Geometric Algorithms

Lecture 2:

Line segment intersection for map overlay
Map layers

In a geographic information system (GIS) data is stored in separate layers.

A layer stores the geometric information about some theme, like land cover, road network, municipality boundaries, red fox habitat, ...
Map overlay is the combination of two (or more) map layers

It is needed to answer questions like:

- What is the total length of roads through forests?
- What is the total area of corn fields within 1 km from a river?
To solve map overlay questions, we need (at the least) intersection points from two sets of line segments (possibly, boundaries of regions)
The (easy) problem

Let’s first look at the easiest version of the problem:

Where did the caribous cross roads?

Given a set of \( n \) line segments in the plane, find all intersection points efficiently.
**Algorithm**  \textsc{FindIntersections}(S)

*Input.* A set $S$ of line segments in the plane.

*Output.* The set of intersection points among the segments in $S$.

1. \textbf{for} each pair of line segments $e_i, e_j \in S$
2. \textbf{do if} $e_i$ and $e_j$ intersect
3. \textbf{then} report their intersection point

**Question:** Why can we say that this algorithm is optimal?
Motivation: Map overlay
Problem
Output-sensitive algorithms
Some attempts
Output-sensitive algorithm

The asymptotic running time of an algorithm is always input-sensitive (depends on $n$)

We may also want the running time to be output-sensitive: if the output is large, it is fine to spend a lot of time, but if the output is small, we want a fast algorithm.
**Question:** How many intersection points do we typically expect in our applications?

If this number is $k$, and if $k = O(n)$, it would be nice if the algorithm runs in $O(n \log n)$ time.
First attempt

**Observation:** Two line segments can only intersect if their $y$-spans have an overlap.

So, how about only testing pairs of line segments that intersect in the $y$-projection?

1-D problem: Given a set of intervals on the real line, find all partly overlapping pairs.

- $(s_1, s_2)$
- $(s_4, s_6)$
- $(s_5, s_6)$
**Refined observation:** Two line segments can only intersect if their $y$-spans have an overlap, and they are adjacent in the $x$-order at that $y$-coordinate (they are *horizontal neighbors*)
The **plane sweep technique**: Imagine a horizontal line passing over the plane from top to bottom, solving the problem as it moves

- The sweep line stops and the algorithm computes at certain positions ⇒ **events**
- The algorithm stores the relevant situation at the current position of the sweep line ⇒ **status**
- The algorithm knows everything it needs to know above the sweep line, and found all intersection points
Sweep
Sweep and status

computed

status

unexplored
The status of this particular plane sweep algorithm, at the current position of the sweep line, is the set of line segments intersecting the sweep line, ordered from left to right.

The events occur when the status changes, and when output is generated.

\[ \text{event} \approx \text{interesting } y\text{-coordinate} \]
Line segment intersection
Plane sweep

Introduction
Events, status, structures
Event handling
Efficiency

add \( s_2 \) after \( s_1 \)

\[ s_1 \]
\[ s_2 \]
\[ s_3 \]
\[ s_4 \]
\[ s_5 \]
\[ s_6 \]
\[ s_7 \]
\[ s_8 \]
add $s_3$ between $s_1$ and $s_2$
Line segment intersection
Plane sweep

Introduction
Events, status, structures
Event handling
Efficiency

Geometric Algorithms
Lecture 2: Line segment intersection

add \( s_4 \) before \( s_1 \)

Diagram: Line segments \( S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8 \) with intersections indicated.
report intersection \((s_1,s_2)\); swap \(s_1\) and \(s_3\)
Introduction
Events, status, structures
Event handling
Efficiency

Line segment intersection
Plane sweep

Geometric Algorithms
Lecture 2: Line segment intersection

$S_1$ $S_3$ $S_2$
$S_4$ $S_5$ $S_6$ $S_7$ $S_8$

remove $S_2$
Geometric Algorithms
Lecture 2: Line segment intersection

Line segment intersection
Plane sweep

Introduction
Events, status, structures
Event handling
Efficiency

remove $s_1$
Introduction
Events, status, structures
Event handling
Efficiency

Geometric Algorithms | Lecture 2: Line segment intersection

Line segment intersection
Plane sweep

$s_1$, $s_2$, $s_3$, $s_4$, $s_5$, $s_6$, $s_7$, $s_8$

add $s_5$
report intersection \((s_3, s_4)\); swap \(s_3\) and \(s_4\)
... and so on ...
The events

When do the events happen? When the sweep line is

- at an upper endpoint of a line segment
- at a lower endpoint of a line segment
- at an intersection point of a line segment

At each type, the status changes; at the third type output is found too
We will at first exclude degenerate cases:

- No two endpoints have the same $y$-coordinate
- No more than two line segments intersect in a point
- ...

**Question:** Are there more degenerate cases?
Event list and status structure

The **event list** is an abstract data structure that stores all events in the order in which they occur.

The **status structure** is an abstract data structure that maintains the current status.

*Here:* The status is the subset of currently intersected line segments in the order of intersection by the sweep line.
We use a balanced binary search tree with the line segments in the leaves as the **status structure**.
Status structure

Upper endpoint: search, and insert
Status structure

Upper endpoint: search, and insert
Status structure

Upper endpoint: search, and insert
Sweep line reaches lower endpoint of a line segment: delete from the status structure

Sweep line reaches intersection point: swap two leaves in the status structure (and update information on the search paths)
Before the sweep algorithm starts, we know all upper endpoint events and all lower endpoint events.

