DCEL: Doubly-Connected Edge List

Claudio Mirolo

Dip. di Scienze Matematiche, Informatiche e Fisiche
Università di Udine, via delle Scienze 206 – Udine

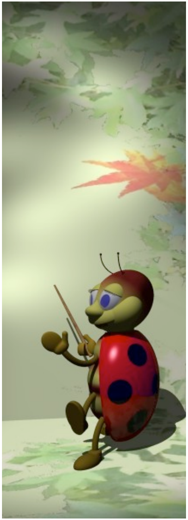
claudio.mirolo@uniud.it

Computational Geometry

users.dimi.uniud.it/∼claudio.mirolo

Outline

1. DCEL data structure
   - representation
   - example
2. Application of DCELs
   - overlay of two subdivisions
   - plane sweep: edges and vertices
   - plane sweep: faces
3. Further annotations



What about?

- Planar subdivision
- Planar embedding of a graph
  - node → *vertex*
  - arc → *edge*
  - + *face*

What about?

- Planar subdivision
- Planar embedding of a graph
  - node $\rightarrow$ *vertex*
  - arc $\rightarrow$ *edge*
  - $+$ *face*

What about?

- Planar subdivision
- Planar embedding of a graph
  - node $\rightarrow$ *vertex*
  - arc $\rightarrow$ *edge*
  - $+$ *face*

What about?

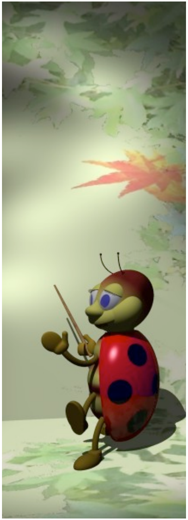
- Planar subdivision
- Planar embedding of a graph
  - node $\rightarrow$ *vertex*
  - arc $\rightarrow$ *edge*
  - $+$ *face*

What about?

- Planar subdivision
- Planar embedding of a graph
  - node $\rightarrow$ *vertex*
  - arc $\rightarrow$ *edge*
  - $+$ *face*

Outline

1. DCEL data structure
   - representation
   - example
2. Application of DCELs
   - overlay of two subdivisions
   - plane sweep: edges and vertices
   - plane sweep: faces
3. Further annotations



DCEL: Basic ingredients

Basic items:

- *vertices*
- *edges* = line segments
- *faces*

As well as topological relationships between such items: *incidence*

DCEL: Basic ingredients

Basic items:

- *vertices*
- *edges* = line segments
- *faces*

As well as topological relationships between such items: *incidence*

DCEL: Basic ingredients

Basic items:

- *vertices*
- *edges* = line segments
- *faces*

As well as topological relationships between such items: *incidence*

DCEL: Basic access operations

Basic *local* access operations:

- walk around the boundary of a given face
- move from a face to an adjacent one (via a common edge)
- visit all the edges around a given vertex

Possibly, store/get additional data associated to, e.g., a face: *attribute information*

DCEL: Basic access operations

Basic *local* access operations:

- walk around the boundary of a given face
- move from a face to an adjacent one (via a common edge)
- visit all the edges around a given vertex

Possibly, store/get additional data associated to, e.g., a face: *attribute information*

DCEL: Basic access operations

Basic *local* access operations:

- walk around the boundary of a given face
- move from a face to an adjacent one (via a common edge)
- visit all the edges around a given vertex

Possibly, store/get additional data associated to, e.g., a face: *attribute information*

DCEL: Basic access operations

Basic *local* access operations:

- walk around the boundary of a given face
- move from a face to an adjacent one (via a common edge)
- visit all the edges around a given vertex

Possibly, store/get additional data associated to, e.g., a face: *attribute information*

DCEL: Representational tricks

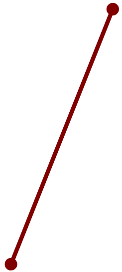
One record for each item. . .

- edge → two directed *half-edges* . . .
- . . . → *twin* half-edges
- face lies to the *left* of its bounding half-edges . . .

DCEL: Representational tricks

One record for each item. . .

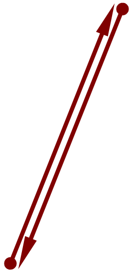
- edge $\rightarrow$ two directed *half-edges* . . .
- . . . $\rightarrow$ *twin* half-edges
- face lies to the *left* of its bounding half-edges . . .



DCEL: Representational tricks

One record for each item…

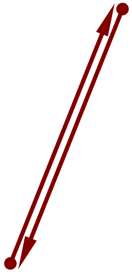
- edge $\rightarrow$ two directed *half-edges* …
- … $\rightarrow$ *twin* half-edges
- face lies to the *left* of its bounding half-edges …



DCEL: Representational tricks

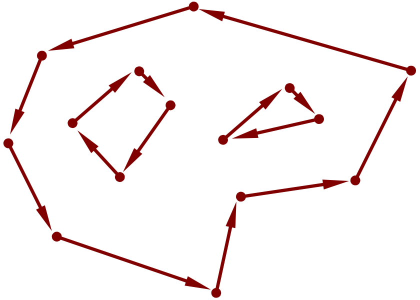
One record for each item...

- edge $\rightarrow$ two directed *half-edges* ...
- ... $\rightarrow$ *twin* half-edges
- face lies to the *left* of its bounding half-edges ...



