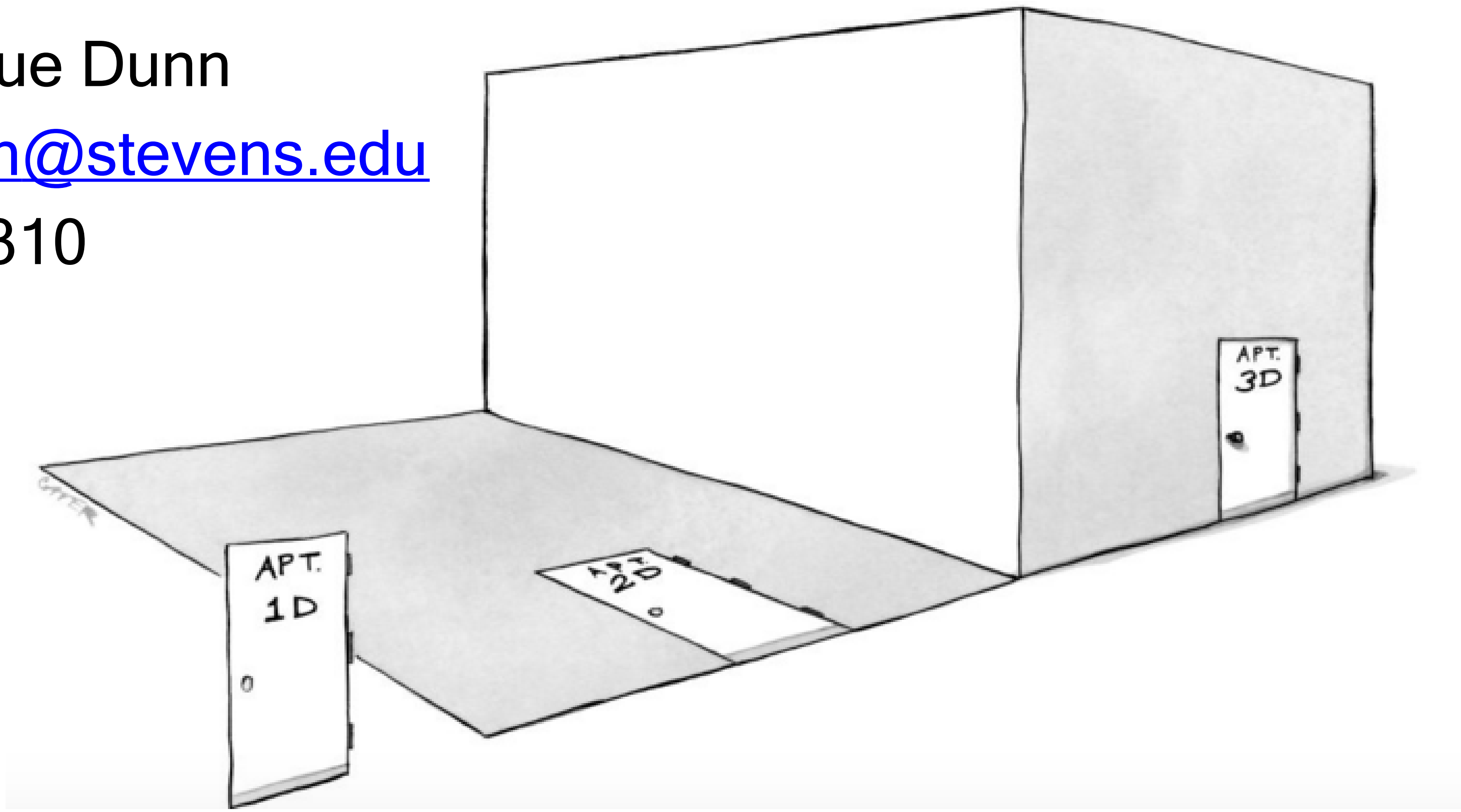
CS 532: 3D Computer Vision
Lecture 10

Enrique Dunn
edunn@stevens.edu
Lieb 310



Homework # 4 available on Canvas

Due Nov 22

Lecture Outline

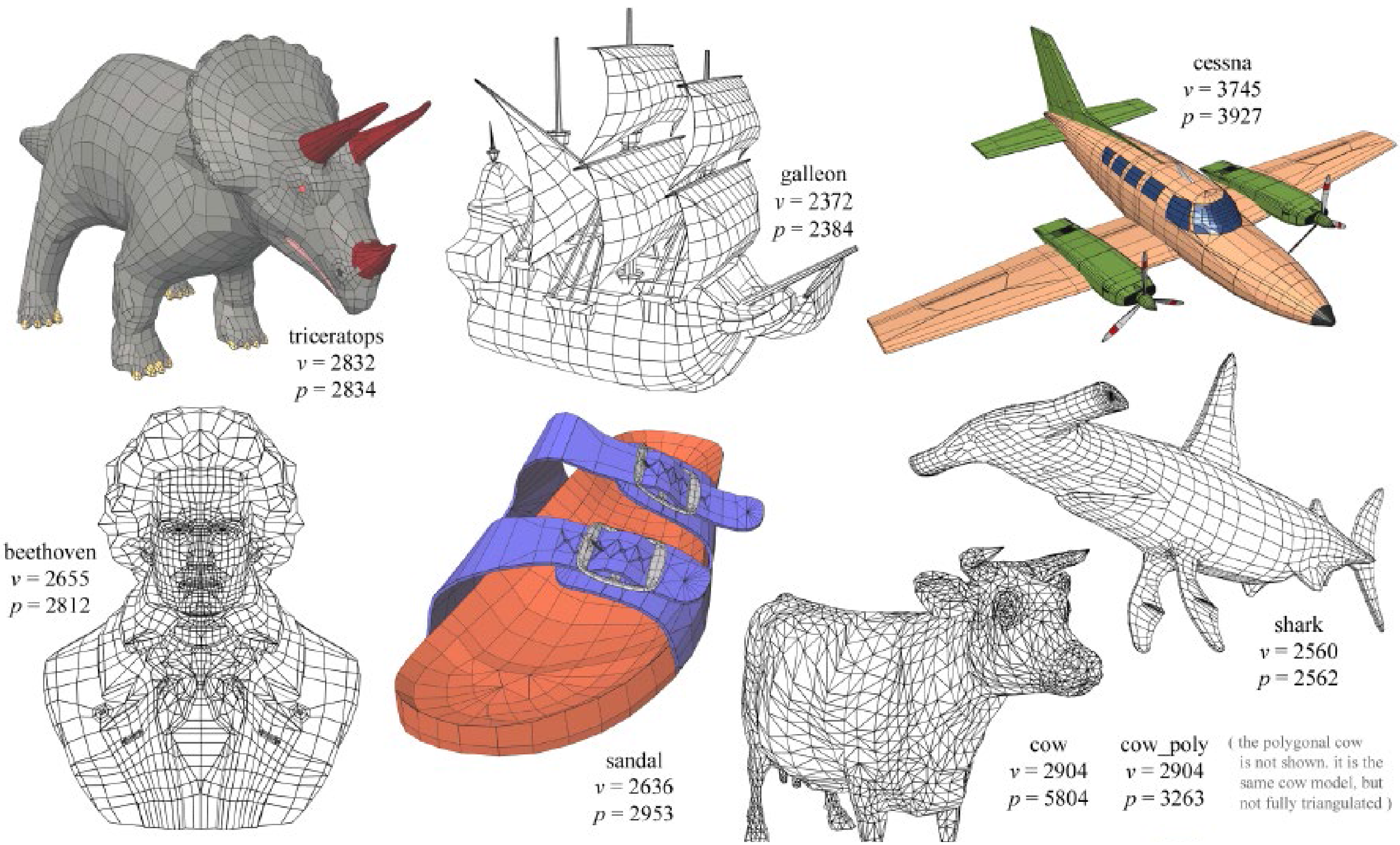
- Meshes

- Slides by:
  - S. Rusinkiewicz, T. Liu and V. Kim (Princeton University)
  - **Ching-Kuang Shene (Michigan Tech. Univ.)**

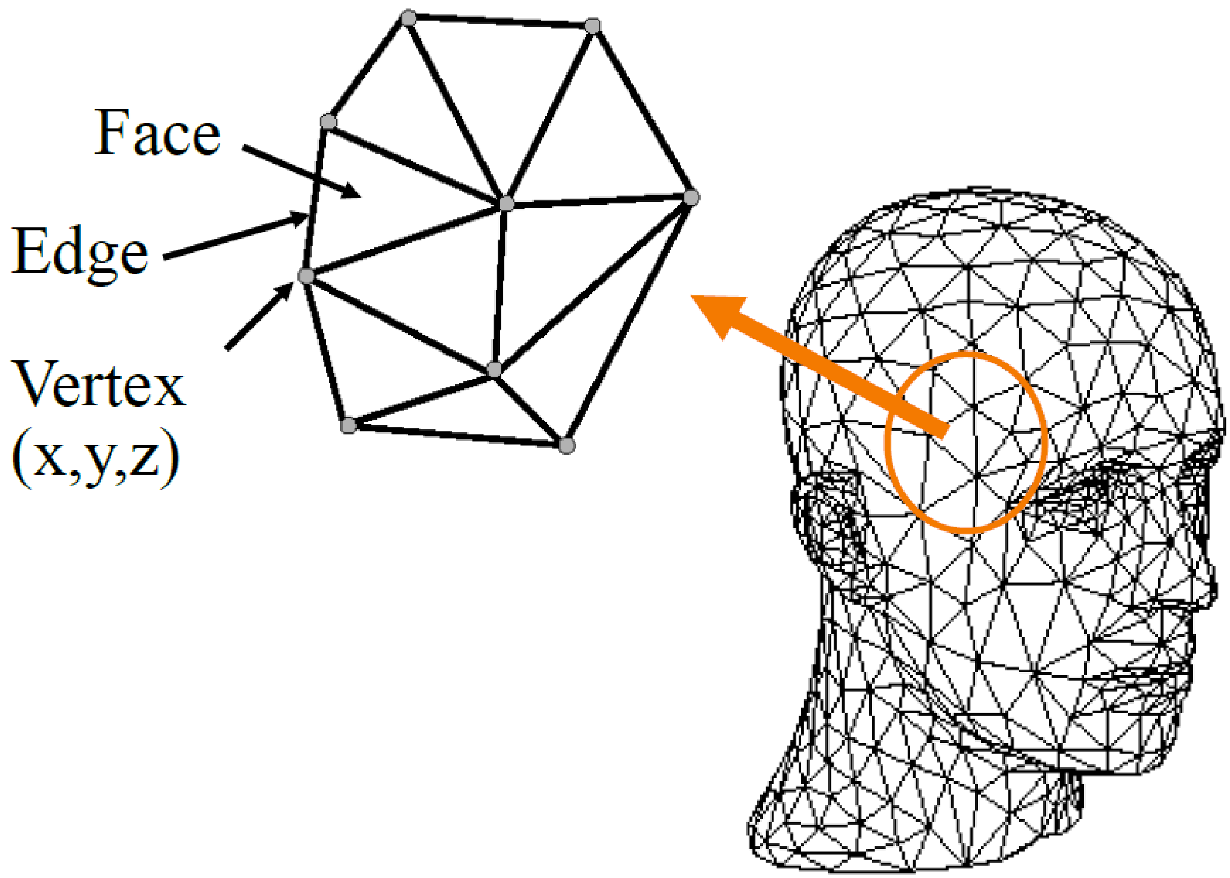
- David M. Mount, CMSC 754: Computational Geometry lecture notes, Department of Computer Science, University of Maryland, Spring 2012
  - Lecture 22

3D Polygonal Mesh

- Set of polygons representing a 2D surface embedded in 3D

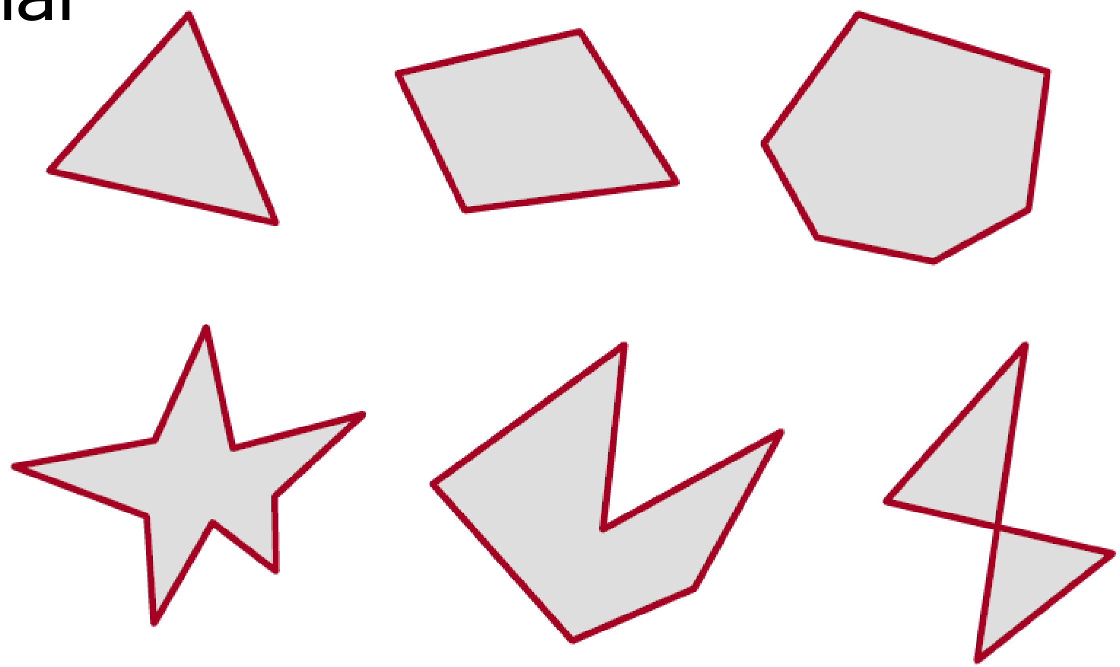


3D Polygonal Mesh



3D Polygon

- Region “inside” a sequence of coplanar points
- Points in counter-clockwise order
  - Define normal

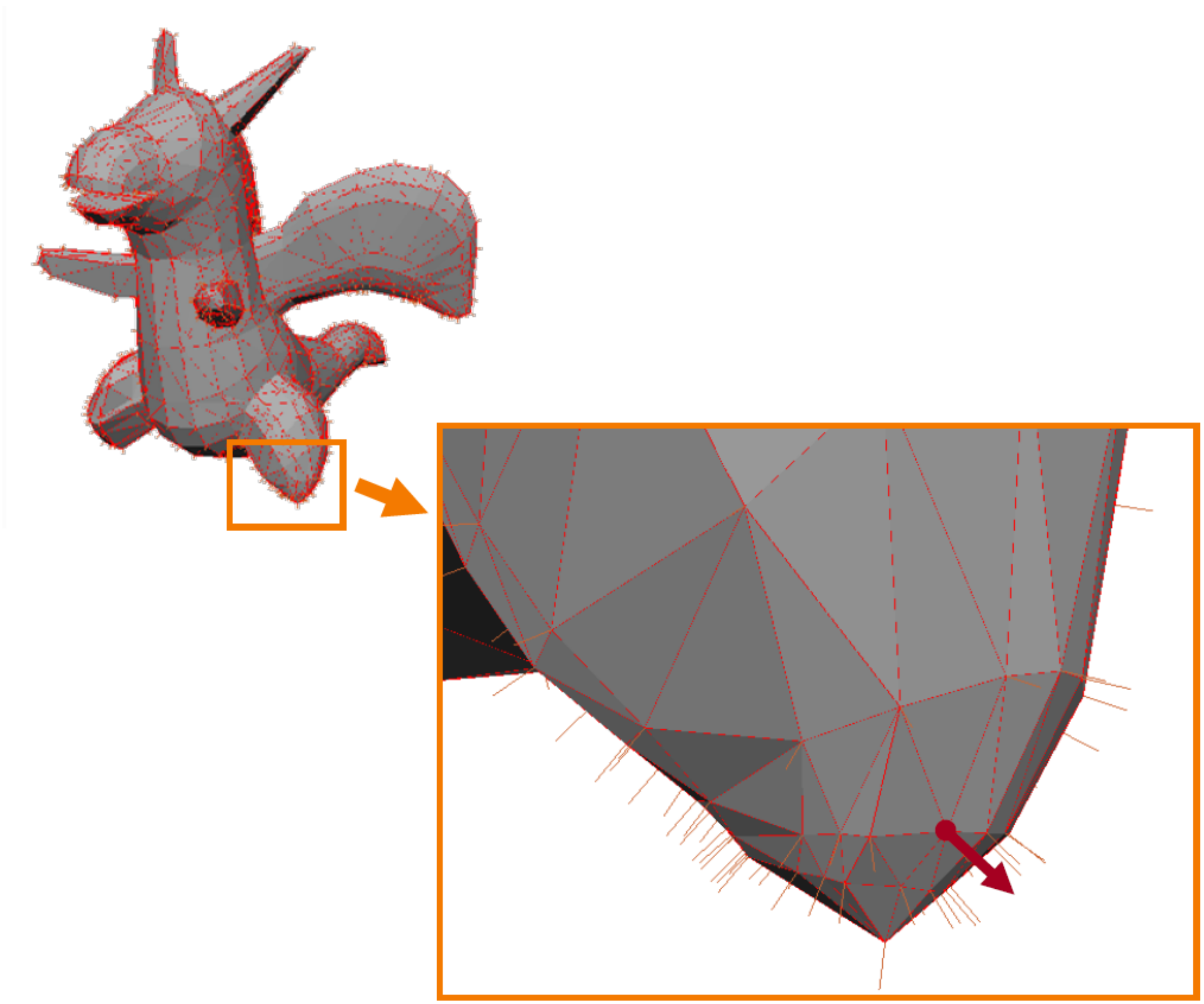


3D Polygonal Meshes

Why are they of interest?

