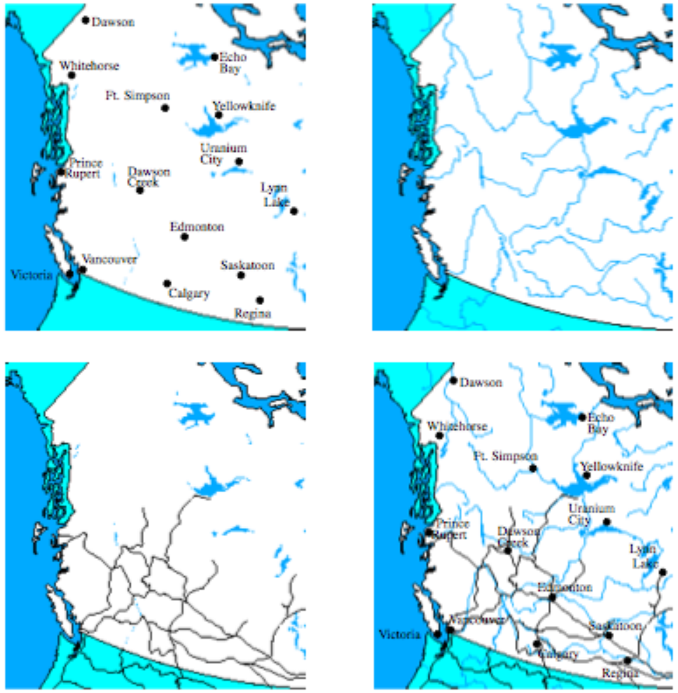
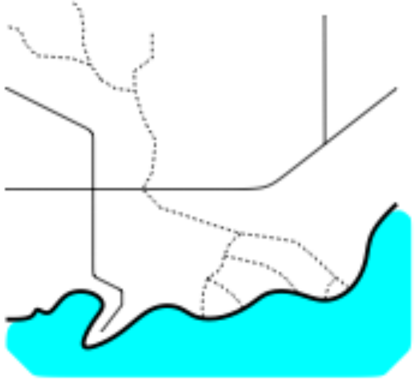
Thematic Map Overlay



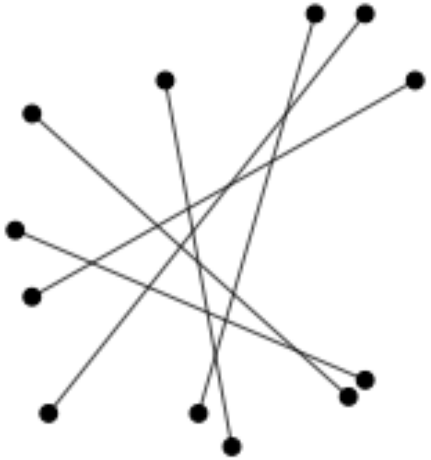
Computational Geometry
2 : Line Segment Intersection
pages 19-41



Compute all intersections among the segments !

Brute force algorithm :

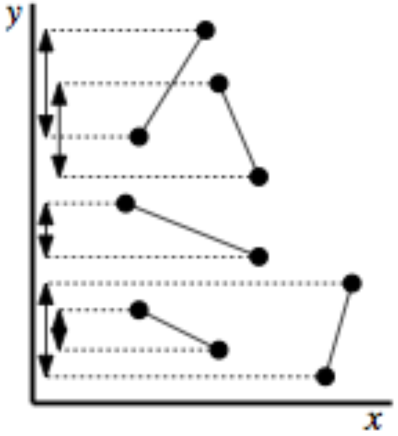
Simply take each pair of segments, compute whether they intersect, and, if so, report their intersection point.



The brute-force algorithm clearly requires O(n^2) time.

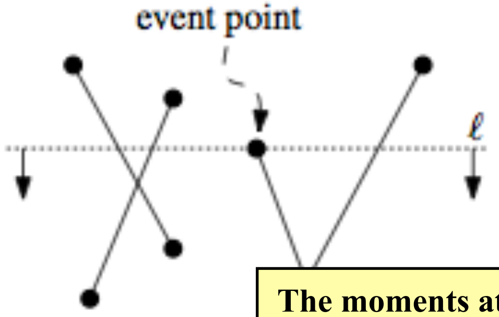
In a sense this is optimal here ! When each pair of segments intersects, any algorithm must take Ω(n^2) time, because it has to report all intersections.c

An output sensitive algorithm ?



How can we avoid testing all pairs of segments for intersection?

Plane sweep algorithm

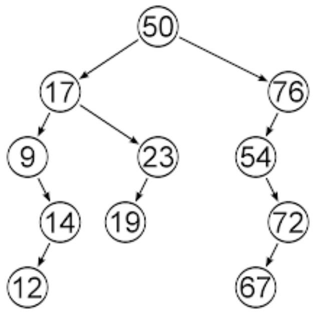


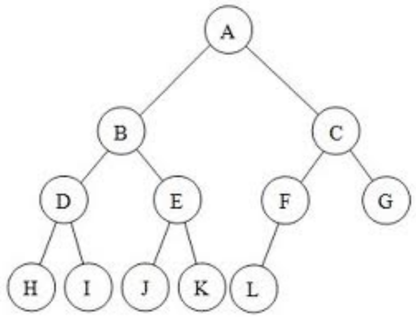
The moments at which the sweep line reaches an event point are the only moments when we do something:

- Update the status of the sweep line.
- Performs some intersection tests.

