3D Geometric Modeling

Geometric Modeling

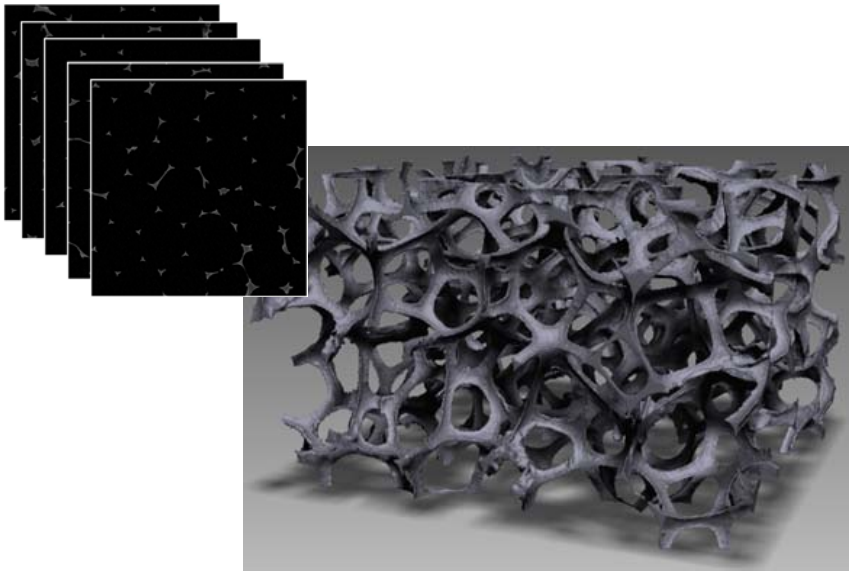
- Surface representations
  - Industrial design

Geometric Modeling

- Surface representations
  - Industrial design
  - Movies and animation

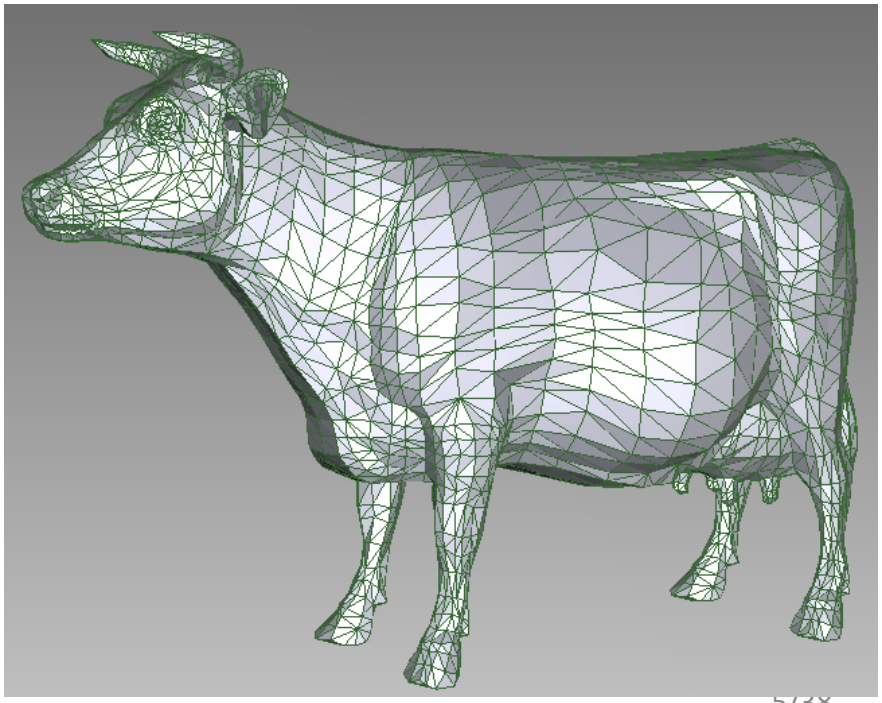
Geometric Modeling

- Surface representations
  - Industrial design
  - Movies and animation
- Surface reconstruction/Visualization



Polygonal Representations

- Stores the boundary of a solid
  - Geometry: vertex locations
  - Topology: connectivity information
    - Vertices
    - Edges
    - Faces



Polygonal Representations

- Constant time adjacency information
  - For each vertex,
    - Find edges/faces touching vertex
  - For each edge,
    - Find vertices/faces touching edge
  - For each face,
    - Find vertices/edges touching face

Polygonal Representations

- Typically assume the surface is *manifold*

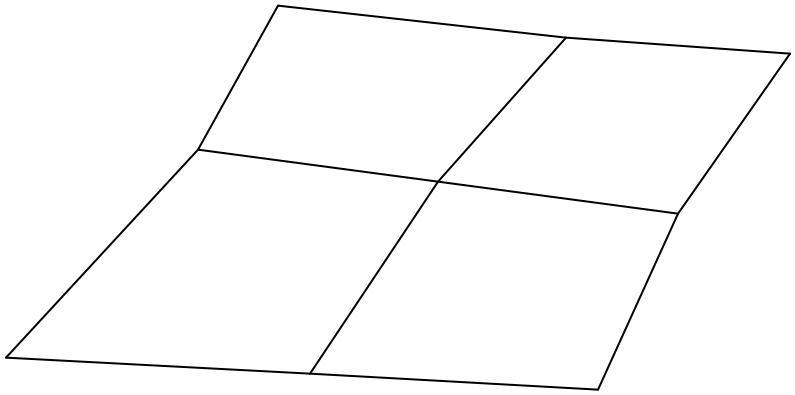
- A surface is *manifold* if, for every point, locally the surface is equivalent to an open disk

Polygonal Representations

- Typically assume the surface is *manifold*

- A surface is *manifold* if, for every point, locally the surface is equivalent to an open disk

