Computational Geometry: Line Arrangements

Luis Peñaranda

IMPA

January 21-25, 2013

Course Outline

Lectures

1. Line Segment Arrangements in the Plane
2. Robustness in Geometric Computations
3. Arrangements in CGAL

Bibliography

Bibliography

- M. de Berg, M. van Kreveld, M. Overmars, O. Cheong. *Computational Geometry: Algorithms and Applications*. 3rd edition, Springer, 2008.
- F. Preparata and M. Shamos. *Computational Geometry: an Introduction*. Springer, 1985.
- H. Edelsbrunner. *Algorithms in Combinatorial Geometry*. Springer, 1987.
- J. Goodman and J. O'Rourke, eds. *Handbook of Discrete and Computational Geometry*. 2nd edition, Chapman & Hall, 2004.
- J.-D. Boissonnat and M. Teillaud, eds. *Effective Computational Geometry for Curves and Surfaces*. Springer, 2007.

Today's Lecture

Line Segment Arrangements in the Plane

- Line Segment Intersection
- The DCEL
- Overlays
- Application: Boolean Operations

Introduction

maps information

- cities
- rivers
- railroads
- regions
- ...

Introduction

maps information

- cities
- rivers
- railroads
- regions
- ...

the data

- are related
- are stored independently
- provide information when overlapped

First Approach

segment intersection

- a map is a set of n segments
- find all their intersections

First Approach

segment intersection

- a map is a set of n segments
- find all their intersections

in other words

- compute the *arrangement* of the n segments

First Approach

segment intersection

- a map is a set of n segments
- find all their intersections

in other words

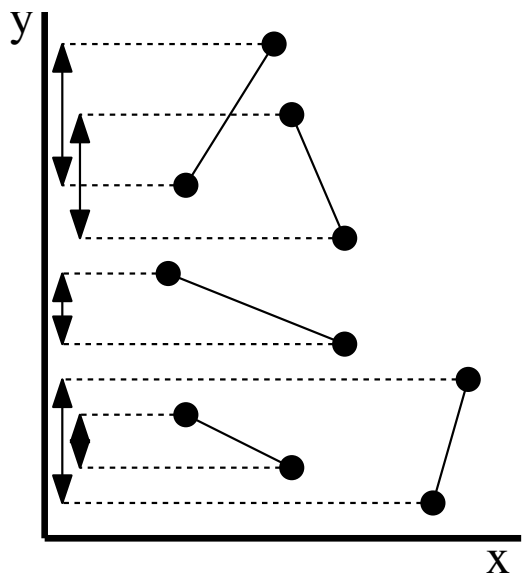
- compute the *arrangement* of the n segments

complexity

- brute force $O(n^2)$
- in some cases, $\Omega(n^2)$
- can we do better?

