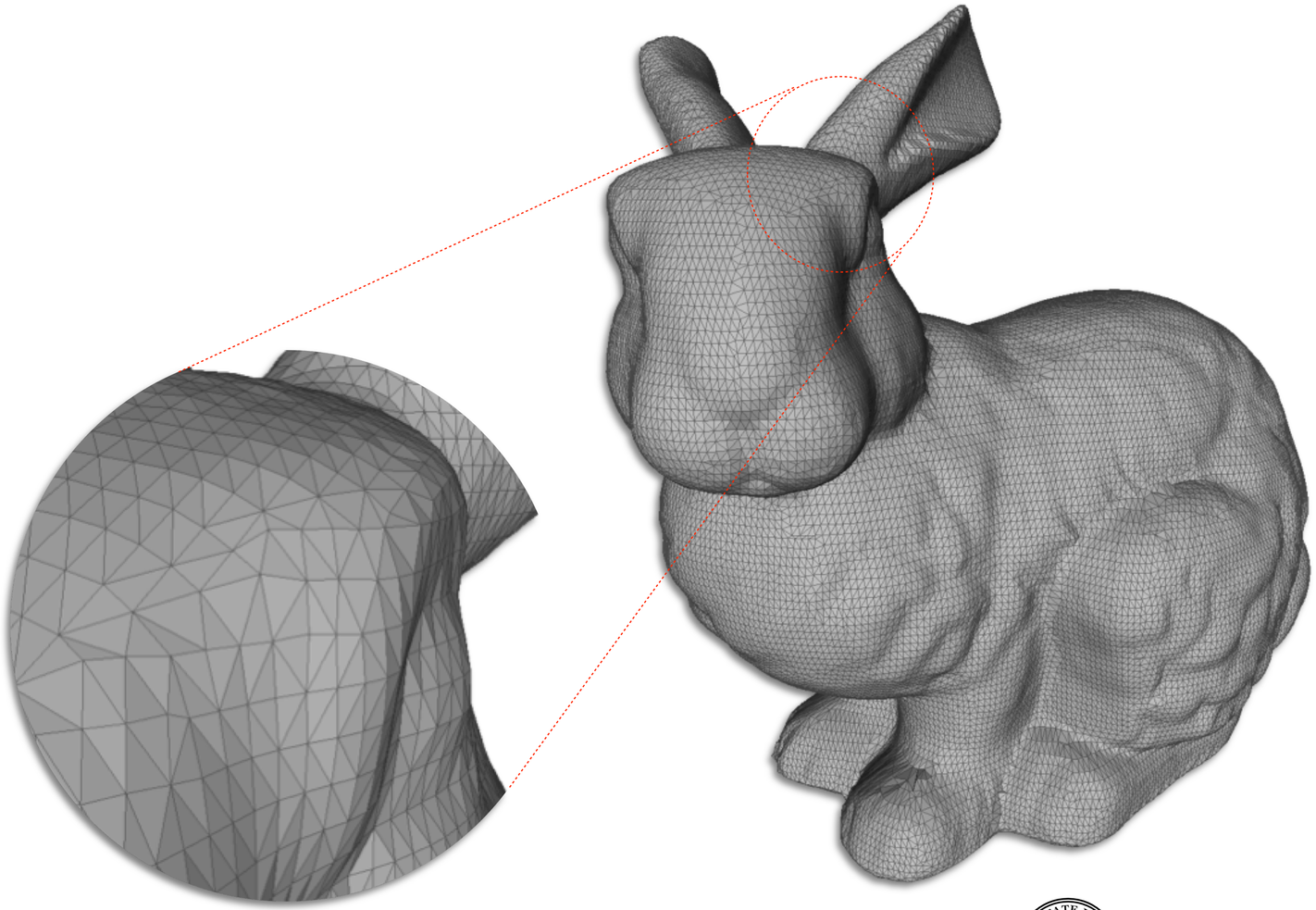
19 - Meshes

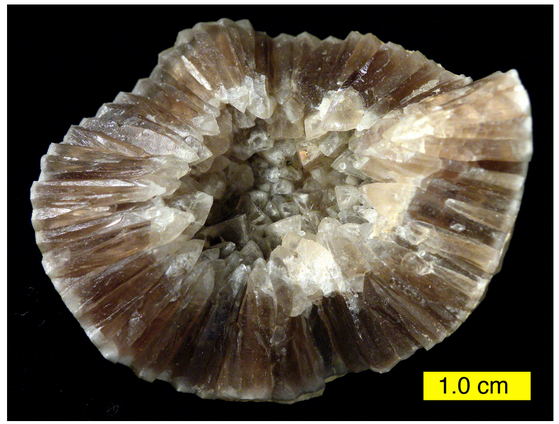
Acknowledgement: Daniele Panozzo, Enrico Puppo

What is a mesh?

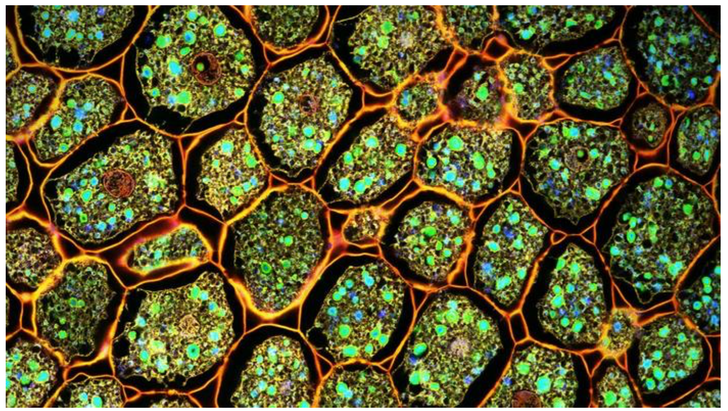
A surface made of polygonal faces glued at common edges

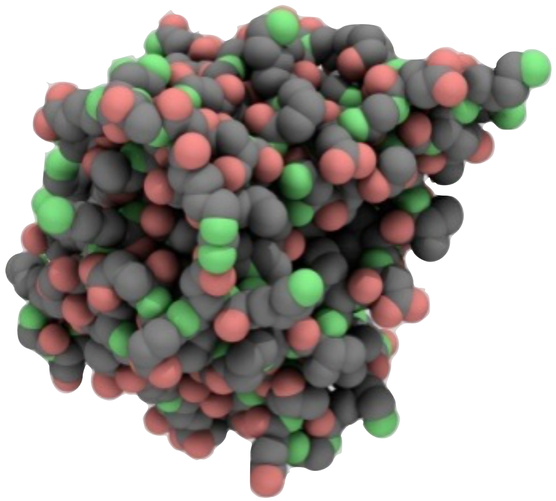


Origin of Meshes



- 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***



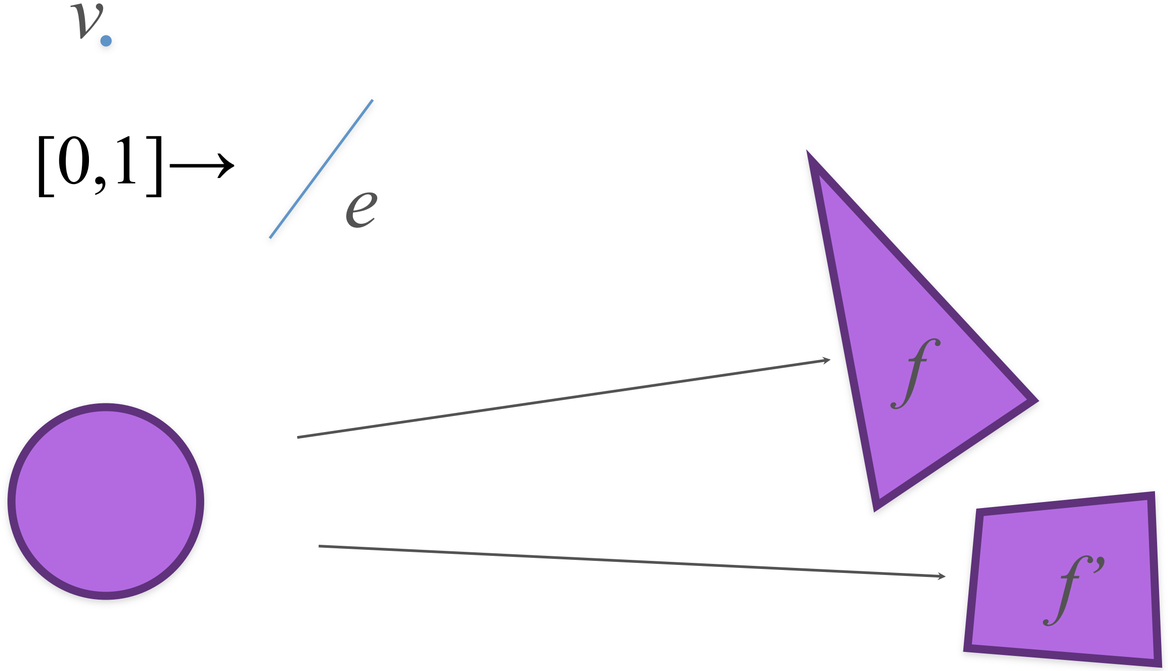


Credit: Fernan Federici Moment Getty Images

By Mark A. Wilson (Department of Geology, The College of Wooster).[1] - Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=12666593

