Computational Geometry · Lecture
Line Segment Intersection

Tamara Mchedlidze
25.04.2018
Overlaying Map Layers

**Example:** Given two different map layers whose intersection is of interest.

Land use

Precipitation

Map combining themes
Overlaying Map Layers

**Example:** Given two different map layers whose intersection is of interest.

- **Land use**
- **Precipitation**
- **Map combining themes**

- Regions are polygons
- Polygons are line segments
- **Calculate all line segment intersections**
- Compute regions
Problem Formulation

**Given:** Set $S = \{s_1, \ldots, s_n\}$ of line segments in the plane

**Output:** • all intersections of two or more line segments
• for each intersection, the line segments involved.
Problem Formulation

**Given:** Set $S = \{s_1, \ldots, s_n\}$ of line segments in the plane

**Output:**
- all intersections of two or more line segments
- for each intersection, the line segments involved.

**Def:** Line segments are closed
Problem Formulation

Given: Set \( S = \{s_1, \ldots, s_n\} \) of line segments in the plane

Output: • all intersections of two or more line segments
  • for each intersection, the line segments involved.

Def: Line segments are closed

Discussion:
– How can you solve this problem naively?
– Is this already optimal?
– Are there better approaches?
The Sweep-Line Method: An Example
The Sweep-Line Method: An Example

Events
The Sweep-Line Method: An Example

Diagram showing an example of line segment intersection using the sweep-line method.
The Sweep-Line Method: An Example
The Sweep-Line Method: An Example
The Sweep-Line Method: An Example
The Sweep-Line Method: An Example
The Sweep-Line Method: An Example
The Sweep-Line Method: An Example

The diagram illustrates the sweep-line method for detecting line segment intersections. The sweep line, represented by a red horizontal line, moves from left to right. As it sweeps across the plane, it intersects with line segments, and the points of intersection are marked. This method efficiently computes all intersections among a set of line segments.
The Sweep-Line Method: An Example

Graph showing line segments intersecting on a plane with a sweep line.
The Sweep-Line Method: An Example

Line Segment Intersection
The Sweep-Line Method: An Example
The Sweep-Line Method: An Example
The Sweep-Line Method: An Example

Line Segment Intersection
The Sweep-Line Method: An Example
The Sweep-Line Method: An Example
The Sweep-Line Method: An Example
The Sweep-Line Method: An Example
The Sweep-Line Method: An Example
The Sweep-Line Method: An Example
The Sweep-Line Method: An Example
The Sweep-Line Method: An Example
1.) Event Queue $Q$

- define $p \prec q \iff \text{def. } y_p > y_q \lor (y_p = y_q \land x_p < x_q)$
Data Structures

1.) Event Queue $Q$

- Define $p \prec q \iff_{\text{def.}} y_p > y_q \lor (y_p = y_q \land x_p < x_q)$

- Store events by $\prec$ in a balanced binary search tree → e.g., AVL tree
Data Structures

1.) Event Queue $Q$

- define $p \prec q \iff \text{def. } y_p > y_q \lor (y_p = y_q \land x_p < x_q)$

- Store events by $\prec$ in a balanced binary search tree
  → e.g., AVL tree

- Operations insert, delete and nextEvent in $O(\log |Q|)$ time
Data Structures

1.) Event Queue $Q$

- define $p \prec q \iff \text{def. } y_p > y_q \lor (y_p = y_q \land x_p < x_q)$

- Store events by $\prec$ in a balanced binary search tree → e.g., AVL tree

- Operations insert, delete and nextEvent in $O(\log |Q|)$ time

2.) Sweep-Line Status $\mathcal{T}$

- Stores $\ell$ cut lines ordered from left to right
Data Structures

1.) Event Queue $Q$

• define $p \prec q \iff \text{def. } y_p > y_q \lor (y_p = y_q \land x_p < x_q)$

\[ p \quad \cdots \quad q \]

• Store events by $\prec$ in a balanced binary search tree

→ e.g., AVL tree

• Operations insert, delete and nextEvent in $O(\log |Q|)$ time

2.) Sweep-Line Status $T$

• Stores $\ell$ cut lines ordered from left to right

• Required operations insert, delete, findNeighbor
Data Structures

1.) Event Queue $Q$

- define $p \prec q \iff \text{def. } y_p > y_q \lor (y_p = y_q \land x_p < x_q)$

- Store events by $\prec$ in a balanced binary search tree
  \[\rightarrow \text{ e.g., AVL tree}\]