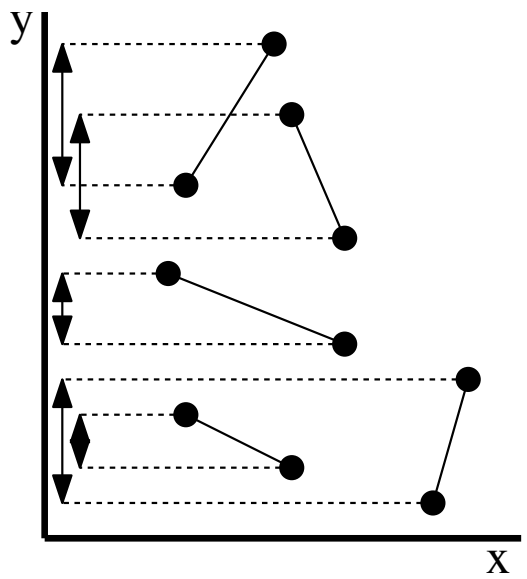
Definitions 1

y-interval



Definitions 1

y-interval



sweep line

an horizontal line that sweeps the plane, from top to bottom

More Definitions

status

segments intersected by the sweep line at a given moment

More Definitions

status

segments intersected by the sweep line at a given moment

event points

the points on which an update of the status is required

Algorithm Outline

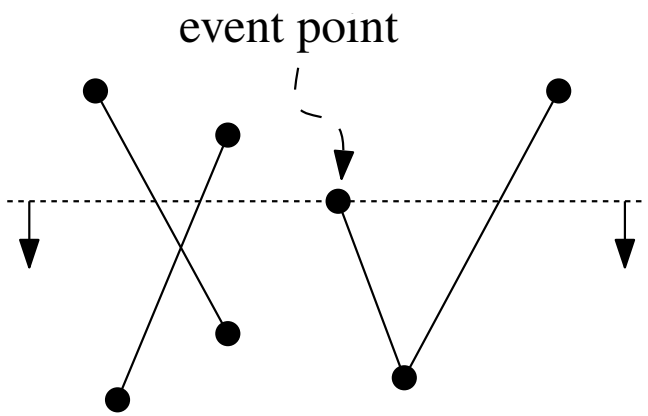
Sweep-line Algorithm

- order segments intersecting the sweep from left to right

Algorithm Outline

Sweep-line Algorithm

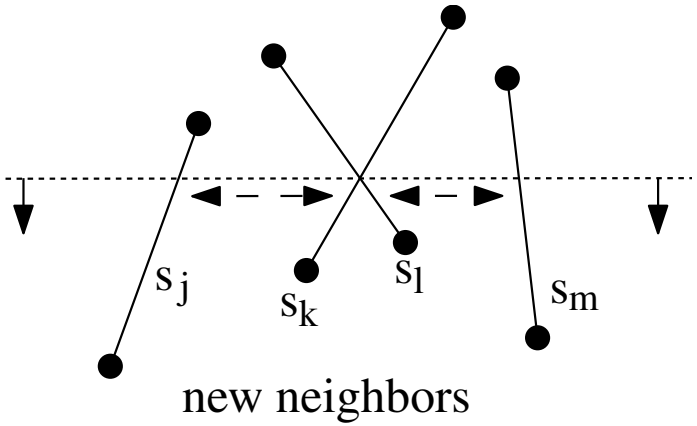
- order segments intersecting the sweep from left to right
- test segments only when they are adjacent



Algorithm Outline

Sweep-line Algorithm

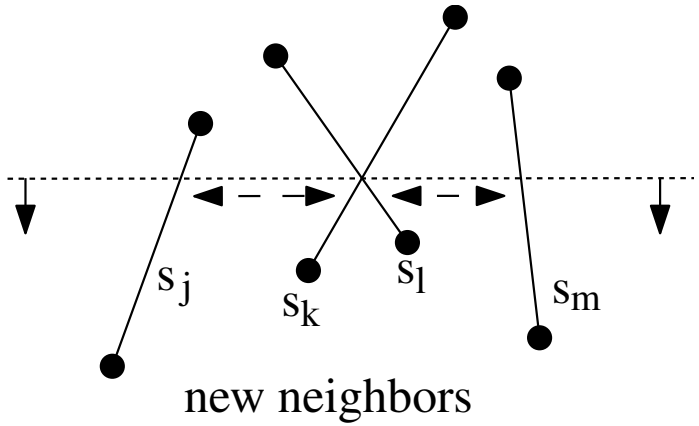
- order segments intersecting the sweep from left to right
- test segments only when they are adjacent
- update the status at each move of the sweep line



Algorithm Outline

Sweep-line Algorithm

- order segments intersecting the sweep from left to right
- test segments only when they are adjacent
- update the status at each move of the sweep line



Invariant

All intersection points above the sweep line were computed correctly.

Correctness

Lemma 1

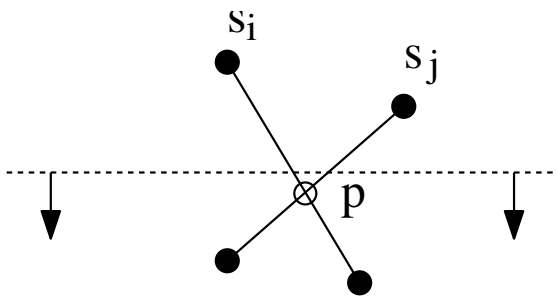
Let s_i and s_j be two non-horizontal segments whose interiors intersect in a single point p, and assume there is no third segment passing through p. Then there is an event point above p where s_i and s_j become adjacent and are tested for intersection.

Correctness

Lemma 1

Let s_i and s_j be two non-horizontal segments whose interiors intersect in a single point p, and assume there is no third segment passing through p. Then there is an event point above p where s_i and s_j become adjacent and are tested for intersection.

Proof



Data structures

event queue Q

- operations: remove next and return; insert
- implementation: balanced BST with event points, ordered w.r.t. $\prec$
- all operations in $O(\log m)$
- sorting of event points $\rightsquigarrow$ *symbolic perturbation*
- why not a heap?

