

Introduction To Geometry

- Implicit & Explicit Geometry
- Manifold Geometry
- Local Geometric Operations

Some Motivation



*"I hate meshes.
I cannot believe how hard this is.
Geometry is hard."*

"why won't you subdivide"

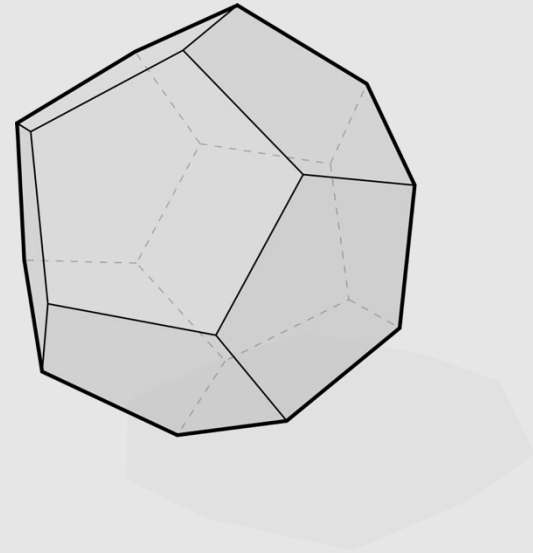


-- David Baraff
Senior Research Scientist
Pixar Animation Studios
(also a former CMU prof.)

What Is Geometry?

“Earth” “measure”
g e o m e t r y /jē'ämətrē/ *n.*

1. The study of shapes, sizes, patterns, and positions.
2. The study of spaces where some quantity (lengths, angles, etc.) can be *measured*.



Computer Graphics is just operating on a bunch of numbers.
If we can measure it, we can represent it on our computer!

How To Represent Geometry

[IMPLICIT]

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

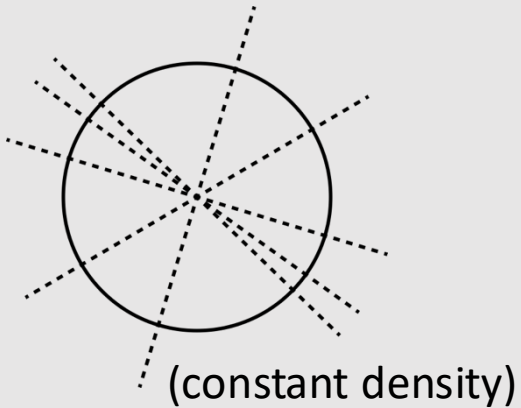
[LINGUISTIC]

“unit circle”

[EXPLICIT]

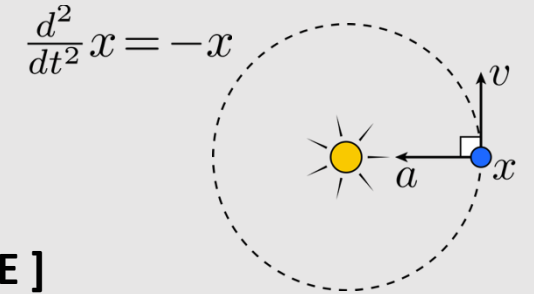
$$\underbrace{(\cos \theta)}_x, \underbrace{(\sin \theta)}_y$$

[TOMOGRAPHIC]



which is best?

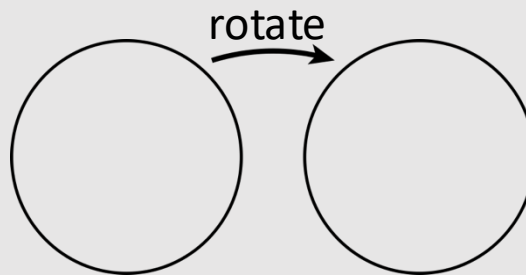
[DYNAMIC]



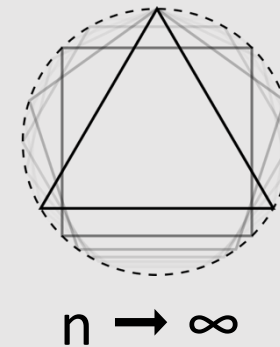
[CURVATURE]

$$\kappa = 1$$

[SYMMETRIC]



[DISCRETE]



How To Represent Humans



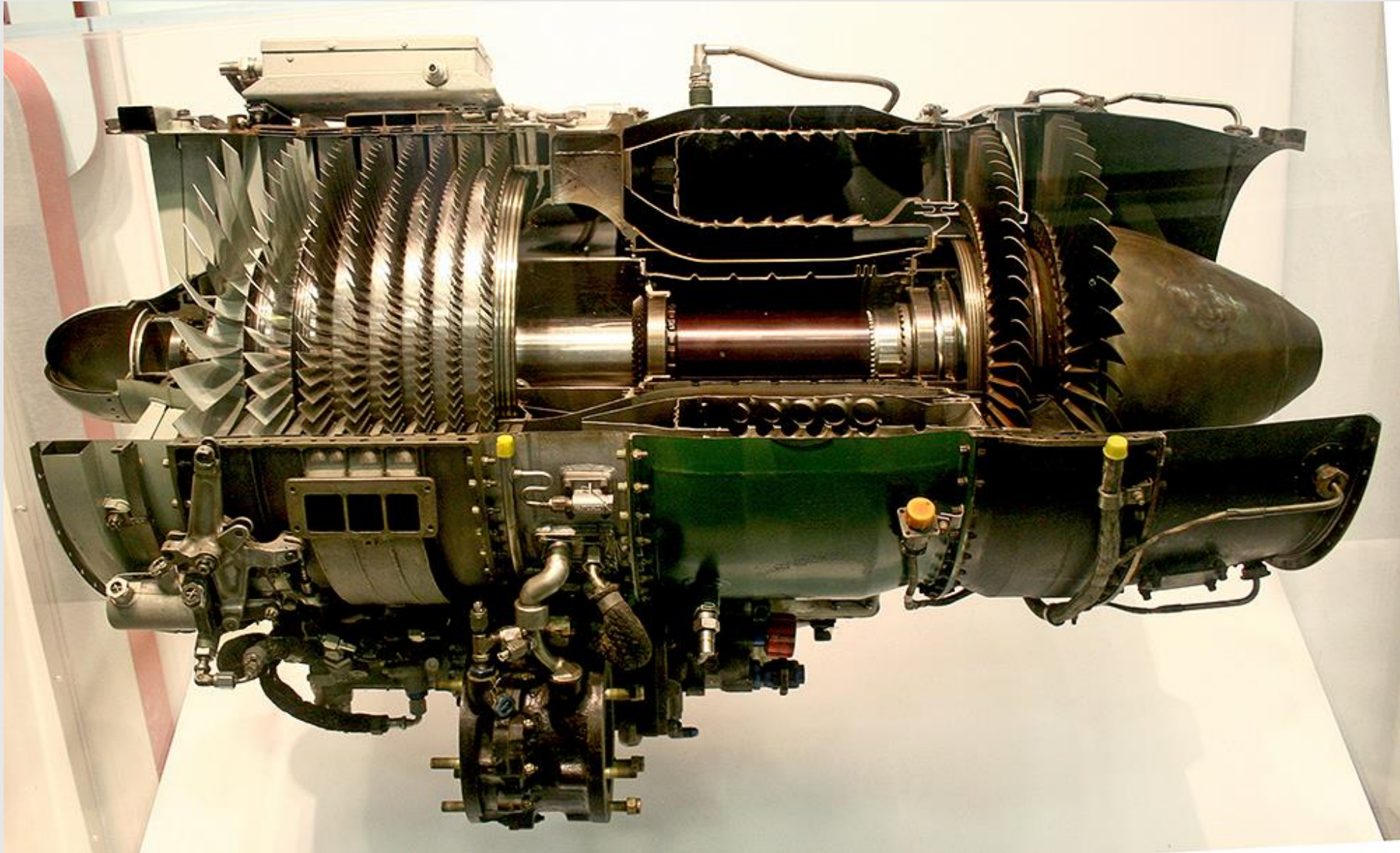
How To Represent Water



How To Represent Cloth



How To Represent Machines

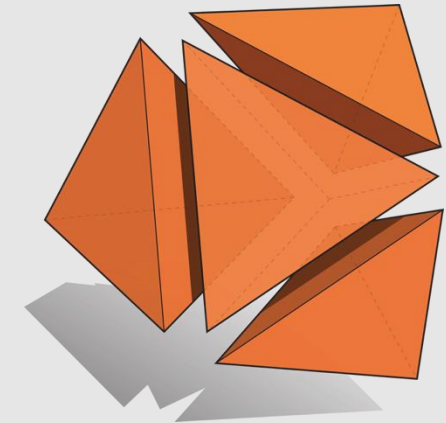
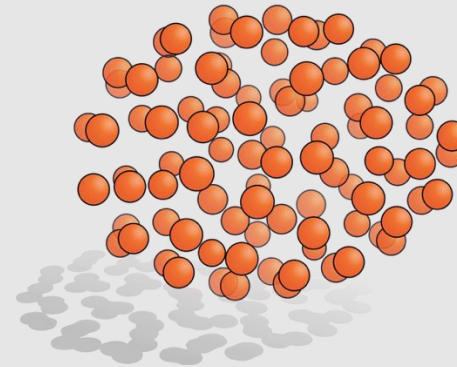
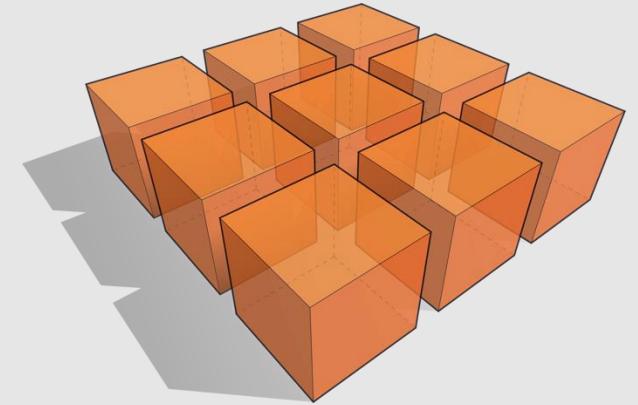
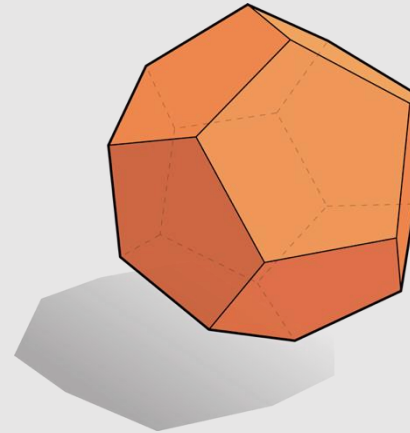
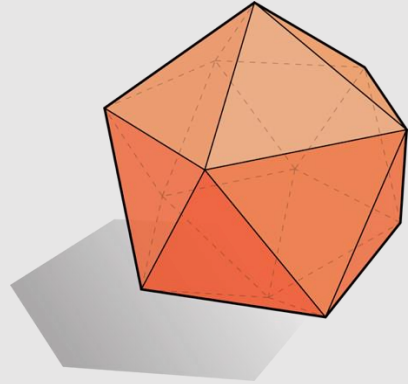


How To Represent This Thing



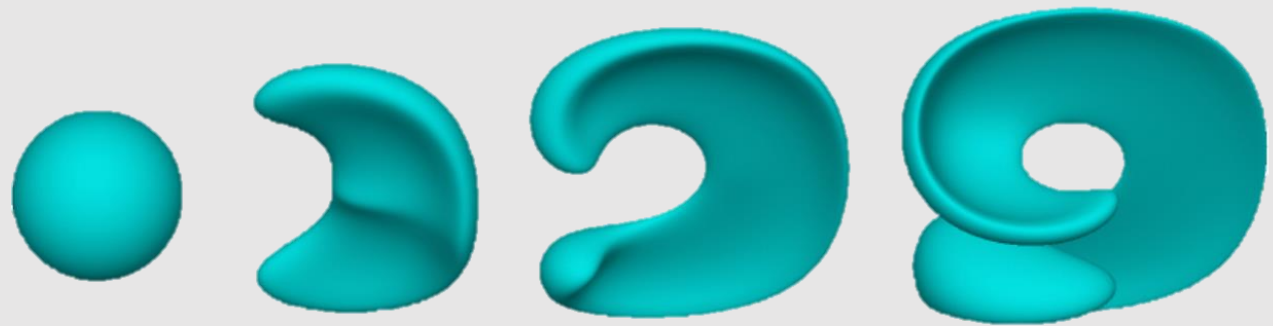
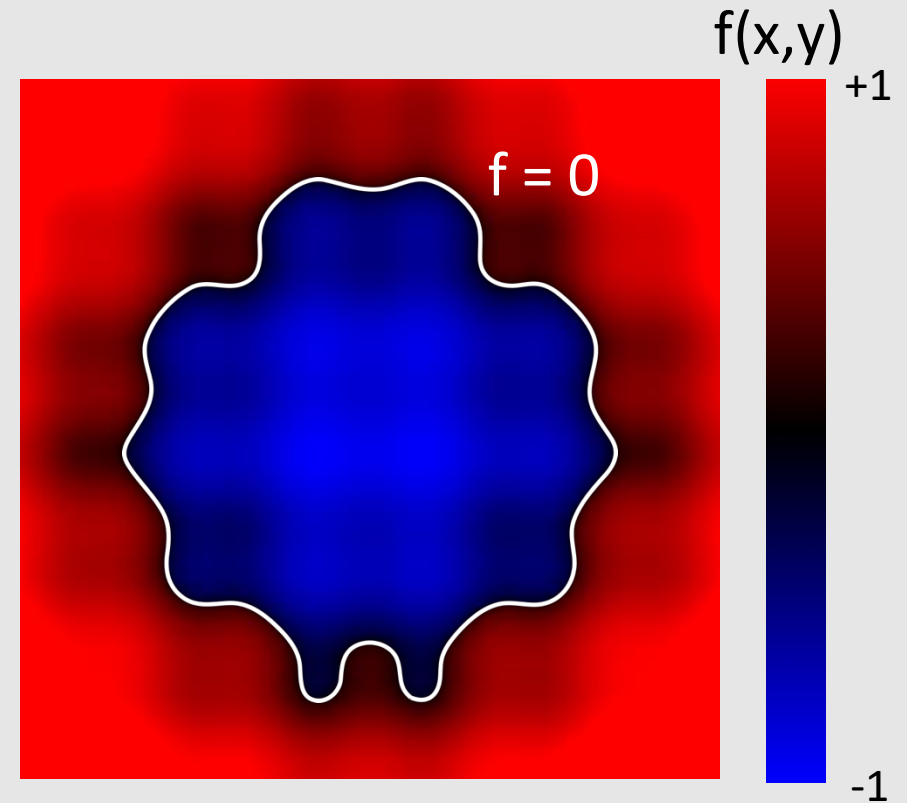
Many Ways To Encode Geometry