- Simple, common representation
- Rendering with hardware support
- Output of many acquisition tools
- Input to many simulation/analysis tools

Surface Normals



Curvature

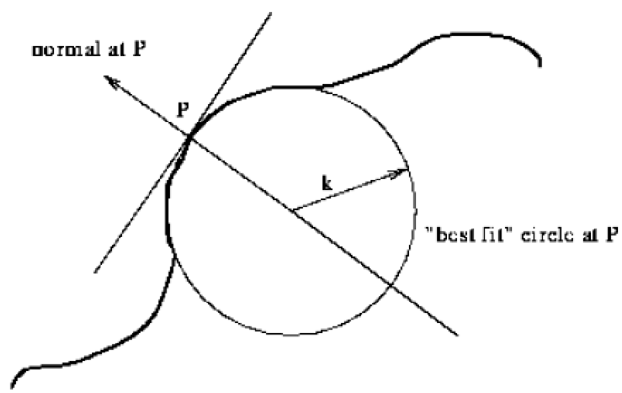
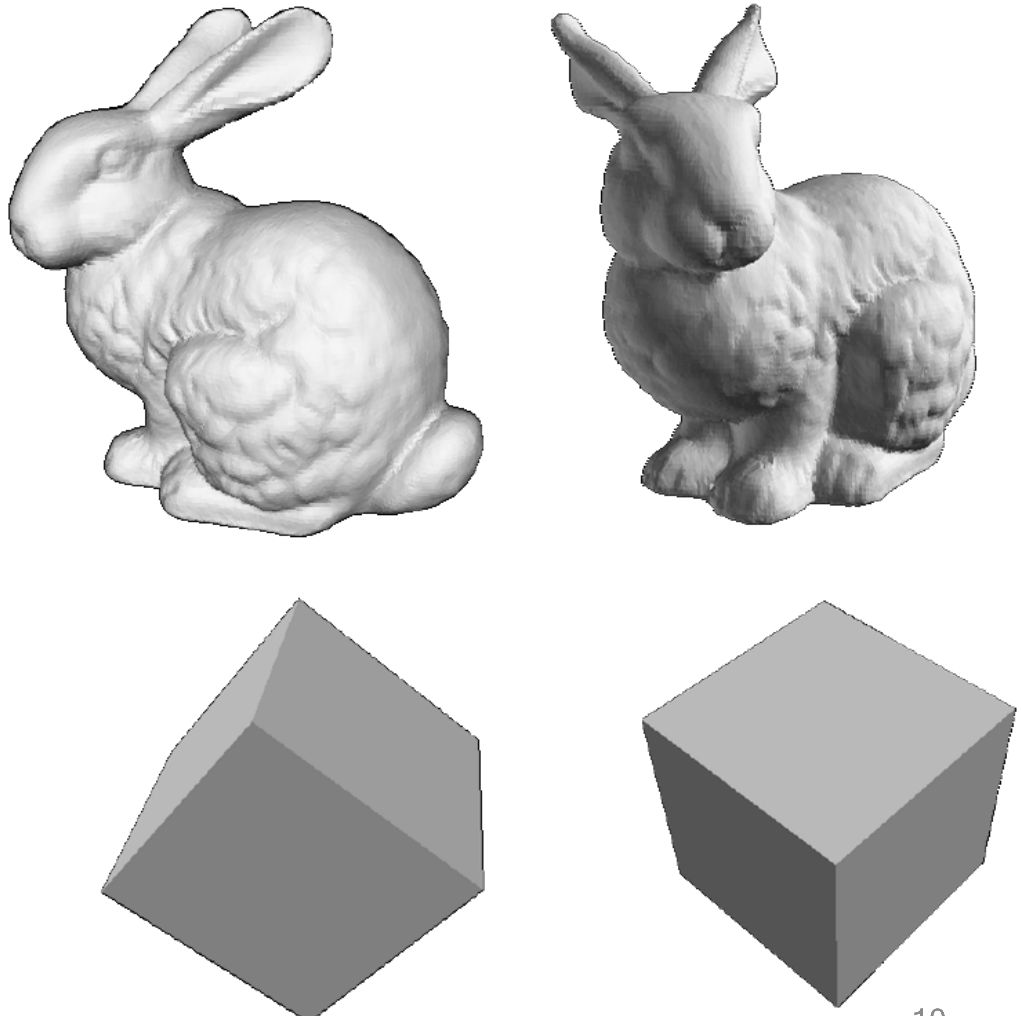


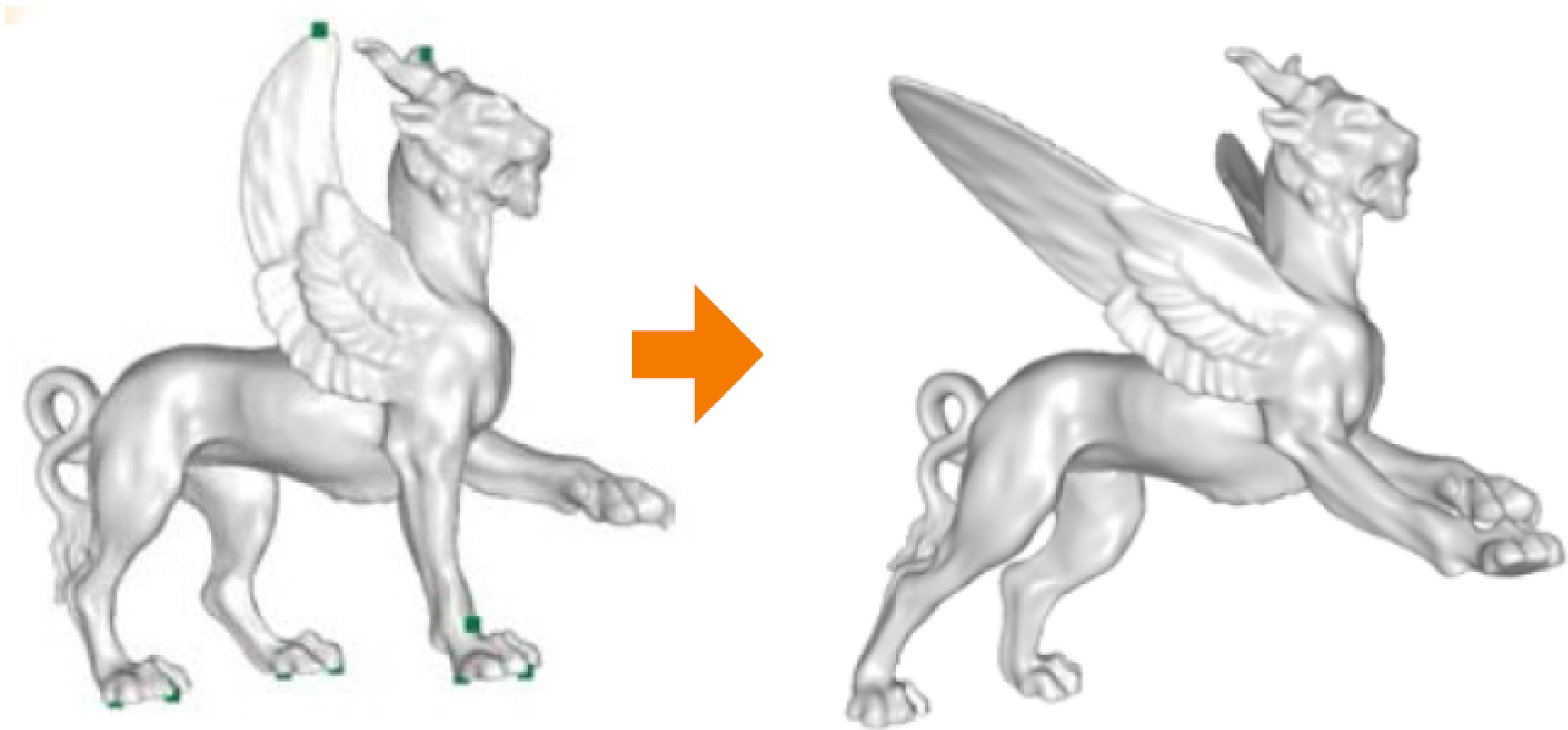
Figure 32: curvature of curve at P is $1/k$

Rigid Transformations

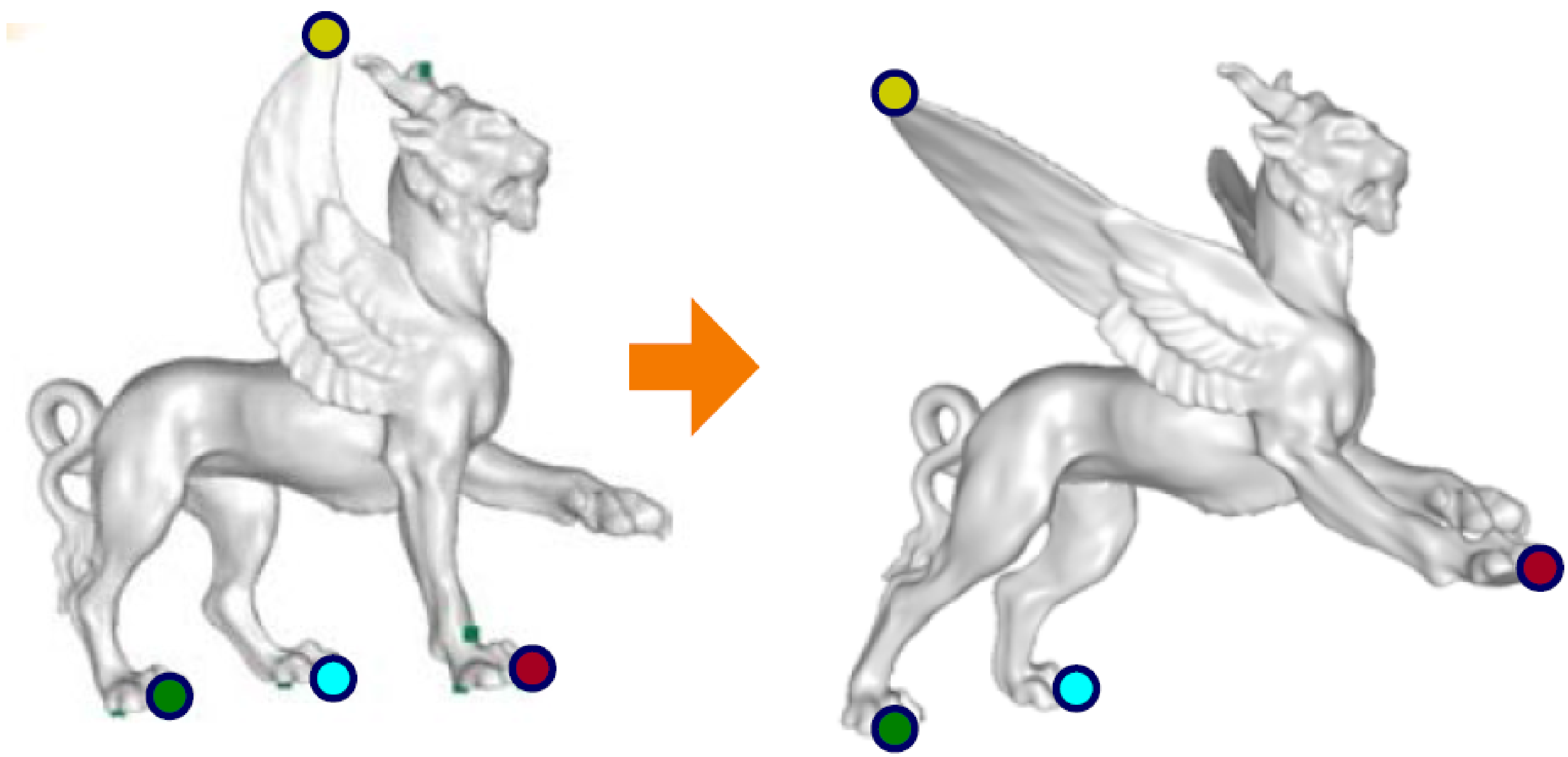
- Compare with implicit representations
  - level sets



