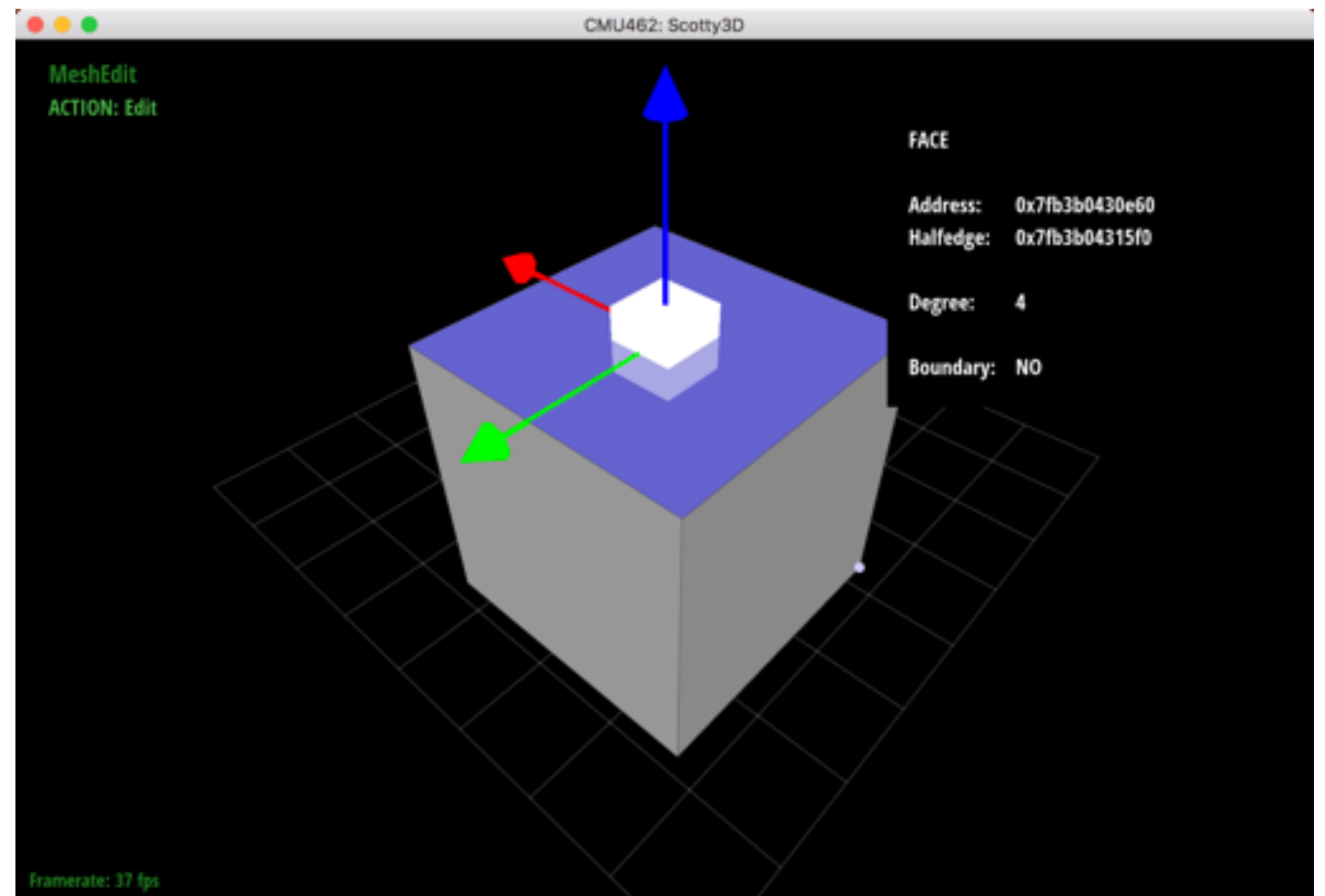


Introduction to Geometry

Computer Graphics
CMU 15-462/15-662

Assignment 2

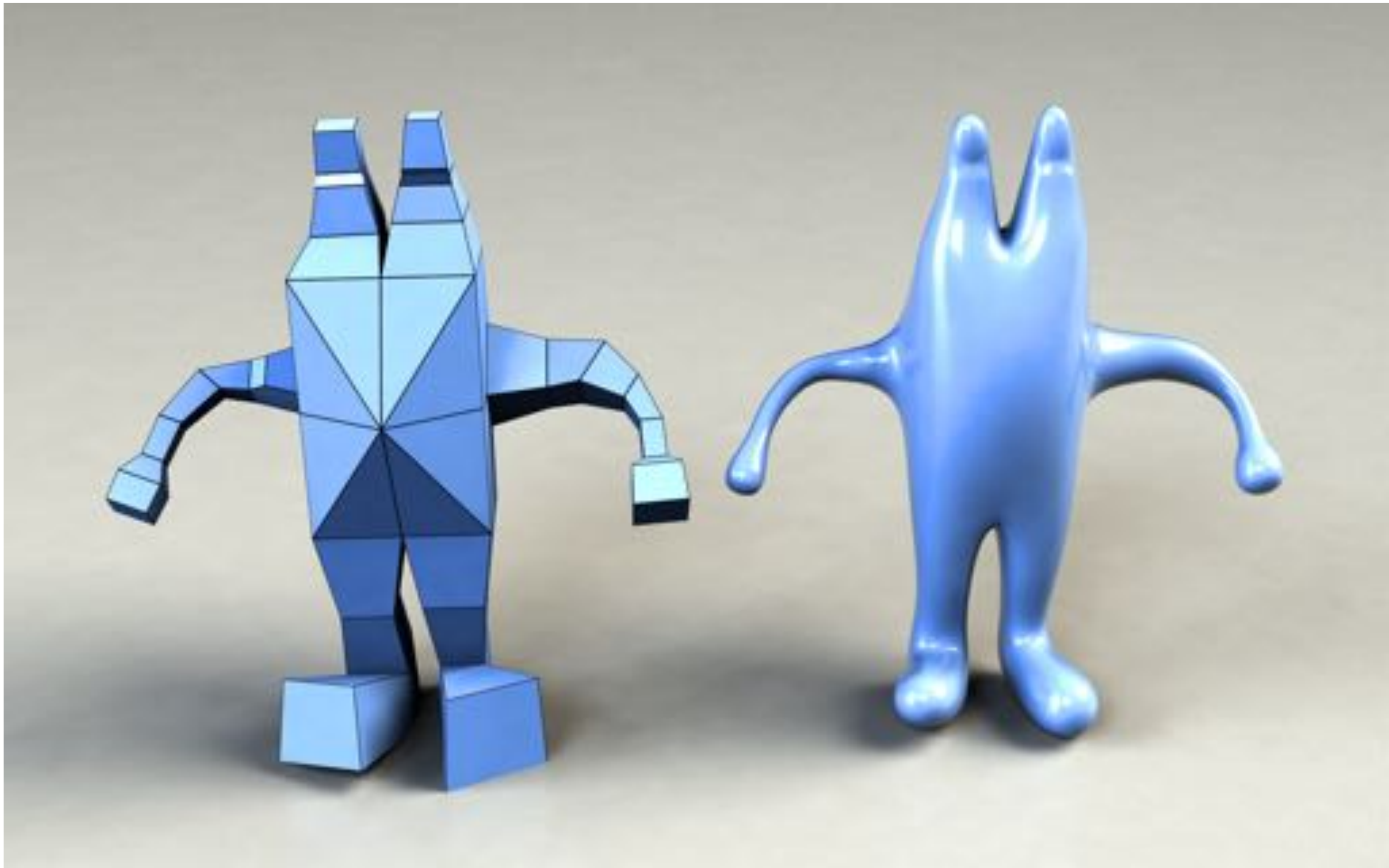
- Start building up “Scotty3D”; first part is 3D modeling



(Start from the cube you described in Lecture 1!)

3D Modeling Competition

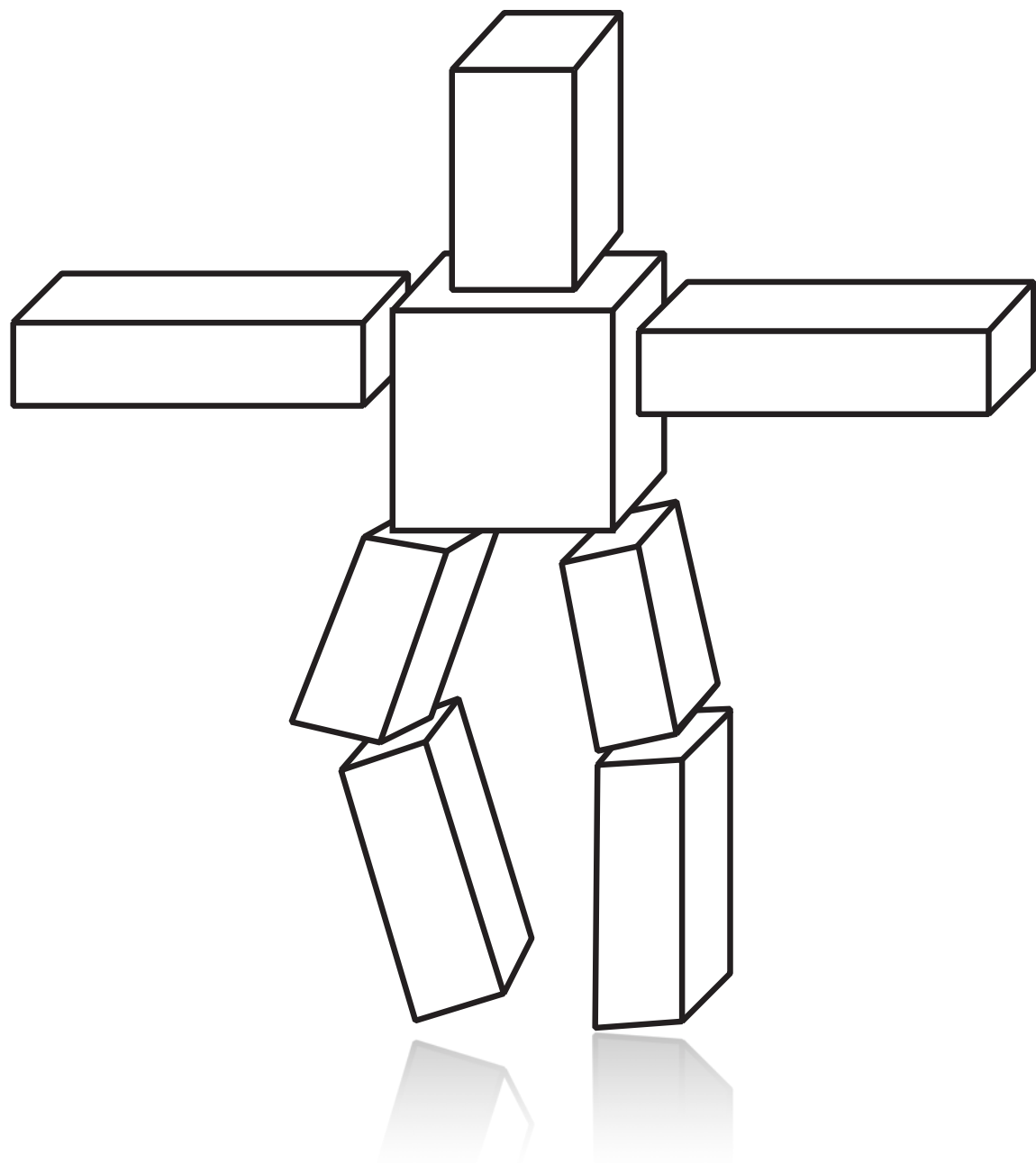
- Don't just make great software... make great art! :-)



(This mesh was created in Scotty3D in about 5 minutes... you can do much better!)

Increasing the complexity of our models

Transformations



Geometry



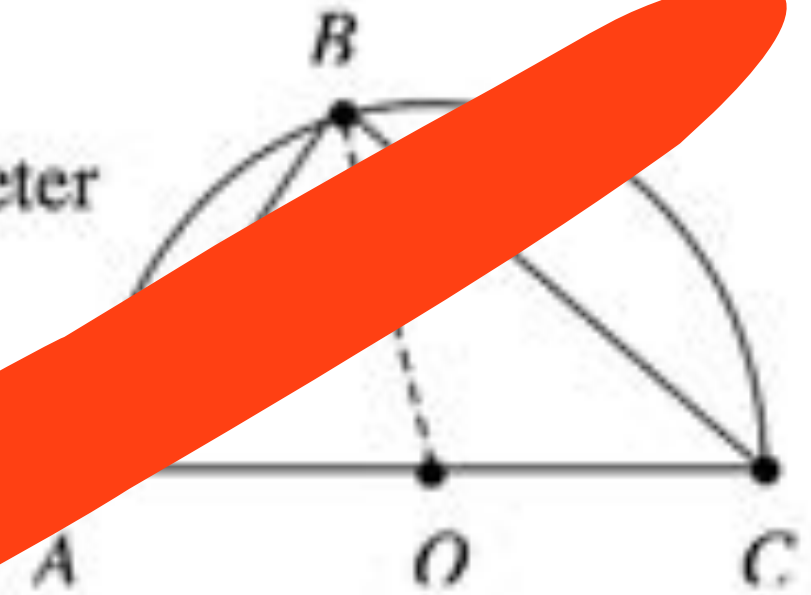
Materials, lighting, ...



Q: What is geometry?

A: Geometry is the study of two-column proofs.

THEOREM 9.5. Let $\triangle ABC$ be inscribed in a semicircle with diameter \overline{AC} . Then $\angle ABC$ is a right angle.



Proof:

Statement

1. Draw radius OB . Then $OB = OC = OA$.
2. $m\angle OBC = m\angle BCA$
 $m\angle OBA = m\angle BAC$
3. $m\angle ABC = m\angle OBA + m\angle OBC$
4. $m\angle ABC + m\angle BCA + m\angle BAC = 180$
5. $m\angle ABC + m\angle OBA + m\angle OBC = 180$
6. $2m\angle ABC = 180$
7. $m\angle ABC = 90$
8. $\angle ABC$ is a right angle

Given

1. Isosceles Triangle Theorem
2. Angle Addition Postulate
3. The sum of angles of a triangle is 180
4. Substitution (line 1)
5. Substitution (line 3)
6. Substitution (line 3)
7. Division Property of Equality
8. Definition of Right Angle

Ceci n'est pas géométrie.

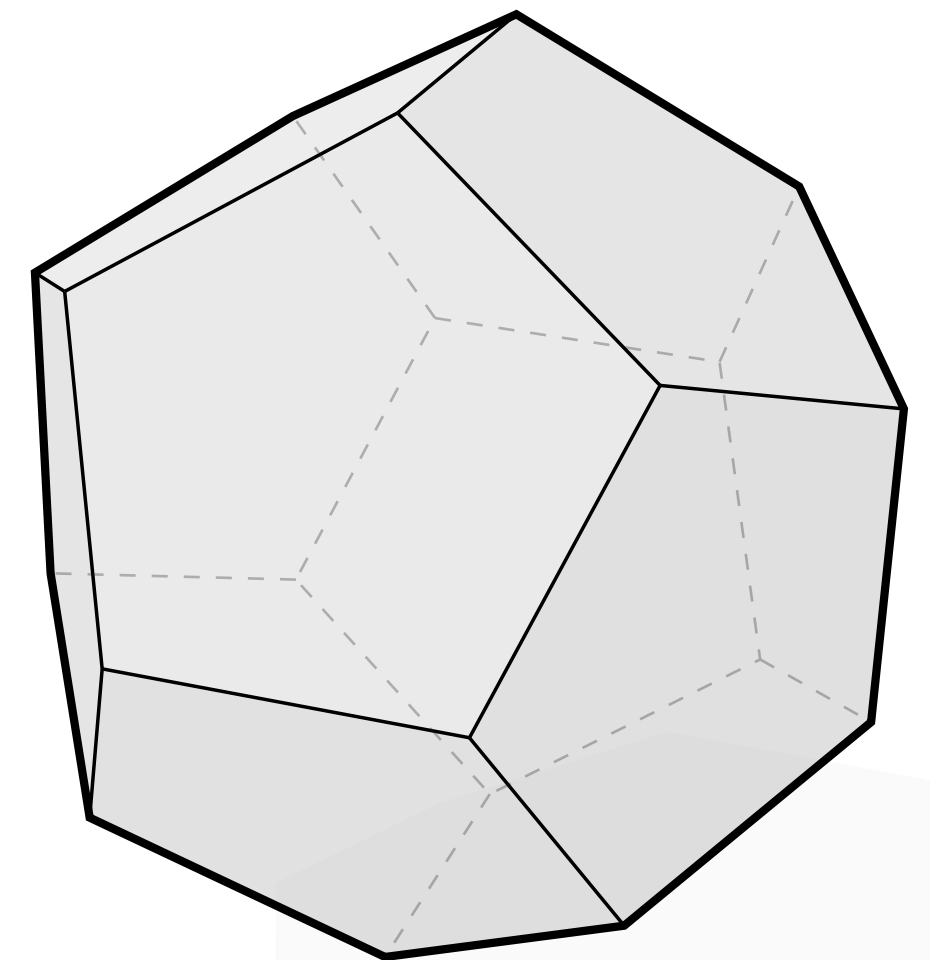
What is geometry?

“Earth”

“measure”

ge • om • et • ry /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*.



Plato: “...the earth is in appearance like one of those balls which have leather coverings in twelve pieces...”

How can we describe geometry?

IMPLICIT

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

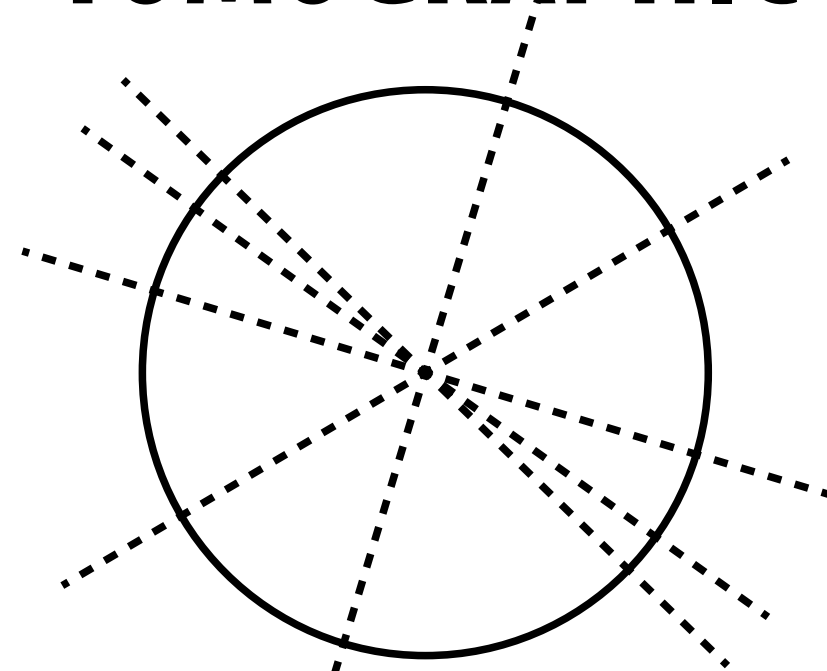
LINGUISTIC

“unit circle”

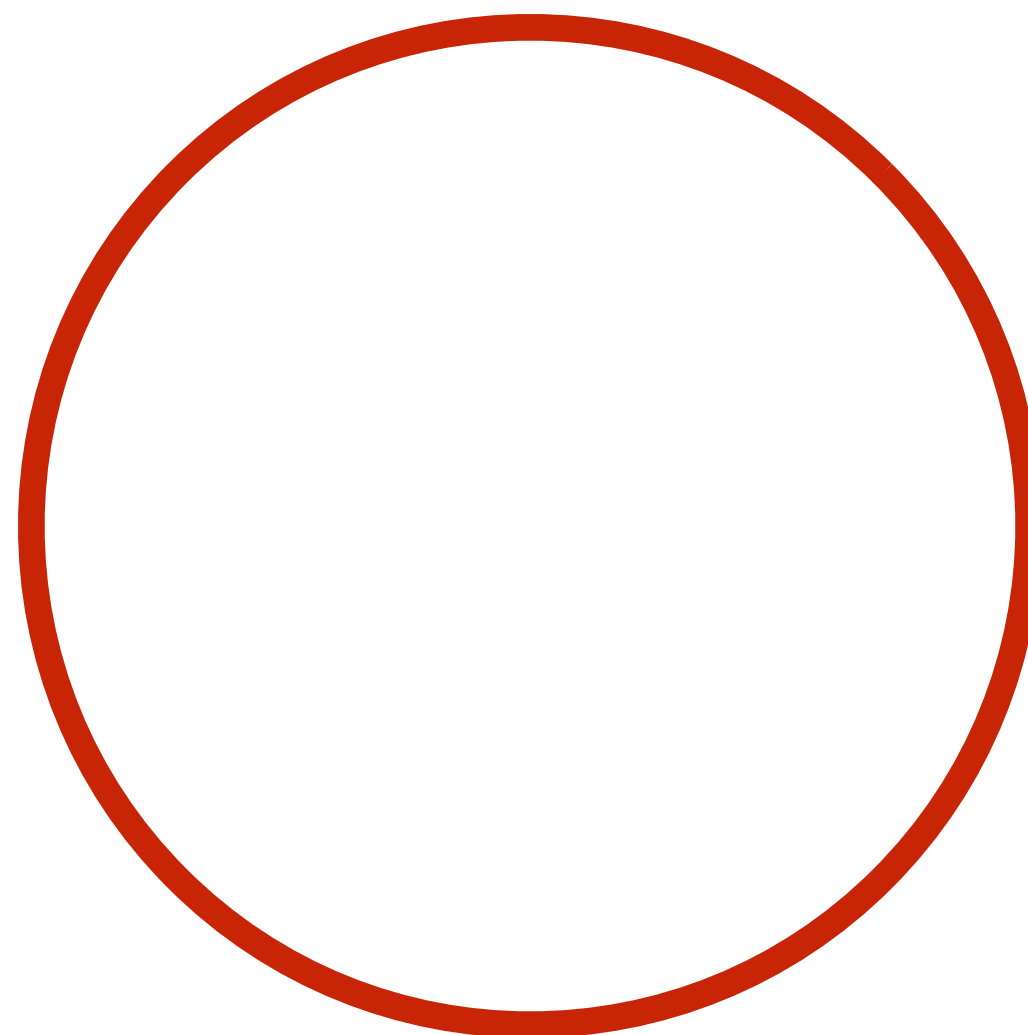
EXPLICIT

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

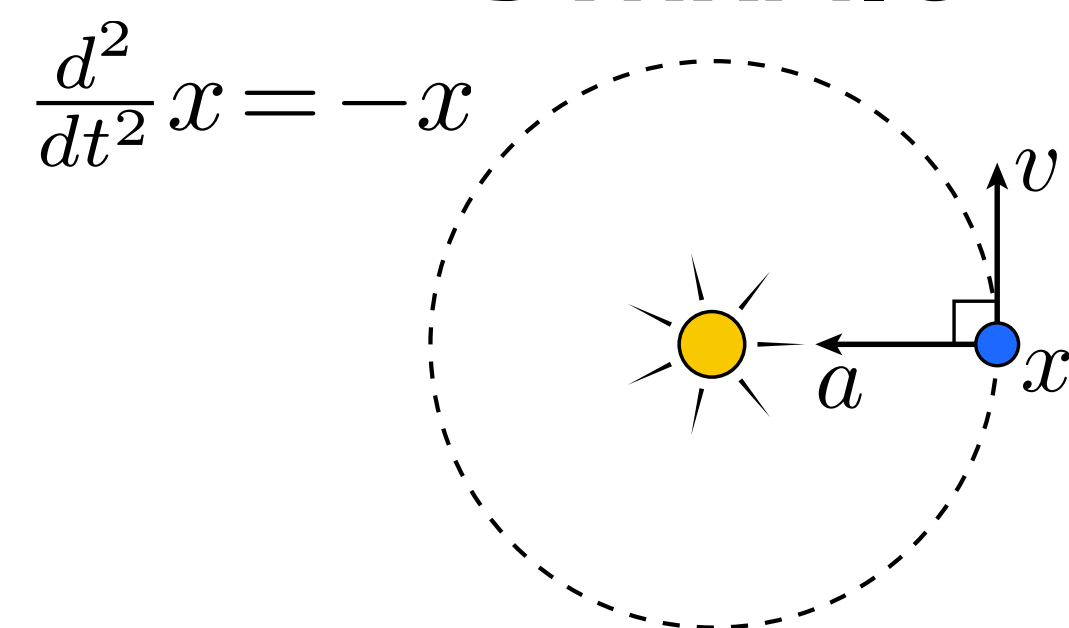
TOMOGRAPHIC



(constant density)



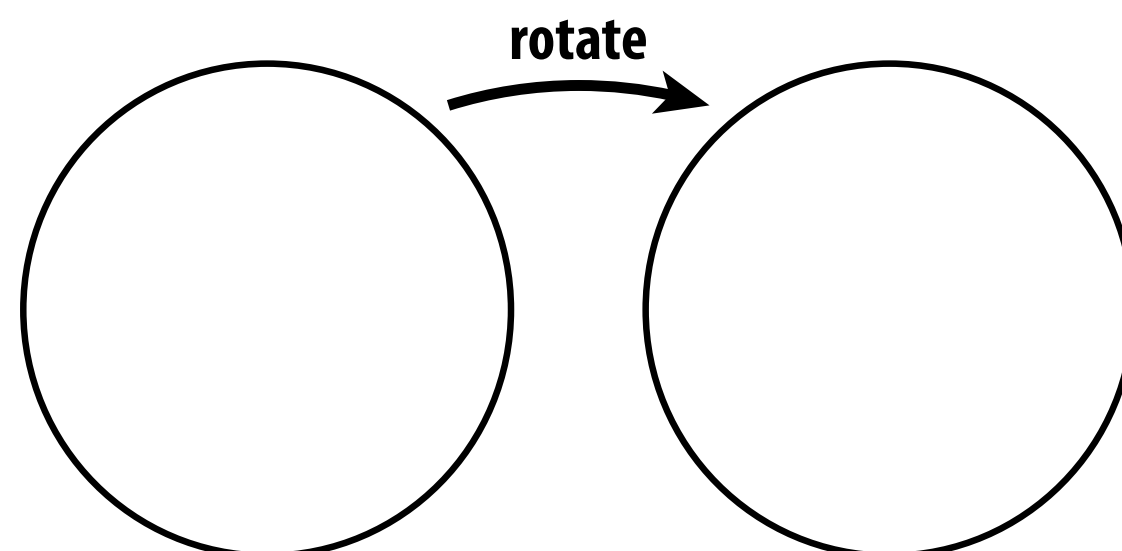
DYNAMIC



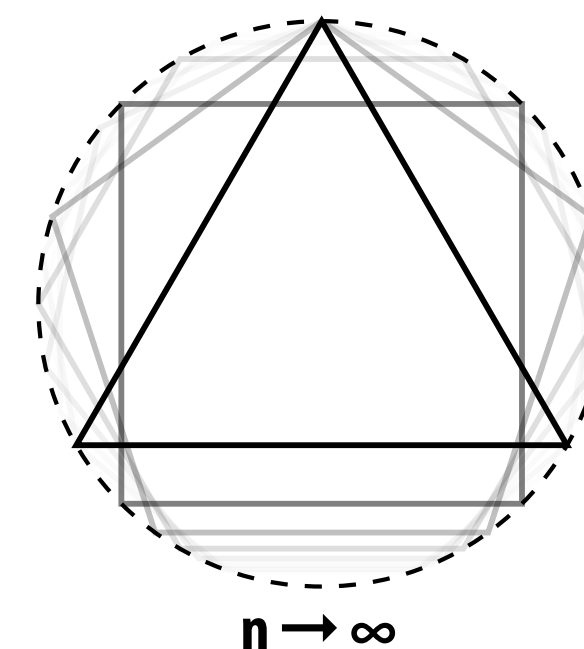
CURVATURE

$$\kappa = 1$$

SYMMETRIC

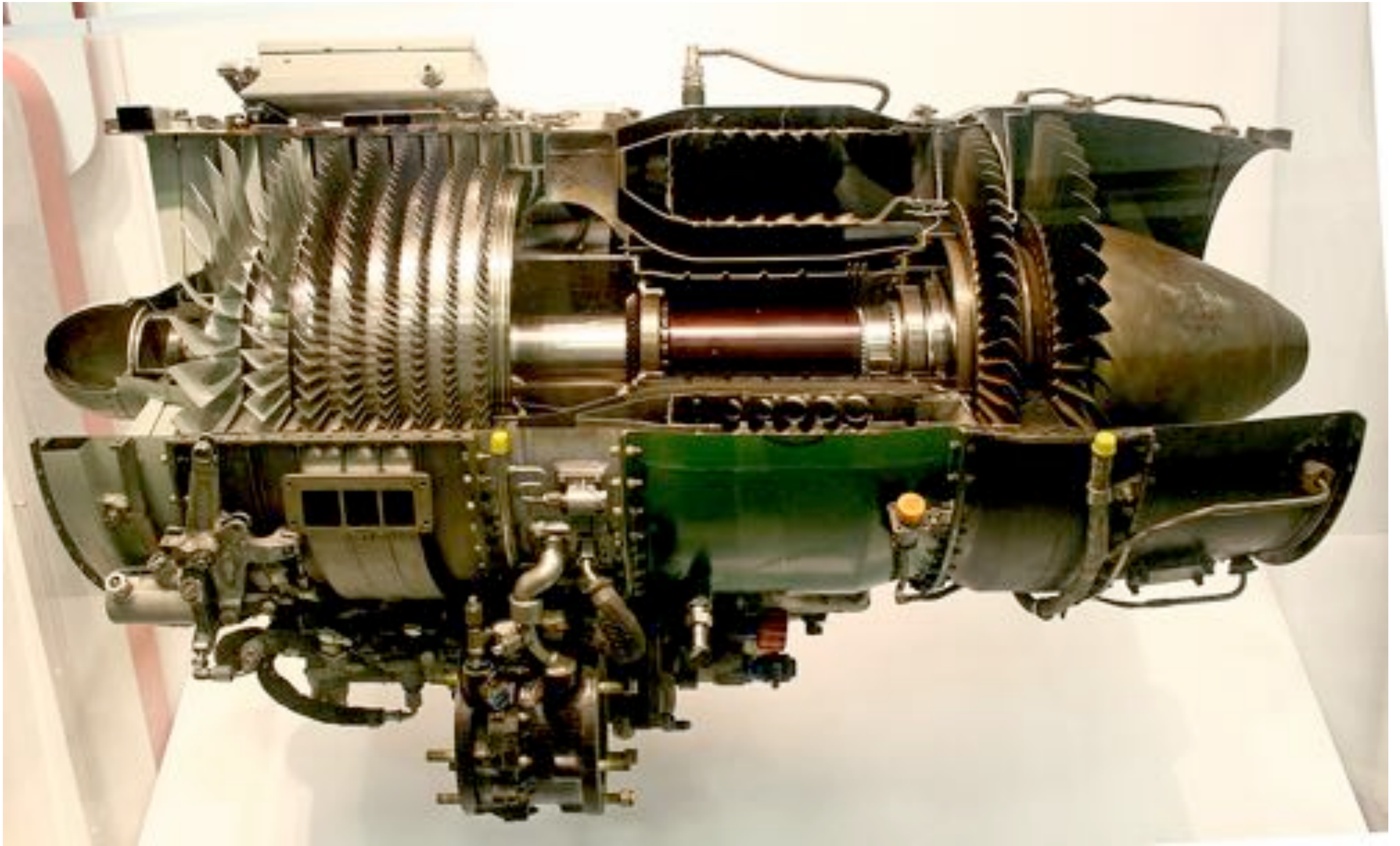


DISCRETE

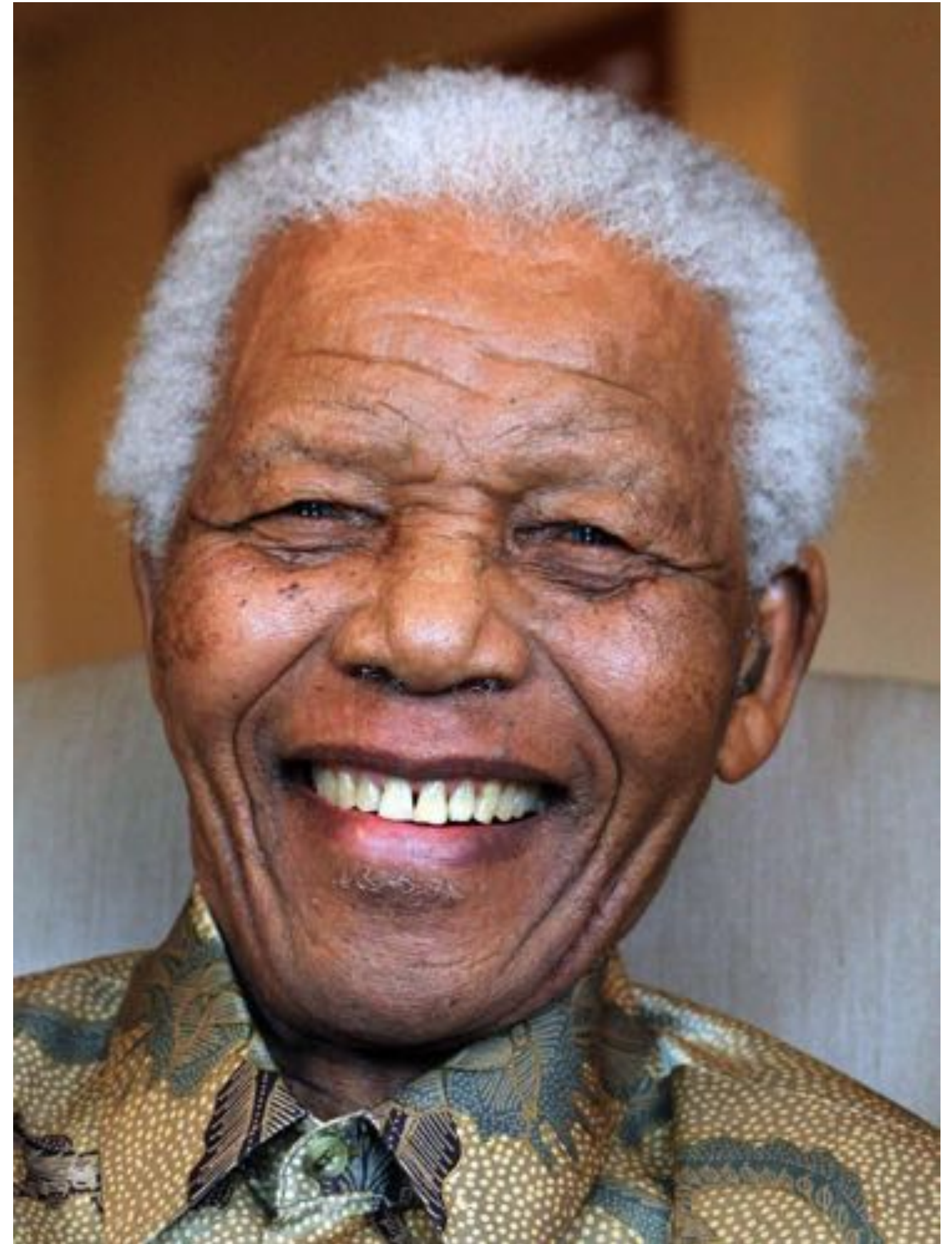


Given all these options, what's the best way to encode geometry on a computer?

