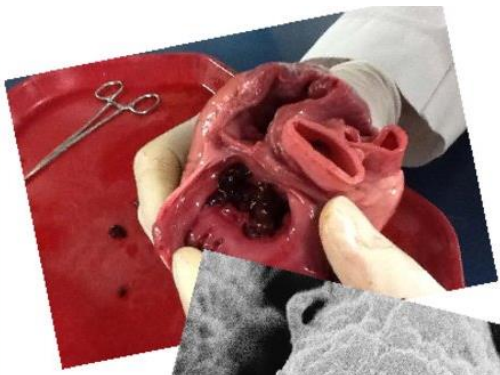


- **Geometry representations**
 - **Polygon Meshes**

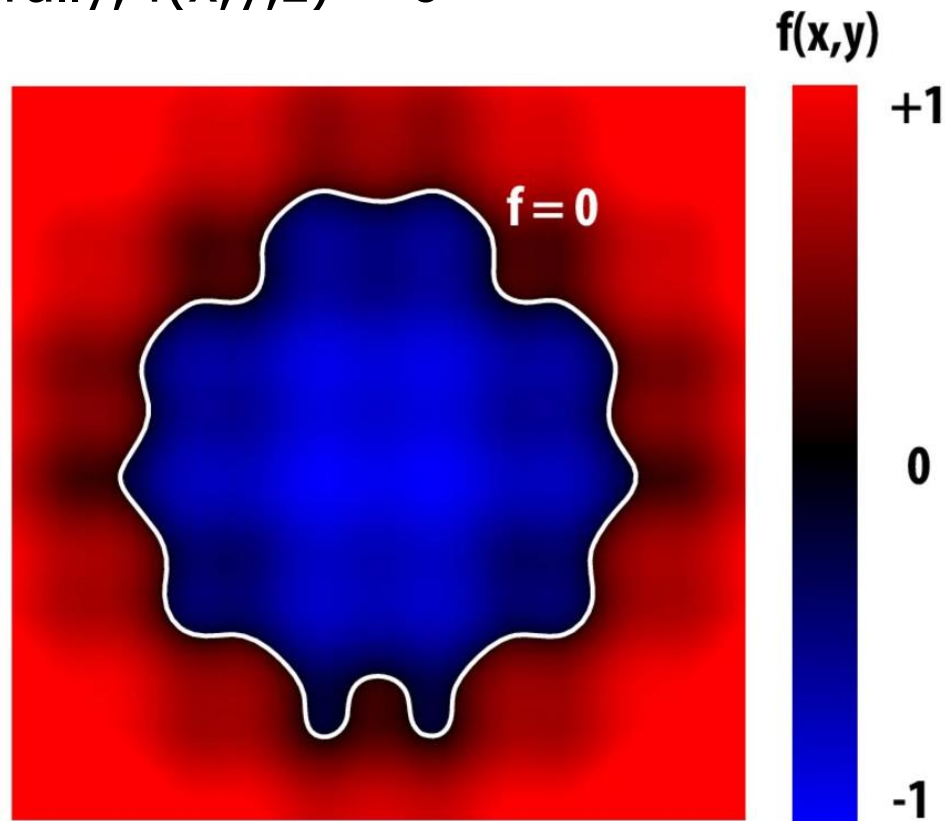


Geometry representations

- Explicit
 - Point cloud
 - Polygon mesh
 - Subdivision surfaces, NURBS
 -
- Implicit
 - Algebraic surfaces
 - Level set
 - ...
- Each choice best suited to a different task/type of geometry

Implicit Representations

- Points are not known directly, but satisfy some relationship
- E.g., unit sphere is all points such that $x^2+y^2+z^2=1$
- More generally, $f(x,y,z) = 0$



Implicit surfaces

Given an implicit surface $f(x,y,z) = x^2 + y^2 + z^2 - 1$

- Find ordered points on it
 - Tasks like sampling is hard
- Determine if a given point is *inside* it

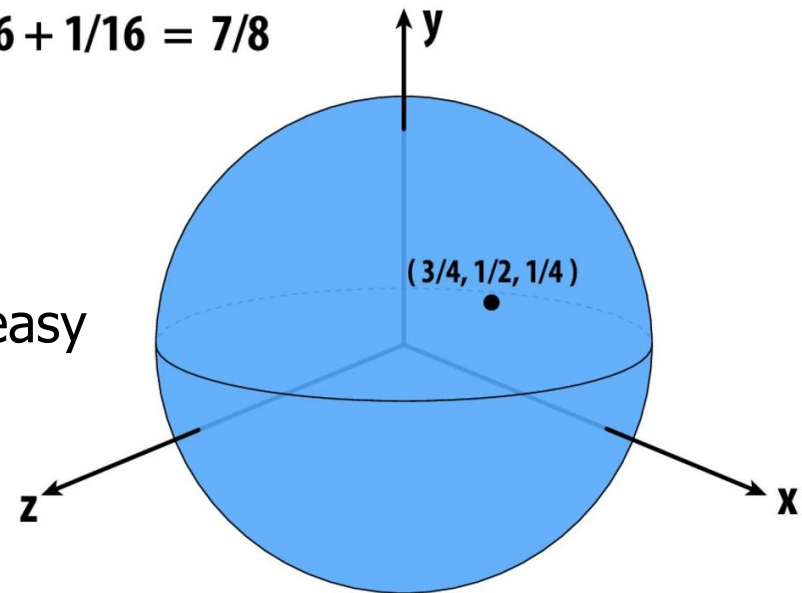
How about the point $(3/4, 1/2, 1/4)$?

$$9/16 + 4/16 + 1/16 = 7/8$$

$$7/8 < 1$$

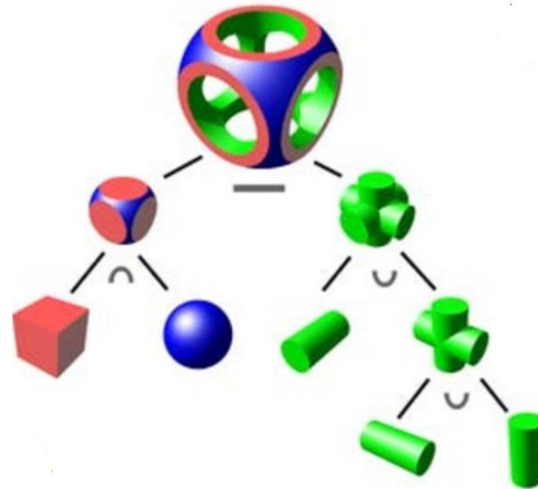
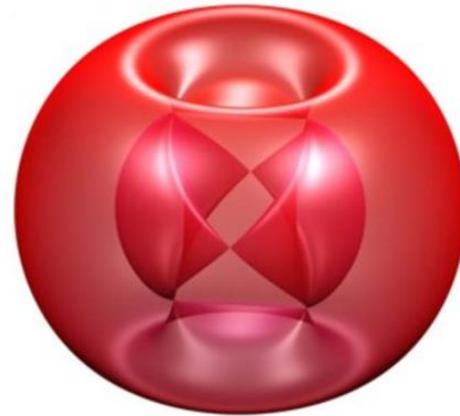
YES.

- Tasks like inside/outside test is easy



Many implicit representations

- Algebraic surfaces
- Constructive solid geometry
- Blobby surfaces
- Level set methods
-



Algebraic surfaces

- Surface is the zero set of a polynomial in x, y, z
- Examples



$$x^2 + y^2 + z^2 = 1$$



$$(R - \sqrt{x^2 + y^2})^2 + z^2 = r^2$$

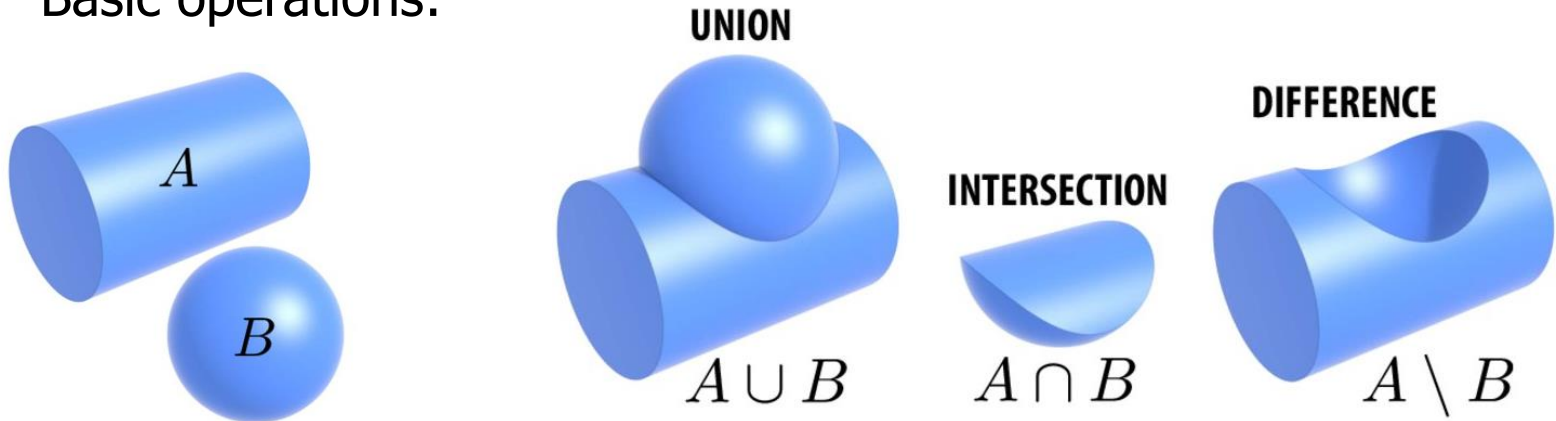


$$\left(x^2 + \frac{9y^2}{4} + z^2 - 1\right)^3 = x^2 z^3 + \frac{9y^2 z^3}{80}$$

