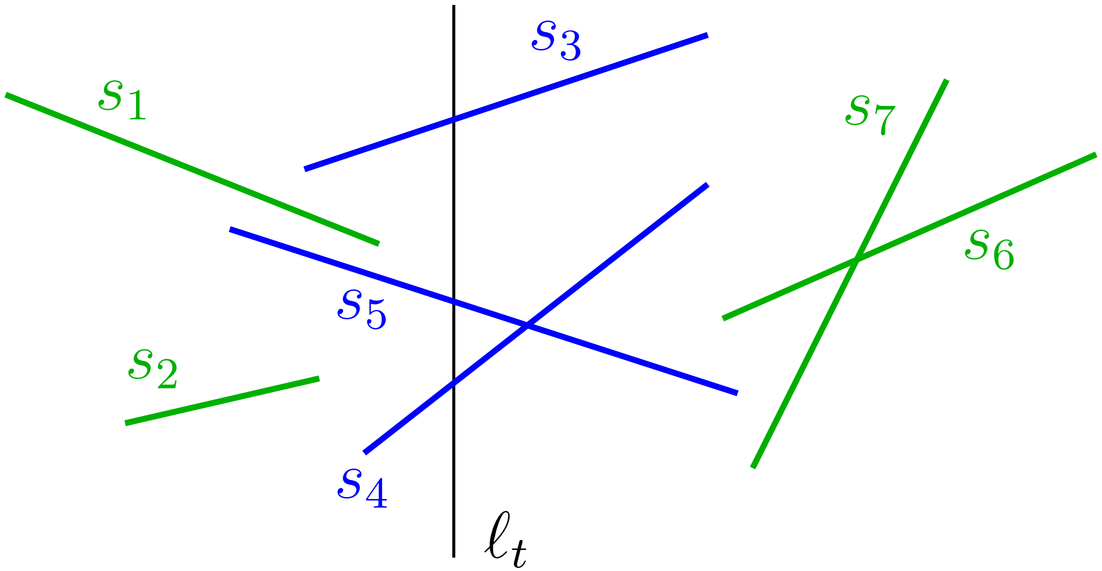


3 collinear endpoints   same x-coordinate   multiple intersection

Intersection Detection

Plane sweep algorithm

- let ℓ_t be the vertical line with equation $x = t$
- S_t is the sequence of the segments that intersect ℓ_t, in vertical order of their intersection with ℓ_t

