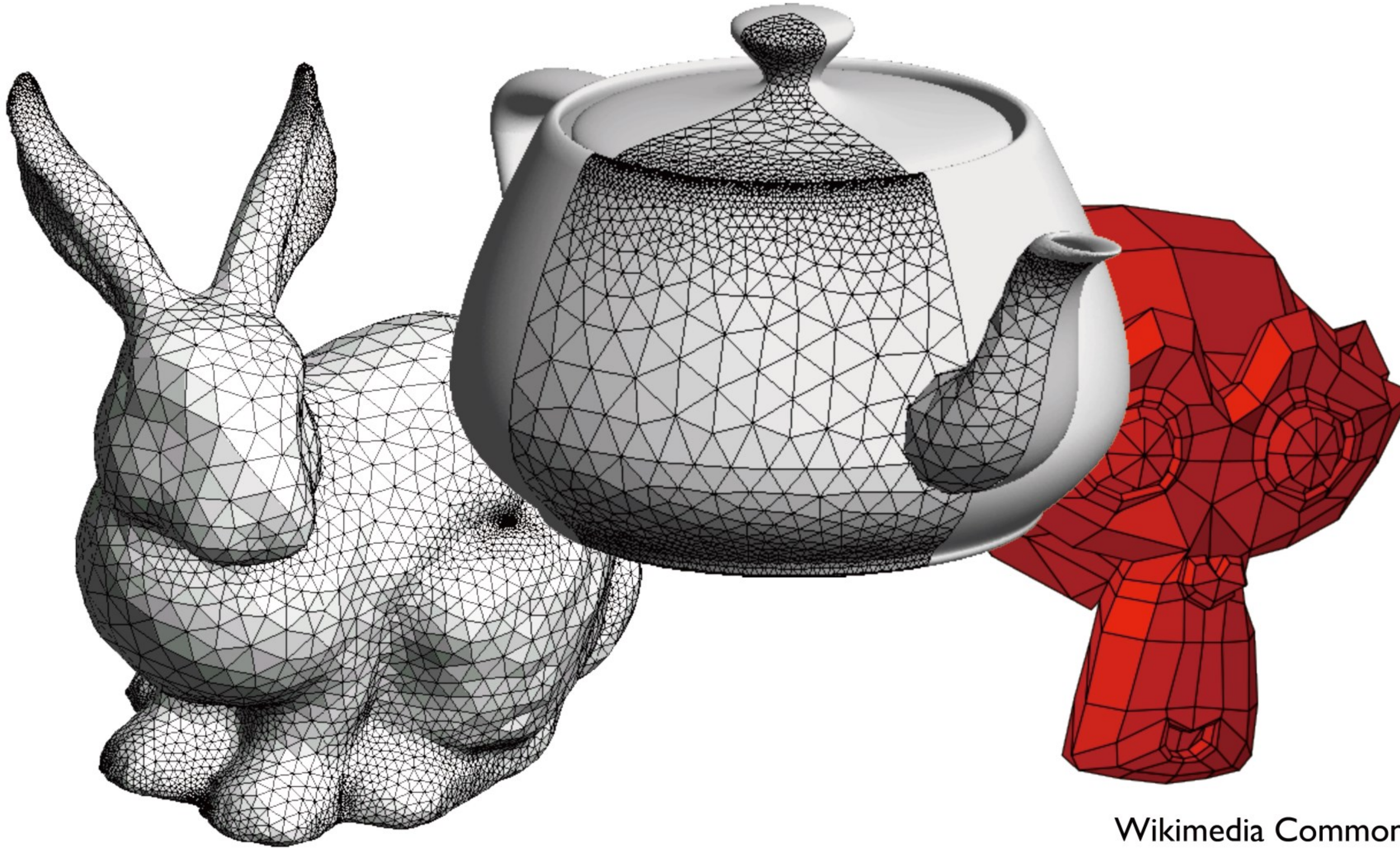
CS100433
Polygonal Mesh

Junqiao Zhao 赵君峤
Department of Computer Science and Technology
College of Electronics and Information Engineering
Tongji University

What is a mesh?

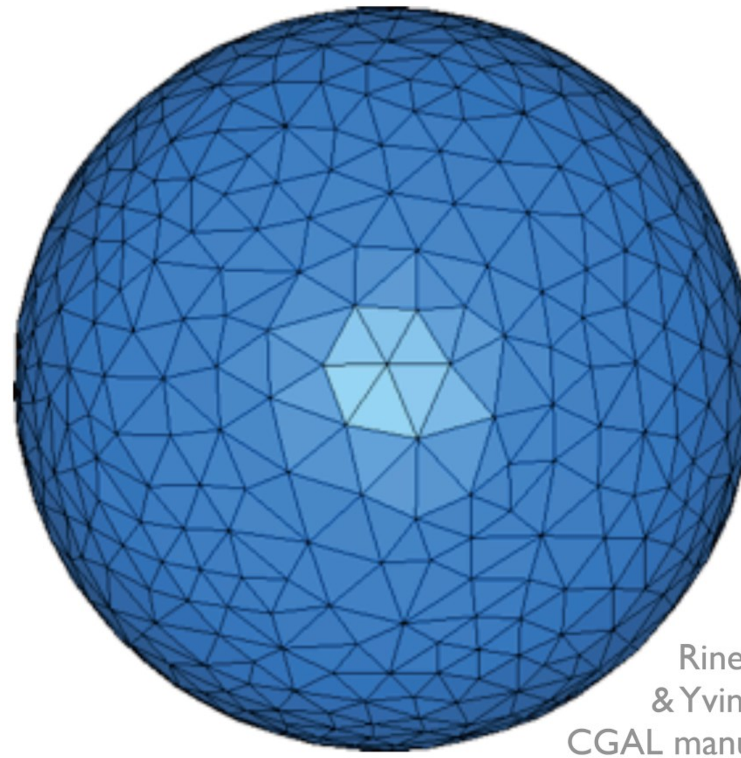


Wikimedia Commons

What is a mesh?

Andrzej Barabasz

Spheres

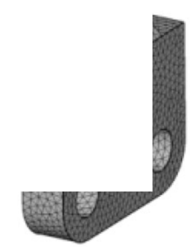


Rineau
& Yvinec
CGAL manual

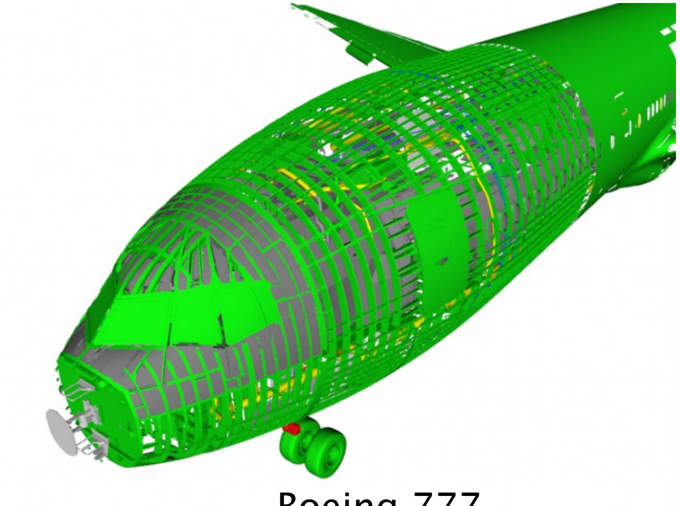
Approximate sphere

3D polygonal meshes

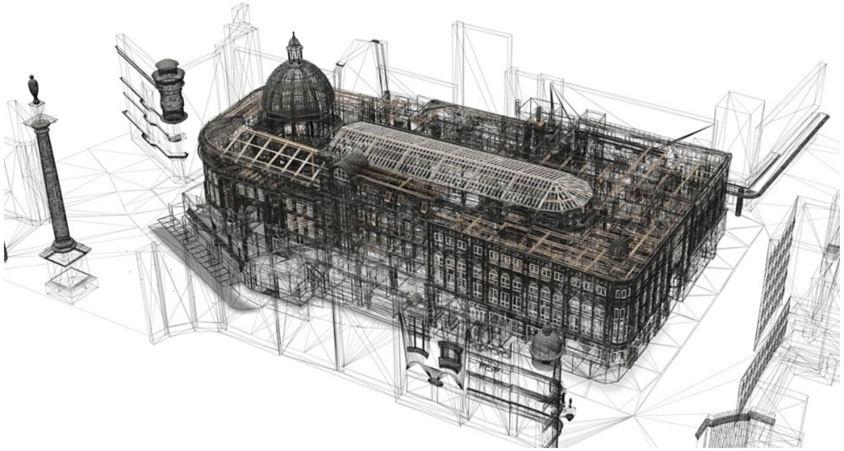
– Representing a 2D surface embedded in $\mathbb{R}^3$ by using a set of polygons



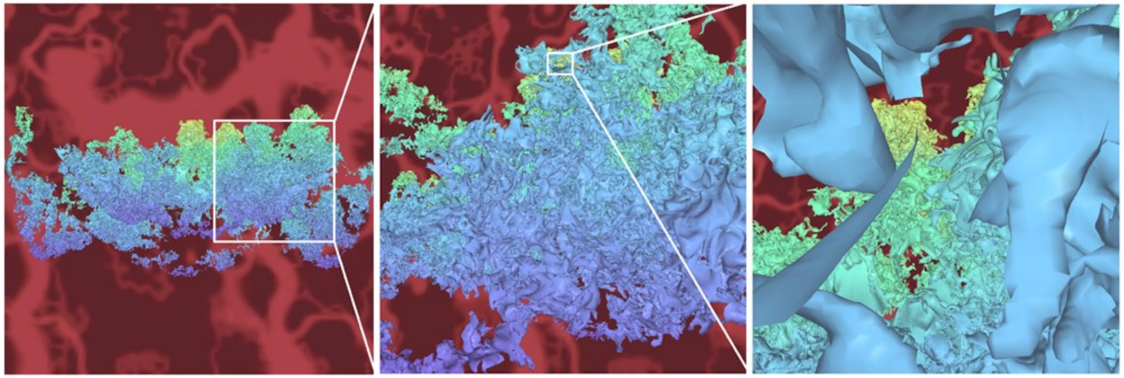
Application Domain



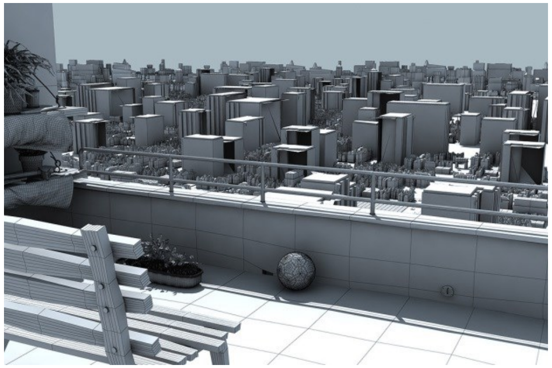
Boeing 777
(718k meshes 470 million triangles)



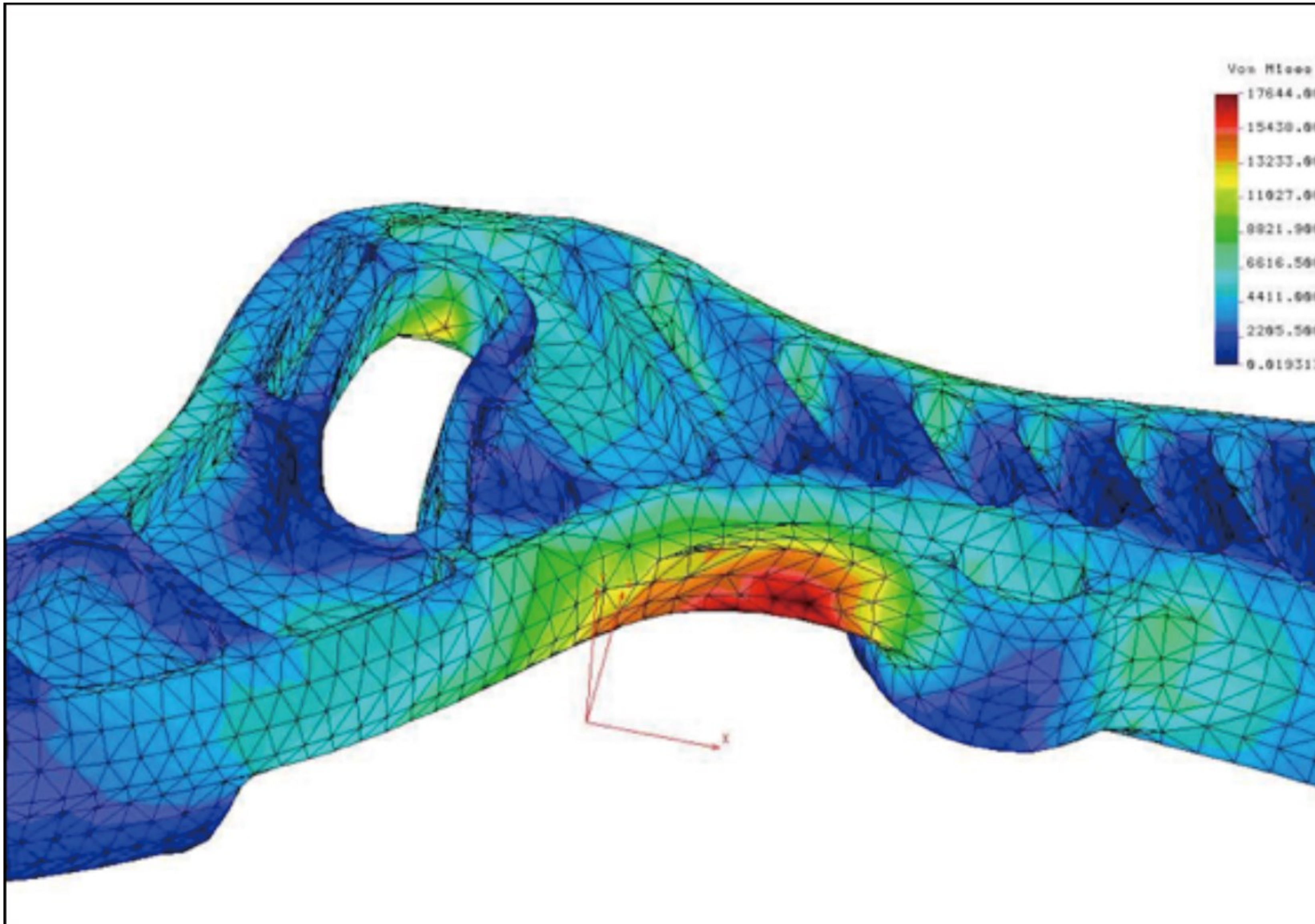
Architectural model
(eBIM)