Examples of geometry



Examples of geometry



Examples of geometry



Examples of geometry



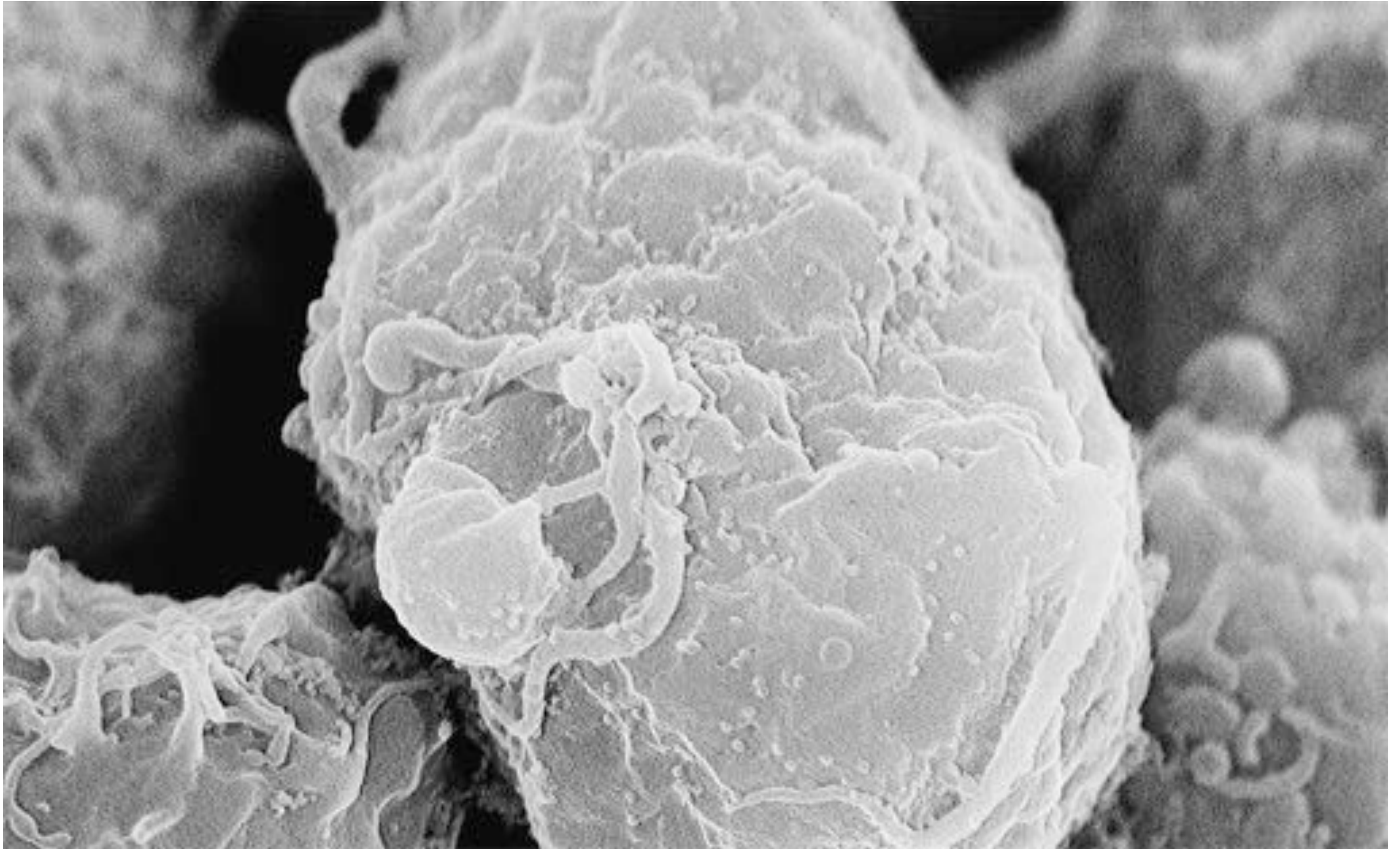
Examples of geometry



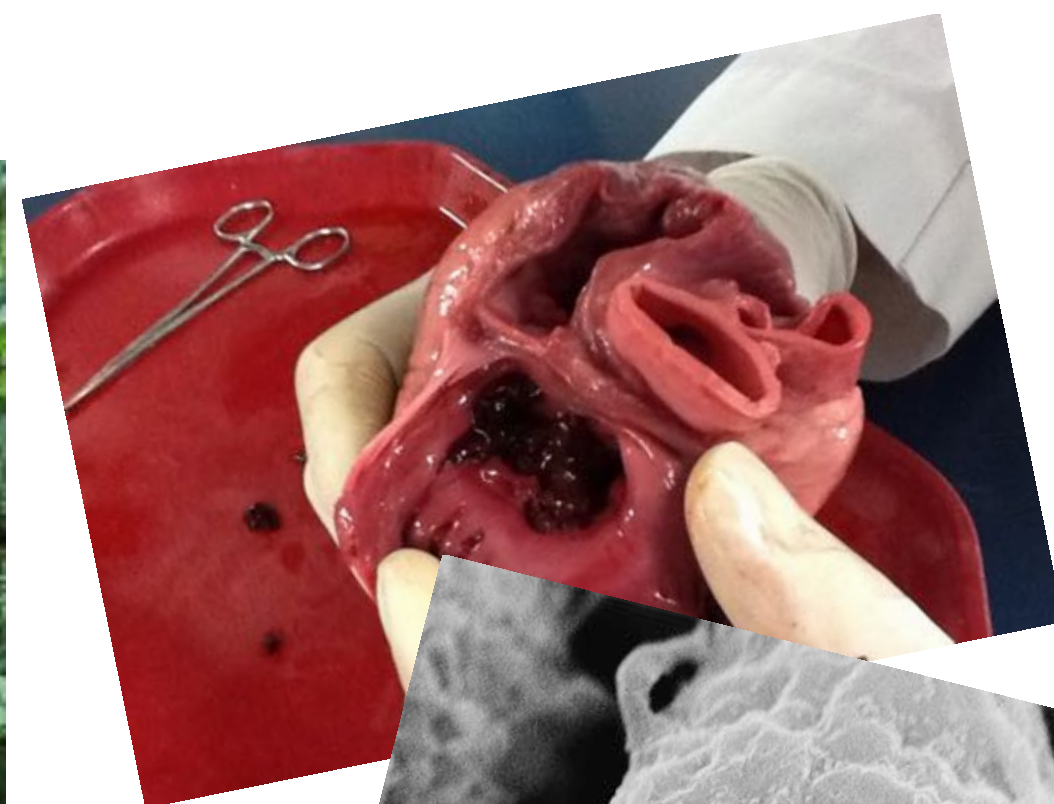
Examples of geometry



Examples of geometry



It's a Jungle Out There!



No one “best” choice—geometry is hard!

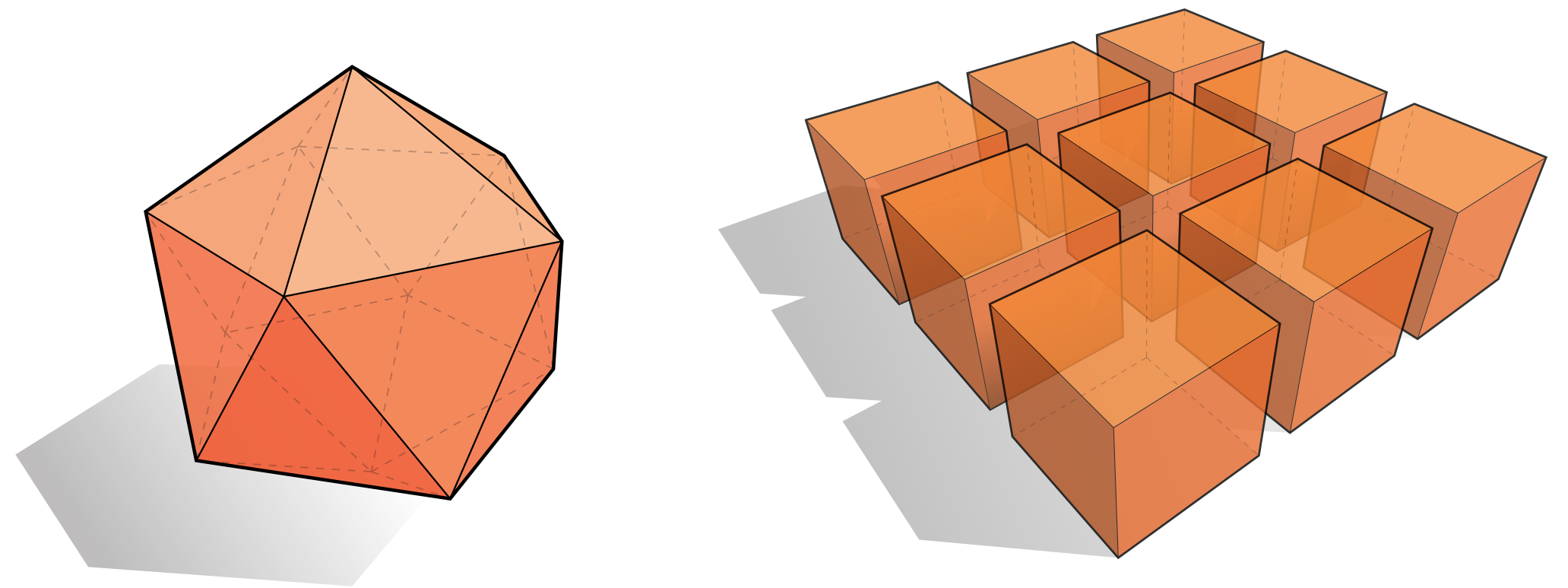
**“I hate meshes.
I cannot believe how hard this is.
Geometry is hard.”**

—David Baraff
Senior Research Scientist
Pixar Animation Studios

Many ways to digitally encode geometry

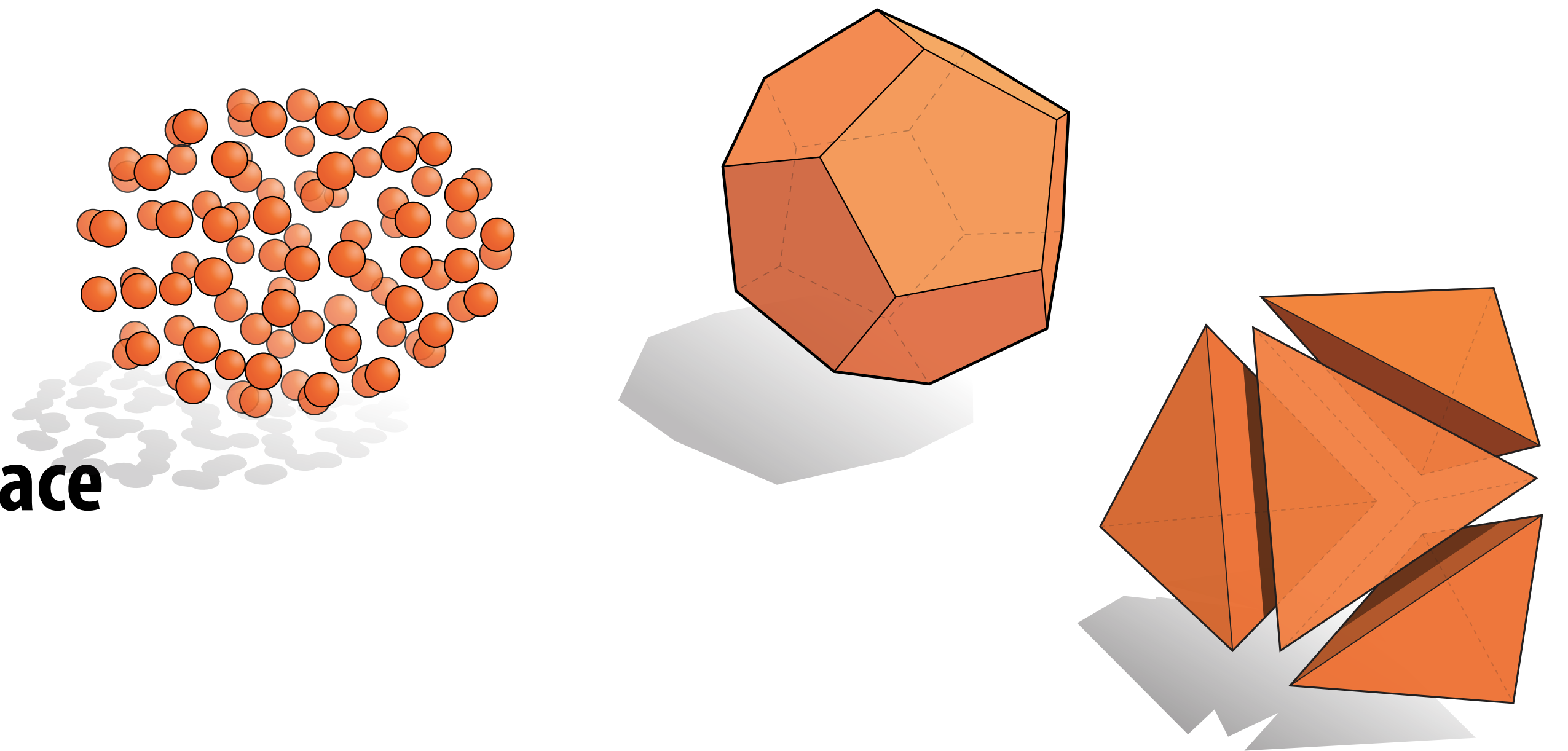
■ EXPLICIT

- point cloud
- polygon mesh
- subdivision, NURBS
- ...



■ IMPLICIT

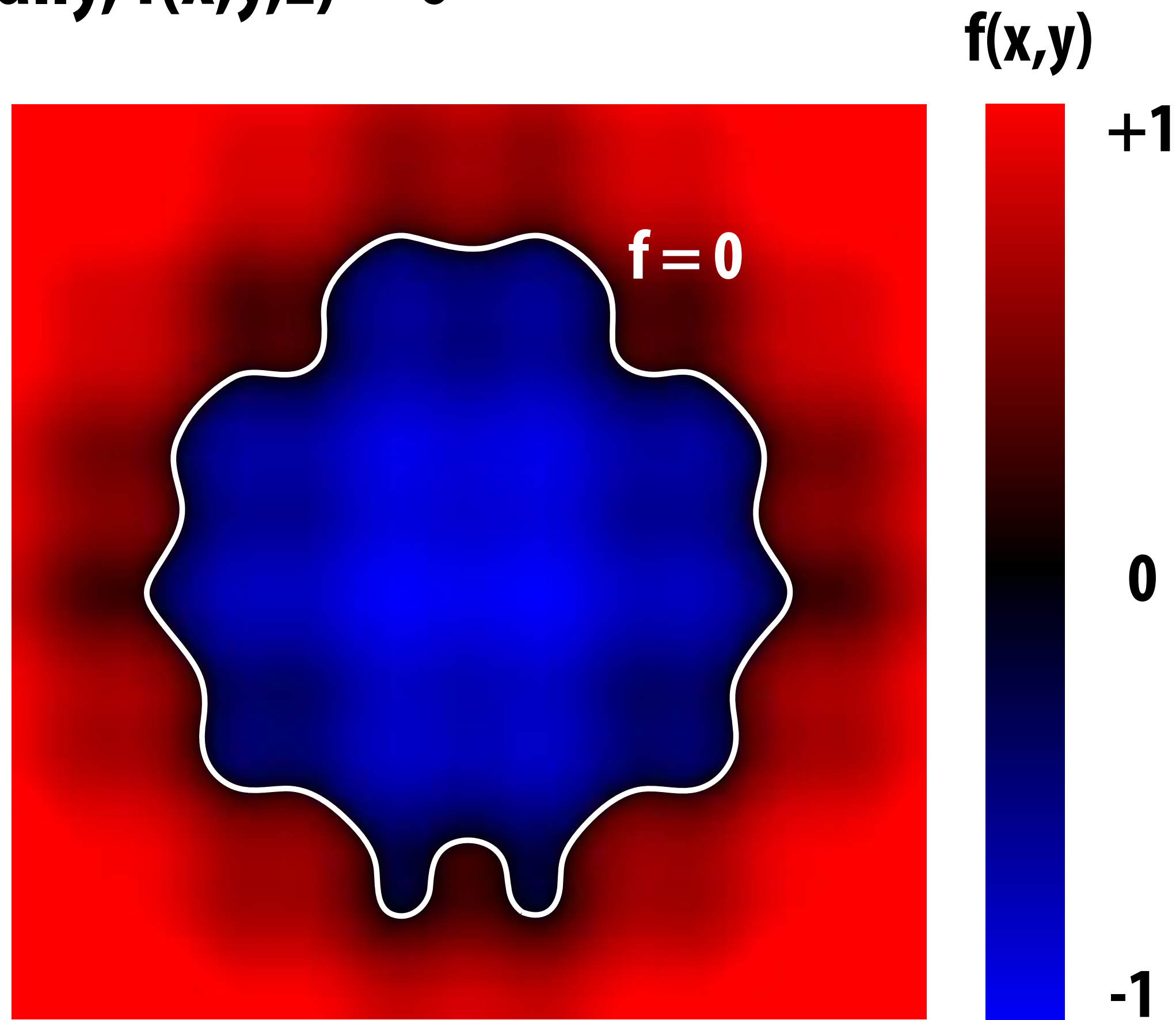
- level set
- algebraic surface
- L-systems
- ...



■ Each choice best suited to a different task/type of geometry

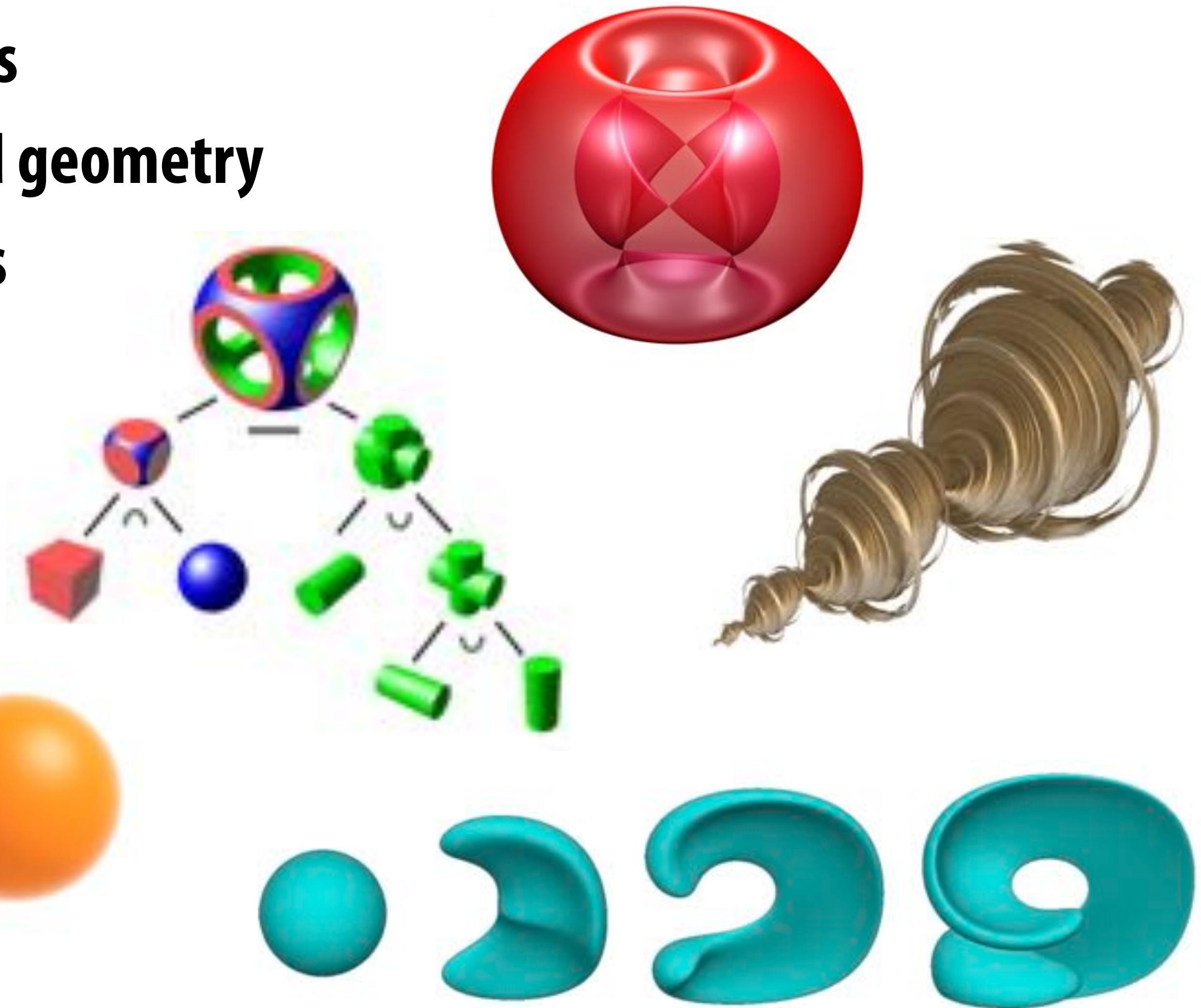
“Implicit” Representations of Geometry

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



Many implicit representations in graphics

- algebraic surfaces
- constructive solid geometry
- level set methods
- blobby surfaces
- fractals
- ...

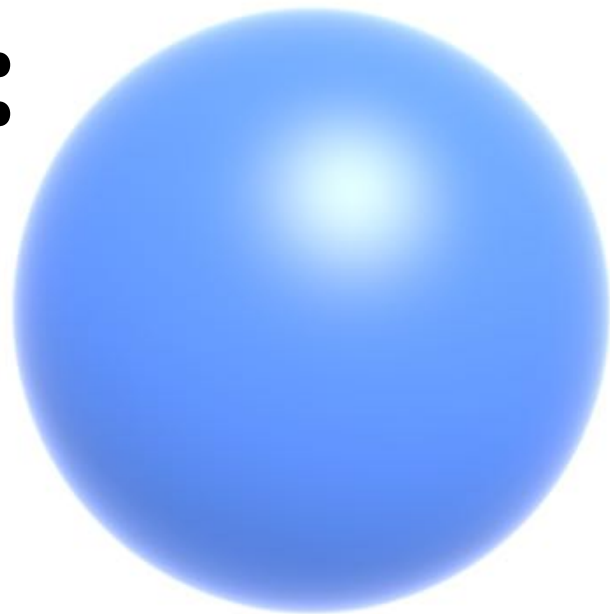


(Will see some of these a bit later.)

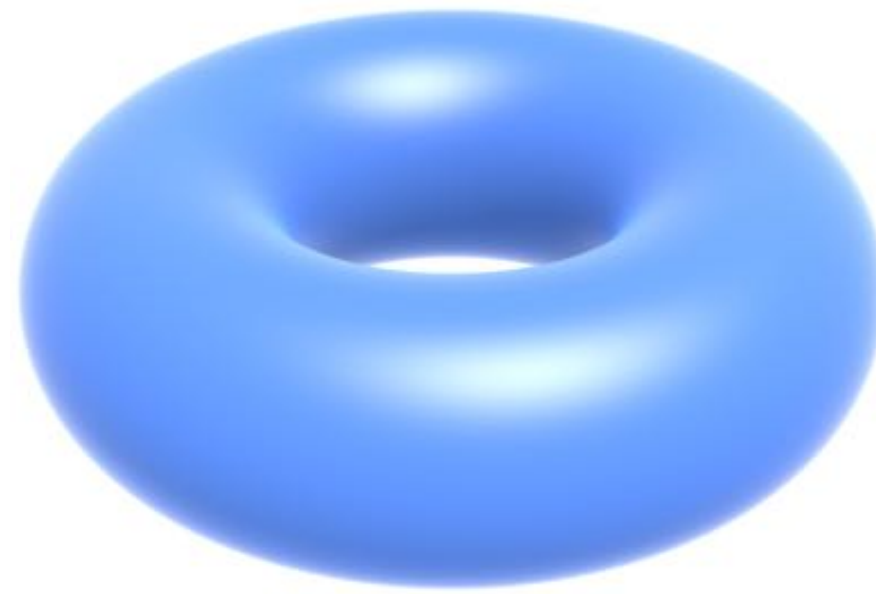
Algebraic Surfaces (Implicit)

- Surface is zero set of a polynomial in x, y, z (“algebraic variety”)

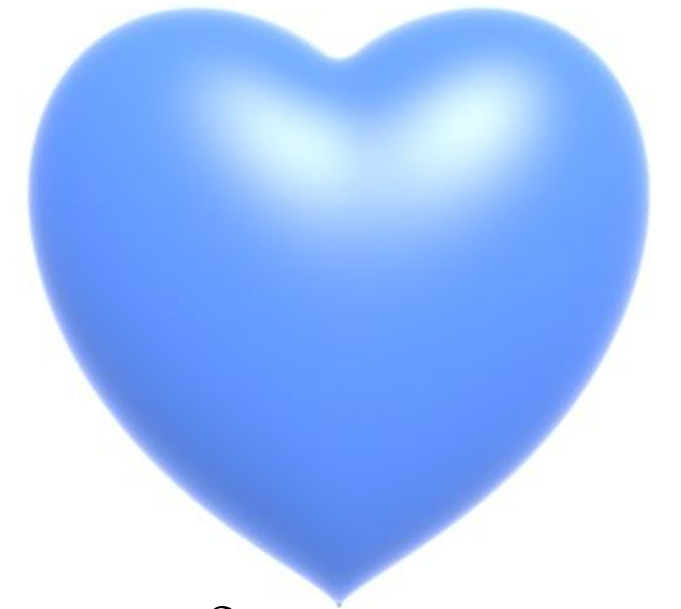
- Examples:



