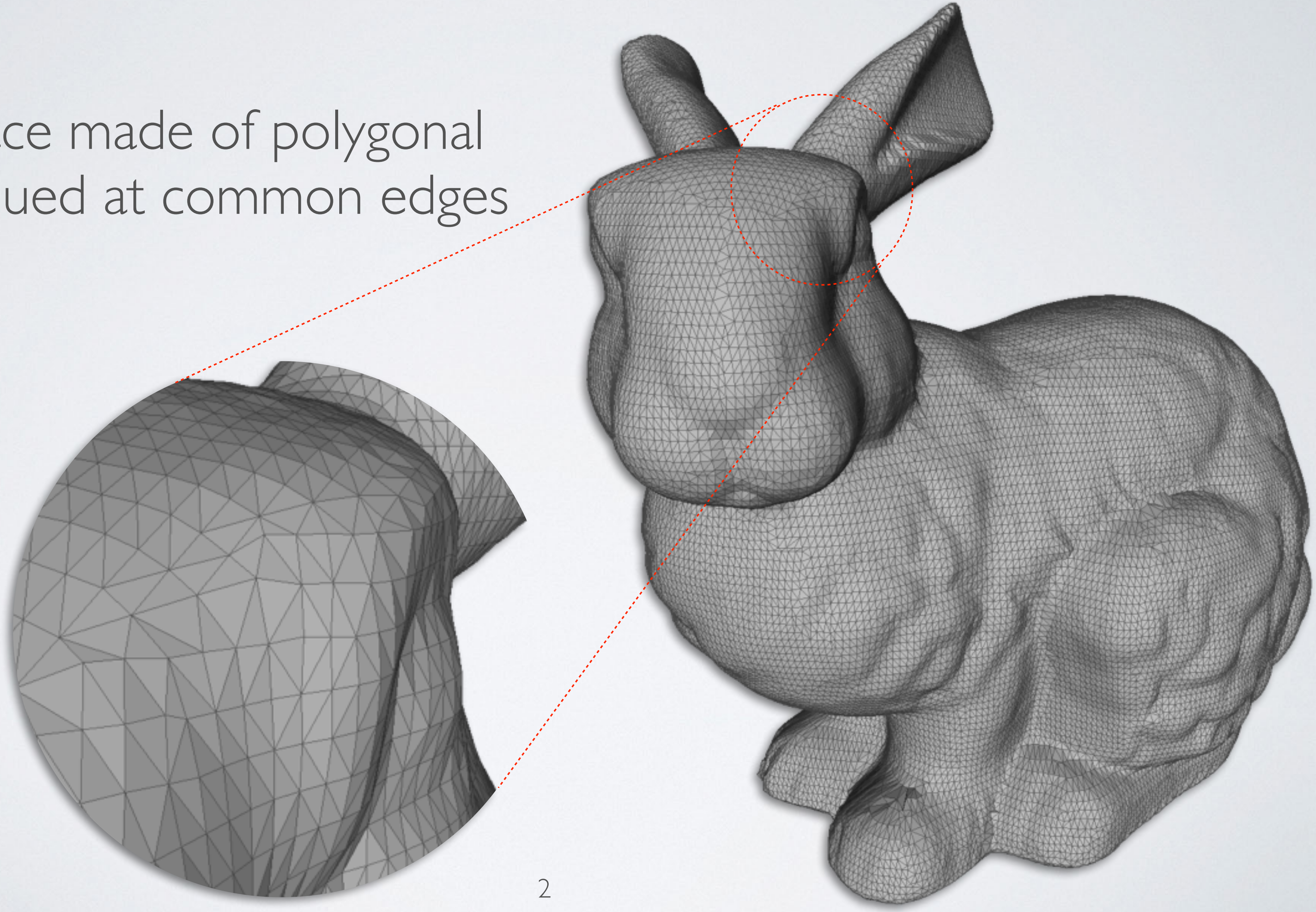
POLYGONAL MESHES

Lecture 3

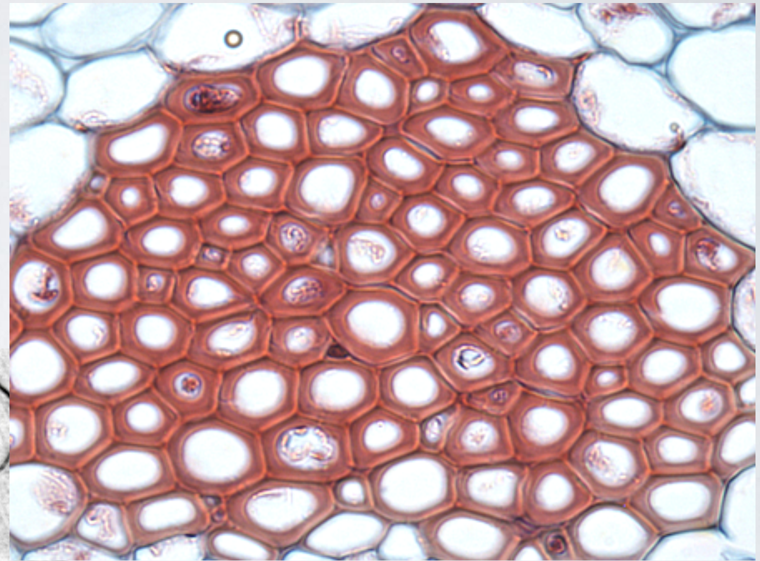
WHAT IS A MESH?

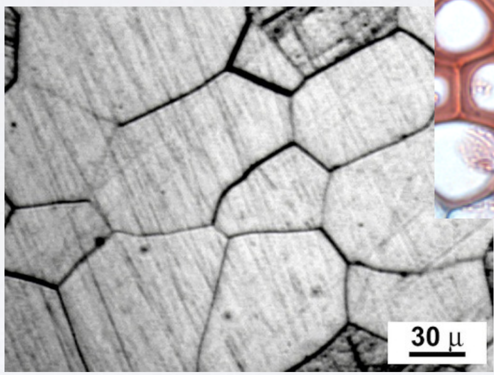
A surface made of polygonal faces glued at common edges

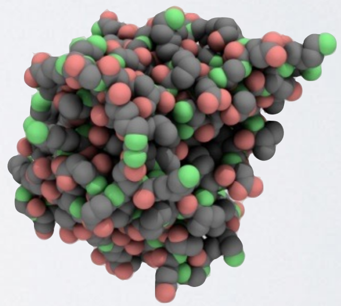


ORIGIN OF MESHES

- In nature, meshes arise in a variety of contexts:
  - –Cells in organic tissues
  - –Crystals
  - –Molecules
  - –…
  - –Mostly ***convex*** but ***irregular*** cells
  - –Common concept: ***complex*** shapes can be described as ***collections*** of ***simple building blocks***

