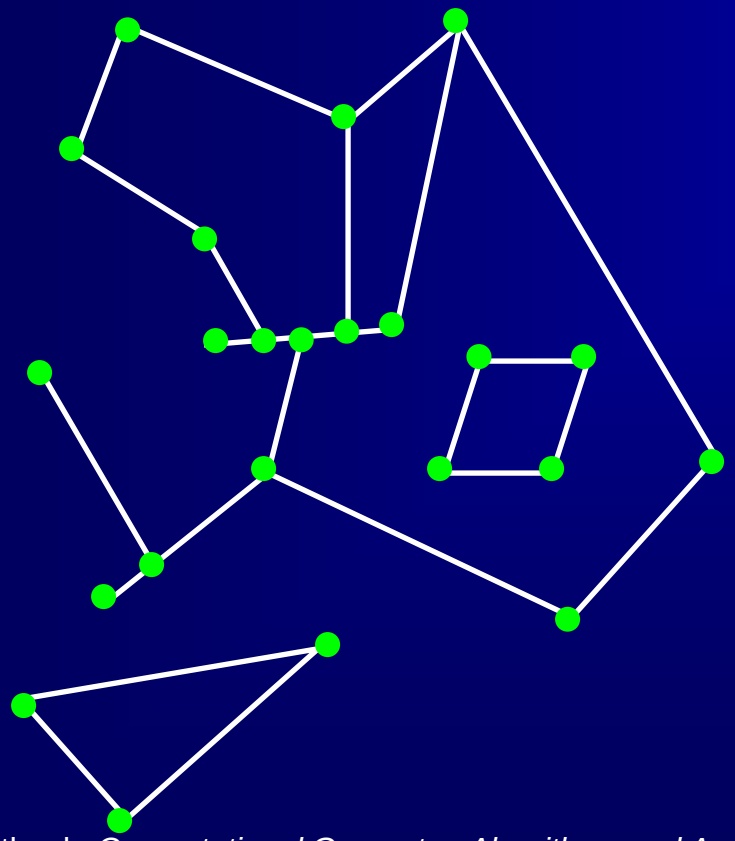
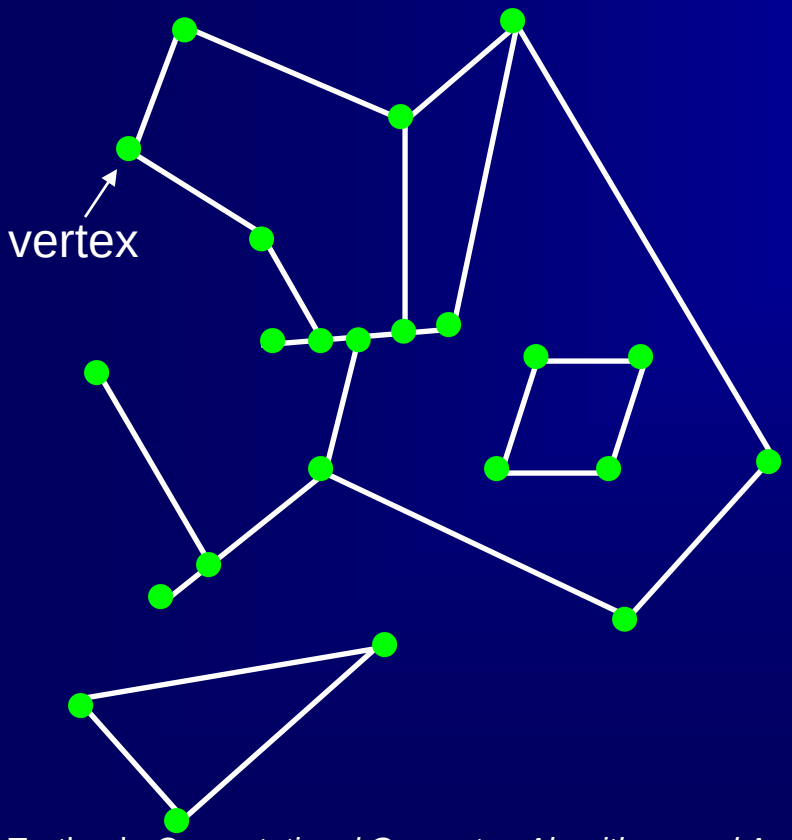
Planar Subdivision

Induced by planar embedding of a graph.



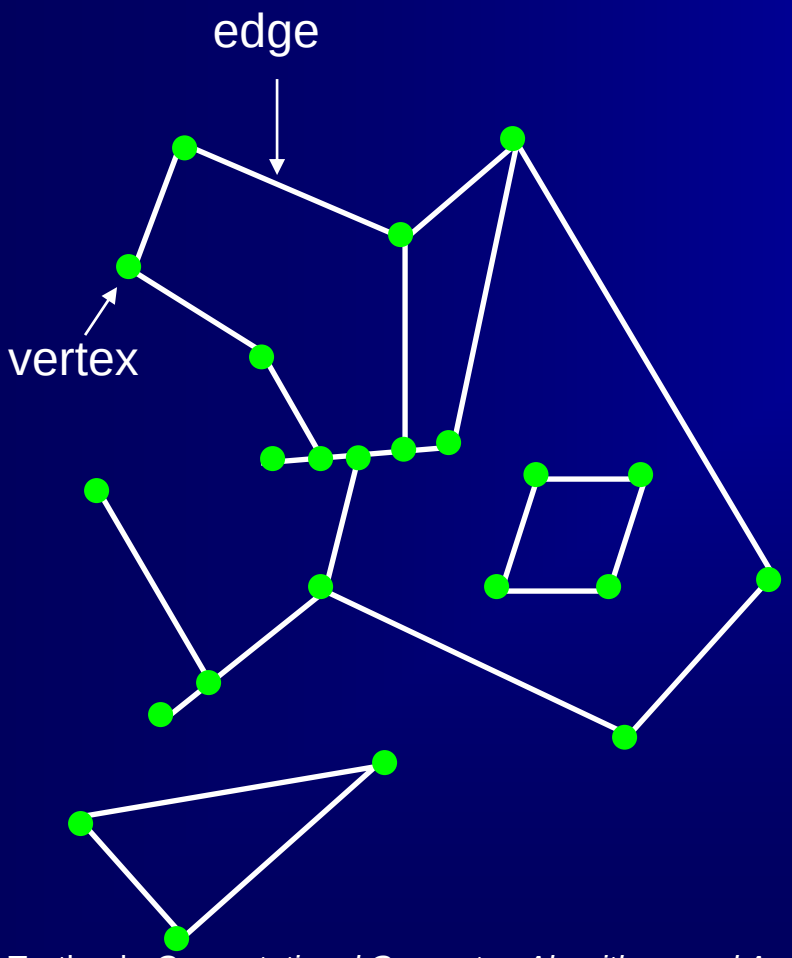
Planar Subdivision

Induced by planar embedding of a graph.



