Computational Geometry

Line-Segment Intersection
or
Map Overlay

Lecture #2

Dr. Philipp Kindermann

Winter Semester 2018
Map Overlay in Geographic Information Systems (GIS)
Map Overlay in Geographic Information Systems (GIS)
Map Overlay in Geographic Information Systems (GIS)
Map Overlay in Geographic Information Systems (GIS)

Here:

= bridge
Line-Segment Intersection

Definition:
Line-Segment Intersection

**Definition:** Is an intersection?
Line-Segment Intersection

Definition: Is \( \text{an intersection?} \)

Answer: Depends...
Line-Segment Intersection

Definition: Is an intersection?

Answer: Depends... 

Problem: Given a set $S$ of $n$ closed non-overlapping line segments in the plane, compute...
Line-Segment Intersection

**Definition:** Is an intersection?

**Answer:** Depends...

**Problem:** Given a set $S$ of $n$ closed non-overlapping line segments in the plane, compute...
- all points where at least two segments intersect and
- for each such point report all segments that contain it.
Line-Segment Intersection

**Definition:** Is there an intersection?

**Answer:** Depends…

**Problem:** Given a set $S$ of $n$ closed non-overlapping line segments in the plane, compute…
- all points where at least two segments intersect and
- for each such point report all segments that contain it.

Yes!
Line-Segment Intersection

Definition: Is an intersection?

Answer: Depends. . .

Problem: Given a set $S$ of $n$ closed non-overlapping line segments in the plane, compute . . .

- all points where at least two segments intersect and
- for each such point report all segments that contain it.

Task: Discuss with your neighbor: how would you do it?
Example
Example

Brute Force?

$O(n^2)$ ... can we do better?
Example

Brute Force?

$O(n^2)$ ... can we do better?

Idea:
Process segments top-to-bottom using a "sweep line".
Sweep-Line Algorithm
Sweep-Line Algorithm

Which active segments should be compared?
Sweep-Line Algorithm

Which active segments should be compared?
Which active segments should be compared?
Sweep-Line Algorithm

Which active segments should be compared?
Sweep-Line Algorithm

Which active segments should be compared?
Sweep-Line Algorithm

Which active segments should be compared?
Which active segments should be compared?
Sweep-Line Algorithm

Which active segments should be compared?
Which active segments should be compared?
Which active segments should be compared?
Sweep-Line Algorithm

Which active segments should be compared?
Sweep-Line Algorithm

Which active segments should be compared?
Sweep-Line Algorithm

Which active segments should be compared?
Which active segments should be compared?
Sweep-Line Algorithm

Which active segments should be compared?
Which active segments should be compared?
Sweep-Line Algorithm

Which active segments should be compared?
Which active segments should be compared?
Sweep-Line Algorithm

Which active segments should be compared?
Which active segments should be compared?
Which active segments should be compared?
Data Structures

1) event (-point) queue $Q$

2) (sweep-line) status $\mathcal{T}$
1) event (-point) queue $Q$

$p \prec q \iff \text{def.}$

2) (sweep-line) status $T$
Data Structures

1) event (-point) queue $Q$

$p \prec q \iff \text{def. } y_p > y_q$

2) (sweep-line) status $T$
Data Structures

1) event (-point) queue \( \mathcal{Q} \)

\[ p \prec q \iff_{\text{def.}} y_p > y_q \]

2) (sweep-line) status \( \mathcal{T} \)
Data Structures

1) event (-point) queue $Q$ 

\[ p \prec q \iff \text{def. } y_p > y_q \text{ or } (y_p = y_q \text{ and } x_p < x_q) \]

2) (sweep-line) status $\mathcal{T}$
1) event (-point) queue $\mathcal{Q}$

$\forall p, q \in \mathcal{Q}$

\[ p \prec q \iff_{\text{def.}} y_p > y_q \quad \text{or} \quad (y_p = y_q \text{ and } x_p < x_q) \]

2) (sweep-line) status $\mathcal{T}$
1) event (-point) queue $Q$

$p \prec q \iff \text{def. } y_p > y_q \quad \text{or} \quad (y_p = y_q \text{ and } x_p < x_q)$

2) (sweep-line) status $T$

\[\ell \quad p \quad q\]
Data Structures

1) event (-point) queue $Q$

\[ p \prec q \iff \text{def. } y_p > y_q \quad \text{or} \quad (y_p = y_q \text{ and } x_p < x_q) \]

Store event pts in balanced binary search tree acc. to $\prec$

2) (sweep-line) status $T$

\[ \ell \]

\[ p \quad q \]

\[ y_p > y_q \]
Data Structures

1) event (-point) queue \( Q \)

\[ p \prec q \iff y_p > y_q \quad \text{or} \quad (y_p = y_q \text{ and } x_p < x_q) \]

Store event pts in balanced binary search tree acc. to \( \prec \)

\[ \Rightarrow \text{nextEvent()} \text{ and del/insEvent()} \text{ take } O(\log |Q|) \text{ time} \]

2) (sweep-line) status \( T \)
Data Structures

1) event (-point) queue $\mathcal{Q}$

\[ p \prec q \iff_{\text{def.}} y_p > y_q \quad \text{or} \quad (y_p = y_q \text{ and } x_p < x_q) \]

Store event pts in *balanced binary search tree* acc. to $\prec$

$\Rightarrow$ nextEvent() and del/insEvent() take $O(\log |\mathcal{Q}|)$ time

2) (sweep-line) status $\mathcal{T}$
Data Structures

1) event (-point) queue $\mathcal{Q}$

$p \prec q \iff y_p > y_q \quad \text{or} \quad (y_p = y_q \text{ and } x_p < x_q)$

Store event pts in balanced binary search tree acc. to $\prec$

$\Rightarrow$ nextEvent() and del/insEvent() take $O(\log |\mathcal{Q}|)$ time

2) (sweep-line) status $\mathcal{T}$

Store the segments intersected by $\ell$ in left-to-right order.
Data Structures

1) event (-point) queue $\mathcal{Q}$

$p ≺ q \iff \text{def. } y_p > y_q \quad \text{or} \quad (y_p = y_q \text{ and } x_p < x_q)$

Store event pts in \textit{balanced binary search tree} acc. to $≺$

$⇒$ nextEvent() and del/insEvent() take $O(\log |\mathcal{Q}|)$ time

2) (sweep-line) status $\mathcal{T}$

Store the segments intersected by $\ell$ in left-to-right order.