- Explicit:
 - point cloud
 - polygon meshes
 - subdivision surfaces
 - NURBS
- Implicit:
 - level set
 - constructive solid geometry
 - algebraic surface
 - L-systems
 - Fractals
- Not one best geometric representation!
 - Tradeoffs between:
 - Accuracy
 - Memory
 - Performance (searching/operating)



Implicit Geometry

- Points not known directly, but satisfy some relationship
 - **Ex:** unit sphere is all points such that $x^2+y^2+z^2=1$
- More generally, in the form $f(x,y,z) = 0$
- Finding example points is **hard**
 - Requires solving equation
- Checking if points are inside/outside is **easy**
 - Evaluate the function with a given point



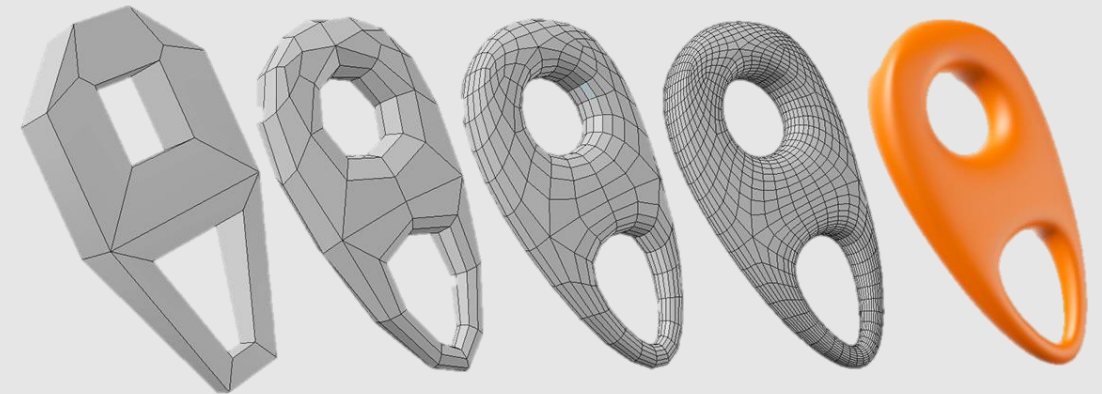
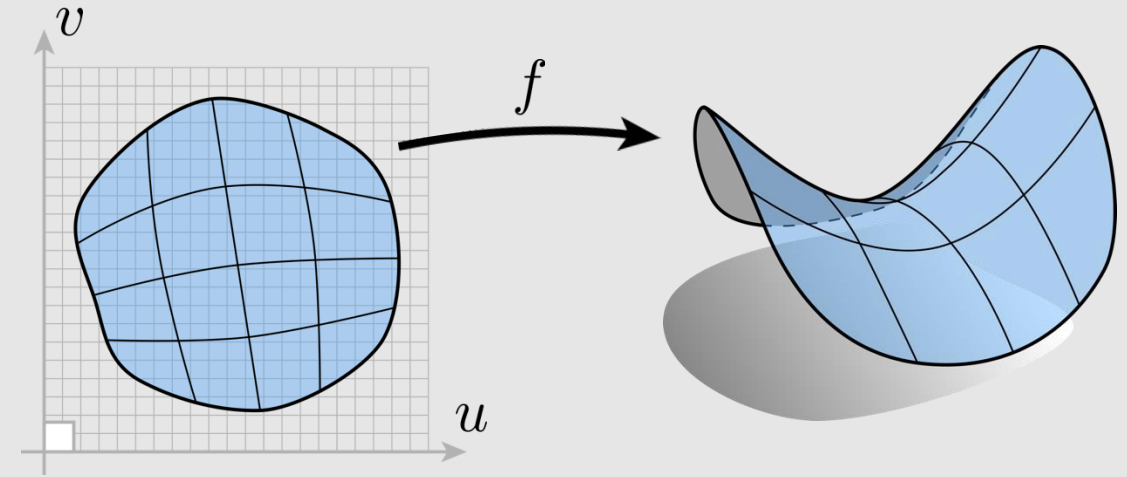
Explicit Geometry

- All points given directly

- More generally:

$$f : \mathbb{R}^2 \rightarrow \mathbb{R}^3; (u, v) \mapsto (x, y, z)$$

- Given any (u, v) , can find a point on the surface
- Can limit (u, v) to some range
 - **Ex:** triangle with barycentric coordinates
- Finding example points is **easy**
 - We are given them for free
- Checking if points are inside/outside is **hard**
 - We are given the output values and need to find input values that satisfy the geometry



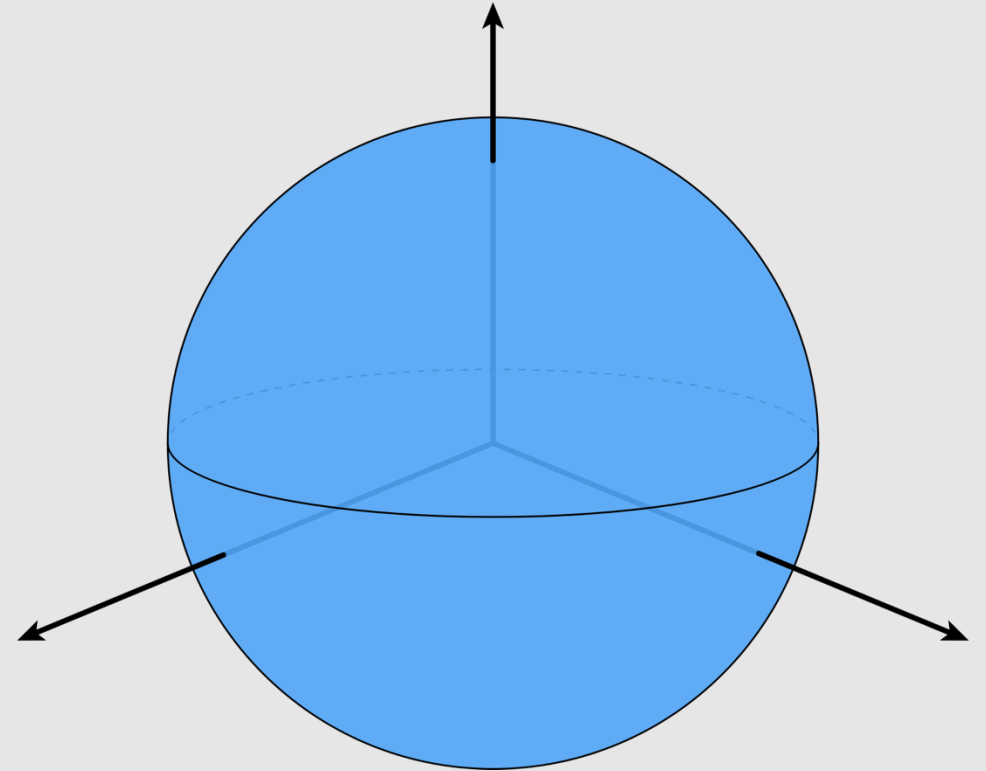
What does **easy** and **hard** mean?

Implicit Geometry [Hard]

- Given the unit sphere:

$$f(x, y, z) = x^2 + y^2 + z^2 = 1$$

- Find a point that exists on it.
- Answer:** (1,0,0)
 - Not so difficult, but how did you arrive at the answer?
 - We are given a constraint, and need to find parameters (x, y, z) that satisfy the constraint
 - Keep guessing and checking

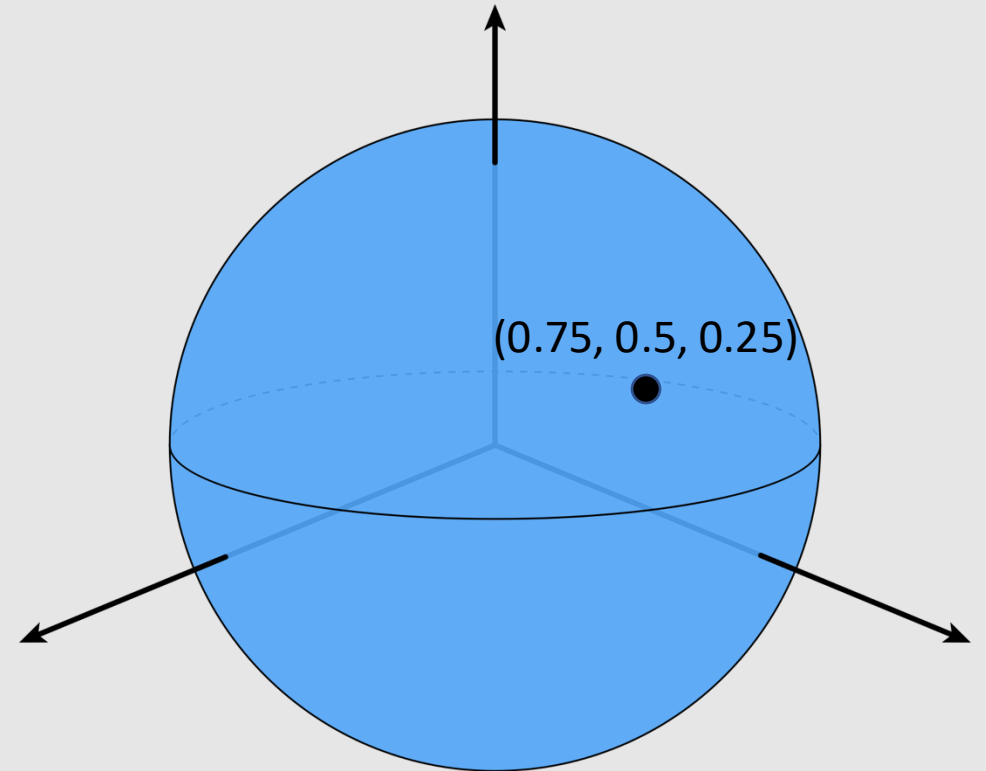


Implicit Geometry [Easy]

- Given the unit sphere:

$$f(x, y, z) = x^2 + y^2 + z^2 = 1$$

- Find if the point $(0.75, 0.5, 0.25)$ lives inside it.
- Answer:** Yes!
 - $f(0.75, 0.5, 0.25) = 0.75^2 + 0.5^2 + 0.25^2 = 0.875 < 1$
 - Easy to check! Just evaluate the sign of the function at the desired point

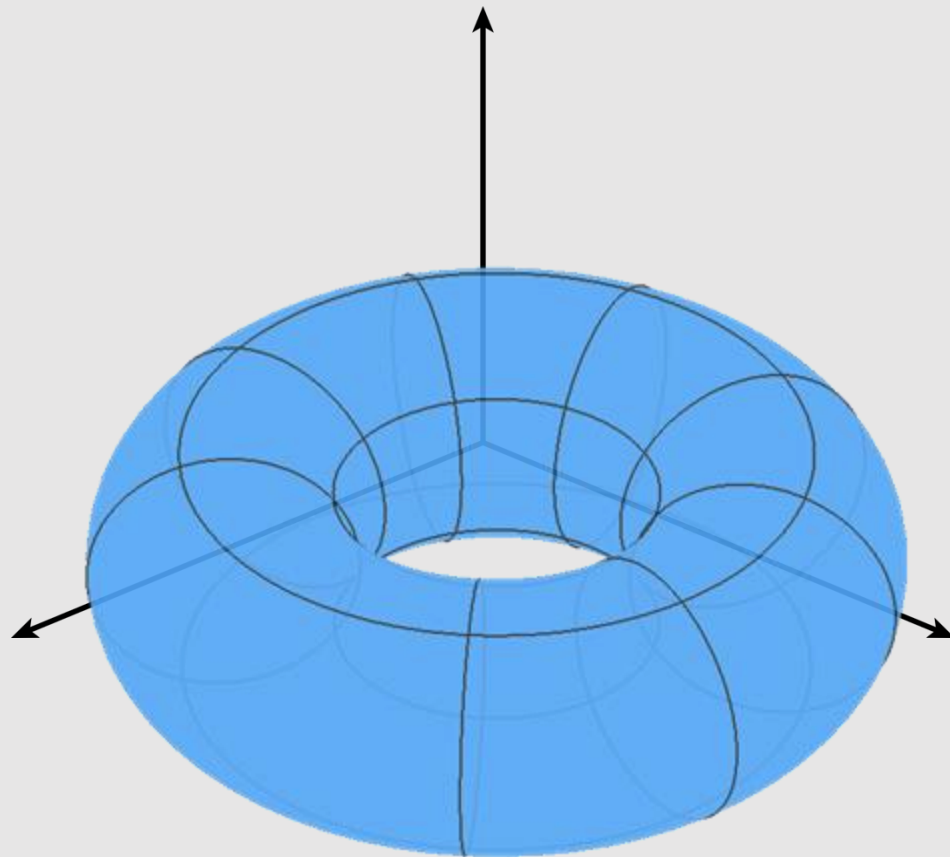


Explicit Geometry [Easy]

- Given the torus:

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

- Find a point that exists on it.
- Answer:** (3,0,0)
 - Just plug in any value of (u, v) !
 - We plugged in $(u, v) = (0,0)$

