CS60064 Spring 2022

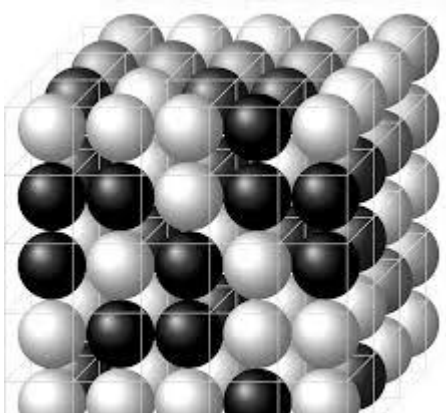
Computational Geometry

Instructors

Bhargab B. Bhattacharya (BBB)
Partha Bhowmick (PB)

Lecture 12
02 February 2022

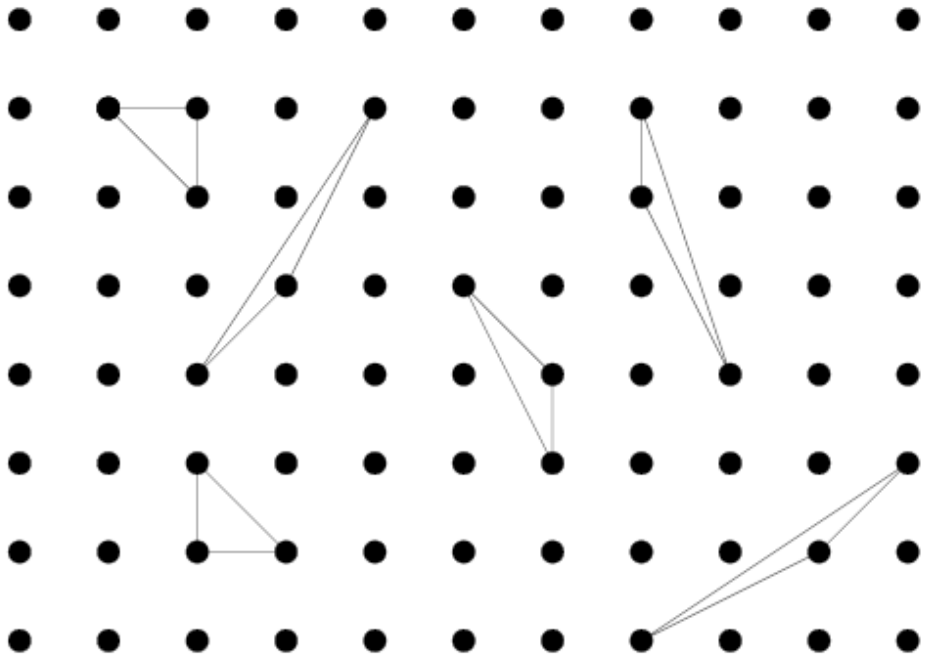
Indian Institute of Technology Kharagpur
Computer Science and Engineering



Lattice Polygons

Doubly Connected Edge List (DCEL)

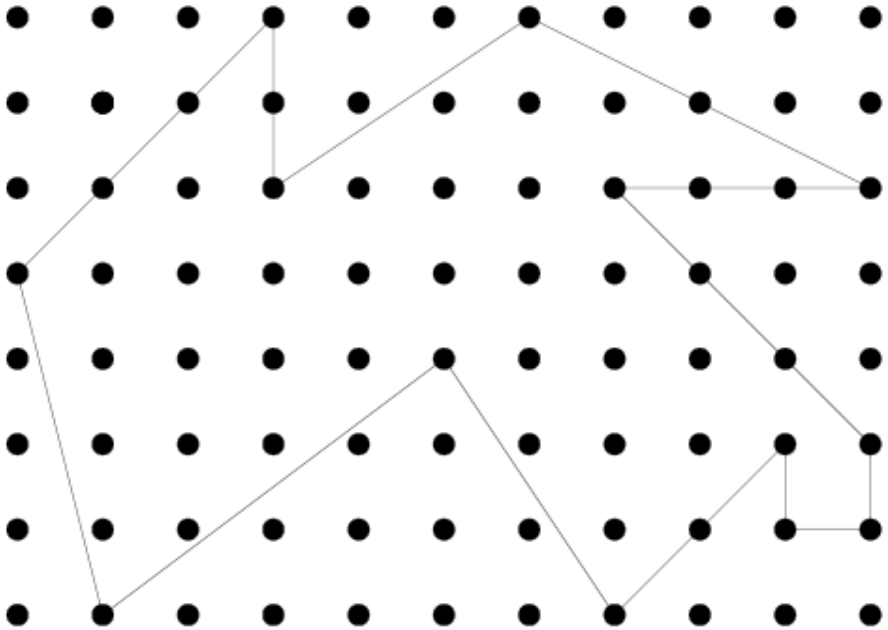
Problem of the Day: Lattice Polygons



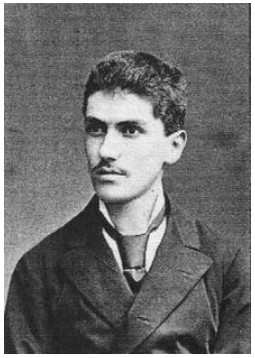
Unit-distance square lattice (dots are at integer coordinates)
What is the area of an elementary triangle (a triangle whose vertices coincide with the lattice points and does not have points in the interior)?

