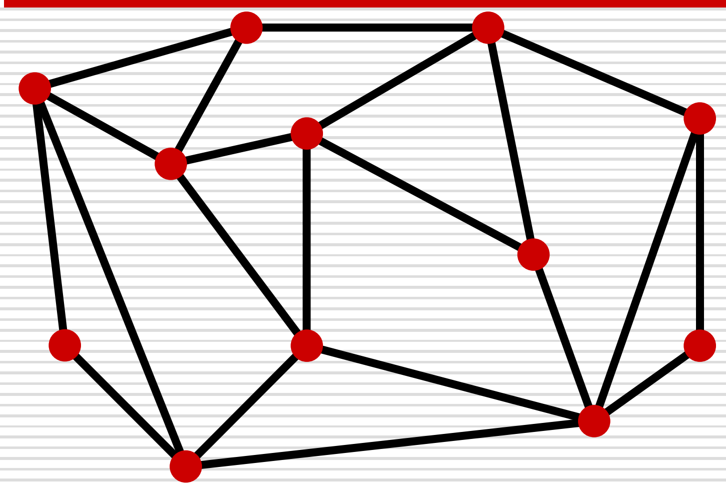
Geometric Modeling

Bing-Yu Chen
National Taiwan University
The University of Tokyo

Definitions and Data Structures of Meshes

- Graph
- Mesh
- Properties of Mesh
- Triangle Meshes
- Mesh Data Structures

Standard Graph Definitions

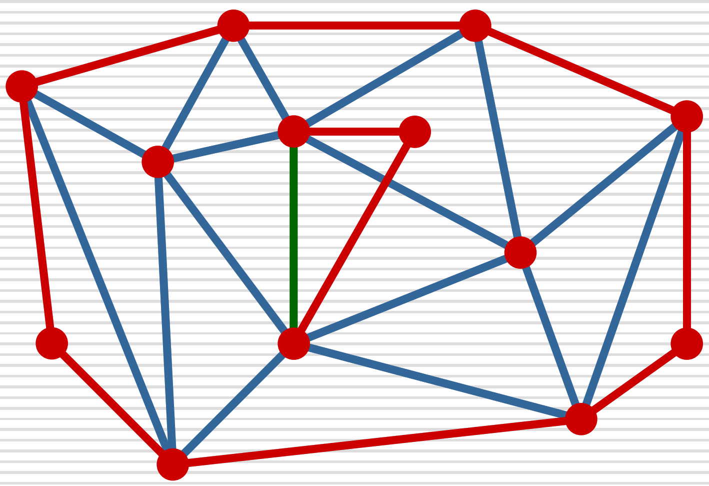


G=<**V**,**E**>
V=vertices={A,B,C,D,E,F,G,H,I,J,K,L}
E=edges=
{(A,B),(B,C),(C,D),(D,E),(E,F),(F,G),
(G,H),(H,A),(A,J),(A,G),(B,J),(K,F),
(C,L),(C,I),(D,I),(D,F),(F,I),(G,K),
(J,L),(J,K),(K,L),(L,I)}

Vertex degree (valence)=number of edges incident on vertex
Ex. deg(J)=4, deg(H)=2
***k*-regular** graph=graph whose vertices all have degree *k*

Face: cycle of vertices/edges which cannot be shortened
F=faces=
{(A,H,G),(A,J,K,G),(B,A,J),(B,C,L,J),(C,I,J),(C,D,I),
(D,E,F),(D,I,F),(L,I,F,K),(L,J,K),(K,F,G)}

Meshes



Mesh: straight-line graph embedded in R^3

Boundary edge: adjacent to exactly *one* face
Regular edge: adjacent to exactly *two* faces
Singular edge: adjacent to more than *two* faces

Corners $\subseteq$ V x F
Half-edges $\subseteq$ E x F

Closed Mesh: mesh with no boundary edges
Manifold Mesh: mesh with no singular edges

1-Manifolds

- What is 1-manifolds ?
  - every point on a 1-manifold has some arbitrarily small neighborhood of points around it that can be considered topologically the same as a line

2-Manifolds

- What is 2-manifolds ?
  - every point on a 2-manifold has some arbitrarily small neighborhood of points around it that can be considered topologically the same as a disk in the plane
  - every edge is shared by exactly two triangles and every triangle shares an edge with exactly three neighboring triangles