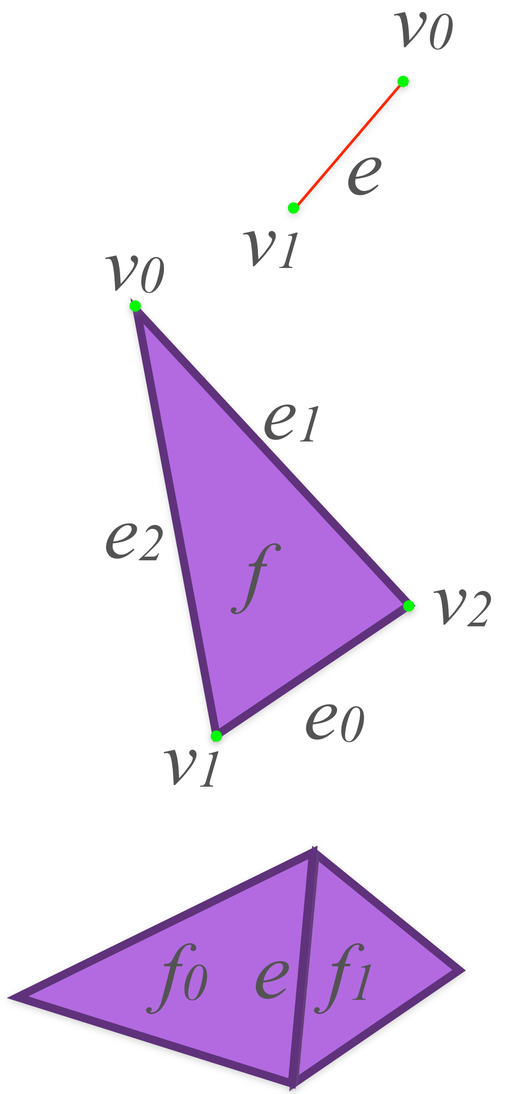
Basic Math of Meshes

- A ***n-cell*** is a set homeomorphic to a Euclidean disc of dimension n:
  - 0-cell: ***vertex***
  - 1-cell: ***edge***
  - 2-cell: ***face***



Structure

- 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



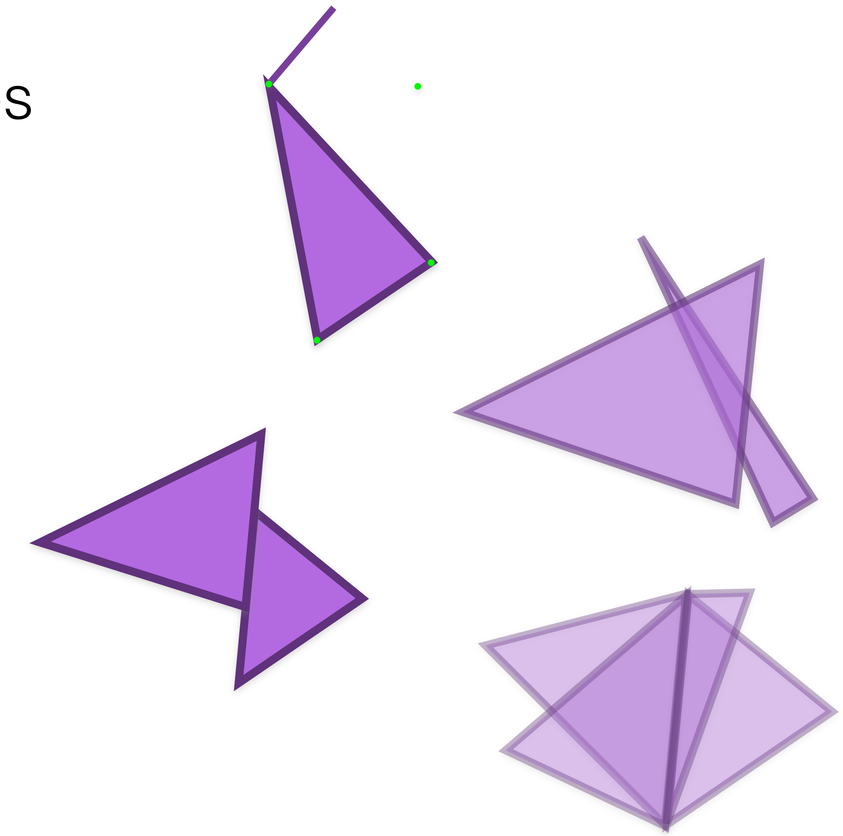
Structure

- 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

Forbidden configurations:

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



Topological Informations

- A mesh can be treated as a purely combinatorial structure

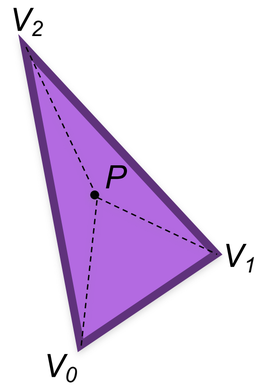
$M = (V, E, F)$

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

Geometrical Information

- 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

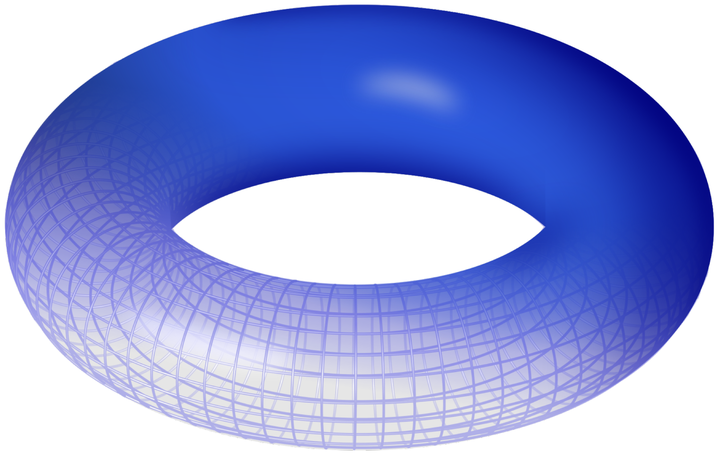
$$P = \lambda_0 V_0 + \lambda_1 V_1 + \lambda_2 V_2 \qquad \lambda_i \in [0,1] \qquad \lambda_0 + \lambda_1 + \lambda_2 = 1$$

  - 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 characteristics of the surface it represents:

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



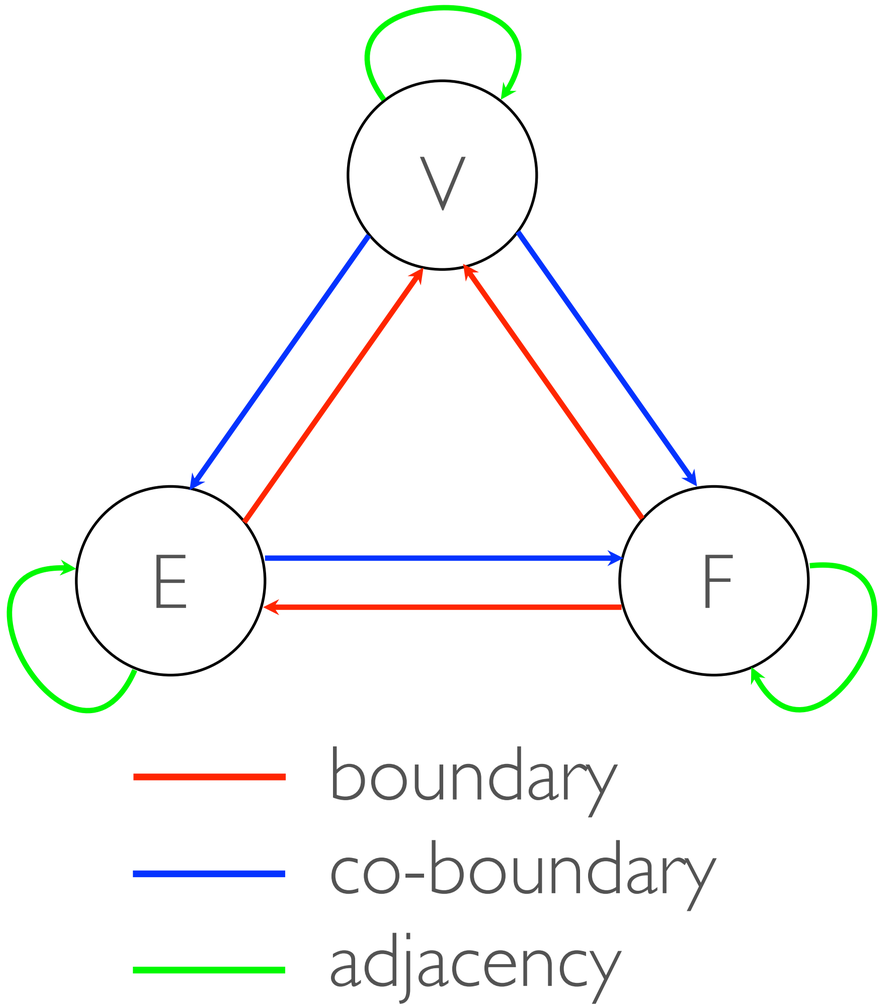
- Shells s = # connected components
- Genus g = # handles (ex.: sphere: genus 0; torus: genus 1)
- Holes h = # boundary loops (watertight: h = 0)