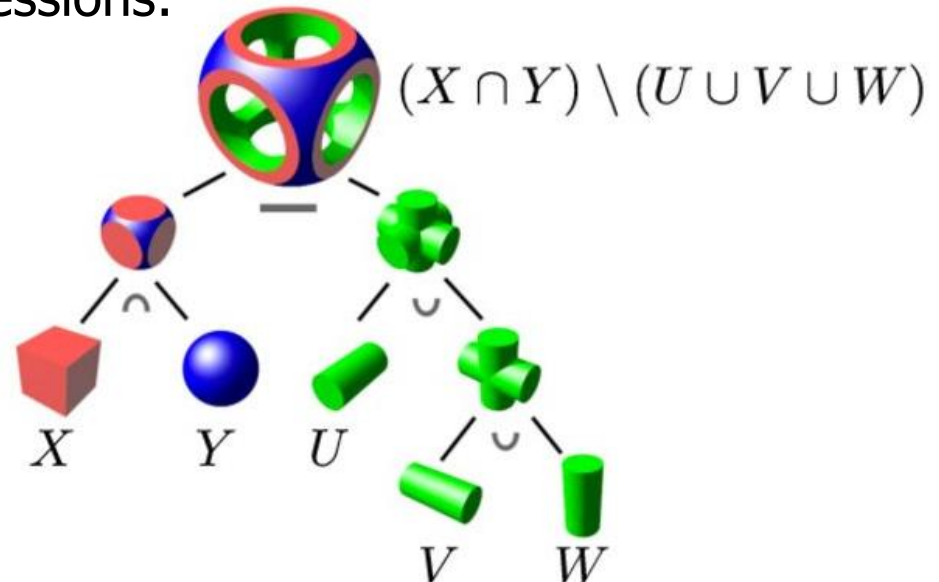
- What about more complicated shapes?
 - Very hard to come up with the polynomials!

Constructive Solid Geometry

- Build more complicated shapes via Boolean operations
- Basic operations:

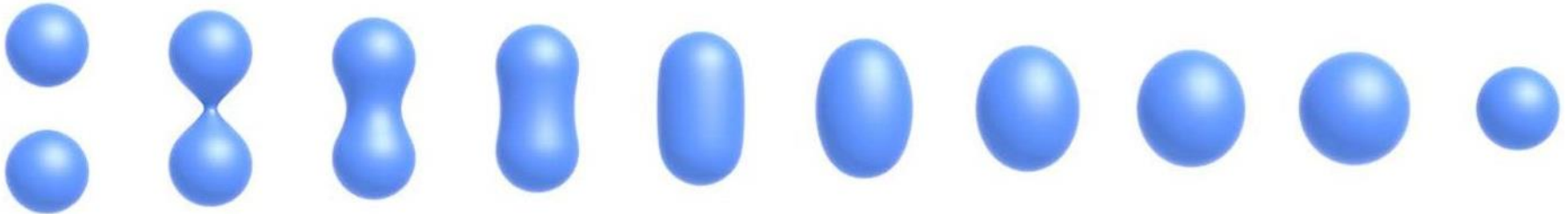


- Then chain together expressions:



Bloppy Surfaces

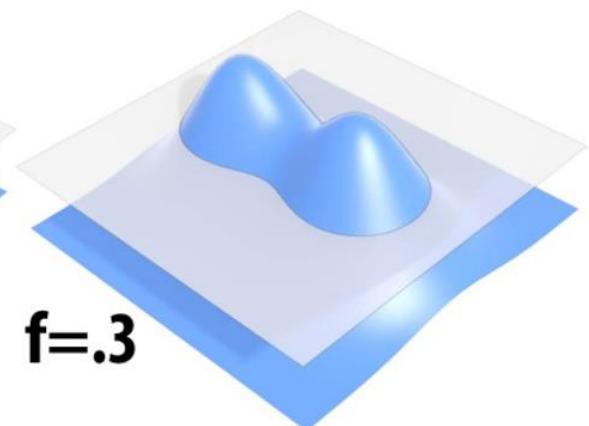
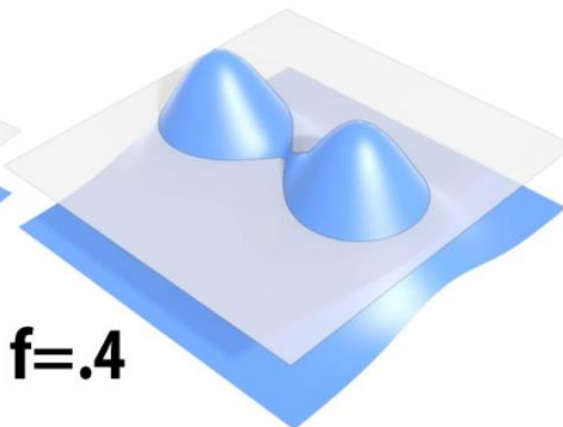
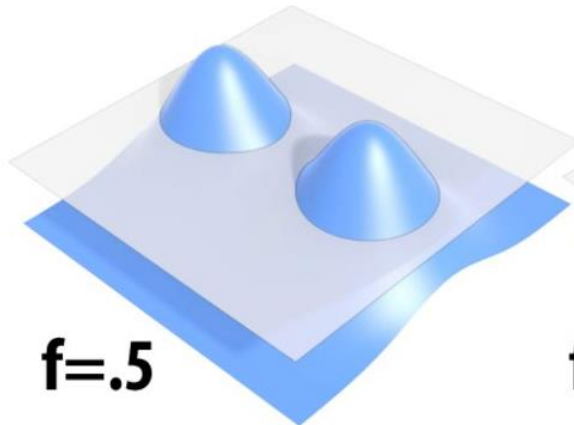
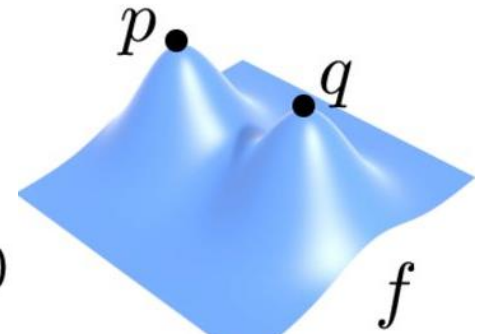
- Instead of Booleans, gradually blend surfaces together:



- Easier to understand in 2D

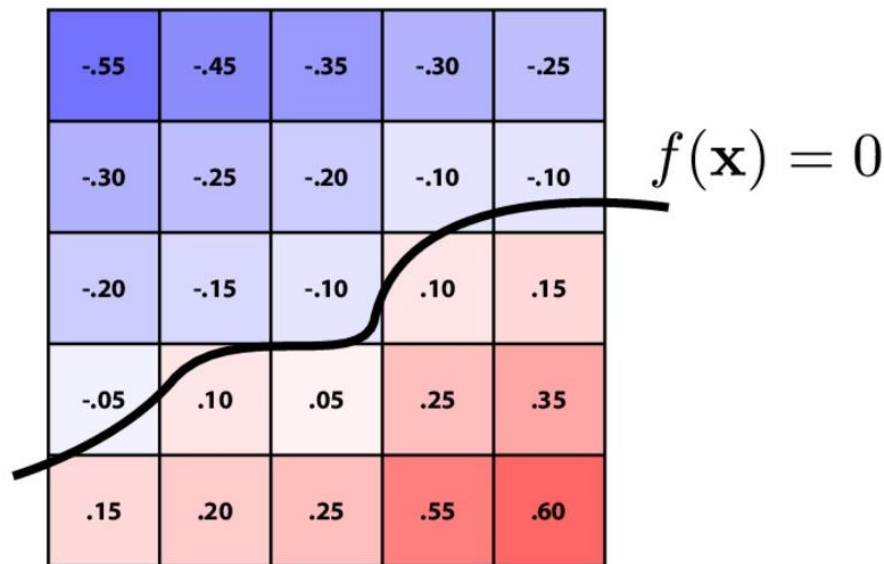
$$\phi_p(x) := e^{-|x-p|^2} \quad \text{(Gaussian centered at p)}$$

$$f := \phi_p + \phi_q \quad \text{(Sum of Gaussians centered at different points)}$$