How?
Data Structures

1) event (-point) queue \( Q \)

\[ p \prec q \iff \text{def. } y_p > y_q \quad \text{or} \quad (y_p = y_q \text{ and } x_p < x_q) \]

Store event pts in balanced binary search tree acc. to \( \prec \)

\[ \Rightarrow \text{nextEvent()} \text{ and del/insEvent()} \text{ take } O(\log |Q|) \text{ time} \]

2) (sweep-line) status \( \mathcal{T} \)

Store the segments intersected by \( \ell \) in left-to-right order.

How? In a balanced binary search tree!
Pseudo-code

findIntersections(S)

Input: set $S$ of $n$ non-overlapping closed line segments

Output: – set $I$ of intersection pts
– for each $p \in I$ every $s \in S$ with $p \in s$
Pseudo-code

findIntersections(S)

Input: set $S$ of $n$ non-overlapping closed line segments

Output: – set $I$ of intersection pts
– for each $p \in I$ every $s \in S$ with $p \in s$

$Q \leftarrow \emptyset; \quad T \leftarrow \langle \text{vertical lines at } x = -\infty \text{ and } x = +\infty \rangle \quad \text{// sentinels}

\text{foreach } s \in S \text{ do}
\quad \text{// initialize event queue } Q
\quad \text{foreach endpoint } p \text{ of } s \text{ do}
\quad \quad \text{if } p \notin Q \text{ then } Q.\text{insert}(p); \quad L(p) = U(p) = \emptyset
\quad \quad \text{if } p \text{ lower endpt of } s \text{ then } L(p).\text{append}(s)
\quad \quad \text{if } p \text{ upper endpt of } s \text{ then } U(p).\text{append}(s)
Pseudo-code

**findIntersections**($S$)

**Input:** set $S$ of $n$ non-overlapping closed line segments

**Output:**
- set $I$ of intersection pts
- for each $p \in I$ every $s \in S$ with $p \in s$

\[
Q \leftarrow \emptyset; \quad T \leftarrow \langle \text{vertical lines at } x = -\infty \text{ and } x = +\infty \rangle \quad \text{// sentinels}
\]

\[
\text{foreach } s \in S \text{ do} \quad \text{// initialize event queue } Q
\]

\[
\quad \text{foreach endpoint } p \text{ of } s \text{ do}
\]

\[
\quad \quad \text{if } p \notin Q \text{ then } Q.\text{insert}(p); \quad L(p) = U(p) = \emptyset
\]

\[
\quad \quad \text{if } p \text{ lower endpt of } s \text{ then } L(p).\text{append}(s)
\]

\[
\quad \quad \text{if } p \text{ upper endpt of } s \text{ then } U(p).\text{append}(s)
\]

\[
\text{while } Q \neq \emptyset \text{ do}
\]

\[
\quad p \leftarrow Q.\text{nextEvent}()
\]

\[
\quad Q.\text{deleteEvent}(p)
\]

\[
\quad \text{handleEvent}(p)
\]
findIntersections($S$)

Input: set $S$ of $n$ non-overlapping closed line segments

Output: – set $I$ of intersection pts
– for each $p \in I$ every $s \in S$ with $p \in s$

$Q \leftarrow \emptyset$; $T \leftarrow \langle$ vertical lines at $x = -\infty$ and $x = +\infty$ $\rangle$ // sentinels

foreach $s \in S$ do // initialize event queue $Q$
  foreach endpoint $p$ of $s$ do
    if $p \notin Q$ then $Q$.insert($p$); $L(p) = U(p) = \emptyset$
    if $p$ lower endpt of $s$ then $L(p)$.append($s$)
    if $p$ upper endpt of $s$ then $U(p)$.append($s$)

while $Q \neq \emptyset$ do
  $p \leftarrow Q$.nextEvent()
  $Q$.deleteEvent($p$)
  handleEvent($p$)

This subroutine does the real work. How would you implement it?
findIntersections($S$)

Input: set $S$ of $n$ non-overlapping closed line segments

Output: – set $I$ of intersection pts
– for each $p \in I$ every $s \in S$ with $p \in s$

$Q \leftarrow \emptyset$; $T \leftarrow \langle \text{vertical lines at } x = -\infty \text{ and } x = +\infty \rangle$  // sentinels

foreach $s \in S$ do
  foreach endpoint $p$ of $s$ do
    if $p \notin Q$ then $Q$.insert($p$); $L(p) = U(p) = \emptyset$
    if $p$ lower endpt of $s$ then $L(p)$.append($s$)
    if $p$ upper endpt of $s$ then $U(p)$.append($s$)

while $Q \neq \emptyset$ do
  $p \leftarrow Q$.nextEvent()
  $Q$.deleteEvent($p$)
  handleEvent($p$)

This subroutine does the real work. How would you implement it?
Handling an Event

handleEvent(event p)

if |U(p) ∪ L(p) ∪ C(p)| > 1 then
  report intersection in p, report segments in U(p) ∪ L(p) ∪ C(p)
  delete L(p) ∪ C(p) from T
  insert U(p) ∪ C(p) into T in their order slightly below ℓ
  if U(p) ∪ C(p) = ∅ then
    bleft / bright = left/right neighbor of p in T
    findNewEvent(bleft, bleft, p)
    findNewEvent(bright, bright, p)
  else
    sleft / sright = leftmost/rightmost segment in U(p) ∪ C(p)
    bleft = left neighbor of sleft in T
    bright = right neighbor of sright in T
    findNewEvent(bleft, sleft, p)
    findNewEvent(bright, sright, p)
Handling an Event

```
handleEvent(event p)
if |U(p) ∪ L(p) ∪ C(p)| > 1 then
  report intersection in p, report segments in U(p) ∪ L(p) ∪ C(p)
```

handleEvent(event p)

if |U(p) ∪ L(p) ∪ C(p)| > 1 then

- report intersection in p, report segments in U(p) ∪ L(p) ∪ C(p)
- delete L(p) ∪ C(p) from T  // consecutive in T!
- insert U(p) ∪ C(p) into T in their order slightly below ℓ
**Handling an Event**

![Diagram of event p with segments C(p), L(p), U(p)]

`handleEvent(event p)`

1. **if** `|U(p) ∪ L(p) ∪ C(p)| > 1` **then**
   - report intersection in p, report segments in `U(p) ∪ L(p) ∪ C(p)`
   - delete `L(p) ∪ C(p)` from `T` // consecutive in `T`!