Locally manifold

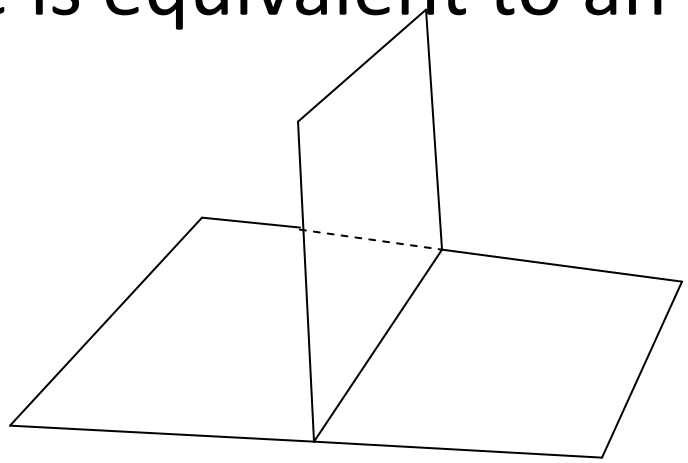


Polygonal Representations

- Typically assume the surface is *manifold*

- A surface is *manifold* if, for every point, locally the surface is equivalent to an open disk

Non-manifold edge

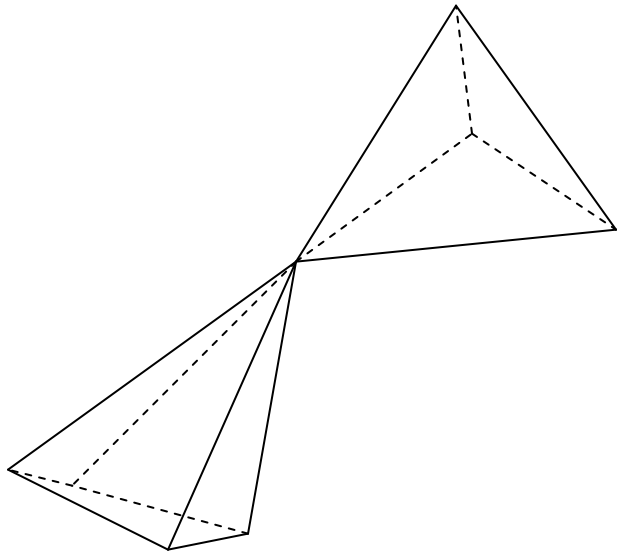


Polygonal Representations

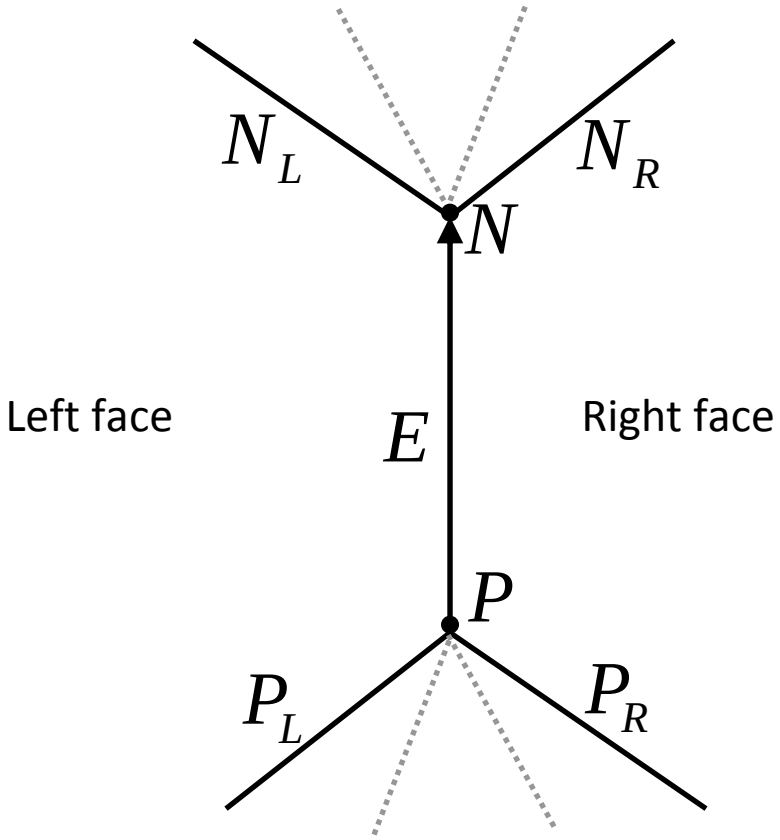
- Typically assume the surface is *manifold*

- A surface is *manifold* if, for every point, locally the surface is equivalent to an open disk