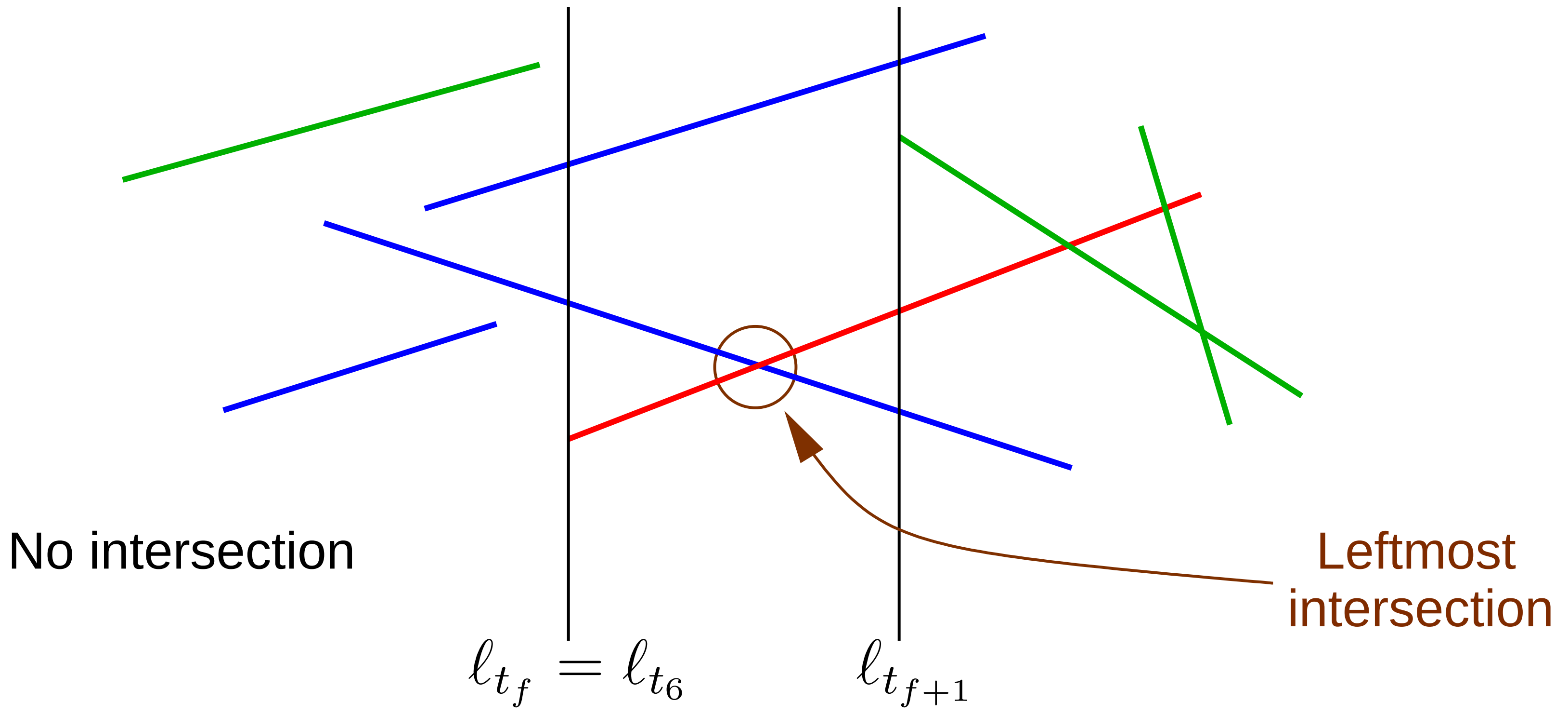


$$S_t = (s_4, s_5, s_3)$$

Intersection Detection

- Idea: maintain S_t while ℓ_t moves from left to right until an intersection is found.
- invariants:
  - there is no intersection to the left of ℓ_t
  - we know S_t
- let $t_1 < t_2 < \ldots < t_{2n}$ be the x–coordinates of the endpoints
- ℓ_t stops when it reaches $t = t_i$ for some index i
- let f be the last index the sweep–line ℓ_{t_i} will reach
- in other words, let f be the largest index such that there is no intersection point with abscissa smaller than t_f