Iso-surface (1000 million triangles)



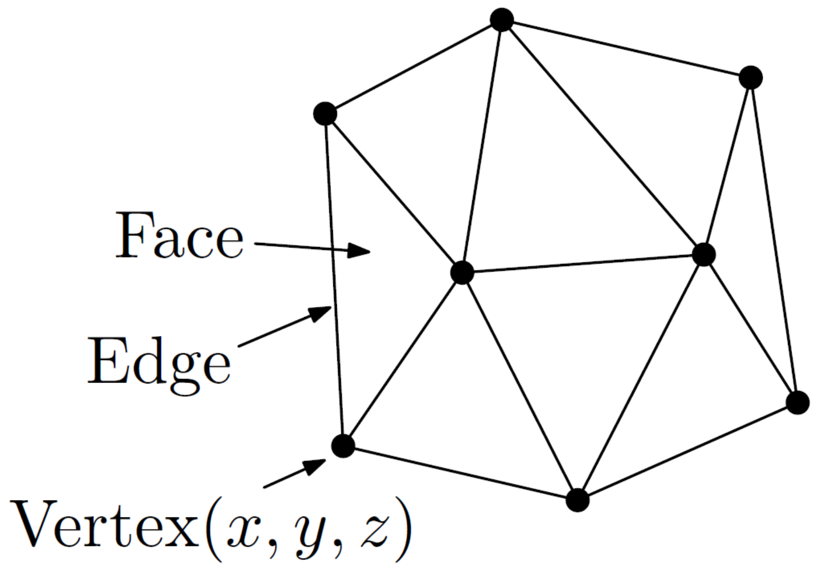
Massive urban scene

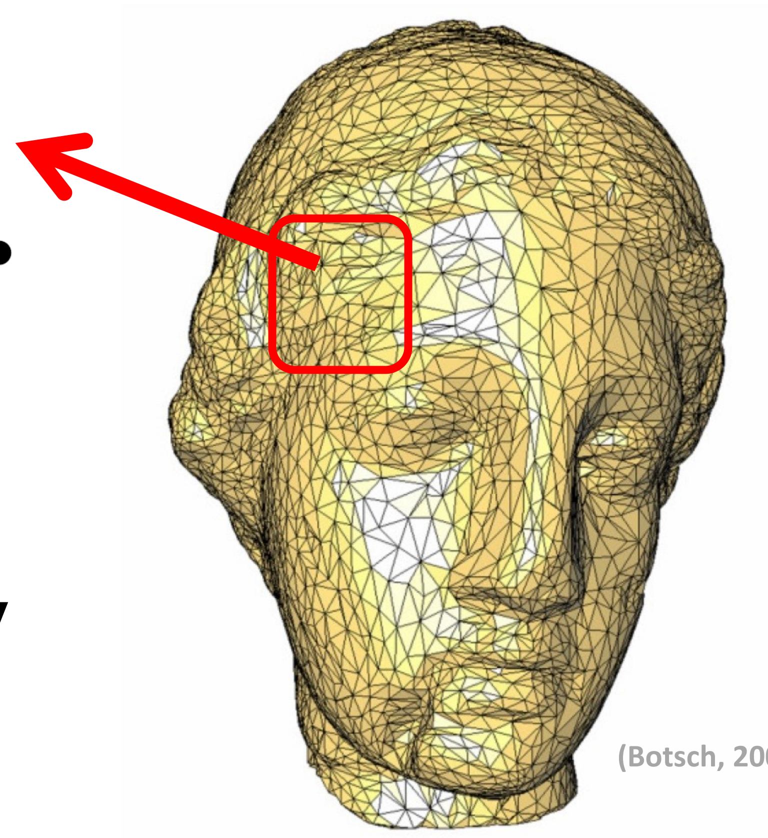


PATRIOT Engineering

finite element analysis

Combinatorial structure



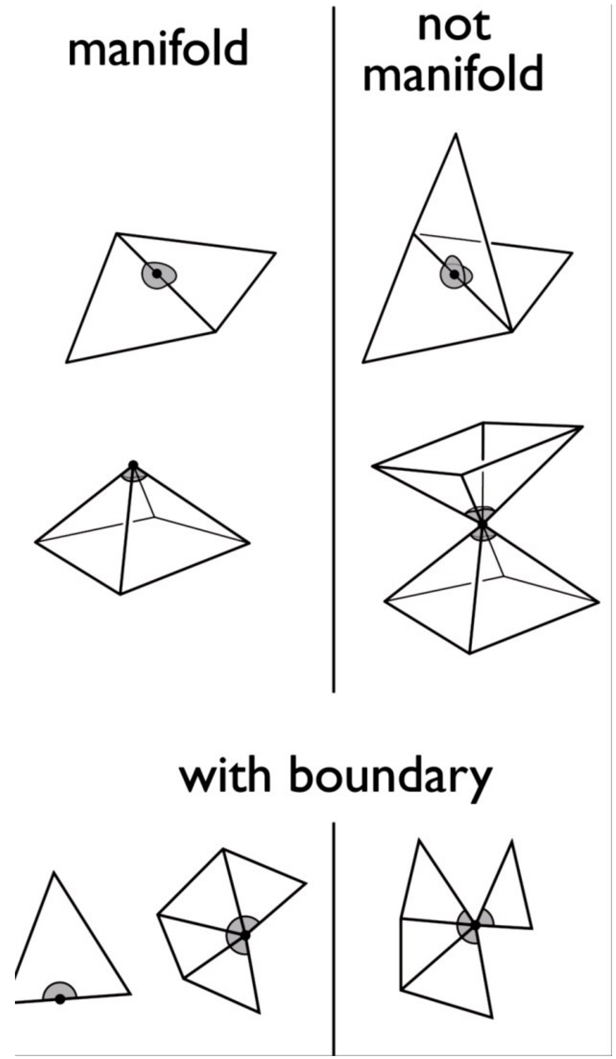


(Botsch, 2007)

- Geometry &Topology

Mesh validity

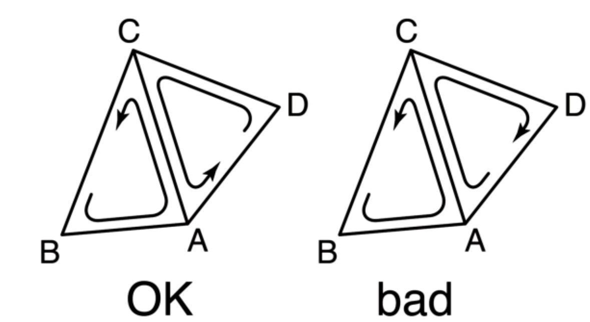
- manifold
  - no edge is shared by more than two polygons; the faces adjacent to a vertex form a single ring
  - edge points: each edge must have exactly 2 triangles
  - vertex points: each vertex must have one loop of triangles
- manifold with boundary
  - weaken rules to allow boundaries
  - The faces adjacent to a vertex form an incomplete ring

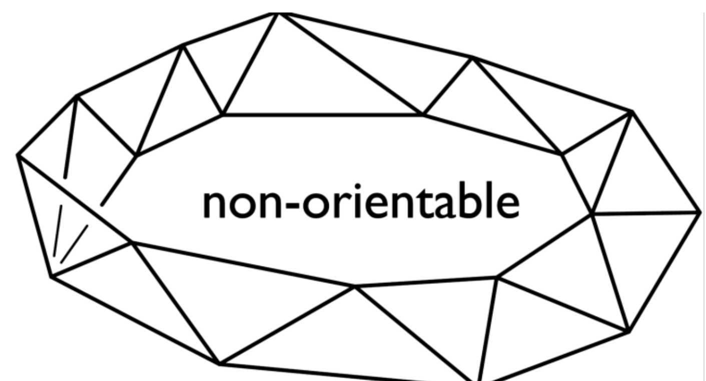


(Steve Marschner 2013)

Topological validity

- Consistent orientation
  - Which side is the “front” or “outside” of the surface and which is the “back” or “inside?”
  - rule: you are on the outside when you see the vertices in counter-clockwise order
  - in mesh, neighboring triangles should agree about which side is the front!
  - caution: not always possible

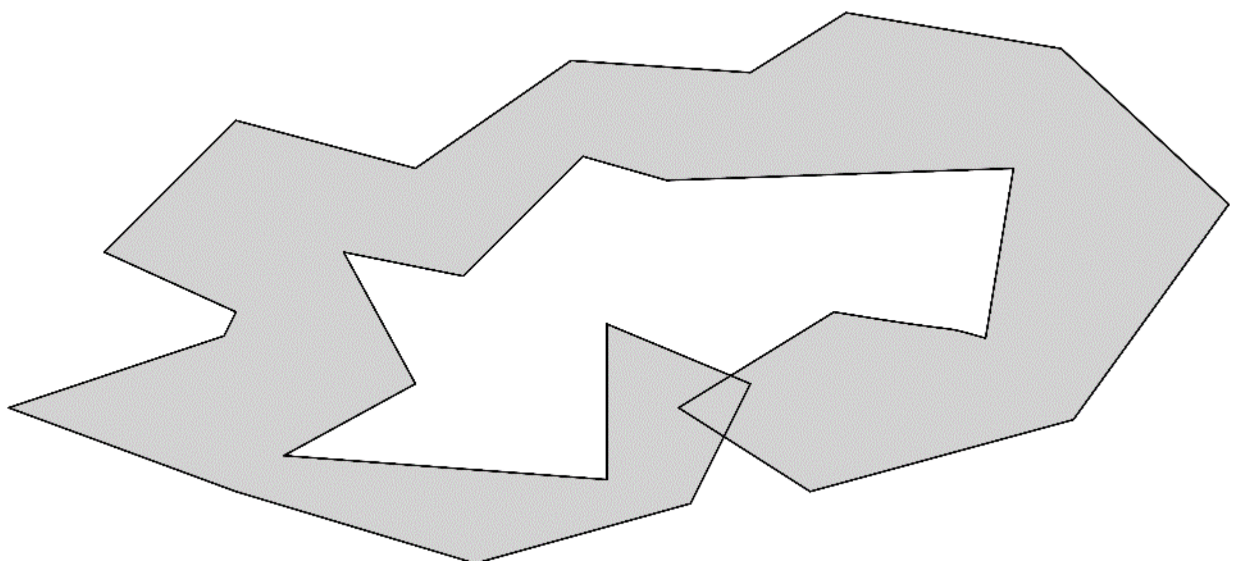




(Steve Marschner 2013)

Geometric Validity

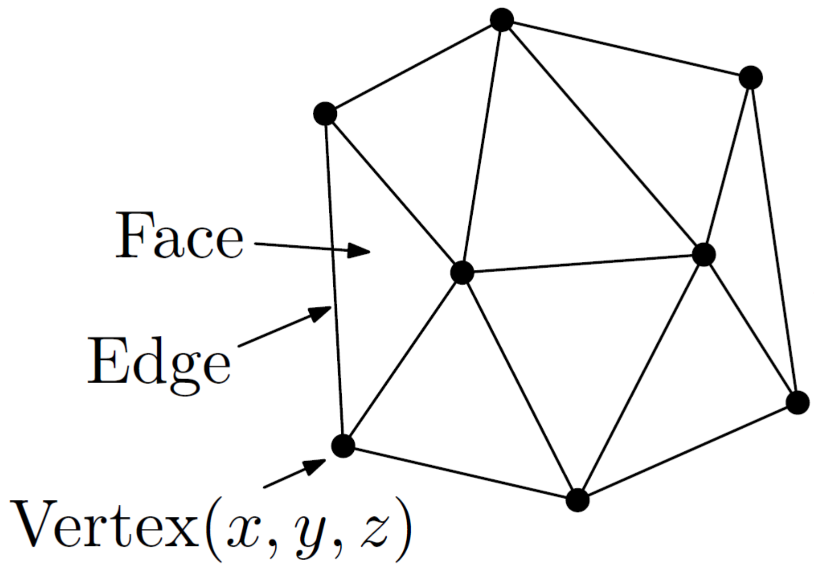
- generally want non-self-intersecting surface
- hard to guarantee in general
  - because far-apart parts of mesh might intersect