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



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



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

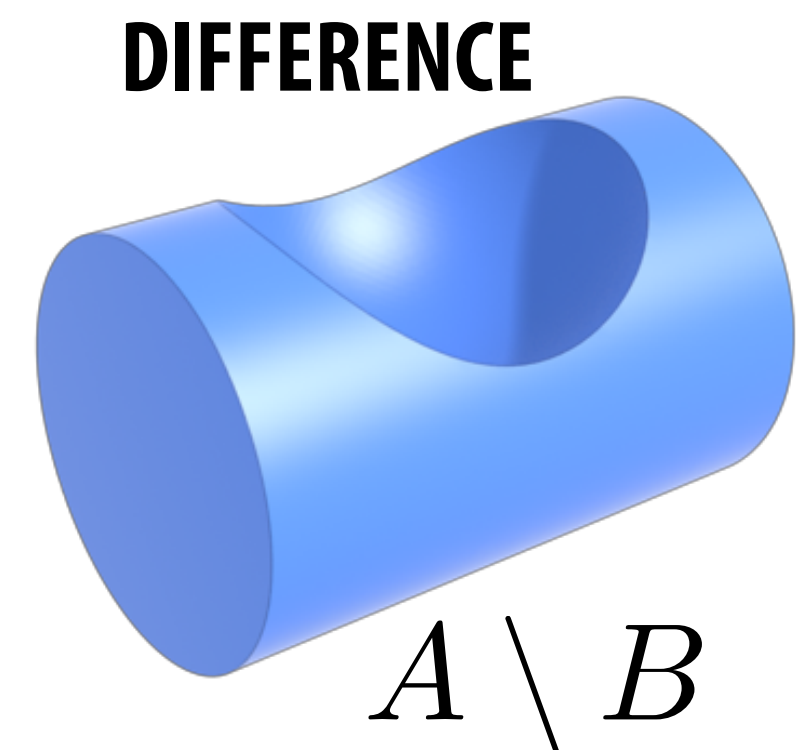
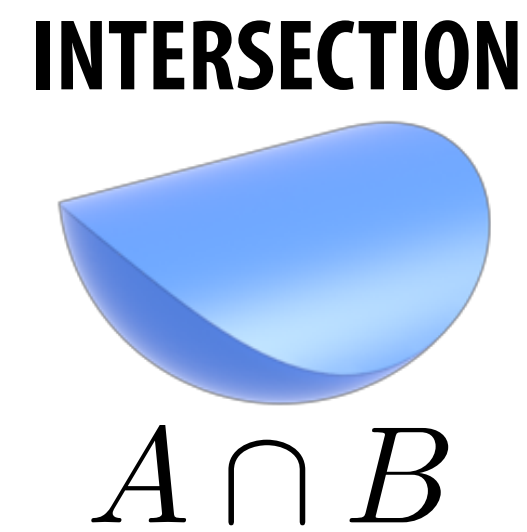
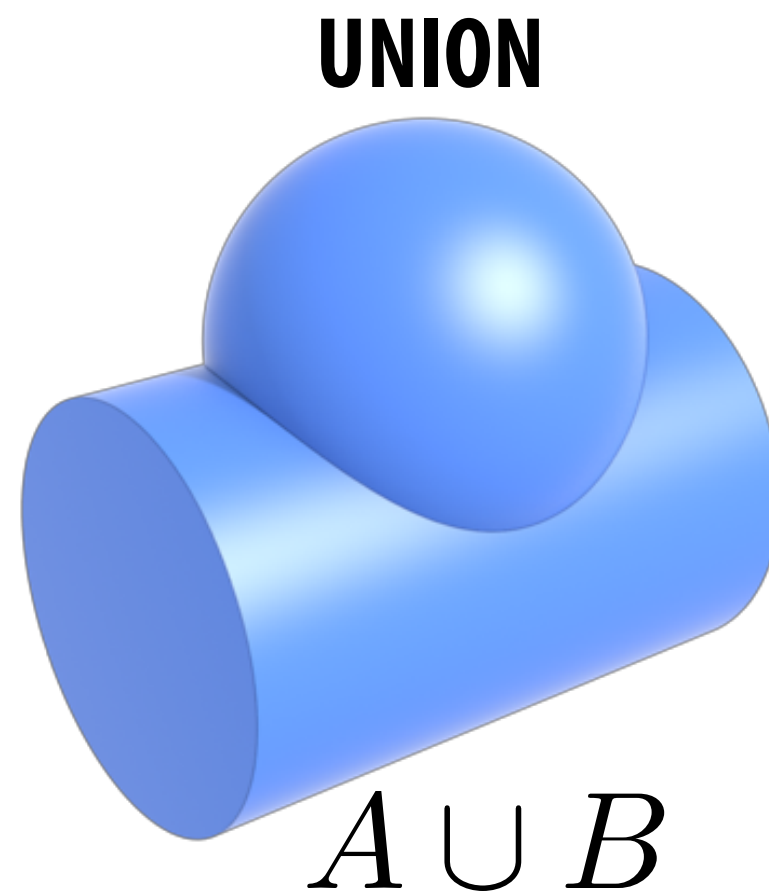
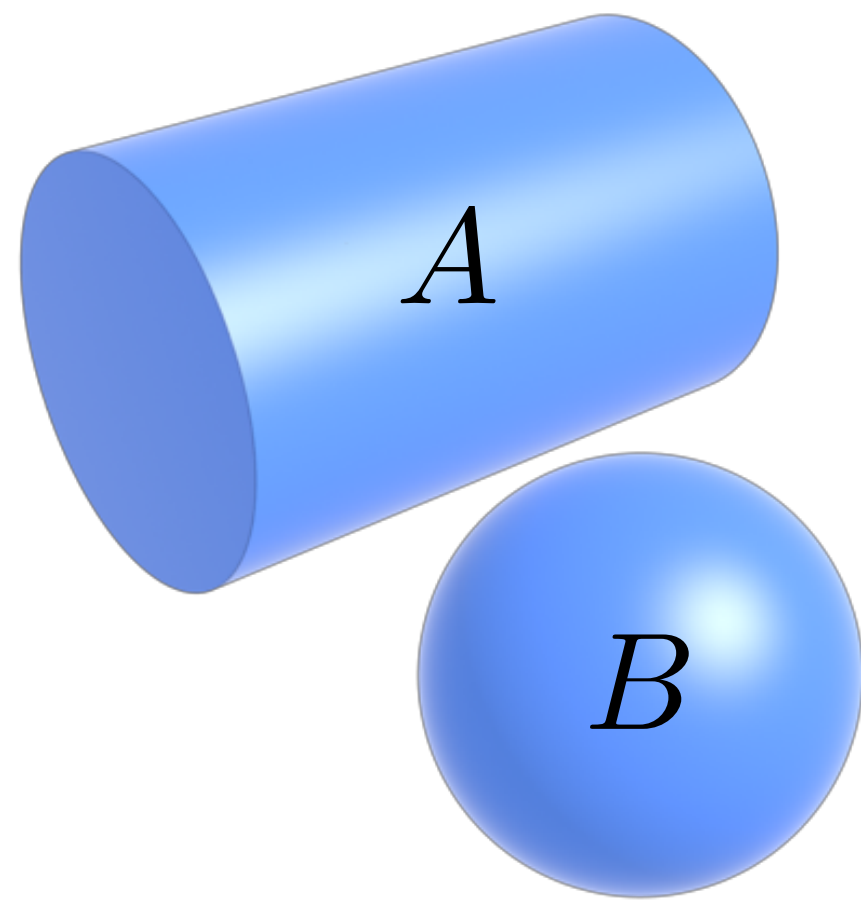
- What about more complicated shapes?



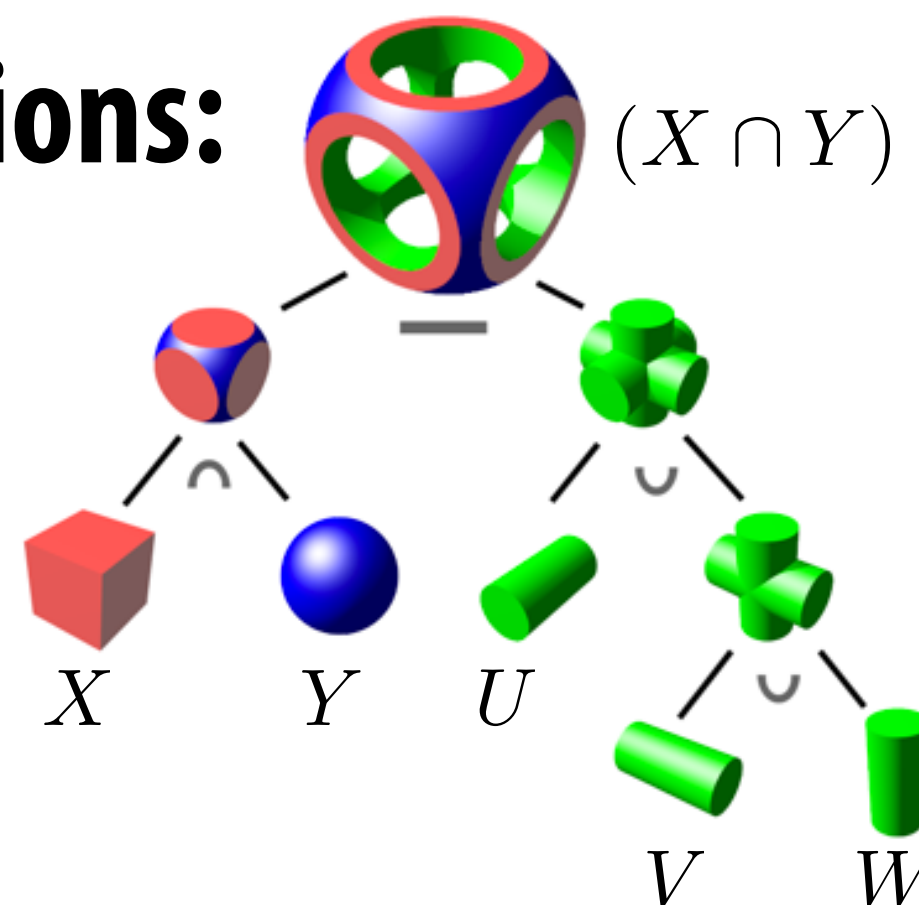
- Very hard to come up with polynomials!

Constructive Solid Geometry (Implicit)

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

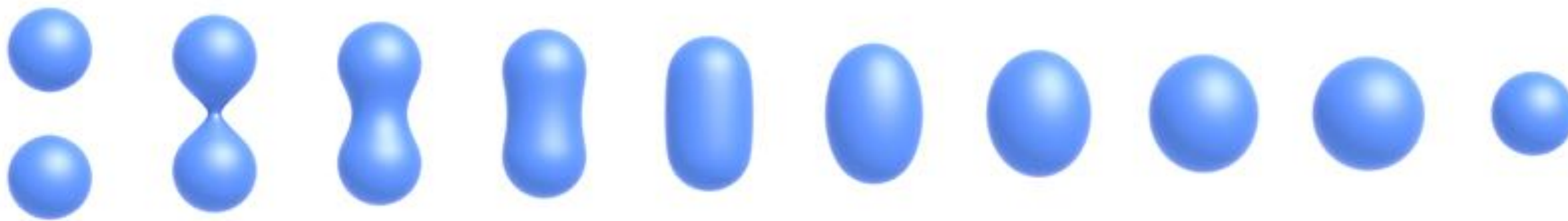


- Then chain together expressions: $(X \cap Y) \setminus (U \cup V \cup W)$



Bloppy Surfaces (Implicit)

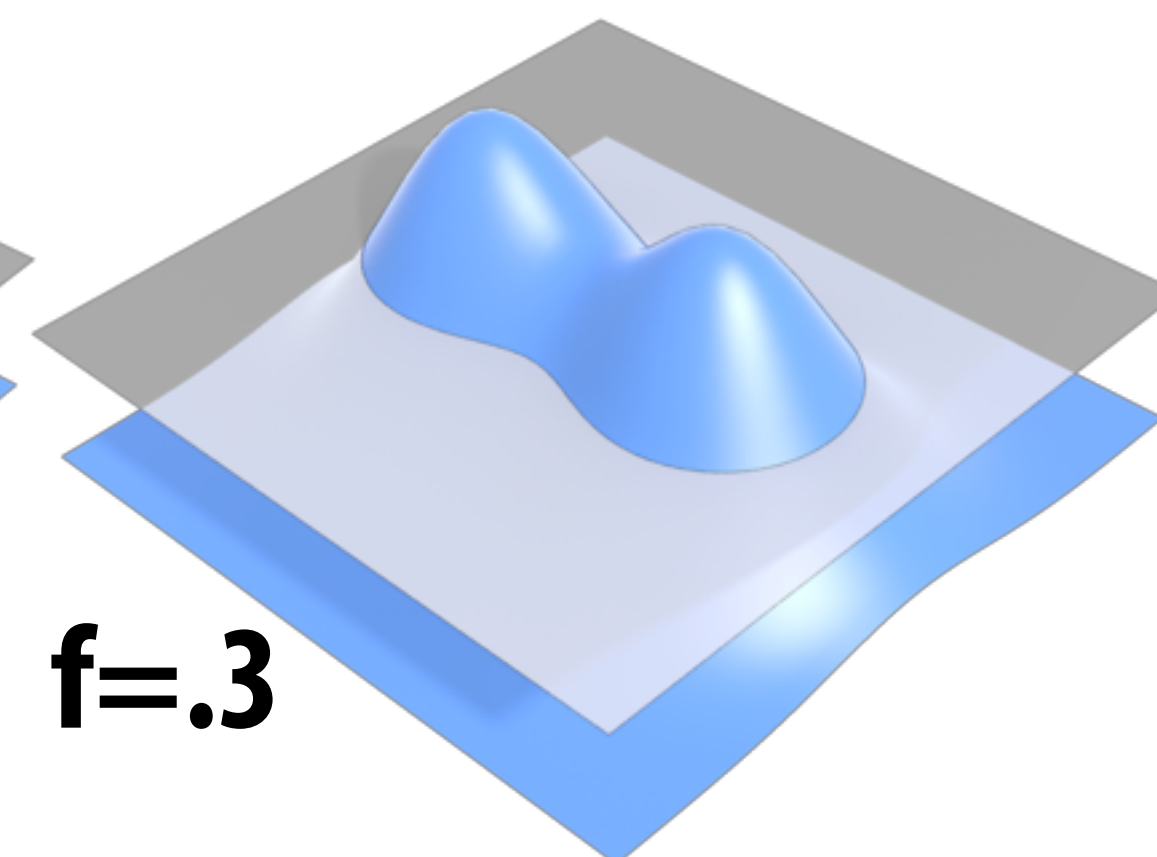
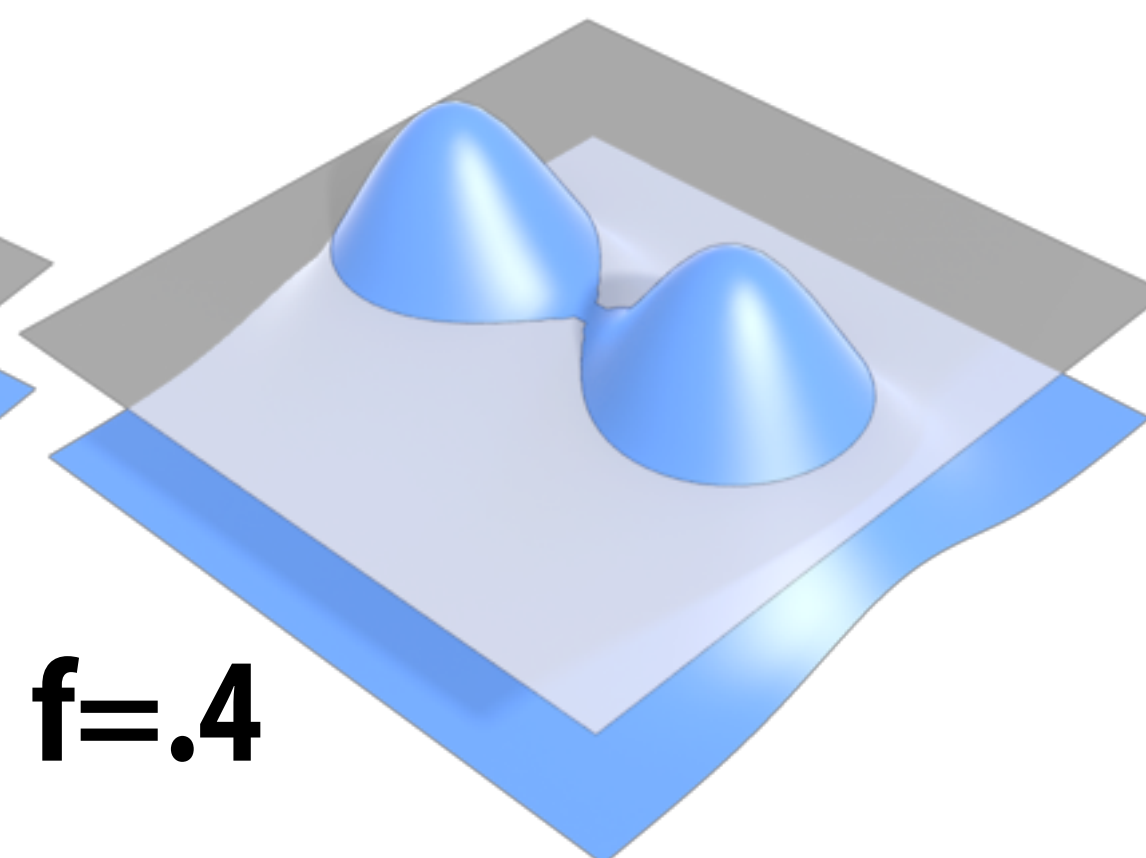
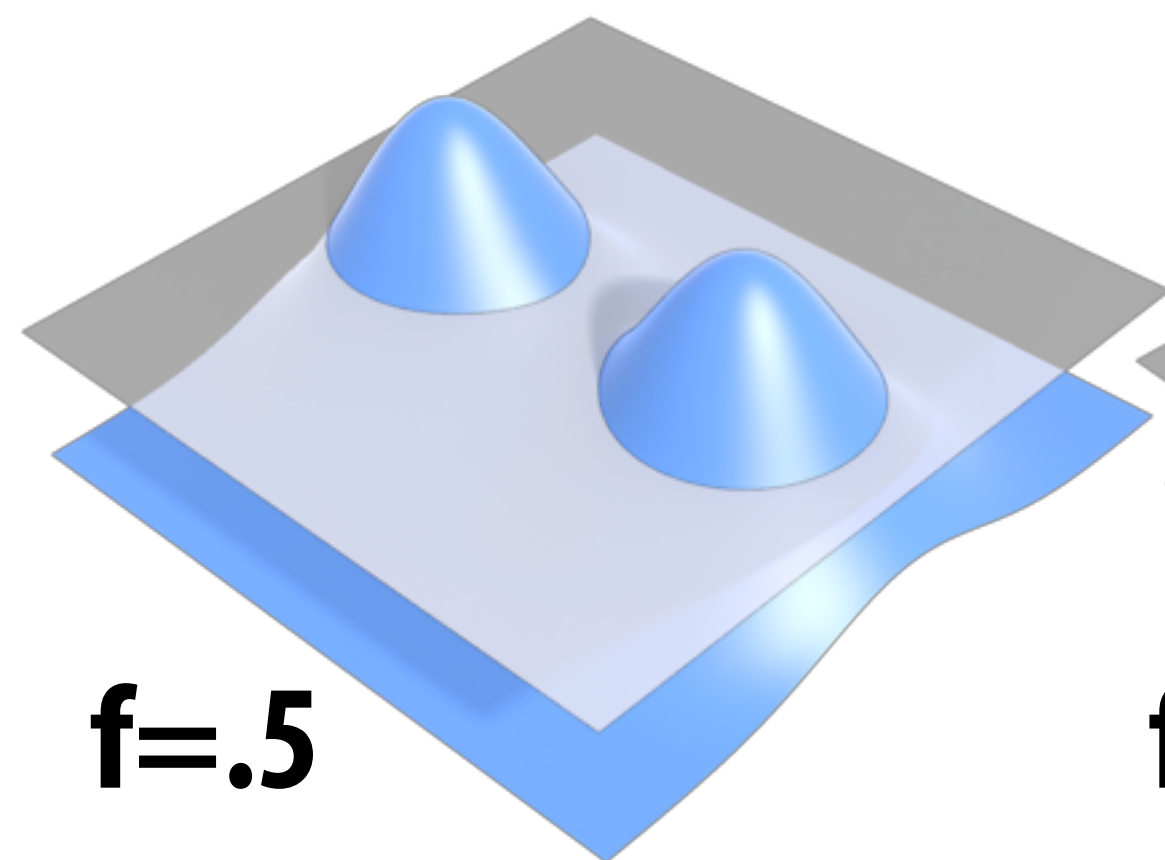
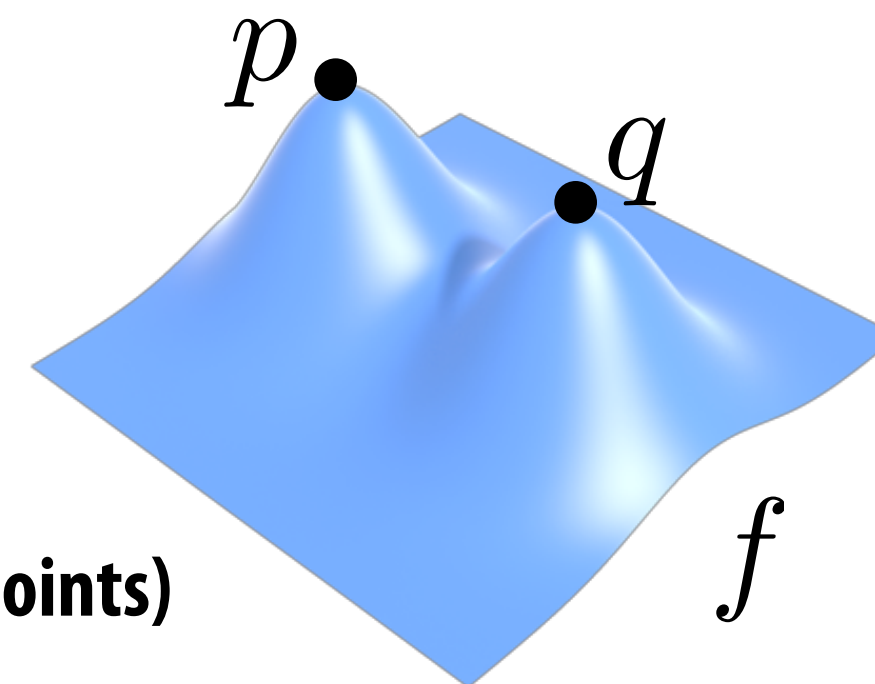
- Instead of Booleans, gradually blend surfaces together:



- Easier to understand in 2D:

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

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



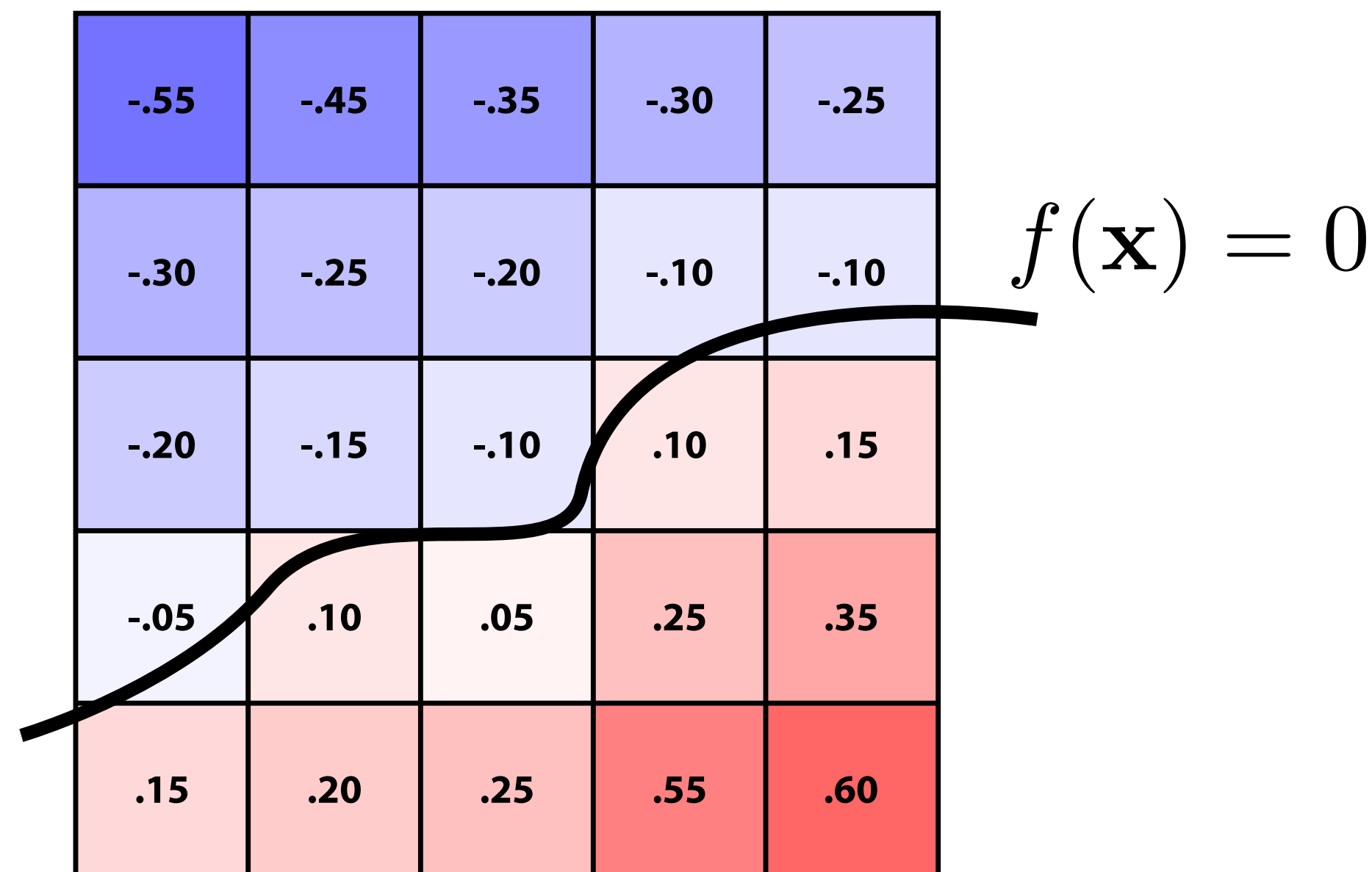
Scene using implicit functions (not easy!)



See <http://iquilezles.org/www/material/nvscene2008/nvscene2008.htm>

Level Set Methods (Implicit)

- 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
- Provides much more explicit control over shape (like a texture)
- Often demands sophisticated filtering (trilinear, tricubic...)

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

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



Implicit Representations - Pros & Cons

■ Pros:

- **description can be very compact (e.g., a polynomial)**
- **easy to determine if a point is in our shape (just plug it in!)**
- **other queries may also be easy (e.g., distance to surface)**
- **for simple shapes, exact description/no sampling error**
- **easy to handle changes in topology (e.g., fluid)**

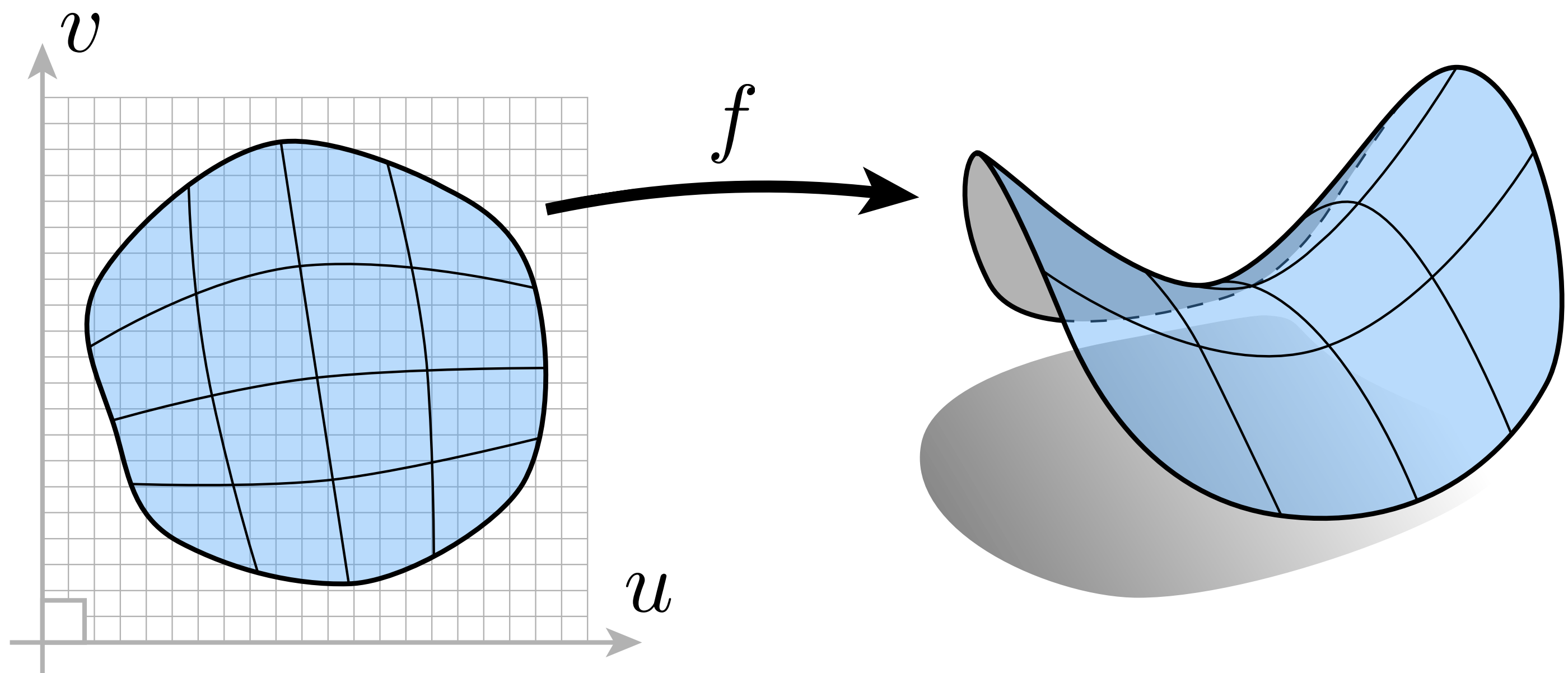
■ Cons:

- **expensive to find all points in the shape (e.g., for drawing)**
- **very difficult to model complex shapes**

What about explicit representations?

