Computer Graphics
- **Meshes and Manifolds**

Junjie Cao @ DLUT

Spring 2019

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

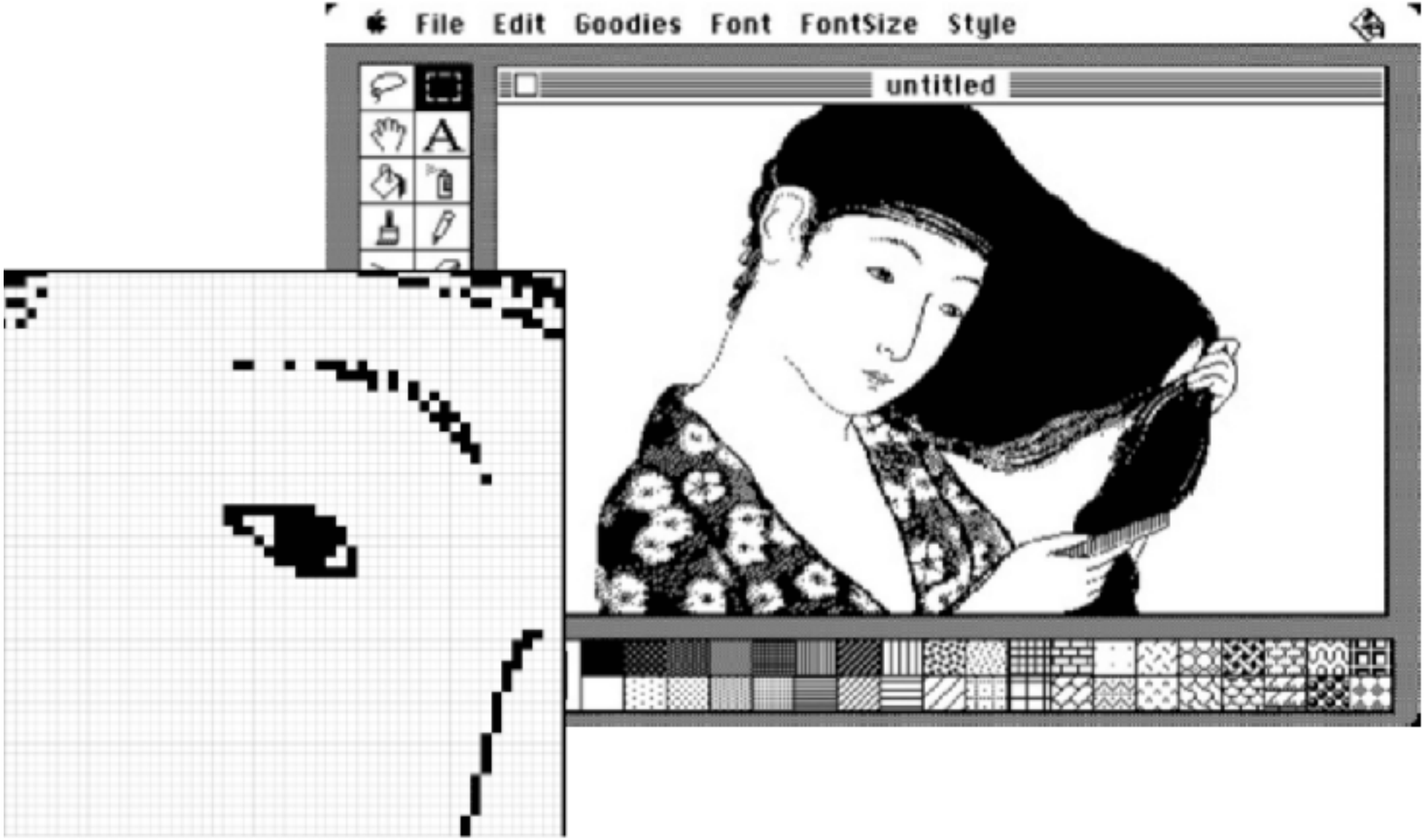
Music is dynamic, while score is static;
Movement is dynamic, while law is static.

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

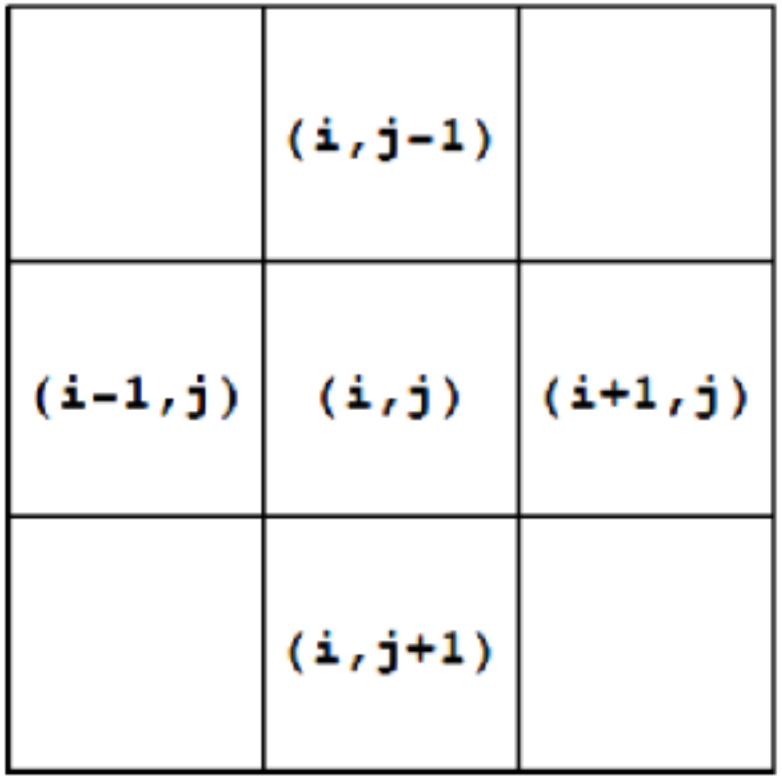
- To encode images, we used a *regular grid* of pixels:



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

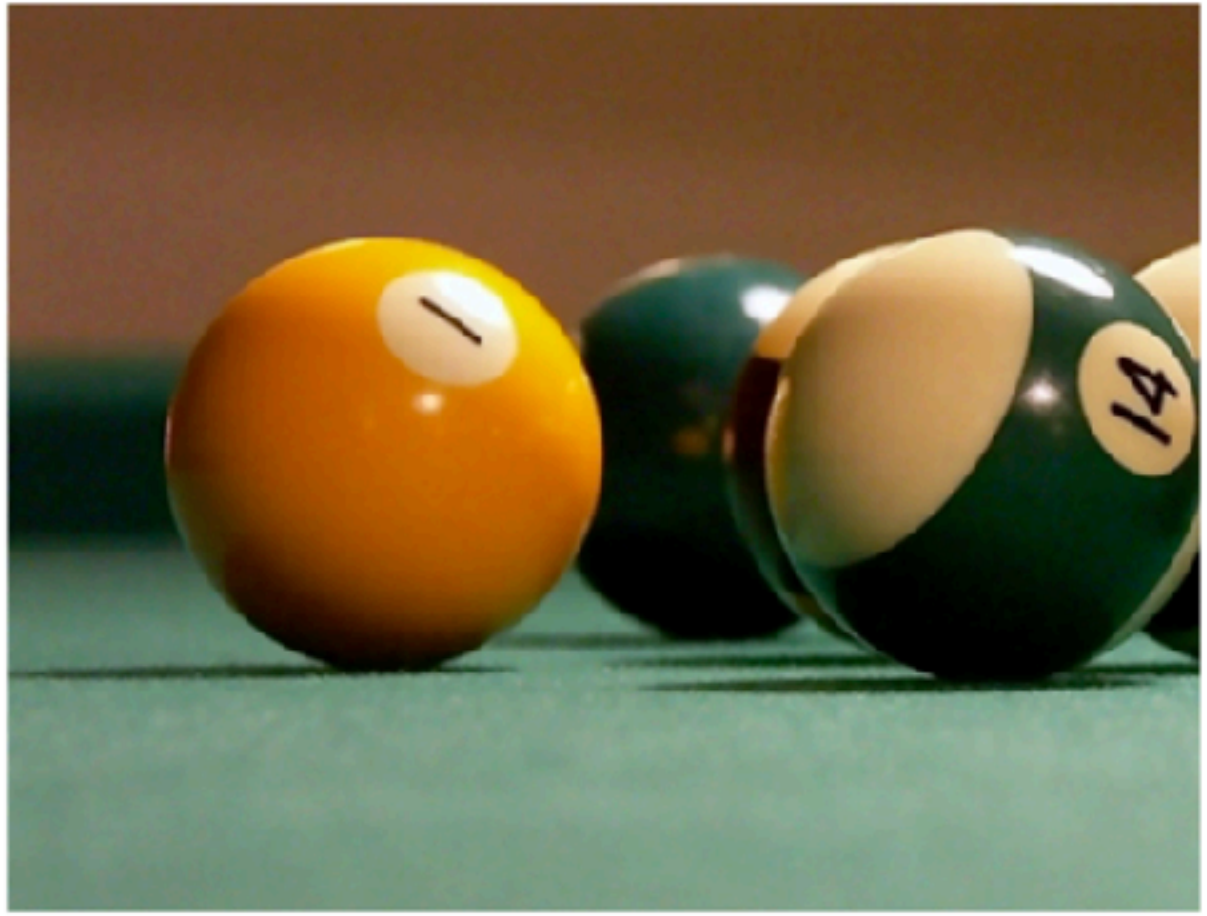
…rather than dozens of alternatives?

Regular grids make life easy



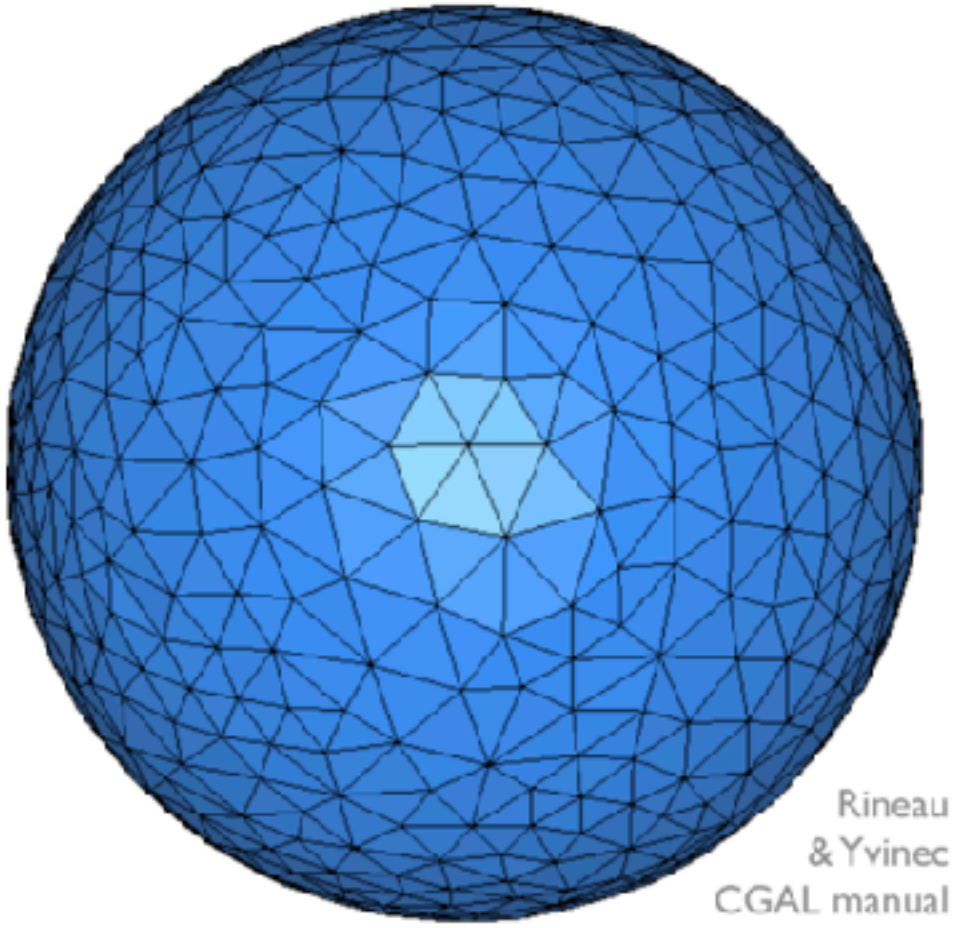
- One reason: SIMPLICITY / EFFICIENCY
  - E.g., always have four neighbors
  - Easy to index, easy to filter…
  - Storage is just a list of numbers
- Another reason: GENERALITY
  - Can encode basically any image
- Are regular grids *always* the best choice for bitmap images?
  - No! E.g., suffer from 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