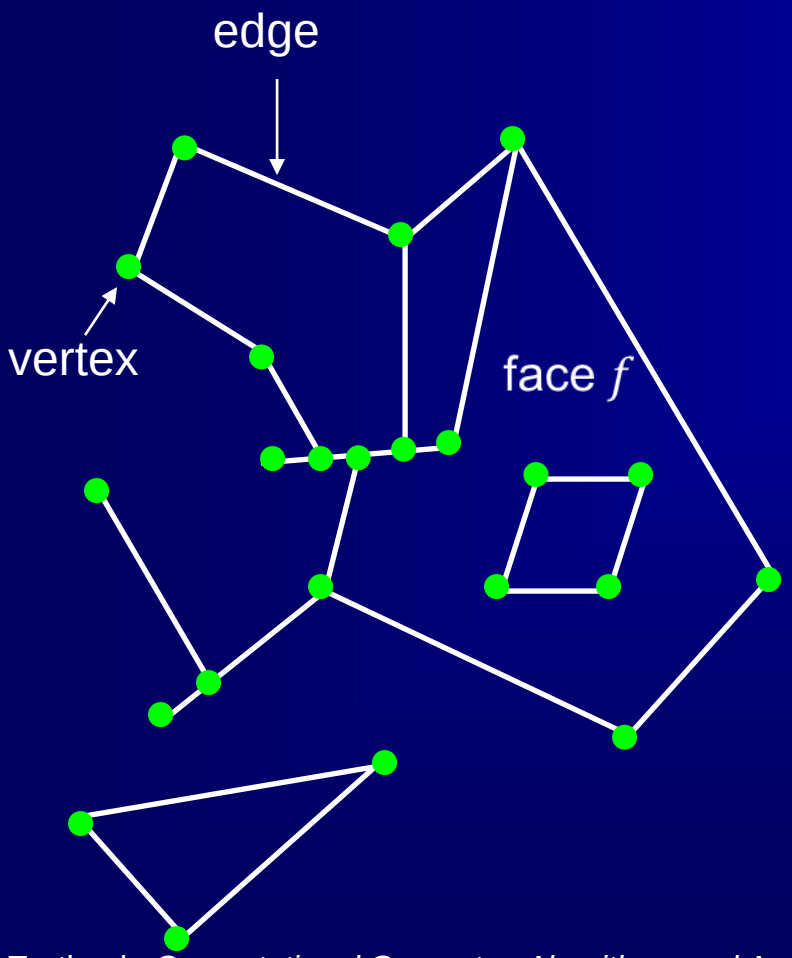
Planar Subdivision

Induced by planar embedding of a graph.



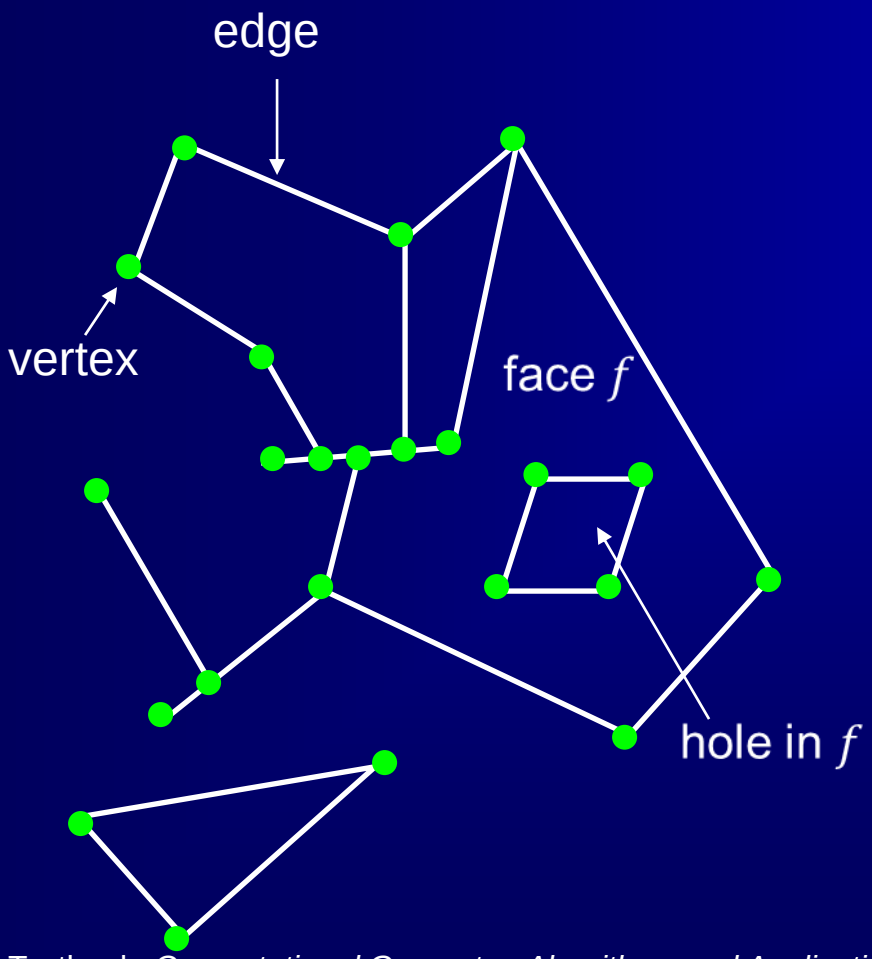
Planar Subdivision

Induced by planar embedding of a graph.



Planar Subdivision

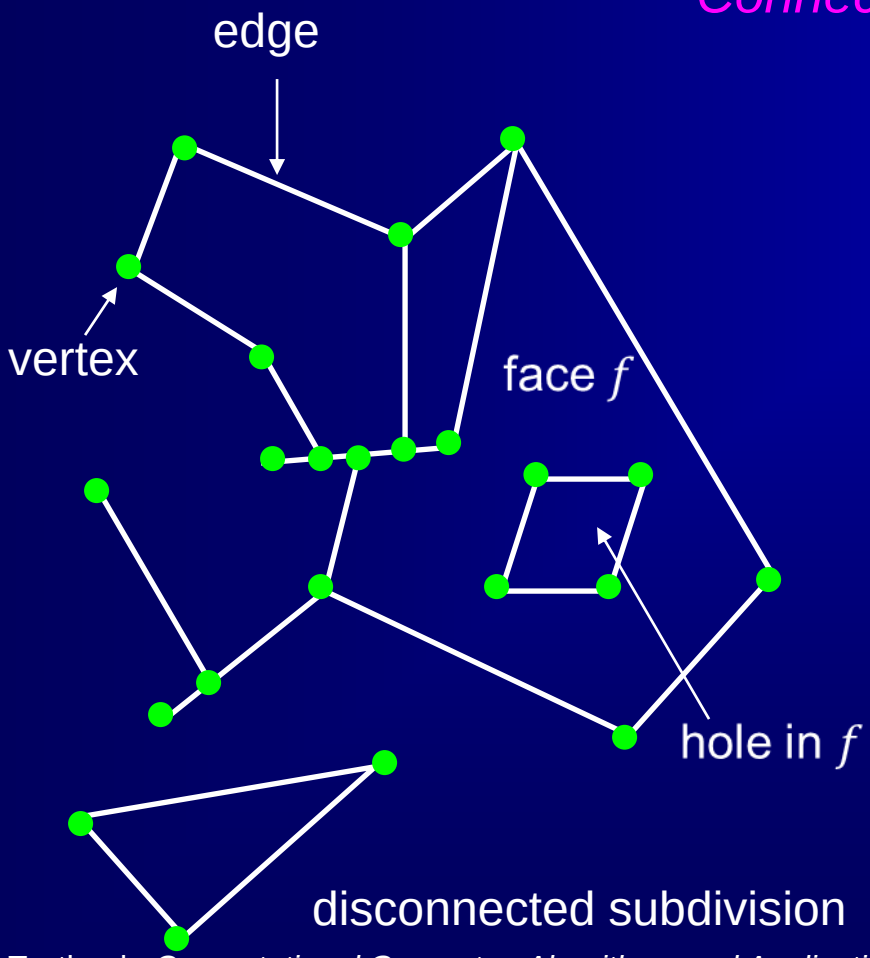
Induced by planar embedding of a graph.



Planar Subdivision

Induced by planar embedding of a graph.

Connected if the underlying graph is.