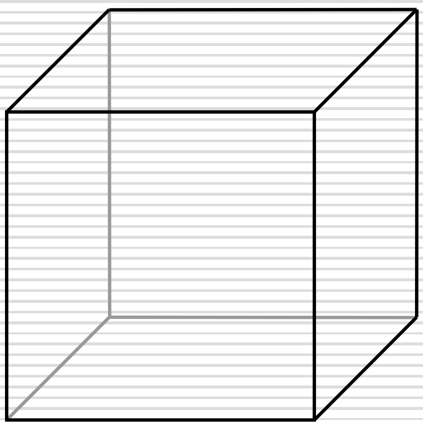
1-Manifold & 2-Manifold Examples

- 1-Manifolds
  - line
  - Circle
  - …
- 2-Manifolds
  - sphere
  - torus
  - cylinder
  - …

Euler's Formula

- Polyhedron
  - a solid that is bounded by a set of polygons whose edges are each a member of an even number of polygons
- Simple Polyhedron
  - a polyhedron that can be deformed into a sphere
- Euler's Formula
  - a simple polyhedron satisfies $V - E + F = 2$

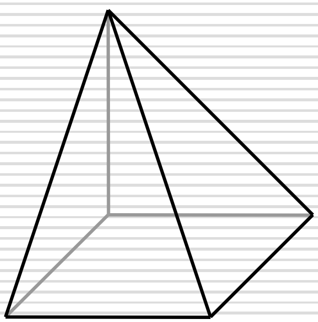
Simple Polyhedra Example



$V = 8$

$E = 12$

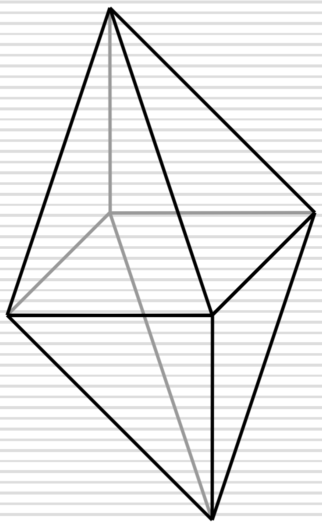
$F = 6$



$V = 5$

$E = 8$

$F = 5$



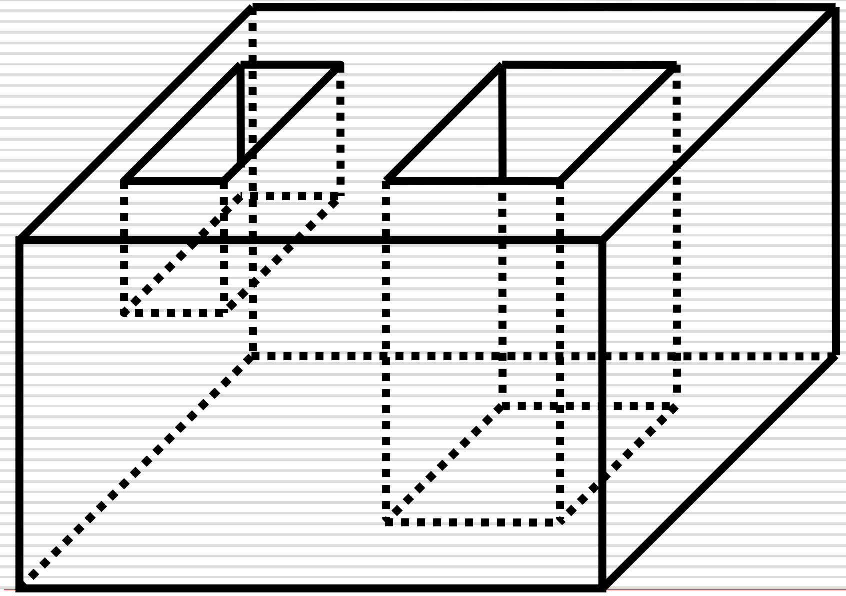
$V = 6$

$E = 12$

$F = 8$

Euler's Formula Applies to 2-Manifolds with Holes

- $V - E + F - H = 2(C - G)$
  - H: the number of holes in the faces
  - G: the number of holes that pass through the object
  - C: the number of separate components



