Computational Geometry

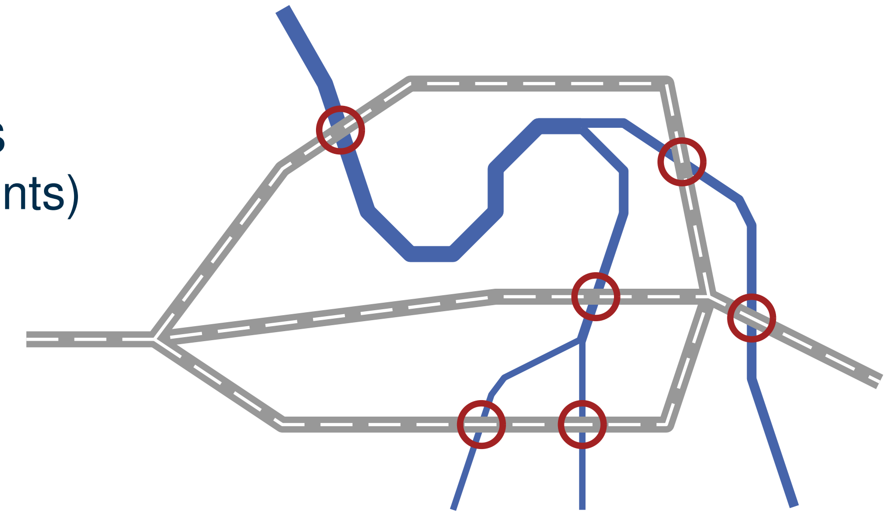
Line Segment Intersection

Thomas Bläsius

Line Segment Intersection: Motivation

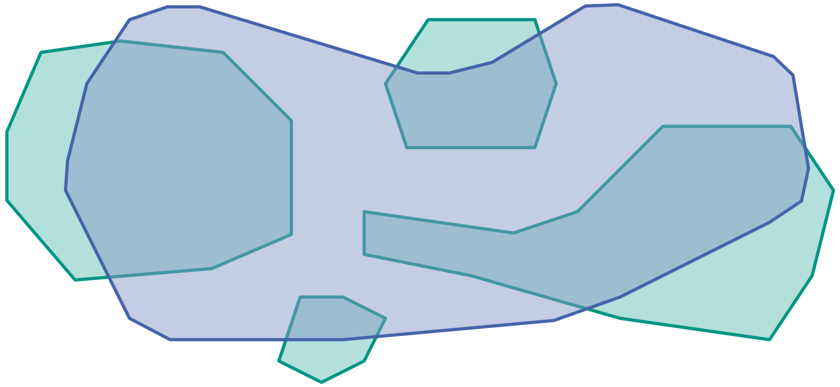
Where Are Bridges?

- **given:** roads and rivers
  (each as sets of line segments)
- **goal:** find all bridges



Forests With A Lot Of Rainfall

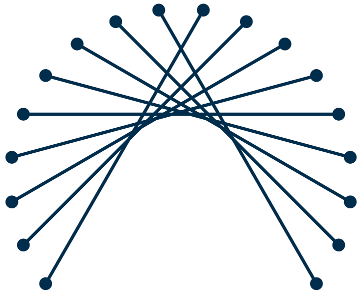
- **given:** forests and regions with more than 1500mm rainfall
  (each as polygons)
- **goal:** compute the intersection of both



Problem: Line Segment Intersection
Given n line segments, compute all pairwise intersections.

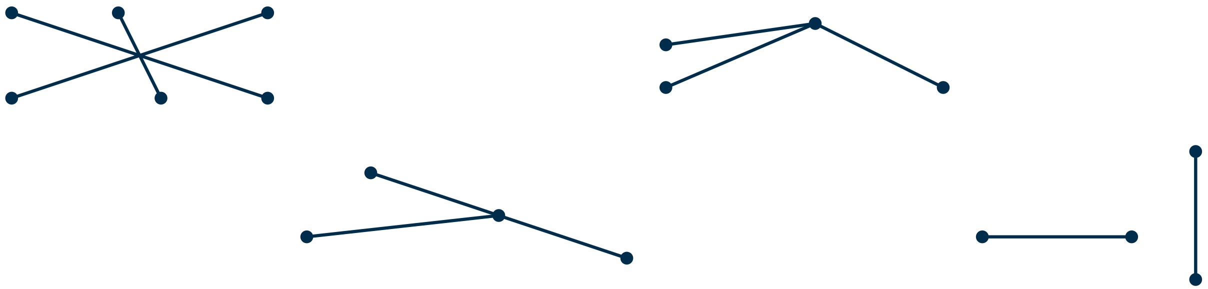
Some First Observations

Problem: Line Segment Intersection
Given n line segments, compute all pairwise intersections.