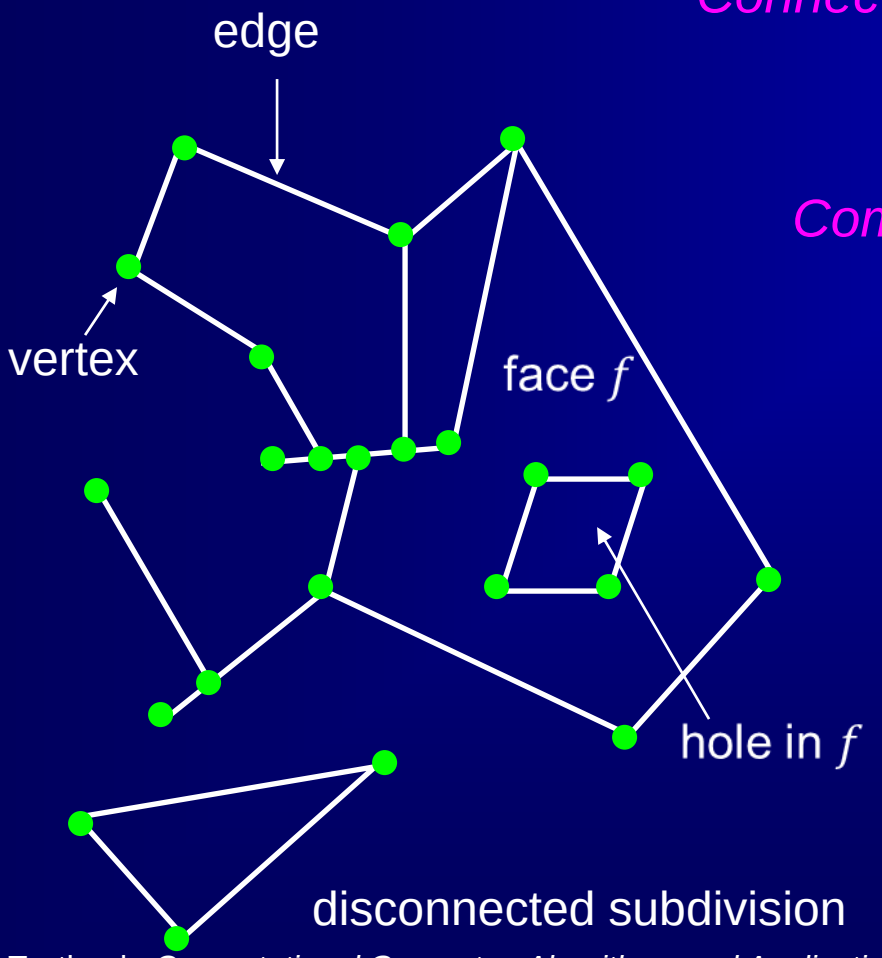


Planar Subdivision

Induced by planar embedding of a graph.

Connected if the underlying graph is.

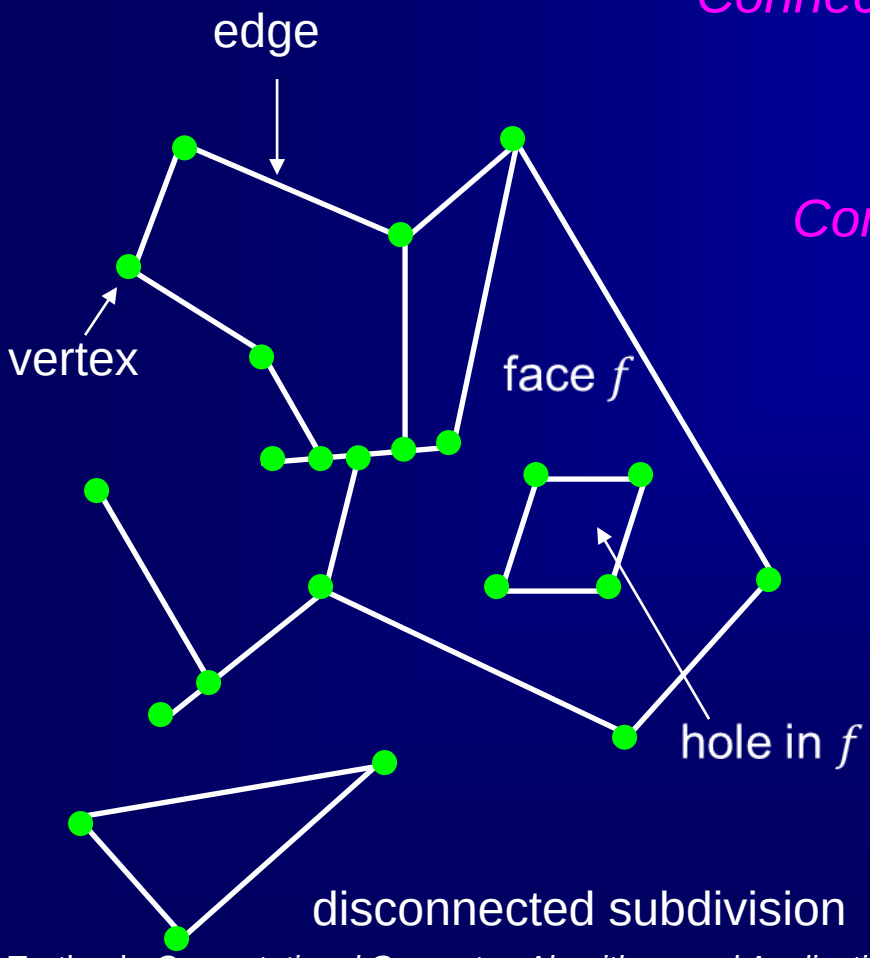
Complexity = #vertices + #edges + #faces



Planar Subdivision

Induced by planar embedding of a graph.

Connected if the underlying graph is.



Complexity = #vertices + #edges + #faces

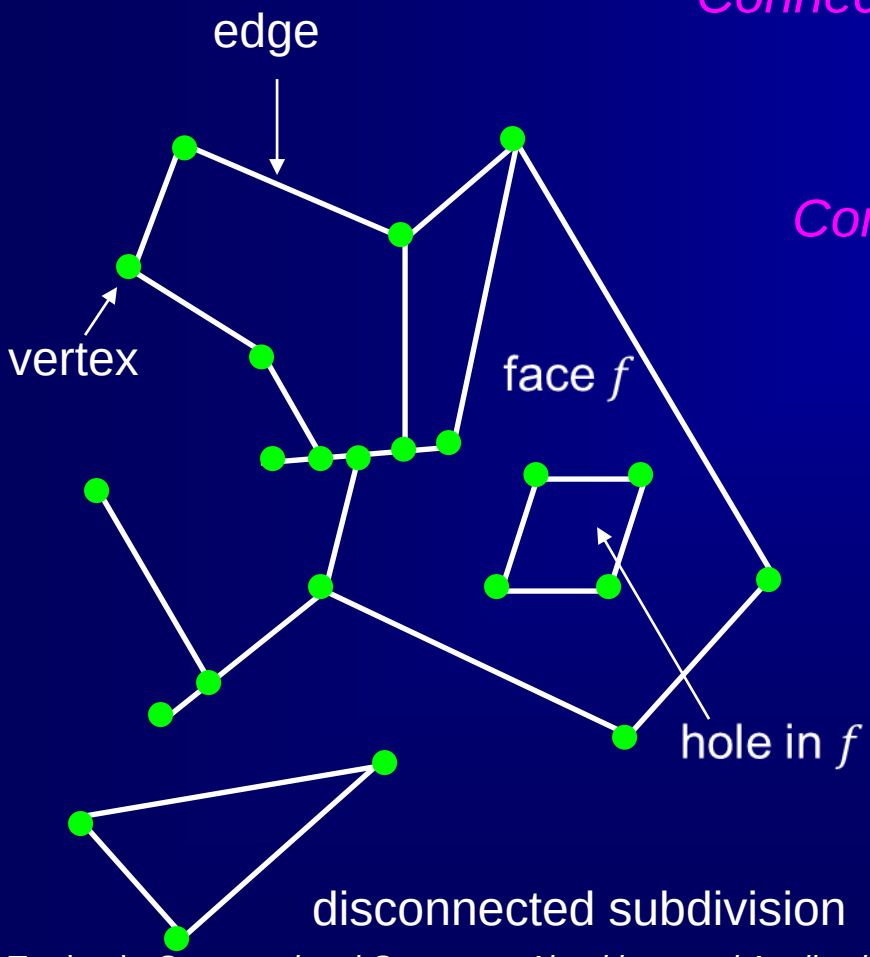
Typical operations:

- Walk around a face.

