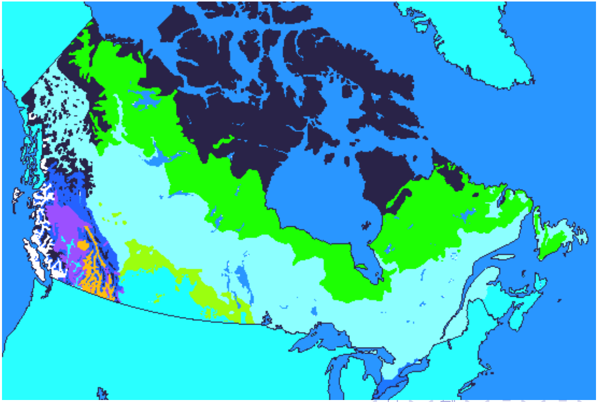
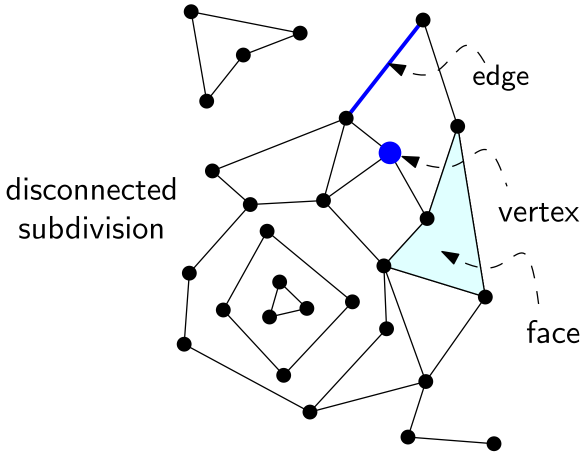
Computing the Overlay of Two Subdivisions

1389-2

- We have solved the easiest case of the map overlay problem, where the two maps are networks represented as collections of line segments.
- In general, maps have a more complicated structure: they are subdivisions of the plane into labeled regions.

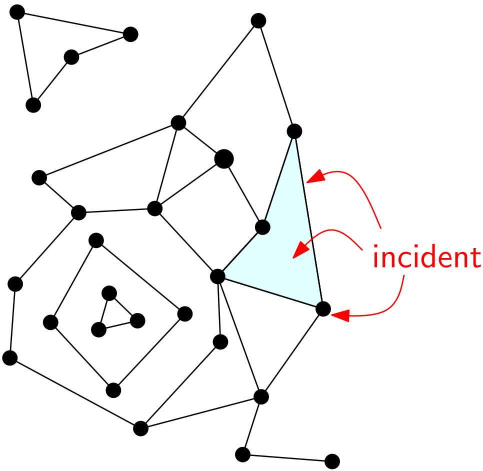


- Before we can give an algorithm for computing the overlay of two subdivisions, we must develop a suitable representation for a subdivision.
- Storing a subdivision as a collection of line segments is not such a good idea.
- Operations like reporting the boundary of a region would be rather complicated.
- Add topological information: which segments bound a given region, which regions are adjacent, and so on.



Complexity of a subdivision

#faces+#edges+#vertices.



Complexity of a subdivision

#faces+#edges+#vertices.

What kind of queries?

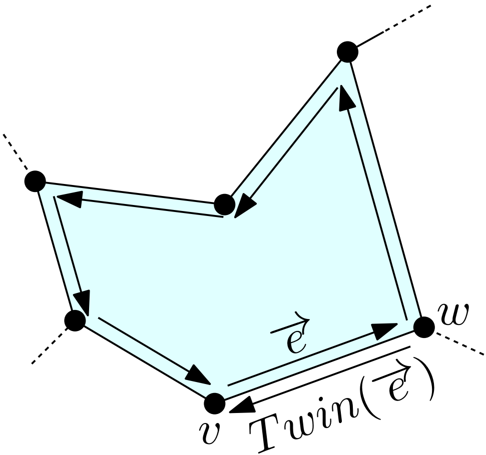
- What is the face containing a given point? (TOO MUCH!)
- Walking around the boundary of a given face,
- Find the face from an adjacent one if we are given a common edge,
- Visit all the edges around a given vertex.