The status corresponds to the ordered sequence of segments intersecting the sweep line.

Event queue

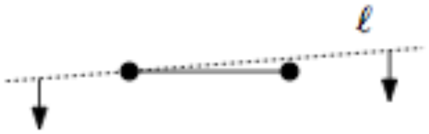




We store the eventsin a balanced binary search tree, ordered according to the altitude.

Fetching the next event and inserting an event take O(log m) time,
where m is the number of events in Q.

We do not use a heap to implement the event queue, because we have to be able to test whether a given event is already present in Q.



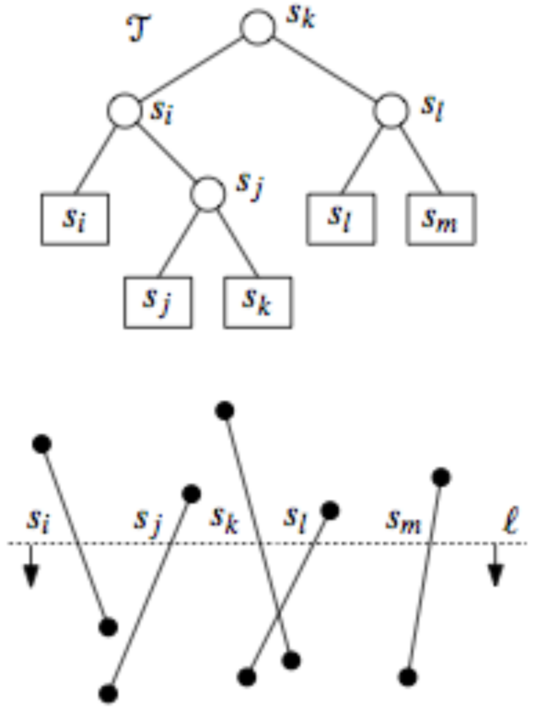
The event queue stores the events.
We denote the event queue by Q.

We need an operation that removes the next event that will occur from Q, and returns it so that it can be treated.

Status tree

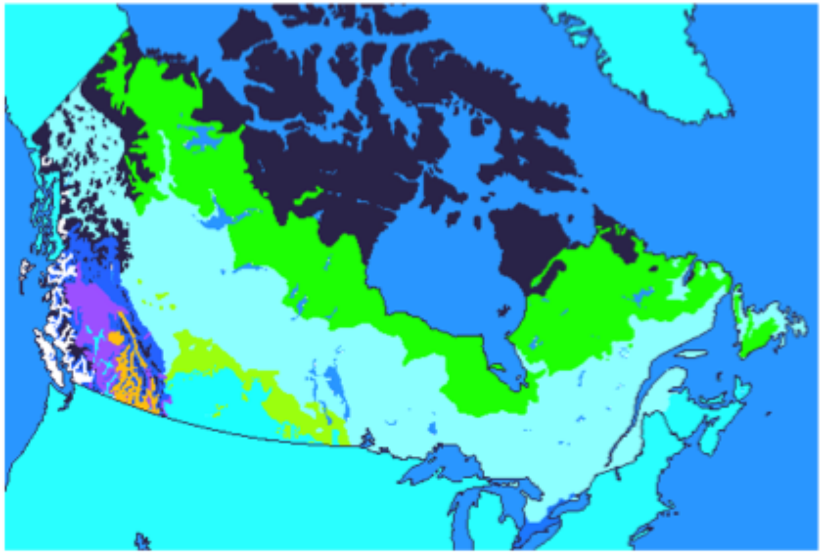
We also use a **balanced binary search tree** !

Each update and neighbor search operation takes O(log n) time.



The **status tree** stores the ordered sequence of segments intersecting the sweep line.
We denote the status tree by T.

We need an operation that removes the next event that will occur from Q, and returns it so that it can be treated.



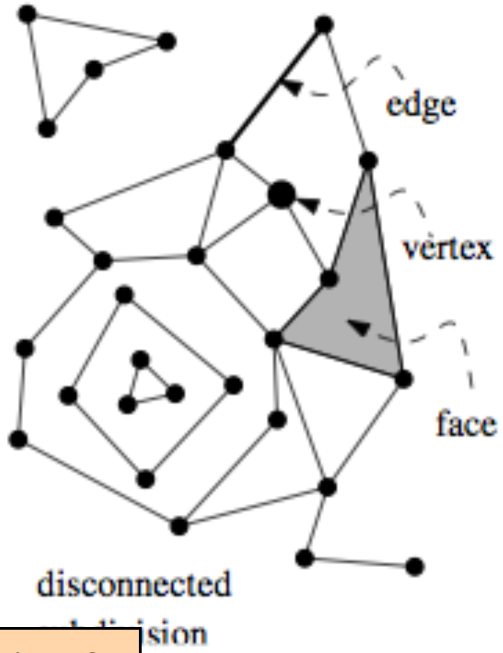
Subdivisions of the plane into labeled regions.