Planar Subdivision

Induced by planar embedding of a graph.

Connected if the underlying graph is.



Complexity = #vertices + #edges + #faces

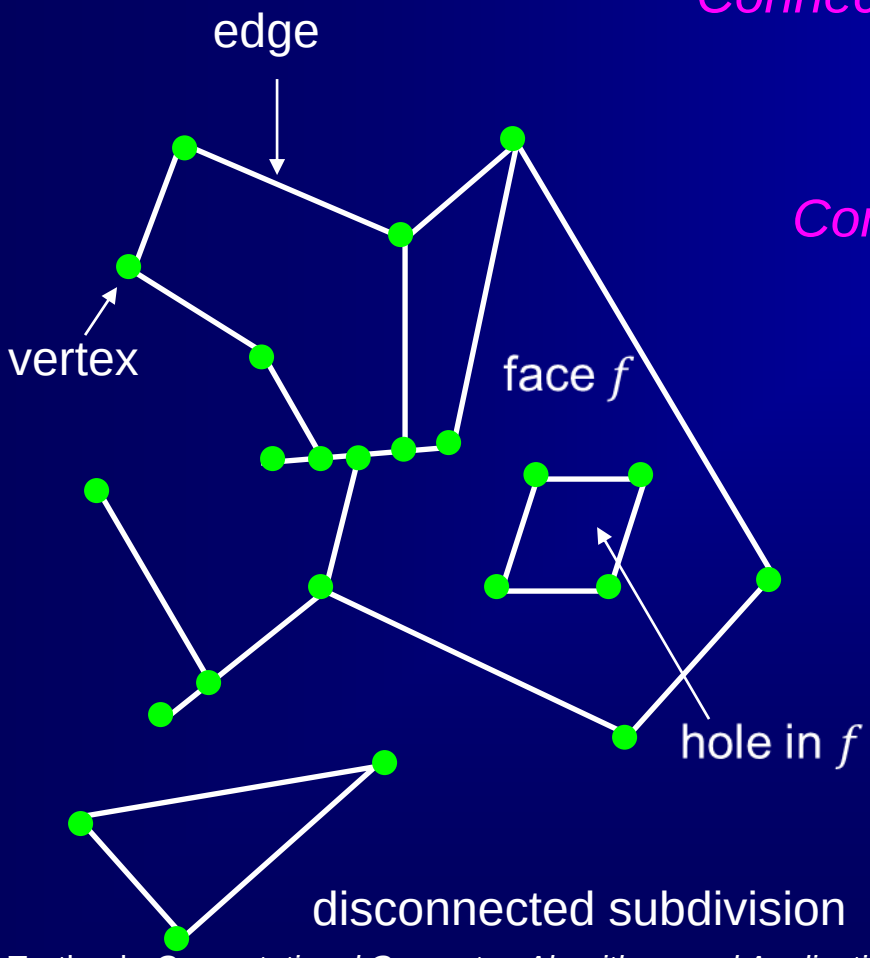
Typical operations:

- Walk around a face.
- Access one face from an adjacent one via a common edge.

Planar Subdivision

Induced by planar embedding of a graph.

Connected if the underlying graph is.



Complexity = #vertices + #edges + #faces

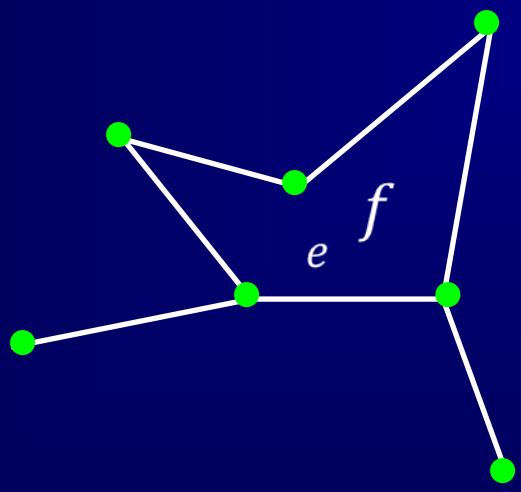
Typical operations:

- Walk around a face.
- Access one face from an adjacent one via a common edge.
- Visit all the edges adjacent to a vertex.

Doubly-Connected Edge List

Stores geometric and topological information.

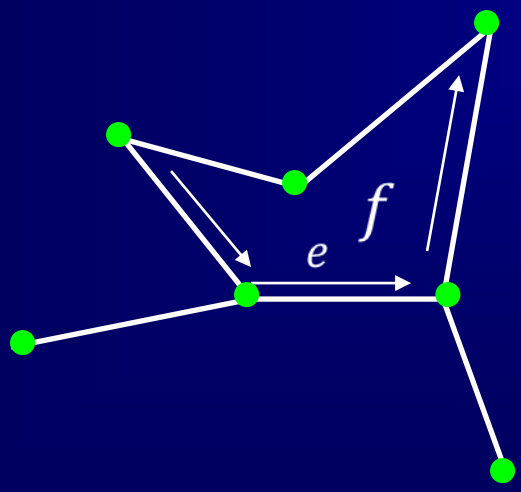
To walk around a face (counterclockwise), we set up



Doubly-Connected Edge List

Stores geometric and topological information.

To walk around a face (counterclockwise), we set up

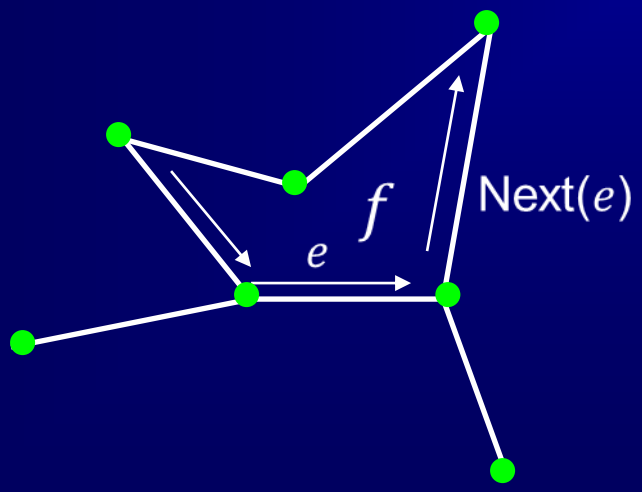


Doubly-Connected Edge List

Stores geometric and topological information.

To walk around a face (counterclockwise), we set up

- a pointer to the next edge

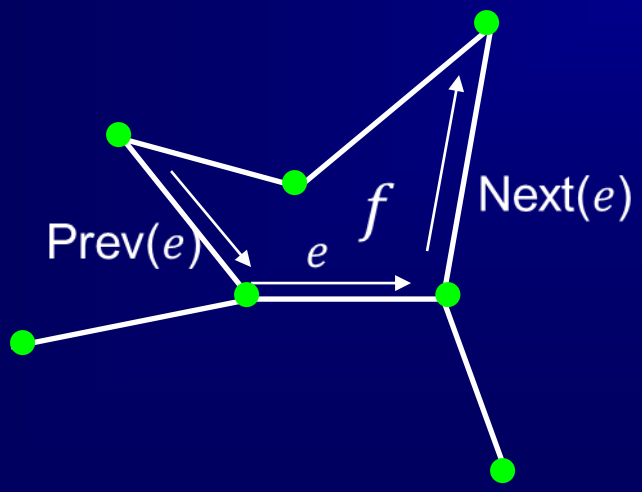


Doubly-Connected Edge List

Stores geometric and topological information.

