Computer Graphics

- **Meshes and Manifolds**

Junjie Cao @ DLUT

Spring 2019

http://jjcao.github.io/ComputerGraphics/

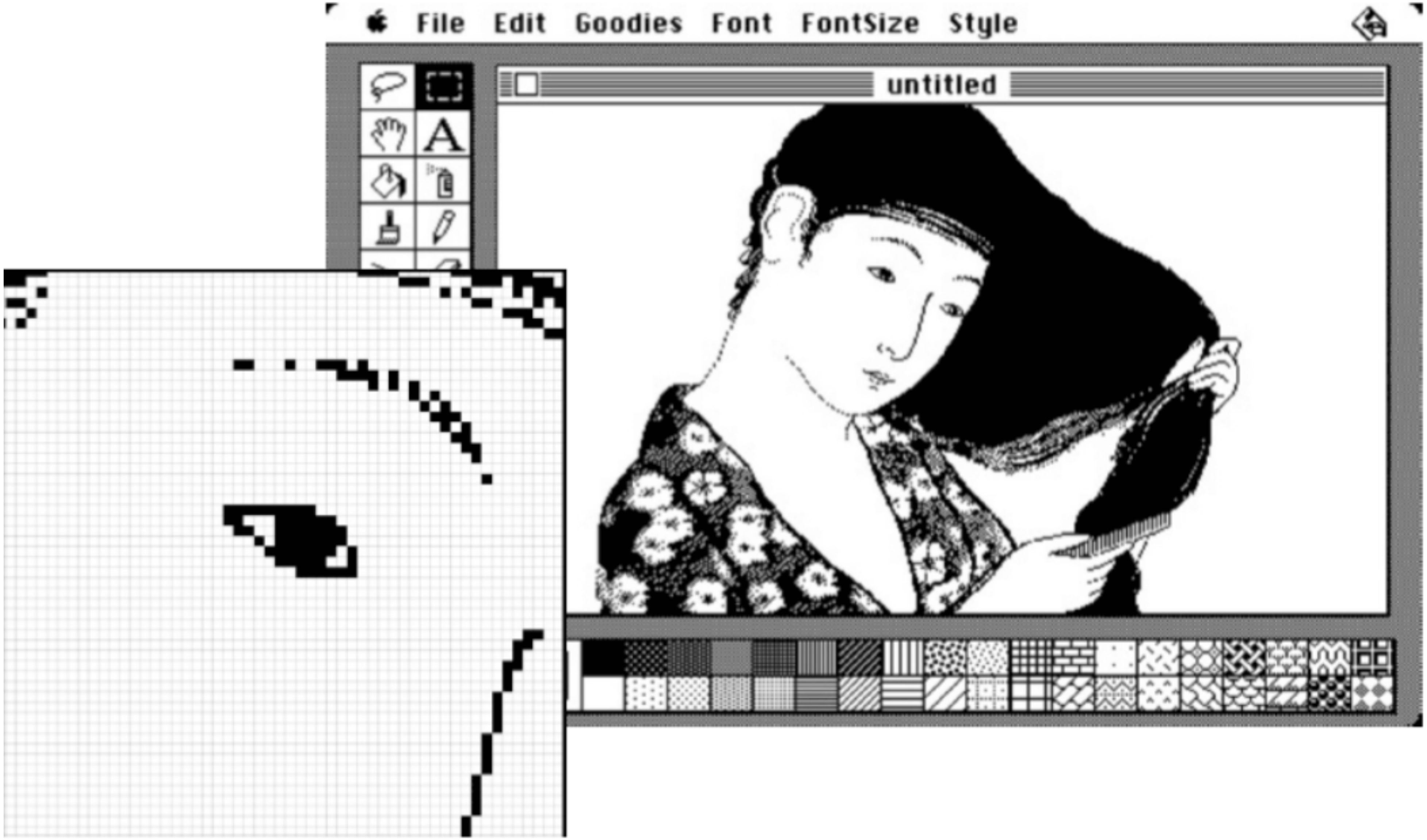
Music is dynamic, while score is static;
Movement is dynamic, while law is static.

Review: overview of geometry

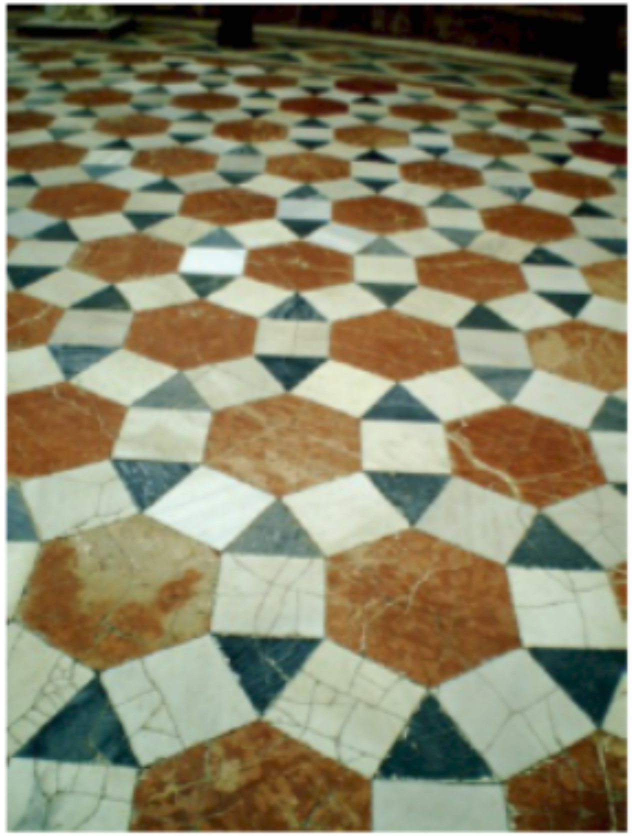
- **Many types of geometry in nature**
- **Demand sophisticated representations**
- **Two major categories:**
  - **IMPLICIT - “tests” if a point is in shape**
  - **EXPLICIT - directly “lists” points**
- **Lots of representations for both**

Bitmap Images, Revisited

- **To encode images, we used a *regular grid* of pixels:**

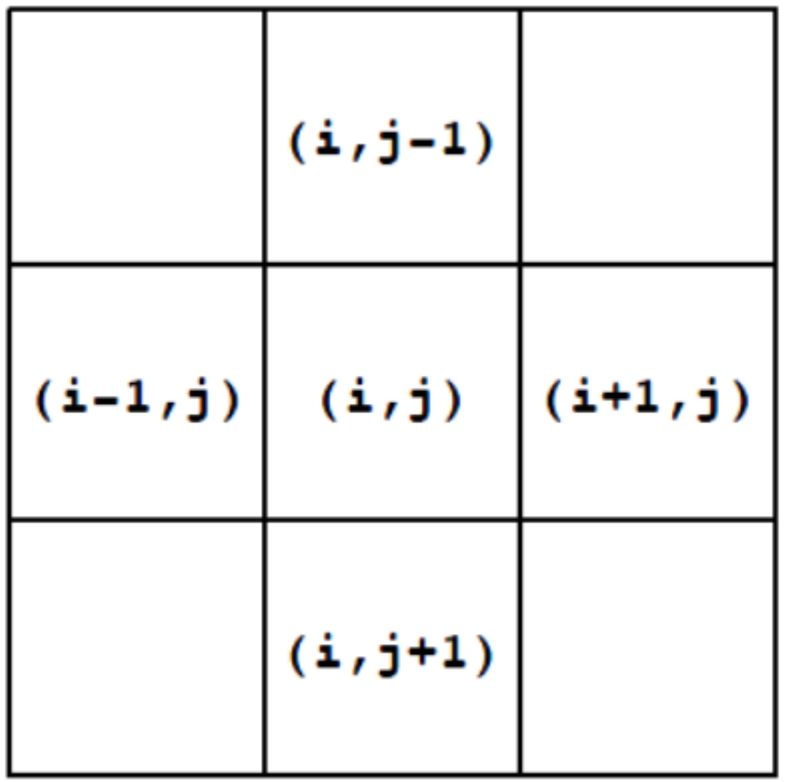


But images are not fundamentally made of little squares:
So why did we choose a square grid?



…rather than dozens of alternatives?

Regular grids make life easy

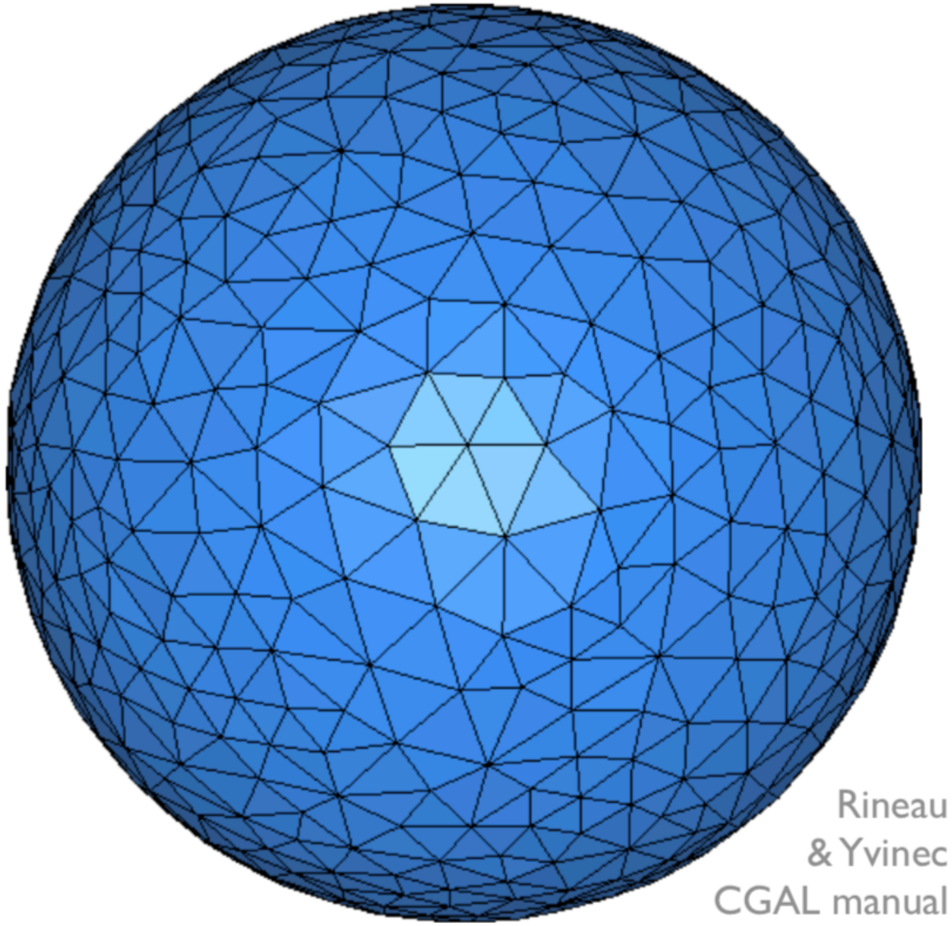


- One reason: SIMPLICITY / EFFICIENCY
  - E.g., always have four neighbors
  - Easy to index, easy to filter…
  - Storage is just a list of numbers
- Another reason: GENERALITY
  - Can encode basically any image
- Are regular grids *always* the best choice for bitmap images?
  - No! E.g., suffer from anisotropy, don't capture edges, ...
  - But *more often than not* are a pretty good choice
- Will see a similar story with geometry...

So, how should we encode surfaces?

Andrzej Barabasz

spheres



Rineau
& Yvinec
CGAL manual

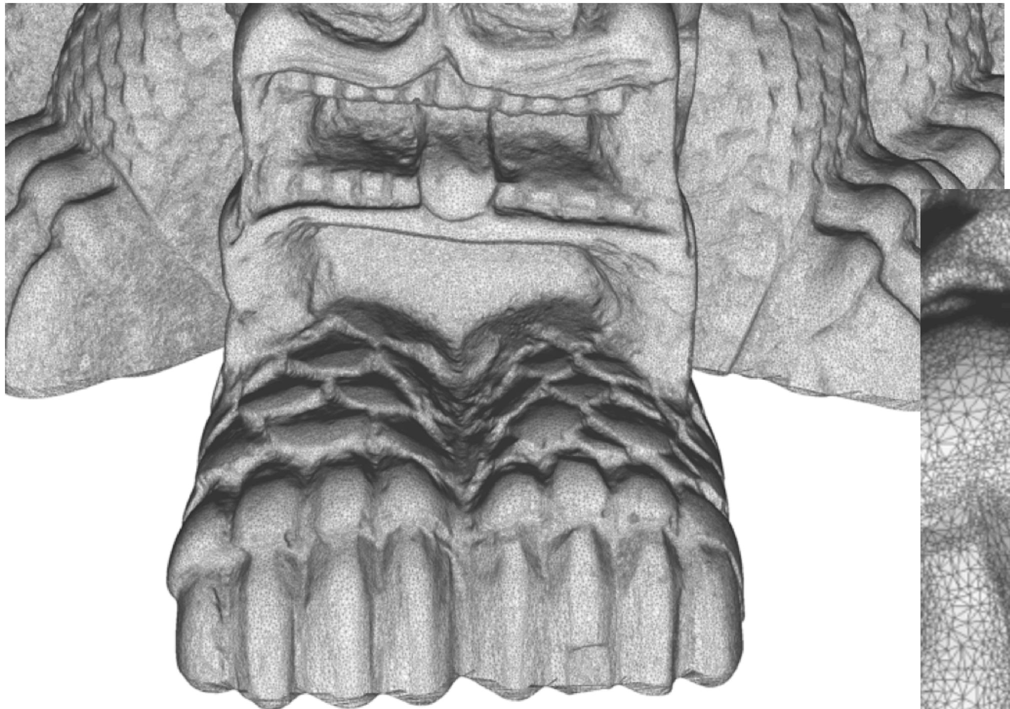
approximate sphere

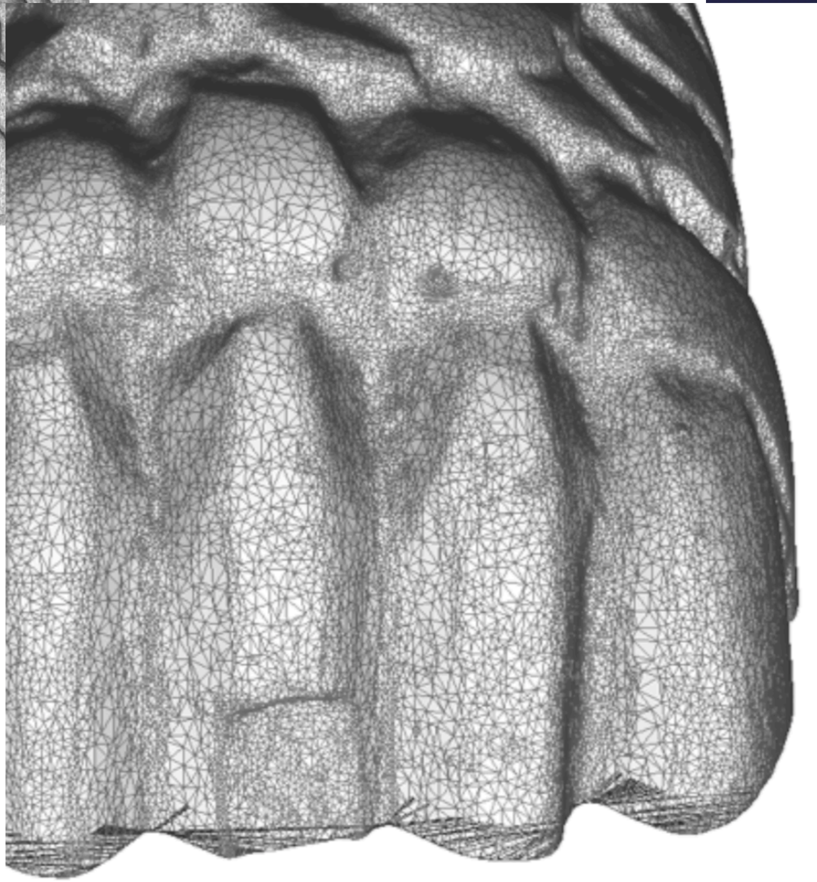
Where Meshes Come From

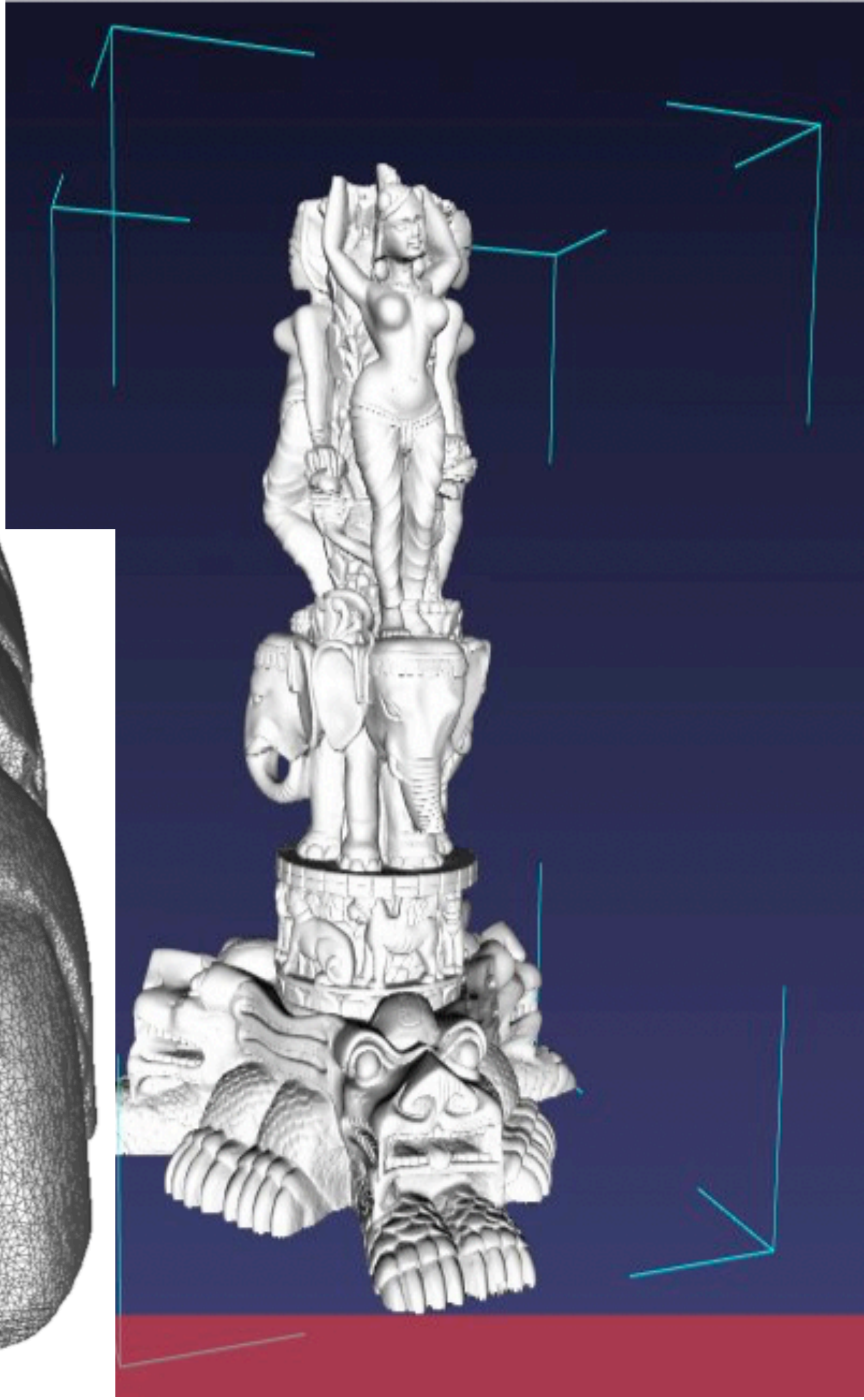
- Model manually
  - Write out all polygons
  - Write some code to generate them
  - Interactive editing: move vertices in space
- Acquisition from real objects
  - 3D scanners, vision systems
  - Generate set of points on the surface
  - Need to convert to polygons

A large mesh

- 10 million triangles from a high-resolution 3D scan



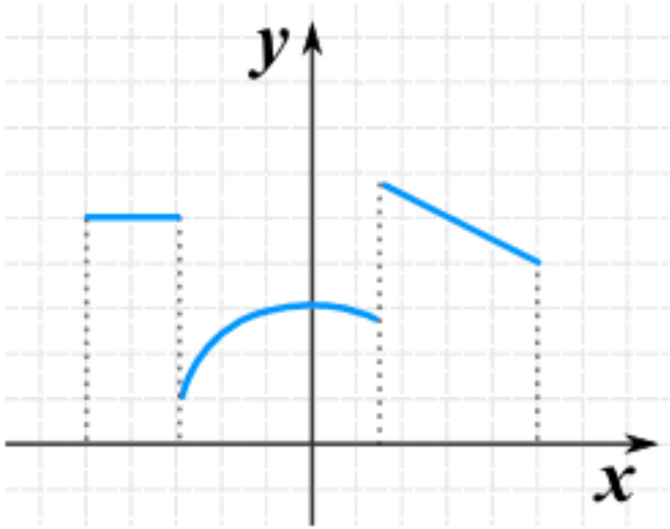




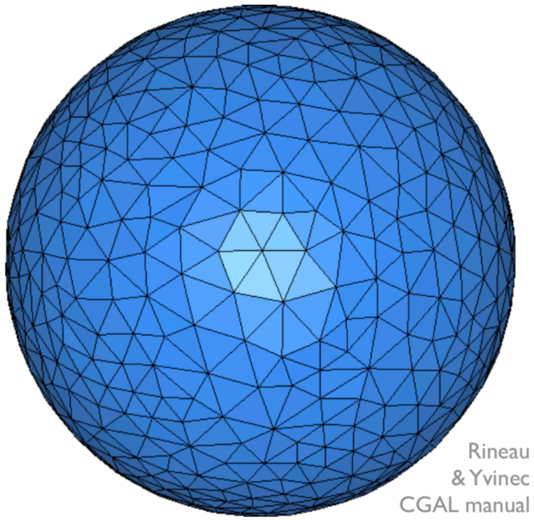
about a trillion-triangle worldwide model
from semi-automatically processed
satellite, aerial, and street photography

Polygon Mesh

- Polygon meshes are C^0 piecewise linear surface representations.
- Analogous to piecewise functions:



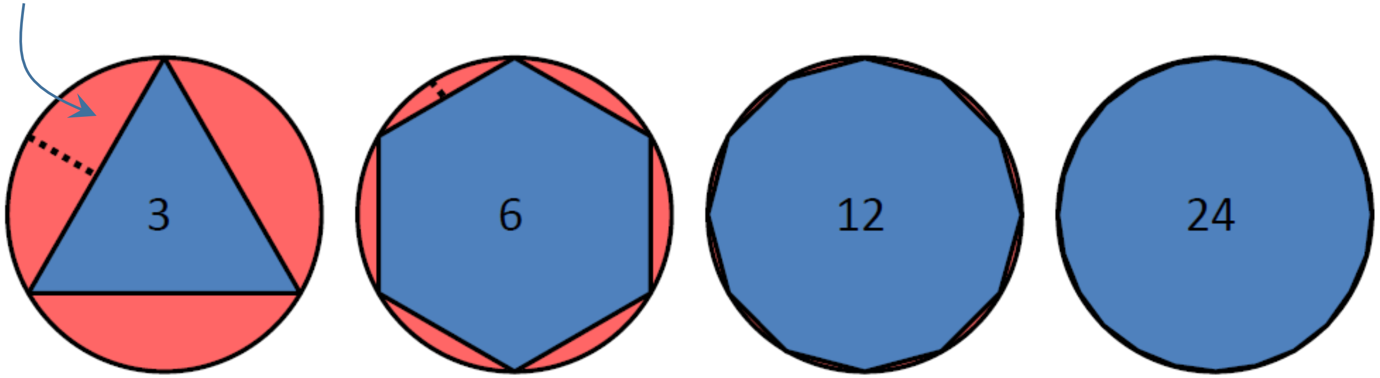
$$f(x) = \begin{cases} 6 & \text{if } x < -2 \\ x^2 & \text{if } x > -2 \text{ and } x \leq 2 \\ 10 - x & \text{if } x > -2 \end{cases}$$



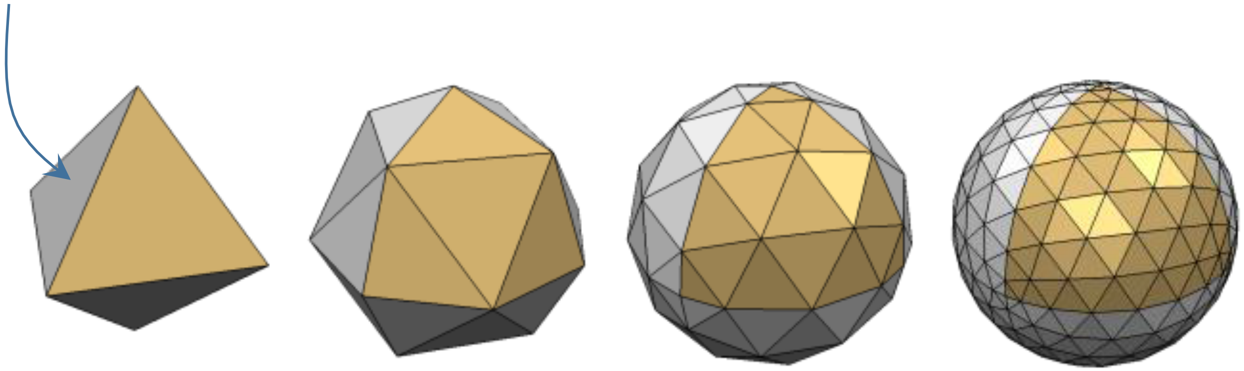
Rineau & Yvinec CGAL manual

Polygon Mesh

✓1D: This line piece approximates the given shape (circle) only locally.