$V = 24$

$E = 36$

$F = 15$

$H = 3$

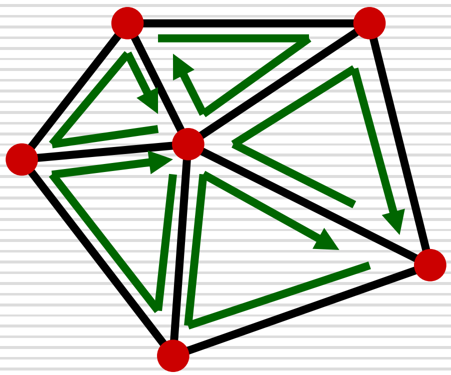
$C = 1$

$G = 1$

if $C = 1$

G is its **genus**

Orientability



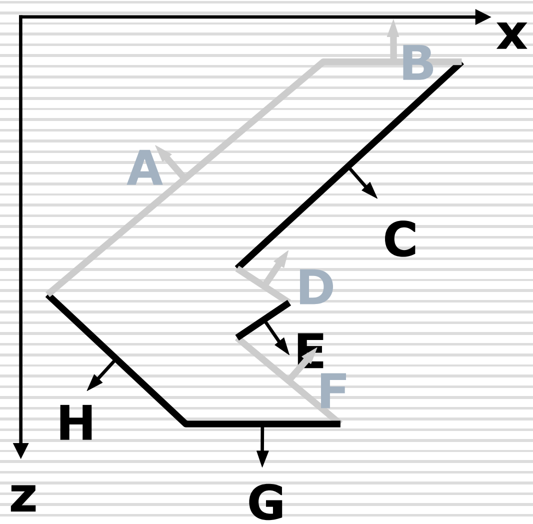
Orientation of a face is clockwise or anticlockwise order in which its vertices and edges are lists

This defines the direction of face **normal**

Oriented
F={(L,J,B),(B,C,L),(L,C,I),
(I,K,L),(L,K,J)}

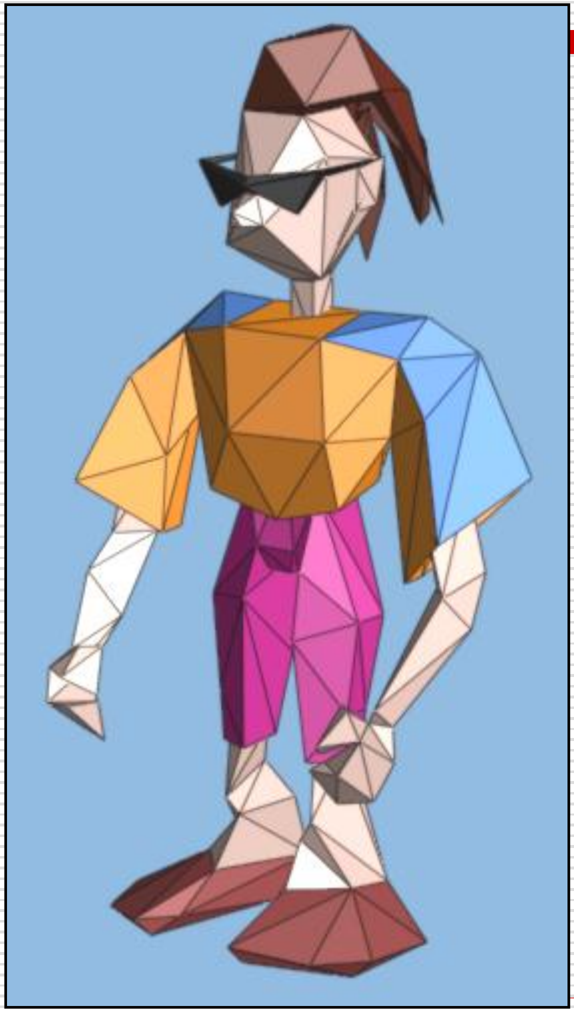
Not Oriented
F={(B,J,L),(B,C,L),(L,C,I),
(L,I,K),(L,K,J)}

Straight line graph is **orientable** if orientations of its faces can be chosen so that each edge is oriented in *both* directions



Back Face Culling = Front Facing

Definitions of Triangle Meshes



[Hoppe 99']