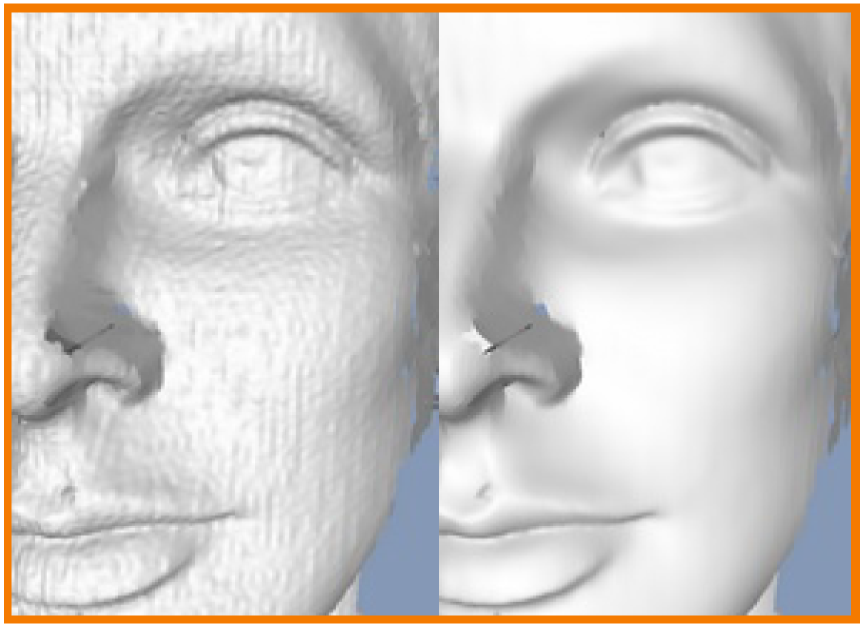
Deformations



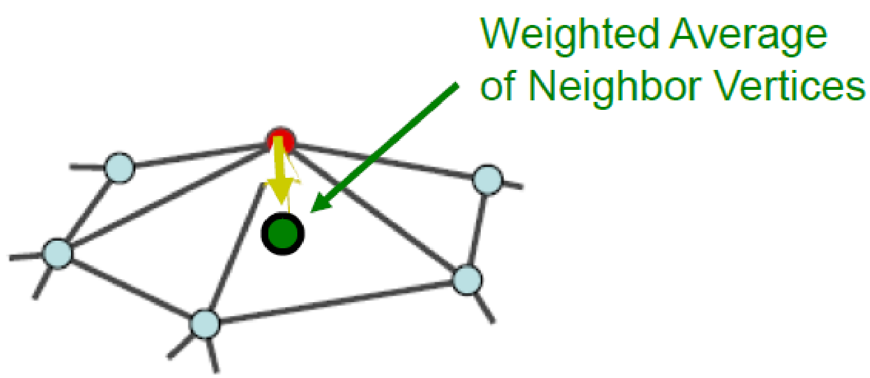
Deformations



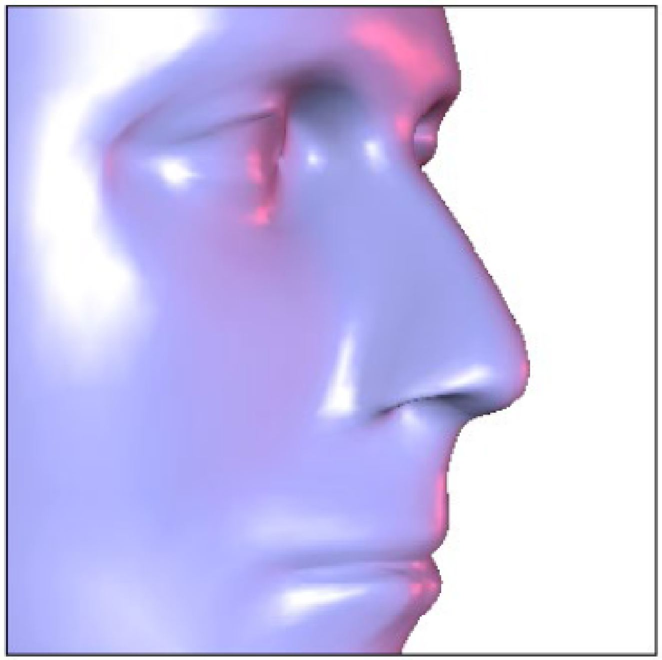
Smoothing

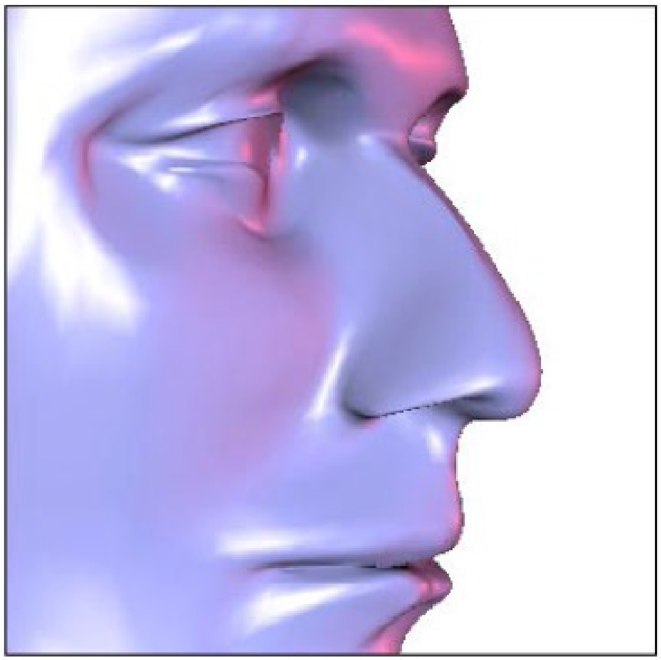


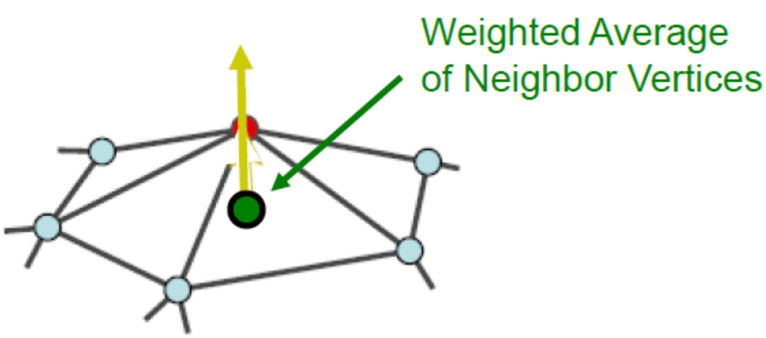
Thouis "Ray" Jones



Sharpen





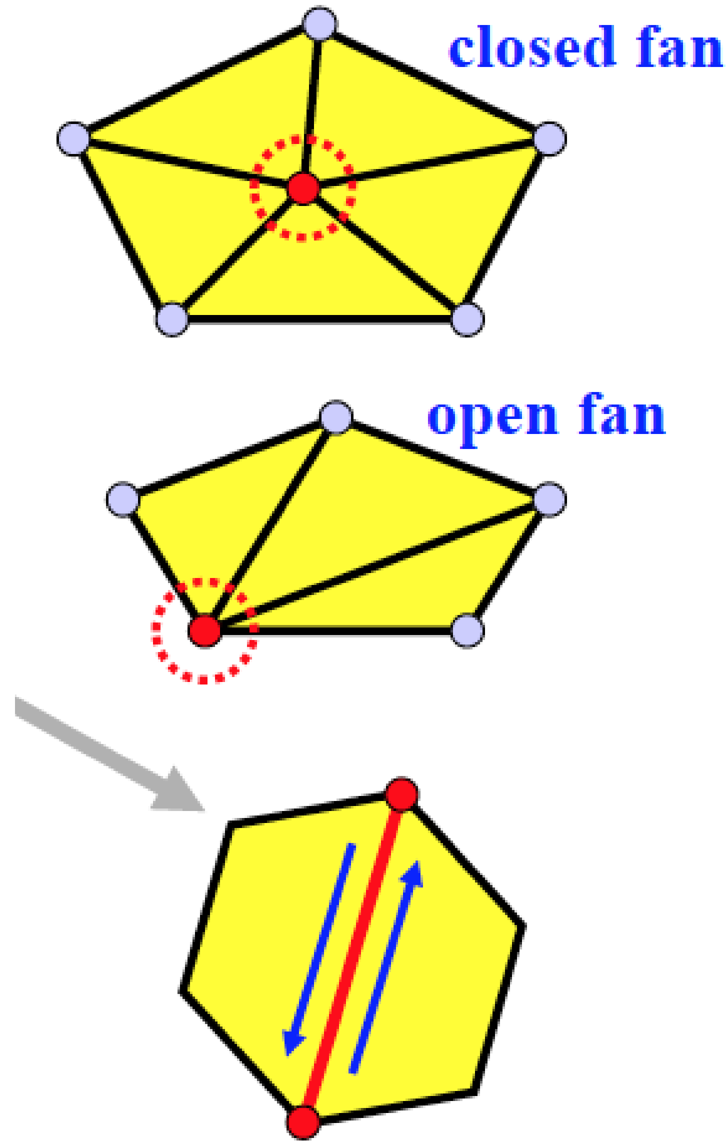


Definitions

- A polygonal mesh consists of three kinds of mesh elements: vertices, edges, and faces.
- The information describing the mesh elements are mesh connectivity and mesh geometry.
- The mesh connectivity, or topology, describes the incidence relations among mesh elements (e.g., adjacent vertices and edges of a face, etc).
- The mesh geometry specifies the position and other geometric characteristics of each vertex.

Definitions

- A polygonal mesh is a manifold if
  - Each edge is incident to only one or two faces, and
  - The faces incident to a vertex form a closed or an open fan.
- The orientation of a face is a cyclic ordering of the incident vertices.
- The orientation of a pair of adjacent faces is compatible, if the two vertices of the single common edge are in opposite order.
- A manifold mesh is orientable if any two adjacent faces have compatible orientation



Definitions

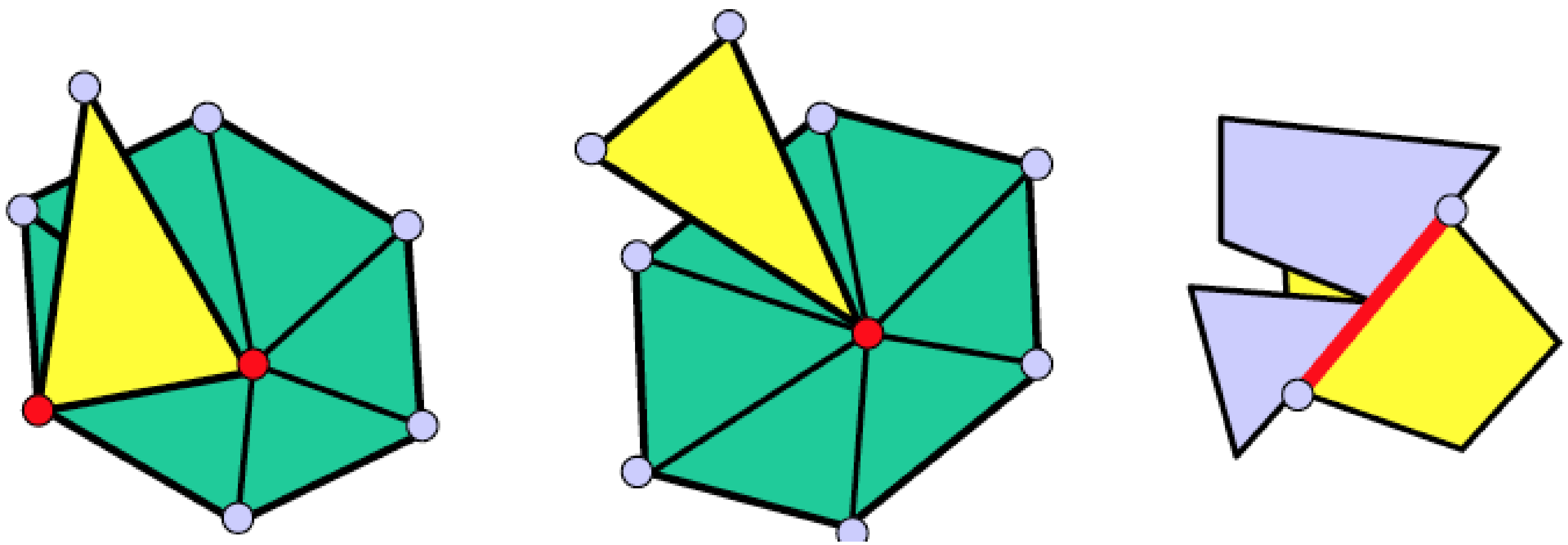
- Boundary edge: adjacent to exactly 1 face
- Regular edge: adjacent to exactly 2 faces
- Singular edge: adjacent to more than 2 faces
- Closed mesh: mesh with no boundary edges

Another Mesh Definition…

- A set of finite number of closed polygons
  - Intersection of inner polygonal areas is empty
  - Intersection of 2 polygons from is either empty, a point or an edge
  - Every edge belongs to at least one polygon
  - The set of all edges which belong only to one polygon are called edges of the polygonal mesh and are either empty or form a single closed polygon

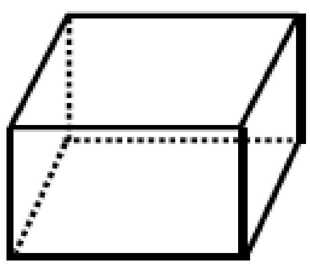
Non-Manifold Meshes

- Manifold Conditions:
  - Each edge is incident to only one or two faces,
  - The faces incident to a vertex form a closed or an open fan.
- The following examples are non-manifold meshes!

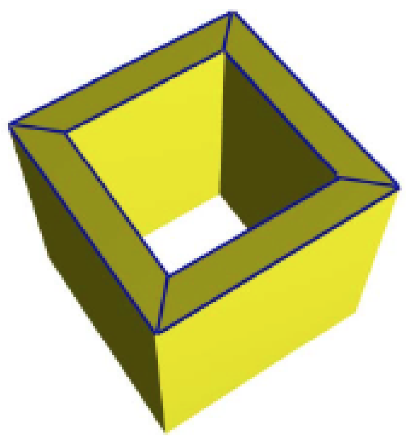


Euler-Poincaré Characteristic

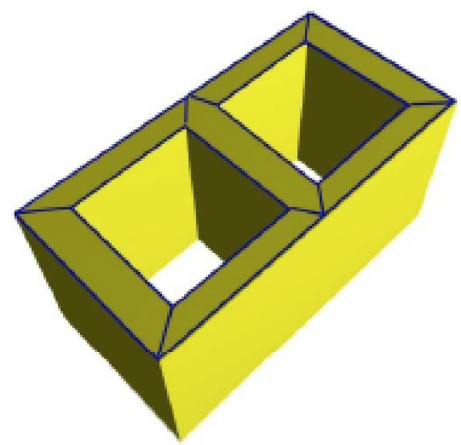
- Given a 2-manifold mesh **M** without boundary, the Euler-Poincaré characteristic of **M** is $\chi(\mathbf{M}) = V - E + F$, where V, E and F are the number of vertices, number of edges, and number of faces.



$V=8$, $E=12$, $F=6$
$\chi(\mathbf{M}) = V-E+F=2$



$V=16$, $E=32$, $F=16$
$\chi(\mathbf{M}) = V-E+F=0$

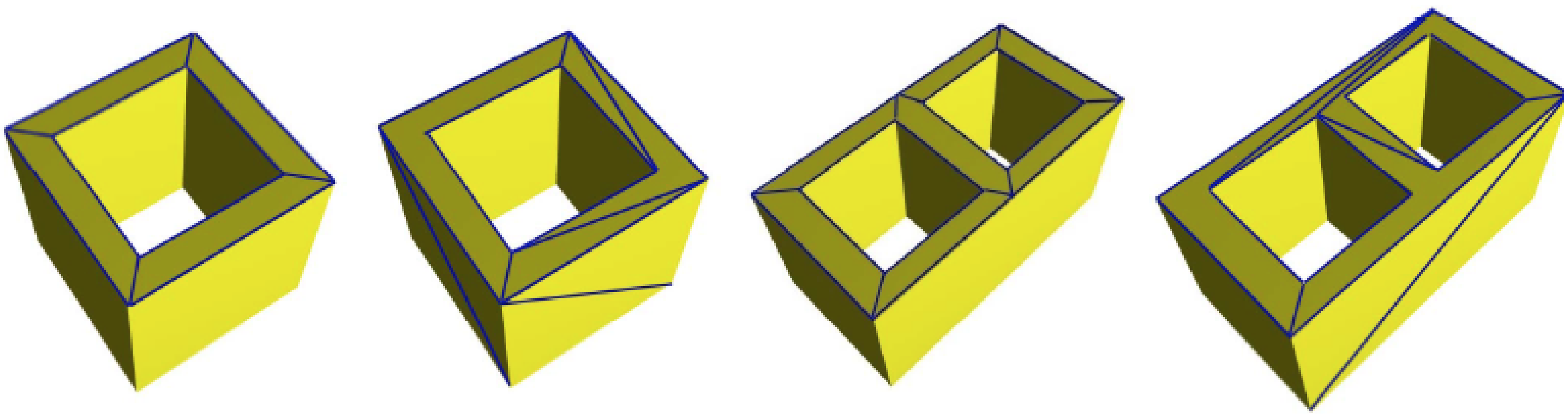


$V=28$, $E=56$, $F=26$
$\chi(\mathbf{M}) = V-E+F=-2$

Euler-Poincaré Characteristic

- **Euler-Poincaré characteristic $\chi(\mathbf{M}) = V\text{-}E\text{+}F$ is independent of tessellation.**

V=24, E=48, F=22
$\chi(\mathbf{M}) = V-E+F = -2$



V=16, E=32, F=16
$\chi(\mathbf{M}) = V-E+F = 0$

V=16, E=36, F=20
$\chi(\mathbf{M}) = V-E+F = 0$

V=28, E=56, F=26
$\chi(\mathbf{M}) = V-E+F = -2$

Homeomorphism

- Two 2-manifold meshes **A** and **B** are **homeomorphic** if their surfaces can be transformed to the other by twisting, squeezing, and stretching without cutting and gluing.
- Thus, boxes, spheres and ellipsoids are homeomorphic to each other.

Homeomorphism

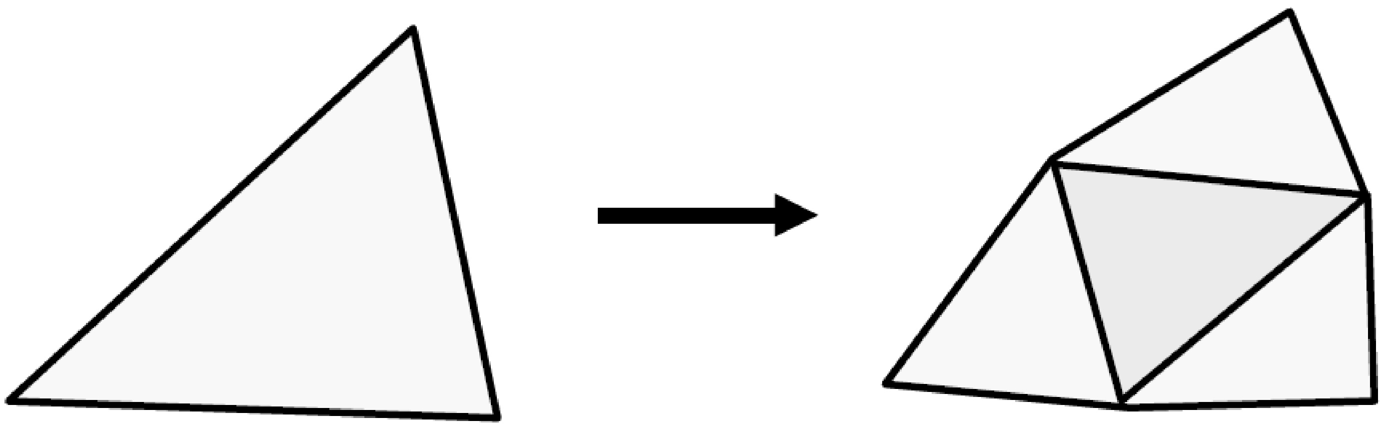
Two orientable 2-manifold meshes without boundary are *homeomorphic* if and only if they have the same Euler-Poincaré characteristic.