Non-manifold vertex

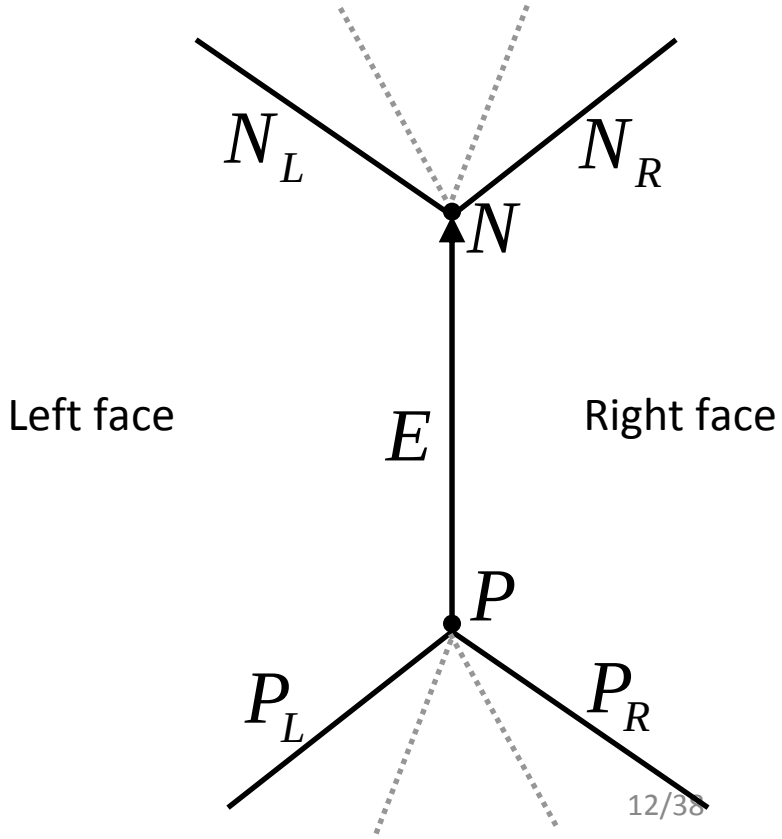


Winged Edge Data Structure



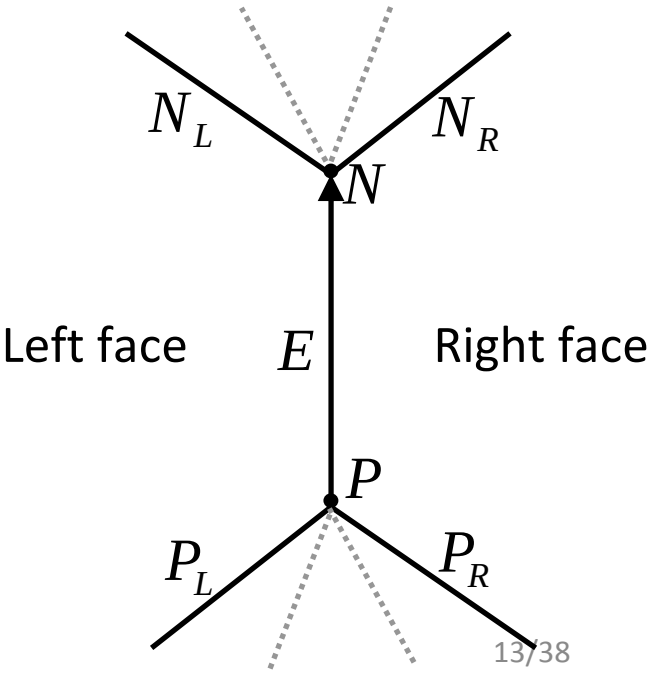
Winged Edge Data Structure

```
WingedEdge {
    WingedEdge nextLeft, nextRight,
                    prevLeft, prevRight;
    Face leftFace, rightFace;
    Vertex prevVert, nextVert;
}

Face {
    WingedEdge edge;
}

Vertex {
    WingedEdge edge;
}
```

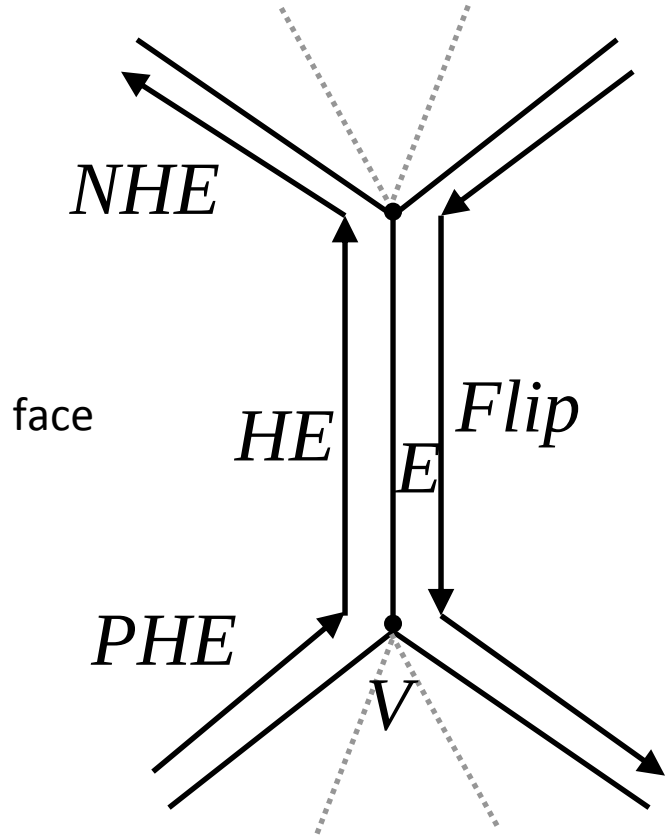


Winged Edge Data Structure

- Given a face, find all vertices touching that face
- Given a vertex, find all edge-adjacent vertices
- Given a face, find all adjacent faces