Euler-Poincaré Formula

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

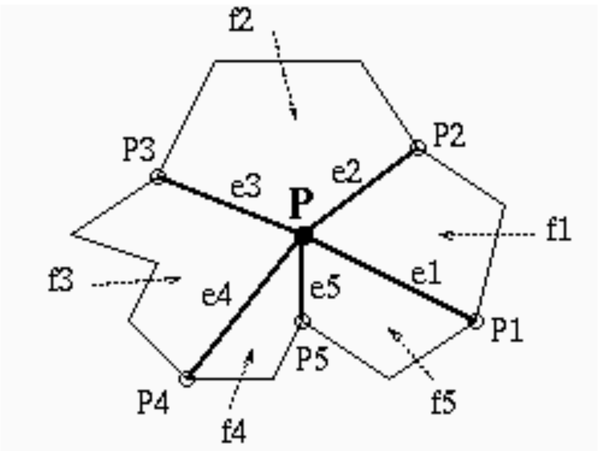
Topological Relations

- **Boundary relations:**
  - for each cell c of dimension n, all cells of dimension <n that belong to its boundary
- **Co-boundary relations:**
  - for each cell c of dimension n, all cells of dimension >n such that c belongs to their boundary
- **Adjacency relations:**
  - for each cell c of dimension n=1,2, all cells of dimension =n such that share some of their boundary with c



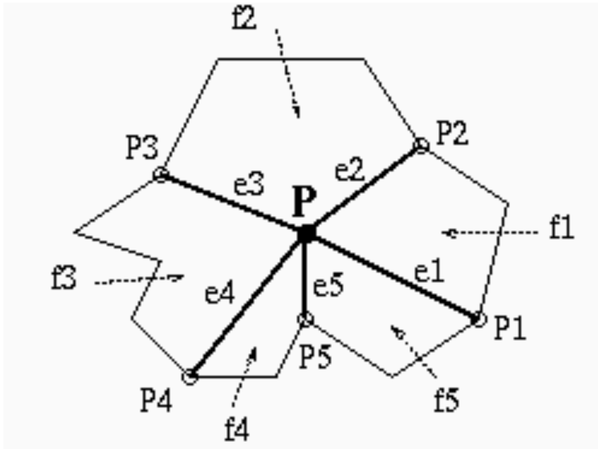
Vertex-Based relations

- 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



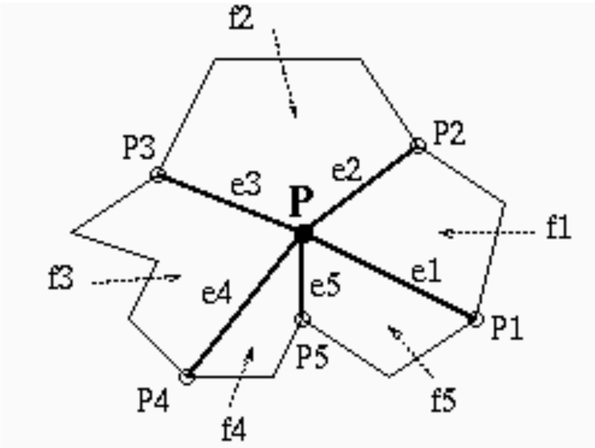
Vertex-Based relations

- VV (Vertex-Vertex):
  - for each vertex v, the list of vertices (v_1,v_2,...,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)



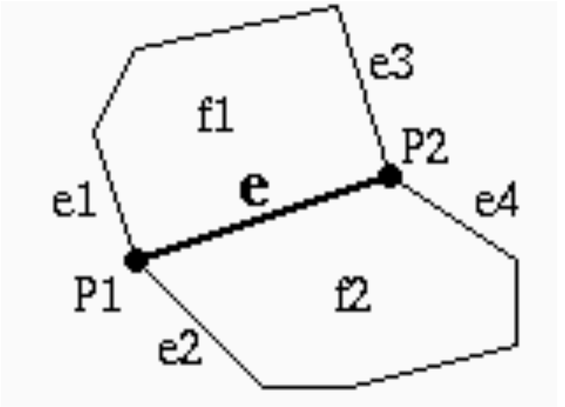
Vertex-Based Relations

- VF (Vertex-Face):
  - for each vertex v, the list of faces (f_1,f_2,…,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)



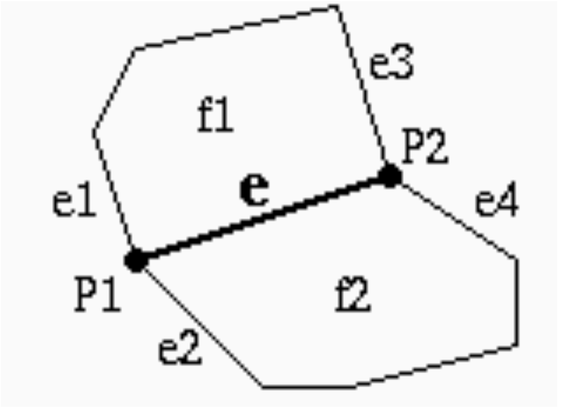
Edge-Based Relations

- 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



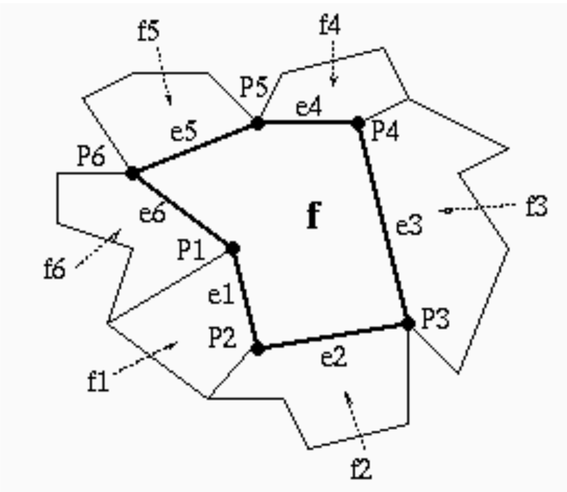
Edge-Based Relations

- 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



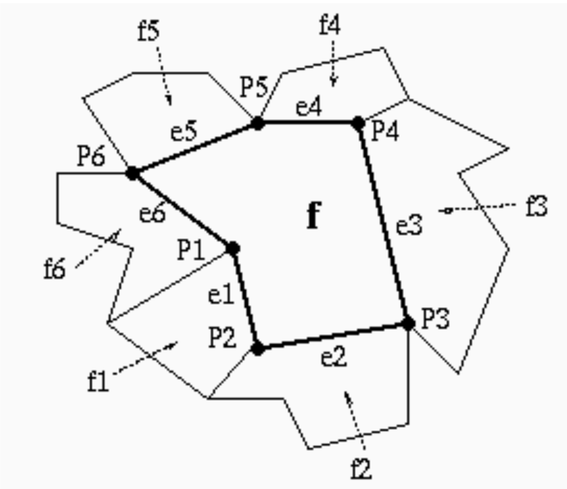
Face-Based Relations

- 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



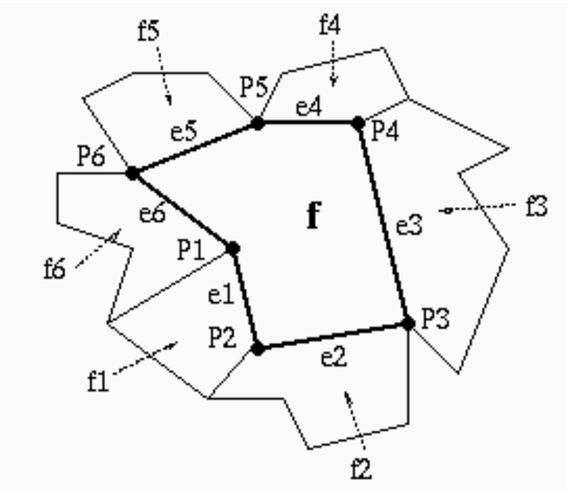
Face-Based Relations

- 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,...,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)

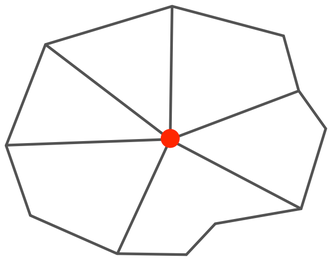


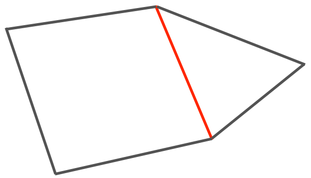
Topological Relations

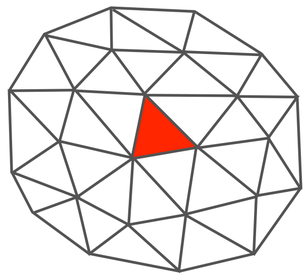
- **Constant relations** return a constant number of elements:
  - EV (each edge has two endpoints)
  - EE (each edge has four adjacent edges)
  - EF (each edge has two incident faces)
- **Variable relations** return a variable number of elements:
  - VV, VE, VF, FV, FE, FF: the number of vertices/edges/faces 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