Low-level Operations

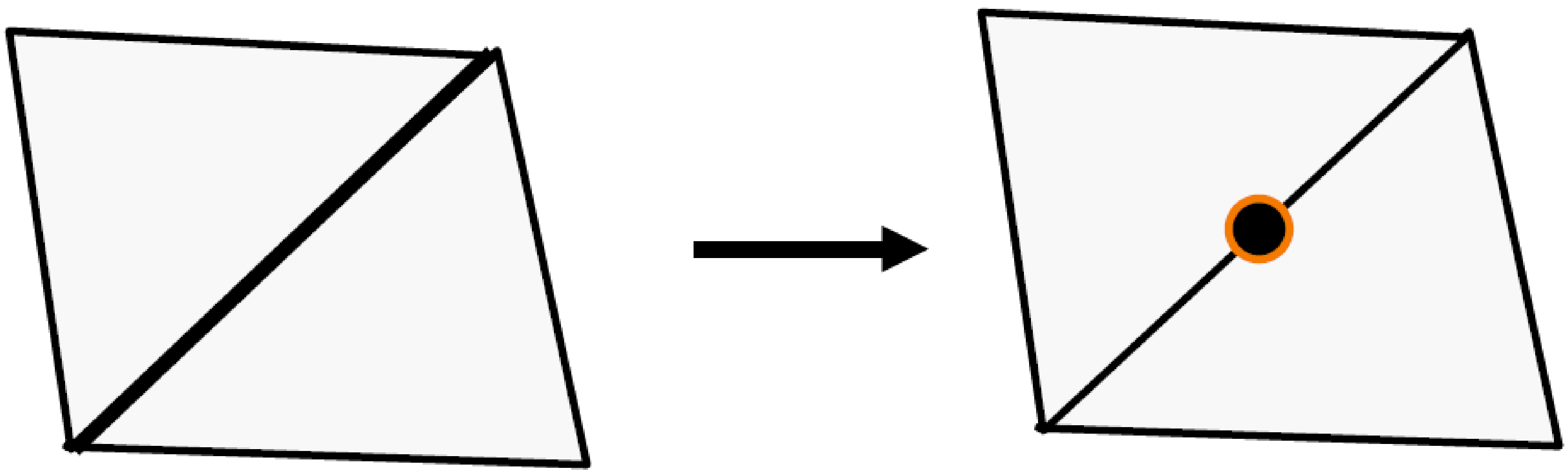
- Subdivide face
- Subdivide edge
- Collapse edge
- Merge vertices
- Remove vertex

Subdivide Face

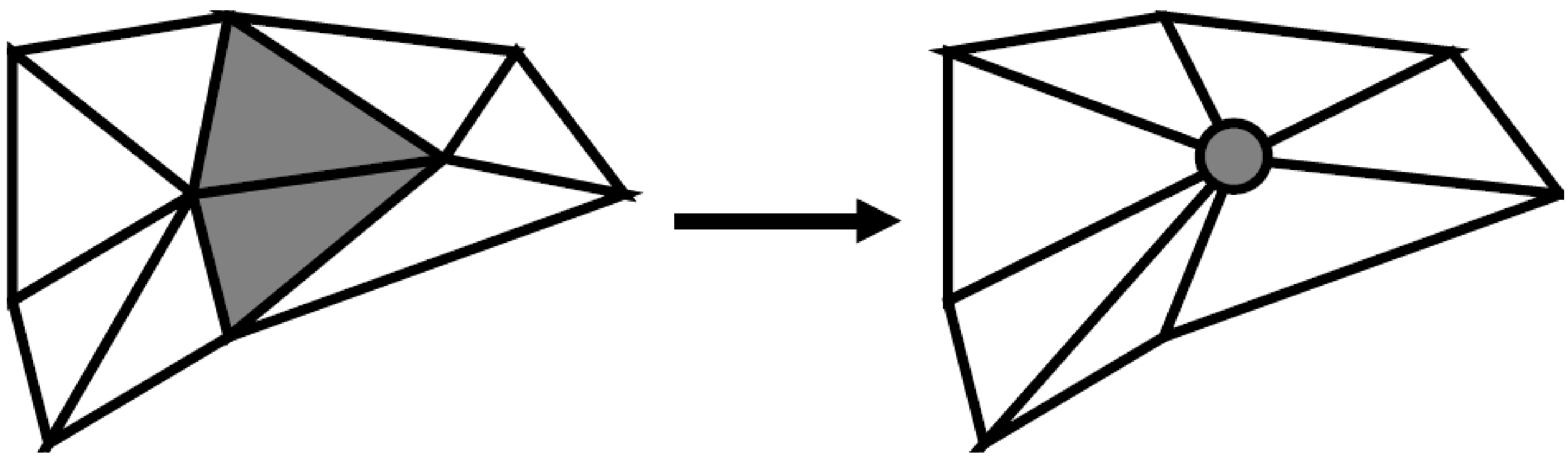
- How should we split current triangle?



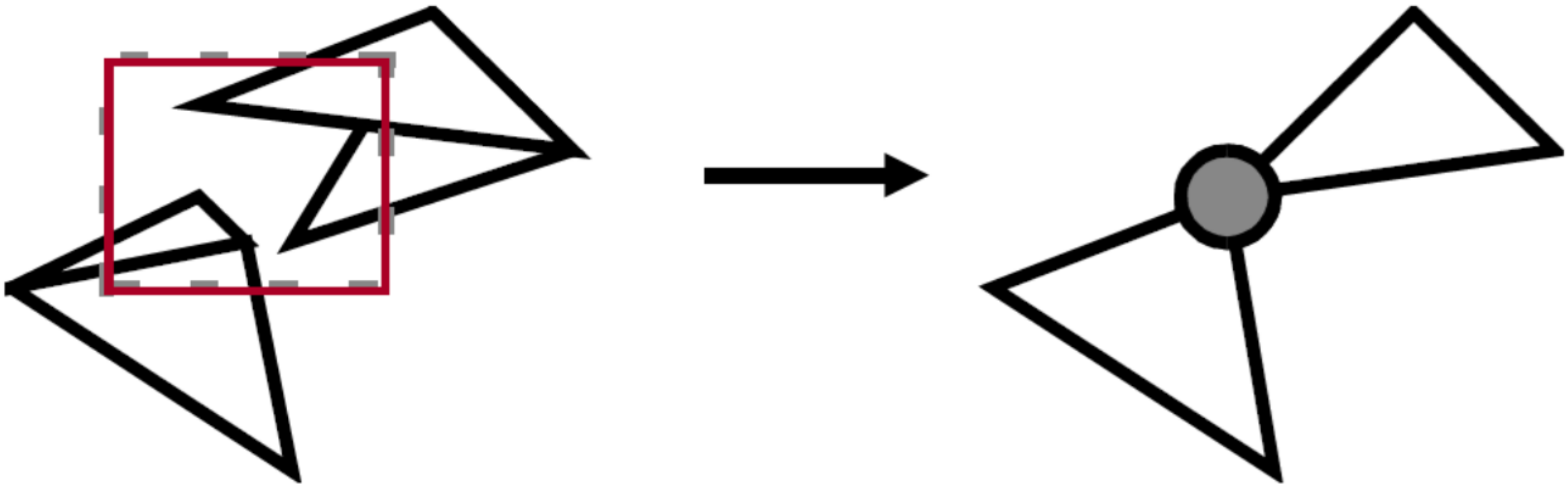
Subdivide Edge



Collapse Edge



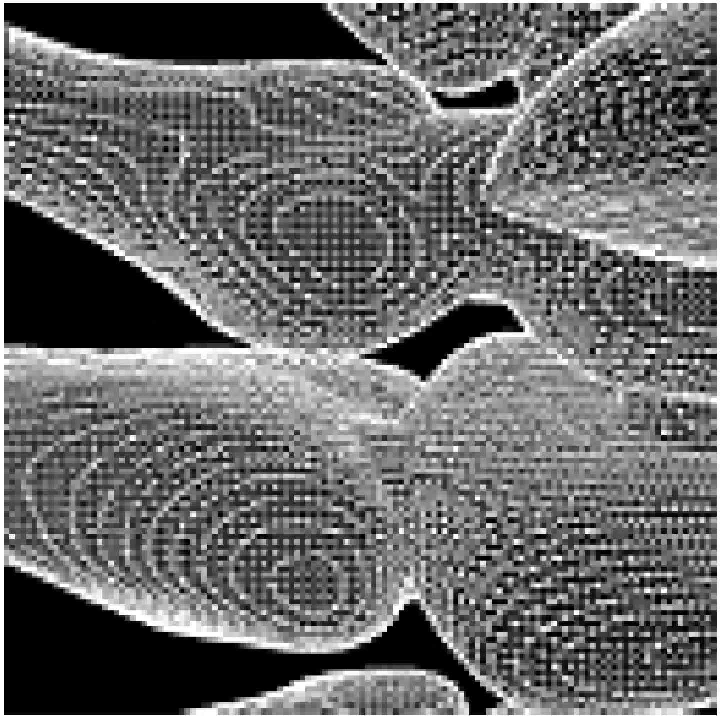
Merge Vertices

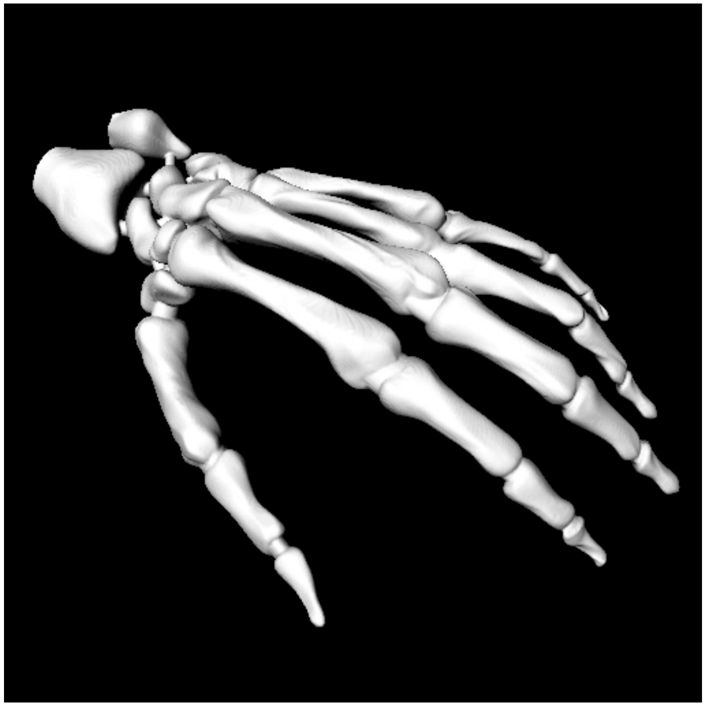


Polygonal Mesh Representation

Important properties of mesh representation

- Efficient traversal of topology
- Efficient use of memory
- Efficient updates



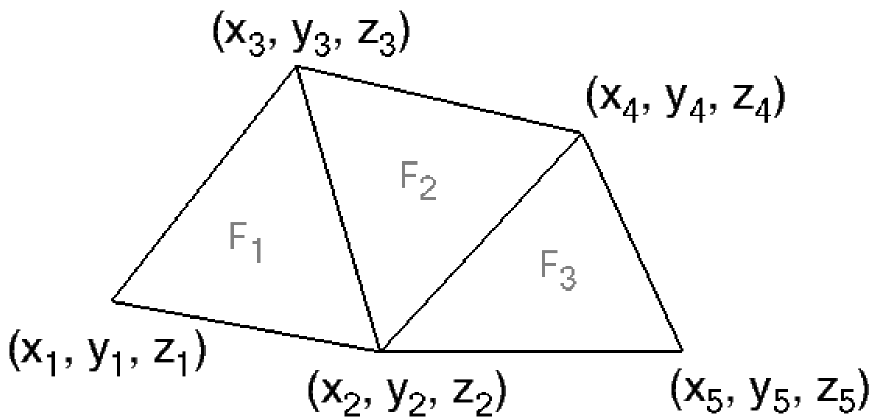


Possible Data Structures

- List of independent faces
- Vertex and face tables
- Adjacency lists
- Winged edge
- Half edge
- etc.

Independent Faces

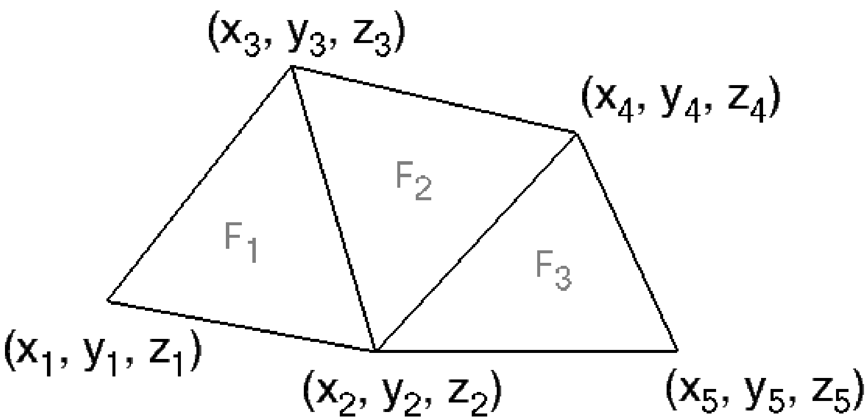
- A.k.a triangle soup
- Each face lists vertex coordinates
  - Redundant vertices
  - No adjacency information



FACE TABLE	
F_1	(x_1, y_1, z_1) (x_2, y_2, z_2) (x_3, y_3, z_3)
F_2	(x_2, y_2, z_2) (x_4, y_4, z_4) (x_3, y_3, z_3)
F_3	(x_2, y_2, z_2) (x_5, y_5, z_5) (x_4, y_4, z_4)

Vertex and Face Tables

- Each face lists vertex references
  - Shared vertices
  - Still no adjacency information

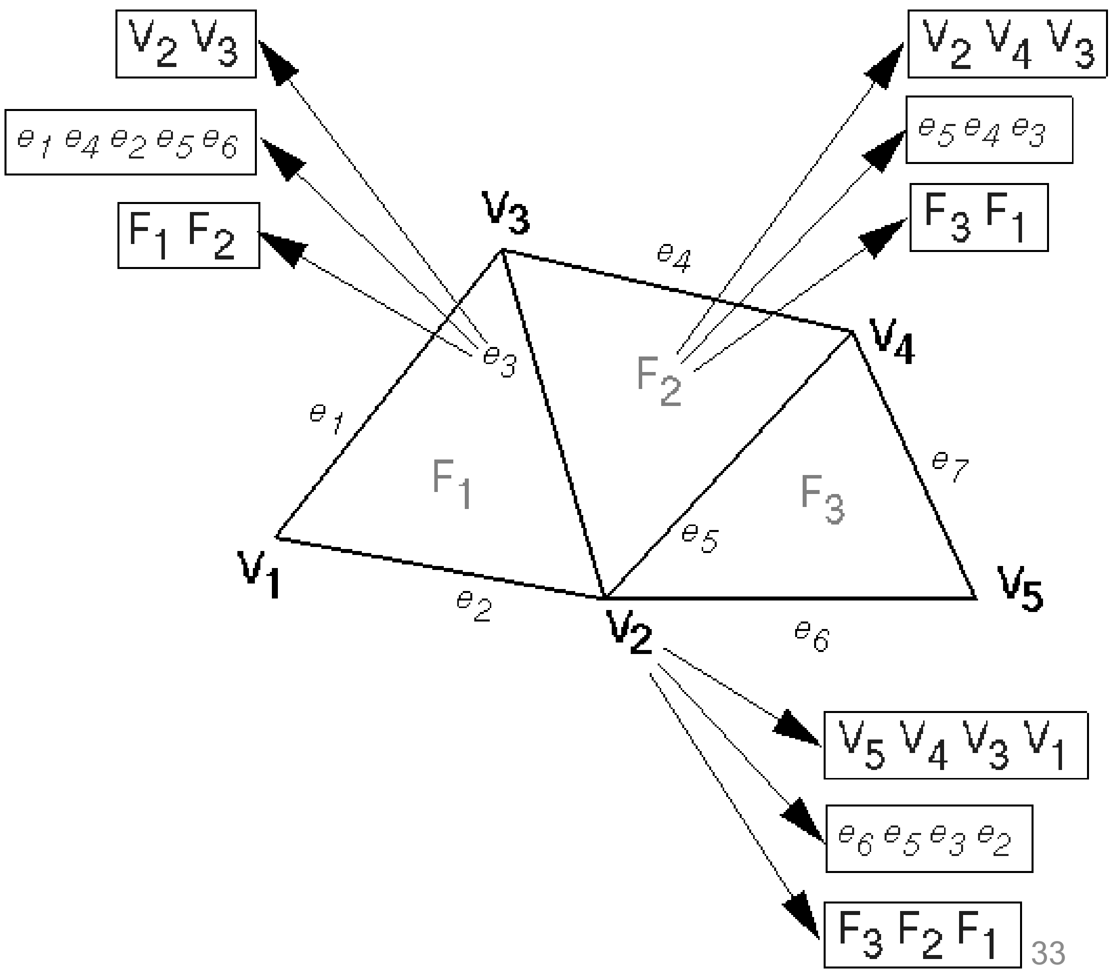


VERTEX TABLE			
V_1	X_1	Y_1	Z_1
V_2	X_2	Y_2	Z_2
V_3	X_3	Y_3	Z_3
V_4	X_4	Y_4	Z_4
V_5	X_5	Y_5	Z_5

FACE TABLE			
F_1	V_1	V_2	V_3
F_2	V_2	V_4	V_3
F_3	V_2	V_5	V_4

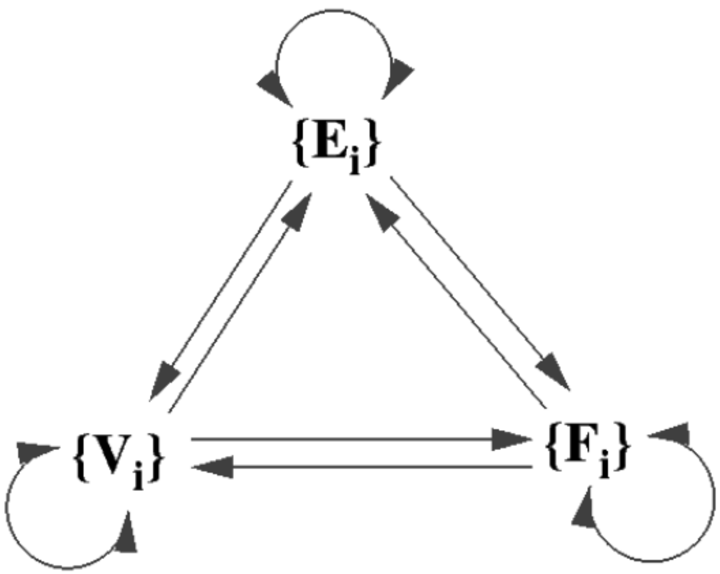
Adjacency Lists

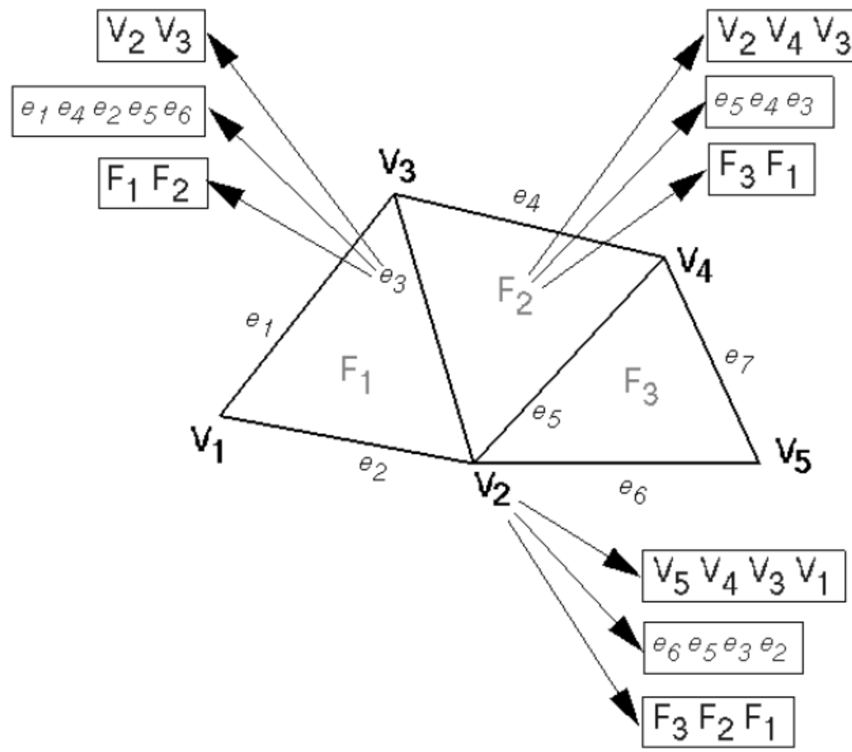
- Store all vertex, edge and face adjacencies
  - Efficient adjacency traversal
  - Extra storage requirements