Assumption: General Position

- at most two segments intersect in one point
- no end point on a different edge
- no shared endpoints
- no horizontal or vertical segments



Trivial Algorithm

- check each pair for intersection $\rightarrow O(n^2)$
- can in general not be improved

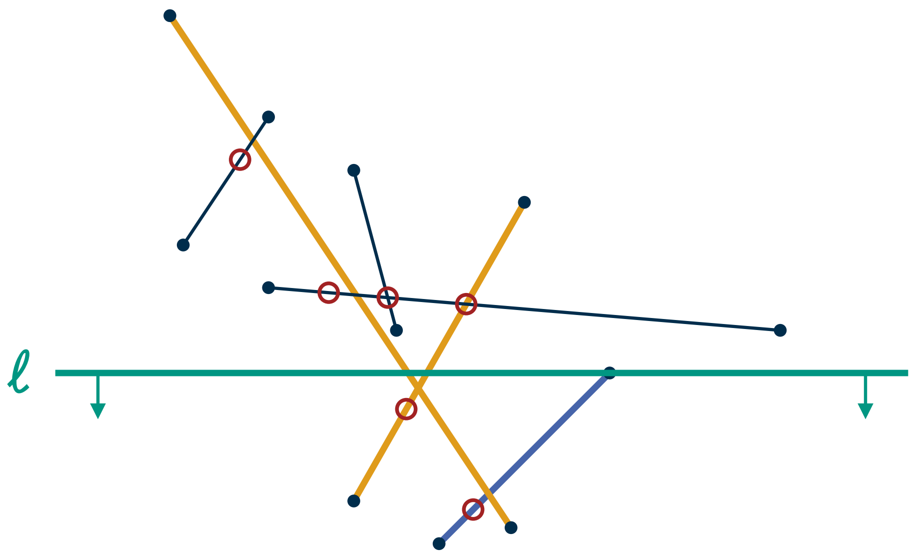
Potential Improvement

- few intersections $\Rightarrow$ better running time
- running time depends on output size
  $\rightarrow$ output sensitive algorithm

A Simple Sweep-Line Algorithm

Idea

- don't compare segments if one lies fully above the other
- move horizontal line ℓ top $\rightarrow$ bottom
- new segment $\rightarrow$ check for intersection with segments intersecting ℓ



Sweep-Line: More Formally

- sweep-line status
  - current status of the sweep line
  - here: set S_ℓ of segments that intersect ℓ
- event queue
  - future positions of the sweep line where something interesting happens (typically status changes)
  - here: start and end points of segments (vertices)
- event handler
  - endpoint of segment s $\rightarrow$ set $S_\ell = S_\ell - s$
  - start point of s $\rightarrow$ check if s intersects segments in S_ℓ and set $S_\ell = S_\ell + s$

A Simple Sweep-Line Algorithm

function FINDINTERSECTIONS(S)
Input: set of line segments S
Output: all intersections (with segment pairs)

```
Q = vertices sorted top to bottom          // event queue
S_ℓ = empty list                            // sweep-line status
while Q ≠ ∅
    p = min{Q} and Q = Q − p
    HandleEventPoint(p)                     // event handler
```

function HANDLEEVENTPOINT(p)

```
s = line segment with vertex p
if s ∈ S_ℓ                                  // segment ends
    S_ℓ = S_ℓ − s
else                                        // segment starts
    for all s' ∈ S_ℓ
        if s ∩ s' ≠ ∅ output (s ∩ s', s, s')
    S_ℓ = S_ℓ + s
```

Problem: slow ($O(n^2)$), if $|S_\ell|$ is usually large

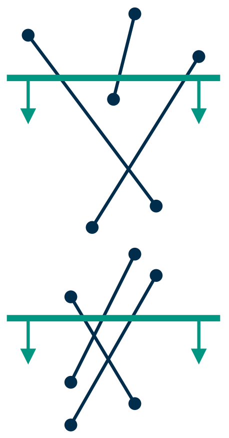
Observation: intersecting segments are at some point next to each other on the sweep line

Solution: only compare adjacent segments (**adjacent:** next to each other on the current sweep line)

Only Checking Adjacent Segments

What Changes?