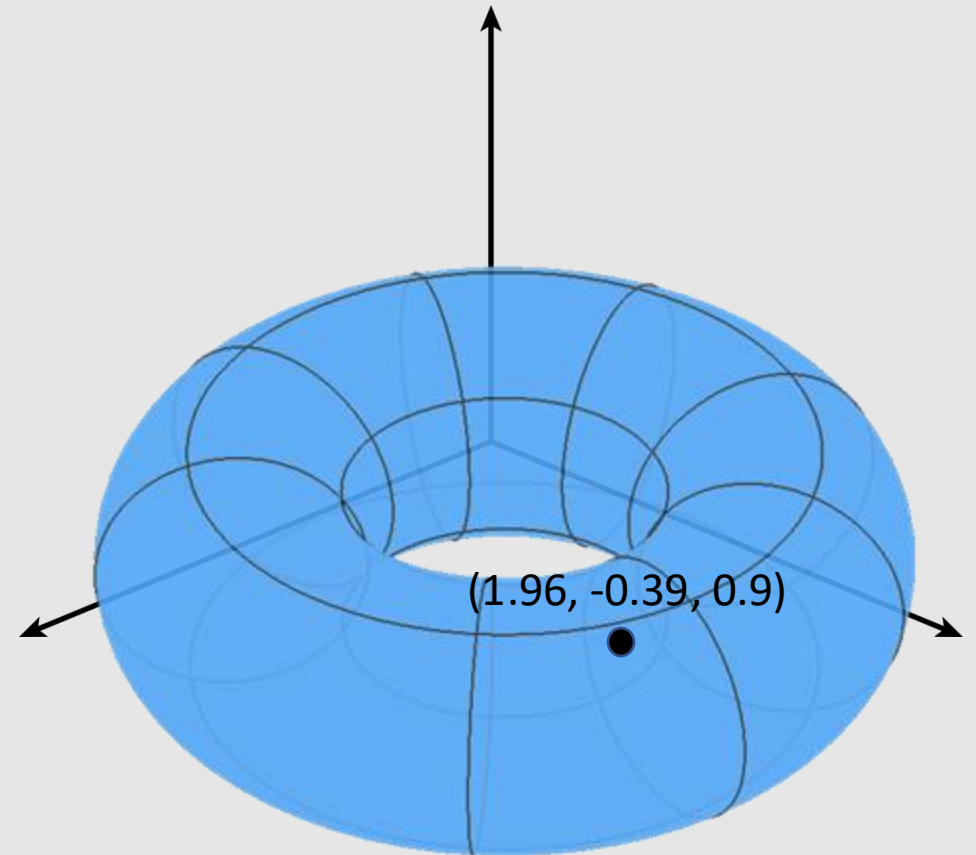


Explicit Geometry [Hard]

- Given the torus:

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

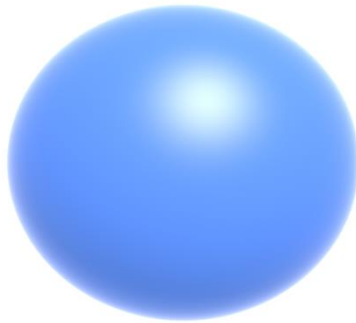
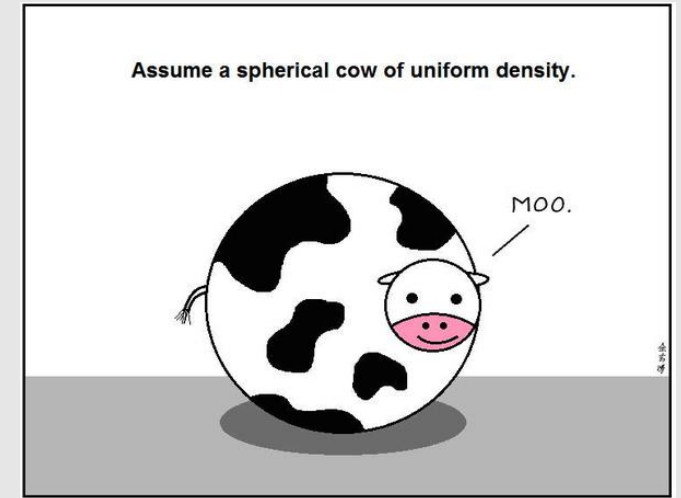
- Find if the point $(1.96, -0.39, 0.9)$ lives inside it.
- Answer:** no, I'm not computing that
 - We are given a constraint, and need to find parameters (u, v) that satisfy the constraint
 - Keep guessing and checking



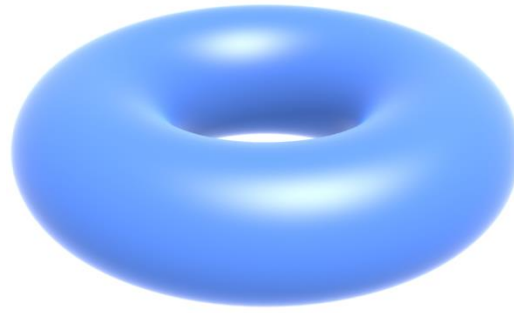
Let's look at some implicit examples...

Algebraic Surfaces [Implicit]

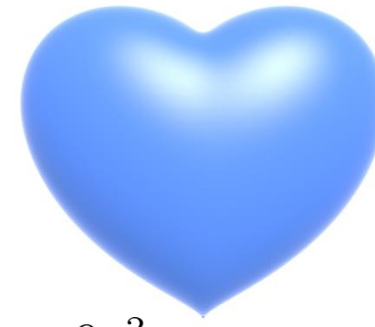
- A surface built with algebra
 - Generally thought of as a surface where points are some radius r away from another point/line/surface
- [+] Generates smooth/symmetric surfaces
- [-] Cannot generate impurities/deformations



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



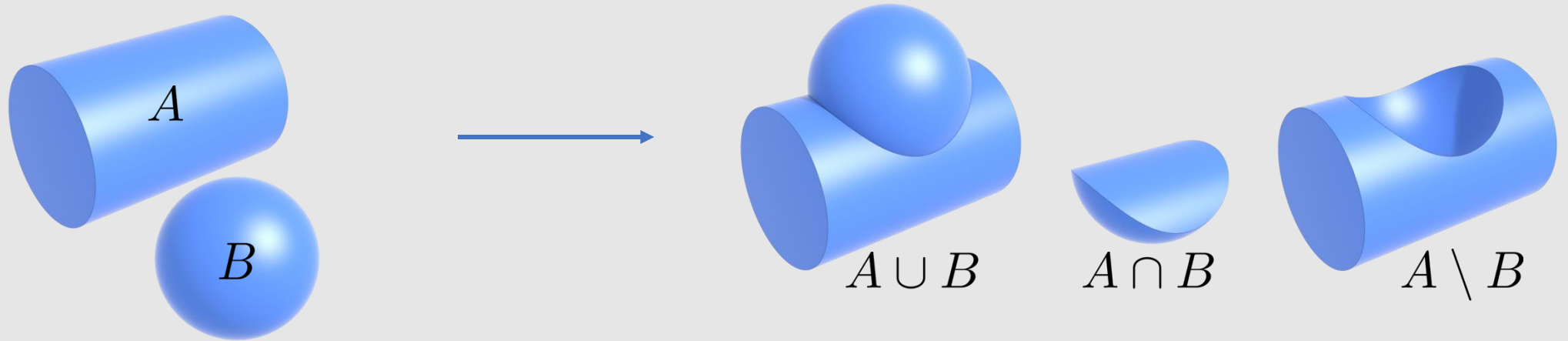
$$(R - \sqrt{x^2 + y^2})^2 + z^2 = r^2$$



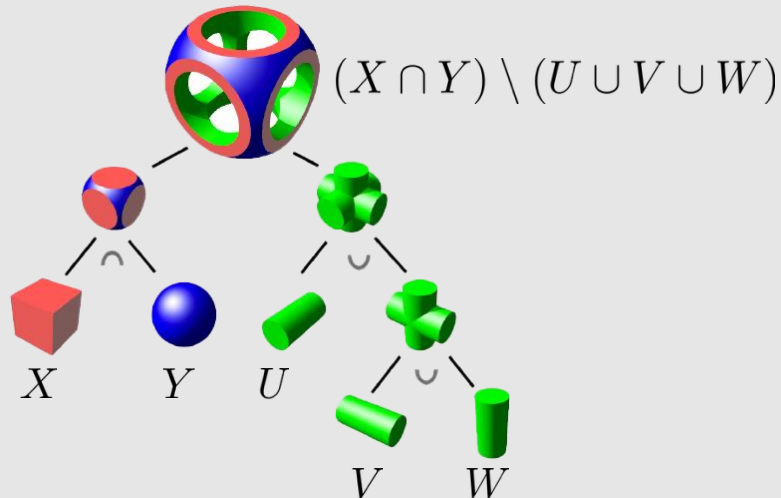
$$(x^2 + \frac{9y^2}{4} + z^2 - 1)^3 = x^2 z^3 + \frac{9y^2 z^3}{80}$$

Constructive Solid Geometry [Implicit]

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

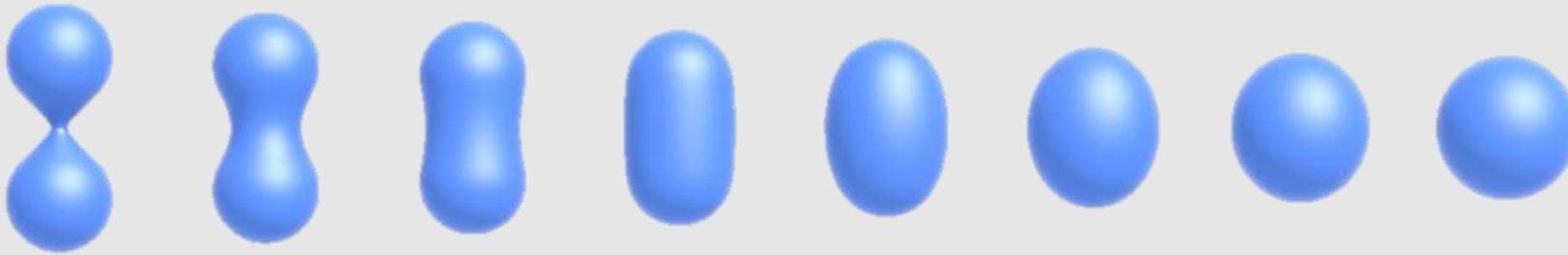


- Can be used to form complex shapes!

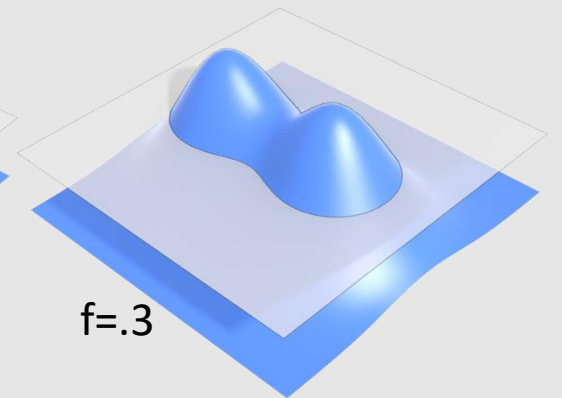
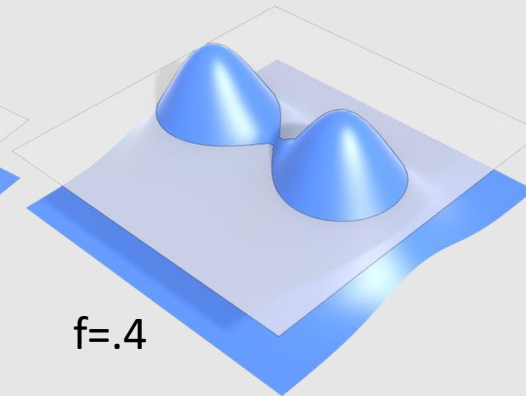
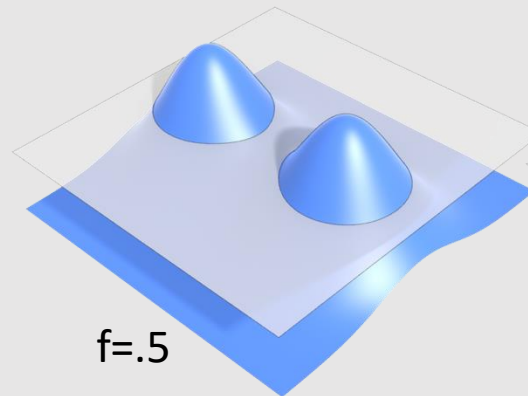


Blobby Surfaces [Implicit]

- Gradually blend surfaces together:

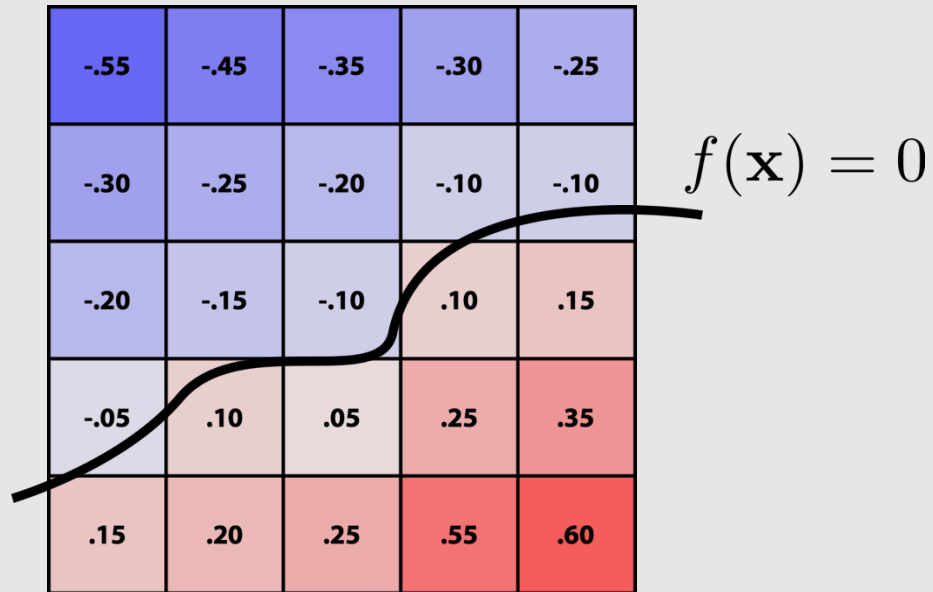


- Easier to understand in 2D:
 $\phi_p(x) := e^{-|x-p|^2}$ (Gaussian centered at p)
 $f := \phi_p + \phi_q$ (Sum of Gaussians centered at different points)