Lemma: Area(Δ) = ½, for any elementary triangle

Lattice Polygons



its area is
30 + (22/2) − 1 = 40 sq. units



Georg Alexander Pick
(1859 – 1952)

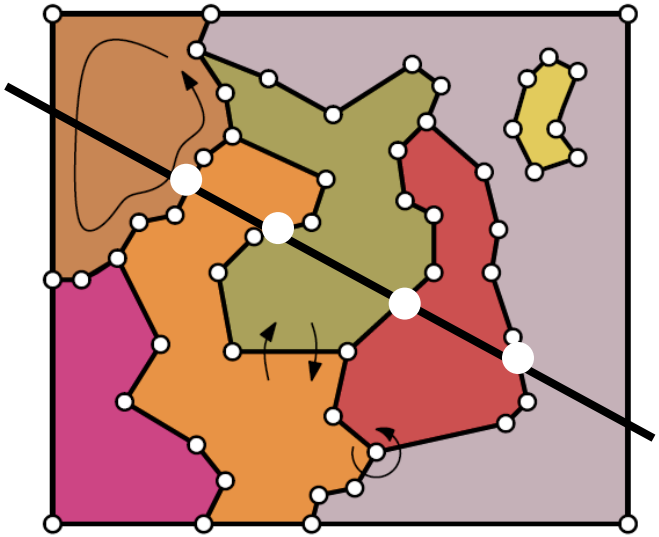
What is the area of a lattice polygon ?

Pick's Theorem (1899):

Area(P) = #interior points + (#boundary points/2) -1

Ref: M. Aigner and G. Ziegler, Proofs from THE BOOK, 6th Edition, Springer, 2018

Doubly Connected Edge List (DCEL)



- Map corresponds to subdivision of plane into polygons, PSLGs, triangulations
- Embedding of planar graph with vertices, edges, faces
- Need a suitable data structure such that most of the operations or updates can be performed efficiently

Operations to be supported by the data structure?

- Given one bounding edge, explore a face by traversing all edges around it
- Walk from face to face crossing edges
- Traverse neighboring vertices in cyclic order
- Update vertex, edge, face information when new edges cross the map, and so on ….

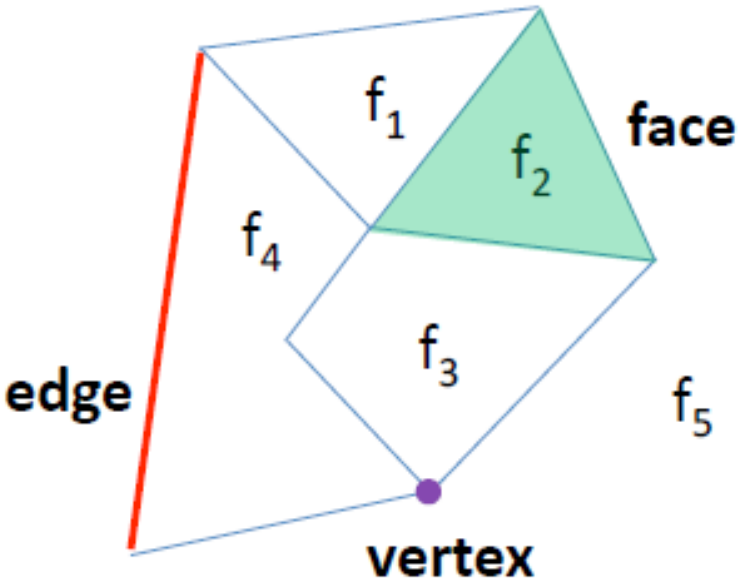
Doubly Connected Edge List (DCEL)

- DCEL is one of the most commonly used representations for planar subdivisions

- It is an edge-based structure which links together the three sets of records:
  - **Vertex**
  - **Edge**
  - **Face**

- It facilitates traversing the faces of planar subdivision, visiting all the edges around a given vertex/face

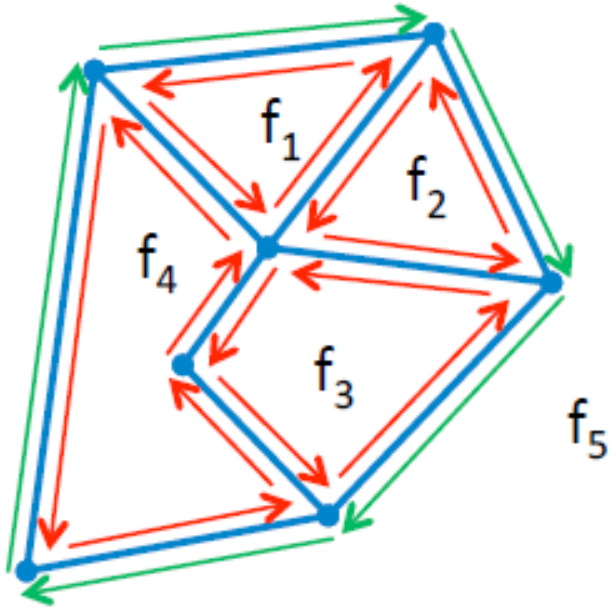
Doubly Connected Edge List (DCEL)



- Record for each face, edge, and vertex
  – Geometric information
  – Topological information
  – Attribute information
- Half-edge (twin-edge) structure

Ref: 4A Textbook

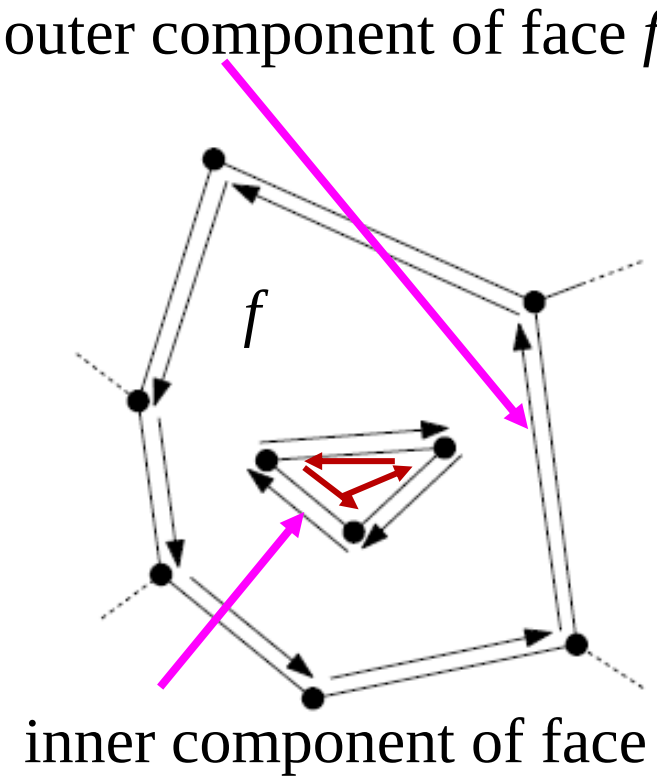
Doubly Connected Edge List (DCEL)