The representation that we shall discuss supports these operations. It is called the doubly-connected edge list (DCEL).

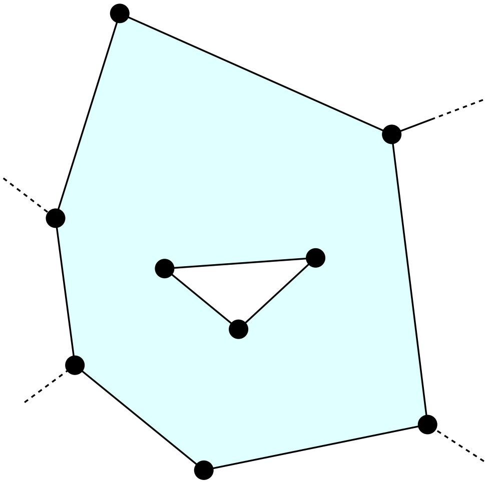
DCEL

DCEL contains:

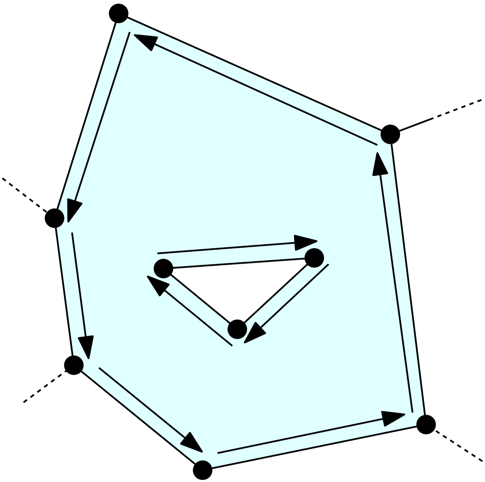
- a record for each edge,
- a record for each vertex,
- a record for each face,
- plus attribute information.



To be able to traverse the boundary of a face, we need to keep a pointer to a half-edge of any boundary component and isolated points.



To be able to traverse the boundary of a face, we need to keep a pointer to a half-edge of any boundary component and isolated points.



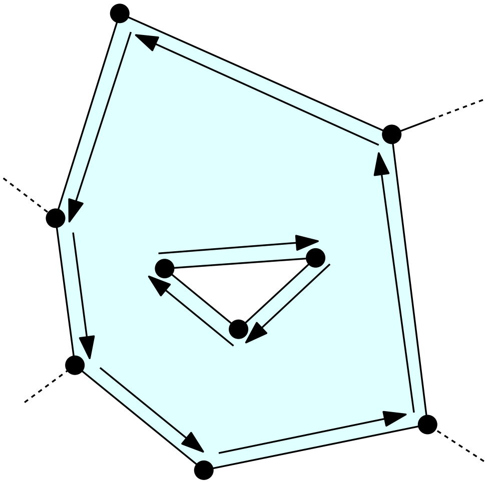
To be able to traverse the boundary of a face, we need to keep a pointer to a half-edge of any boundary component and isolated points.

Yazd Univ.

Computational Geometry

Doubly Connected Edge List (DCEL)

Computing the Overlay of Two Subdivisions



To be able to traverse the boundary of a face, we need to keep a pointer to a half-edge of any boundary component and isolated points.

Yazd Univ.

Computational Geometry

Doubly Connected Edge List (DCEL)

Computing the Overlay of Two Subdivisions

DCEL

DCEL contains:

- a record for each vertex,
  1. $Coordinates(v)$: the coordinates of v,
  2. $IncidentEdge(v)$: a pointer to an arbitrary half-edge that has v as its origin.
- a record for each face,
  1. $OuterComponent(f)$: to some half-edge on its outer boundary (nil if unbounded),
  2. $InnerComponents(f)$: a pointer to some half-edge on the boundary of the hole, for each hole.
- a record for each half-edge $\vec{e}$,
  1. $Origin(\vec{e})$: a pointer to its origin,
  2. $Twin(\vec{e})$ a pointer to its twin half-edge,
  3. $IncidentFace(\vec{e})$: a pointer to the face that it bounds.
  4. $Next(\vec{e})$ and $Prev(\vec{e})$: a pointer to the next and previous edge on the boundary of $IncidentFace(\vec{e})$.