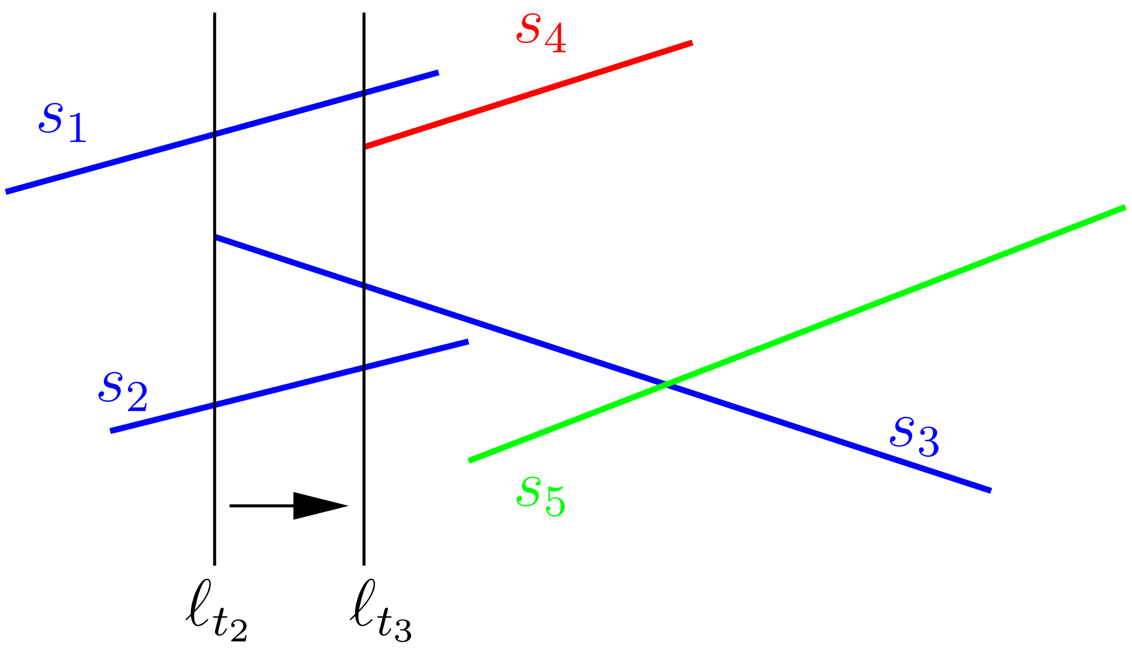
Example



- knowing S_{t_i} for some $i < f$, we can easily find $S_{t_{i+1}}$ (see next two slides)

First case: left endpoint

If t_{i+1} corresponds to the left endpoint of s, insert s into S_{t_i} in order to obtain $S_{t_{i+1}}$.

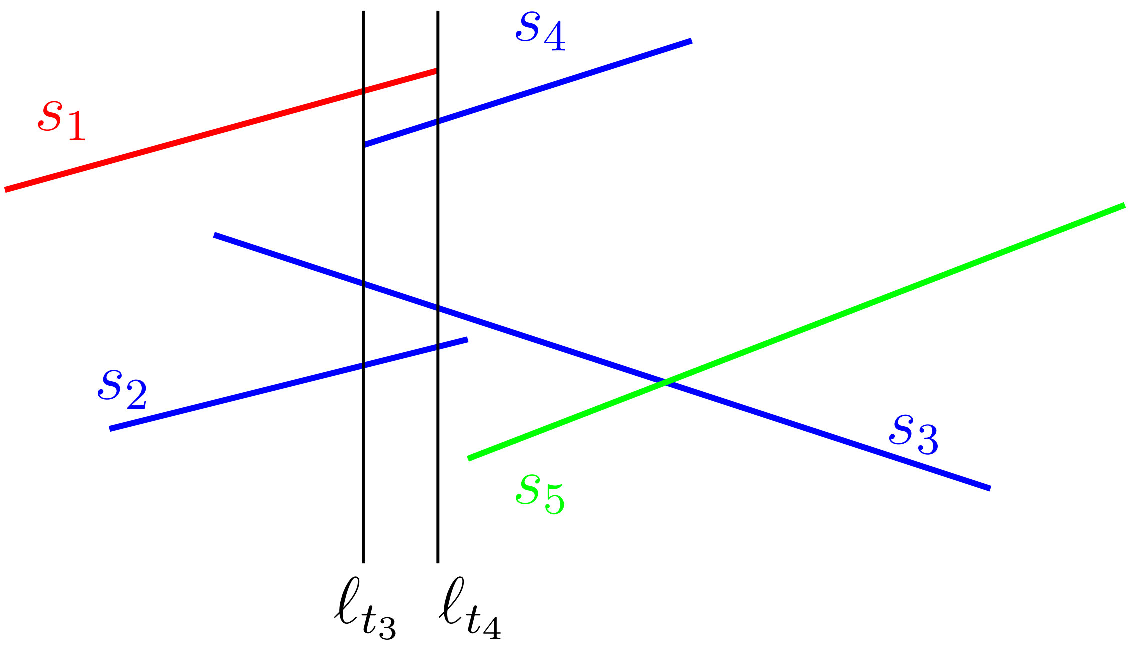


$$S_{t_2} = (s_2, s_3, s_1)$$

$$S_{t_3} = (s_2, s_3, s_4, s_1)$$

Second case: right endpoint

Delete the corresponding segment in order to obtain $S_{t_{i+1}}$.