$\{f_1\} : \{ v_1 , v_2 , v_3 \}$

$\{f_2\} : \{ v_3 , v_2 , v_4 \}$

…

$\{v_1\} : (x,y,z)$

$\{v_2\} : (x,y,z)$

…

$\{f_1\}$: *"skin material"*

$\{f_2\}$: *"brown hair"*

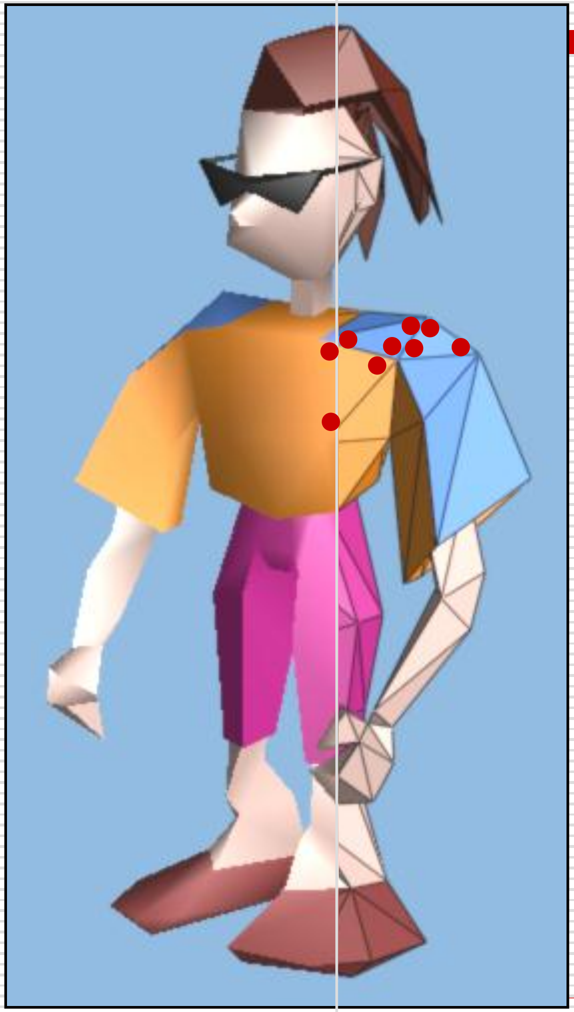
…

connectivity

geometry

face attributes

Definitions of Triangle Meshes



[Hoppe 99']

$\{f_1\} : \{ v_1 , v_2 , v_3 \}$
$\{f_2\} : \{ v_3 , v_2 , v_4 \}$
...

connectivity

$\{v_1\} : (x,y,z)$
$\{v_2\} : (x,y,z)$
...

geometry

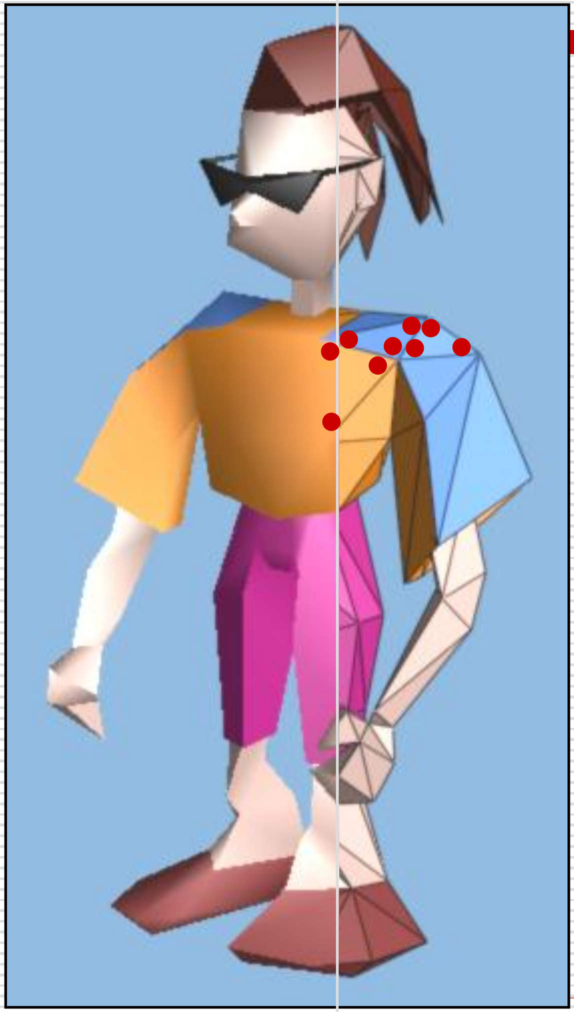
$\{f_1\}$: *"skin material"*
$\{f_2\}$: *"brown hair"*
...

face attributes

$\{v_2,f_1\} : (n_x,n_y,n_z)\ (u,v)$
$\{v_2,f_2\} : (n_x,n_y,n_z)\ (u,v)$
...

corner attributes

Definitions of Triangle Meshes



[Hoppe 99']