Data structures

event queue Q

- operations: remove next and return; insert
- implementation: balanced BST with event points, ordered w.r.t. $\prec$
- all operations in $O(\log m)$
- sorting of event points $\rightsquigarrow$ *symbolic perturbation*
- why not a heap?

status structure T

- operations: insert; delete; find neighbor
- implementation: balanced BST (because it must always be sorted), leaves are segments intersecting sweep line
- all operations in $O(\log n)$

Bentley-Ottman Algorithm #1

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

- initialize an empty event queue Q and insert the segment points into it;
- initialize an empty status structure T;
- **while** Q is not empty **do**
  determine the next event point p;
  HandleEventPoint(p);

Bentley-Ottman Algorithm #2

HandleEventPoint(p)

- $U(p) := \{$segments whose upper endpoint is $p\}$ (these segments are stored with the event point p);
- Search in T 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;
- $C(p) := \{$segments that contain p in their interior$\}$;
- **if** $L(p) \cup U(p) \cup C(p)$ contains more than one segment
  then report p as an intersection, and $L(p)$, $U(p)$ and $C(p)$;
- Delete the segments in $L(p) \cup C(p)$ from T;
- 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);

Bentley-Ottman Algorithm #3

HandleEventPoint(p) *(continued)*

- **if** $U(p) \cup C(p) = \emptyset$
  then
    Let s_l and s_r be the neighbors of p in T;
    FindNewEvent(s_l,s_r,p);
  else
    Let s' be the leftmost segment of $U(p) \cup C(p)$ in T;
    Let s_l be the left neighbor of s' in T;
    FindNewEvent(s_l,s',p);
    Let s'' be the rightmost segment of $U(p) \cup C(p)$ in T;
    Let s_r be the right neighbor of s'' in T;
    FindNewEvent(s'',s_r,p);

Bentley-Ottman Algorithm #4

FindNewEvent(s_l,s_r,p)

- **if** [s_l and s_r intersect below the sweep line
  or (on it and to the right of the current event point p)]
  and the intersection is not yet present as an event in Q
  then
    Insert the intersection point as an event into Q;

Performance (exercise)

Lemma 2

FindIntersections computes all intersection points and the segments that contain it correctly.

Performance (exercise)

Lemma 2

FindIntersections computes all intersection points and the segments that contain it correctly.

Lemma 3

The running time of FindIntersections for a set S of n line segments in the plane is $O(n \log n + I \log n)$, where I is the number of intersection points of segments in S.

Performance (exercise)

Lemma 2

FindIntersections computes all intersection points and the segments that contain it correctly.

Lemma 3

The running time of FindIntersections for a set S of n line segments in the plane is $O(n \log n + I \log n)$, where I is the number of intersection points of segments in S.

Lemma 4

FindIntersections runs in $O(n)$ space.

DCEL: the Doubly-Connected Edge List

Why?

- maps contain subdivisions in regions
- DCEL is a representation of that
- it stores incidence relations. . . and whatever you want

DCEL: the Doubly-Connected Edge List

Why?

- maps contain subdivisions in regions
- DCEL is a representation of that
- it stores incidence relations. . . and whatever you want

DCEL permits

- point-location queries
- visit edges around a vertex

DCEL: the Doubly-Connected Edge List

Why?

- maps contain subdivisions in regions
- DCEL is a representation of that
- it stores incidence relations. . . and whatever you want

DCEL permits

- point-location queries
- visit edges around a vertex

Complexity of a subdivision

Sum of the number of vertices, edges and faces.