- Operations insert, delete and nextEvent in $O(\log |Q|)$ time

2.) Sweep-Line Status $T$

- Stores $\ell$ cut lines ordered from left to right
- Required operations insert, delete, findNeighbor
- This is also a balanced binary search tree with line segments stored in the leaves!
Algorithm

FindIntersections($S'$)

**Input:** Set $S$ of line segments  
**Output:** Set of all intersection points and the line segments involved

$Q \leftarrow \emptyset; \quad T \leftarrow \emptyset$

foreach $s \in S$ do
  $Q$.insert(upperEndPoint($s$))
  $Q$.insert(lowerEndPoint($s$))

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

Algorithm

FindIntersections($S$)

**Input:** Set $S$ of line segments  
**Output:** Set of all intersection points and the line segments involved

$Q \leftarrow \emptyset; \quad T \leftarrow \emptyset$

**foreach** $s \in S$ **do**

$Q$.insert(upperEndPoint($s$))  
$Q$.insert(lowerEndPoint($s$))

**while** $Q \neq \emptyset$ **do**

$p \leftarrow Q$.nextEvent()  
$Q$.deleteEvent($p$)  
handleEvent($p$)

Store the segment together with its upper end point.
Algorithm

FindIntersections($S'$)

**Input:** Set $S$ of line segments

**Output:** Set of all intersection points and the line segments involved

$Q \leftarrow \emptyset; \quad T \leftarrow \emptyset$

**foreach** $s \in S$ **do**

- $Q$.insert(upperEndPoint($s$))
- $Q$.insert(lowerEndPoint($s$))

**while** $Q \neq \emptyset$ **do**

- $p \leftarrow Q$.nextEvent()
- $Q$.deleteEvent($p$)
- `handleEvent(p)`

What happens with duplicates?
Algorithm

FindIntersections($S'$)

**Input**: Set $S$ of line segments  
**Output**: Set of all intersection points and the line segments involved  

$Q \leftarrow \emptyset$; $T \leftarrow \emptyset$

foreach $s \in S$ do

$Q$.insert(upperEndPoint($s$))

$Q$.insert(lowerEndPoint($s$))

while $Q \neq \emptyset$ do

$p \leftarrow Q$.nextEvent()

$Q$.deleteEvent($p$)

handleEvent($p$)

What happens with duplicates?

This is the core of the algorithm!
Algorithm

```plaintext
handleEvent(p)

  U(p) ← Line segments with p as upper endpoint
  L(p) ← Line segments with p as lower endpoint
  C(p) ← Line segments with p as interior point

  if |U(p) ∪ L(p) ∪ C(p)| ≥ 2 then
    report p and U(p) ∪ L(p) ∪ C(p)

  remove L(p) ∪ C(p) from T
  add U(p) ∪ C(p) to T

  if U(p) ∪ C(p) = ∅ then // s_l and s_r, neighbors of p in T
    Q ← check if s_l and s_r intersect below p
  else // s' and s'' leftmost and rightmost line segment in U(p) ∪ C(p)
    Q ← check if s_l and s' intersect below p
    Q ← check if s_r and s'' intersect below p
```

Algorithm

handleEvent\( (p) \)

\[ U(p) \leftarrow \text{Line segments with } p \text{ as upper endpoint} \]
\[ L(p) \leftarrow \text{Line segments with } p \text{ as lower endpoint} \]
\[ C(p) \leftarrow \text{Line segments with } p \text{ as interior point} \]

if \( |U(p) \cup L(p) \cup C(p)| \geq 2 \) then

\[ \text{report } p \text{ and } U(p) \cup L(p) \cup C(p) \]

remove \( L(p) \cup C(p) \) from \( T \)

add \( U(p) \cup C(p) \) to \( T \)

if \( U(p) \cup C(p) = \emptyset \) then   //\( s_l \) and \( s_r \), neighbors of \( p \) in \( T \)

\[ Q \leftarrow \text{check if } s_l \text{ and } s_r \text{ intersect below } p \]

else   //\( s' \) and \( s'' \) leftmost and rightmost line segment in \( U(p) \cup C(p) \)

\[ Q \leftarrow \text{check if } s_l \text{ and } s' \text{ intersect below } p \]

\[ Q \leftarrow \text{check if } s_r \text{ and } s'' \text{ intersect below } p \]
Algorithm