$\{f_1\} : \{ v_1 , v_2 , v_3 \}$
$\{f_2\} : \{ v_3 , v_2 , v_4 \}$
…

connectivity

$\{v_1\} : (x,y,z)$
$\{v_2\} : (x,y,z)$
…

geometry

$\{f_1\}$: *"skin material"*
$\{f_2\}$: *"brown hair"*
…

face attributes

$\{v_1\} : (n_x,n_y,n_z)\ (u,v)$
$\{v_2\} : (n_x,n_y,n_z)\ (u,v)$
…

vertex attributes

Mesh Data Structures

- Uses of mesh data:
  - Rendering
  - Geometry queries
    - What are the vertices of face #3?
    - Are vertices i and j adjacent?
    - Which faces are adjacent face #7?
  - Geometry operations
    - Remove/add a vertex/face
    - Mesh simplification
    - Vertex split, edge collapse
- Storage of generic meshes
  - hard to implement efficiently
- Assume: **orientable**, **manifold** and **triangular**

Storing Mesh Data

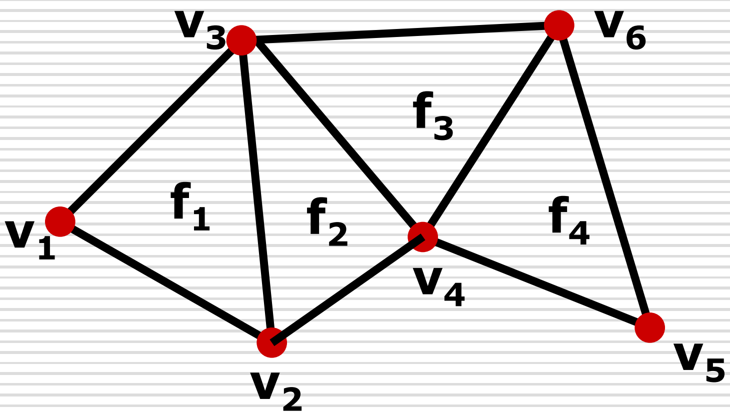
- How “good” is a data structure?
  - Time to construct – preprocessing
  - Time to answer a query
  - Time to perform an operation
    - update the data structure
  - Space complexity
  - Redundancy

1. List of Faces

- List of vertices (coordinates)
- List of faces
  - triplets of pointers to face vertices (c_1,c_2,c_3)
- Queries:
  - What are the vertices of face #3?
    - O(1) – checking the third triplet
  - Are vertices i and j adjacent?
    - A pass over all faces is necessary – NOT GOOD

1. List of Faces

- Example



vertex	coordinate
v_1	(x_1,y_1,z_1)
v_2	(x_2,y_2,z_2)
v_3	(x_3,y_3,z_3)
v_4	(x_4,y_4,z_4)
v_5	(x_5,y_5,z_5)
v_6	(x_6,y_6,z_6)

face	vertices (ccw)
f_1	(v_1,v_2,v_3)
f_2	(v_2,v_4,v_3)
f_3	(v_3,v_4,v_6)
f_4	(v_4,v_5,v_6)

1. List of Faces

- Pros:
  - Convenient and efficient (memory wise)
  - Can represent non-manifold meshes
- Cons:
  - Too simple – not enough information on relations between vertices and faces

OBJ File Format (simple ver.)

- v x y z
- vn i j k
- f v1 // vn1 v2 // vn2 v3 // vn3

2. Adjacency matrix

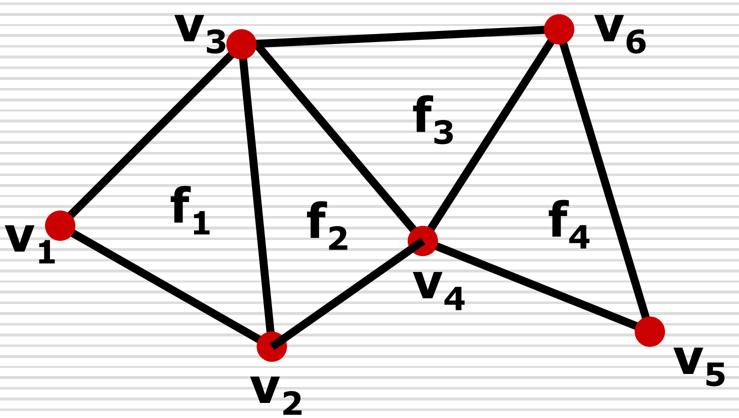
- View mesh as connected graph
- Given n vertices build nxn matrix of adjacency information
  - Entry (i,j) is TRUE value if vertices i and j are adjacent
- Geometric info
  - list of vertex coordinates
- Add faces
  - list of triplets of vertex indices (v_1,v_2,v_3)