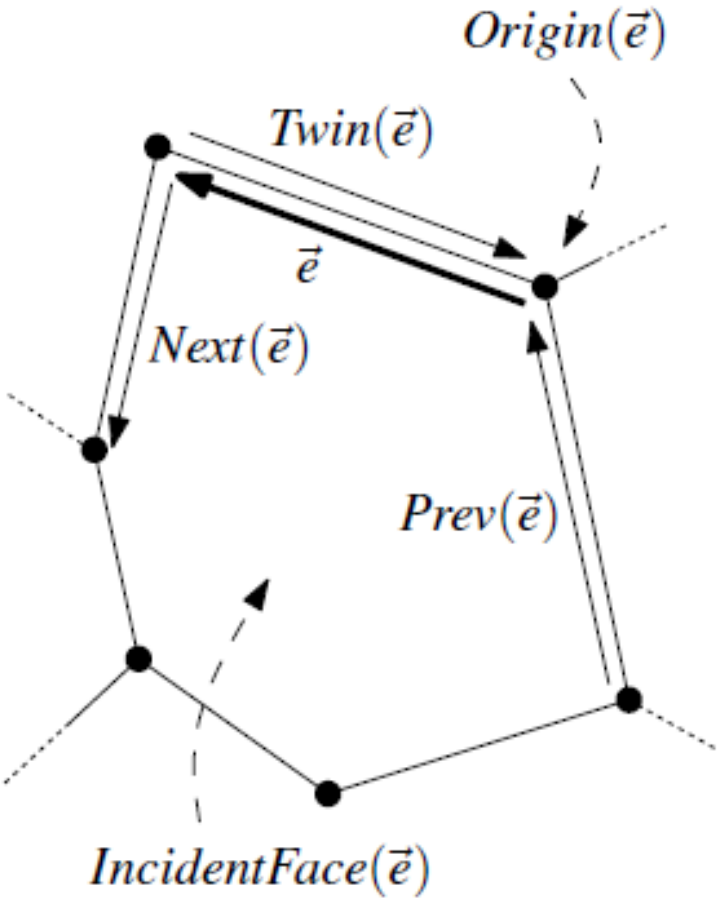
- **Main idea:**
  - Edges are oriented counterclockwise inside each face
  - Since an edge borders two faces, each edge is represented by two half-edges (twin edges), one for each face

Ref: 4A Textbook

Doubly Connected Edge List (DCEL)

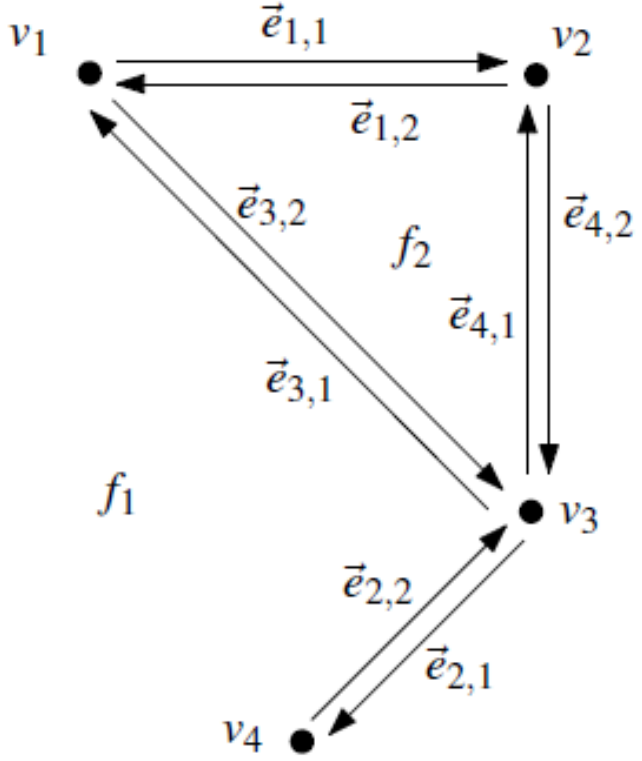


polygonal map with a hole



Ref: 4A Textbook

Doubly Connected Edge List (DCEL)



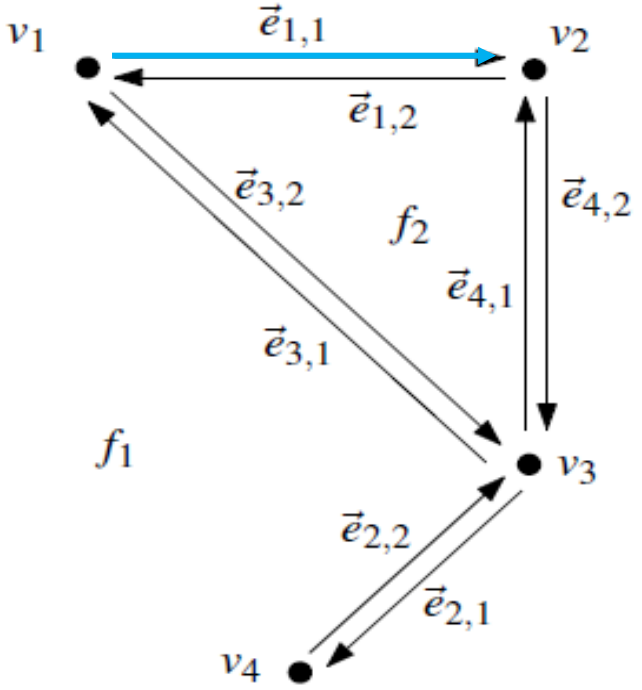
Vertex	Coordinates	IncidentEdge
v_1	$(0,4)$	$\vec{e}_{1,1}$
v_2	$(2,4)$	$\vec{e}_{4,2}$
v_3	$(2,2)$	$\vec{e}_{2,1}$
v_4	$(1,1)$	$\vec{e}_{2,2}$