DCEL: Representational tricks

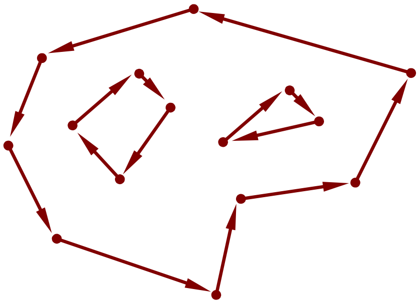
- . . .   including its possible *holes*



Remark “walking” direction!

DCEL: Representational tricks

- . . .   including its possible *holes*



Remark “walking” direction!

DCEL: Vertices

Vertex record v :

- pair of coordinates (x, y)
- pointer to (any) incident half-edge, with v as its origin

DCEL: Vertices

Vertex record v :

- pair of coordinates (x, y)
- pointer to (any) incident half-edge, with v as its origin

DCEL: Vertices

Vertex record v :

- pair of coordinates (x, y)
- pointer to (any) incident half-edge, with v as its origin

DCEL: Vertices

Vertex record v :

- pair of coordinates (x, y)
- pointer to (any) incident half-edge, with v as its origin

DCEL: Faces

Face record f :

- pointer to (any) half-edge of its outer boundary, with f lying to the left of this half-edge
- list of pointers to (any) half-edge of each inner boundary, with f still lying to the left of these half-edges

DCEL: Faces

Face record f :

- pointer to (any) half-edge of its outer boundary, with f lying to the left of this half-edge
- list of pointers to (any) half-edge of each inner boundary, with f still lying to the left of these half-edges

DCEL: Faces

Face record f :

- pointer to (any) half-edge of its outer boundary, with f lying to the left of this half-edge
- list of pointers to (any) half-edge of each inner boundary, with f still lying to the left of these half-edges

DCEL: Faces

Face record f :

- pointer to (any) half-edge of its outer boundary, with f lying to the left of this half-edge
- list of pointers to (any) half-edge of each inner boundary, with f still lying to the left of these half-edges

DCEL: Faces

Face record f :

- pointer to (any) half-edge of its outer boundary, with f lying to the left of this half-edge
- list of pointers to (any) half-edge of each inner boundary, with f still lying to the left of these half-edges

DCEL: Half-edges

Half-edge record e :

- *source* vertex (origin of e)
- incident face, i.e. face to its left
- *twin* half-edge
- *next* half-edge, along boundary of incident face
- *previous* half-edge, along same boundary

DCEL: Half-edges

Half-edge record e :

- *source* vertex (origin of e)
- incident face, i.e. face to its left
- *twin* half-edge
- *next* half-edge, along boundary of incident face
- *previous* half-edge, along same boundary

DCEL: Half-edges

Half-edge record e :

- *source* vertex (origin of e)
- incident face, i.e. face to its left
- *twin* half-edge
- *next* half-edge, along boundary of incident face
- *previous* half-edge, along same boundary

DCEL: Half-edges

Half-edge record e :

- *source* vertex (origin of e)
- incident face, i.e. face to its left
- *twin* half-edge
- *next* half-edge, along boundary of incident face
- *previous* half-edge, along same boundary

DCEL: Half-edges

Half-edge record e :

- *source* vertex (origin of e)
- incident face, i.e. face to its left
- *twin* half-edge
- *next* half-edge, along boundary of incident face
- *previous* half-edge, along same boundary

DCEL: Half-edges

Half-edge record e :

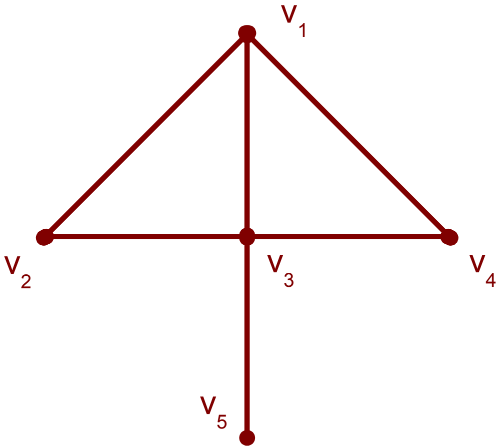
- *source* vertex (origin of e)
- incident face, i.e. face to its left
- *twin* half-edge
- *next* half-edge, along boundary of incident face
- *previous* half-edge, along same boundary

DCEL: Half-edges

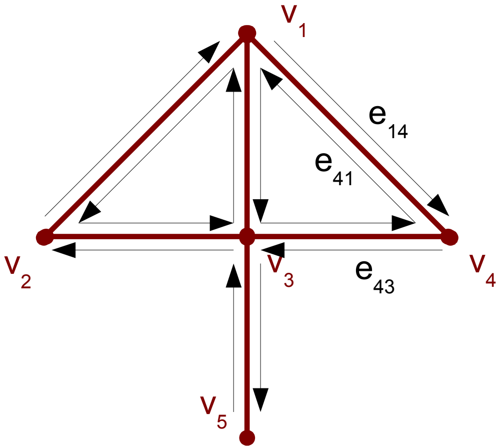
Half-edge record e :

- *source* vertex (origin of e)
- incident face, i.e. face to its left
- *twin* half-edge
- *next* half-edge, along boundary of incident face
- *previous* half-edge, along same boundary