Partial Adjacency Lists

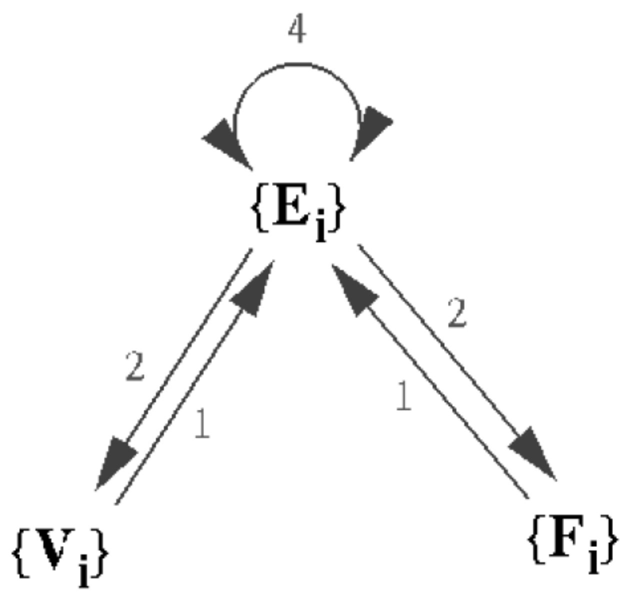
- Can we store only some adjacency relationships and derive others?

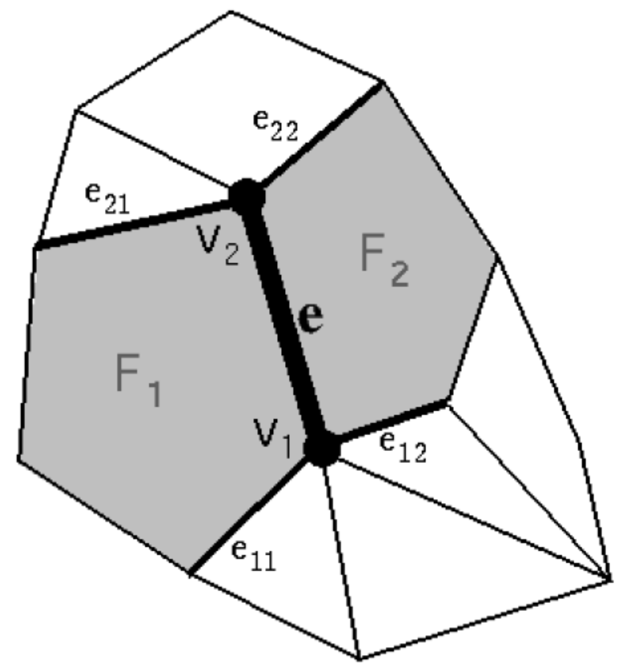




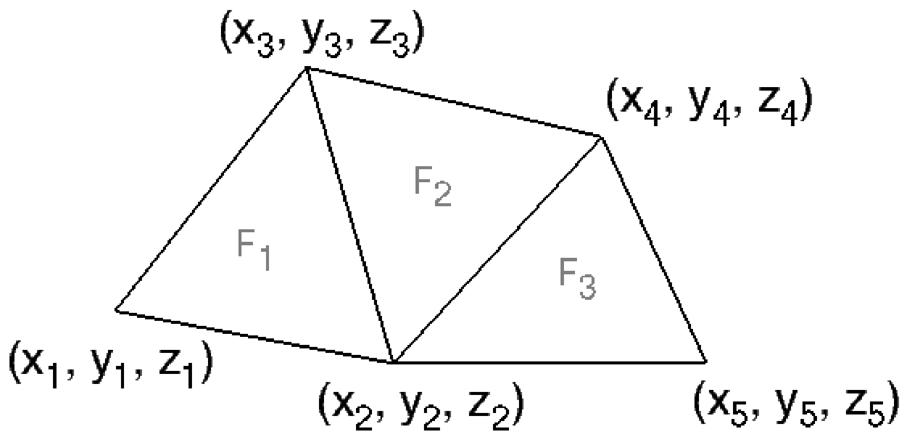
Winged Edge

- Adjacency encoded in edges
  - All adjacencies in O(1) time
- Little extra storage (fixed records)
- Arbitrary polygons





Winged Edge



VERTEX TABLE

V_1	X_1	Y_1	Z_1	e_1
V_2	X_2	Y_2	Z_2	e_6
V_3	X_3	Y_3	Z_3	e_3
V_4	X_4	Y_4	Z_4	e_5
V_5	X_5	Y_5	Z_5	e_6

EDGE TABLE

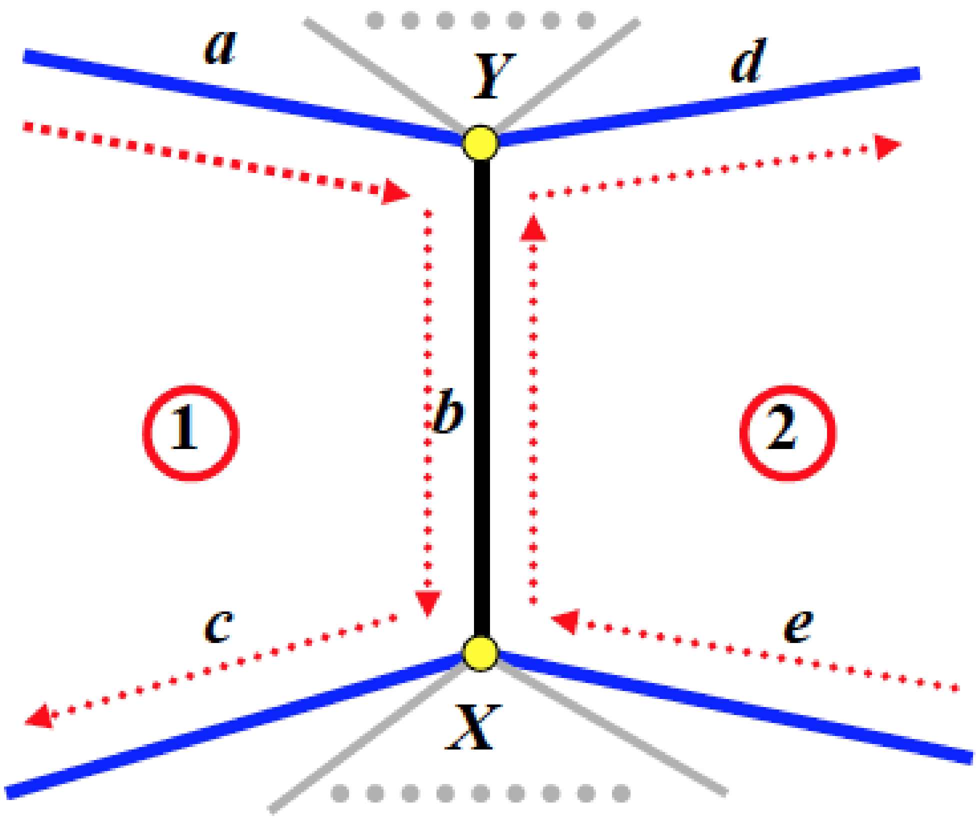
					11	12	21	22
e_1	V_1	V_3		F_1	e_2	e_2	e_4	e_3
e_2	V_1	V_2	F_1		e_1	e_1	e_3	e_6
e_3	V_2	V_3	F_1	F_2	e_2	e_5	e_1	e_4
e_4	$V3$	V_4		F_2	e_1	e_3	e_7	e_5
e_5	V_2	V_4	F_2	F_3	e_3	e_6	e_4	e_7
e_6	V_2	V_5	F_3		e_5	e_2	e_7	e_7
e_7	V_4	V_5		F_3	e_4	e_5	e_6	e_6

FACE TABLE

F_1	e_1
F_2	e_3
F_3	e_5

Winged Edge

If all faces are oriented clock-wise, each edge has *eight* pieces of incident information.

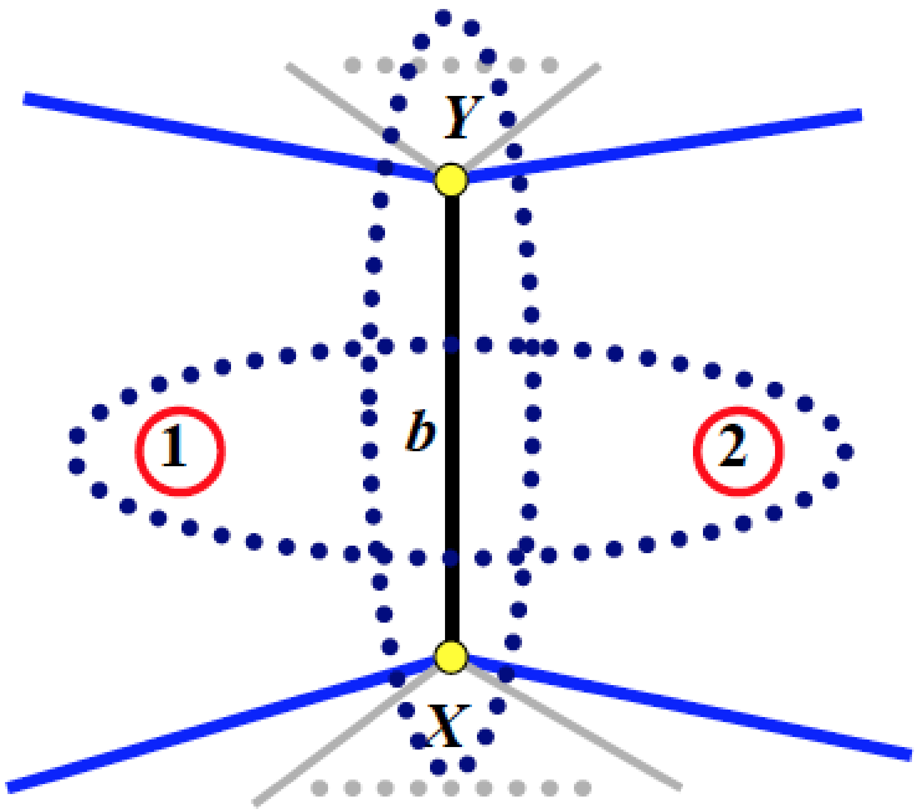


- **Given edge: *b*=*XY***
- **Incident faces: 1 and 2**
- **Pred. & succ. edges of 1**
- **Pred. & succ. edges of 2**

- **The *wings* of edge *b*=*XY* are faces 1 and 2.**
- **Edge *b* is a *winged-edge***

Winged Edge

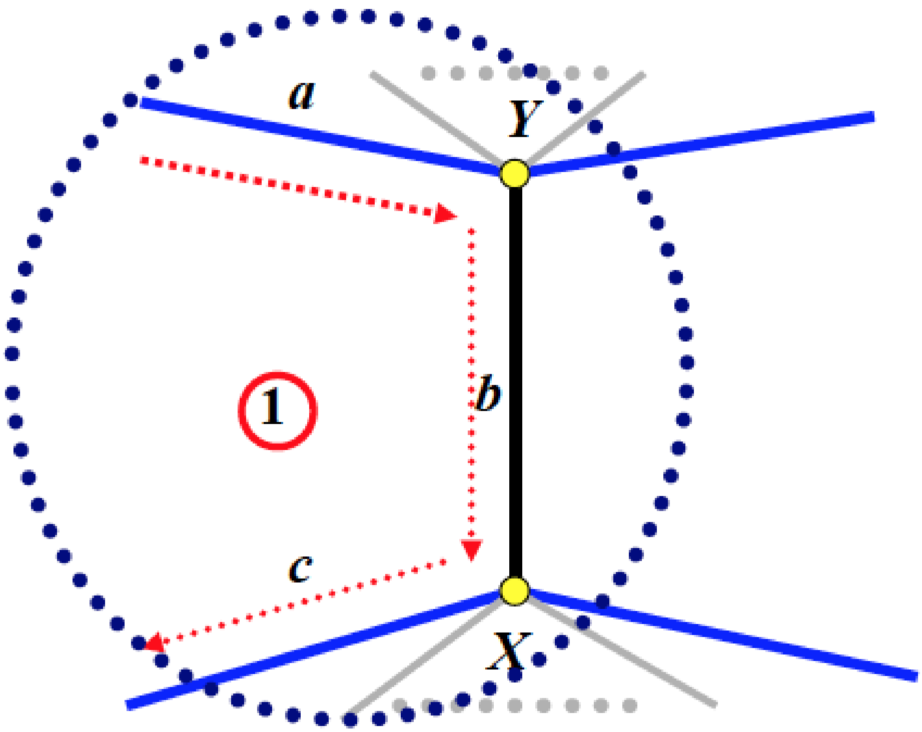
If all faces are oriented clock-wise, each edge has *eight* pieces of incident information.



- **The first *four* pieces:**
  - **The *two* vertices of *b*: *X* and *Y***
  - **The *two* incident faces: 1 and 2**

Winged Edge

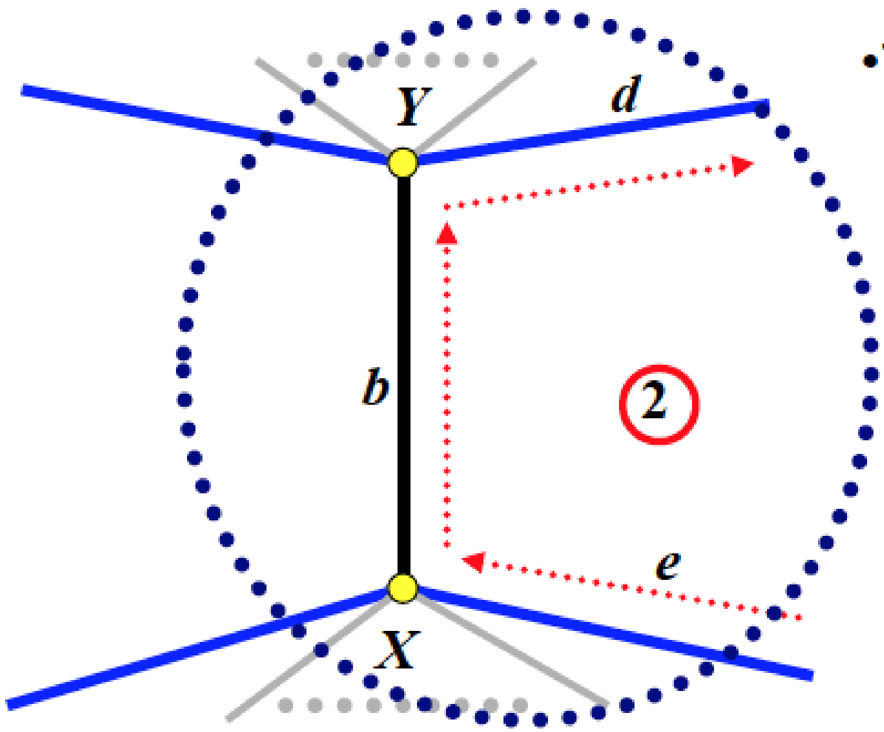
If all faces are oriented clock-wise, each edge has *eight* pieces of incident information.



