CG Lecture 2

Line segment intersection

- Intersecting two line segments
- Line sweep algorithm
- Convex polygon intersection
- Boolean operations on polygons
- Subdivision overlay algorithm

Family of intersection problems

1. **Line segment intersection**: given $n$ line segments, report their intersections efficiently.
   - Worst case: $k = n(n-1)/2 = O(n^2)$ intersections.
   - Optimal algorithm: $O(n \log n + k)$ time and $O(n)$ space.

2. **Polygon intersection**
   - Intersection of two simple polygons is not a simple polygon.
   - Boolean operations of union and difference very similar
   - Optimal algorithm: $O(n \log n + k)$ time, $O(n+m)$ for convex.

3. **Subdivision overlay**: given two planar subdivisions, compute their overlay. Generalization of 2.
Family of intersection problems (2)

4. **Line intersections and arrangements**: given \( n \) lines (half-spaces), compute their intersections and the regions they define efficiently.

5. **Face and polyhedra intersections**
   - Intersection of two polygonal (triangular) faces in space
   - Intersection of two (convex) polyhedra.

Applications: MANY!
- Basic operations in graphics.
- Engineering and VLSI design
- Non-geometric domains: databases, parallelization.

Intersection of two line segments

**Theorem**: Segments \( (p_1,p_2) \) and \( (p_3,p_4) \) intersect in their interior iff
- \( p_1 \) and \( p_2 \) on different sides of line \( p_3p_4 \)
- \( p_3 \) and \( p_4 \) on different sides of line \( p_1p_2 \)

Both conditions can be tested by computing the orientations of *four* triangles. Which ones?

Special cases:
Intersection point computation

**Question:** What is the meaning of other values of $s$ and $t$?

Solve (2D) linear vector equation for $t$ and $s$:

\[
(p_1, q_1) \rightarrow (p_2, q_2)
\]

Line segment intersections

**Problem:** Given $n$ line-segments in the plane, compute all their intersections.

**Variant:** Is there any pair of intersecting segments?

**Assume:** general position
- No line segment is vertical.
- No two segments are collinear.
- No three segments intersect at a common point.

**Naive algorithm:** Check each pair of segments for intersection. Time complexity: $\Theta(n^2)$. 
Line segment intersections

**Goal:**
Output-sensitive algorithm
- \( O(n \log n + k \log n) \) time
- \( O(n) \) space
- not optimal, but good start

Idea:
- Segments that are close together are candidates for intersection.
- Keep track of “closedness” changes.

Sweep line: idea

Segment adjacencies with respect to a horizontal (vertical) line change locally:
- \( s_i \) and \( s_j \) must be adjacent for the intersection to occur.
- \( (s_p, s_j) \) before intersection.
- \( (s_p, s_i) \) after intersection.

Keep track of segment adjacencies as vertical line moves from left to right.
Stop only at *event* points!
Adjacency theorem

**Theorem:** Just before an intersection occurs (infinitesimally-close to it), the two respective segments are adjacent to each other in the sweep-line status.

**Proof:**

**In practice:** Look ahead: whenever two line segments become adjacent along the sweep line, check for their intersection below the sweep line.

---

Sweep line algorithm

- An **event** is any endpoint or intersection point.
- Sweep a horizontal line from top to bottom.

**Maintain two data structures:**

- Event priority queue $Q$: sorted by $y$ coordinate.
- Sweep line status $L$: stores segments currently intersected by the sweep line, sorted by $y$ coordinate.

# of events $\leq n + k$

# of segments in $L \leq n$
Balanced binary tree

- Balanced binary tree sorted by $x$ coordinate
- Insertion, deletion, and neighbor operations in $O(\log n)$

Sweep line algorithm

Initialization:
- Put all segment endpoints in the event queue $Q$, sorted according to $y$ coordinates.
  - Time: $O(n \log n)$.
- Sweep line status is initially empty.

The algorithm proceeds by inserting, deleting, and handling discrete events from the queue $Q$ until it is empty. It also maintains the list $L$. 
Event handling – start of segment

*Event of type A*: Beginning of segment (upper endpoint)

- Locate segment position in the status.
- Insert segment into sweep line status.
- Test for intersection below the sweep line with the segments immediately to the left and right of it. Insert intersection point(s) (if found, and if not already in the queue) into the event queue.

Time complexity: $n$ events, $O(\log n)$ time for each $\rightarrow O(n \log n)$ total.

---

Event handling – end of segment

*Event of type B*: End of segment (lower endpoint)

- Locate segment position in the status.
- Delete segment from sweep line status.
- Test for intersection below the sweep line between the segments immediately to the left and right. Insert intersection point (if found, and if not already in the queue) into the event queue.

Time complexity: $n$ events, $O(\log n)$ time for each $\rightarrow O(n \log n)$ in total.
Event handling – intersection

*Event of type C:* Intersection point

- Report the point.
- Swap the two respective line segments in the sweep line status.
- For the new left and right segments – test each for intersection against the segment immediately adjacent the status line (if exists). Insert intersection point (if found, and if not already in the queue) into the event queue.