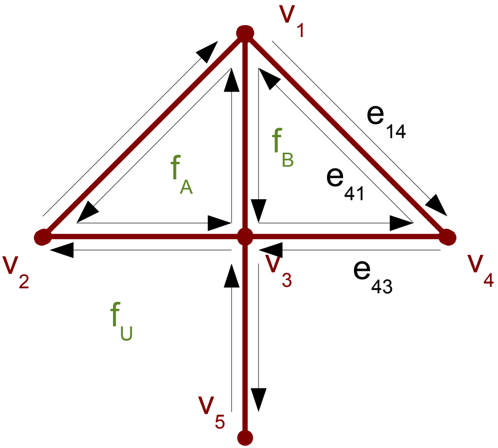
Example: Vertices + Half-edges + Faces



Example: Vertices + Half-edges + Faces

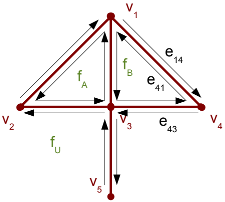


Example: Vertices + Half-edges + Faces



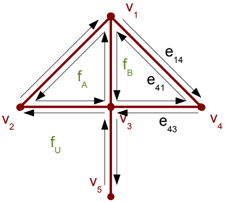
Vertex records

	coord	edge
v_1	$(0.0, 2.5)$	e_{12}
v_2	$(-2.5, 0.0)$	e_{23}
v_3	$(0.0, 0.0)$	e_{35}
v_4	$(2.5, 0.0)$	e_{41}
v_5	$(0.0, -2.5)$	??



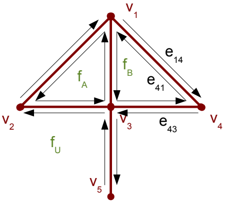
Vertex records

	coord	edge
v_1	$(0.0, 2.5)$	e_{12}
v_2	$(-2.5, 0.0)$	e_{23}
v_3	$(0.0, 0.0)$	e_{35}
v_4	$(2.5, 0.0)$	e_{41}
v_5	$(0.0, -2.5)$	e_{53}



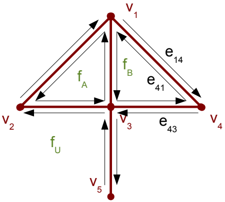
Face records

	outer	inner
f_U	—	$< e_{32} >$
f_A	e_{12}	??
f_B	e_{13}	??



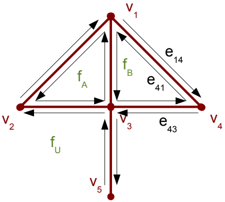
Face records

	outer	inner
f_U	—	$< e_{32} >$
f_A	e_{12}	$<>$
f_B	e_{13}	$<>$



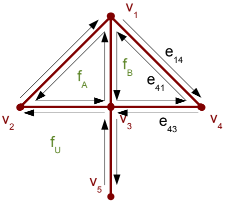
Edge records

	src	face	next	prev	twin
e_{12}	v_1	f_A	e_{23}	e_{31}	e_{21}
e_{23}	v_2	f_A	e_{31}	e_{12}	e_{32}
e_{31}	v_3	f_A	e_{12}	e_{23}	e_{13}
e_{13}	v_1		??		
e_{34}	v_3				
e_{41}	v_4				



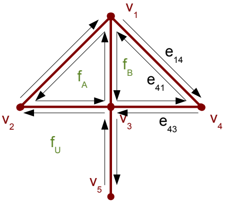
Edge records

	src	face	next	prev	twin
e_{12}	v_1	f_A	e_{23}	e_{31}	e_{21}
e_{23}	v_2	f_A	e_{31}	e_{12}	e_{32}
e_{31}	v_3	f_A	e_{12}	e_{23}	e_{13}
e_{13}	v_1	f_B	e_{34}	e_{41}	e_{31}
e_{34}	v_3	f_B	e_{41}	e_{13}	e_{43}
e_{41}	v_4	f_B	e_{13}	e_{34}	e_{14}



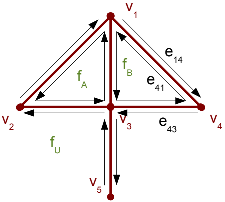
Edge records (continued)

	src	face	next	prev	twin
e_{14}	v_1	f_U	e_{43}	e_{21}	e_{41}
e_{43}	v_4	f_U	e_{35}	e_{14}	e_{34}
??			??		
??			??		
e_{32}	v_3	f_U	e_{21}	e_{53}	e_{23}
e_{21}	v_2	f_U	e_{14}	e_{32}	e_{12}



Edge records (continued)

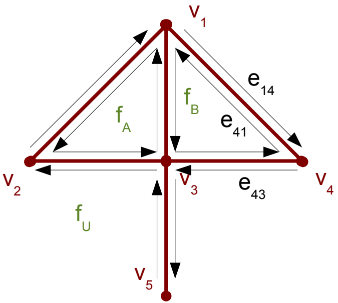
	src	face	next	prev	twin
e_{14}	v_1	f_U	e_{43}	e_{21}	e_{41}
e_{43}	v_4	f_U	e_{35}	e_{14}	e_{34}
e_{35}	v_3	f_U	e_{53}	e_{43}	e_{53}
e_{53}	v_5	f_U	e_{32}	e_{35}	e_{35}
e_{32}	v_3	f_U	e_{21}	e_{53}	e_{23}
e_{21}	v_2	f_U	e_{14}	e_{32}	e_{12}



Questions. . .

How to visit all the edges around a vertex, say v_3, in counterclockwise order?

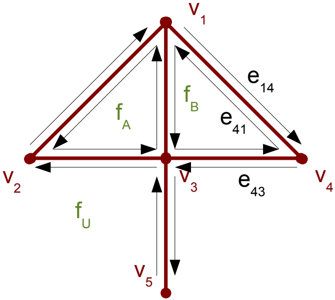
	coord	edge
v_1	...	
v_2	...	
v_3	$(0.0, 0.0)$	e_{35}
v_4	...	
v_5	...	



Questions. . .

