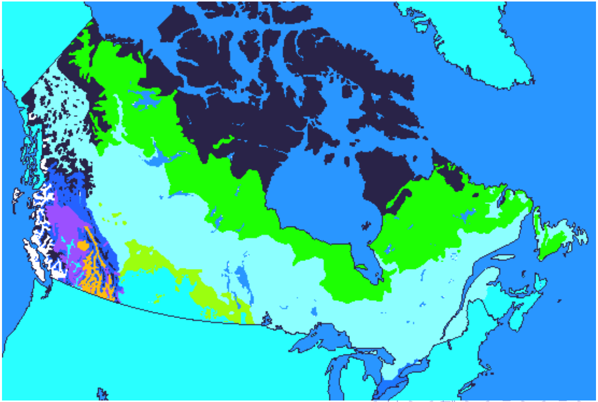
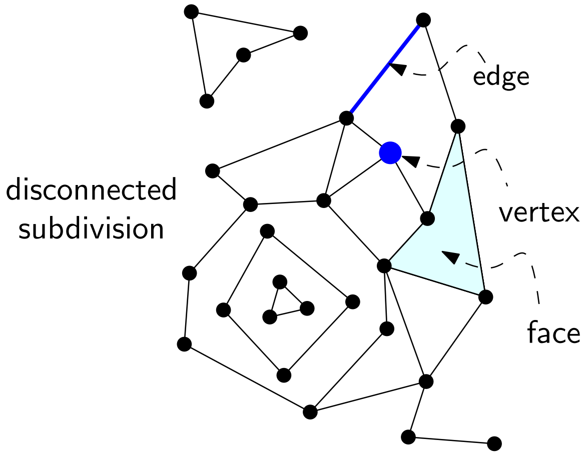
Computing the Overlay of Two Subdivisions

1397-2

- 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.

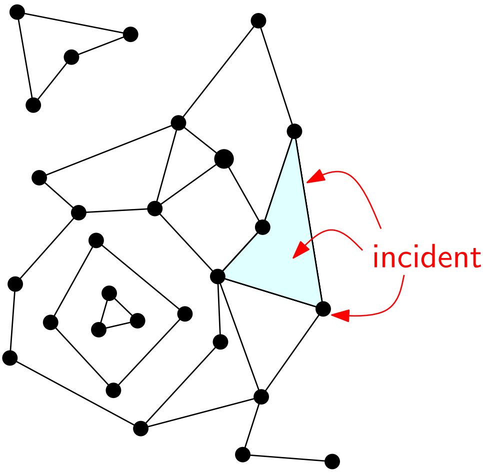


- Before we can give an algorithm for computing the overlay of two subdivisions, we must develop a suitable representation for a subdivision.
- Storing a subdivision as a collection of line segments is not such a good idea.
- Operations like reporting the boundary of a region would be rather complicated.
- 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?

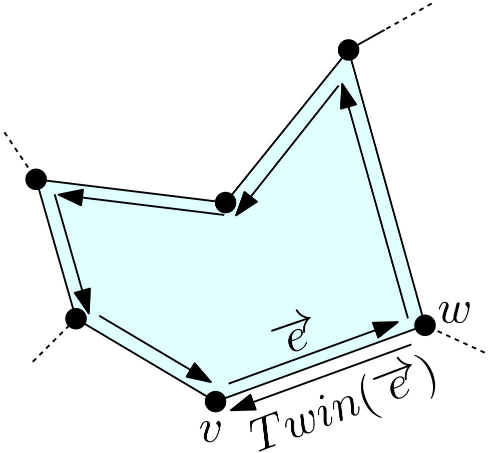
- What is the face containing a given point? (TOO MUCH!)
- Walking around the boundary of a given face,
- Find the face from an adjacent one if we are given a common edge,
- Visit all the edges around a given vertex.