Time complexity: $k$ such events, $O(\log n)$ each $\rightarrow O(k \log n)$ in total.

Example
Complexity analysis

- Basic data structures:
  - Event queue: heap
  - Sweep line status: balanced binary tree
- Each heap/tree operation requires $O(\log n)$ time.
  
  \((\text{Why is } O(\log k) = O(\log n) ?)\)
- Total time complexity: $O((n+k) \log n)$.
  - If $k \approx n^2$ this is slightly worse than the naive algorithm.
  - When $k = o(n^2/\log n)$ then the sweep algorithm is faster.
- Total space complexity: $O(n+k)$.

**Question:** How can this be improved to $O(n)$?

- Note: There exists a better algorithm whose running time is $\Theta(n \log n + k)$ with $O(n+k)$ space complexity.

---

Handling degeneracies

- Horizontal segments with same $y$ coordinate → treat them in arbitrary order
  (i.e., from left to right $y$ coordinate)
- Overlapping segments → report them as special
- Multiple segment intersections at the same point
Handling degeneracies

Line segment intersection algorithm

HANDLE_EVENT_POINT(p)

1. Let \( U(p) \) be the set of segments whose upper endpoint is \( p \); these segments are stored with the event point \( p \). (For horizontal segments, the upper endpoint is by definition the left endpoint.)

2. Search in \( T \) for the set \( S(p) \) of all segments that contain \( p \); they are adjacent in \( T \). Let \( L(p) \subset S(p) \) be the set of segments whose lower endpoint is \( p \), and let \( C(p) \subset S(p) \) be the set of segments that contain \( p \) in their interior.

3. If \( L(p) \cup U(p) \cup C(p) \) contains more than one segment

4. then Report \( p \) as an intersection, together with \( L(p) \), \( U(p) \), and \( C(p) \).

5. Delete the segments in \( L(p) \cup C(p) \) from \( T \).

6. Insert the segments in \( U(p) \cup C(p) \) into \( T \). The order of the segments in \( T \) should correspond to the order in which they are intersected by a sweep line just below \( p \). If there is a horizontal segment, it comes last among all segments containing \( p \).

7. (* Deleting and re-inserting the segments of \( C(p) \) reverses their order. *)

8. if \( U(p) \cup C(p) = \emptyset \)

9. then Let \( s_l \) and \( s_r \) be the left and right neighbors of \( p \) in \( T \).

10. FIND_NEW_EVENT(\( s_l \), \( s_r \), \( p \))

11. else Let \( s' \) be the leftmost segment of \( U(p) \cup C(p) \) in \( T \).

12. Let \( s_l \) be the left neighbor of \( s' \) in \( T \).

13. FIND_NEW_EVENT(\( s_l \), \( s_r' \), \( p \))

14. Let \( s'' \) be the rightmost segment of \( U(p) \cup C(p) \) in \( T \).

15. Let \( s_r \) be the right neighbor of \( s'' \) in \( T \).

16. FIND_NEW_EVENT(\( s'' \), \( s_r \), \( p \))
Intersection of convex polygons

**Input:** Two convex polygons

\[ |A| = n \text{ and } |B| = m. \]

**Output:** one convex polygon

\[ |A \cap B| \leq n + m \text{ in optimal } O(n+m) \text{ time and space.} \]

Observations:

- alternation of \( A \) and \( B \) boundary pieces.
- alternation at intersection points
- order of vertices is kept.
- at each intersection, choose leftmost segment.

\[
A = \{1,2,3,4,5\} \quad B = \{6,7,8,9\} \quad I = \{10,11\} \quad A \cap B = \{6,7,10,8,9,11\}
\]

---

Code for convex polygon intersection

**Algorithm:** INTERSECTION OF CONVEX POLYGONS

Choose \( A \) and \( B \) arbitrarily.

repeat

if \( A \) intersects \( B \) then

Check for termination.

Update an inside flag.

Advance either \( A \) or \( B \), depending on geometric conditions.

until iterations \( > 2(n+m) \)

Handle \( P \cap Q = \emptyset \) and \( P \subseteq Q \) and \( P \supseteq Q \) cases.

<table>
<thead>
<tr>
<th>( A \times B )</th>
<th>Halfplane condition</th>
<th>Advance rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt; 0</td>
<td>( b \in H(A) )</td>
<td>( A )</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>( b \notin H(A) )</td>
<td>( B )</td>
</tr>
<tr>
<td>&lt; 0</td>
<td>( a \in H(B) )</td>
<td>( B )</td>
</tr>
<tr>
<td>&lt; 0</td>
<td>( a \notin H(B) )</td>
<td>( A )</td>
</tr>
</tbody>
</table>
Advance rules

Do not advance on the boundary of either $A$ or $B$ whose current edge may contain a yet-to-be-found intersection.