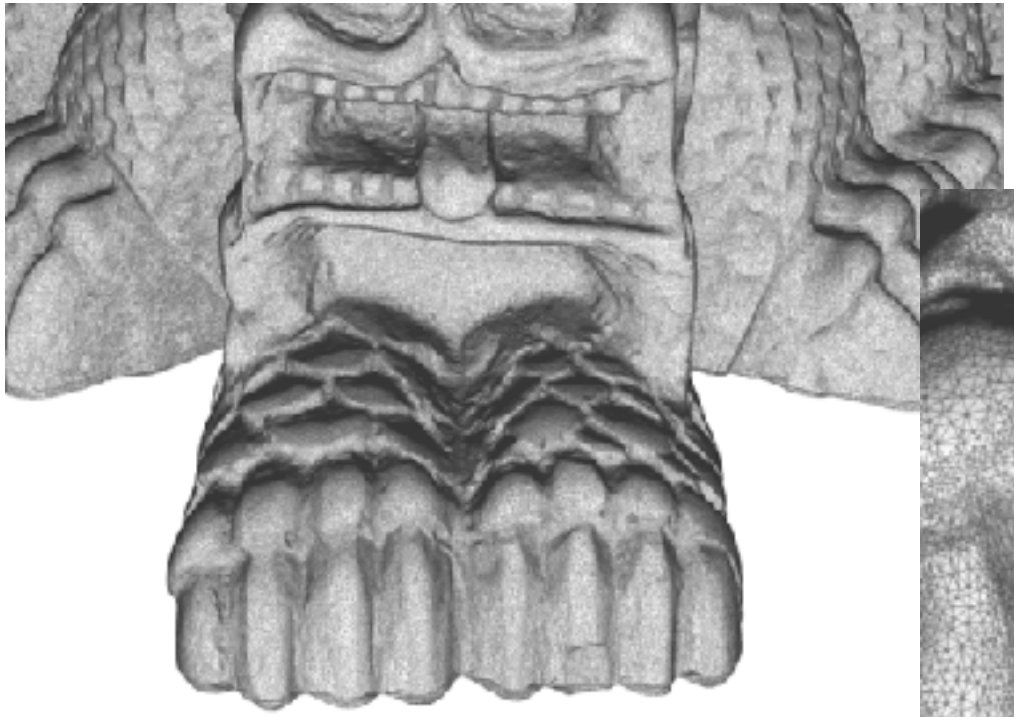


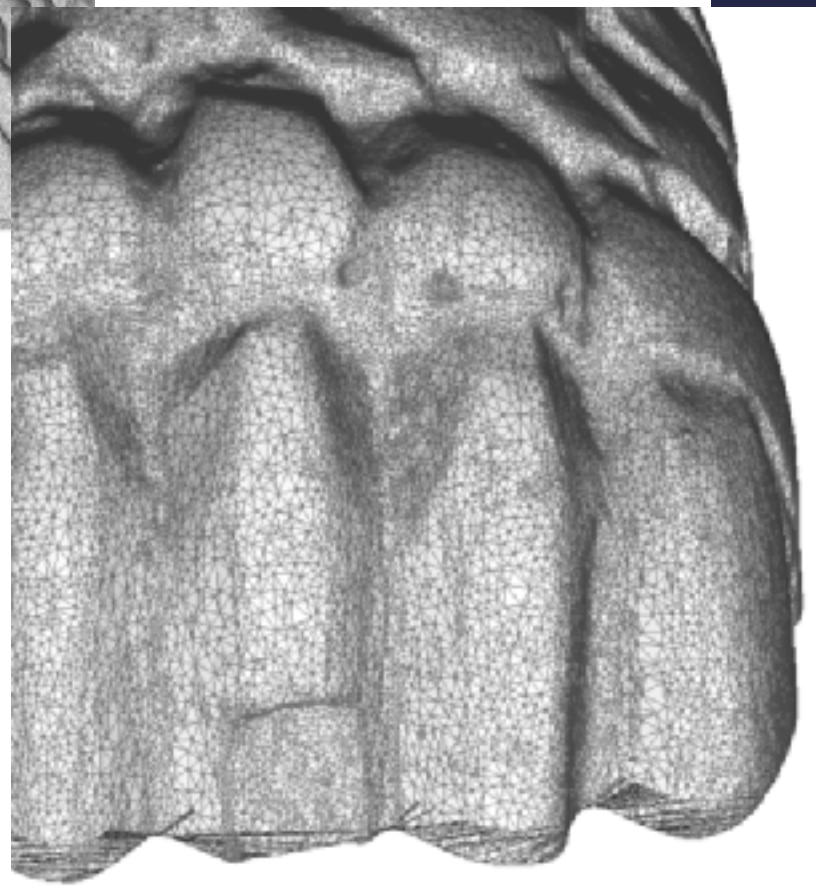
Rineau
& Yvinec
CGAL manual

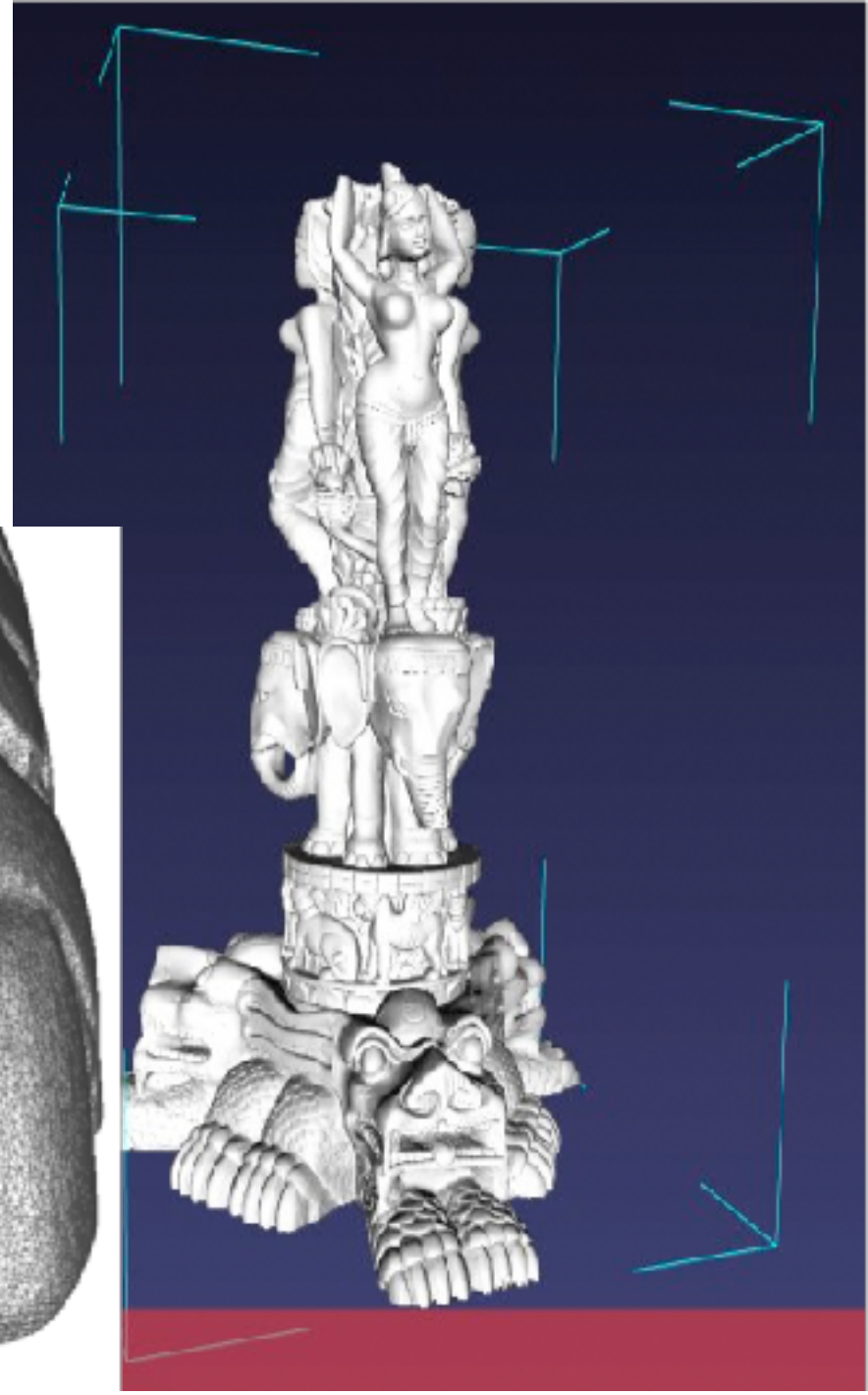
approximate sphere

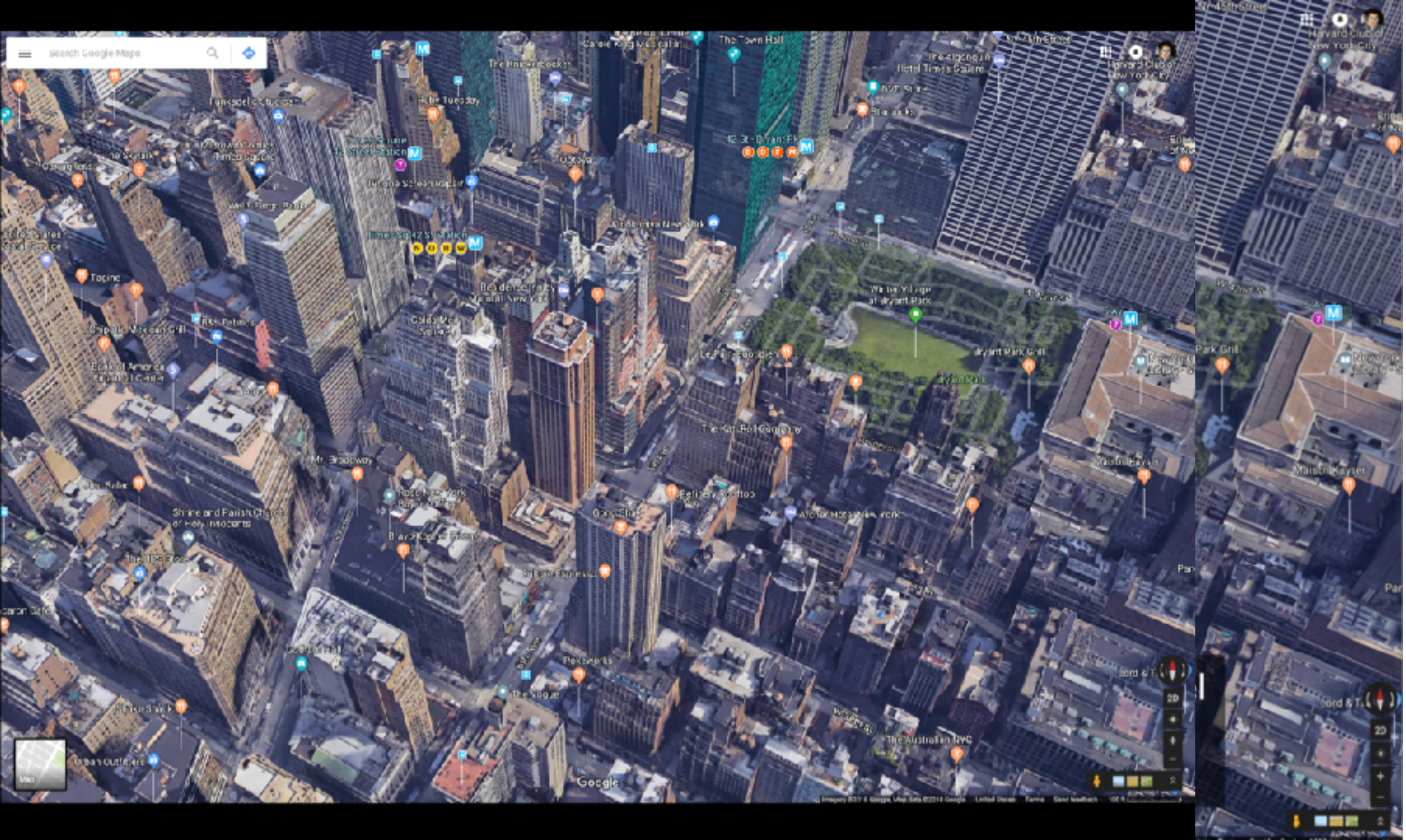
A large mesh

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



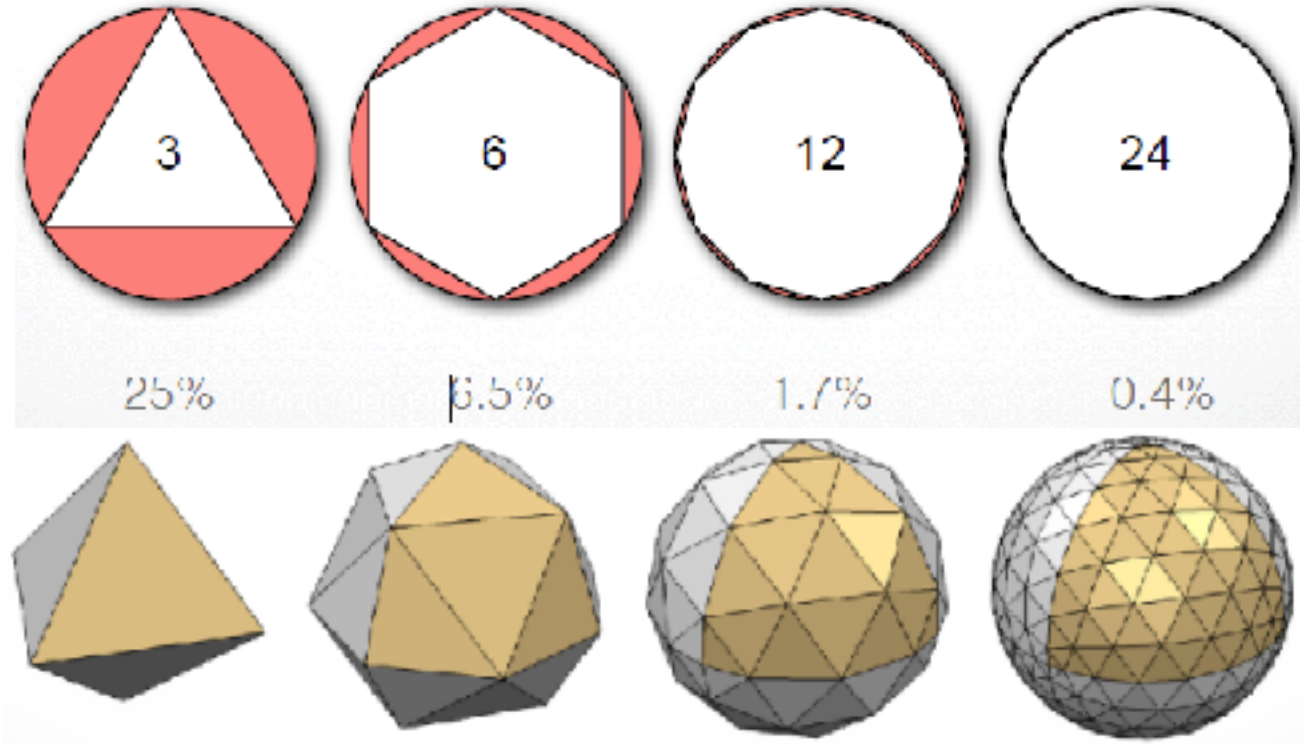


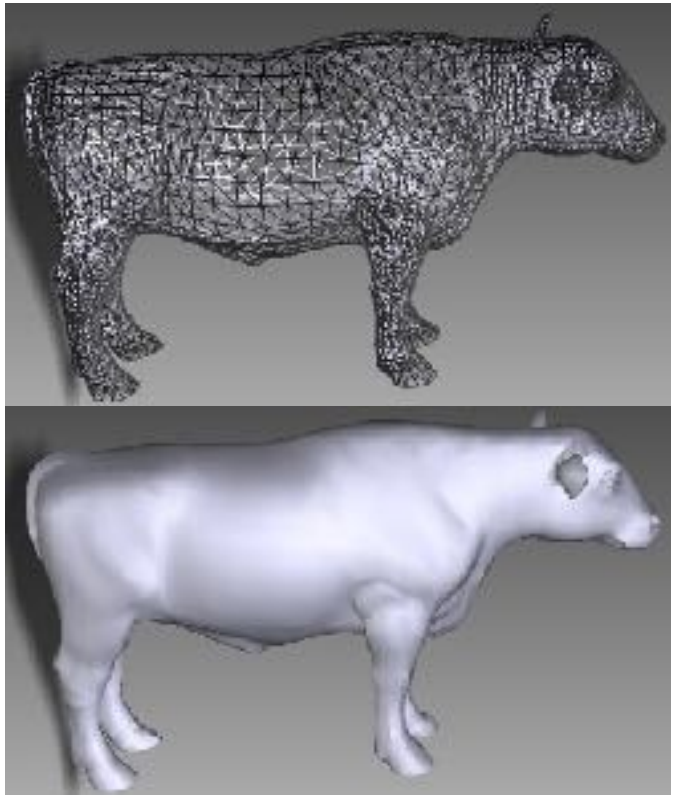



Search Google Maps
The Town Hall
Bryant Park Grill
Google

Polygonal meshes are a good compromise

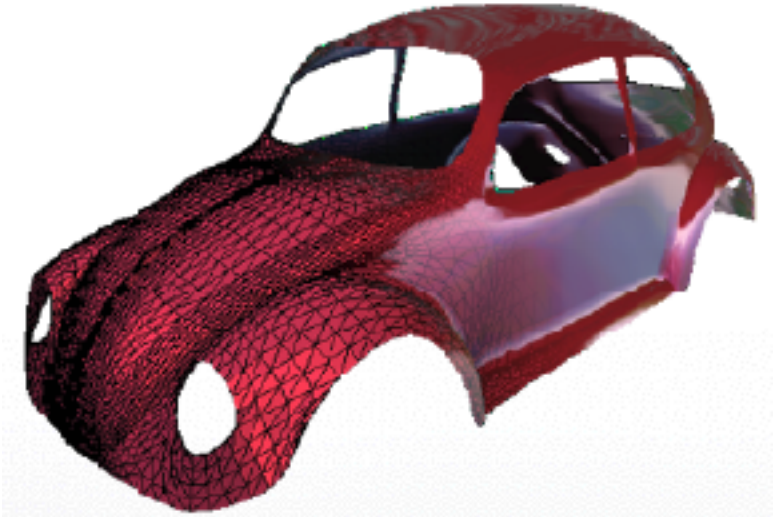
- Piecewise linear approximation → error is O(h^2)
- **Theorem** Given a smooth surface S and a given error , there exists a piecewise linear surface (mesh) M, such that for all points of M.
- Error inversely proportional to #faces





