Subdivision Overlay

Outline

I. The overlay problem

II. Edge update

III. Face update

IV. Algorithm & boolean operations
I. Overlay of Two Subdivisions

$S_1$

doubly-connected edge list 1 (DCEL1)
I. Overlay of Two Subdivisions

\[ S_2 \ (\text{DCEL2}) \]
The overlay is a new planar subdivision.
The Overlay Problem
The Overlay Problem

Compute a doubly-connected edge list for the new planar subdivision.
The Overlay Problem

Compute a doubly-connected edge list for the new planar subdivision.

new intersection.
The Overlay Problem

Compute a doubly-connected edge list for the new planar subdivision.

Half-edge records reusable since the edge is not intersected by those from the other subdivision.

new intersection.
The Overlay Problem

Compute a doubly-connected edge list for the new planar subdivision.

Half-edge records reusable since the edge is not intersected by those from the other subdivision.

Half-edge records need to be generated at new intersection.
The Overlay Problem

- Compute a doubly-connected edge list for the new planar subdivision.
- Every face is labeled with the labels of the containing faces from the input subdivisions.

Half-edge records need to be generated since the edge is not intersected by those from the other subdivision.

**new intersection.**

**Half-edge records need to be generated**
The Overlay Problem

Compute a doubly-connected edge list for the new planar subdivision.

Every face is labeled with the labels of the containing faces from the input subdivisions.

Half-edge records need to be generated.

Half-edge records are reusable since the edge is not intersected by those from the other subdivision.
The Overlay Problem

- Compute a doubly-connected edge list for the new planar subdivision.

- Every face is labeled with the labels of the containing faces from the input subdivisions.

Half-edge records need to be generated for new intersections.
The Overlay Problem

- Compute a doubly-connected edge list for the new planar subdivision.

- Every face is labeled with the labels of the containing faces from the input subdivisions.

Half-edge records reusable since the edge is not intersected by those from the other subdivision.

Half-edge records need to be generated

new intersection.
First, copy the DCELs of two subdivisions.
The General Approach

★ First, copy the DCELs of two subdivisions.
★ Transform the result into a valid DCEL for the subdivision overlay.
The General Approach

★ First, copy the DCELs of two subdivisions.
★ Transform the result into a valid DCEL for the subdivision overlay.

- Compute the intersections of edges from different subdivisions.
- Link together appropriate parts of the two DCELs.
The General Approach

First, copy the DCELs of two subdivisions.

Transform the result into a valid DCEL for the subdivision overlay.

- Compute the intersections of edges from different subdivisions.
- Link together appropriate parts of the two DCELs.
  1. Vertex and half-edge records.
The General Approach

First, copy the DCELs of two subdivisions.

Transform the result into a valid DCEL for the subdivision overlay.

- Compute the intersections of edges from different subdivisions.
- Link together appropriate parts of the two DCELs.
  1. Vertex and half-edge records.
  2. Face records.
The General Approach

First, copy the DCELs of two subdivisions.

Transform the result into a valid DCEL for the subdivision overlay.

- Compute the intersections of edges from different subdivisions.
- Link together appropriate parts of the two DCELs.
  1. Vertex and half-edge records.
  2. Face records.

Modify the plane sweep algorithm!
Line Sweep

new intersection
Line Sweep

new intersection

DCEL1
DCEL2
Line Sweep

new intersection

DCEL1
DCEL2

DCEL for the overlay
Line Sweep

Invariant: the part of overlay to left of the sweep line has been computed correctly.
At an Event Point

Update the event queue and sweep-line status tree as in the segment intersection algorithm.
At an Event Point

- Update the event queue and sweep-line status tree as in the segment intersection algorithm.

- In case the event point is
  - **Vertex** adjacent to edges from one subdivision.
At an Event Point

- Update the event queue and sweep-line status tree as in the segment intersection algorithm.

- In case the event point is
  - Vertex adjacent to edges from one subdivision.

  No additional work!
At an Event Point

- Update the event queue and sweep-line status tree as in the segment intersection algorithm.

- In case the event point is
  - **Vertex** adjacent to edges from one subdivision.
  - **Intersection** of edges from different subdivisions.

  *No additional work!*
At an Event Point

- Update the event queue and sweep-line status tree as in the segment intersection algorithm.

- In case the event point is
  - **Vertex** adjacent to edges from one subdivision.
    - No additional work!
  - **Intersection** of edges from different subdivisions.
    - Link the DCEL1 and DCEL2 at the intersection point.
    - Handle all possible cases.
II. Three Types of Edge Crossing
II. Three Types of Edge Crossing

**Vertex-Vertex**: two vertices from different subdivisions coincide with each other.
II. Three Types of Edge Crossing

**Vertex-Vertex**: two vertices from different subdivisions coincide with each other.

**Vertex-Edge**: an edge from one input subdivision passes through a vertex of another subdivision.
II. Three Types of Edge Crossing

**Vertex-Vertex**: two vertices from different subdivisions coincide with each other.

**Vertex-Edge**: an edge from one input subdivision passes through a vertex of another subdivision.

**Edge-Edge**: two edges from different subdivisions intersect in their interior.
II. Three Types of Edge Crossing

**Vertex-Vertex**: two vertices from different subdivisions coincide with each other.

**Vertex-Edge**: an edge from one input subdivision passes through a vertex of another subdivision.

**Edge-Edge**: two edges from different subdivisions intersect in their interior.

The other two cases are no more difficult.
An edge of one subdivision passes through a vertex of another subdivision.
An edge of one subdivision passes through a vertex of another subdivision.
An edge of one subdivision passes through a vertex of another subdivision.