How to identify all faces adjacent to a given one, say f_B ?

	outer	inner
f_U	...	
f_A	...	
f_B	e_{13}	<>

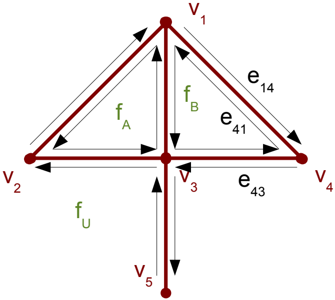


Can the *same* face be met more than once?

Questions. . .

How to identify all faces adjacent to a given one, say f_B ?

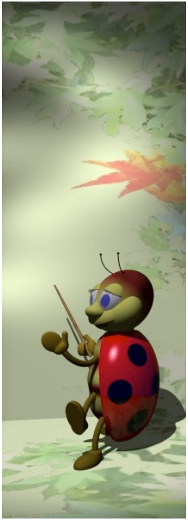
	outer	inner
f_U	. . .	
f_A	. . .	
f_B	e_{13}	<>



Can the *same* face be met more than once?

Outline

1. DCEL data structure
   - representation
   - example
2. Application of DCELs
   - overlay of two subdivisions
   - plane sweep: edges and vertices
   - plane sweep: faces
3. Further annotations



What does *overlay* mean?

Overlay of two subdivisions S_1 and S_2 :

a face f belongs to the overlay $Ov(S_1, S_2)$

if and only if f is a maximal connected subset of $f_1 \cap f_2$

for $f_1 \in S_1$ and $f_2 \in S_2$

Here faces = open sets

What does *overlay* mean?

Overlay of two subdivisions S_1 and S_2 :

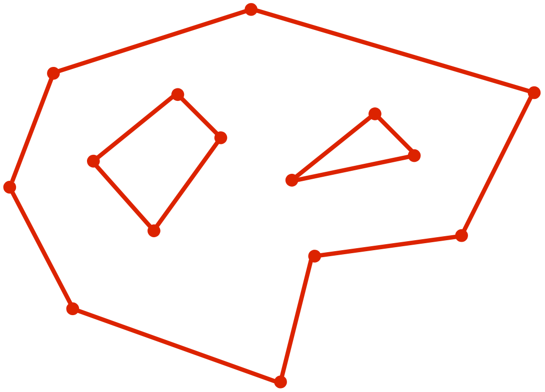
a face f belongs to the overlay $Ov(S_1, S_2)$

if and only if f is a maximal connected subset of $f_1 \cap f_2$

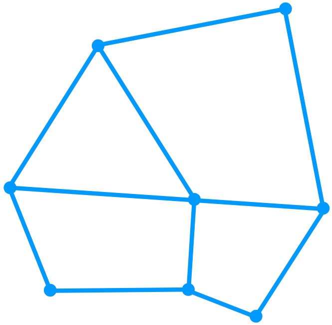
for $f_1 \in S_1$ and $f_2 \in S_2$

Here faces = open sets

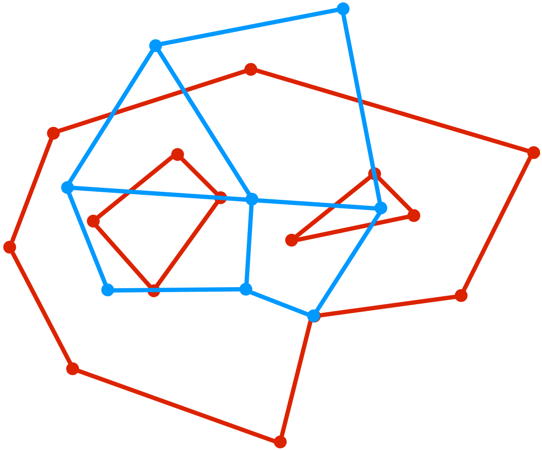
Overlaying subdivisions: S_1



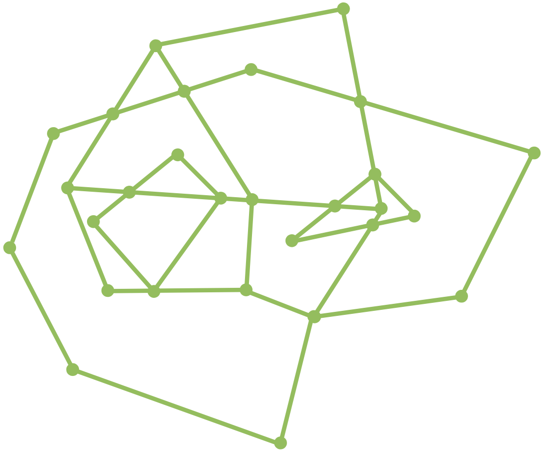
Overlaying subdivisions: S_2



Overlaying subdivisions: $S_1 + S_2$



Overlaying subdivisions: $Ov(S_1, S_2)$



Approach

Much information about the edges can be reused

Approach:

- load a copy of the DCELs of S_1 and S_2
  (not a valid DCEL as such)
- process the resulting network of edges and vertices:
  split edges and link parts together as appropriate
- rebuild face records and assign half-edges' face fields

Approach

Much information about the edges can be reused

Approach:

- load a copy of the DCELs of S_1 and S_2
  (not a valid DCEL as such)
- process the resulting network of edges and vertices:
  split edges and link parts together as appropriate
- rebuild face records and assign half-edges' face fields

Approach

Much information about the edges can be reused

Approach:

- load a copy of the DCELs of S_1 and S_2 (not a valid DCEL as such)
- process the resulting network of edges and vertices: split edges and link parts together as appropriate
- rebuild face records and assign half-edges' face fields