2. Adjacency matrix

- Example

vertex	coordinate
v_1	(x_1,y_1,z_1)
v_2	(x_2,y_2,z_2)
v_3	(x_3,y_3,z_3)
v_4	(x_4,y_4,z_4)
v_5	(x_5,y_5,z_5)
v_6	(x_6,y_6,z_6)

face	vertices (ccw)
f_1	(v_1,v_2,v_3)
f_2	(v_2,v_4,v_3)
f_3	(v_3,v_4,v_6)
f_4	(v_4,v_5,v_6)



	v_1	v_2	v_3	v_4	v_5	v_6
v_1		1	1			
v_2	1		1	1		
v_3	1	1		1		1
v_4		1	1		1	1
v_5				1		1
v_6			1	1	1	

2. Adjacency matrix

- Queries:
  - What are the vertices of face #3?
    - O(1) – checking the third triplet of faces
  - Are vertices i and j adjacent?
    - O(1) – checking adjacency matrix at location (i,j)
  - Which faces are adjacent of vertex j?
    - Full pass on all faces is necessary

2. Adjacency matrix

- Pros:
  - Information on vertices adjacency
  - Stores non-manifold meshes

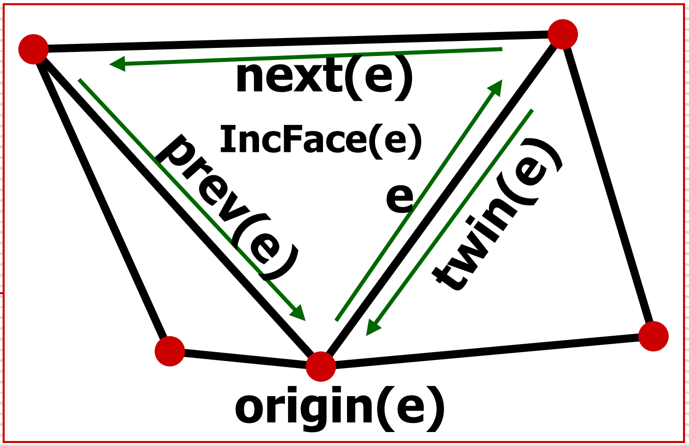
- Cons:
  - Connects faces to their vertices, BUT NO connection between vertex and its face

3. DCEL (Doubly-Connected Edge List)

- Record for each face, edge and vertex
  - Geometric information
  - Topological information
  - Attribute information

- aka Half-Edge Structure

3. DCEL

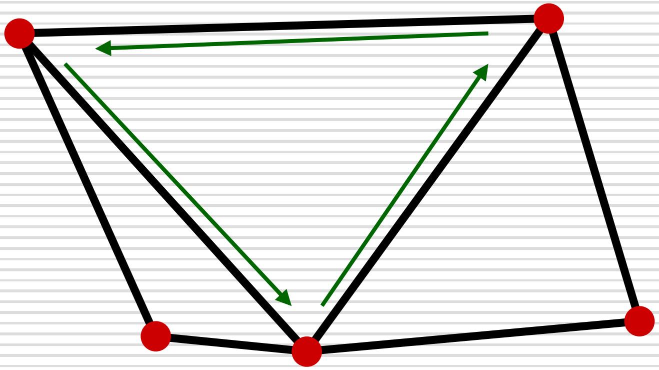


- Vertex record:
  - Coordinates
  - Pointer to one half-edge that has v as its origin
- Face record:
  - Pointer to one half-edge on its boundary
- Half-edge record:
  - Pointer to its origin, origin(e)
  - Pointer to its twin half-edge, twin(e)
  - Pointer to the face it bounds, IncidentFace(e)
    - face lies to left of e when traversed from origin to destination
  - Next and previous edge on boundary of IncidentFace(e)

3. DCEL

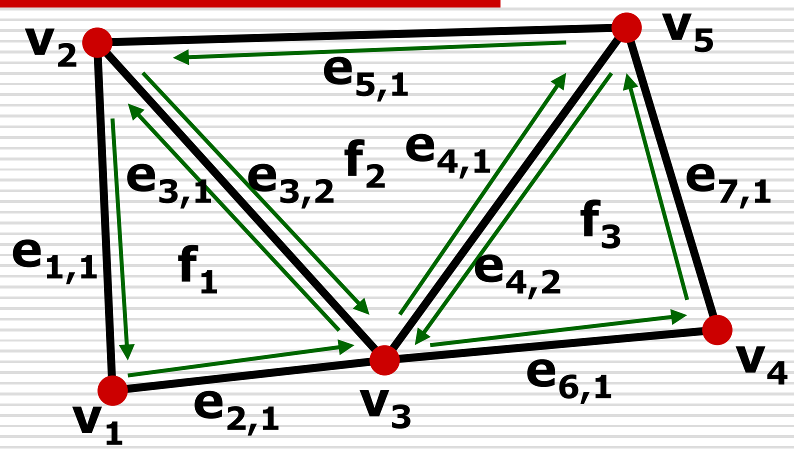
- Operations supported:
  - Walk around boundary of given face
  - Visit all edges incident to vertex v