DCEL Definitions

Half-edges

- twin half-edges
- origin and destination

DCEL Definitions

Half-edges

- twin half-edges
- origin and destination

Implementation

- vertex table
- face table
- half-edge table

Stored Data

Vertex records

- coordinates of the vertex
- pointer to an incident edge

Stored Data

Vertex records

- coordinates of the vertex
- pointer to an incident edge

Face records

- outer component
- inner components

Stored Data

Vertex records

- coordinates of the vertex
- pointer to an incident edge

Face records

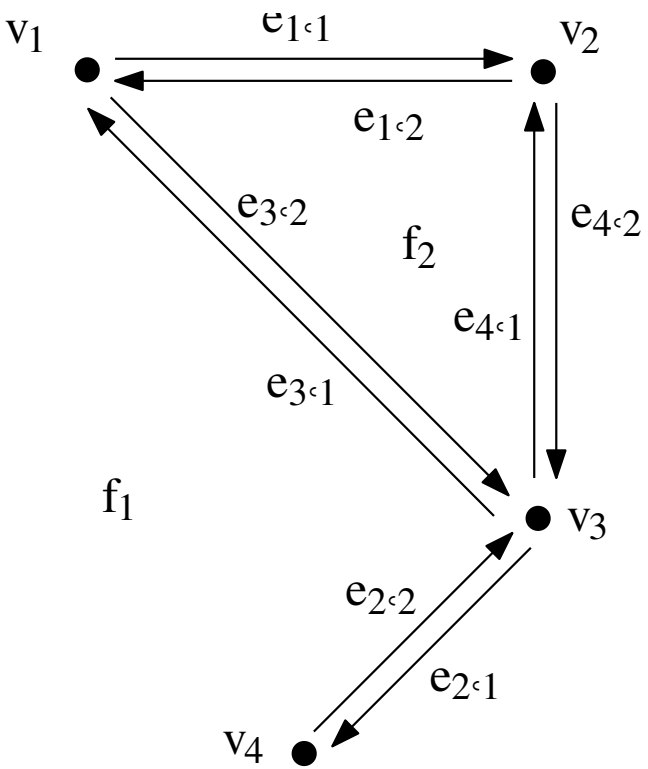
- outer component
- inner components

Half-edge records

- origin
- twin
- incident face
- previous and next

DCEL Example

A simple arrangement



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**

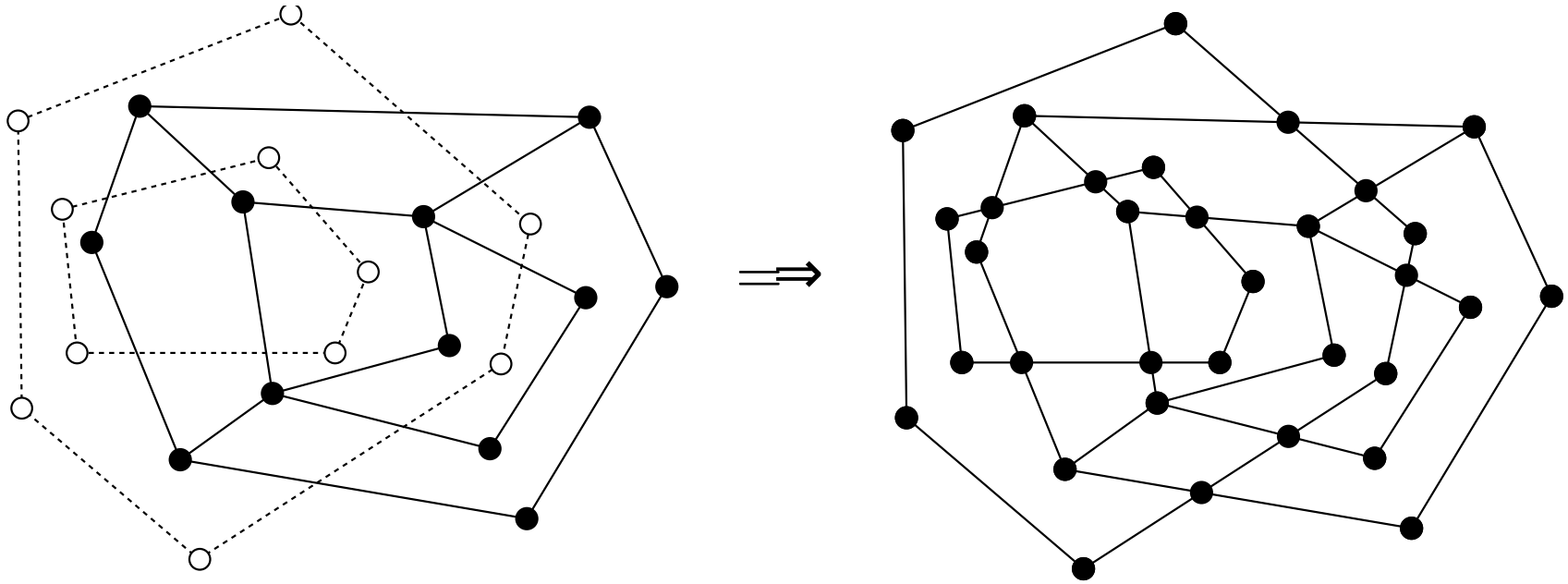
Half-edge	Origin	Twin	IncidentFace	Next	Prev
$\vec{e}_{1,1}$	v_1	$\vec{e}_{1,2}$	f_1	$\vec{e}_{4,2}$	$\vec{e}_{3,1}$
$\vec{e}_{1,2}$	v_2	$\vec{e}_{1,1}$	f_2	$\vec{e}_{3,2}$	$\vec{e}_{4,1}$
$\vec{e}_{2,1}$	v_3	$\vec{e}_{2,2}$	f_1	$\vec{e}_{2,2}$	$\vec{e}_{4,2}$
$\vec{e}_{2,2}$	v_4	$\vec{e}_{2,1}$	f_1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