2. **insert** `U(p) ∪ C(p)` into `T` in their order slightly below `ℓ`

3. **if** `U(p) ∪ C(p) = ∅` **then**
   - else
Handling an Event

\[
\text{handleRequest}(\text{event } p)
\]

\begin{enumerate}
\item \textbf{if } \left| U(p) \cup L(p) \cup C(p) \right| > 1 \textbf{ then}
\item \quad \text{report intersection in } p, \text{ report segments in } U(p) \cup L(p) \cup C(p)
\item \quad \text{delete } L(p) \cup C(p) \text{ from } \mathcal{T} \quad // \quad \text{consecutive in } \mathcal{T}!
\item \quad \text{insert } U(p) \cup C(p) \text{ into } \mathcal{T} \text{ in their order slightly below } \ell
\item \quad \textbf{if } U(p) \cup C(p) = \emptyset \textbf{ then}
\item \qquad b_\text{left}/b_\text{right} = \text{left/right neighbor of } p \text{ in } \mathcal{T}
\item \qquad \text{findNewEvent}(b_\text{left}, b_\text{right}, p)
\item \quad \textbf{else}
\end{enumerate}
Handling an Event

\[ C(p), L(p), U(p) \]

`handleEvent(event p)`

```plaintext
if |U(p) ∪ L(p) ∪ C(p)| > 1 then
  report intersection in p, report segments in U(p) ∪ L(p) ∪ C(p)
delete L(p) ∪ C(p) from \( \mathcal{T} \) // consecutive in \( \mathcal{T} \)
insert U(p) ∪ C(p) into \( \mathcal{T} \) in their order slightly below \( \ell \)
if U(p) ∪ C(p) = ∅ then
  b_{\text{left}}/b_{\text{right}} = \text{left/right neighbor of } p \text{ in } \mathcal{T}
  findNewEvent(b_{\text{left}}, b_{\text{right}}, p)
else
  findNewEvent(s, s', p)
  if s ∩ s' = ∅ then return
  \{x\} = s ∩ s'
  if x below \( \ell \) or to the right of p then
    if x ∉ Q then Q.add(x)
    if x ∈ rel-int(s) then C(x) ← C(x) ∪ \{s\}
    if x ∈ rel-int(s') then
      C(x) ← C(x) ∪ \{s'\}
```

below \( \ell \)

\[ b_{\text{left}}, b_{\text{right}} \]
Handling an Event

```
handleEvent(event p)
if |U(p) ∪ L(p) ∪ C(p)| > 1 then
    report intersection in p, report segments in U(p) ∪ L(p) ∪ C(p)
delete L(p) ∪ C(p) from T // consecutive in T!
    insert U(p) ∪ C(p) into T in their order slightly below ℓ
if U(p) ∪ C(p) = ∅ then
    b_left/b_right = left/right neighbor of p in T
    findNewEvent(b_left, b_right, p)
else
    s_left/s_right = leftmost/rightmost segment in U(p) ∪ C(p)
b_left = left neighbor of s_left in T
b_right = right neighbor of s_right in T
findNewEvent(b_left, s_left, p)
findNewEvent(b_right, s_right, p)
```
Correctness

Lemma. findIntersections() correctly computes all intersection points & the segments that contain them.
Correctness

**Lemma.** findIntersections() correctly computes all intersection points & the segments that contain them.

**Proof.** Let $p$ be an intersection pt.
Correctness

**Lemma.** findIntersections() correctly computes all intersection points & the segments that contain them.

**Proof.** Let \( p \) be an intersection pt. Assume:
- Every int. pt \( q \prec p \) has been computed correctly.
Correctness

Lemma. findIntersections() correctly computes all intersection points & the segments that contain them.

Proof. Let $p$ be an intersection pt. Assume:

- Every int. pt $q \prec p$ has been computed correctly.
- $T$ contains all segments intersecting $\ell$ in left-to-right order.
Correctness

**Lemma.** findIntersections() correctly computes all intersection points & the segments that contain them.

**Proof.** Let $p$ be an intersection pt. Assume (by induction):

- Every int. pt $q \prec p$ has been computed correctly.
- $T$ contains all segments intersecting $\ell$ in left-to-right order.
Correctness

Lemma. findIntersections() correctly computes all intersection points & the segments that contain them.

Proof. Let $p$ be an intersection pt. Assume (by induction):

- Every int. pt $q \prec p$ has been computed correctly.
- $\mathcal{T}$ contains all segments intersecting $\ell$ in left-to-right order.

Case I: $p$ is not an interior pt of a segment.
Correctness

**Lemma.** findIntersections() correctly computes all intersection points & the segments that contain them.

**Proof.** Let \( p \) be an intersection pt. Assume (by induction):

- Every int. pt \( q \prec p \) has been computed correctly.
- \( T \) contains all segments intersecting \( \ell \) in left-to-right order.

**Case I:** \( p \) is not an interior pt of a segment.

\( \Rightarrow \) \( p \) has been inserted in \( Q \) in the beginning.
Correctness

Lemma. findIntersections() correctly computes all intersection points & the segments that contain them.

Proof. Let $p$ be an intersection pt. Assume (by induction):

- Every int. pt $q \prec p$ has been computed correctly.
- $\mathcal{T}$ contains all segments intersecting $\ell$ in left-to-right order.

Case I: $p$ is not an interior pt of a segment.

$\Rightarrow p$ has been inserted in $Q$ in the beginning.

Segm. in $U(p)$ and $L(p)$ are stored with $p$ in the beginning.
Correctness

Lemma. findIntersections() correctly computes all intersection points & the segments that contain them.