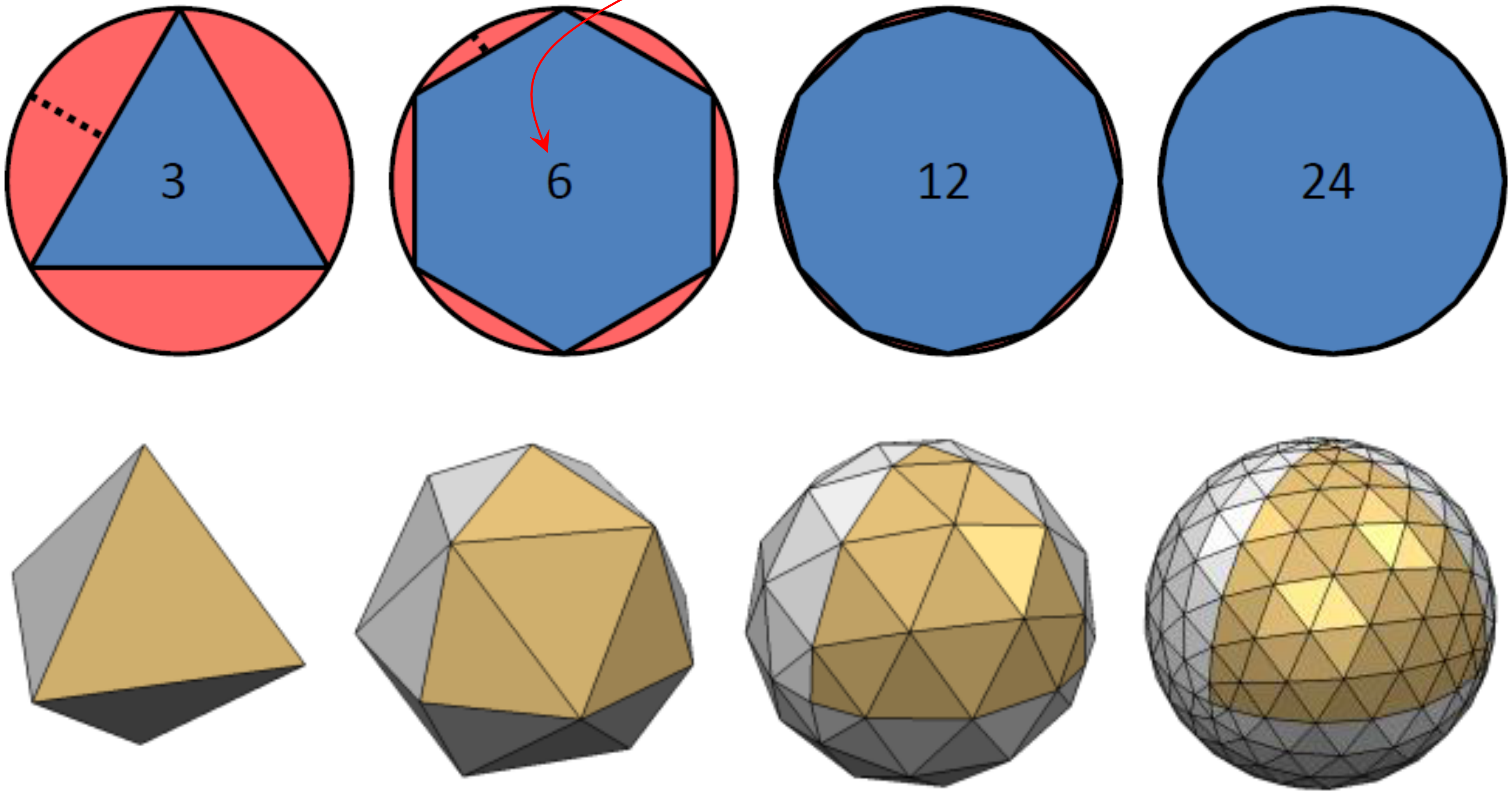


✓ 2D: This triangle piece approxs the given shape (sphere) only locally.



Polygon Mesh

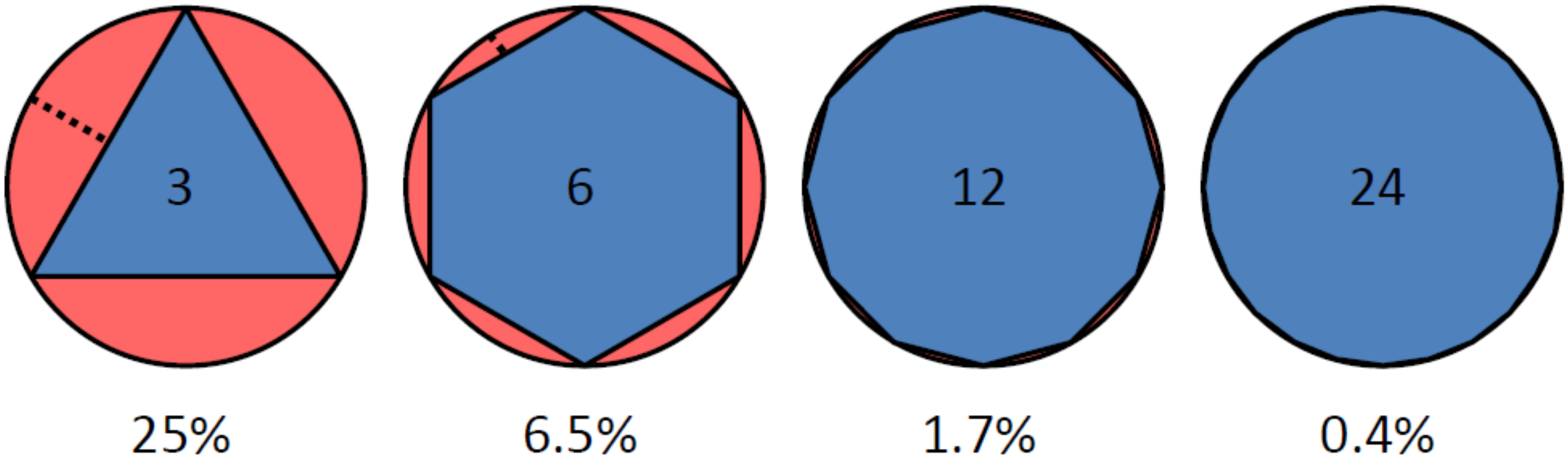
- Approximation error decreases as # pieces increases.



Polygon Mesh

✓Approximation error is quadratic.

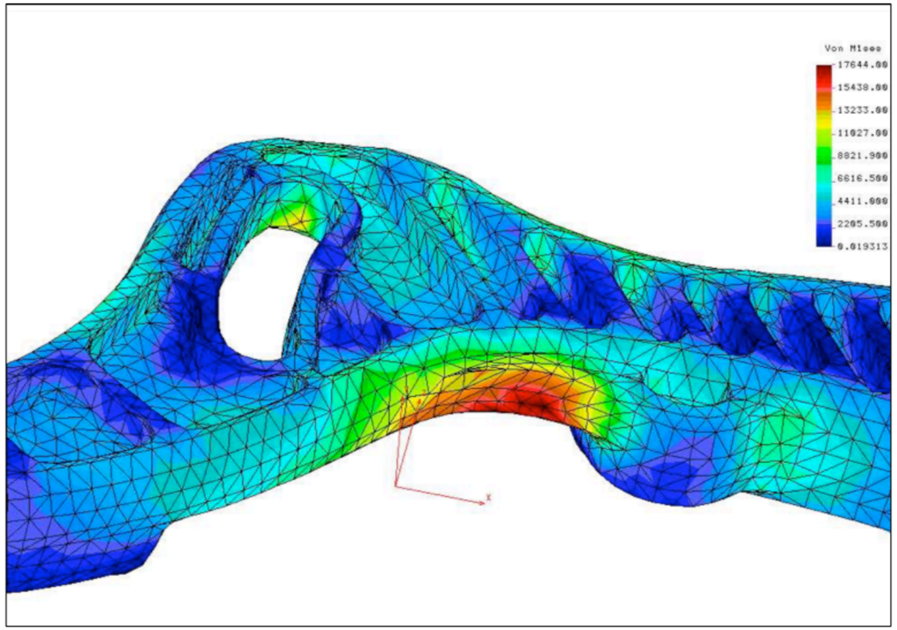
- ✓As # pieces doubled, error decreases one forth.



Polygonal meshes are a good compromise

- **Theorem** Given a smooth surface S and a given error $\varepsilon > 0$, there exists a piecewise linear surface (mesh) M, such that $|\mathrm{M} - S| < \varepsilon$ for all points of M.

- Piecewise linear approximation → error is O(h^2) (Error inversely proportional to #faces)
- Arbitrary topology surfaces
- Piecewise smooth surfaces
- Adaptive sampling
- Efficient GPU-based rendering/processing
- Finite element



PATRIOT Engineering

finite element analysis

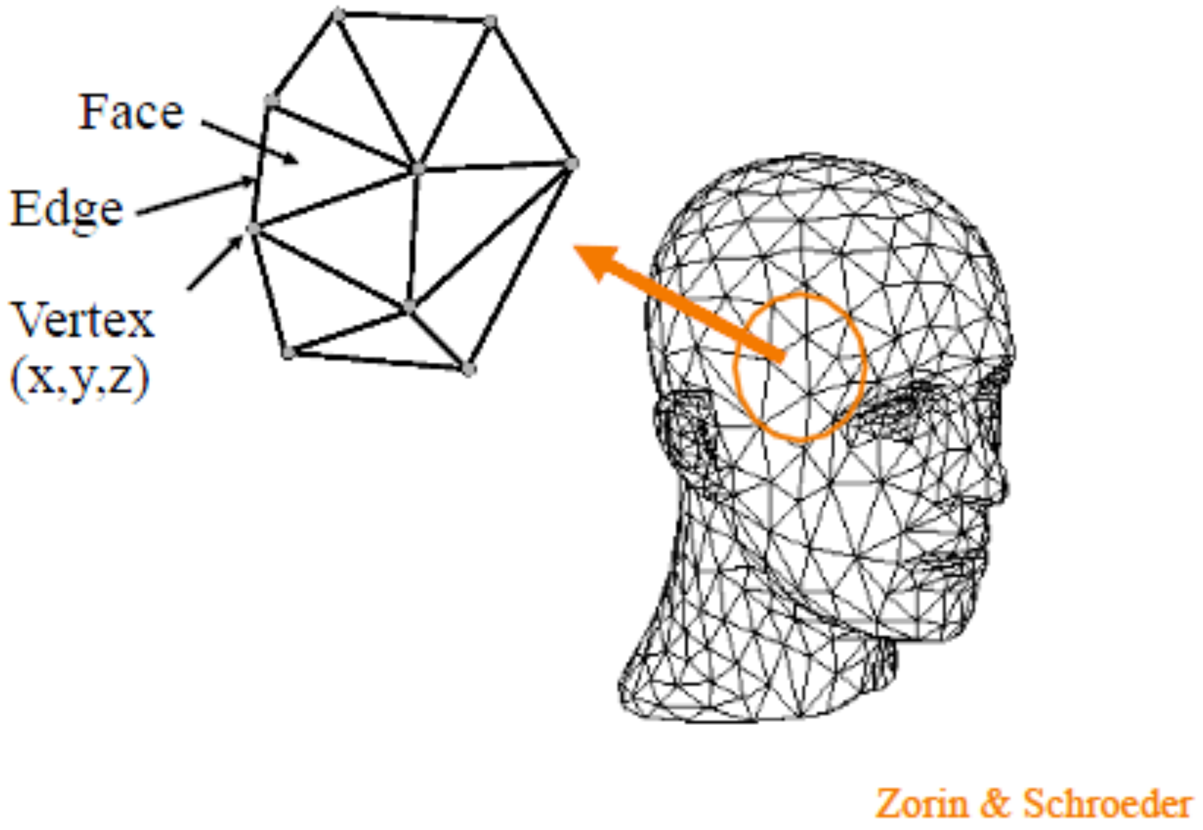
What is a Mesh?

What is a Mesh?

- A Mesh is a pair *(P,K)*, where *P* is a set of point positions $P = \{p_i \in R^3 \mid 1 \le i \le n\}$ and *K* is an abstract simplicial complex which contains all topological information.

$$K = V \cup E \cup F$$

- Vertices $v = \{i\} \in V$
- Edges $e = \{i, j\} \in E$
- Faces $f = \{i_1, i_2, \ldots, i_{n_f}\} \in F$



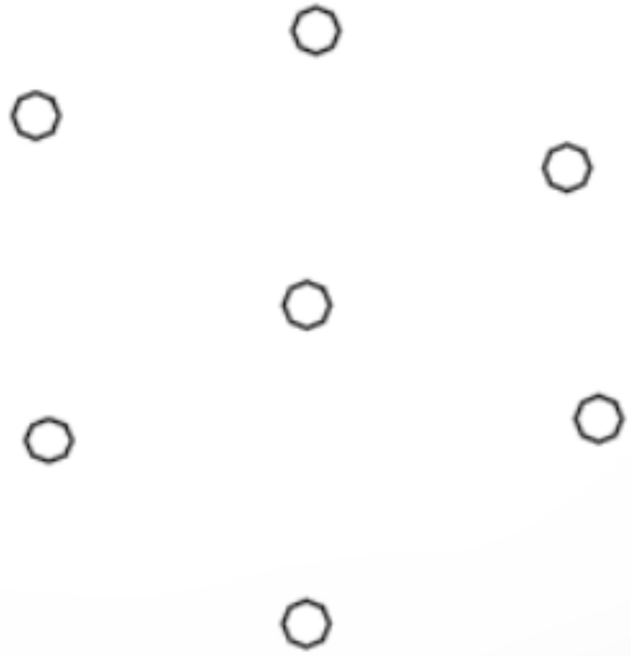
- A **Graph** is a pair G=(V,E)

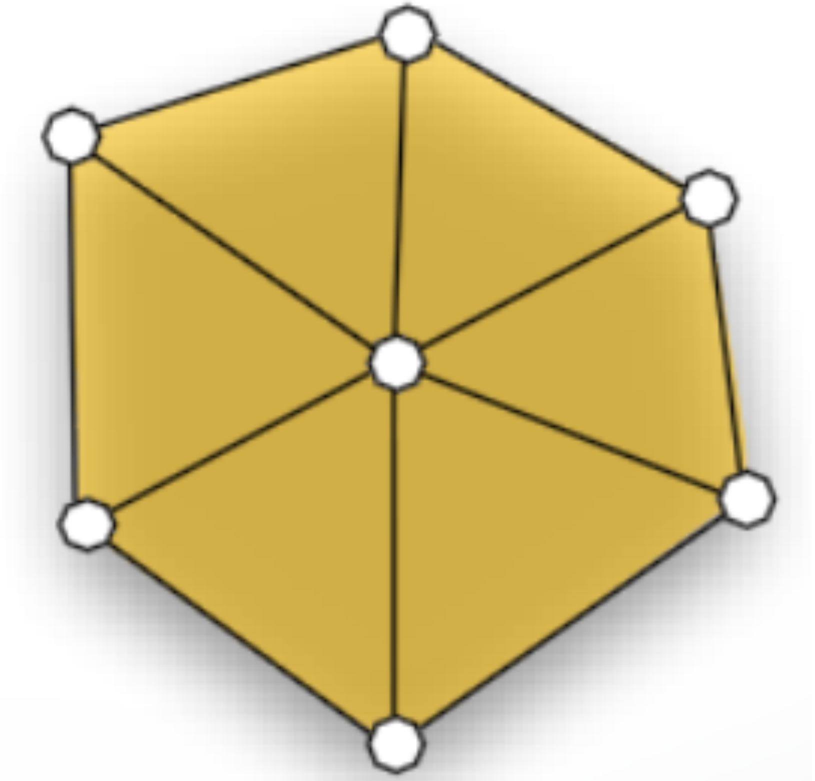
Polygonal Meshes

- **The vertex positions capture the geometry of the surface**
- The mesh connectivity captures the **topology** of the surface

geometry $\mathbf{v}_i \in \mathbb{R}^3$

topology $e_i, f_i \subset \mathbb{R}^3$

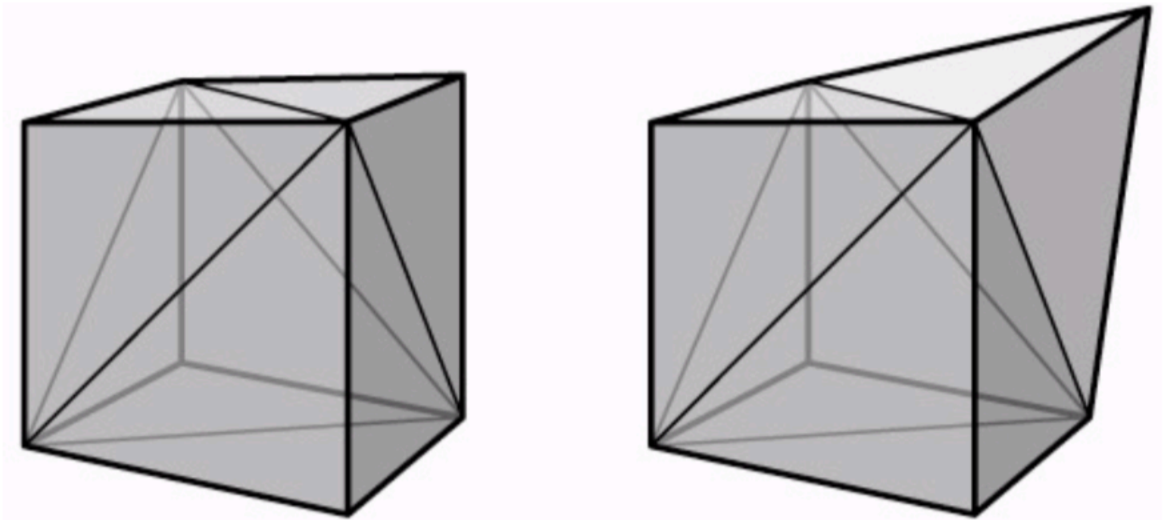




Polygonal Meshes

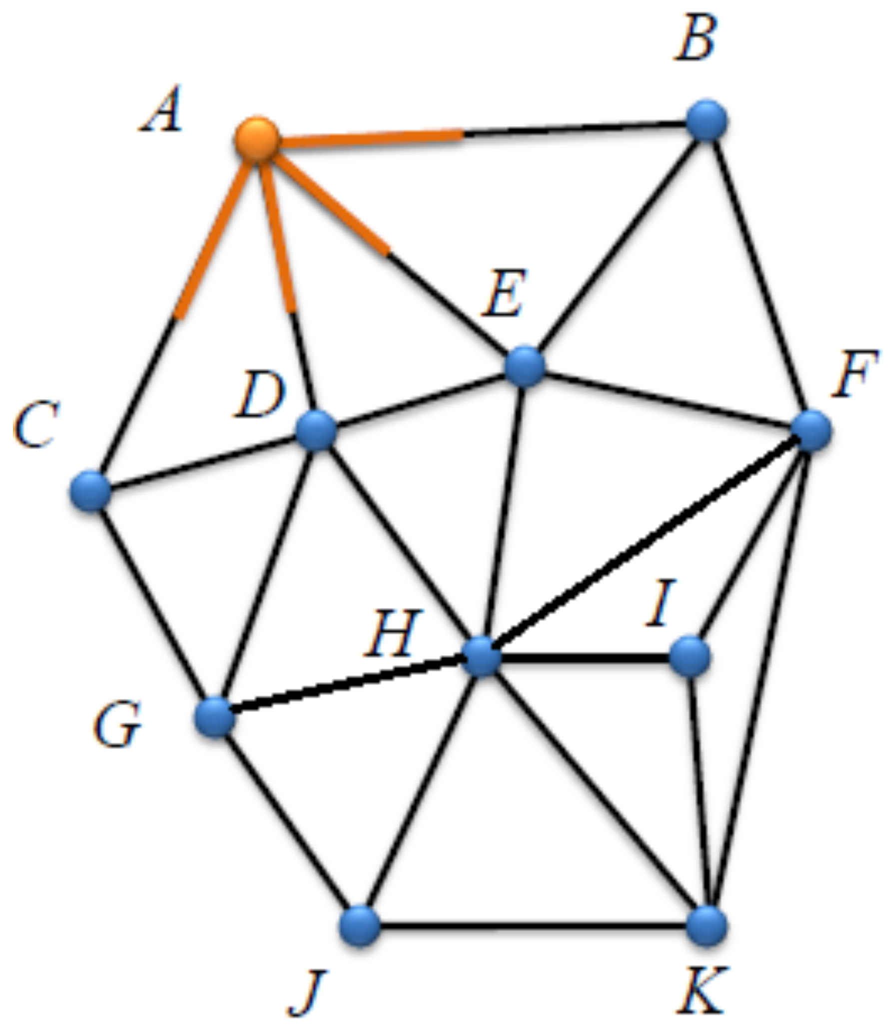
- Geometry
  - Embedding – Vertex coordinates
  - Riemannian metrics – Edge lengths
  - Conformal Structure – Corner angles (and other variant definitions)

- Topology
  - connectivity of the vertices
  - Simplicial Complex, Combinatorics



Triangle Meshes

✓ An undirected graph, with triangle faces.



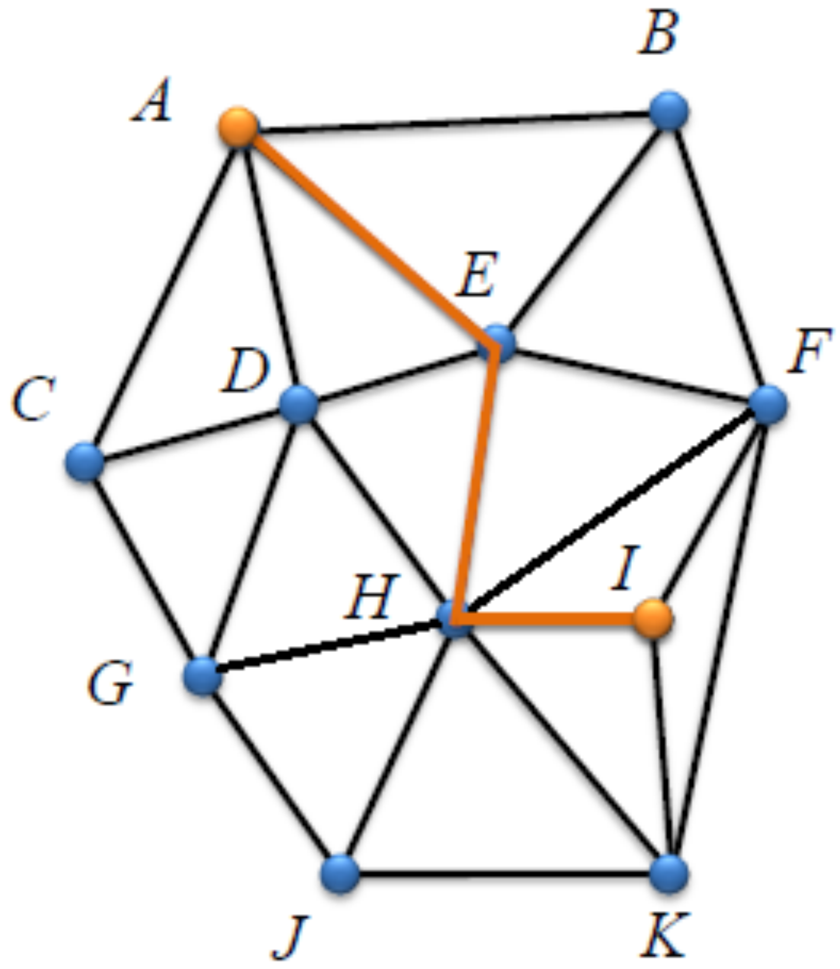
Vertex degree or valence = # incident edges

deg(A) = 4 deg(B) = 3

k-regular mesh if all vertex degrees are equal to k.