DCEL

DCEL contains:

- a record for each vertex,
  1. $Coordinates(v)$: the coordinates of v,
  2. $IncidentEdge(v)$: a pointer to an arbitrary half-edge that has v as its origin.
- a record for each face,
  1. $OuterComponent(f)$: to some half-edge on its outer boundary (nil if unbounded),
  2. $InnerComponents(f)$: a pointer to some half-edge on the boundary of the hole, for each hole.
- a record for each half-edge $\vec{e}$,
  1. $Origin(\vec{e})$: a pointer to its origin,
  2. $Twin(\vec{e})$ a pointer to its twin half-edge,
  3. $IncidentFace(\vec{e})$: a pointer to the face that it bounds.
  4. $Next(\vec{e})$ and $Prev(\vec{e})$: a pointer to the next and previous edge on the boundary of $IncidentFace(\vec{e})$.

DCEL

DCEL contains:

- a record for each vertex,
  1. $Coordinates(v)$: the coordinates of v,
  2. $IncidentEdge(v)$: a pointer to an arbitrary half-edge that has v as its origin.
- a record for each face,
  1. $OuterComponent(f)$: to some half-edge on its outer boundary (nil if unbounded),
  2. $InnerComponents(f)$: a pointer to some half-edge on the boundary of the hole, for each hole.
- a record for each half-edge $\vec{e}$,
  1. $Origin(\vec{e})$: a pointer to its origin,
  2. $Twin(\vec{e})$ a pointer to its twin half-edge,
  3. $IncidentFace(\vec{e})$: a pointer to the face that it bounds.
  4. $Next(\vec{e})$ and $Prev(\vec{e})$: a pointer to the next and previous edge on the boundary of $IncidentFace(\vec{e})$.

DCEL

DCEL contains:

- a record for each vertex,
  1. $Coordinates(v)$: the coordinates of v,
  2. $IncidentEdge(v)$: a pointer to an arbitrary half-edge that has v as its origin.
- a record for each face,
  1. $OuterComponent(f)$: to some half-edge on its outer boundary (nil if unbounded),
  2. $InnerComponents(f)$: a pointer to some half-edge on the boundary of the hole, for each hole.
- a record for each half-edge $\overrightarrow{e}$,
  1. $Origin(\overrightarrow{e})$: a pointer to its origin,
  2. $Twin(\overrightarrow{e})$ a pointer to its twin half-edge,
  3. $IncidentFace(\overrightarrow{e})$: a pointer to the face that it bounds.
  4. $Next(\overrightarrow{e})$ and $Prev(\overrightarrow{e})$: a pointer to the next and previous edge on the boundary of $IncidentFace(\overrightarrow{e})$.

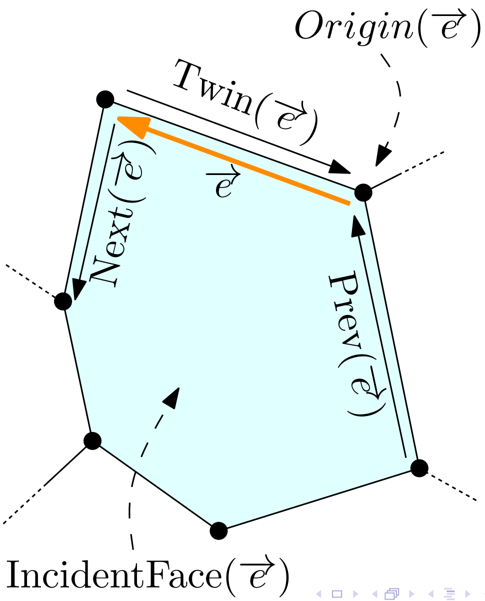
DCEL contains:

- a record for each vertex,
  1. $Coordinates(v)$: the coordinates of v,
  2. $IncidentEdge(v)$: a pointer to an arbitrary half-edge that has v as its origin.
- a record for each face,
  1. $OuterComponent(f)$: to some half-edge on its outer boundary (nil if unbounded),
  2. $InnerComponents(f)$: a pointer to some half-edge on the boundary of the hole, for each hole.
- a record for each half-edge $\vec{e}$,
  1. $Origin(\vec{e})$: a pointer to its origin,
  2. $Twin(\vec{e})$ a pointer to its twin half-edge,
  3. $IncidentFace(\vec{e})$: a pointer to the face that it bounds.
  4. $Next(\vec{e})$ and $Prev(\vec{e})$: a pointer to the next and previous edge on the boundary of $IncidentFace(\vec{e})$.

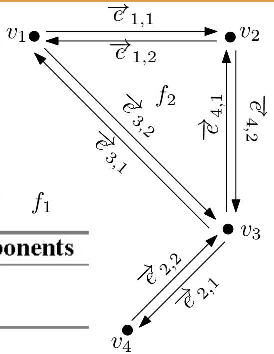
DCEL contains:

- a record for each vertex,
  1. $Coordinates(v)$: the coordinates of v,
  2. $IncidentEdge(v)$: a pointer to an arbitrary half-edge that has v as its origin.
- a record for each face,
  1. $OuterComponent(f)$: to some half-edge on its outer boundary (nil if unbounded),
  2. $InnerComponents(f)$: a pointer to some half-edge on the boundary of the hole, for each hole.
- a record for each half-edge $\vec{e}$,
  1. $Origin(\vec{e})$: a pointer to its origin,
  2. $Twin(\vec{e})$ a pointer to its twin half-edge,
  3. $IncidentFace(\vec{e})$: a pointer to the face that it bounds.
  4. $Next(\vec{e})$ and $Prev(\vec{e})$: a pointer to the next and previous edge on the boundary of $IncidentFace(\vec{e})$.

DCEL


$Origin(\overrightarrow{e})$
$\mathrm{Twin}(\overrightarrow{e})$
$\overrightarrow{e}$
$\mathrm{Next}(\overrightarrow{e})$
$\mathrm{Prev}(\overrightarrow{e})$
$\mathrm{IncidentFace}(\overrightarrow{e})$

DCEL:Example



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	$(1,1)$	$\vec{e}_{2,2}$

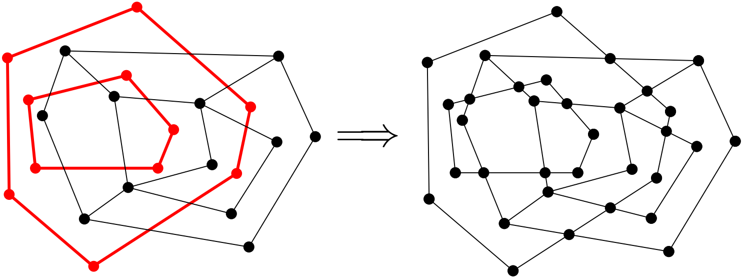
Face	OuterComponent	InnerComponents
f_1	**nil**	$\vec{e}_{1,1}$
f_2	$\vec{e}_{4,1}$	**nil**

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	$\vec{e}_{2,1}$	$\vec{e}_{1,1}$

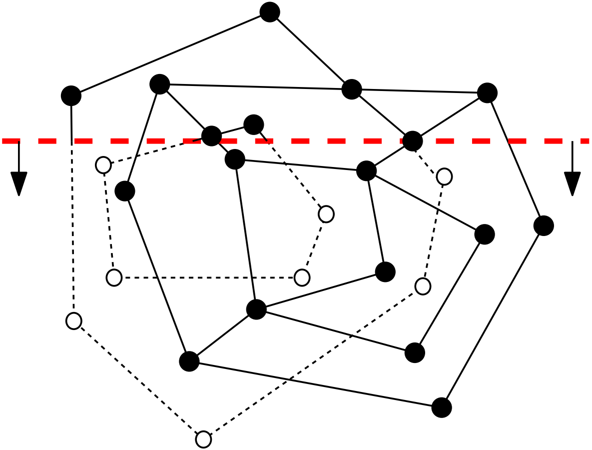
Time complexity of queries?