$$S_{t_3} = (s_2, s_3, s_4, s_1)$$

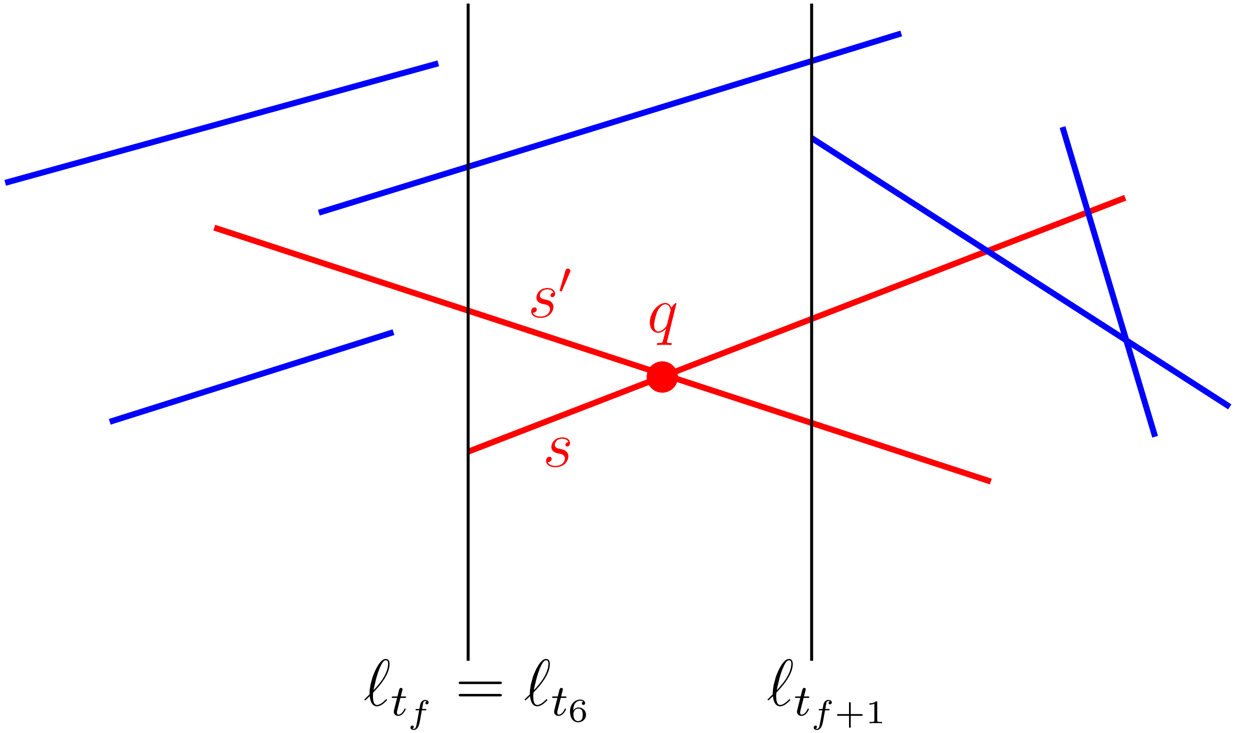
$$S_{t_4} = (s_2, s_3, s_4)$$

Data Structure

- we maintain S_t in a balanced binary search tree
- for $i < f$, we obtain $S_{t_{i+1}}$ from S_{t_i} by performing an insertion or a deletion
- each takes $O(\log n)$ time
- we did not check intersections. How to do it?

A geometric observation

- Let $q = s \cap s'$ be the leftmost intersection point.



- s and s' are adjacent in S_{t_f} (proof next slide)

Proof by contradiction

- assume that $S_{t_f} = (\dots s \dots s'' \dots s' \dots)$
- the right endpoint of s'' cannot be to the left of q, by definition of t_f
- if q is below s'' then s'' intersect s' to the left of q, which contradicts the fact that q is the leftmost intersection
- similarly, q cannot be above s''
- we reached a contradiction

Checking intersection

- we store S_t in a balanced binary search tree $\mathcal{T}$, the order is the vertical order along ℓ_t
- given a segment in $\mathcal{T}$, we can find in $O(\log n)$ time the next and previous segment in vertical order
- when deleting a segment in $\mathcal{T}$, two segments s and s' become adjacent. We can find them in $O(\log n)$ time and check if they intersect
- when inserting a segment s_i in $\mathcal{T}$, it becomes adjacent to two segments s and s'. Check if $s_i \cap s \neq \emptyset$ and if $s_i \cap s' \neq \emptyset$
- in any case, if we find an intersection, we are done

Pseudo–code

Algorithm *DetectIntersection*(S)