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

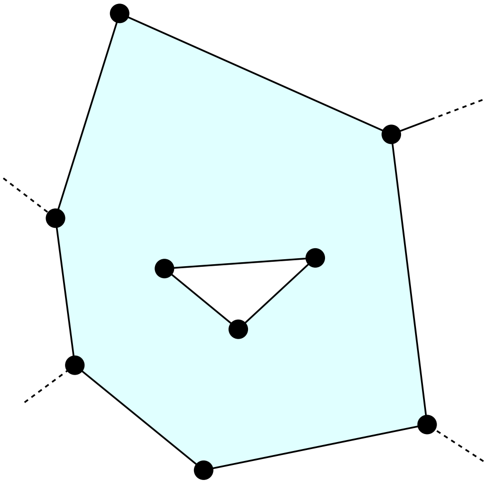
DCEL

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.



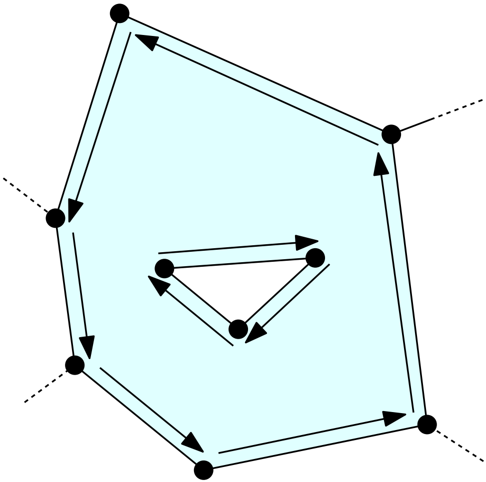
To be able to traverse the boundary of a face, we need to keep a pointer to a half-edge of any boundary component and isolated points.

Yazd Univ.

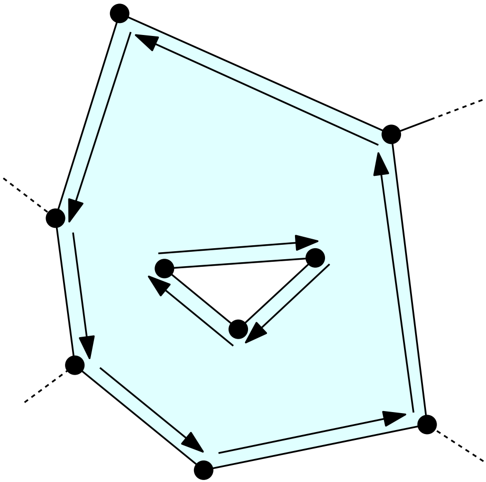
Computational Geometry

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.



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.

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 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

DCEL contains:

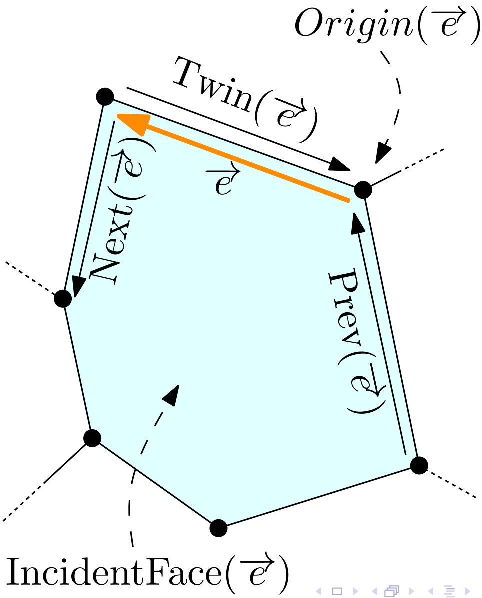
- a record for each vertex,
  1. $Coordinates(v)$: the coordinates of v,
  2. $IncidentEdge(v)$: a pointer to an arbitrary half-edge that has v as its origin.
- a record for each face,
  1. $OuterComponent(f)$: to some half-edge on its outer boundary (nil if unbounded),
  2. $InnerComponents(f)$: a pointer 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 $IncidentFace(\overrightarrow{e})$.