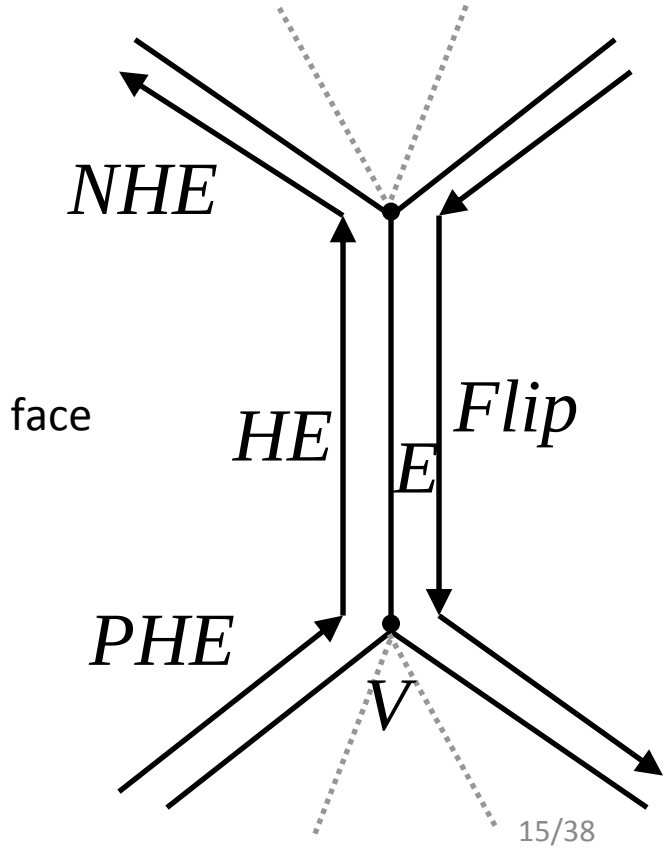


Half Edge Data Structure



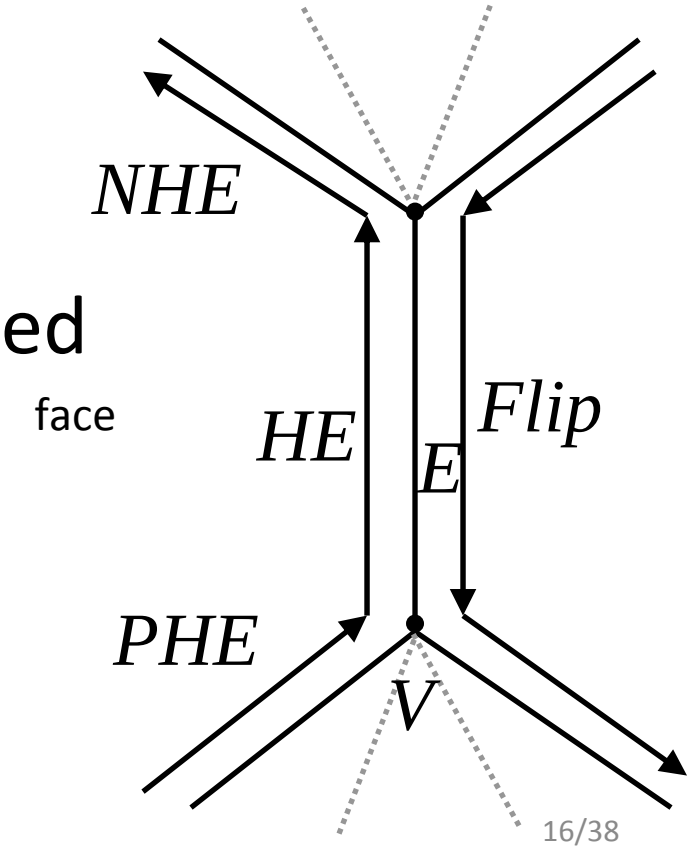
Half Edge Data Structure

```
HalfEdge {
    HalfEdge next, prev, flip;
    Face face;
    Vertex origin;
    Edge edge;
}

Face {
    HalfEdge edge; // part of this face
}

Vertex {
    HalfEdge edge;  // points away
}

Edge {
    HalfEdge he;
}
```



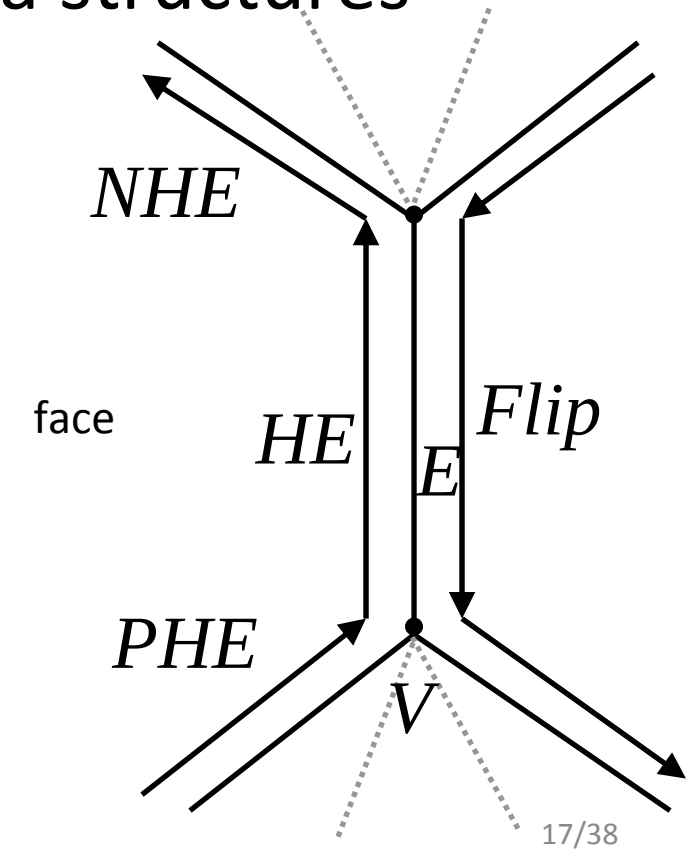
Half Edge Data Structure

- Very similar to WingedEdge Data Structure, but edges have unique orientation
- Slightly more storage
- Less conditional operations
- Assumes polygons are oriented



Building a Topological Data Structure

- Must connect adjacent edges/faces/vertices
- Edges are critical in most data structures



Euler Characteristic

$$V - E + F = 2 - 2G = \chi(G)$$

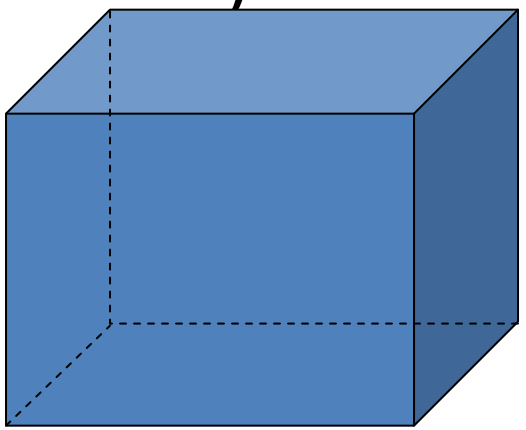
- *V*: number of vertices
- *E*: number of edges
- *F*: number of faces
- *G*: genus of surface (number of holes)