Level Set Methods

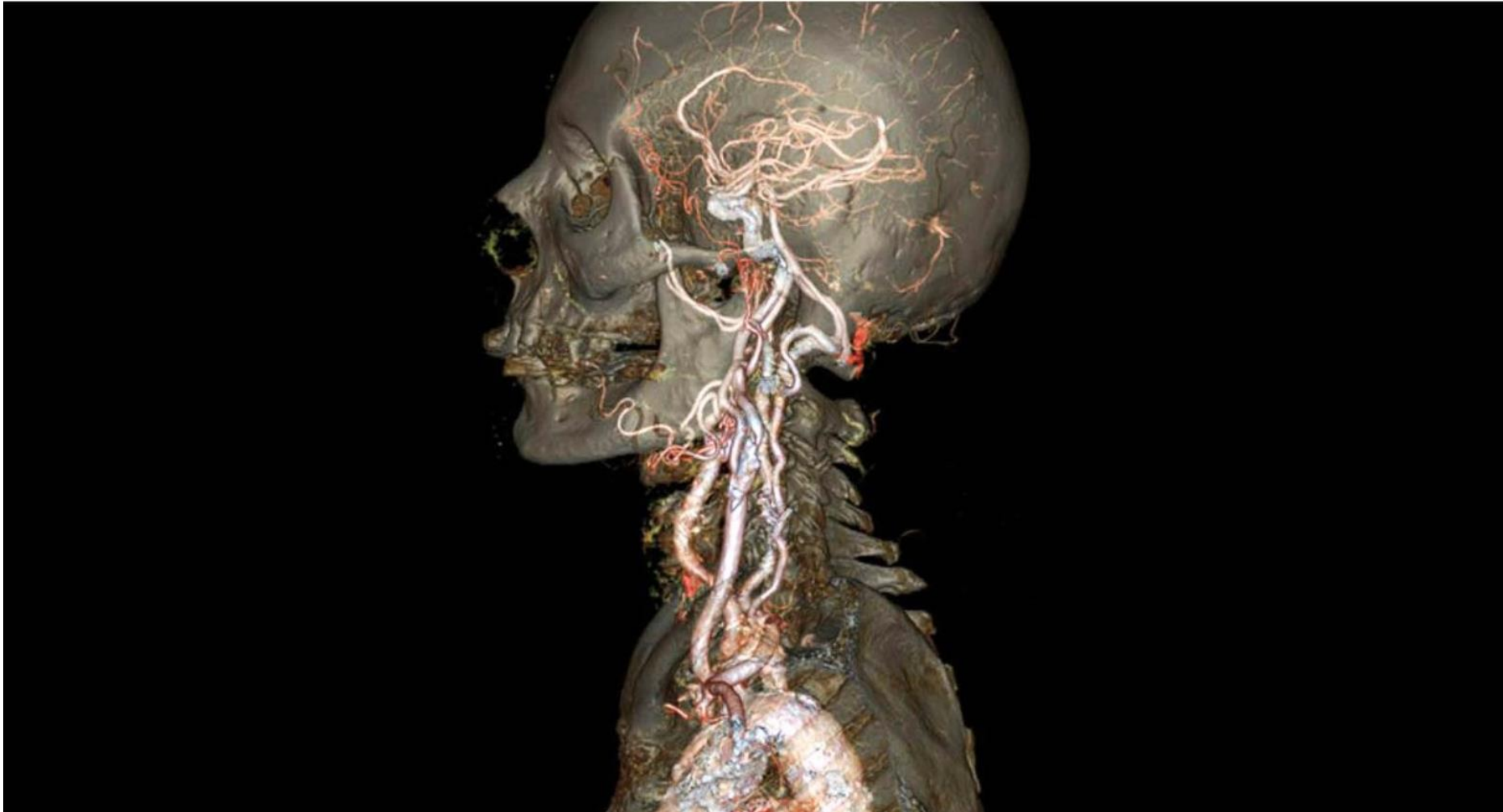
- Implicit surfaces have some nice features (e.g., merging/splitting)
- But hard to describe complex shapes in closed form
- Alternative: store a grid of values approximating function



- Surface is found where **interpolated** values equal zero
- Provide much more explicit control over shape
- Unlike closed-form expressions, run into problems of aliasing!

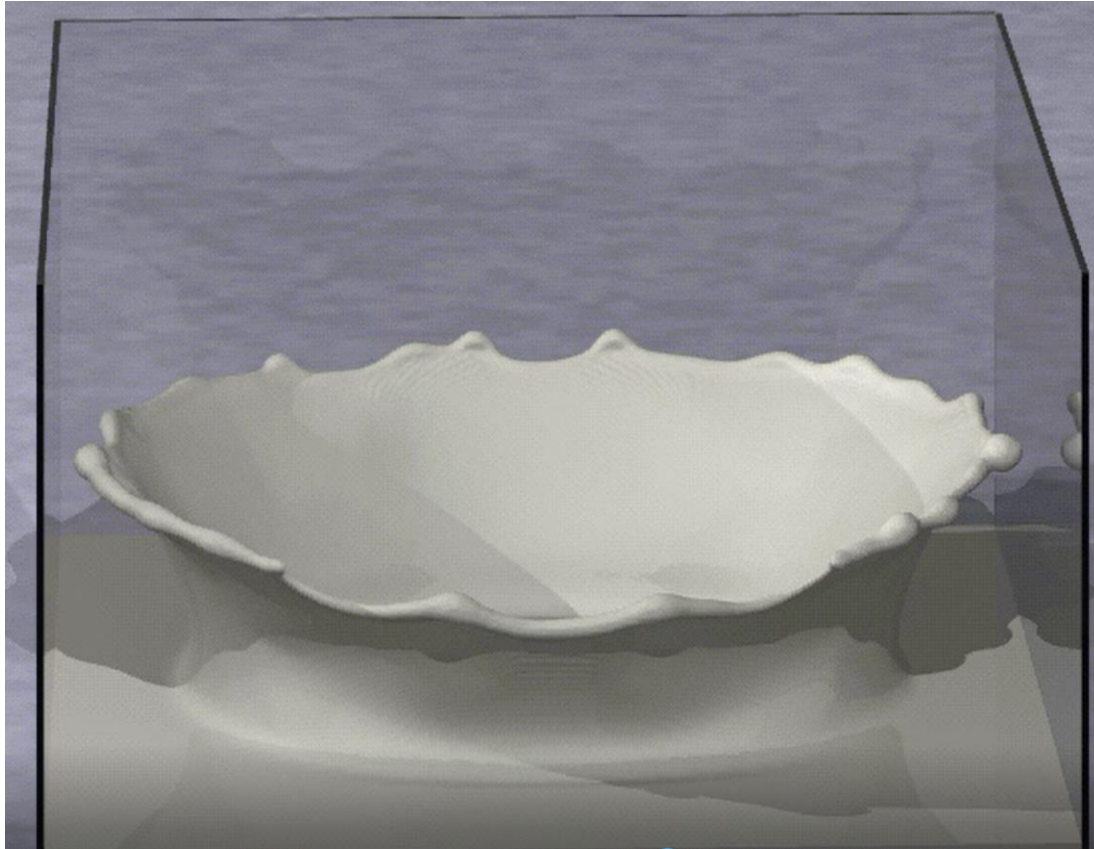
Level sets from Medical Data (CT, MRI, etc.)