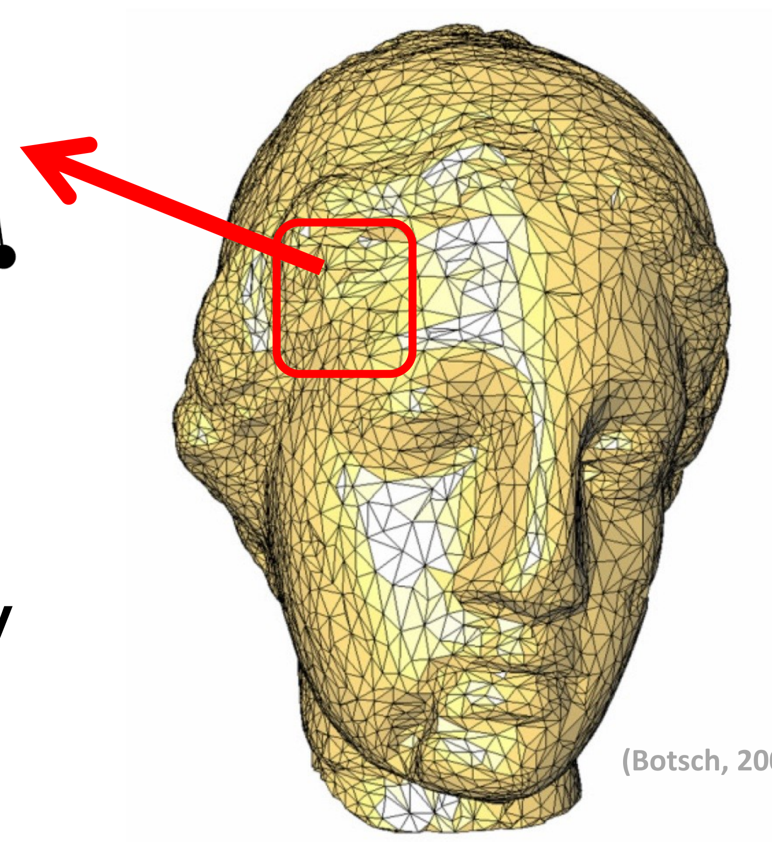


(Steve Marschner 2013)

- questions?

Combinatorial structure





(Botsch, 2007)

- Geometry &Topology

Requirements for mesh data structures

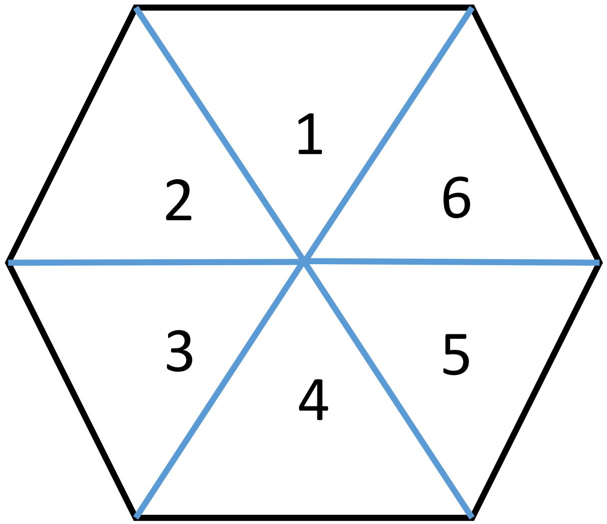
- Compactness
- Efficiency for rendering
- Efficient support for queries, e.g.
  - Given a face, find its vertices
  - Given a face, find neighboring faces
  - Given a vertex, find faces touching it
  - Given a vertex, find neighboring vertices
  - Given an edge, find vertices and faces it touches

Polygon soup

- Polygons specified one-by-one with no explicit information on shared vertices
- Unorganized
- Think about VBO and glDrawElements(GL_TRIANGLES…)

Mesh queries

- Iterate over all elements of a certain type, visiting each only once
- Iterate over all elements (faces, edges, vertices) adjacent to an element



Mesh data structures

- Separate triangles
- Indexed triangle set
- Triangle strips and triangle fans
- Triangle-neighbor data structure
- Winged-edge data structure
- Half-edge data structure

Independent triangles

- Each triangle lists vertex coordinates
  - Redundant vertices
  - No adjacency information

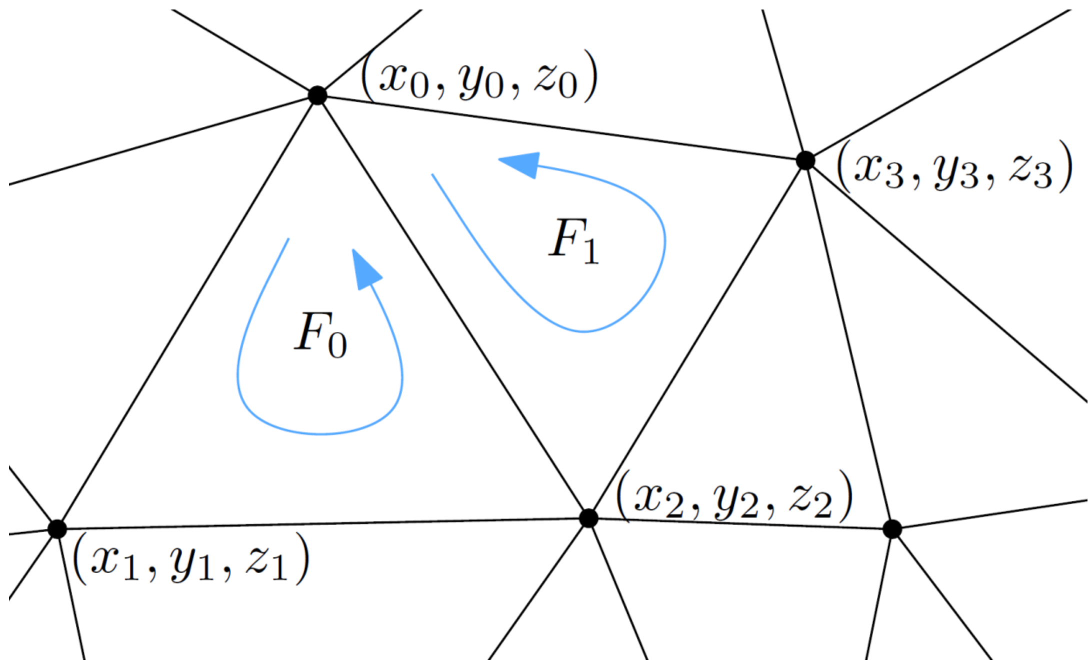
Face Table

F_0:

$(x_0, y_0, z_0), (x_1, y_1, z_1), (x_2, y$

F_1:

$(x_2, y_2, z_2), (x_3, y_3, z_3), (x_0, y$



Note the CCW orientation!

Indexed triangle set

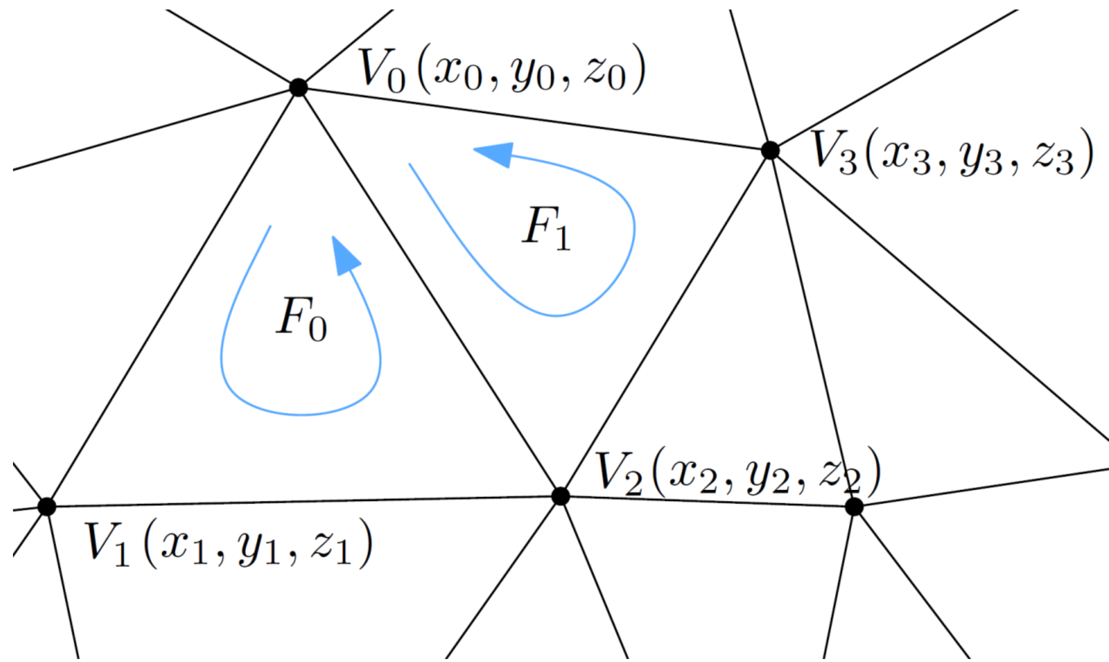
- Each face lists vertex references
  - Shared vertices
  - Still no adjacency information

Vertex Table

$V_0: (x_0, y_0, z_0)$
$V_1: (x_1, y_1, z_1)$
$V_2: (x_2, y_2, z_2)$
$V_3: (x_3, y_3, z_3)$

Face Table

$F_0: V_0, V_1, V_2$
$F_1: V_2, V_3, V_0$

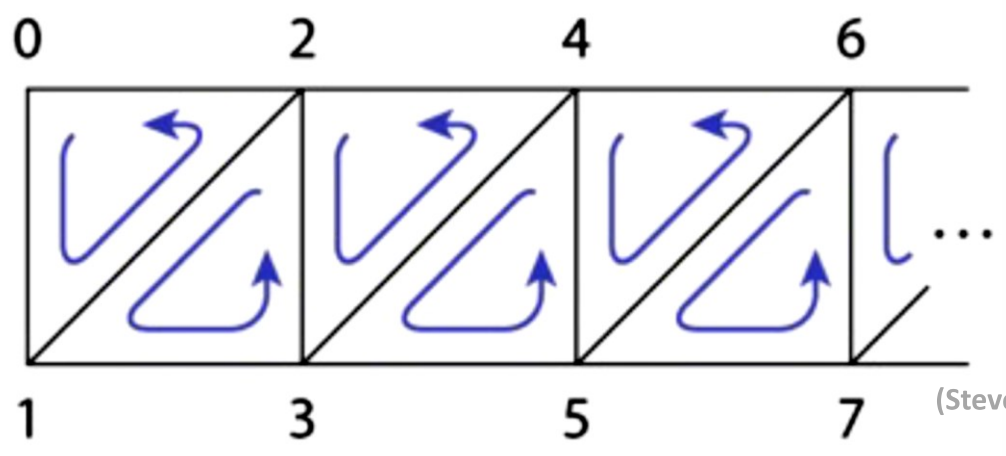


OBJ data format

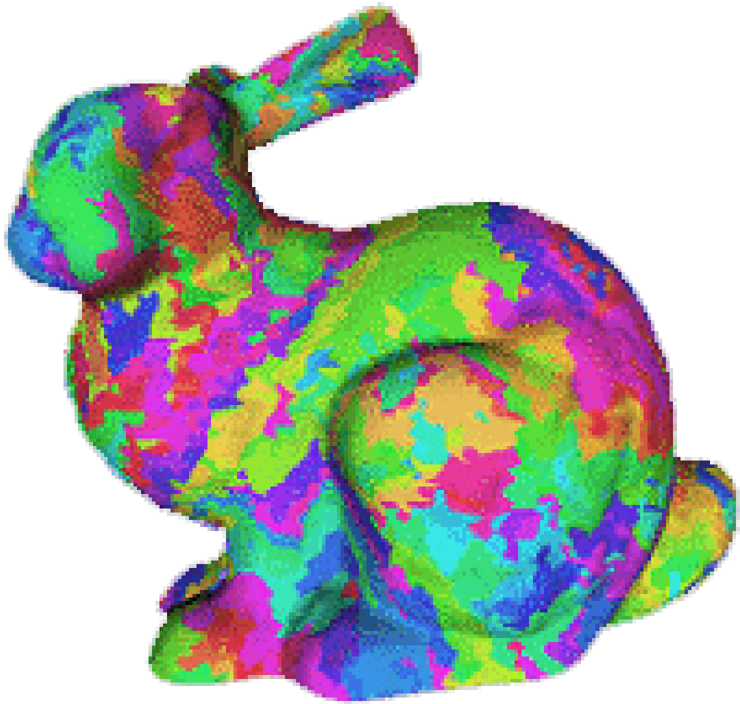
- *.obj Wavefront obj
  - #vertices
  - v x, y, z
  - …
  - #faces
  - f i1, i2, i3, …, in
- Note there can be more information such as normal, texture etc.
  - #normal
  - vn nx, ny, nz
  - vt u, v, w
  - …