Euler Characteristic

$$V - E + F = 2 - 2G = \chi(G)$$

- *V*: number of vertices
- *E*: number of edges
- *F*: number of faces
- *G*: genus of surface (number of holes)

$$8 - 12 + 6 = 2 - 2 * 0$$

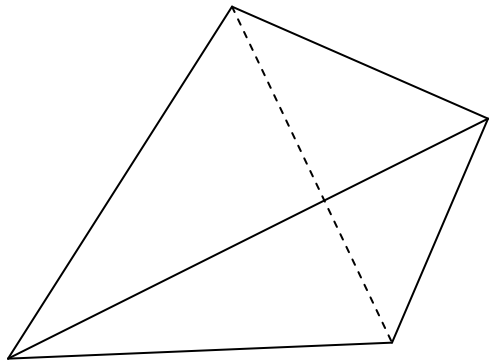


Euler Characteristic

$$V - E + F = 2 - 2G = \chi(G)$$

- *V*: number of vertices
- *E*: number of edges
- *F*: number of faces
- *G*: genus of surface (number of holes)

$$4 - 6 + 4 = 2 - 2 * 0$$

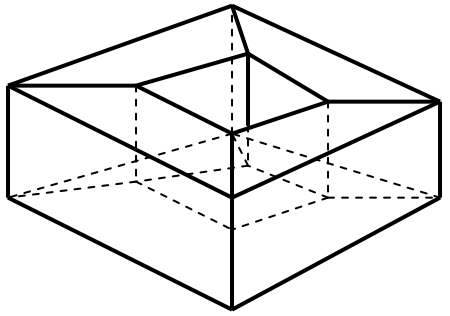


Euler Characteristic

$$V - E + F = 2 - 2G = \chi(G)$$

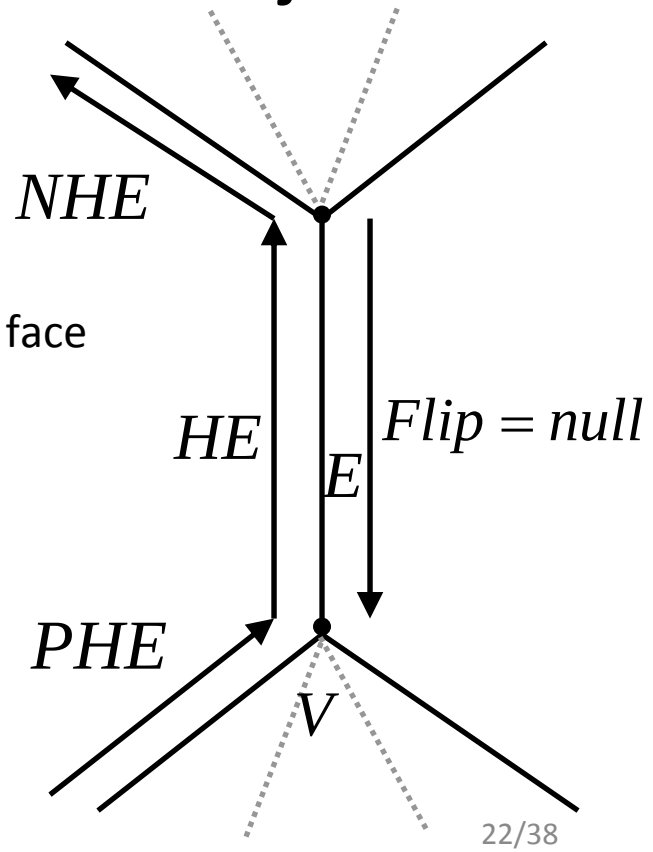
- *V*: number of vertices
- *E*: number of edges
- *F*: number of faces
- *G*: genus of surface (number of holes)

$$16 - 32 + 16 = 2 - 2*1$$



Topological Operations: Hole Filling

- Find a half-edge whose *flip* is null
- Use *next* and *flip* points to find *next* adjacent half-edge with null *flip*
- Repeat until back at original half-edge
- Create new half-edges along boundary and face containing those edges



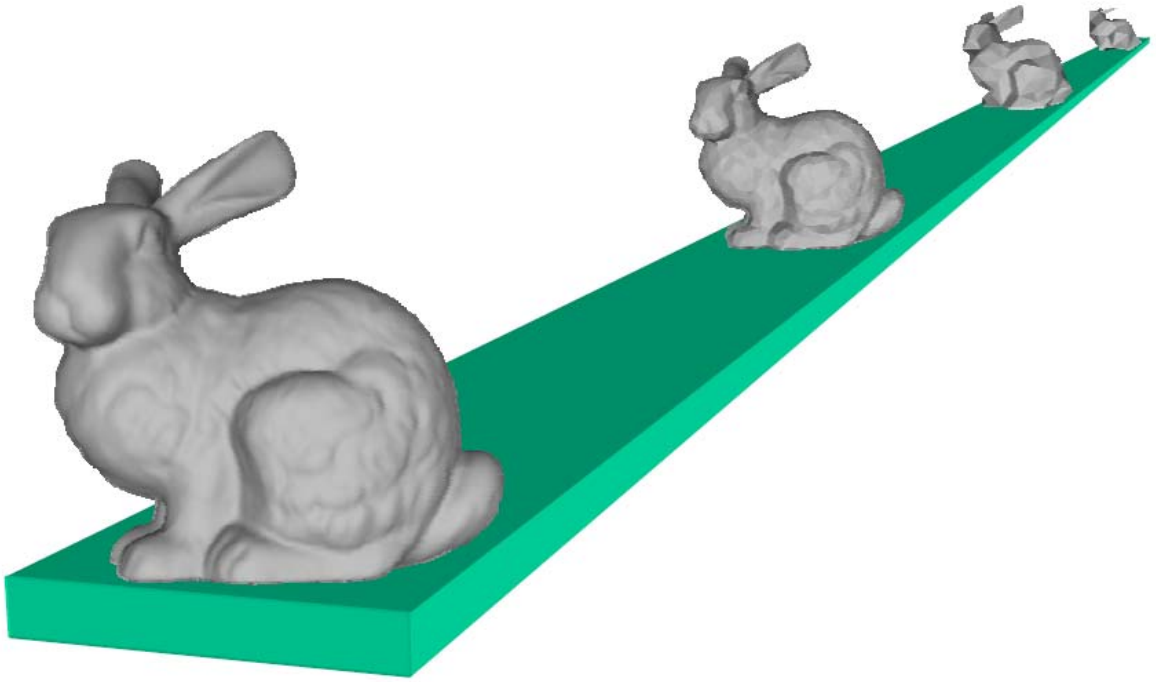
Mesh Simplification

- Create levels of detail (LOD) of objects:

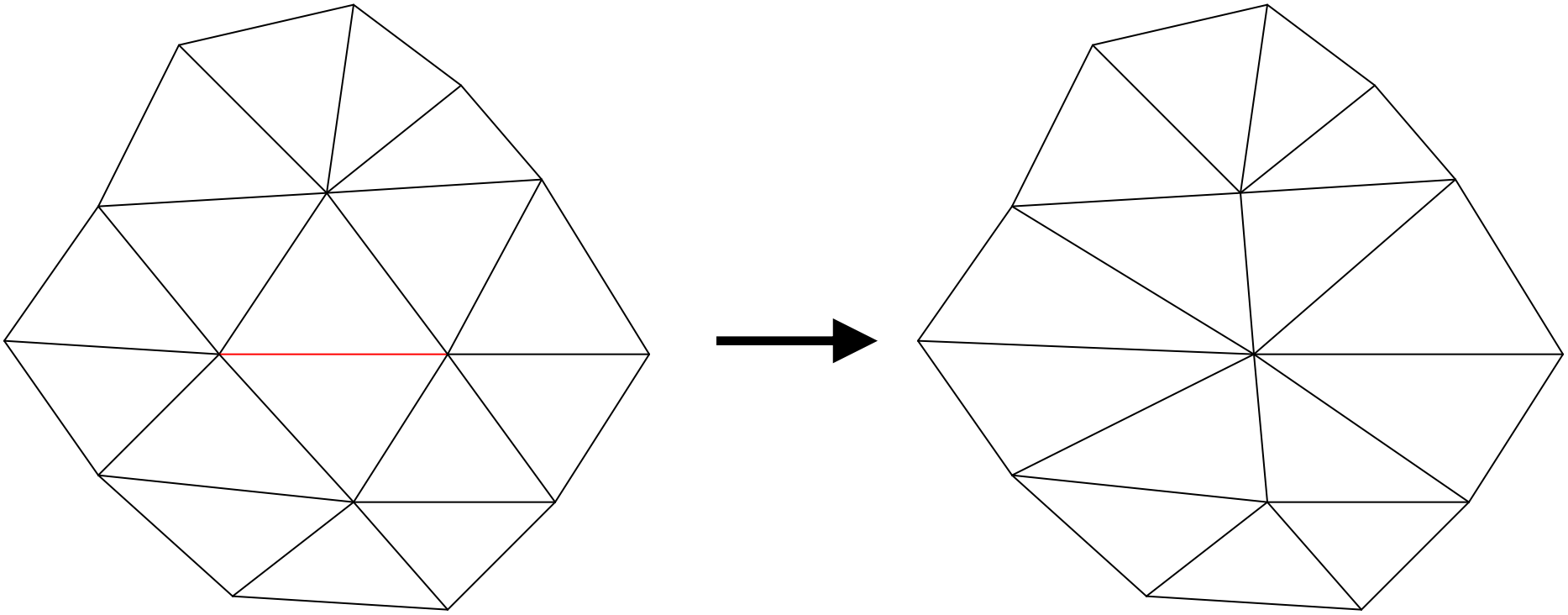
69,451 polys 2,502 polys 251 polys 76 polys

Level of Details (LOD)

- Distant objects use coarser LODs:

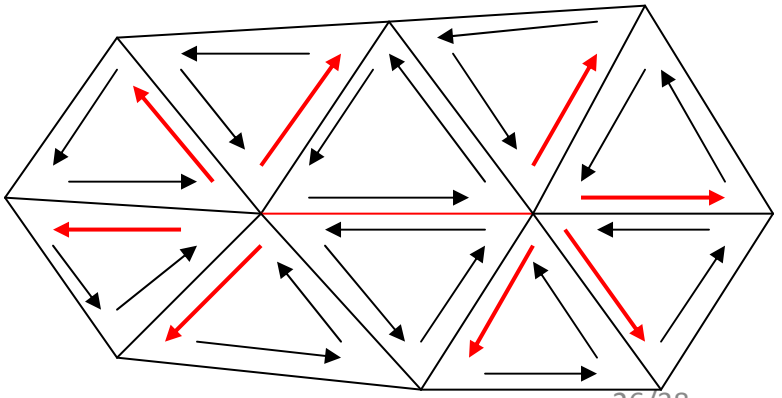


Topological Operations: Edge Collapse



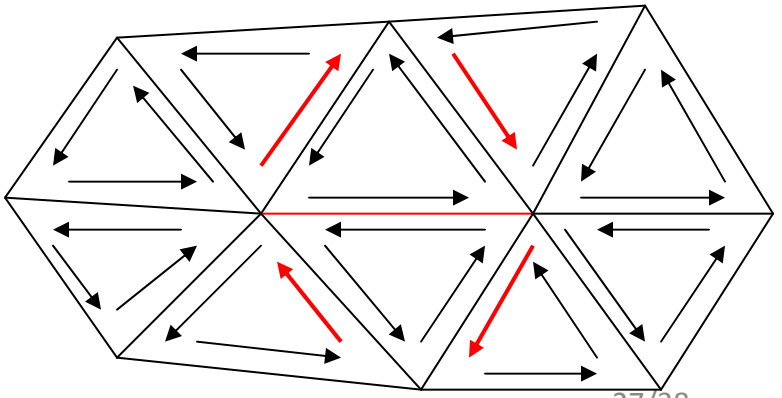
Topological Operations: Edge Collapse

- Set *origin* of all edges pointing outwards from two vertices to new vertex and vertex to one of the half-edges