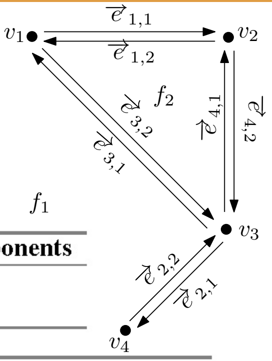
DCEL



$Origin(\overrightarrow{e})$
Twin($\overrightarrow{e}$)
$\overrightarrow{e}$
Next($\overrightarrow{e}$)
Prev($\overrightarrow{e}$)
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	$\vec{e}_{2,1}$	$\vec{e}_{1,1}$

DCEL

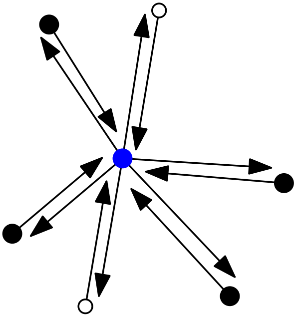
Time complexity of queries?

- 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.

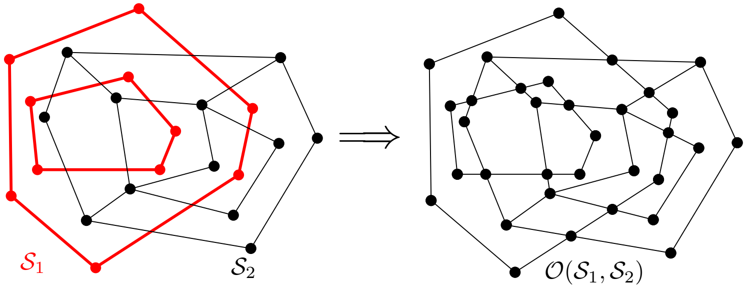
DCEL

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.

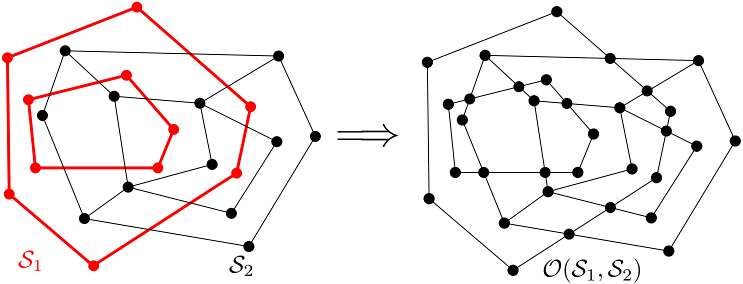


Computing the Overlay of Two Subdivisions



Which records are (almost) the same as input in the overlay?

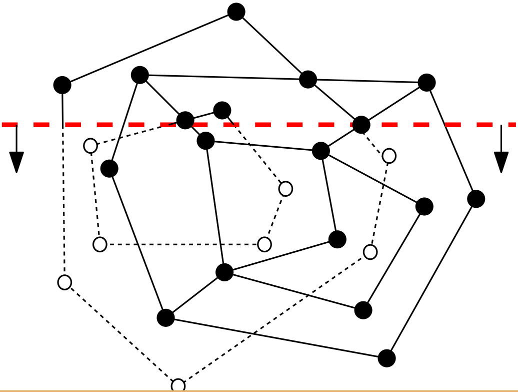
- vertex records (except the incident edge)
- edge records that are not intersected (except some fields)



Main Idea in computing $\mathcal{O}(\mathcal{S}_1, \mathcal{S}_2)$

- Copy DCEL of $\mathcal{S}_1$ and $\mathcal{S}_2$ into new DCEL (Not a Valid DCEL).
- Make the new DCEL valid.

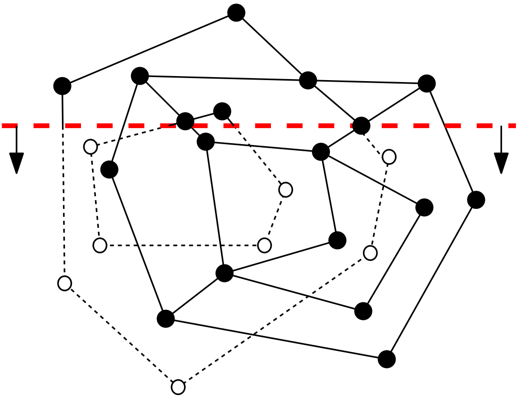
Computing the Overlay of Two Subdivisions



What the algorithm should do?