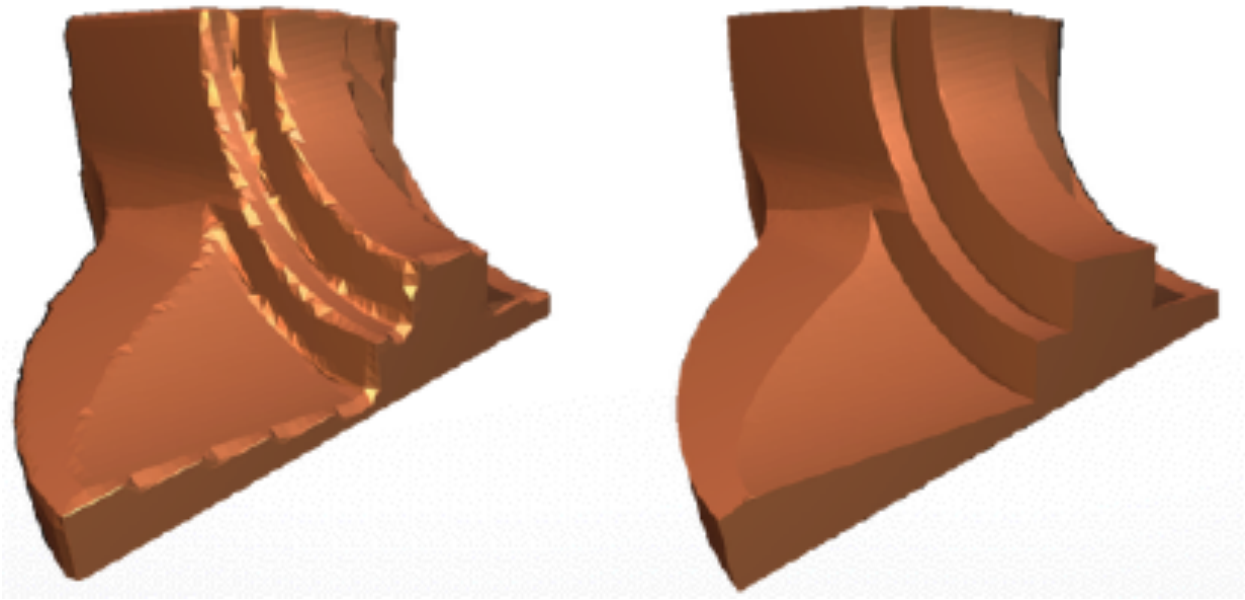
Polygonal meshes are a good compromise

- Piecewise linear approximation → error is O(h^2)
- Arbitrary topology surfaces



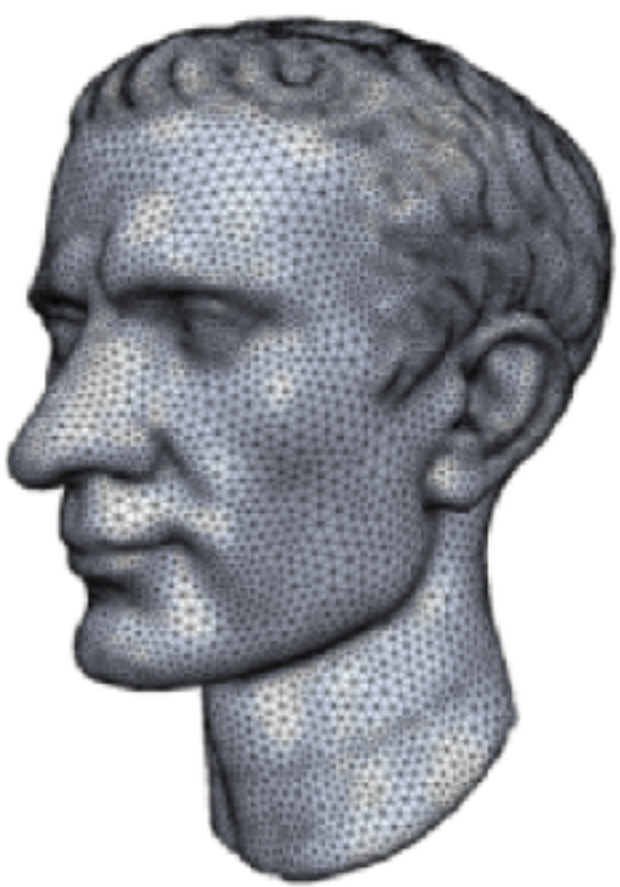
Polygonal meshes are a good compromise

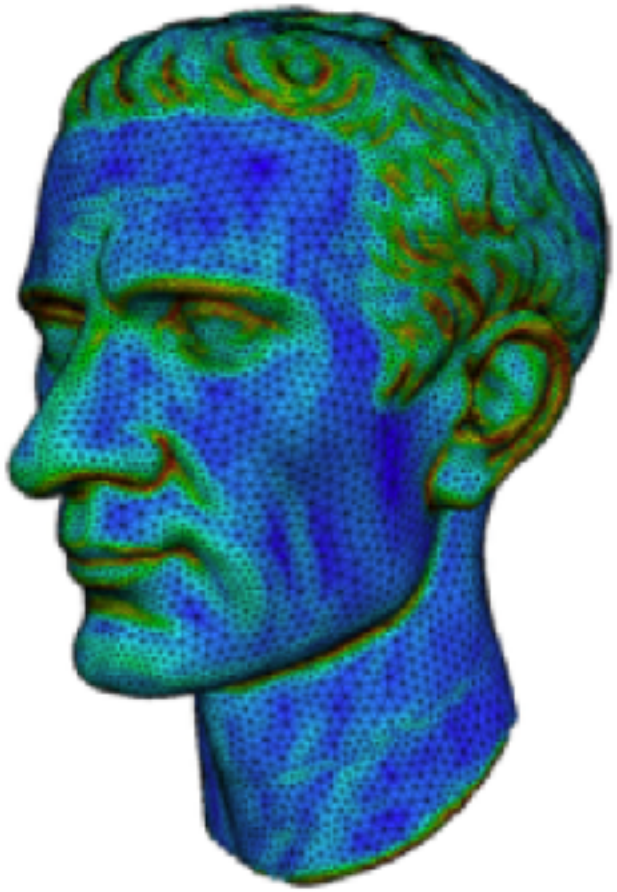
- Piecewise linear approximation → error is O(h^2)
- Arbitrary topology surfaces
- Piecewise smooth surfaces



Polygonal meshes are a good compromise

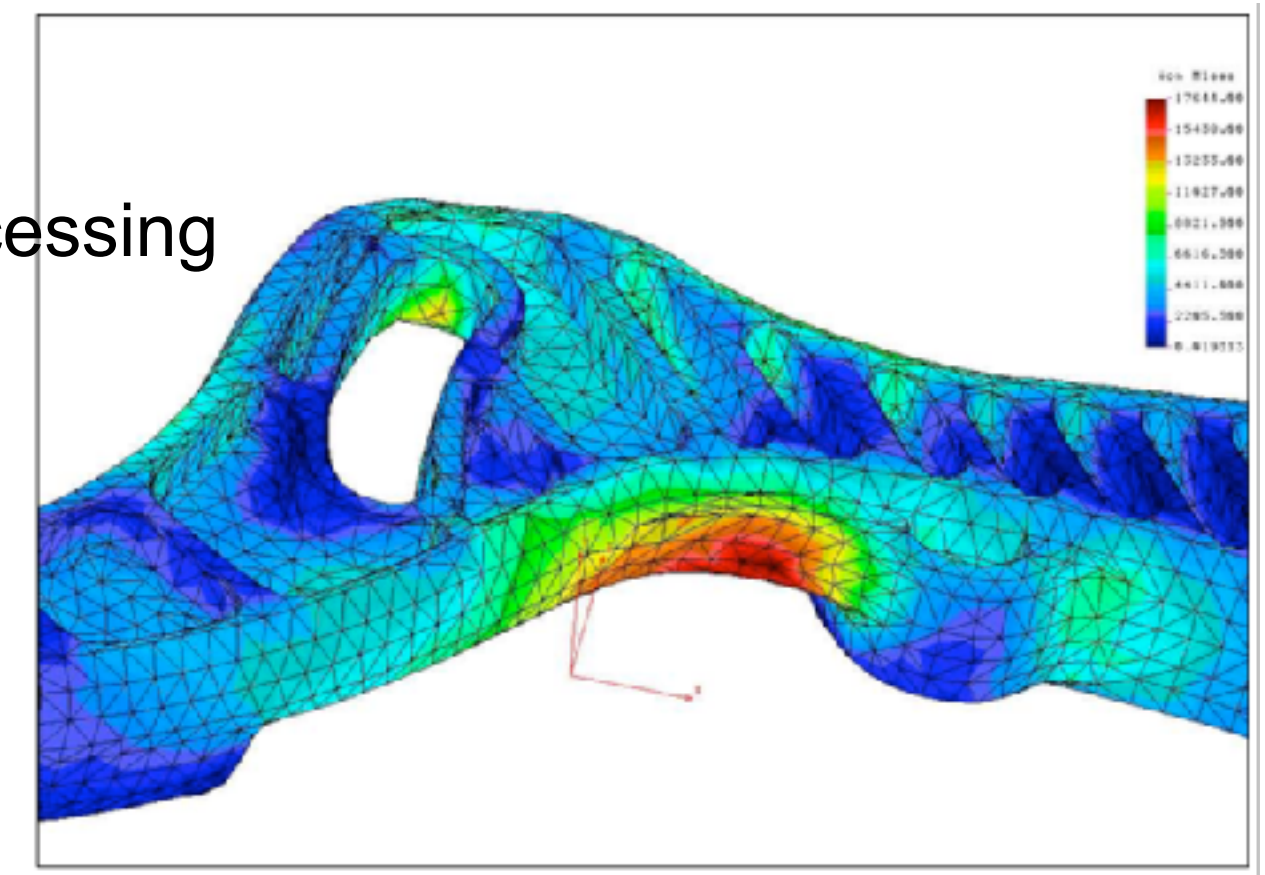
- Piecewise linear approximation → error is O(h^2)
- Arbitrary topology surfaces
- Piecewise smooth surfaces
- Adaptive sampling





Polygonal meshes are a good compromise

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



PATRIOT Engineering

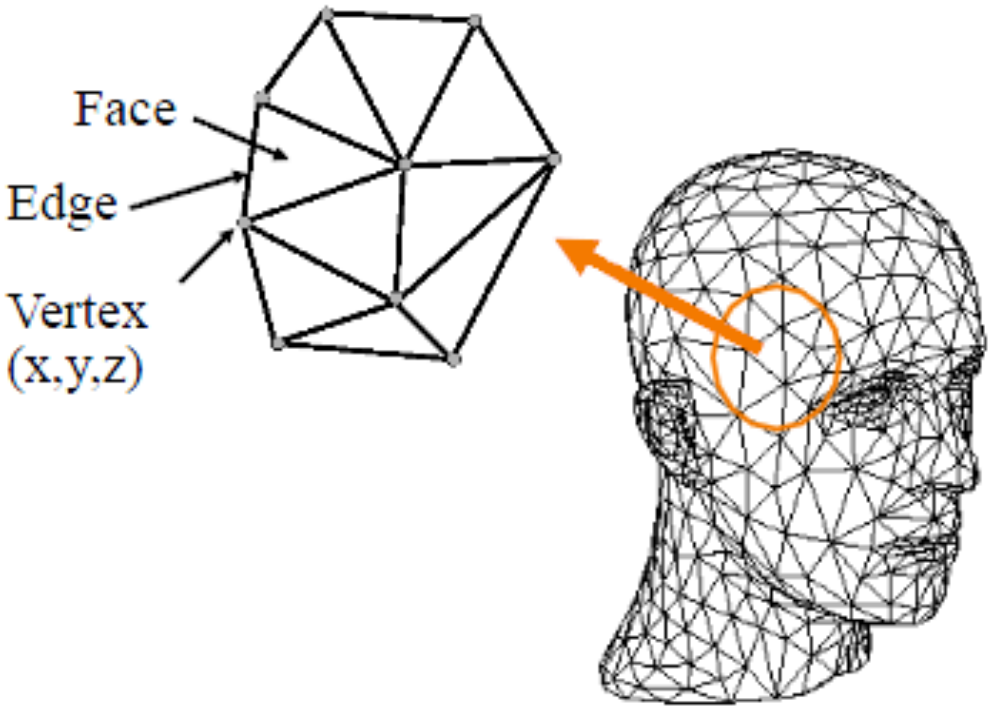
finite element analysis

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* is a set of subsets of {1, ?, N} : $K = V \cup E \cup F$
  - Vertices $v = \{i\} \in V$
  - Edges $e = \{i, j\} \in E$
  - Faces $f = \{i_1, i_2, \ldots, i_{n_f}\} \in F$



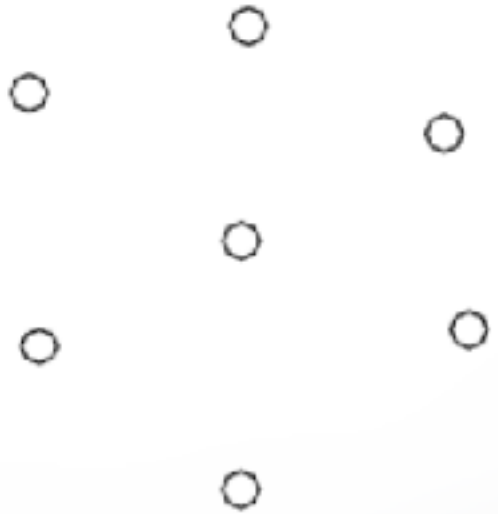
- A **Graph** is a pair G=(V,E)

Zorin & Schroeder

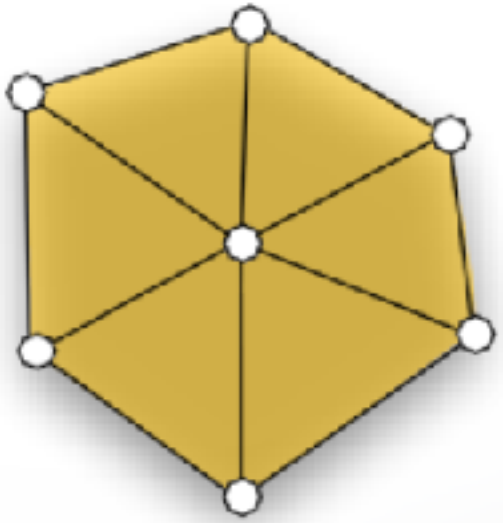
Polygonal Meshes

$$\mathcal{M} = (\{\mathbf{v}_i\}, \{e_j\}, \{f_k\})$$

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



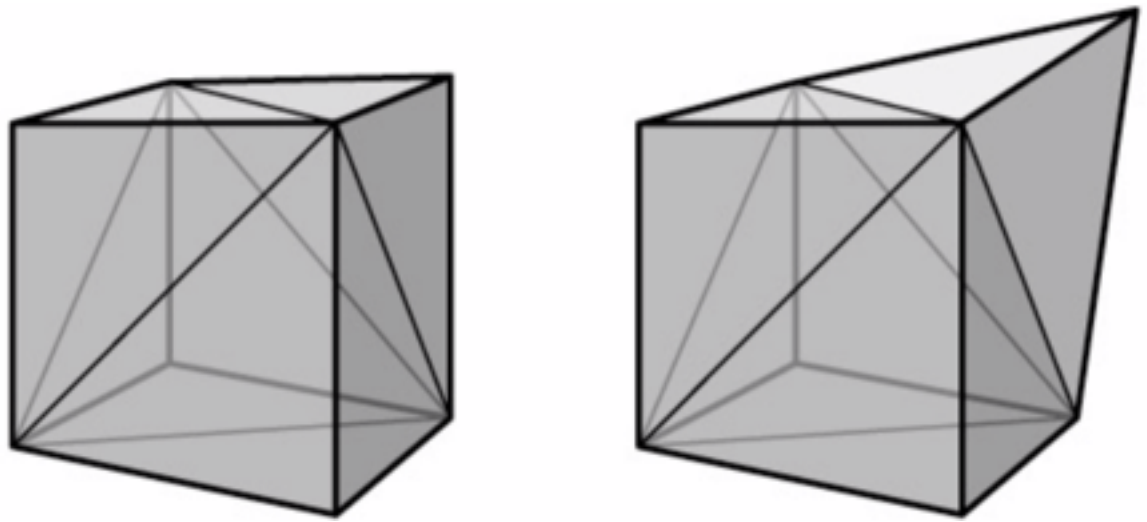
topology $e_i, f_i \subset \mathbb{R}^3$



Polygonal Meshes

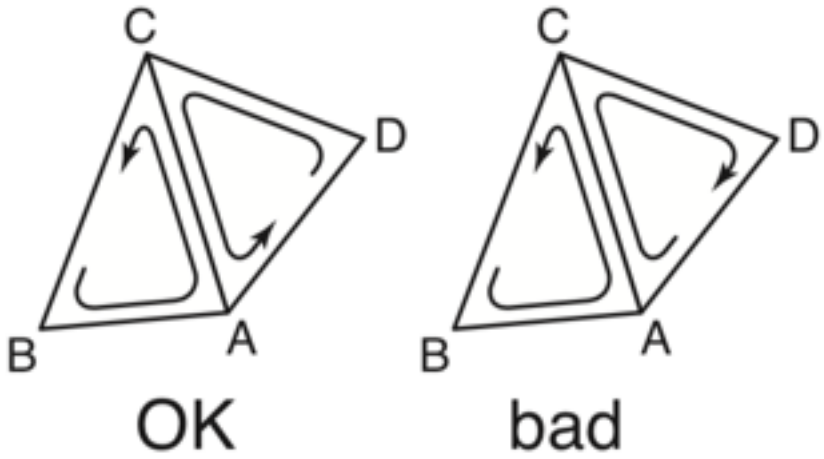
- Geometry
  - Embedding – Vertex coordinates
  - Riemannian metrics – Edge lengths
  - Conformal Structure – Corner angles (and other variant definitions)