General Version of the DCEL

Variants

- enough to walk around
- associate data with vertex, faces and half-edges
- store less information
- higher dimensions

Overlay of Subdivisions



Definition of Overlay

Formal

The overlay of two subdivisions S_1 and S_2 is the subdivision $O(S_1, S_2)$ such that there is a face f in $O(S_1, S_2)$ if and only if there are faces f_1 in S_1 and f_2 in S_2 such that f is a maximal connected subset of $f_1 \cap f_2$.

Definition of Overlay

Formal

The overlay of two subdivisions S_1 and S_2 is the subdivision $O(S_1, S_2)$ such that there is a face f in $O(S_1, S_2)$ if and only if there are faces f_1 in S_1 and f_2 in S_2 such that f is a maximal connected subset of $f_1 \cap f_2$.

Informal

The overlay $O(S_1, S_2)$ is the subdivision of the plane induced by the edges from S_1 and S_2.

Definition of Overlay

Formal

The overlay of two subdivisions S_1 and S_2 is the subdivision $O(S_1, S_2)$ such that there is a face f in $O(S_1, S_2)$ if and only if there are faces f_1 in S_1 and f_2 in S_2 such that f is a maximal connected subset of $f_1 \cap f_2$.

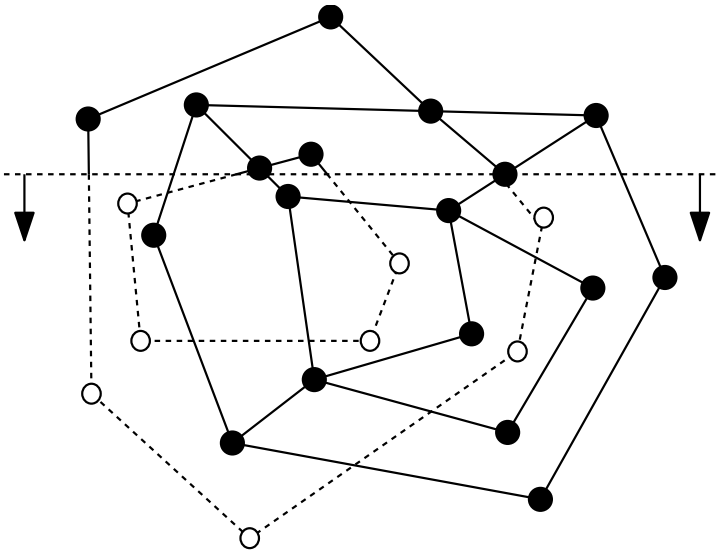
Informal

The overlay $O(S_1, S_2)$ is the subdivision of the plane induced by the edges from S_1 and S_2.

- each face in $O(S_1, S_2)$ must be labelled with the faces which contain it
- we can do it with the DCEL's of S_1 and S_2

How?