To walk around a face (counterclockwise), we set up

- a pointer to the next edge
- a pointer to the previous edge



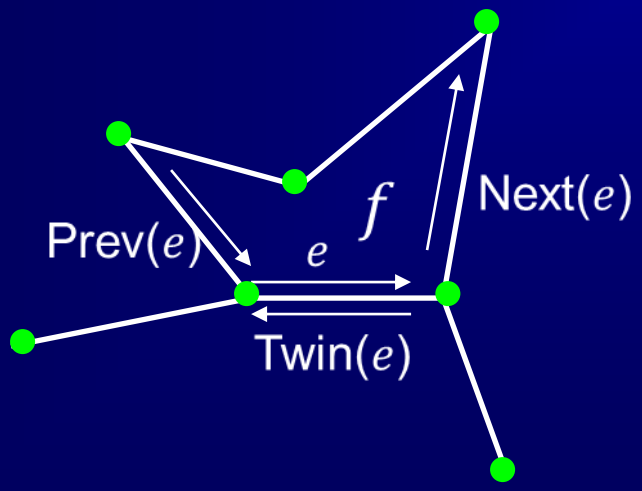
Doubly-Connected Edge List

Stores geometric and topological information.

To walk around a face (counterclockwise), we set up

- a pointer to the next edge
- a pointer to the previous edge

Every edge has two distinct half-edges (*twins*) in opposite directions.



Doubly-Connected Edge List

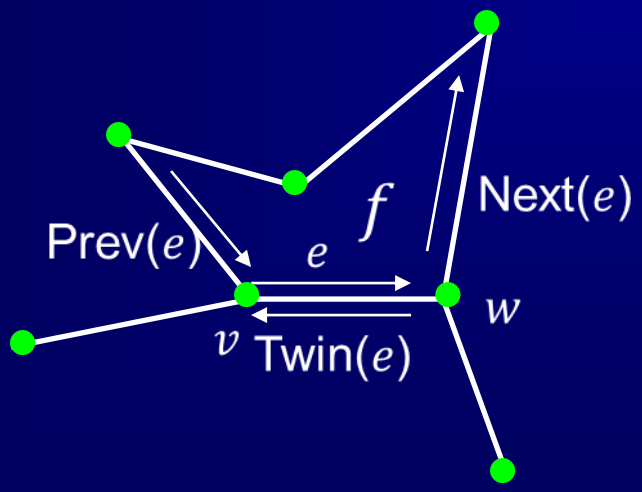
Stores geometric and topological information.

To walk around a face (counterclockwise), we set up

- a pointer to the next edge
- a pointer to the previous edge

Every edge has two distinct half-edges (*twins*) in opposite directions.

Edge e has *origin* v and *destination* w.



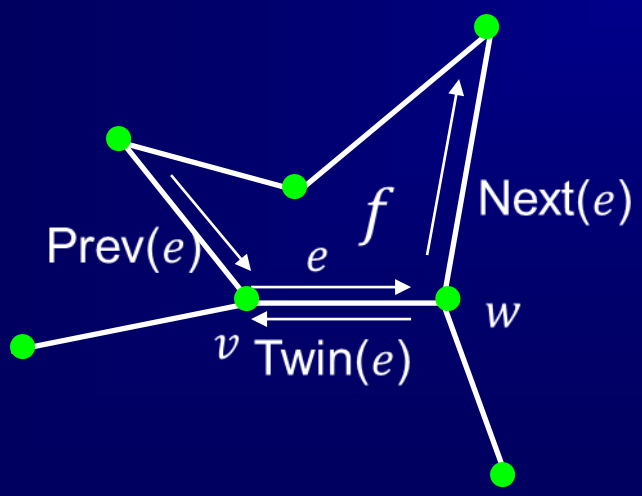
Doubly-Connected Edge List

Stores geometric and topological information.

To walk around a face (counterclockwise), we set up

- a pointer to the next edge
- a pointer to the previous edge

Every edge has two distinct half-edges (*twins*) in opposite directions.



Edge e has *origin* v and *destination* w.
Edge Twin(e) has origin w and destination v.

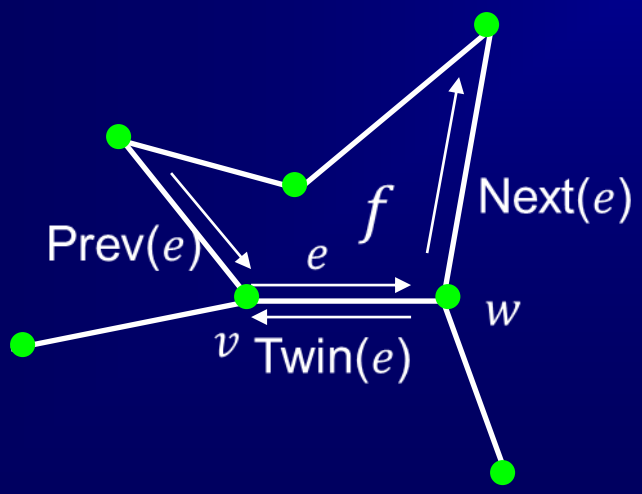
Doubly-Connected Edge List

Stores geometric and topological information.

To walk around a face (counterclockwise), we set up

- a pointer to the next edge
- a pointer to the previous edge

Every edge has two distinct half-edges (*twins*) in opposite directions.



Edge e has *origin* v and *destination* w.
Edge Twin(e) has origin w and destination v.

- For each face f store one pointer to an *arbitrary* half-edge bounding the face.

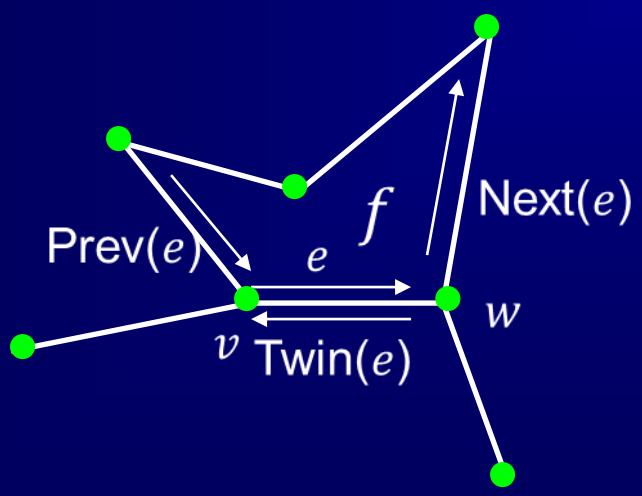
Doubly-Connected Edge List

Stores geometric and topological information.

To walk around a face (counterclockwise), we set up

- a pointer to the next edge
- a pointer to the previous edge

Every edge has two distinct half-edges (*twins*) in opposite directions.



Edge e has *origin* v and *destination* w.
Edge Twin(e) has origin w and destination v.

- For each face f store one pointer to an *arbitrary* half-edge bounding the face.

  From that half-edge traverse the face counterclockwise through the next pointer.