Triangle strips

- Take advantage of the mesh property
  - each triangle is usually adjacent to the previous
  - let every vertex create a triangle by reusing the second and third vertices of the previous triangle
  - e. g., 0, 1, 2, 3, 4, 5, 6, 7, … leads to (0 1 2), (2 1 3), (2 3 4), (4 3 5), (4 5 6), (6 5 7), …
  - for long strips, this requires about one index per triangle



(Steve Marschner 2013)

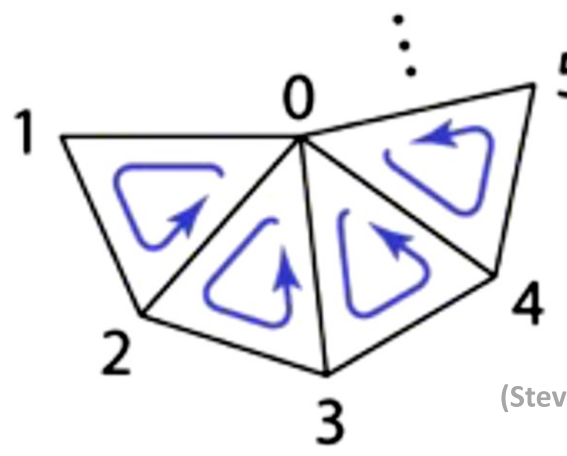


J. El-Sana

M Isenburg

Triangle fans

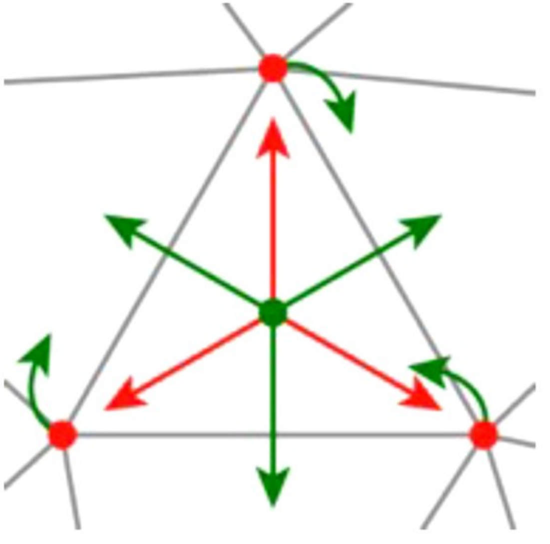
- Same idea as triangle strips, but keep oldest rather than newest
  - every sequence of three vertices produces a triangle
  - e. g., 0, 1, 2, 3, 4, 5, … leads to (0 1 2), (0 2 3), (0 3 4), (0 3 5), …
  - Memory considerations exactly the same as triangle strip



(Steve Marschner 2013)

Triangle-neighbor data structure

- Extension to indexed triangle set
- Triangle points to its three neighboring triangles
- Vertex points to a single neighboring triangle
- Can now enu around a vertex



(Steve Marschner 2013)

Winged-edge data structure

- Based on edges
- Store all vertex, face, and edge adjacencies

Edge Adjacency Table

$e_0: V_0, V_1; F_0, \emptyset; \emptyset, e_2, e_1, \emptyset$

$\boxed{e_2: V_2, V_0; F_0, F_1; e_3, e_1, e_0, e_4}$

$e_1: V_1, V_2; F_0, \emptyset; \emptyset, e_0, e_2, \emptyset$

Face Adjacency Table

$\boxed{F_0: V_0, V_1, V_2; F_1, \emptyset, \emptyset; e_0, e_1, e_2}$

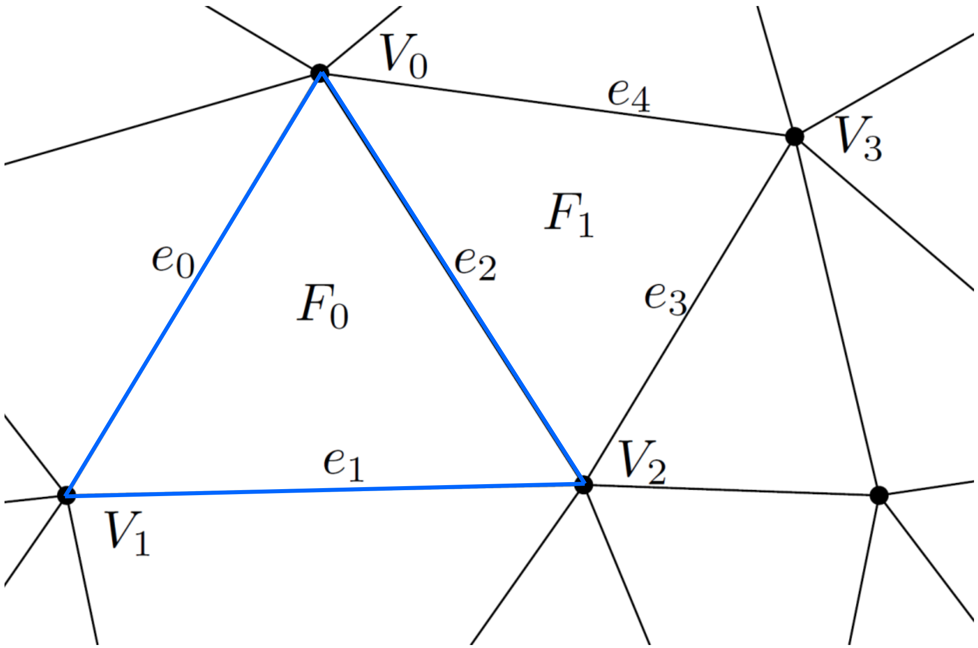
$F_1: V_2, V_3, V_0; F_0, \emptyset, \emptyset; e_2, e_3, e_4$

Vertex Adjacency Table

$V_0: V_1, V_2, V_3; F_0, F_1; e_0, e_2, e_4$

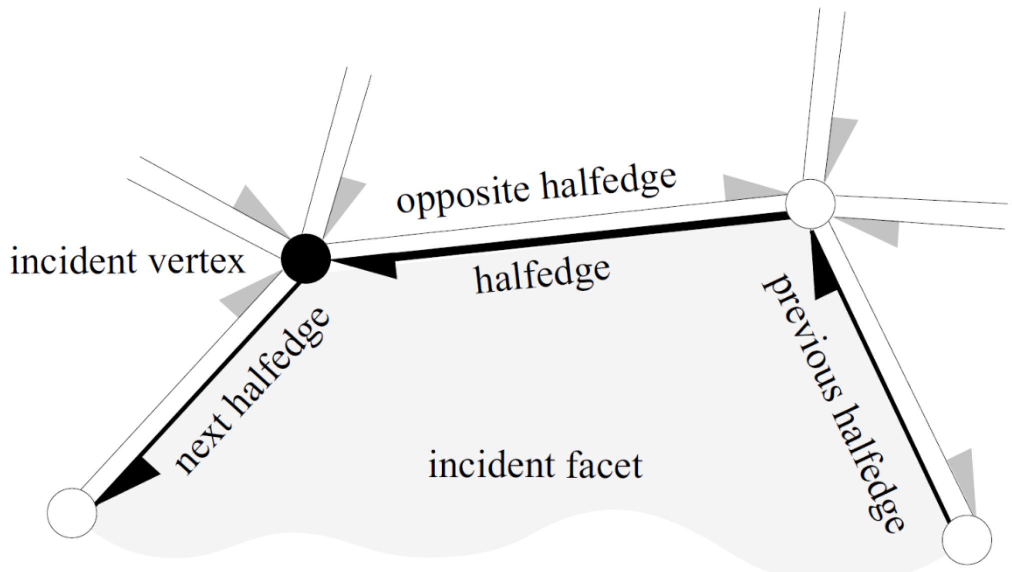
$V_1: V_2, V_0; F_0; e_1, e_0$

...



Half-edge data structure

- A half-edge data structure is an **edge-centered** data structure capable of maintaining incidence information of vertices, edges and faces
- Instead of a single edge, 2 **oriented** “half edges”



(CGAL, 2013)

Half-edge data structure

Half Edge Table

$h_0: V_2 ;\ F_1; h_1; h_3$

$\boxed{h_2: V_0 ;\ F_1; h_0; h_9}$

$h_3: V_0 ; F_0; h_4; h_0$

. . .

Face Table

F_0: h_3

F_1: h_2

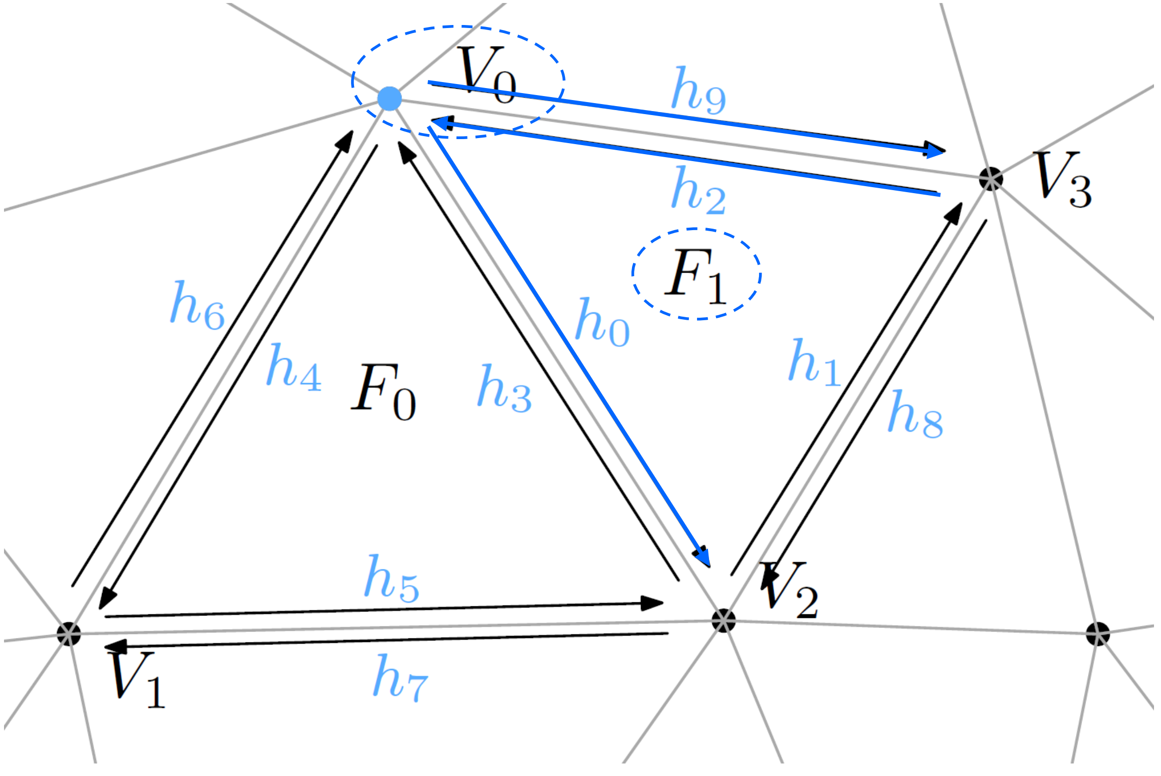
. . .

Vertex Table

V_0: h_2

V_1: h_4

. . .