Triangle Meshes

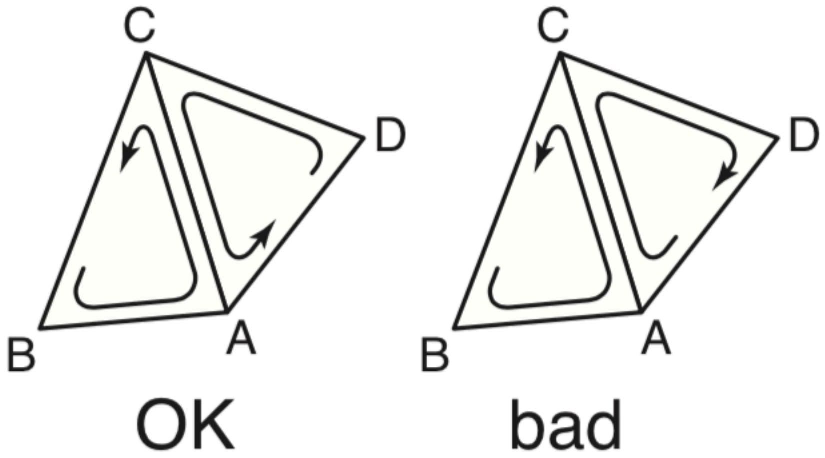
✓ An undirected graph, with triangle faces.



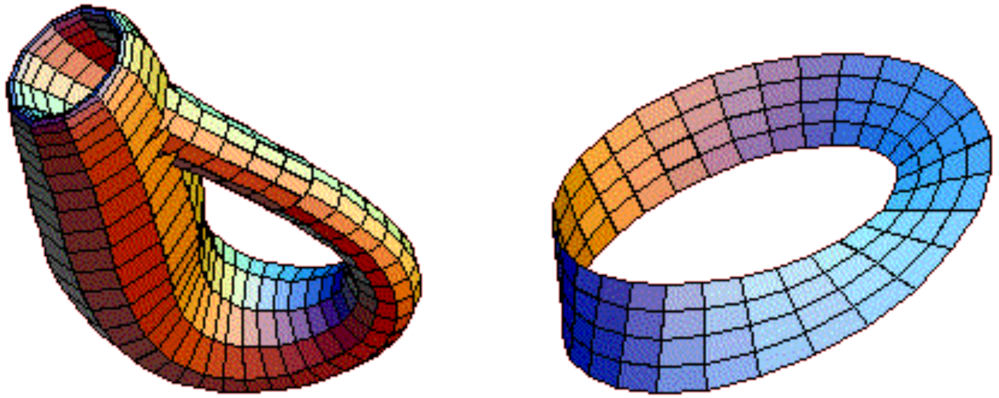
connected if every pair of vertices are connected by a path (of edges).

Topological validity - **Consistent orientation**

- Orientation of a face is defined by ordering of its vertices, which determines its normal direction, it can be clockwise or counter-clockwise => “front”
- A mesh is consistent oriented (orientable) if all faces can be oriented consistently (all CCW or all CW) such that each edge has two opposite orientations for its two adjacent faces

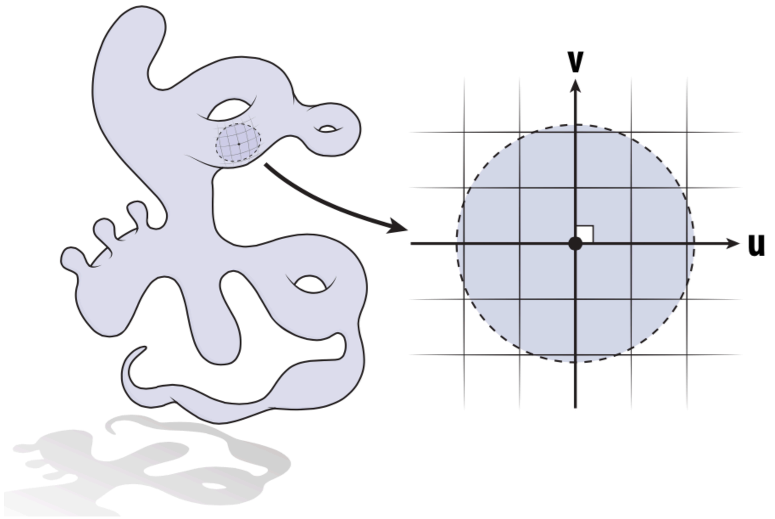


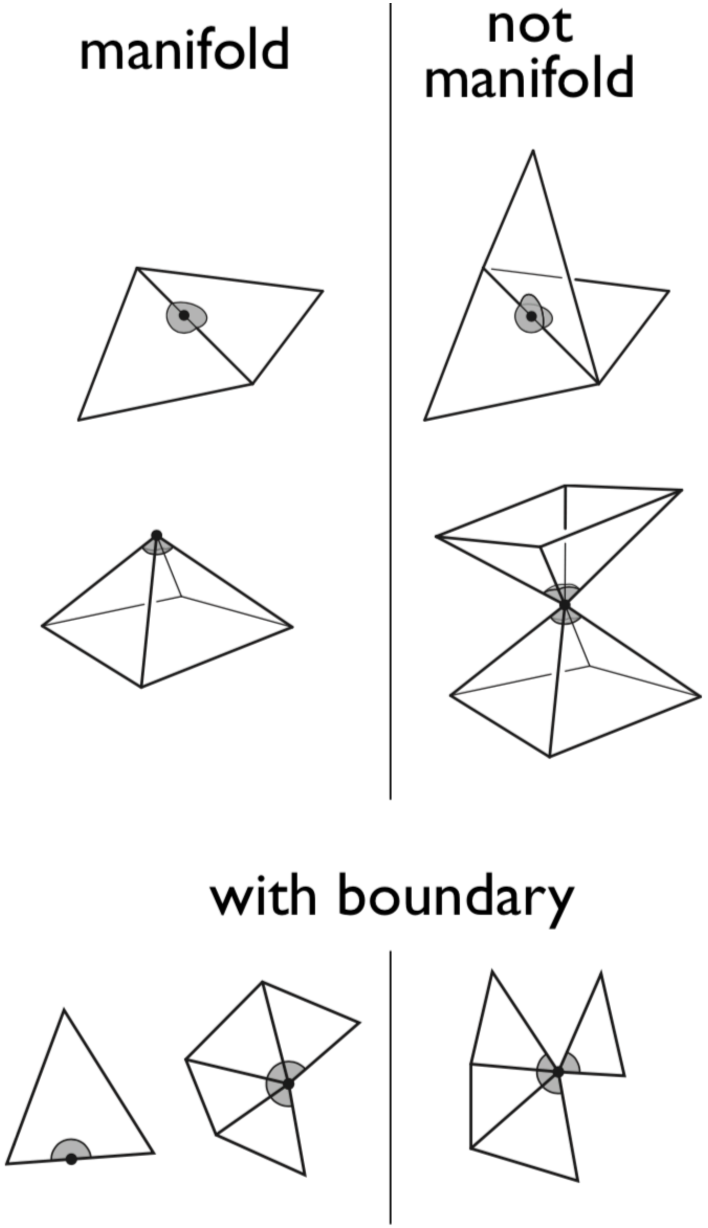
- Not every mesh can be well oriented.



non-orientable surfaces

Topological validity -- Manifold assumption

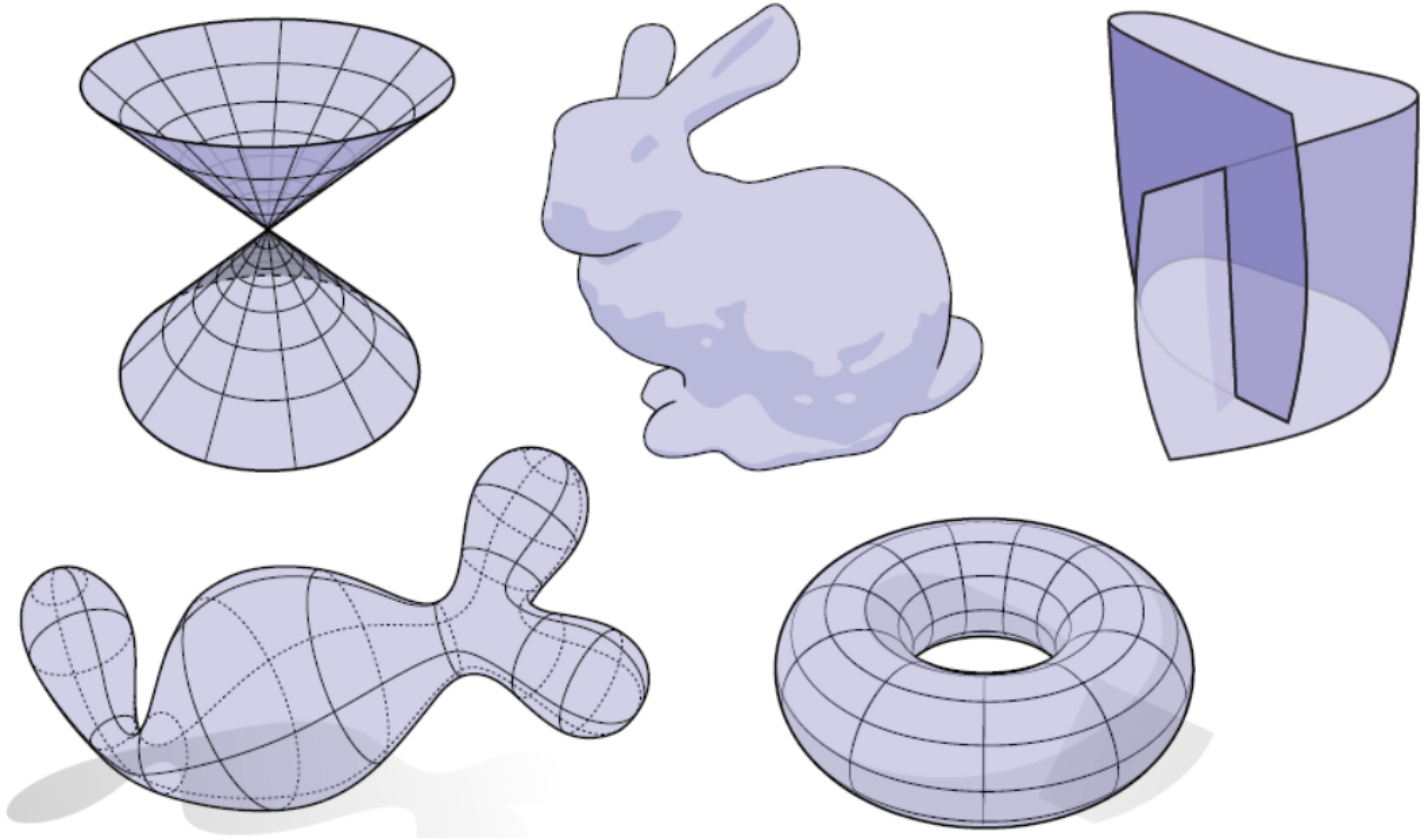




- **strongest property: be a manifold**
  - edge: each edge must have exactly 2 triangles
  - vertex: each vertex must have one loop of triangles
- **slightly looser: manifold with boundary**

Isn’t every shape manifold?

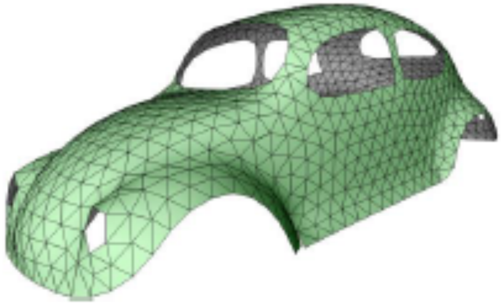
- **Which of these shapes are manifold?**



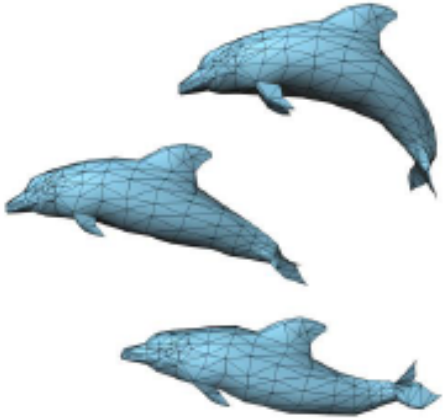
Center point never looks like the plane, no matter how close we get.

Polygon Mesh Types

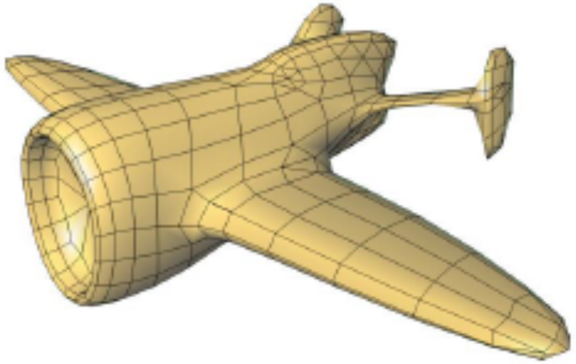
Single component,
closed, triangular,
2-manifold



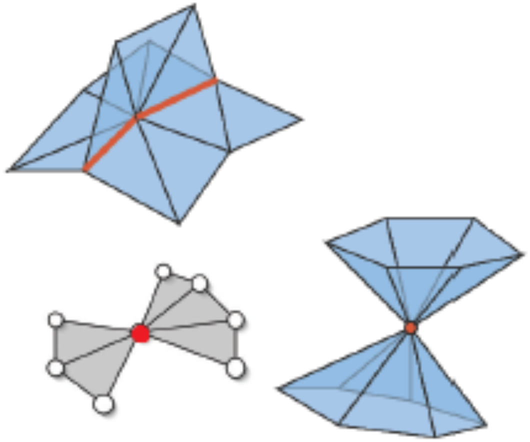
With boundaries
2-manifold



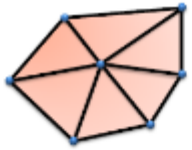
Multiple components
2-manifold



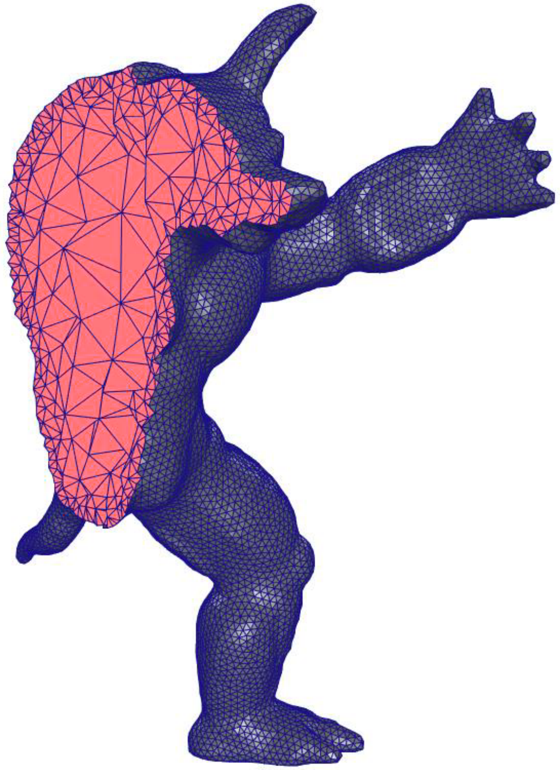
Not only triangles
2-manifold



Non manifold



2-manifoldness

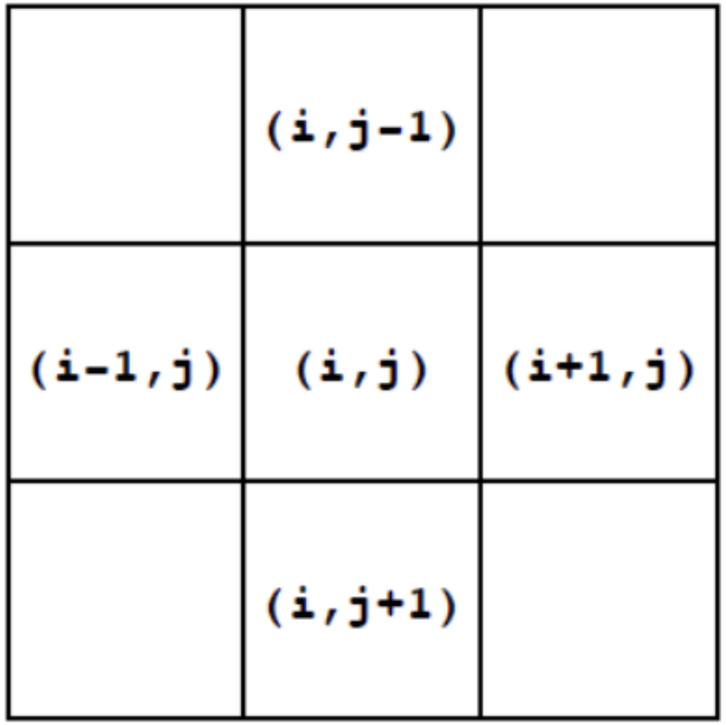


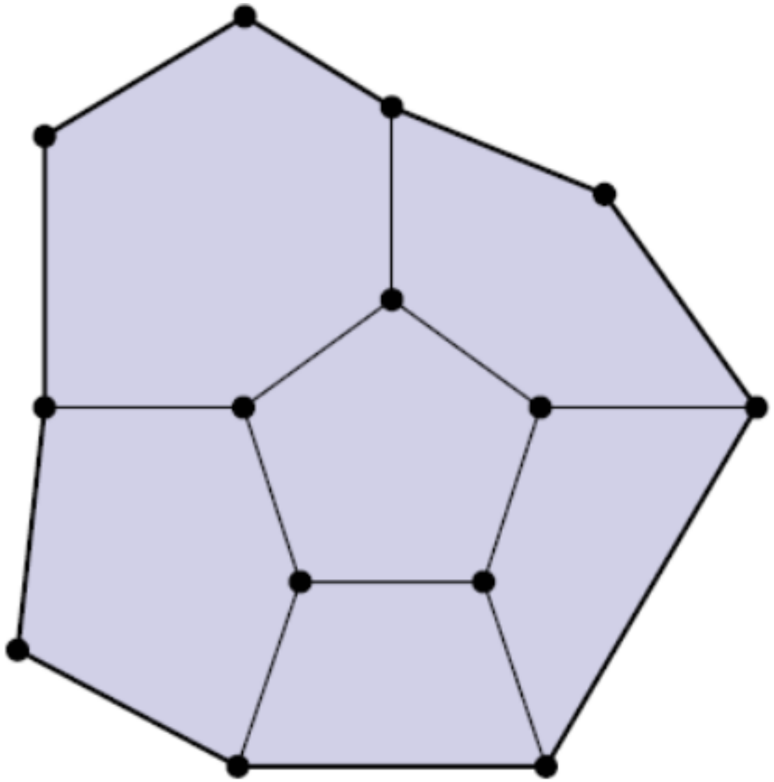
A collection of tetrahedrons

Ok, but why is the manifold assumption *useful?*

Keep it Simple!

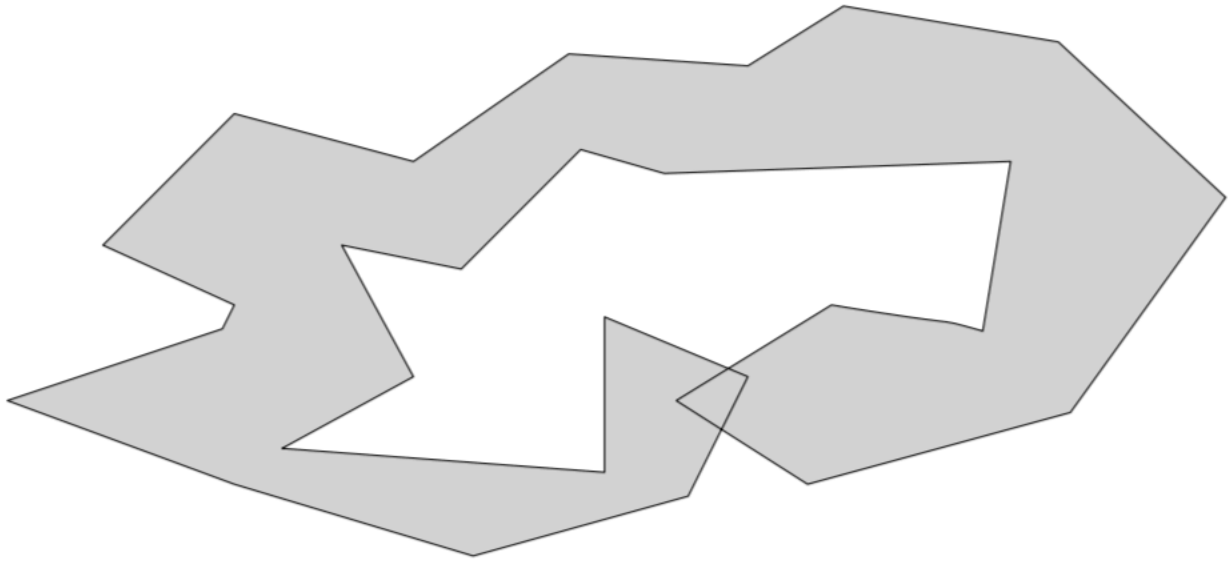
- **Same motivation as for images:**
  - **make some assumptions about our geometry to keep data structures/algorithms simple and efficient**
  - **in *many common cases*, doesn't fundamentally limit what we can do with geometry**





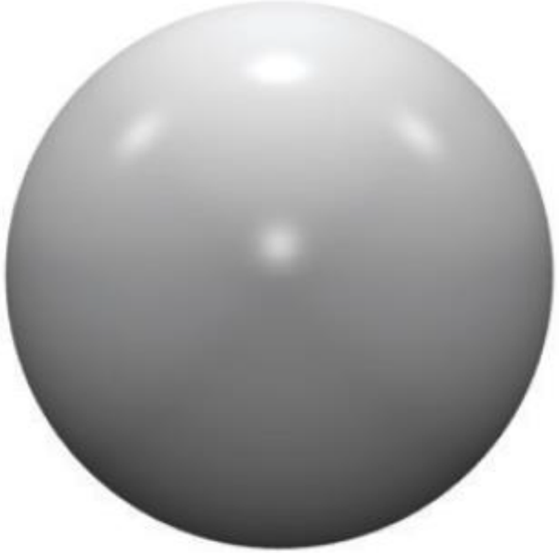
Geometric validity

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



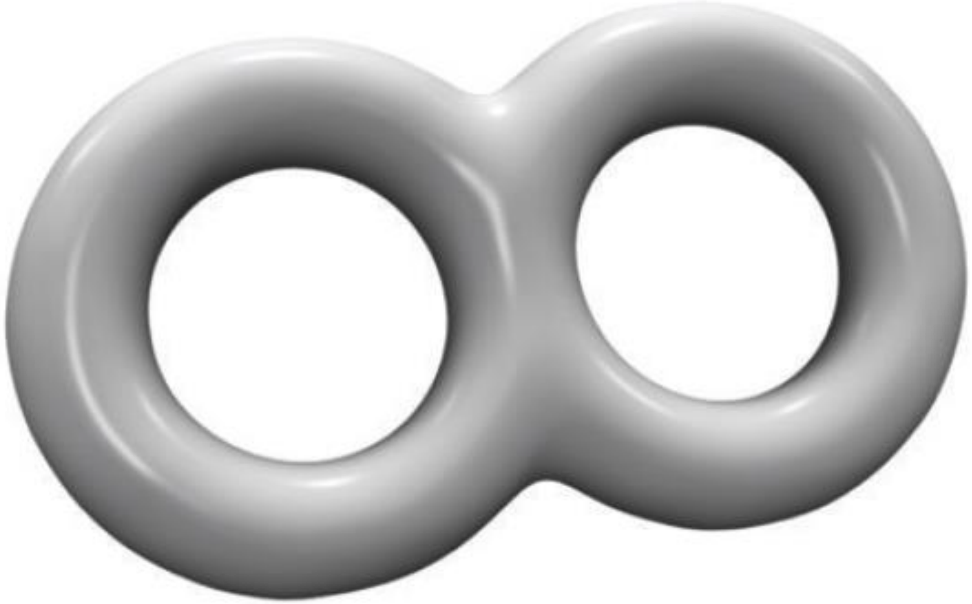
Global Topology: **Genus**

- Genus: Maximal number of closed simple cutting curves that do not disconnect the graph into multiple components.



