Line Segment Intersection

Thematic Map Overlay

Min-Te Sun, Ph.D.

Map Overlay

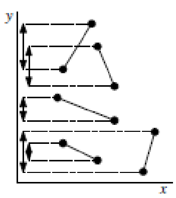
- Two of Classical Computational Geometry Problems
  - Line Segment Intersection
    - Finding bridges from rivers and road networks
  - Computing the Overlay of Two Subdivisions
    - Locating the region that holds two specific attributes (e.g., the region that produces apples with average temperature 20 degree)

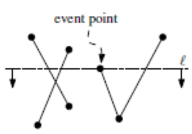
Line Segment Intersection

- Problem statement: Given a set S of n closed segments in the plane, report all intersection points among the segments in S.
- Intuition – Compute the intersection of all pairs of line segments
  - Time Complexity Ω (n^2)
  - Good enough?

Output-Sensitive Algorithm

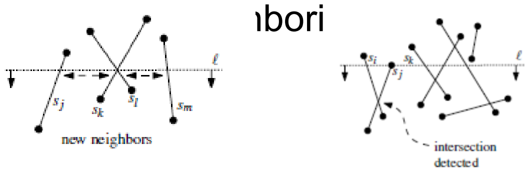
- Avoid testing all pairs for intersection
  - Segments whose y-intervals not overlap won't intersect
  - Segments whose x-intervals not overlap won't intersect
  - Result => plane sweep algorithm





What DS Do We Need?

- Event Points (B. BST Q)
  - End points of segments
  - Intersections
  - Order?
- Status "Active" Line Segments (B. BST T)
  - Only n ıbori

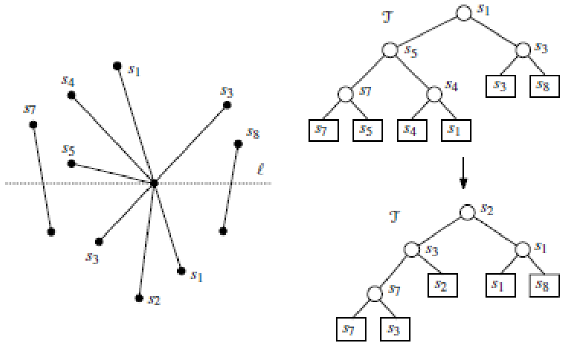


Algorithm

Algorithm FINDINTERSECTIONS(S)
Input. A set S of line segments in the plane.
Output. The set of intersection points among the segments in S, with for each intersection point the segments that contain it.

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

3 Segments meet at 1 Point?



```
if L(p) ∪ U(p) ∪ C(p) contains more than one segment
   then Report p as an intersection, together with L(p), U(p), and C(p).
Delete the segments in L(p) ∪ C(p) from T.
Insert the segments in U(p) ∪ 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.
(* Deleting and re-inserting the segments of C(p) reverses their order. *)
```

Running Time?

- Our algorithm takes O(nlogn + Ilogn), where I is the # of intersections in S
  - Each operation on T takes O(logn)
  - The # of operations is linear to

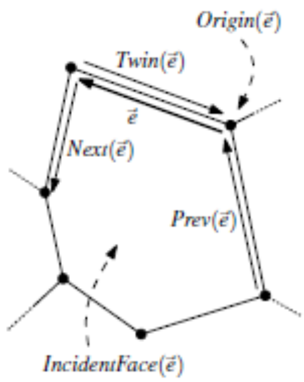
    m(p) = card(L(p) ∪U(p) ∪C(p))
  - m(p) = O(n + I)
    - n_v <= 2n + I
    - Every face of the planar graph is bounded by at least 3 edges and an edge can bound at most two different faces => n_f <= $2n_e/3$
    - Euler's formula => $n_v - n_e + n_f >= 2$
    - $2 <= (2n + I) - n_e + 2n_e/3 = (2n + I) - n_e/3$
    - n_e <= 6n + 3I - 6, and m <= 12n + 6I - 12

How to Represent a Subdivision?

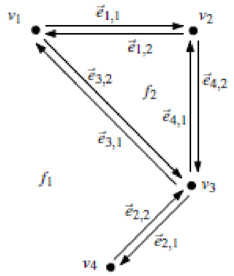
- Operations needed:
  - walk around the boundary of a face
  - access one face from an adjacent one
  - and visit all the edges around a vertex
- A doubly-connected edge list consists of 3 collections of records
  - Vertices
  - Faces
  - Half-edges

Doubly-Connected Edge List

- Vertex record v stores the coordinates(v) and a pointer IncidentEdge(v) to an arbitrary half-edge that has v as its origin
- Face record f stores pointer OuterComponent(f), and a list InnerComponents(f)
- Half-edge record e stores pointer Origin(e), pointer Twin(e), pointer IncidentFace(e), Next(e), and Prev(e)