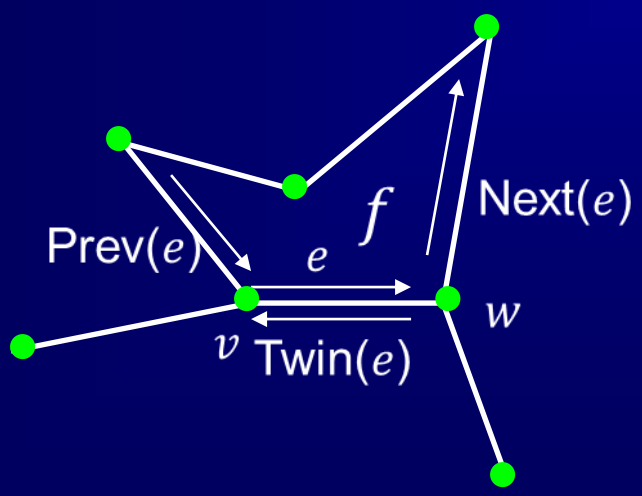
Doubly-Connected Edge List

Stores geometric and topological information.

To walk around a face (counterclockwise), we set up

- a pointer to the next edge
- a pointer to the previous edge

Every edge has two distinct half-edges (*twins*) in opposite directions.



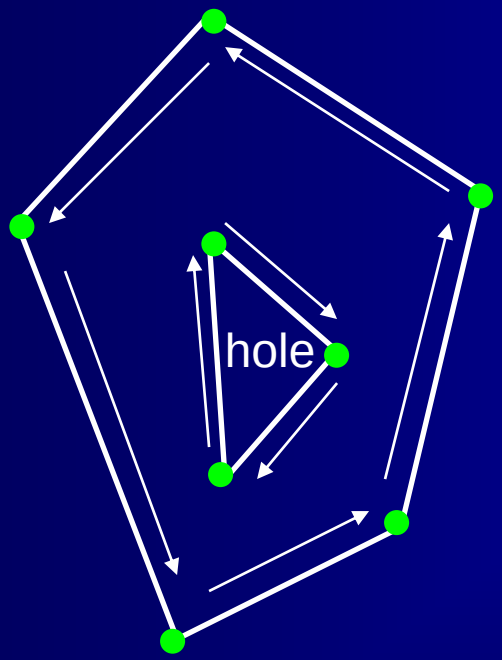
Edge e has *origin* v and *destination* w.
Edge Twin(e) has origin w and destination v.

- For each face f store one pointer to an *arbitrary* half-edge bounding the face.

  From that half-edge traverse the face counterclockwise through the next pointer.

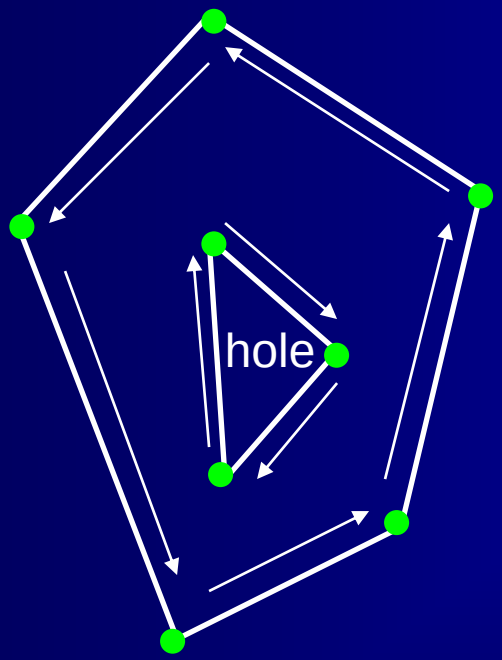
  This seems good if there's no hole …

Holes



Edges on the boundary of a hole are traversed in clockwise order so that the face still lies to the left.

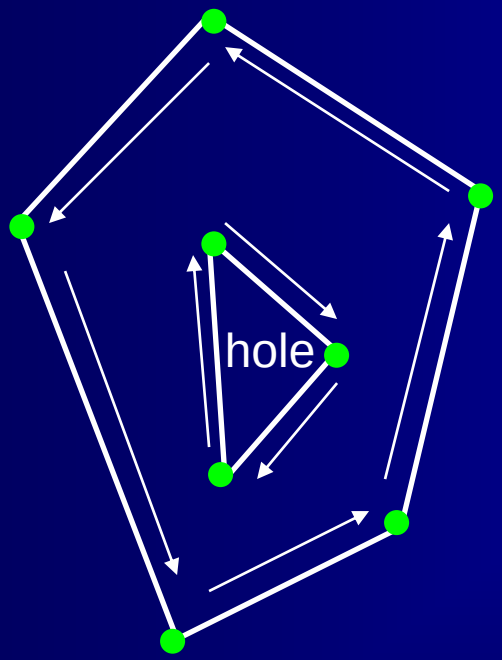
Holes



Edges on the boundary of a hole are traversed in clockwise order so that the face still lies to the left.

A face always lies to the left of any half-edge on its boundary.

Holes

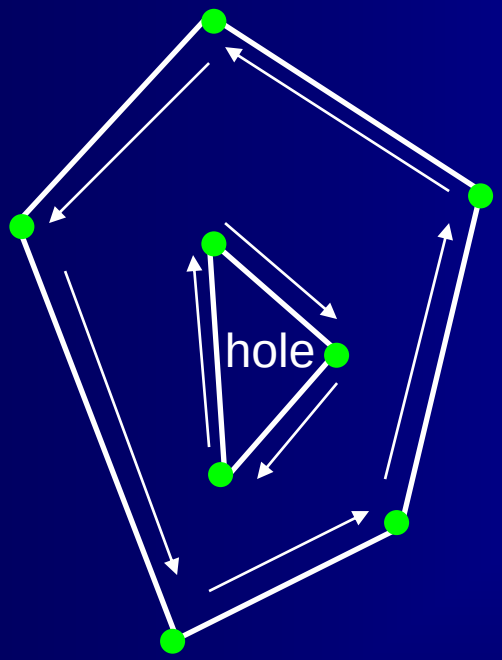


Edges on the boundary of a hole are traversed in clockwise order so that the face still lies to the left.

A face always lies to the left of any half-edge on its boundary.

To traverse the face, we need *a pointer to a half-edge in every boundary component.*

Holes



Edges on the boundary of a hole are traversed in clockwise order so that the face still lies to the left.

A face always lies to the left of any half-edge on its boundary.

To traverse the face, we need *a pointer to a half-edge in every boundary component.*

isolated vertex

Add a pointer to every isolated vertex in the face.

Summary on DCE List

- Every vertex v
  - Coordinates(v)
  - IncidentEdge(v) – pointer to an arbitrary half-edge originated from v.
- Every face f
  - OuterComponent(f) – pointer to some half-edge on outer boundary.
  - InnerComponents(f) – a list of pointers, each to some half-edge on the boundary of a different hole in the face.
- Every edge e
  - Origin(e)
  - Twin(e)
  - Next(e)
  - Prev(e)
  - IncidentFace(e) – face lying to the left of the half-edge

Summary on DCE List

- Every vertex v // $O(1)$ information
  - Coordinates(v)
  - IncidentEdge(v) – pointer to an arbitrary half-edge originated from v.
- Every face f
  - OuterComponent(f) – pointer to some half-edge on outer boundary.
  - InnerComponents(f) – a list of pointers, each to some half-edge on the boundary of a different hole in the face.
- Every edge e
  - Origin(e)
  - Twin(e)
  - Next(e)
  - Prev(e)
  - IncidentFace(e) – face lying to the left of the half-edge