- sweep-line status: segments crossing ℓ, ordered from left to right
- event handler
  - segment starts: check only with adjacent segments
  - segment starts: insert segment into status $\rightarrow$ maintain left-to-right ordering
  - segment ends: check for newly adjacent segments
  - intersection: order in sweep-line status changes $\rightarrow$ check newly adjacent
  - intersection found: insert intersection into event queue



Which Data Structure?

- sweep-line status: insert, delete, find successor / predecessor
- event queue: insert, extract minimum, search

} search tree: $O(\log n)$
(e.g., (a, b)-tree, red-black tree)

$\uparrow$ one way to handle finding the same intersection multiple times

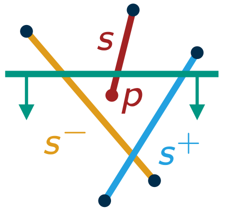
Improved Sweep-Line Algorithm

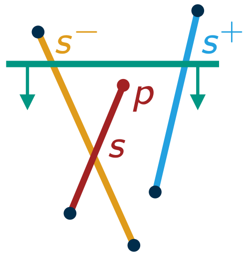
function FINDINTERSECTIONS(S)
Input: set of line segments S
Output: all intersections (with segment pairs)

- $Q =$ empty queue // event queue
- **for** $pq \in S$
  - $Q = Q + p + q$
- $T =$ search tree // status
- **while** $Q \neq \emptyset$
  - $p = \min\{Q\}$ and $Q = Q - p$
  - HANDLEEVENT(p) // event handler

function HANDLEEVENT(p)

- **if** p is vertex of a segment
  - HANDLEVERTEX(p)
- **else**
  - HANDLEINTERSECTION(p)

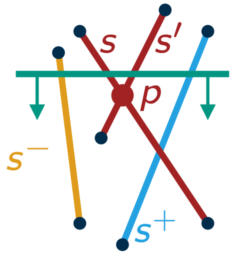




function HANDLEVERTEX(p)

- $s =$ segment with vertex p
- $s^- =$ predecessor of s in T
- $s^+ =$ successor of s in T
- **if** $s \in T$ // segment ends
  - CHECKINTERSECTION(s^-, s^+, p)
  - $T = T - s$
- **else** // segment starts
  - $T = T + s$
  - CHECKINTERSECTION(s^-, s, p)
  - CHECKINTERSECTION(s^+, s, p)

Running time?



function HANDLEINTERSECTION(p)

- $s, s' =$ segments intersecting in p
- **output** (p, s, s') ($s < s'$ in T)
- swap s and s' in T ($s' < s$)
- $s^- =$ predecessor of s' in T
- $s^+ =$ successor of s in T
- CHECKINTERSECTION(s^-, s', p)
- CHECKINTERSECTION(s^+, s, p)

function CHECKINTERSECTION(s^-, s^+, p)

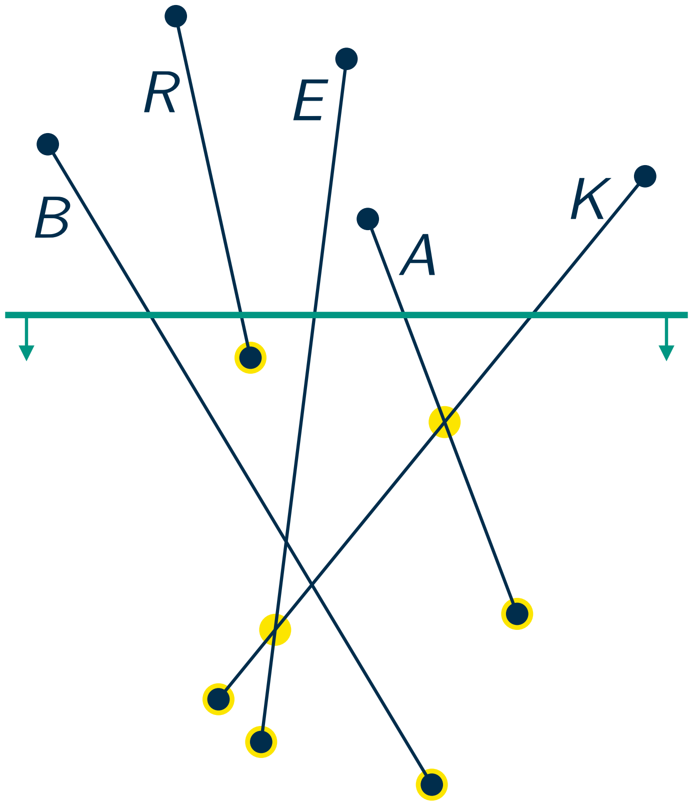
- $q = s^- \cap s^+$
- **if** $q \neq \emptyset$ **and** $p_y < q_y$ **and** $q \notin Q$
  - $Q = Q + q$