Approach

Much information about the edges can be reused

Approach:

- load a copy of the DCELs of S_1 and S_2 (not a valid DCEL as such)
- process the resulting network of edges and vertices: split edges and link parts together as appropriate
- rebuild face records and assign half-edges' face fields

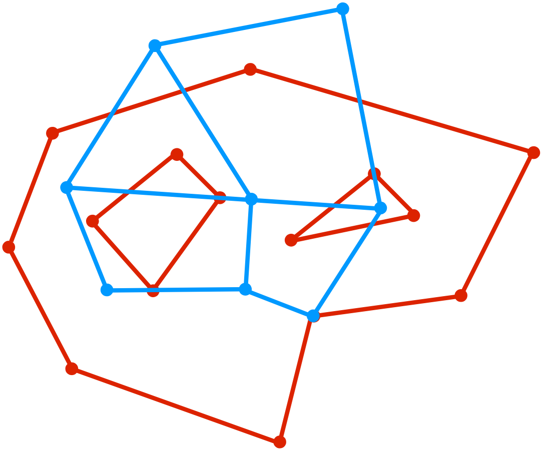
Approach

Much information about the edges can be reused

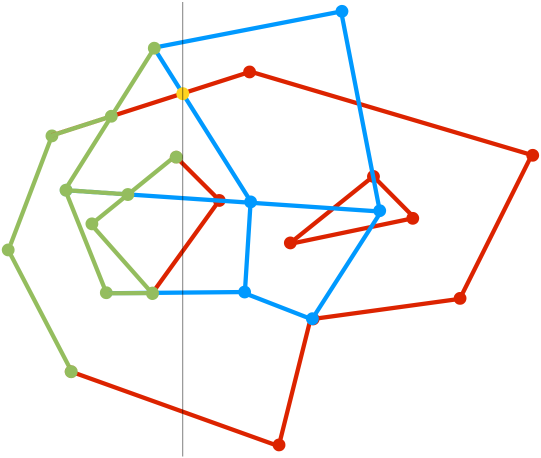
Approach:

- load a copy of the DCELs of S_1 and S_2
  (not a valid DCEL as such)
- process the resulting network of edges and vertices:
  split edges and link parts together as appropriate
- rebuild face records and assign half-edges' face fields

Plane sweep



Plane sweep



Plane sweep

- *Invariant*:
  valid DCEL of $Ov(S_1, S_2)$ on the left of the sweep line
  (except from face information, to be processed later)

- event queue (EQ) and sweep line structure (SL)
  treated as usual. . .

  - all edges incident at event point p come from the same original subdivision: can be used as it is

  - event point p involves items of both subdivisions: "tedious but not difficult" local updates

Plane sweep

- *Invariant*:
  valid DCEL of $Ov(S_1, S_2)$ on the left of the sweep line
  (except from face information, to be processed later)

- event queue (*EQ*) and sweep line structure (*SL*)
  treated as usual…

  - all edges incident at event point p come from
    the same original subdivision:

  - event point p involves items of both subdivisions:

Plane sweep

- *Invariant*:
  valid DCEL of $Ov(S_1, S_2)$ on the left of the sweep line
  (except from face information, to be processed later)

- event queue (EQ) and sweep line structure (SL)
  treated as usual. . .
  - all edges incident at event point p come from
    the same original subdivision: can be used as it is
  - event point p involves items of both subdivisions:
    "tedious but not difficult" local updates

Plane sweep

- *Invariant*:
  valid DCEL of $Ov(S_1, S_2)$ on the left of the sweep line
  (except from face information, to be processed later)

- event queue (*EQ*) and sweep line structure (*SL*)
  treated as usual...
  - all edges incident at event point p come from
    the same original subdivision: can be used as it is
  - event point p involves items of both subdivisions:
    "tedious but not difficult" local updates

Plane sweep

- *Invariant*:
  valid DCEL of $Ov(S_1, S_2)$ on the left of the sweep line
  (except from face information, to be processed later)

- event queue (*EQ*) and sweep line structure (*SL*)
  treated as usual...
  - all edges incident at event point *p* come from
    the same original subdivision: can be used as it is
  - event point *p* involves items of both subdivisions:
    "tedious but not difficult" local updates

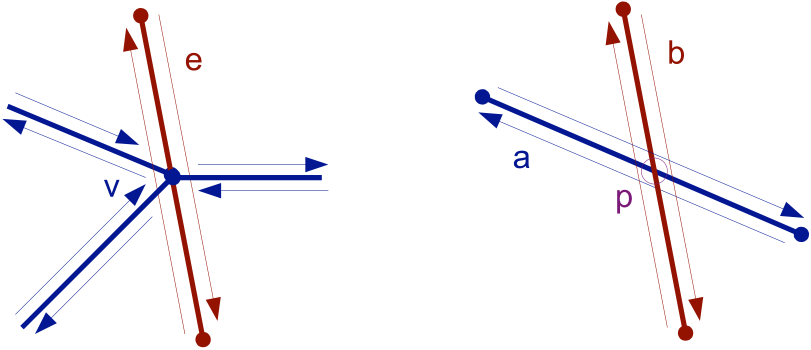
Plane sweep

- *Invariant*:
  valid DCEL of $Ov(S_1, S_2)$ on the left of the sweep line (except from face information, to be processed later)
- event queue (EQ) and sweep line structure (SL) treated as usual...
  - all edges incident at event point p come from the same original subdivision: can be used as it is
  - event point p involves items of both subdivisions: "tedious but not difficult" local updates