Example of Doubly-Connected Edge List



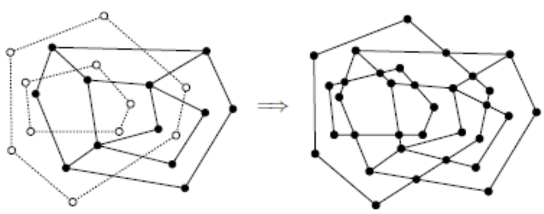
Vertex	Coordinates	IncidentEdge
v_1	$(0,4)$	$\vec{e}_{1,1}$
v_2	$(2,4)$	$\vec{e}_{4,2}$
v_3	$(2,2)$	$\vec{e}_{2,1}$
v_4	$(1,1)$	$\vec{e}_{2,2}$

Face	OuterComponent	InnerComponents
f_1	nil	$\vec{e}_{1,1}$
f_2	$\vec{e}_{4,1}$	nil

Half-edge	Origin	Twin	IncidentFace	Next	Prev
$\vec{e}_{1,1}$	v_1	$\vec{e}_{1,2}$	f_1	$\vec{e}_{4,2}$	$\vec{e}_{3,1}$
$\vec{e}_{1,2}$	v_2	$\vec{e}_{1,1}$	f_2	$\vec{e}_{3,2}$	$\vec{e}_{4,1}$
$\vec{e}_{2,1}$	v_3	$\vec{e}_{2,2}$	f_1	$\vec{e}_{2,2}$	$\vec{e}_{4,2}$
$\vec{e}_{2,2}$	v_4	$\vec{e}_{2,1}$	f_1	$\vec{e}_{3,1}$	$\vec{e}_{2,1}$
$\vec{e}_{3,1}$	v_3	$\vec{e}_{3,2}$	f_1	$\vec{e}_{1,1}$	$\vec{e}_{2,2}$
$\vec{e}_{3,2}$	v_1	$\vec{e}_{3,1}$	f_2	$\vec{e}_{4,1}$	$\vec{e}_{1,2}$
$\vec{e}_{4,1}$	v_3	$\vec{e}_{4,2}$	f_2	$\vec{e}_{1,2}$	$\vec{e}_{3,2}$
$\vec{e}_{4,2}$	v_2	$\vec{e}_{4,1}$	f_1	$\vec{e}_{2,1}$	$\vec{e}_{1,1}$

Computing the Overlay of Two Subdivisions

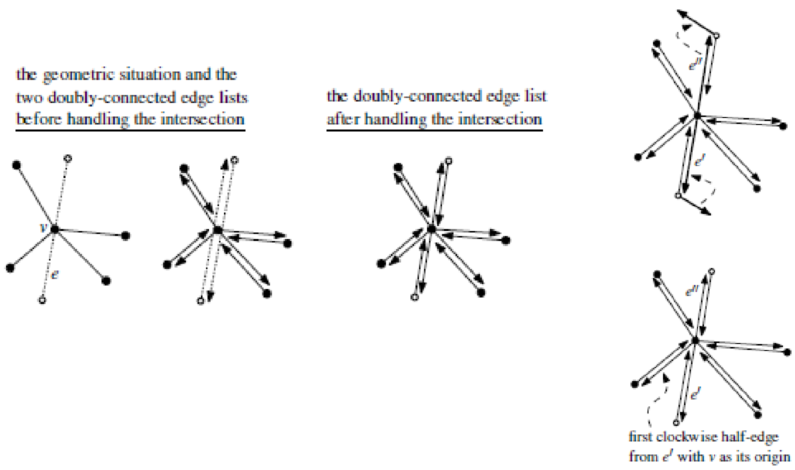
- Problem Statement: Given two subdivisions S1 and S2 (two doubly-connected edge lists), we want to compute a doubly-connected edge list for O(S1, S2)



A Thought Before Algorithm Discussion

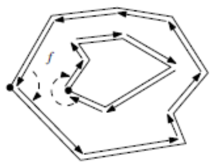
- The problem of overlay of two subdivisions is similar to line segment intersection
  - Each half-edge record will become one or several half-edge records after overlay depending on the # of intersections
  - Treat half-edges as line segments and use line segment intersection algorithm to obtain new points that "break" half-edges into shorter half-edges
    - The vertex and half-edge records can be adjusted accordingly when the algorithm is executed
  - The face record and incident face of each half-edge will be computed later

Half-edge Record Adjustment Example



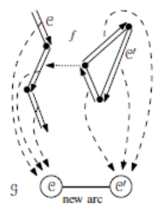
Face Records?