Proof. Let $p$ be an intersection pt. Assume (by induction):
- Every int. pt $q \prec p$ has been computed correctly.
- $\mathcal{T}$ contains all segments intersecting $\ell$ in left-to-right order.

Case I: $p$ is not an interior pt of a segment.
$\Rightarrow p$ has been inserted in $Q$ in the beginning.
Segm. in $U(p)$ and $L(p)$ are stored with $p$ in the beginning.
When $p$ is processed, we output all segm. in $U(p) \cup L(p)$. 
Correctness

**Lemma.** `findIntersections()` correctly computes all intersection points & the segments that contain them.

**Proof.** Let $p$ be an intersection pt. Assume (by induction):
- Every int. pt $q \prec p$ has been computed correctly.
- $\mathcal{T}$ contains all segments intersecting $\ell$ in left-to-right order.

**Case I:** $p$ is not an interior pt of a segment.

$\Rightarrow$ $p$ has been inserted in $Q$ in the beginning.

Segm. in $U(p)$ and $L(p)$ are stored with $p$ in the beginning.

When $p$ is processed, we output all segm. in $U(p) \cup L(p)$.

$\Rightarrow$ All segments that contain $p$ are reported.
Correctness (Case II)

Case II: $p$ is an interior point of some segment.
Correctness (Case II)

Case II: \( p \) is an interior point of some segment, i.e., \( C(p) \neq \emptyset \).
Correctness (Case II)

Case II: $p$ is an interior point of some segment, i.e., $C(p) \neq \emptyset$. If $p$ is not an endpt, need that $p$ is inserted into $Q$ before $\ell$ reaches $p$. 
Correctness (Case II)

Case II: $p$ is an interior point of some segment, i.e., $C(p) \neq \emptyset$. If $p$ is not an endpt, need that $p$ is inserted into $Q$ before $\ell$ reaches $p$. 

![Diagram showing a point $p$ and various lines intersecting at $p$.]
Correctness (Case II)

Case II: $p$ is an interior point of some segment, i.e., $C(p) \neq \emptyset$. If $p$ is not an endpt, need that $p$ is inserted into $Q$ before $\ell$ reaches $p$.

Let $s, s' \in C(p)$ be neighbors in the circular ordering of $C(p) \cup \{\ell\}$ around $p$. 
Correctness (Case II)

Case II: \( p \) is an interior point of some segment, i.e., \( C(p) \neq \emptyset \). If \( p \) is not an endpt, need that \( p \) is inserted into \( Q \) before \( \ell \) reaches \( p \).

Let \( s, s' \in C(p) \) be neighbors in the circular ordering of \( C(p) \cup \{ \ell \} \) around \( p \). Imagine moving \( \ell \) slightly back in time.
Correctness (Case II)

**Case II:** \( p \) is an interior point of some segment, i.e., \( C(p) \neq \emptyset \).

If \( p \) is not an endpt, need that \( p \) is inserted into \( Q \) before \( \ell \) reaches \( p \).

Let \( s, s' \in C(p) \) be neighbors in the circular ordering of \( C(p) \cup \{\ell\} \) around \( p \). Imagine moving \( \ell \) slightly back in time. Then \( s, s' \) were neighbors in the left-to-right order on \( \ell \) (in \( T \)).
Correctness (Case II)

Case II: \( p \) is an interior point of some segment, i.e., \( C(p) \neq \emptyset \).
If \( p \) is not an endpt, need that \( p \) is inserted into \( Q \) before \( \ell \) reaches \( p \).

Let \( s, s' \in C(p) \) be neighbors in the circular ordering of \( C(p) \cup \{\ell\} \) around \( p \). Imagine moving \( \ell \) slightly back in time. Then \( s, s' \) were neighbors in the left-to-right order on \( \ell \) (in \( T \)). At the beginning of the alg., they weren’t neighbors in \( T \).
Correctness (Case II)

Case II: $p$ is an interior point of some segment, i.e., $C(p) \neq \emptyset$. If $p$ is not an endpt, need that $p$ is inserted into $Q$ before $\ell$ reaches $p$.

Let $s, s' \in C(p)$ be neighbors in the circular ordering of $C(p) \cup \{\ell\}$ around $p$. Imagine moving $\ell$ slightly back in time. Then $s, s'$ were neighbors in the left-to-right order on $\ell$ (in $T$). At the beginning of the alg., they weren’t neighbors in $T$. ⇒ There was some moment when they became neighbors!
Correctness (Case II)

**Case II:** \( p \) is an interior point of some segment, i.e., \( C(p) \neq \emptyset \). If \( p \) is not an endpt, need that \( p \) is inserted into \( Q \) before \( \ell \) reaches \( p \).

Let \( s, s' \in C(p) \) be neighbors in the circular ordering of \( C(p) \cup \{\ell\} \) around \( p \). Imagine moving \( \ell \) slightly back in time. Then \( s, s' \) were neighbors in the left-to-right order on \( \ell \) (in \( T \)). At the beginning of the alg., they weren’t neighbors in \( T \).

⇒ There was some moment when they became neighbors! This is when \( \{p\} = s \cap s' \) was inserted into \( Q \).
Correctness (Case II)

Case II: \( p \) is an interior point of some segment, i.e., \( C(p) \neq \emptyset \).
If \( p \) is not an endpt, need that \( p \) is inserted into \( Q \) before \( \ell \) reaches \( p \).