Sweep first

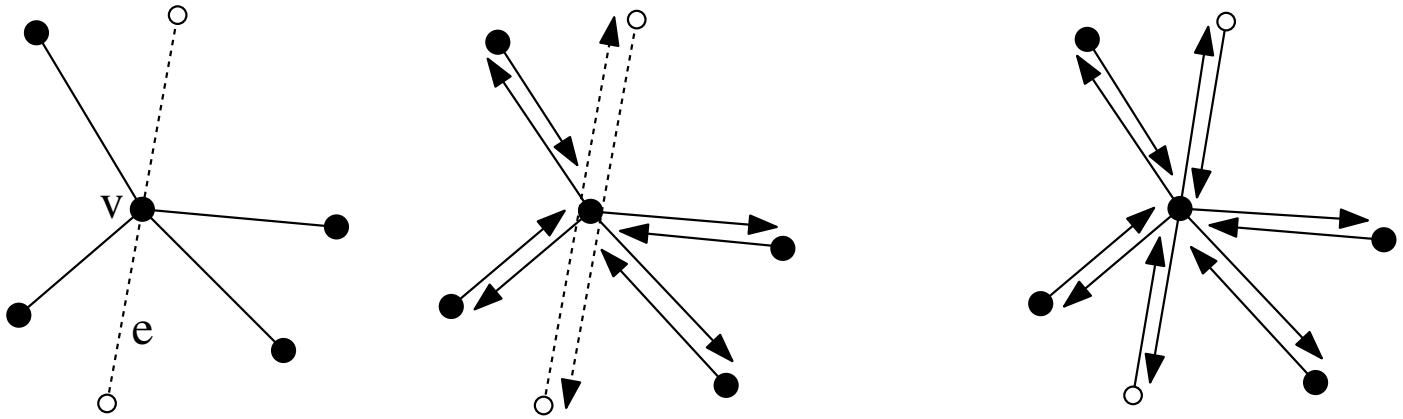


And

Handle intersections

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

the doubly-connected edge list after handling the intersection

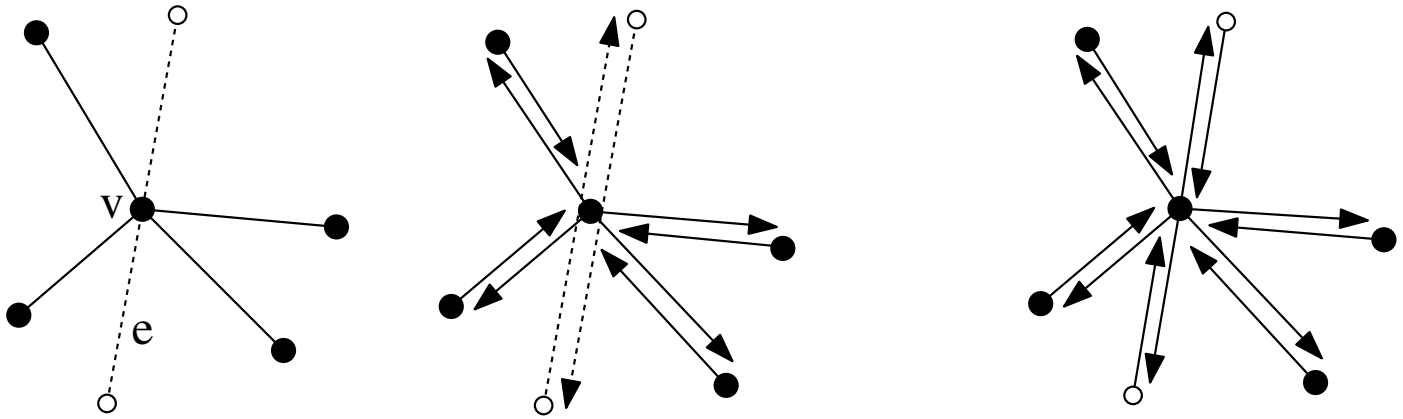


And

Handle intersections

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

the doubly-connected edge list after handling the intersection



What is left?

Overlay Faces

Graph G

- one node for each boundary cycle (inner or outer)
- one node for the imaginary outer boundary
- one arc between two cycles iff one of them is the boundary of a hole and the other has a half-edgde immediatly to the left of the leftmost vertex of that hole cycle
- no half edge to the left $\rightsquigarrow$ link the node representing the cycle with the unbounded face