Before:

DCEL1

2 old half-edges

DCEL2

After:

4 new half-edges
Operations in the Update
1. Edge $e$ splits into two edges $e'$ and $e''$ at intersection $v$. 
Operations in the Update

1. Edge $e$ splits into two edges $e'$ and $e''$ at intersection $v$. 
Operations in the Update

1. Edge $e$ splits into two edges $e'$ and $e''$ at intersection $v$.

2. Shorten half-edge $e_a = (u, w)$ to $e'_a = (u, v)$. 
Operations in the Update

1. Edge $e$ splits into two edges $e'$ and $e''$ at intersection $v$.

2. Shorten half-edge $e_a = (u, w)$ to $e_a' = (u, v)$.

3. Shorten half-edge $e_b = (w, u)$ to $e_a'' = (w, v)$.
Operations in the Update

1. Edge $e$ splits into two edges $e'$ and $e''$ at intersection $v$.

2. Shorten half-edge $e_a = (u, w)$ to $e_a' = (u, v)$.

3. Shorten half-edge $e_b = (w, u)$ to $e_a'' = (w, v)$.

4. Create their twin half-edges with $v$ as the origin. Update the Twin pointers.
Operations in the Update

1. Edge $e$ splits into two edges $e'$ and $e''$ at intersection $v$.

2. Shorten half-edge $e_a = (u, w)$ to $e_a' = (u, v)$.

3. Shorten half-edge $e_b = (w, u)$ to $e_a'' = (w, v)$.

4. Create their twin half-edges with $v$ as the origin. Update the Twin pointers.
5. Set the Next and Prev pointers of the four new half-edges.

Next\( (e_b') \leftarrow Next(e_b) \)
Next\( (e_b'') \leftarrow Next(e_a) \)
Prev\( (\text{Next}(e_b)) \leftarrow e_b' \)
Prev\( (\text{Next}(e_a)) \leftarrow e_b'' \)

Prev\( (e_a') \leftarrow \text{Prev}(e_a) \)
Prev\( (e_a'') \leftarrow \text{Prev}(e_b) \)
Next\( (\text{Prev}(e_a)) \leftarrow e_a' \)
Next\( (\text{Prev}(e_b)) \leftarrow e_a'' \)
Set the Next and Prev pointers of the four half-edges (colored red) incident to $v$ from the other subdivision.

Next($e''_a$) = ?

Since $e''_a$ has destination $v$, we need only set its Next pointer to the next edge on the face it bounds.
Set the Next and Prev pointers of the four half-edges (colored red) incident to $v$ from the other subdivision.

$\text{Next}(e''') = ?$

Since $e''$ has destination $v$, we need only set its Next pointer to the next edge on the face it bounds.

This is the first half-edge seen clockwise from $e'''$ with $v$ as its origin.
Previous and Next Edges

Set the Next and Prev pointers of the four half-edges (colored red) incident to \( v \) from the other subdivision.

\[ \text{Next}(e''_a) = ? \]

Since \( e''_a \) has destination \( v \), we need only set its Next pointer to the next edge on the face it bounds.

This is the first half-edge seen clockwise from \( e''_a \) with \( v \) as its origin.

How to find it using the DCEL2?
Set the Next and Prev pointers of the four half-edges (colored red) incident to $v$ from the other subdivision.

$$\text{Next}(e_\alpha'') = ?$$

Since $e_\alpha''$ has destination $v$, we need only set its Next pointer to the next edge on the face it bounds.

This is the first half-edge seen clockwise from $e_\alpha''$ with $v$ as its origin.

How to find it using the DCEL2?

Scan outgoing edges from $v$ in cyclic order.
How to find it using the DCEL2?

Set the Next and Prev pointers of the four half-edges (colored red) incident to \( v \) from the other subdivision.

\[
\text{Next}(e''') = ?
\]

Since \( e''' \) has destination \( v \), we need only set its Next pointer to the next edge on the face it bounds.

This is the first half-edge seen clockwise from \( e''' \) with \( v \) as its origin.

How to find it using the DCEL2?

Scan outgoing edges from \( v \) in cyclic order.

Similarly, Next(\( e'_a \)), Previous(\( e'_b \)), Previous(\( e''_b \))
How to find it using the DCEL2?

Set the Next and Prev pointers of the four half-edges (colored red) incident to \( v \) from the other subdivision.

Since \( e_a'' \) has destination \( v \), we need only set its Next pointer to the next edge on the face it bounds.

This is the first half-edge seen clockwise from \( e_a'' \) with \( v \) as its origin.

How to find it using the DCEL2?
Scan outgoing edges from \( v \) in cyclic order.

Similarly, \( \text{Next}(e_a'), \text{Previous}(e_b'), \text{Previous}(e_b'') \)
Time Cost for Updating Vertex and Half-Edge Records

Locating where $e'$, $e''$ appear in the cyclic order around $v$ takes time linear in the $\text{deg}(v)$. 
Locating where $e'$, $e''$ appear in the cyclic order around $v$ takes time linear in the $\text{deg}(v)$.

Each of the other operations takes time $O(1)$. 
Time Cost for Updating Vertex and Half-Edge Records

- Locating where $e'$, $e''$ appear in the cyclic order around $v$ takes time linear in the $\deg(v)$.

- Each of the other operations takes time $O(1)$.

$O(m)$ time where $m = \#$ edges (incident on the event point during the sweep).
Time Cost for Updating Vertex and Half-Edge Records

- Locating where $e', e''$ appear in the cyclic order around $v$ takes time linear in the $\text{deg}(v)$.

- Each of the other operations takes time $O(1)$.

$O(m)$ time where $m = \#$ edges (incident on the event point during the sweep).
Time Cost for Updating Vertex and Half-Edge Records

- Locating where $e'$, $e''$ appear in the cyclic order around $v$ takes time linear in the $\text{deg}(v)$.

- Each of the other operations takes time $O(1)$.

$O(m)$ time where $m = \# \text{ edges}$ (incident on the event point during the sweep).

Generalizes over the vertex-vertex and edge-edge cases.
Locating where $e', e''$ appear in the cyclic order around $v$ takes time linear in the $\deg(v)$.