- Level sets encode, e.g., constant tissue density



Level sets in physical simulation

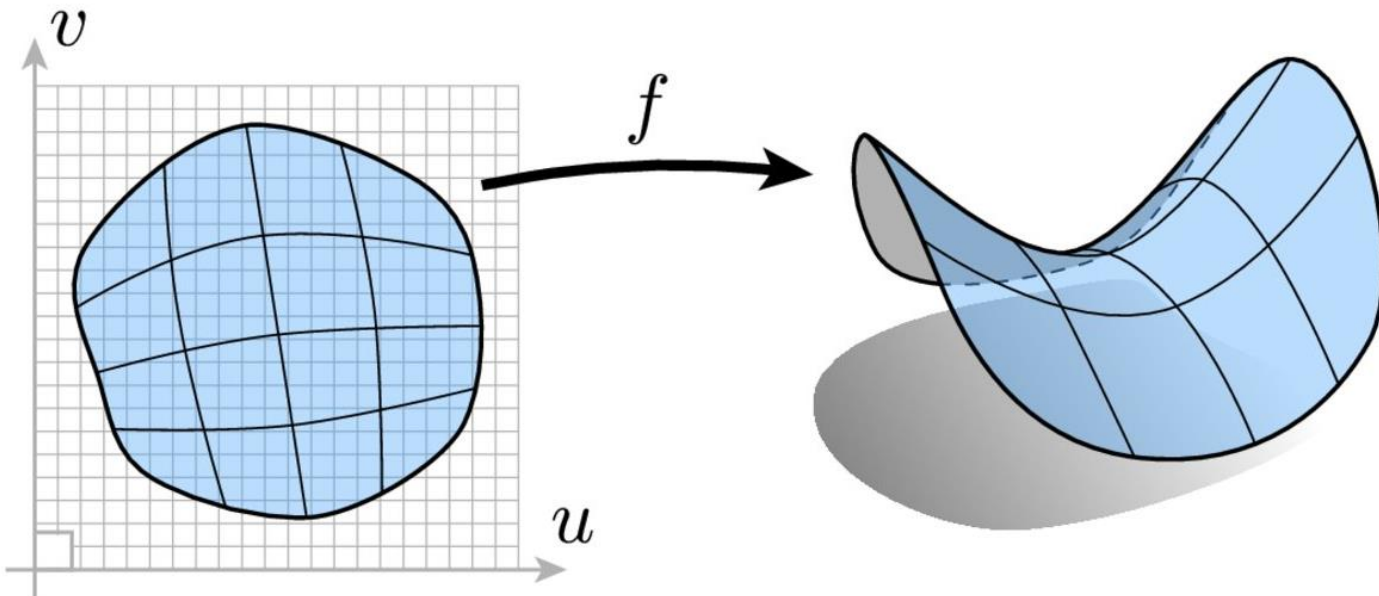
- Level set encodes distance to air-liquid boundary



- <http://physbam.Stanford.edu>

Explicit Representations

- All points are given directly
- E.g., points on sphere are $(\cos(u) \sin(v), \sin(u) \sin(v), \cos(v))$,
for $0 \leq u < 2\pi$ and $0 \leq v \leq \pi$
- More generally: $f : \mathbb{R}^2 \rightarrow \mathbb{R}^3; (u, v) \mapsto (x, y, z)$



Explicit Surfaces

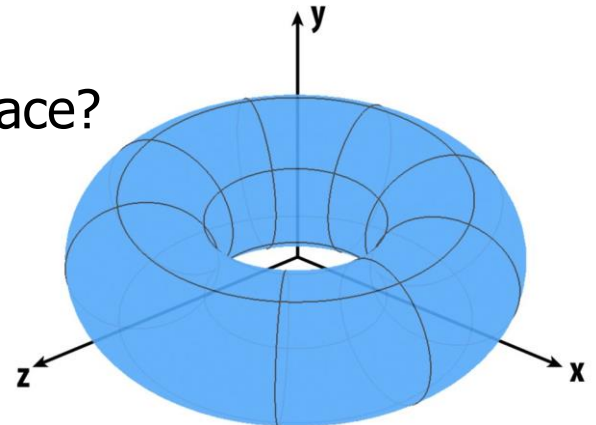
Given a surface $f(u,v)$: $(\cos(u) \sin(v), \sin(u) \sin(v), \cos(v))$,
for $0 \leq u < 2\pi$ and $0 \leq v \leq \pi$

- Find an ordered list of points on the surface
 - Just plug in, e.g., $u=0, v=0, 0.1, 0.2, \dots$
 - tasks like sampling is easy

Given a surface

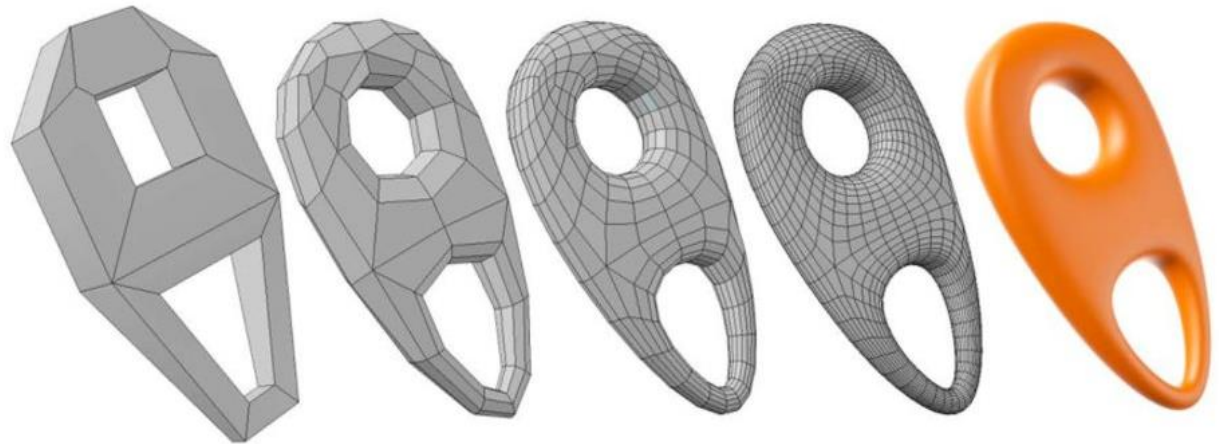
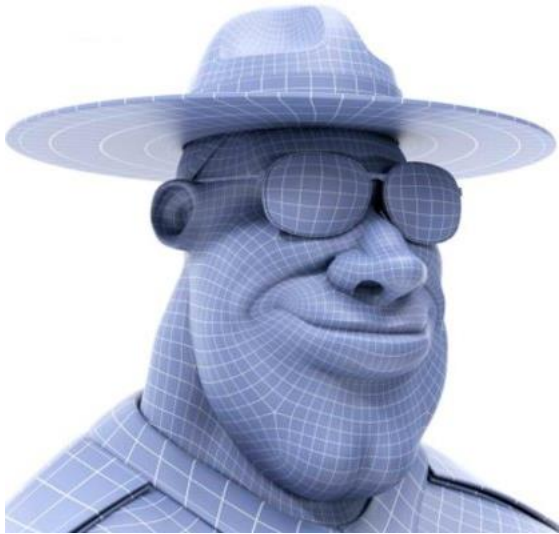
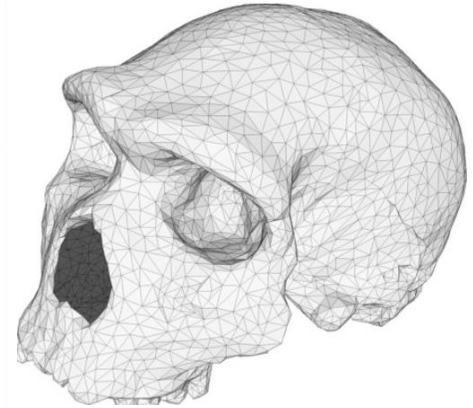
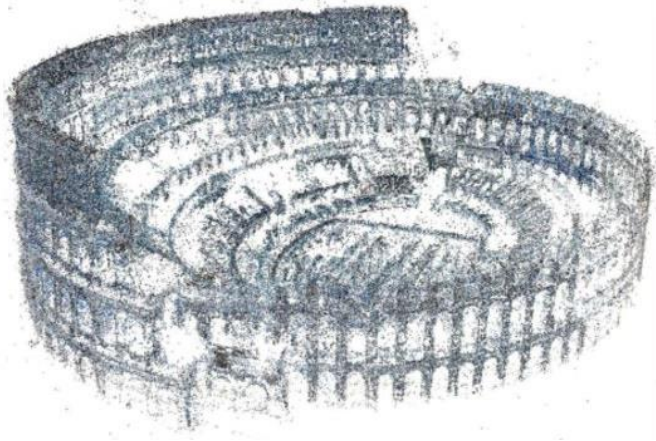
$$\mathbf{f}(u,v) = ((2+\cos u)\cos v, (2+\cos u)\sin v, \sin u)$$

- Is the point $(1.96, -0.39, 0.9)$ on the surface?
 - Tasks like inside/outside task is hard



Many explicit representations

- Point clouds
- Polygon meshes
- Subdivision surfaces
- NURBS
-



What is a Polygon Mesh?