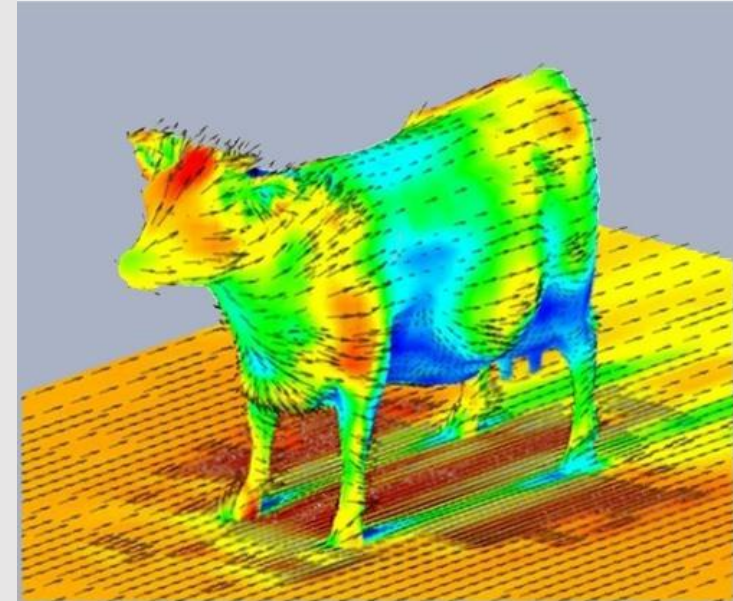


Level Set Methods [Implicit]

- Store a grid of values approximating function



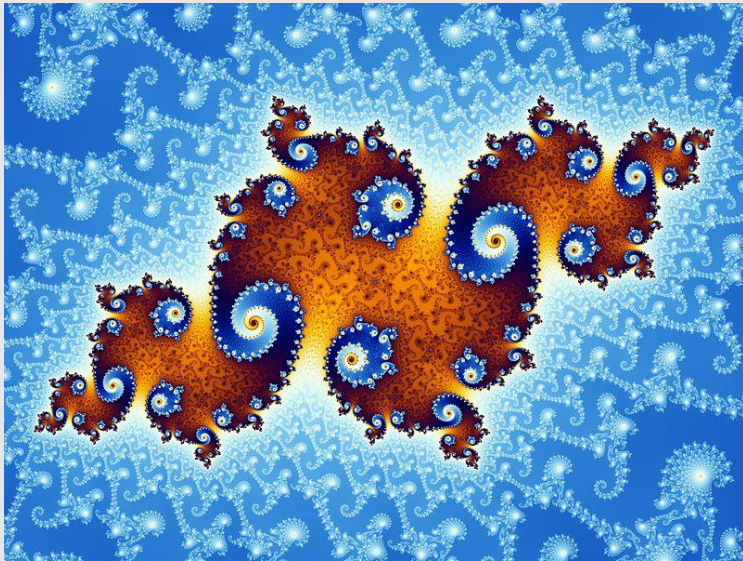
The aerodynamics of a cow:



- Surface is found where interpolated values equal zero
- [+] Provides much more explicit control over shape
- [-] Runs into aliasing problems

Fractals [Implicit]

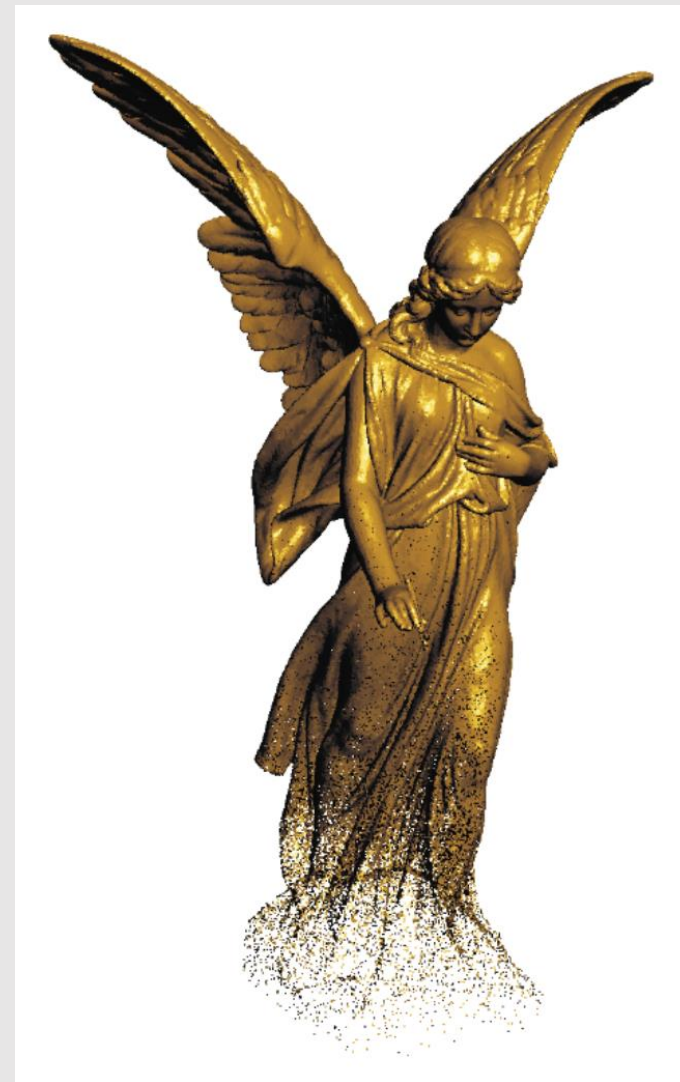
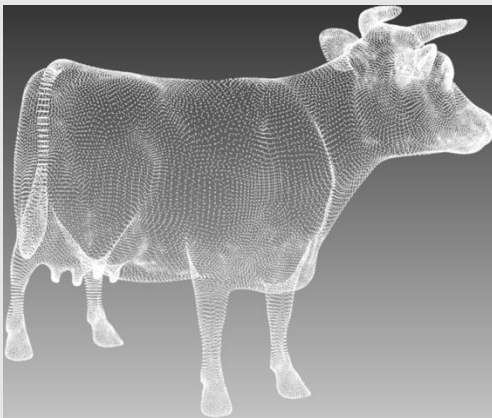
- No precise definition; exhibit self-similarity, detail at all scales
- [+] New “language” for describing natural phenomena
- [-] Hard to control shape!



Let's look at some explicit examples...

Point Cloud [Explicit]

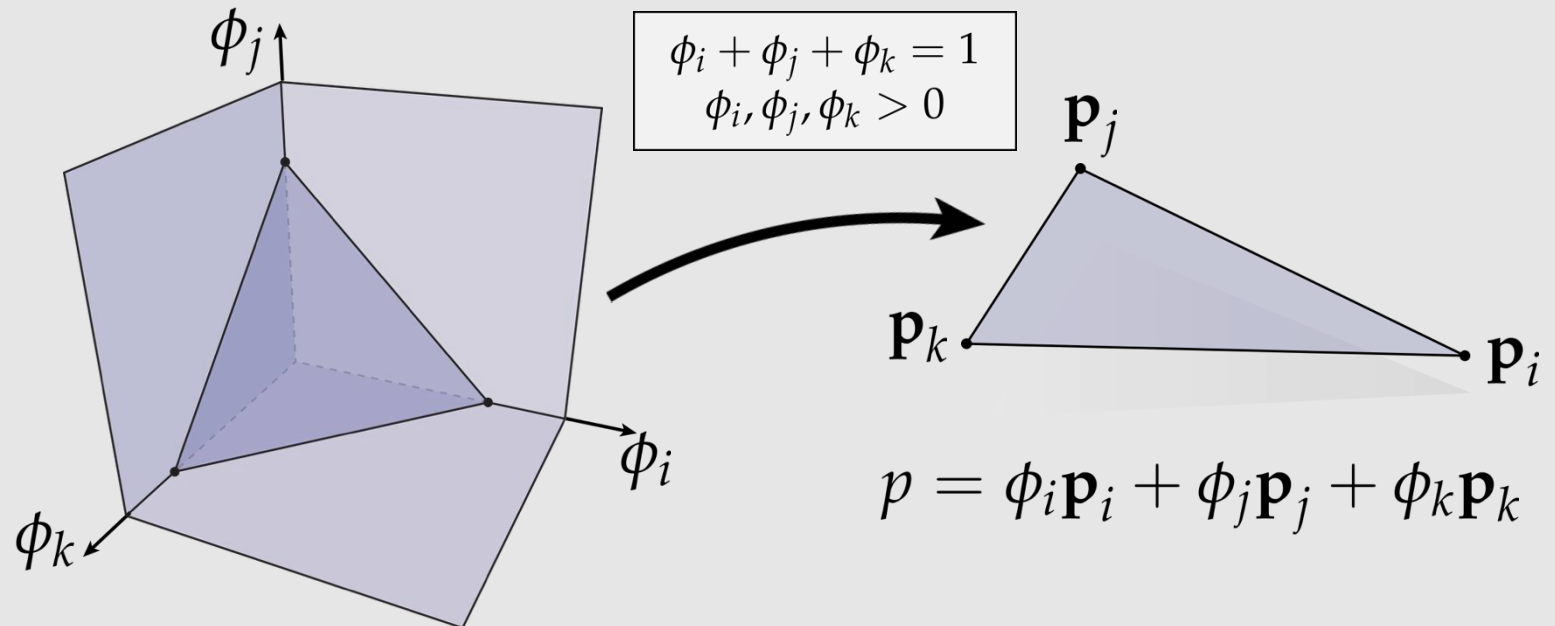
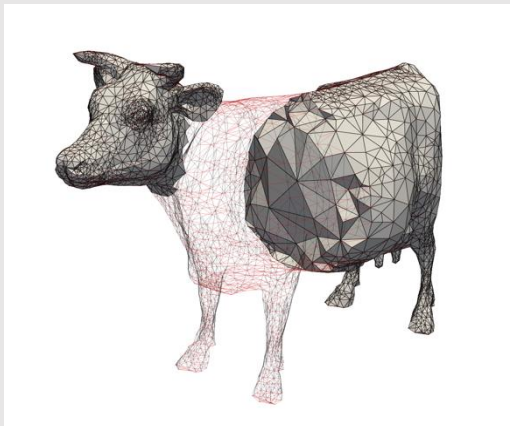
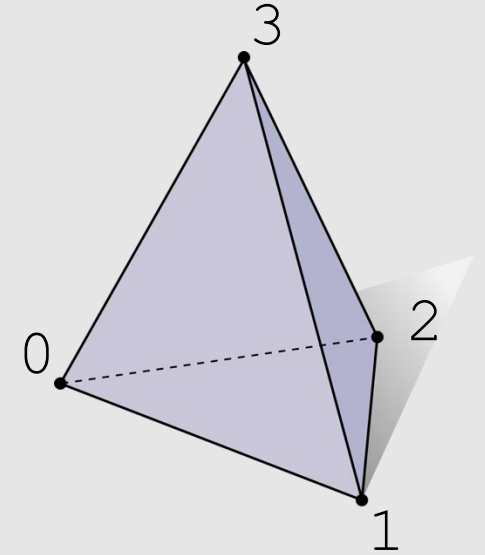
- A list of points (x, y, z)
 - Often augmented with normals
- [+] Easily represent any kind of geometry
- [+] Easy to draw dense cloud ($\gg 1$ point/pixel)
- [+] Easy for simulation
- [-] Large lookup time
- [-] Large memory overhead
 - Hard to interpolate undersampled regions
 - Expensive to do processing/simulation
 - Result is just as good as the scan



Triangle Mesh [Explicit]

- [+] Easy interpolation with good approximation
 - Barycentric interpolation
- [-] Large memory overhead
 - Store vertices as triples of coordinates (x,y,z)
 - Store triangles as triples of indices (i,j,k)
- Polygonal Mesh: shapes do not need to be triangles
 - Ex: quads