Let \( s, s' \in C(p) \) be neighbors in the circular ordering of \( C(p) \cup \{\ell\} \) around \( p \). Imagine moving \( \ell \) slightly back in time. Then \( s, s' \) were neighbors in the left-to-right order on \( \ell \) (in \( T \)). At the beginning of the alg., they weren’t neighbors in \( T \).
⇒ There was some moment when they became neighbors!
This is when \( \{p\} = s \cap s' \) was inserted into \( Q \). □
\[ Q \leftarrow \emptyset; \ T \leftarrow \langle \text{vertical lines at } x = -\infty \text{ and } x = +\infty \rangle \] // sentinels

\textbf{foreach } s \in S \textbf{ do}

\quad \textbf{foreach endpoint } p \textbf{ of } s \textbf{ do}

\quad \quad \textbf{if } p \notin Q \textbf{ then } Q.\text{insert}(p); \ L(p) = U(p) = \emptyset

\quad \quad \textbf{if } p \text{ lower endpt of } s \textbf{ then } L(p).\text{append}(s)

\quad \quad \textbf{if } p \text{ upper endpt of } s \textbf{ then } U(p).\text{append}(s)

\textbf{while } Q \neq \emptyset \textbf{ do}

\quad p \leftarrow Q.\text{nextEvent}()

\quad Q.\text{deleteEvent}(p)

\quad \text{handleEvent}(p)

\textbf{Running time?}
\[ Q \leftarrow \emptyset; \ T \leftarrow \langle \text{vertical lines at } x = -\infty \text{ and } x = +\infty \rangle \]  // sentinels

\[
\text{foreach } s \in S \text{ do}
\]

\[
\quad \text{foreach endpoint } p \text{ of } s \text{ do}
\]

\[
\quad \quad \text{if } p \notin Q \text{ then } Q.\text{insert}(p); \quad L(p) = U(p) = \emptyset
\]

\[
\quad \quad \text{if } p \text{ lower endpt of } s \text{ then } L(p).\text{append}(s)
\]

\[
\quad \quad \text{if } p \text{ upper endpt of } s \text{ then } U(p).\text{append}(s)
\]

\[
\text{while } Q \neq \emptyset \text{ do}
\]

\[
\quad p \leftarrow Q.\text{nextEvent()}
\]

\[
\quad Q.\text{deleteEvent}(p)
\]

\[
\quad \text{handleEvent}(p)
\]

\[
\text{Running time?}
\]
$Q \leftarrow \emptyset; \ T \leftarrow \langle \text{vertical lines at } x = -\infty \text{ and } x = +\infty \rangle // \text{ sentinels}$

foreach $s \in S$ do

    foreach endpoint $p$ of $s$ do

        if $p \not\in Q$ then $Q$.insert($p$); $L(p) = U(p) = \emptyset$

        if $p$ lower endpt of $s$ then $L(p)$.append($s$)

        if $p$ upper endpt of $s$ then $U(p)$.append($s$)

while $Q \neq \emptyset$ do

    $p \leftarrow Q$.nextEvent()

    $Q$.deleteEvent($p$)

    handleEvent($p$)

    if $|U(p) \cup L(p) \cup C(p)| > 1$ then

        report intersection in $p$, report segments in $U(p) \cup L(p)$

        delete $L(p) \cup C(p)$ from $T$ // consecutive in $T$!

        insert $U(p) \cup C(p)$ into $T$ in their order slightly below $\ell$

        if $U(p) \cup C(p) = \emptyset$ then

            $b_{\text{left}}/b_{\text{right}} = \text{left/right neighbor of } p \text{ in } T$

            findNewEvent($b_{\text{left}}, b_{\text{right}}, p$)\rightarrow \{x\} = s \cap s'$

            if $x \not\in Q$ then $Q$.insert($x$)

        else

            $s_{\text{left}}/s_{\text{right}} = \text{leftmost/rightmost segment in } U(p) \cup C(p)$

            $b_{\text{left}} = \text{left neighbor of } s_{\text{left}} \text{ in } T$

            $b_{\text{right}} = \text{right neighbor of } s_{\text{right}} \text{ in } T$

            findNewEvent($b_{\text{left}}, s_{\text{left}}, p$)

            findNewEvent($b_{\text{right}}, s_{\text{right}}, p$)

Running time?
\[ Q \leftarrow \emptyset; \quad T \leftarrow \langle \text{vertical lines at } x = -\infty \text{ and } x = +\infty \rangle \quad // \text{sentinels} \]

\begin{algorithm}
\begin{algorithmic}
\State \textbf{foreach} \( s \in S \) \textbf{do}
\State \quad \textbf{foreach} endpoint \( p \) of \( s \) \textbf{do}
\State \quad \quad \textbf{if} \( p \notin Q \) \textbf{then} \( Q.\text{insert}(p); L(p) = U(p) = \emptyset \)
\State \quad \quad \textbf{if} \( p \) lower endpt of \( s \) \textbf{then} \( L(p).\text{append}(s) \)
\State \quad \quad \textbf{if} \( p \) upper endpt of \( s \) \textbf{then} \( U(p).\text{append}(s) \)
\State \end{algorithmic}
\end{algorithm}

\begin{algorithm}
\begin{algorithmic}
\State \textbf{while} \( Q \neq \emptyset \) \textbf{do}
\State \quad \( p \leftarrow Q.\text{nextEvent()} \)
\State \quad \( Q.\text{deleteEvent}(p) \)
\State \quad \textbf{handleEvent}(p)
\State \quad \textbf{if} \( |U(p) \cup L(p) \cup C(p)| > 1 \) \textbf{then}
\State \quad \quad \textbf{report intersection in} \( p \), \textbf{report segments in} \( U(p) \cup L(p) \)
\State \quad \quad \textbf{delete} \( L(p) \cup C(p) \) \textbf{from} \( T \) \quad // \text{consecutive in} \( T \)!
\State \quad \quad \textbf{insert} \( U(p) \cup C(p) \) \textbf{into} \( T \) \textbf{in their order slightly below} \( l \)
\State \quad \quad \textbf{if} \( U(p) \cup C(p) = \emptyset \) \textbf{then}
\State \quad \quad \quad \( b_{\text{left}}/b_{\text{right}} = \text{left/right neighbor of} \ p \ \text{in} \ T \)
\State \quad \quad \quad \textbf{findNewEvent}(b_{\text{left}}, b_{\text{right}}, p) \rightarrow \{x\} = s \cap s' \)
\State \quad \quad \quad \textbf{if} \( x \notin Q \) \textbf{then} \( Q.\text{insert}(x) \)
\State \quad \quad \textbf{else}
\State \quad \quad \quad \( s_{\text{left}}/s_{\text{right}} = \text{leftmost/rightmost segment in} \ U(p) \cup C(p) \)
\State \quad \quad \quad \( b_{\text{left}} = \text{left neighbor of} \ s_{\text{left}} \ \text{in} \ T \)
\State \quad \quad \quad \( b_{\text{right}} = \text{right neighbor of} \ s_{\text{right}} \ \text{in} \ T \)
\State \quad \quad \quad \textbf{findNewEvent}(b_{\text{left}}, s_{\text{left}}, p)
\State \quad \quad \quad \textbf{findNewEvent}(b_{\text{right}}, s_{\text{right}}, p)
\State \end{algorithmic}
\end{algorithm}