Plane sweep

- *Invariant*:
  valid DCEL of $Ov(S_1, S_2)$ on the left of the sweep line
  (except from face information, to be processed later)
- event queue (*EQ*) and sweep line structure (*SL*)
  treated as usual. . .
  - all edges incident at event point p come from
    the same original subdivision: can be used as it is
  - event point p involves items of both subdivisions:
    "tedious but not difficult" local updates

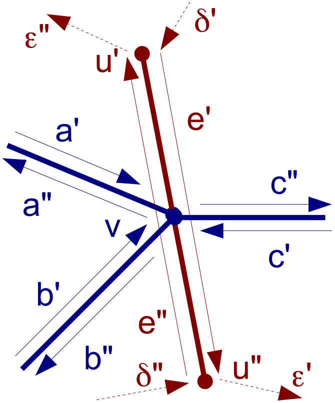
Which local updates?



What must be changed?

	coord	edge
v	...	a''
u'	...	β
u''	...	e''

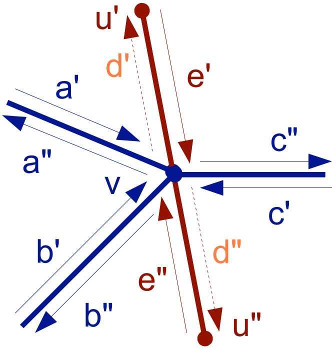
	src	...	next	prev	twin
e'	u'	...	ε'	δ'	e''
e''	u''	...	ε''	δ''	e'
a'	ϕ	...	c''	α	a''
c''	v	...	γ	a'	c'



What must be changed?

	coord	edge
v	...	a''
u'	...	β
u''	...	e''

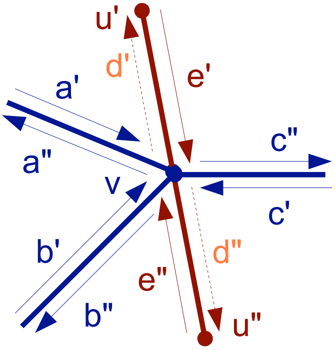
	src	...	next	prev	twin
e'	u'	...	ε'	δ'	e''
e''	u''	...	ε''	δ''	e'
a'	ϕ	...	c''	α	a''
c''	v	...	γ	a'	c'



And how?

	coord	edge
v	...	a''
u'	...	β
u''	...	e''

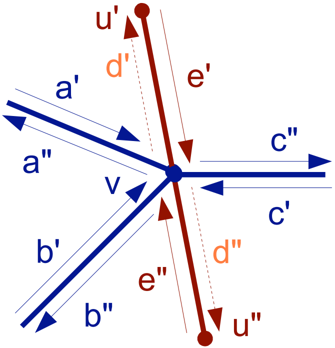
	src	...	next	prev	twin
e'	u'	...	ε'	δ'	e''
e''	u''	...	ε''	δ''	e'
a'	ϕ	...	c''	α	a''
c''	v	...	γ	a'	c'



How to find *next/prev*? How much does it cost?

	coord	edge
v	...	a''
u'	...	β
u''	...	e''

	src	...	next	prev	twin
e'	u'	...	c''	δ'	d'
e''	u''	...	b''	δ''	d''
a'	ϕ	...	d'	α	a''
c''	v	...	γ	e'	c'
d'	v	...	ε''	a'	e'
d''	v	...	ε'	c'	e''



Event types

Events, in summary:

- vertex of S_1 only (nothing to do)
- edge of S_1 passing through a vertex of S_2
- ...and symmetric configurations, of course
- vertex of S_1 and S_2
- intersection of edges of S_1 and S_2

Event types

Events, in summary:

- vertex of S_1 only (nothing to do)
- edge of S_1 passing through a vertex of S_2
- . . . and symmetric configurations, of course
- vertex of S_1 and S_2
- intersection of edges of S_1 and S_2

Event types

Events, in summary:

- vertex of S_1 only (nothing to do)
- edge of S_1 passing through a vertex of S_2
- . . . and symmetric configurations, of course
- vertex of S_1 and S_2
- intersection of edges of S_1 and S_2

Event types

Events, in summary:

- vertex of S_1 only (nothing to do)
- edge of S_1 passing through a vertex of S_2
- ... and symmetric configurations, of course
- vertex of S_1 and S_2
- intersection of edges of S_1 and S_2

Event types

Events, in summary:

- vertex of S_1 only (nothing to do)
- edge of S_1 passing through a vertex of S_2
- ... and symmetric configurations, of course
- vertex of S_1 and S_2
- intersection of edges of S_1 and S_2

Event types

Events, in summary:

- vertex of S_1 only (nothing to do)
- edge of S_1 passing through a vertex of S_2
- ... and symmetric configurations, of course
- vertex of S_1 and S_2
- intersection of edges of S_1 and S_2

Analysis (. . . so far)

Vertex u of $Ov(S_1, S_2)$ . . .

- local changes: $O(\ deg(u)\ )$

  $\rightarrow \quad O(\ \sum_u\ deg(u)\ ) = O(\ n+k\ )$ [formula di Eulero]

- overall: $O(\ (n+k)\log n\ )$

n complexity of $S_1 + S_2$;
k (additional) complexity of $Ov(S_1, S_2)$

Analysis (. . . so far)

Vertex u of $Ov(S_1, S_2)$ . . .