`handleEvent(p)`

- `U(p) ← Line segments with p as upper endpoint`
- `L(p) ← Line segments with p as lower endpoint`
- `C(p) ← Line segments with p as interior point`

**if** `|U(p) ∪ L(p) ∪ C(p)| ≥ 2` **then**
- report `p` and `U(p) ∪ L(p) ∪ C(p)`

**remove** `L(p) ∪ C(p)` **from** `T`
**add** `U(p) ∪ C(p)` **to** `T`

**if** `U(p) ∪ C(p) = ∅` **then**
- `Q ← check if sl and sr intersect below p`
- 
- **else**
- `Q ← check if s' and s'' intersect below p`
- `Q ← check if sl and s' intersect below p`
- `Q ← check if sr and s'' intersect below p`
Algorithm

handleEvent(p)

\[ U(p) \leftarrow \text{Line segments with } p \text{ as upper endpoint} \]
\[ L(p) \leftarrow \text{Line segments with } p \text{ as lower endpoint} \]
\[ C(p) \leftarrow \text{Line segments with } p \text{ as interior point} \]
\[ \text{if } |U(p) \cup L(p) \cup C(p)| \geq 2 \text{ then} \]
\[ \quad \text{report } p \text{ and } U(p) \cup L(p) \cup C(p) \]
\[ \text{remove } L(p) \cup C(p) \text{ from } \mathcal{T} \]
\[ \text{add } U(p) \cup C(p) \text{ to } \mathcal{T} \]
\[ \text{if } U(p) \cup C(p) = \emptyset \text{ then} \]
\[ \quad Q \leftarrow \text{check if } s_l \text{ and } s_r \text{ intersect below } p \]
\[ \text{else} \]
\[ \quad Q \leftarrow \text{check if } s'_l \text{ and } s''_l \text{ intersect below } p \]
\[ \quad Q \leftarrow \text{check if } s'_r \text{ and } s''_r \text{ intersect below } p \]
What Happens Exactly?

\( S \)

\( T \)

\( s_1 \)

\( s_2 \)

\( s_3 \)

\( s_4 \)

\( s_5 \)

\( s_6 \)

\( s_7 \)

\( p \)
What Happens Exactly?

\[ U(p) = \{s_2\} \]
\[ L(p) = \]
\[ C(p) = \]
What Happens Exactly?

\[ U(p) = \{ s_2 \} \]
\[ L(p) = \]
\[ C(p) = \]
What Happens Exactly?

\[ U(p) = \{s_2\} \]
\[ L(p) = \{s_4, s_5\} \]
\[ C(p) = \{s_1, s_3\} \]
What Happens Exactly?

\[ U(p) = \{s_2\} \]
\[ L(p) = \{s_4, s_5\} \]
\[ C(p) = \{s_1, s_3\} \]

\[
\text{Report} \ (p, \{s_1, s_2, s_3, s_4, s_5\})
\]
What Happens Exactly?

\[ U(p) = \{s_2\} \]
\[ L(p) = \{s_4, s_5\} \]
\[ C(p) = \{s_1, s_3\} \]

Delete \( L(p) \cup C(p) \); add \( U(p) \cup C(p) \)
What Happens Exactly?

\[ S \]

\[ \begin{align*}
U(p) &= \{s_2\} \\
L(p) &= \{s_4, s_5\} \\
C(p) &= \{s_1, s_3\}
\end{align*} \]

Add event \( p' = s_1 \times s_7 \) in \( Q \).
Correctness

Lemma 1: Algorithm FindIntersections finds all intersection points and the line segments involved
Correctness

**Lemma 1:** Algorithm FindIntersections finds all intersection points and the line segments involved

**Proof:**
Induction on the number of events processed, ordered by their priority.

Let \( p \) be an intersection point and all intersection points \( q \prec p \) are already correctly computed.

**Case 1:** \( p \) is a line segment endpoint
- \( p \) was inserted in \( Q \)
- \( U(p) \) are stored with \( p \)
- \( L(p) \) and \( C(p) \) are in \( T \)
Correctness

Lemma 1: Algorithm FindIntersections finds all intersection points and the line segments involved

Proof:
Induction on the number of events processed, ordered by their priority.

Let \( p \) be an intersection point and all intersection points \( q \prec p \) are already correctly computed.