\textbf{Running time?}
\( Q \leftarrow \emptyset; \ T \leftarrow \langle \text{vertical lines at } x = -\infty \text{ and } x = +\infty \rangle \quad // \text{sentinels} \\
\text{foreach } s \in S \text{ do} \\
\quad \text{foreach endpoint } p \text{ of } s \text{ do} \\
\quad\quad \text{if } p \notin Q \text{ then } Q.\text{insert}(p); \ L(p) = U(p) = \emptyset \\
\quad\quad \text{if } p \text{ lower endpt of } s \text{ then } L(p).\text{append}(s) \\
\quad\quad \text{if } p \text{ upper endpt of } s \text{ then } U(p).\text{append}(s) \\
\text{while } Q \neq \emptyset \text{ do} \\
\quad p \leftarrow Q.\text{nextEvent}() \\
\quad Q.\text{deleteEvent}(p) \\
\quad \text{handleEvent}(p) \\
\text{handleEvent}(p) \\
\text{if } |U(p) \cup L(p) \cup C(p)| > 1 \text{ then} \\
\quad \text{report intersection in } p, \text{ report segments in } U(p) \cup L(p) \\
\quad \text{delete } L(p) \cup C(p) \text{ from } T \quad // \text{consecutive in } T! \\
\quad \text{insert } U(p) \cup C(p) \text{ into } T \text{ in their order slightly below } \ell \\
\quad \text{if } U(p) \cup C(p) = \emptyset \text{ then} \\
\quad\quad \text{b}_{\text{left}}/\text{b}_{\text{right}} = \text{left/right neighbor of } p \text{ in } T \\
\quad\quad \text{findNewEvent}(\text{b}_{\text{left}}, \text{b}_{\text{right}}, p) \rightarrow \{x\} = s \cap s' \\
\quad\quad \text{if } x \notin Q \text{ then } Q.\text{insert}(x) \\
\quad\quad \text{else} \\
\quad\quad\quad \text{s}_{\text{left}}/\text{s}_{\text{right}} = \text{leftmost/rightmost segment in } U(p) \cup C(p) \\
\quad\quad\quad \text{b}_{\text{left}} = \text{left neighbor of } s_{\text{left}} \text{ in } T \\
\quad\quad\quad \text{b}_{\text{right}} = \text{right neighbor of } s_{\text{right}} \text{ in } T \\
\quad\quad\quad \text{findNewEvent}(\text{b}_{\text{left}}, s_{\text{left}}, p) \\
\quad\quad\quad \text{findNewEvent}(\text{b}_{\text{right}}, s_{\text{right}}, p) \\
\text{Running time?}
Lemma. findIntersections() finds $I$ intersection points among $n$ non-overlapping line segments in $O((n + I) \log n)$ time.
Lemma. findIntersections() finds $I$ intersection points among $n$ non-overlapping line segments in $O((n + I) \log n)$ time.

Proof. Let $p$ be an event $pt$,

$m(p) = |L(p) \cup C(p)| + |U(p) \cup C(p)|$

and $m = \sum_p m(p)$.

Then it’s clear that the runtime is $O((m + n) \log n)$. 

Check your knowledge about planar graphs!
Proof. Let $p$ be an event pt, 
$m(p) = |L(p) ∪ C(p)| + |U(p) ∪ C(p)|$
and $m = ∑_p m(p)$. 
Then it’s clear that the runtime is $O((m + n) \log n)$. 
We show that $m ∈ O(n + I)$. 

Lemma. findIntersections() finds $I$ intersection points 
among $n$ non-overlapping line segments in 
$O((n + I) \log n)$ time.
Running Time

Lemma. findIntersections() finds \( I \) intersection points among \( n \) non-overlapping line segments in \( O((n + I) \log n) \) time.

Proof. Let \( p \) be an event pt, 
\[
m(p) = |L(p) \cup C(p)| + |U(p) \cup C(p)|
\]
and 
\[
m = \sum_p m(p).
\]
Then it’s clear that the runtime is \( O((m + n) \log n) \).
We show that \( m \in O(n + I) \). (⇒ lemma)
Running Time

Lemma. findIntersections() finds $I$ intersection points among $n$ non-overlapping line segments in $O((n + I) \log n)$ time.

Proof. Let $p$ be an event pt,

$$m(p) = |L(p) \cup C(p)| + |U(p) \cup C(p)|$$

and $m = \sum_p m(p)$.

Then it’s clear that the runtime is $O((m + n) \log n)$.

We show that $m \in O(n + I)$. ($\Rightarrow$ lemma)

Define (geometric) graph $G = (V, E)$ with $V = \{\text{endpts, intersection pts}\}$

Check your knowledge about planar graphs!
Lemma. \( \text{findIntersections()} \) finds \( I \) intersection points among \( n \) non-overlapping line segments in \( O((n + I) \log n) \) time.

Proof. Let \( p \) be an event pt,
\[
m(p) = |L(p) \cup C(p)| + |U(p) \cup C(p)|
\]
and \( m = \sum_p m(p) \).

Then it’s clear that the runtime is \( O((m + n) \log n) \).

We show that \( m \in O(n + I) \). (⇒ lemma)

Define (geometric) graph \( G = (V, E) \) with
\[
V = \{ \text{endpts, intersection pts} \}.
\]
Running Time

Lemma. findIntersections() finds $I$ intersection points among $n$ non-overlapping line segments in $O((n + I) \log n)$ time.

Proof. Let $p$ be an event pt,

$$m(p) = |L(p) \cup C(p)| + |U(p) \cup C(p)|$$

and $m = \sum_p m(p)$.

Then it’s clear that the runtime is $O((m + n) \log n)$.