- **The pred. and succ. edges of *b* with respect to face 1: *a* and *c***

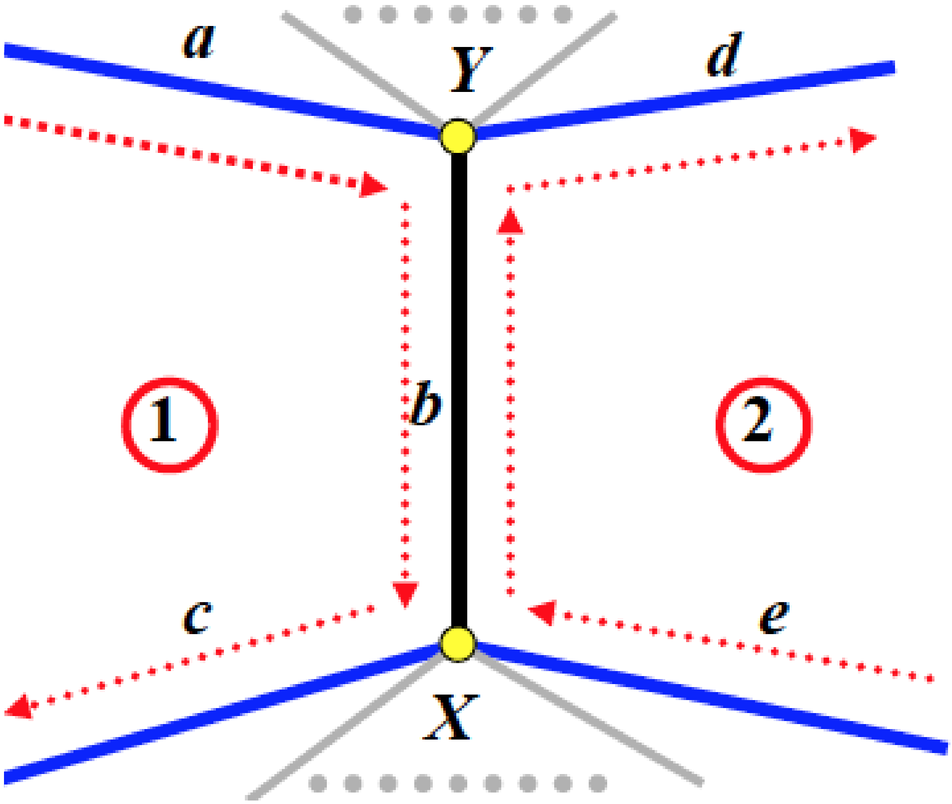
Winged Edge

If all faces are oriented clock-wise, each edge has *eight* pieces of incident information.



- **The pred. and succ. edges of *b* with respect to face 2: *e* and *d***

Left vs. Right Faces

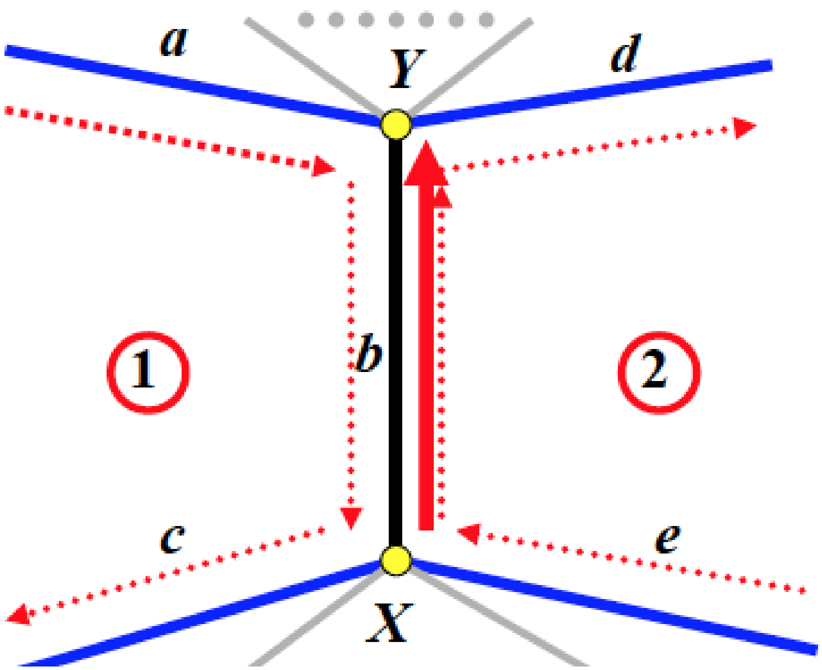


- Which one is left, 1 or 2?
- Choose a direction for edge *b*, say from *X* to *Y* or from *Y* to *X*.
- Going from the start vertex to the end vertex, we know which face is the left one!
- If the start vertex is *X*, the *left* face is face 1.

Winged Edge Info

Information for Edge *b* (from *X* to *Y*)

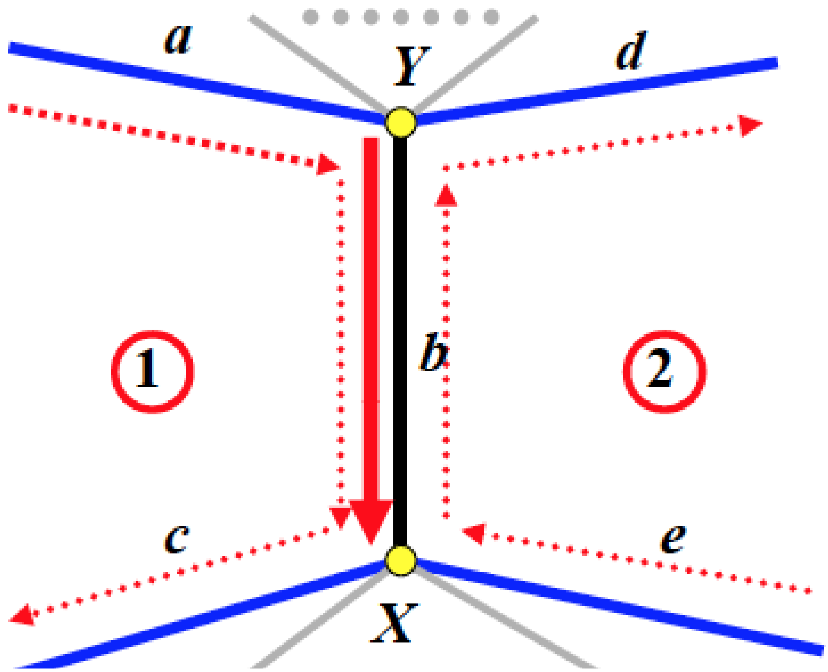
Start Vertex	End Vertex	Left Face	Right Face	Left Pred.	Left Succ.	Right Pred.	Right Succ.
X	*Y*	1	2	*a*	*c*	*e*	*d*



Winged Edge Info

Information for Edge *b* (from *Y* to *X*)

Start Vertex	End Vertex	Left Face	Right Face	Left Pred.	Left Succ.	Right Pred.	Right Succ.
Y	*X*	2	1	*e*	*d*	*a*	*c*

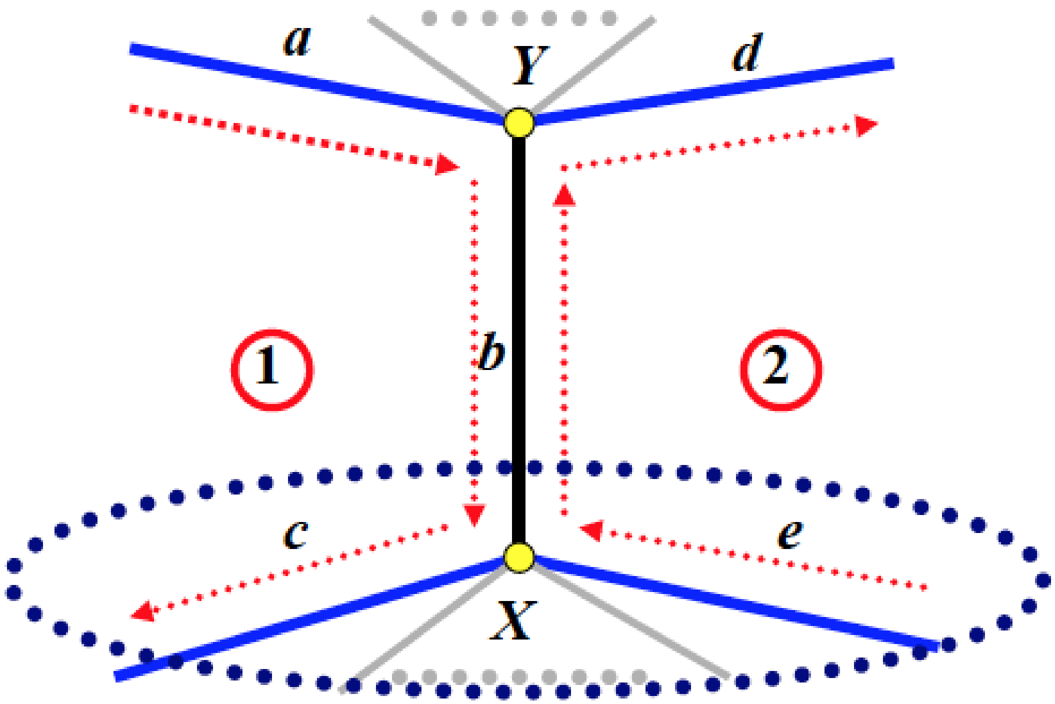


Winged Edged Data Structure

- The winged-edge data structure has three tables: **edge** table, vertex table, and face table.
- Each edge has one row in the edge table.
  - Each row contains the eight pieces of information of that edge.
- Each vertex has one entry in the vertex table.
  - Each entry has a pointer to an incident edge (in the edge table) of that vertex.
- Each face has one entry in the face table.
  - Each entry has a pointer to an incident edge (in the edge table) of that face.

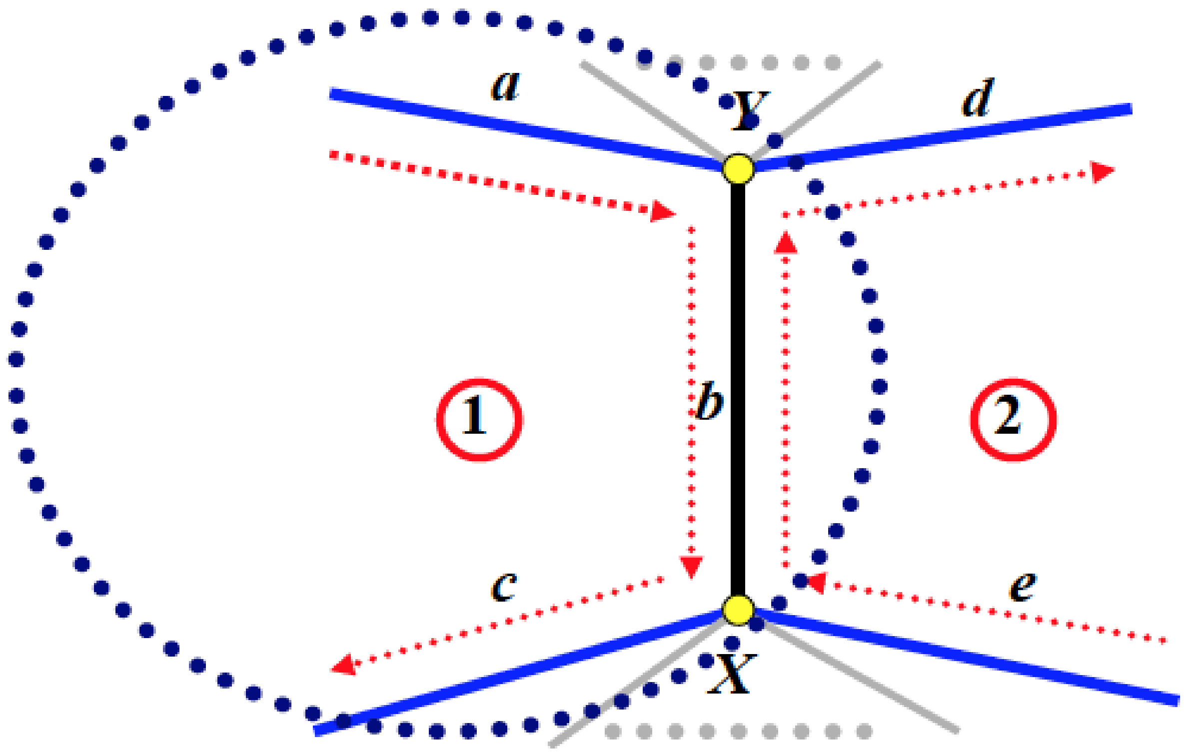
Winged Edged Data Structure

The vertex table entry for vertex *X* may be the edge table entry of edges *c*, *b*, *e*, or any other incident edge.



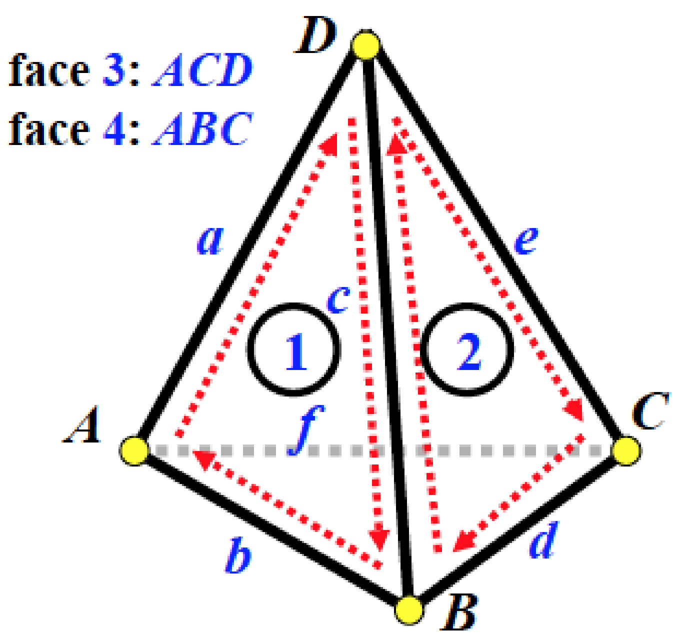
Winged Edged Data Structure

The face table entry for face 1 may be the edge table entry of edges *a*, *b*, *c*, or any other incident edge.



Winged Edged Data Structure

- The following tetrahedron has four vertices A, B, C and D, six edges a, b, c, d, e, f, and four faces 1, 2, 3 and 4.



Vertex Table

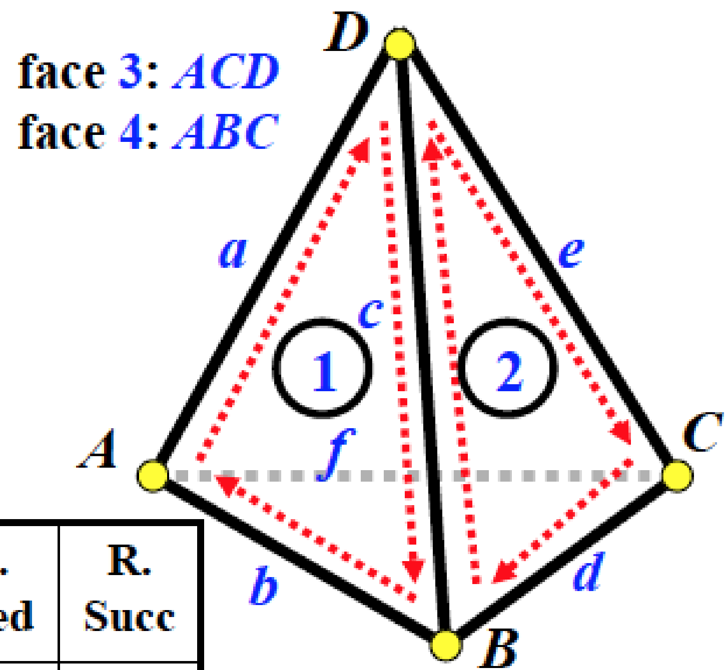
Vertex	Edge
A	*a*
B	*b*
C	*d*
D	*a*

Face Table

Face	Edge
1	*a*
2	*c*
3	*a*
4	*b*

Winged Edged Data Structure

- The following tetrahedron has four vertices A, B, C and D, six edges a, b, c, d, e, f, and four faces 1, 2, 3 and 4.



Edge Table

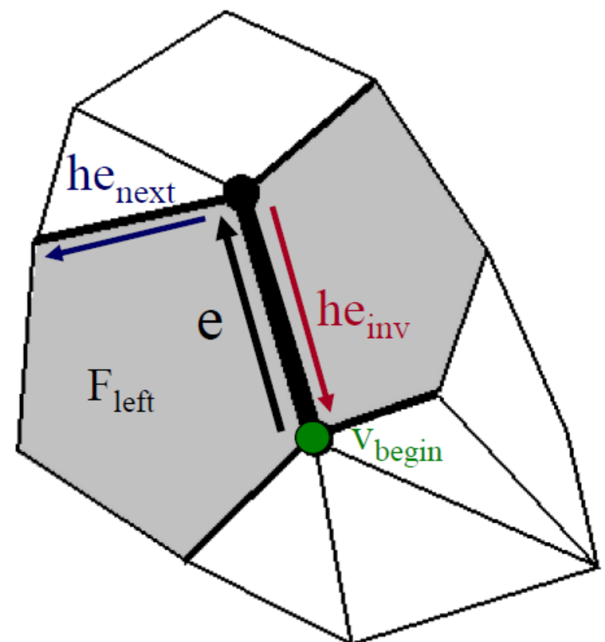
Edge	Start Vtx	End Vtx	L. Face	R. Face	L. Pred	L. Succ	R. Pred	R. Succ
a	*A*	*D*	3	1	*e*	*f*	*b*	*c*
b	*A*	*B*	1	4	*c*	*a*	*f*	*d*
c	*B*	*D*	1	2	*a*	*b*	*d*	*e*
d	*B*	*C*	2	4	*e*	*c*	*b*	*f*
e	*C*	*D*	2	3	*c*	*d*	*f*	*a*
f	*A*	*C*	4	3	*d*	*b*	*a*	*e*

Winged Edged Data Structure

- The winged-edge data structure permits a program to answer many topological inquires very efficiently.
- If (1) V, E and F are the numbers of vertices, edges, and faces and (2) each entry in the table uses one memory unit, the vertex table, edge, table, and face table require V, 8E and F memory units, respectively.