An algorithm for computing the overlay of two subdivisions

Overlay of two planar subdivisions

Computational Geometry
2 : Line Segment Intersection
pages 29-41

Planar subdivisions induced by planar embeddings of graph

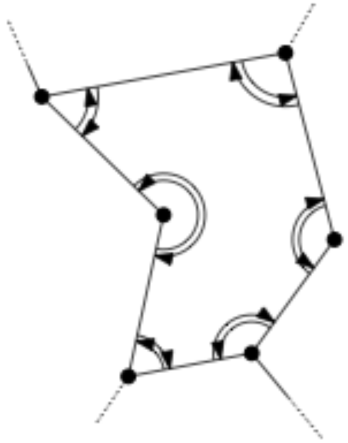


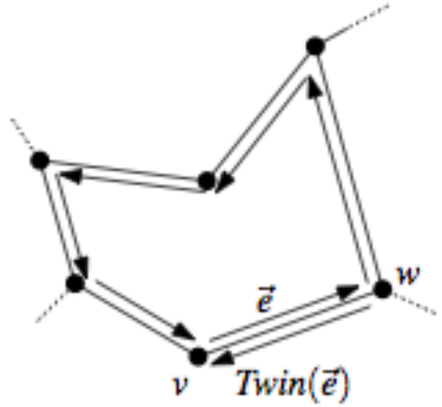
Storing a subdivision as a collection of line segments is not such a good idea.

Operations like reporting the boundary of a region would be rather complicated.

What should we require from a representation of a subdivision ?

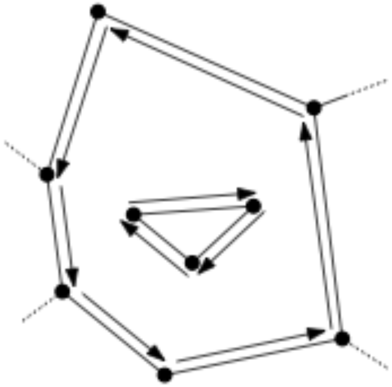
Walking around the boundary of a given face
Finding the faces containing a given vertex
Visiting all edges around a given vertex