Improved Sweep-Line Algorithm

Theorem **assumption:** general position

The k intersections of n line segments can be computed in $O((n+k)\log n)$ time.

How Many Events Are In The Event Queue?



7

Do We Need General Position?

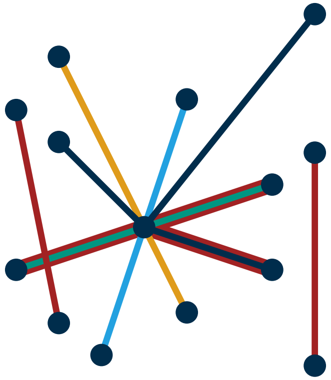
Theorem **assumption:** general position

The k intersections of n line segments can be computed in $O((n + k) \log n)$ time.

Problem: Multiple Events At The Same Point

- three things can happen at an event point:
  - one or multiple segments start
  - one or multiple segments end
  - segments intersect

  } plan: handle of them together



- let $\mathtt{start}(p)$, $\mathtt{end}(p)$, and $\mathtt{int}(p)$ be the sets of segments that start at, end at, and intersect p
- updating the sweep-line status T:
  - remove $\mathtt{end}(p) \cup \mathtt{int}(p)$
  - insert $\mathtt{int}(p) \cup \mathtt{start}(p)$ (using the order slightly below p)
- check newly adjacent segments afterwards

What happens to horizontal edges?

Sweep-Line Algo – No General Position Assumption

HANDLEEVENT(p)

get / find start(p), end(p), int(p) // end(p), int(p) can be found in T, start(p) has to be saved with p

if $|\text{start}(p) \cup \text{end}(p) \cup \text{int}(p)| > 1$

  output p (with $\text{start}(p) \cup \text{end}(p) \cup \text{int}(p)$)

remove $\text{end}(p) \cup \text{int}(p)$ from T

insert $\text{int}(p) \cup \text{start}(p)$ into T

if $\text{start}(p) \cup \text{int}(p) = \emptyset$

  $s^- =$ predecessor of p in T

  $s^+ =$ successor of p in T

  CHECKINTERSECTION(s^-, s^+, p)

else

  $s^- =$ left-most segment in $\text{start}(p) \cup \text{int}(p)$

  $s^+ =$ right-most segment in $\text{start}(p) \cup \text{int}(p)$

  $\hat{s}^- =$ predecessor of s^- in T

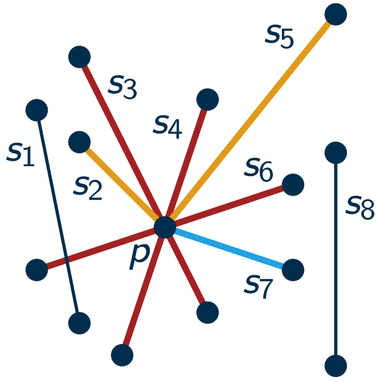
  $\hat{s}^+ =$ successor of s^+ in T

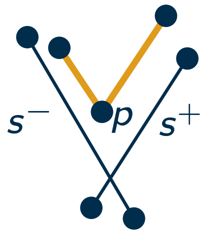
  CHECKINTERSECTION($s^-, \hat{s}^-, p$)