- A Mesh is a pair (P, K) , where
 - P contains the **geometry information**
i.e., a set of **vertex positions**
$$P = \{ p_i \in R^3 \mid 1 \leq i \leq n \}$$
 - K contains all **topological (connectivity) information**

K is an abstract simplicial complex

i.e., K contains subsets of $\{1, \dots, N\}$

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

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

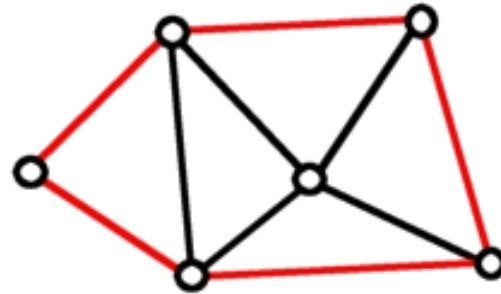
What is a Mesh?

- Edges are generally not oriented, denoted $\{\mathbf{v}_i, \mathbf{v}_j\}$
- Each edge must belong to at least one face, i.e.

$$e = \{j, k\} \in E \text{ iff } \exists f = \{i_1, \dots, j, k, \dots, i_{n_f}\} \in F$$

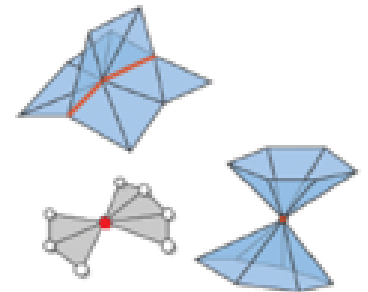
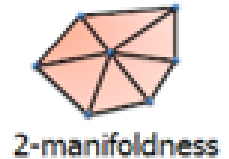
- Each vertex must belong to at least one edge, i.e.

$$v = \{j\} \in V \text{ iff } \exists e = \{i, j\} \in E$$



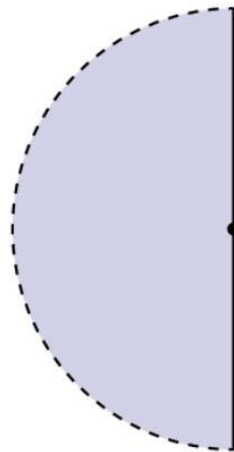
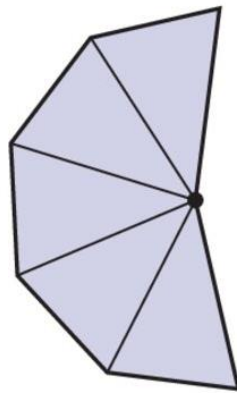
Manifold Meshes

- Manifold meshes:
 - In images, every pixel has 4 neighbors.
 - In meshes, want to have
 - Every edge is contained in 2 faces
 - Adjacent polygons of every vertex form 1 disk/fan
- Manifold meshes keep data structure/algorithms simple and efficient
- Closed mesh if no boundary edges.
- Watertight meshes
 - Suitable for simulations and 3D printing.
- Euler's formula (see later slide) applies.



What about boundary?

- The boundary is where the surface “ends”
- Locally, looks like a **half** disk
- Globally, each boundary forms a loop



YES



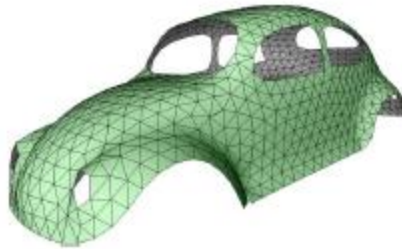
Manifold meshes:

- Each Boundary edge has 1 adjacent face
- Adjacent vertices of a boundary vertex forms a half disk

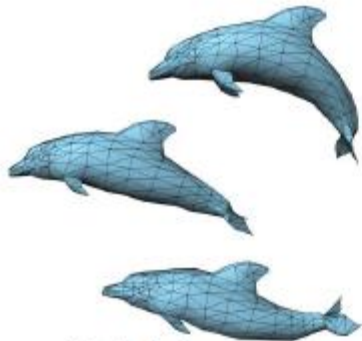
Types of Polygon Meshes



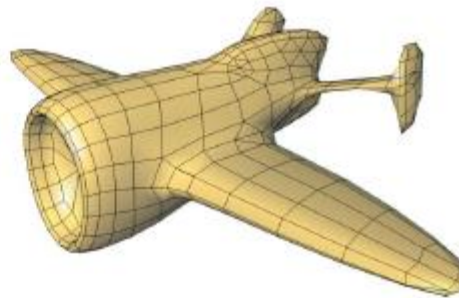
Single component,
closed, triangular,
2-manifold



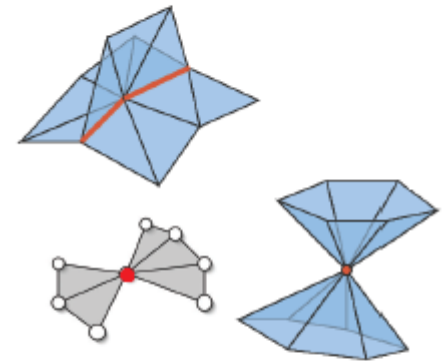
With boundaries
2-manifold



Multiple components
2-manifold



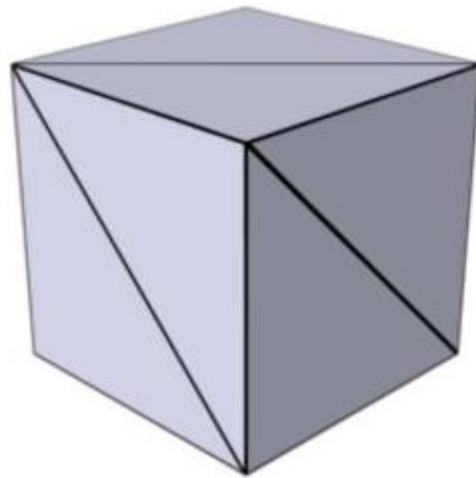
Not only triangles
2-manifold



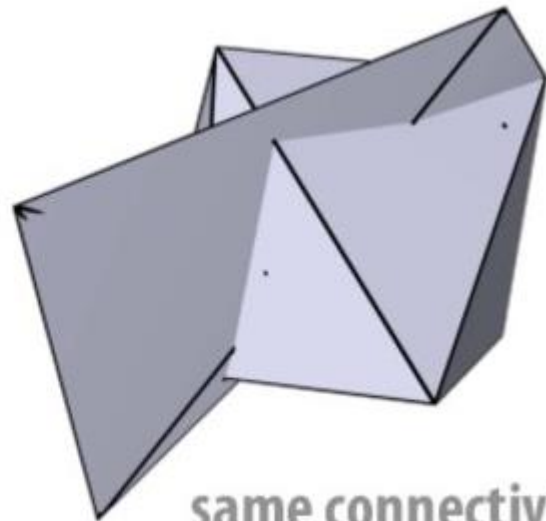
Non manifold

Connectivity vs Geometry

- Manifold conditions say nothing about vertex positions! Just connectivity
- Can have perfect good connectivity, yet awful geometry



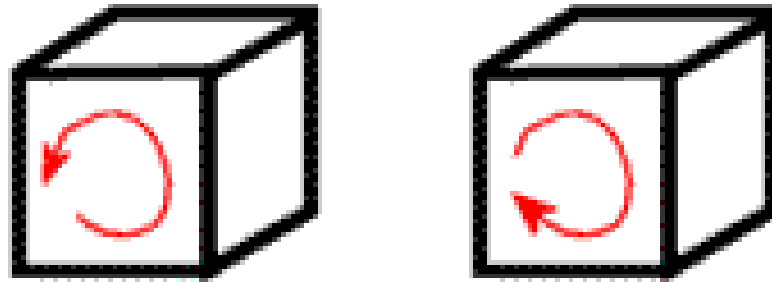
cube (manifold)



same connectivity,
random vertex positions

Orientation of Faces

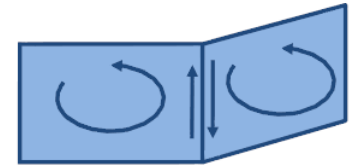
- Each **face** can be assigned an **orientation** by defining an ordering on its vertices
- Orientation can be clockwise or counter-clockwise.



- The orientation determines the sign of the normal direction.
 - Convention: **counterclockwise** is the “**front**” side.

Orientation of Faces

- A **mesh is orientable** if the incident faces of each edge are oriented consistently
 - The two edge directions of the shared edge (induced by the face orientations) are opposing
- Every **non-orientable closed** mesh embedded in \mathbb{R}^3 intersects itself
- The surface of a polyhedron is always orientable.