Overlay Faces

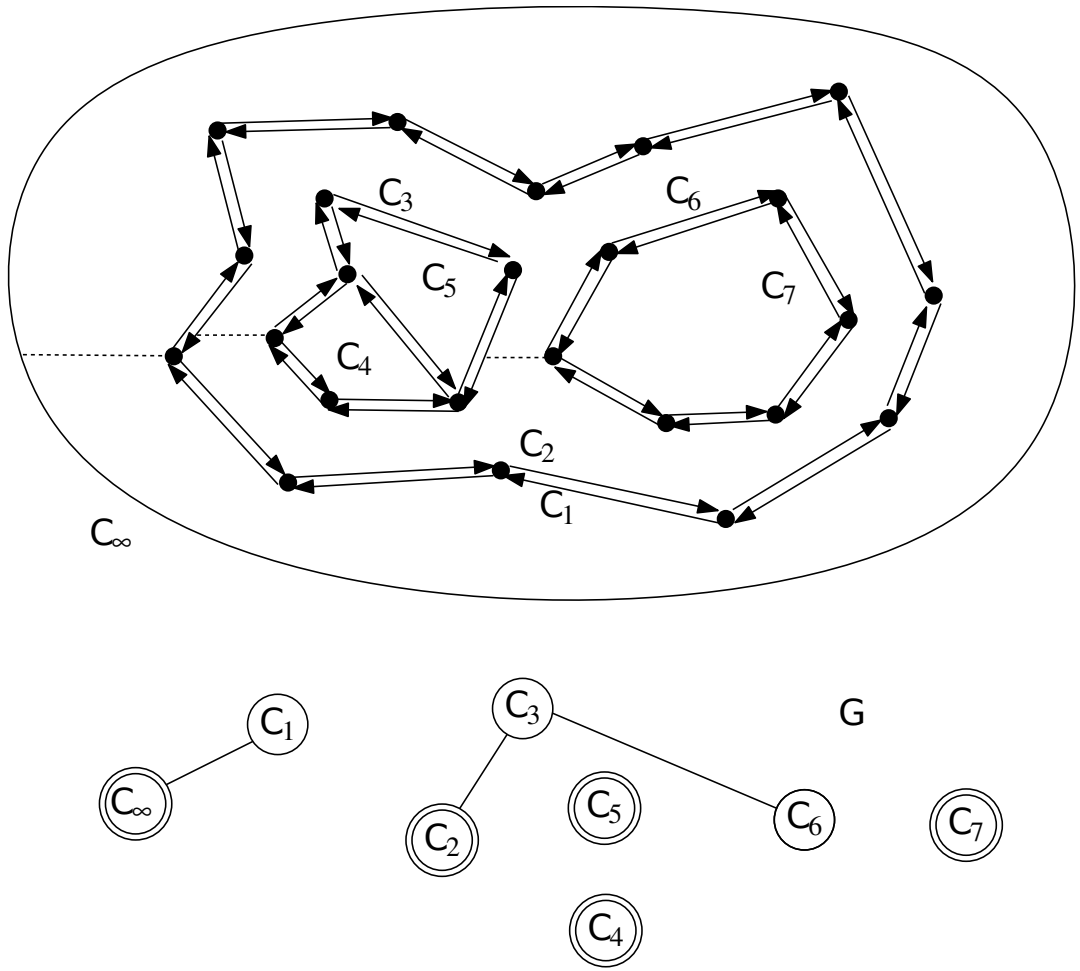
Graph G

- one node for each boundary cycle (inner or outer)
- one node for the imaginary outer boundary
- one arc between two cycles iff one of them is the boundary of a hole and the other has a half-edgde immediatly to the left of the leftmost vertex of that hole cycle
- no half edge to the left $\rightsquigarrow$ link the node representing the cycle with the unbounded face

Lemma

Each connected component of the graph G corresponds exactly to the set of cycles incident to one face.

Example of Graph



Overlay Algorithm #1

MapOverlay(S_1,S_2)

Input: subdivisions S_1 and S_2, stored in DCEL's
Output: their overlay, stored in a DCEL D

- copy DCEL's from S_1 and S_2 to a new DCEL D;
- compute all intersections between edges from S_1 and S_2 with the plane sweep algorithm. In addition to the actions on T and Q, do the following:
  - Update D if the event involves edges of both S_1 and S_2;
  - store the half-edge immediately to the left of the event point at the vertex in D representing it;

  (now D is the DCEL for $O(S_1, S_2)$, except that the information about the faces has not been computed yet)
- determine the boundary cycles in $O(S_1, S_2)$ by traversing D;

Overlay Algorithm #2

MapOverlay(S_1,S_2) *(continued)*

- construct the graph 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;
- **for** each connected component in G **do**
  let C be the unique outer boundary cycle in the component;
  let f denote the face bounded by the cycle;
  create a face record for f;
  OuterComponent(f) := some half-edge of C;
  InnerComponents(f) := {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;
- label each face of $O(S_1, S_2)$ with the names of the faces of S_1 and S_2 containing it;

Complexity of the overlay algorithm

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.