- local changes: $O(\ deg(u)\ )$

  $\rightarrow \quad O(\ \sum_u deg(u)\ ) = O(\ n + k\ )$ [formula di Eulero]

- overall: $O(\ (n + k) \log n\ )$

n complexity of $S_1 + S_2$;

k (additional) complexity of $Ov(S_1, S_2)$

Analysis (...so far)

Vertex u of $Ov(S_1, S_2)$ ...

- local changes: $O(\ deg(u)\ )$

$$\rightarrow \quad O(\ \sum_u deg(u)\ ) \ = \ O(\ n + k\ )$$ [formula di Eulero]

- overall: $O(\ (n+k) \log n\ )$

n complexity of $S_1 + S_2$;

k (additional) complexity of $Ov(S_1, S_2)$

Analysis (...so far)

Vertex u of $Ov(S_1, S_2)$ ...

- local changes: $O(\ deg(u)\ )$

  $\rightarrow \quad O(\ \sum_u\ deg(u)\ ) \ = \ O(\ n + k\ )$ [formula di Eulero]

- overall: $O(\ (n + k) \log n\ )$

n complexity of $S_1 + S_2$;

k (additional) complexity of $Ov(S_1, S_2)$

Analysis (...so far)

Vertex u of $Ov(S_1, S_2)$ ...

- local changes: $O(\ deg(u)\ )$

  $\rightarrow \quad O(\ \sum_u\ deg(u)\ ) \ = \ O(\ n + k\ )$ [formula di Eulero]

- overall: $O(\ (n+k) \log n\ )$

n complexity of S_1 + S_2;

k (additional) complexity of $Ov(S_1, S_2)$

Re-processing faces: integrations

Faces:

- outer boundary
- list of inner boundaries

Half-edges:

- incident face
- everything else is already computed

Re-processing faces: integrations

Faces:

- outer boundary
- list of inner boundaries

Half-edges:

- incident face
- everything else is already computed

Re-processing faces: getting information

How many faces?

- as many as the outer boundaires…
- …+ 1 (unbounded face)

How to find the boundary cycles?

- graph: half-edges + next/prev links
- connected components

Re-processing faces: getting information

How many faces?

- as many as the outer boundaires...
- ... + 1 (unbounded face)

How to find the boundary cycles?

- graph: half-edges + next/prev links
- connected components

Re-processing faces: getting information

How many faces?

- as many as the outer boundaires...
- ... + 1 (unbounded face)

How to find the boundary cycles?

- graph: half-edges + next/prev links
- connected components

Re-processing faces: getting information

How many faces?

- as many as the outer boundaires. . .
- . . . + 1 (unbounded face)

How to find the boundary cycles?

- graph: half-edges + next/prev links
- connected components

Re-processing faces: getting information

How to know if a boundary is “outer” or “inner”?

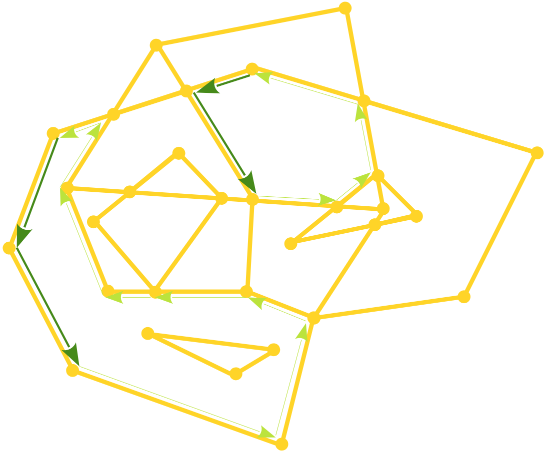
- follow the direction of half-edges
- at the (lexicographically) leftmost vertex
- does next half-edge turn left or right?

Re-processing faces: getting information

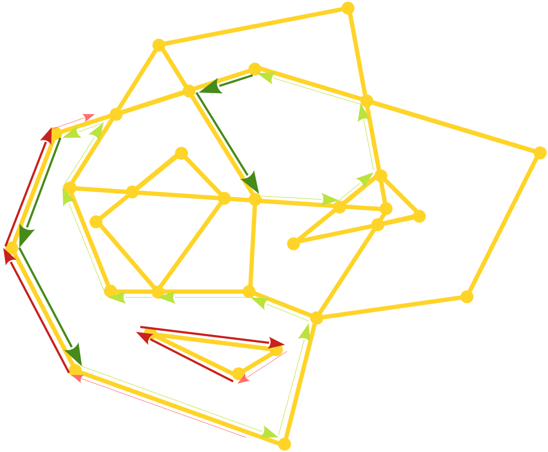
How to know if a boundary is “outer” or “inner”?

- follow the direction of half-edges
- at the (lexicographically) leftmost vertex
- does next half-edge turn left or right?

Outer boundaries vs. inner boundaries



Outer boundaries vs. inner boundaries



Re-processing faces: face records

Face records. . .

- create face records for each outer boundary and for the unbounded face
- set the corresponding field to point to a half-edge of the outer boundary
- set the incident face field accordingly for each half-edge of the outer boundary

Re-processing faces: face records

Face records…

- create face records for each outer boundary and for the unbounded face
- set the corresponding field to point to a half-edge of the outer boundary
- set the incident face field accordingly for each half-edge of the outer boundary

Re-processing faces: face records

Face records. . .

- create face records for each outer boundary and for the unbounded face
- set the corresponding field to point to a half-edge of the outer boundary
- set the incident face field accordingly for each half-edge of the outer boundary

Re-processing faces: outer vs. inner boundaries

Which disconnected edge cycles bound the same face?