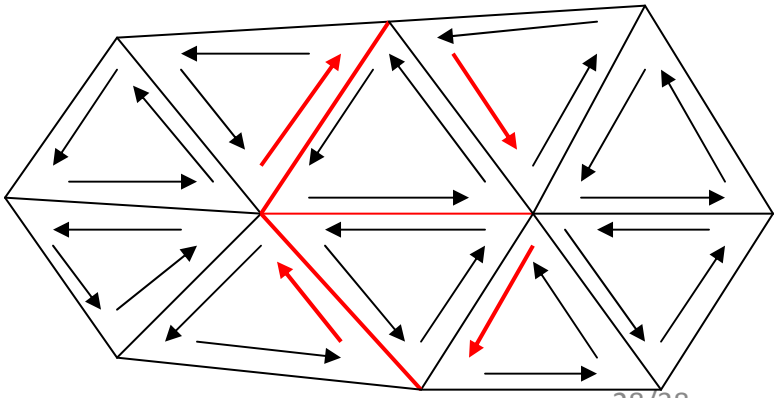
Topological Operations: Edge Collapse

- Set *origin* of all edges pointing outwards from two vertices to new vertex and vertex to one of the half-edges
- Set *flip* on outer edges to point to opposite edge



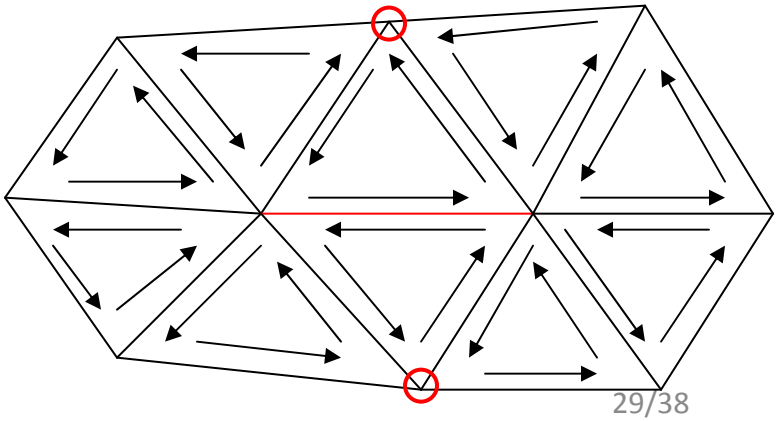
Topological Operations: Edge Collapse

- Set *origin* of all edges pointing outwards from two vertices to new vertex and vertex to one of the half-edges
- Set *flip* on outer edges to point to opposite edge
- Make sure half-edge pointer for edges point to correct half-edge



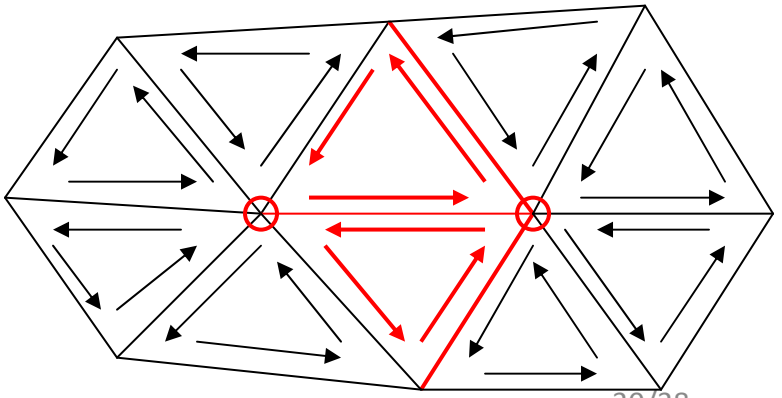
Topological Operations: Edge Collapse

- Set *origin* of all edges pointing outwards from two vertices to new vertex and vertex to one of the half-edges
- Set *flip* on outer edges to point to opposite edge
- Make sure half-edge pointer for edges point to correct half-edge
- Update half-edge for wing-vertices



Topological Operations: Edge Collapse

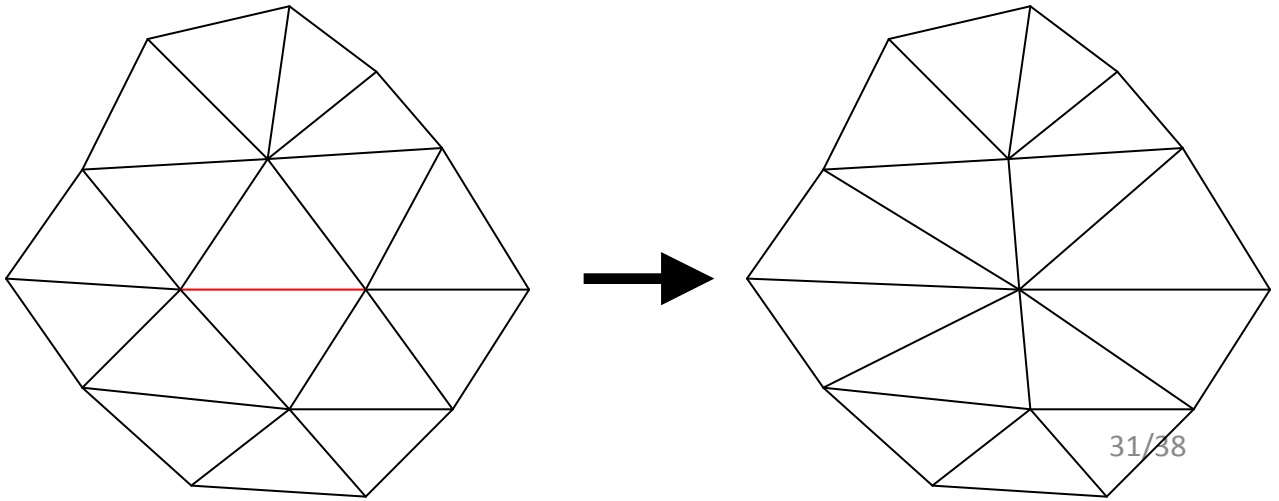
- Set *origin* of all edges pointing outwards from two vertices to new vertex and vertex to one of the half-edges
- Set *flip* on outer edges to point to opposite edge
- Make sure half-edge pointer for edges point to correct half-edge
- Update half-edge for wing-vertices
- Delete faces/edges/vertices