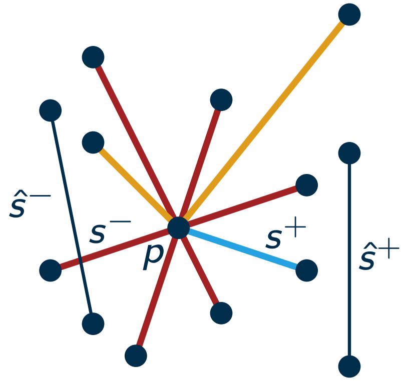
  CHECKINTERSECTION($s^+, \hat{s}^+, p$)

T: s_1 s_2 s_3 s_4 s_5 s_6 s_8 $\longrightarrow$ s_1 s_8

T: s_1 s_8 $\longrightarrow$ s_1 s_6 s_4 s_3 s_7 s_8







How can this be done using a search tree?

→ comparator depending on position of the sweep line

What is left to show?

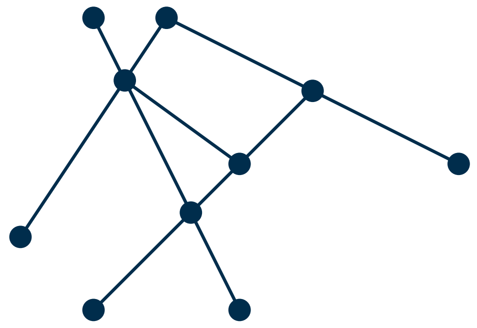
Theorem
The k intersections of n line segments can be computed in $O((n + k) \log n)$ time.

Running Time Analysis

Theorem
The k intersections of n line segments can be computed in $O((n+k)\log n)$ time.

Proof (Running Time)

- initialization: insert $2n$ event points into queue $O(n \log n)$
- queue operations at event: extract minimum, insert at most 2 new events $O((n+k)\log n)$
- operations on the sweep-line status
  - $m(p)$ intersecting segments at event point $p \Rightarrow \Theta(m(p))$ operations
  - cost over all event points: $m \log n$ mit $m = \sum_p m(p)$ $O(m \log n)$
- segments form plane graph $G = (V, E)$
- $|V| \leq 2n + k$ und $2|E| = m$
- for planar graphs: $|E| \leq 3|V| - 6$
  $\Rightarrow m \in O(n+k)$



Does $m \in O(n+k)$ hold?

Memory Consumption

Why Should We Care?

- memory is a more critical resource than time
- you can wait for an algorithm with running time $O(n^2)$
- $O(n^2)$ memory consumption is usually not ok

How Large Can The Sweep-Line Status?

- at most n Segments $\rightarrow O(n)$

How Large Can The Event Queue Be?

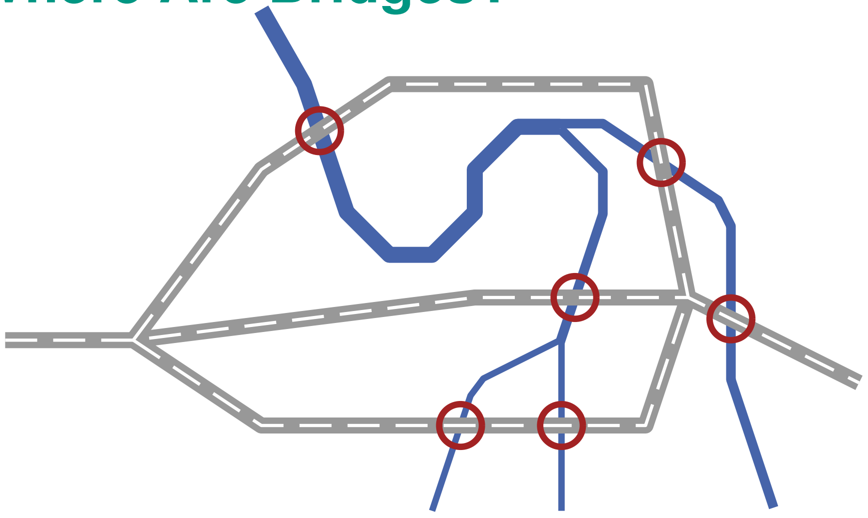
- obvious bound: $n + k$
- intersections may be in the queue for quite some time before they are processed
- option: only keep intersections in the queue corresponding to adjacent segments $\rightarrow O(n)$