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

Ref: 4A Textbook

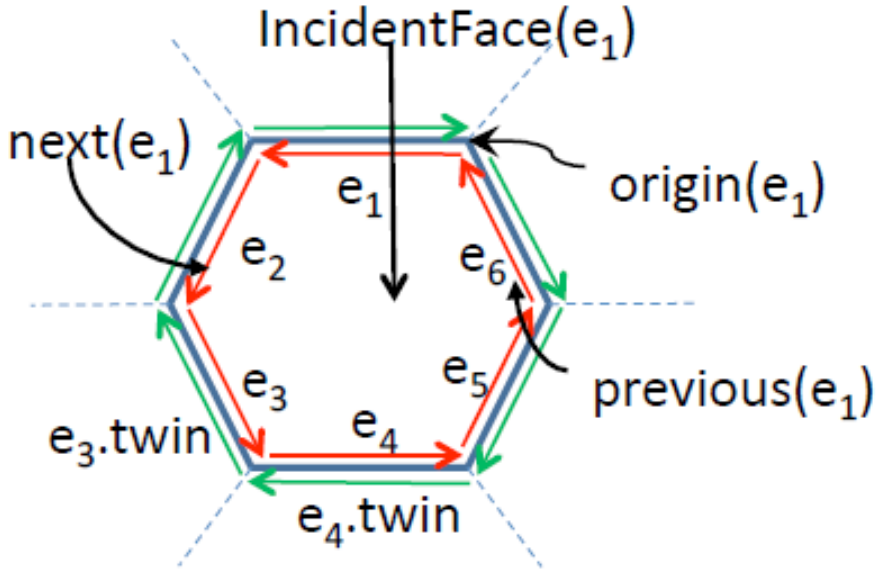
DCEL

Ref: 4A Textbook



Half-edge	Origin	Twin	IncidentFace	Next	Prev
$\vec{e}_{1,1}$	v_1	$\vec{e}_{1,2}$	f_1	$\vec{e}_{4,2}$	$\vec{e}_{3,1}$
$\vec{e}_{1,2}$	v_2	$\vec{e}_{1,1}$	f_2	$\vec{e}_{3,2}$	$\vec{e}_{4,1}$
$\vec{e}_{2,1}$	v_3	$\vec{e}_{2,2}$	f_1	$\vec{e}_{2,2}$	$\vec{e}_{4,2}$
$\vec{e}_{2,2}$	v_4	$\vec{e}_{2,1}$	f_1	$\vec{e}_{3,1}$	$\vec{e}_{2,1}$
$\vec{e}_{3,1}$	v_3	$\vec{e}_{3,2}$	f_1	$\vec{e}_{1,1}$	$\vec{e}_{2,2}$
$\vec{e}_{3,2}$	v_1	$\vec{e}_{3,1}$	f_2	$\vec{e}_{4,1}$	$\vec{e}_{1,2}$
$\vec{e}_{4,1}$	v_3	$\vec{e}_{4,2}$	f_2	$\vec{e}_{1,2}$	$\vec{e}_{3,2}$
$\vec{e}_{4,2}$	v_2	$\vec{e}_{4,1}$	f_1	$\vec{e}_{2,1}$	$\vec{e}_{1,1}$

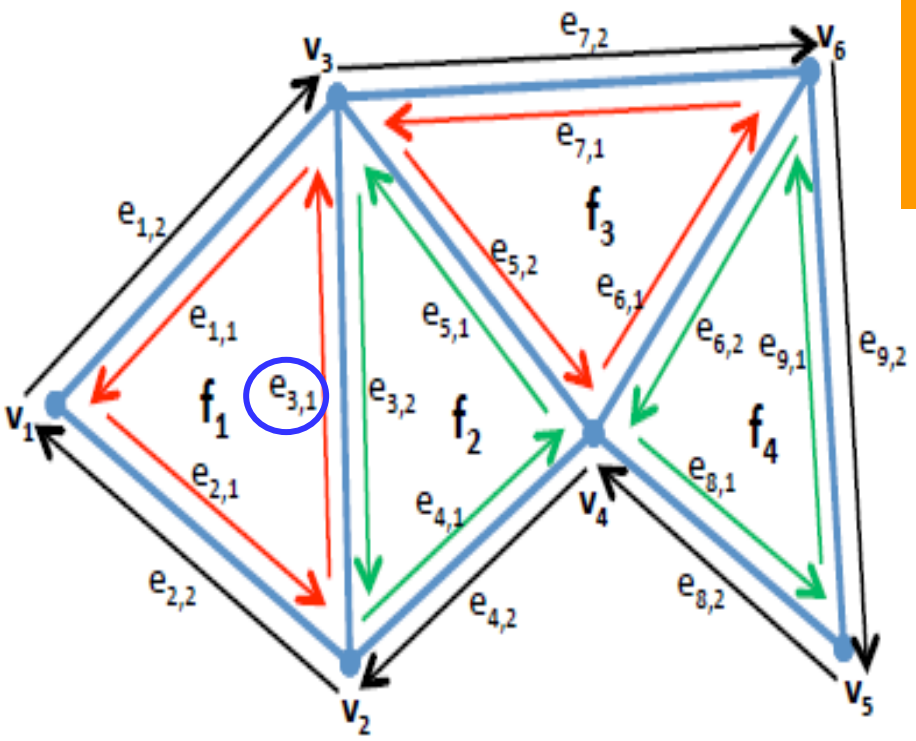
Half-Edge Records in DCEL



The half-edge record of a half-edge *e* stores pointers to:

- Origin (*e*), i.e., the *vertex* from which directed *e* has originated
- Twin of *e*, *e*.twin or twin(*e*), the same edge in reverse direction corresponding to its neighboring face
- The face to its left (IncidentFace(*e*))
- Next(*e*) : next half-edge on the boundary of IncidentFace(*e*)
- Previous(*e*) : previous half-edge

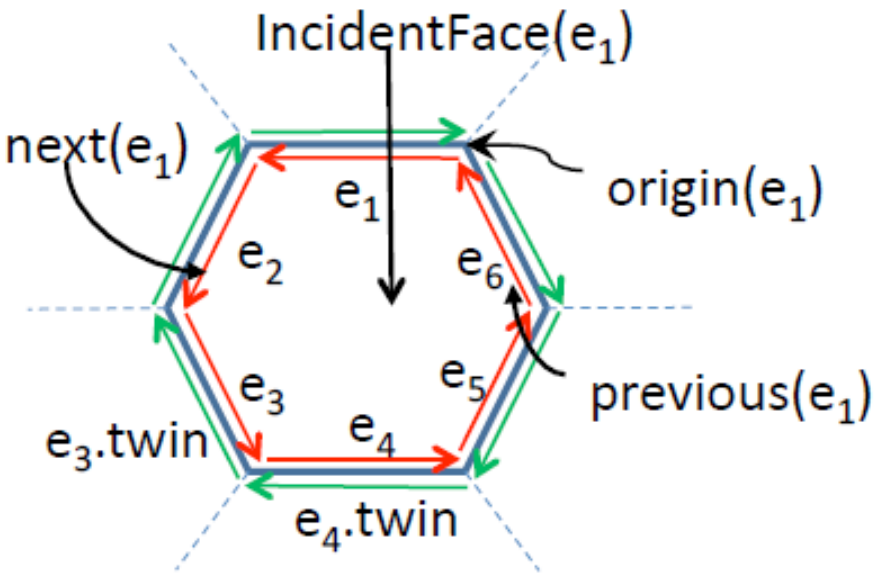
Example: Half-Edge Records in DCEL



- Origin(*e*); Twin of *e*; The face to its left (IncidentFace(*e*));
- Next(*e*): next half-edge on the IncidentFace(*e*);
- Previous(*e*): previous half-edge

Half-edge	Origin	Twin	IncidentFace	Next	Previous
$e_{3,1}$	v_2	$e_{3,2}$	f_1	$e_{1,1}$	$e_{2,1}$
$e_{3,2}$	v_3	$e_{3,1}$	f_2	$e_{4,1}$	$e_{5,1}$
$e_{4,1}$	v_2	$e_{4,2}$	f_2	$e_{5,1}$	$e_{3,2}$
$e_{4,2}$	v_4	$e_{4,1}$	f_5	$e_{2,2}$	$e_{8,2}$
...	...	...	...	...	...

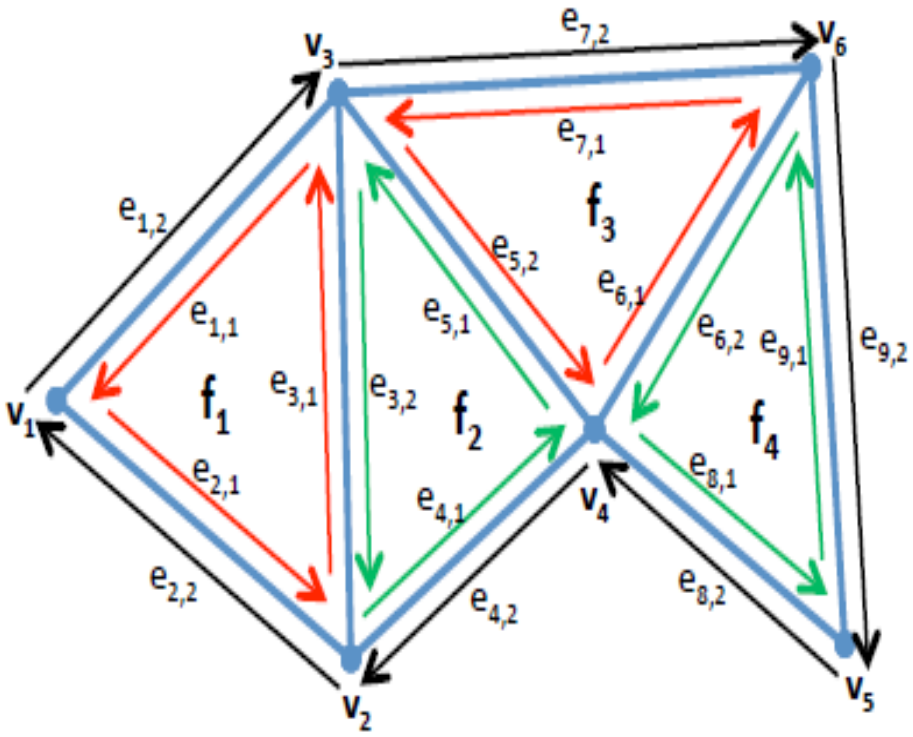
Vertex and Face Records in DCEL



The vertex record of a vertex *v* stores the coordinates of *v*. It also stores a pointer **IncidentEdge**(*v*) to any half-edge that has *v* as its origin (i.e., an outgoing edge emanating from *v*)

• The face record of a face *f* stores a pointer to *any* half-edge on its boundary which can be used as a starting point to traverse *f* in counter-clockwise order