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

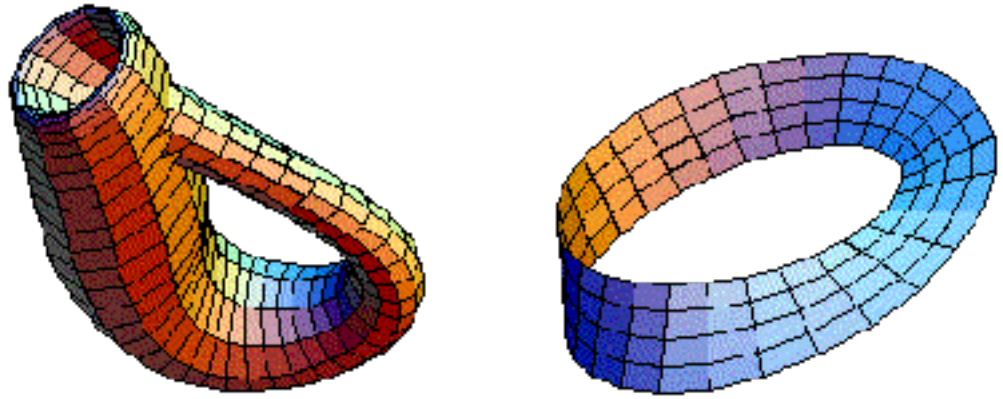


Topological validity - **Consistent orientation**

- 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

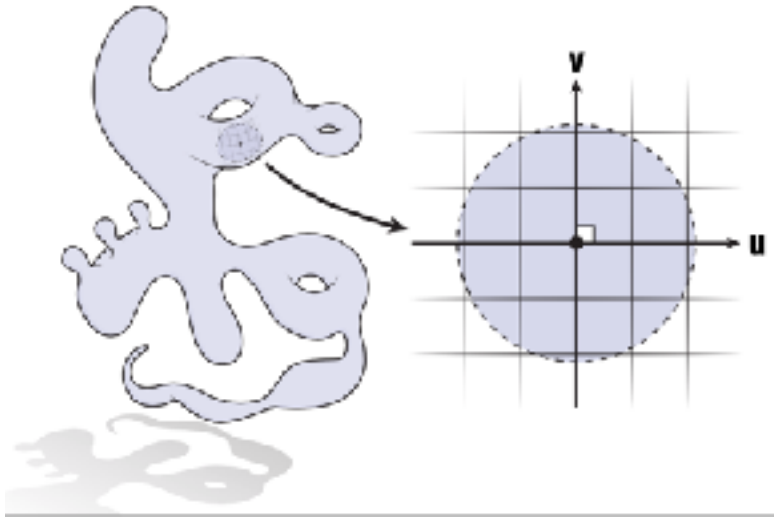


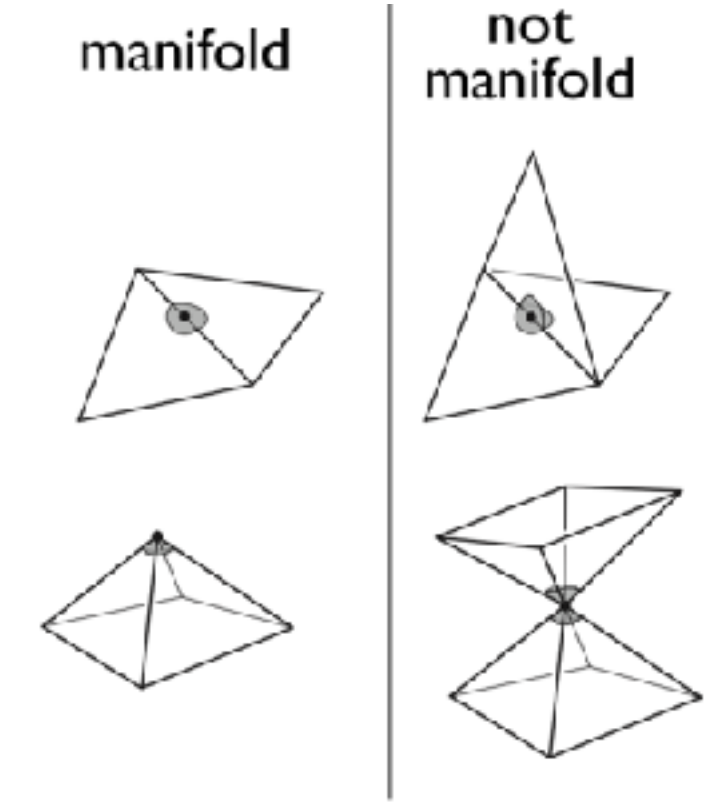
- Not every mesh can be well oriented.



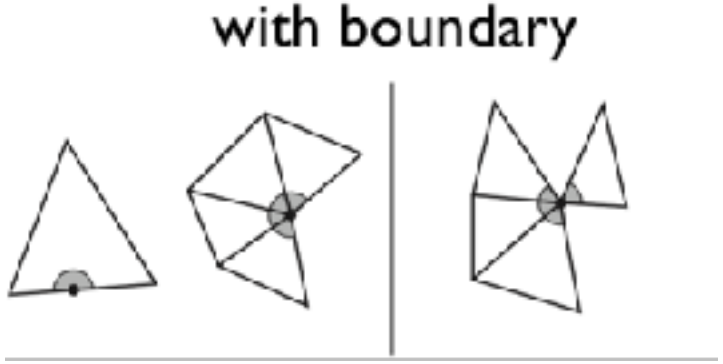
non-orientable surfaces

Topological validity -- Manifold assumption



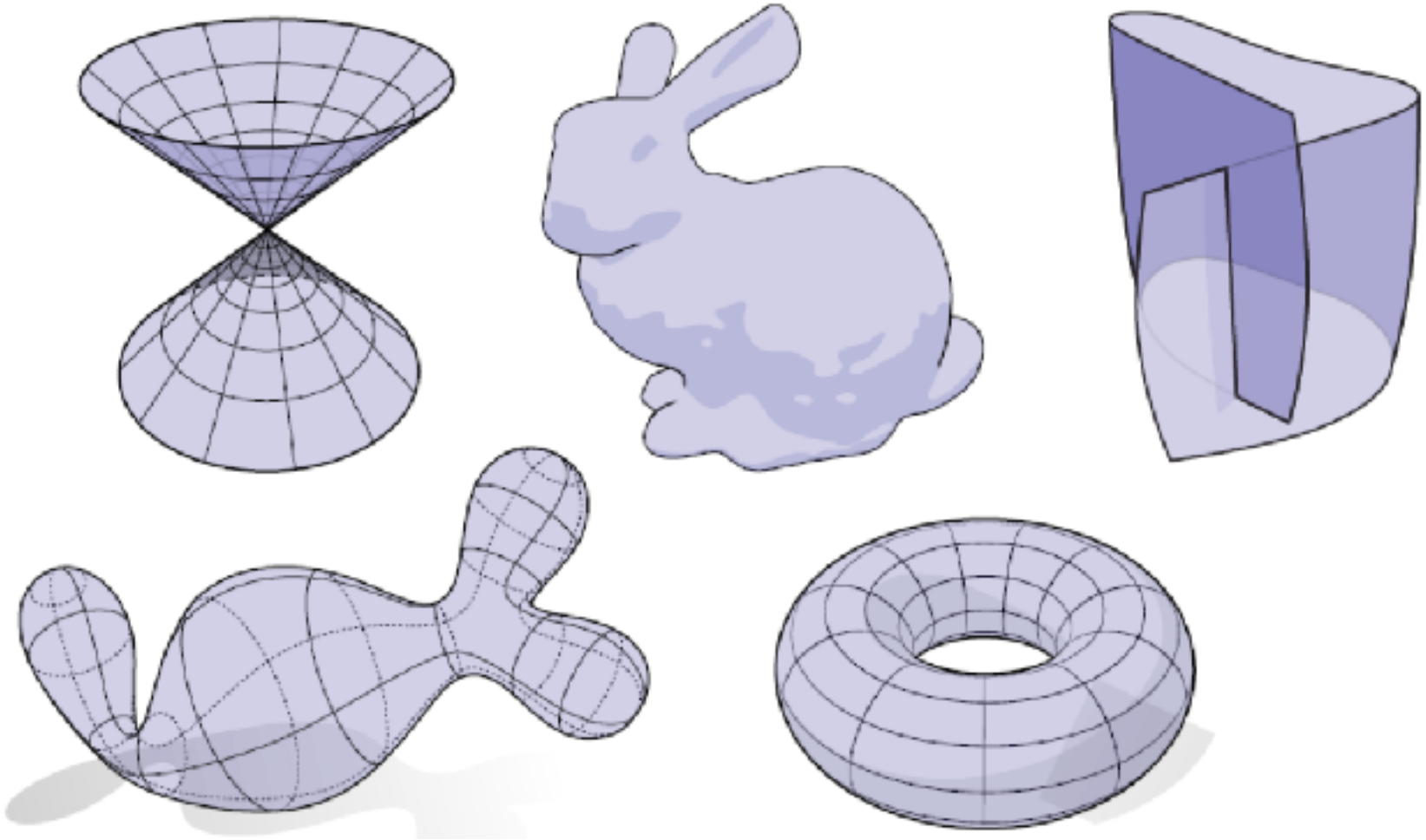


- **strongest property: be a manifold**
  - edge: each edge must have exactly 2 triangles
  - vertex: each vertex must have one loop of triangles
- **slightly looser: manifold with boundary**



Isn’t every shape manifold?

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



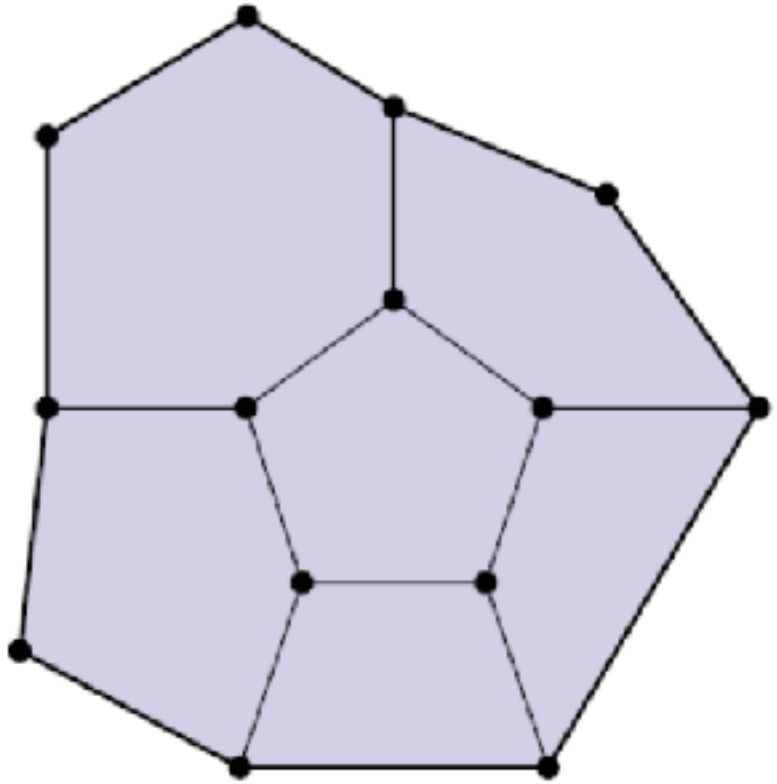
Center point never looks like the plane, no matter how close we get.

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

Keep it Simple!

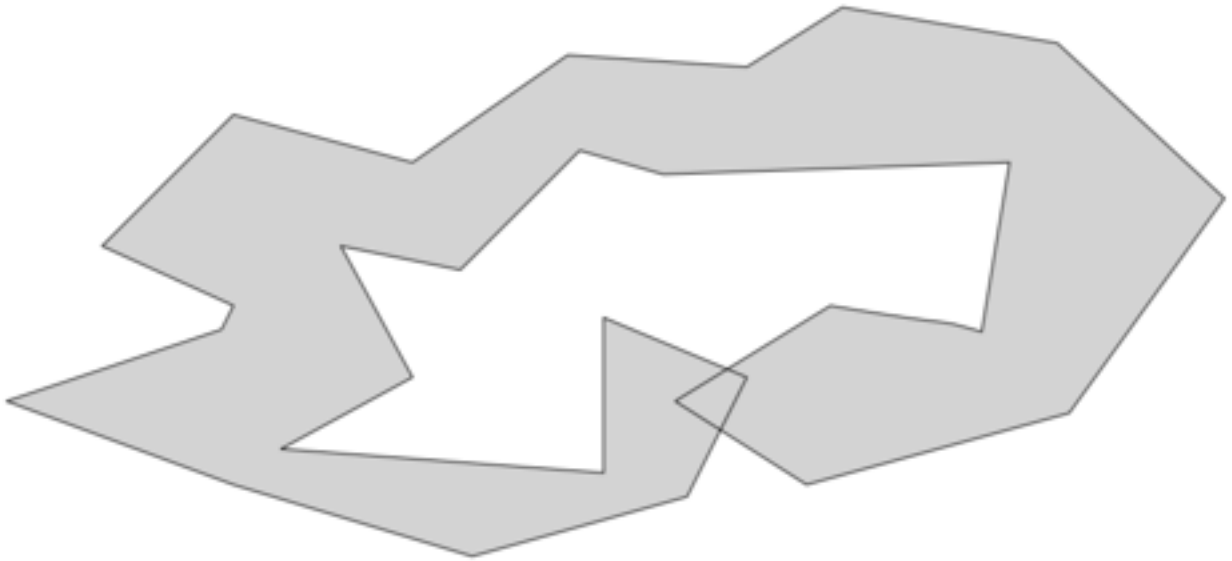
- **Same motivation as for images:**
  - **make some assumptions about our geometry to keep data structures/algorithms simple and efficient**
  - **in *many common cases*, doesn't fundamentally limit what we can do with geometry**

	(i,j-1)	
(i-1,j)	(i,j)	(i+1,j)
	(i,j+1)	



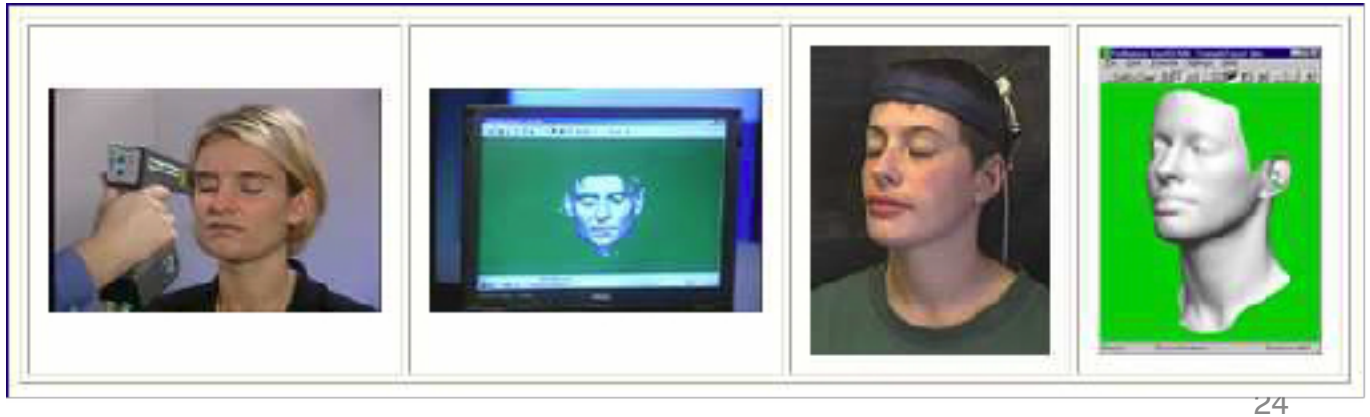
Geometric validity

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



Where Meshes Come From

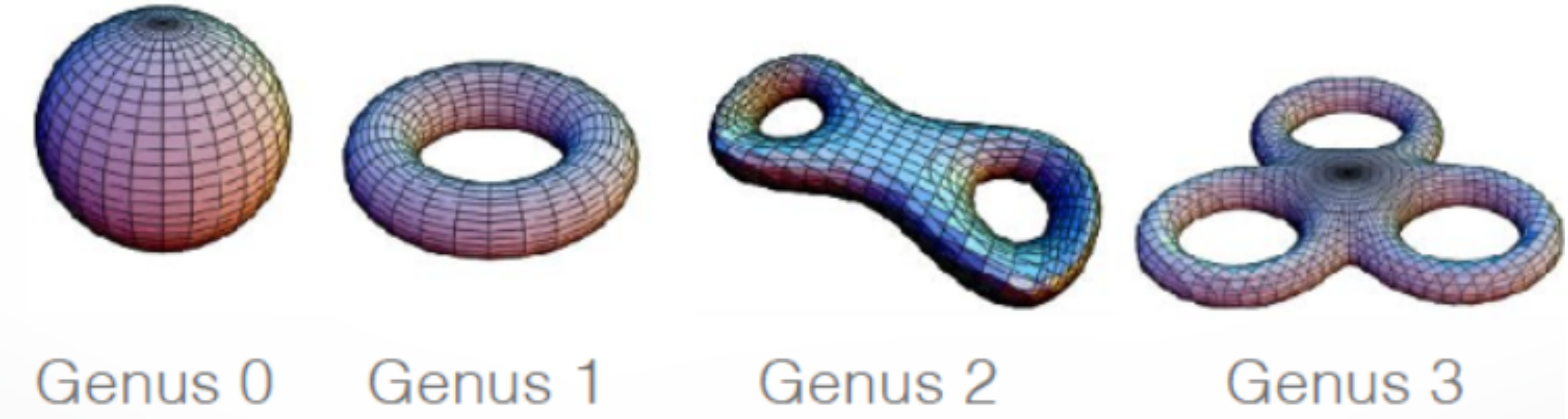
- Model manually
  - Write out all polygons
  - Write some code to generate them
  - Interactive editing: move vertices in space
- Acquisition from real objects
  - 3D scanners, vision systems
  - Generate set of points on the surface
  - Need to convert to polygons