Level of Details

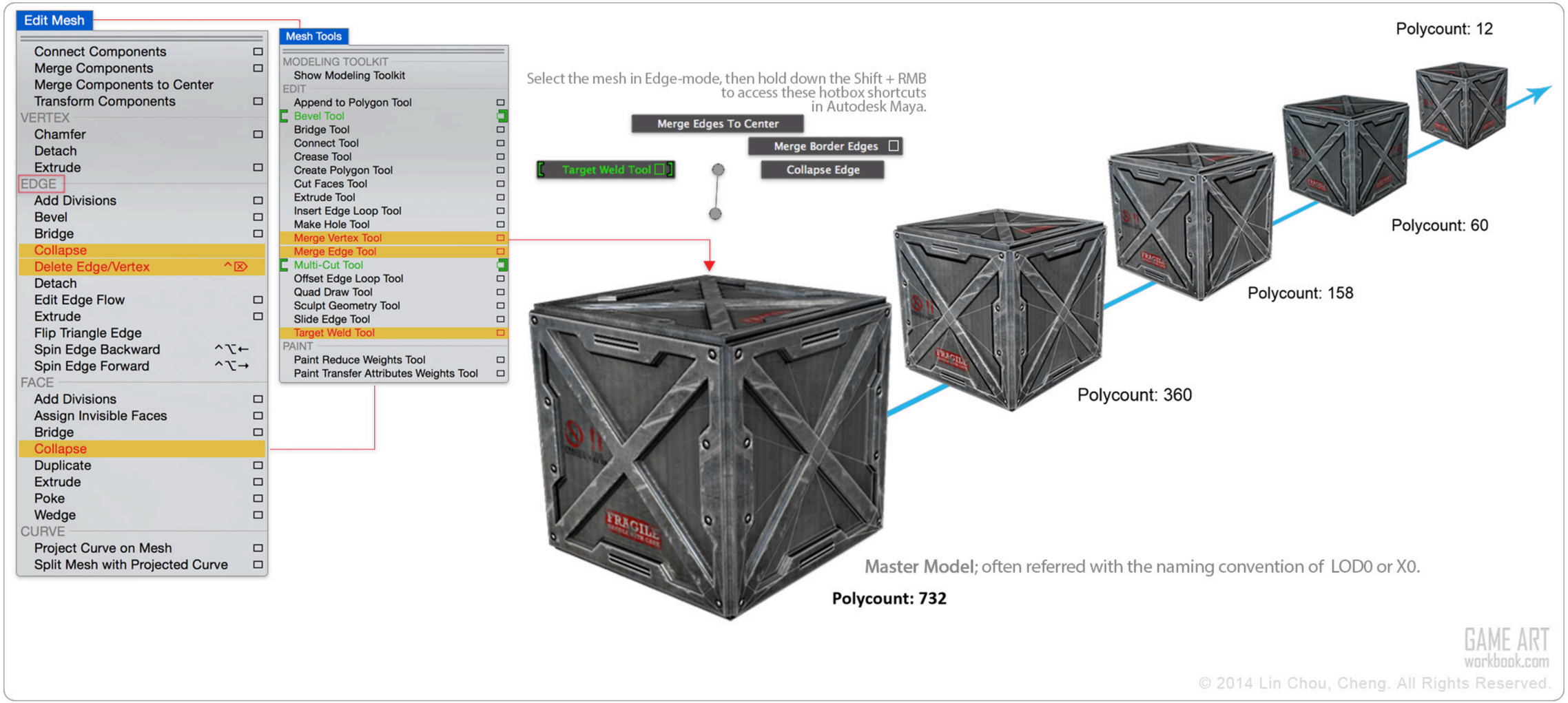
Continuous Methods

- Widely used for rendering terrains

Image					
Vertices	~5500	~2880	~1580	~670	140
Notes	Maximum detail, for closeups.				Minimum detail, very far objects.

Source: http://en.wikipedia.org/wiki/Level_of_detail

Discrete LOD



Editing Operators

Euler Operators

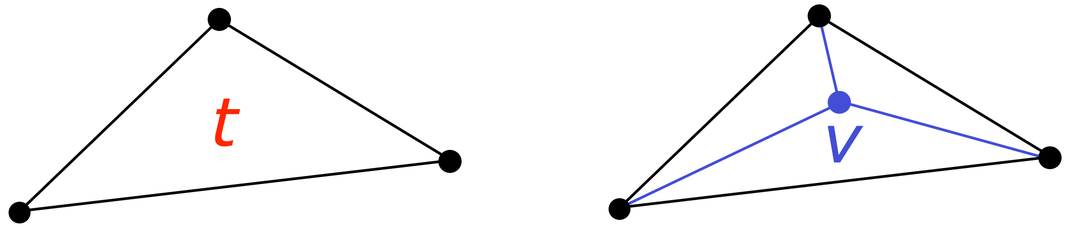
- Change a mesh while fulfilling the Euler formula v - e + f = 2s - 2g - h
- Challenging to implement
- Examples:
  - MVS - MakeVertexShell: creates a new connected component composed of a single vertex
  - MEV - MakeEdgeVertex: creates a new vertex and a new edge, joining it to an existing vertex
  - 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 specific for triangle meshes
- **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

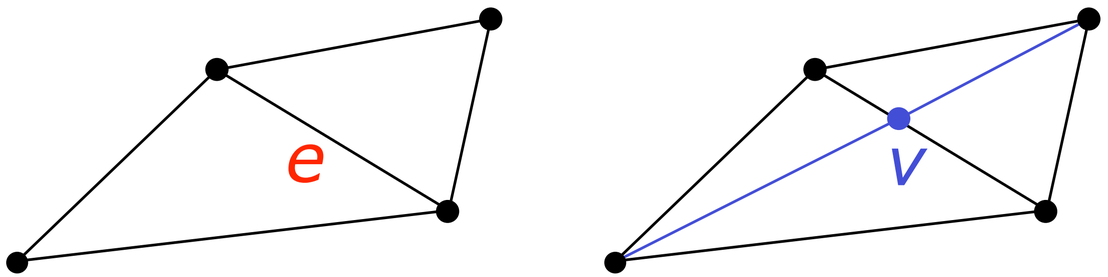
Refinement Operators

- **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



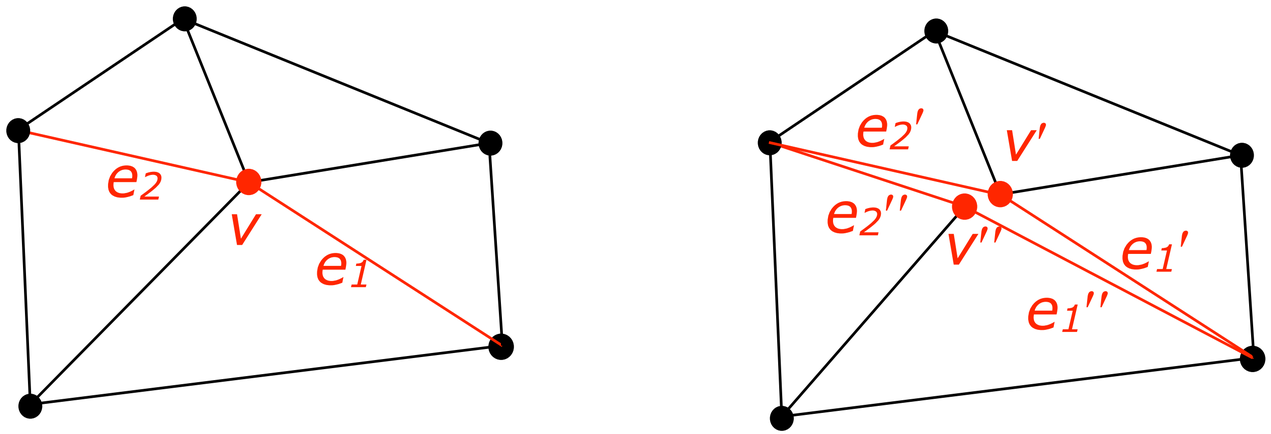
Refinement Operators

- **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



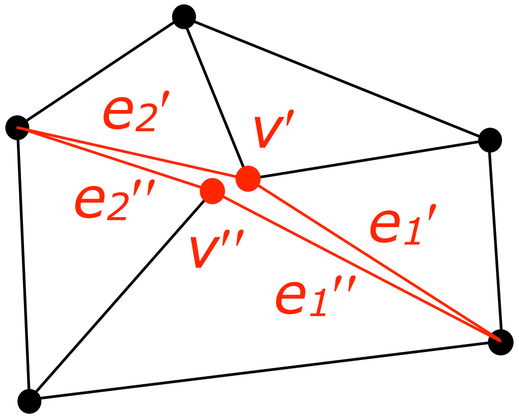
Refinement Operators

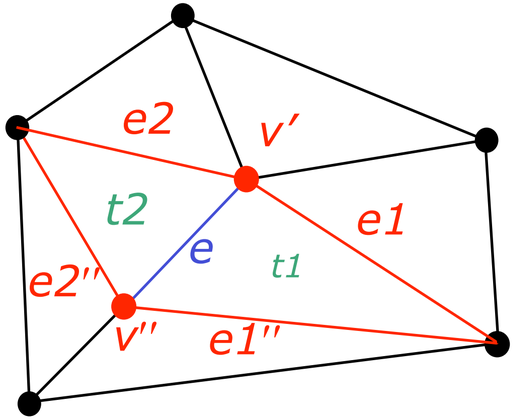
- **Vertex split:**
  - cut open the mesh along two edges e_1 and e_2 incident at a common vertex v, by duplicating such edges as well as v