g=0

g=1



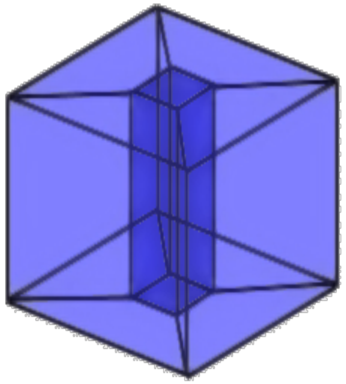
g=2

A disc (plane with boundary) / planar graph has genus zero

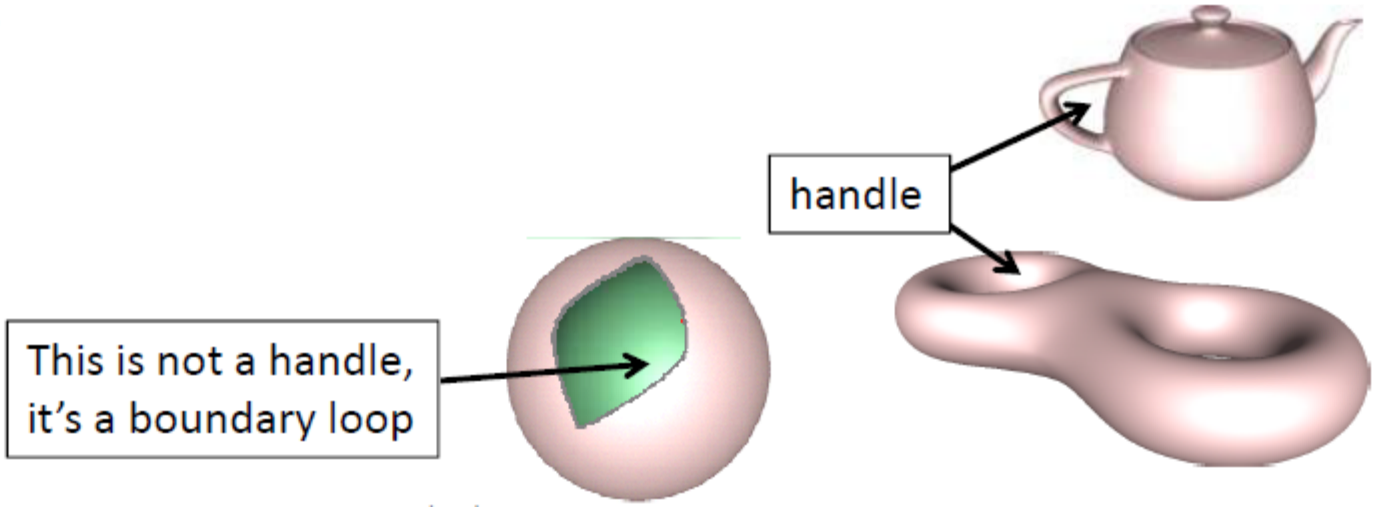
Euler-Poincaré Formula

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

- **Euler characteristic χ= V-E+F=2(C-G)-B**
  - V : number of vertices
  - E : number of edges
  - F : number of faces
  - C : number of connected components
  - G : number of genus (holes, handles)
  - B : number of boundaries
- Euler Formula: V-E+F = 2 when C=1, G=0

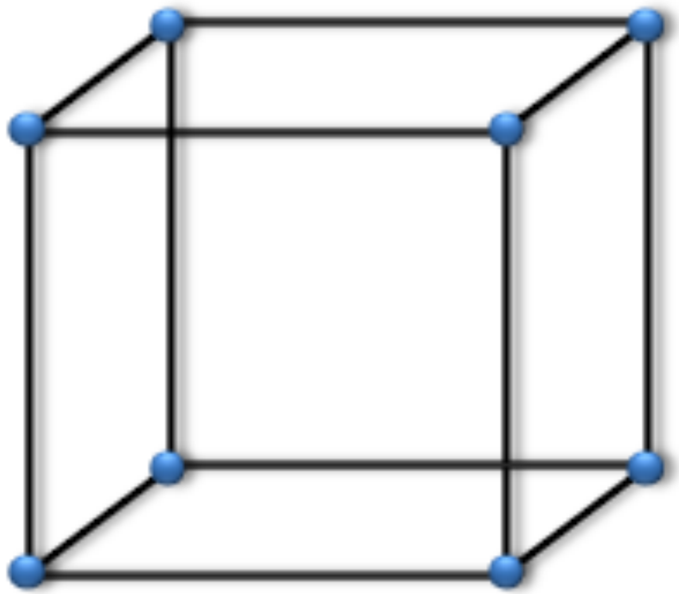


V = 16
E = 32
F = 16
C = 1
G = 1
B = 0
16 – 32 + 16 = 2 (1 - 1) - 0

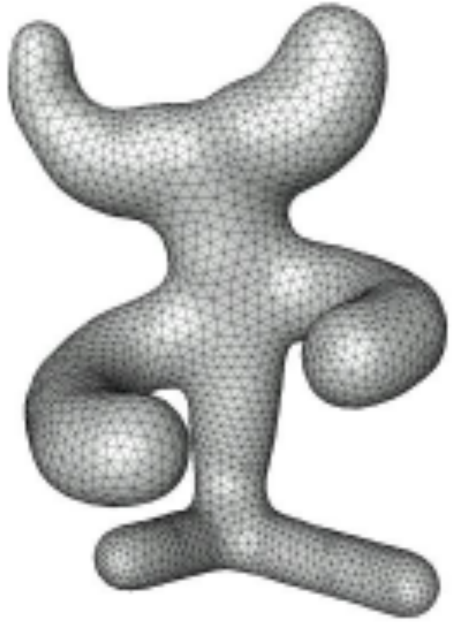


Euler Formula V-E+F = 2

Euler formula for planar graphs help us derive cool mesh statistics.



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



$$V = 3890$$
$$E = 11664$$
$$F = 7776$$
$$\chi = \mathbf{2}$$

Average Valence of Closed Triangle Mesh

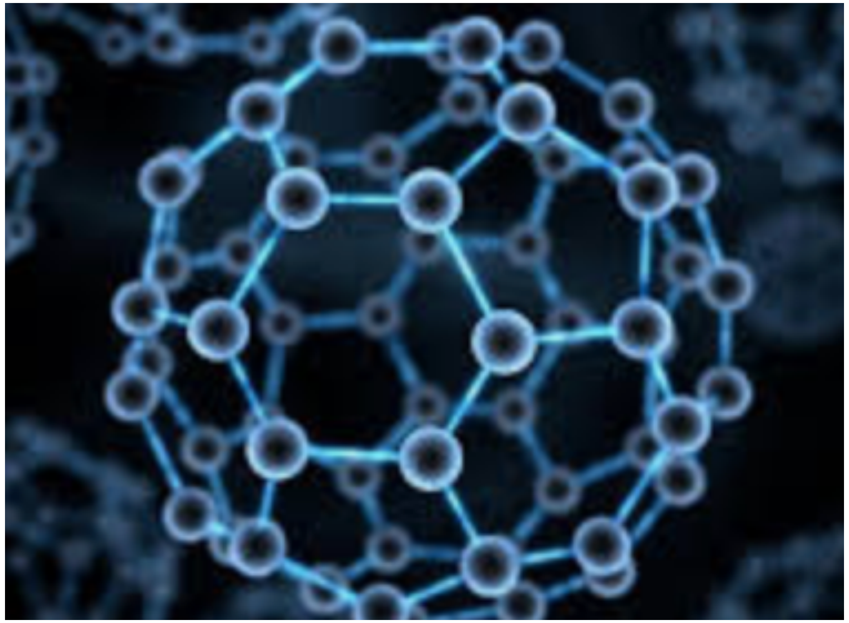
- **Theorem:** For any closed manifold triangle mesh
  - ✓F ~ 2V
  - ✓E ~ 3V
  - ✓Average vertex degree D is 6.
- **Proof:**
  - Each face has 3 edges & each edge is counted twice: 3F = 2E
  - by Euler's formula: 2=V-E+F = V-3/2F+F=V-1/2F => F=2V-4 ~2V for large V
  - Similar approach => E ~3V

  - DV=2E => D=?
  - by Euler's formula: V+F-E = V+2E/3-E = 2-2g
  - Thus E = 3(V-2+2g)

  - => D = 2E/V = 6(V-2+2g)/V ~ 6 for large V

How many pentagons六边形?

- every **vertex has valence 3**



fullerene (carbon)

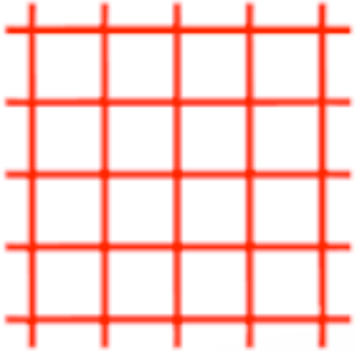
Euler Consequences

Triangle mesh statistics

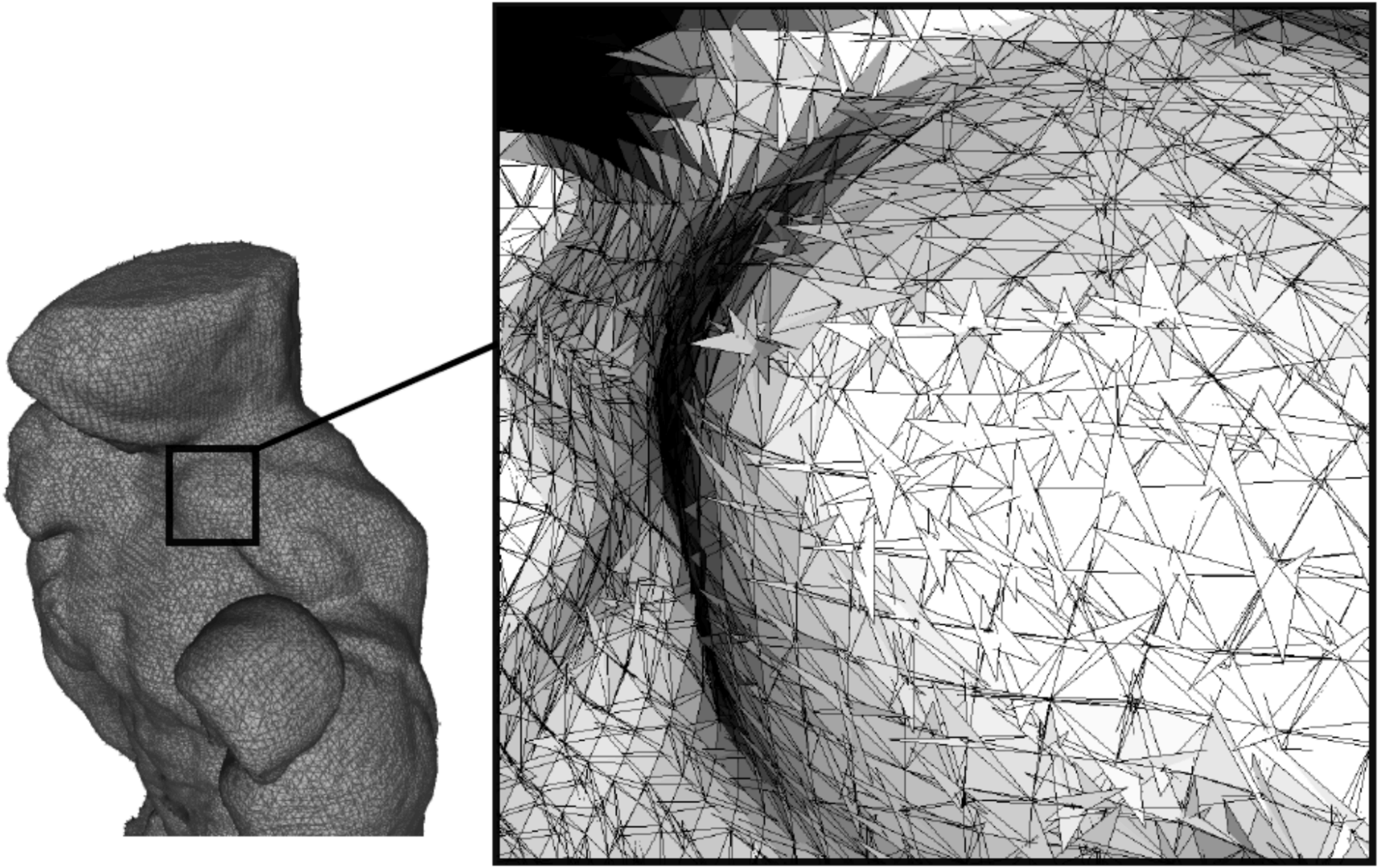
- $F \approx 2V$
- $E \approx 3V$
- Average valence = 6

Quad mesh statistics

- $F \approx V$
- $E \approx 2V$
- Average valence = 4



How do we actually encode all this data?

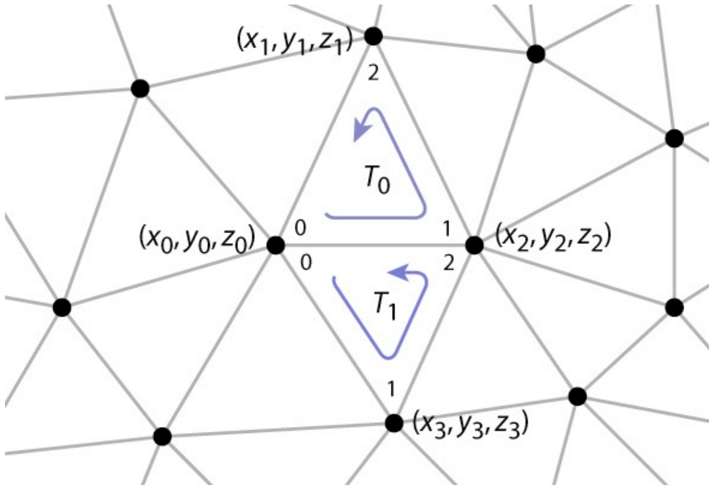


Face set (STL) - Polygon Soups / Separate triangles

	[0]	[1]	[2]
tris[0]	x_0, y_0, z_0	x_2, y_2, z_2	x_1, y_1, z_1
tris[1]	x_0, y_0, z_0	x_3, y_3, z_3	x_2, y_2, z_2
	⋮	⋮	⋮

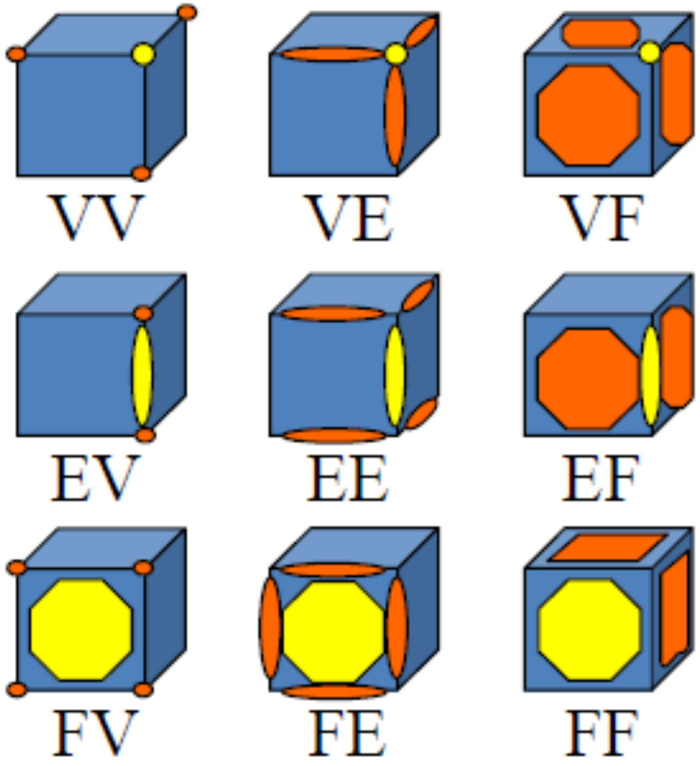
- **array of triples of points**
  - float[*nf*][3][3]: about 72 bytes per vertex
    - 4 bytes per coordinate (float)
    - 3 coordinates per vertex
    - 3 vertices per triangle => 36 byte per face
    - 2 triangles per vertex (on average, Euler Consequences: |F|~2|V|)

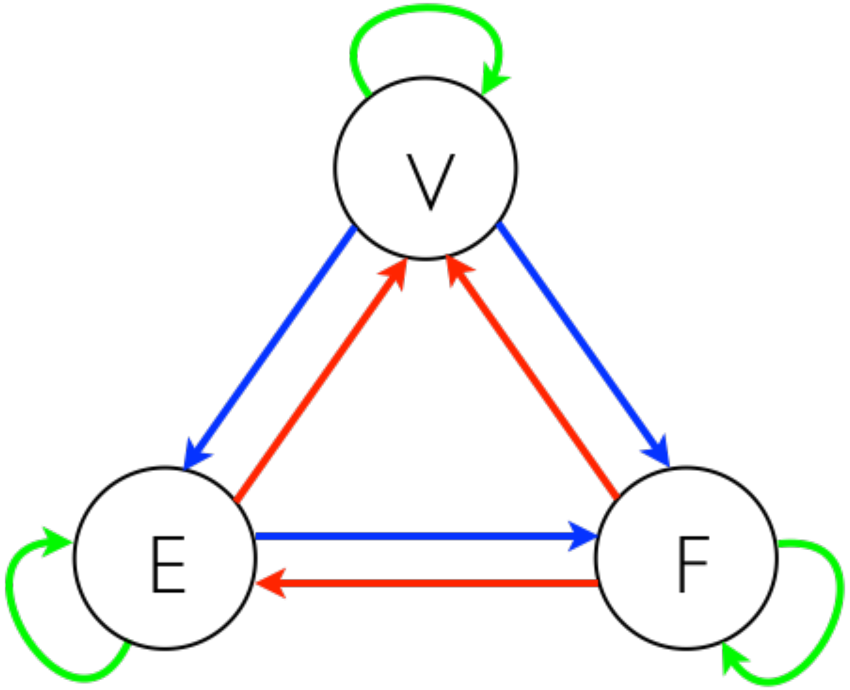
- **various problems**
  - wastes space (each vertex stored 6 times)
  - cracks due to roundoff
  - difficulty of finding neighbors at all



Neighborhood relations [Weiler 1985]

1.	Vertex	– Vertex	VV
2.	Vertex	– Edge	VE
3.	Vertex	– Face	VF
4.	Edge	– Vertex	EV
5.	Edge	– Edge	EE
6.	Edge	– Face	EF
7.	Face	– Vertex	FV
8.	Face	– Edge	FE
9.	Face	– Face	FF

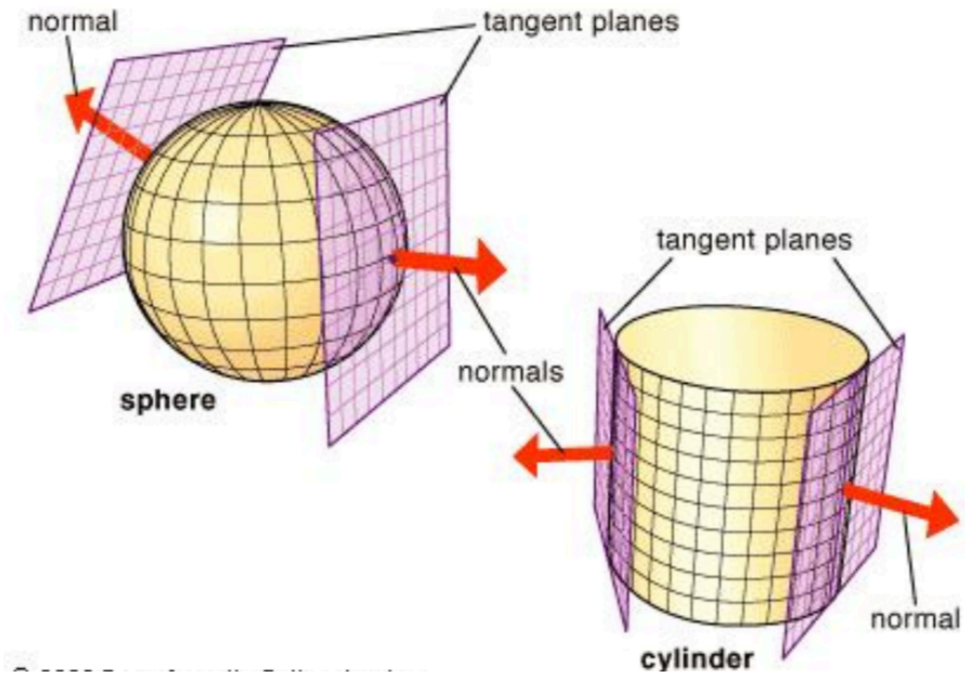




Knowing some types of relation, we can discover other (but not necessary all) topological information
e.g. if in addition to VV, VE and VF, we know neighboring vertices of a face, we can discover all neighboring edges of the face

Data Structures

- **What should be stored?**
  - Geometry: 3D vertex coordinates
  - Connectivity: Vertex adjacency
  - Attributes:
    - normals, color, texture coordinates, etc.
    - Per Vertex, per face, per edge

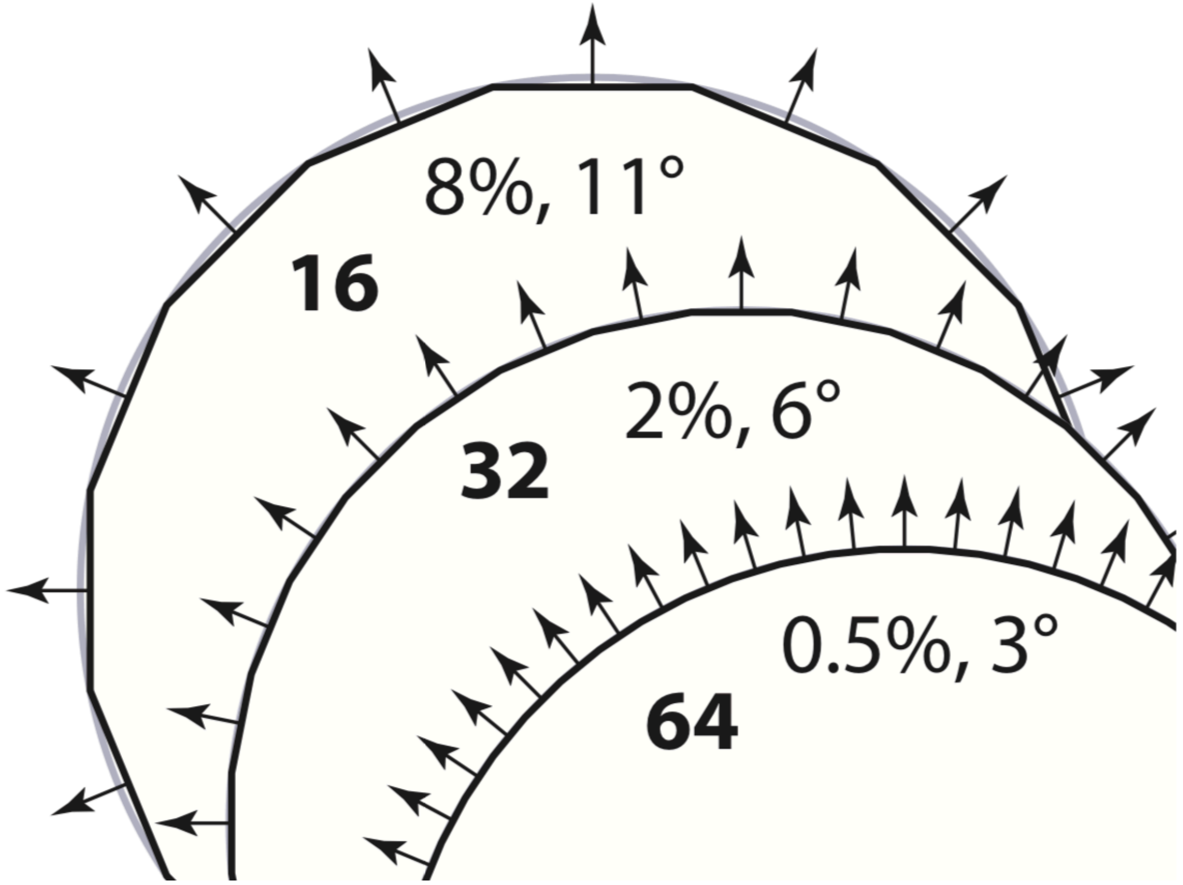


How to think about vertex normals

- **Piecewise planar approximation converges pretty quickly to the smooth geometry as the number of triangles increases**
  - For mathematicians: error is O(h2)
- **But the surface normals don't converge so well**
  - normal is constant over each triangle, with discontinuous jumps across edges
  - for mathematicians: error is only O(h)
- **Better: store the "real" normal at each vertex, and *interpolate* to get normals that vary gradually across triangles**