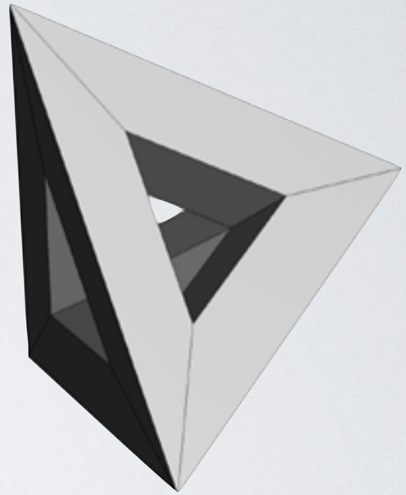






ORIGIN OF MESHES

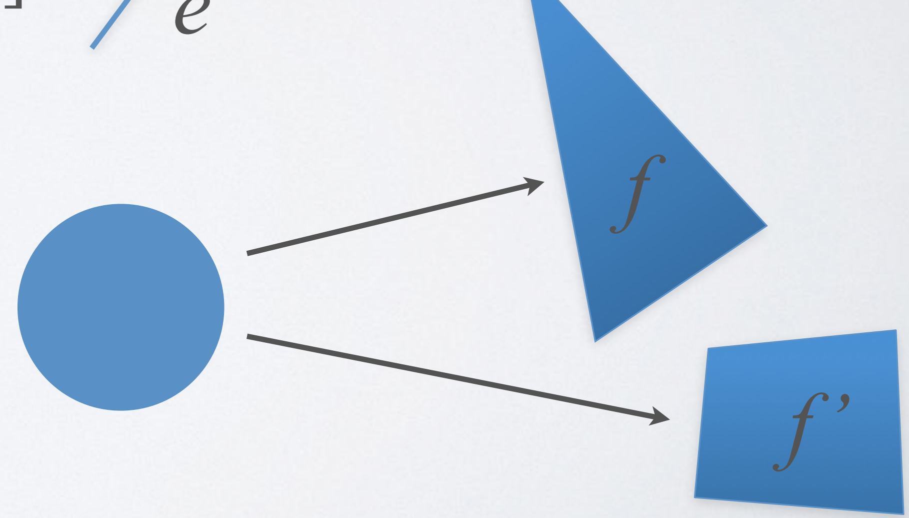
- In math, meshes come from algebraic topology:
  - A manifold is decomposed into a collection of simple ***cells*** (each homeomorphic to a disc)
  - Properties of the manifold are studied by analyzing the structure of the resulting ***cell complex***
  - Both simplicial and hypercubic complexes may be used (for surfaces: tris and quads)



BASIC MATH OF MESHES

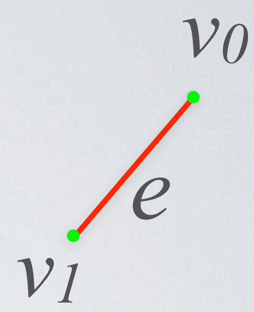
- A ***n*-cell** is a set homeomorphic to a Euclidean disc of dimension n:
  - 0-cell: ***vertex*** v
  - 1-cell: ***edge*** $[0,1] \rightarrow$ 

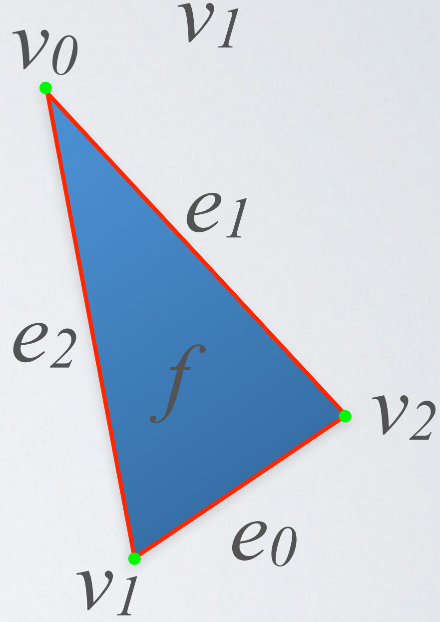
  - 2-cell: ***face***

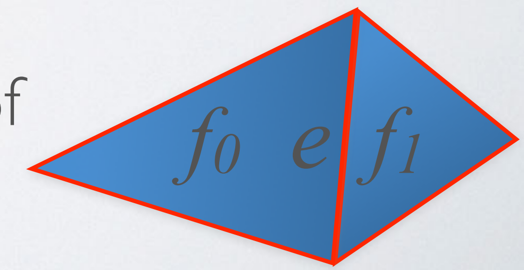


STRUCTURE OF MESHES

- A ***mesh*** $M=(V,E,F)$ of dimension 2 is made of a collection of k-cells for $k = 0, 1, 2$:
    - 0-cells of V lie on the boundary of 1-cells of E
    - 1-cells of E lies on the boundary of 2-cells of F
        - **(manifoldness)** each 1-cell of E lies on the boundary of either one or two 2-cells of F
    - the intersection of two distinct 1- / 2-cells is either empty or it coincides with a collection of 0- / 1-cells







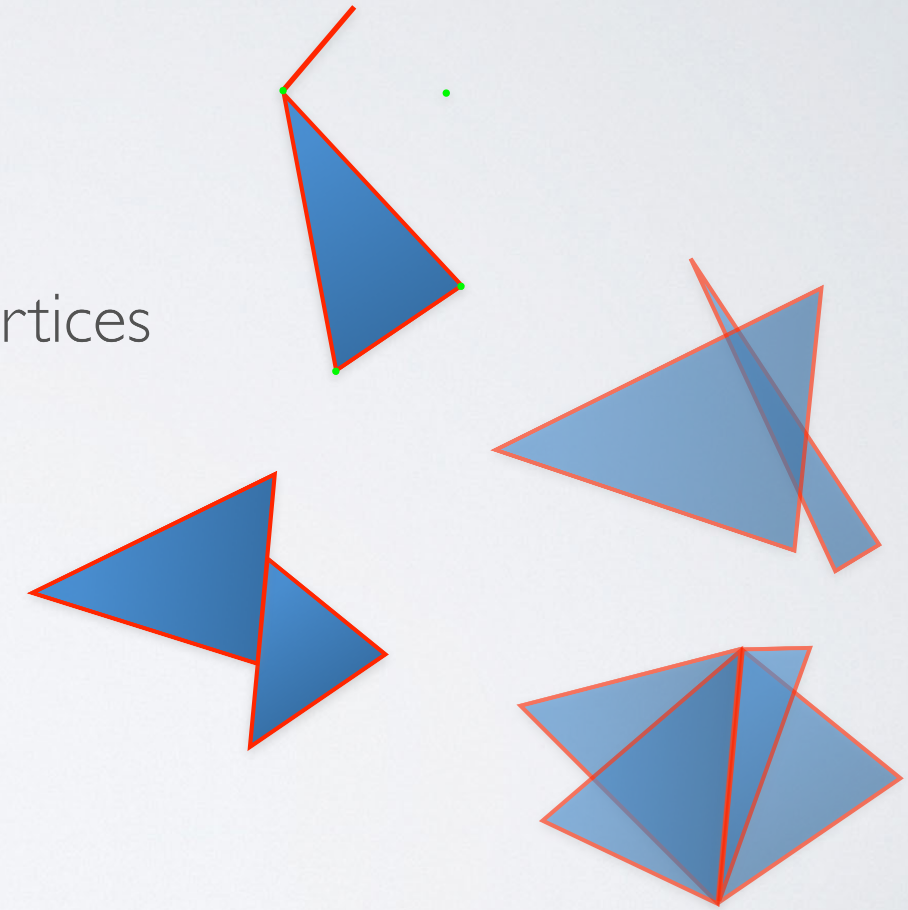
STRUCTURE OF MESHES

- Properties 1. and 2. guarantee that there are no “dangling” edges and isolated vertices
- Property 3. guarantees that faces abut properly
- Property 2.1 extends property 2. to guarantee that the ***carrier*** of the mesh (i.e., the union of all its cells) is a manifold (i.e., a surface)

STRUCTURE OF MESHES

Forbidden configurations:

- Dangling edges and isolated vertices
- Intersecting faces
- Non-conforming adjacency
- Non-manifold edges



STRUCTURE OF MESHES

- A mesh can be treated as a purely combinatorial structure

$$M = (V, E, F)$$

- For some applications, geometry of edges and faces it not relevant. Just encode:
  - vertices as singletons (V)
  - how vertices are connected among them (E)
  - how cycles of vertices bound faces (F)