Case 1: \( p \) is a line segment endpoint
- \( p \) was inserted in \( Q \)
- \( U(p) \) are stored with \( p \)
- \( L(p) \) and \( C(p) \) are in \( T \)

Case 2: \( p \) is not a line segment endpoint

Consider why \( p \) must be in \( Q \)!
Running-Time Analysis

FindIntersections($S$)

**Input:** Set $S$ of line segments  
**Output:** Set of all intersections with their line segments

```plaintext
Q ← ∅; T ← ∅

foreach $s ∈ S$ do
  Q.insert(upperEndPoint($s$))
  Q.insert(lowerEndPoint($s$))

while $Q ≠ ∅$ do
  $p ← Q$.nextEvent()
  Q.deleteEvent($p$)
  handleEvent($p$)
```

---

Dr. Tamara Mchedlidze · Dr. Darren Strash · Computational Geometry Lecture
Running-Time Analysis

FindIntersections\( (S') \)

**Input:** Set \( S \) of line segments

**Output:** Set of all intersections with their line segments

\[
Q \leftarrow \emptyset; \quad T \leftarrow \emptyset
\]

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

\[
Q.\text{insert}(\text{upperEndPoint}(s))
\]

\[
Q.\text{insert}(\text{lowerEndPoint}(s))
\]

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

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

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

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

\[O(1)\]
# Running-Time Analysis

**FindIntersections**($S$)

**Input:** Set $S$ of line segments  
**Output:** Set of all intersections with their line segments

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$Q \leftarrow \emptyset$; $T \leftarrow \emptyset$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td><strong>foreach</strong> $s \in S$ <strong>do</strong></td>
<td>$O(n \log n)$</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>$Q$.insert(upperEndPoint($s$))</td>
<td></td>
</tr>
<tr>
<td>$Q$.insert(lowerEndPoint($s$))</td>
<td></td>
</tr>
<tr>
<td><strong>while</strong> $Q \neq \emptyset$ <strong>do</strong></td>
<td></td>
</tr>
<tr>
<td>$p \leftarrow Q$.nextEvent()</td>
<td></td>
</tr>
<tr>
<td>$Q$.deleteEvent($p$)</td>
<td></td>
</tr>
<tr>
<td>handleEvent($p$)</td>
<td></td>
</tr>
</tbody>
</table>

10  
Dr. Tamara Mchedlidze · Dr. Darren Strash · Computational Geometry Lecture  
Line Segment Intersection
## Running-Time Analysis

**FindIntersections**($S'$)

**Input:** Set $S$ of line segments

**Output:** Set of all intersections with their line segments

<table>
<thead>
<tr>
<th>Q $\leftarrow \emptyset$; $T \leftarrow \emptyset$</th>
<th>$O(1)$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>foreach</strong> $s \in S$ <strong>do</strong></td>
<td></td>
</tr>
<tr>
<td>$Q$.insert(upperEndPoint($s$))</td>
<td>$O(n \log n)$</td>
</tr>
<tr>
<td>$Q$.insert(lowerEndPoint($s$))</td>
<td></td>
</tr>
<tr>
<td><strong>while</strong> $Q \neq \emptyset$ <strong>do</strong></td>
<td></td>
</tr>
<tr>
<td>$p \leftarrow Q$.nextEvent()</td>
<td>$O(\log</td>
</tr>
<tr>
<td>$Q$.deleteEvent($p$)</td>
<td></td>
</tr>
<tr>
<td>handleEvent($p$)</td>
<td></td>
</tr>
</tbody>
</table>
Running-Time Analysis

FindIntersections($S'$)

**Input:** Set $S$ of line segments  
**Output:** Set of all intersections with their line segments

<table>
<thead>
<tr>
<th>$Q \leftarrow \emptyset$; $T \leftarrow \emptyset$</th>
<th>$O(1)$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>foreach</strong> $s \in S$ <strong>do</strong></td>
<td></td>
</tr>
<tr>
<td>$Q$.insert(upperEndPoint($s$))</td>
<td>$O(n \log n)$</td>
</tr>
<tr>
<td>$Q$.insert(lowerEndPoint($s$))</td>
<td></td>
</tr>
<tr>
<td><strong>while</strong> $Q \neq \emptyset$ <strong>do</strong></td>
<td></td>
</tr>
<tr>
<td>$p \leftarrow Q$.nextEvent()</td>
<td>$O(\log</td>
</tr>
<tr>
<td>$Q$.deleteEvent($p$)</td>
<td>?</td>
</tr>
<tr>
<td>handleEvent($p$)</td>
<td></td>
</tr>
</tbody>
</table>
Running-Time Analysis