- during plane sweep, register the neighbor edge below the leftmost vertex of each inner boundary cycle
- such items are adjacent along *SL* !
- The resulting links connect (directly or indirectly) inner and outer boundaries of a same face…

Re-processing faces: outer vs. inner boundaries

Which disconnected edge cycles bound the same face?

- during plane sweep, register the neighbor edge below the leftmost vertex of each inner boundary cycle
- such items are adjacent along *SL* !
- The resulting links connect (directly or indirectly) inner and outer boundaries of a same face…

Re-processing faces: outer vs. inner boundaries

Which disconnected edge cycles bound the same face?

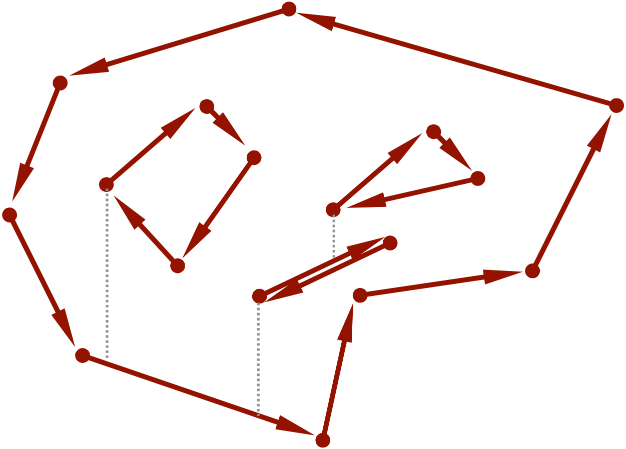
- during plane sweep, register the neighbor edge below the leftmost vertex of each inner boundary cycle
- such items are adjacent along *SL* !
- The resulting links connect (directly or indirectly) inner and outer boundaries of a same face...

Re-processing faces: outer vs. inner boundaries

Which disconnected edge cycles bound the same face?

- during plane sweep, register the neighbor edge below the leftmost vertex of each inner boundary cycle
- such items are adjacent along *SL* !
- The resulting links connect (directly or indirectly) inner and outer boundaries of a same face. . .

Graph of boundary cycles



Connected components of the graph

Are such links sound and complete?

- The interval on the sweep line between a vertex and the linked edge must belong to just one face, hence both related cycles bound the same face
- The leftmost vertex of each inner boundary must be linked to some component, which must bound the same face

Connected components of the graph

Are such links sound and complete?

- The interval on the sweep line between a vertex and the linked edge must belong to just one face, hence both related cycles bound the same face
- The leftmost vertex of each inner boundary must be linked to some component, which must bound the same face

Connected components of the graph

Are such links sound and complete?

- The interval on the sweep line between a vertex and the linked edge must belong to just one face, hence both related cycles bound the same face
- The leftmost vertex of each inner boundary must be linked to some component, which must bound the same face

Further processing

- Set the incident face field for each half-edge of each inner boundary

- We may want to link each face of the new subdivision with the two overlapping original faces
(information gathered from plane sweep processing. . .)

Further processing

- Set the incident face field for each half-edge of each inner boundary
- We may want to link each face of the new subdivision with the two overlapping original faces
  (information gathered from plane sweep processing. . .)

Further processing

- Set the incident face field for each half-edge of each inner boundary
- We may want to link each face of the new subdivision with the two overlapping original faces (information gathered from plane sweep processing. . .)

Analysis (overall)

- additional cross pointers to work efficiently
- face information within edge and face records can be set in $O(\, n + k \,)$ after the plane sweep
- overall costs sum up to: $O(\, (n + k) \log n \,)$

where $n = |\, S_1 \,| + |\, S_2 \,|$; $k = |\, Ov(S_1, S_2) \,|$

Analysis (overall)

- additional cross pointers to work efficiently
- face information within edge and face records can be set in $O(\ n + k\ )$ after the plane sweep
- overall costs sum up to: $O(\ (n + k) \log n\ )$

where $n = |\ S_1\ | + |\ S_2\ |$; $k = |\ Ov(S_1, S_2)\ |$

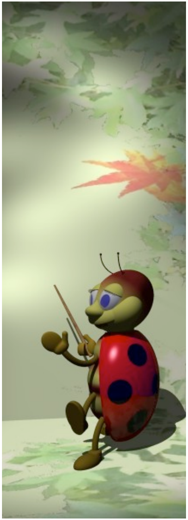
Analysis (overall)

- additional cross pointers to work efficiently
- face information within edge and face records can be set in $O(\ n+k\ )$ after the plane sweep
- overall costs sum up to: $O(\ (n+k)\log n\ )$

where $n = |\ S_1\ | + |\ S_2\ |;\quad k = |\ Ov(S_1, S_2)\ |$

Outline

1. DCEL data structure
   - representation
   - example
2. Application of DCELs
   - overlay of two subdivisions
   - plane sweep: edges and vertices
   - plane sweep: faces
3. Further annotations



Set operations

As special cases of the above technique:

- union of (simple) polygons
- intersection of polygons
- difference of polygons
- all in $O(\ (n+k)\log n\ )$
- results are not always polygons

Set operations

As special cases of the above technique:

- union of (simple) polygons
- intersection of polygons
- difference of polygons
- all in $O(\ (n+k)\log n\ )$
- results are not always polygons

Set operations

As special cases of the above technique:

- union of (simple) polygons
- intersection of polygons
- difference of polygons
- all in $O(\ (n+k)\log n\ )$
- results are not always polygons

Riferimenti

D.E. Muller & F.P. Preparata (1978)
Finding the Intersection of Two Convex Polyhedra
Theoretical Computer Science, 7(2)