Queries

- Examples:

  `h = h->next`

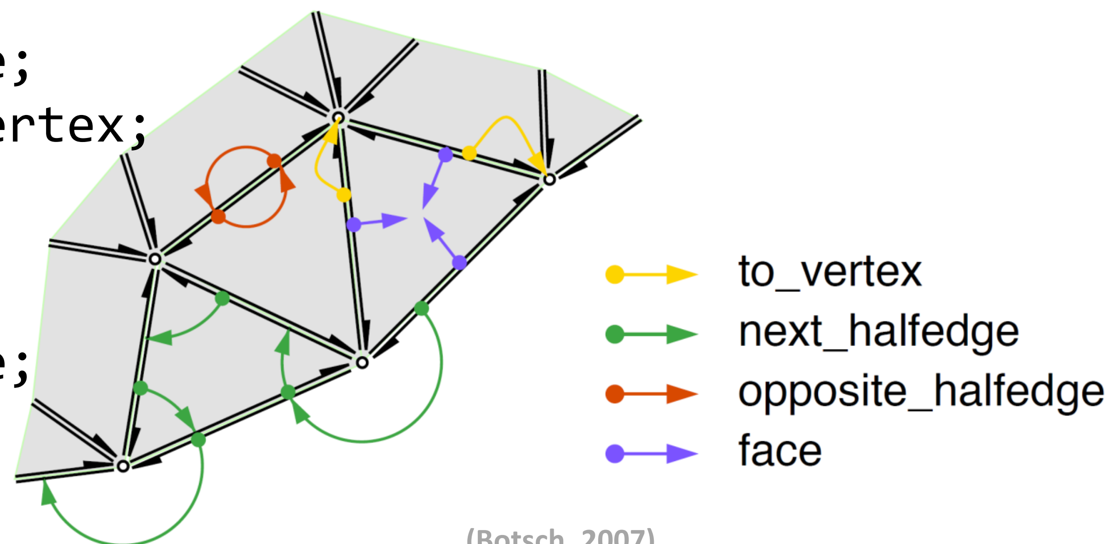
  cycle counterclockwise around the face and traverse all halfedges incident to this facet

  `h = h->next->opposite`

  cycle clockwise around the vertex and traverse all halfedges incident to this vertex

Half-edge data structure

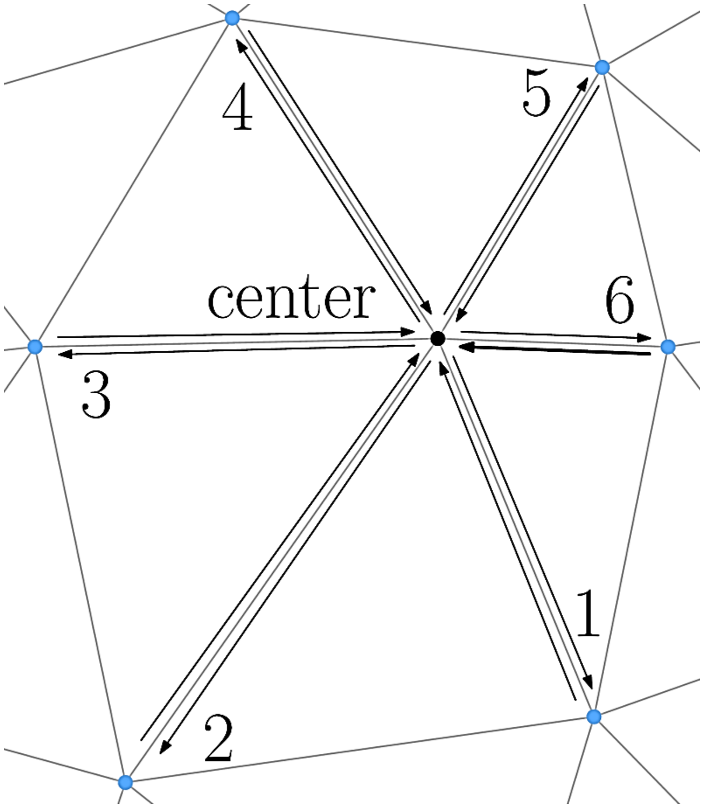
```
struct Halfedge {
        Halfedge* next;
        Halfedge* opposite;
        Face* incident_face;
        Vertex* incident_vertex;
}

struct Face{
        Halfedge * halfedge;
}

struct Vertex{
        Halfedge* incident_halfedge;
}
```



(Botsch, 2007)

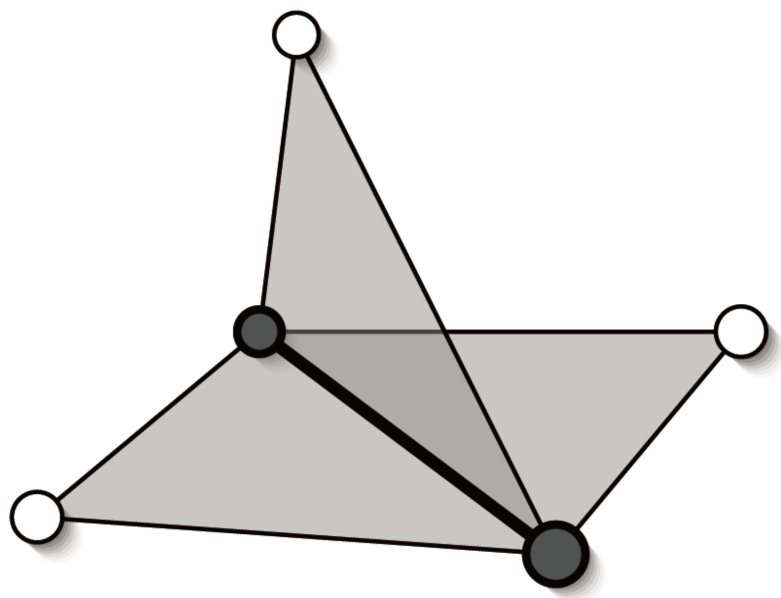
Question

- How to enumerate the STAR of a vertex?



Think

- Can you represent this shape with half-edge data structure?



Summary

- For rendering purpose
  - Triangle strips/fans
  - Independent triangles
  - Indexed triangle sets
- For query purpose
  - Winged-edge
  - Half-edge

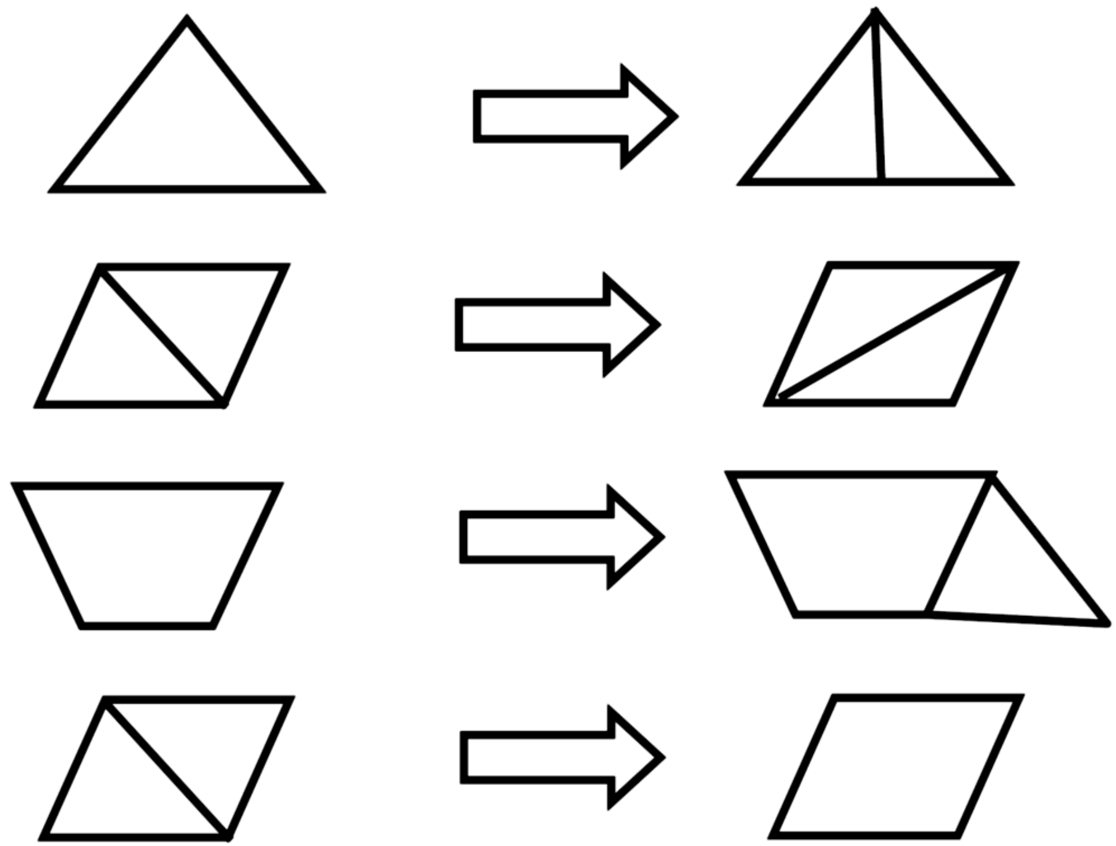
Mesh operations

Refinement

Edge Flips

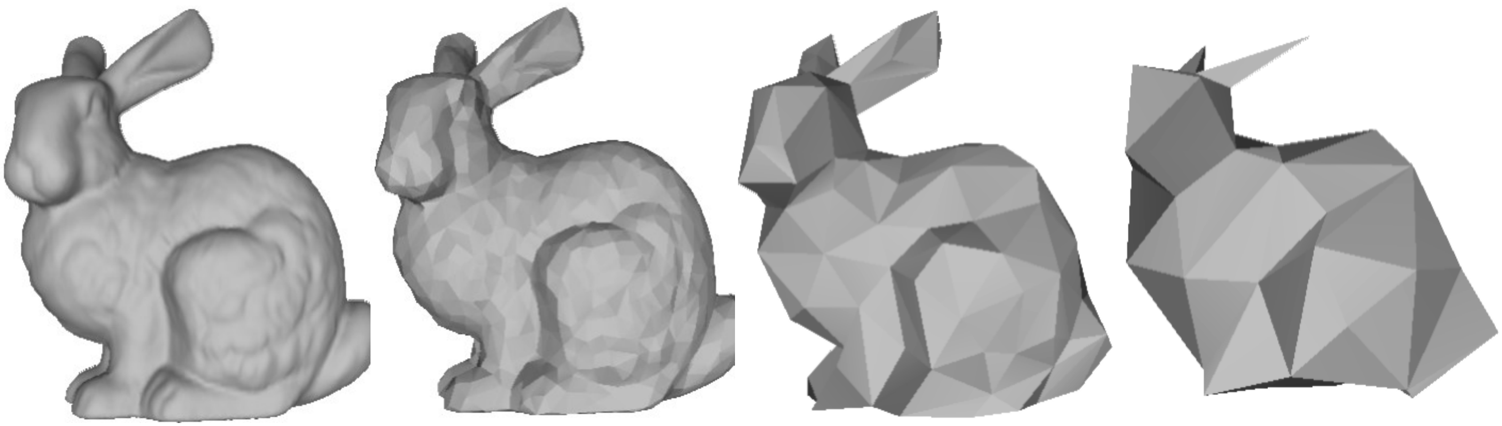
Face Addition/
Deletion

Face Merge



Level of Detail (LoD)

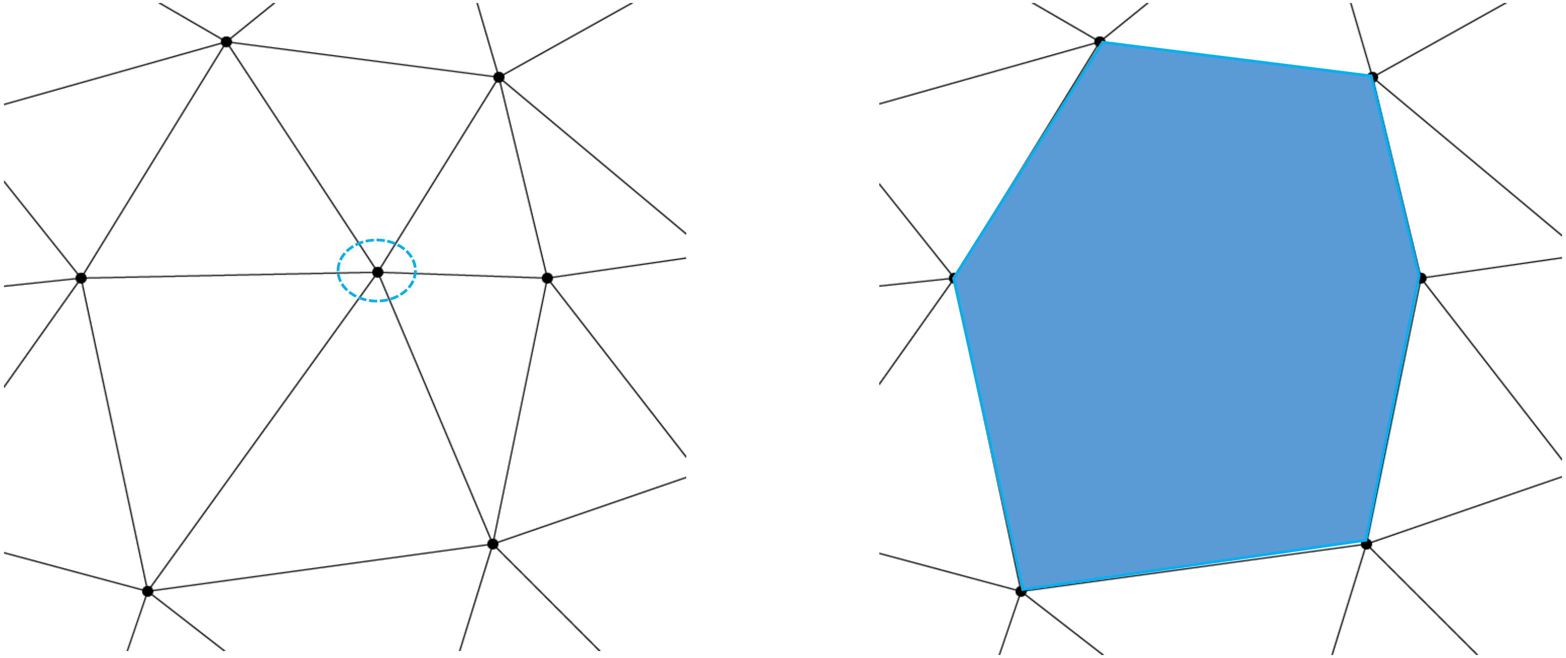
- Requirements
  - Decimate vertices/triangles
  - Preserve features