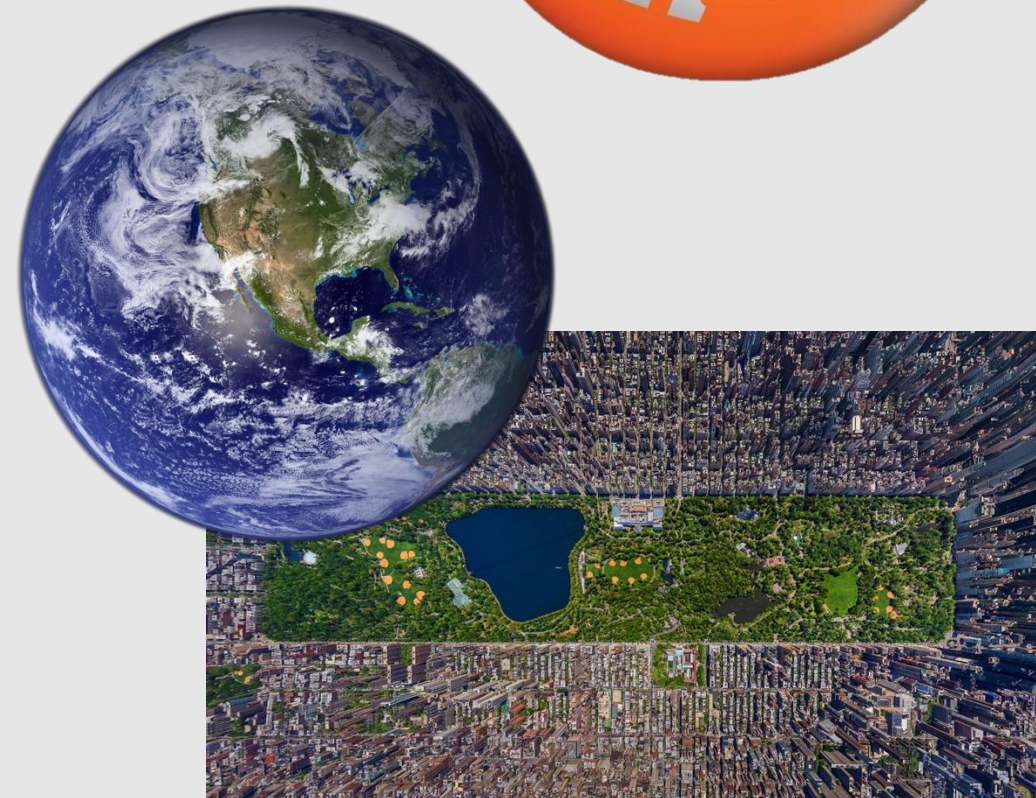
	[VERTICES]			[TRIANGLES]		
	x	y	z	i	j	k
0:	-1	-1	-1	0	2	1
1:	1	-1	1	0	3	2
2:	1	1	-1	3	0	1
3:	-1	1	1	3	1	2



- ~~Implicit & Explicit Geometry~~
- Manifold Geometry
- Local Geometric Operations

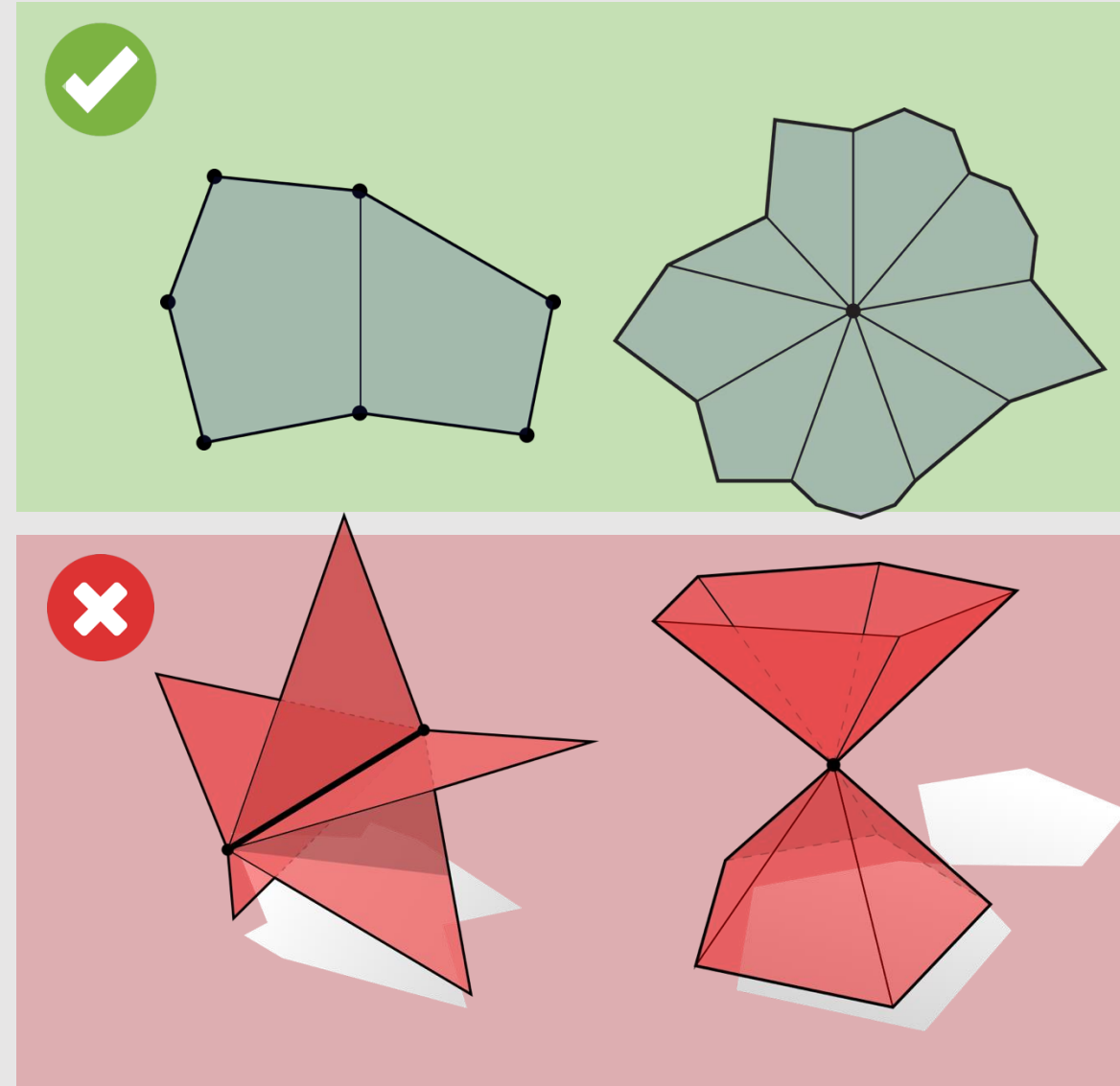
Manifold Assumption

- A mesh is **manifold** if and only if it can exist in real life
 - Important for simulation/3D printing
- Everything in real life has volume
 - Every manifold surface encases some volume
 - Think of manifold surfaces as 'shells' to an inner volume
 - **Example:** M&Ms
- Everything in real life, when zoomed in far enough, has a rectangular coordinate grid
 - Every manifold surface is planar when zoomed in far enough
 - **Example:** Planet Earth

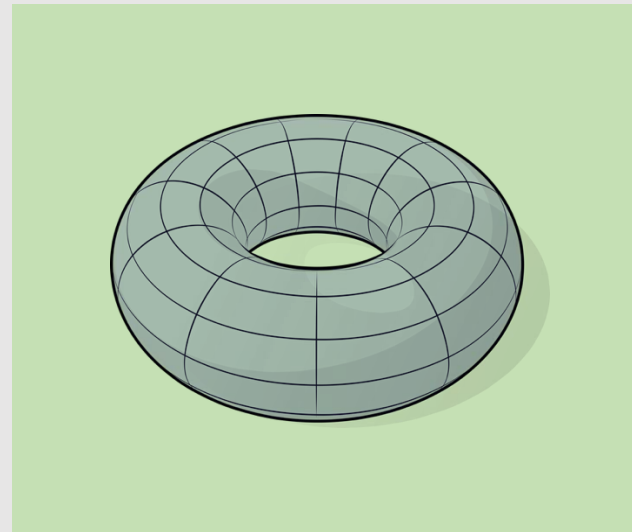
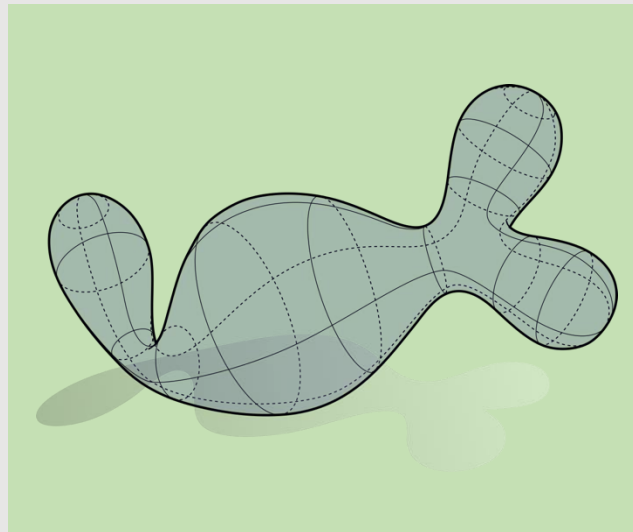
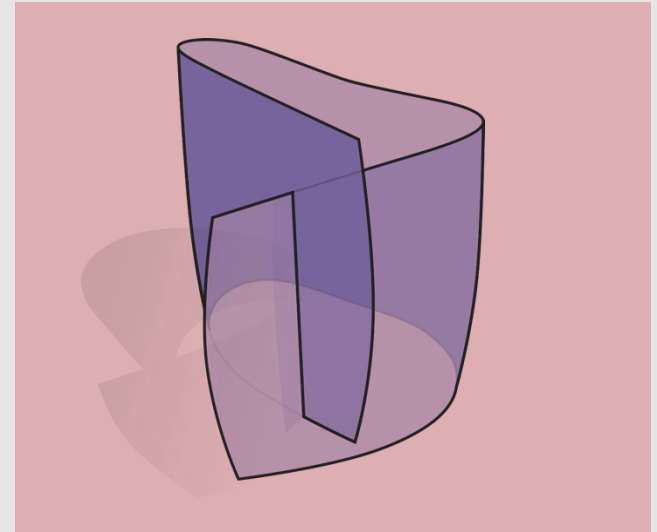
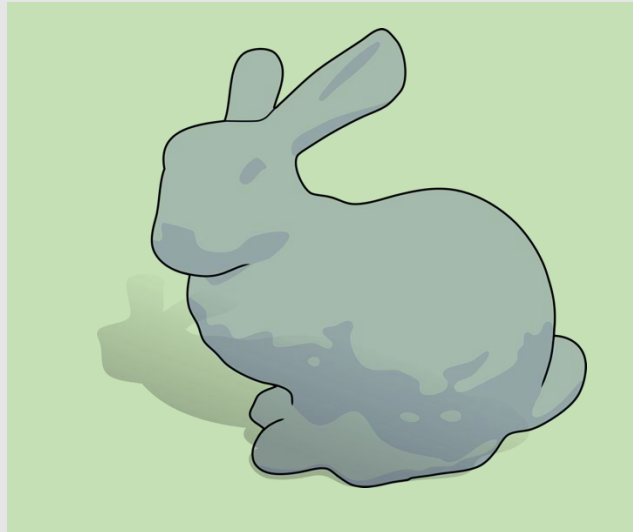
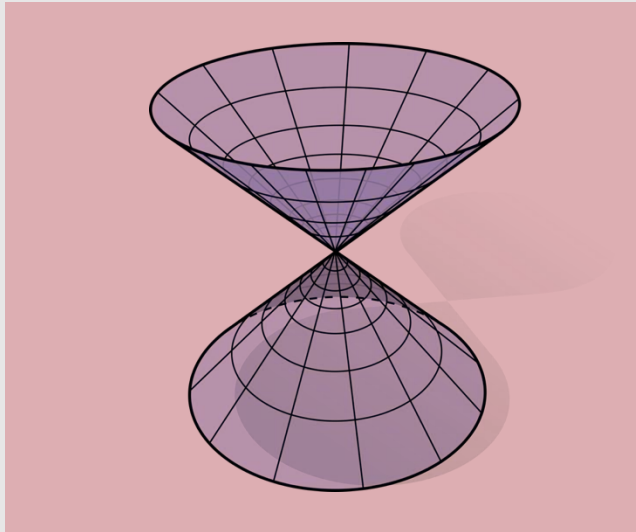