- How many face records?
  - # of outer boundary + 1 (unbounded one)
- How to determine a cycle is a outer boundary?
  - Check the angle of the half-edges between the (lowest) leftmost point in the cycle
    - If < 180^o => Outer boundary, hole otherwise

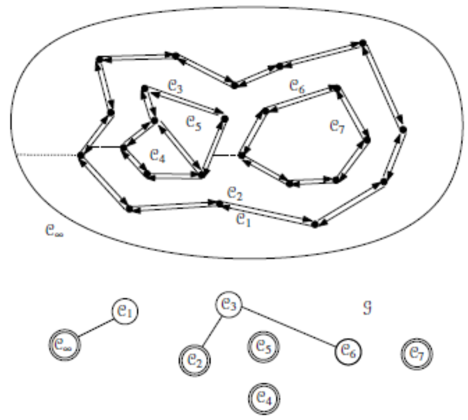


Association Between Outer Boundary and Holes

- How to determine if a hole is inside a outer boundary
  - Draw a dual graph G, create one node for each cycle (one additional node for unbounded boundary) => Draw an arc between two cycles if and only if one of the cycles is the boundary of a hole and the other cycle has a half-edge immediately to the left of the leftmost vertex of that hole cycle
- Each connected component of the graph G corresponds exactly to the set of cycles incident to one face

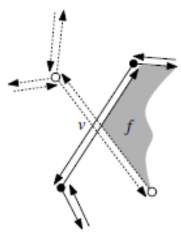


Example of Boundary Association



Labeling New Faces

- Each new face needs to know which old face in S_1 and which old face in S_2 contain it
  - Because we need to aggregate the info from two subdivisions into the new one
- How to check which face of S_1 (and S_2) contains a face adjacent to vertex v
  - If v is the intersection of e_1 from S_1 and e_2 from S2, we can check the IncidentFace of e_1 and e_2
  - If v is a vertex of S_1, we need to find the face of S_2 which contains v
    - This requires another plane-sweep procedure



Map Overlay Algorithm

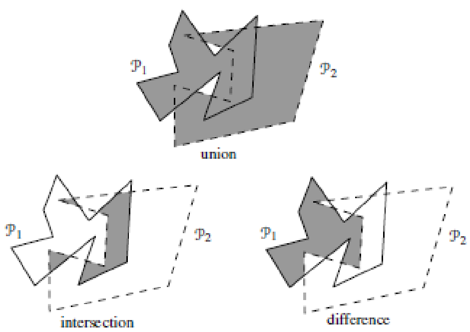
Algorithm MAPOVERLAY($\mathcal{S}_1, \mathcal{S}_2$)
Input. Two planar subdivisions $\mathcal{S}_1$ and $\mathcal{S}_2$ stored in doubly-connected edge lists.
Output. The overlay of $\mathcal{S}_1$ and $\mathcal{S}_2$ stored in a doubly-connected edge list $\mathcal{D}$.

1. Copy the doubly-connected edge lists for $\mathcal{S}_1$ and $\mathcal{S}_2$ to a new doubly-connected edge list $\mathcal{D}$.
2. Compute all intersections between edges from $\mathcal{S}_1$ and $\mathcal{S}_2$ with the plane sweep algorithm of Section 2.1. In addition to the actions on $\mathcal{T}$ and $\mathcal{Q}$ required at the event points, do the following:
   - Update $\mathcal{D}$ as explained above if the event involves edges of both $\mathcal{S}_1$ and $\mathcal{S}_2$. (This was explained for the case where an edge of $\mathcal{S}_1$ passes through a vertex of $\mathcal{S}_2$.)
   - Store the half-edge immediately to the left of the event point at the vertex in $\mathcal{D}$ representing it.
3. (* Now $\mathcal{D}$ is the doubly-connected edge list for $\mathcal{O}(\mathcal{S}_1, \mathcal{S}_2)$, except that the information about the faces has not been computed yet. *)
4. Determine the boundary cycles in $\mathcal{O}(\mathcal{S}_1, \mathcal{S}_2)$ by traversing $\mathcal{D}$.
5. Construct the graph $\mathcal{G}$ whose nodes correspond to boundary cycles and whose arcs connect each hole cycle to the cycle to the left of its leftmost vertex, and compute its connected components. (The information to determine the arcs of $\mathcal{G}$ has been computed in line 2, second item.)
6. **for** each connected component in $\mathcal{G}$
7.   **do** Let $\mathcal{C}$ be the unique outer boundary cycle in the component and let f denote the face bounded by the cycle. Create a face record for f, set *OuterComponent*(f) to some half-edge of $\mathcal{C}$, and construct the list *InnerComponents*(f) consisting of pointers to one half-edge in each hole cycle in the component. Let the *IncidentFace*() pointers of all half-edges in the cycles point to the face record of f.
8. Label each face of $\mathcal{O}(\mathcal{S}_1, \mathcal{S}_2)$ with the names of the faces of $\mathcal{S}_1$ and $\mathcal{S}_2$ containing it, as explained above.

Theorem: Let S_1 be a planar subdivision of complexity n_1, let S_2 be a subdivision of complexity n_2, and let $n = n_1 + n_2$. The overlay of S_1 and S_2 can be constructed in $O(n\log n + k \log n)$ time, where k is the complexity of the overlay.

Boolean Operations

- Map overlay algorithm can be used to compute union, intersection, and difference of two polygons P_1 and P_2



Homework Assignment 1

Page 15

- 1.1
- 1.3

Page 41 ~ 42

- 2.1
- 2.5
- 2.6