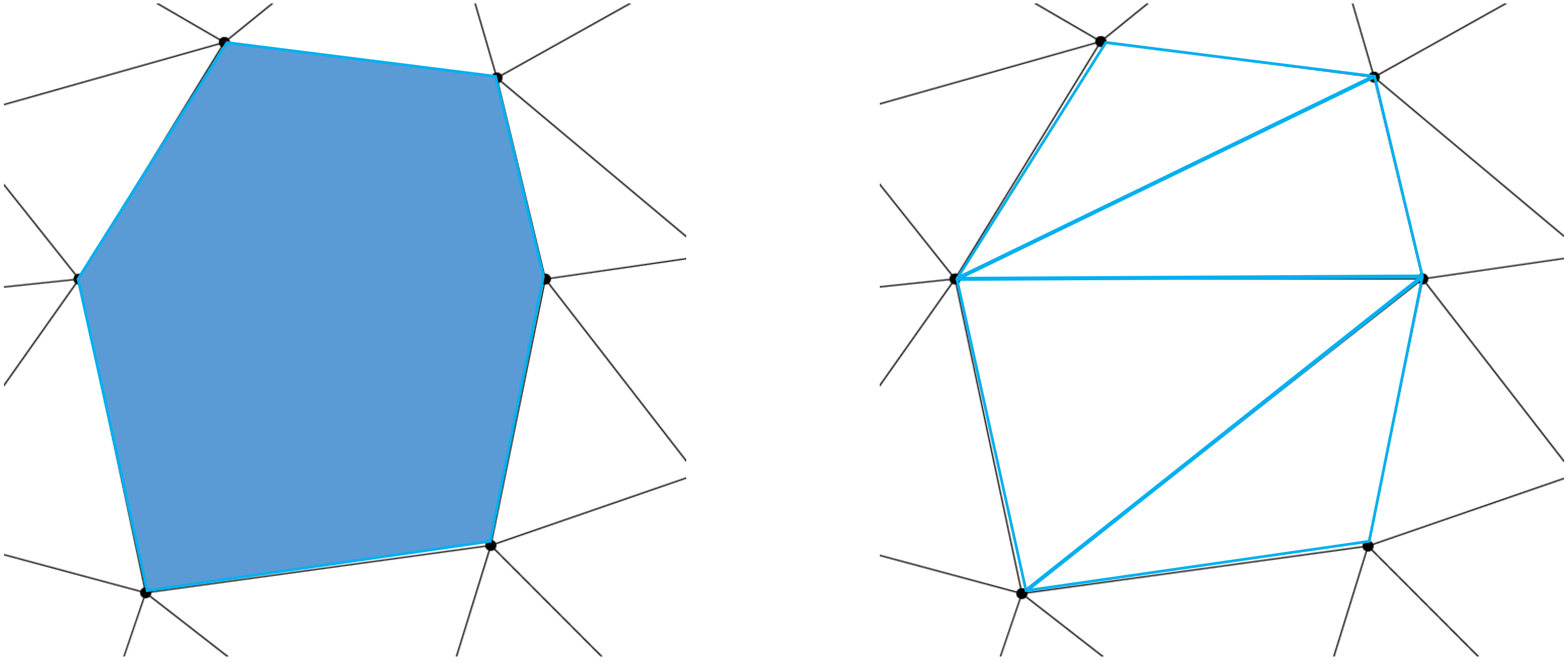
69,451 triangles   2,502 triangles   251 triangles   76 triangles

(Courtesy Stanford 3D Scanning Repository)

Vertex Removal

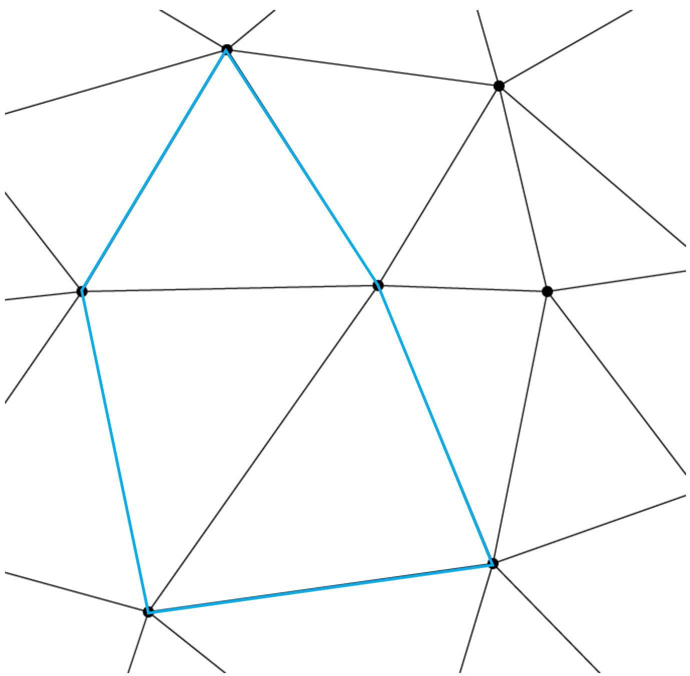


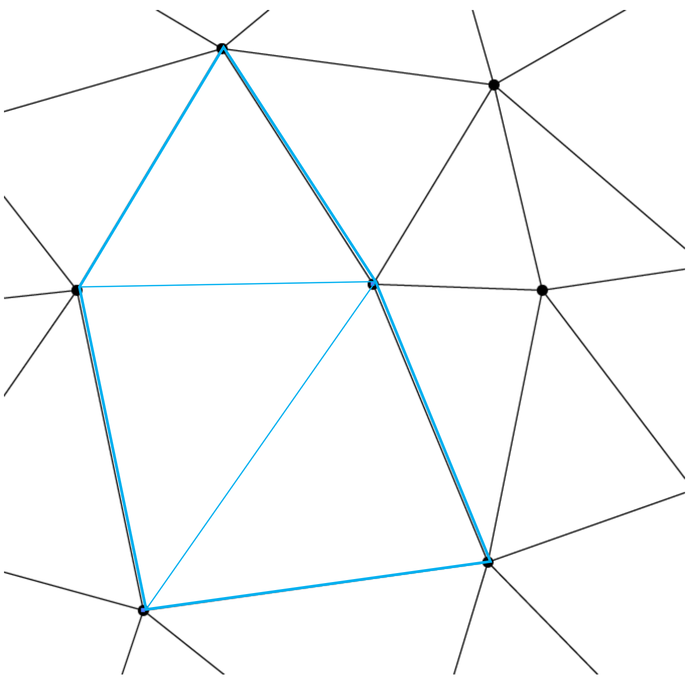
Vertex Removal



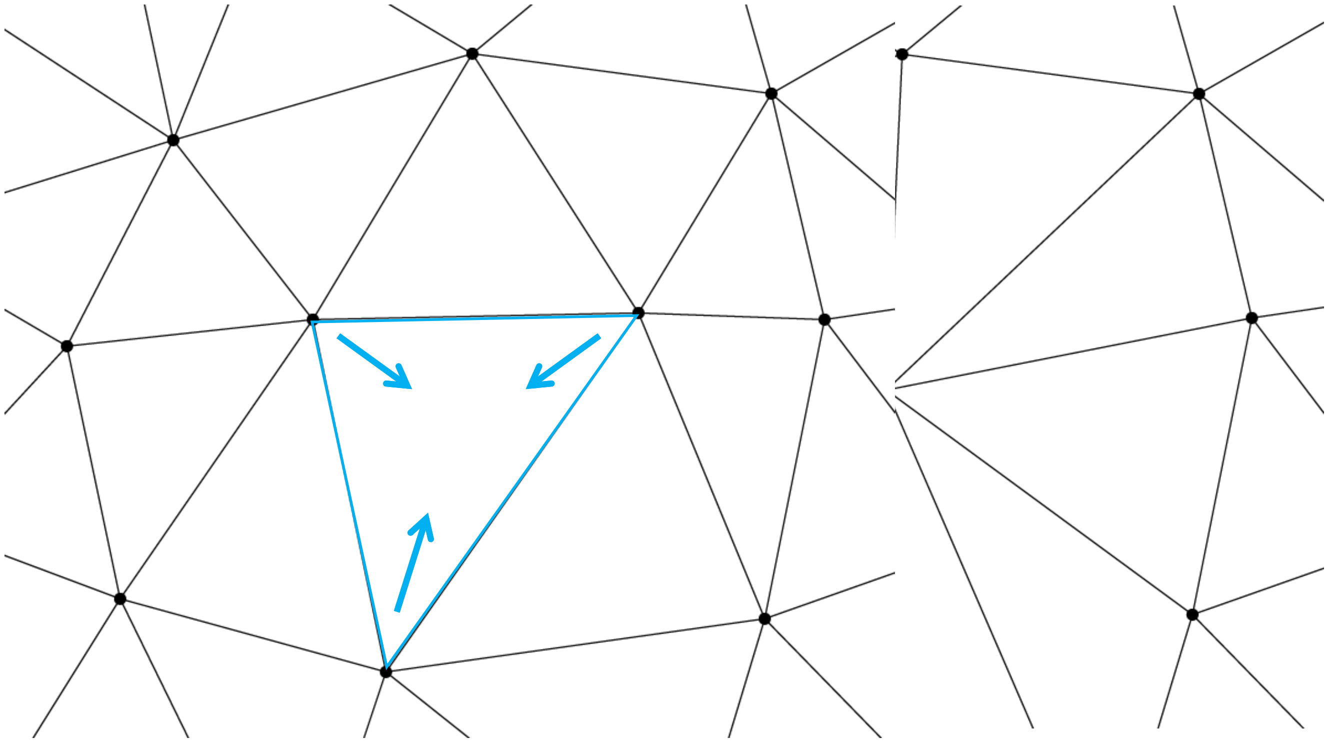
Decimate Faces

- Can we delete faces?