Each of the other operations takes time $O(1)$.

$O(m)$ time where $m = \#\text{ edges (incident on the event point during the sweep)}$.

Generalizes over the vertex-vertex and edge-edge cases.

Updating vertex and half-edge records does not increase the asymptotic running time of the line segment intersection algorithm.
Time Cost for Updating Vertex and Half-Edge Records

Locating where $e', e''$ appear in the cyclic order around $v$ takes time linear in the $\text{deg}(v)$.

Each of the other operations takes time $O(1)$.

$O(m)$ time where $m = \# \text{ edges}$ (incident on the event point during the sweep).

Generalizes over the vertex-vertex and edge-edge cases.

Updating vertex and half-edge records does not increase the asymptotic running time of the line segment intersection algorithm.

every intersection is a vertex of the overlay.
Time Cost for Updating Vertex and Half-Edge Records

- Locating where $e'$, $e''$ appear in the cyclic order around $v$ takes time linear in $\text{deg}(v)$.

- Each of the other operations takes time $O(1)$.

$O(m)$ time where $m = \# \text{ edges}$ (incident on the event point during the sweep).

Generalizes over the vertex-vertex and edge-edge cases.

Updating vertex and half-edge records does not increase the asymptotic running time of the line segment intersection algorithm.

$O(n \log n + k \log n)$ every intersection is a vertex of the overlay.
Locating where $e'$, $e''$ appear in the cyclic order around $v$ takes time linear in the $\text{deg}(v)$.

Each of the other operations takes time $O(1)$.

$O(m)$ time where $m = \# \text{ edges}$ (incident on the event point during the sweep).

Generalizes over the vertex-vertex and edge-edge cases.

Updating vertex and half-edge records does not increase the asymptotic running time of the line segment intersection algorithm.

$O(n \log n + k \log n)$ every intersection is a vertex of the overlay.

combined complexity of 2 input subdivisions complexity of the overlay
III. Face Update

For each face $f$:

- $\text{OuterComponent}(f)$
- $\text{InnerComponents}(f)$
- $\text{IncidentFace}(e) \leftarrow f$, for each bounding half-edge $e$.
- label $f$ as $(F, G)$ where $F$ and $G$ are faces in the two old subdivisions that contain $f$.

#face records = 1 + #outer boundary cycles

Easy to extract all boundary cycles from DCEL.
Outer Boundary Cycle

How to distinguish an outer boundary cycle from one that bounds a hole?
Outer Boundary Cycle

How to distinguish an outer boundary cycle from one that bounds a hole?

Look at the leftmost vertex $v$ of the cycle with incident edges $e$ and $\text{Next}(e)$. 
Outer Boundary Cycle

How to distinguish an outer boundary cycle from one that bounds a hole?

Look at the leftmost vertex $\nu$ of the cycle with incident edges $e$ and $\text{Next}(e)$. 

Outer Boundary Cycle

How to distinguish an outer boundary cycle from one that bounds a hole?

Look at the leftmost vertex \( v \) of the cycle with incident edges \( e \) and \( \text{Next}(e) \).
Outer Boundary Cycle

How to distinguish an outer boundary cycle from one that bounds a hole?

Look at the leftmost vertex \( v \) of the cycle with incident edges \( e \) and Next(e).

Let \( \theta \) be the angle of rotation from \( e \) to Next \( (e) \).
Outer Boundary Cycle

How to distinguish an outer boundary cycle from one that bounds a hole?

Look at the leftmost vertex $v$ of the cycle with incident edges $e$ and $\text{Next}(e)$.

Let $\theta$ be the angle of rotation from $e$ to $\text{Next}(e)$. 
How to distinguish an outer boundary cycle from one that bounds a hole?

Look at the leftmost vertex \( v \) of the cycle with incident edges \( e \) and \( \text{Next}(e) \).

Let \( \theta \) be the angle of rotation from \( e \) to \( \text{Next}(e) \).

\[ \theta > \pi \Rightarrow \text{hole boundary} \]
How to distinguish an outer boundary cycle from one that bounds a hole?

Look at the leftmost vertex $v$ of the cycle with incident edges $e$ and $\text{Next}(e)$.

Let $\theta$ be the angle of rotation from $e$ to $\text{Next}(e)$.

- $\theta > \pi \Rightarrow$ hole boundary
Outer Boundary Cycle

How to distinguish an outer boundary cycle from one that bounds a hole?

Look at the leftmost vertex $v$ of the cycle with incident edges $e$ and $\text{Next}(e)$.

Let $\theta$ be the angle of rotation from $e$ to $\text{Next}(e)$.

$\diamond \theta > \pi \Rightarrow$ hole boundary
Outer Boundary Cycle

How to distinguish an outer boundary cycle from one that bounds a hole?

Look at the leftmost vertex \( v \) of the cycle with incident edges \( e \) and Next(\( e \)).

Let \( \theta \) be the angle of rotation from \( e \) to Next(\( e \)).

\[ \theta > \pi \Rightarrow \text{hole boundary} \]
\[ \theta < \pi \Rightarrow \text{outer boundary} \]
Outer Boundary Cycle

How to distinguish an outer boundary cycle from one that bounds a hole?

Look at the leftmost vertex $v$ of the cycle with incident edges $e$ and $\text{Next}(e)$.

Let $\theta$ be the angle of rotation from $e$ to $\text{Next}(e)$.

- $\theta > \pi \Rightarrow$ hole boundary
- $\theta < \pi \Rightarrow$ outer boundary
Outer Boundary Cycle

How to distinguish an outer boundary cycle from one that bounds a hole?

Look at the leftmost vertex $v$ of the cycle with incident edges $e$ and $\text{Next}(e)$.

Let $\theta$ be the angle of rotation from $e$ to $\text{Next}(e)$.