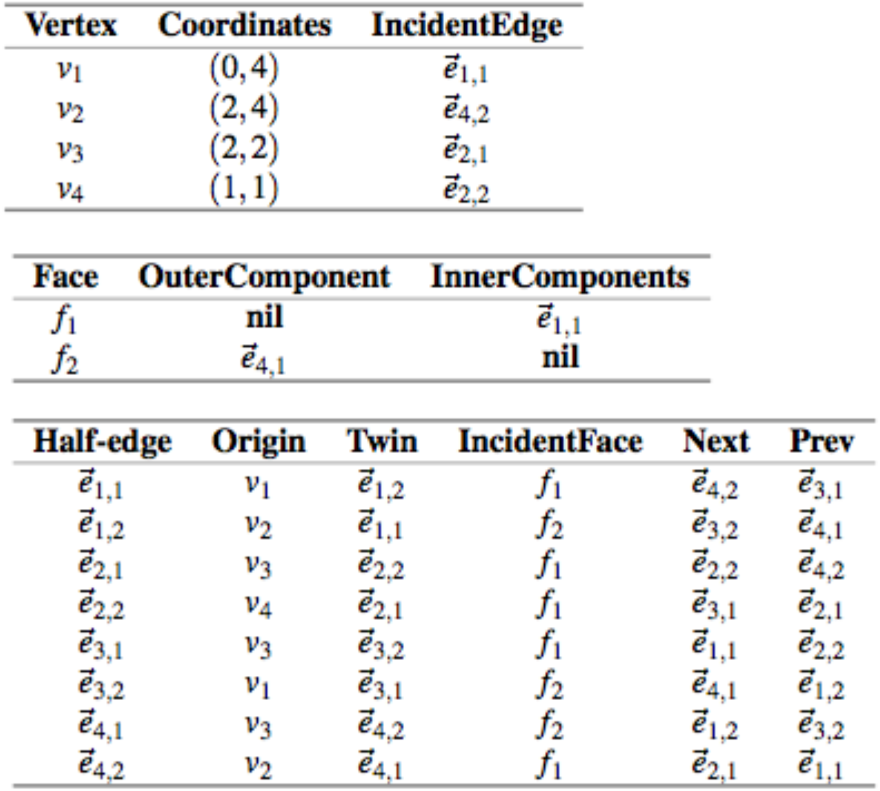




Half-edge with an origin and a destination
Twin of an half-edge

The doubly-connected edge list

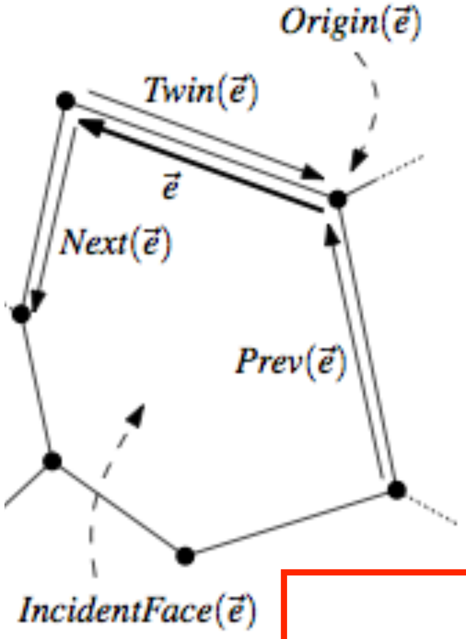




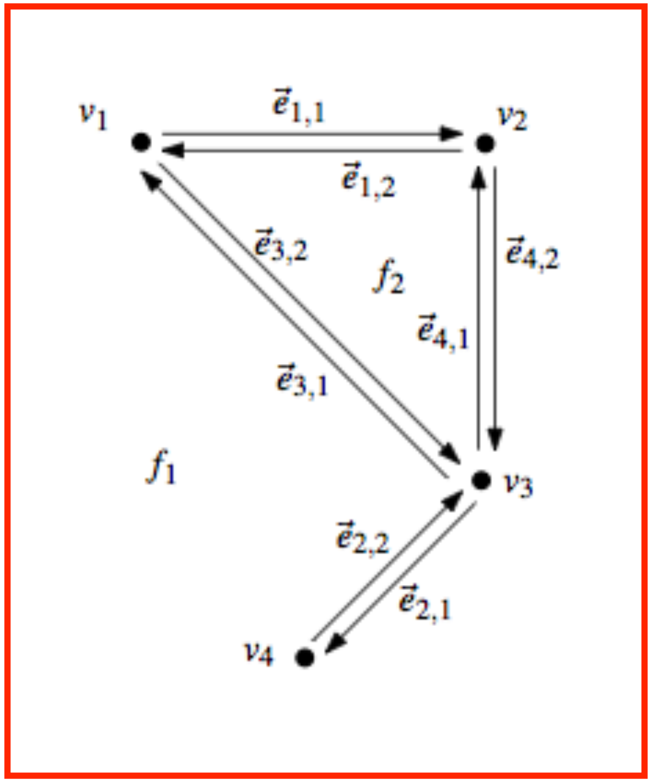
Vertex	Coordinates	IncidentEdge
v_1	$(0,4)$	$\vec{e}_{1,1}$
v_2	$(2,4)$	$\vec{e}_{4,2}$
v_3	$(2,2)$	$\vec{e}_{2,1}$
v_4	$(1,1)$	$\vec{e}_{2,2}$

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

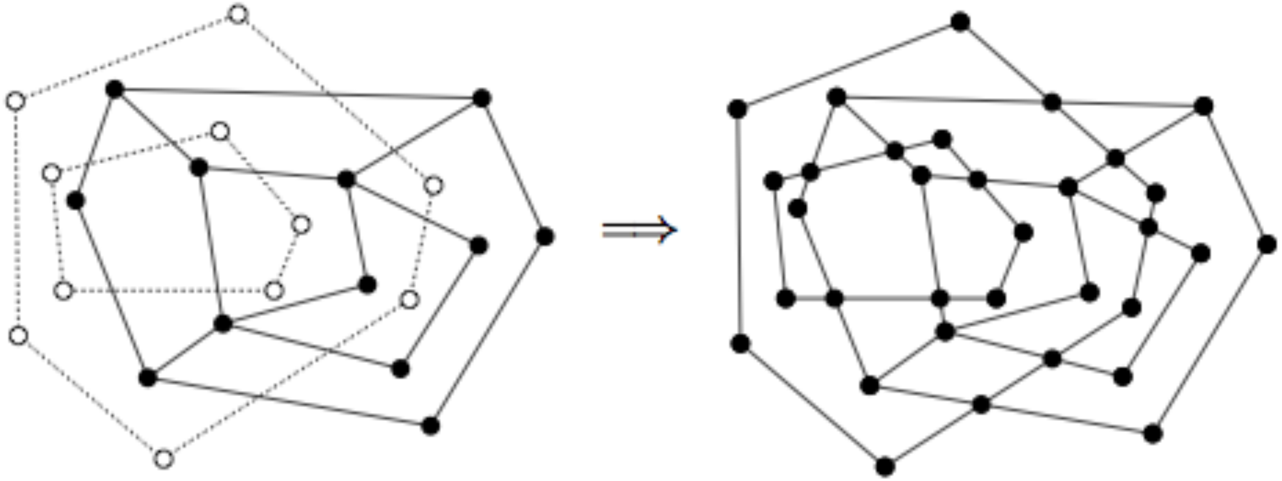
Half-edge	Origin	Twin	IncidentFace	Next	Prev
$\vec{e}_{1,1}$	v_1	$\vec{e}_{1,2}$	f_1	$\vec{e}_{4,2}$	$\vec{e}_{3,1}$
$\vec{e}_{1,2}$	v_2	$\vec{e}_{1,1}$	f_2	$\vec{e}_{3,2}$	$\vec{e}_{4,1}$
$\vec{e}_{2,1}$	v_3	$\vec{e}_{2,2}$	f_1	$\vec{e}_{2,2}$	$\vec{e}_{4,2}$
$\vec{e}_{2,2}$	v_4	$\vec{e}_{2,1}$	f_1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



The doubly-connected edge list



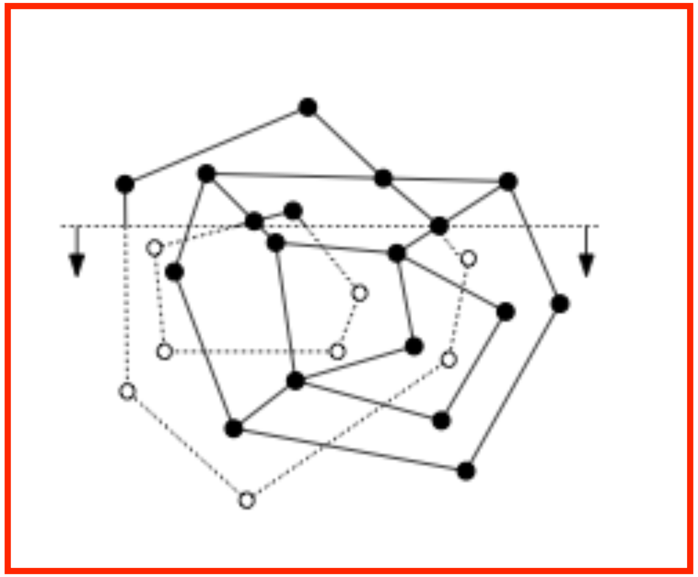
Computing the overlay of two subdivisions



How much information from the doubly-connected edge lists for S1 and S2 we can re-use in the doubly-connected edge list for O(S1,S2) ?