- Walking around the boundary of a given face,
- Find the face from an adjacent one if we are given a common edge,
- Visit all the edges around a given vertex.

Computing the Overlay of Two Subdivisions



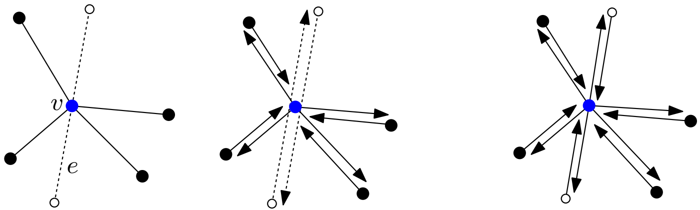
Computing the Overlay of Two Subdivisions



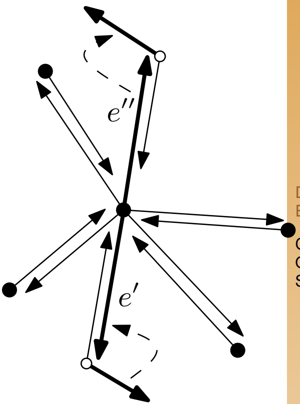
Computing the Overlay of Two Subdivisions

Updating half-edges

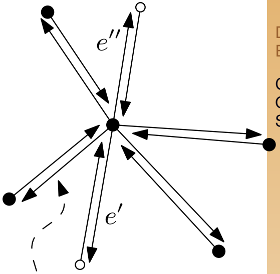
the geometric situation and the two doubly-connected edge lists before handling the intersection	the doubly-connected edge list after handling the intersection



- $Next()$ pointers of the two new half-edges each copy the $Next()$ pointer of the old half-edge that is not its twin.
- The half-edges to which these pointers point must also update their $Prev()$ pointer and set it to the new half-edges.

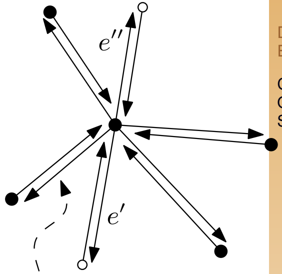


- The half-edge for e' that has v as its destination must be linked to the first half-edge, seen clockwise from e', with v as its origin.
- The half-edge for e' with v as its origin must be linked to the first counterclockwise half-edge with v as its destination.
- The same for e''.
- Time complexity: $\mathcal{O}(m)$ (m: degree of v).



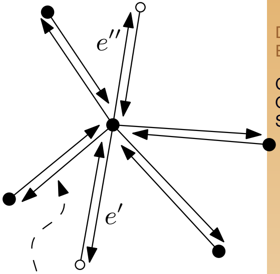
first clockwise half-edge from e' with v as its origin

- The half-edge for e' that has v as its destination must be linked to the first half-edge, seen clockwise from e', with v as its origin.
- The half-edge for e' with v as its origin must be linked to the first counterclockwise half-edge with v as its destination.
- The same for e''.
- Time complexity: $\mathcal{O}(m)$ (m: degree of v).



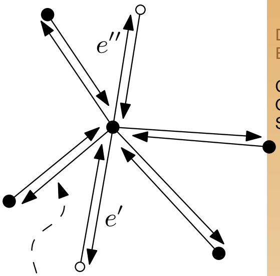
first clockwise half-edge from e' with v as its origin

- The half-edge for e' that has v as its destination must be linked to the first half-edge, seen clockwise from e', with v as its origin.
- The half-edge for e' with v as its origin must be linked to the first counterclockwise half-edge with v as its destination.
- The same for e''.
- Time complexity: $\mathcal{O}(m)$ (m: degree of v).