Möbius strip



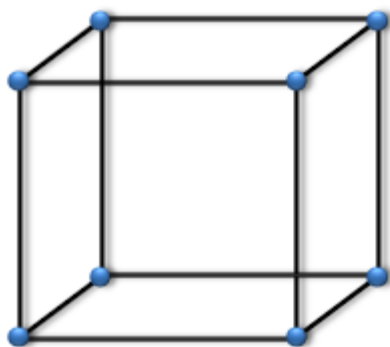
Klein bottle



Mesh Statistics

- **Euler formula** helps us derive mesh statistics.
- If a mesh M is homeomorphic to a sphere, then

$$V - E + F = 2$$

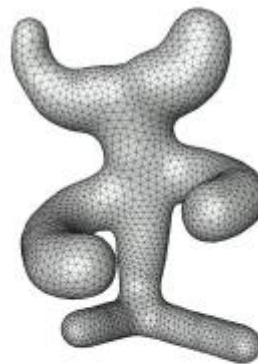


$$V = 8$$

$$E = 12$$

$$F = 6$$

$$\chi = 8 + 6 - 12 = 2$$



$$V = 3890$$

$$E = 11664$$

$$F = 7776$$

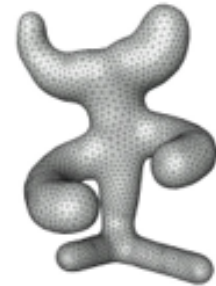
$$\chi = 2$$

Mesh Statistics

In large meshes, which of V , E , F is largest?

Based on Euler's formula, we can derive:

- $F \sim 2V$
- $E \sim 3V$
- Average vertex **valence** (degree) is 6 for large triangular meshes



$V = 3890$
 $E = 11664$
 $F = 7776$

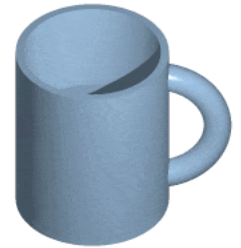
Proofs:

- Each face has 3 edges, but each edge counted twice, thus $2E = 3F$
- $V - E + F = 2 \rightarrow V - E + 2E/3 = 2_{\text{negligible}} \rightarrow E \sim 3V$
- $V - E + F = 2 \rightarrow V - 3F/2 + F = 2_{\text{negligible}} \rightarrow F \sim 2V$
- Average valence $= \sum_v \deg(v)/V = 2E / V \sim 6V/V = 6$

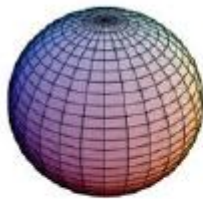
More on Euler's Formula

Generalized Euler Formula:

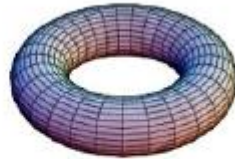
$$V - E + F = 2(1-g)$$



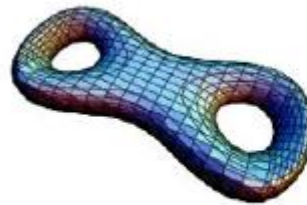
where g is the **genus** of the surface (i.e., # handles of the object)



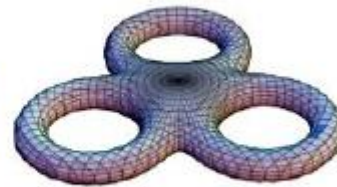
Genus 0



Genus 1



Genus 2



Genus 3

Mesh Data Structure

Mesh Data Structure

- Polygon mesh:
 - set of *polygons* embedded in 2D or 3D.
 - set of *vertices, edges, and faces* embedded in 2D or 3D.
- Let's handle these vertices, edges, faces in a structured way, hence the mesh data structure.

Mesh Data Structure

- How to store **geometry** & **connectivity** of a mesh.



3D vertex coordinates






























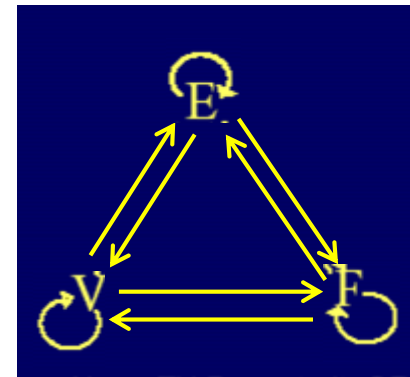
Vertex adjacency

- Also store attributes: normal, color, texture coords, etc.
- Support efficient algorithms to get local neighborhood information:
e.g.,
 - What are the vertices of face 77?
 - Is vertex 7 adjacent to vertex 17?
 - Which edges are incident to vertex 27?
 - Which faces are incident to vertex 27?

Neighborhood Relations

- All neighborhood relationships:

1. vertex	all neighboring vertices	VV			
2. vertex	all neighboring edges	VE			
3. vertex	all neighboring faces	VF			
4. edge	all neighboring vertices	EV			
5. edge	all neighboring edges	EE			
6. edge	all neighboring faces	EF			
7. face	all neighboring vertices	FV			
8. face	all neighboring edges	FE			
9. face	all neighboring faces	FF			



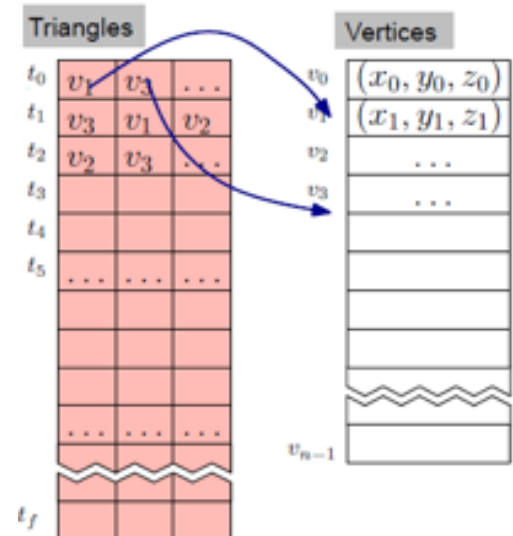
Polygon Soup

- **Face-Set** data structure

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}

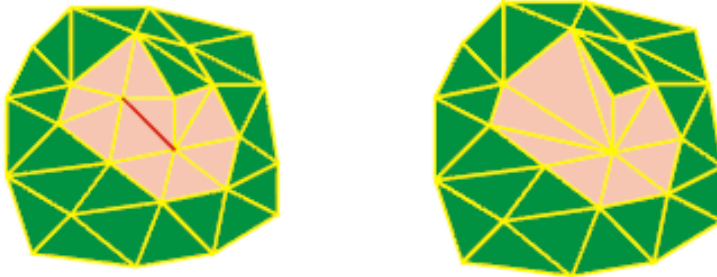
vertices and associated data are replicated.

- **Indexed face-set** data structure (obj, off, ply formats).
 - Less space requirement than face-set

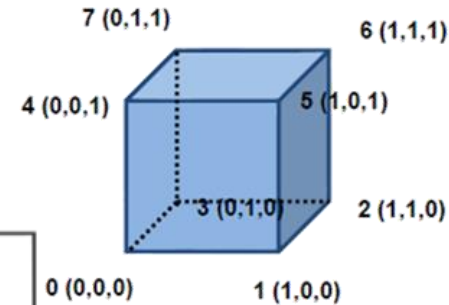


Indexed face set problems

- Information about neighbors is not explicit
 - Finding neighboring vertices/edges/faces etc.
 - $O(V)$** time!
 - Local mesh modifications cost **$O(V)$**



- Hard to do anything beyond drawing



Vertex list
(Coordinate3)

0	0.0	0.0	0.0
1	1.0	0.0	0.0
2	1.0	1.0	0.0
3	0.0	1.0	0.0
4	0.0	0.0	1.0
5	1.0	0.0	1.0
6	1.0	1.0	1.0
7	0.0	1.0	1.0

Face list
(IndexedFaceSet)