handleEvent(p)

\[ U(p) \leftarrow \text{Line segments with } p \text{ as upper endpoint} \]
\[ L(p) \leftarrow \text{Line segments with } p \text{ as lower endpoint} \]
\[ C(p) \leftarrow \text{Line segments with } p \text{ as interior point} \]
\[ \text{if } |U(p) \cup L(p) \cup C(p)| \geq 2 \text{ then} \]
\[ \quad \text{return } p \text{ and } U(p) \cup L(p) \cup C(p) \]
\[ \text{remove } L(p) \cup C(p) \text{ from } \mathcal{T} \]
\[ \text{add } U(p) \cup C(p) \text{ to } \mathcal{T} \]
\[ \text{if } U(p) \cup C(p) = \emptyset \text{ then} \quad // s_l \text{ and } s_r, \text{ neighbors of } p \text{ in } \mathcal{T} \]
\[ \quad Q \leftarrow \text{check if } s_l \text{ and } s_r \text{ intersect below } p \]
\[ \text{else} \quad // s' \text{ and } s'' \text{ leftmost and rightmost line segment in } U(p) \cup C(p) \]
\[ \quad Q \leftarrow \text{check if } s_l \text{ and } s' \text{ intersect below } p \]
\[ \quad Q \leftarrow \text{check if } s_r \text{ and } s'' \text{ intersect below } p \]

Lemma 2: Algorithm FindIntersections has running time
\[ O(n \log n + I \log n), \text{ where } I \text{ is the number of intersection points.} \]
Summary

**Thm 1:** Let $S$ be a set of $n$ line segments in the plane. Then we can compute intersections in $S$ together with the involved line segments in $O((n + I) \log n)$ time and $O(?)$ space.
Summary

**Thm 1:** Let $S$ be a set of $n$ line segments in the plane. Then we can compute intersections in $S$ together with the involved line segments in $O((n + I) \log n)$ time and $O(?)$ space.

**Proof:**
- Correctness ✓
- Running time ✓
- Space
Summary

**Thm 1:** Let \( S \) be a set of \( n \) line segments in the plane. Then we can compute intersections in \( S \) together with the involved line segments in \( O((n + I) \log n) \) time and \( O(?) \) space.

**Proof:**

- Correctness ✓
- Running time ✓
- Space

Consider how much space the data structures need!
Summary

**Thm 1:** Let $S$ be a set of $n$ line segments in the plane. Then we can compute intersections in $S$ together with the involved line segments in $O((n + I) \log n)$ time and $O(n)$ space.

**Proof:**
- Correctness ✓
- Running time ✓
- Space
  - $T$ has at most $n$ elements
  - $Q$ has at most $O(n + I)$ elements
  - reduction of $Q$ to $O(n)$ space: an exercise

Consider how much space the data structures need!
Discussion

Is the Sweep-Line Algorithm always better than the naive one?
Discussion

Is the Sweep-Line Algorithm always better than the naive one?

No, because if \( I \in \Omega(n^2) \) then the algorithm has running time \( O(n^2 \log n) \).
Discussion

Is the Sweep-Line Algorithm always better than the naive one?

No, because if $I \in \Omega(n^2)$ then the algorithm has running time $O(n^2 \log n)$.

Can we do better?
Discussion

Is the Sweep-Line Algorithm always better than the naive one?

No, because if $I \in \Omega(n^2)$ then the algorithm has running time $O(n^2 \log n)$.

Can we do better?

Yes, in $\Theta(n \log n + I)$ time and $\Theta(n)$ space [Balaban, 1995].
Discussion

Is the Sweep-Line Algorithm always better than the naive one?
No, because if $I \in \Omega(n^2)$ then the algorithm has running time $O(n^2 \log n)$.

Can we do better?
Yes, in $\Theta(n \log n + I)$ time and $\Theta(n)$ space [Balaban, 1995].

How does this solve the map overlay problem?
Discussion

Is the Sweep-Line Algorithm always better than the naive one?
No, because if $I \in \Omega(n^2)$ then the algorithm has running time $O(n^2 \log n)$.

Can we do better?
Yes, in $\Theta(n \log n + I)$ time and $\Theta(n)$ space [Balaban, 1995].

How does this solve the map overlay problem?
Using an appropriate data structure (doubly-connected edgelist) for planar graphs we can compute in $O((n + I) \log n)$ time the overlay of two maps.
(Details in Ch. 2.3 of the book)