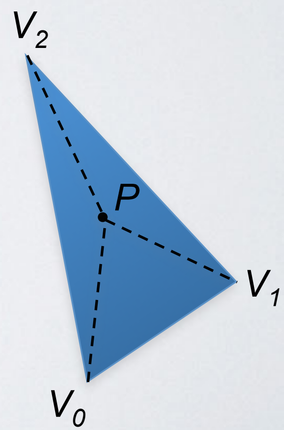
STRUCTURE OF MESHES

- Geometric embedding:
  - position in space for each 0-cell (vertex - point)
  - geometry for each 1-cell (edge - line) and 2-cell (face - disk-like surface)
- Polygonal meshes are embedded:
  - edges are straight-line segments
  - are faces flat? not always true: vertices of a face might be ***not coplanar***

TRIANGLE MESHES

- A *triangle mesh* is a polygonal mesh with all triangular faces
  - all faces are flat (there exist a unique plane for three points)
- All cells are *simplices*, i.e., they are the convex combinations of their vertices

$$P = \lambda_0 V_0 + \lambda_1 V_1 + \lambda_2 V_2 \qquad \lambda_i \in [0,1] \qquad \lambda_0 + \lambda_1 + \lambda_2 = 1$$



  - embedding of vertices + combinatorial structure characterize the embedding of the whole mesh

EULER-POINCARÉ FORMULA

- Relates the number of cells in a mesh with the genus of the surface it represents:

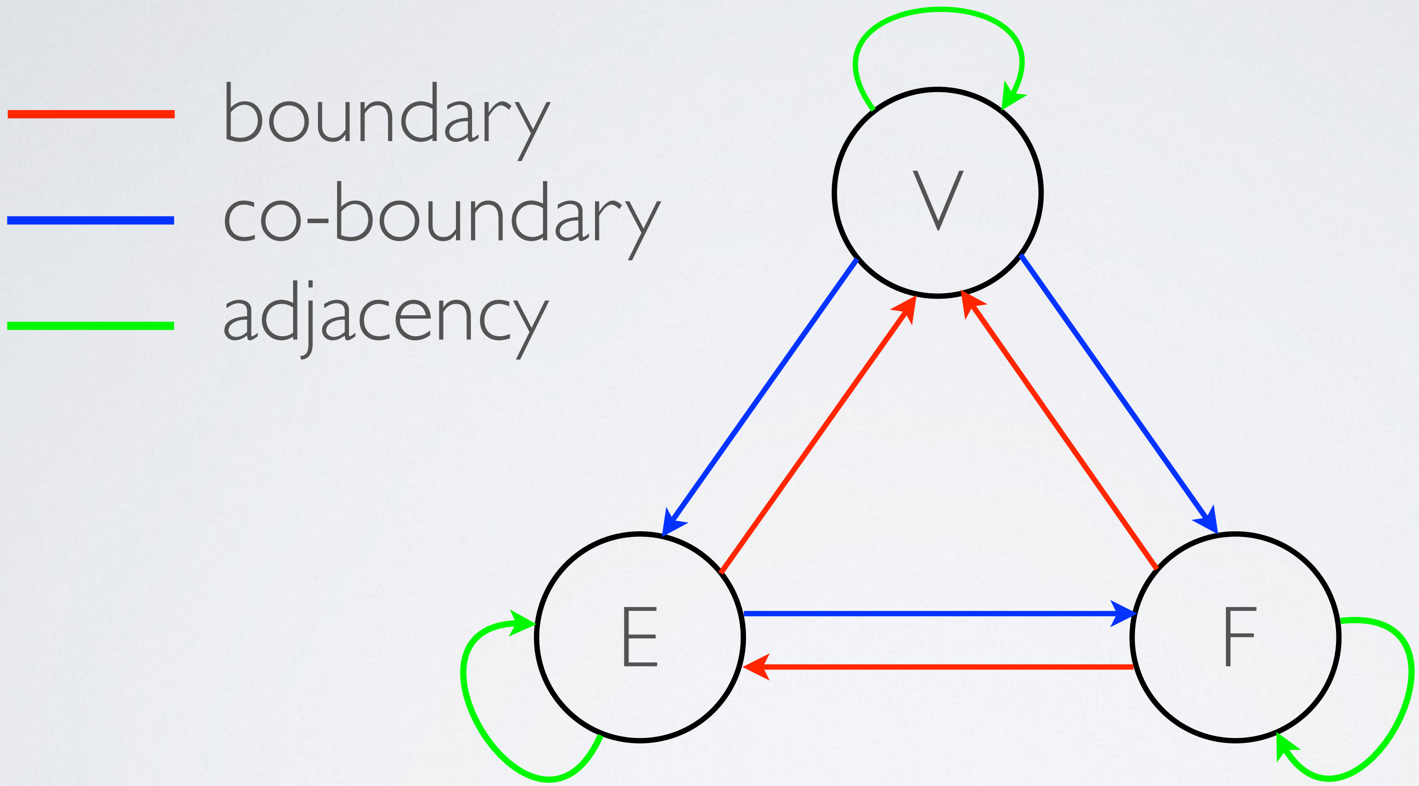
$$v - e + f = 2 - 2g - h$$

- Genus $g =$ # handles (ex.: sphere: genus 0; torus: genus 1)
- Holes $h =$ # boundary loops (watertight: no boundary)

EULER-POINCARÉ FORMULA

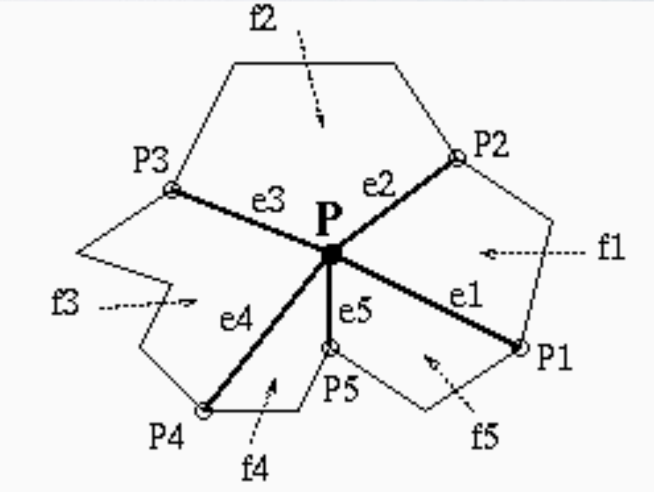
- In a (watertight manifold) triangle mesh:
  - each face has three edges, each edge is shared by two faces:
    - $2e = 3f$
    - $e = 3v + 6g - 6 \approx 3v$
    - $f = 2v + 4g - 4 \approx 2v$
- The formula can be adapted to bordered surfaces to take into account boundary loops and edges

TOPOLOGICAL RELATIONS



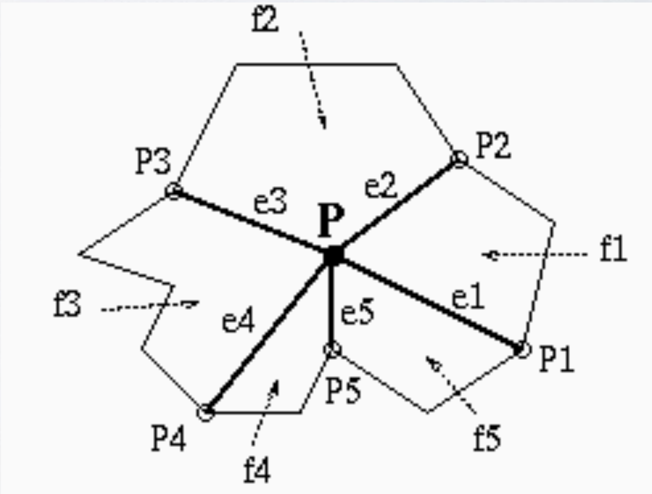
VERTEX-BASED RELATIONS

- VE (Vertex-Edge):
  - for each vertex v, the list of edges (e_1,e_2,...,e_r) having an endpoint in v (*incident edges*) arranged in counter-clockwise radial order around v
  - list is circular: initial vertex e_1 is arbitrarily chosen