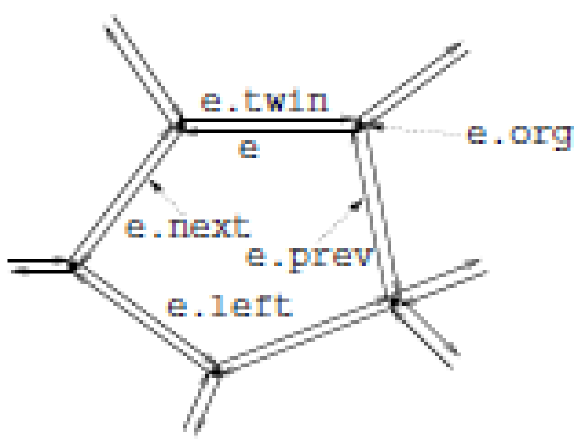
Half Edge

- Adjacency encoded in edges
  - All adjacencies in O(1) time
  - Little extra storage (fixed records)
  - Arbitrary polygons
- Similar to winged-edge, except adjacency encoded in half-edges



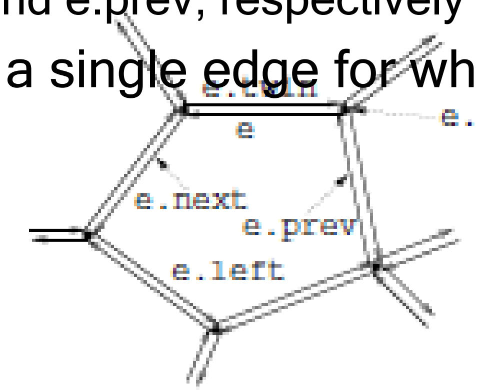
Half Edge

- Each undirected edge represented by two *directed* half edges
  - Unambiguously defines left and right
- Assume that there are no holes in faces



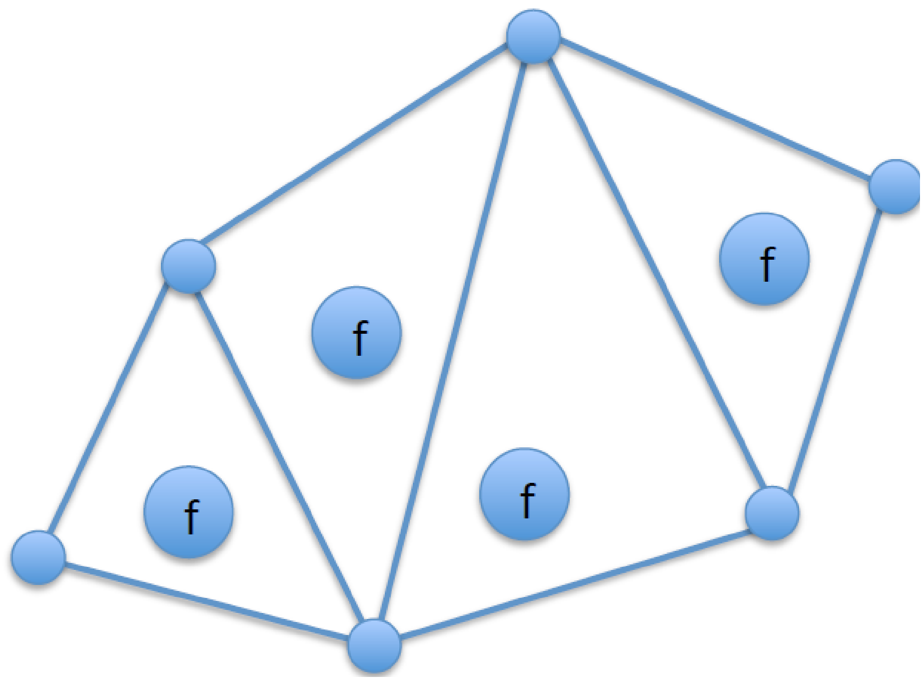
Half Edge

- Each vertex stores:
  - its coordinates
  - a pointer v.inc_edge to any directed edge that has vertex as its origin
- Each directed edge is associated with:
  - a pointer to the oppositely directed edge, called its twin
  - an origin and destination vertex
  - two faces, one to its left and one to its right.
- We only store:
  - a pointer to the origin vertex e.org (e.dest can be accessed as e.twin.org)
  - a pointer to the face to the left of the edge e.left (we can access the face to the right from the twin edge)
  - pointers to the next and previous directed edges in counterclockwise order about the incident face, e.next and e.prev, respectively
- Each face f stores a pointer to a single edge for which this face is the incident face, f.inc_edge



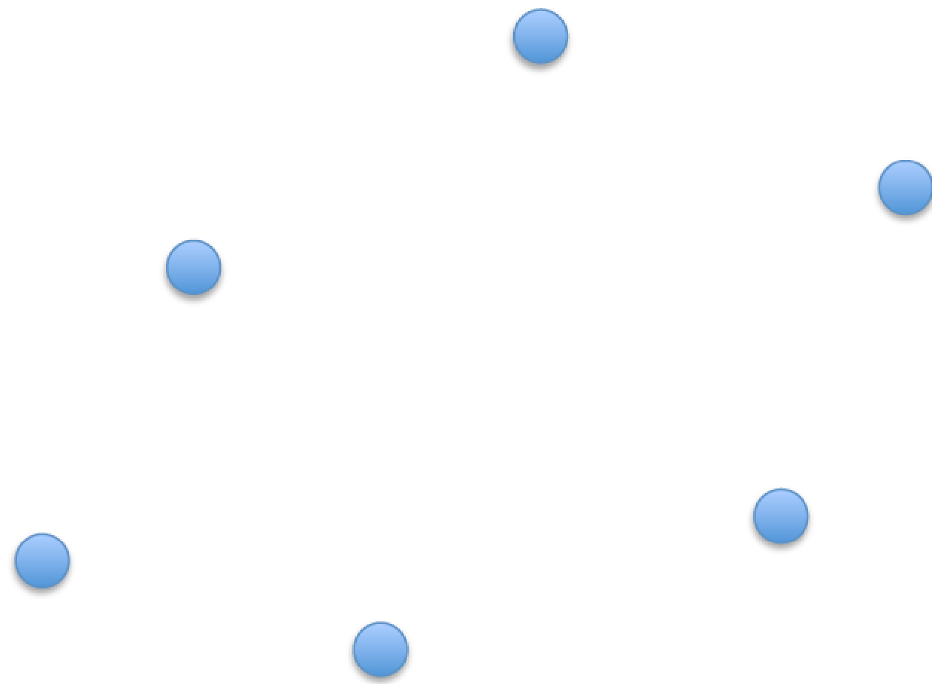
How to Load a Shape

- From file with vertices and triangles



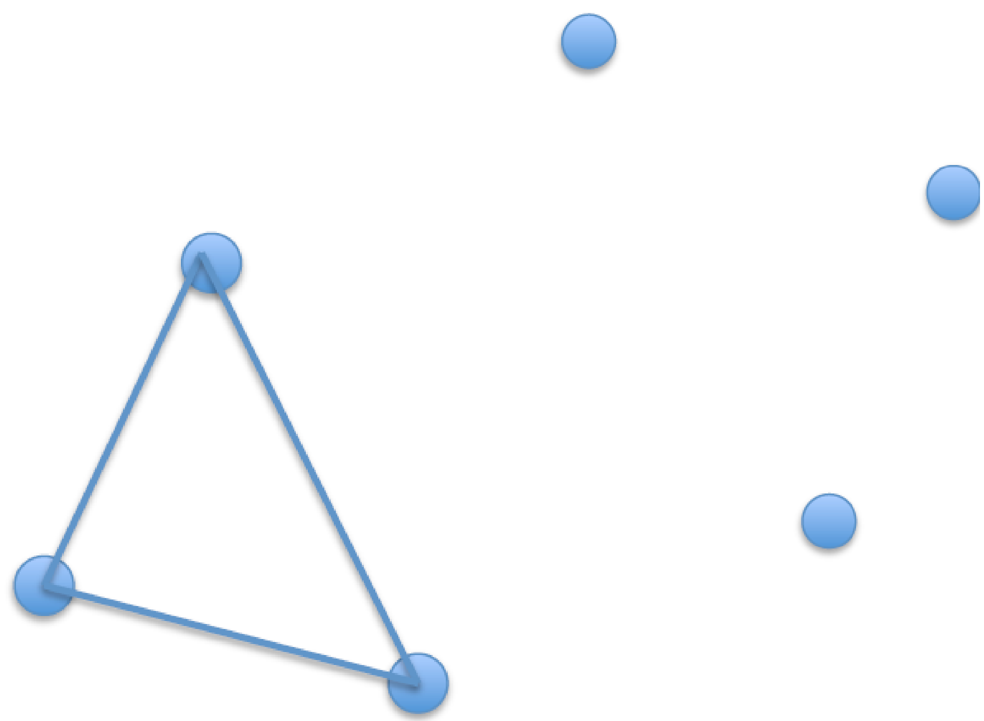
How to Load a Shape

- Add vertex coordinates to list



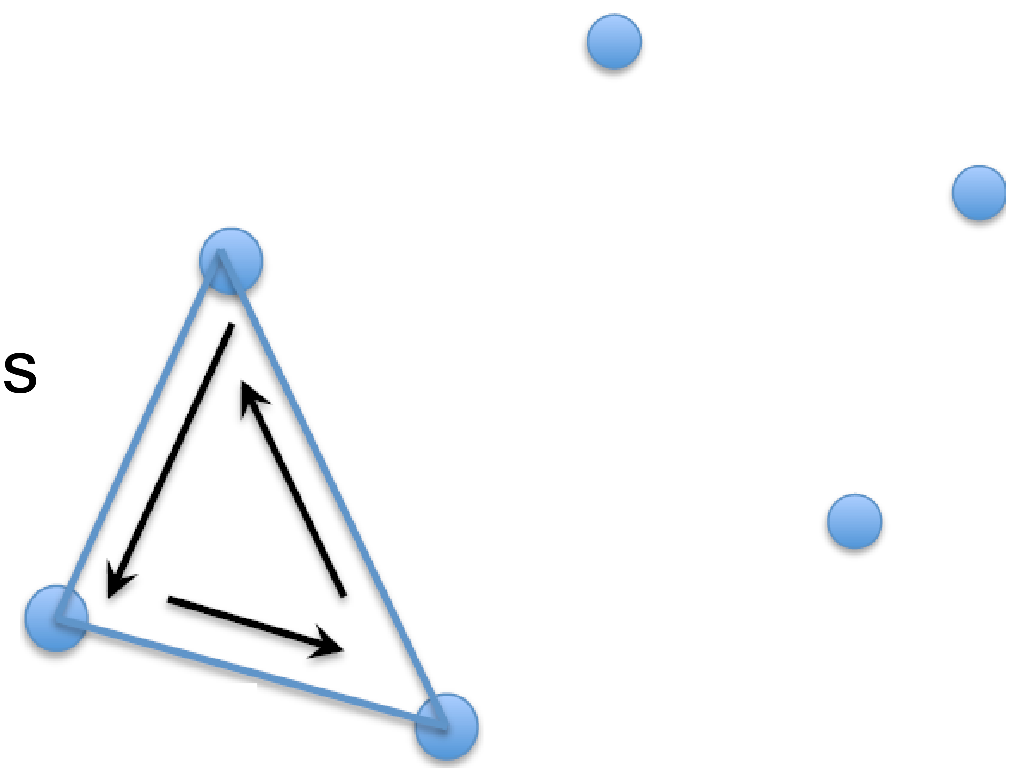
How to Load a Shape

- Add vertex coordinates to list
- Add half-edges with faces



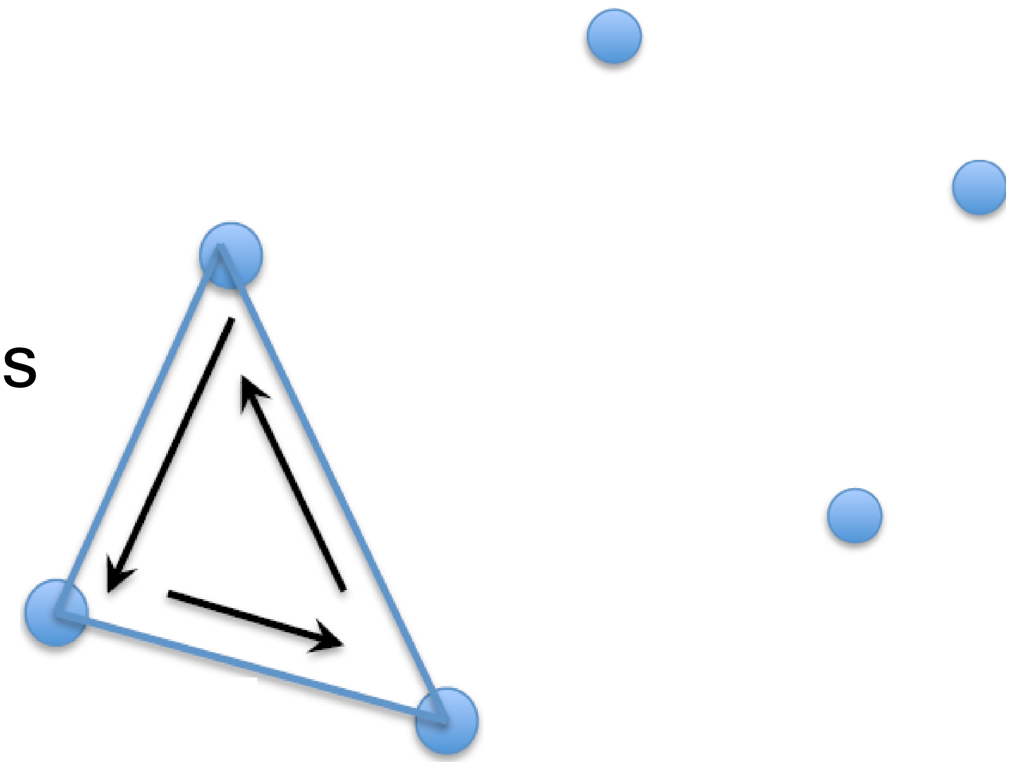
How to Load a Shape

- Add vertex coordinates to list
- Add half-edges with faces
  - Inner half-edges are sufficient



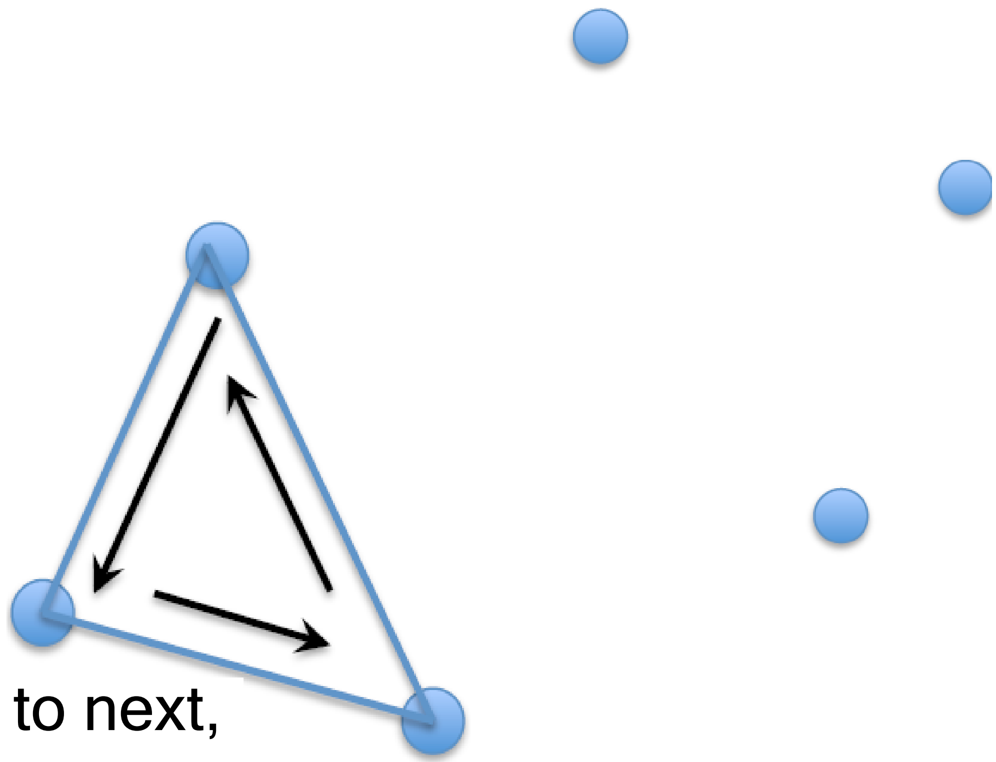
How to Load a Shape

- Add vertex coordinates to list
- Add half-edges with faces
  - Inner half-edges are sufficient
  - Update vertex pointers to half-edges