Global Topology: **Genus**

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

- Informally, the number of holes or handles

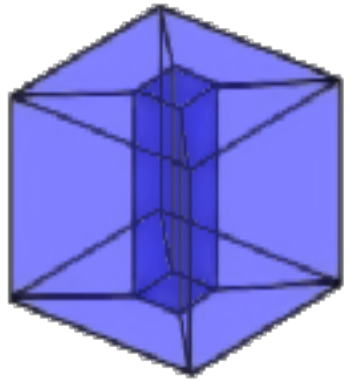


A disc (plane with boundary) / planar graph has genus zero

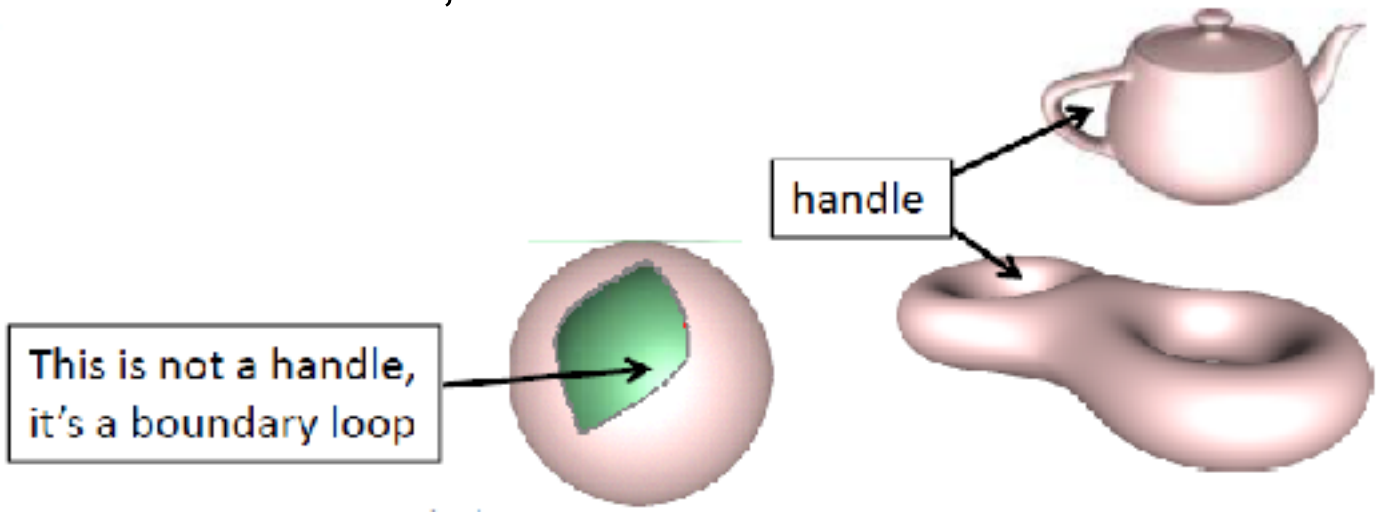
Euler-Poincaré Formula

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

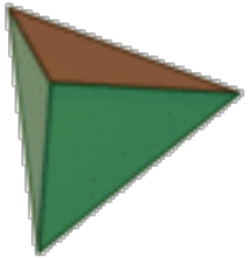
- **Euler characteristic = V-E+F=2(C-G)-B**
  - V : number of vertices
  - E : number of edges
  - F : number of faces
  - C : number of connected components
  - G : number of genus (holes, handles)
  - B : number of boundaries
- Euler Formula: V-E+F = 2 when C=1, G=0



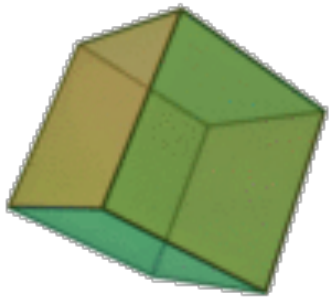
V = 16
E = 32
F = 16
C = 1
G = 1
B = 0
16 – 32 + 16 = 2 (1 - 1) - 0



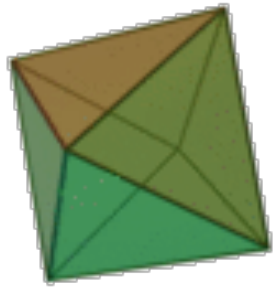
Euler Formula



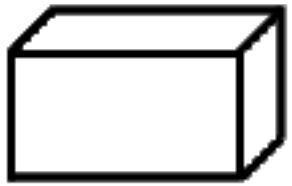
- Tetrahedron
  - V = 4
  - E = 6
  - F = 4
  - 4 - 6 + 4 = 2



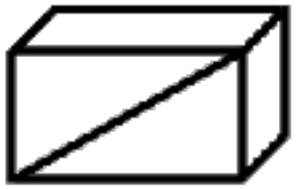
- Cube
  - V = 8
  - E = 12
  - F = 6
  - 8 -12 + 6 = 2



- Octahedron
  - V = 6
  - E = 12
  - F = 8
  - 6 -12 + 8 = 2



V = 8
E = 12
F = 6
8 - 12 + 6 = 2



V = 8
E = 12 + 1 = 13
F = 6 + 1 = 7
8 - 13 + 7 = 2

Average Valence of Closed Triangle Mesh

- **Theorem:** Average vertex degree in a closed manifold triangle mesh is ~6 (V-E+F=2)

- **Proof:** Each face has 3 edges and each edge is counted twice: 3F = 2E

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

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

How many pentagons六边形?

- every **vertex has valence 3**

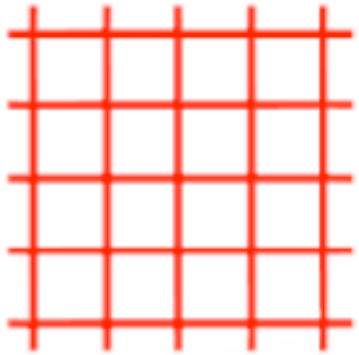
Euler Consequences

Triangle mesh statistics

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

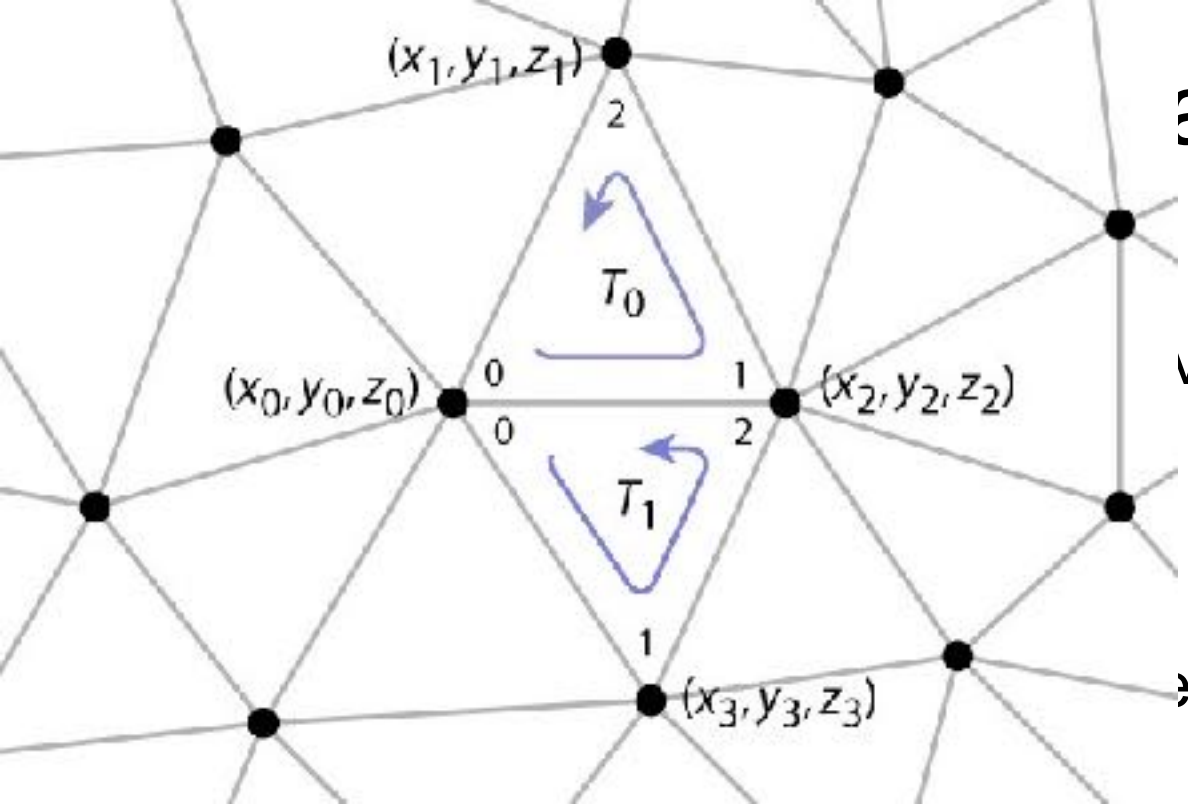
Quad mesh statistics

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



How do we actually encode all this data?

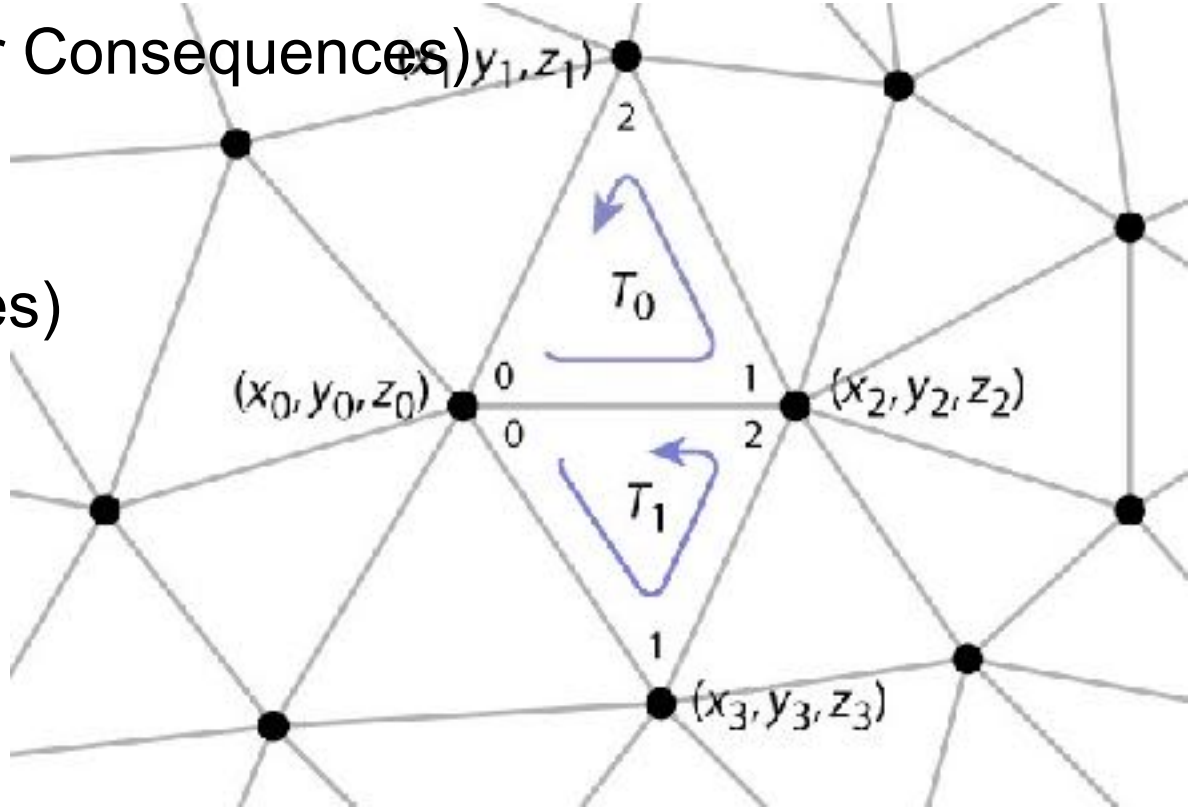
arate triangles



vertex

	[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
	⋮	⋮	⋮

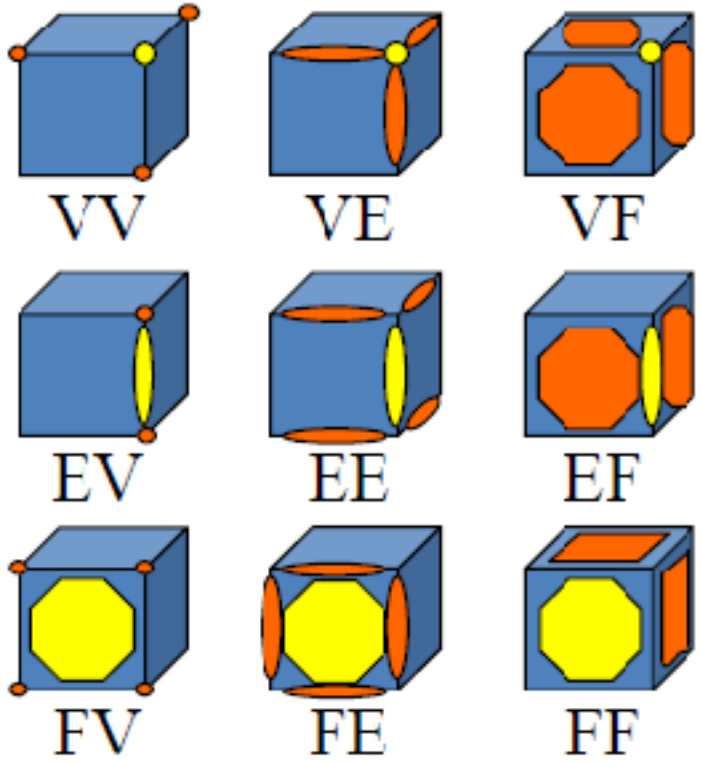
e, Euler Consequences)

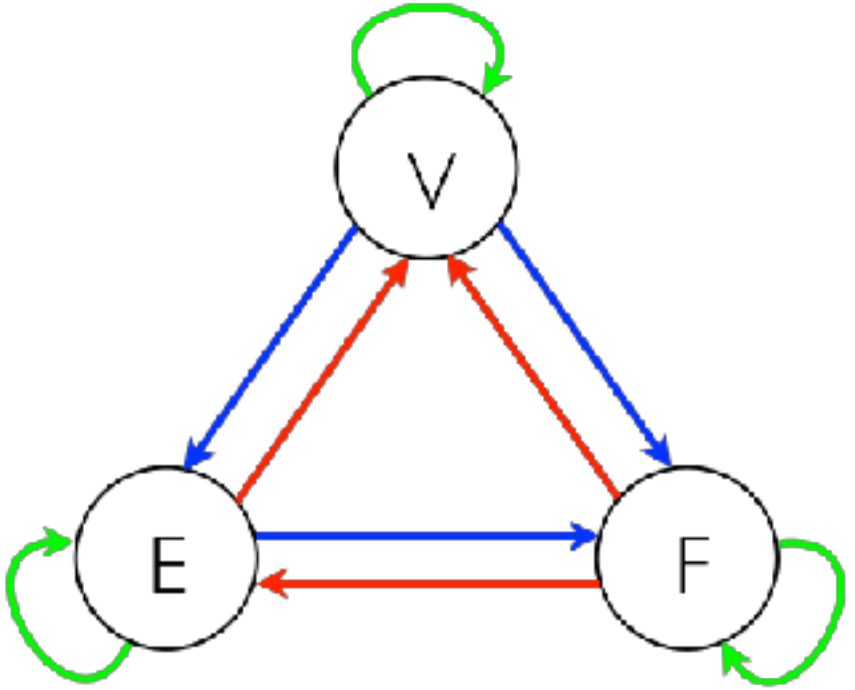


- **various problems**
  - wastes space (each vertex stored 6 times)
  - cracks due to roundoff
  - difficulty of finding neighbors at all