- $\theta > \pi \Rightarrow$ hole boundary
- $\theta < \pi \Rightarrow$ outer boundary

Compute $\theta$?
Outer Boundary Cycle

How to distinguish an outer boundary cycle from one that bounds a hole?

Look at the leftmost vertex $v$ of the cycle with incident edges $e$ and Next($e$).

Let $\theta$ be the angle of rotation from $e$ to Next($e$).

- $\theta > \pi \Rightarrow$ hole boundary
- $\theta < \pi \Rightarrow$ outer boundary

Compute $\theta$?

Use cross product: $e \times \text{Next}(e)$
Cycles Bounding the Same Face

Construct an undirected graph $G$ such that
Cycles Bounding the Same Face

Construct an undirected graph $G$ such that

- Every boundary cycle corresponds to a node $C_i$.

One node $C_\infty$ for the imaginary outer boundary cycle of the unbounded face.
Cycles Bounding the Same Face

Construct an undirected graph $G$ such that

- Every boundary cycle corresponds to a node $C_i$.
- One node $C_\infty$ for the imaginary outer boundary cycle of the unbounded face.
Cycles Bounding the Same Face

Construct an undirected graph $G$ such that

✦ Every boundary cycle corresponds to a node $C_i$.
✦ One node $C_\infty$ for the imaginary outer boundary cycle of the unbounded face.
✦ $(C_i, C_j)$ is an edge iff
  
  (a) one of the corresponding cycles is the boundary of a hole.
  
  (b) the other cycle has a half-edge immediately below the lowest vertex of the first cycle.
Graph Example

Five faces in total.
Graph Example

Five faces in total.
Graph Example

Five faces in total.
Graph Example

Five faces in total.
Five faces in total.
Graph Example

Five faces in total.

Inner boundary

Outer boundary
Five faces in total.

Graph $G$ induces the record for every face in DCEL ($O(n + k)$ time).
Construction of Graph $G$

Modify the plane sweep algorithm to construct $G$.

The algorithm checks the segment immediately below the event point.
Construction of Graph $G$

Modify the plane sweep algorithm to construct $G$.

The algorithm checks the segment immediately below the event point.

Make a node for every cycle.
Construction of Graph $G$

Modify the plane sweep algorithm to construct $G$.

The algorithm checks the segment immediately below the event point.

- Make a node for every cycle.
- When the event point $v$ is the lowest vertex bounding a hole $C'$.
Construction of Graph $G$

Modify the plane sweep algorithm to construct $G$.

The algorithm checks the segment immediately below the event point.

- Make a node for every cycle.
- When the event point $v$ is the lowest vertex bounding a hole $C'$

1. Locate the edge $e$ below $v$ and the cycle $C$ it is on.
Construction of Graph $G$

Modify the plane sweep algorithm to construct $G$.

The algorithm checks the segment immediately below the event point.

- Make a node for every cycle.
- When the event point $v$ is the lowest vertex bounding a hole $C'$

1. Locate the edge $e$ below $v$ and the cycle $C$ it is on.
**Construction of Graph $G$**

Modify the plane sweep algorithm to construct $G$.

The algorithm checks the segment immediately below the event point.

- Make a node for every cycle.
- When the event point $v$ is the lowest vertex bounding a hole $C'$
  1. Locate the edge $e$ below $v$ and the cycle $C$ it is on.
  2. Add an edge $(C, C')$ to $G$. 
Construction of Graph $G$

Modify the plane sweep algorithm to construct $G$.

The algorithm checks the segment immediately below the event point.

- Make a node for every cycle.
- When the event point $v$ is the lowest vertex bounding a hole $C'$
  
  1. Locate the edge $e$ below $v$ and the cycle $C$ it is on.
  2. Add an edge $(C, C')$ to $G$. 
Construction of Graph $G$

Modify the plane sweep algorithm to construct $G$.

The algorithm checks the segment immediately below the event point.

- Make a node for every cycle.
- When the event point $v$ is the lowest vertex bounding a hole $C'$
  
  1. Locate the edge $e$ below $v$ and the cycle $C$ it is on.
  2. Add an edge $(C, C')$ to $G$.

Maintain a pointer from every half-edge to the node in $G$ representing the cycle it is on.
Construction of Graph $G$

Modify the plane sweep algorithm to construct $G$.

The algorithm checks the segment immediately below the event point.

- Make a node for every cycle.
- When the event point $v$ is the lowest vertex bounding a hole $C'$
  
  1. Locate the edge $e$ below $v$ and the cycle $C$ it is on.
  2. Add an edge $(C, C')$ to $G$.

Maintain a pointer from every half-edge to the node in $G$ representing the cycle it is on.
Labeling a Face

Every face in the overlay is labeled with the names of the faces in the old subdivisions that contain the face.
Labeling a Face

Every face in the overlay is labeled with the names of the faces in the old subdivisions that contain the face.

Consider an arbitrary vertex $\nu$: 
Labeling a Face

Every face in the overlay is labeled with the names of the faces in the old subdivisions that contain the face.

Consider an arbitrary vertex $v$: 

intersection of edges from different subdivisions.
Labeling a Face

Every face in the overlay is labeled with the names of the faces in the old subdivisions that contain the face.

Consider an arbitrary vertex $v$:

- intersection of edges from different subdivisions.
- Look up the IncidentFace() pointer of the two corresponding half-edges.
Labeling a Face

Every face in the overlay is labeled with the names of the faces in the old subdivisions that contain the face.

Consider an arbitrary vertex $v$:

\[ \text{intersection of edges from different subdivisions.} \]

Look up the IncidentFace() pointer of the two corresponding half-edges.
Labeling a Face

Every face in the overlay is labeled with the names of the faces in the old subdivisions that contain the face.

Consider an arbitrary vertex $v$:

Look up the IncidentFace() pointer of the two corresponding half-edges.
Labeling a Face

Every face in the overlay is labeled with the names of the faces in the old subdivisions that contain the face.