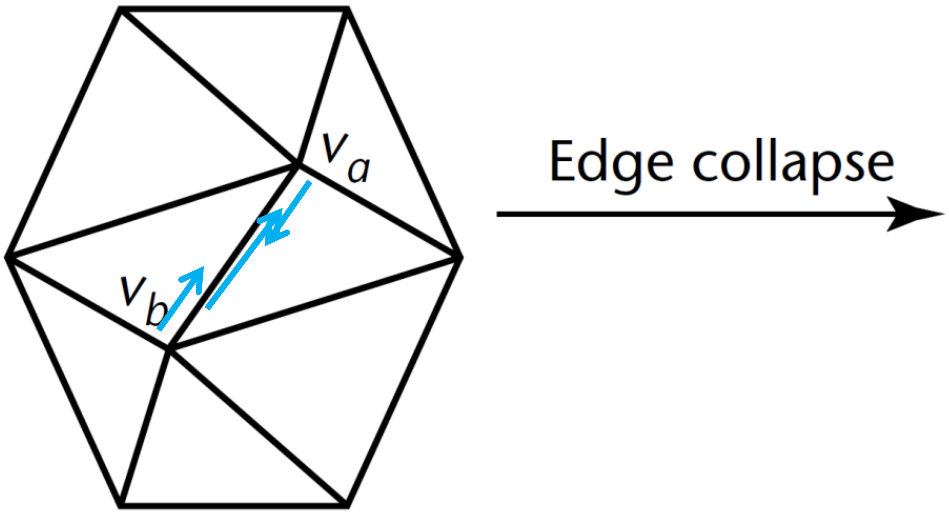




Face Collapse

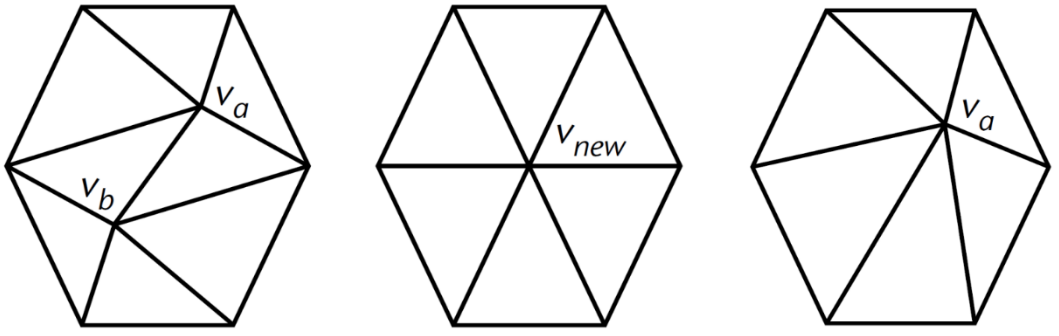


Edge Collapse



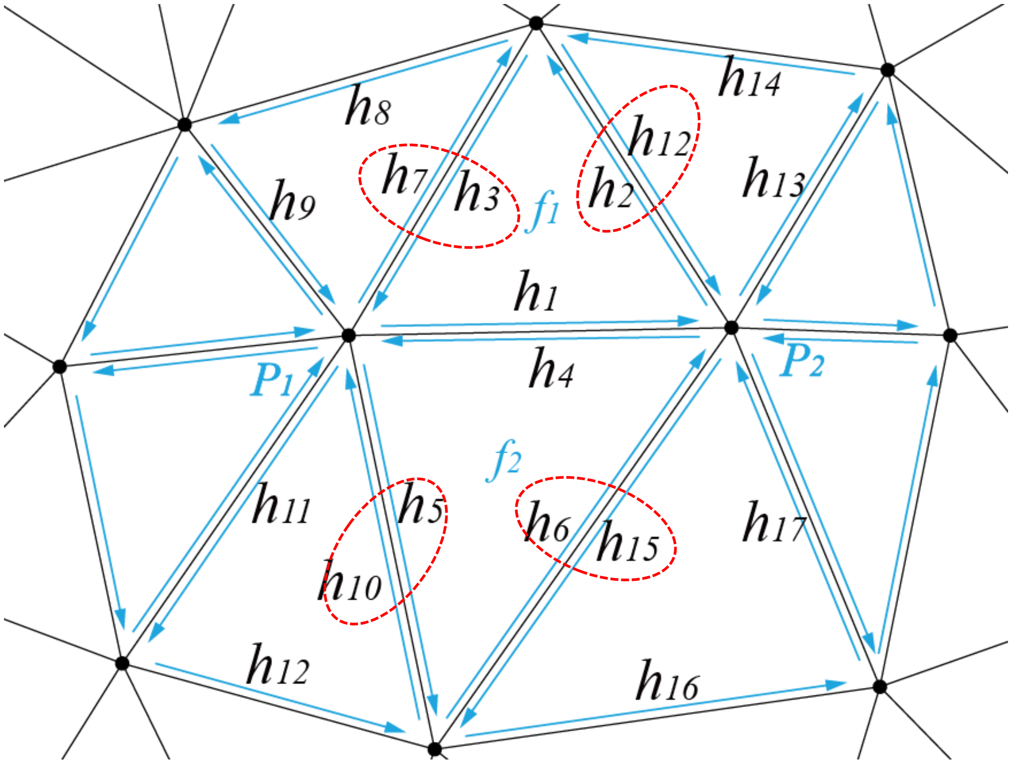
Comparison

- Vertex removal *vs* Edge collapse
  - Re-triangulation



Implementation

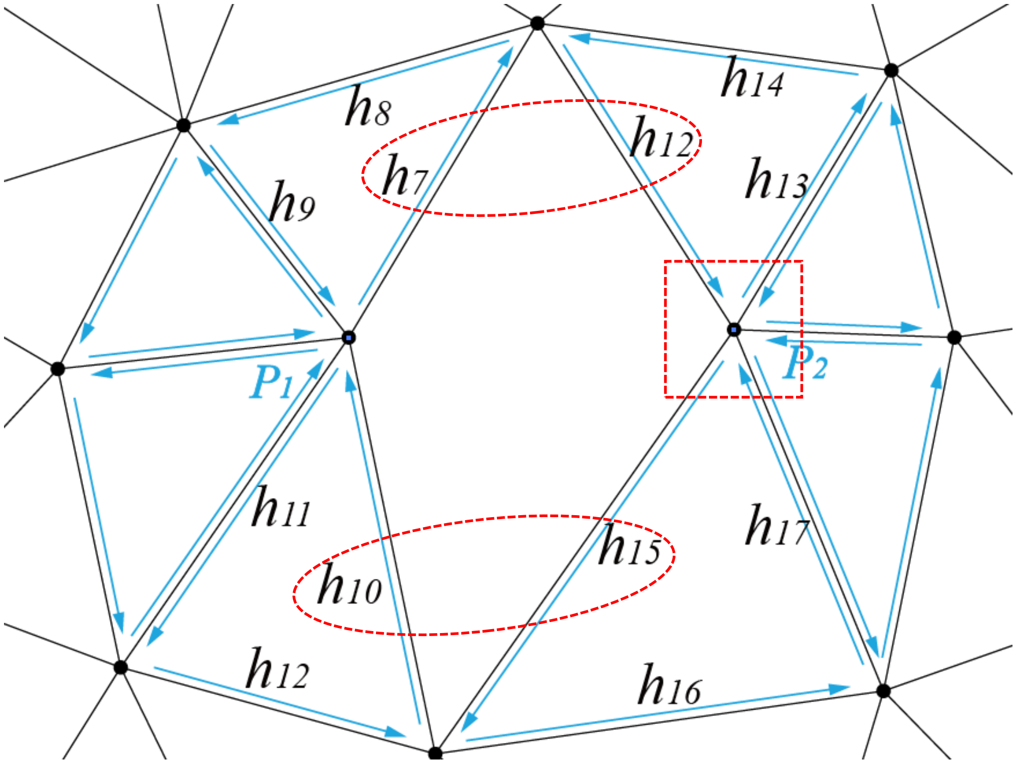
- **Half-edge data structure**
  - $h_{12}.opposite = h_2$
  - $h_7.opposite = h_3$
  - $h_{15}.opposite = h_6$
  - $h_{10}.opposite = h_5$
  - vice verse



- **STEP1**

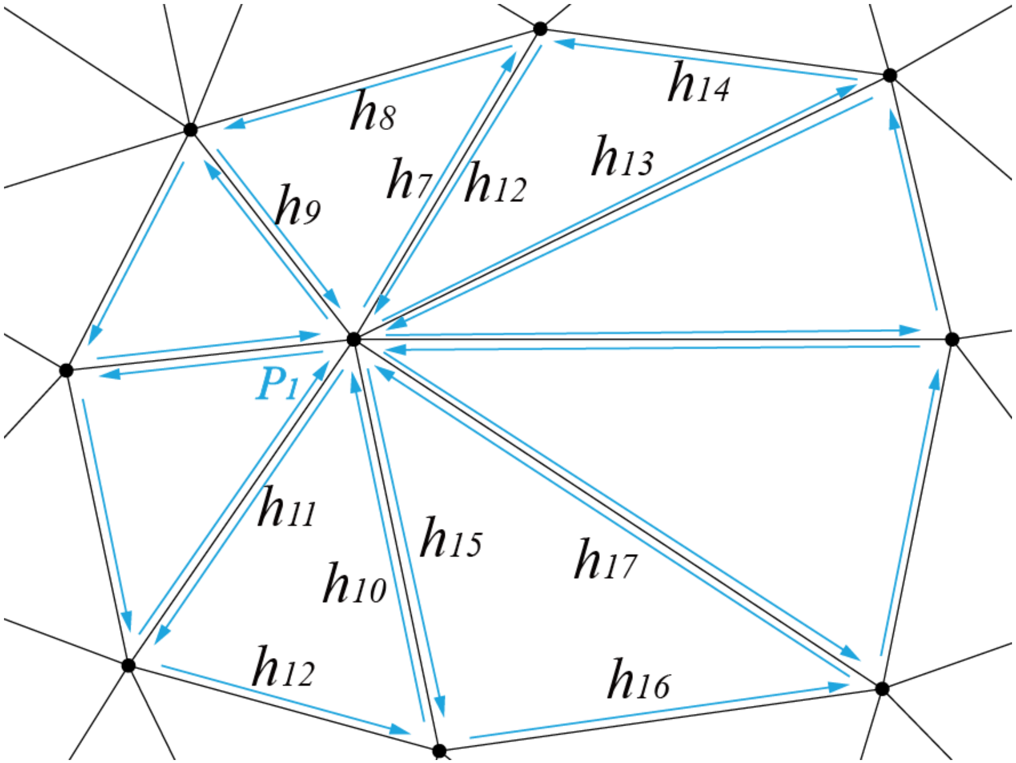
Implementation

- **Half-edge data structure**
- $h_{12}.opposite = h_7$
- $h_7.opposite = h_{12}$
- $h_{15}.opposite = h_{10}$
- $h_{10}.opposite = h_{15}$
- $h_{12}.vertex = P_2$
- $h_{17}.vertex = P_2$
- … …



Implementation

- **Half-edge data structure**
- $h_{12}.vertex = P_1$
- $h_{17}.vertex = P_1$

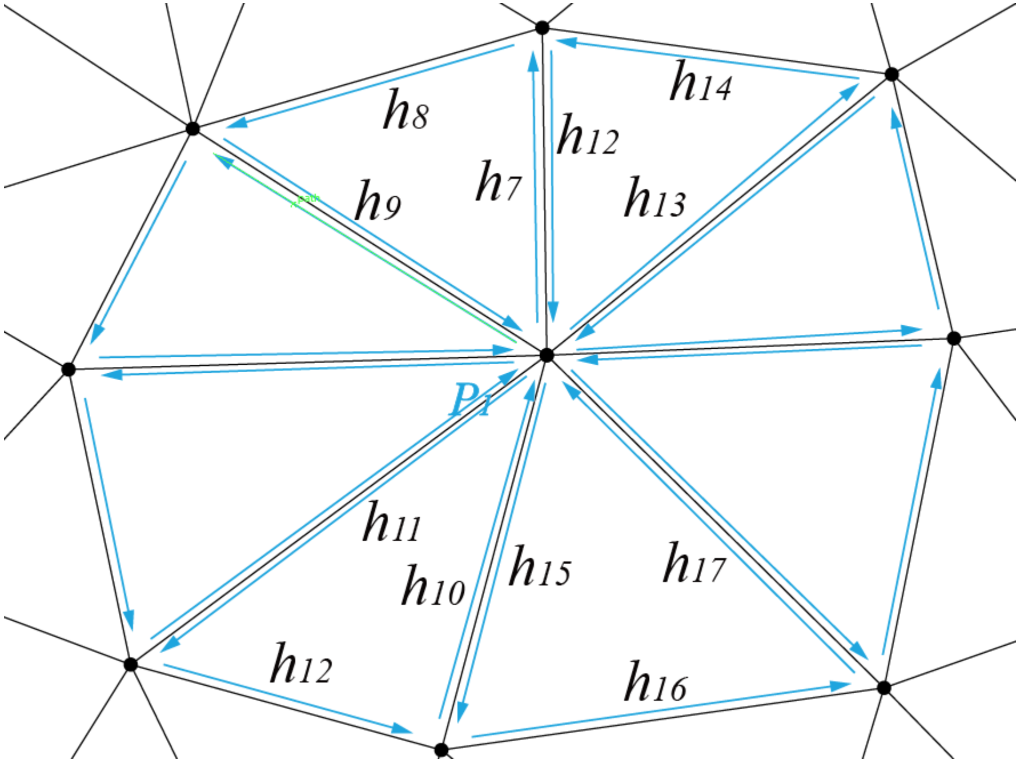


- **STEP2**

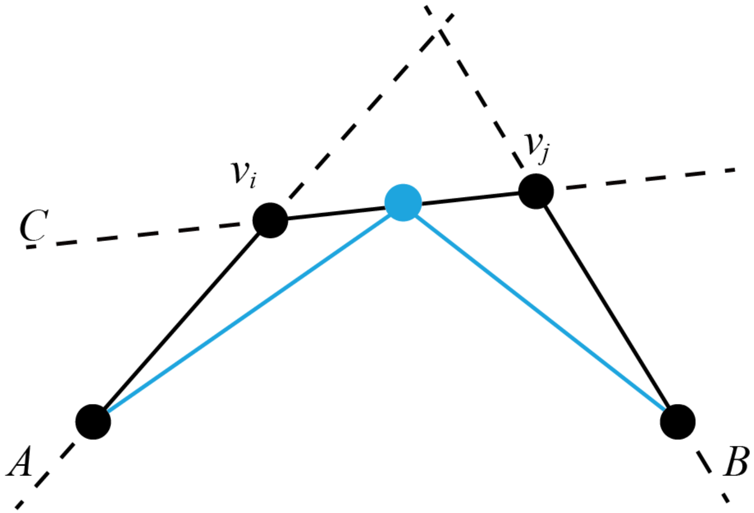
Implementation

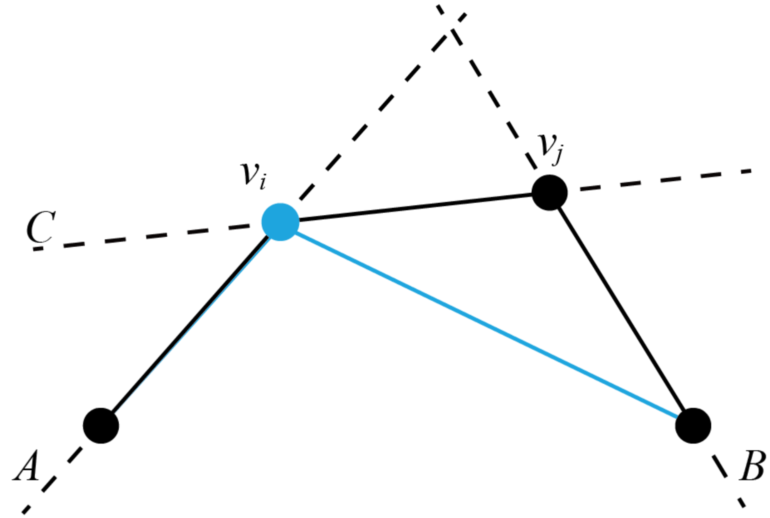
- Half-edge data structure
  - $h_{12}.vertex = P_1$
  - $h_{17}.vertex = P_1$

- ?