- Queries:
  - Most queries are O(1)



3. DCEL

- Example

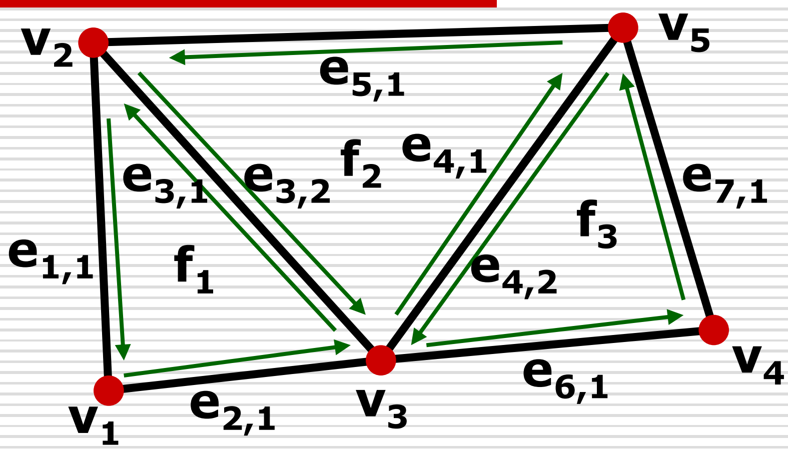


vertex	coordinate	IncidentEdge
v_1	(x_1,y_1,z_1)	$e_{2,1}$
v_2	(x_2,y_2,z_2)	$e_{1,1}$
v_3	(x_3,y_3,z_3)	$e_{4,1}$
v_4	(x_4,y_4,z_4)	$e_{7,1}$
v_5	(x_5,y_5,z_5)	$e_{5,1}$

face	edge
f_1	$e_{1,1}$
f_2	$e_{3,2}$
f_3	$e_{4,2}$

3. DCEL

- Example

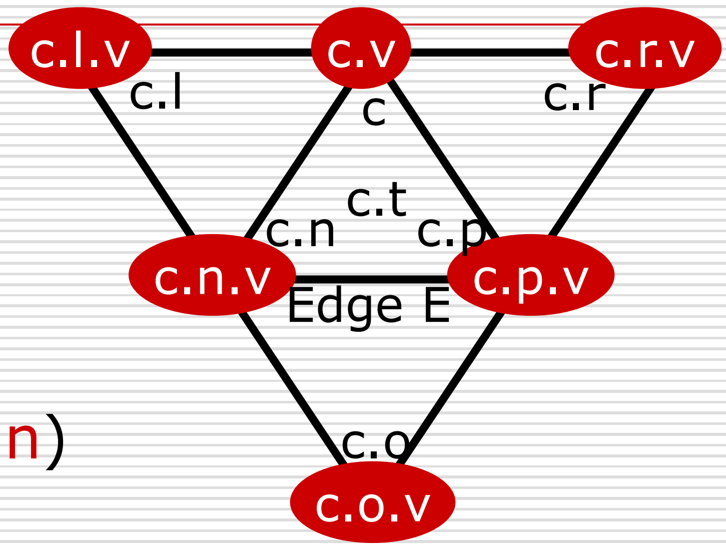


Half-edge	origin	twin	Incident Face	next	prev
$e_{3,1}$	v_3	$e_{3,2}$	f_1	$e_{1,1}$	$e_{2,1}$
$e_{3,2}$	v_2	$e_{3,1}$	f_2	$e_{4,1}$	$e_{5,1}$
$e_{4,1}$	v_3	$e_{4,2}$	f_2	$e_{5,1}$	$e_{3,2}$
$e_{4,2}$	v_5	$e_{4,1}$	f_3	$e_{6,1}$	$e_{7,1}$

3. DCEL

- Pros:
  - All queries in O(1) time
  - All operations are (usually) O(1)
- Cons:
  - Represents only manifold meshes