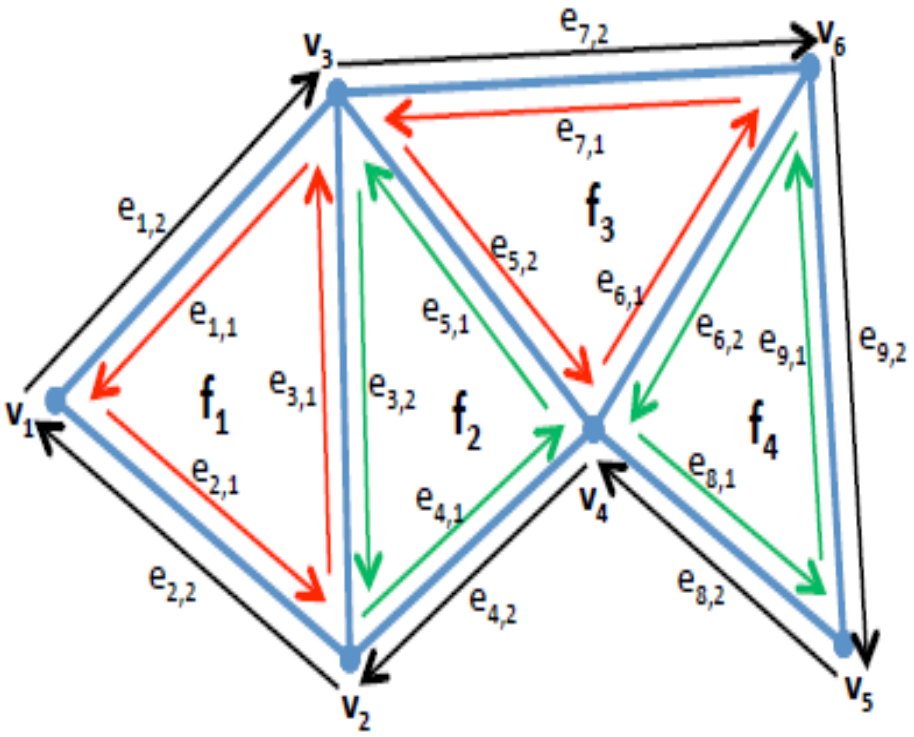
Example: Vertex Records in DCEL



Vertex	Coordinates	IncidentEdge
v_1	(x_1, y_1)	$e_{2,1}$
v_2	(x_2, y_2)	$e_{4,1}$
v_3	(x_3, y_3)	$e_{3,2}$
v_4	(x_4, y_4)	$e_{6,1}$
v_5	(x_5, y_5)	$e_{9,1}$
v_6	(x_6, y_6)	$e_{7,1}$

The vertex record of a vertex *v* stores the coordinates of *v*. It also stores a pointer **IncidentEdge**(*v*) to any half-edge that has *v* as its origin (i.e., any outgoing edge emanating from *v*)

Example: Face Records in DCEL

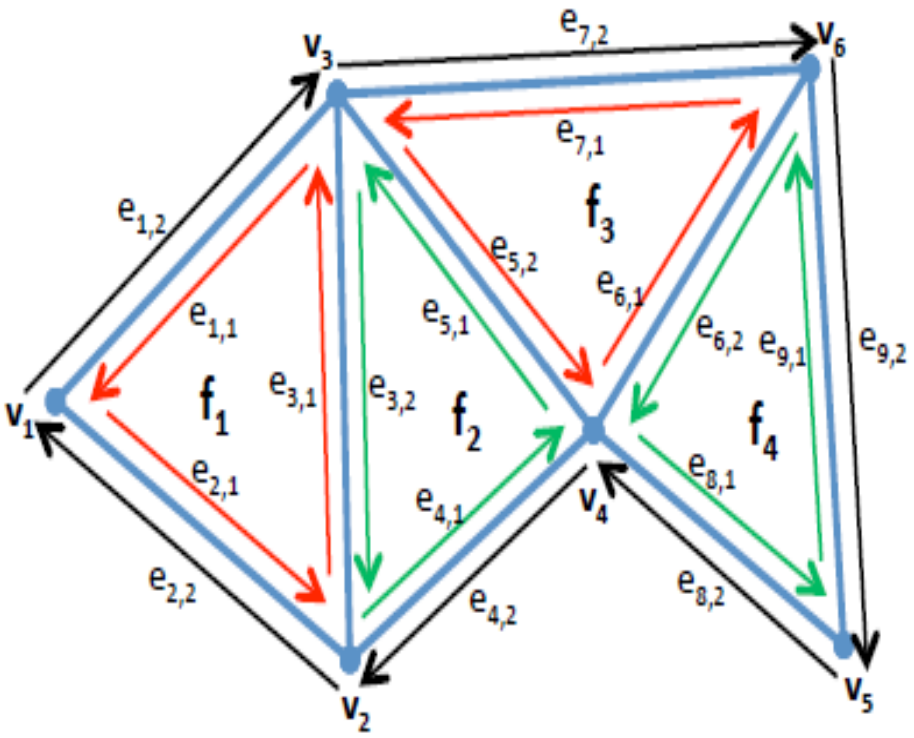


f_5

Face	Edge
f_1	$e_{1,1}$
f_2	$e_{5,1}$
f_3	$e_{5,2}$
f_4	$e_{8,1}$
f_5	$e_{9,2}$

• The face record of a face *f* stores a pointer to *any* half-edge on its boundary which can be used as a starting point to traverse *f* in counter-clockwise order

Storage Requirement and Operations in DCEL

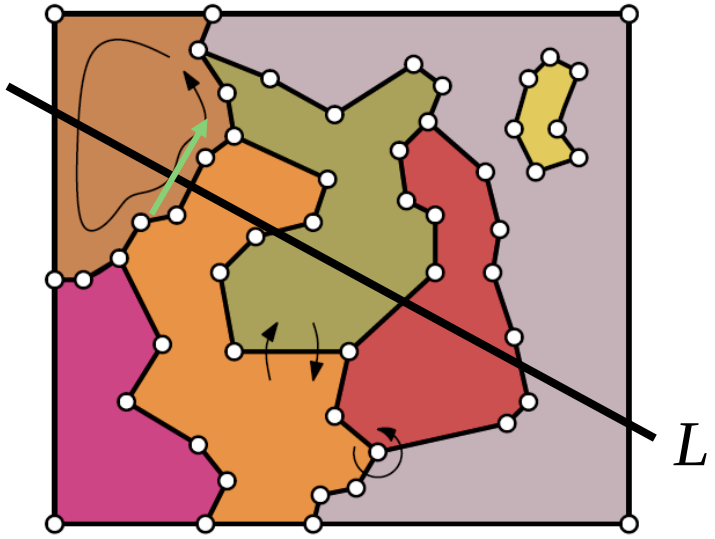


- Storage space requirement:
  - Linear in the number of vertices (or on edges or faces)