Complexity of the overlay algorithm

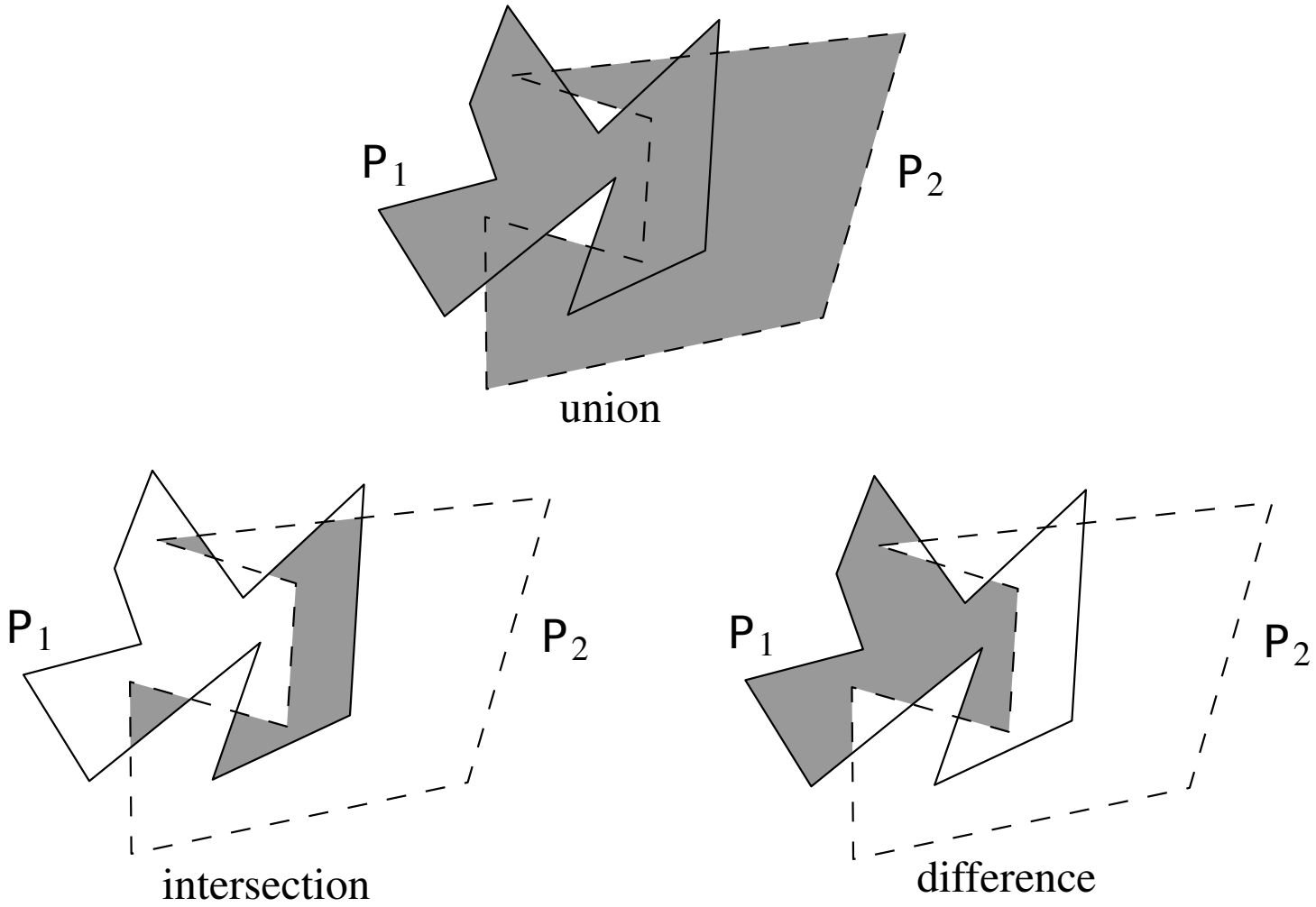
Theorem

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.

Proof

Exercise

Application: Boolean Operations



Boolean Operations

- a very important application
- computed as overlays of polygonal *regions* (can contain holes)

How to compute them

- use the overlay algorithm, but...
- intersection: extract only faces labeled with P_1 and P_2,
- union: extract P_1 or P_2, and
- difference: extract faces labeled with P_1 and not with P_2

Polygon Operations

Corollary 7

Let P_1 be a polygon with n_1 vertices and P_2 a polygon with n_2 vertices, and let $n = n_1 + n_2$. Then $P_1 \cap P_2$, $P_1 \cup P_2$ and $P_1 \setminus P_2$ can each be computed in $O(n \log n + k \log n)$ time, where k is the complexity of the output.