“Explicit” Representations of Geometry

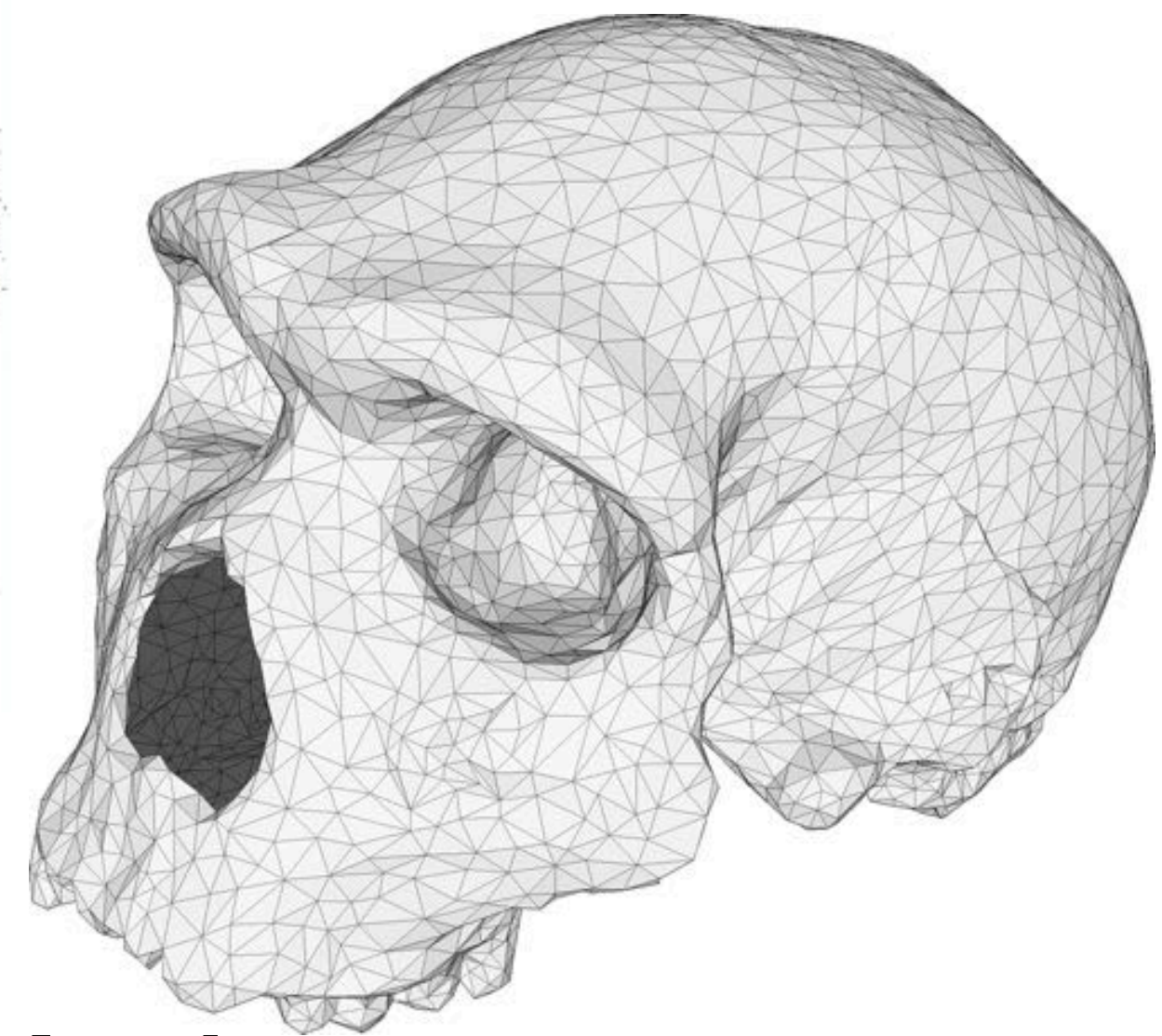
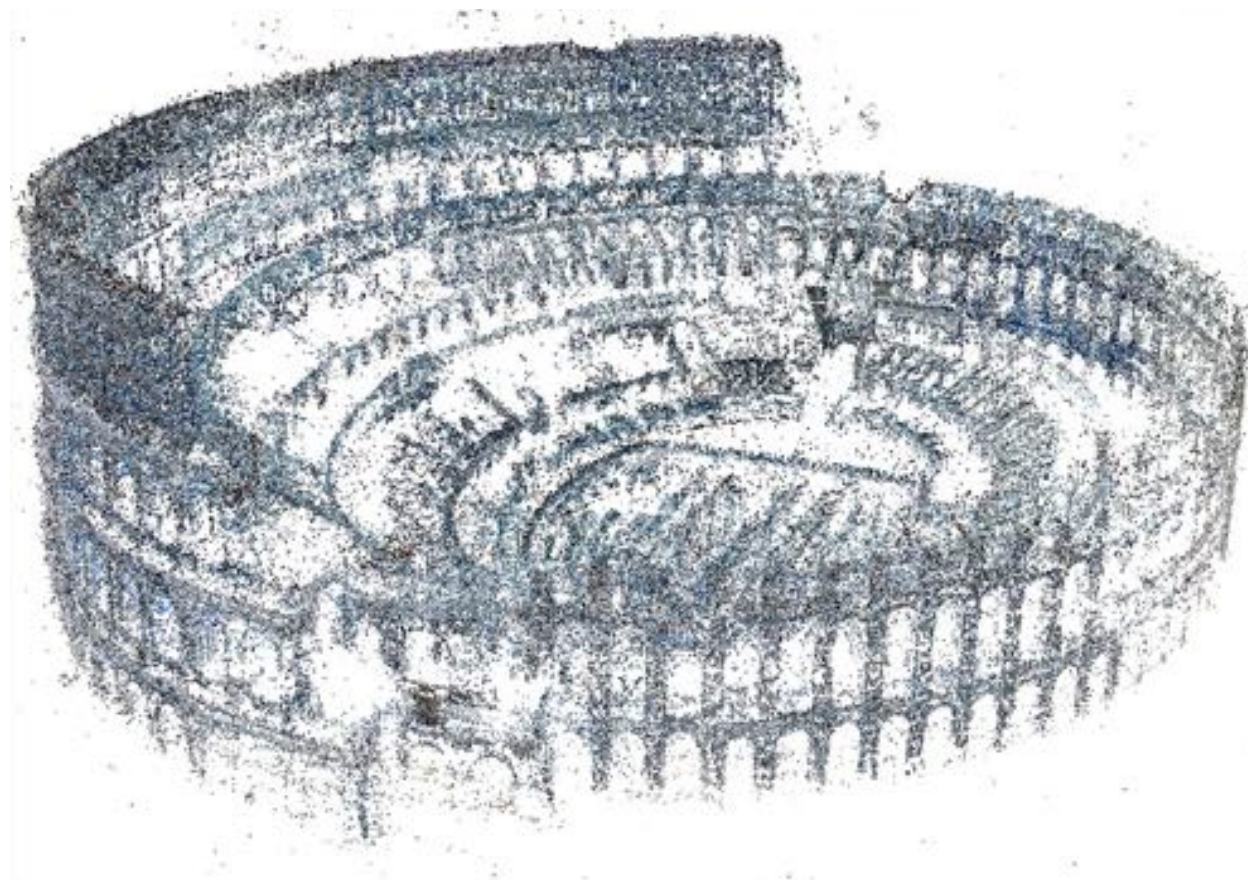
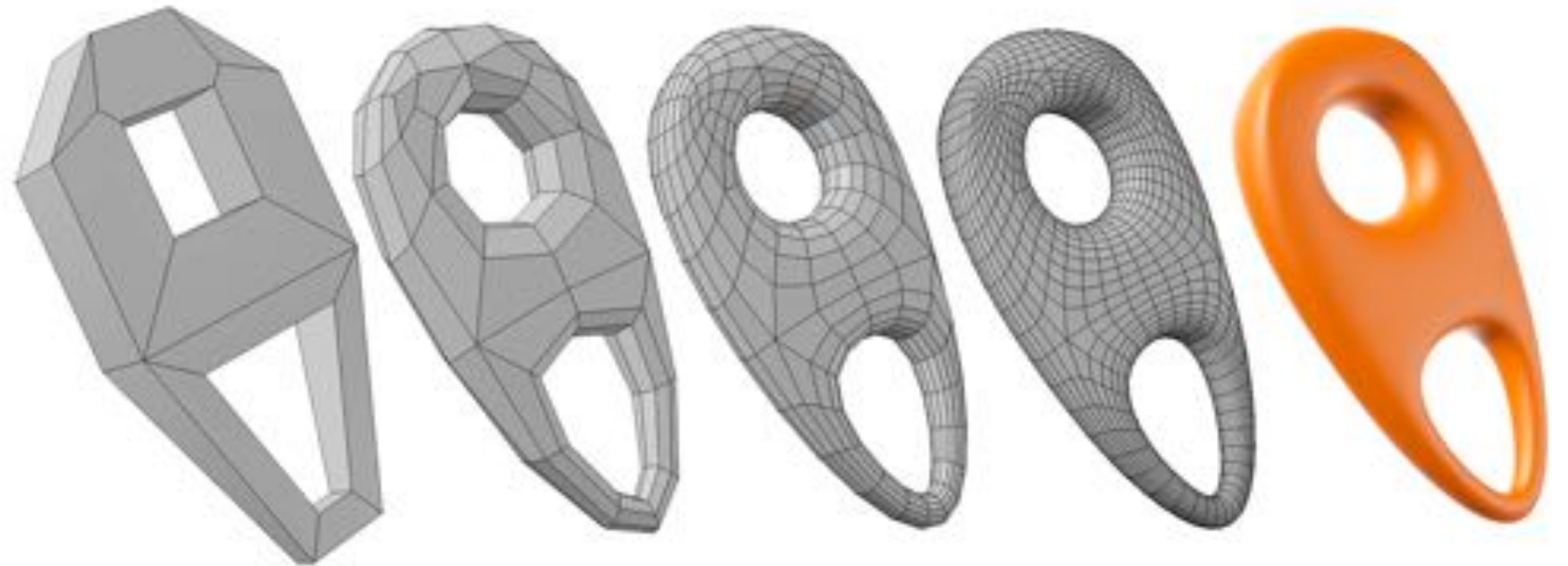
- All points are given directly
- E.g., points on sphere are $(\cos(u) \sin(v), \sin(u) \sin(v), \cos(v))$,
for $0 \leq u < 2\pi$ and $0 \leq v \leq \pi$
- More generally: $f : \mathbb{R}^2 \rightarrow \mathbb{R}^3; (u, v) \mapsto (x, y, z)$



- (Might have a bunch of these maps, e.g., one per triangle!)

Many explicit representations in graphics

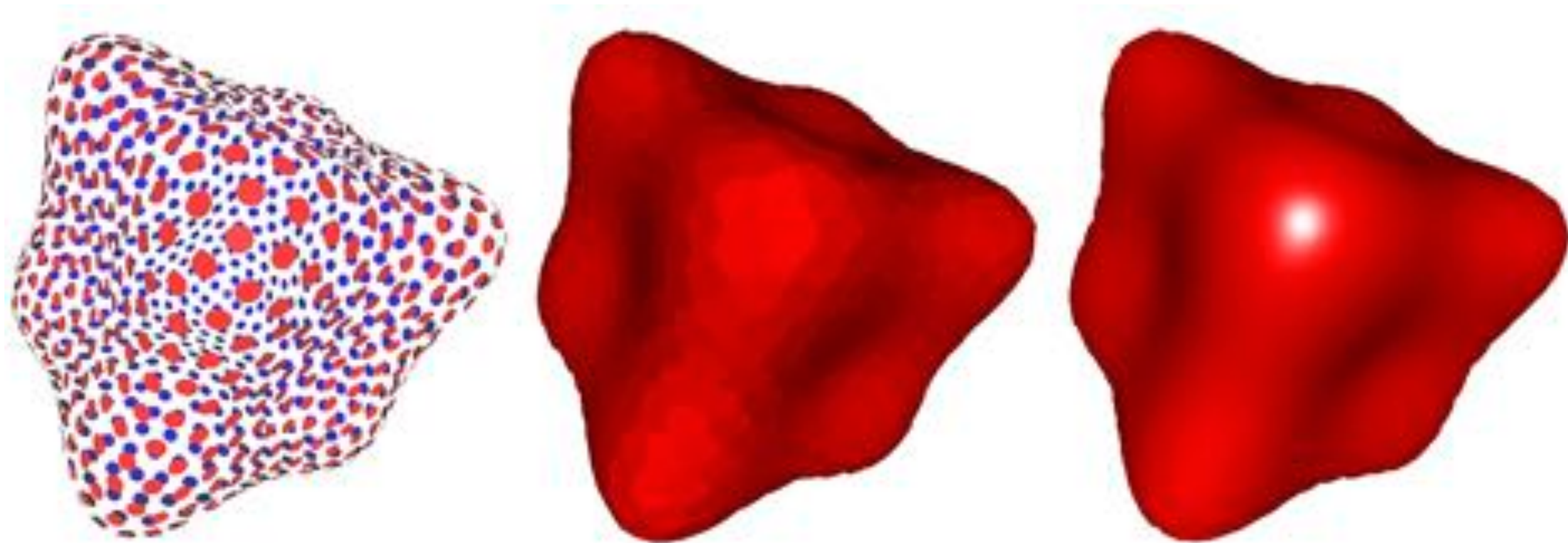
- triangle meshes
- polygon meshes
- subdivision surfaces
- NURBS
- point clouds
- ...



(Will see some of these a bit later.)

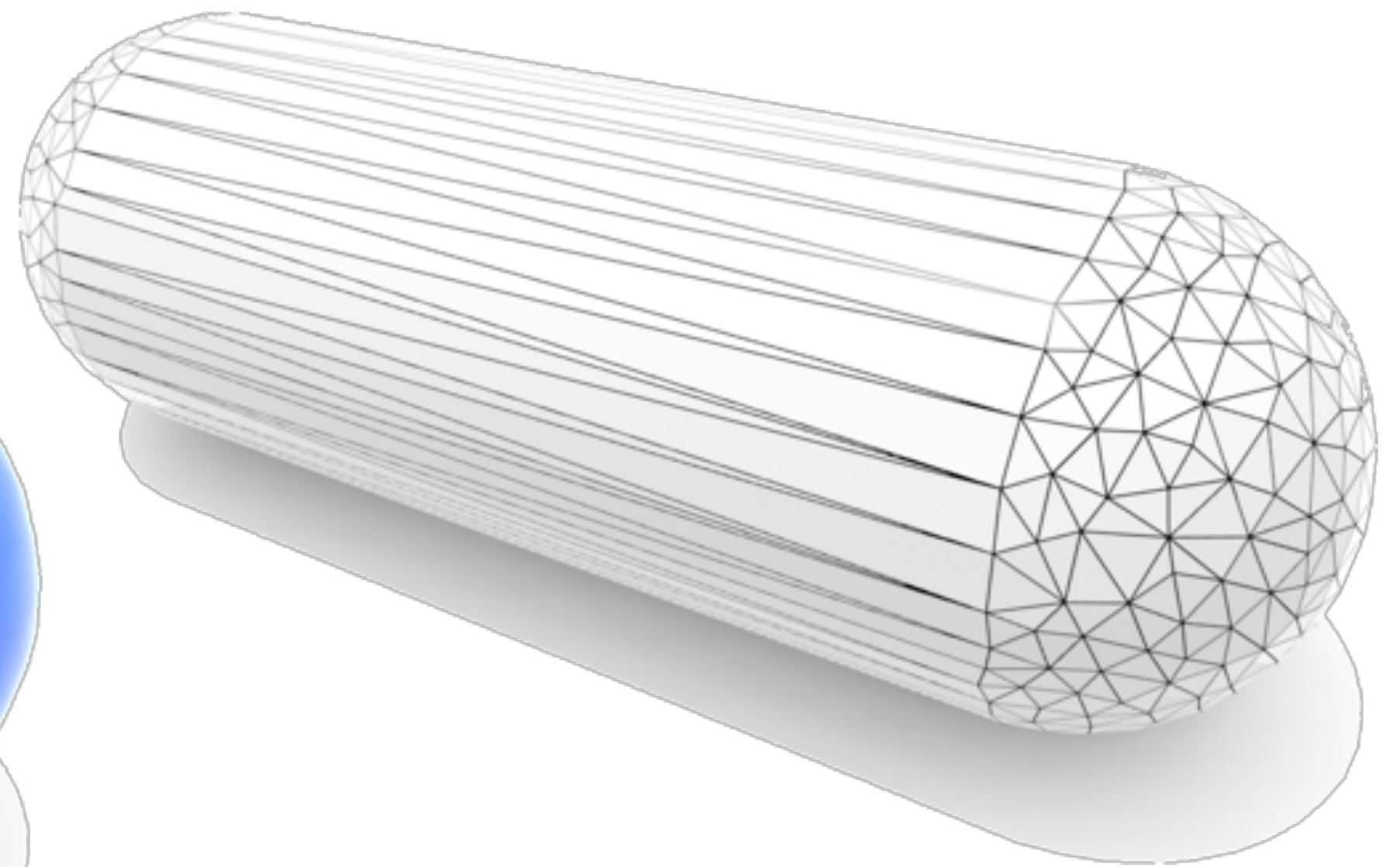
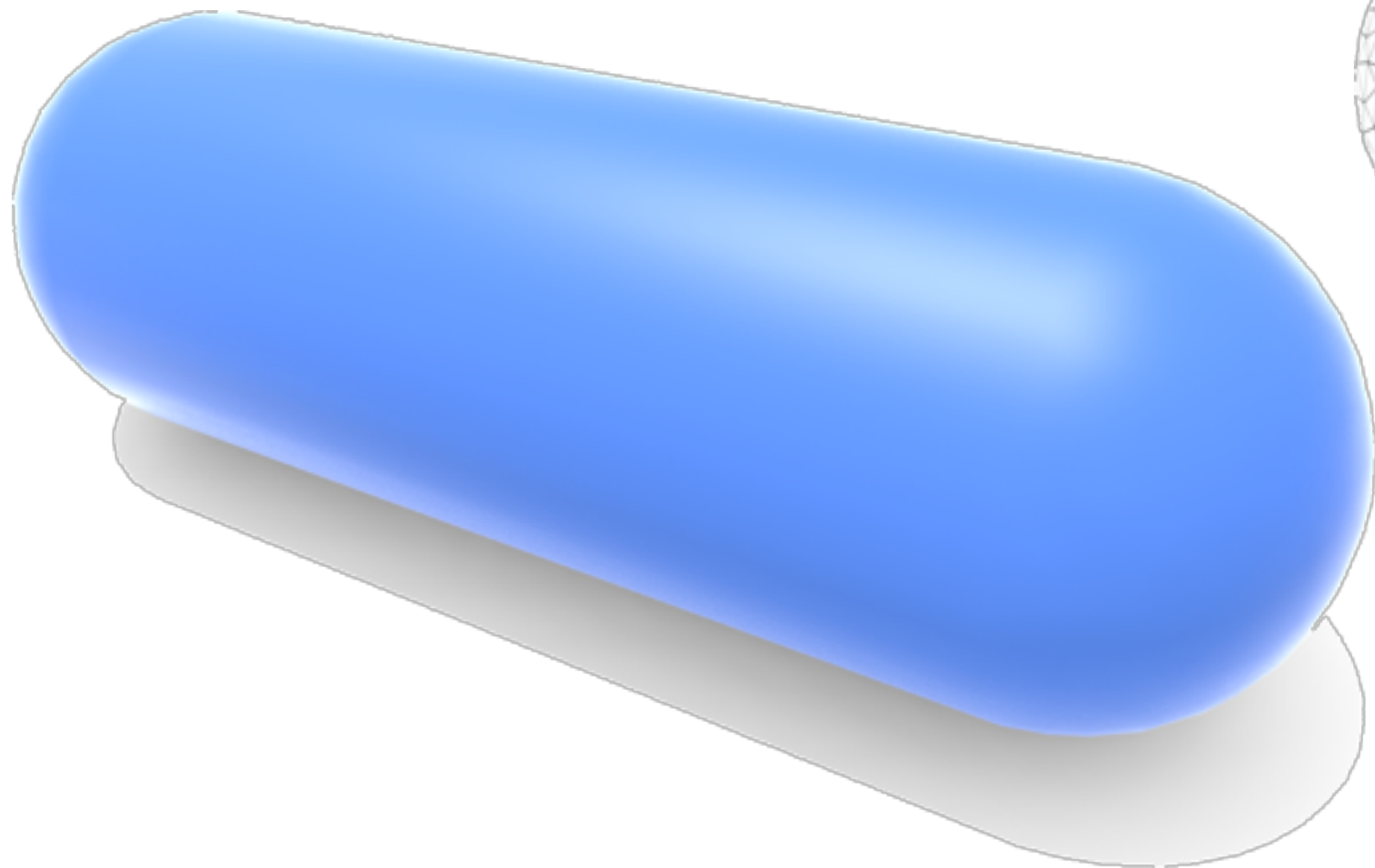
Point Cloud (Explicit)

- Easiest representation: list of points (x,y,z)
- Often augmented with normals
- Easily represent any kind of geometry
- Useful for LARGE datasets ($\gg 1$ point/pixel)
- Hard to interpolate undersampled regions
- Hard to do processing / simulation / ...



Polygon Mesh (Explicit)

- **Store vertices and polygons (most often triangles or quads)**
- **Easier to do processing/simulation, adaptive sampling**
- **More complicated data structures**
- **Perhaps most common representation in graphics**



(Much more about polygon meshes in upcoming lectures!)

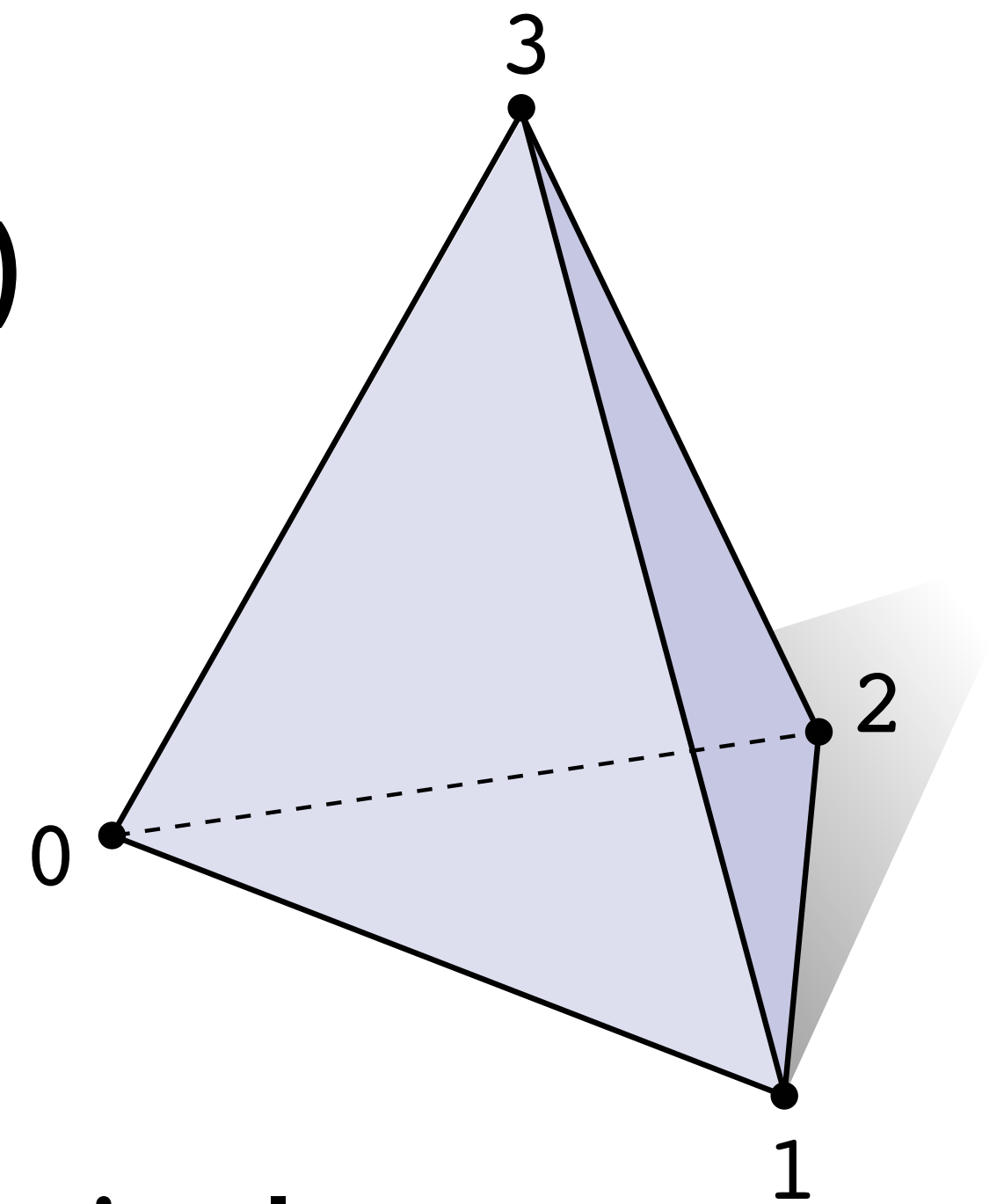
Triangle Mesh (Explicit)

- Store vertices as triples of coordinates (x,y,z)

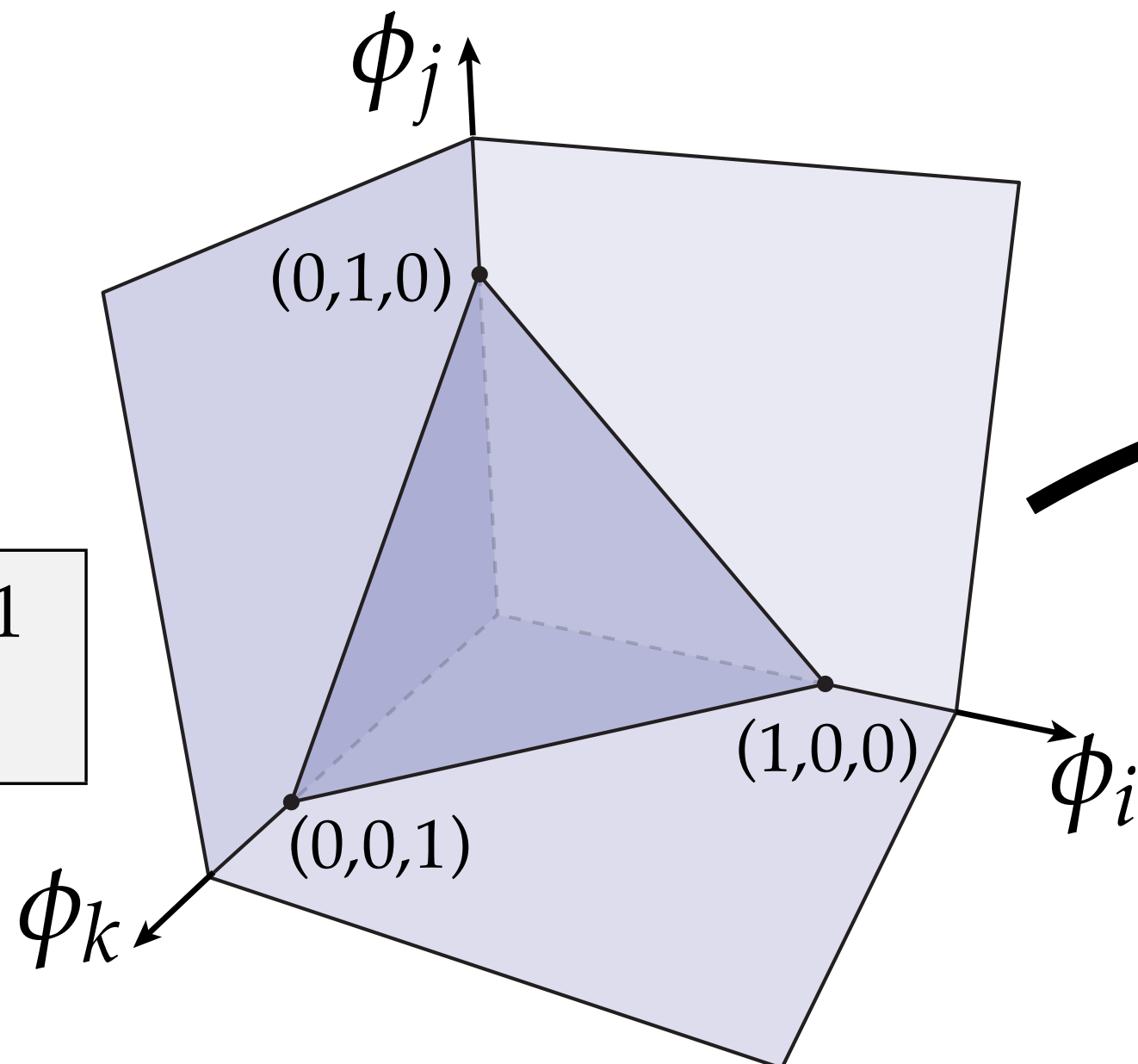
- Store triangles as triples of indices (i,j,k)

- E.g., tetrahedron:

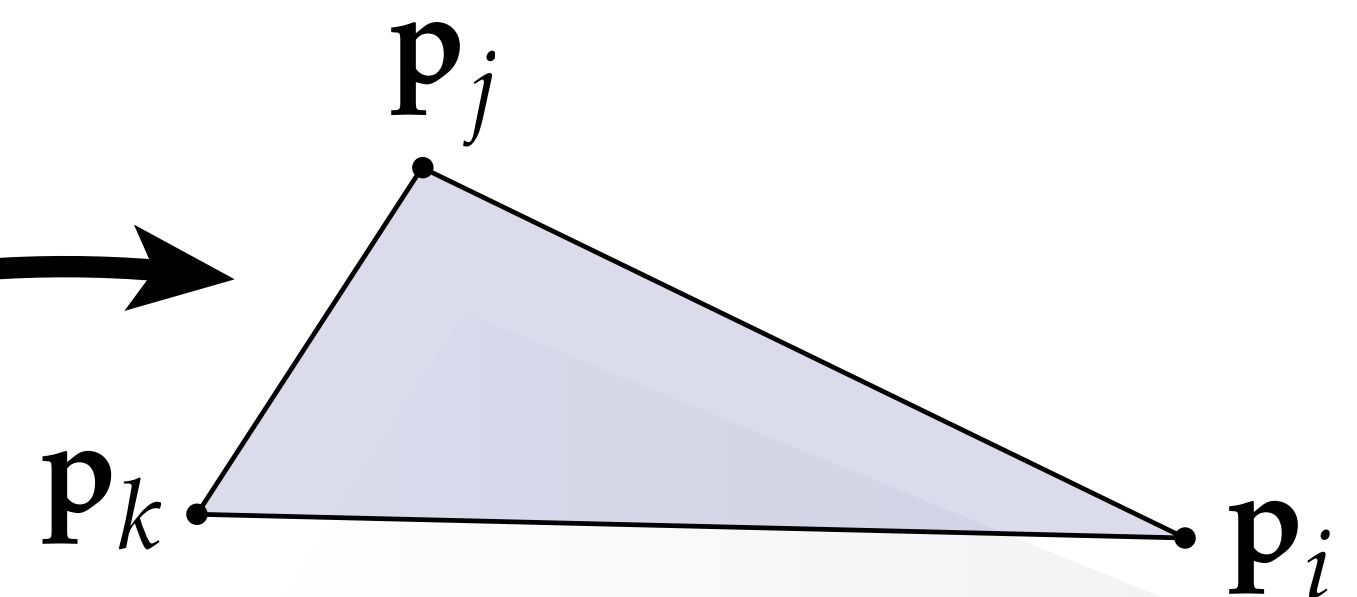
	VERTICES			TRIANGLES		
	x	y	z	i	j	k
0:	-1	-1	-1	0	2	1
1:	1	-1	1	0	3	2
2:	1	1	-1	3	0	1
3:	-1	1	1	3	1	2



- Use barycentric interpolation to define points inside triangles:



$$\begin{aligned} \phi_i + \phi_j + \phi_k &= 1 \\ \phi_i, \phi_j, \phi_k &> 0 \end{aligned}$$



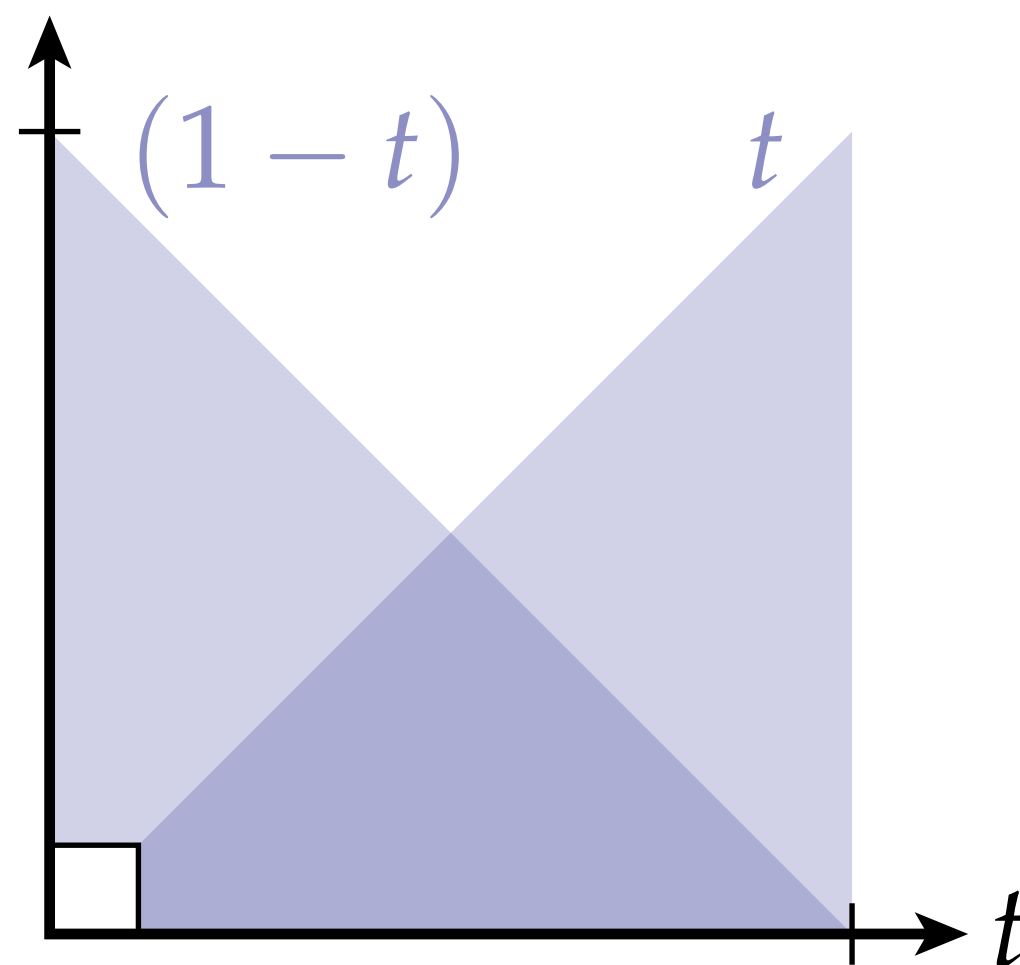
$$p = \phi_i \mathbf{p}_i + \phi_j \mathbf{p}_j + \phi_k \mathbf{p}_k$$

Recall: Linear Interpolation (1D)