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

- The half-edge for e' that has v as its destination must be linked to the first half-edge, seen clockwise from e', with v as its origin.
- The half-edge for e' with v as its origin must be linked to the first counterclockwise half-edge with v as its destination.
- The same for e''.
- Time complexity: $\mathcal{O}(m)$ (m: degree of v).

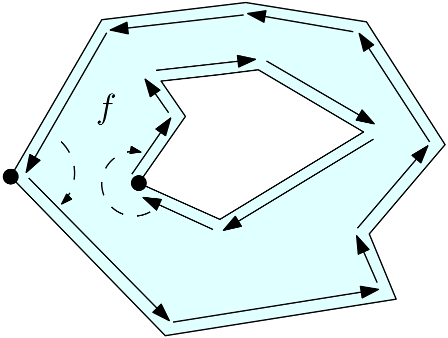


first clockwise half-edge from e' with v as its origin

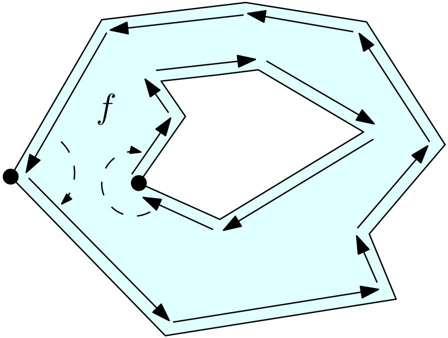
Computing the Overlay of Two Subdivisions

Updating faces

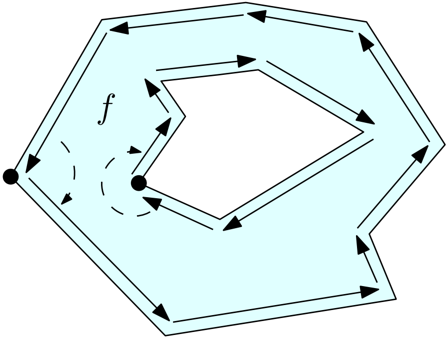
- \# faces= # outer boundaries +1 (unbounded face).
- From half-edges we can construct the boundaries.
- To determine weather the boundary is outer boundary or boundary of a hole:



- # faces= # outer boundaries +1 (unbounded face).
- From half-edges we can construct the boundaries.
- To determine weather the boundary is outer boundary or boundary of a hole:



- $\#$ faces= $\#$ outer boundaries +1 (unbounded face).
- From half-edges we can construct the boundaries.
- To determine weather the boundary is outer boundary or boundary of a hole:



Which boundary cycles bound the same face?

- Construct a graph $\mathcal{G}$.
- Every boundary cycle is a node in $\mathcal{G}$.
- One node for the imaginary outer boundary of the unbounded face.
- Add 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.
- If there is no half-edge to the left of the leftmost vertex of a cycle, then the node representing the cycle is linked to the node of the unbounded face.

Which boundary cycles bound the same face?

- Construct a graph $\mathcal{G}$.
- Every boundary cycle is a node in $\mathcal{G}$.
- One node for the imaginary outer boundary of the unbounded face.
- Add 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.
- If there is no half-edge to the left of the leftmost vertex of a cycle, then the node representing the cycle is linked to the node of the unbounded face.

Which boundary cycles bound the same face?

- Construct a graph $\mathcal{G}$.
- Every boundary cycle is a node in $\mathcal{G}$.
- One node for the imaginary outer boundary of the unbounded face.
- Add 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.
- If there is no half-edge to the left of the leftmost vertex of a cycle, then the node representing the cycle is linked to the node of the unbounded face.

Which boundary cycles bound the same face?