Consider the network of edges and vertices of S1.
This network is cut into pieces by the edges of S2.
These pieces are for a large part re-usable.
Only the edges that have been cut by the edges of S2 should be renewed.

Using the plane sweep algorithm



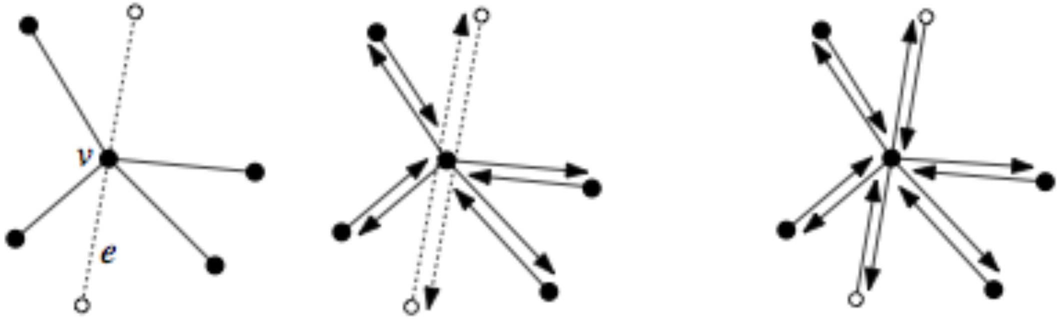
Event queue : Q

Segments intersecting the sweep line : T

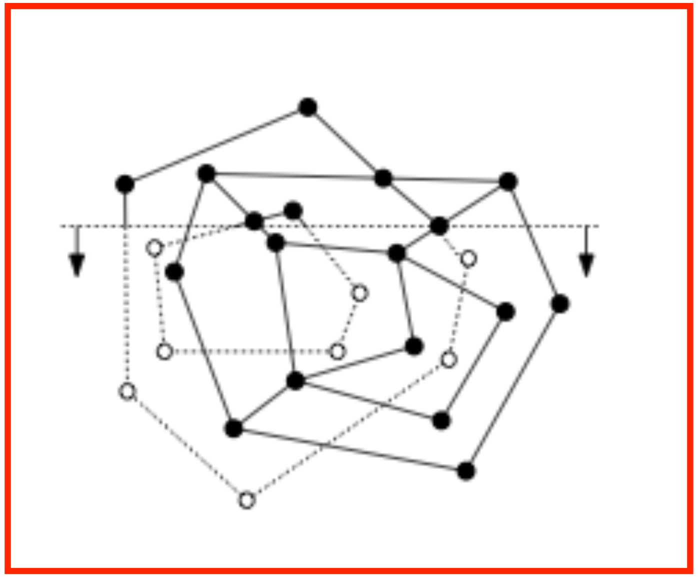
A doubly-connected edge list : D

the geometric situation and the two doubly-connected edge lists before handling the intersection

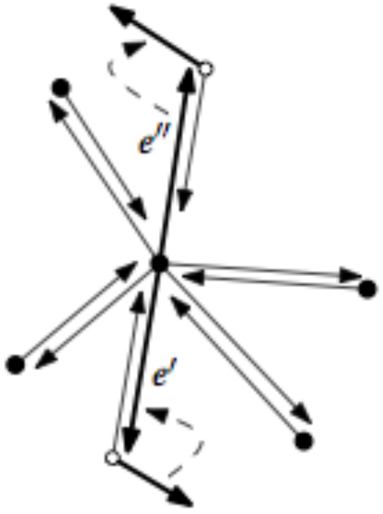
the doubly-connected edge list after handling the intersection

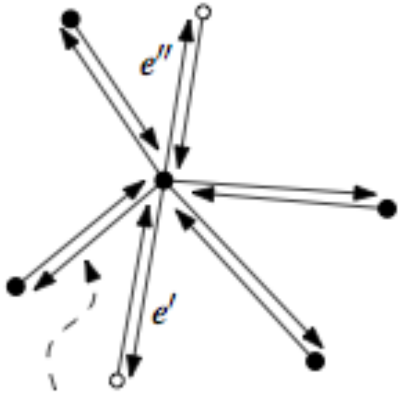


Using the plane sweep algorithm



Event queue : Q
Segments intersecting the sweep line : T
A doubly-connected edge list : D