- Interpolate vertex positions using linear interpolation; in 1D:

$$\hat{f}(t) = (1 - t)f_i + tf_j$$

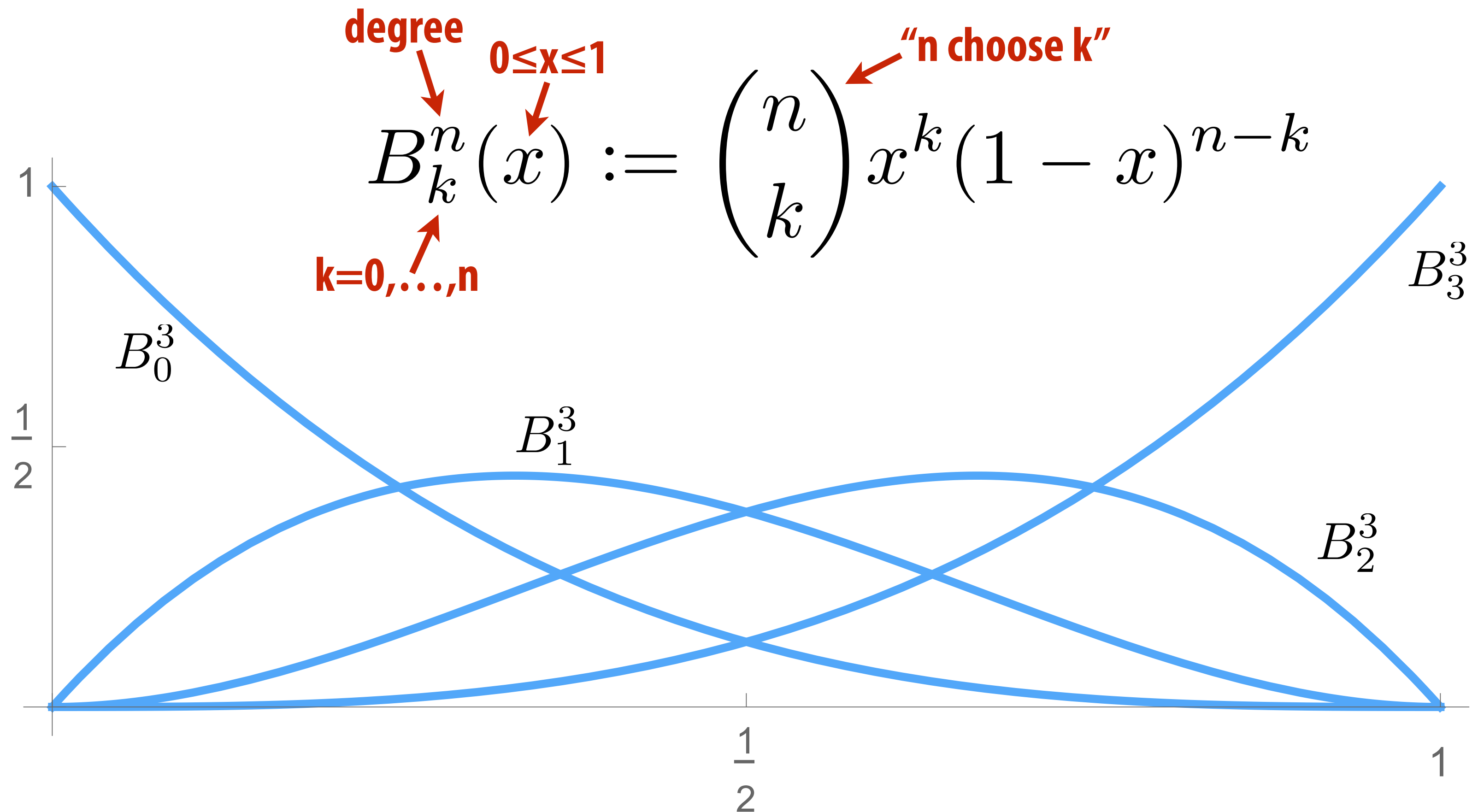
- Can think of this as a linear combination of two functions:



- As we move closer to $t=0$, we approach the value of f at x_i
- As we move closer to $t=1$, we approach the value of f at x_j

Bernstein Basis

- Why limit ourselves to just linear interpolation?
- More flexibility by using higher-order polynomials
- Instead of usual basis $(1, x, x^2, x^3, \dots)$, use Bernstein basis:

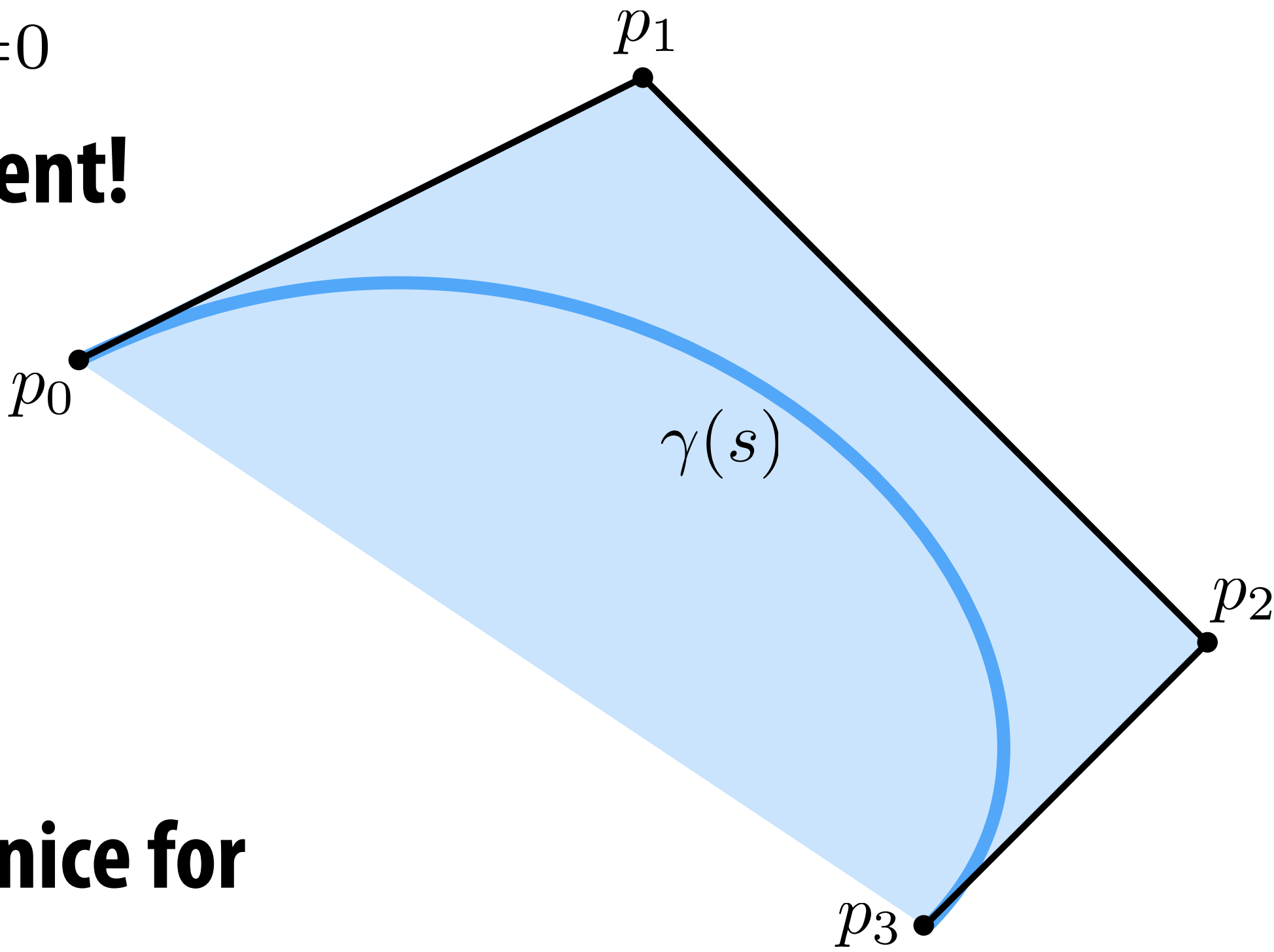


Bézier Curves (Explicit)

- A Bézier curve is a curve expressed in the Bernstein basis:

$$\gamma(s) := \sum_{k=0}^n B_{n,k}(s) p_k$$

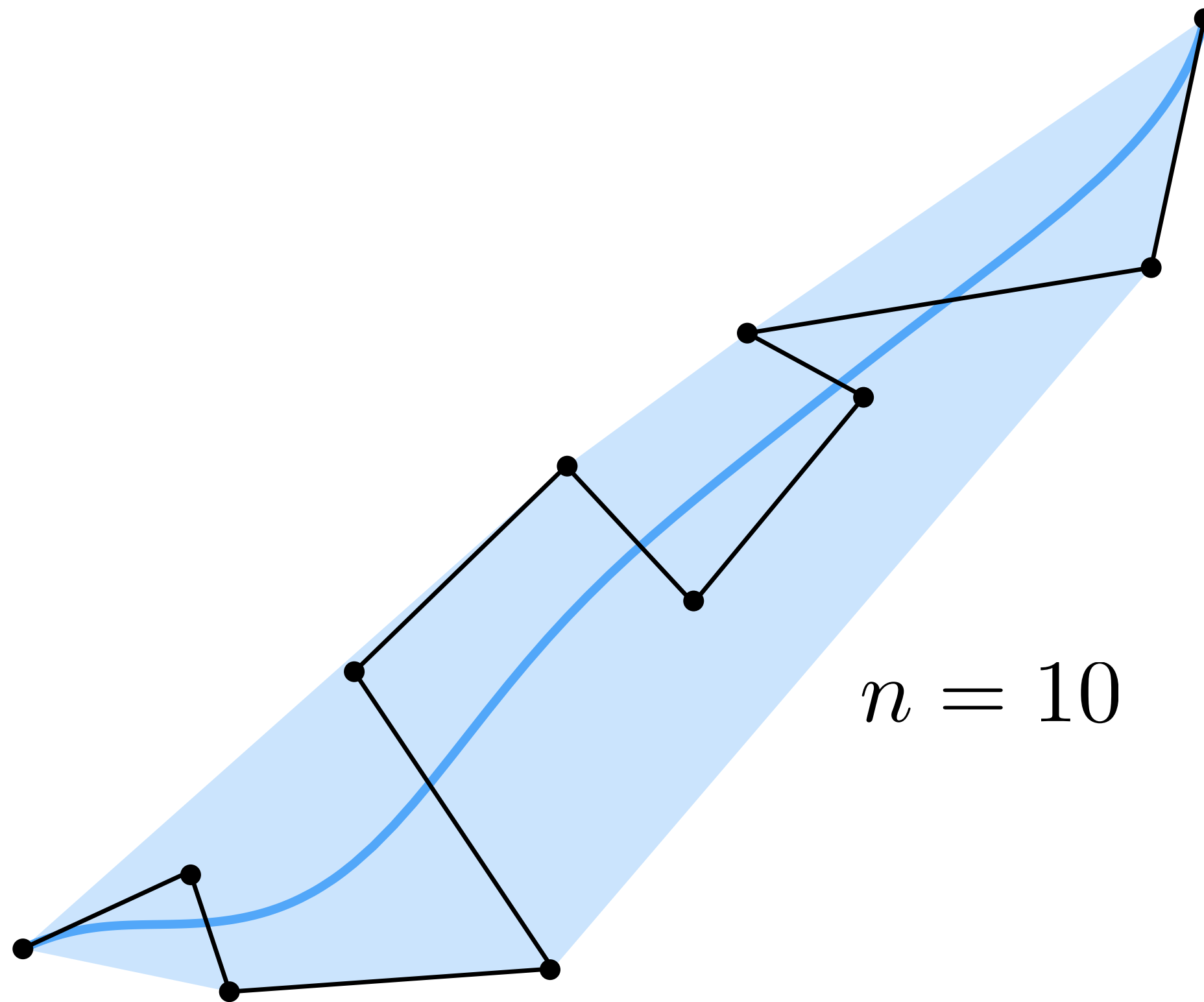
control points



- For $n=1$, just get a line segment!
- For $n=3$, get “cubic Bézier”:
- Important features:
 1. interpolates endpoints
 2. tangent to end segments
 3. contained in convex hull (nice for rasterization)

Just keep going...?

- What if we want an even more interesting curve?
- High-degree Bernstein polynomials don't interpolate well:



Very hard to control!

Piecewise Bézier Curves (Explicit)

- Alternative idea: piece together many Bézier curves
- Widely-used technique (Illustrator, fonts, SVG, etc.)



- Formally, piecewise Bézier curve:

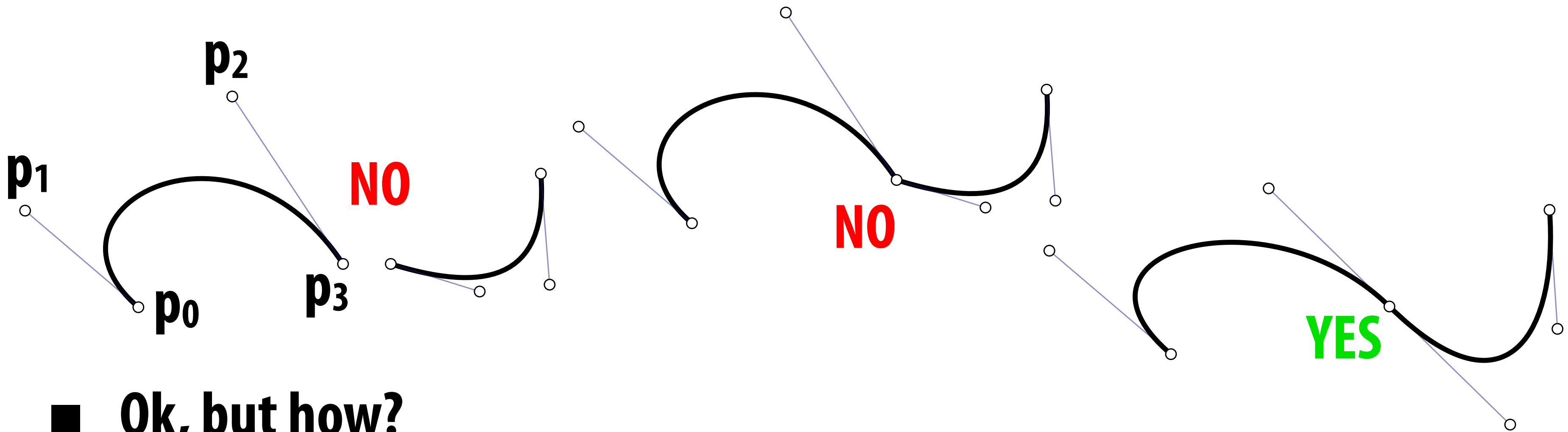
piecewise Bézier

$$\gamma(u) := \gamma_i \left(\frac{u - u_i}{u_{i+1} - u_i} \right), \quad u_i \leq u < u_{i+1}$$

single Bézier

Bézier Curves — tangent continuity

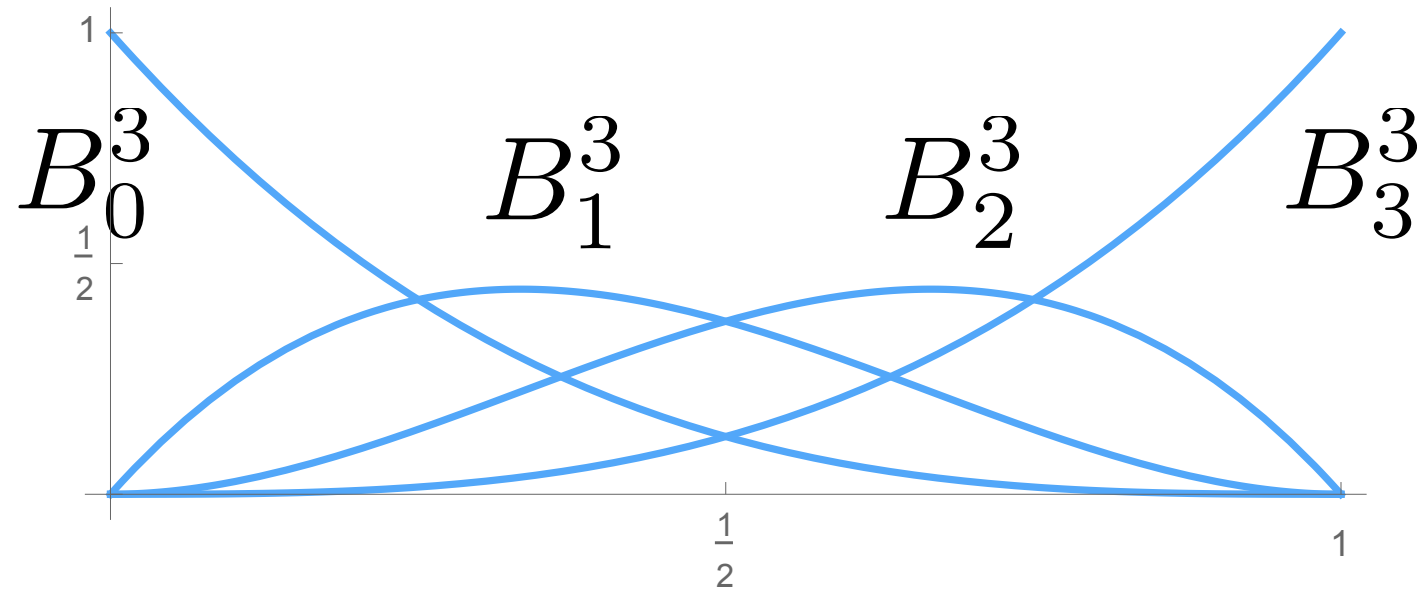
- To get “seamless” curves, need points and tangents to line up:



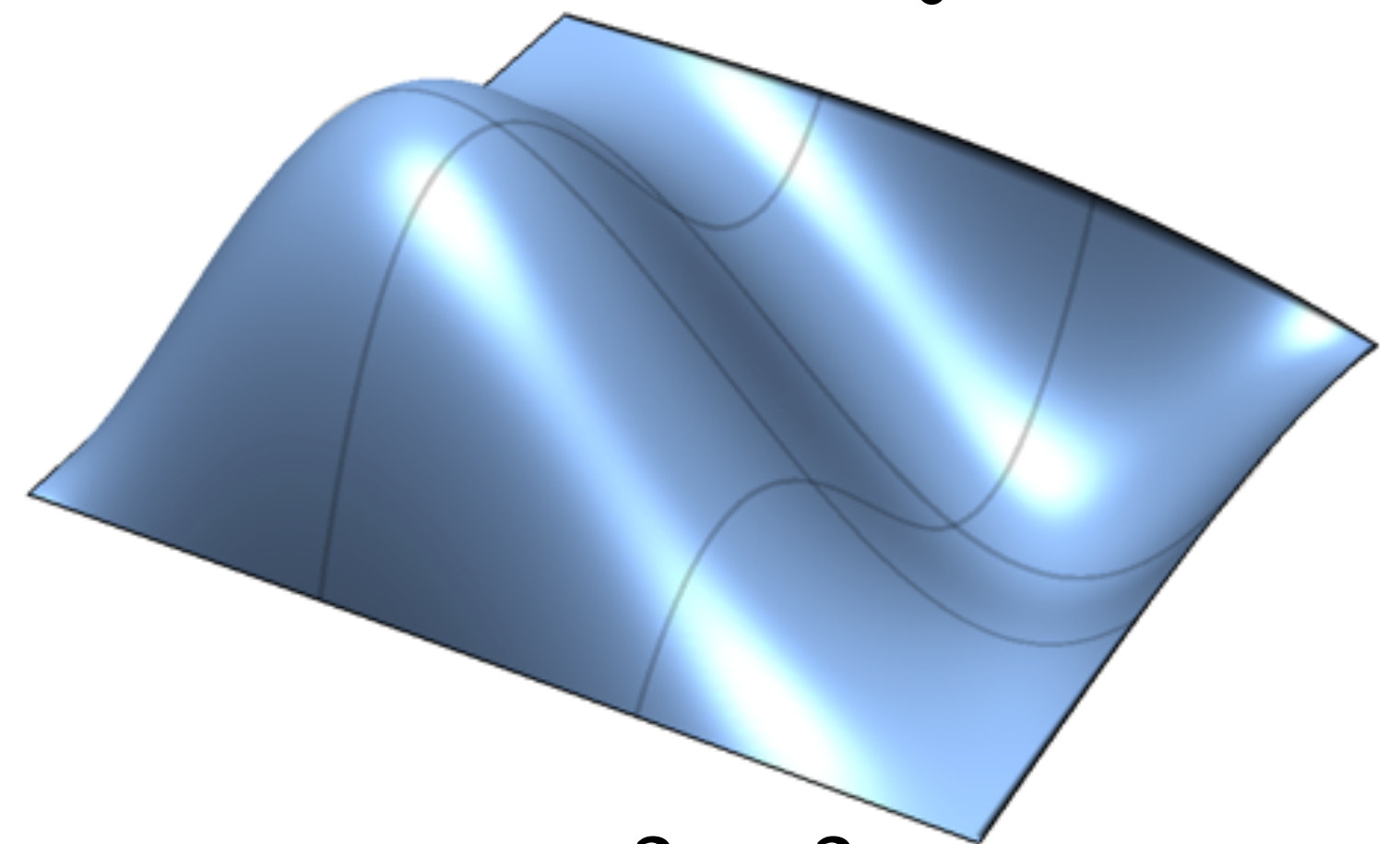
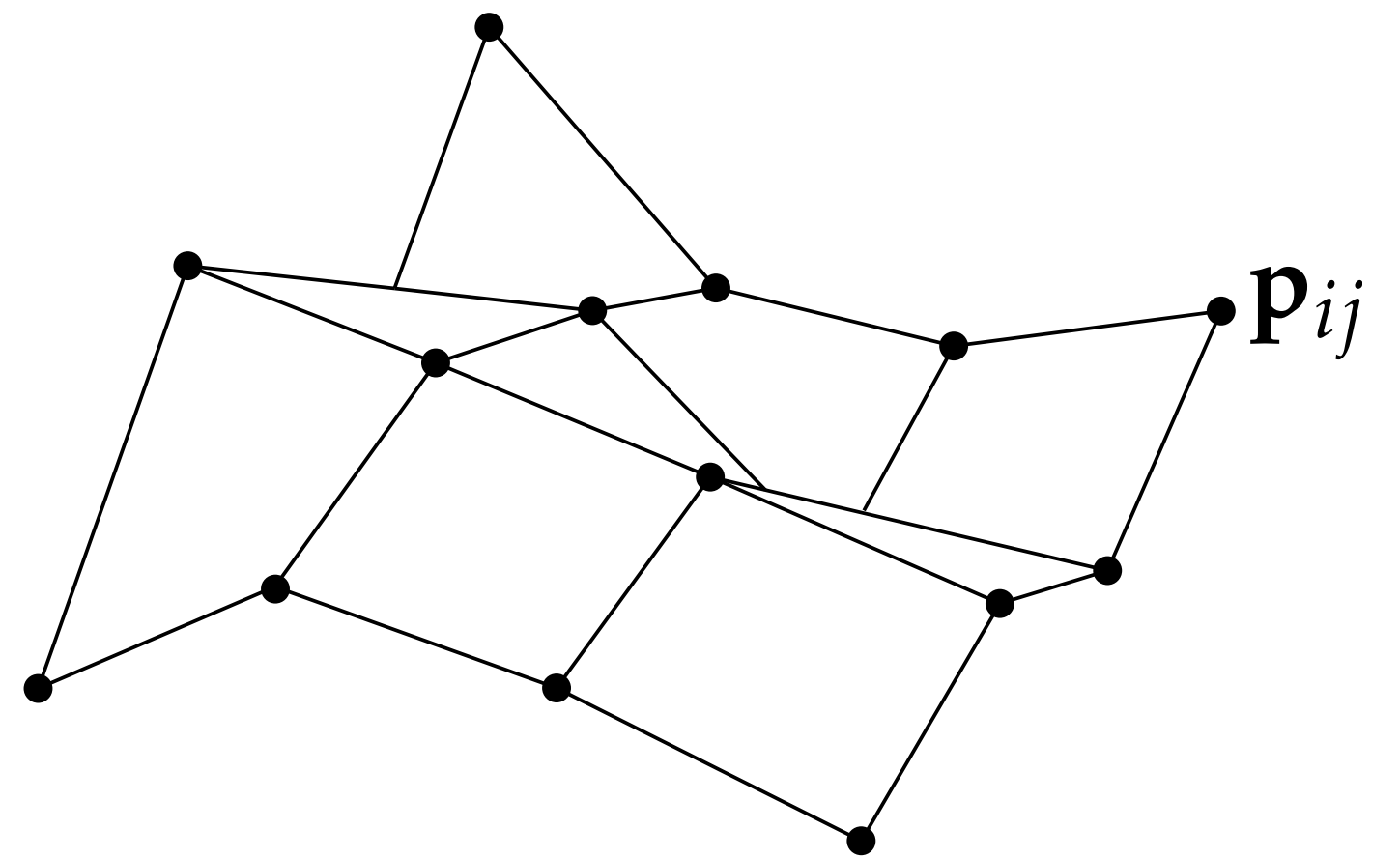
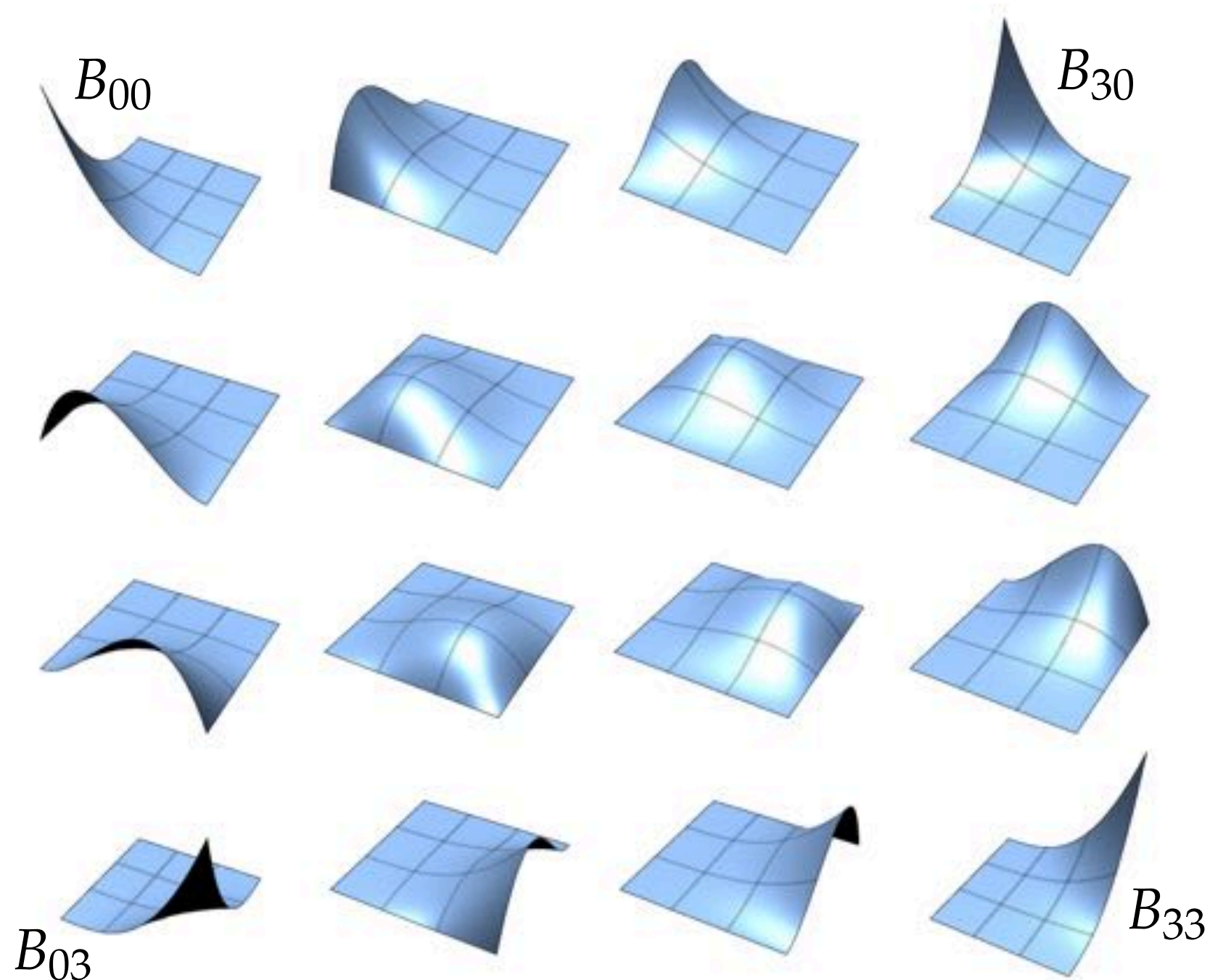
- Ok, but how?
- Each curve is cubic: $u^3p_0 + 3u^2(1-u)p_1 + 3u(1-u)^2p_2 + (1-u)^3p_3$
- Want endpoints of each segment to meet
- Want tangents at endpoints to meet
- Q: How many constraints vs. degrees of freedom?
- Q: Could you do this with quadratic Bézier? Linear Bézier?