- Compute the intersections and add them to the DCEL.
- Update half-edge records that have intersection.
- Add records for new faces and update previous ones. (will do it later)

Computing the Overlay of Two Subdivisions

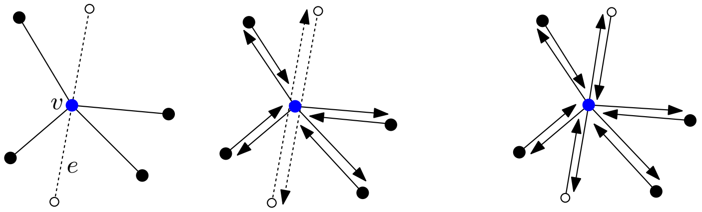


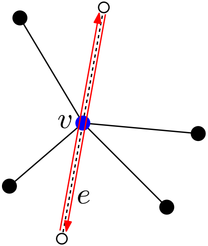
- The algorithm is similar to the algorithm for computing line segments intersections.

Case 1: an edge e of one subdivision passes through a vertex v of other subdivision:

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

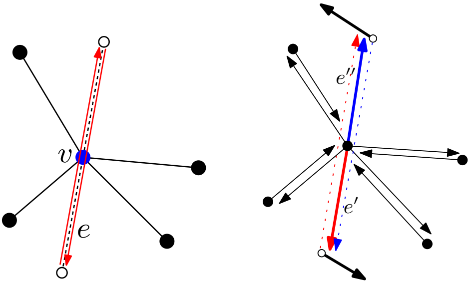
the doubly-connected edge list after handling the intersection





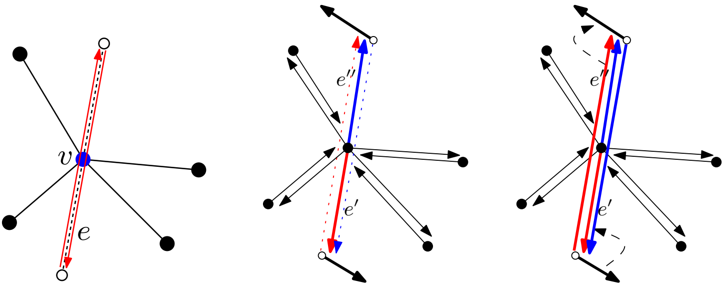
Update Algorithm:

- Make two new edge with origin v.
- Set $Twin$ of new edges.
- Set $Next()$ of the two new half-edges.
- Set $Prev()$ of the half-edges to which these pointers point.



Update Algorithm:

- Make two new edge with origin v.
- Set $Twin$ of new edges.
- Set $Next()$ of the two new half-edges.
- Set $Prev()$ of the half-edges to which these pointers point.

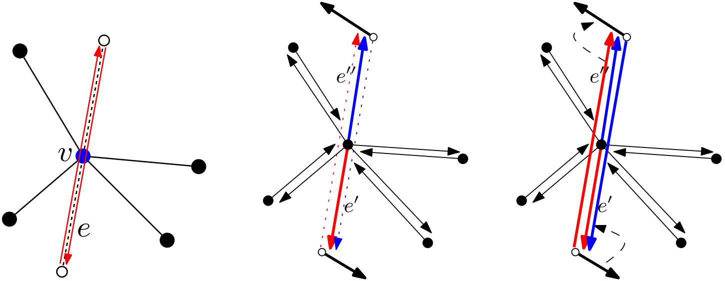


Update Algorithm:

- Make two new edge with origin v.
- Set $Twin$ of new edges.
- Set $Next()$ of the two new half-edges.
- Set $Prev()$ of the half-edges to which these pointers point.

Computing the Overlay of Two Subdivisions

Updating half-edges

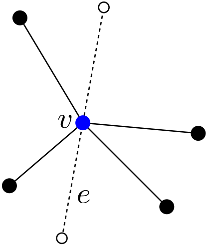


Update Algorithm:

- Make two new edge with origin v.
- Set $Twin$ of new edges.
- Set $Next()$ of the two new half-edges.
- Set $Prev()$ of the half-edges to which these pointers point.