Consider an arbitrary vertex $v$:

\[ (f, g) e_1 e_2 g \]

intersection of edges from different subdivisions.

Look up the `IncidentFace()` pointer of the two corresponding half-edges.
Labeling a Face

Every face in the overlay is labeled with the names of the faces in the old subdivisions that contain the face.

Consider an arbitrary vertex $v$:

- intersection of edges from different subdivisions.
- Look up the IncidentFace( ) pointer of the two corresponding half-edges.
- existing vertex of one subdivision.
Labeling a Face

Every face in the overlay is labeled with the names of the faces in the old subdivisions that contain the face.

Consider an arbitrary vertex $v$:

- Intersection of edges from different subdivisions.
- Look up the IncidentFace() pointer of the two corresponding half-edges.
- Existing vertex of one subdivision.
- Know only one generating face – the one from the same subdivision. Therefore,
Labeling a Face

Every face in the overlay is labeled with the names of the faces in the old subdivisions that contain the face.

Consider an arbitrary vertex $v$:

- Intersection of edges from different subdivisions.

  Look up the IncidentFace() pointer of the two corresponding half-edges.

- Existing vertex of one subdivision.

  Know only one generating face – the one from the same subdivision. Therefore,

  *for each vertex in one subdivision, we keep track of its containing face in the other subdivision.*
Labeling a Face

Every face in the overlay is labeled with the names of the faces in the old subdivisions that contain the face.

Consider an arbitrary vertex $v$: 

- intersection of edges from different subdivisions.
- Look up the `IncidentFace()` pointer of the two corresponding half-edges.
- existing vertex of one subdivision.
- Know only one generating face – the one from the same subdivision. Therefore,

*for each vertex in one subdivision, we keep track of its containing face in the other subdivision.*

Plane sweep again (or do it in the same sweep).
IV. The Overlay Algorithm

\textbf{MapOverlay}(S_1, S_2)

\textbf{Input}: two planar subdivisions \(S_1\) and \(S_2\) stored in DCELs.

\textbf{Output}: the overlay of \(S_1\) and \(S_2\) in a DCEL \(D\).

1. Copy the DCELs for \(S_1\) and \(S_2\) into a new DCEL \(D\).
2. Use plane sweep to compute all intersections between edges from \(S_1\) and \(S_2\).

While updating the event queue \(Q\) and sweep-line status \(T\), do the following:
- Update vertex and edge records in \(D\) whenever the event involves edges from both \(S_1\) and \(S_2\). Let the corresponding vertex be \(v\).
- Store the half-edge immediately below the event point at \(v\).

3. Traverse \(D\) (using depth-first search) to determine all boundary cycles.
4. Construct the graph \(G\).
5. for each connected component in \(G\)
6. do \(C \leftarrow\) the unique outer boundary cycle
   \(f \leftarrow\) the face bounded by \(C\)
   create a face record for \(f\)
   OuterComponent(\(f\)) \(\leftarrow\) some half-edge of \(C\)
   InnerComponents(\(f\)) \(\leftarrow\) pointers to one half-edge \(e\) in each hole
   IncidentFace(\(e\)) \(\leftarrow f\) for all half-edges bounding the cycle \(C\) and holes
7. Label each face in the overlay.
IV. The Overlay Algorithm

MapOverlay($S_1, S_2$)  // total complexity of DCELs is $n$

**Input:** two planar subdivisions $S_1$ and $S_2$ stored in DCELs.

**Output:** the overlay of $S_1$ and $S_2$ in a DCEL $D$.

1. Copy the DCELs for $S_1$ and $S_2$ into a new DCEL $D$.
2. Use plane sweep to compute all intersections between edges from $S_1$ and $S_2$.

While updating the event queue $Q$ and sweep-line status $T$, do the following:

- Update vertex and edge records in $D$ whenever the event involves edges from both $S_1$ and $S_2$. Let the corresponding vertex be $v$.
- Store the half-edge immediately below the event point at $v$.

3. Traverse $D$ (using depth-first search) to determine all boundary cycles.
4. Construct the graph $G$
5. for each connected component in $G$
6. do $C \leftarrow$ the unique outer boundary cycle
   $f \leftarrow$ the face bounded by $C$
   create a face record for $f$
   OuterComponent($f$) $\leftarrow$ some half-edge of $C$
   InnerComponents($f$) $\leftarrow$ pointers to one half-edge $e$ in each hole
   IncidentFace($e$) $\leftarrow f$ for all half-edges bounding the cycle $C$ and holes
7. Label each face in the overlay.
IV. The Overlay Algorithm

\( \text{MapOverlay}(S_1, S_2) \)  \hspace{1em} // \text{total complexity of DCELs is } n

**Input:** two planar subdivisions \( S_1 \) and \( S_2 \) stored in DCELs.

**Output:** the overlay of \( S_1 \) and \( S_2 \) in a DCEL \( D \).

1. Copy the DCELs for \( S_1 \) and \( S_2 \) into a new DCEL \( D \).
2. Use plane sweep to compute all intersections between edges from \( S_1 \) and \( S_2 \).

While updating the event queue \( Q \) and sweep-line status \( T \), do the following:

- Update vertex and edge records in \( D \) whenever the event involves edges from both \( S_1 \) and \( S_2 \). Let the corresponding vertex be \( v \).
- Store the half-edge immediately below the event point at \( v \).