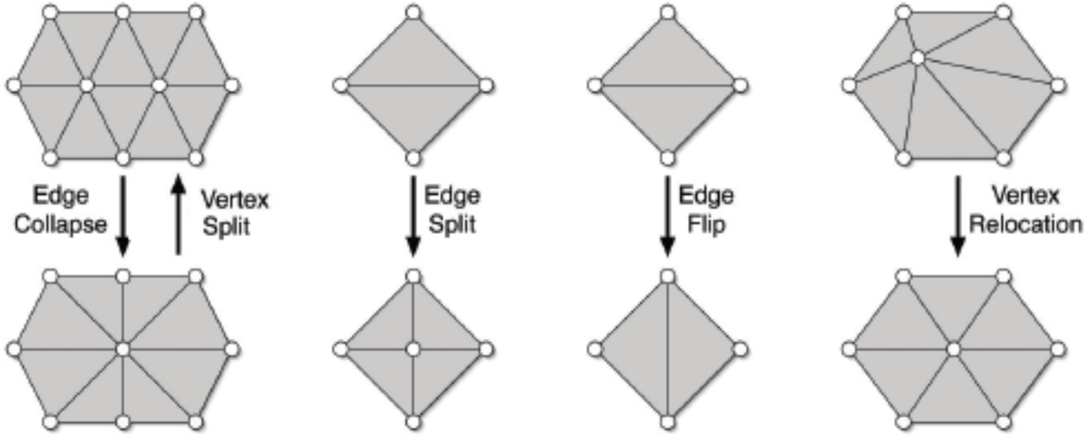
Interpolated normals—2D example

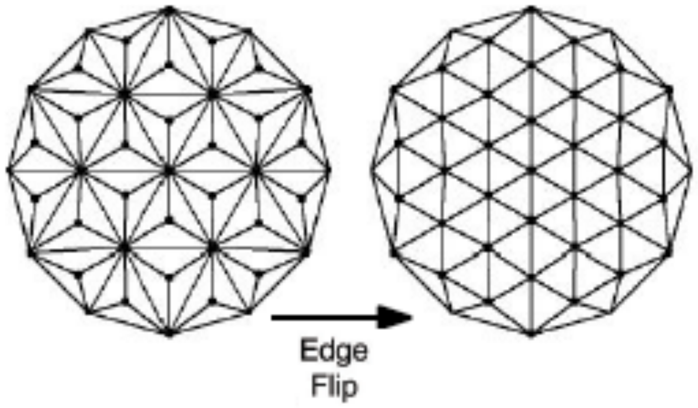
- Approximating circle with increasingly many segments
- Max error in position error drops by factor of 4 at each step
- Max error in normal only drops by factor of 2



Mesh Data Structures

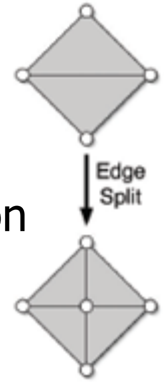
- **How to store geometry & connectivity?**
  - **Compact storage and file formats**
  - **Efficient algorithms on meshes**
    - Rendering
    - Queries
      - What are the vertices of face #3?
      - Is vertex #6 adjacent to vertex #12?
      - Which faces are adjacent to face #7?
    - Modifications
      - Remove/add a vertex/face
      - Vertex split, edge collapse

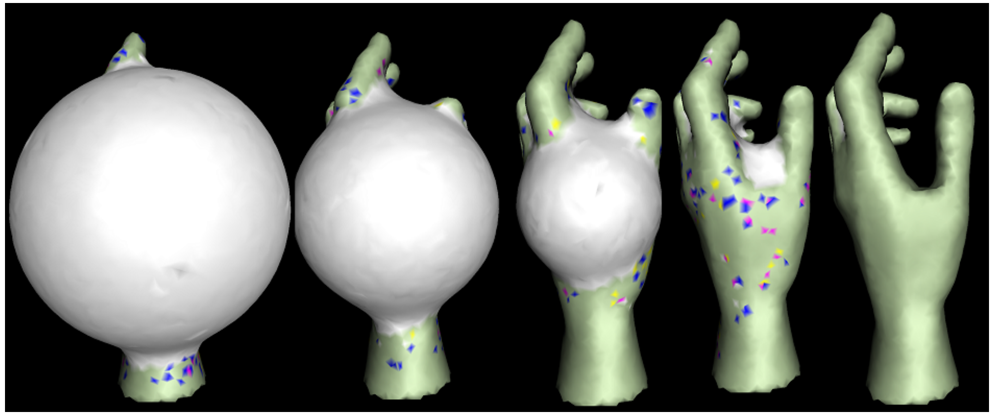




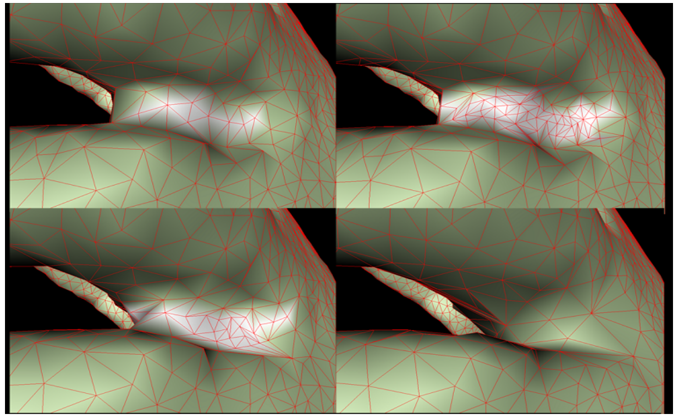
Mesh Data Structures

- ✓ Applications of edge split:
- ✓ Increase resolution to catch details in 3D reconstruction
  - ✓ Paper: Shape from silhouette using topology-adaptive mesh deformation



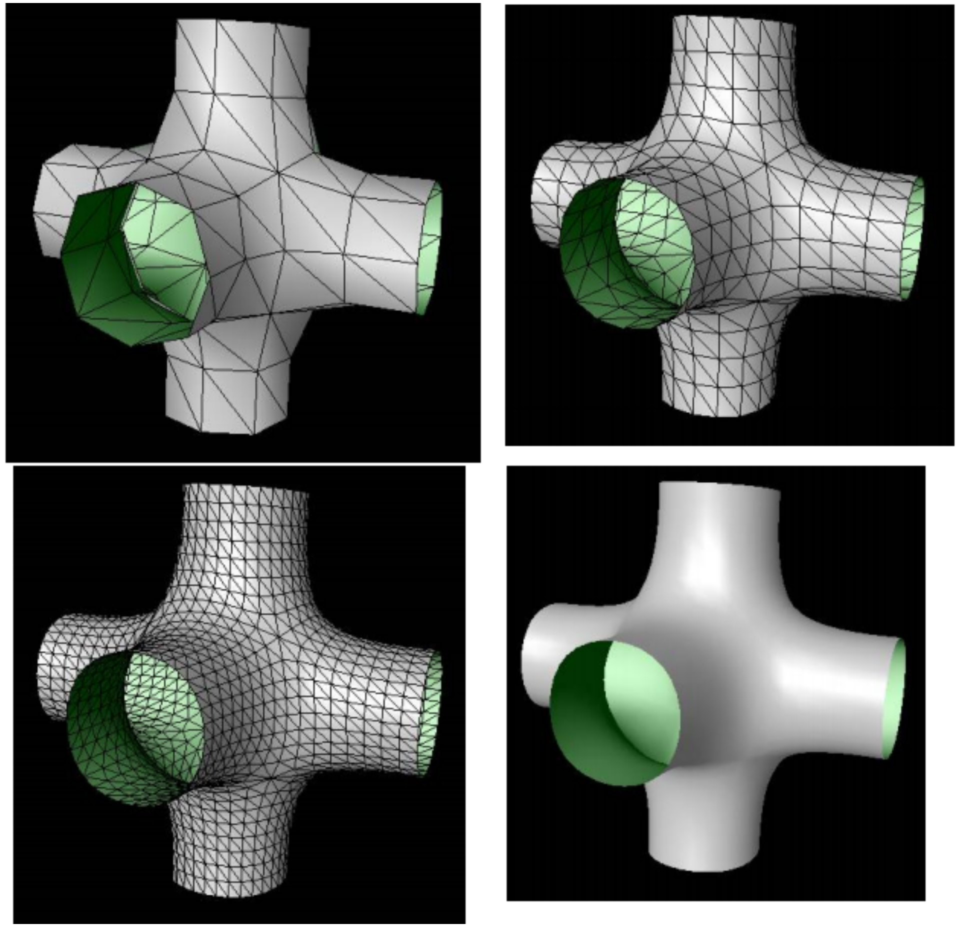


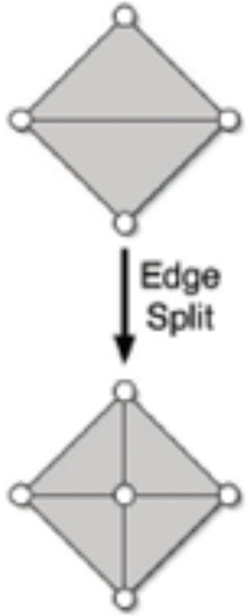
- ✓ Split short edge if midpoint is OUT:



Mesh Data Structures

- ✓ Applications of edge split:
- ✓ Increase resolution for smoother surfaces: Subdivision Surfaces
  - ✓ Loop subdivision

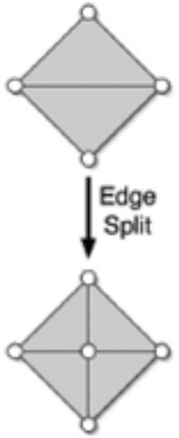




- ✓ 32 (original) to 1628 vertices in 3 iterations:

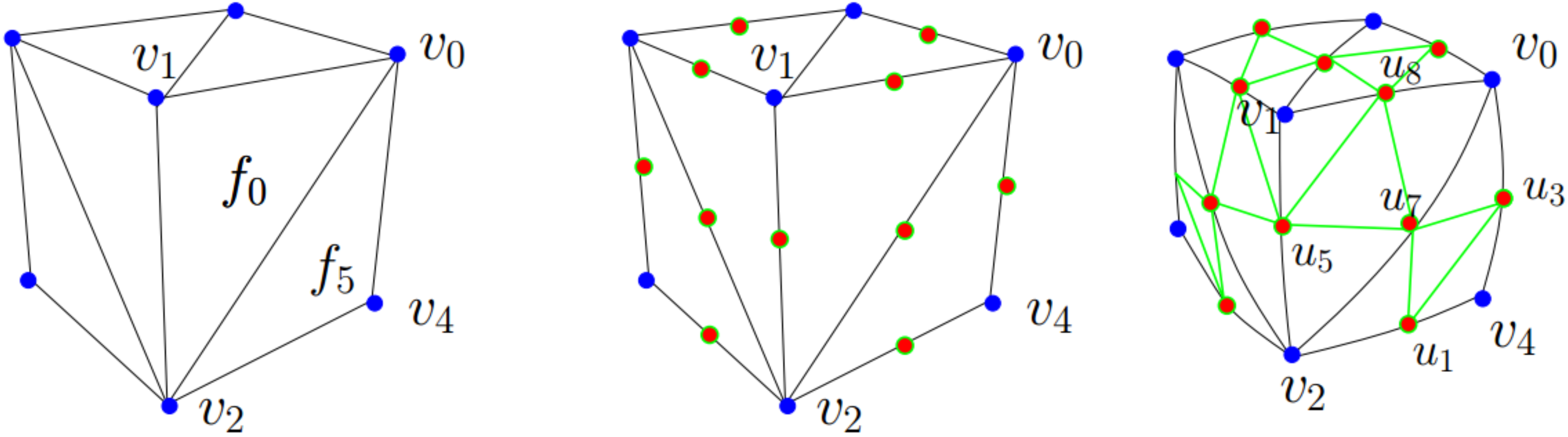
Mesh Data Structures

- ✓ Applications of edge split:
- ✓ Increase resolution for smoother surfaces: Subdivision Surfaces
  - ✓ Loop subdivision
  - ✓ Updating the topology (connectivity)

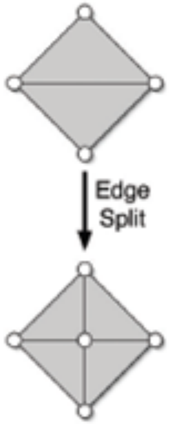


split all edges, by inserting a midpoint

subdivide each face into 4 triangles

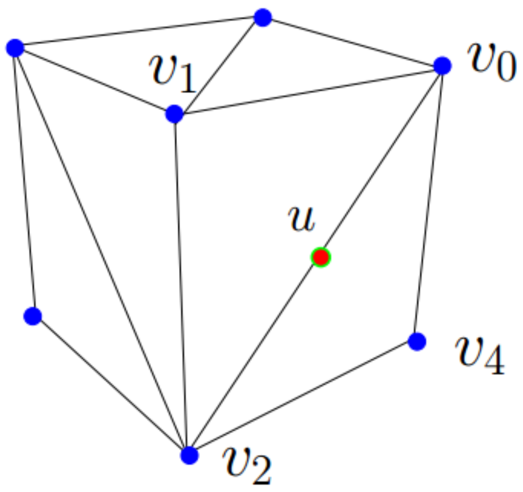


Mesh Data Structures



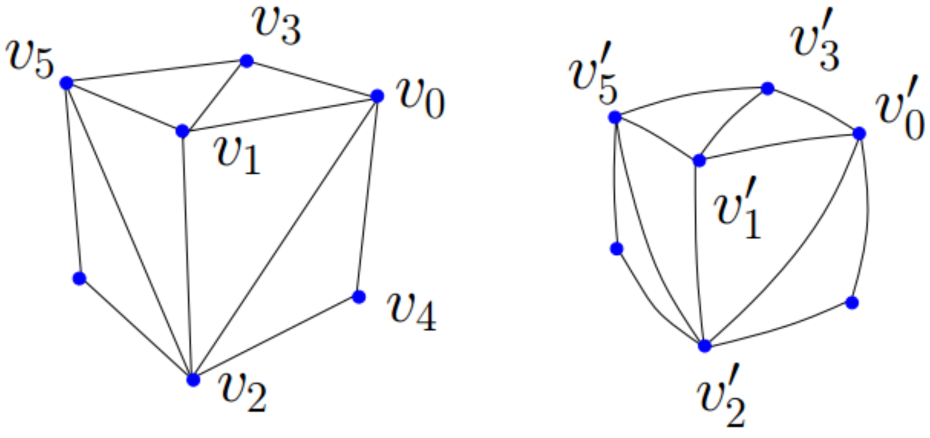
- ✓ Applications of edge split:
- ✓ Increase resolution for smoother surfaces: Subdivision Surfaces
  - ✓ Loop subdivision
  - ✓ Updating the geometry (coordinates)

First compute edge points u_k



$$u = \frac{3}{8}v_0 + \frac{3}{8}v_2 + \frac{1}{8}v_1 + \frac{1}{8}v_4$$

Compute new locations v_i' of initial vertices



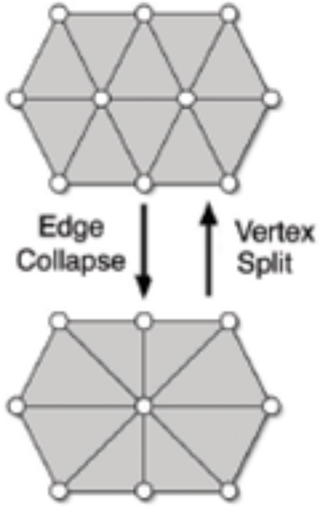
$$v_i' = (1 - \alpha d)v_i + \alpha \sum_{j=1}^{d} v_{i_j}$$

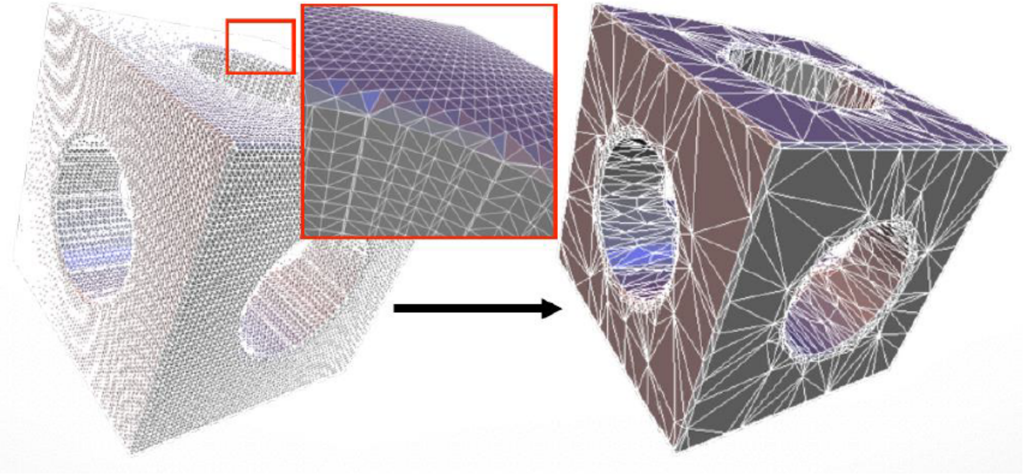
d is the *degree* of vertex v_i
v_{i_j} is the j-th neighbor of v_i

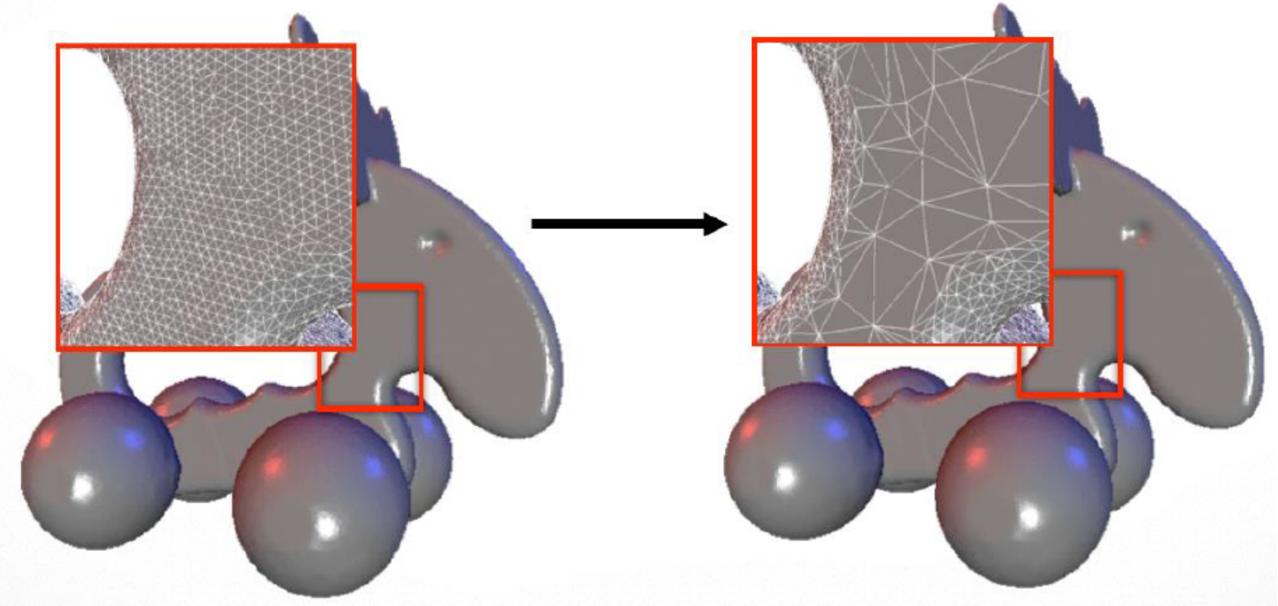
$$\left[\begin{array}{l} \alpha = \frac{3}{16}, \text{ if } d = 3 \\ \alpha = \frac{3}{8d}, \text{ if } d > 3 \end{array}\right.$$

Mesh Data Structures

- ✓ Applications of edge collapse:
- ✓ Decrease resolution for efficiency
  - ✓ Detail-preserving

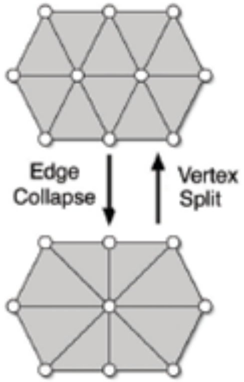


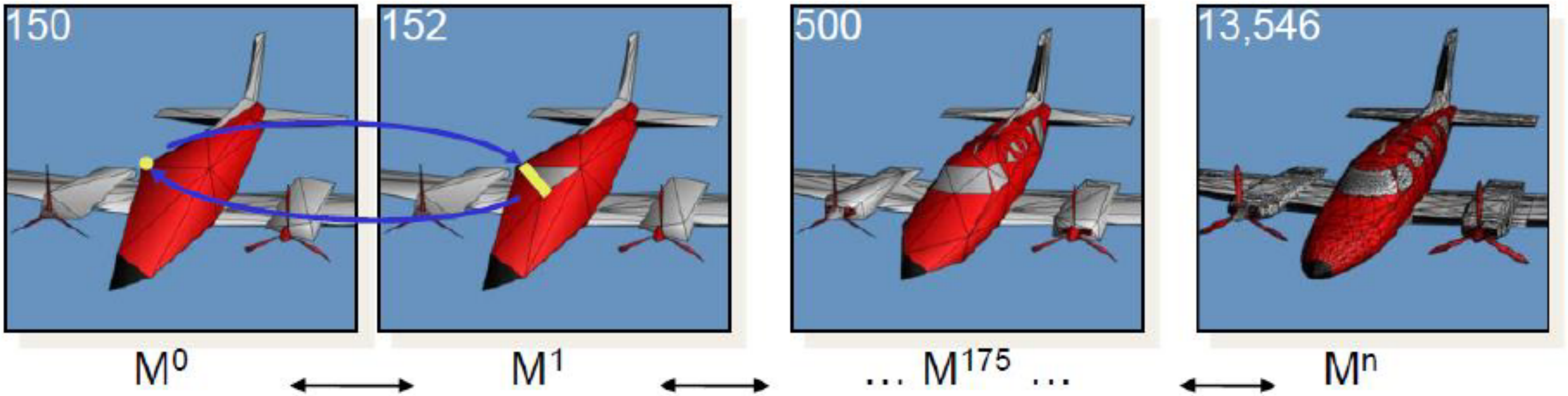




Mesh Data Structures

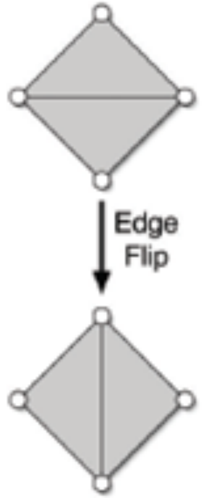
- ✓ Applications of edge collapse:
- ✓ Decrease resolution for efficiency
  - ✓ Detail-oblivious (level-of-detail)