Refinement Operators

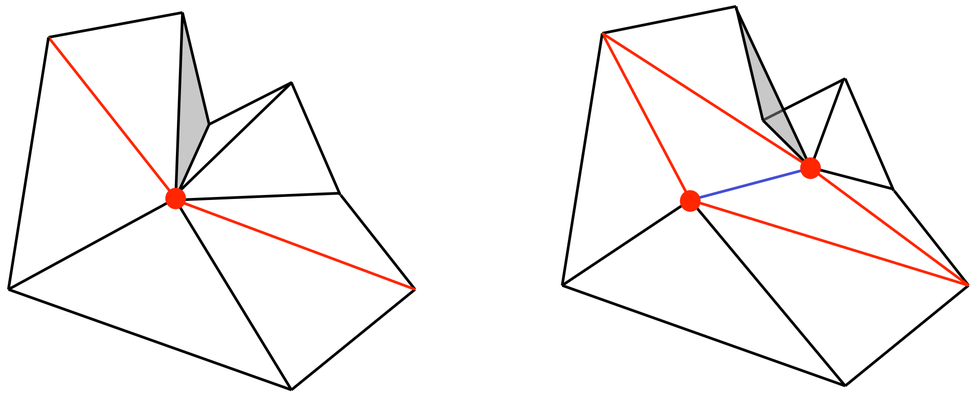
- **Vertex split:**
  - cut open the mesh along two edges e_1 and e_2 incident at a common vertex v, by duplicating such edges as well as v
  - fill the quadrangular hole with two new triangles and an edge joining the two copies of v





Refinement Operators

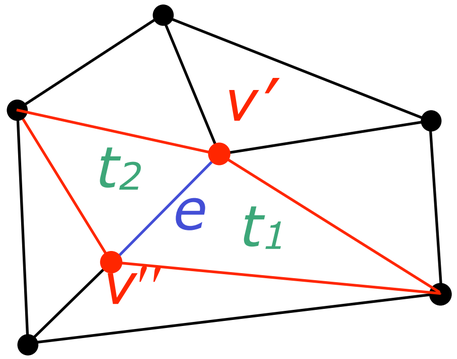
- **Vertex split:**
  - possible inconsistencies because of *triangle flip*

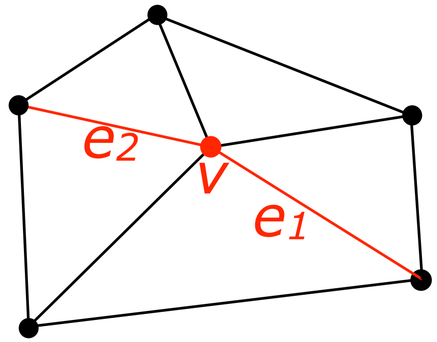


  - flips can be detected by a local test on the orientation of faces: flips changes orientation from clockwise to counter-clockwise and vice-versa

Simplification Operators

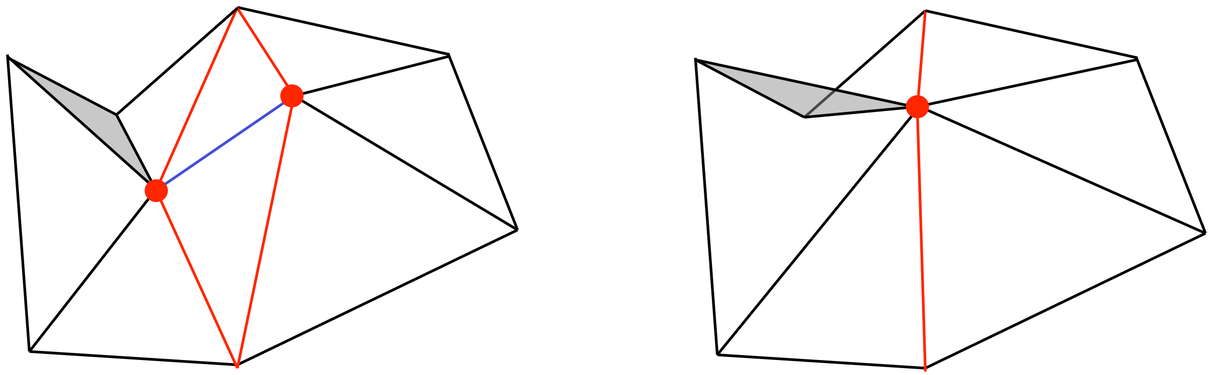
- **Edge collapse (reverse of vertex split):**
  - collapse an edge ***e*** to a single point
  - ***e*** is removed together with its two incident triangles
  - the endpoints of ***e*** are identified
  - the other edges bounding the deleted triangles are pairwise identified





Simplification Operators

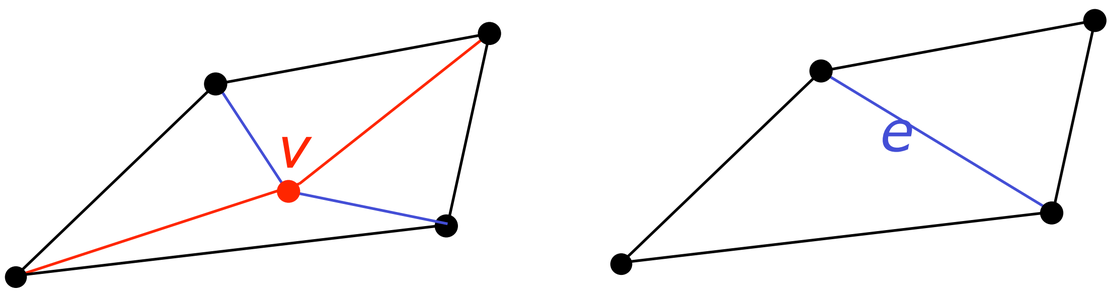
- **Edge collapse:**
  - possible inconsistencies because of *triangle flip*



  - consistency check analogous to vertex split

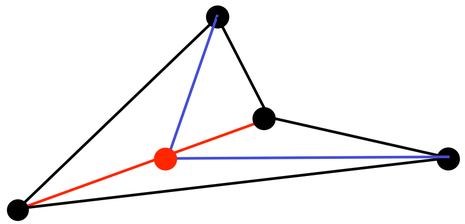
Simplification Operators

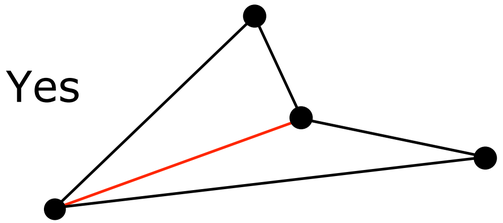
- **Edge merge (reverse of edge split):**
  - take an internal vertex ***v*** with valence 4
  - delete ***v*** together with its incident triangles and edges and fill the hole with two new triangles sharing a new edge ***e***