3. Traverse \( D \) (using depth-first search) to determine all boundary cycles.
4. Construct the graph \( G \)
5. for each connected component in \( G \)
6. \hspace{1em} do \hspace{1em} \( C \leftarrow \) the unique outer boundary cycle
   \hspace{1em} \( f \leftarrow \) the face bounded by \( C \)
   \hspace{1em} create a face record for \( f \)
   \hspace{1em} OuterComponent(\( f \)) \leftarrow \) some half-edge of \( C \)
   \hspace{1em} InnerComponents(\( f \)) \leftarrow \) pointers to one half-edge \( e \) in each hole
   \hspace{1em} IncidentFace(\( e \)) \leftarrow \( f \) for all half-edges bounding the cycle \( C \) and holes
7. Label each face in the overlay.
IV. The Overlay Algorithm

MapOverlay($S_1, S_2$)  // total complexity of DCELs is $n$

**Input:** two planar subdivisions $S_1$ and $S_2$ stored in DCELs.

**Output:** the overlay of $S_1$ and $S_2$ in a DCEL $D$.

1. Copy the DCELs for $S_1$ and $S_2$ into a new DCEL $D$. $\ll O(n)$
2. Use plane sweep to compute all intersections between edges from $S_1$ and $S_2$.

While updating the event queue $Q$ and sweep-line status $T$, do the following:

- Update vertex and edge records in $D$ whenever the event involves edges from both $S_1$ and $S_2$. Let the corresponding vertex be $v$.
- Store the half-edge immediately below the event point at $v$.

3. Traverse $D$ (using depth-first search) to determine all boundary cycles.
4. Construct the graph $G$
5. for each connected component in $G$
6. do $C \leftarrow$ the unique outer boundary cycle
7. $f \leftarrow$ the face bounded by $C$
8. create a face record for $f$
9. OuterComponent($f$)$\leftarrow$ some half-edge of $C$
10. InnerComponents($f$)$\leftarrow$ pointers to one half-edge $e$ in each hole
11. IncidentFace($e$)$\leftarrow f$ for all half-edges bounding the cycle $C$ and holes
12. Label each face in the overlay.
IV. The Overlay Algorithm

MapOverlay($S_1, S_2$) // total complexity of DCELs is $n$

Input: two planar subdivisions $S_1$ and $S_2$ stored in DCELs.

Output: the overlay of $S_1$ and $S_2$ in a DCEL $D$.

1. Copy the DCELs for $S_1$ and $S_2$ into a new DCEL $D$. // $O(n)$
2. Use plane sweep to compute all intersections between edges from $S_1$ and $S_2$.

While updating the event queue $Q$ and sweep-line status $T$, do the following:

- Update vertex and edge records in $D$ whenever the event involves edges from both $S_1$ and $S_2$. Let the corresponding vertex be $v$.
- Store the half-edge immediately below the event point at $v$.

3. Traverse $D$ (using depth-first search) to determine all boundary cycles.
4. Construct the graph $G$
5. for each connected component in $G$
6. do $C \leftarrow$ the unique outer boundary cycle
   $f \leftarrow$ the face bounded by $C$
   create a face record for $f$
   OuterComponent($f$) $\leftarrow$ some half-edge of $C$
   InnerComponents($f$) $\leftarrow$ pointers to one half-edge $e$ in each hole
   IncidentFace($e$) $\leftarrow f$ for all half-edges bounding the cycle $C$ and holes
7. Label each face in the overlay.
IV. The Overlay Algorithm

MapOverlay($S_1, S_2$) \hspace{1em} // total complexity of DCELs is $n$

**Input:** two planar subdivisions $S_1$ and $S_2$ stored in DCELs.

**Output:** the overlay of $S_1$ and $S_2$ in a DCEL $D$.

1. Copy the DCELs for $S_1$ and $S_2$ into a new DCEL $D$. \hspace{1em} // $O(n)$
2. Use plane sweep to compute all intersections between edges from $S_1$ and $S_2$. \hspace{1em} // $O(n \log n + k \log n)$ where $k$ is the complexity of the overlay.

While updating the event queue $Q$ and sweep-line status $T$, do the following:

- Update vertex and edge records in $D$ whenever the event involves edges from both $S_1$ and $S_2$. Let the corresponding vertex be $v$.
- Store the half-edge immediately below the event point at $v$.

3. Traverse $D$ (using depth-first search) to determine all boundary cycles.
4. Construct the graph $G$
5. for each connected component in $G$
6. do $C \leftarrow$ the unique outer boundary cycle
   $f \leftarrow$ the face bounded by $C$
   create a face record for $f$
   OuterComponent($f$) $\leftarrow$ some half-edge of $C$
   InnerComponents($f$) $\leftarrow$ pointers to one half-edge $e$ in each hole
   IncidentFace($e$) $\leftarrow f$ for all half-edges bounding the cycle $C$ and holes

7. Label each face in the overlay.
IV. The Overlay Algorithm

MapOverlay($S_1, S_2$) // total complexity of DCELs is $n$

**Input:** two planar subdivisions $S_1$ and $S_2$ stored in DCELs.

**Output:** the overlay of $S_1$ and $S_2$ in a DCEL $D$. // $O(n)$

1. Copy the DCELs for $S_1$ and $S_2$ into a new DCEL $D$. // $O(n \log n + k \log n)$ where $k$ is the complexity of the overlay.
2. Use plane sweep to compute all intersections between edges from $S_1$ and $S_2$. 

While updating the event queue $Q$ and sweep-line status $T$, do the following:

- Update vertex and edge records in $D$ whenever the event involves edges from both $S_1$ and $S_2$. Let the corresponding vertex be $v$.
- Store the half-edge immediately below the event point at $v$.

3. Traverse $D$ (using depth-first search) to determine all boundary cycles.
4. Construct the graph $G$
5. for each connected component in $G$
6. do $C \leftarrow$ the unique outer boundary cycle 
   $f \leftarrow$ the face bounded by $C$
   create a face record for $f$
   OuterComponent($f$) $\leftarrow$ some half-edge of $C$
   InnerComponents($f$) $\leftarrow$ pointers to one half-edge $e$ in each hole
   IncidentFace($e$) $\leftarrow f$ for all half-edges bounding the cycle $C$ and holes
7. Label each face in the overlay.
IV. The Overlay Algorithm

\texttt{MapOverlay}(S_1, S_2) \quad // \text{total complexity of DCELs is } n

\textbf{Input:} two planar subdivisions \(S_1\) and \(S_2\) stored in DCELs.