Manifold Properties

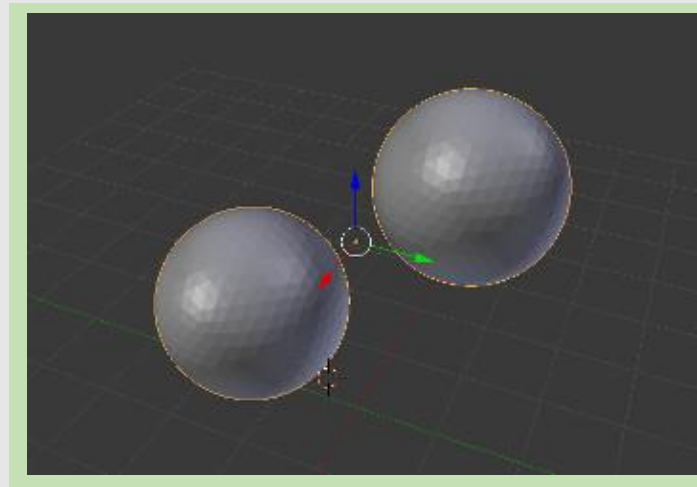
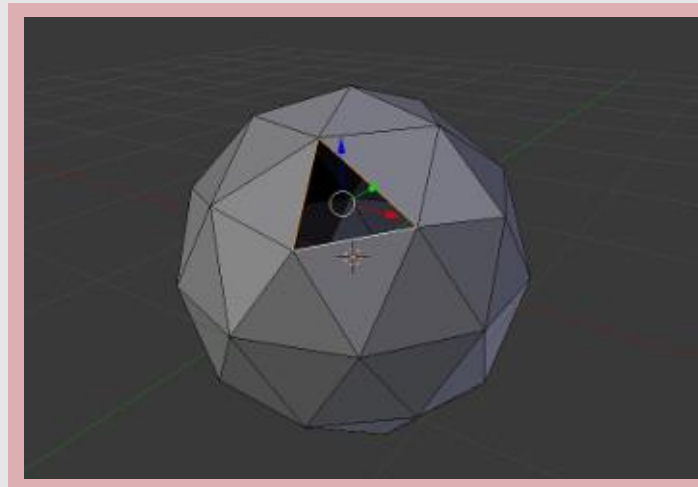
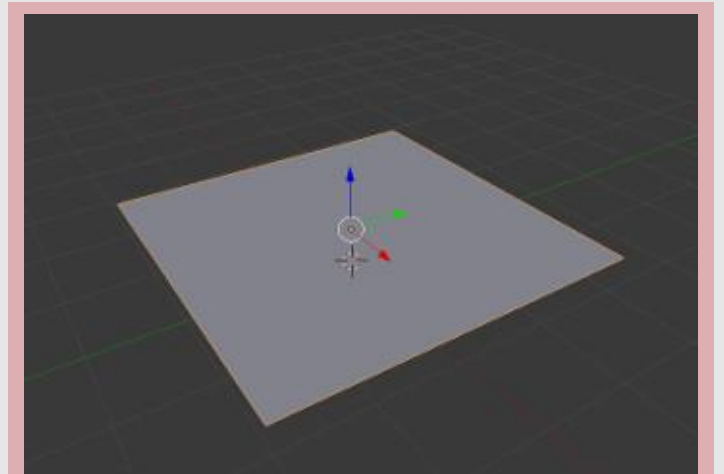
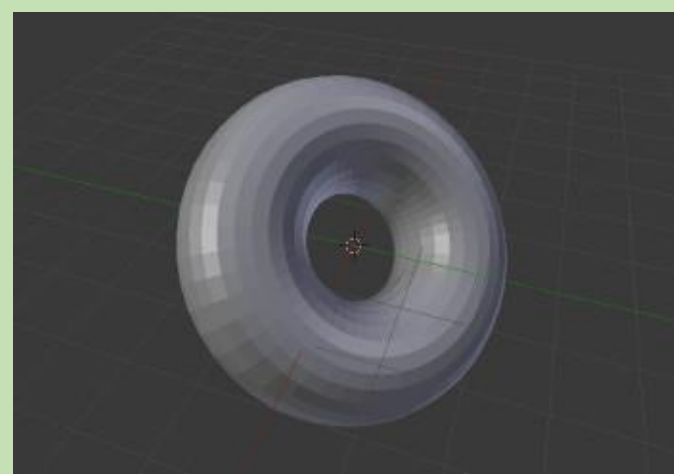
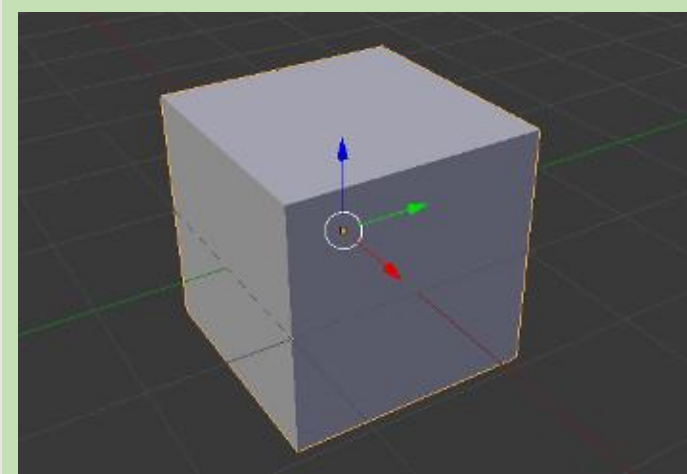
- For polygonal surfaces, check for “**fins**” and “**fans**”
- Every edge is contained in only two polygons (no “**fins**”)
 - Fins break planar zoom requirement
- The polygons containing each vertex make a single “**fan**”
 - Multiple fans break volume requirement



Manifold Check



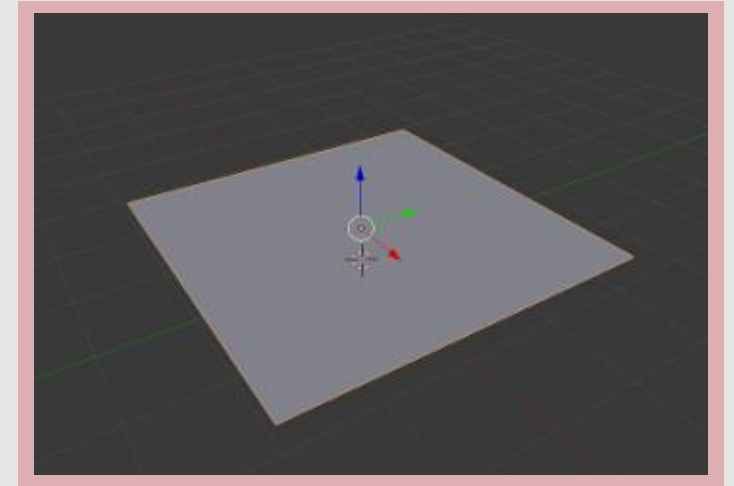
Manifold Check



******<https://github.com/rlguy/Blender-FLIP-Fluids/wiki/Manifold-Meshes>

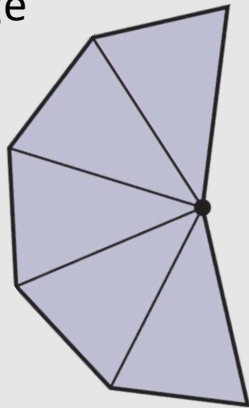
Planes Are Not Manifold

- Each edge only touches 1 polygon
 - Breaks the “fin” constraint
- More intuitively: **no notion of thickness!**
 - Can not be represented in real life
 - Paper (best approximation of plane) still has thickness
- **How to make manifold:** add a second polygon that overlaps with the first plane, connecting all the edges
- **How to make manifold:** add a new type of edge denoting it as a boundary
 - The “boundary” edge



Boundary Edges

- Objects in real life (Ex: pants) have boundaries
 - Boundary geometry loops around to create the inner seams of the pants
 - The volume enclosed by pants are not where your legs go, but the physical thickness of the pants
- Representing both the inside and outside of pants is expensive!
 - Use boundary edges
- A boundary edge has 1 polygon per edge
 - **This does not mean planes are manifold!** This just gives us a way to represent complex manifold geometry as simpler non-manifold geometry



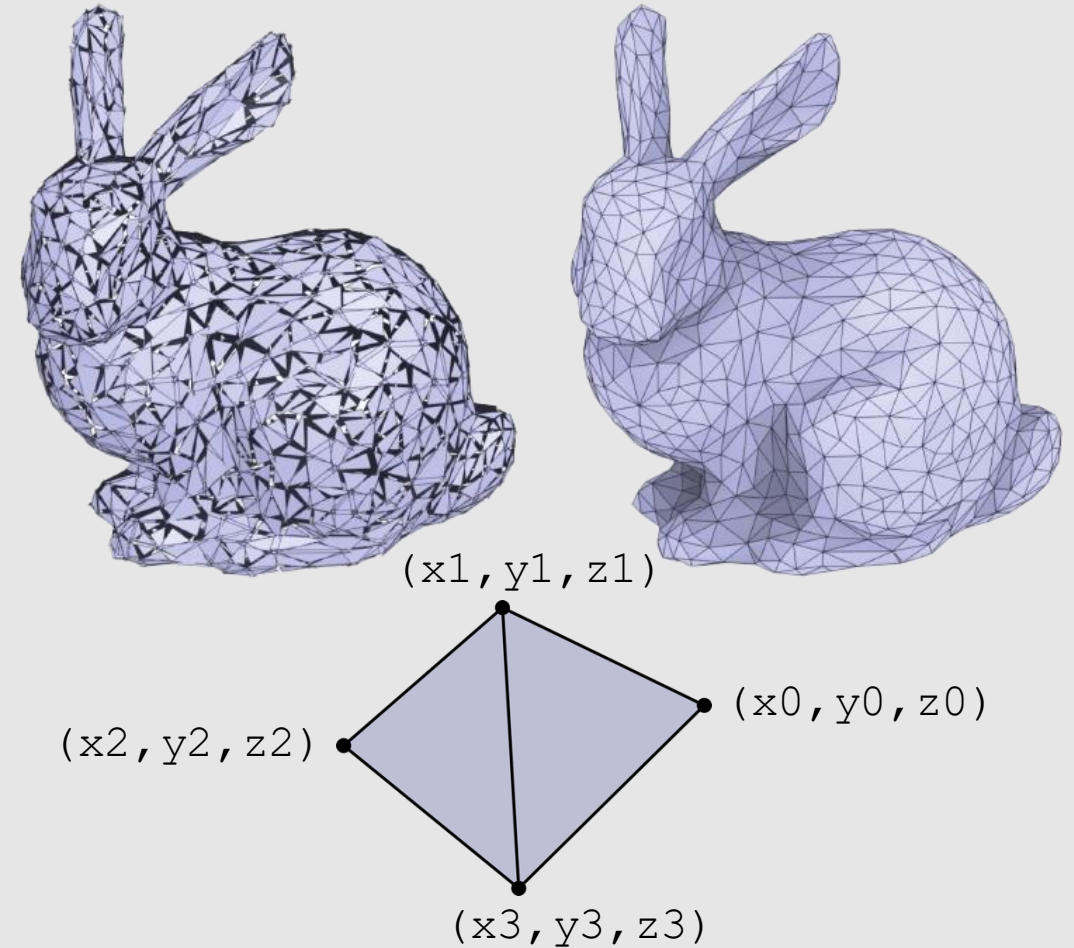
YES



What are some ways to describe the connectivity of geometry?

Polygon Soup

- Most basic idea:
 - For each triangle, store three coordinates
 - No other information about connectivity
 - Not much different from point cloud
 - A “Triangle cloud”?
- **Pros:**
 - [+] Really stupid simple
- **Cons:**
 - [-] Really stupid
 - [-] Redundant storage of vertices
 - [-] Very difficult to find neighboring polygons

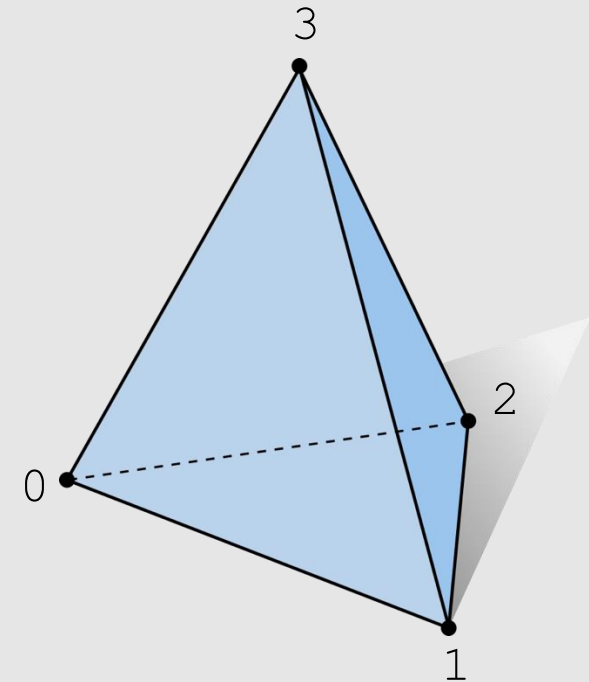


x_0, y_0, z_0	x_1, y_1, z_1	x_3, y_3, z_3
x_1, y_1, z_1	x_2, y_2, z_2	x_3, y_3, z_3

Adjacency List

- A little more complicated:
 - Store triples of coordinates (x,y,z)
 - Store tuples of indices referencing coordinates for each triangle
- **Pros:**
 - [+] No duplicate coordinates
 - [+] Lower memory footprint
 - [+] Easy to keep geometry manifold
 - [+] Supports nonmanifold geometry
 - [+] Easy to change connectivity of geometry
- **Cons:**
 - [-] Very difficult to find neighboring polygons
 - [-] Difficult to add/remove mesh elements

<u>VERTICES</u>				<u>POLYGONS</u>		
	x	y	z	i	j	k
0:	-1	-1	-1	0	2	1
1:	1	-1	1	0	3	2
2:	1	1	-1	3	0	1
3:	-1	1	1	3	1	2



Incidence Matrices

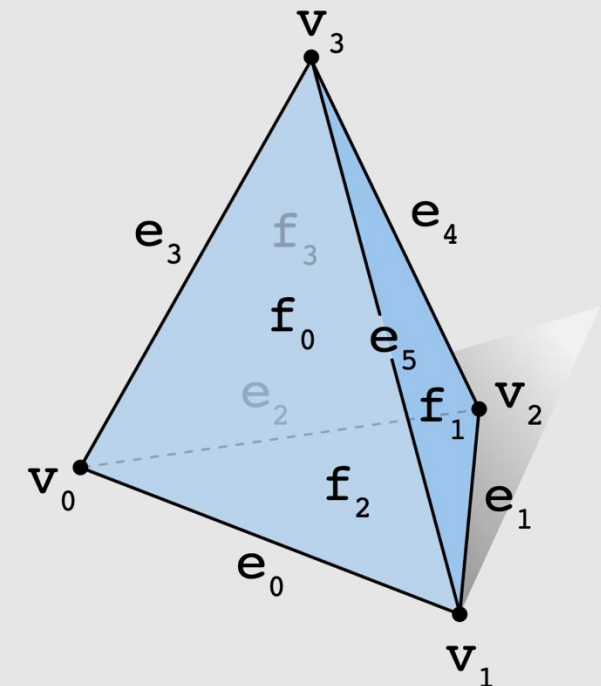
- If we want to know our neighbors, store incidence matrix between vertices + edges, and edges + faces
 - 1 means touch, 0 means no touch
 - Store as sparse matrix
- **Pros:**
 - [+] No duplicate coordinates
 - [+] Finding neighbors is $O(1)$
 - [+] Easy to keep geometry manifold
 - [+] Supports nonmanifold geometry
- **Cons:**
 - [-] Larger memory footprint
 - [-] Hard to change connectivity with fixed indices
 - [-] Difficult to add/remove mesh elements