Topological Operation: Edge Collapse

- Edge collapse preserves topology as long as the local Euler characteristic of the surface remains the same

$$V - E + F = \chi(G)$$



Topological Operation: Edge Collapse

- Edge collapse preserves topology as long as the local Euler characteristic of the surface remains the same

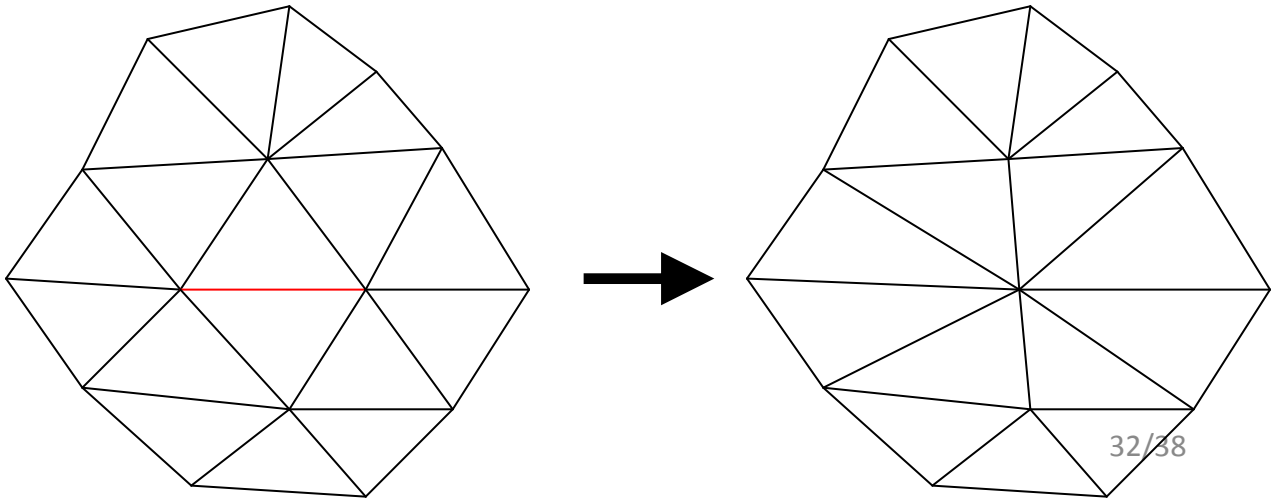
$$V - E + F = \chi(G)$$

$$\hat{V} - \hat{E} + \hat{F} = (V - 1) - (E - 3) + (F - 2) = V - E + F$$

$$\hat{V} = V - 1$$

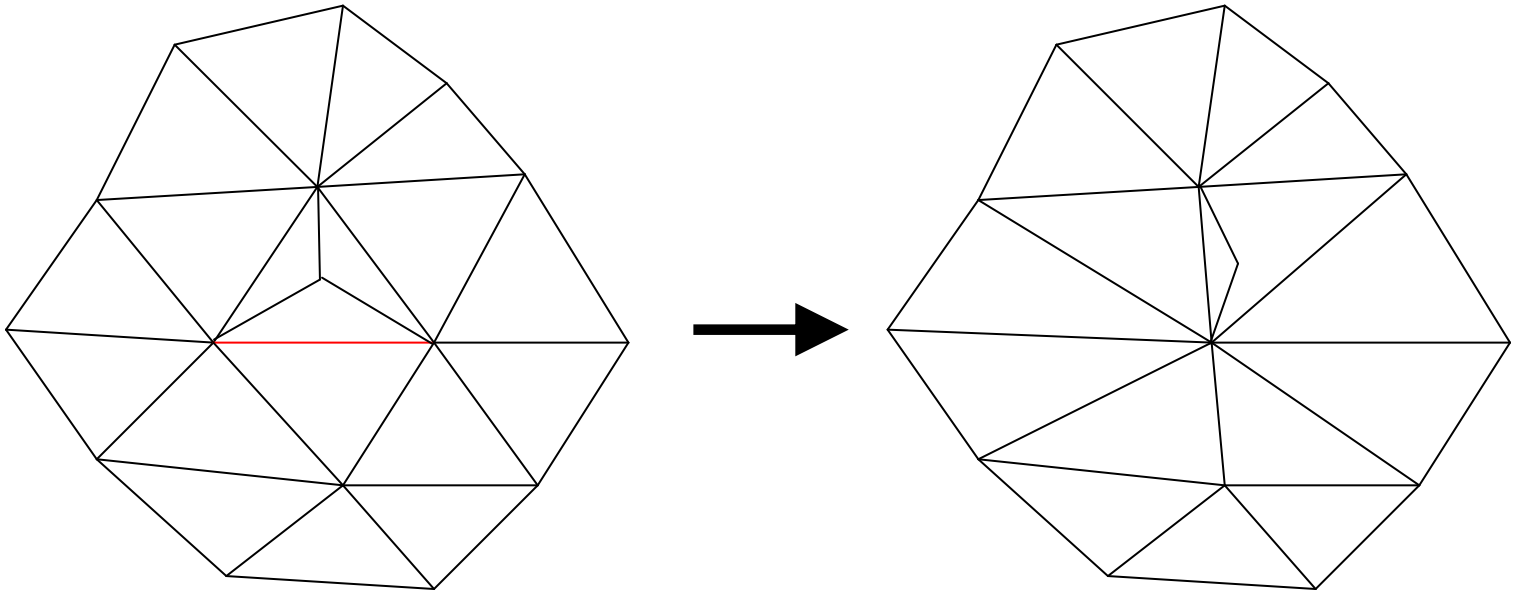
$$\hat{E} = E - 3$$

$$\hat{F} = F - 2$$



Topological Operation: Edge Collapse

- Edge collapse does NOT always preserve topology



Detecting Unsafe Edge Collapses

- Let $N(v)$ be the set of vertices edge-adjacent to v
- Safe to collapse if $\left|N(v_1) \cap N(v_2)\right| = 2$