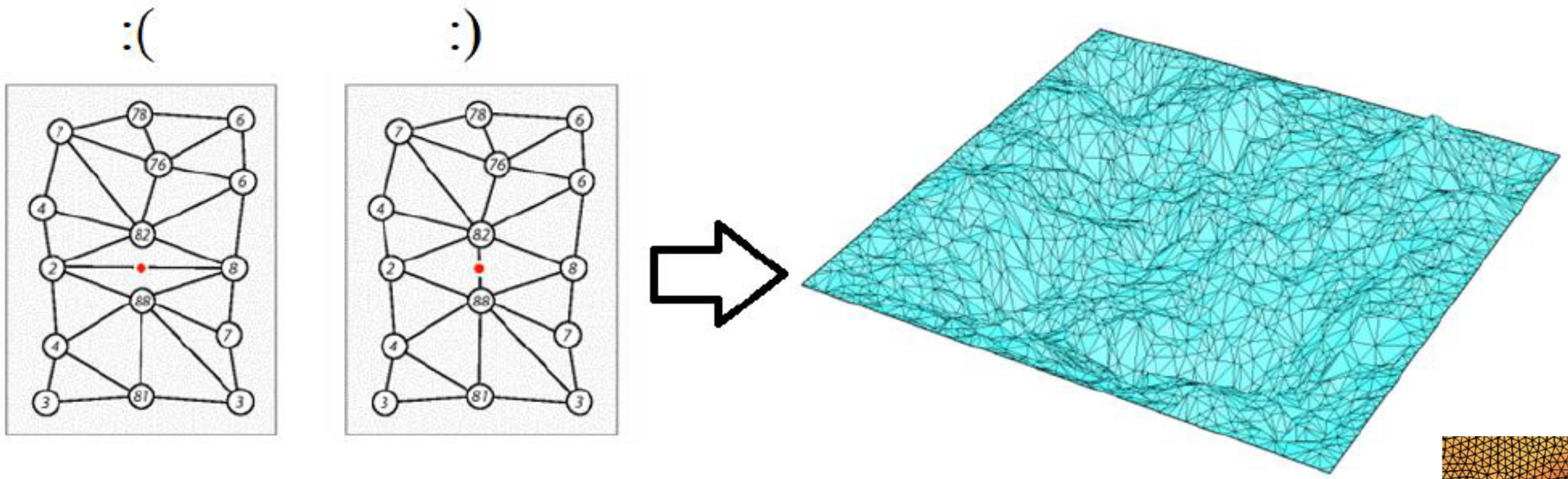


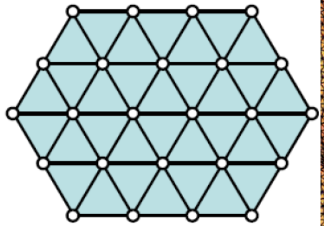


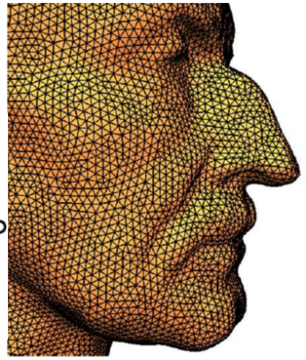
Mesh Data Structures

- ✓ Applications of edge flip:
- ✓ Better triangulations, e.g., w/ less skinny triangles
- ✓ Finite element modeling, simulations, terrain construction









Different Data Structures

- Time to construct (preprocessing)
- Time to answer a query
  - Random access to vertices/edges/faces
  - Fast mesh traversal
  - Fast Neighborhood query
- Time to perform an operation
  - split/merge
- Space complexity
- Redundancy
- Most important ones are face and edge-based (since they encode connectivity)

Mesh Representations

- **Face-vertex meshes**
  - **Problem: different topological structure for triangles and quadrangles**
- **Winged-edge meshes**
  - **Problem: traveling the neighborhood requires one case distinction**
- **Half-edge meshes**
- **Quad-edge meshes, Corner-tables, Vertex-vertex meshes, …**
- **LR (*Laced Ring*): more compact than halfedge [siggraph2011: compact connectivity representation for triangle meshes]**
  - **Suited for processing meshes with fixed connectivity**

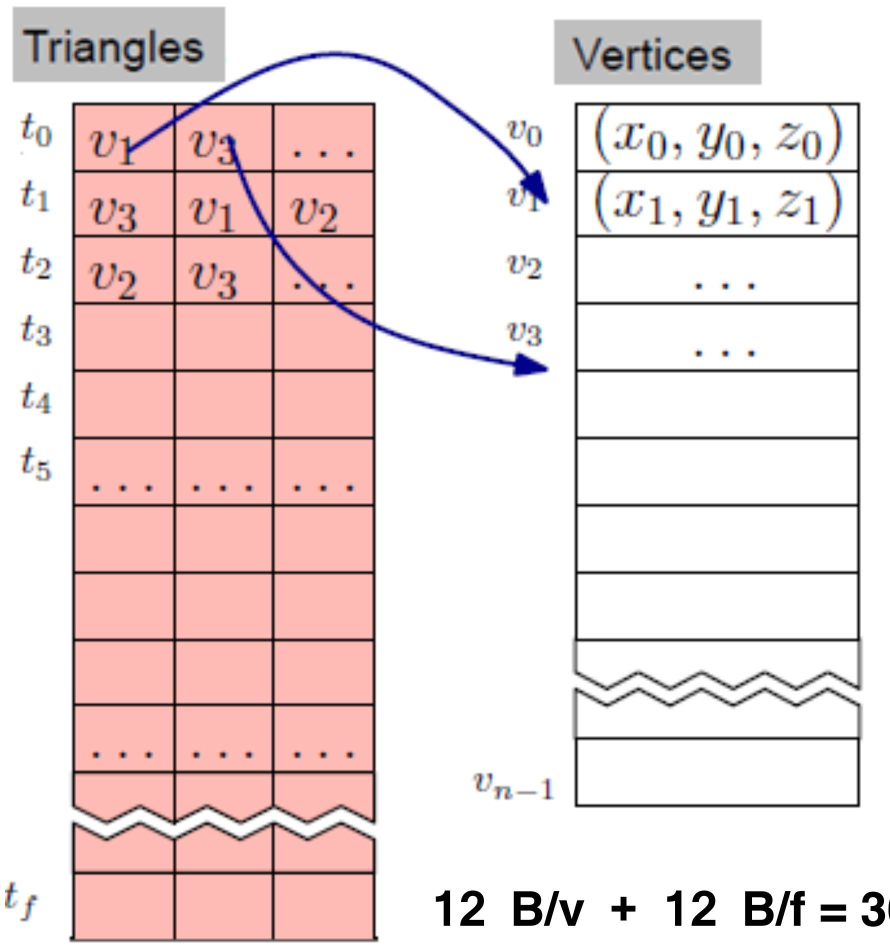
Mesh Representations

- Choice
  - Each of the representations above have particular advantages & drawbacks
  - Choice is governed by
    - Application,
    - Performance required,
    - Size of the data,
    - and Operations to be performed.

- Example
  - it is easier to deal with triangles than general polygons, especially in computational geometry.
  - For certain operations it is necessary to have a fast access to topological information such as edges or neighboring faces; this requires more complex structures such as half-edge representation.
  - For hardware rendering, compact, simple structures are needed; thus the corner-table (triangle fan) is commonly incorporated into low-level rendering APIs such as DirectX and OpenGL.

Indexed Face set - Shared Vertex (OBJ,OFF)

- **Store each vertex once**
- **Each triangle points to its three vertices**



12 B/v + 12 B/f = 36B/v

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}

Face-Set data structure with various problems

- wastes space (each vertex stored 6 times)
- cracks due to roundoff
- difficulty of finding neighbors at all

Transversal operations

- Most operations are slow for the connectivity info is not explicit.
- Need a more efficient representation

iterate over / collect adjacent	V	E	F
V	quadratic	quadratic	linear
E	quadratic	quadratic	linear
F	quadratic	quadratic	linear

Example1: Iterate {fi}; find fi’ s vertices for computing face normal: linear operations

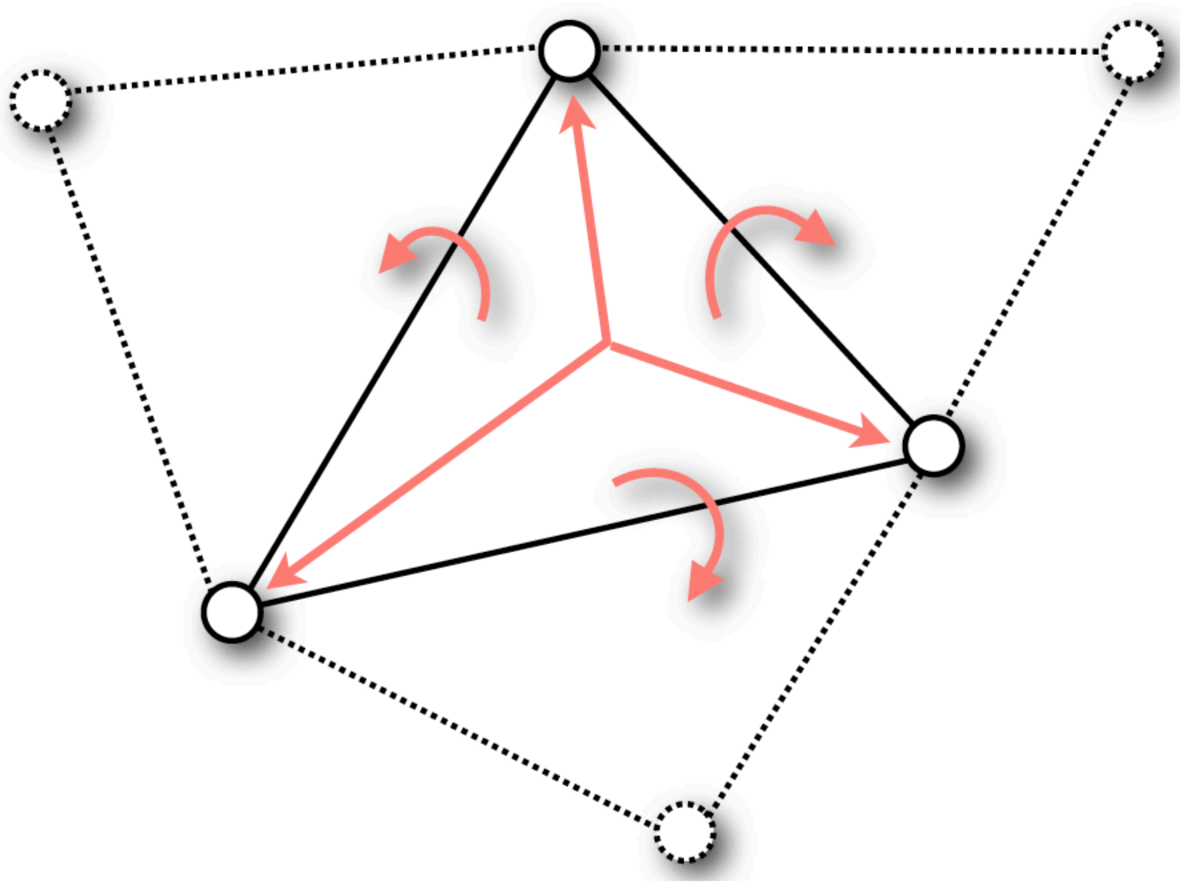
1. Iterate {fi}: O(|F|), |F|~|2V|, so O(V);
2. For each fi, find its vertices: O(1).

Example2: Iterate {vi}; find 1-ring vertex neighbors of each vi to compute Laplacian or averaging some vertex property: quadratic operations

1. Iterate {vi}: O(V);
2. For vi, search {fi} to find all faces {fj}’ containing vi: O(|F|), |F|~|2V|, so O(V);
3. For each fj of vi’ s 1-ring faces, find vi’ s 1-ring vertices: O(1).

Face-Based Connectivity

- Vertex:
  - position
  - 1 face
- Face:
  - 3 vertices
  - 3 face neighbors



12(v position4*3) + 12*2(f vertices4*3) + 4(v 1 face) + 12*2(f 3face neighbors)=64 B/v

Face-vertex meshes

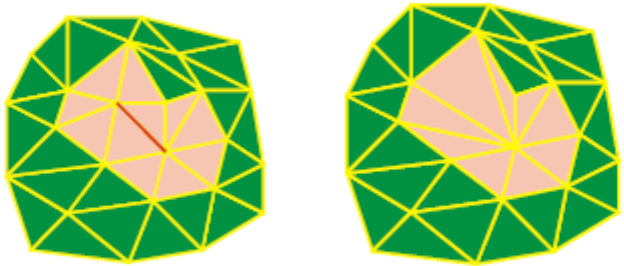
1. locating neighboring faces and vertices is constant time
2. a search is still needed to find all the faces surrounding a given face.
3. Other dynamic operations, such as splitting or merging a face, are also difficult with face-vertex meshes.

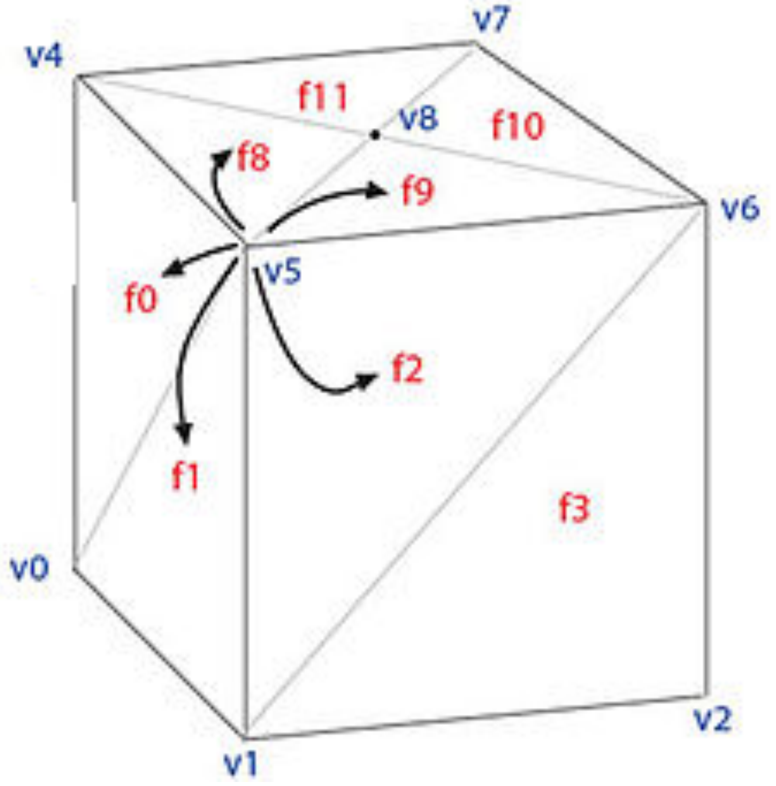
Face List

f0	v0 v4 v5
f1	v0 v5 v1
f2	v1 v5 v6
f3	v1 v6 v2
f4	v2 v6 v7
f5	v2 v7 v3
f6	v3 v7 v4
f7	v3 v4 v0
f8	v8 v5 v4
f9	v8 v6 v5
f10	v8 v7 v6
f11	v8 v4 v7
f12	v9 v5 v4
f13	v9 v6 v5
f14	v9 v7 v6
f15	v9 v4 v7

Vertex List

v0	0,0,0	f0 f1 f12 f15 f7
v1	1,0,0	f2 f3 f13 f12 f1
v2	1,1,0	f4 f5 f14 f13 f3
v3	0,1,0	f6 f7 f15 f14 f5
v4	0,0,1	f6 f7 f0 f8 f11
v5	1,0,1	f0 f1 f2 f9 f8
v6	1,1,1	f2 f3 f4 f10 f9
v7	0,1,1	f4 f5 f6 f11 f10
v8	.5,.5,0	f8 f9 f10 f11
v9	.5,.5,1	f12 13 14 15



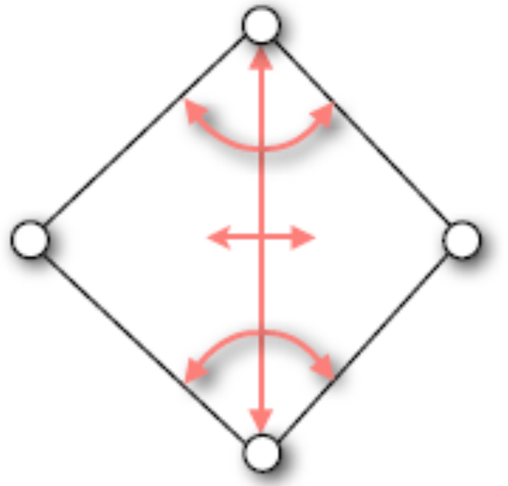


Edges always have the same topological structure

Efficient handling of polygons with variable valence

(Winged) **Edge-Based Connectivity**

- **Vertex:**
  - position
  - 1 edge
- **Edge:**
  - 2 vertices
  - 2 faces
  - 4 edges
- **Face:**
  - 1 edge

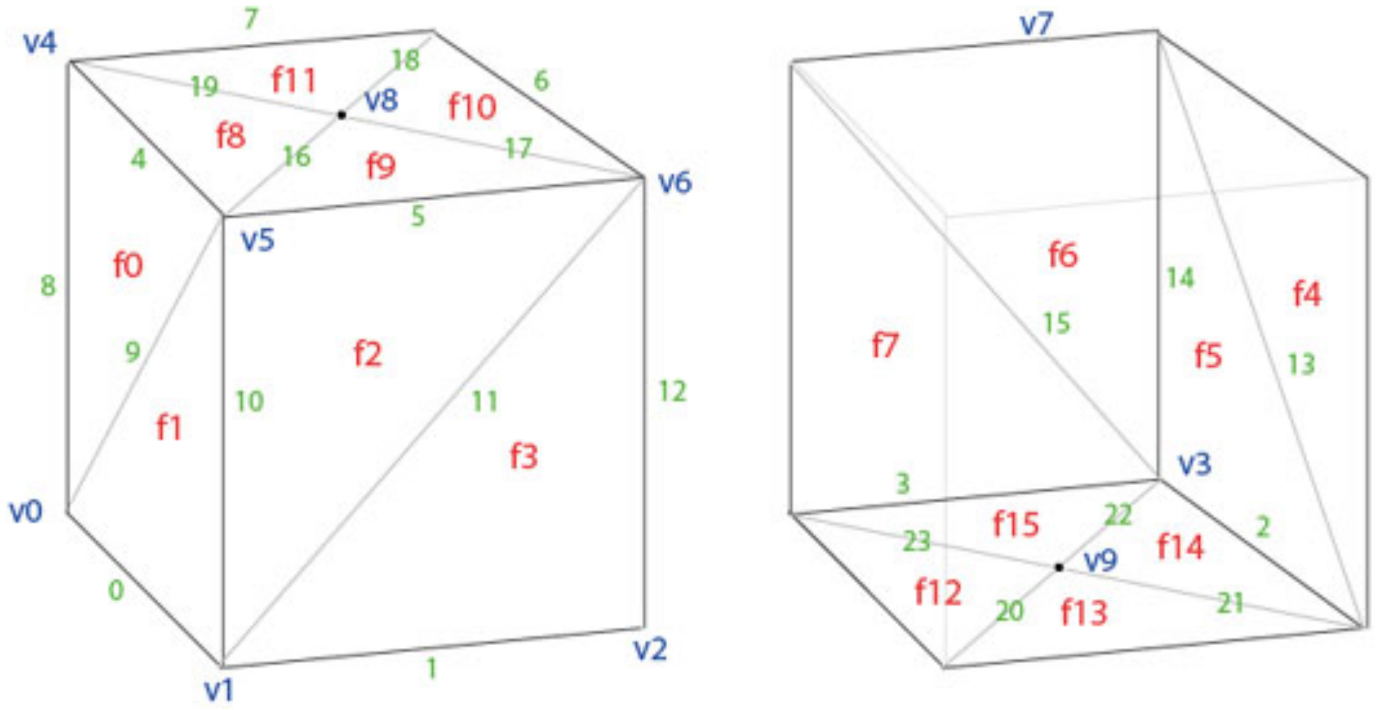


120 B/v

Edges have no orientation: special case handling for neighbors

Winged-edge meshes

- explicitly represent the vertices, faces, and edges of a mesh.
- greatest flexibility in dynamically changing the mesh
- large storage requirements and increased complexity due to maintaining many indic…



Winged-edge meshes

Face List

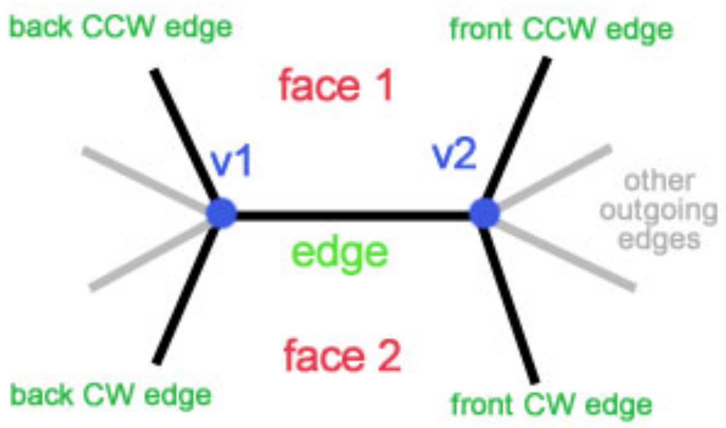
f0	4 8 9
f1	0 10 9
f2	5 10 11
f3	1 12 11
f4	6 12 13
f5	2 14 13
f6	7 14 15
f7	3 8 15
f8	4 16 19
f9	5 17 16
f10	6 18 17
f11	7 19 18
f12	0 23 20
f13	1 20 21
f14	2 21 22
f15	3 22 23

Edge List

e0	v0 v1	f1 f12	9 23 10 20
e1	v1 v2	f3 f13	11 20 12 21
e2	v2 v3	f5 f14	13 21 14 22
e3	v3 v0	f7 f15	15 22 8 23
e4	v4 v5	f0 f8	19 8 16 9
e5	v5 v6	f2 f9	16 10 17 11
e6	v6 v7	f4 f10	17 12 18 13
e7	v7 v4	f6 f11	18 14 19 15
e8	v0 v4	f7 f0	3 9 7 4
e9	v0 v5	f0 f1	8 0 4 10
e10	v1 v5	f1 f2	0 11 9 5
e11	v1 v6	f2 f3	10 1 5 12
e12	v2 v6	f3 f4	1 13 11 6
e13	v2 v7	f4 f5	12 2 6 14
e14	v3 v7	f5 f6	2 15 13 7
e15	v3 v4	f6 f7	14 3 7 15
e16	v5 v8	f8 f9	4 5 19 17
e17	v6 v8	f9 f10	5 6 16 18
e18	v7 v8	f10 f11	6 7 17 19
e19	v4 v8	f11 f8	7 4 18 16
e20	v1 v9	f12 f13	0 1 23 21
e21	v2 v9	f13 f14	1 2 20 22
e22	v3 v9	f14 f15	2 3 21 23
e23	v0 v9	f15 f12	3 0 22 20

Vertex List

v0	0,0,0	8 9 0 23 3
v1	1,0,0	10 11 1 20 0
v2	1,1,0	12 13 2 21 1
v3	0,1,0	14 15 3 22 2
v4	0,0,1	8 15 7 19 4
v5	1,0,1	10 9 4 16 5
v6	1,1,1	12 11 5 17 6
v7	0,1,1	14 13 6 18 7
v8	.5,.5,0	16 17 18 19
v9	.5,.5,1	20 21 22 23



Winged Edge Structure

Render dynamic meshes

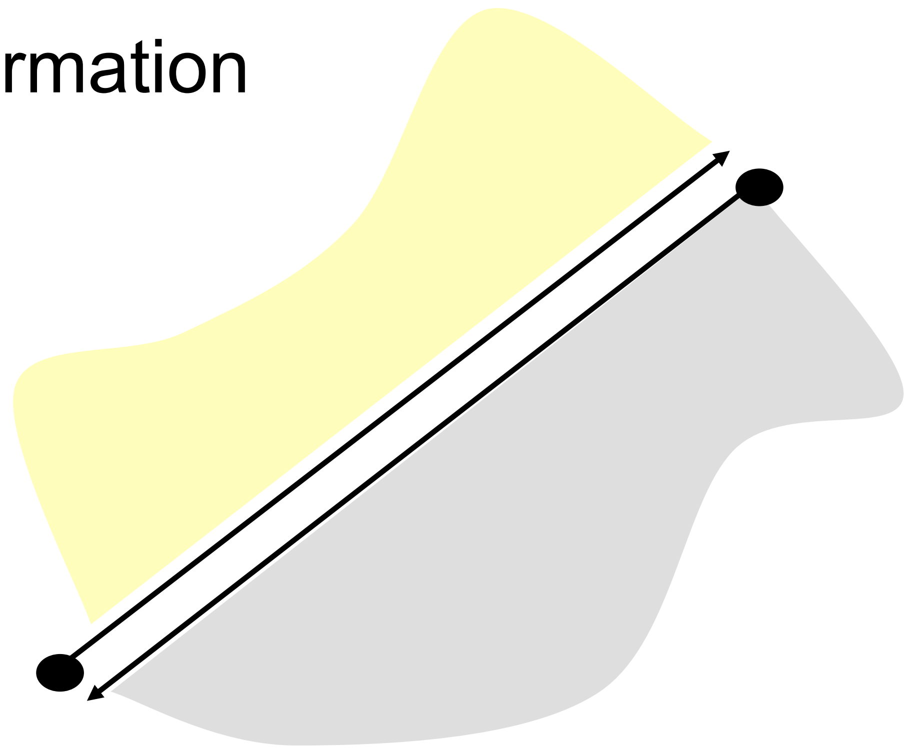
- combines winged-edge meshes and face-vertex meshes
- require slightly less storage space than standard winged-edge meshes,
- and can be directly rendered by graphics hardware since the face list contains an index of vertices.