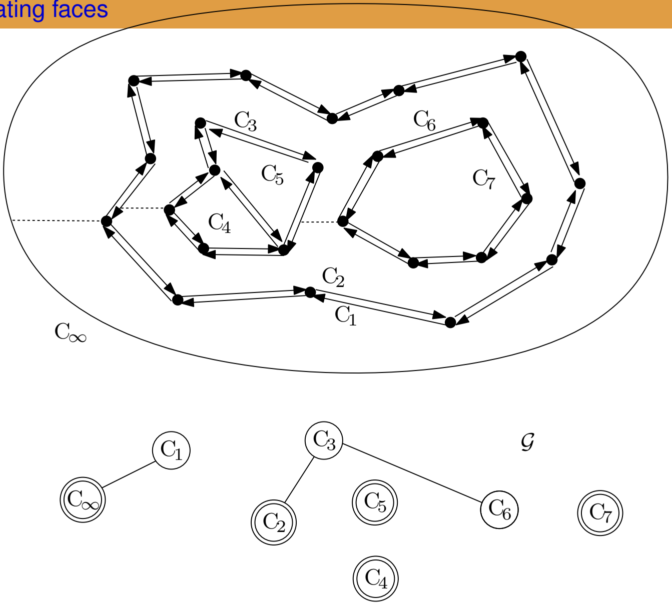
- Construct a graph $\mathcal{G}$.
- Every boundary cycle is a node in $\mathcal{G}$.
- One node for the imaginary outer boundary of the unbounded face.
- Add an arc between two cycles if and only if one of the cycles is the boundary of a hole and the other cycle has a half-edge immediately to the left of the leftmost vertex of that hole cycle.
- If there is no half-edge to the left of the leftmost vertex of a cycle, then the node representing the cycle is linked to the node of the unbounded face.

Which boundary cycles bound the same face?

- Construct a graph $\mathcal{G}$.
- Every boundary cycle is a node in $\mathcal{G}$.
- One node for the imaginary outer boundary of the unbounded face.
- Add 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.
- If there is no half-edge to the left of the leftmost vertex of a cycle, then the node representing the cycle is linked to the node of the unbounded face.

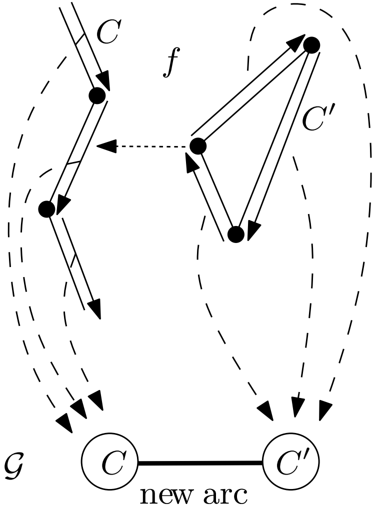
Computing the Overlay of Two Subdivisions

Updating faces



Lemma 2.5

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

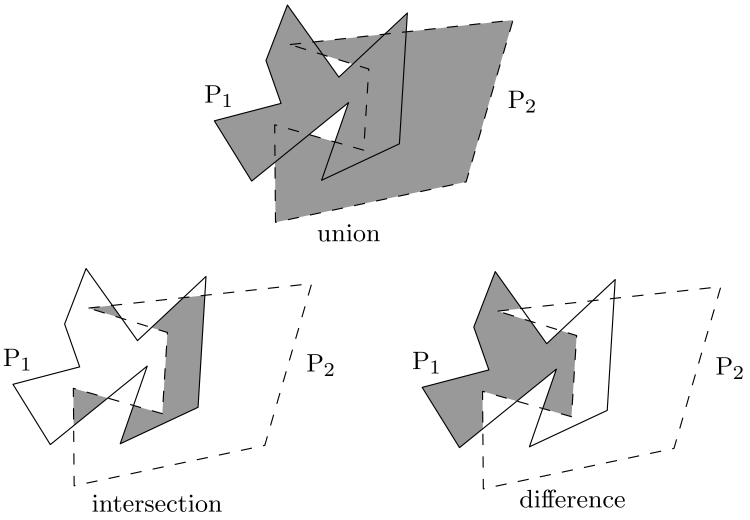

C
f
C'
$\mathcal{G}$
C
C'
new arc

Computing the Overlay of Two Subdivisions

Theorem 2.6

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 $\mathcal{O}(n \log n + k \log n)$ time, where k is the complexity of the overlay.

Application:Boolean Operations



END.