Error Metrics

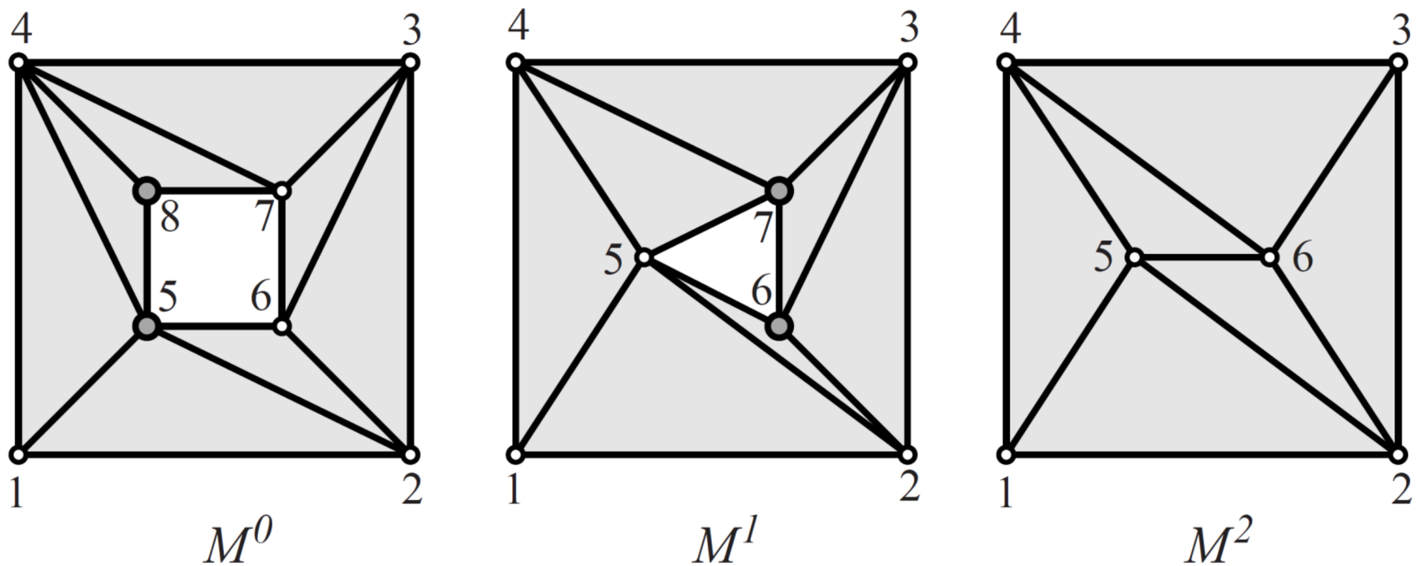




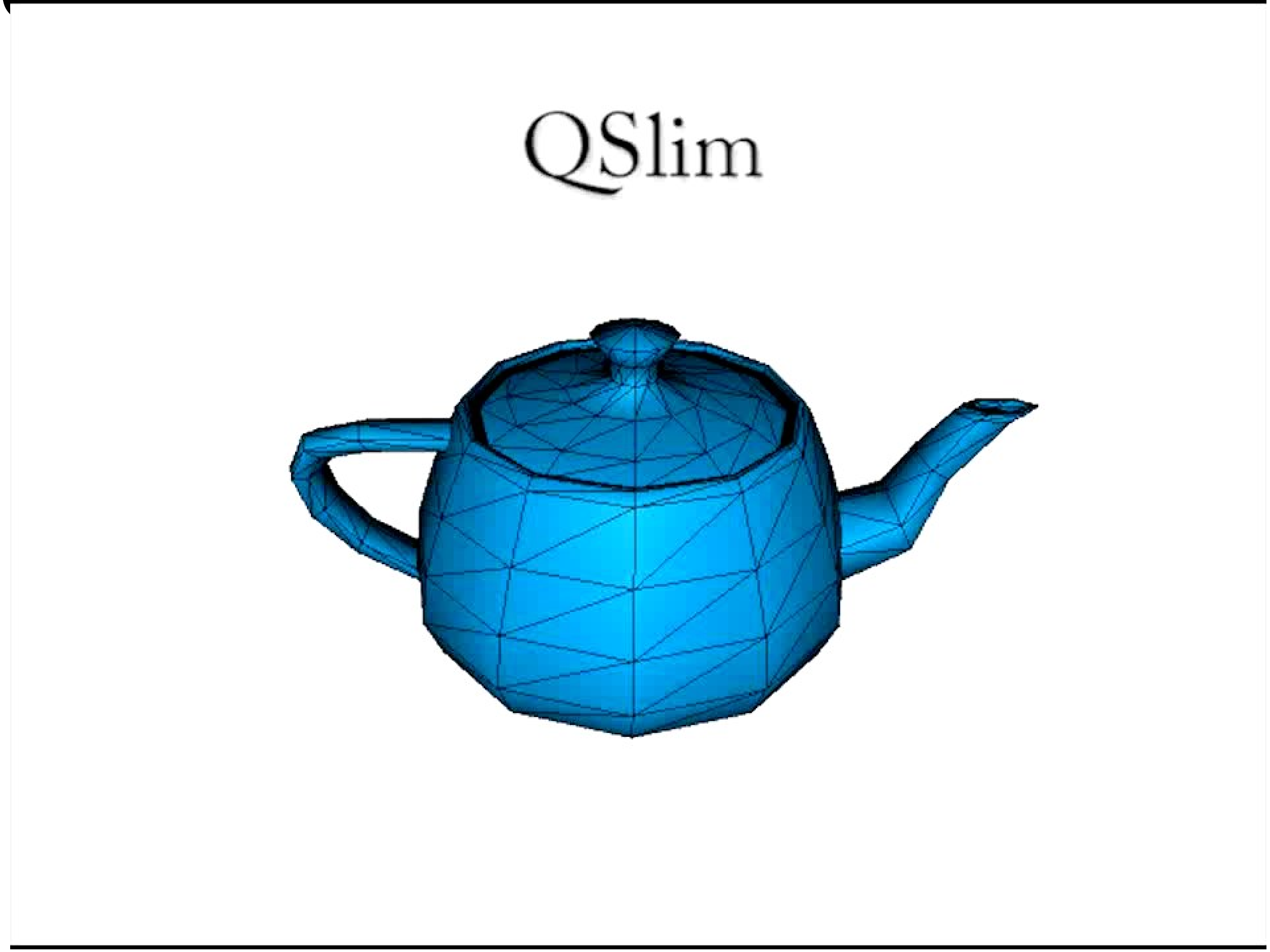
Quadric Error Metrics (QEM)

Iterative Collapsing

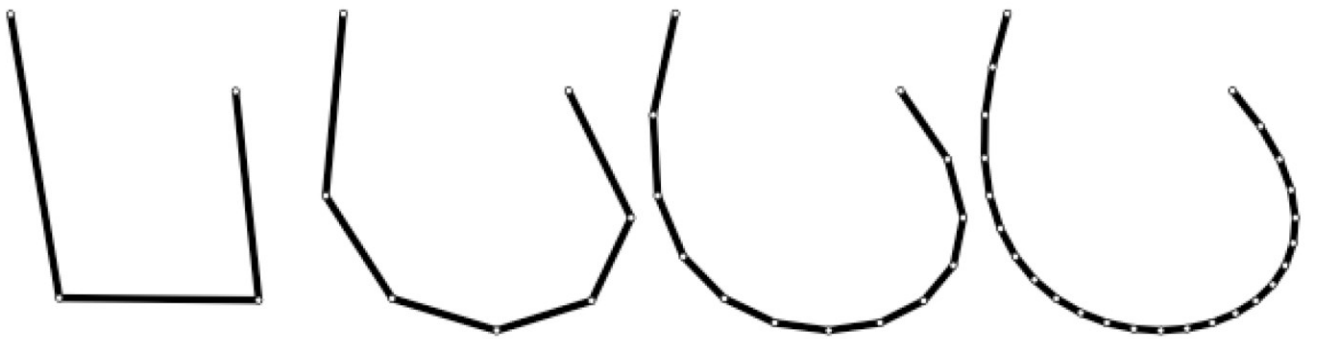
- Finding optimal approximations of surfaces is an NP-Hard problem
- Greedy algorithm

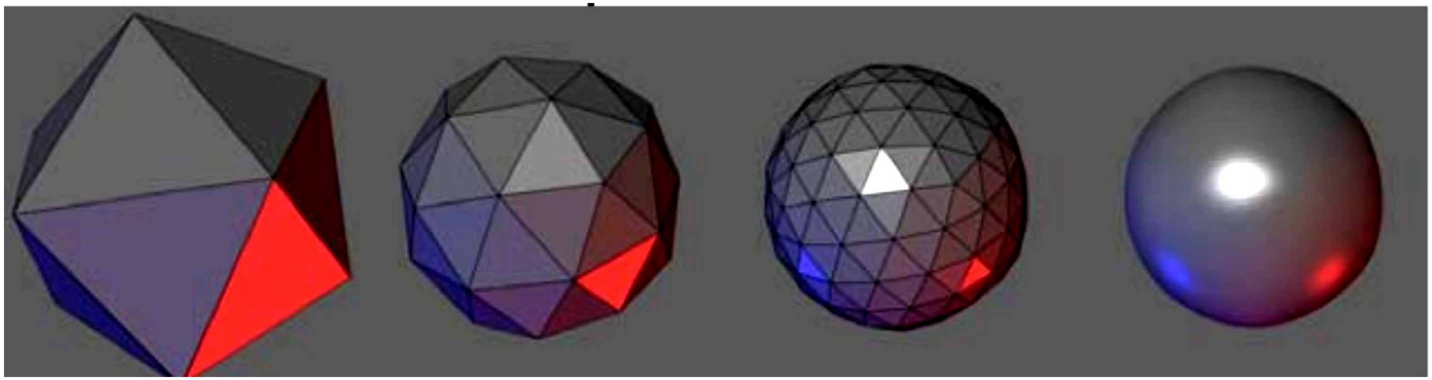


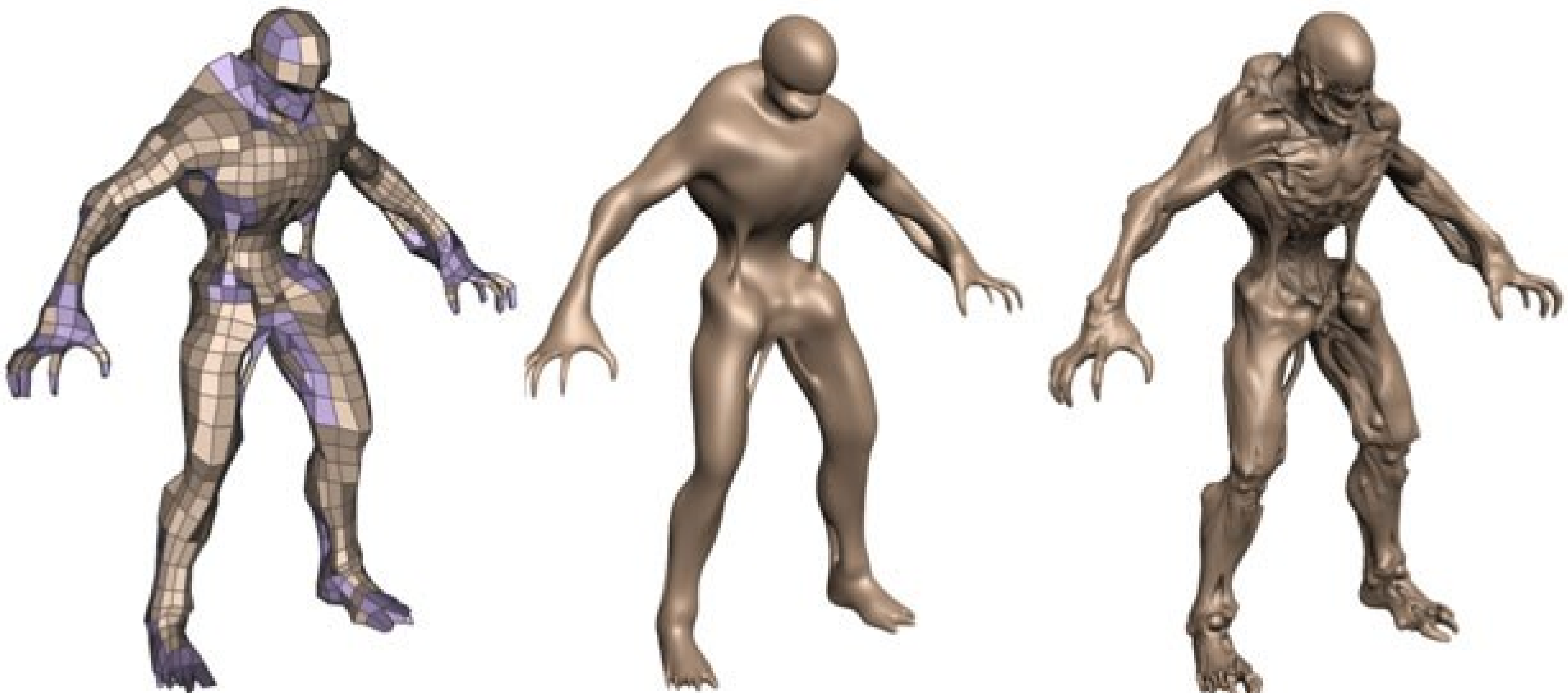
Demo



Subdivision Surface







After a coarse model (left) goes through tessellation, a smooth model is produced (middle). When displacement mapping is applied (right), characters approach film-like realism. © Kenneth Scott, id Software 2008

References

- Steve Marschner, CS4620/5620 Computer Graphics, Cornell
- Elif Tosun, Computer Graphics, The University of New York

- Questions?