Summary on DCE List

- Every vertex v // $O(1)$ information
  - Coordinates(v)
  - IncidentEdge(v) – pointer to an arbitrary half-edge originated from v.
- Every face f
  - OuterComponent(f) – pointer to some half-edge on outer boundary.
  - InnerComponents(f) – a list of pointers, each to some half-edge on the boundary of a different hole in the face.
- Every edge e // $O(1)$ information
  - Origin(e)
  - Twin(e)
  - Next(e)
  - Prev(e)
  - IncidentFace(e) – face lying to the left of the half-edge

Summary on DCE List

- Every vertex v // $O(1)$ information
  - Coordinates(v)
  - IncidentEdge(v) – pointer to an arbitrary half-edge originated from v.
    // information depends on # inner components.
- Every face f
  - OuterComponent(f) – pointer to some half-edge on outer boundary.
  - InnerComponents(f) – a list of pointers, each to some half-edge on the boundary of a different hole in the face.
- Every edge e // $O(1)$ information
  - Origin(e)
  - Twin(e)
  - Next(e)
  - Prev(e)
  - IncidentFace(e) – face lying to the left of the half-edge

Summary on DCE List

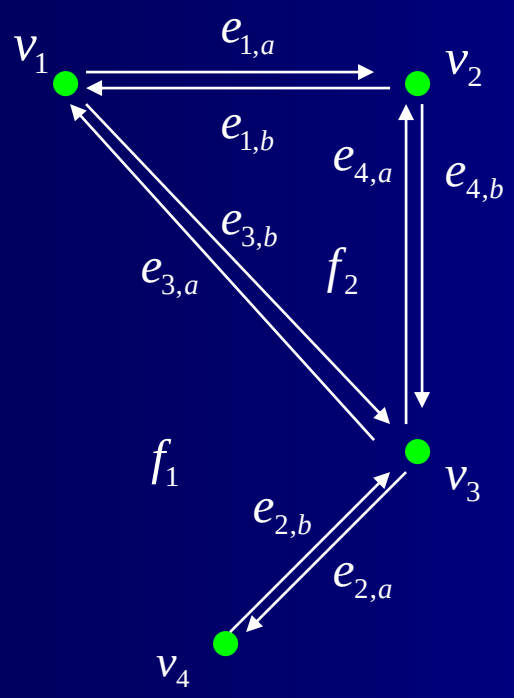
- Every vertex v // $O(1)$ information
  - Coordinates(v)
  - IncidentEdge(v) – pointer to an arbitrary half-edge originated from v.
- Every face f // information depends on # inner components.
  // any edge is pointed to at most once from the list
  // InnerComponents.
  - OuterComponent(f) – pointer to some half-edge on outer boundary.
  - InnerComponents(f) – a list of pointers, each to some half-edge on the boundary of a different hole in the face.
- Every edge e // $O(1)$ information
  - Origin(e)
  - Twin(e)
  - Next(e)
  - Prev(e)
  - IncidentFace(e) – face lying to the left of the half-edge

Summary on DCE List

- Every vertex v // $O(1)$ information
  - Coordinates(v)
  - IncidentEdge(v) – pointer to an arbitrary half-edge originated from v.
- Every face f // information depends on # inner components.
  // any edge is pointed to at most once from the list
  // InnerComponents.
  - OuterComponent(f) – pointer to some half-edge on outer boundary.
  - InnerComponents(f) – a list of pointers, each to some half-edge on the boundary of a different hole in the face.
- Every edge e // $O(1)$ information
  - Origin(e)
  - Twin(e)
  - Next(e)
  - Prev(e)
  - IncidentFace(e) – face lying to the left of the half-edge

total storage: O(#vertices + #edges + #faces)
i.e. linear in subdivision complexity.

An Example

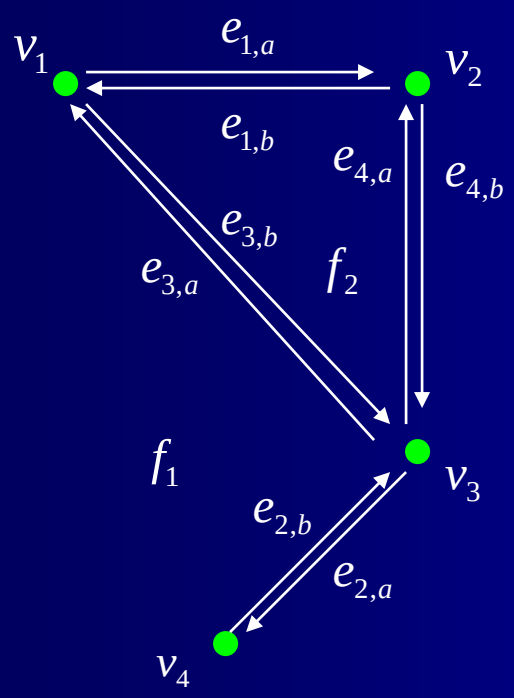


face	Outercomponent	InnerComponents
f_1	nil	$e_{1,a}$
f_2	$e_{4,a}$	nil

vertex	Coordinates	IncidentEdge
v_1	(0,4)	$e_{1,a}$
v_2	(2,4)	$e_{4,b}$
v_3	(2,2)	$e_{2,a}$
v_4	(1,1)	$e_{2,b}$

half-edge	origin	Twin	IncidentFace	Next	Prev
$e_{1,a}$	v_1	$e_{1,b}$	f_1	$e_{4,b}$	$e_{3,a}$
$e_{1,b}$	v_2	$e_{1,a}$	f_2	$e_{3,b}$	$e_{4,a}$
$e_{2,a}$	v_3	$e_{2,b}$	f_1	$e_{2,b}$	$e_{4,b}$
$e_{2,b}$	v_4	$e_{2,a}$	f_1	$e_{3,a}$	$e_{2,a}$
$e_{3,a}$	v_3	$e_{3,b}$	f_1	$e_{1,a}$	$e_{2,b}$
$e_{3,b}$	v_1	$e_{3,a}$	f_2	$e_{4,a}$	$e_{1,b}$
$e_{4,a}$	v_3	$e_{4,b}$	f_2	$e_{1,b}$	$e_{3,b}$
$e_{4,b}$	v_2	$e_{4,a}$	f_1	$e_{2,a}$	$e_{1,a}$

An Example



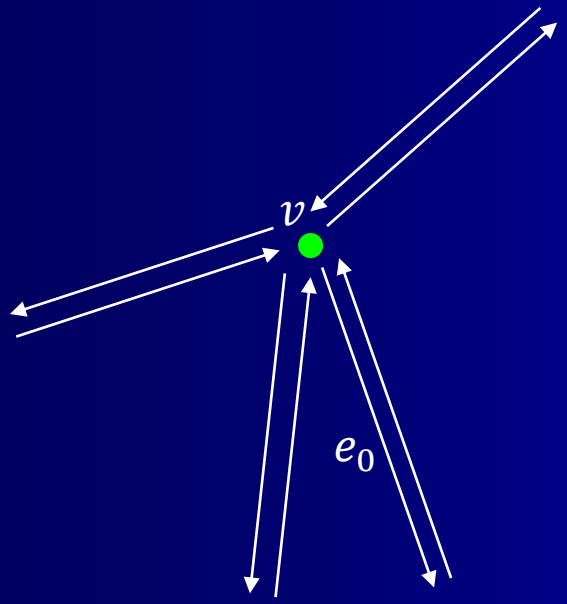
face	Outercomponent	InnerComponents
f_1	nil	$e_{1,a}$
f_2	$e_{4,a}$	nil

vertex	Coordinates	IncidentEdge
v_1	(0,4)	$e_{1,a}$
v_2	(2,4)	$e_{4,b}$
v_3	(2,2)	$e_{2,a}$
v_4	(1,1)	$e_{2,b}$