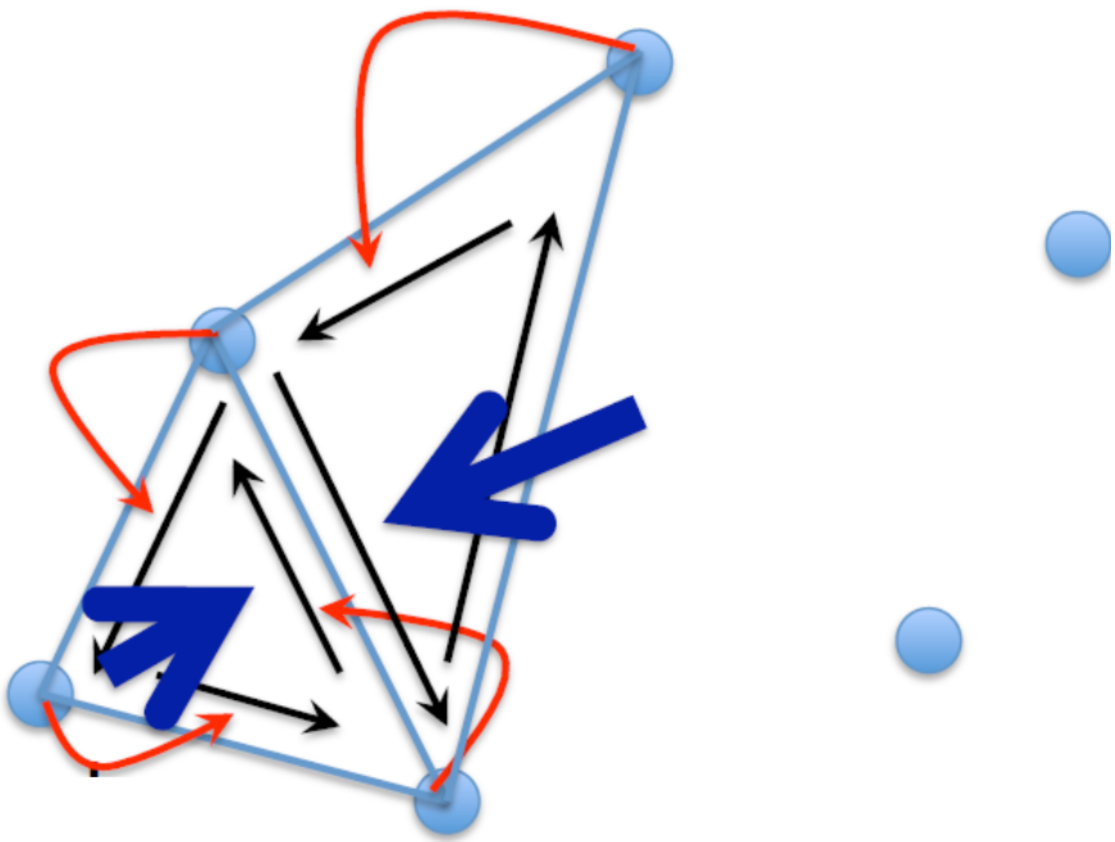
How to Load a Shape

- Add vertex coordinates to list
- Add half-edges with faces
  - Inner half-edges are sufficient
  - Update vertex pointers to half-edges
  - Half-edges: pointer to next, pointer to face
  - Faces: pointer to one of the inner half-edges



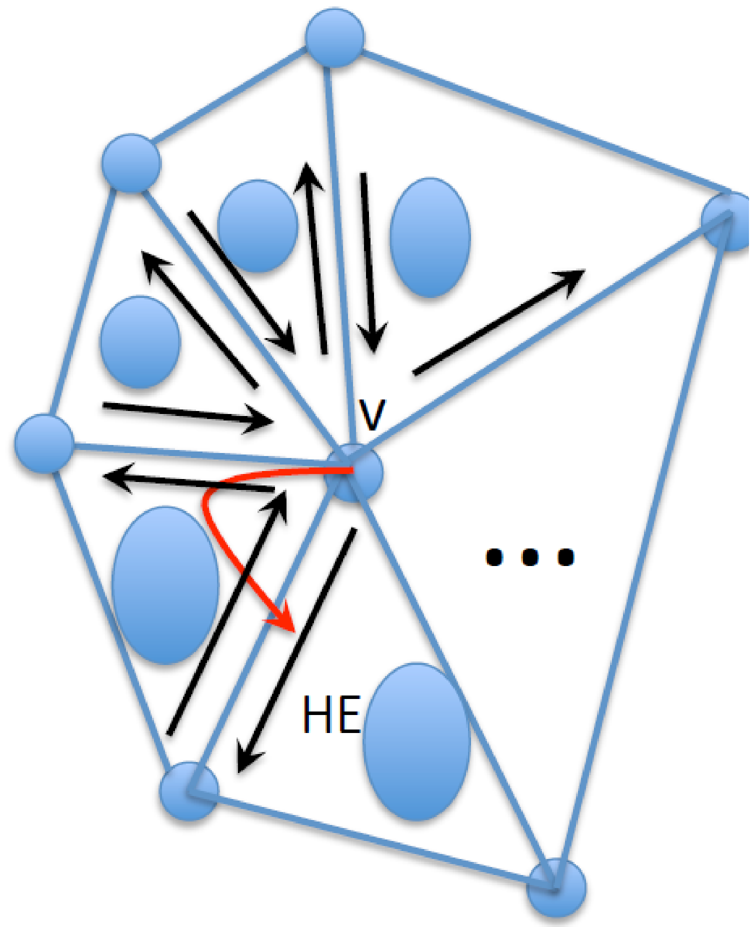
How to Load a Shape

- Continue adding incrementally



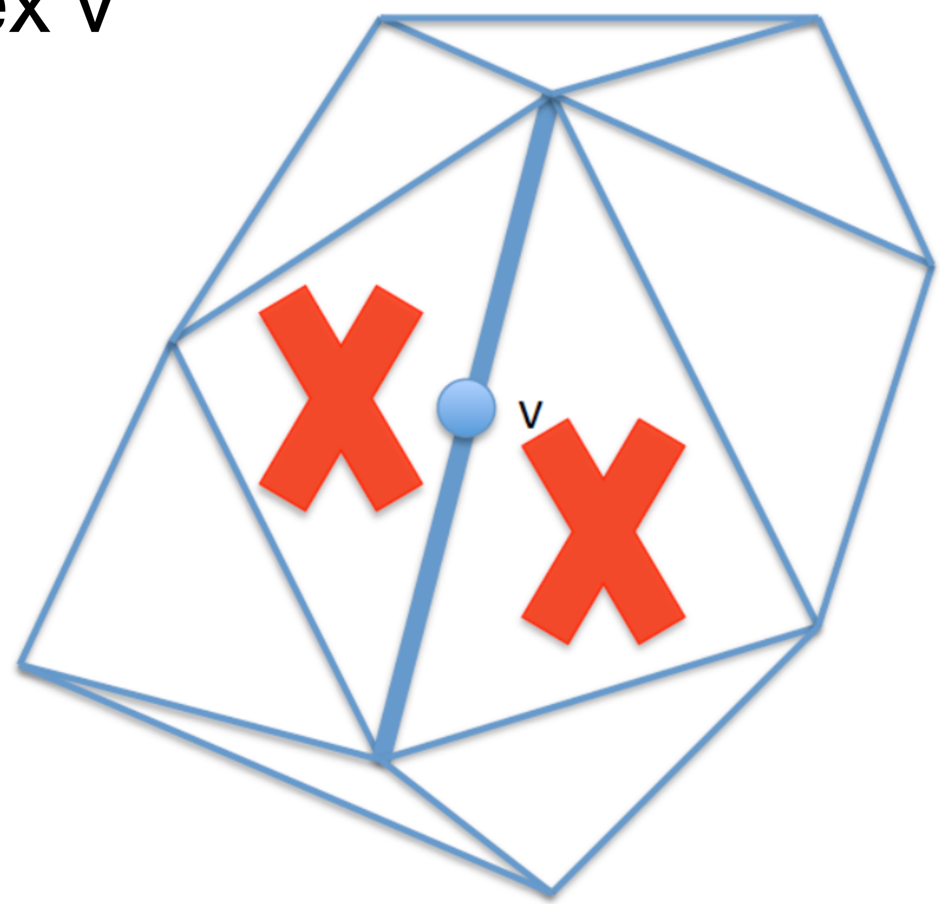
Finding Adjacent Faces

- Check all outgoing half edges
  - V points to a half edge HE
  - ADD_FACE(HE)
  - Iterate:
    - X=HE.twin
    - Y=X.next
    - ADD_FACE(Y)
    - HE:=Y



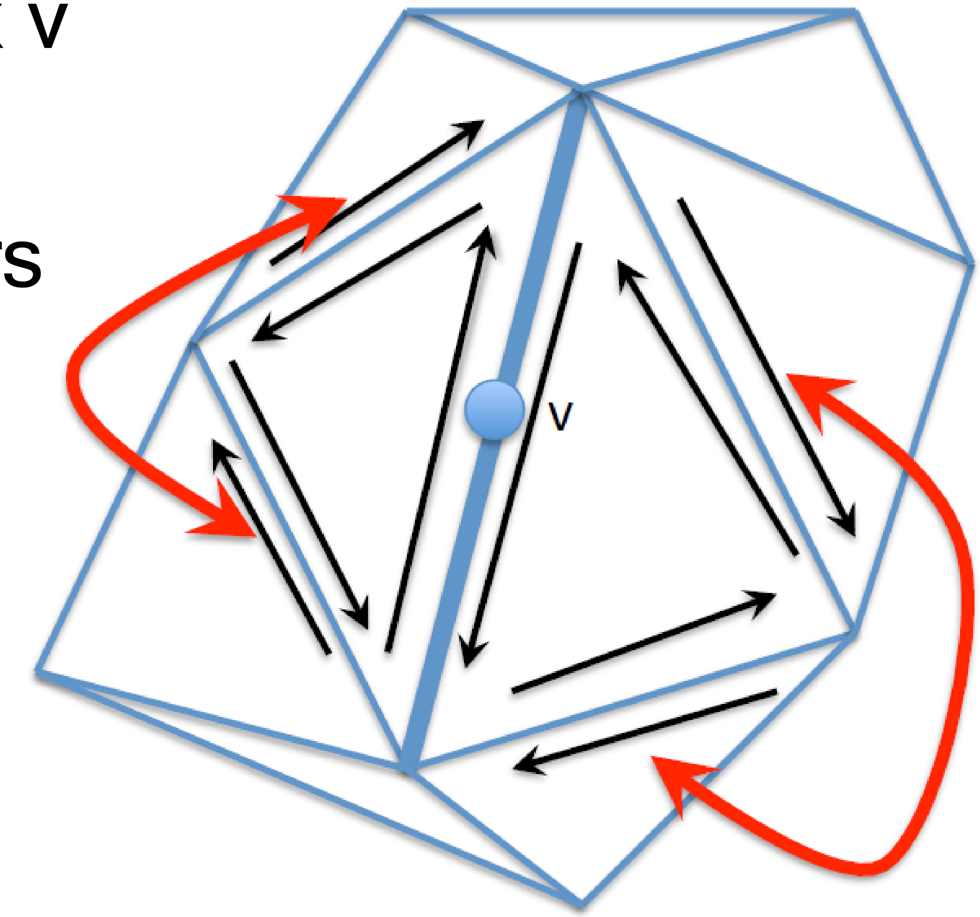
Collapsing an Edge

- Create a new vertex v
- Remove faces



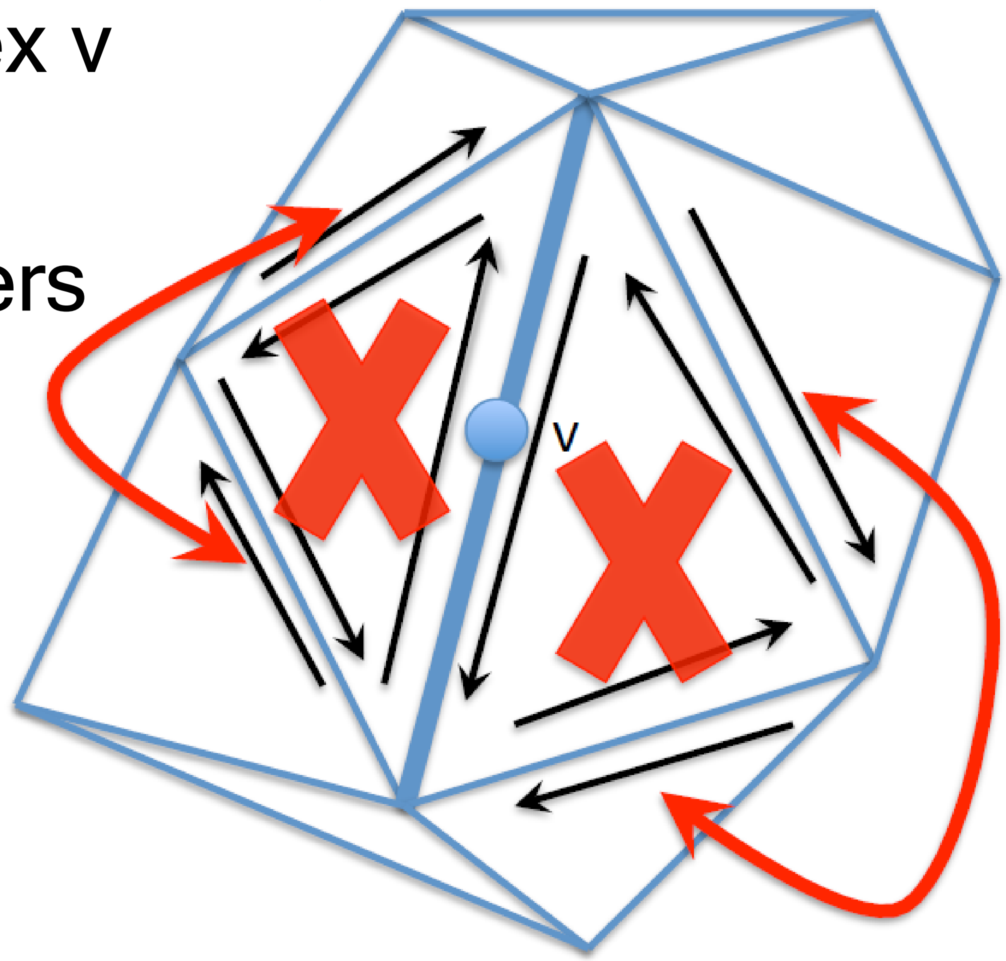
Collapsing an Edge

- Create a new vertex v
- Remove faces
- Change twin pointers



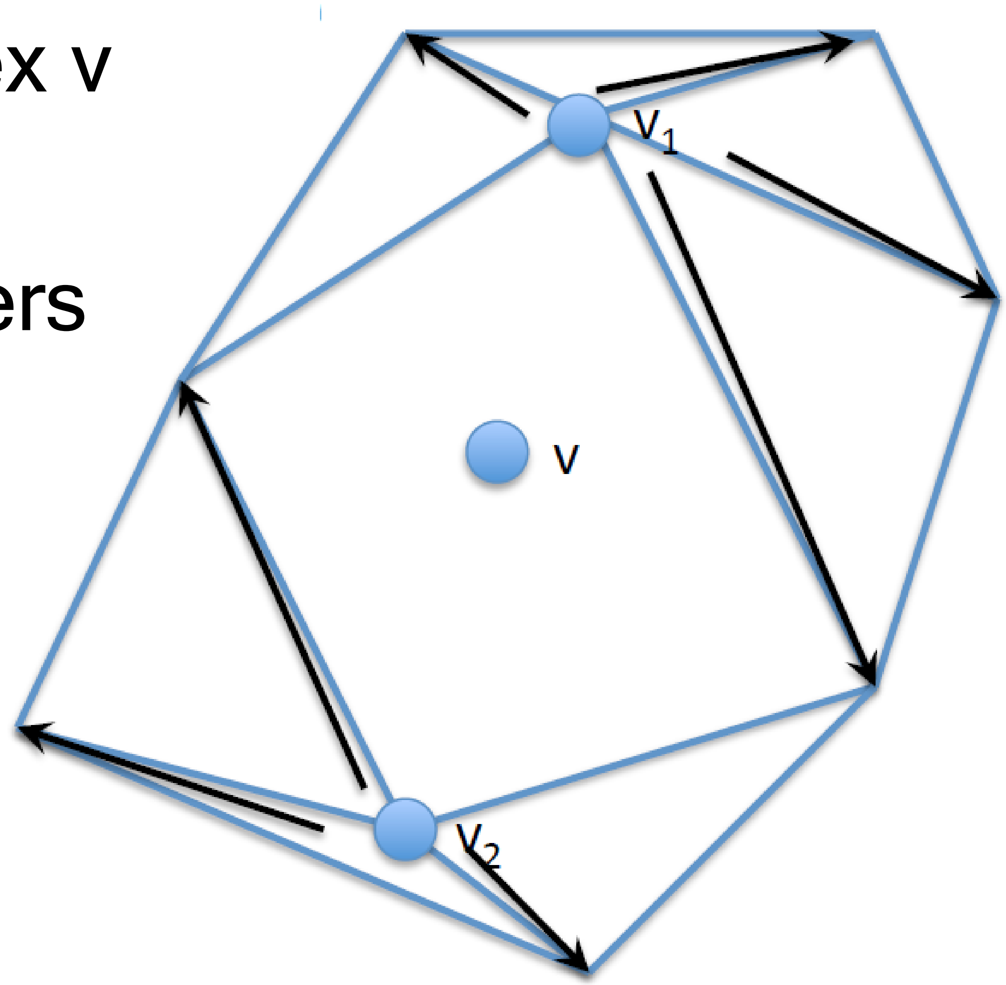
Collapsing an Edge

- Create a new vertex v
- Remove faces
- Change twin pointers
- Remove edges



Collapsing an Edge

- Create a new vertex v
- Remove faces
- Change twin pointers
- Remove edges
- Change pointers from half-edges to v_1 and v_2



Collapsing an Edge

- Create a new vertex v
- Remove faces
- Change twin pointers
- Remove edges
- Change pointers from half-edges to v_1 and v_2
- Remove v_1 and v_2
- Pick an outgoing edge for v