Bézier Patches

- Bézier patch is sum of products of Bernstein bases



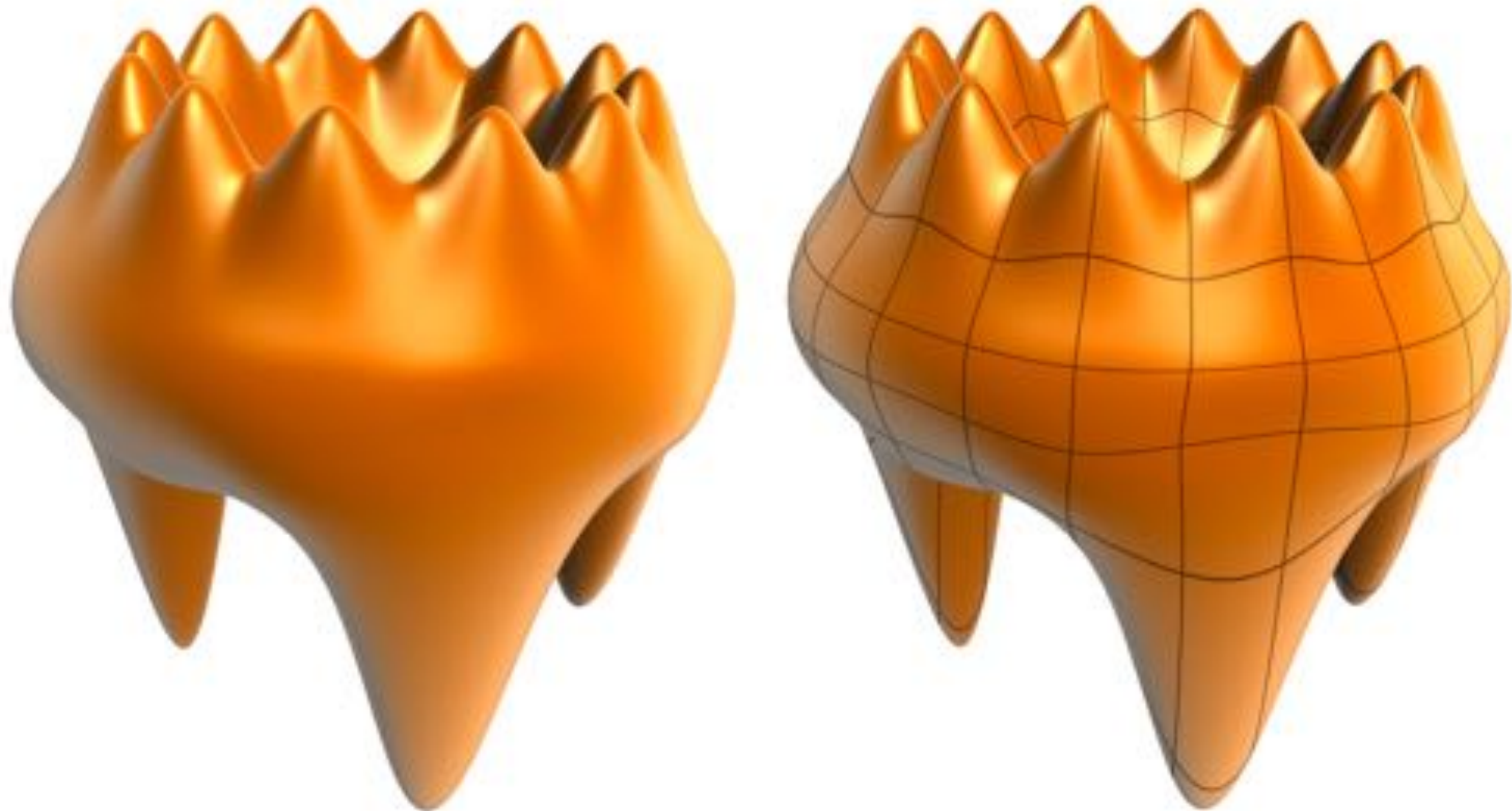
$$B_{i,j}^3(u, v) := B_i^3(u) B_j^3(v)$$



$$S(u, v) := \sum_{i=0}^3 \sum_{j=0}^3 B_{i,j}^3(u, v) \mathbf{p}_{ij}$$

Bézier Surface

- Just as we connected Bézier curves, can connect Bézier patches to get a surface:



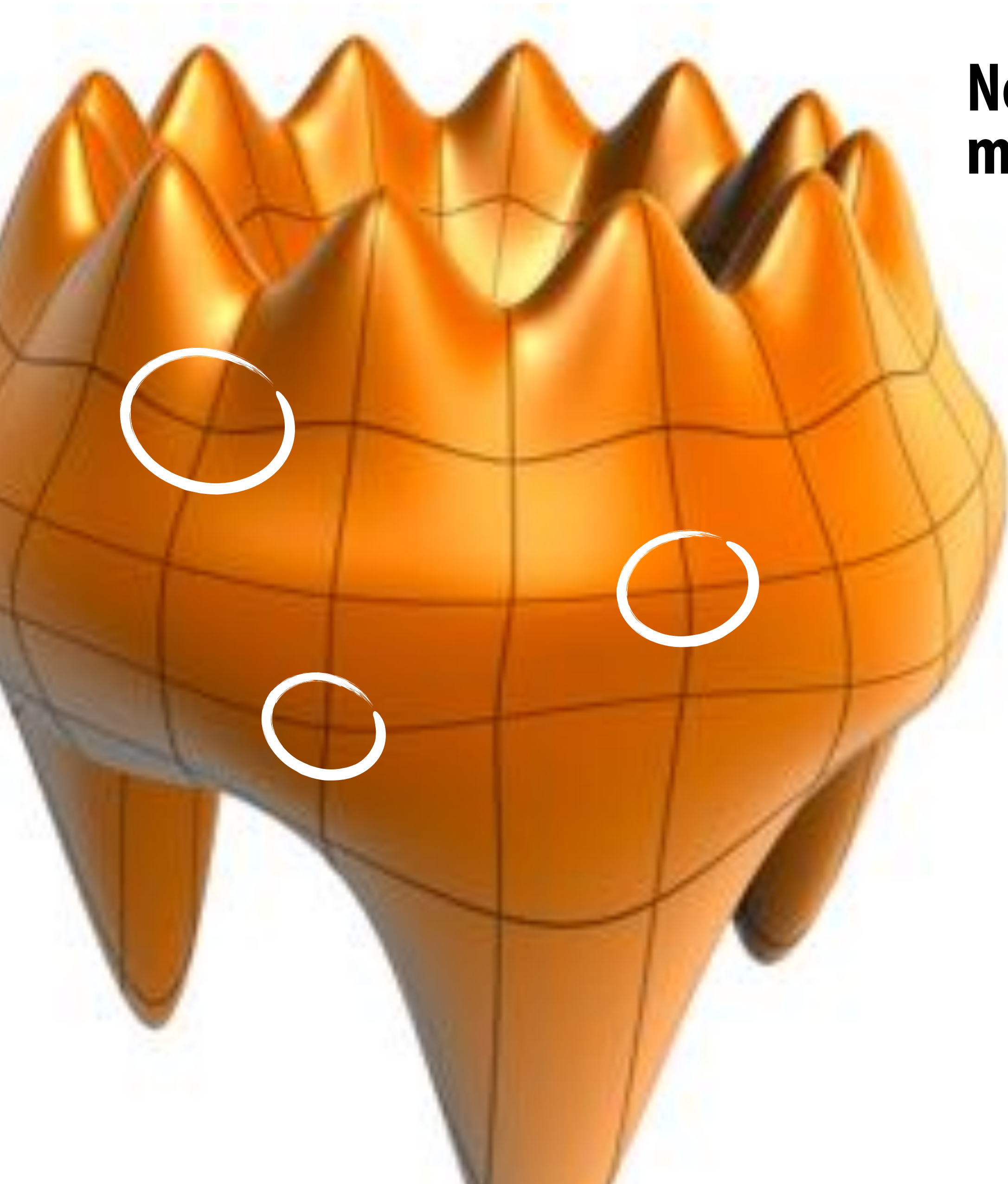
- Very easy to draw: just dice each patch into regular (u,v) grid!

Q: Can we always get tangent continuity?

(Think: how many constraints? How many degrees of freedom?)

**Notice anything fishy
about the last picture?**

Bézier Patches are Too Simple



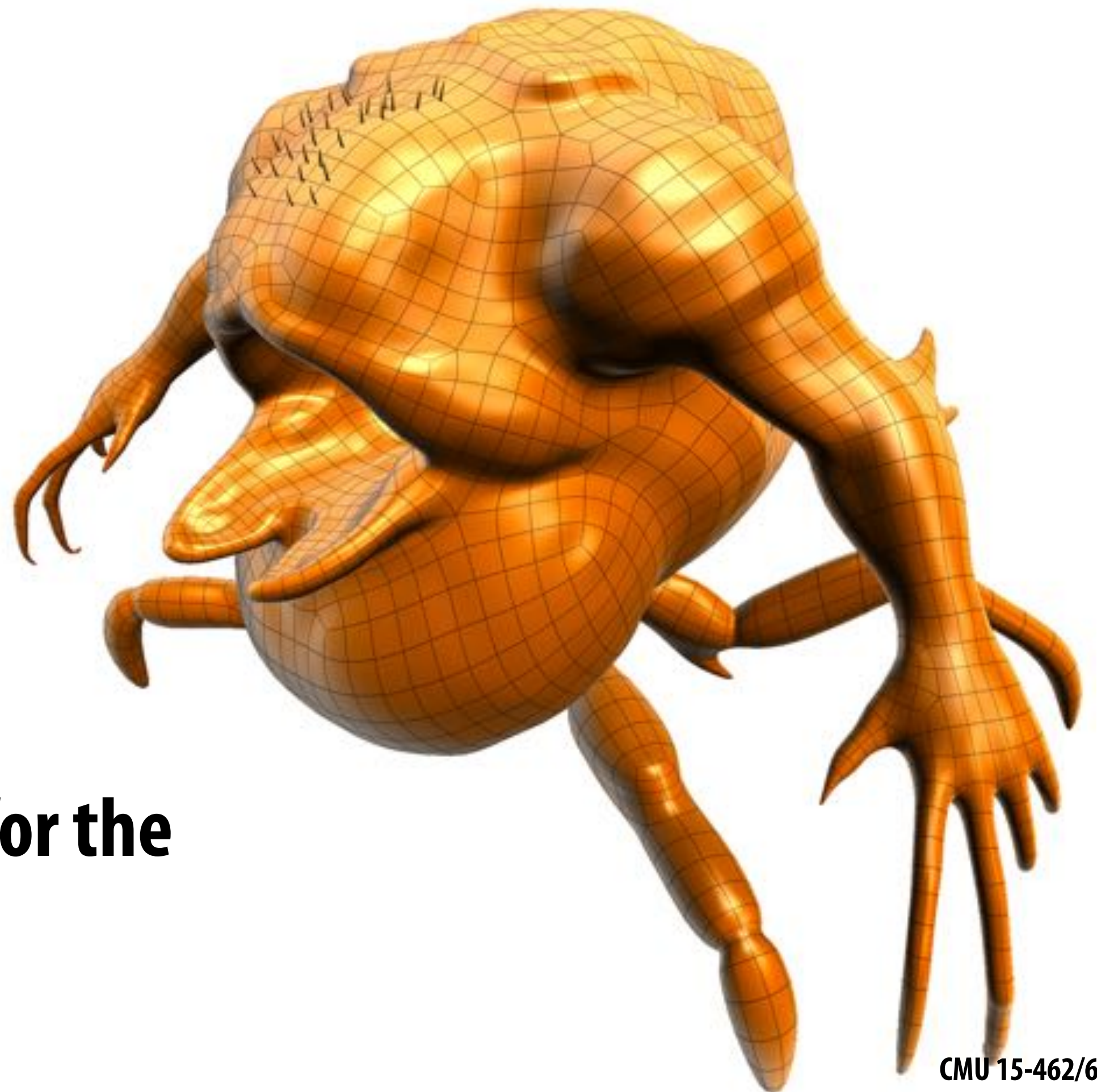
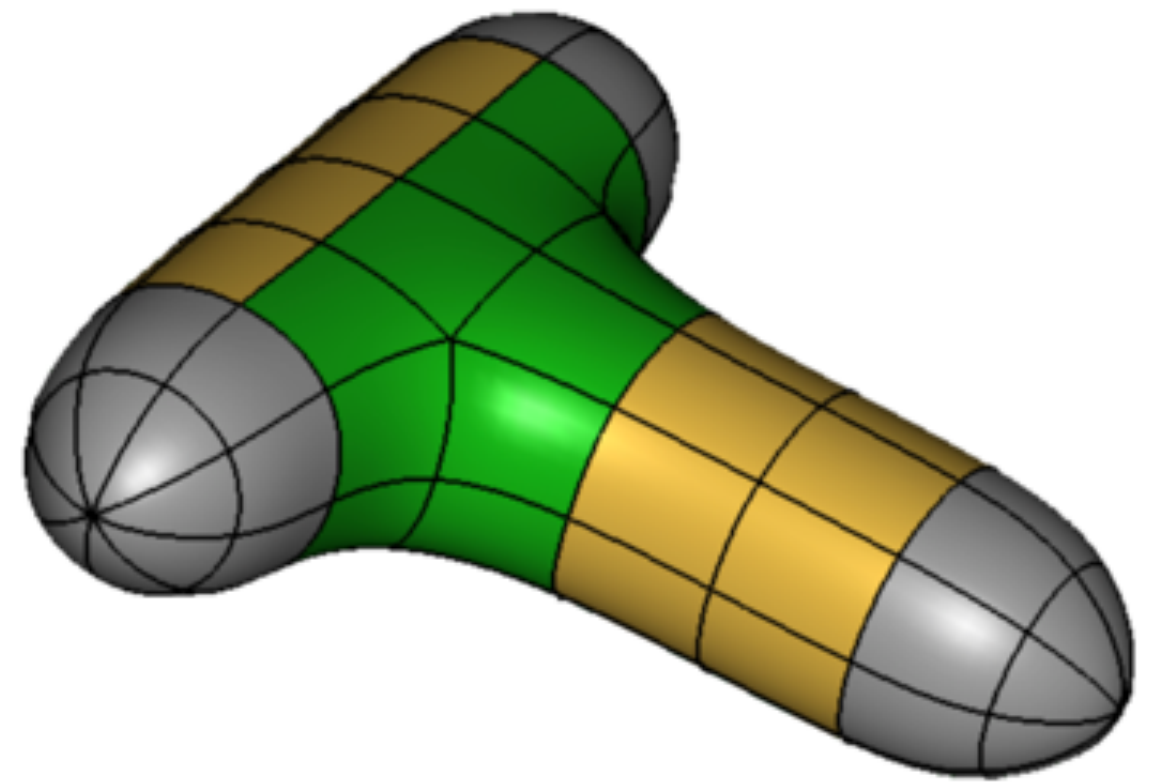
Notice that exactly four patches meet around every vertex!

In practice, far too constrained.

To make interesting shapes (with good continuity), we need patches that allow more interesting connectivity...

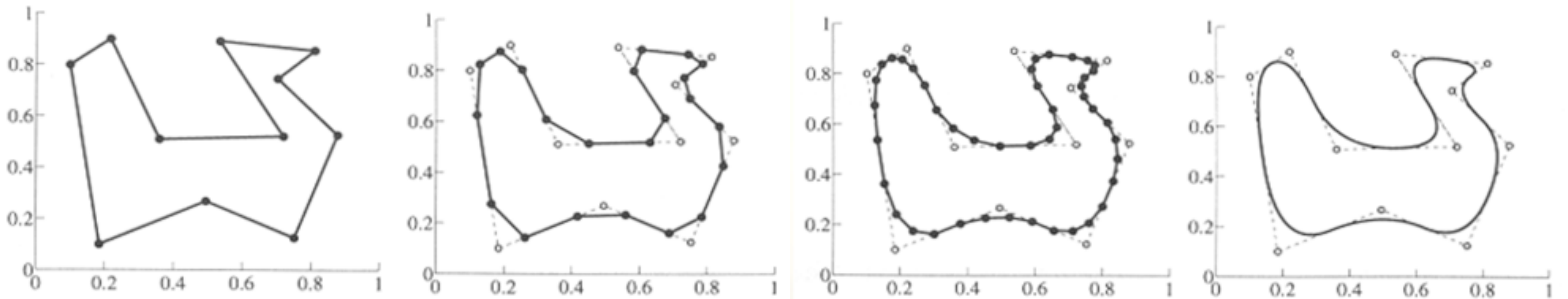
Spline patch schemes

- There are many alternatives!
- NURBS, Gregory, Pm, polar...
- Tradeoffs:
 - degrees of freedom
 - continuity
 - difficulty of editing
 - cost of evaluation
 - generality
 - ...
- As usual: pick the right tool for the job!



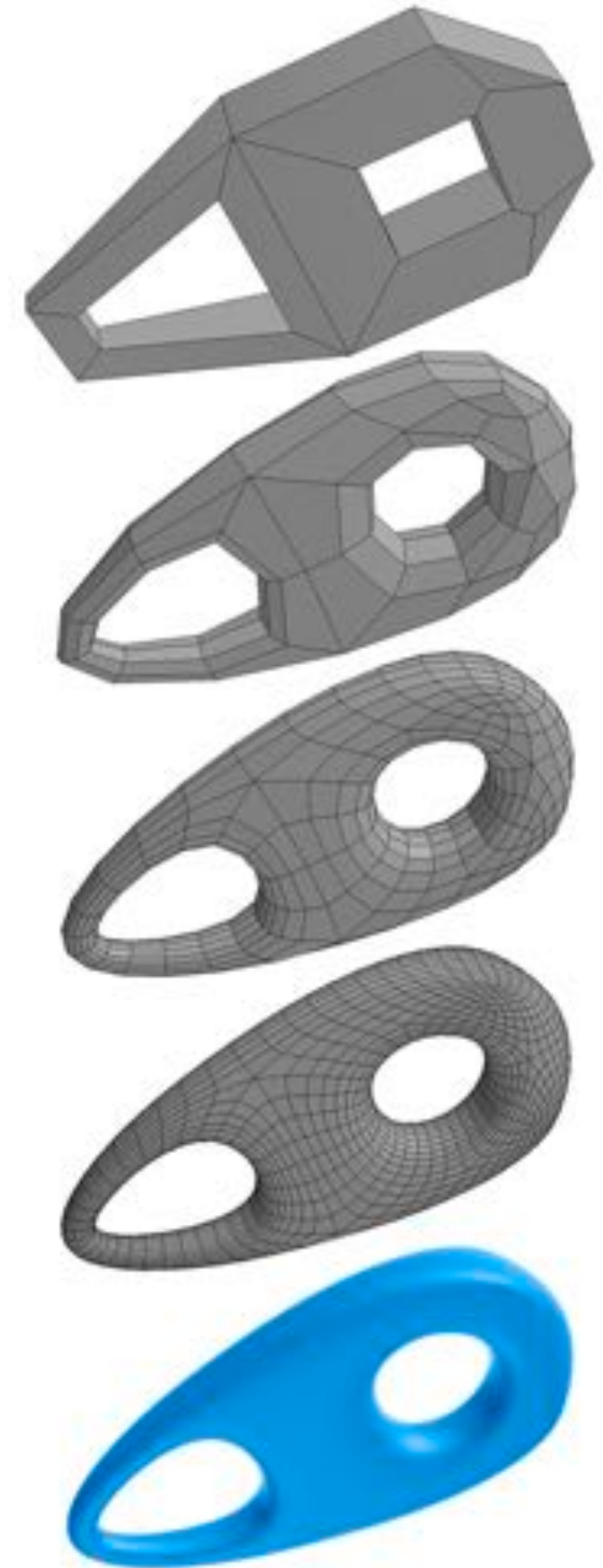
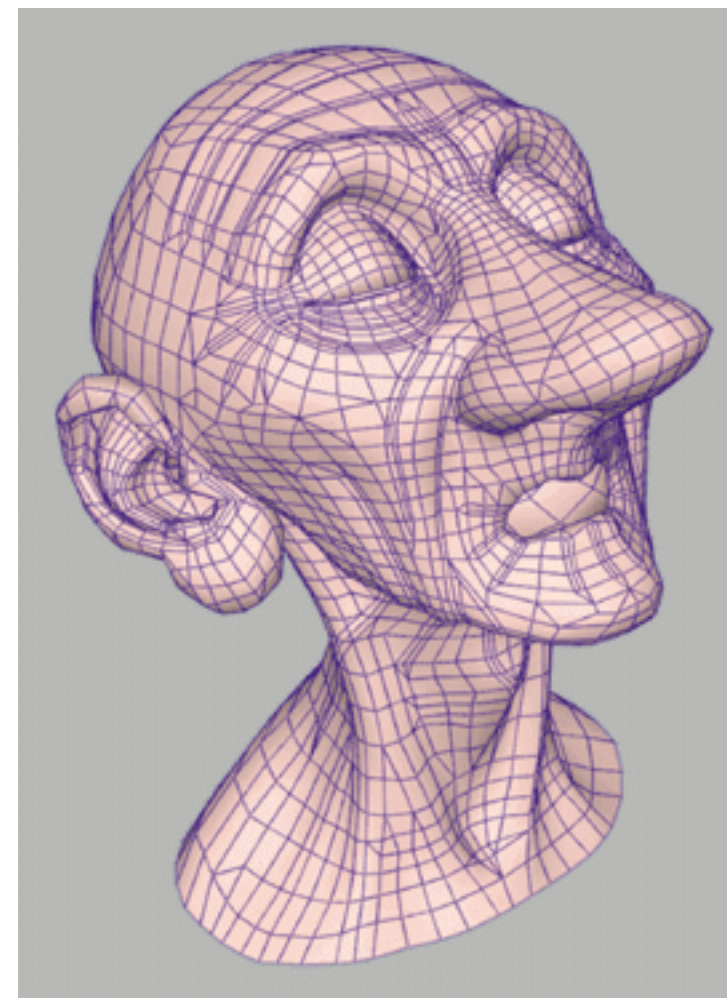
Subdivision (Explicit or Implicit?)

- **Alternative starting point for curves/surfaces: subdivision**
- **Start with control curve**
- **Insert new vertex at each edge midpoint**
- **Update vertex positions according to fixed rule**
- **For careful choice of averaging rule, yields smooth curve**
 - **Some subdivision schemes correspond to well-known spline schemes!**



Subdivision Surfaces (Explicit)

- Start with coarse polygon mesh (“control cage”)
- Subdivide each element
- Update vertices via local averaging
- Many possible rule:
 - Catmull-Clark (quads)
 - Loop (triangles)
 - ...
- Common issues:
 - interpolating or approximating?
 - continuity at vertices?
- Easier than splines for modeling; harder to evaluate pointwise



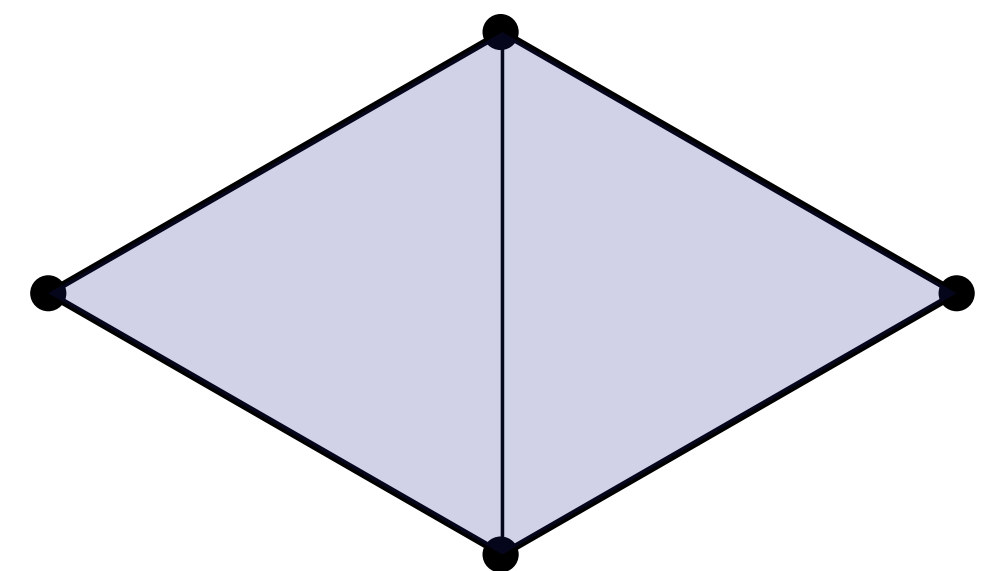
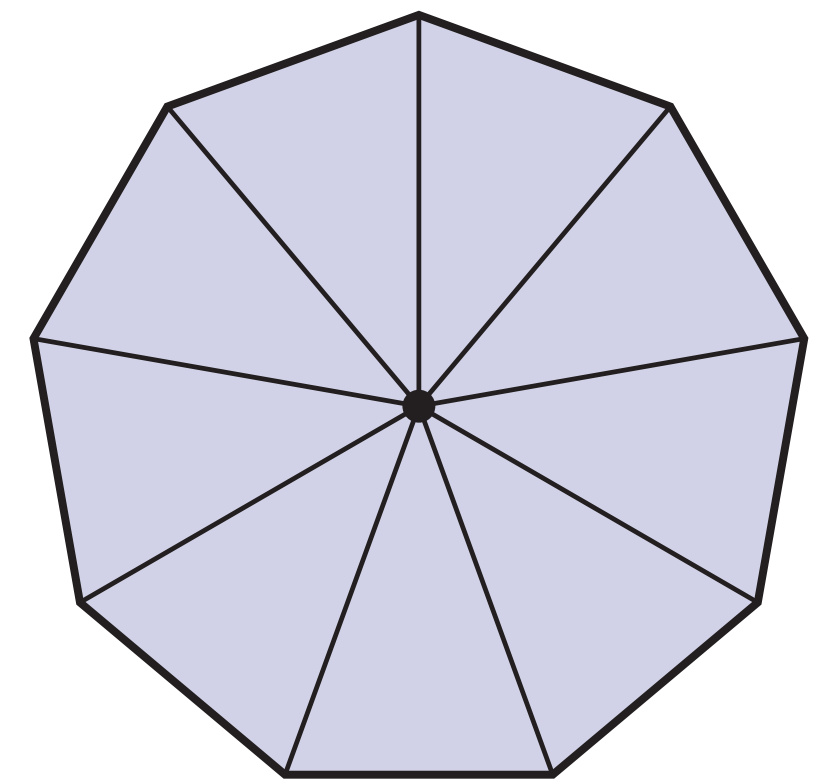
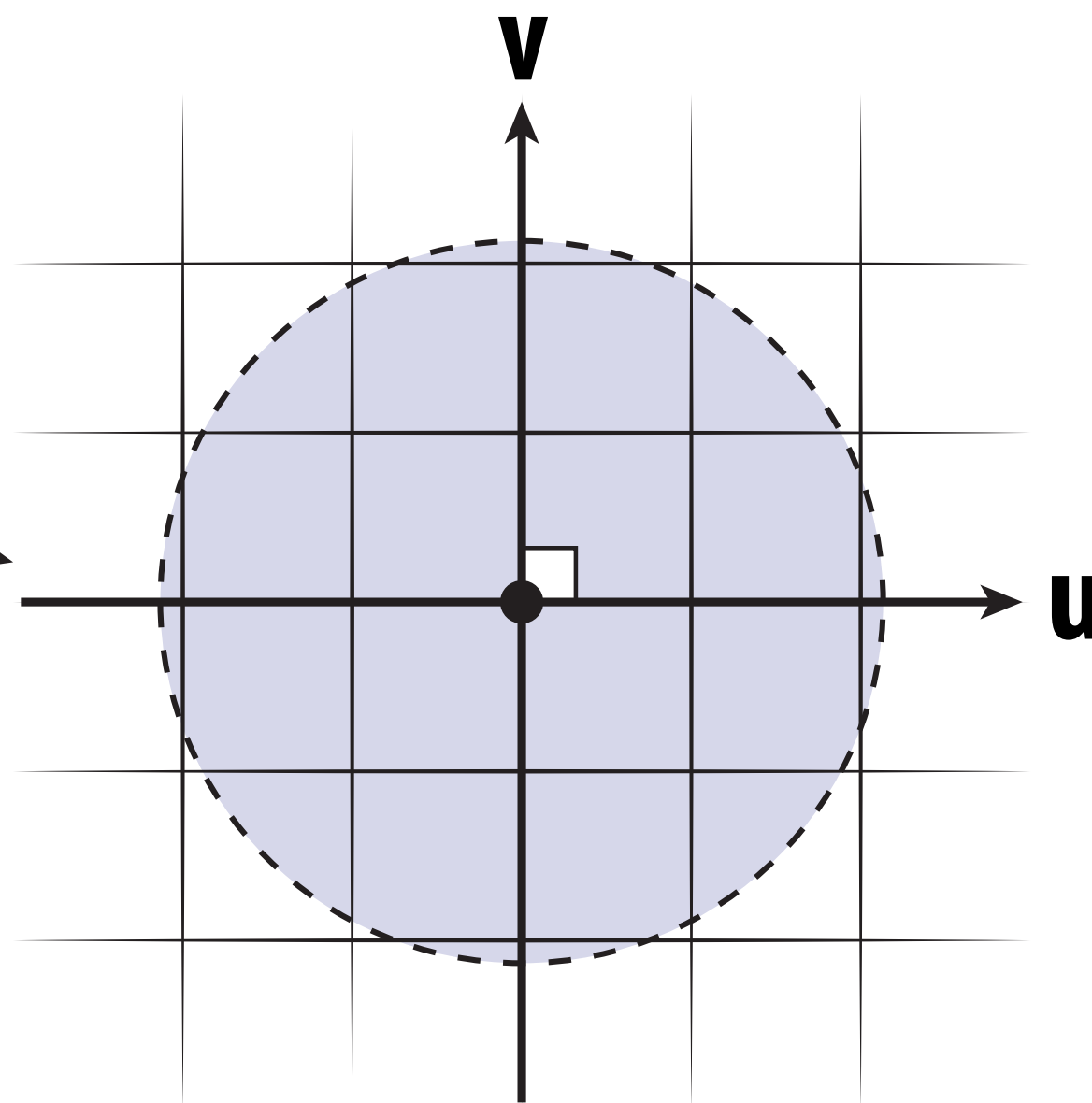
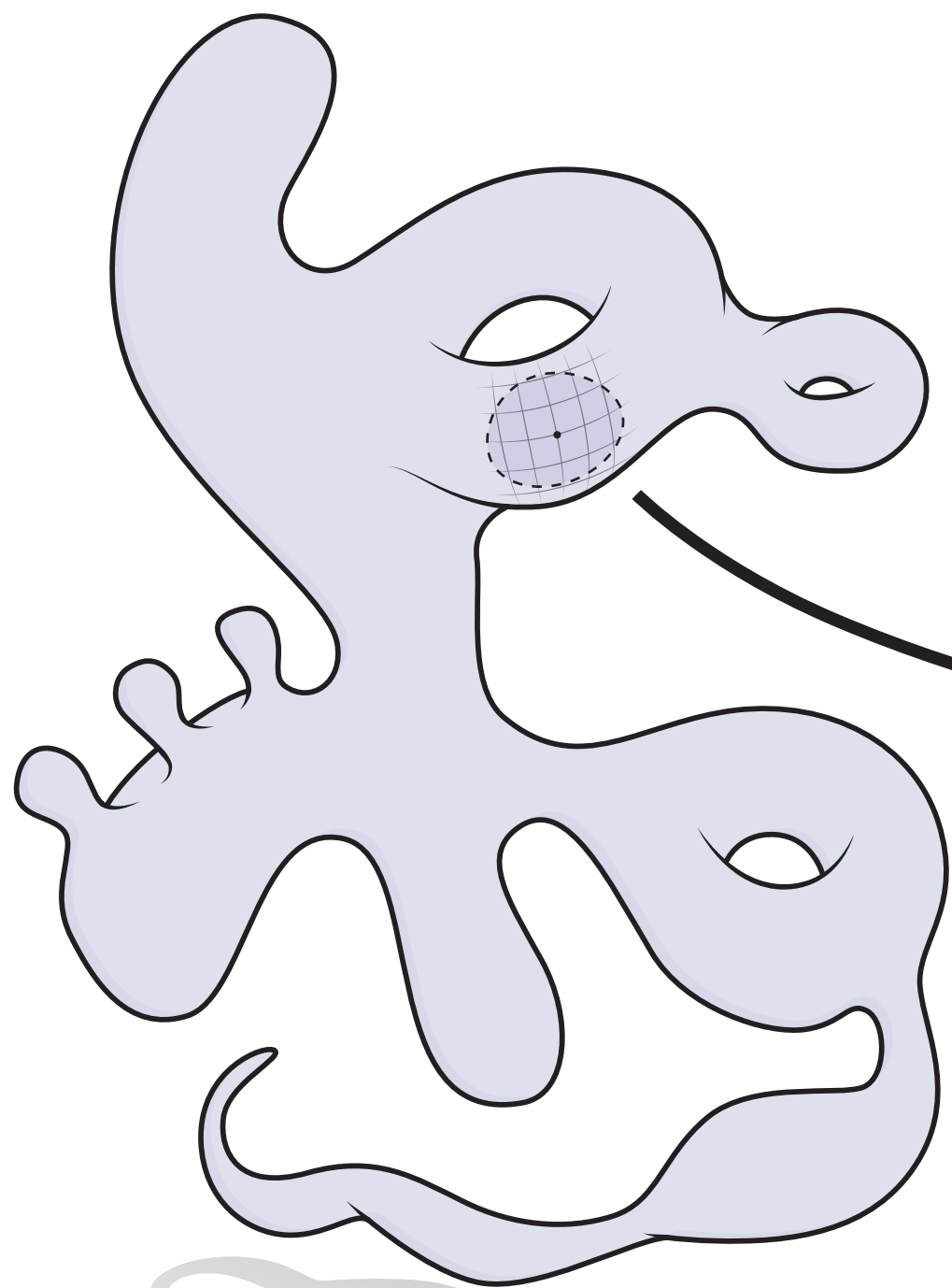
Subdivision in Action (Pixar's "Geri's Game")