Operation		Vertex-vertex	Face-vertex	Winged-edge	Render dynamic
V-V	All vertices around vertex	Explicit	V → f1, f2, f3, ... → v1, v2, v3, ...	V → e1, e2, e3, ... → v1, v2, v3, ...	V → e1, e2, e3, ... → v1, v2, v3, ...
E-F	All edges of a face	F(a, b, c) → {a, b}, {b, c}, {a, c}	F → {a, b}, {b, c}, {a, c}	Explicit	Explicit
V-F	All vertices of a face	F(a, b, c) → {a, b, c}	Explicit	F → e1, e2, e3 → a, b, c	Explicit
F-V	All faces around a vertex	Pair search	Explicit	V → e1, e2, e3 → f1, f2, f3, ...	Explicit
E-V	All edges around a vertex	V → {v, v1}, {v, v2}, {v, v3}, ...	V → f1, f2, f3, ... → v1, v2, v3, ...	Explicit	Explicit
F-E	Both faces of an edge	List compare	List compare	Explicit	Explicit
V-E	Both vertices of an edge	E(a, b) → {a, b}	E(a, b) → {a, b}	Explicit	Explicit
Flook	Find face with given vertices	F(a, b, c) → {a, b, c}	Set intersection of v1, v2, v3	Set intersection of v1, v2, v3	Set intersection of v1, v2, v3
Storage size		V*avg(V, V)	3F + V*avg(F, V)	3F + 8E + V*avg(E, V)	6F + 4E + V*avg(E, V)
		Example with 10 vertices, 16 faces, 24 edges:			
		10 * 5 = 50	3*16 + 10*5 = 98	3*16 + 8*24 + 10*5 = 290	6*16 + 4*24 + 10*5 = 242

Figure 6: summary of mesh representation operations

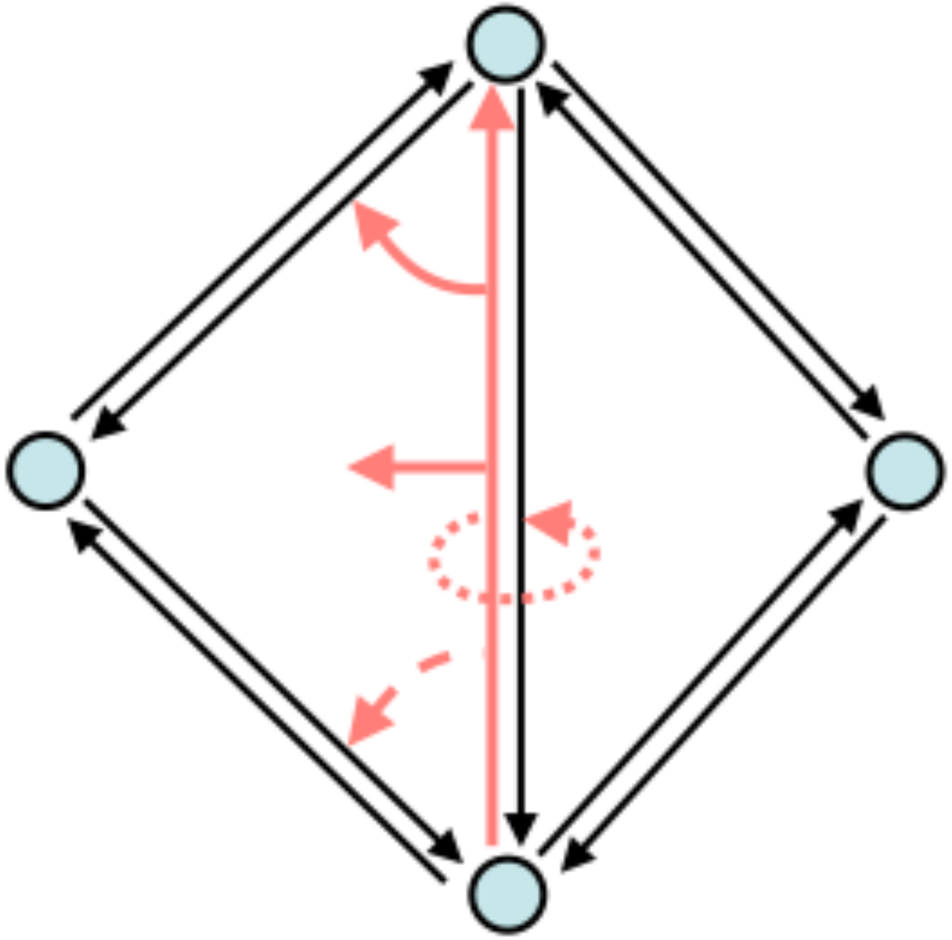
Half-Edge Data Structure

- 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

- Vertex:
  - position
  - 1 halfedge
- Edge:
  - 1 vertex
  - 1 face
  - 1, 2, or 3 halfedges
- Face:
  - 1 halfedge



96 to 144 B/v

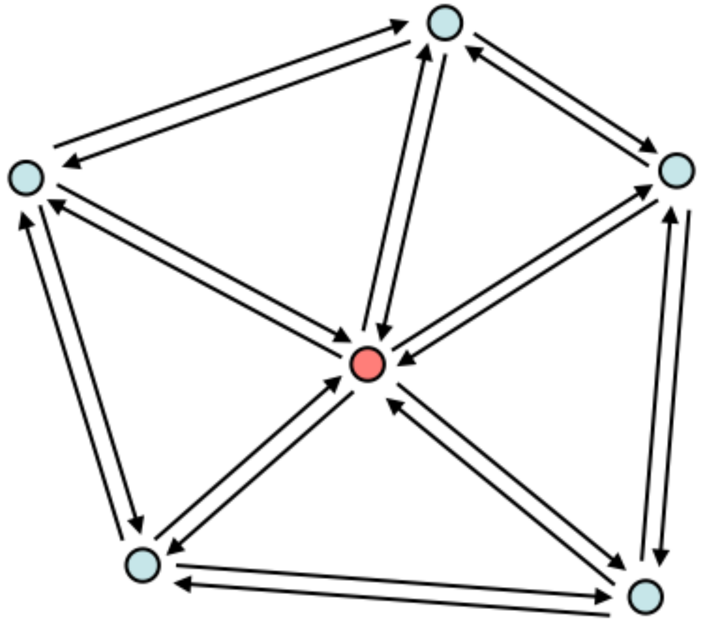
Half-Edge Data Structure

- 64-144 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

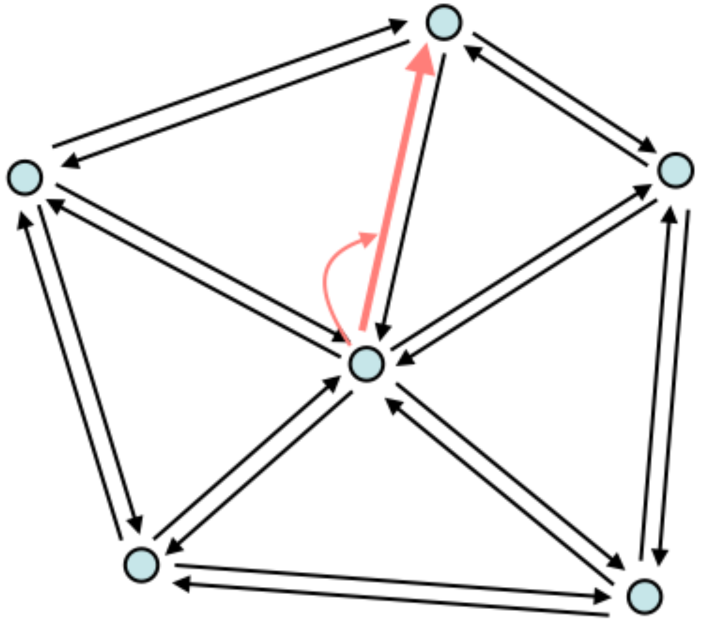
Half-Edge Data Structure

- One-ring traversal (V* relations):
    1. start at vertex



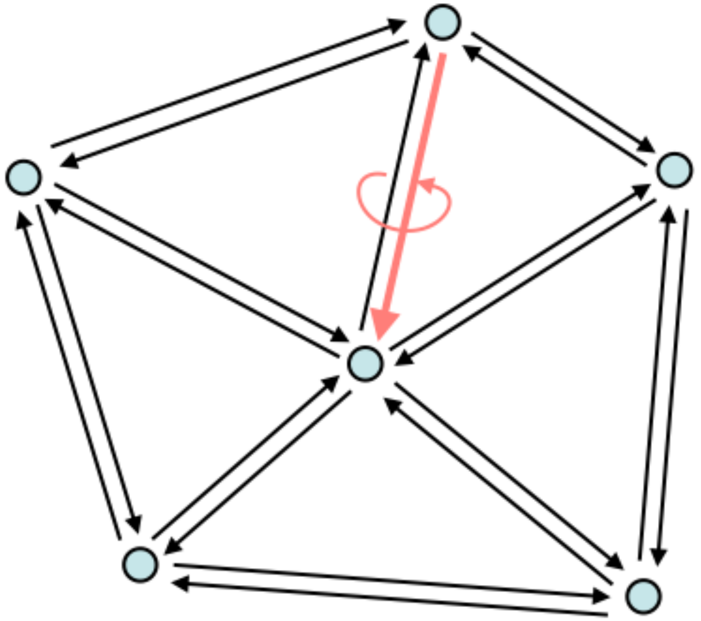
Half-Edge Data Structure

- One-ring traversal (V* relations):
  1. start at vertex
  2. outgoing half-edge



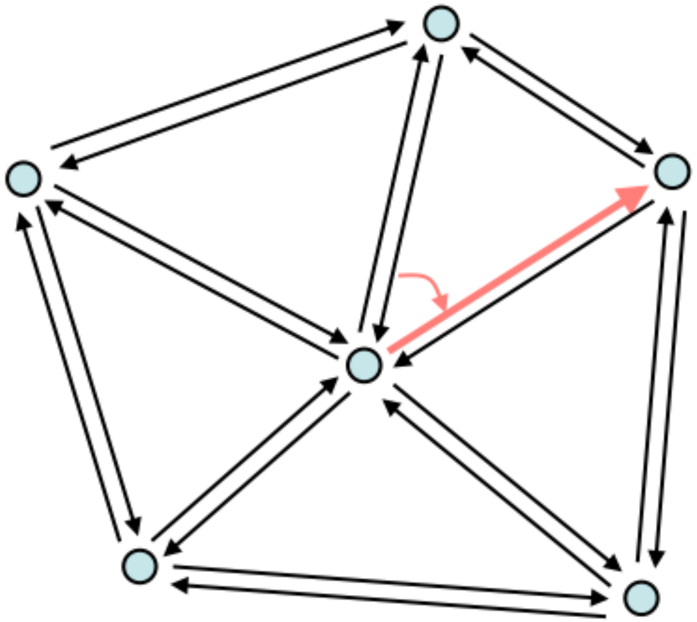
Half-Edge Data Structure

- One-ring traversal (V* relations):
  1. start at vertex
  2. outgoing half-edge
  3. opposite half-edge



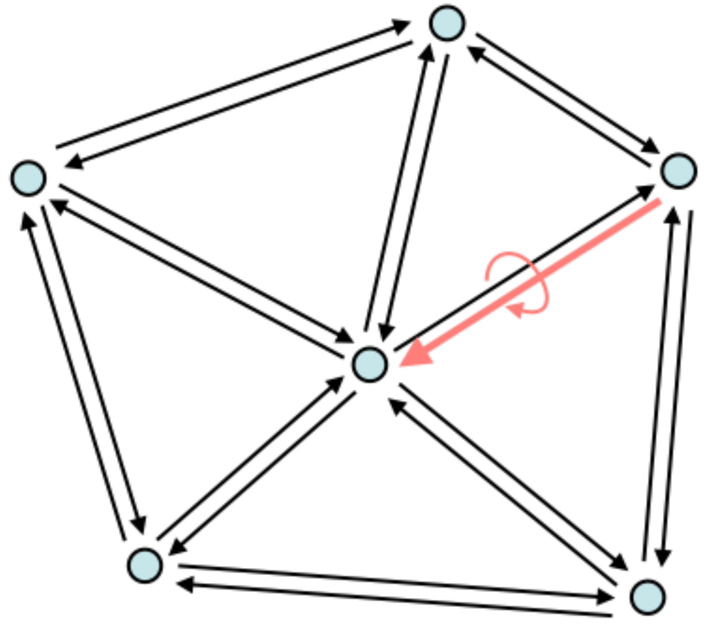
Half-Edge Data Structure

- One-ring traversal (V* relations):
  1. start at vertex
  2. outgoing half-edge
  3. opposite half-edge
  4. next half-edge



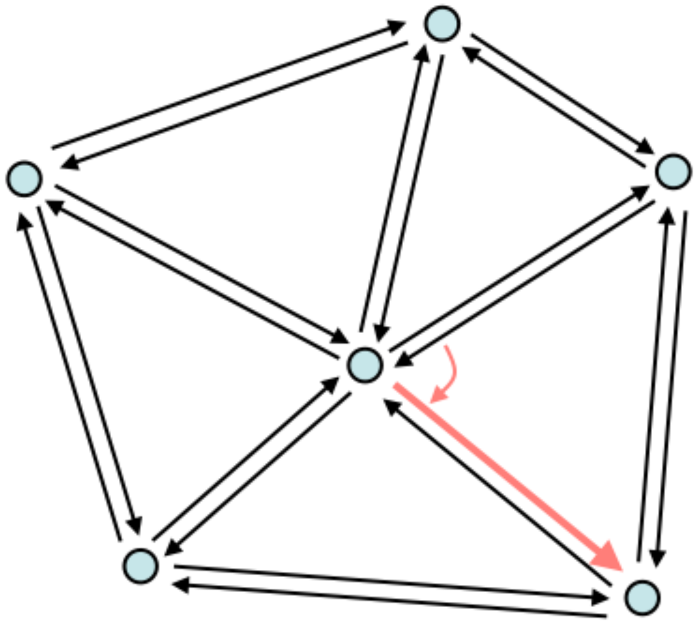
Half-Edge Data Structure

- One-ring traversal (V* relations):
  1. start at vertex
  2. outgoing half-edge
  3. opposite half-edge
  4. next half-edge
  5. opposite

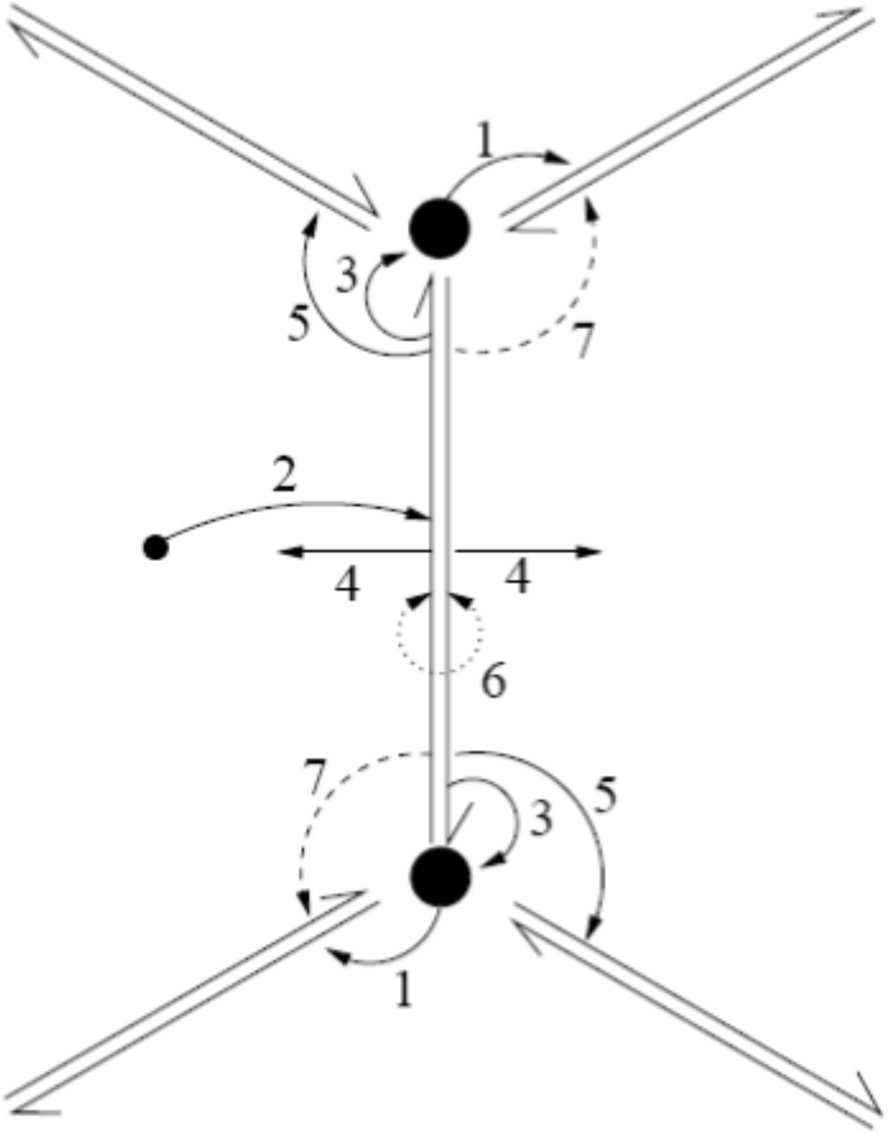


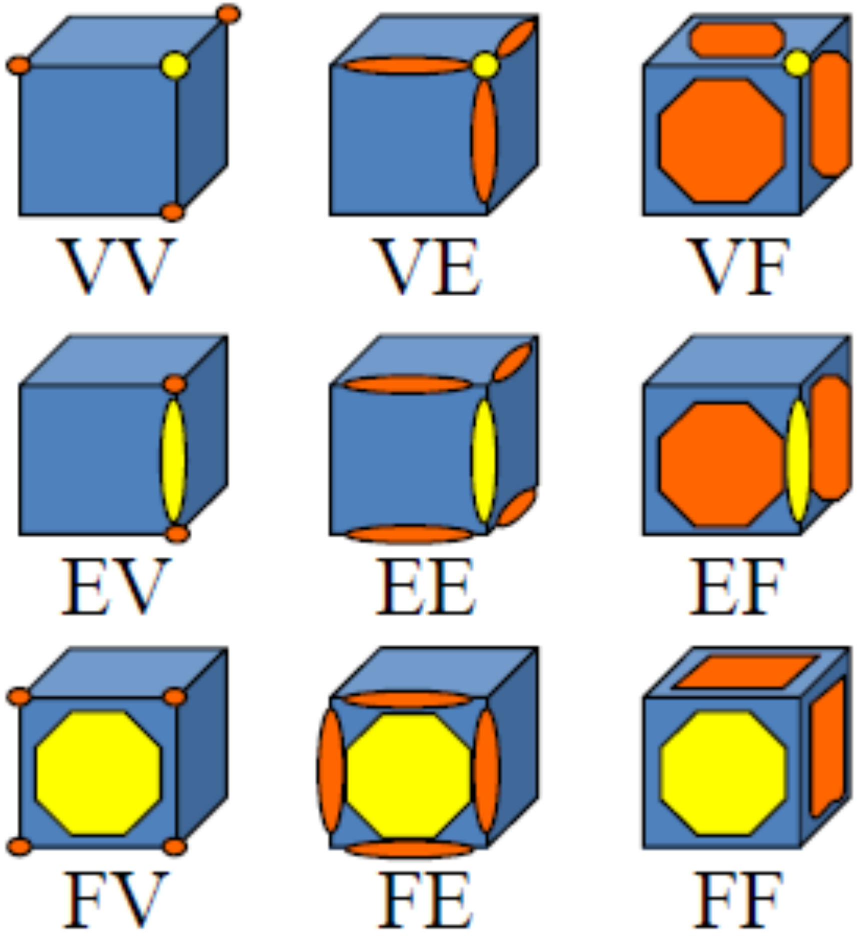
Half-Edge Data Structure

- 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.....



How HDS can -- OpenMesh

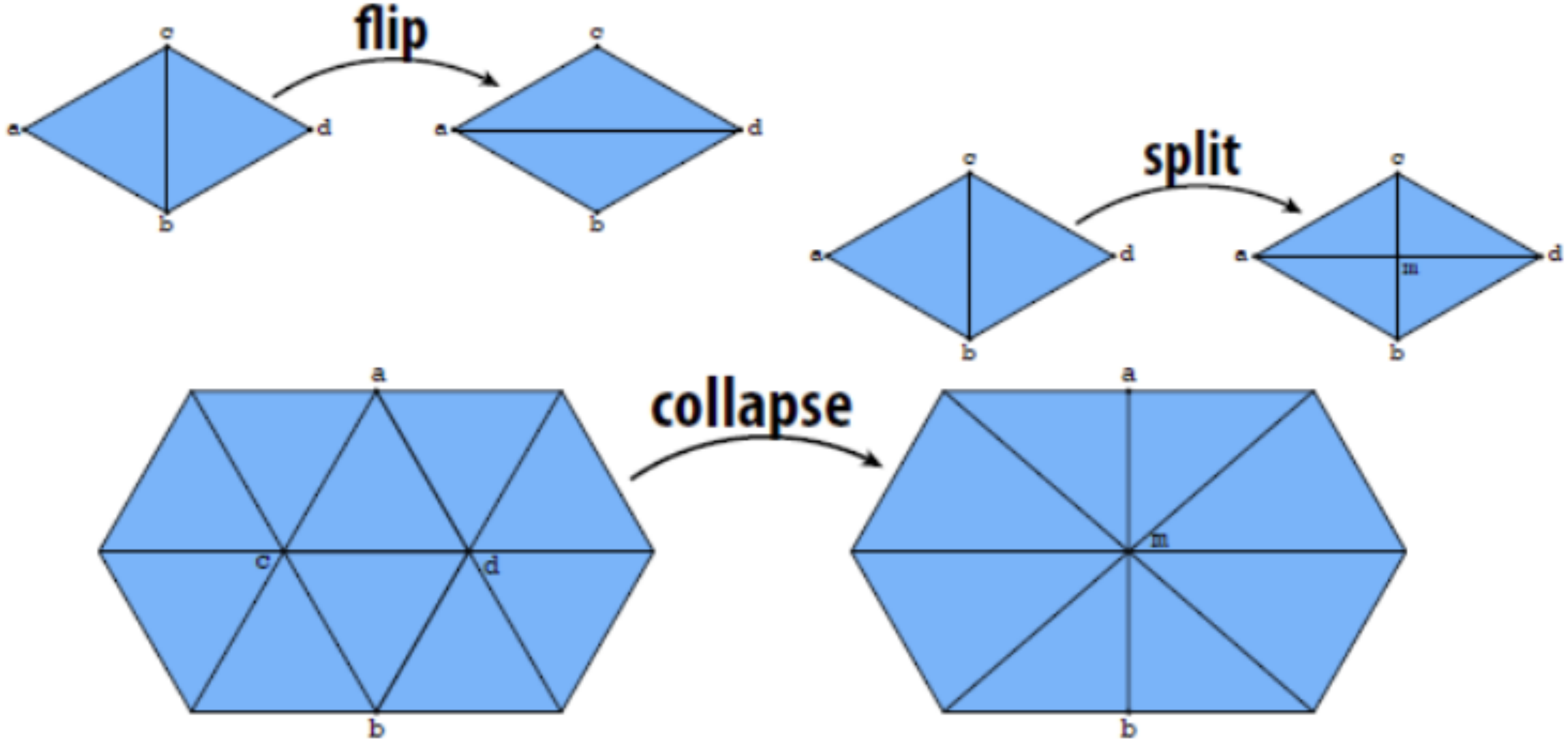




All basic queries take constant O(1) time!