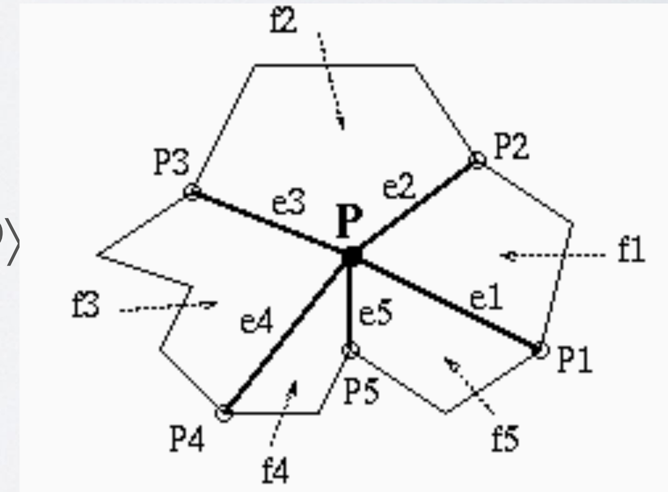
VERTEX-BASED RELATIONS

- VV (Vertex-Vertex):
  - for each vertex v, the list of vertices $(v_1, v_2, \ldots, v_r)$ connected to v with an edge (*adjacent vertices*) arranged in counter-clockwise radial order around v
  - consistency rule: vertex v_i in VV(v) is an endpoint of edge e_i in VE(v)



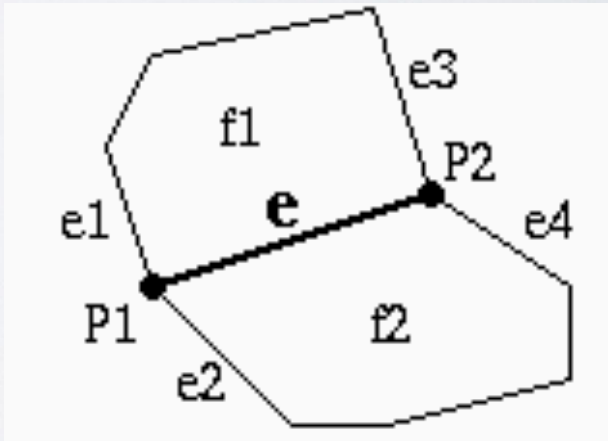
VERTEX-BASED RELATIONS

- VF (Vertex-Face):
  - for each vertex v, the list of faces $(f_1,f_2,\dots,f_r)$ having v on their boundary (*incident faces*) arranged in counter-clockwise radial order around v
  - consistency rule: face f_i in VF(v) is bounded by edges e_i and e_{i+1} in VE(v)



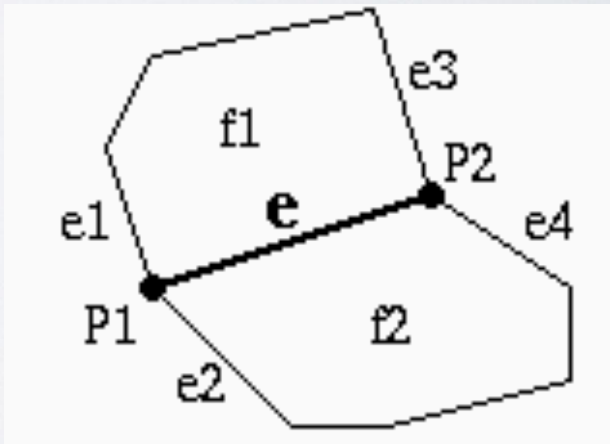
EDGE-BASED RELATIONS

- EV (Edge-Vertex):
  - for each edge e, the two endpoints (v_1,v_2) of e (*incident vertices*)
- EF (Edge-Face):
  - for each edge e, the two faces (f_1,f_2) having e on their boundary (*incident faces*)
- Consistency rule: face f_1 [f_2] is on the left [right] of the oriented line from v_1 to v_2



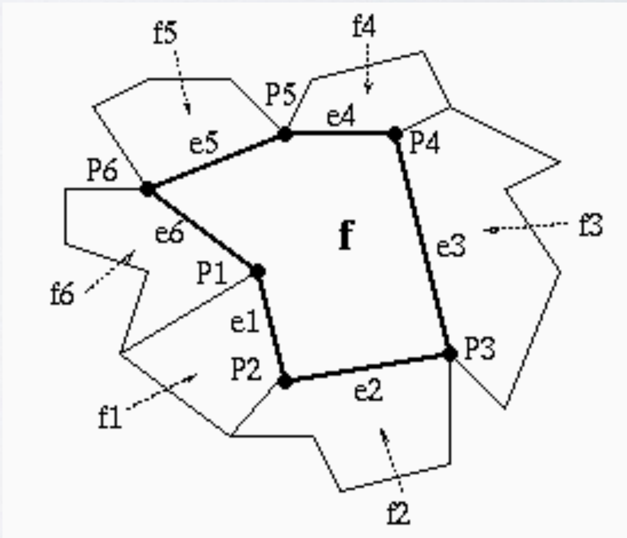
EDGE-BASED RELATIONS

- EE (Edge-Edge):
    - for each edge e, two pairs of edges ((e_1,e_2), (e_3,e_4)) that share a vertex and a face with e (*adjacent edges*)
- Consistency rule:
    - e_1 is incident on v_1 and f_1
    - e_2 is incident on v_1 and f_2
    - e_3 is incident on v_2 and f_1
    - e_4 is incident on v_2 and f_2



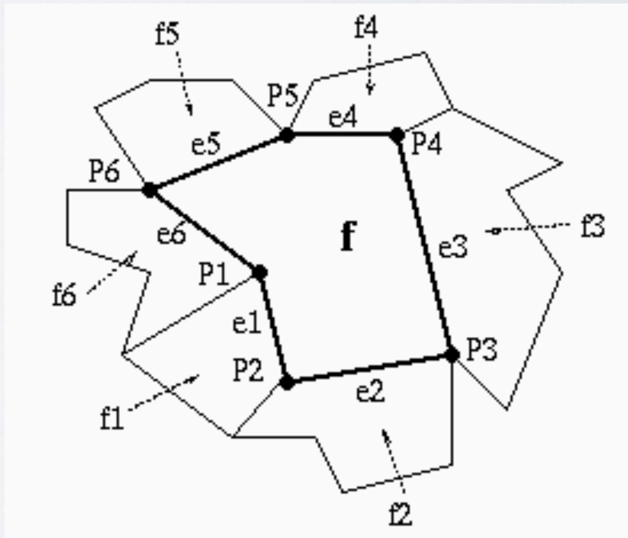
FACE-BASED RELATIONS

- FE (Face-Edge):
  - for each face f, the list (e_1,e_2,...,e_m) of edges of its boundary (*incident edges*), in counter-clockwise order about f
  - list is circular: initial vertex e_1 is arbitrarily chosen