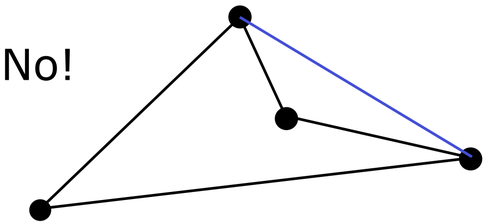


Simplification Operators

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

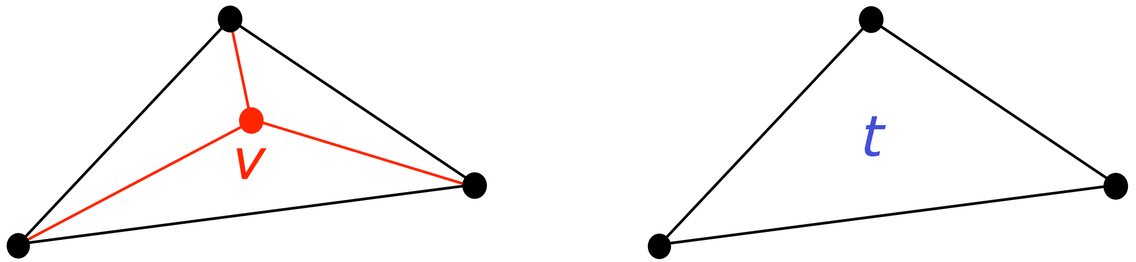






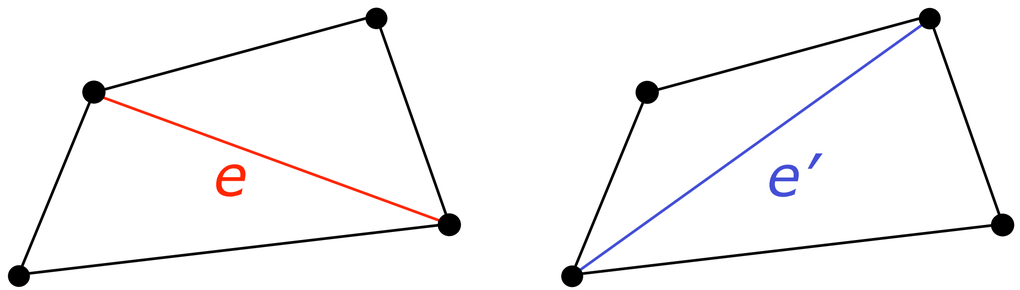
Simplification Operators

- **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



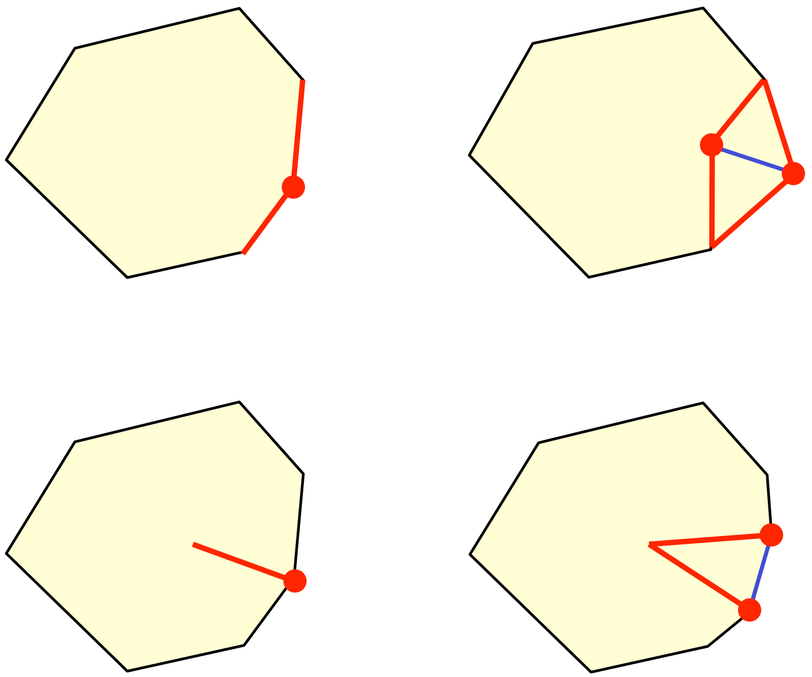
Neutral Operator

- **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



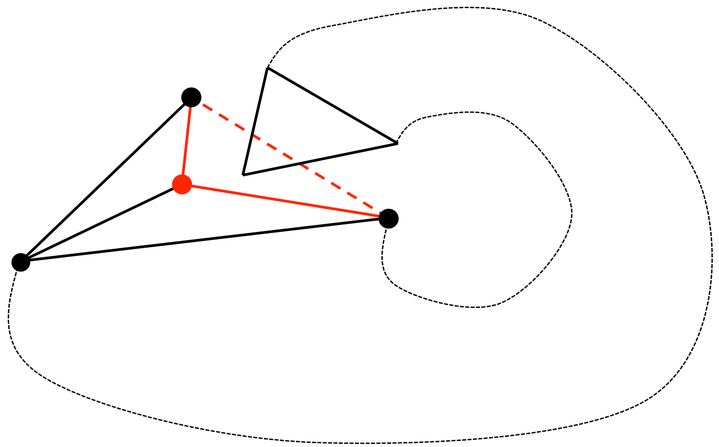
Boundary Cases

- Vertex split / Edge collapse:



Boundary Cases

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



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

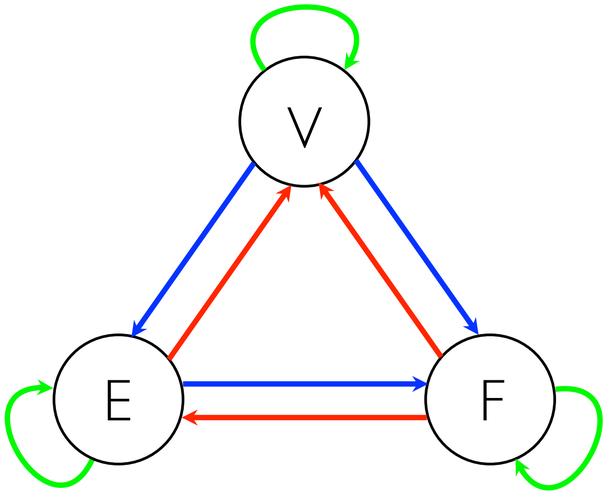
Mesh Data Structures

Data Structures

- Storing a mesh:
  - geometry: position of vertices
  - connectivity: edges, faces
  - topology: topological relations
- Compactness vs efficiency trade-off: data structures should be compact, while efficiently supporting time-critical operations
  - storage requirements
  - traversal operations
  - update operations

Data Structures

- Information to store/retrieve:
  - Entities: vertices, edges, faces
  - Relations: VV, VE, VF, EV, EE, EF, FV, FE, FF
  - Additional properties attached to any entity (Positions, Normals, Colors)
- The data structure to use is application-dependent



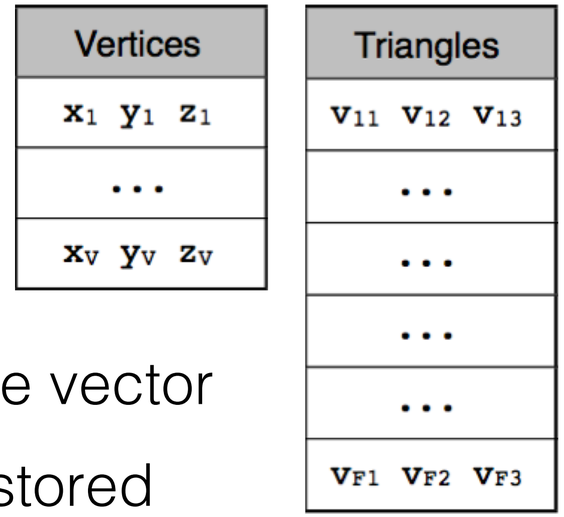
Polygon Soup

Triangles		
x_{11} y_{11} z_{11}	x_{12} y_{12} z_{12}	x_{13} y_{13} z_{13}
x_{21} y_{21} z_{21}	x_{22} y_{22} z_{22}	x_{23} y_{23} z_{23}
...	...	...
x_{F1} y_{F1} z_{F1}	x_{F2} y_{F2} z_{F2}	x_{F3} y_{F3} z_{F3}

- a.k.a. Face set - STL files
- Just store a list of faces
- For each face, store positions of its vertices
  - For general polygonal mesh, the number of vertices of each face must also be stored
  - Number implicit for triangle meshes
- For a triangle mesh: 36 bytes/face ≈ 72 bytes/vertex
- No connectivity! just a collection of polygons
- Streaming structure: no need to store it all in memory to render the mesh

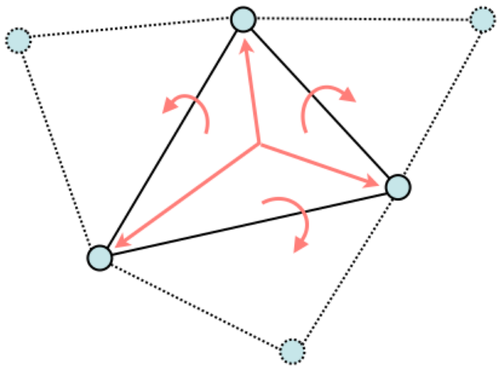
Indexed Structure

- Avoid replication of vertices - OBJ, OFF, PLY files
- Maintain a vector of vertices and a vector of faces
  - For each vertex: position
  - For each face: references to positions of its vertices (FV) in the vector
  - in general, the number of vertices of each face must also be stored
  - number implicit for triangle meshes
- For a triangle mesh: 12 bytes/vertex +12 bytes/face ≈ 36 bytes/vertex
- Encodes connectivity through the FV relation
- Main memory structure: need to store in memory at least the whole list of vertices



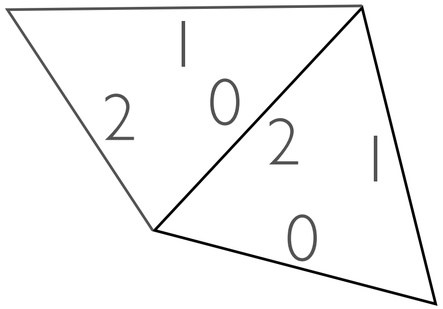
Indexed Structure with Adjacencies

- Just for triangle meshes
- Extends indexed structure with some topological relations:
  - For each vertex:
    - position
    - one reference to an incident triangle (VF* relation)
  - For each face:
    - references to its three vertices (FV)
    - references to its three adjacent triangles (FF)



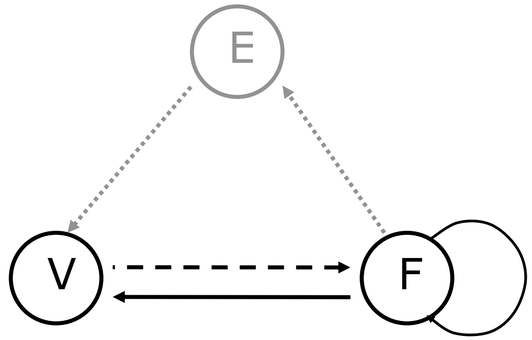
Indexed Structure with Adjacencies

- No explicit edges!
  - implicitly defined as pairs of vertices (unique)
  - or as pairs (f,i) with f triangle and i index in 0,1,2 (not unique!)
- Attributes for edges require care, especially when modifying the mesh with editing operators