Halfedge meshes are easy to edit

- Remember key feature of linked list: insert/delete elements
- Same story with halfedge mesh (“linked list on steroids”)
- E.g., for triangle meshes, several atomic operations:



- How? Allocate/delete elements; reassigning pointers.
- Must be careful to preserve manifoldness!

Comparison of Polygon Mesh Data Structures

Case study: triangles.	Polygon Soup	Incidence Matrices	Halfedge Mesh
storage cost*	~3 x #vertices	~33 x #vertices	~36 x #vertices
constant-time neighborhood access?	NO	YES	YES
easy to add/remove mesh elements?	NO	NO	YES
nonmanifold geometry?	YES	YES	NO

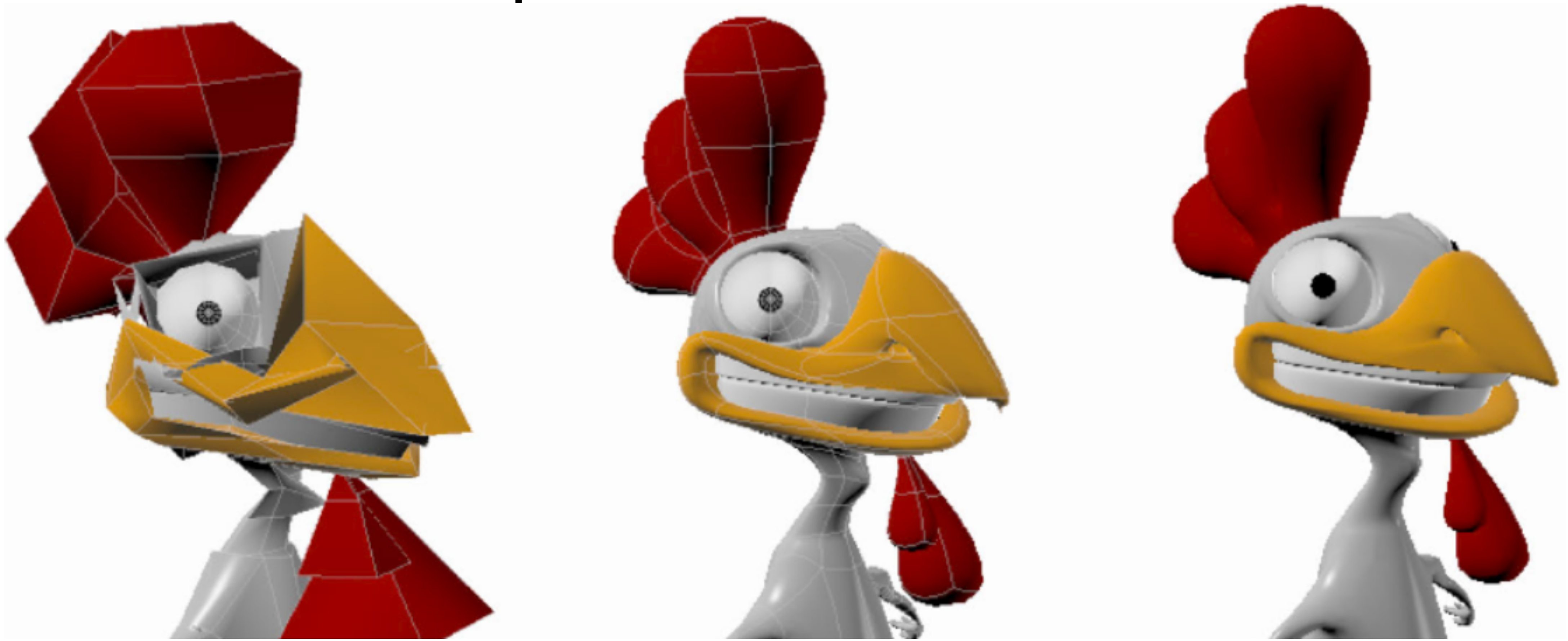
Conclusion: pick the right data structure for the job!

*number of integer values and/or pointers required to encode *connectivity*
(all data structures require same amount of storage for vertex positions)

Ok, but what can we actually *do* with our fancy new data structure?

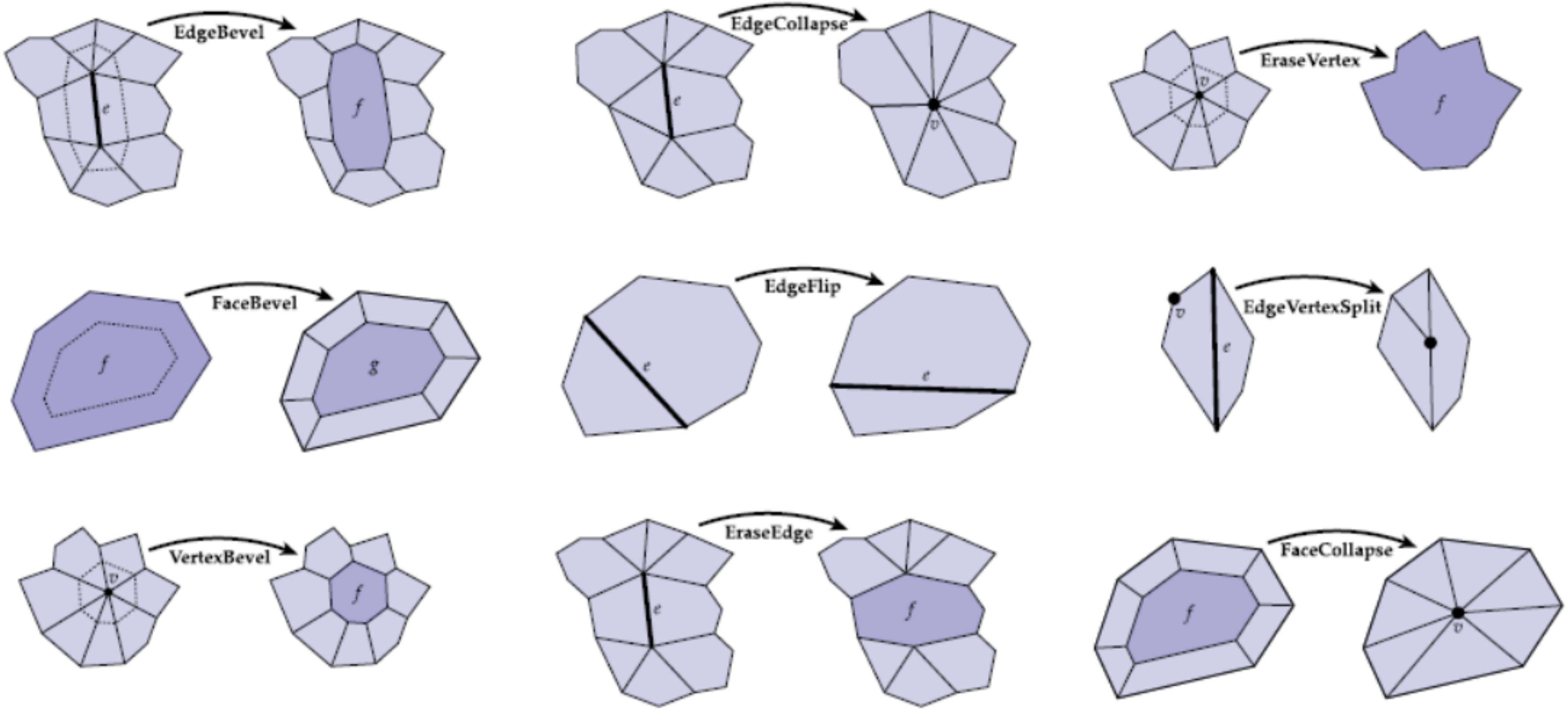
Subdivision Modeling

- Common modeling paradigm in modern 3D tools:
  - Coarse “control cage”
  - Perform local operations to control/edit shape
  - Global subdivision process determines final surface



Subdivision Modeling—Local Operations

- **For general polygon meshes, we can dream up lots of local mesh operations that might be useful for modeling:**



…and many, many more!

TOOLS

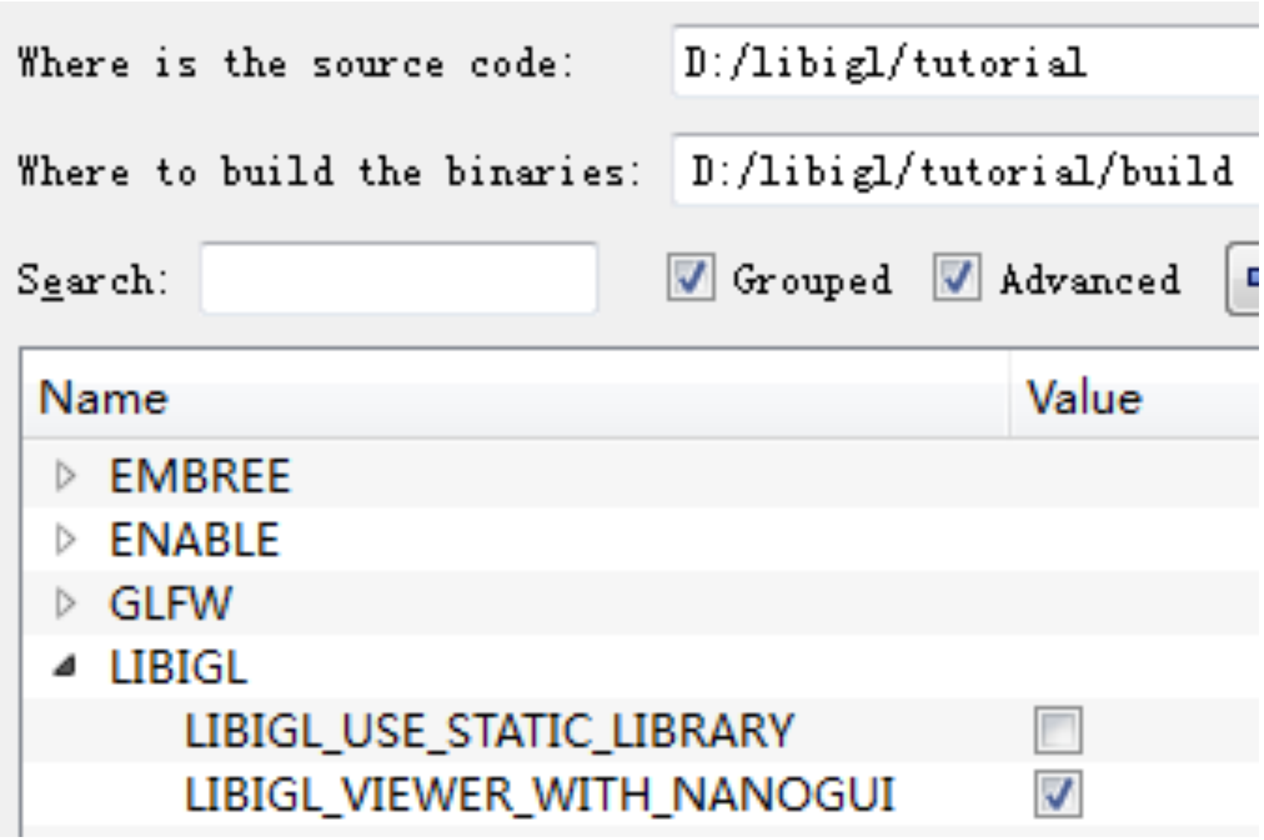
- Meshlab (`meshlab.sourceforge.net`) - free:
  - triangle mesh processing with many features
  - based on the VCGlib
- OpenFlipper (www.openflipper.org) - free:
  - polygon mesh modeling and processing
  - based on OpenMesh
- Graphite (alice.loria.fr) - free:
  - polygon mesh modeling, processing and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

Environment – c++

- Visual studio 2015 community
- CMAKE
- Eigen

- Libigl (Indexed based)
- VCGlib (Adjacency based)

- CGAL (Half-edge based)
- OpenMesh (Half-edge based)



Environment - Matlab

- Matlab 2015b
- jjcao_code: https://github.com/jjcao/jjcao_code.git

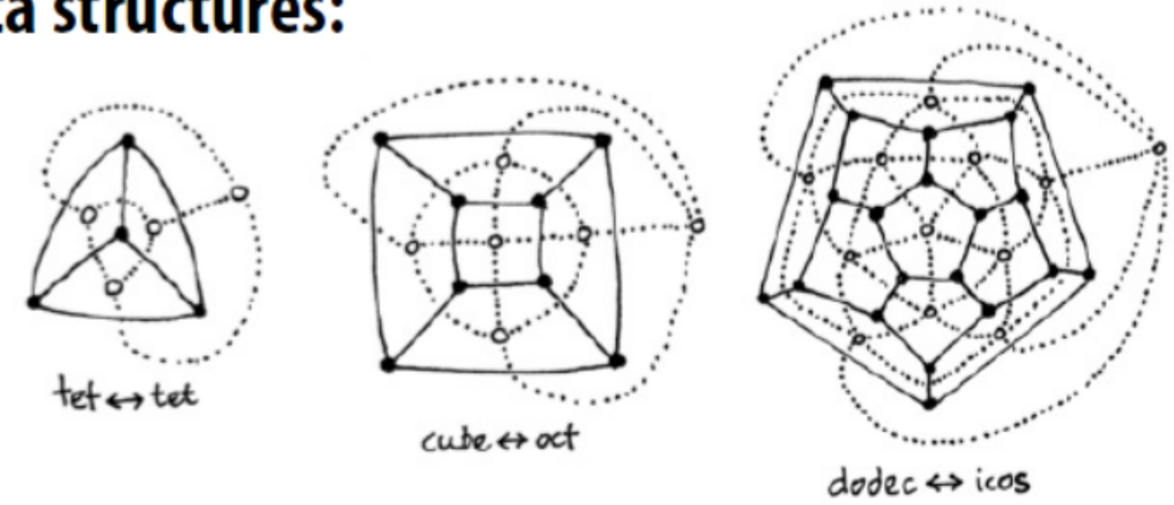
Lab

- Lab1
  - **Chapter 1 of libigl tutorial** or jjcao_code\toolbox\jjcao_plot\eg_trisurf.m
- Lab2 [optional]
  - See User manual of Halfedge Data Structures of CGAL
  - run the examples or jjcao_code\toolbox\jjcao_mesh\datastructure\test_to_halfedge.m

Alternatives to Halfedge

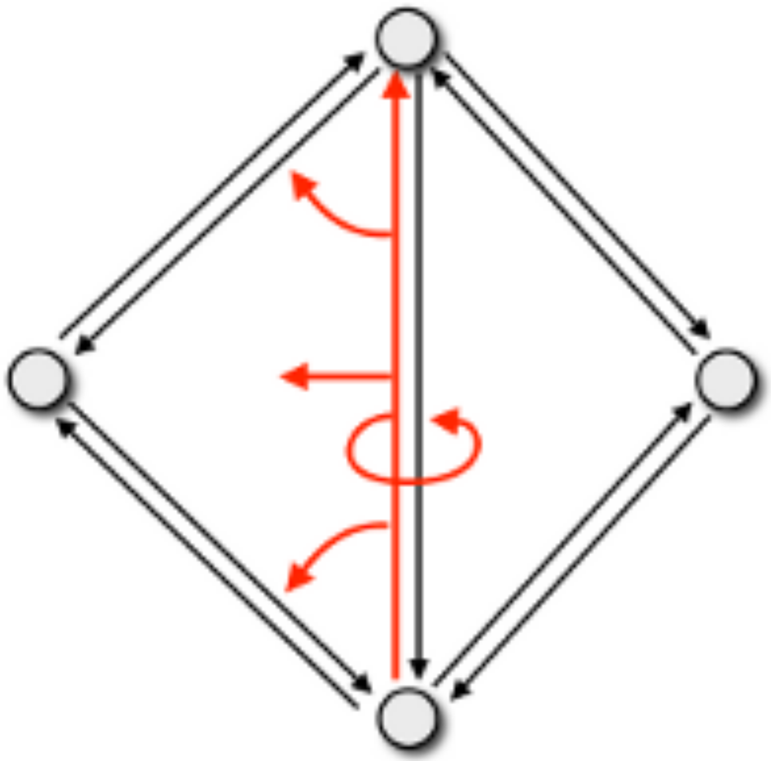
Paul Heckbert (former CMU prof.)
quadedge code - http://bit.ly/1QZLHos

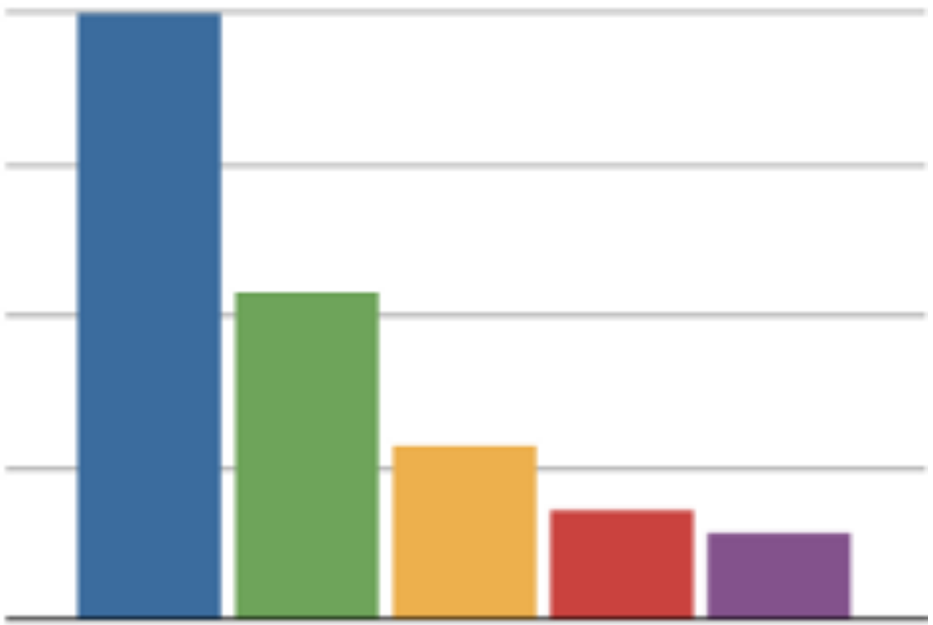
- Many very similar data structures:
  - winged edge
  - corner table
  - quadedge
  - ...



- Each stores local neighborhood information
- Similar tradeoffs relative to simple polygon list:
  - CONS: additional storage, incoherent memory access
  - PROS: better access time for individual elements, intuitive traversal of local neighborhoods
- (Food for thought: can you design a halfedge-like data structure with reasonably coherent data storage?)

Design, Implementation, and Evaluation of theSurface_mesh Data Structure





Design, Implementation and Evaluation of the Surface_mesh Data Structure, *IMR 2011*. has code

Resources

- *https://github.com/jjcao/jjcao_code.git*
- SourceTree

- Gabriel Peyre’s numerical tour!

- Wiki
- OFF file format specification

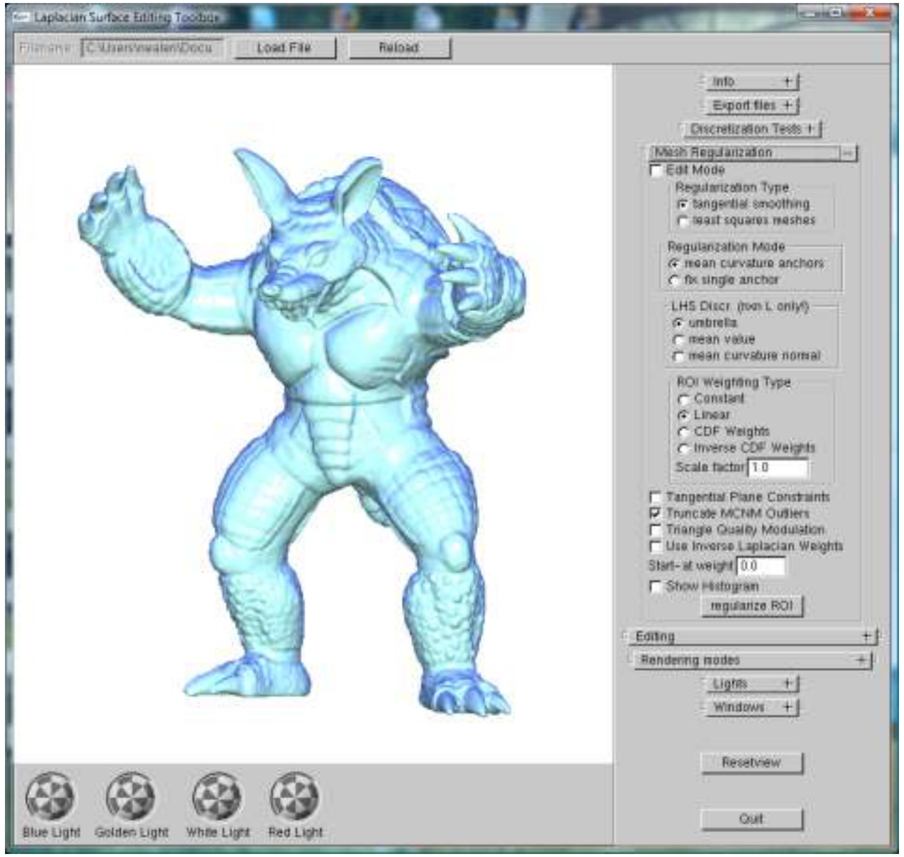
- Xianfeng Gu, lecture_8_halfedge_data_structure

Thanks!

Old assignment

Assignment 1: Mesh processing "Hello World"

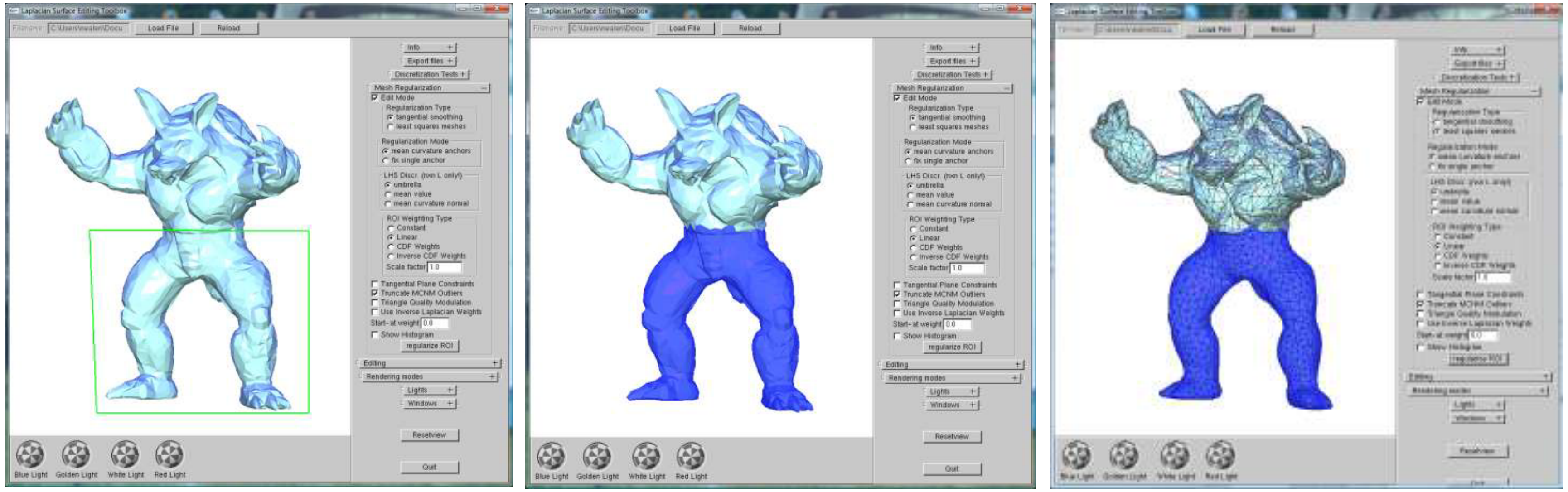
- Goals: learn basic mesh data structure programming + rendering (flat/gouraud shaded, wireframe) + basic GUI programming
- by **MATLAB** or **VC**



You can ask the help from school senior!

Assignment 2: selection + operation tools

- Goals: implement image-space selection tools and perform local operations (smoothing, etc.) on selected region
- VC



Final Project

- Implementation/extension of a space or surface based editing tool
  - makes use of assignments 1 + 2
  - Your own suggestion, with instructor approval
- Includes written project report & presentation
  - Latex style files will be provided?
  - Power Point examples will be provided?