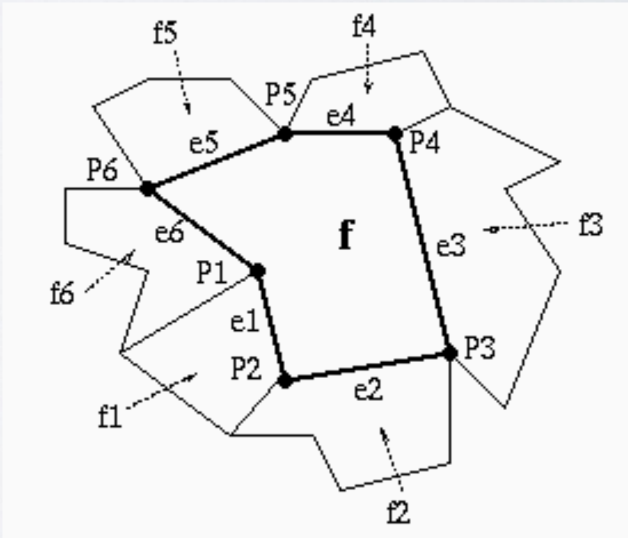
FACE-BASED RELATIONS

- FV (Face-Vertex):
  - for each face f, the list $(v_1,v_2,...,v_m)$ of vertices of its boundary (*incident vertices*), in counter-clockwise order about f
  - consistency rule: edge e_i in FE(f) has endpoints v_i and v_{i+1}



FACE-BASED RELATIONS

- FF (Face-Face):
  - for each face f, the list $(f_1,f_2,\dots,f_m)$ of faces that share an edge with f (*adjacent faces*), in counter-clockwise order about f
  - consistency rule: face f_i in FF(f) shares edge e_i in FE(f)

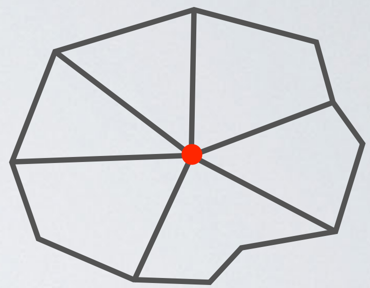


TOPOLOGICAL RELATIONS

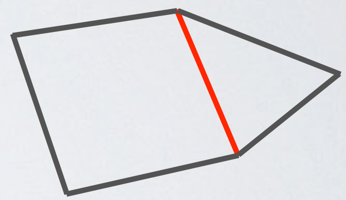
- **Constant relations** return a constant number of elements:
  - EV (each edge has two endpoints)
  - EE (each edge has four adjacent edges)
  - EF (each edge has two incident faces)
- **Variable relations** return a variable number of elements:
  - VV, VE, VF, FV, FE, FF: the number of vertices/edges/faces incident/ adjacent to a given vertex/face is not constant and it can be of the same order of the total number of vertices/edges/faces

STARS AND RINGS

- The **star** of a vertex v is formed by v plus the set of cells incident at v (edges and faces of its co-boundary)

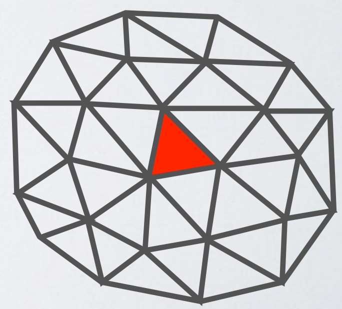


- The **star** of an edge e is formed by e plus the set of faces incident at e (faces of its co-boundary)



- The **1-ring** of a face f is the formed by the union of the stars of its boundary vertices

- The **k-ring** of a face f, for k>1 is the formed by the union of the 1-rings of faces in its (k-1)-ring



EDITING OPERATIONS

- Mesh processing requires editing operations to change both the geometry and the connectivity of meshes
- Editing based on **primitive operations** that warrant consistency of meshes:

  consistent mesh M → $\boxed{\text{editing op.}}$ → consistent mesh M'

- Consistency is important for the implementation on data structures

EULER OPERATORS

- for general polygonal meshes
- inherited from geometric modeling
- always fulfill the Euler formula $v - e + f = 2s - 2g - h$
- not closed on meshes: intermediate results can be not meshes
- require more general data structures

EULER OPERATORS

- Examples:
  - MVS - MakeVertexShell: creates a new connected component composed of a single vertex
  - MEV - MakeEdgeVertex: creates a new vertex and a new edge, joining it to an existing vertex
  - MEF - MakeEdgeFace: connects two existing vertices with an edge creating a new face - this can either make and fill a loop, or split an existing face into two
  - KHMF - KillHoleMakeFace: fills a hole loop with a face
  - .....

OPERATORS FOR TRIANGLE MESHES

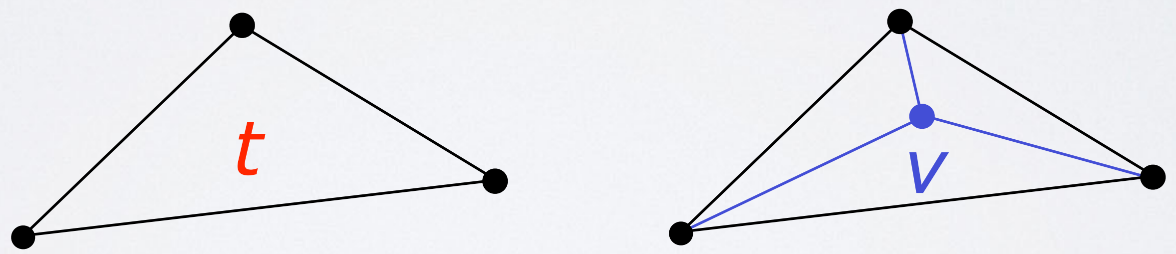
- Specific operators that are closed on triangle meshes:

  triangle mesh *M* → $\boxed{\text{editing op.}}$ → triangle mesh *M'*

- Refinement operators: produce a mesh with more vertices/ edges/faces
- Simplification operators: produce a mesh with less vertices/ edges/faces
- Can be implemented on any topological data structure for triangle meshes

REFINEMENT OPERATORS

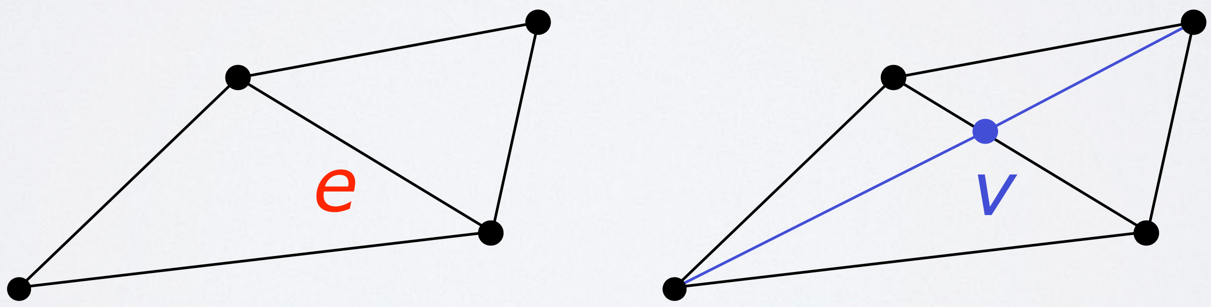
- **Triangle split:**
  - insert a new vertex *v* in a triangle *t* and connect *v* to the vertices of *t* by splitting it into three triangles



Note: this is possible on general meshes with star-shaped faces - a.k.a. *starring*