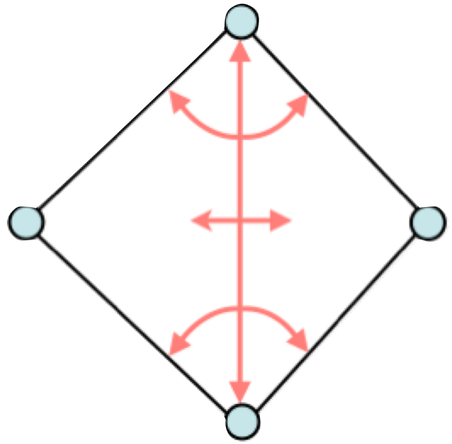
Indexed Structure with Adjacencies

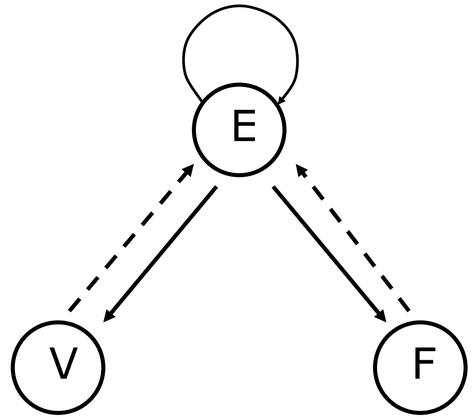
- Evaluation of topological relations:
  - FV, FF: encoded - optimal
  - VF = VF* + FF + FV optimal
  - VV analogous to VF
- Relations involving edges are either implicit or they can be evaluated in optimal time by the (f,i) encoding



Winged Edge Data Structure

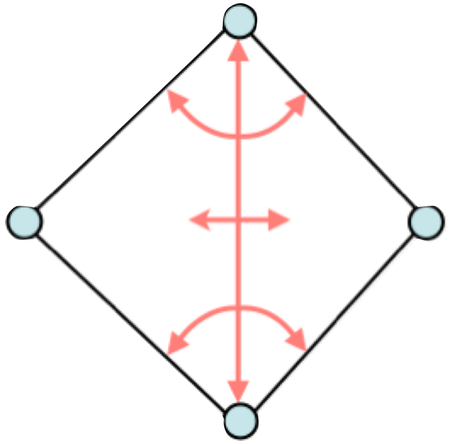
- Topological structure for general polygonal meshes
  - For each vertex:
    - position
    - one reference to an incident edge (VE*)
  - For each edge:
    - references to its two vertices (EV)
    - references to its two incident faces (EF)
    - references to its four adjacent edges (EE)
  - For each face:
    - one reference to an incident edge (FE*)

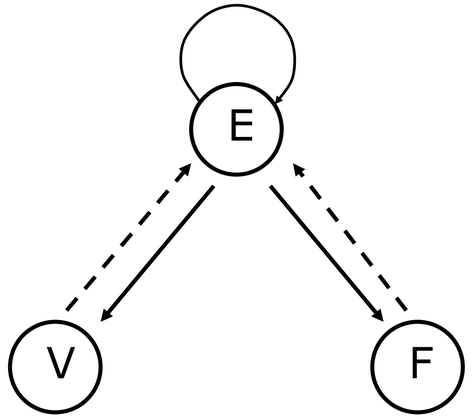




Winged Edge Data Structure

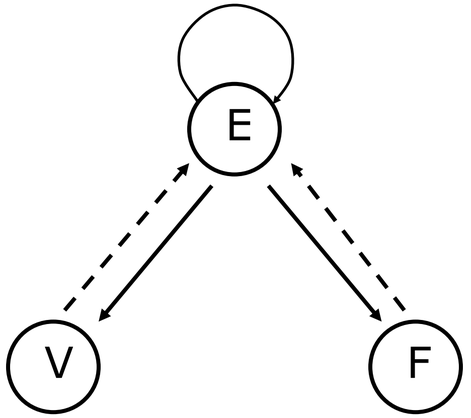
- 120 bytes/vertex
- all entities are represented explicitly
- all topological relations can be retrieved in optimal time
- ambiguous orientation of edges:
  - does EV(e) return (v_i,v_j) or (v_j,v_i)?

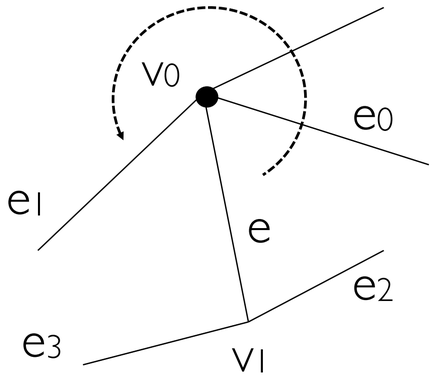




Winged Edge Data Structure

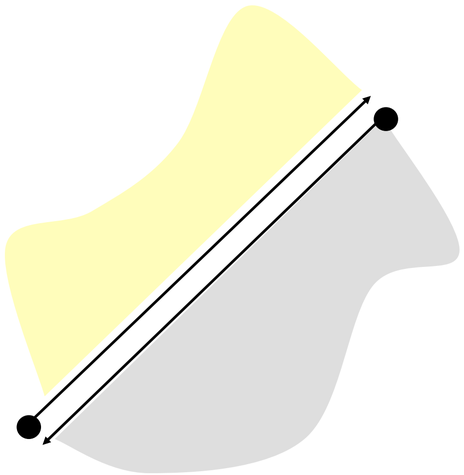
- Example: evaluation of VE(v):
  - get first edge e = VE*(v)
  - evaluate (e_0,e_1,e_2,e_3) = EE(e)
  - evaluate (v_0,v_1) = EV(e)
  - in counter-clockwise order, set next edge at either e_0 or e_3, depending on whether $v=v_0$ or $v=v_1$
  - cycle until hitting e





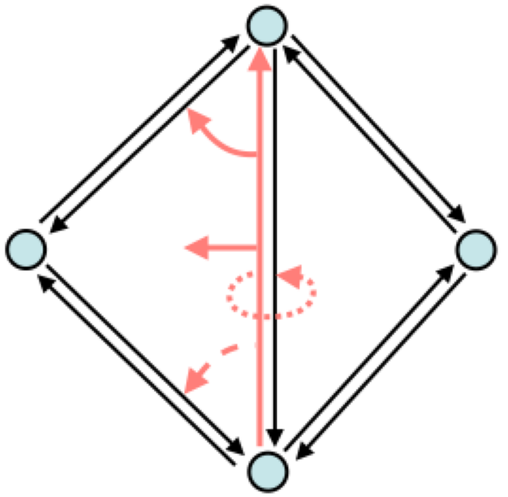
Half-Edge Data Structure

- Half-edge: each edge is duplicated by also considering its orientation
- An edge corresponds to a pair of sibling half-edges with opposite orientations
- Each half-edge stores half topological information concerning the edge



Half-Edge Data Structure

- For each vertex:
  - position
  - one reference to an incident half-edge
- For each half-edge:
  - reference to one its two vertices (origin)
  - references to one of its two incident faces (face on its left)
  - references to: next/previous half-edge on the same face, sibling half-edge
- For each face:
  - one reference to an incident half-edge (FE*)



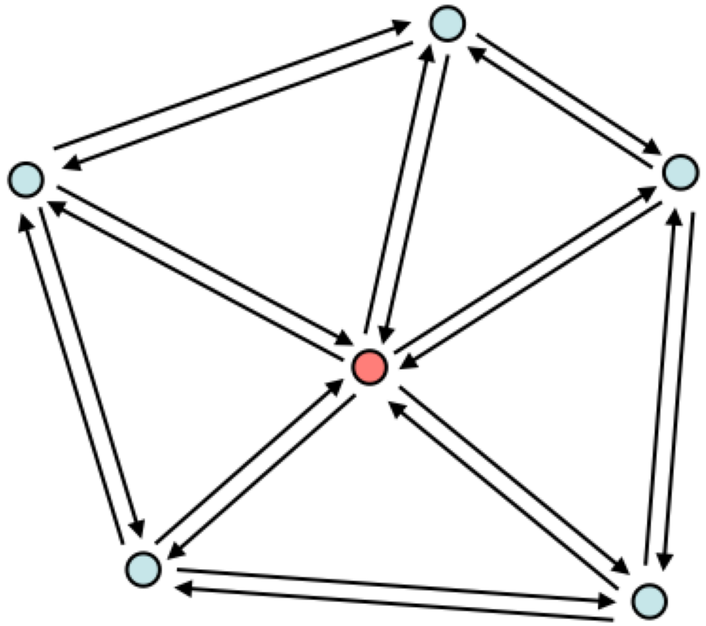
Half-Edge Data Structure

- 96-144 bytes/vertex depending on number of references to adjacent edges
  - reference to sibling half-edge can be avoided by storing siblings at consecutive entries of a vector
  - for triangle meshes, just one reference to either next or previous half-edge is sufficient
- Efficient traversal and update operations
- Attributes for edges must be stored separately

Half-Edge Data Structure

- One-ring traversal (V* relations):