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

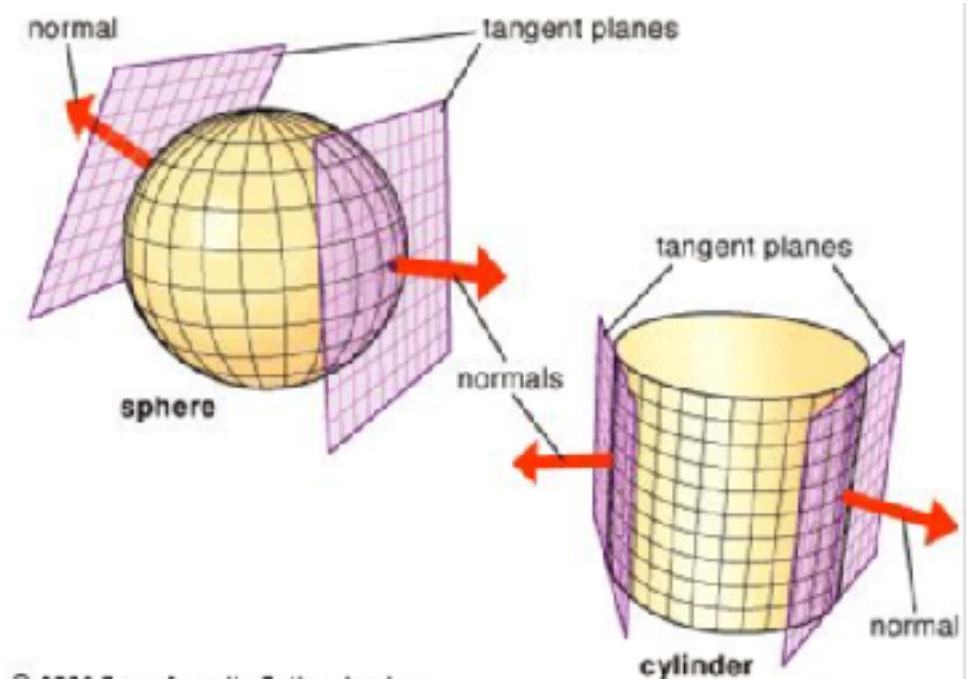




Knowing some types of relation, we can discover other (but not necessary all) topological information
e.g. if in addition to VV, VE and VF, we know neighboring vertices of a face, we can discover all neighboring edges of the face

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

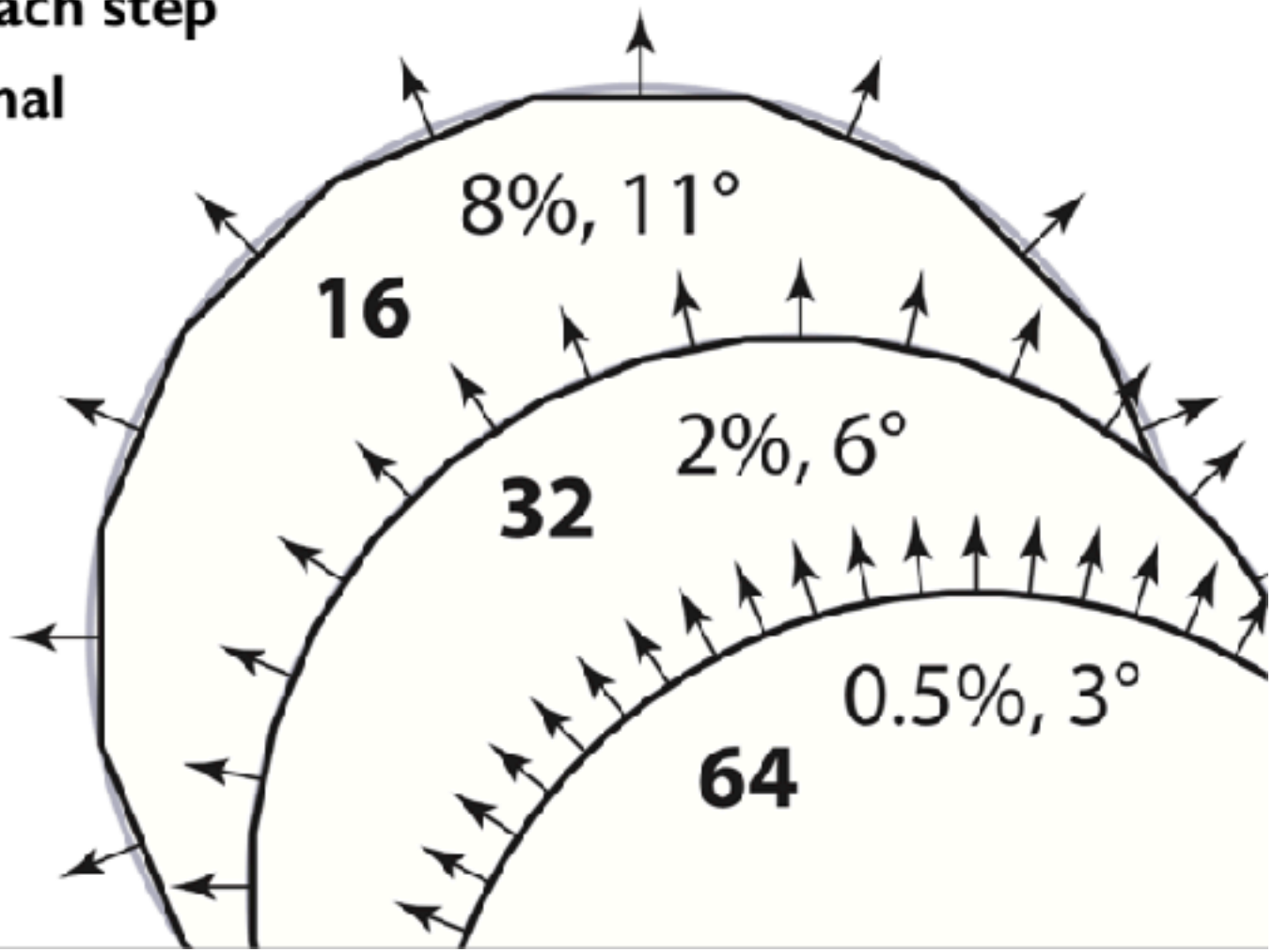


How to think about vertex normals

- **Piecewise planar approximation converges pretty quickly to the smooth geometry as the number of triangles increases**
  - For mathematicians: erroris O(*h*2)
- **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

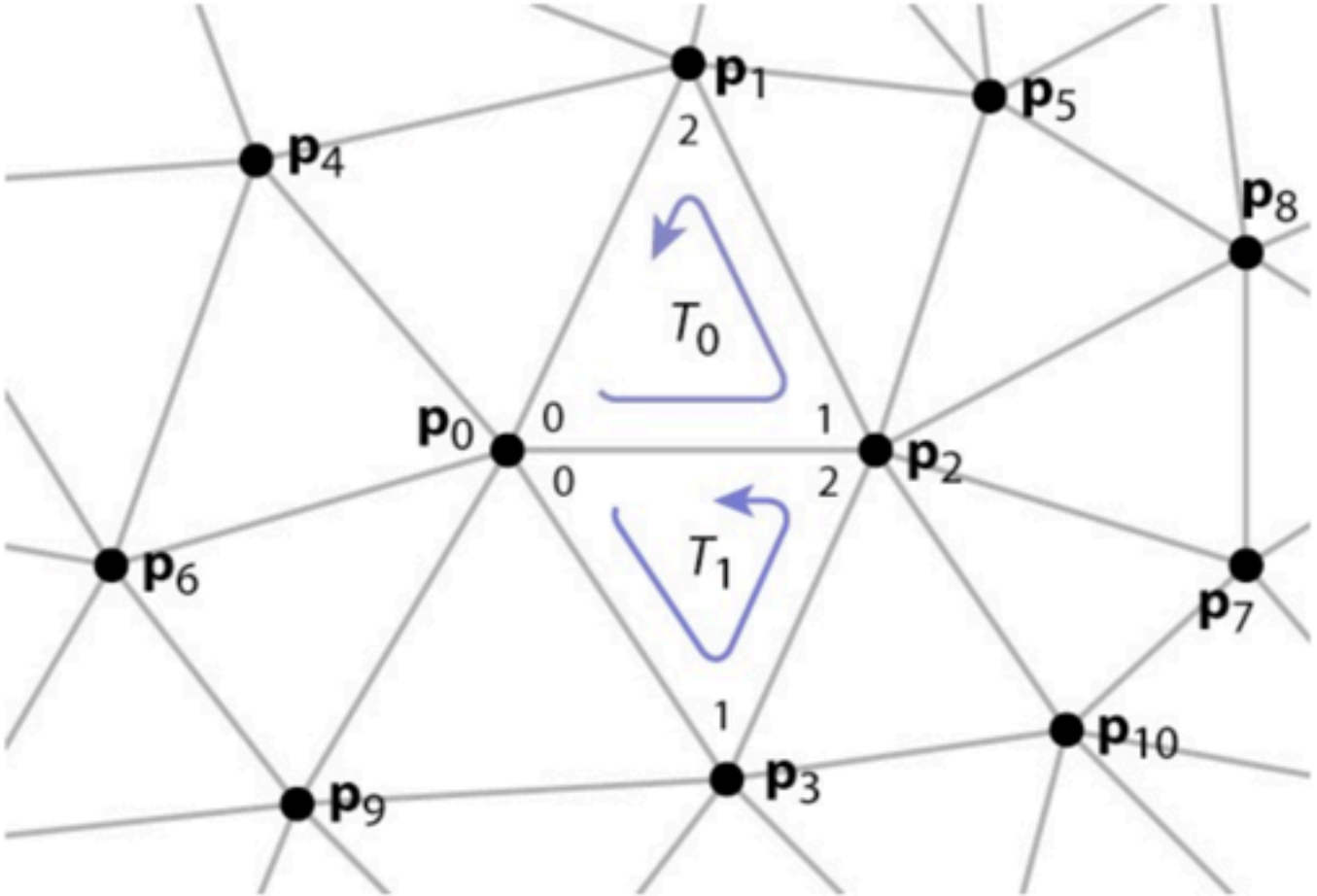


Indexed Face set - Shared Vertex (OBJ,OFF)

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

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

tInd[0]	0, 2, 1
tInd[1]	0, 3, 2
	⋮



Face-vertex meshes

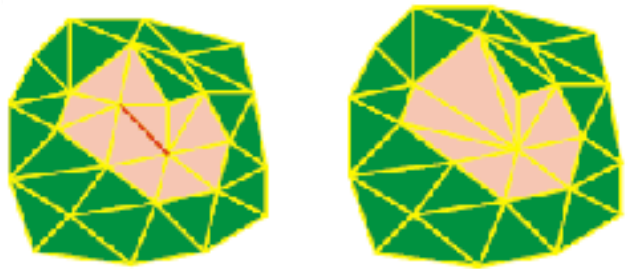
1. locating neighboring faces and vertices is constant time
2. a search is still needed to find all the faces surrounding a given face.
3. Other dynamic operations, such as splitting or merging a face, are also difficult with face-vertex meshes.

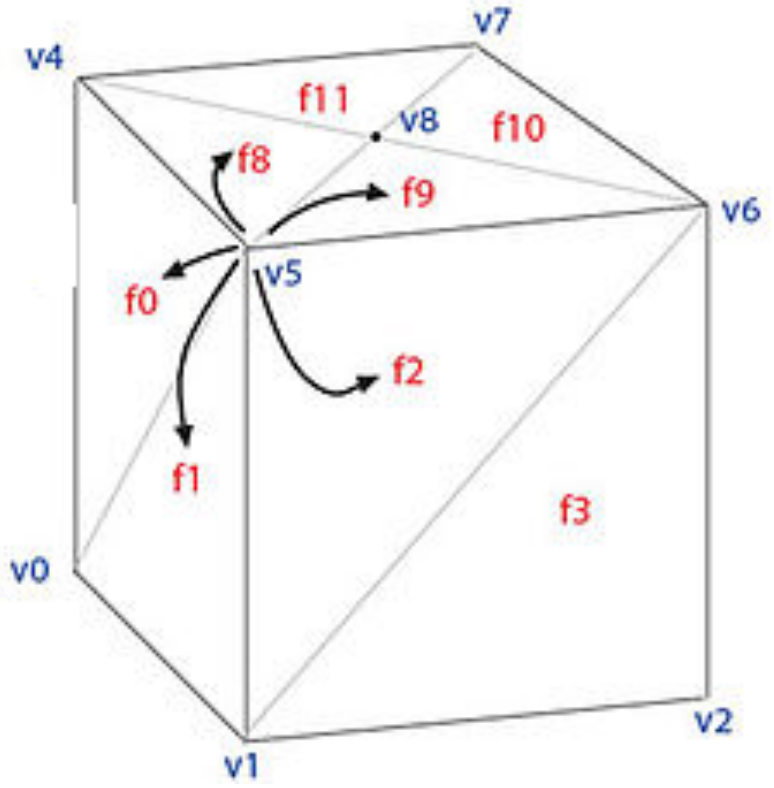
Face List

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



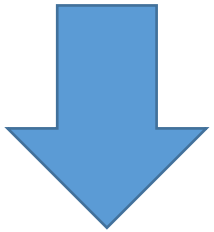


Transversal operations

- 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

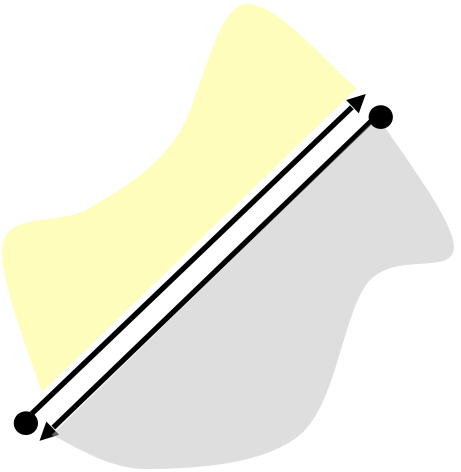
Edges always have the same topological structure



Efficient handling of polygons with variable valence

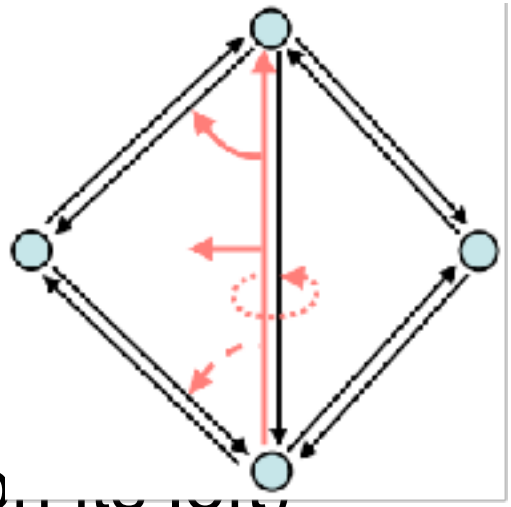
Half-Edge Data Structure

- 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 o… 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*)



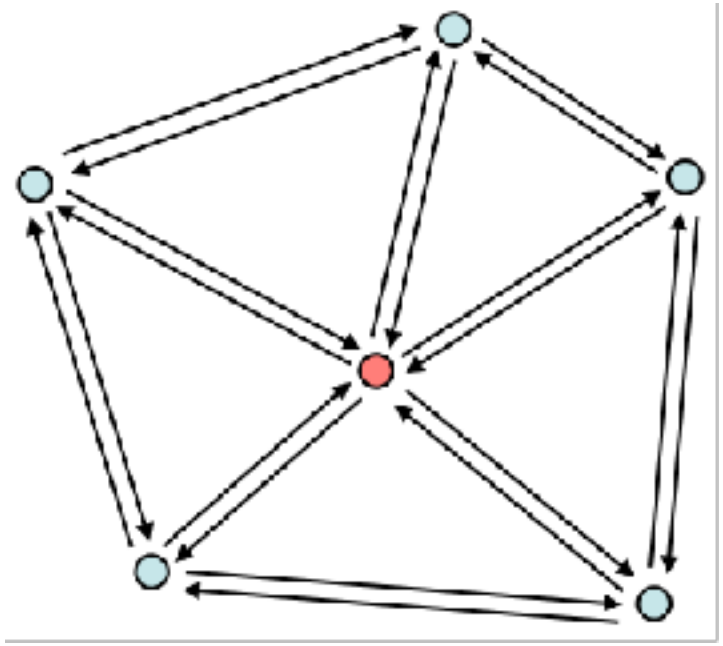
Half-Edge Data Structure

- 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

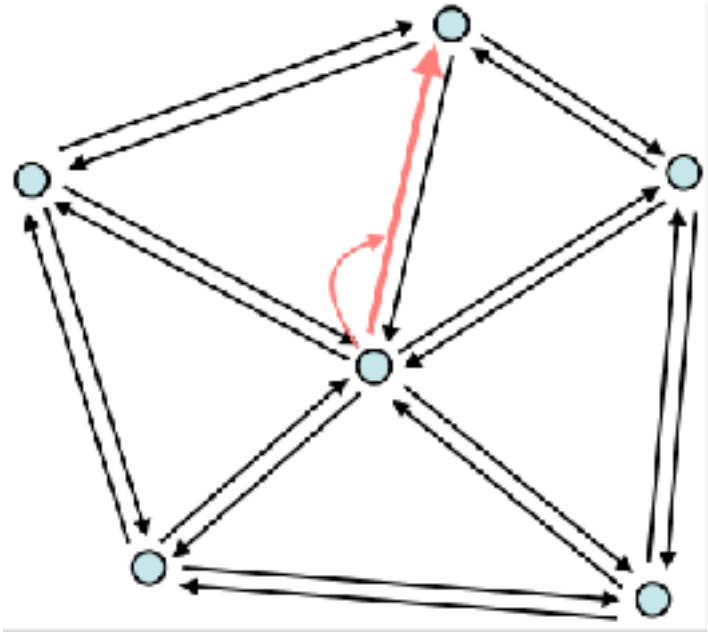
- One-ring traversal (V* relations):