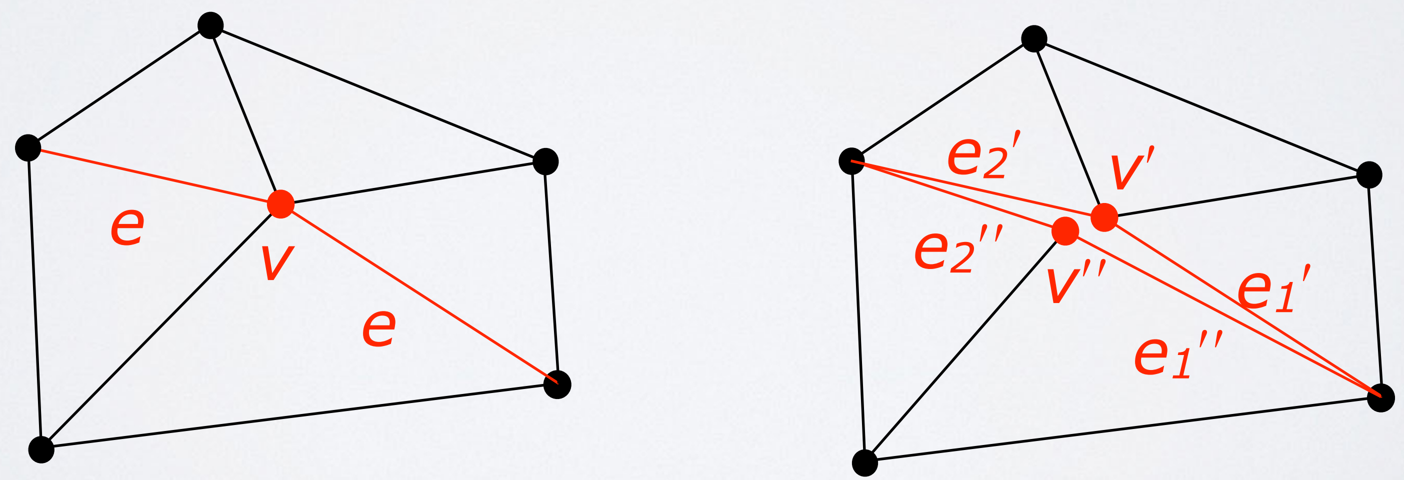
REFINEMENT OPERATORS

- **Edge split:**
  - insert a new vertex *v* on an edge e and connect *v* to the opposite vertices of triangles incident at e by splitting e as well as each such triangle into two



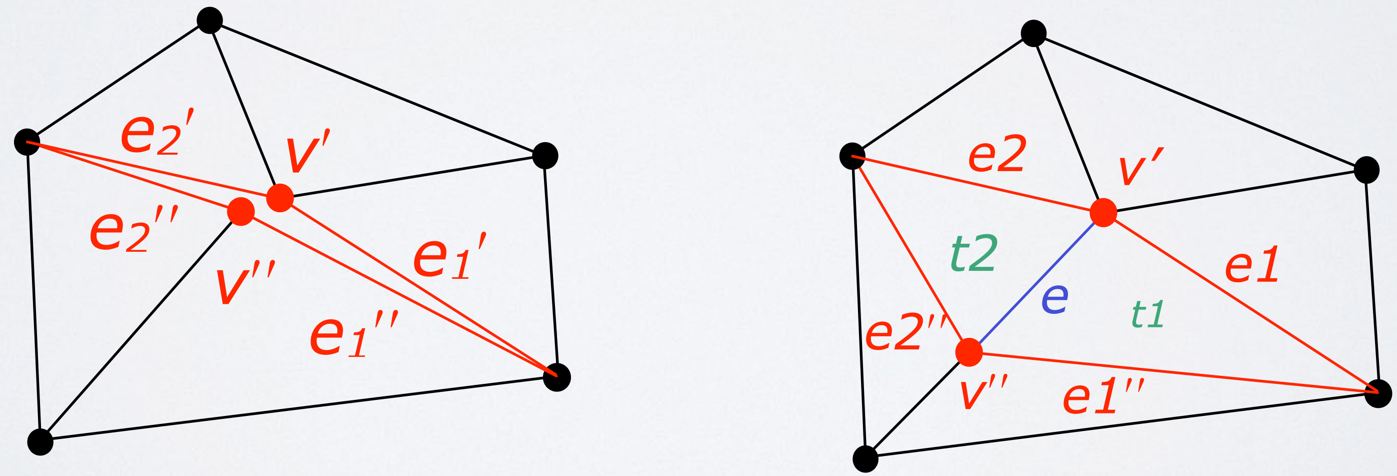
REFINEMENT OPERATORS

- **Vertex split:**
  - cut open the mesh along two edges e_1 and e_2 incident at a common vertex *v*, by duplicating such edges as well as *v*



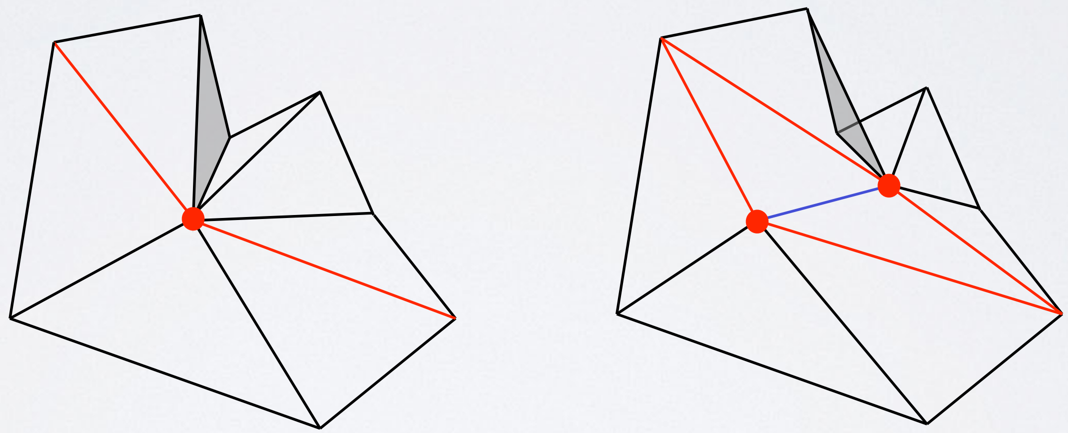
REFINEMENT OPERATORS

- **Vertex split:**
    - cut open the mesh along two edges e_1 and e_2 incident at a common vertex *v*, by duplicating such edges as well as *v*
    - fill the quadrangular hole with two new triangles and an edge joining the two copies of *v*



REFINEMENT OPERATORS

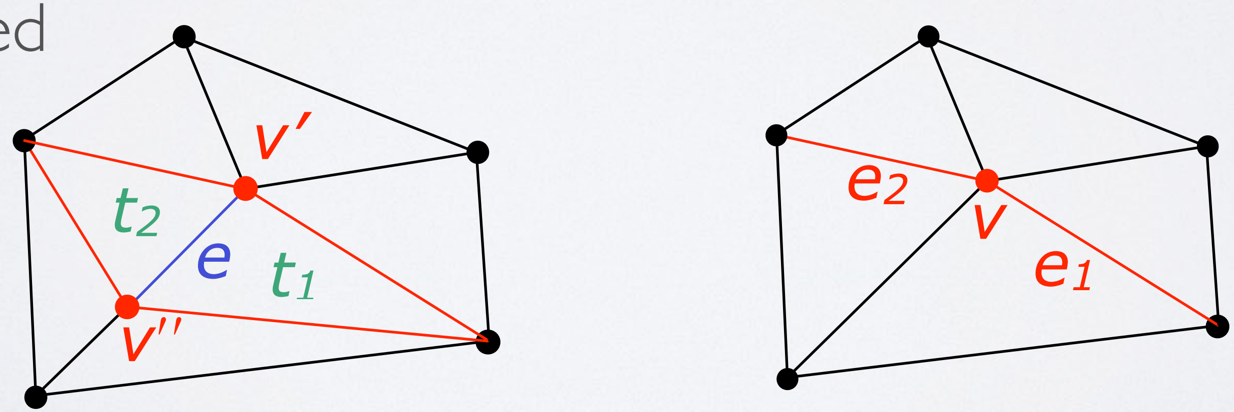
- **Vertex split:**
  - possible inconsistencies because of *triangle flip*