Computing the Overlay of Two Subdivisions

Updating half-edges



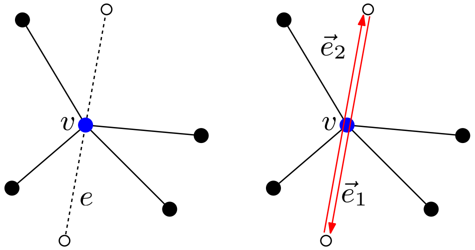
Updates:

half-edge	Origin	Twin	Face	Next	Prev
$\vec{e}_1$	No Change	$\vec{e'}_1$	?		No Change
$\vec{e'}_1$	v	$\vec{e}_1$	?	$Next(\vec{e}_1)$	
$\vec{e}_2$	No Change	$\vec{e'}_2$	?		No Change
$\vec{e'}_2$	v	$\vec{e}_2$	?	$Next(\vec{e}_2)$	

Update $Prev(Next(\vec{e}_1))$ and $Prev(Next(\vec{e}_2))$

Computing the Overlay of Two Subdivisions

Updating half-edges



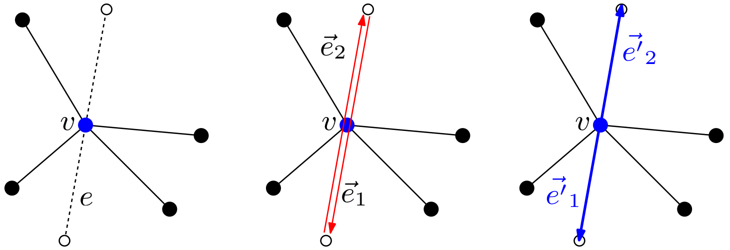
Updates:

half-edge	Origin	Twin	Face	Next	Prev
$\vec{e}_1$	No Change	$\vec{e'}_1$	?		No Change
$\vec{e'}_1$	v	$\vec{e}_1$	?	$Next(\vec{e}_1)$	
$\vec{e}_2$	No Change	$\vec{e'}_2$	?		No Change
$\vec{e'}_2$	v	$\vec{e}_2$	?	$Next(\vec{e}_2)$	

Update $Prev(Next(\vec{e}_1))$ and $Prev(Next(\vec{e}_2))$

Computing the Overlay of Two Subdivisions

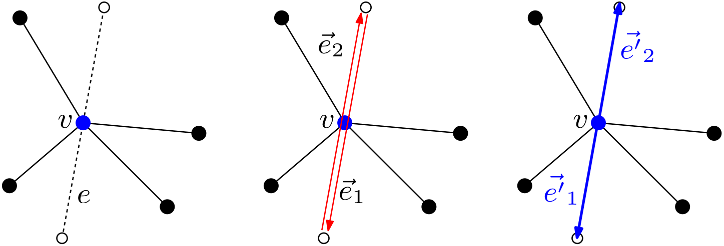
Updating half-edges



Updates:

half-edge	Origin	Twin	Face	Next	Prev
$\vec{e}_1$	No Change	$\vec{e'}_1$	?		No Change
$\vec{e'}_1$	v	$\vec{e}_1$	?	$Next(\vec{e}_1)$	
$\vec{e}_2$	No Change	$\vec{e'}_2$	?		No Change
$\vec{e'}_2$	v	$\vec{e}_2$	?	$Next(\vec{e}_2)$	

Update $Prev(Next(\vec{e}_1))$ and $Prev(Next(\vec{e}_2))$



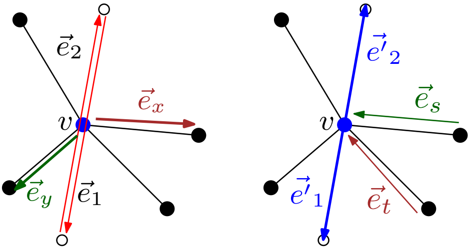
Updates:

half-edge	Origin	Twin	Face	Next	Prev
$\vec{e}_1$	No Change	$\vec{e'}_1$	?		No Change
$\vec{e'}_1$	v	$\vec{e}_1$	?	$Next(\vec{e}_1)$	
$\vec{e}_2$	No Change	$\vec{e'}_2$	?		No Change
$\vec{e'}_2$	v	$\vec{e}_2$	?	$Next(\vec{e}_2)$	