    1. start at vertex



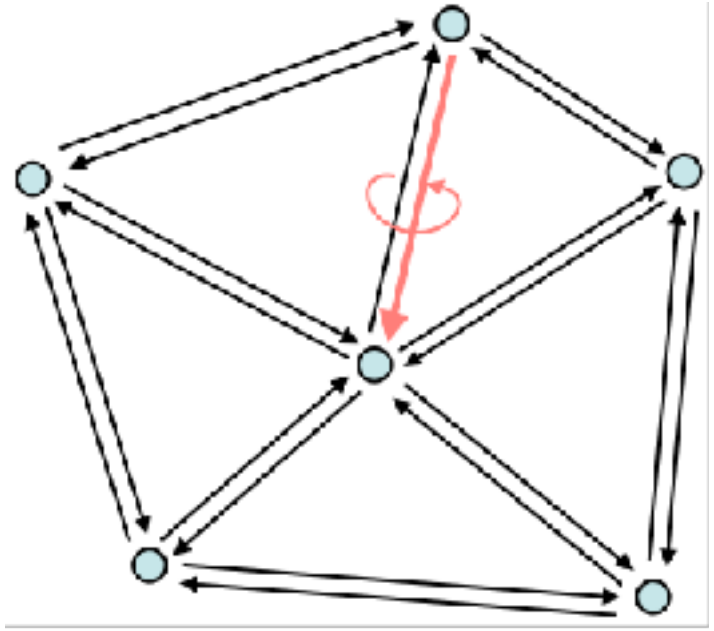
Half-Edge Data Structure

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



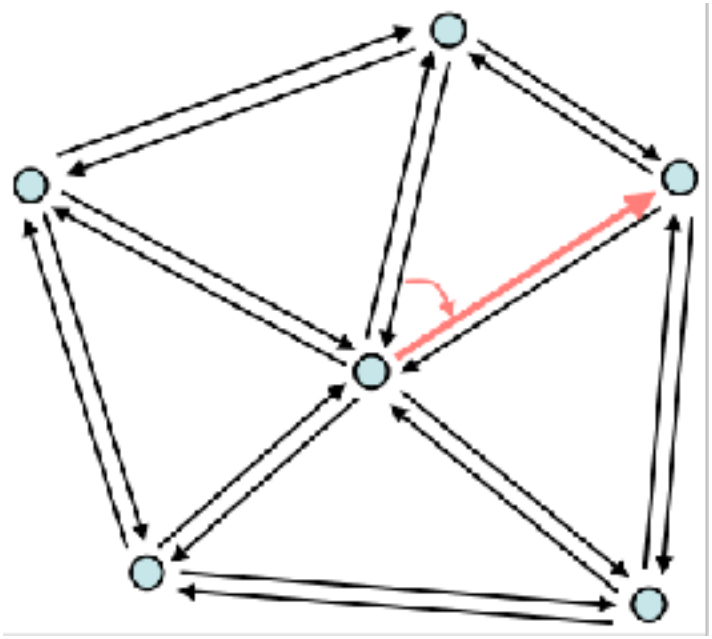
Half-Edge Data Structure

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



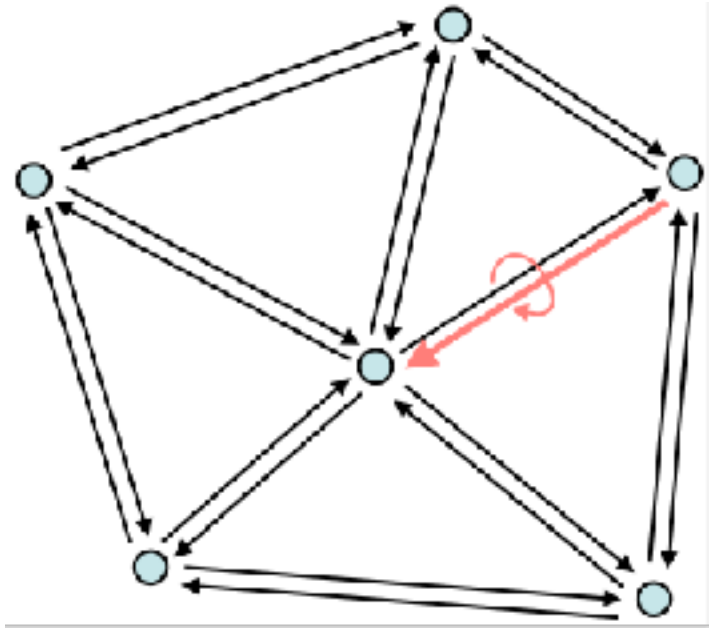
Half-Edge Data Structure

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



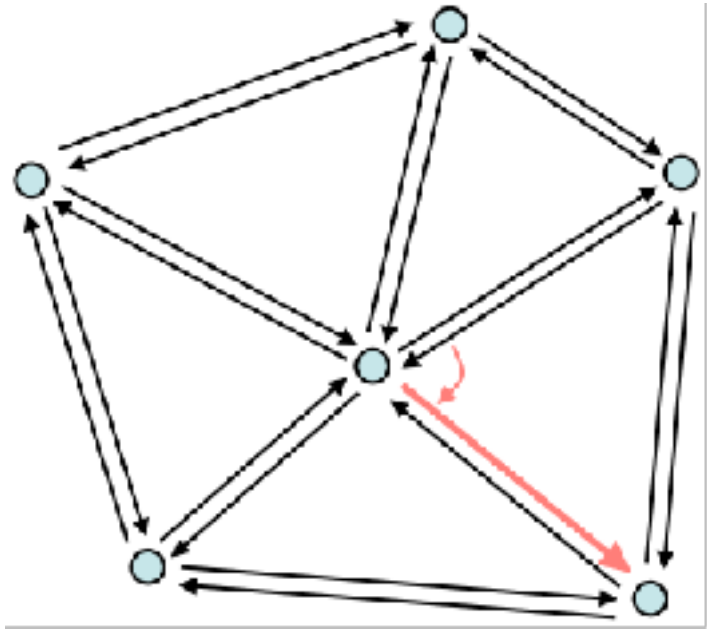
Half-Edge Data Structure

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

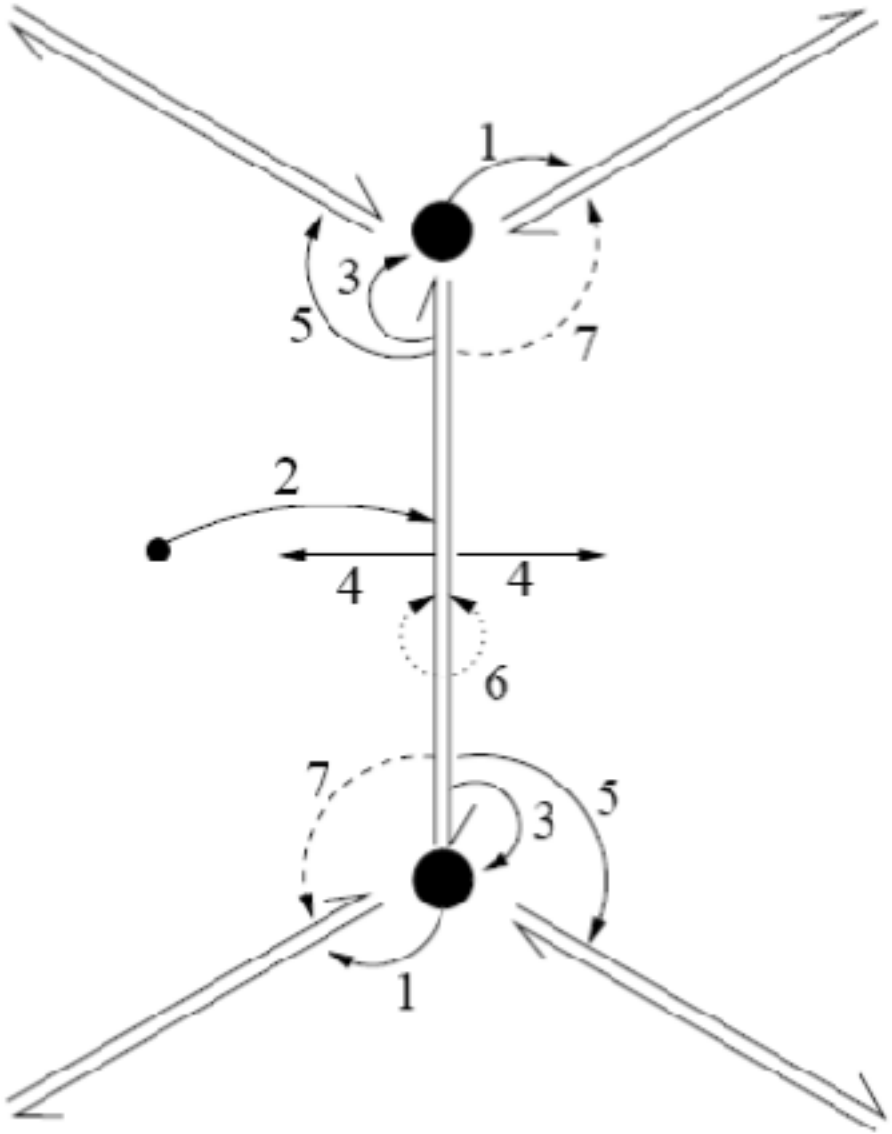


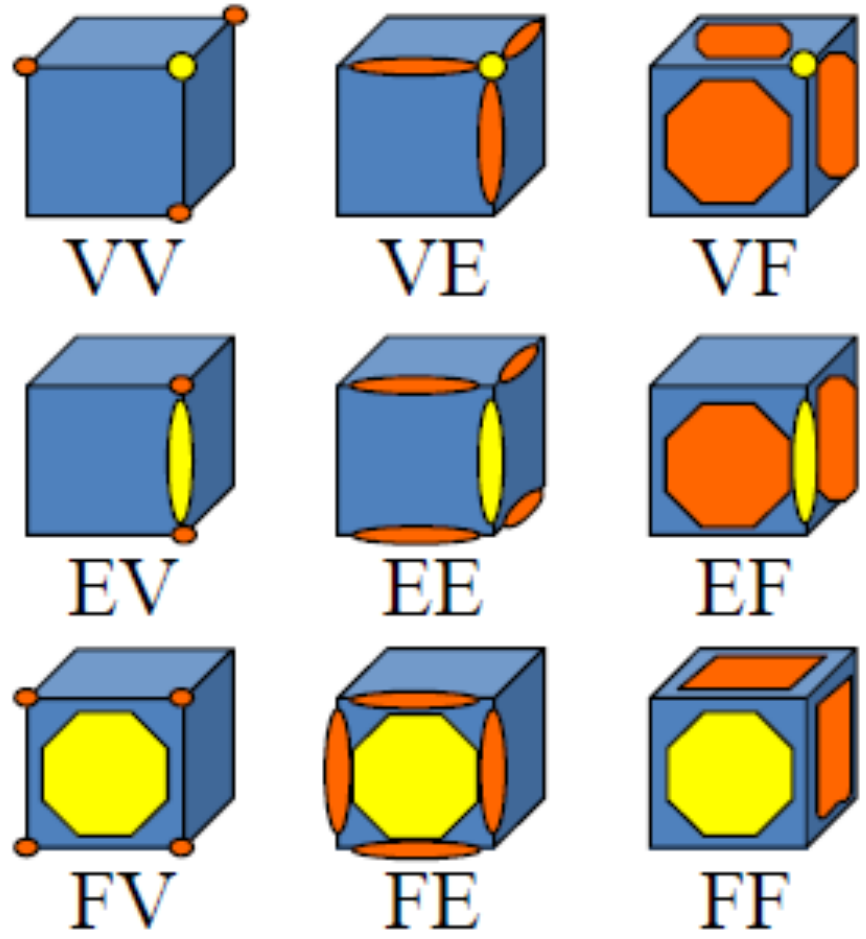
Half-Edge Data Structure

- 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.....



How HDS can -- OpenMesh





All basic queries take constant O(1) time!

Comparison of Polygon Mesh Data Structures

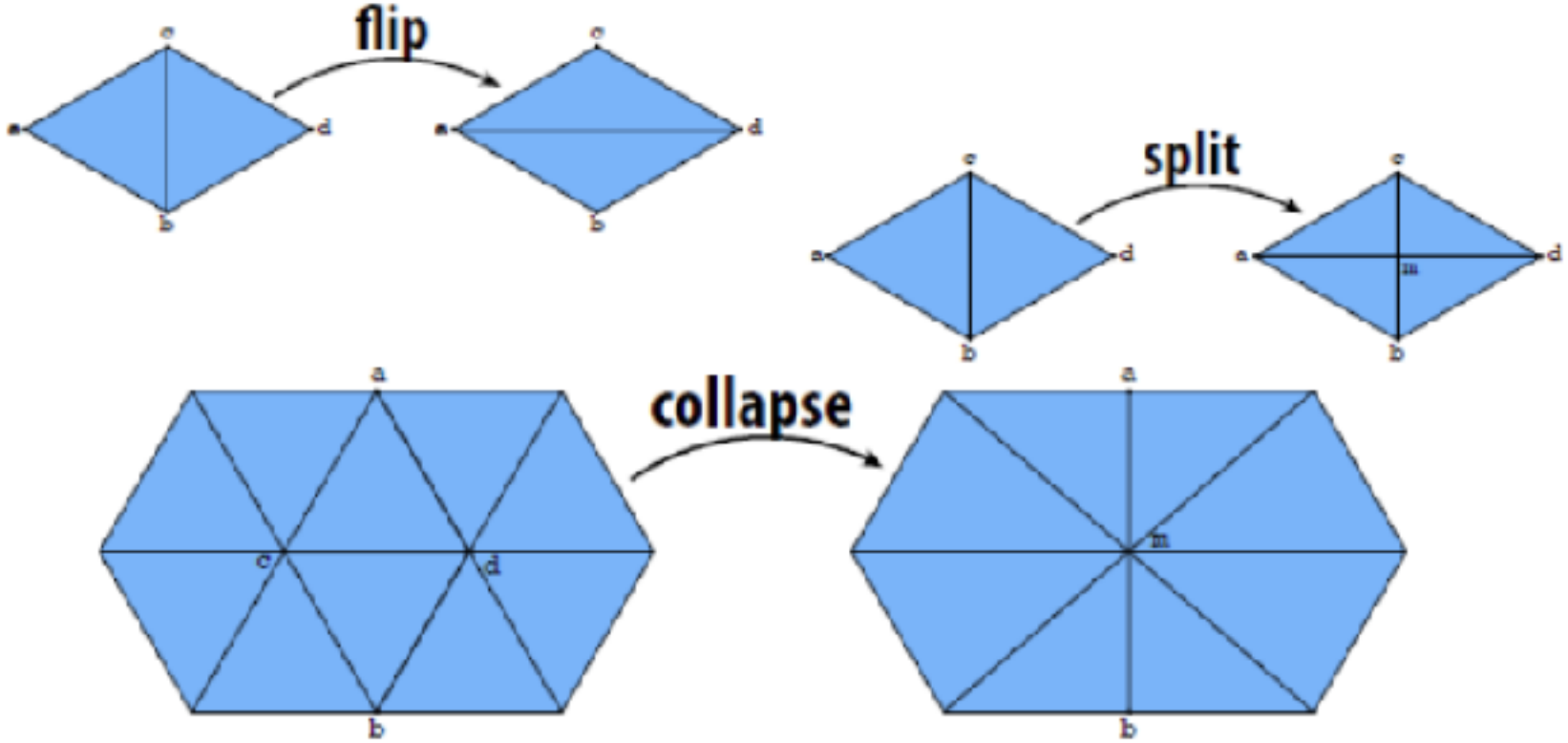
Case study: triangles.	Polygon Soup	Incidence Matrices	Halfedge Mesh
storage cost*	~3 x #vertices	~33 x #vertices	~36 x #vertices
constant-time neighborhood access?	NO	YES	YES
easy to add/remove mesh elements?	NO	NO	YES
nonmanifold geometry?	YES	YES	NO