  - flips can be detected by a local test on the orientation of faces: flips changes orientation from clockwise to counter-clockwise and vice-versa

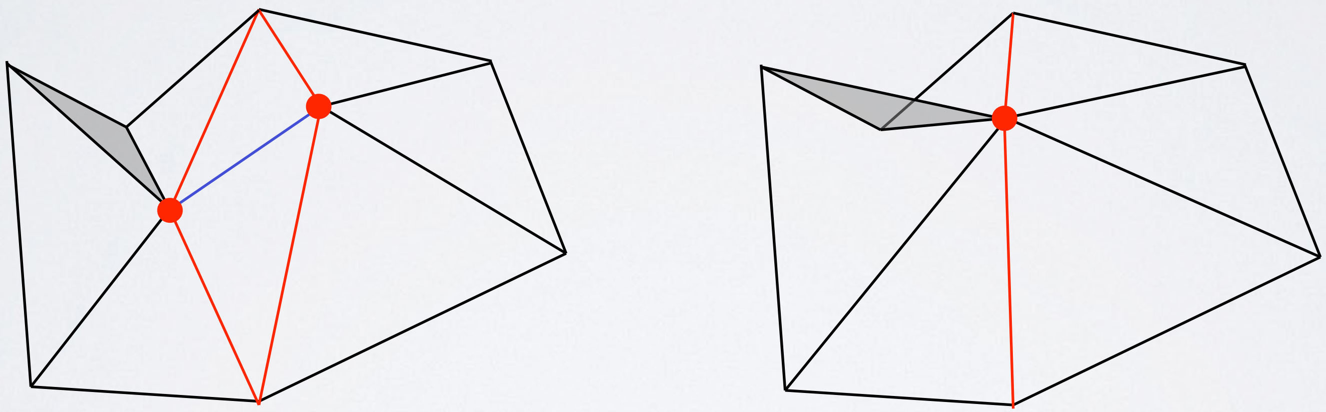
SIMPLIFICATION OPERATORS

- **Edge collapse (reverse of vertex split):**
  - collapse an edge e to a single point
  - e is removed together with its two incident triangles
  - the endpoints of e are identified
  - the other edges bounding the deleted triangles are pairwise identified



SIMPLIFICATION OPERATORS

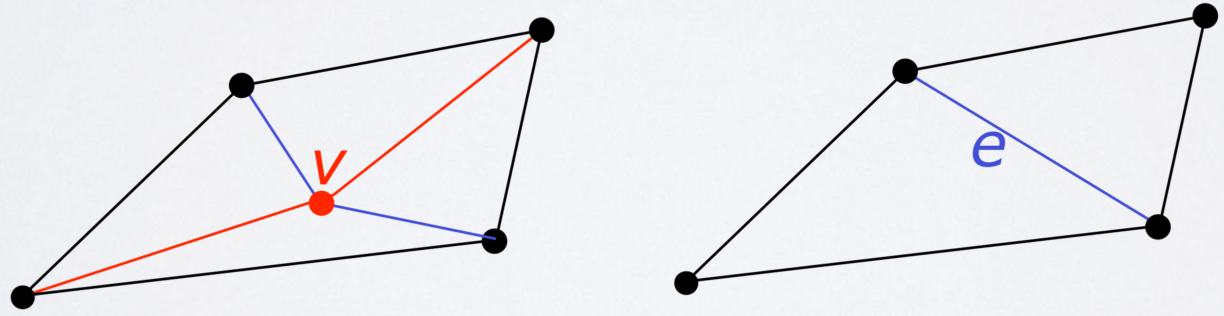
- **Edge collapse:**
    - possible inconsistencies because of *triangle flip*



- consistency check analogous to vertex split

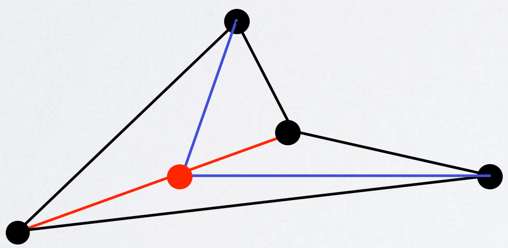
SIMPLIFICATION OPERATORS

- **Edge merge (reverse of edge split):**
  - take an internal vertex *v* with valence 4
  - delete *v* together with its incident triangles and edges and fill the hole with two new triangles sharing a new edge e

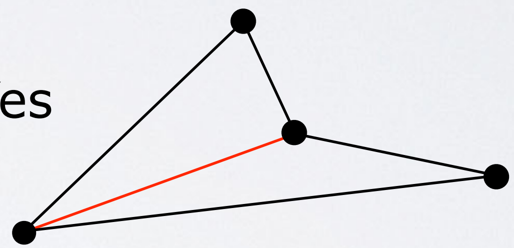


SIMPLIFICATION OPERATORS

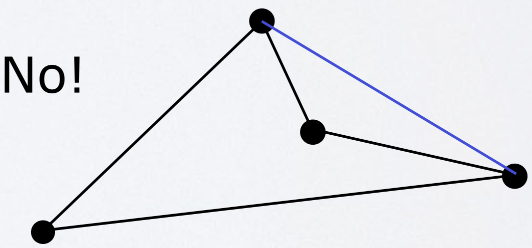
- **Edge merge:**
  - if the hole is not convex, only one diagonal edge can be inserted



Yes

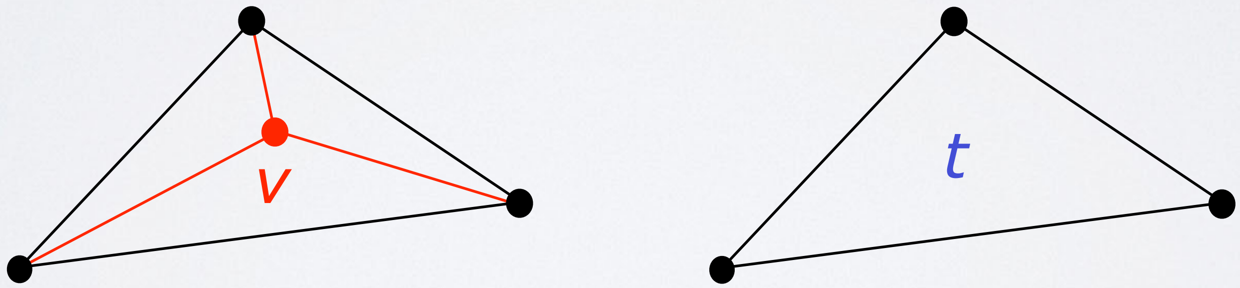


No!