Operations:

- Walk around the boundary of a given face in CCW order
- Access a face from an adjacent one
- Visit all the edges around a given vertex

Queries in DCEL



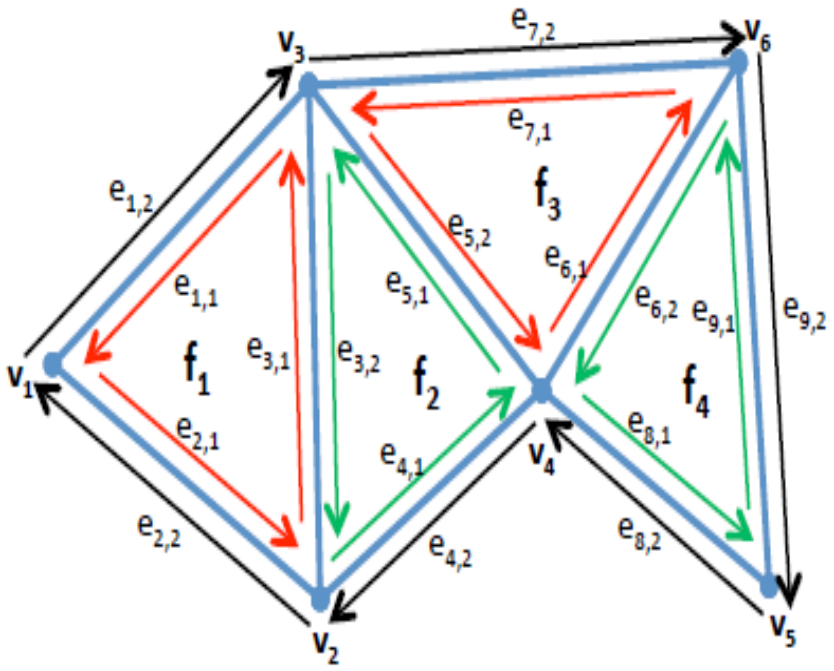
Half-edge	Origin	Twin	IncidentFace	Next	Previous
$e_{3,1}$	v_2	$e_{3,2}$	f_1	$e_{1,1}$	$e_{2,1}$
$e_{3,2}$	v_3	$e_{3,1}$	f_2	$e_{4,1}$	$e_{5,1}$
$e_{4,1}$	v_2	$e_{4,2}$	f_2	$e_{5,1}$	$e_{3,2}$
$e_{4,2}$	v_4	$e_{4,1}$	f_5	$e_{2,2}$	$e_{8,2}$
...	...	...	...	...	...

- **Interesting Query:**

– Given a DCEL description, a line *L* and a half-edge that this line cuts, efficiently find all the faces cut by *L*

How to resolve it?

Queries in DCEL



Face	Edge
f_1	$e_{1,1}$
f_2	$e_{5,1}$
f_3	$e_{5,2}$
f_4	$e_{8,1}$
f_5	$e_{9,2}$

Traversing all edges incident on a vertex v

– Note: we only output the half-edges whose origin is v

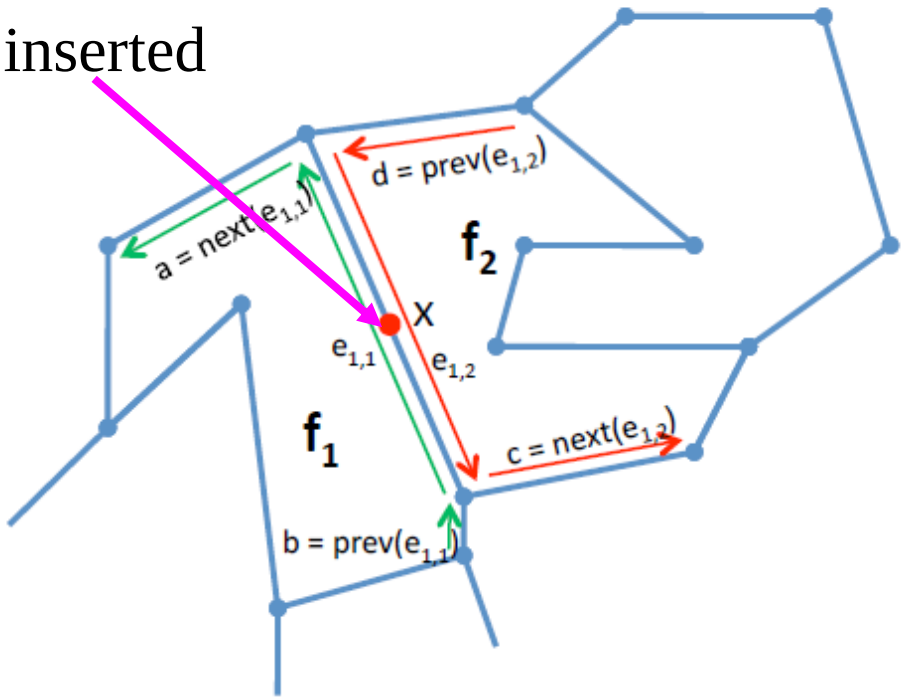
– Given: a half-edge e with the origin at v