We show that $m \in O(n + I)$. $(\Rightarrow$ lemma$)$

Define (geometric) graph $G = (V, E)$ with $V = \{\text{endpts, intersection pts}\} \Rightarrow |V| \leq 2n + I.$
Lemma. findIntersections() finds $I$ intersection points among $n$ non-overlapping line segments in $O((n + I) \log n)$ time.

Proof. Let $p$ be an event pt, 
$m(p) = |L(p) \cup C(p)| + |U(p) \cup C(p)|$
and 
$m = \sum_p m(p)$.

Then it’s clear that the runtime is $O((m + n) \log n)$.

We show that $m \in O(n + I)$. (⇒ lemma)
Define (geometric) graph $G = (V, E)$ with 
$V = \{ \text{endpts, intersection pts} \} \Rightarrow |V| \leq 2n + I$.

For any $p \in V$: $m(p) = \deg(p)$. 

Check your knowledge about planar graphs!
Lemma. findIntersections() finds $I$ intersection points among $n$ non-overlapping line segments in $O((n + I) \log n)$ time.

Proof. Let $p$ be an event pt, 
\[ m(p) = |L(p) \cup C(p)| + |U(p) \cup C(p)| \]
and 
\[ m = \sum_p m(p). \]
Then it’s clear that the runtime is $O((m + n) \log n)$. We show that $m \in O(n + I)$. ($\Rightarrow$ lemma) Define (geometric) graph $G = (V, E)$ with 
\[ V = \{ \text{endpts, intersection pts} \} \Rightarrow |V| \leq 2n + I. \]
For any $p \in V: m(p) = \deg(p)$. 
\[ \Rightarrow m = \sum_p \deg(p) = 2|E| \]
Running Time

Lemma. findIntersections() finds \( I \) intersection points among \( n \) non-overlapping line segments in \( O((n + I) \log n) \) time.

Proof. Let \( p \) be an event pt,

\[
m(p) = |L(p) \cup C(p)| + |U(p) \cup C(p)|
\]

and \( m = \sum_p m(p) \).

Then it’s clear that the runtime is \( O((m + n) \log n) \).

We show that \( m \in O(n + I) \). (\( \Rightarrow \) lemma)

Define (geometric) graph \( G = (V, E) \) with
\[
V = \{ \text{endpts, intersection pts} \} \Rightarrow |V| \leq 2n + I.
\]

For any \( p \in V: m(p) = \text{deg}(p) \).

\[
\Rightarrow m = \sum_p \text{deg}(p) = 2|E| \leq ...
\]
Running Time

Lemma. findIntersections() finds $I$ intersection points among $n$ non-overlapping line segments in $O((n + I) \log n)$ time.

Proof. Let $p$ be an event pt, 

$$m(p) = |L(p) \cup C(p)| + |U(p) \cup C(p)|$$

and $m = \sum_p m(p)$.

Then it's clear that the runtime is $O((m + n) \log n)$.

We show that $m \in O(n + I)$. ($\Rightarrow$ lemma)

Define (geometric) graph $G = (V, E)$ with $V = \{ \text{endpts, intersection pts} \} \Rightarrow |V| \leq 2n + I$.

For any $p \in V$: $m(p) = \deg(p)$.

$\Rightarrow m = \sum_p \deg(p) = 2|E| \leq$ Euler (G is planar!!)
Running Time

**Lemma.** findIntersections() finds $I$ intersection points among $n$ non-overlapping line segments in $O((n + I) \log n)$ time.

**Proof.** Let $p$ be an event pt,

$m(p) = |L(p) \cup C(p)| + |U(p) \cup C(p)|$

and $m = \sum_p m(p)$.

Then it’s clear that the runtime is $O((m + n) \log n)$.

We show that $m \in O(n + I)$. ($\Rightarrow$ Lemma)

Define (geometric) graph $G = (V, E)$ with

$V = \{ \text{endpts, intersection pts} \}$

$|V| \leq 2n + I$.

For any $p \in V$: $m(p) = \deg(p)$.

$\Rightarrow m = \sum_p \deg(p) = 2|E| \leq 2 \cdot (3|V| - 6)$

Euler (G is planar!!)
Running Time

Lemma. findIntersections() finds \( I \) intersection points among \( n \) non-overlapping line segments in \( O((n + I) \log n) \) time.

Proof. Let \( p \) be an event pt,

\[ m(p) = |L(p) \cup C(p)| + |U(p) \cup C(p)| \]

and \( m = \sum_p m(p) \).

Then it’s clear that the runtime is \( O((m + n) \log n) \).

We show that \( m \in O(n + I) \). (\( \Rightarrow \) lemma)

Define (geometric) graph \( G = (V, E) \) with

\[ V = \{ \text{endpts, intersection pts} \} \Rightarrow |V| \leq 2n + I. \]

For any \( p \in V: m(p) = \deg(p) \).

\[ \Rightarrow m = \sum_p \deg(p) = 2|E| \leq 2 \cdot (3|V| - 6) \]

Euler (\( G \) is planar!!)
Proof. Let \( p \) be an event pt,
\[
m(p) = |L(p) \cup C(p)| + |U(p) \cup C(p)|
\]
and
\[
m = \sum_p m(p).
\]
Then it’s clear that the runtime is \( O((m + n) \log n) \).

We show that \( m \in O(n + I) \). (\( \Rightarrow \) lemma)

Define (geometric) graph \( G = (V, E) \) with
\[
V = \{ \text{endpts, intersection pts} \} \Rightarrow |V| \leq 2n + I.
\]
For any \( p \in V \): \( m(p) = \deg(p) \).
\[
\Rightarrow m = \sum_p \deg(p) = 2|E| \leq 2 \cdot (3|V| - 6)
\]
\[
\in O(n + I) \quad \square
\]

Lemma. findIntersections() finds \( I \) intersection points among \( n \) non-overlapping line segments in \( O((n + I) \log n) \) time.
Today’s Main Result

**Theorem.** We can report all $I$ intersection points among $n$ non-overlapping line segments in the plane and report the segments involved in the intersections in $O((n + I) \log n)$ time and $O(n)$ space.
Theorem. We can report all \( I \) intersection points among \( n \) non-overlapping line segments in the plane and report the segments involved in the intersections in \( O((n + I) \log n) \) time and \( O(n) \) space.
Today’s Main Result

**Theorem.** We can report all $I$ intersection points among $n$ non-overlapping line segments in the plane and report the segments involved in the intersections in $O((n + I) \log n)$ time and $O(n)$ space.