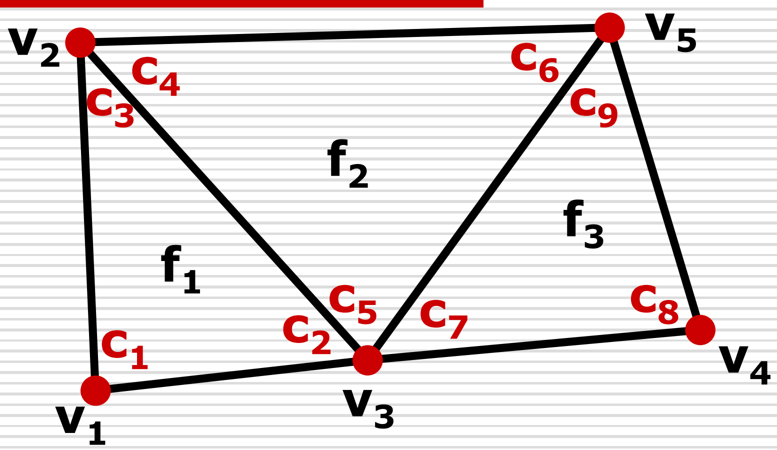
4. Corner table



- Corner c contains:
  - Triangle – c.t
  - Vertex – c.v
  - Next corner in c.t (ccw) – c.n
  - Previous corner – c.p (==c.n.n)
  - Corner opposite – c.o
    - E edge opposite c – not incident on c.v
    - c.o couples triangle T adjacent to c.t across E with vertex of T not incident on E
  - Right corner – c.r – corner opposite c.n (==c.n.o)
  - Left corner – c.l (== c.p.o == c.n.n.o)

4. Corner table

- Example

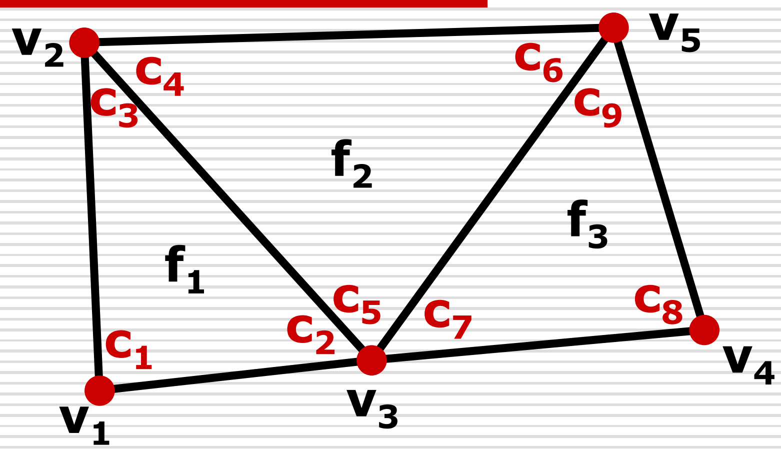


vertex	coordinate	corner
v_1	(x_1,y_1,z_1)	c_1
v_2	(x_2,y_2,z_2)	c_3
v_3	(x_3,y_3,z_3)	c_2
v_4	(x_4,y_4,z_4)	c_8
v_5	(x_5,y_5,z_5)	c_6

face	corners (ccw)
f_1	(c_1,c_2,c_3)
f_2	(c_4,c_5,c_6)
f_3	(c_7,c_8,c_9)

4. Corner table

- Example



corner	c.v	c.t	c.n	c.p	c.o	c.r	c.l
c_1	v_1	f_1	c_2	c_3	c_6	NULL	NULL
c_2	v_3	f_1	c_3	c_1	NULL	NULL	c_6
c_3	v_2	f_1	c_1	c_2	NULL	c_6	NULL
c_4	v_2	f_2	c_5	c_6	c_8	NULL	c_1
c_5	v_3	f_2	c_6	c_4	NULL	c_1	c_8
c_6	v_5	f_2	c_4	c_5	c_1	c_8	NULL

4. Corner table

- Pros:
  - All queries in O(1) time
  - All operations are (usually) O(1)

- Cons:
  - Represents only manifold meshes
  - High redundancy (but not too high...)

4. Corner table

- Queries:
  - What are the vertices of face #3?
    - Check c.v of corners 7, 8, 9
  - Are vertices i and j adjacent?
    - Scan all corners of vertex i, check if c.p.v or c.n.v are j
  - Which faces are adjacent to vertex j?
    - Check c.t of all corners of vertex j