Back To Our Motivation

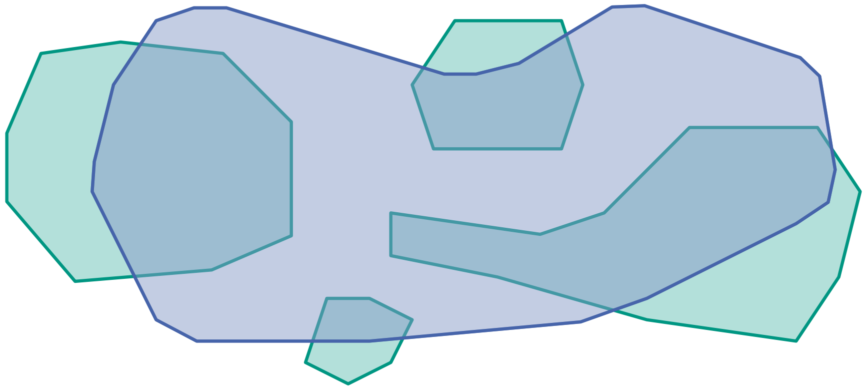
Did We Reach Our Goal?
- we can find bridges
- we cannot yet compute the intersection of polygons

In The Following
- data structure that helps with computing the intersection
- actually using it: exercise sheet

Where Are Bridges?



Forests With A Lot Of Rainfall



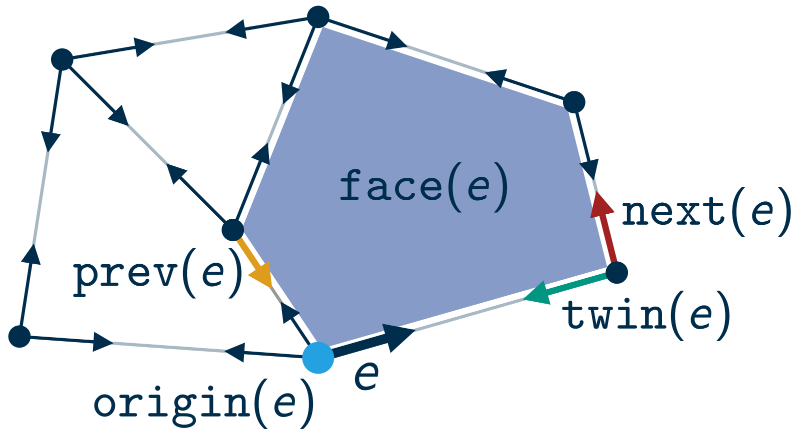
Doubly-Connected Edge List

Doubly Connected Edge List

- each edge has two incident vertices $\to$ store each edge twice

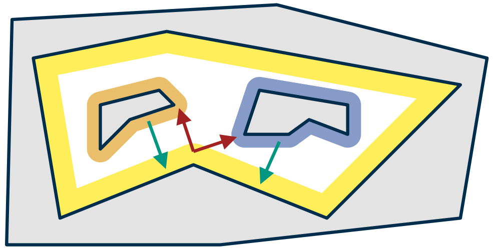
For Every "Half Edge" e

- corresponding vertex: $\texttt{origin}(e)$
- half edge at the opposite vertex: $\texttt{twin}(e)$
- incident face (left): $\texttt{face}(e)$
- next/previous edge along this face: $\texttt{next}(e)$, $\texttt{prev}(e)$
- for every node v and face f: one incident edge $\texttt{edge}(v)$ / $\texttt{edge}(f)$



Derived Operations & Notes

- clockwise successor of e around $\texttt{origin}(e)$: $\texttt{next}(\texttt{twin}(e))$
- clockwise predecessor of e around $\texttt{origin}(e)$: $\texttt{twin}(\texttt{prev}(e))$
- you should adapt this to your use case



adaptation for multiple connected components

Wrap-Up

What Have We Learned Today?

- output sensitive algorithm for segment intersection: $O((n + k) \log n)$
- sweep line technique: discretization of continuous geometry using a finite set of events
- like last week: initially ignoring special cases helps
- doubly-connected edge list

What Else Is There?

- extension to map overlay and Boolean operations on polygons
- lower bound: $\Omega(n \log n + k)$
- can be solved in $O(n \log n + k)$ time with $O(n)$ space
- extensions to the sweep-line approach
  - the sweep line might move differently (e.g., rotate)
  - the sweep line does not need to be a line