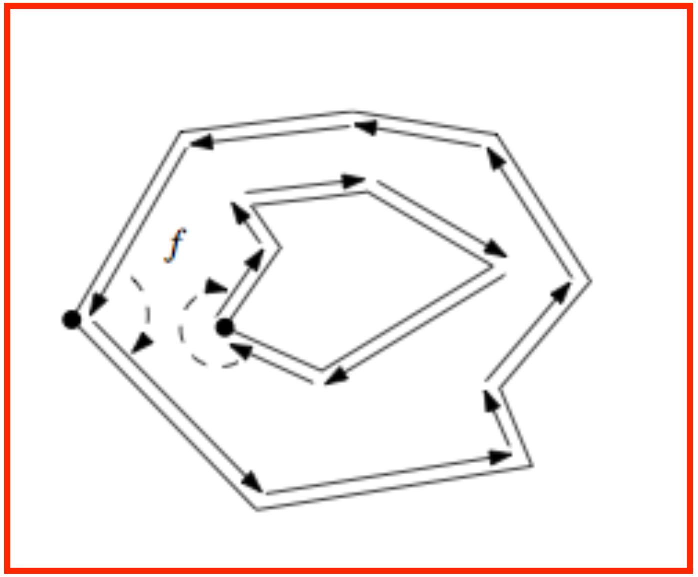


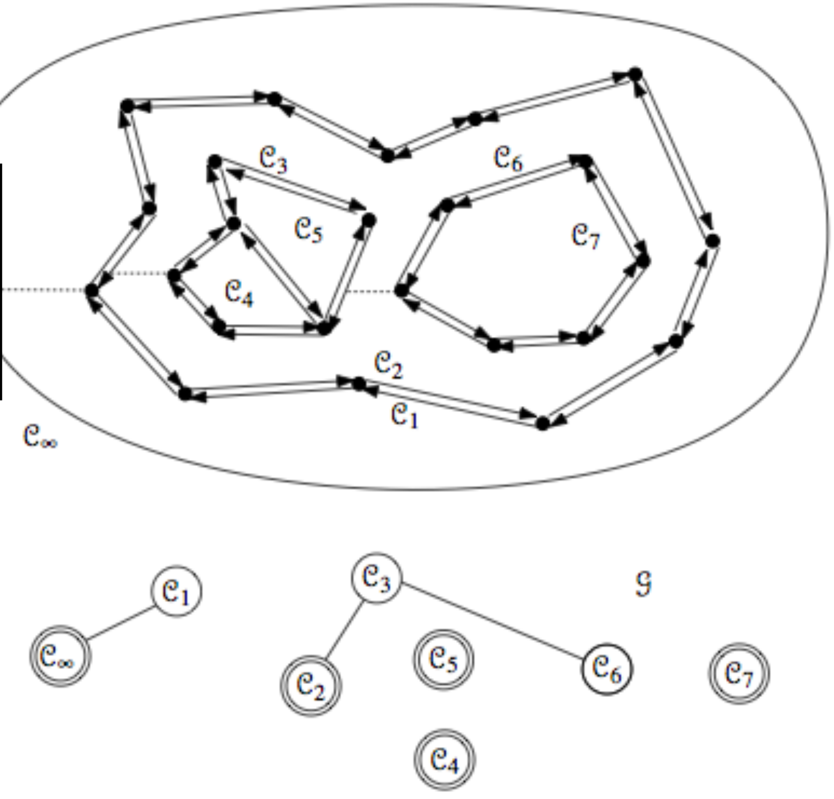


first clockwise half-edge
from e' with v as its origin

Which boundary cycles do bind the same face?

Event queue : Q
Segments intersecting the sweep line : T
A doubly-connected edge list : D
A graph : G

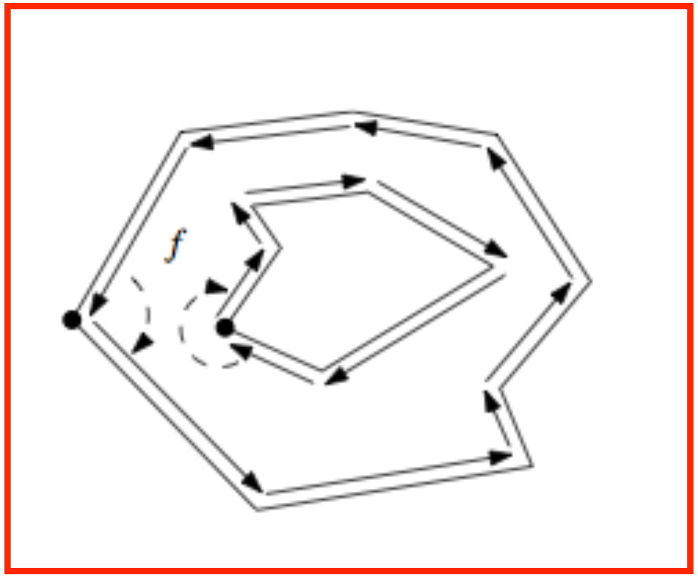


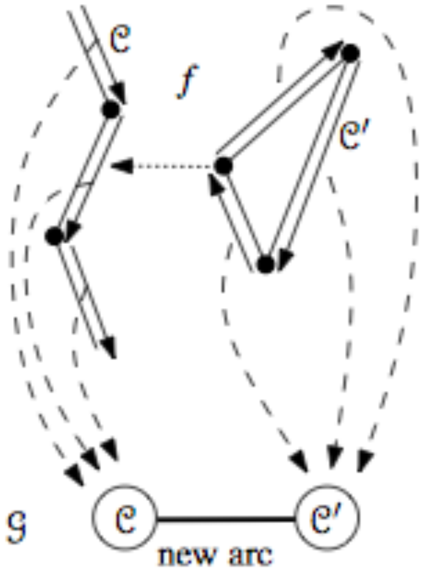


Lemma 2.5 *Each connected component of the graph $\mathcal{G}$ corresponds exactly to the set of cycles incident to one face.*

How can we construct the graph G ?