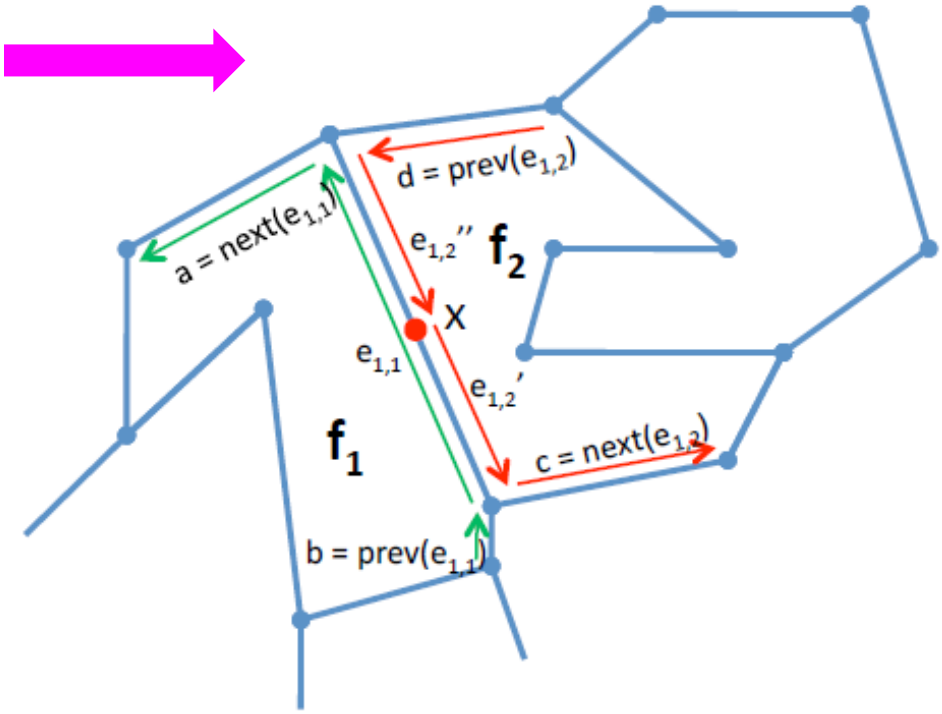
1. Start_edge ← e
2. While next(twin(e)) ≠ start_edge then

e ← next(twin(e))

Adding a new vertex in DCEL

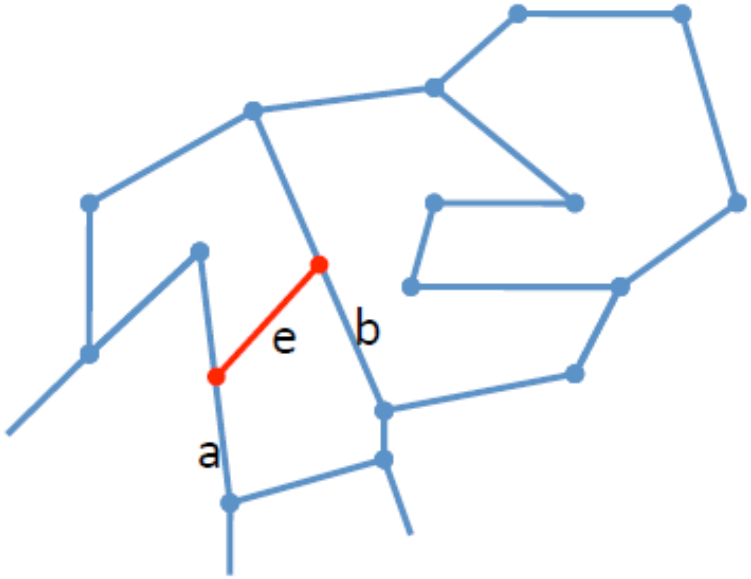
A new vertex *x* is inserted





Update vertex, half-edge, and face records in $O(1)$ time

Other Operations on DCEL

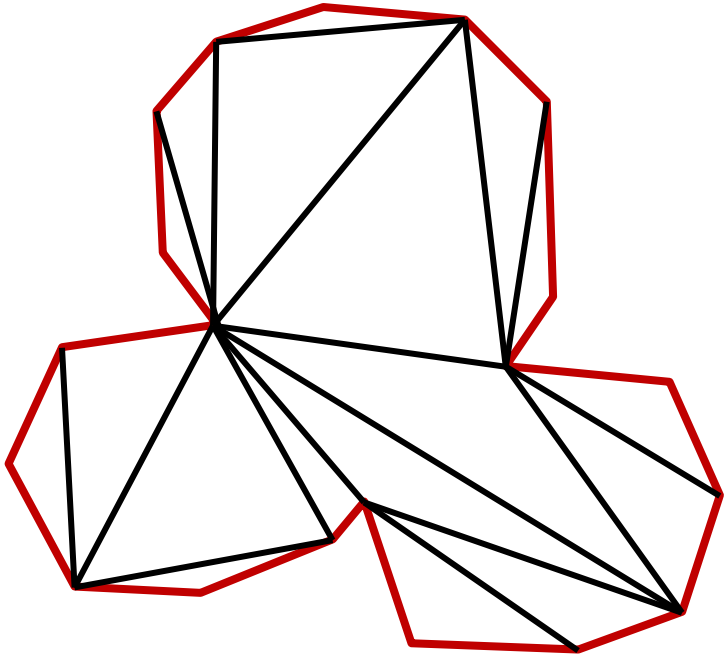


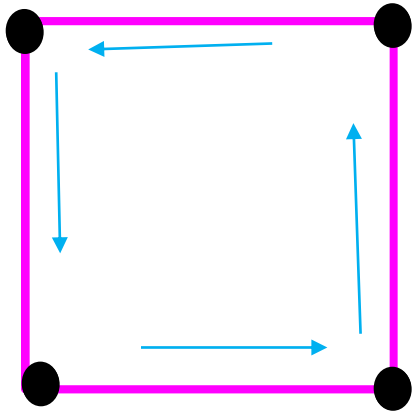
- **Add an Edge**
  - Planar subdivision
  - e is added
  - DCEL can be updated in constant time once the edges a and b are known

Can you implement H-M Algorithm using DCEL?

H-M Algorithm: Example

How do I store a triangulation instance and check the "essentiality" of diagonals sequentially?





Modify DCEL after removing an edge: update vertex, edge, and face records in $O(1)$ time