\textbf{Output:} the overlay of \(S_1\) and \(S_2\) in a DCEL \(D\).

1. Copy the DCELs for \(S_1\) and \(S_2\) into a new DCEL \(D\). \(\quad // O(n)\)

2. Use plane sweep to compute all intersections between edges from \(S_1\) and \(S_2\).
   \(\quad // O(n \log n + k \log n)\) where \(k\) is the complexity of the overlay.

While updating the event queue \(Q\) and sweep-line status \(T\), do the following:

\begin{itemize}
  \item Update vertex and edge records in \(D\) whenever the event involves
  edges from both \(S_1\) and \(S_2\). Let the corresponding vertex be \(v\).
  \item Store the half-edge immediately below the event point at \(v\).
\end{itemize}

3. Traverse \(D\) (using depth-first search) to determine all boundary cycles. \(\quad // O(k)\)

4. Construct the graph \(G\)

5. for each connected component in \(G\)

6. \hspace{1em} do \(C \leftarrow\) the unique outer boundary cycle
   \hspace{1em} \(f \leftarrow\) the face bounded by \(C\)
   \hspace{1em} create a face record for \(f\)
   \hspace{1em} OuterComponent(\(f\)) \leftarrow\) some half-edge of \(C\)
   \hspace{1em} InnerComponents(\(f\)) \leftarrow\) pointers to one half-edge \(e\) in each hole
   \hspace{1em} IncidentFace(\(e\)) \leftarrow f\) for all half-edges bounding the cycle \(C\) and holes

7. Label each face in the overlay.
IV. The Overlay Algorithm

MapOverlay($S_1, S_2$)  // total complexity of DCELs is $n$

**Input:** two planar subdivisions $S_1$ and $S_2$ stored in DCELs.

**Output:** the overlay of $S_1$ and $S_2$ in a DCEL $D$.

1. Copy the DCELs for $S_1$ and $S_2$ into a new DCEL $D$.  // $O(n)$
2. Use plane sweep to compute all intersections between edges from $S_1$ and $S_2$.  
   // $O(n \log n + k \log n)$ where $k$ is the complexity of the overlay.

While updating the event queue $Q$ and sweep-line status $T$, do the following:

- Update vertex and edge records in $D$ whenever the event involves
  edges from both $S_1$ and $S_2$. Let the corresponding vertex be $v$.
- Store the half-edge immediately below the event point at $v$.

3. Traverse $D$ (using depth-first search) to determine all boundary cycles.  // $O(k)$
4. Construct the graph $G$
5. for each connected component in $G$
6. do $C \leftarrow$ the unique outer boundary cycle
7. $f \leftarrow$ the face bounded by $C$
8. create a face record for $f$
9. OuterComponent($f$) $\leftarrow$ some half-edge of $C$
10. InnerComponents($f$) $\leftarrow$ pointers to one half-edge $e$ in each hole
11. IncidentFace($e$) $\leftarrow f$ for all half-edges bounding the cycle $C$ and holes
7. Label each face in the overlay.
IV. The Overlay Algorithm

MapOverlay($S_1, S_2$) // total complexity of DCELs is $n$

**Input:** two planar subdivisions $S_1$ and $S_2$ stored in DCELs.

**Output:** the overlay of $S_1$ and $S_2$ in a DCEL $D$.

1. Copy the DCELs for $S_1$ and $S_2$ into a new DCEL $D$. // $O(n)$
2. Use plane sweep to compute all intersections between edges from $S_1$ and $S_2$. // $O(n \log n + k \log n)$ where $k$ is the complexity of the overlay.

While updating the event queue $Q$ and sweep-line status $T$, do the following:
- Update vertex and edge records in $D$ whenever the event involves edges from both $S_1$ and $S_2$. Let the corresponding vertex be $v$.
- Store the half-edge immediately below the event point at $v$.

3. Traverse $D$ (using depth-first search) to determine all boundary cycles. // $O(k)$

4. Construct the graph $G$

5. for each connected component in $G$

6. do $C \leftarrow$ the unique outer boundary cycle

   $f \leftarrow$ the face bounded by $C$

   create a face record for $f$

   OuterComponent($f$) $\leftarrow$ some half-edge of $C$

   InnerComponents($f$) $\leftarrow$ pointers to one half-edge $e$ in each hole

   IncidentFace($e$) $\leftarrow f$ for all half-edges bounding the cycle $C$ and holes

7. Label each face in the overlay.
IV. The Overlay Algorithm

MapOverlay($S_1, S_2$)  // total complexity of DCELs is $n$

**Input:** two planar subdivisions $S_1$ and $S_2$ stored in DCELs.

**Output:** the overlay of $S_1$ and $S_2$ in a DCEL $D$.

1. Copy the DCELs for $S_1$ and $S_2$ into a new DCEL $D$.  // $O(n)$

2. Use plane sweep to compute all intersections between edges from $S_1$ and $S_2$.  // $O(n \log n + k \log n)$ where $k$ is the complexity of the overlay.

While updating the event queue $Q$ and sweep-line status $T$, do the following:

- Update vertex and edge records in $D$ whenever the event involves edges from both $S_1$ and $S_2$. Let the corresponding vertex be $v$.
- Store the half-edge immediately below the event point at $v$.