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)

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:

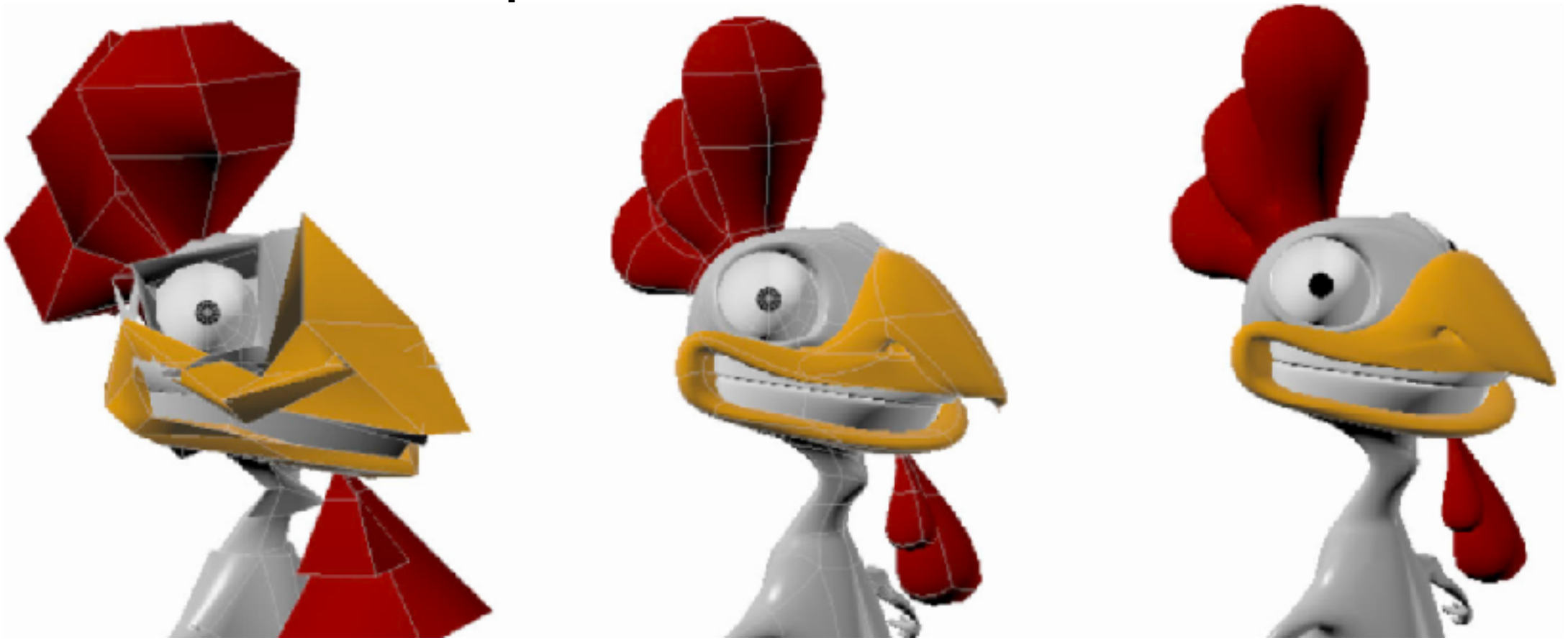


- How? Allocate/delete elements; reassigning pointers.
- Must be careful to preserve manifoldness!

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

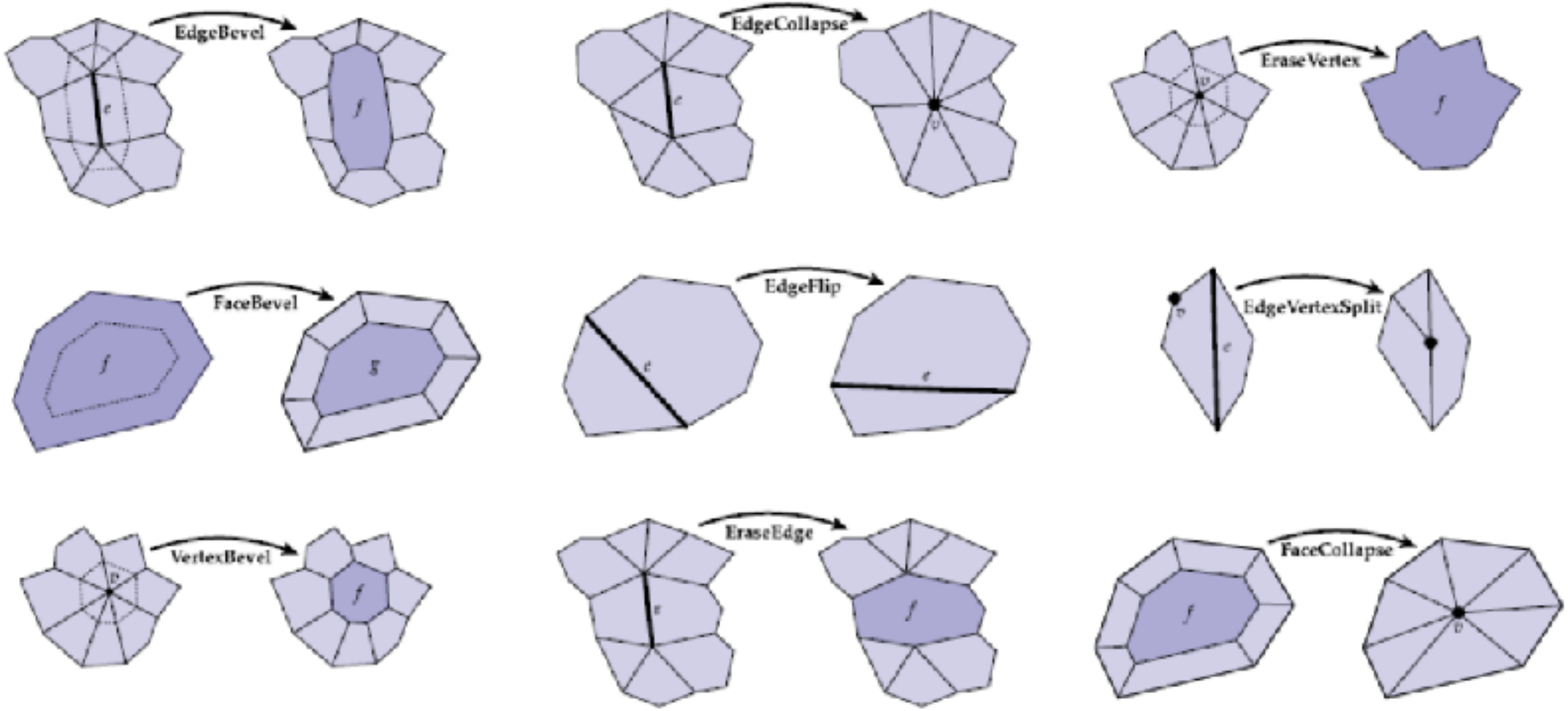
Subdivision Modeling

- 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!

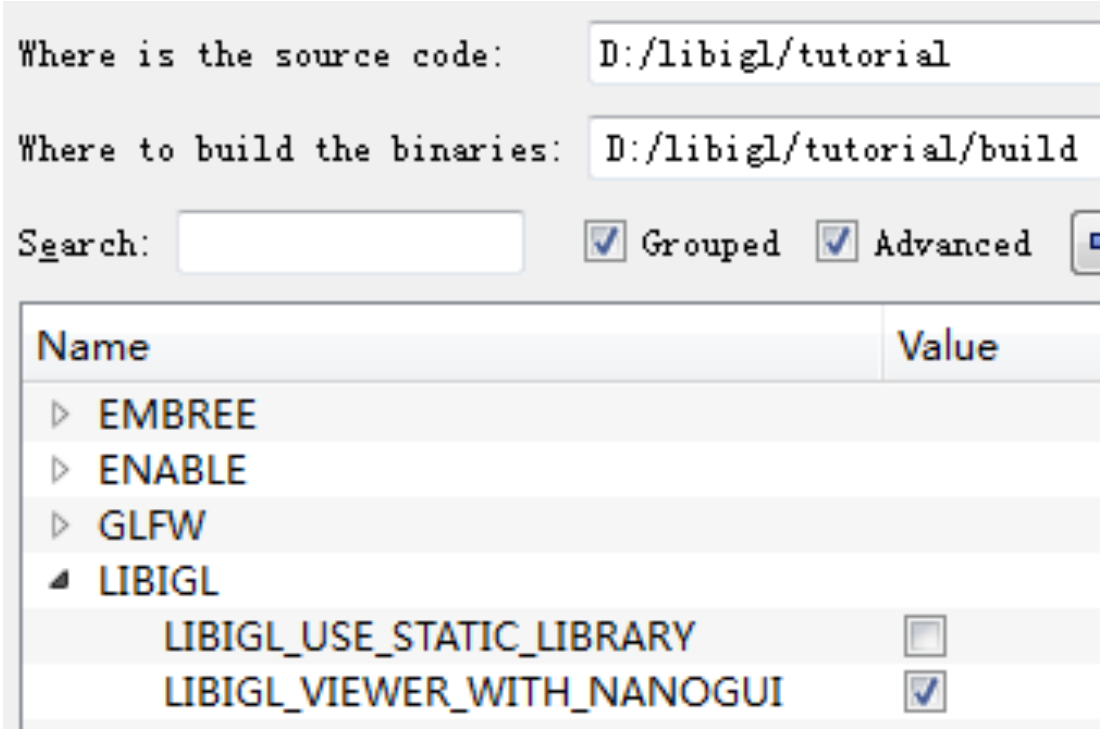
TOOLS

- 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

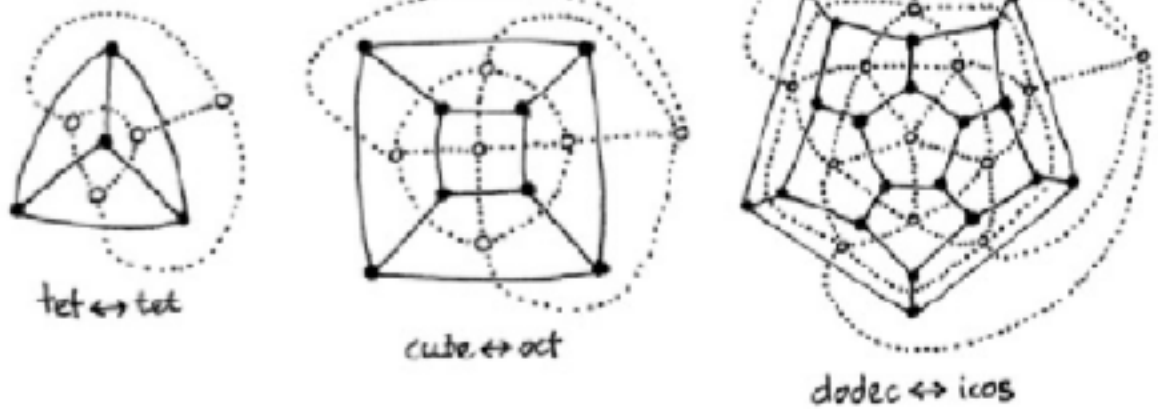
Lab

- 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

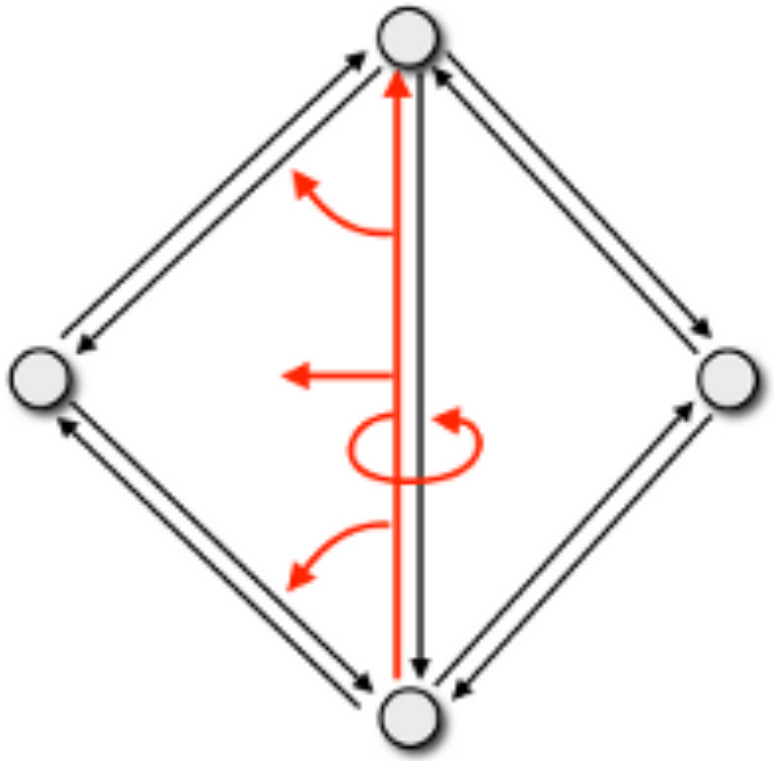
Paul Heckbert (former CMU prof.)
quadedge code - http://bit.ly/1QZLHos

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

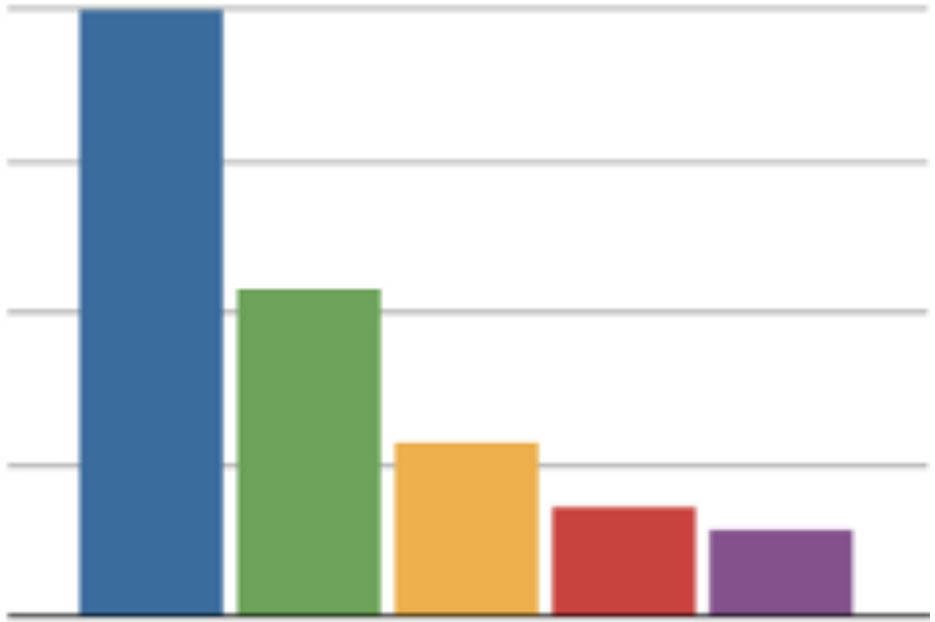


- Each stores local neighborhood information
- Similar tradeoffs relative to simple polygon list:
  - CONS: additional storage, incoherent memory access
  - PROS: better access time for individual elements, intuitive traversal of local neighborhoods
- (Food for thought: can you design a halfedge-like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

Design, Implementation, and Evaluation of theSurface_mesh Data Structure



```
class Vertex_iterator
{
public:

    /// Default constructor
    Vertex_iterator(Vertex v=Vertex()) : hnd_(v) {}

    /// Cast to the vertex the iterator refers to
    operator Vertex() const { return hnd_; }

    /// are two iterators equal?
    bool operator==(const Vertex_iterator& rhs) const
    {
        return hnd_==rhs.hnd_;
    }

    /// are two iterators different?
    bool operator!=(const Vertex_iterator& rhs) const
    {
        return !operator==(rhs);
    }

    /// pre-increment iterator
    Vertex_iterator& operator++()
    {
        ++hnd_.idx_;
        return *this;
    }

    /// pre-decrement iterator
    Vertex_iterator& operator--()
    {
        --hnd_.idx_;
        return *this;
    }

private:
    Vertex hnd_;
};
```



[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!