  1.start at vertex

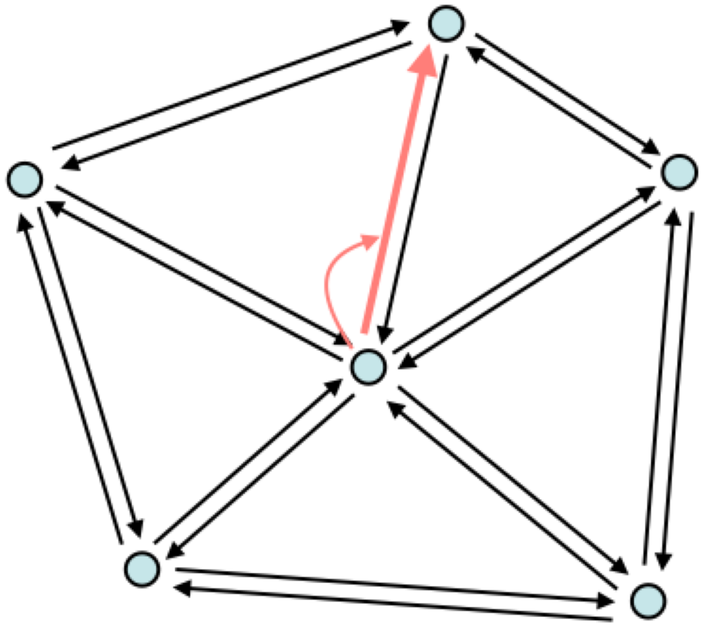


Half-Edge Data Structure

- One-ring traversal (V* relations):

  1.start at vertex

  2.outgoing half-edge



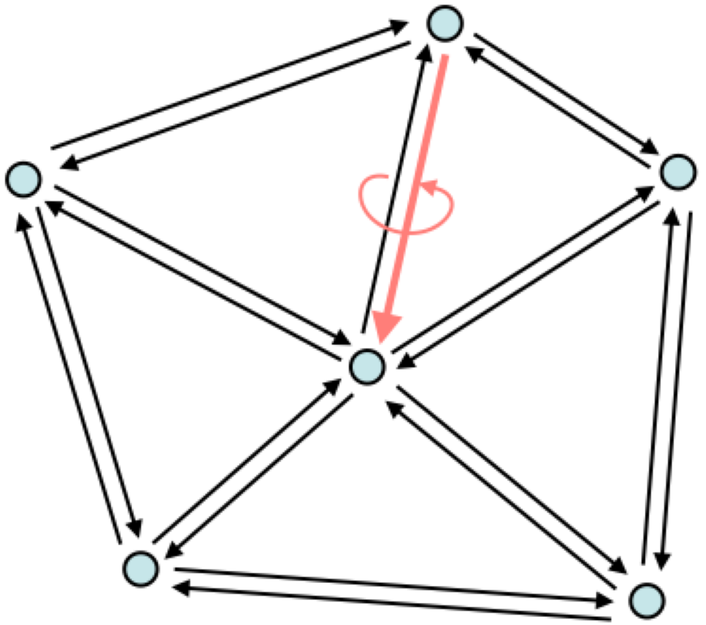
Half-Edge Data Structure

- One-ring traversal (V* relations):

  1.start at vertex

  2.outgoing half-edge

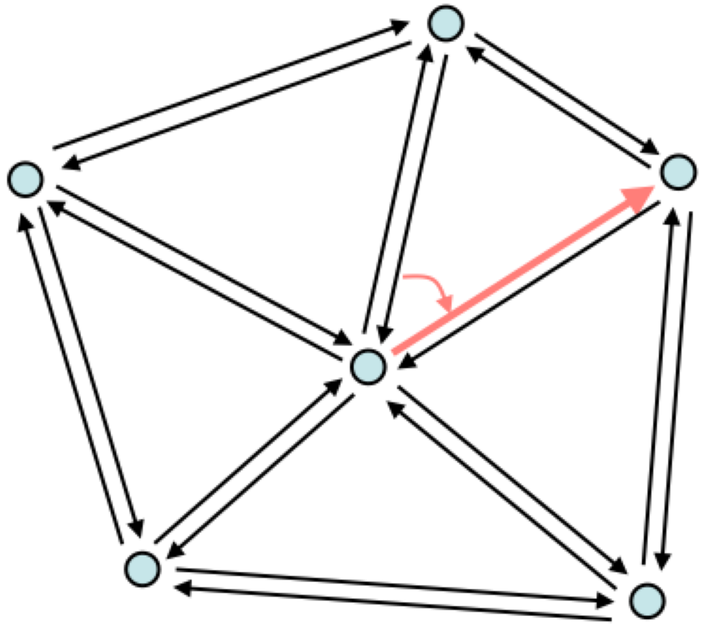
  3.opposite half-edge



Half-Edge Data Structure

- One-ring traversal (V* relations):

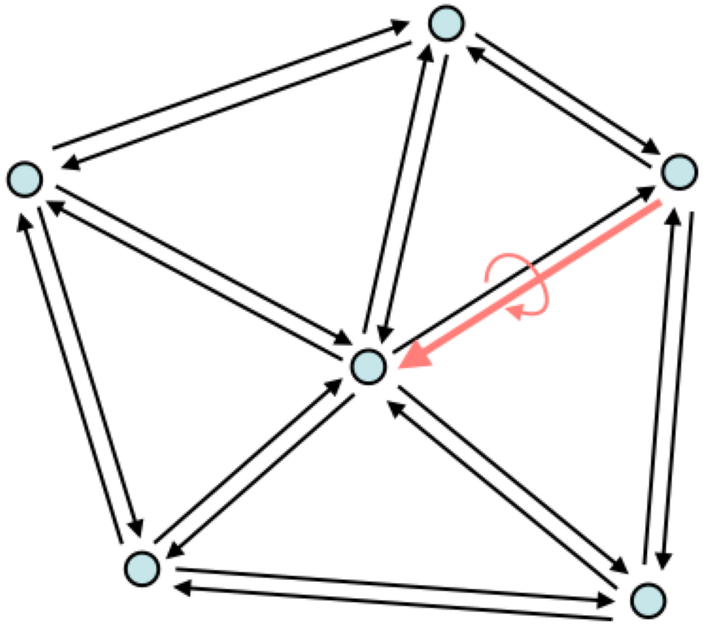
  1. start at vertex
  2. outgoing half-edge
  3. opposite half-edge
  4. next half-edge



Half-Edge Data Structure

- One-ring traversal (V* relations):

  1. start at vertex
  2. outgoing half-edge
  3. opposite half-edge
  4. next half-edge
  5. opposite



Half-Edge Data Structure

- One-ring traversal (V* relations):

  1.start at vertex

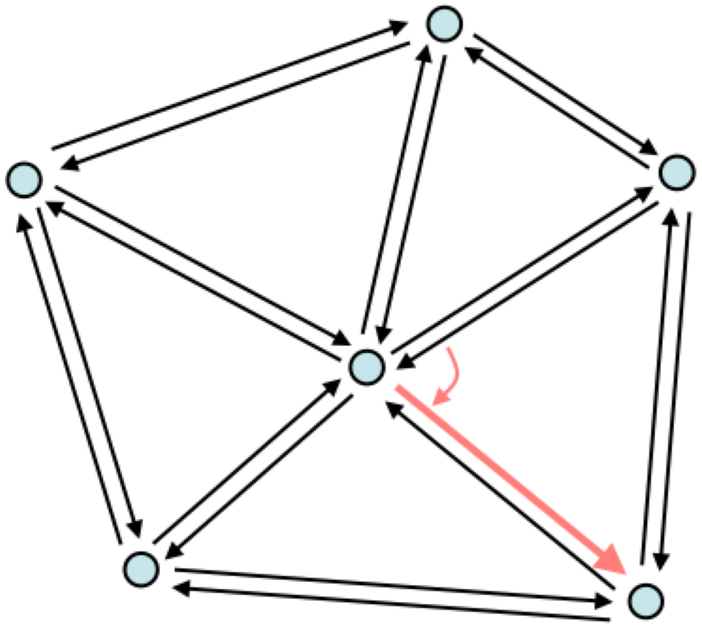
  2.outgoing half-edge

  3.opposite half-edge

  4.next half-edge

  5.opposite

  6.next.....



C++ libraries

- Indexed based:
  - libigl (`igl.ethz.ch/projects/libigl/`):
    - light mesh representation compatible with numerical software (e.g., Eigen, Matlab)
    - topological relations are computed on-the-fly and can be stored for later use
    - several geometry processing algorithms algorithms
    - free, MPL2 licence

C++ libraries

- Adjacency based:
  - VCGlib (`vcg.sourceforge.net`):
    - optimized for triangle and tetrahedral meshes
    - extensions with half-edge for more general meshes
    - several geometry processing algorithms and spatial data structures
    - free, LGPL licence

C++ libraries

- Half-edge based:
  - CGAL (`www.cgal.org`):
    - rich, complex
    - computational geometry algorithms
    - free for non-commercial use
  - OpenMesh (`www.openmesh.org`):
    - mesh processing algorithms
    - free, LGPL licence

TOOLS

- Meshlab (`meshlab.sourceforge.net`) - free:
  - triangle mesh processing with many features
  - based on the VCGlib
- OpenFlipper (`www.openflipper.org`) - free:
  - polygon mesh modeling and processing
  - based on OpenMesh
- Graphite (`alice.loria.fr`) - free:
  - polygon mesh modeling, processing and rendering
  - based on CGAL

References

Fundamentals of Computer Graphics, Fourth Edition
4th Edition **by** **Steve Marschner**, **Peter Shirley**
Chapter 12