VERTEX ↔ EDGE

	v0	v1	v2	v3
e0	1	1	0	0
e1	0	1	1	0
e2	1	0	1	0
e3	1	0	0	1
e4	0	0	1	1
e5	0	1	0	1

EDGE ↔ FACE

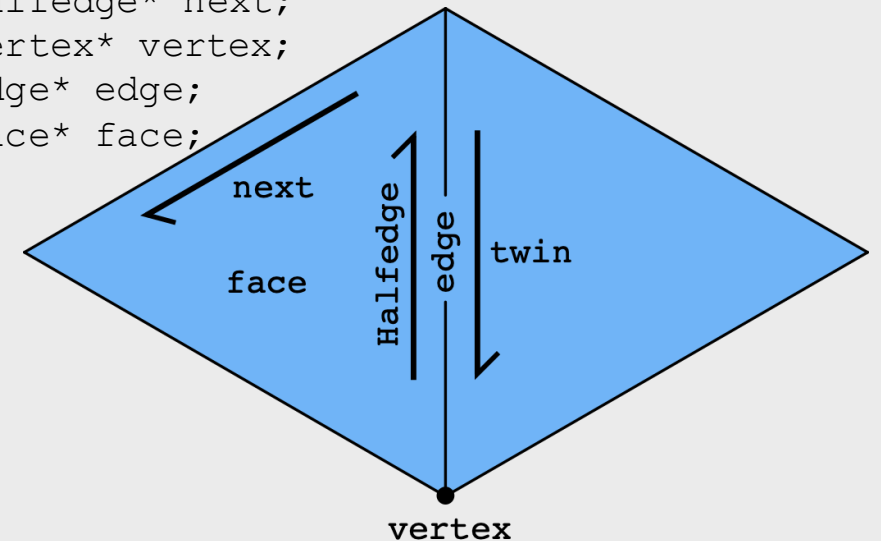
	e0	e1	e2	e3	e4	e5
f0	1	0	0	1	0	1
f1	0	1	0	0	1	1
f2	1	1	1	0	0	0
f3	0	0	1	1	1	0



Halfedge Data Structure

- Let's add more about our neighbors:
 - Each edge gets 2 halfedges
 - Each halfedge "glues" an edge to a face
- **Pros:**
 - [+] No duplicate coordinates
 - [+] Finding neighbors is $O(1)$
 - [+] Easy to traverse geometry
 - [+] Easy to change mesh connectivity
 - [+] Easy to add/remove mesh elements
 - [+] Easy to keep geometry manifold
- **Cons:**
 - [-] Does not support nonmanifold geometry

```
struct Halfedge
{
    Halfedge* twin;
    Halfedge* next;
    Vertex* vertex;
    Edge* edge;
    Face* face;
};
```



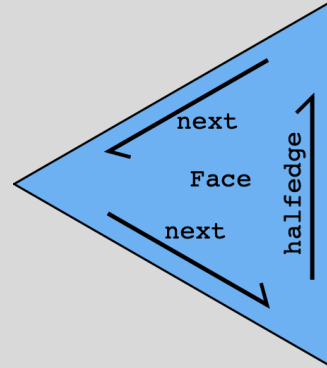
Halfedge Data Structure

- Makes mesh traversal easy
 - Use “twin” and “next” pointers to move around the mesh
 - Use “vertex”, “edge”, and “face” pointers to grab element

```
struct Halfedge
{
    Halfedge* twin;
    Halfedge* next;
    Vertex* vertex;
    Edge* edge;
    Face* face;
};
```

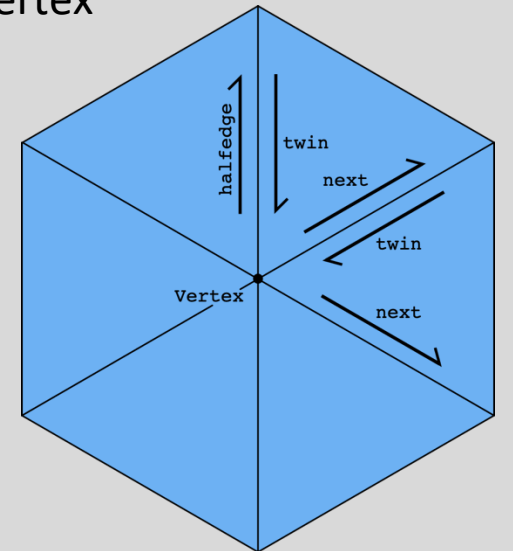
Example: visit all vertices in a face

```
Halfedge* h = f->halfedge;
do {
    h = h->next;
    // do something w/ h->vertex
}
while( h != f->halfedge );
```



Example: visit all neighbors of a vertex

```
Halfedge* h = v->halfedge;
do {
    h = h->twin->next;
}
while( h != v->halfedge );
```



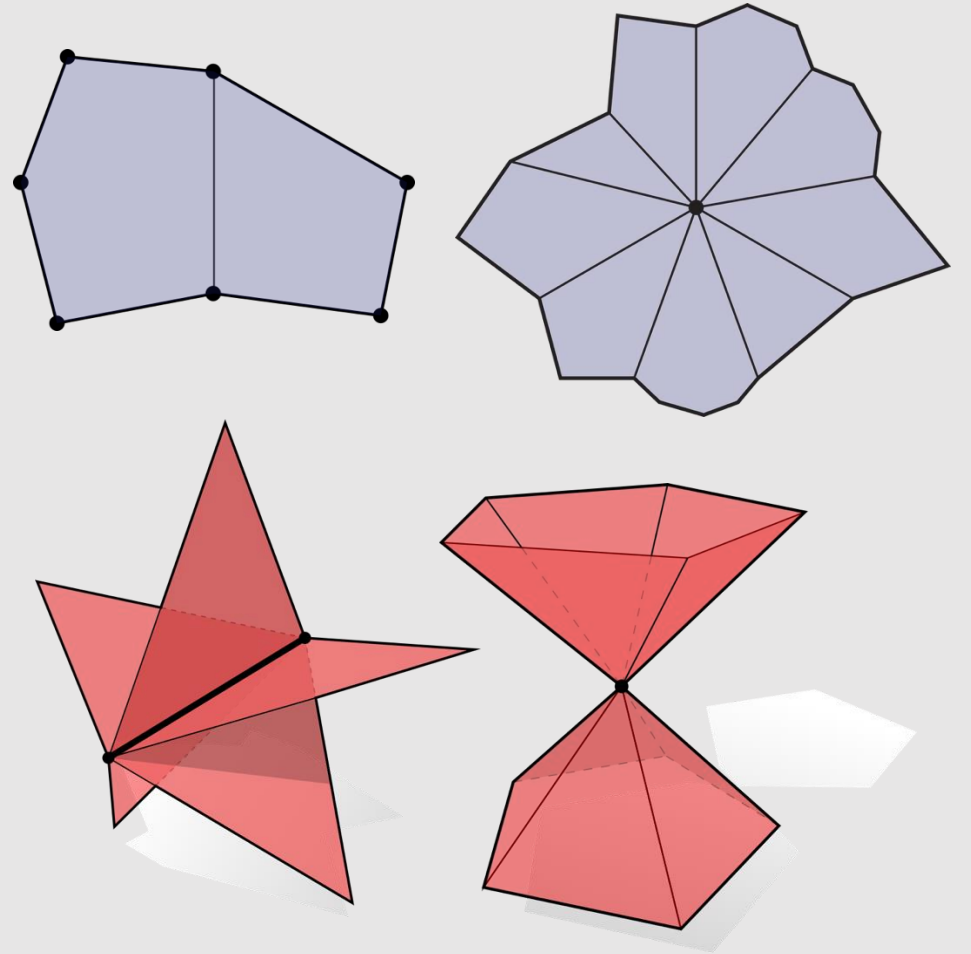
Note: only makes sense if mesh is manifold!

Halfedge Data Structure

- Halfedge meshes are always manifold!
- Have the following constraints:

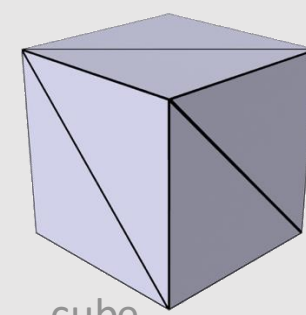
```
h->twin->twin == h // my twin's twin is me  
h->twin != h // I am not my own twin  
h2->next = h //every h is someone's "next"
```

- Keep following **next** and you'll traverse a face
- Keep following **twin** and you'll traverse an edge
- Keep following **next->twin** and you'll traverse a vertex
- **Q: Why is it impossible to encode the red figures?**
 - First shape violates 1st and 2nd conditions
 - Second shape violates 3rd condition

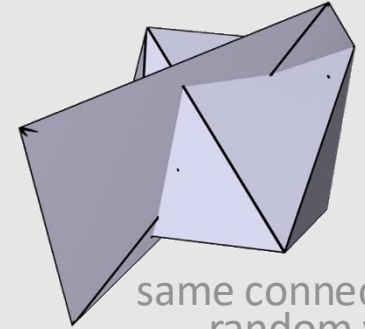


Connectivity vs Geometry

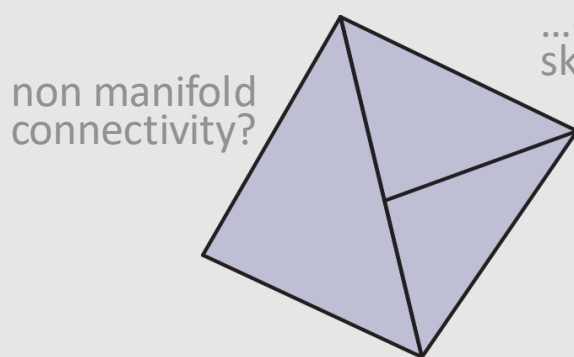
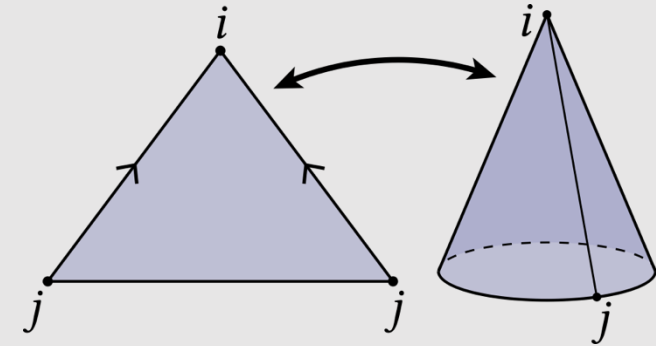
- Recall manifold conditions (fans not fins):
 - These conditions say nothing about vertex positions! Just connectivity
- Leads to confusion when debugging:
 - Mesh looks “bad”, even though connectivity is fine
 - Scotty3D should still be able to support these “bad” meshes



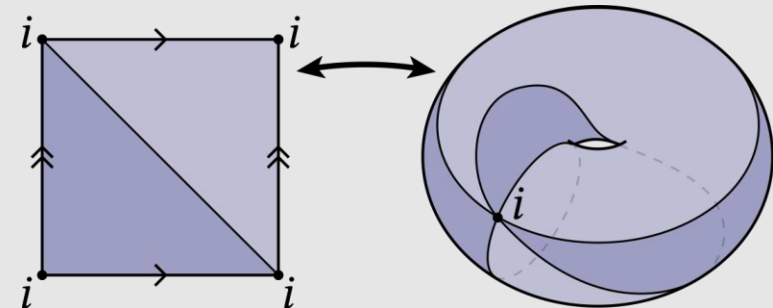
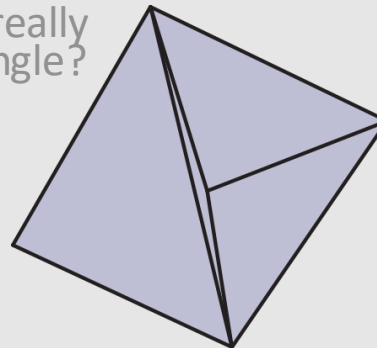
cube
(manifold)



same connectivity,
random vertex
positions



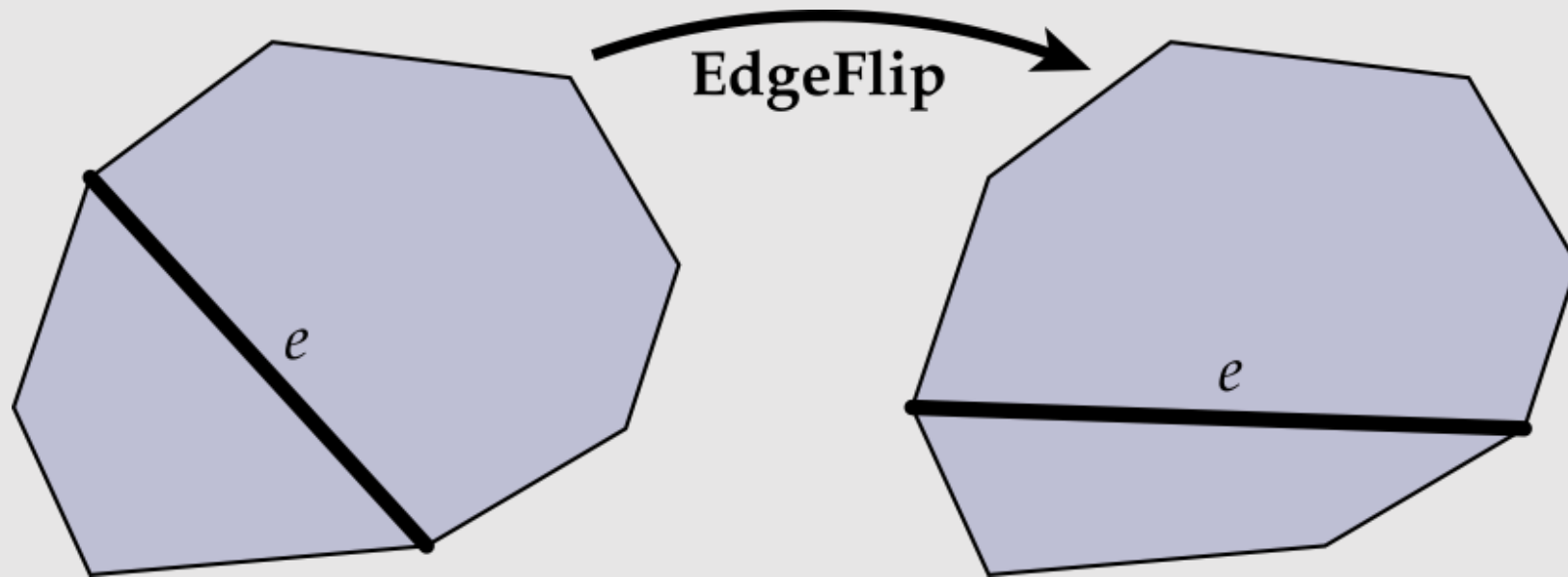
...or just a really
skinny triangle?