Update $Prev(Next(\vec{e}_1))$ and $Prev(Next(\vec{e}_2))$

Computing the Overlay of Two Subdivisions

Updating half-edges



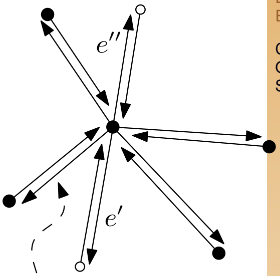
Updates: (around v)

half-edge	Origin	Twin	Face	Next	Prev
$\vec{e}_1$	No Change	$\vec{e'}_1$	?	$\vec{e}_x$	No Change
$\vec{e'}_1$	v	$\vec{e}_1$	?	$Next(\vec{e}_1)$	$\vec{e}_t$
$\vec{e}_2$	No Change	$\vec{e'}_2$	?	$\vec{e}_y$	No Change
$\vec{e'}_2$	v	$\vec{e}_2$	?	$Next(\vec{e}_2)$	$\vec{e}_s$

Update $Prev(\vec{e}_x)$, $Prev(\vec{e}_y)$, $Prev(\vec{e}_t)$, $Prev(\vec{e}_s)$.

Fix the situation around v

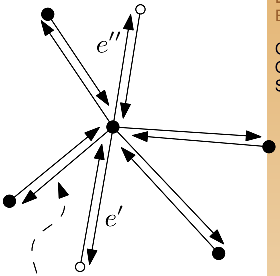
- The half-edge for e' that has v as its destination must be linked to the first half-edge, seen clockwise from e', with v as its origin.
- The half-edge for e' with v as its origin must be linked to the first counterclockwise half-edge with v as its destination.
- The same for e''.
- Time complexity: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



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

Fix the situation around v

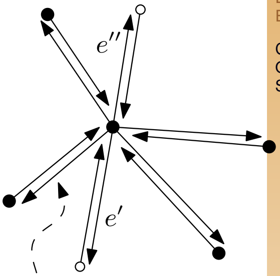
- The half-edge for e' that has v as its destination must be linked to the first half-edge, seen clockwise from e', with v as its origin.
- The half-edge for e' with v as its origin must be linked to the first counterclockwise half-edge with v as its destination.
- The same for e''.
- Time complexity: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



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

Fix the situation around v

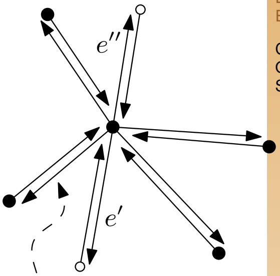
- The half-edge for e' that has v as its destination must be linked to the first half-edge, seen clockwise from e', with v as its origin.
- The half-edge for e' with v as its origin must be linked to the first counterclockwise half-edge with v as its destination.
- The same for e''.
- Time complexity: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



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

Fix the situation around v

- 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

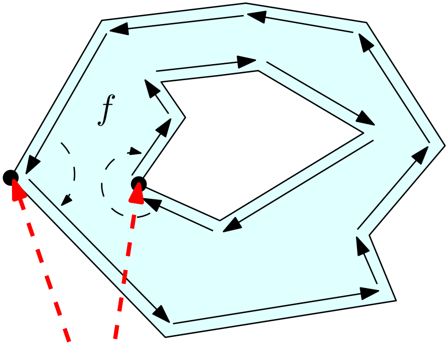
Updating Faces:

- Create a face record for each $f \in \mathcal{O}(\mathcal{S}_1, \mathcal{S}_2)$.
- Set $OuterComponent(f)$ and $InnerComponent(f)$.

- # 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?