$(e_1, e_2 \ldots e_{2n}) \leftarrow$endpoints, ordered w.r.t. x

$\mathcal{T} \leftarrow$empty balanced BST

for $i = 1$ to $2n$

  if e_i is left endpoint of some $s \in S$

    then insert s into $\mathcal{T}$

      if s intersects $next(s)$ or $prev(s)$

        then return TRUE

  if e_i is right endpoint of some $s \in S$

    then delete s from $\mathcal{T}$

      if $next(s)$ intersects $prev(s)$

        then return TRUE

return FALSE

Analysis

- line 1: $\theta(n \log n)$ time by mergesort
- line 2: $O(1)$ time
- line 5 and 6: $O(\log n)$ time
- $\Rightarrow$ loop 3–11: $O(n \log n)$ time
- $\Rightarrow$ our algorithm runs in $\theta(n \log n)$ time

Lower bound

- Element Uniqueness: given a set of n real numbers, are they distinct?
- takes $\Omega(n \log n)$ time in the algebraic decision tree model
  - we can only evaluate polynomials
  - we cannot use the floor function for instance
  - more details in the book by Preparata and Shamos
- it is a simpler version, in one dimension, of our problem
- thus our algorithm is optimal in this computation model

Intersection reporting

Intersection reporting

- brute force: $\theta(n^2)$ time which is optimal in the worst case
- if there is ≤ 1 intersection, the detection algorithm does it in $O(n \log n)$ time which is better
- $\Rightarrow$ we need to look at the running time in a different way

Output sensitive algorithm

- let $k =$ #intersecting pairs
- here $k =$ # intersection points since we assume general position
- k is the *output size*
- sweep line algorithm reports intersections in $O((n+k)\log n)$ time (see later)
- this is an *output sensitive* algorithm
- it is faster for smaller output
- $\Omega(n+k)$ is a lower bound
- $\Rightarrow$ nearly optimal (within an $O(\log n)$ factor)

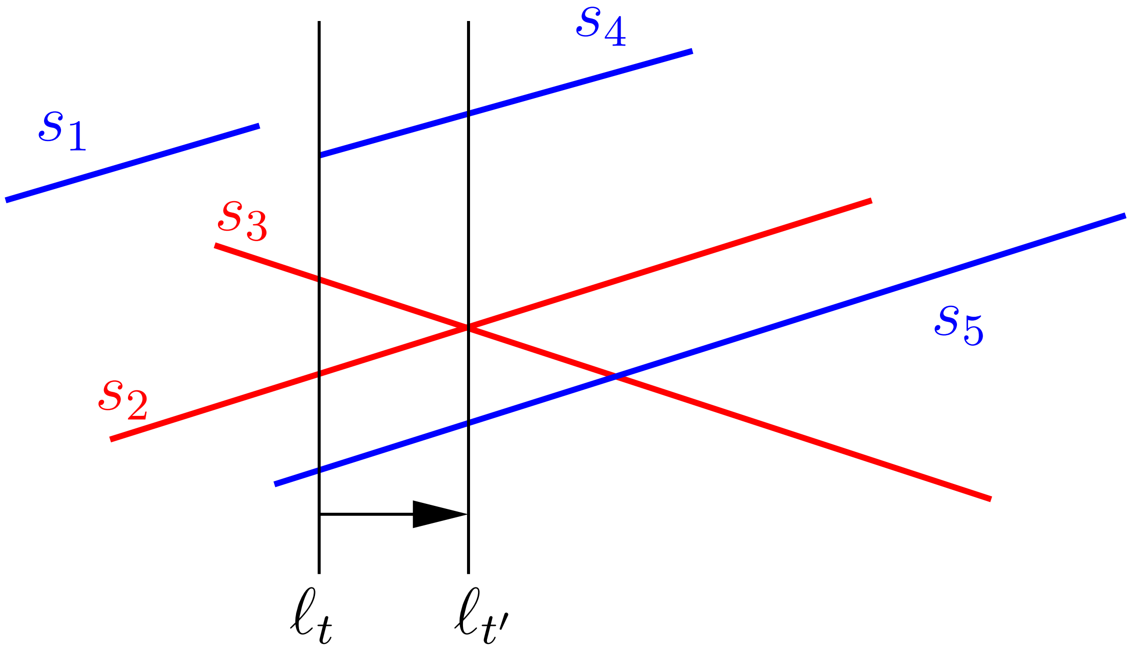
Algorithm

- similar to intersection detection
- two kinds of event points: endpoints and intersection points
- we do not know intersection points in advance
  $\Rightarrow$we cannot sort them all in advance
  $\Rightarrow$we will use an *event queue* $\mathcal{Q}$
- $\mathcal{Q}$ contains event points
- ordered according to x–coordinates
- implementation: a min–heap
- we can insert an event point in $O(\log n)$ time
- we can dequeue the event point with smallest x–coordinate in $O(\log n)$ time.