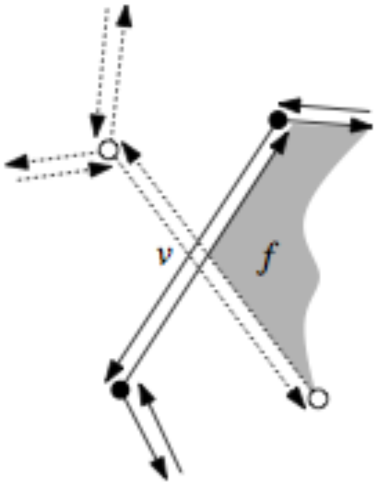
Event queue : Q
Segments intersecting the sweep line : T
A doubly-connected edge list : D
A graph : G

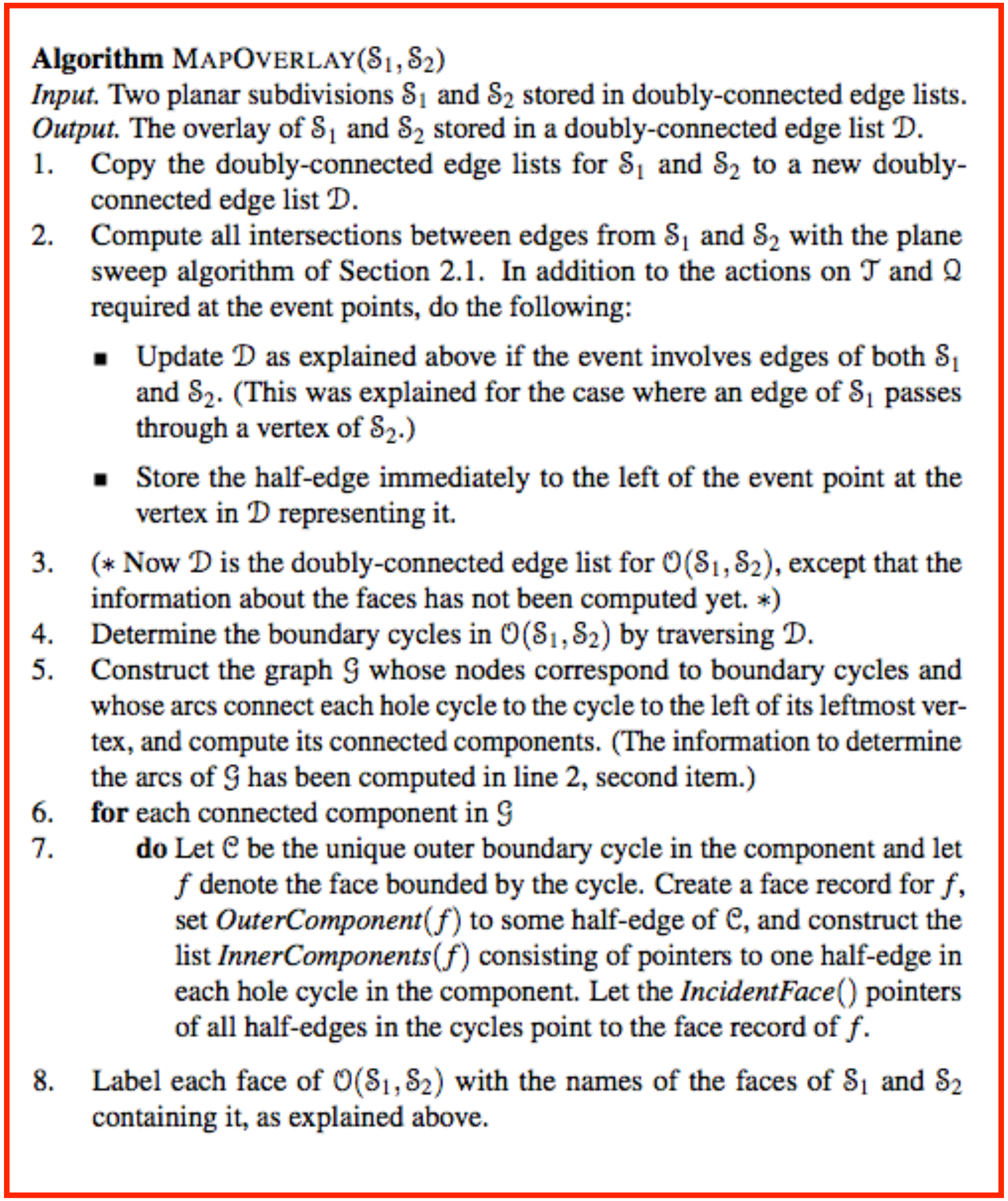




Lemma 2.5 *Each connected component of the graph $\mathcal{G}$ corresponds exactly to the set of cycles incident to one face.*