- ~~Implicit & Explicit Geometry~~
- ~~Manifold Geometry~~
- Local Geometric Operations

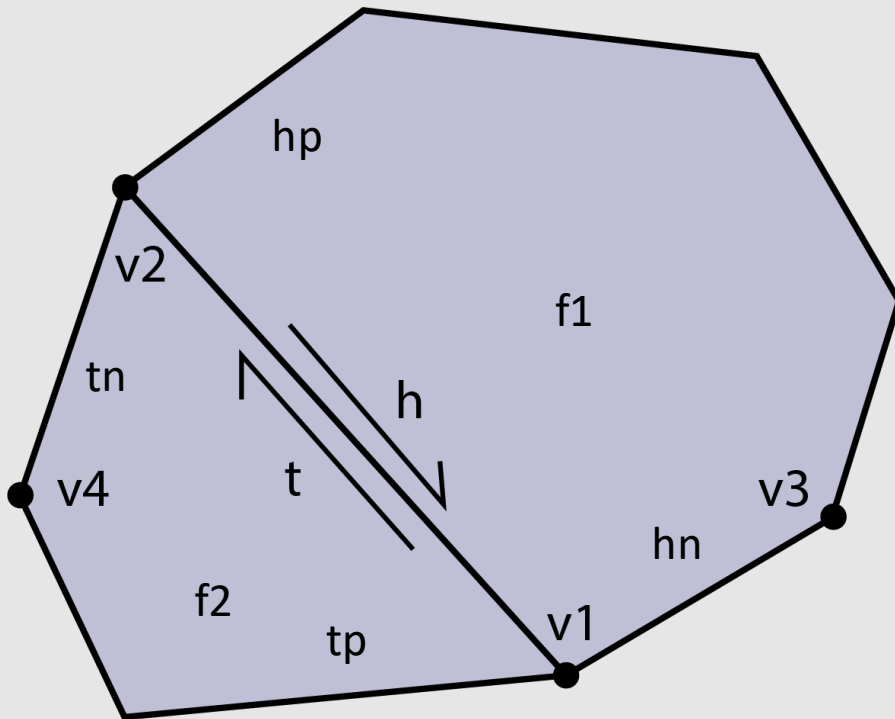
Edge Flip

Goal: Move edge e around faces adjacent to it:



- No elements created/destroyed, just pointer reassignment
- Flipping the same edge multiple times yields original results

Edge Flip



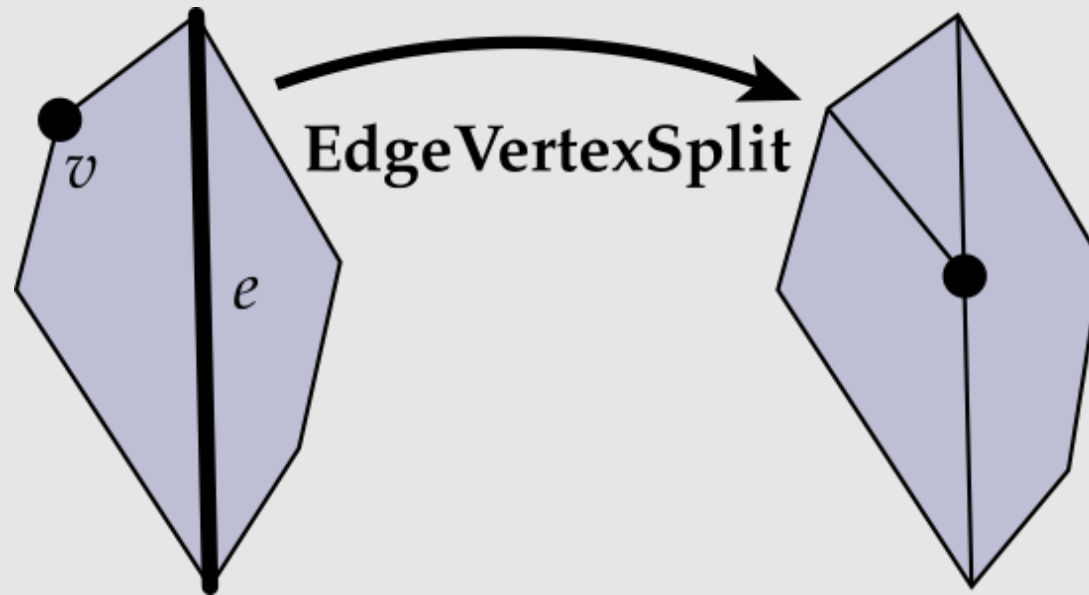
Q: what are the “edge case”?

```
// collect
h = e->halfedge;
t = h->twin;
v1 = h->next->vertex;
v2 = t->next->vertex;
v3 = h->next->next->vertex;
v4 = t->next->next->vertex;
f1 = h->face;      f2 = t->face;
hn = h->next;      tn = t->next;
hp = h->next; while(hp->next != h) { hp = hp->next; }
tp = t->next; while(tp->next != t) { tp = tp->next; }

// connect
v1->halfedge = hn;  v2->halfedge = tn;
f1->halfedge = h;   f2->halfedge = t;
t->vertex = v3;      h->vertex = v4;
tp->next = hn;       hp->next = tn;
hn->next = t;        tn->next = h;
```

Edge Vertex Split

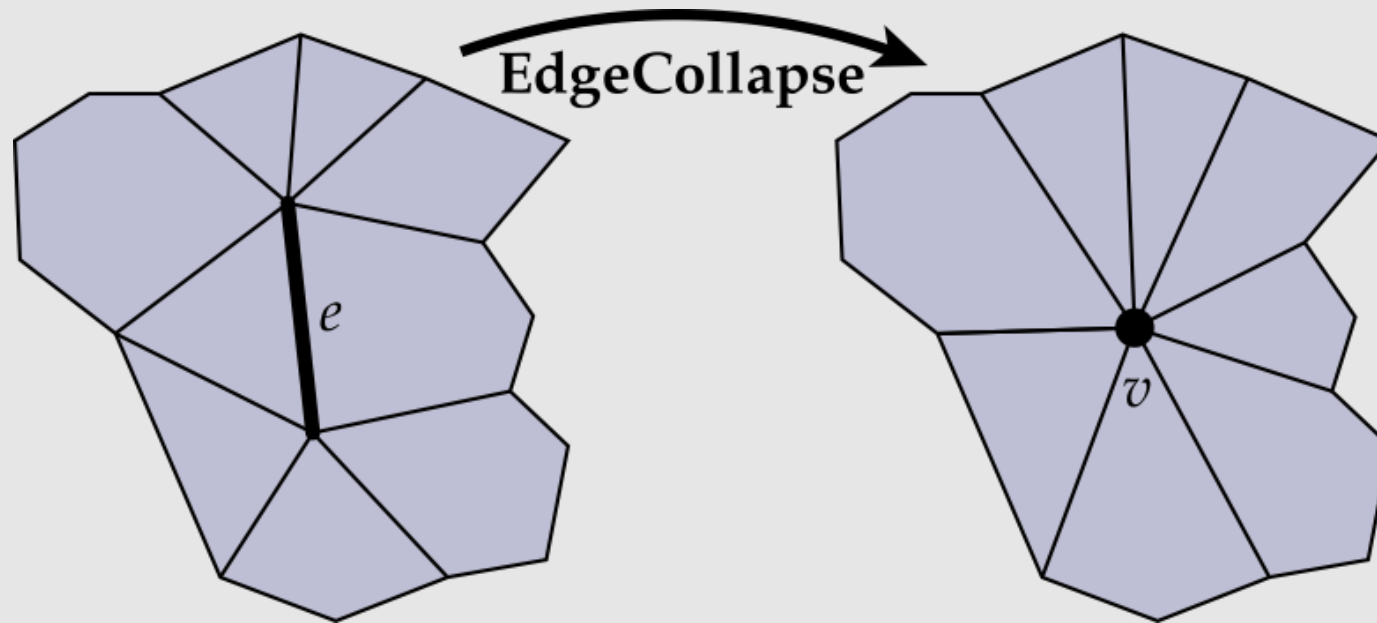
Goal: Insert edge between vertex v and midpoint of edge e :



- Creates a new vertex, new edge, and new face
- Involves much more pointer reassignments

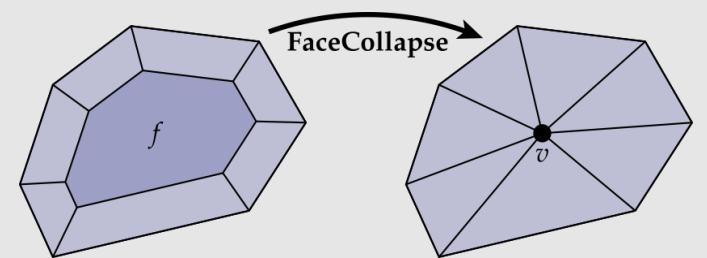
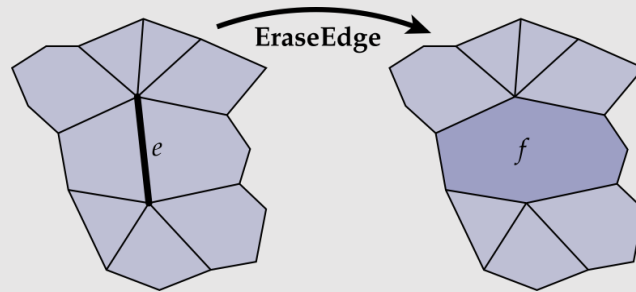
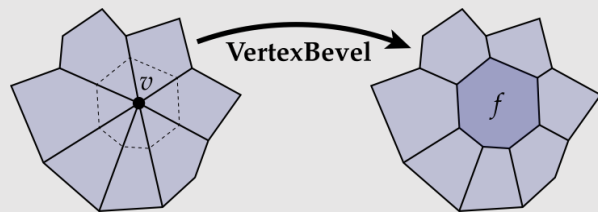
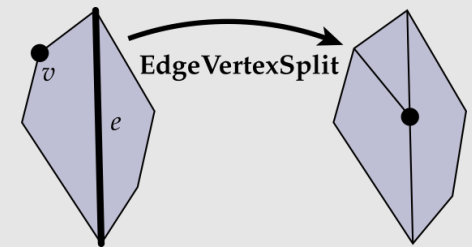
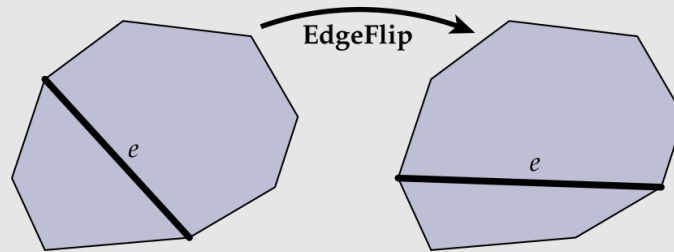
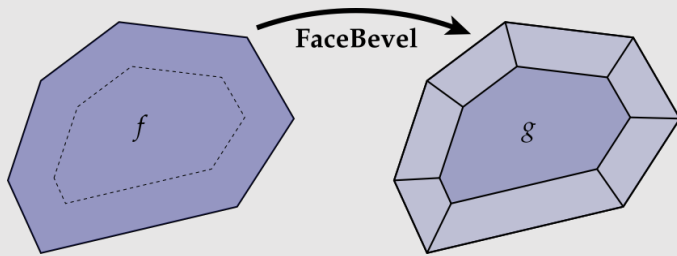
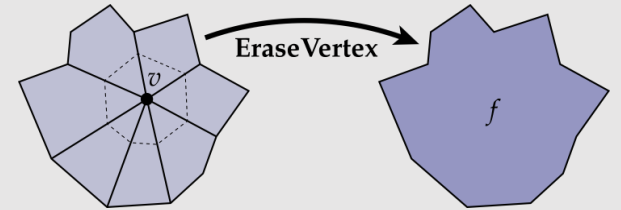
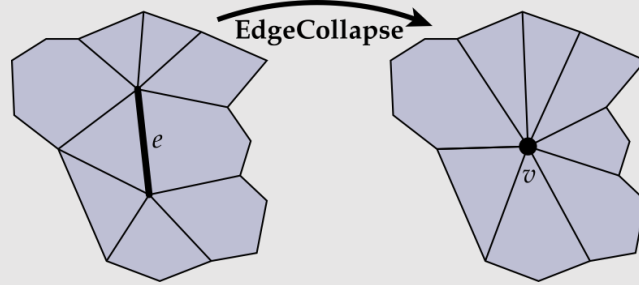
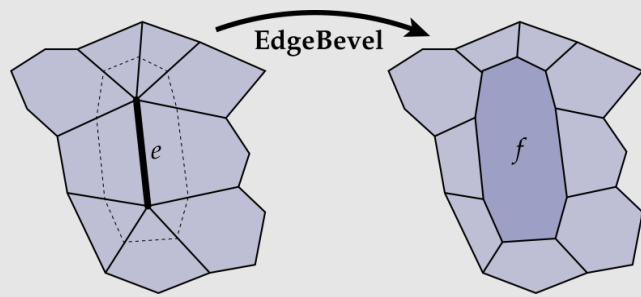
Edge Collapse

Goal: Replace edge (c,d) with a single vertex m:



- Deletes a vertex, (up to) 3 edges, and (up to) 2 faces
 - Depends on the degree of the original faces

Local Operations



Many other local operations you will explore in your homework...

Local Operation Tips

- Always draw out a diagram
 - We've given you some unlabeled diagrams
 - With pen + paper, label the elements you'll need to collect/create
- Stage your code in the following way:
 - Create
 - Collect
 - Connect
 - Delete
- Write asserts around your code
 - Check elements that should be deleted were deleted
 - Make sure there are no references to anything deleted
 - Make sure every element that you connect is valid
 - What it means for a vertex to be valid is not the same as what it means for an edge to be valid, etc.