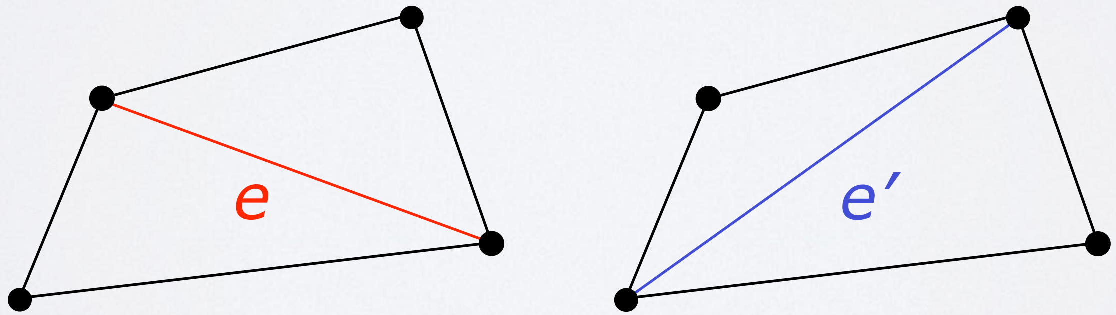
SIMPLIFICATION OPERATORS

- **Delete vertex (reverse of triangle split):**
  - remove an internal vertex *v* of valence 3 together with its incident triangles and edges and fille the hole with a new triangle



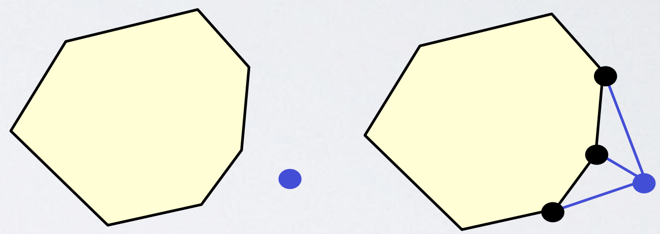
NEUTRAL OPERATOR

- **Edge swap:**
  - consider an edge e such that its two incident triangles form a convex quadrilateral
  - replace e with the opposite diagonal of the quadrilateral, rearranging the two incident triangles accordingly

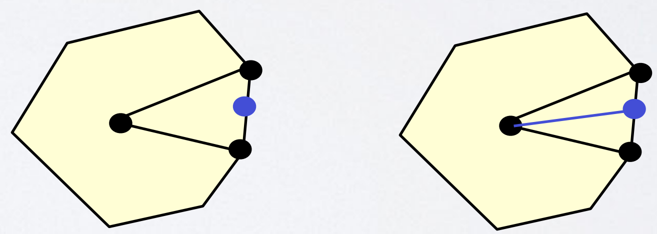


BOUNDARY CASES

- Triangle split / Delete vertex:



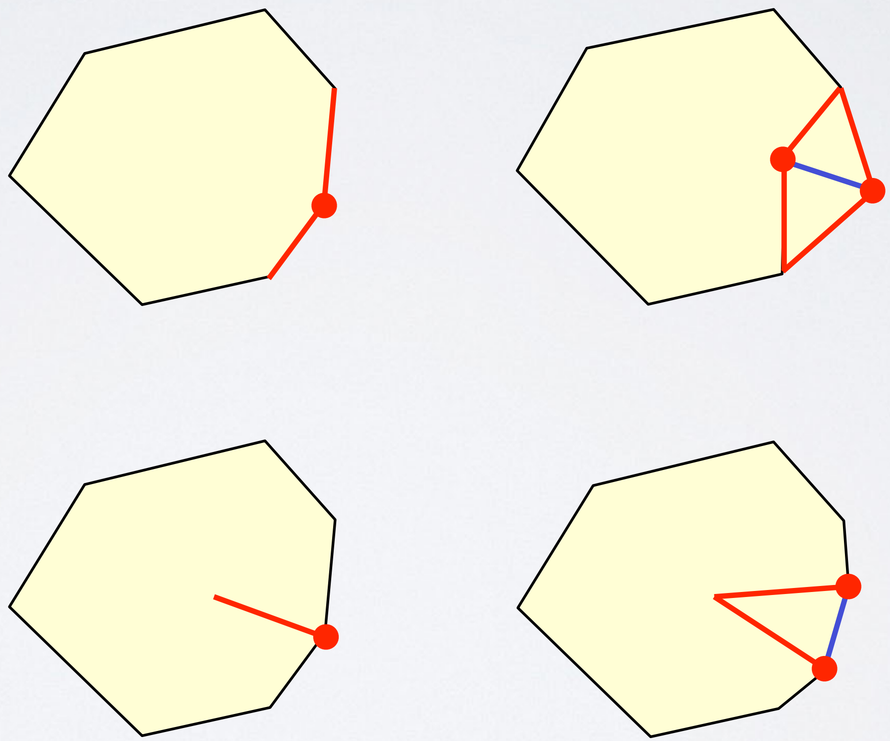
- Edge split / Edge merge:



- Edge swap: NO

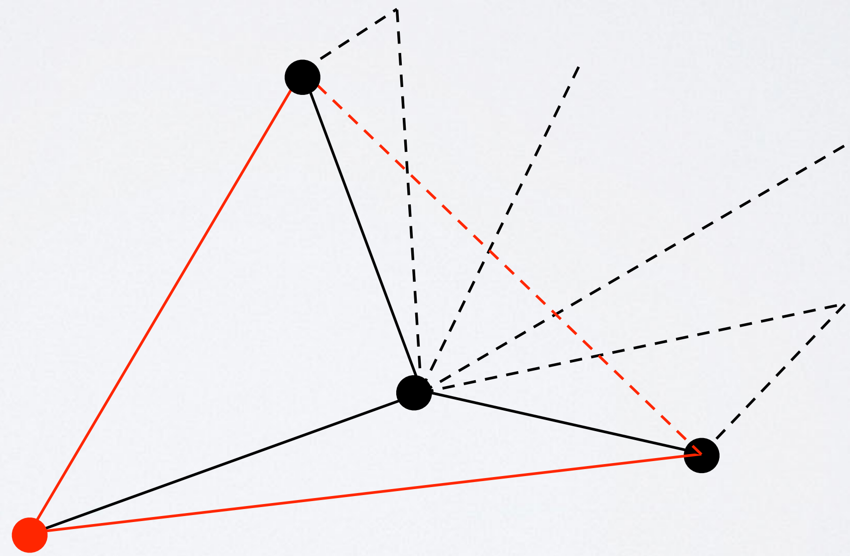
BOUNDARY CASES

- Vertex split / Edge collapse:



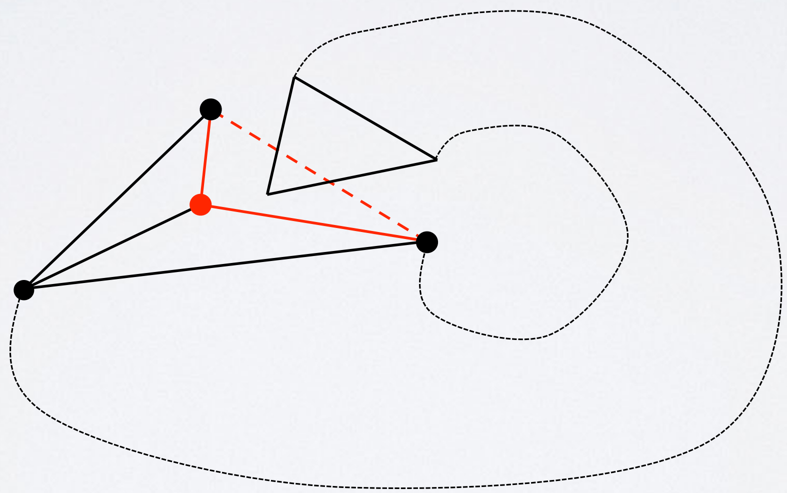
BOUNDARY CASES

- Edge merge on a convex boundary may cause inconsistencies if the quadrilateral formed from the two triangles that merge is not convex
- local check



BOUNDARY CASES

- Edge merge on a concave boundary may cause self-intersection of the mesh
- global check! intersecting parts may be far on the mesh



- similar problems with edge collapse on concave boundary and vertex split on convex boundary