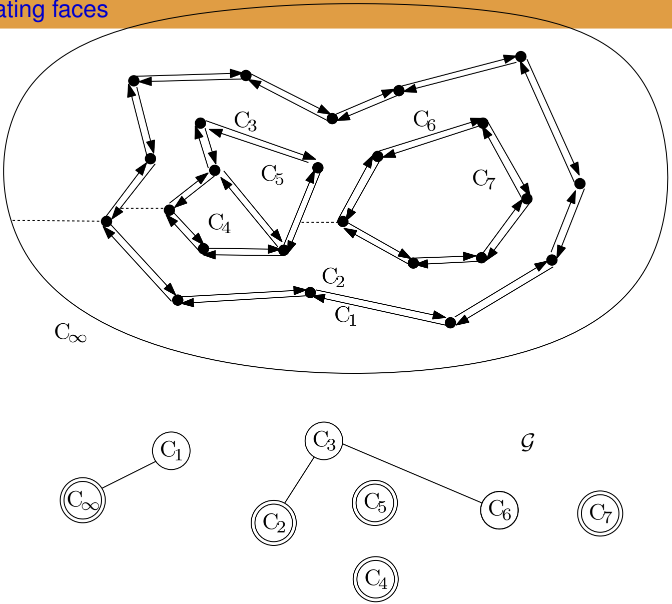
- Construct a graph $\mathcal{G}$.
- Every boundary cycle is a node in $\mathcal{G}$.
- One node for the imaginary outer boundary of the unbounded face.
- Add an arc between two cycles if and only if one of the cycles is the boundary of a hole and the other cycle has a half-edge immediately to the left 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

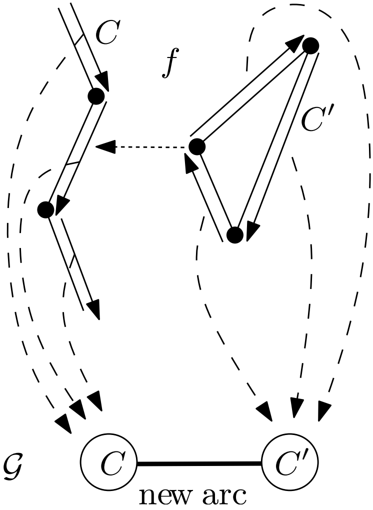


Lemma 2.5

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

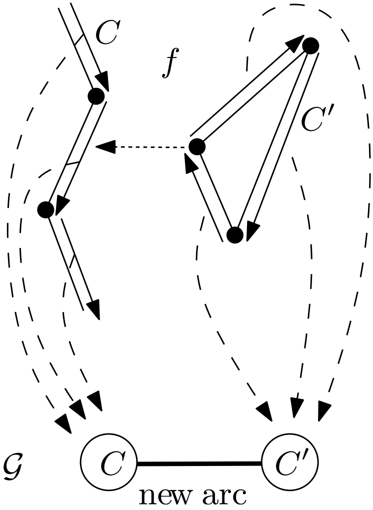
Constructing $\mathcal{G}$

- Similar to line segment intersection algorithm.
- Find the edge immediately to the left of point.
- We need a pointer from each edge to its boundary in $\mathcal{G}$.



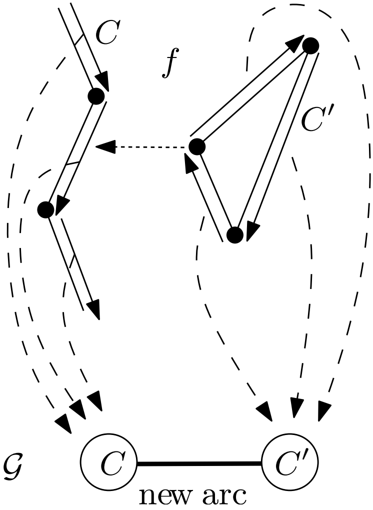
Constructing $\mathcal{G}$

- Similar to line segment intersection algorithm.
- Find the edge immediately to the left of point.
- We need a pointer from each edge to its boundary in $\mathcal{G}$.



Constructing $\mathcal{G}$

- Similar to line segment intersection algorithm.
- Find the edge immediately to the left of point.
- We need a pointer from each edge to its boundary in $\mathcal{G}$.



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.

Computing the Overlay of Two Subdivisions

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

Application:Boolean Operations