3. Traverse $D$ (using depth-first search) to determine all boundary cycles.  // $O(k)$

4. Construct the graph $G$

5. for each connected component in $G$

6. do $C \leftarrow$ the unique outer boundary cycle

   $f \leftarrow$ the face bounded by $C$

   create a face record for $f$

   OuterComponent($f$) $\leftarrow$ some half-edge of $C$

   InnerComponents($f$) $\leftarrow$ pointers to one half-edge $e$ in each hole

   IncidentFace($e$) $\leftarrow f$ for all half-edges bounding the cycle $C$ and holes

7. Label each face in the overlay.  // $O(n \log n + k \log n)$
IV. The Overlay Algorithm

\( \text{MapOverlay}(S_1, S_2) \)  // total complexity of DCELs is \( n \)

**Input:** two planar subdivisions \( S_1 \) and \( S_2 \) stored in DCELs.

**Output:** the overlay of \( S_1 \) and \( S_2 \) in a DCEL \( D \).

1. Copy the DCELs for \( S_1 \) and \( S_2 \) into a new DCEL \( D \). \( \mathcal{O}(n) \)
2. Use plane sweep to compute all intersections between edges from \( S_1 \) and \( S_2 \).
   \( \mathcal{O}(n \log n + k \log n) \) where \( k \) is the complexity of the overlay.

While updating the event queue \( Q \) and sweep-line status \( T \), do the following:

- Update vertex and edge records in \( D \) whenever the event involves
  edges from both \( S_1 \) and \( S_2 \). Let the corresponding vertex be \( v \).
- Store the half-edge immediately below the event point at \( v \).

3. Traverse \( D \) (using depth-first search) to determine all boundary cycles. \( \mathcal{O}(k) \)
4. Construct the graph \( G \)
5. for each connected component in \( G \)
6. do \( C \leftarrow \) the unique outer boundary cycle
   \( f \leftarrow \) the face bounded by \( C \)
   create a face record for \( f \)
   OuterComponent(\( f \)) \( \leftarrow \) some half-edge of \( C \)
   InnerComponents(\( f \)) \( \leftarrow \) pointers to one half-edge \( e \) in each hole
   IncidentFace(\( e \)) \( \leftarrow f \) for all half-edges bounding the cycle \( C \) and holes
7. Label each face in the overlay. \( \mathcal{O}(n \log n + k \log n) \)
Theorem: The overlay of two planar subdivisions with total complexity $n$ can be constructed in $O(n \log n + k \log n)$, where $k$ is the complexity of the overlay.
Boolean Operations

Operations on polygonal regions:
Boolean Operations

Operations on polygonal regions:
Boolean Operations

Operations on polygonal regions:
Boolean Operations

Operations on polygonal regions:
Boolean Operations

Operations on polygonal regions:

$P \cup Q$
Boolean Operations

Operations on polygonal regions:

$P \cup Q$

faces labeled $P$ or $Q$. 
Boolean Operations

Operations on polygonal regions:

\[ P \cup Q \]

faces labeled \( P \) or \( Q \).
Boolean Operations

Operations on polygonal regions:

$P \cup Q$

faces labeled $P$ or $Q$

$P \cap Q$
Boolean Operations

Operations on polygonal regions:

$P \cup Q$  \hspace{1cm} faces labeled $P$ or $Q$

$P \cap Q$
Boolean Operations

Operations on polygonal regions:

$P \cup Q$

faces labeled $P$ or $Q$.

$P \cap Q$
Boolean Operations

Operations on polygonal regions:

- $P \cup Q$: faces labeled $P$ or $Q$
- $P \cap Q$: faces labeled $(P, Q)$
Boolean Operations

Operations on polygonal regions:

- \( P \cup Q \) (faces labeled \( P \) or \( Q \))
- \( P \cap Q \) (faces labeled \( (P, Q) \))
Boolean Operations

Operations on polygonal regions:

$P \cup Q$
- faces labeled $P$ or $Q$.

$P \cap Q$
- faces labeled $(P, Q)$. 
Boolean Operations

Operations on polygonal regions:

- $P \cup Q$: faces labeled $P$ or $Q$.
- $P \cap Q$: faces labeled $(P, Q)$.
- $P - Q$.
Boolean Operations

Operations on polygonal regions:

- $P \cup Q$: $P$ or $Q$, faces labeled $P$ or $Q$.
- $P \cap Q$: $P, Q$, faces labeled $(P, Q)$.
- $P - Q$: $P$ minus $Q$. 
Boolean Operations

Operations on polygonal regions:

- $P \cup Q$: faces labeled $P$ or $Q$
- $P \cap Q$: faces labeled $(P, Q)$
- $P - Q$
Boolean Operations

Operations on polygonal regions:

- $P \cup Q$: faces labeled $P$ or $Q$.
- $P \cap Q$: faces labeled $(P, Q)$.
- $P - Q$: faces labeled $P$ but not $Q$. 
Boolean Operations

Operations on polygonal regions:

- $P \cup Q$: faces labeled $P$ or $Q$, $n$ vertices in total
- $P \cap Q$: faces labeled $(P, Q)$
- $P - Q$: $P$ but not $Q$
Boolean Operations

Operations on polygonal regions:

- \( P \cup Q \): faces labeled \( P \) or \( Q \)
  - \( n \) vertices in total
  - \( k \) complexity of overlay

- \( P \cap Q \): faces labeled \( (P, Q) \)

- \( P - Q \): faces labeled \( P \) but not \( Q \)
Boolean Operations

Operations on polygonal regions:

- $P$ or $Q$
- but not $Q$

$P \cup Q$

Faces labeled $P$ or $Q$

$n$ vertices in total

$k$ complexity of overlay

$P \cap Q$

Faces labeled $(P, Q)$

Corollary $P \cup Q$, $P \cap Q$, and $P - Q$ can each be computed in time $O(n \log n + k \log n)$. 

$P - Q$

Faces labeled $P$ but not $Q$
Boolean Operations

Operations on polygonal regions:

- $P \cup Q$: faces labeled $P$ or $Q$.
- $P \cap Q$: faces labeled $(P, Q)$.
- $P - Q$: faces labeled $P$ but not $Q$.

$n$ vertices in total
$k$ complexity of overlay

**Corollary** $P \cup Q$, $P \cap Q$, and $P - Q$ can each be computed in time $O(n \log n + k \log n)$. 