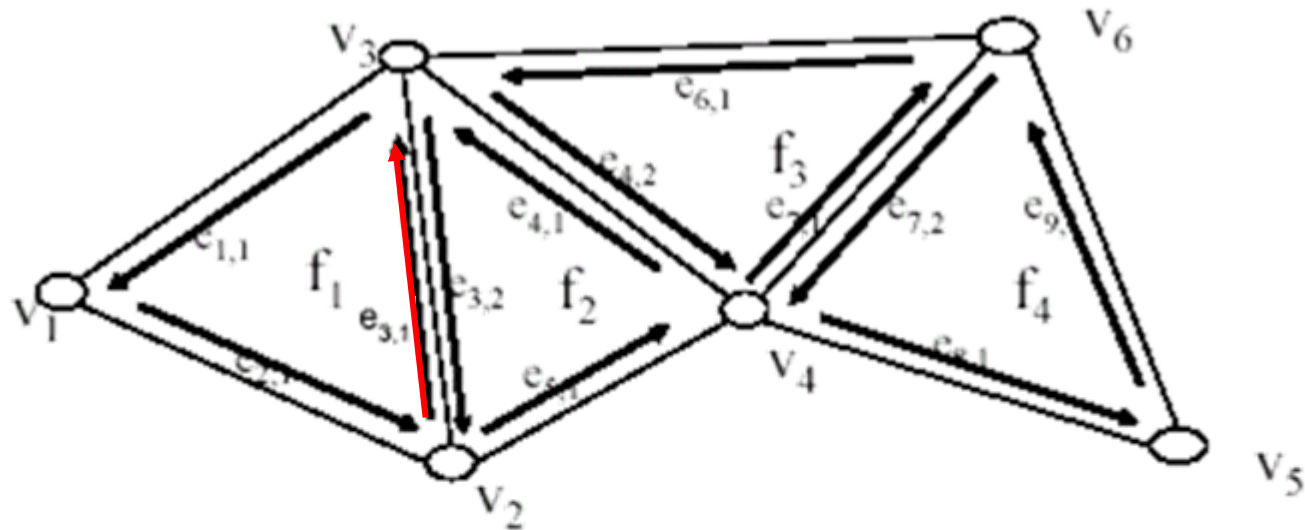
0	0	3	2	1
1	0	1	5	4
2	1	2	6	5
3	2	3	7	6
4	0	4	7	3
5	4	5	6	7

e.g., find all faces incident to vertex 2

Edge-Based Data Structures

- Face-based data structures
 - Different topological structures for triangles and quads
- Edge-based data structures
 - Explicit storage of edges and their connectivity information
 - For each vertex, stores a link to one edge
 - For each face, stores a link to one edge
 - Used in ready-to-use mesh processing libraries and software:
 - CGAL (lib)
 - OpenMesh (lib)
 - MeshLab (sw)

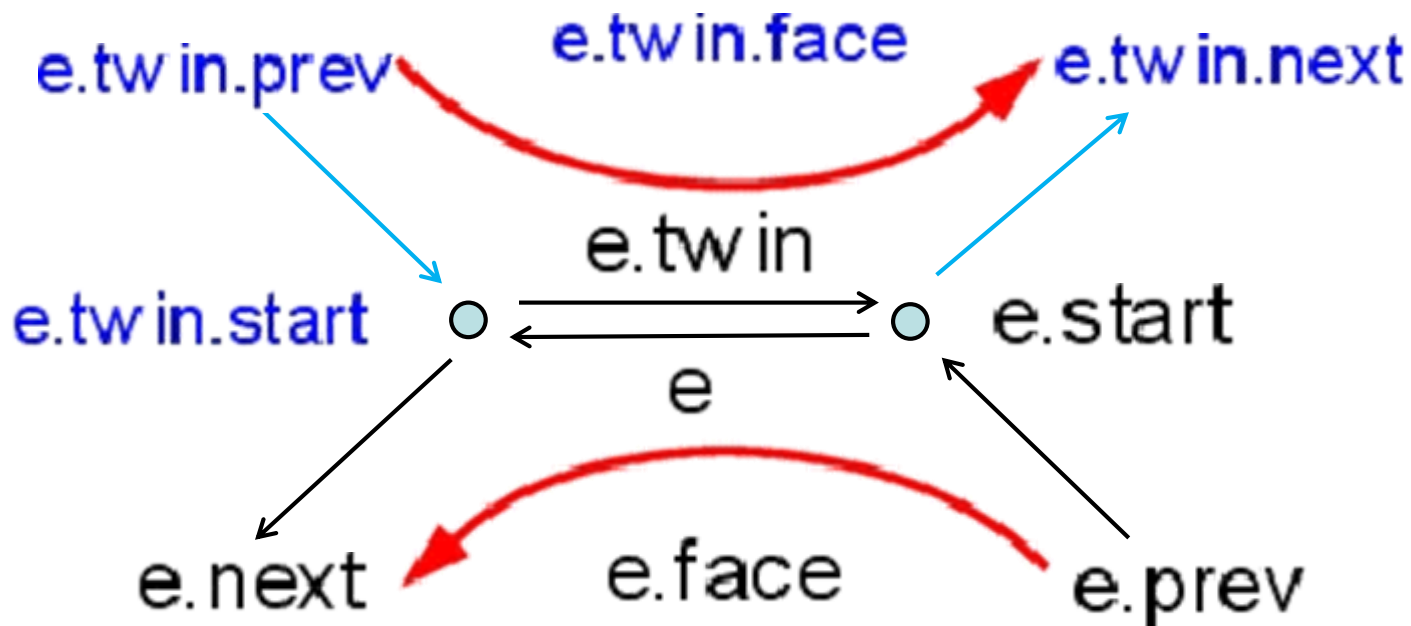
Half-edge data structure: example



Half-edge	origin	twin	IncidentFace	next	prev
$e_{3,1}$	v_2	$e_{3,2}$	f_1	$e_{1,1}$	$e_{2,1}$
$e_{3,2}$	v_3	$e_{3,1}$	f_2	$e_{5,1}$	$e_{4,1}$
$e_{4,1}$	v_4	$e_{4,2}$	f_2	$e_{3,2}$	$e_{5,1}$
$e_{4,2}$	v_3	$e_{4,1}$	f_3	$e_{7,1}$	$e_{6,1}$

Half-Edge Data Structure (aka doubly connected edge list)

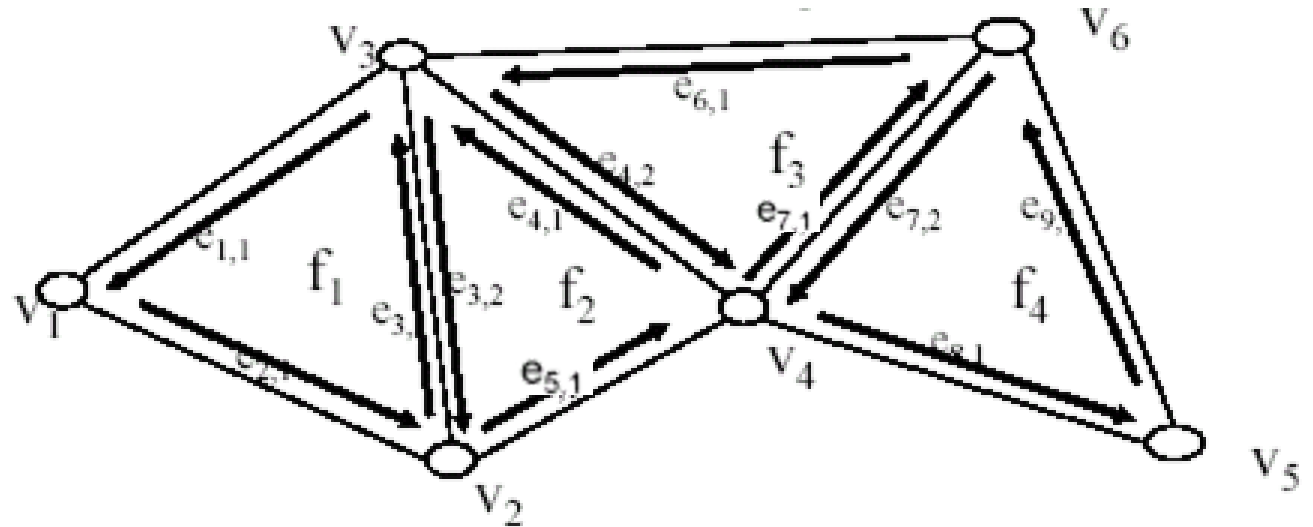
- Each half-edge has:
 - Pointer to **twin** half-edge
 - Pointer to **start** vertex
 - Pointer to **next** half-edge
 - Pointer to **previous** half-edge (optional)
 - Pointer to **incident face** (left side, assume counter-clockwise orientation)



Half-edge data structure

- In addition:
- **Vertex** has:
 - Pointer to (any) one emanating half-edge
- **Face** has:
 - Pointer to (any) one of its enclosing half-edge