But: How do we know intersection point events???
(those we were trying to find . . .)

Recall: Two line segments can only intersect if they are horizontal neighbors.
Finding events

**Lemma:** Two line segments $s_i$ and $s_j$ can only intersect after (= below) they have become horizontal neighbors

**Proof:** Just imagine that the sweep line is ever so slightly above the intersection point of $s_i$ and $s_j$, but below any other event  □

Also: some earlier (= higher) event made $s_i$ and $s_j$ horizontally adjacent!!!
The **event list** must be a balanced binary search tree, because during the sweep, we discover **new events** that will happen later and we want to be able to test whether an event is already in the list.

We know upper endpoint events and lower endpoint events beforehand; we find intersection point events when the involved line segments become horizontal neighbors.
Algorithm $\text{FindIntersections}(S)$

*Input.* A set $S$ of line segments in the plane.

*Output.* The intersection points of the segments in $S$, with for each intersection point the segments that contain it.

1. Initialize an empty event queue $Q$. Next, insert the segment endpoints into $Q$; when an upper endpoint is inserted, the corresponding segment should be stored with it.
2. Initialize an empty status structure $T$.
3. **while** $Q$ is not empty
4. **do** Determine next event point $p$ in $Q$ and delete it
5. $\text{HandleEventPoint}(p)$
If the event is an upper endpoint event, and \( s \) is the line segment that starts at \( p \):

1. Search with \( p \) in \( T \), and insert \( s \)
2. If \( s \) intersects its left neighbor in \( T \), then determine the intersection point and insert it \( Q \)
3. If \( s \) intersects its right neighbor in \( T \), then determine the intersection point and insert it \( Q \)
If the event is a lower endpoint event, and $s$ is the line segment that ends at $p$:

1. Search with $p$ in $T$, and delete $s$.
2. Let $s_l$ and $s_r$ be the left and right neighbors of $s$ in $T$ (before deletion). If they intersect below the sweep line, then insert their intersection point as an event in $Q$. 

\[ \text{sweep line} \]

\[ p \]

\[ s \]
If the event is an intersection point event where $s$ and $s'$ intersect at $p$:
If the event is an intersection point event where $s$ and $s'$ intersect at $p$:

1. Exchange $s$ and $s'$ in $T$
2. ...
3. ...
4. ...
Event handling

If the event is an intersection point event where \( s \) and \( s' \) intersect at \( p \):

1. Exchange \( s \) and \( s' \) in \( T \)
2. If \( s' \) and its new left neighbor in \( T \) intersect below the sweep line, then insert this intersection point in \( Q \)

\[
\text{\ldots}
\]

\[
\text{\ldots}
\]
If the event is an **intersection point** event where $s$ and $s'$ intersect at $p$:

1. Exchange $s$ and $s'$ in $T$
2. If $s'$ and its new left neighbor in $T$ intersect below the sweep line, then insert this intersection point in $Q$
3. If $s$ and its new right neighbor in $T$ intersect below the sweep line, then insert this intersection point in $Q$
4. ...
If the event is an intersection point event where $s$ and $s'$ intersect at $p$:

1. Exchange $s$ and $s'$ in $T$
2. If $s'$ and its new left neighbor in $T$ intersect below the sweep line, then insert this intersection point in $Q$
3. If $s$ and its new right neighbor in $T$ intersect below the sweep line, then insert this intersection point in $Q$
4. Report the intersection point
Can it be that new horizontal neighbors already intersected above the sweep line?

Can it be that we insert a newly detected intersection point event, but it already occurs in $Q$?
Can it be that new horizontal neighbors already intersected above the sweep line?

Can it be that we insert a newly detected intersection point event, but it already occurs in $Q$?

Insert events only once!
Efficiency

How much time to handle an event?

At most one search in $T$ and/or one insertion, deletion, or swap

At most twice finding a neighbor in $T$

At most one deletion from and two insertions in $Q$

Since $T$ and $Q$ are balanced binary search trees, handling an event takes only $O(\log n)$ time
How many events?

- $2n$ for the upper and lower endpoints
- $k$ for the intersection points, if there are $k$ of them

In total: $O(n + k)$ events
Efficiency

Initialization takes $O(n \log n)$ time (to put all upper and lower endpoint events in $Q$).

Each of the $O(n + k)$ events takes $O(\log n)$ time.

The algorithm takes $O(n \log n + k \log n)$ time.

If $k = O(n)$, then this is $O(n \log n)$.

Note that if $k$ is really large, the brute force $O(n^2)$ time algorithm is more efficient.
Question: How much storage does the algorithm take?
Question: Given that the event list is a binary tree that may store $O(k) = O(n^2)$ events, is the efficiency in jeopardy?
Solution:
Only store intersection points of currently adjacent segments.
Degenerate cases

How do we deal with degenerate cases?

For two different events with the same $y$-coordinate, we treat them from left to right $\Rightarrow$ the “upper” endpoint of a horizontal line segment is its left endpoint.
Degenerate cases

How about multiply coinciding event points?

Let $U(p)$ and $L(p)$ be the line segments that have $p$ as upper and lower endpoint, and $C(p)$ the ones that contain $p$.

**Question**: How do we handle this multi-event?
For every sweep algorithm:

- Define the status
- Choose the status structure and the event list
- Figure out how events must be handled (with sketches!)
- To analyze, determine the number of events and how much time they take

Then deal with degeneracies