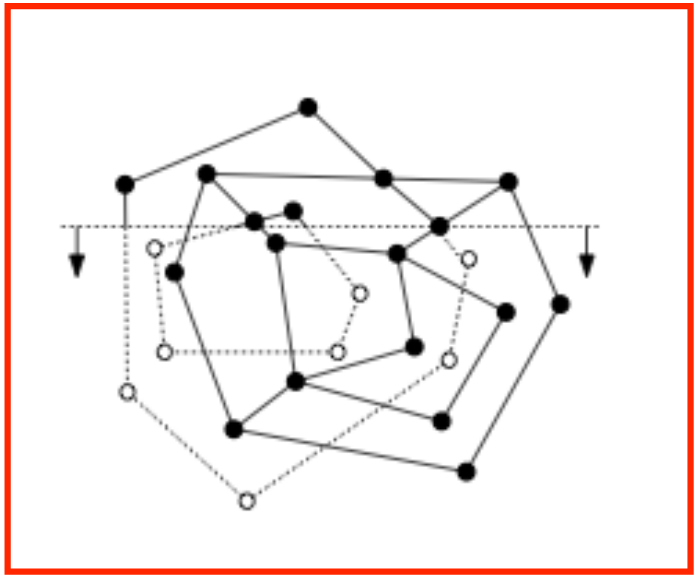
The algorithm





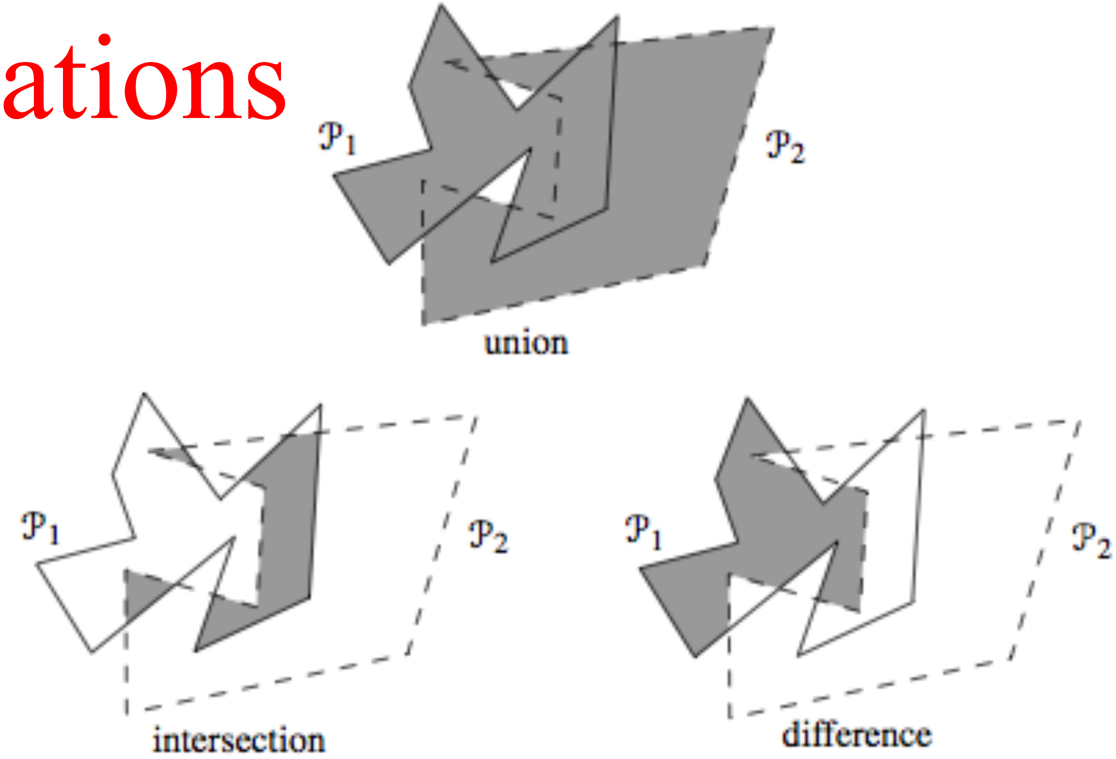
```
Algorithm MapOverlay($\mathcal{S}_1, \mathcal{S}_2$)
Input. Two planar subdivisions $\mathcal{S}_1$ and $\mathcal{S}_2$ stored in doubly-connected edge lists.
Output. The overlay of $\mathcal{S}_1$ and $\mathcal{S}_2$ stored in a doubly-connected edge list $\mathcal{D}$.
1.  Copy the doubly-connected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

Complexity of the sweep algorithm



Theorem 2.6 *Let $\mathcal{S}_1$ be a planar subdivision of complexity n_1, let $\mathcal{S}_2$ be a subdivision of complexity n_2, and let $n := n_1 + n_2$. The overlay of $\mathcal{S}_1$ and $\mathcal{S}_2$ can be constructed in $O(n \log n + k \log n)$ time, where k is the complexity of the overlay.*

Applications :-)
Boolean operations

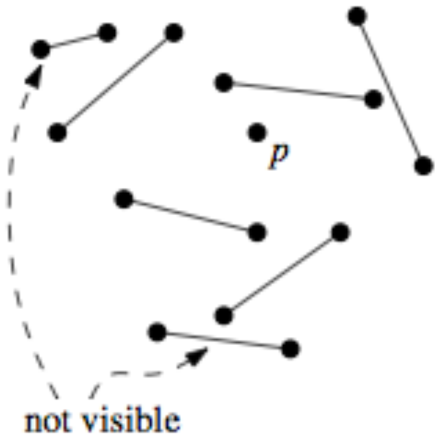


The line segment intersection problem is one of the most fundamental problems in computational geometry.

The O(n log n + k log n) solution presented was given by Bentley and Ottmann [47] in 1979.

Exercice 3

2.14 Let S be a set of n disjoint line segments in the plane, and let p be a point not on any of the line segments of S. We wish to determine all line segments of S that p can see, that is, all line segments of S that contain some point q so that the open segment $\overline{pq}$ doesn't intersect any line segment of S. Give an $O(n \log n)$ time algorithm for this problem that uses a rotating half-line with its endpoint at p.


p
not visible