Half-Edge Data Structure

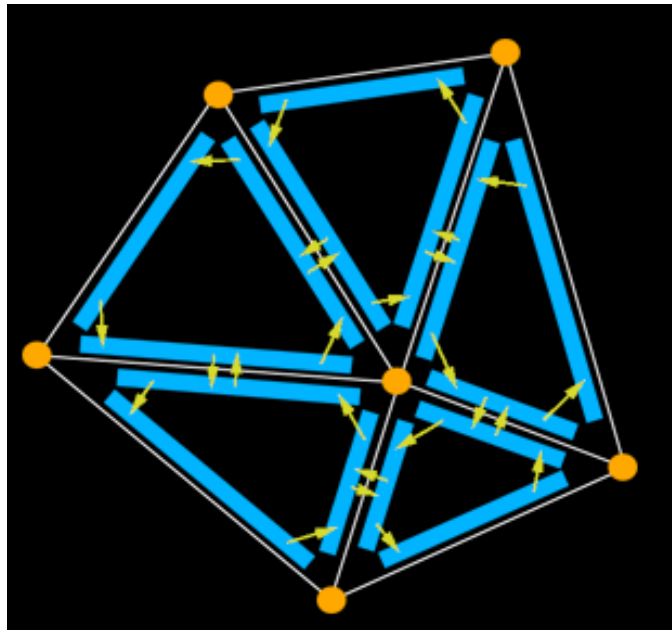


vertex	coordinate	IncidentEdge
v_1	(x_1, y_1, z_1)	$e_{2,1}$
v_2	(x_2, y_2, z_2)	$e_{5,1}$
v_3	(x_3, y_3, z_3)	$e_{1,1}$
v_4	(x_4, y_4, z_4)	$e_{7,1}$
v_5	(x_5, y_5, z_5)	$e_{9,1}$
v_6	(x_6, y_6, z_6)	$e_{7,2}$

face	edge
f_1	$e_{1,1}$
f_2	$e_{5,1}$
f_3	$e_{4,2}$
f_4	$e_{8,1}$

Half-Edge Data Structure

- Find the edges of a face
- Next pointers provide links around each face in counterclockwise



- prev pointers can be used to go around in clockwise
 - No need to store prev pointers otherwise

Half-Edge Data Structure

- Find all vertices adjacent to vertex v (i.e., 1-ring neighbors).

```
/* Assume closed mesh and  
counterclockwise order */
```

```
HalfEdge he = v.he;
```

```
HalfEdge curr = he;
```

```
output (curr.twin.start);
```

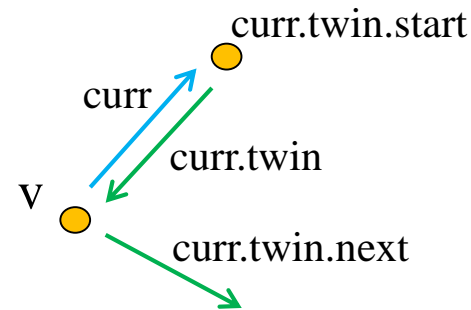
```
while (curr.twin.next != he) {
```

```
    curr = curr.twin.next;
```

```
    output (curr.twin.start);
```

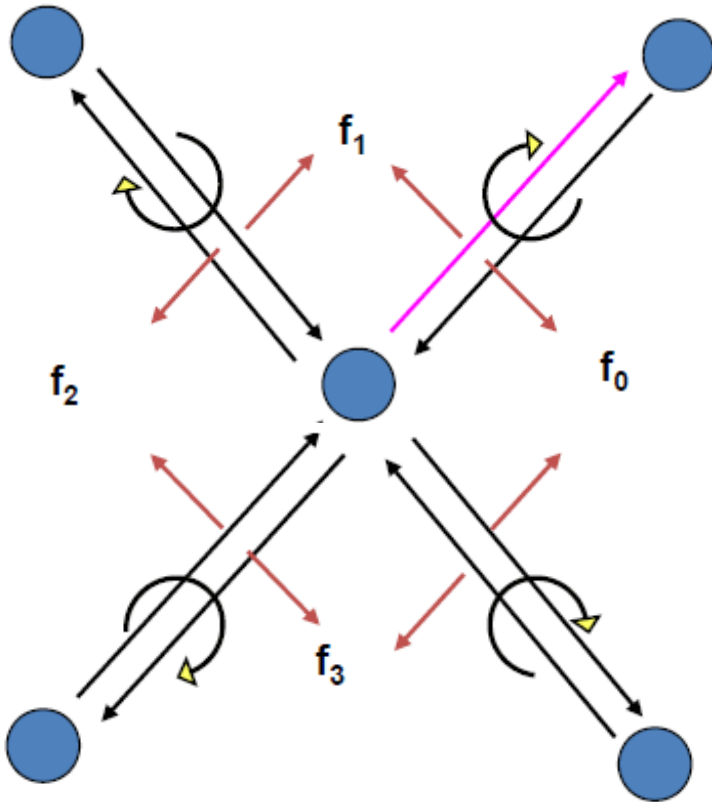
```
}
```

- Take $O(1)$ time



Half-Edge Data Structure

- All local neighboring queries take constant time ($O(1)$)
 - Query time independent of model size



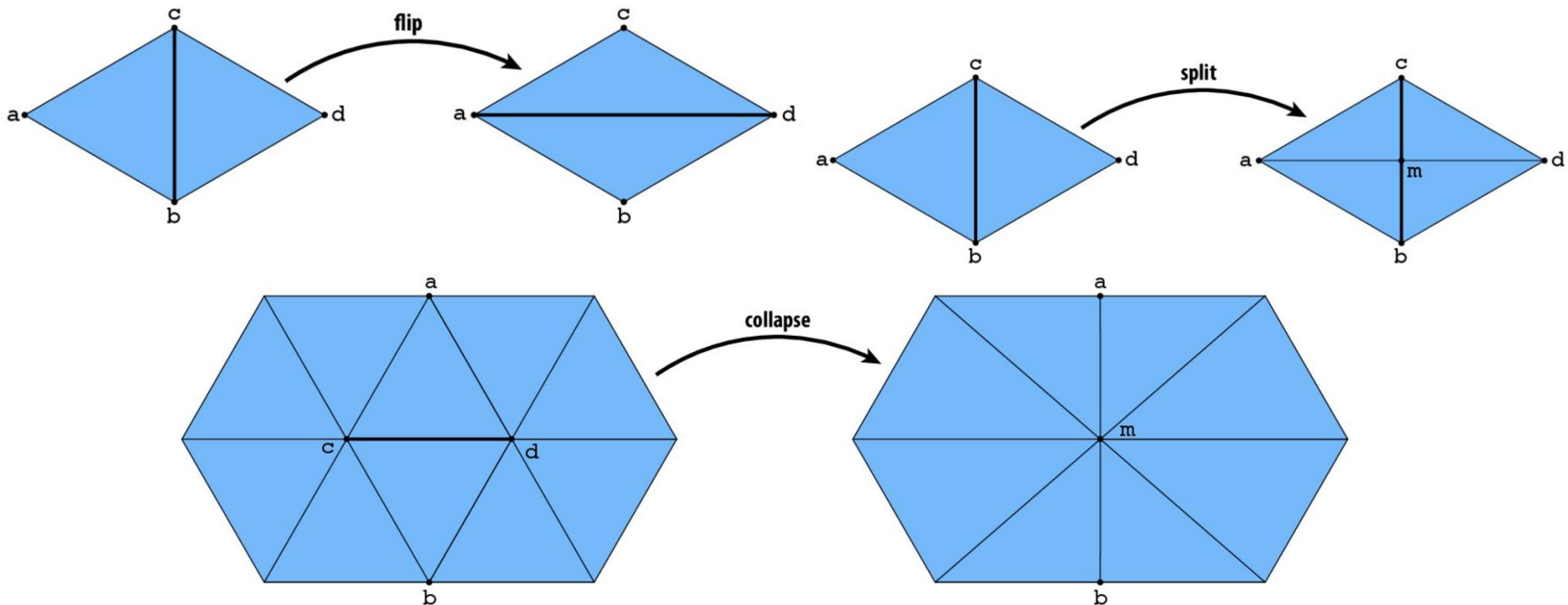
Half-edge	origin	twin	IncidentFace	next	prev
$e_{3,1}$	v_1	$e_{3,2}$	f_1	$e_{1,1}$	$e_{2,1}$
$e_{3,2}$	v_3	$e_{3,1}$	f_2	$e_{5,1}$	$e_{4,1}$
$e_{4,1}$	v_4	$e_{4,2}$	f_2	$e_{3,2}$	$e_{5,1}$
$e_{4,2}$	v_3	$e_{4,1}$	f_3	$e_{7,1}$	$e_{6,1}$

vertex	coordinate	IncidentEdge
v_1	(x_1, y_1, z_1)	$e_{2,1}$
v_2	(x_2, y_2, z_2)	$e_{5,1}$
v_3	(x_3, y_3, z_3)	$e_{1,1}$

face	edge
f_1	$e_{1,1}$
f_2	$e_{5,1}$

Atomic operations on halfedge meshes

- Key feature of linked list: insert/delete elements
- Same story with half-edge meshes
- Atomic **edge operations** on triangle meshes:



- How? Allocate/delete mesh elements; reassigning pointers
- Can we "reverse" split? Can we "reverse" edge collapse?
- Must be careful to preserve manifoldness

Alternatives to half-edge

- 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