Meshes and Manifolds

Computer Graphics
CMU 15-462/15-662

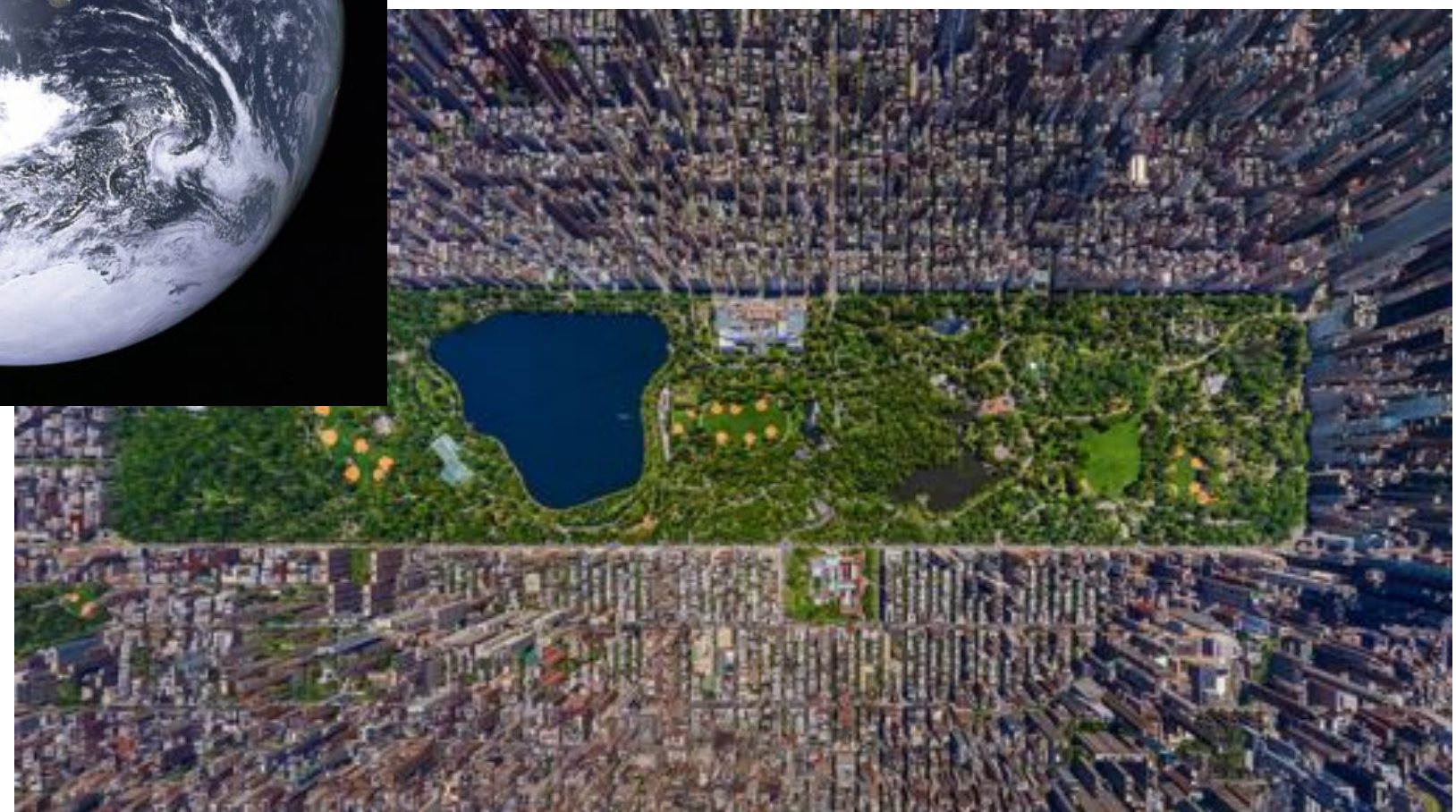
Manifold Assumption

- Next, we're going to introduce the idea of manifold geometry



Smooth Surfaces

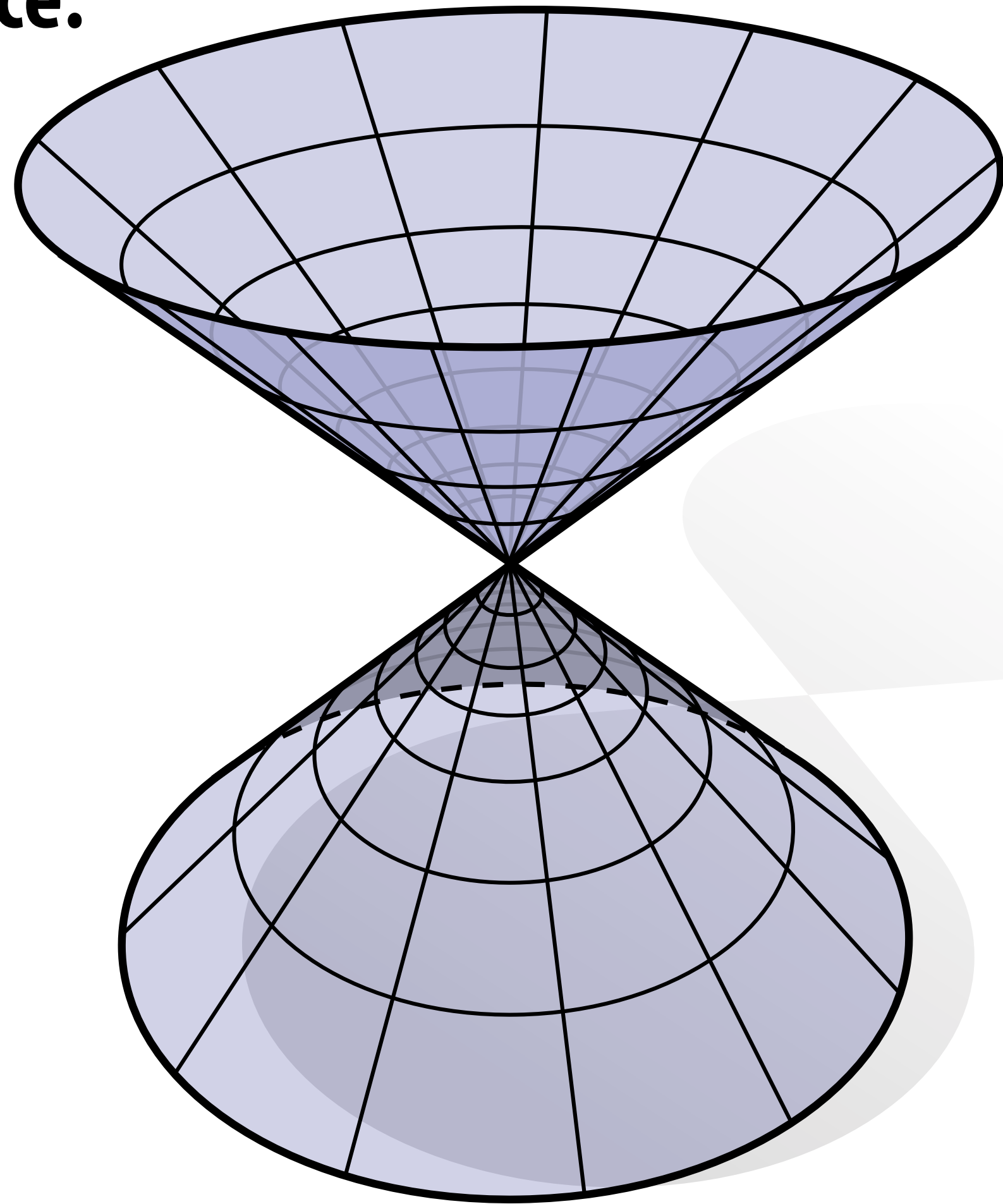
- Intuitively, a surface is the boundary or “shell” of an object
- (Think about the candy shell, not the chocolate.)
- Surfaces are manifold:
 - If you zoom in far enough (at any point) looks like a plane*
 - E.g., the Earth from space vs. from the ground



*...or can easily be flattened into the plane, without cutting or ripping.

Isn't every shape manifold?

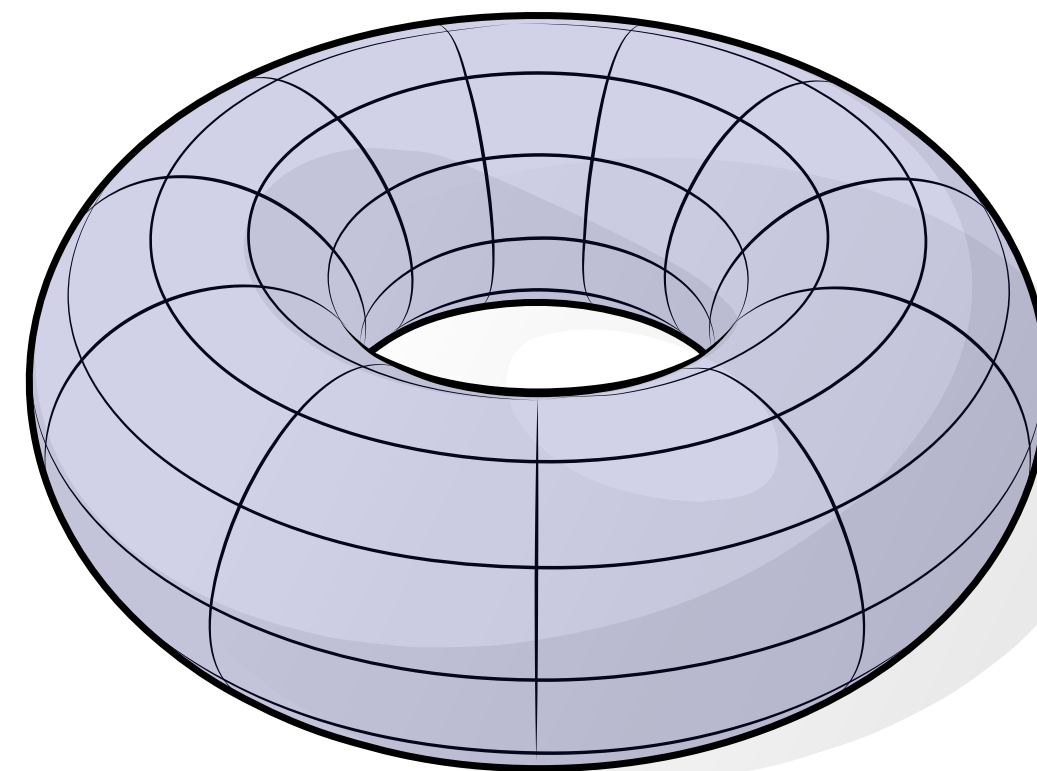
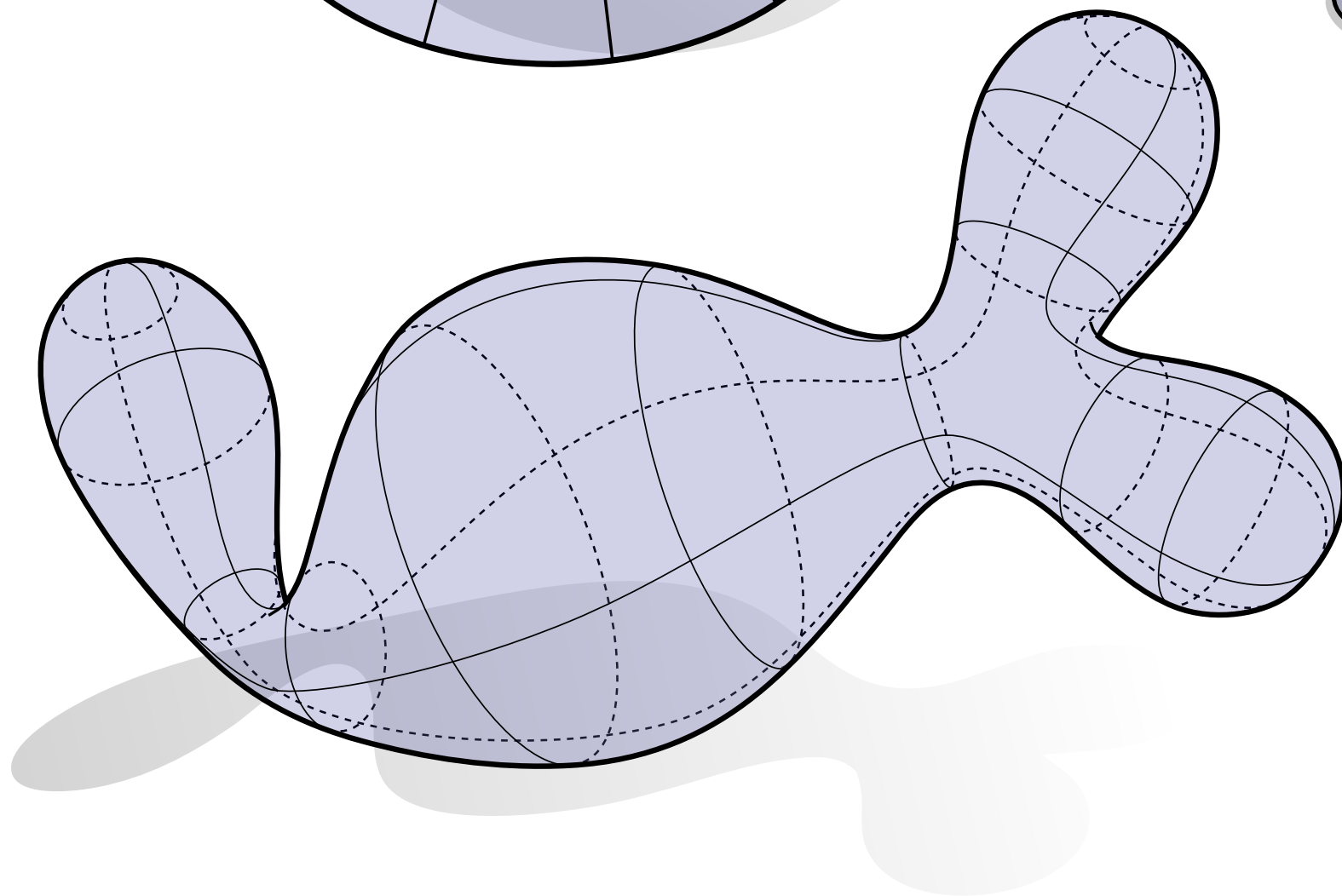
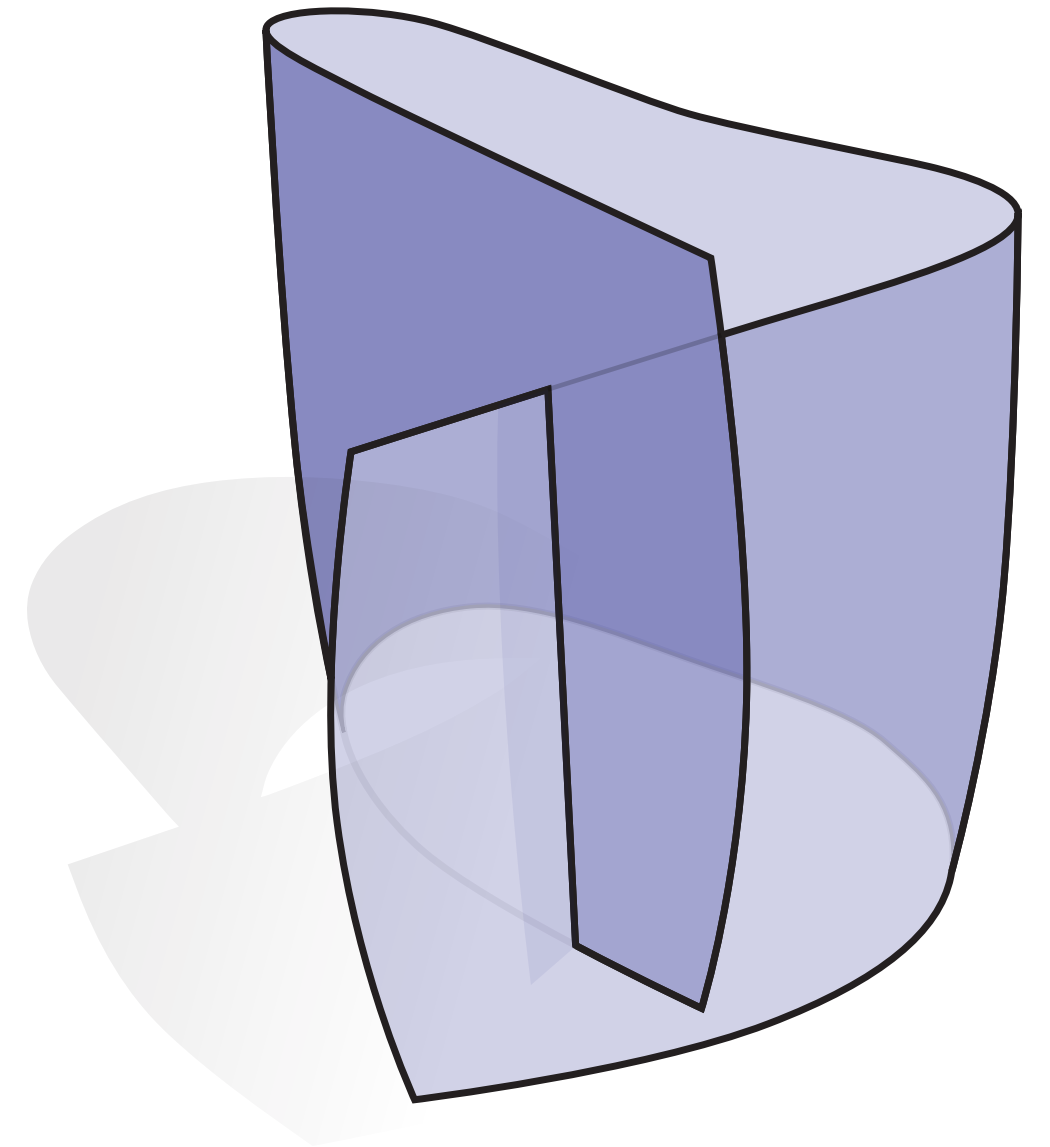
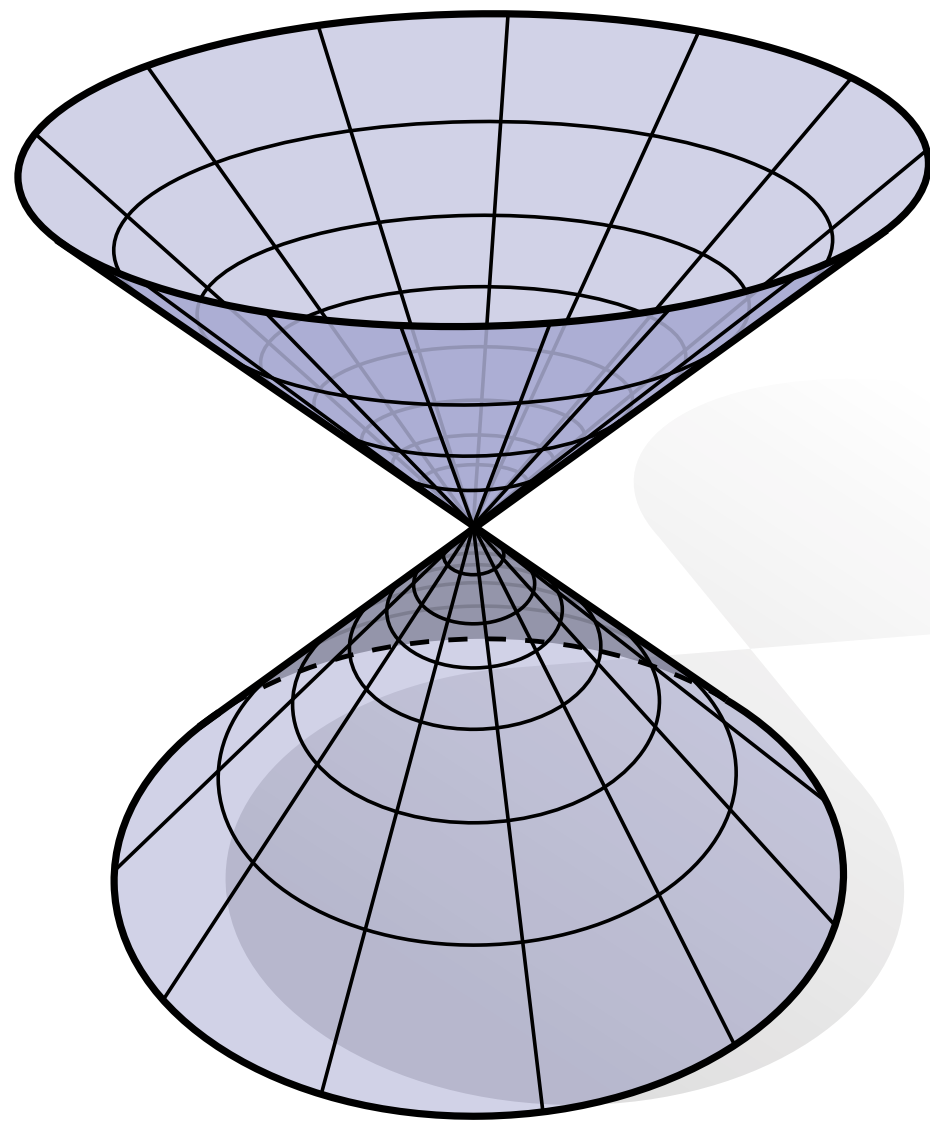
- No, for instance:



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

More Examples of Smooth Surfaces

- Which of these shapes are manifold?



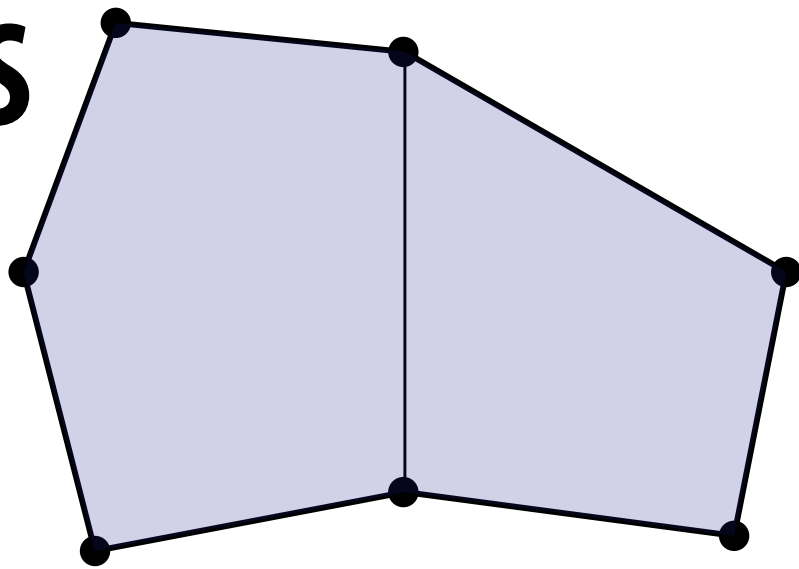
A manifold polygon mesh has fans, not fins

■ For polygonal surfaces just two easy conditions to check:

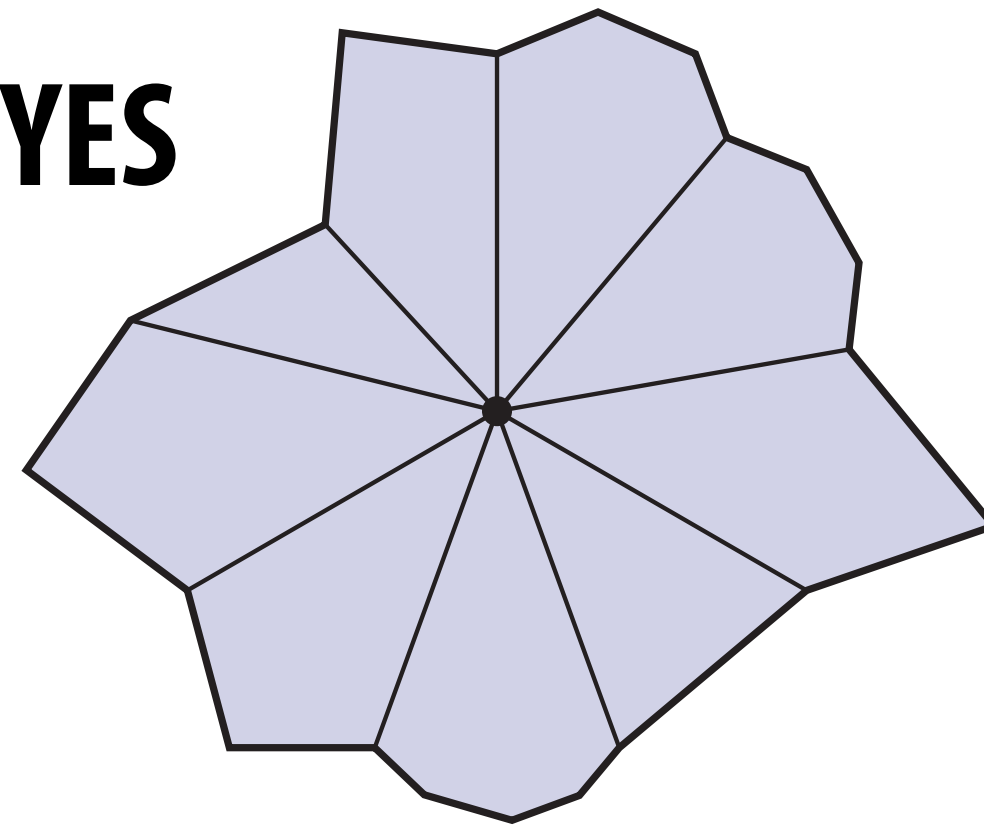
1. Every edge is contained in only two polygons (no “fins”)

2. The polygons containing each vertex make a single “fan”

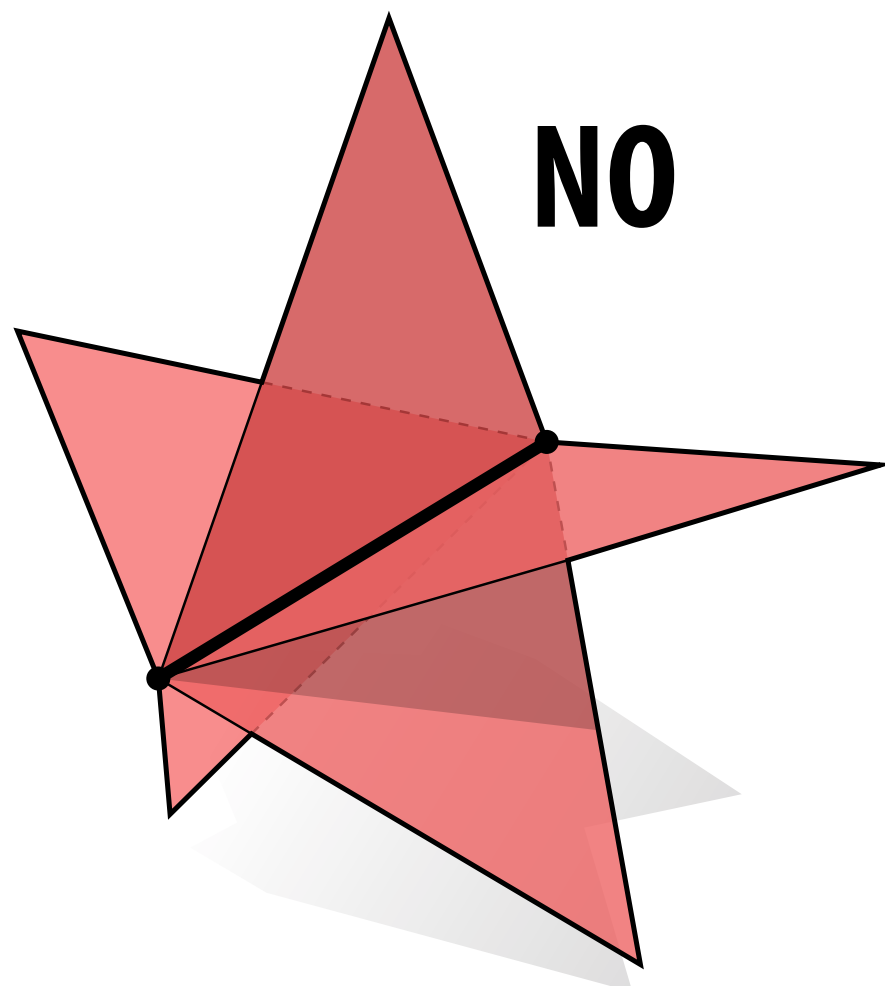
YES



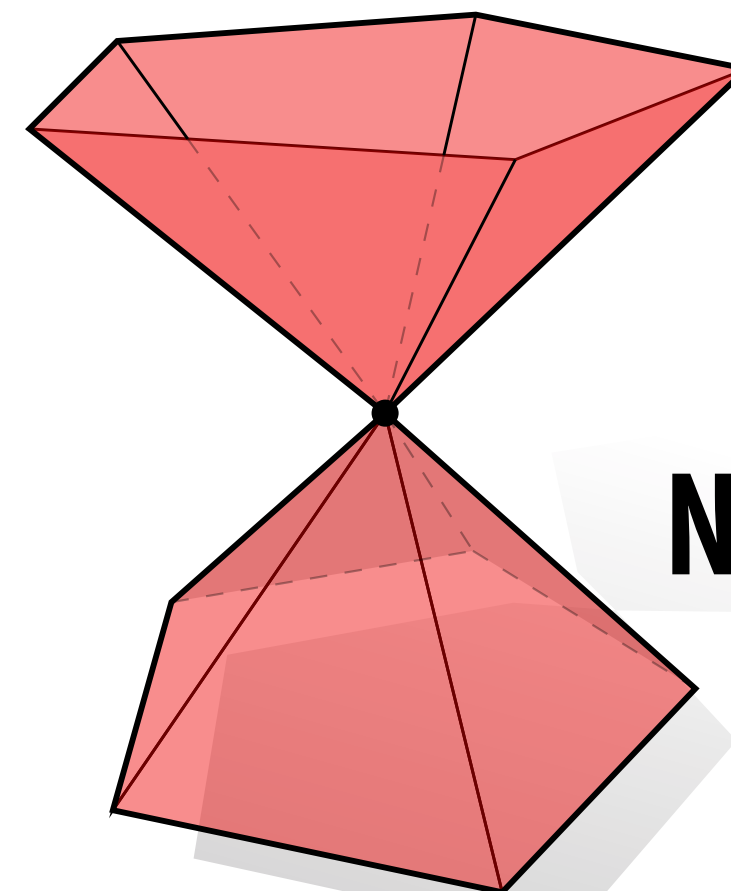
YES



NO

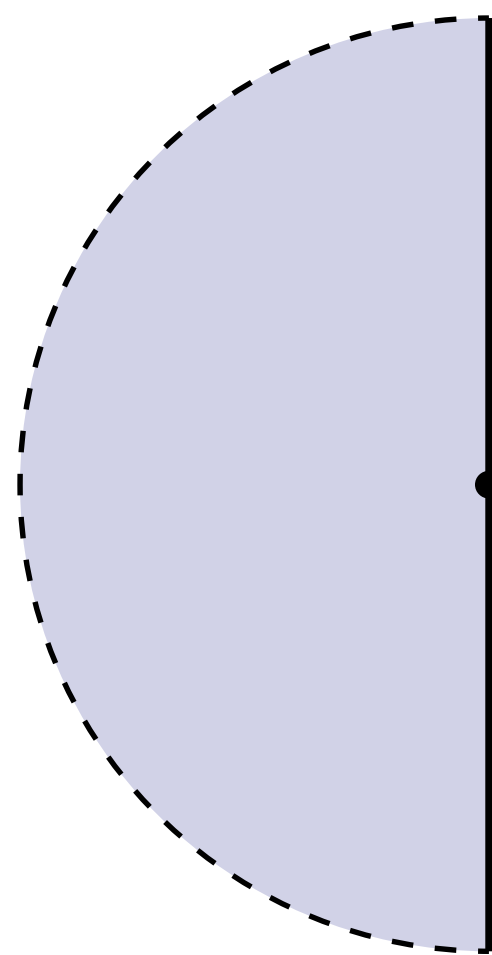
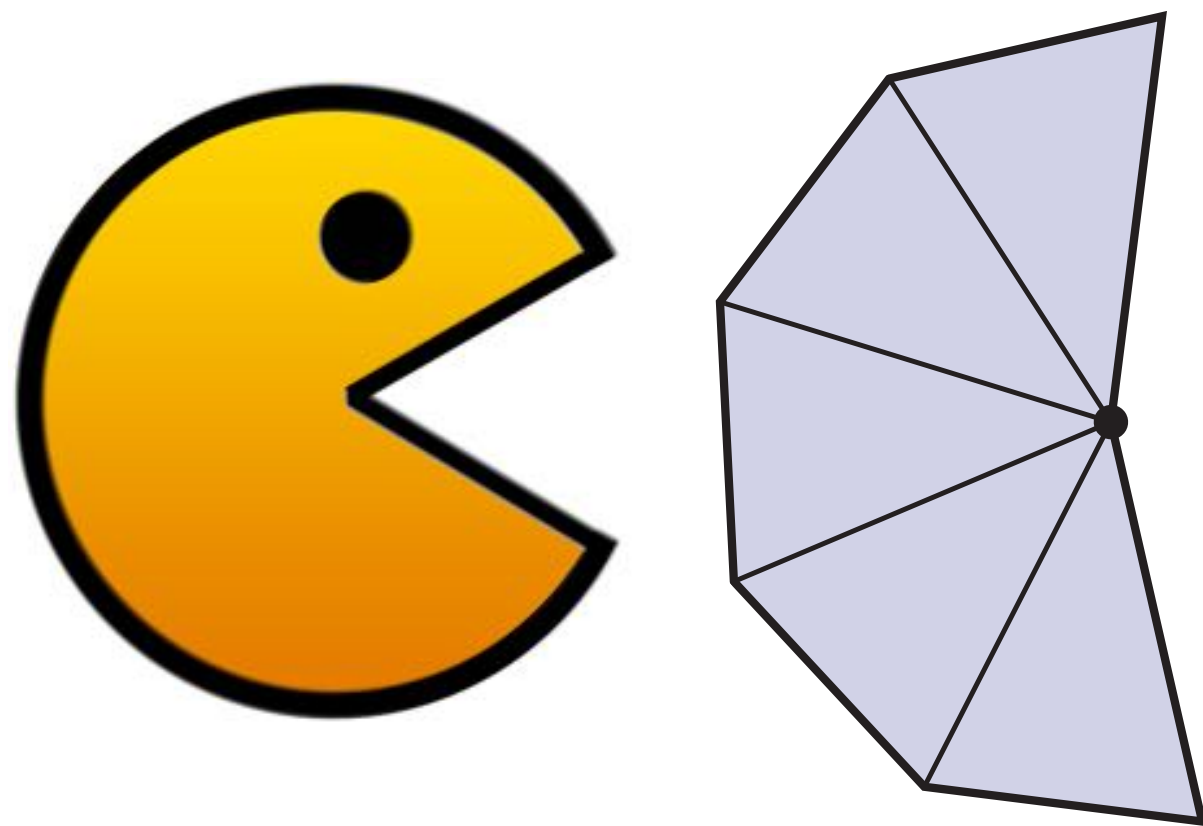


NO



What about boundary?