Algorithm

- initially, $\mathcal{Q}$ contains all the endpoints
- the sweep line moves from left to right
- it stops at each event point of $\mathcal{Q}$, when it does we dequeue the corresponding event point.
- each time we find an intersection, we insert it into $\mathcal{Q}$
- when we reach an intersection point, we swap the corresponding segments in $\mathcal{T}$ and check for intersection the newly adjacent segments.

Intersection event



$$S_t = (s_5, s_2, s_3, s_4)$$

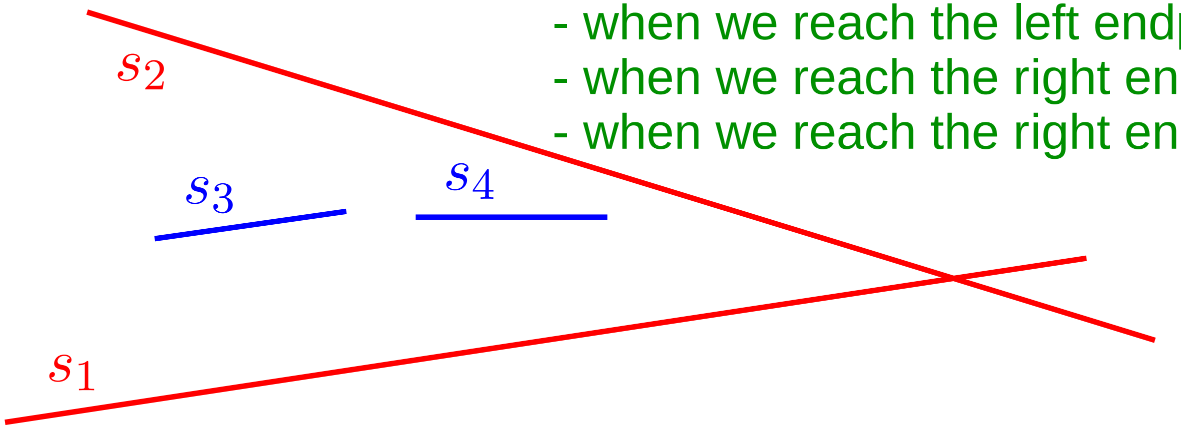
$$S_{t'} = (s_5, s_3, s_2, s_4)$$

Check $s_4 \cap s_2$
and $s_3 \cap s_5$

At time t, the event queue $\mathcal{Q}$ contains all the endpoints with abscissa larger than t and the intersection point $s_2 \cap s_3$. At time t', we insert $s_3 \cap s_5$.

Analysis

- we insert new events and extract the next event in $O(\log n)$ time
- problem: an intersection point may be inserted several times into $\mathcal{Q}$



$s_1 \cap s_2$ is inserted three times.
- when we reach the left endpoint of s_2
- when we reach the right endpoint of s_3
- when we reach the right endpoint of s_4

- when we reach an event point that is present several times in $\mathcal{Q}$, we just extract it repeatedly until a new event is found

Analysis

- some intersection points are inserted several times into $\mathcal{Q}$. How many times does it happen?
- at most twice at each event $\Rightarrow$ at most $4n + 2k$ times
- it follows that we insert $O(n + k)$ events into $\mathcal{Q}$
- so the algorithm runs in $O((n + k) \log(n + k))$ time
- $\log(n + k) < \log(n + n^2) = O(\log n)$
- the running time is $O((n + k) \log n)$

Space complexity

- in the worst case $\mathcal{Q}$ contains $\Theta(n+k)$ points
- we say that this algorithm requires $\Theta(n+k)$ *space*
- can we improve this?
- yes, at time t only keep intersections between segments that are adjacent in S_t
- then the algorithm requires only $O(n)$ space, which is optimal

Conclusion

- new computing paradigm: Plane Sweep
- idea: a sweep line moves from left to right and draws the picture
- it stops at a finite set of event points, where the data structure is updated
- the event points are usually stored in a priority queue
- can be used for various algorithms and applications
- for instance, map overlay: compute a full description of the map