Example of convex intersection trace
Binary operations on polygons

**Operations**: intersection, union, difference.

Observations:
- Operations on simple connected polygons can lead to holes and disconnected pieces.
- Operations are similar, once the intersection points are found. Only the boundary reconstruction rules change.
- Maximum number of pieces/holes is $O(nm)$

Maximum number of pieces/holes

Each piece has $O(n)$ slabs.
The intersection has $O(n^2)$ pieces.
Intersection of polygons

\[ A = \{1,2,3,4,5\} \]
\[ B = \{6,7,8\} \]
\[ I = \{9,10,11,12,13,14\} \]
\[ A \cap B = \{9,10,11,12\} \cup \{13,14\} \]

**Boundary rule:**
Follow segment with **both** interiors to the left

---

Union of polygons

\[ A = \{1,2,3,4,5\} \]
\[ B = \{6,7,8\} \]
\[ I = \{9,10,11,12,13,14\} \]
\[ A \cup B = ((6,10,1,11,8,7,14,4,5),(9,2,13)) \]

**Boundary rule:**
Follow segment with either interior to the left
Difference of polygons

\[ A = \{1, 2, 3, 4, 5\} \]
\[ B = \{6, 7, 8\} \]
\[ I = \{9, 10, 11, 12, 13, 14\} \]
\[ A - B = \{9, 12, 2, 13, 14, 4, 5\} \]

Boundary rule:
Follow segment with A's interior to the left

Subdivision overlay intersection

Algorithm for a more general class of shapes subdivision overlays, a subdivision of faces in the plane represented by a planar graph.

Idea: use the sweep line technique to detect intersections and build the new overlay by adding vertices and updating pointers.

- Start with an overlay consisting of two disconnected components, \( S_1 \) and \( S_2 \).
- Add points and links as events occur.
Subdivision overlay representation

A face is the maximal connected subset of the plane that does not contain in its interior a point on an edge or vertex. The incident edges and vertices form its boundary.

Representation as a doubly-connected edge list. Directed edges are half-edges
- **Vertex**: coordinates, incident edge
- **Face**: inner and outer components
- **Half-edge**: origin, twin edge, next and previous edges, incident face.

Example of subdivision
Event handling – sketch

- Use a sweep line, stopping at vertices. Keep a doubly connected list $D$, initially containing edges of $S_1$ and $S_2$.
- If the event involves only edges from either $S_1$ or $S_2$, there is no need to update $D$.
- Otherwise, locally update the data structures. (pointer manipulations – tedious details omitted).
- All operations take $O(m)$, where $m$ is the cardinality of the vertex.
- At the end of the sweep line, the vertex and edges have been properly handled in the data structure.
- The face information (old and new faces) must now be updated to create $O(S_1, S_2)$.

Event handling – example

![Diagram](image)
Boundary and face tracing

- Boundary cycles are extracted directly from $D$ by following pointers $\rightarrow$ faces are directly created and extracted.
- Missing information: is the boundary an outer or an inner face? Which face does it belong to?
- Inner or outer face: find the angle of the leftmost vertex. If it is greater than $\pi$, then it is an outer face, otherwise it is an outer face.

Face handling – connected components

holes

outer boundaries
Subdivision intersection algorithm (1)

1. Initialize $D$ with $S_1$ and $S_2$
2. Compute all intersections between edges from $S_1$ and edges of $S_2$ with line-sweep. In addition to the actions on $T$ and $Q$, do
   1. Update $D$ when the event involves edges from both $S_1$ and $S_2$.
   2. Store first half-edge to the left of the event point at corresponding vertex in $D$ (used later for face computation).
3. Find boundary cycles.
4. Compute connected components of graph $G$ whose nodes are boundary cycles and whose arcs connect each hole cycle to the cycle left of its leftmost vertex.

Subdivision intersection algorithm (2)

5. For each connected component in $G$ do:
   Let $C$ be the unique outer boundary cycle in the component, and let $f$ be the face bounded by the cycle.
   a. Create a face record for $f$
   b. Set $OuterComponent(f)$ to a half-edge of $C$
   c. Construct the list $InnerComponents(f)$ with pointers to one half-edge in each hole cycle in the component.
   d. Make the $IncidentFace()$ pointers of all half-edges in the cycles point to the face record of $f$.

Theorem: The overlay of two planar subdivisions $S_1$ and $S_2$ of complexity $n$ can be computed in $O(n \log n + k \log n)$ where $k$ is the complexity of the resulting overlay.
Line segment intersection algorithms

- Naïve $O(n^2)$
- Bentley-Ottman, 1988: $O((n+k) \log n)$
- Edelsbrunner-Chazelle, 1990: $O(n \log n + k)$
  requires $O(n \log n)$ space, complex.
- Clarkson-Shor, Mumuley, 1992:
  $O(n \log n + k)$ with randomized $O(n)$ space
- Balaban, 1995: $O(n \log n + k)$
  with $O(n)$ space, optimal deterministic.
  Solved a long open problem!