Sure? The event-point queue $Q$ contains

- all segment end pts *below the sweep line*
- all intersection pts *below the sweep line*

$\Rightarrow$ (worst-case) space consumption $\in$
Theorem. We can report all \( I \) intersection points among \( n \) non-overlapping line segments in the plane and report the segments involved in the intersections in \( O((n + I) \log n) \) time and \( O(n) \) space.

Sure? The event-point queue \( Q \) contains

- all segment end pts below the sweep line
- all intersection pts below the sweep line

\( \Rightarrow \) (worst-case) space consumption \( \in \Theta(n + I) \) :-(
Today’s Main Result

**Theorem.** We can report all $I$ intersection points among $n$ non-overlapping line segments in the plane and report the segments involved in the intersections in $O((n + I) \log n)$ time and $O(n)$ space.

Sure? The event-point queue $Q$ contains
- all segment end pts below the sweep line
- all intersection pts below the sweep line

$\Rightarrow$ (worst-case) space consumption $\in \Theta(n + I)$ :-(

Can we do better?
Theorem. We can report all $I$ intersection points among $n$ non-overlapping line segments in the plane and report the segments involved in the intersections in $O((n + I) \log n)$ time and $O(n)$ space.

Sure? The event-point queue $Q$ contains

- all segment end pts below the sweep line
- all intersection pts below the sweep line

$\Rightarrow$ (worst-case) space consumption $\in \Theta(n + I)$ :-(

Can we do better?
Today’s Main Result

**Theorem.** We can report all $I$ intersection points among $n$ non-overlapping line segments in the plane and report the segments involved in the intersections in $O((n + I) \log n)$ time and $O(n)$ space.

Sure? The event-point queue $Q$ contains
- all segment end pts **below the sweep line**
- all intersection pts **below the sweep line**

$\Rightarrow$ (worst-case) space consumption $\in \Theta(n + I)$ :-(

Can we do better?
**Today’s Main Result**

**Theorem.** We can report all \( I \) intersection points among \( n \) non-overlapping line segments in the plane and report the segments involved in the intersections in \( O((n + I) \log n) \) time and \( O(n) \) space.

Sure? The event-point queue \( Q \) contains
- all segment end pts *below the sweep line*
- all intersection pts *below the sweep line*

\( \Rightarrow \) (worst-case) space consumption \( \in \Theta(n + I) \) :-(

Can we do better?

\(-\) insert \( s \cap s' \) into \( Q \)
Theorem. We can report all $I$ intersection points among $n$ non-overlapping line segments in the plane and report the segments involved in the intersections in $O((n + I) \log n)$ time and $O(n)$ space.

Sure? The event-point queue $Q$ contains
- all segment end pts below the sweep line
- all intersection pts below the sweep line
\[\Rightarrow \text{(worst-case) space consumption } \in \Theta(n + I) \quad :-(\]

Can we do better?
- insert $s \cap s'$ into $Q$
Today’s Main Result

**Theorem.** We can report all \( I \) intersection points among \( n \) non-overlapping line segments in the plane and report the segments involved in the intersections in \( O((n + I) \log n) \) time and \( O(n) \) space.

Sure? The event-point queue \( Q \) contains

- all segment end pts below the sweep line
- all intersection pts below the sweep line

\[ \Rightarrow \text{(worst-case) space consumption } \in \Theta(n + I) \]

Can we do better? Theorem.

- insert \( s \cap s' \) into \( Q \)
- remove \( s \cap s' \) from \( Q \)
Theorem. We can report all $I$ intersection points among $n$ non-overlapping line segments in the plane and report the segments involved in the intersections in $O((n + I) \log n)$ time and $O(n)$ space.

Sure? The event-point queue $Q$ contains

- all segment end pts below the sweep line
- all intersection pts below the sweep line

$\Rightarrow$ (worst-case) space consumption $\in \Theta(n + I) :-(

Can we do better?

- insert $s \cap s'$ into $Q$
- remove $s \cap s'$ from $Q$
Theorem. We can report all $I$ intersection points among $n$ non-overlapping line segments in the plane and report the segments involved in the intersections in $O((n + I) \log n)$ time and $O(n)$ space.

Sure? The event-point queue $Q$ contains
- all segment end pts below the sweep line
- all intersection pts below the sweep line

$\Rightarrow$ (worst-case) space consumption $\in \Theta(n + I)$

Can we do better?
- insert $s \cap s'$ into $Q$
- remove $s \cap s'$ from $Q$
- re-insert $s \cap s'$ into $Q$
**Theorem.** We can report all $I$ intersection points among $n$ non-overlapping line segments in the plane and report the segments involved in the intersections in $O((n + I) \log n)$ time and $O(n)$ space.

Sure? The event-point queue $Q$ contains

- all segment end pts below the sweep line
- all intersection pts below the sweep line

$\Rightarrow$ (worst-case) space consumption $\in \Theta(n + I)$ :-(

Can we do better?

- insert $s \cap s'$ into $Q$
- remove $s \cap s'$ from $Q$
- re-insert $s \cap s'$ into $Q$

$\Rightarrow$ need just $O(n)$ space;
Today’s Main Result

**Theorem.** We can report all $I$ intersection points among $n$ non-overlapping line segments in the plane and report the segments involved in the intersections in $O((n + I) \log n)$ time and $O(n)$ space.

**Sure?** The event-point queue $Q$ contains
• all segment end pts below the sweep line
• all intersection pts below the sweep line
⇒ (worst-case) space consumption $\in \Theta(n + I)$ :-(

**Can we do better?**

- insert $s \cap s'$ into $Q$
- remove $s \cap s'$ from $Q$
- re-insert $s \cap s'$ into $Q$
⇒ need just $O(n)$ space;
(asymptotic) running time doesn’t change