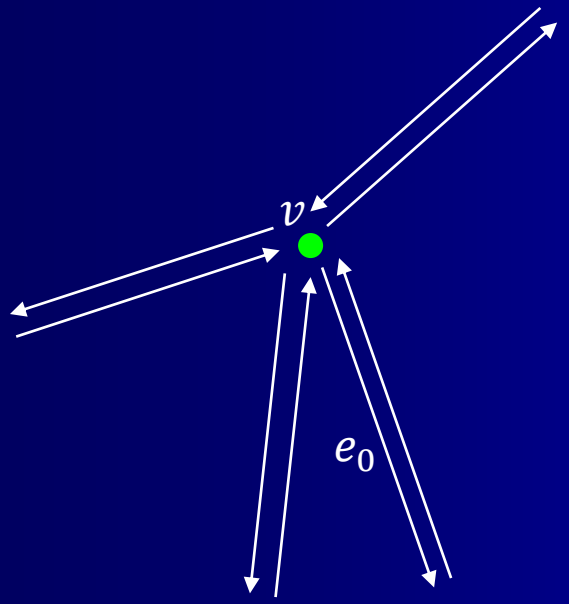
half-edge	origin	Twin	IncidentFace	Next	Prev
$e_{1,a}$	v_1	$e_{1,b}$	f_1	$e_{4,b}$	$e_{3,a}$
$e_{1,b}$	v_2	$e_{1,a}$	f_2	$e_{3,b}$	$e_{4,a}$
$e_{2,a}$	v_3	$e_{2,b}$	f_1	$e_{2,b}$	$e_{4,b}$
$e_{2,b}$	v_4	$e_{2,a}$	f_1	$e_{3,a}$	$e_{2,a}$
$e_{3,a}$	v_3	$e_{3,b}$	f_1	$e_{1,a}$	$e_{2,b}$
$e_{3,b}$	v_1	$e_{3,a}$	f_2	$e_{4,a}$	$e_{1,b}$
$e_{4,a}$	v_3	$e_{4,b}$	f_2	$e_{1,b}$	$e_{3,b}$
$e_{4,b}$	v_2	$e_{4,a}$	f_1	$e_{2,a}$	$e_{1,a}$

How to find all incident edges to a vertex?

Edges Incident on a Vertex

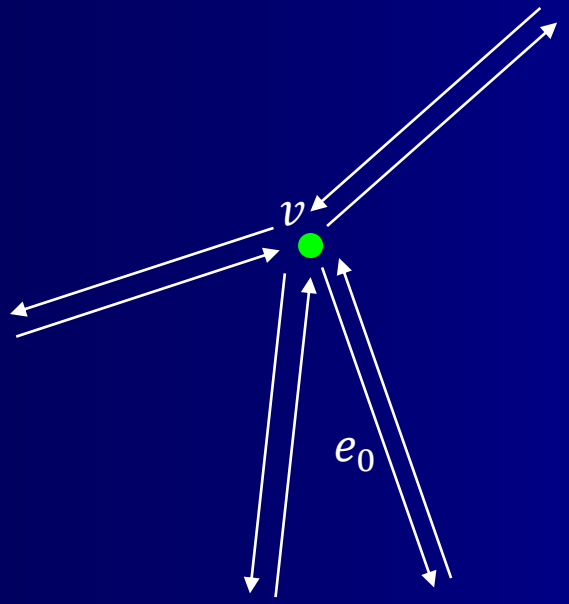


Edges Incident on a Vertex



Start with a half-edge e_0 that has the origin v.

Edges Incident on a Vertex



Start with a half-edge e_0 that has the origin v.

$e \leftarrow e_0$

do

output e

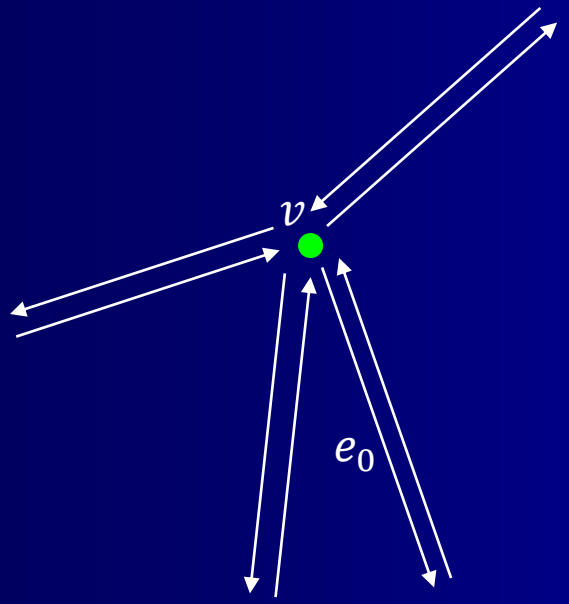
$e \leftarrow Twin(e)$

output e

$e \leftarrow Next(e)$

until $e = e_0$

Edges Incident on a Vertex



Start with a half-edge e_0 that has the origin v.

$e \leftarrow e_0$

do

output e

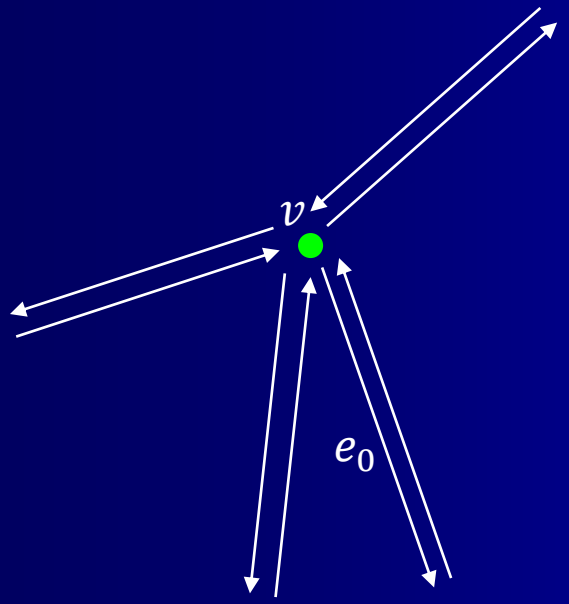
$e \leftarrow Twin(e)$

output e

$e \leftarrow Next(e)$

until $e = e_0$

Edges Incident on a Vertex



Start with a half-edge e_0 that has the origin v.

$e \leftarrow e_0$

do

output e

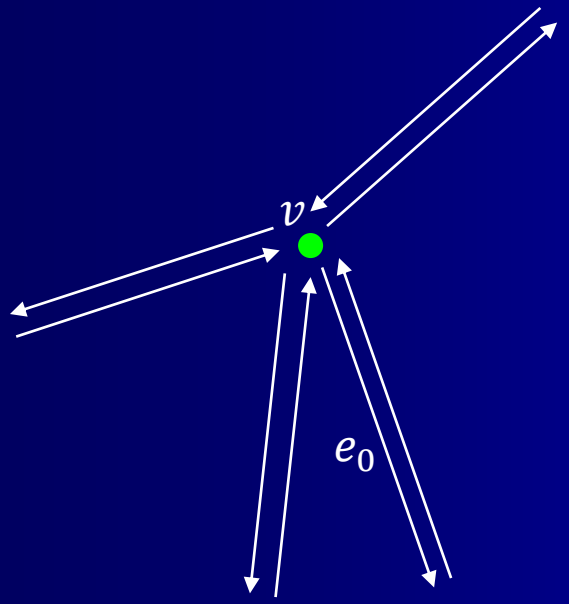
$e \leftarrow Twin(e)$

output e

$e \leftarrow Next(e)$

until $e = e_0$

Edges Incident on a Vertex



Start with a half-edge e_0 that has the origin v.

$e \leftarrow e_0$

do

output e

$e \leftarrow Twin(e)$

output e

$e \leftarrow Next(e)$

until $e = e_0$