- The boundary is where the surface “ends.”
- E.g., waist & ankles on a pair of pants.
- Locally, looks like a half disk
- Globally, each boundary forms a loop



YES

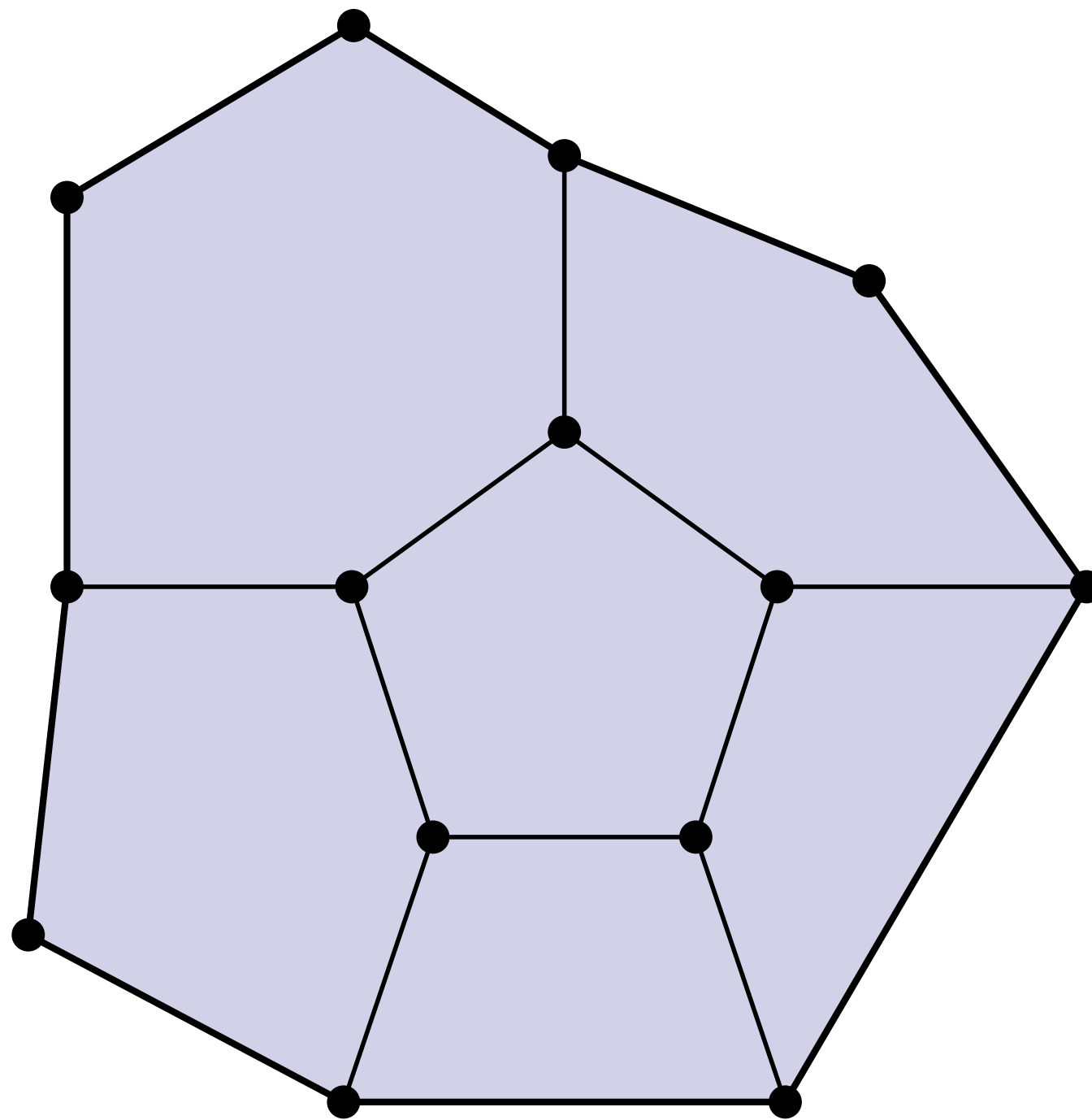


- Polygon mesh:
 - one polygon per boundary edge
 - boundary vertex looks like “pacman”

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

Keep it Simple!

- **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**



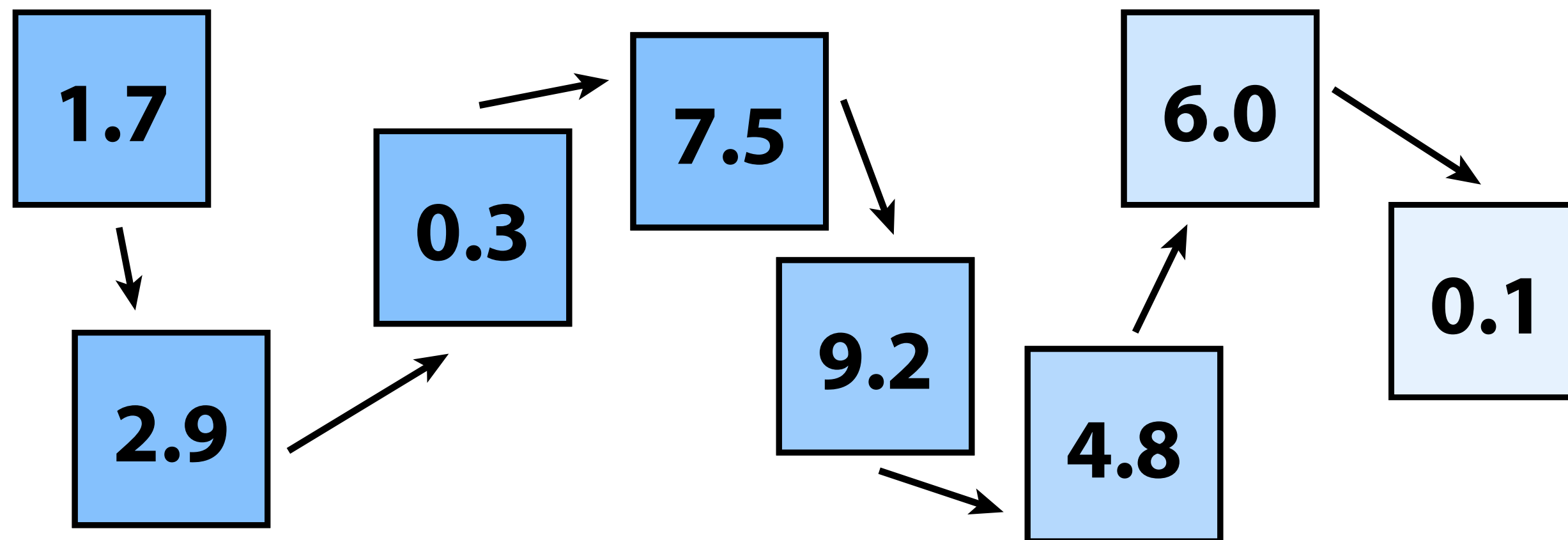
How do we actually encode all this data?

Warm up: storing numbers

- **Q: What data structures can we use to store a list of numbers?**
- **One idea: use an array (constant time lookup, coherent access)**



- **Alternative: use a linked list (linear lookup, incoherent access)**



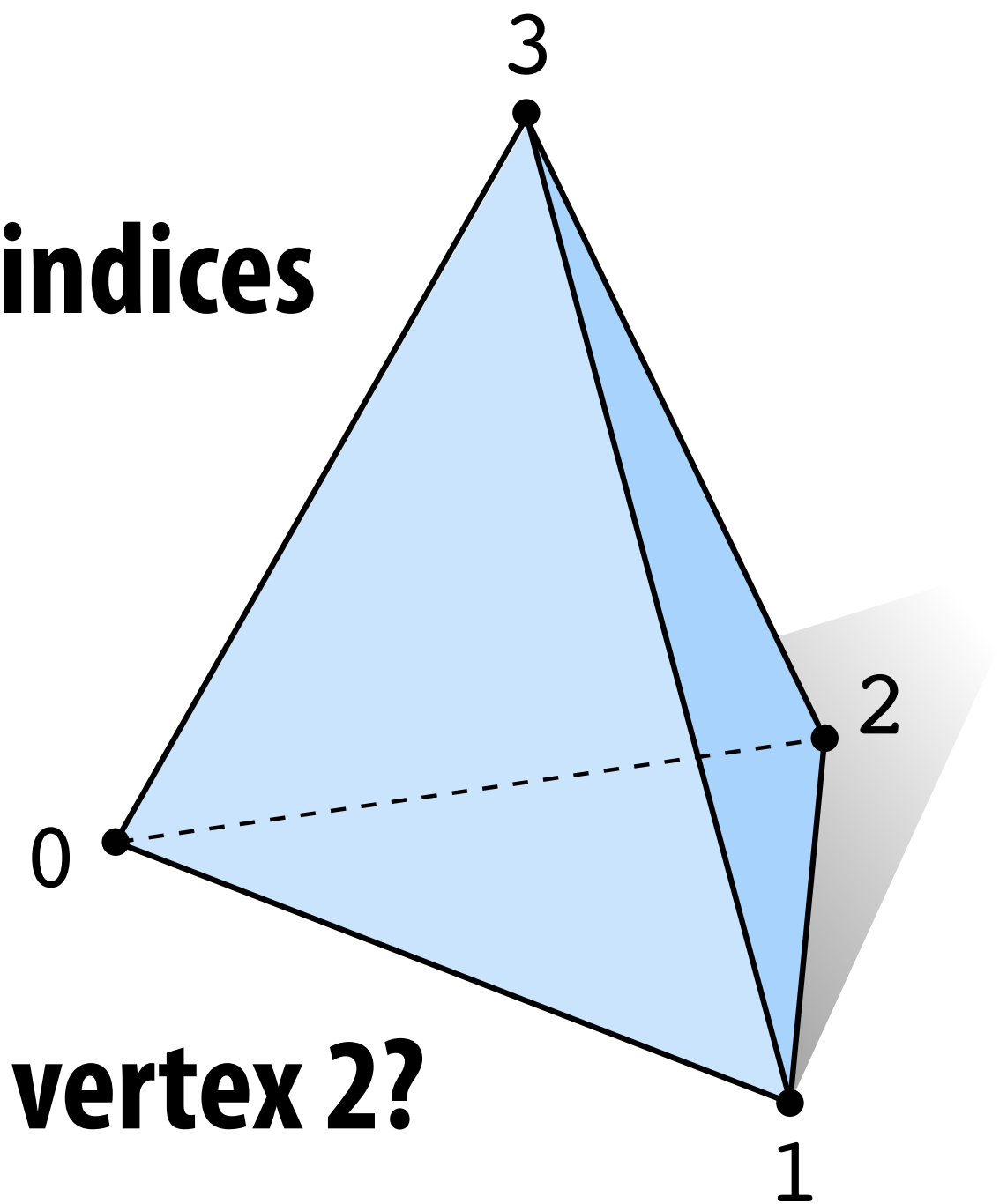
- **Q: Why bother with the linked list?**
- **A: For one, we can easily insert numbers wherever we like...**

Adjacency List (Array-like)

- Store triples of coordinates (x,y,z) , tuples of indices

- E.g., tetrahedron:

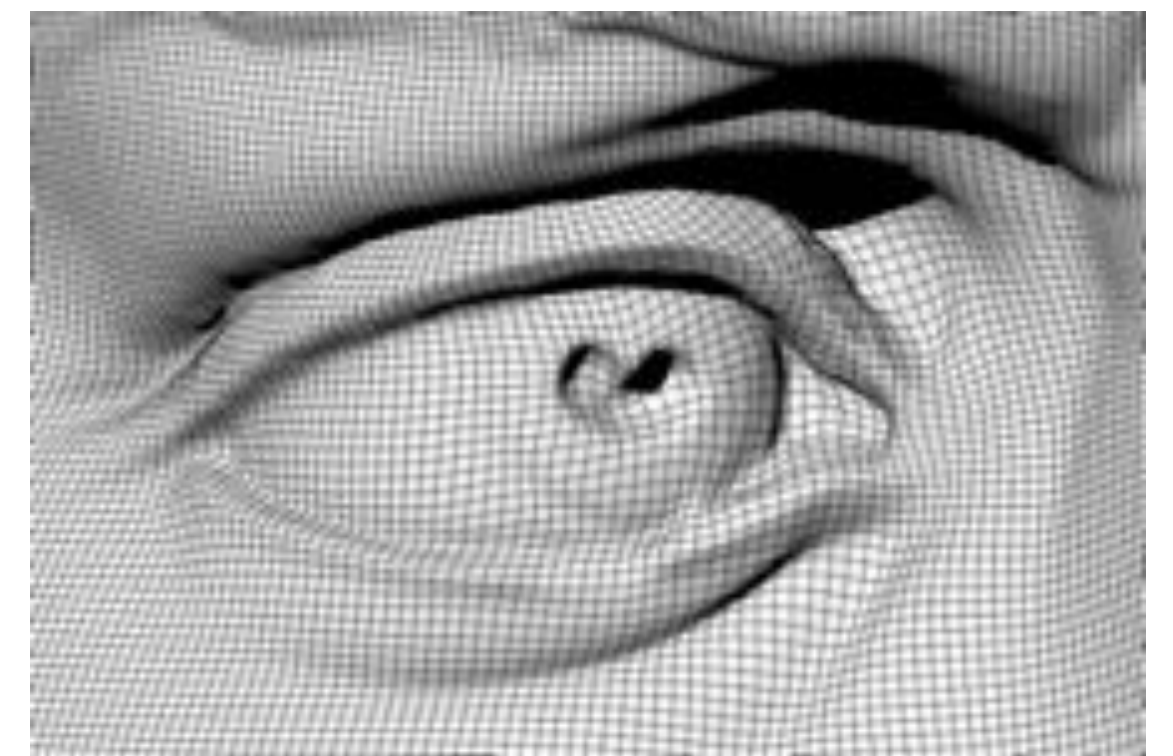
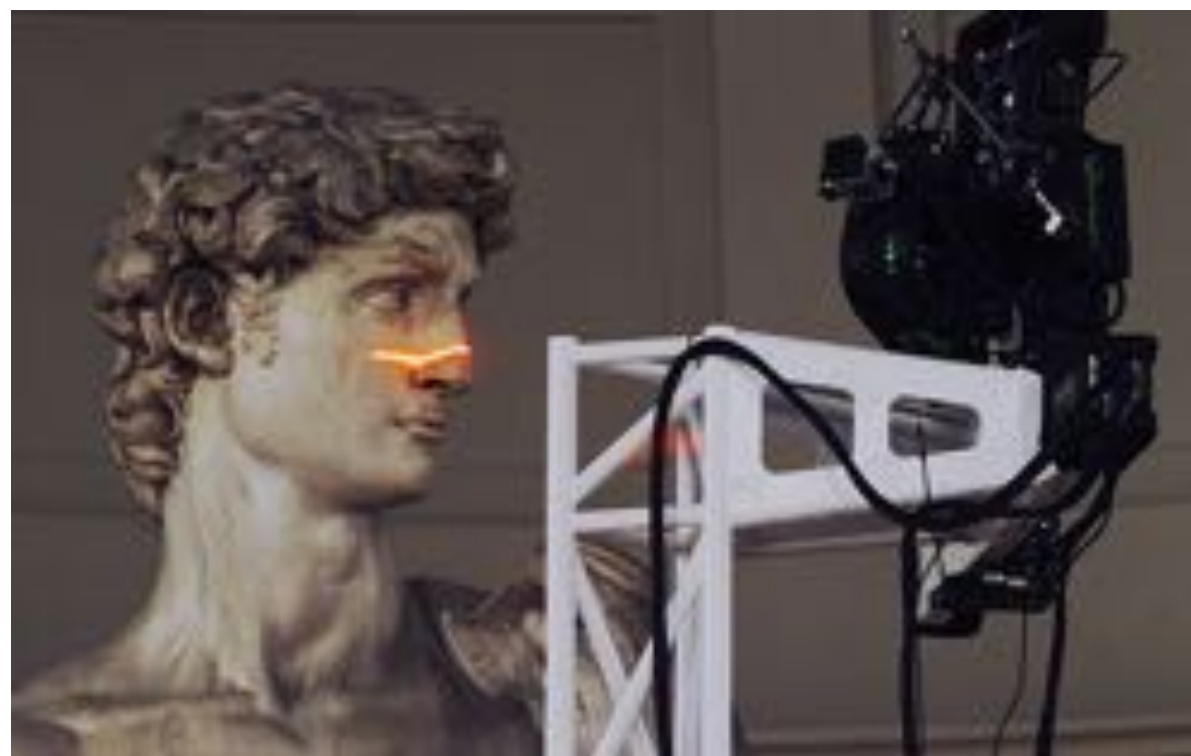
	VERTICES			POLYGONS		
	x	y	z	i	j	k
0:	-1	-1	-1	0	2	1
1:	1	-1	1	0	3	2
2:	1	1	-1	3	0	1
3:	-1	1	1	3	1	2



- Q: How do we find all the polygons touching vertex 2?

- Ok, now consider a more complicated mesh:

~1 billion polygons

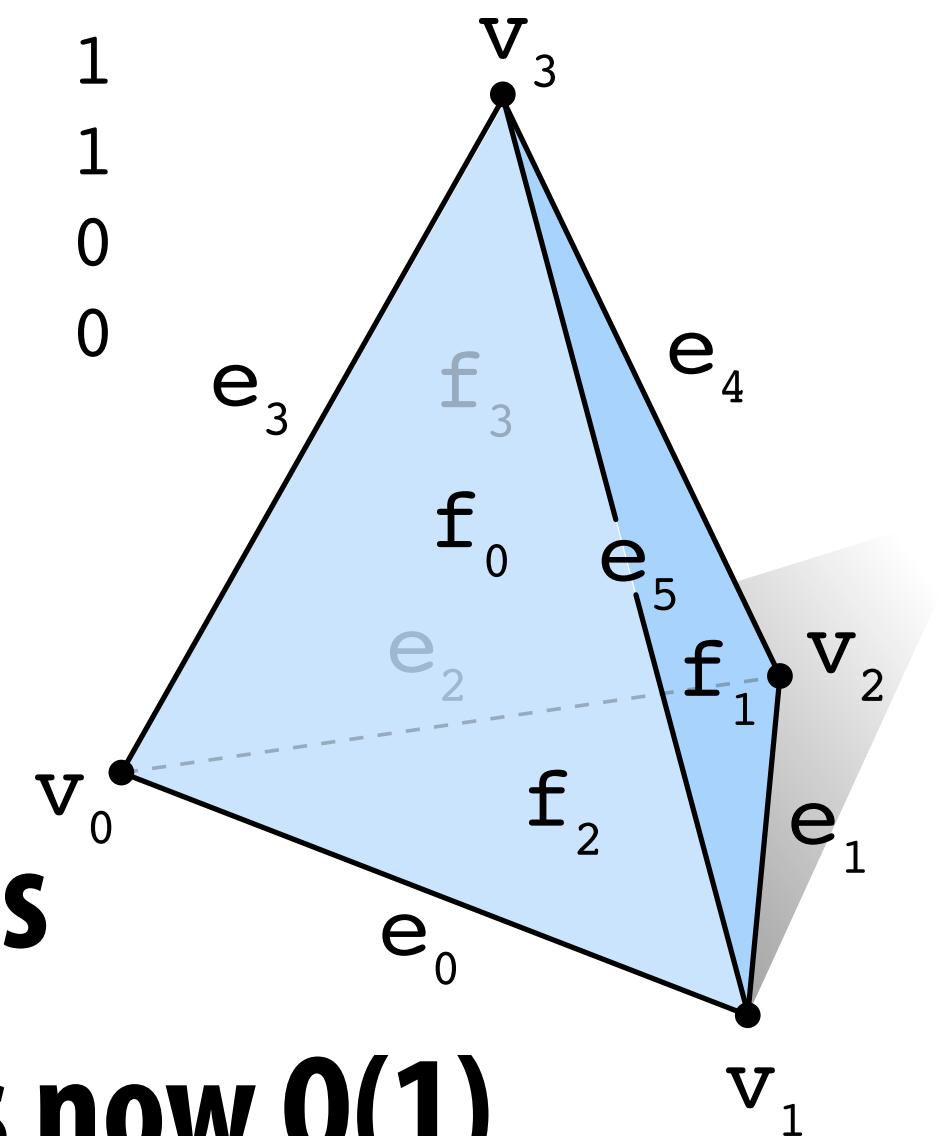


Very expensive to find the neighboring triangles! (What's the cost?)

Incidence Matrices

- If we want to answer neighborhood queries, why not simply store a list of neighbors?
- Can encode all neighbor information via incidence matrices
- E.g., tetrahedron: **VERTEX ↔ EDGE** **EDGE ↔ FACE**

	v0	v1	v2	v3		e0	e1	e2	e3	e4	e5
e0	1	1	0	0	f0	1	0	0	1	0	1
e1	0	1	1	0	f1	0	1	0	0	1	1
e2	1	0	1	0	f2	1	1	1	0	0	0
e3	1	0	0	1	f3	0	0	1	1	1	0
e4	0	0	1	1							
e5	0	1	0	1							

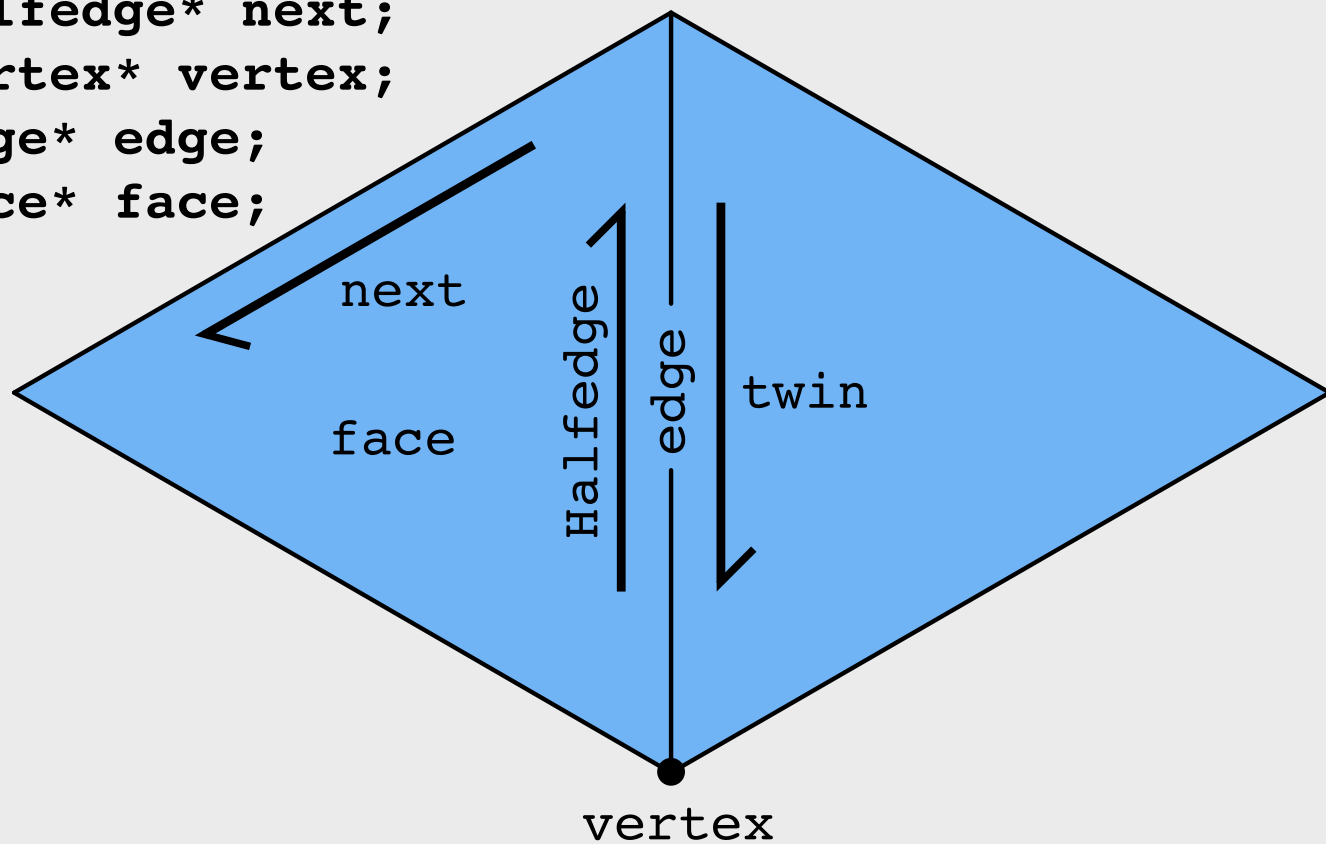


- 1 means “touches”; 0 means “does not touch”
- Instead of storing lots of 0’s, use sparse matrices
- Still large storage cost, but finding neighbors is now $O(1)$
- Hard to change connectivity, since we used fixed indices
- Bonus feature: mesh does not have to be manifold

Halfedge Data Structure (Linked-list-like)

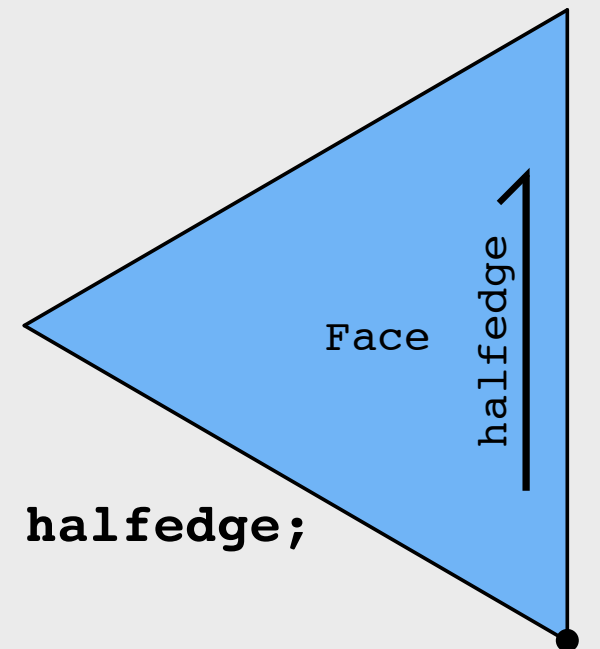
- Store some information about neighbors
- Don't need an exhaustive list; just a few key pointers
- Key idea: two halfedges act as "glue" between mesh elements:

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

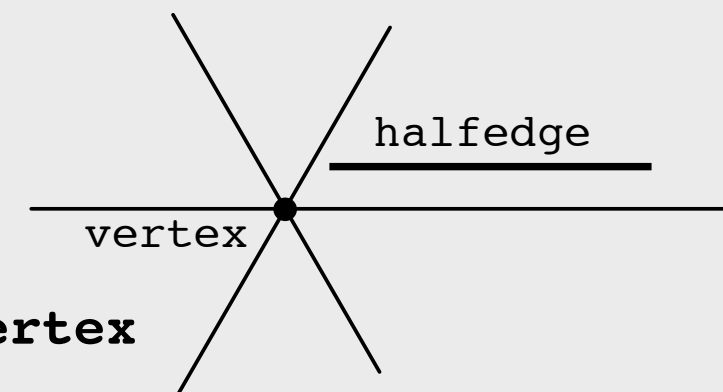


```
struct Edge
{
  Halfedge* halfedge;
};
```

```
struct Face
{
  Halfedge* halfedge;
};
```



```
struct Vertex
{
  Halfedge* halfedge;
};
```



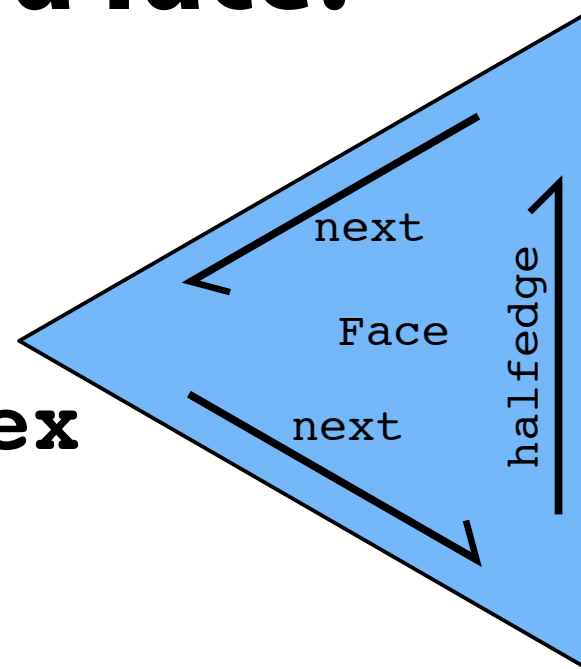
- Each vertex, edge face points to just one of its halfedges.

Halfedge makes mesh traversal easy

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

- Example: visit all vertices of a face:

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

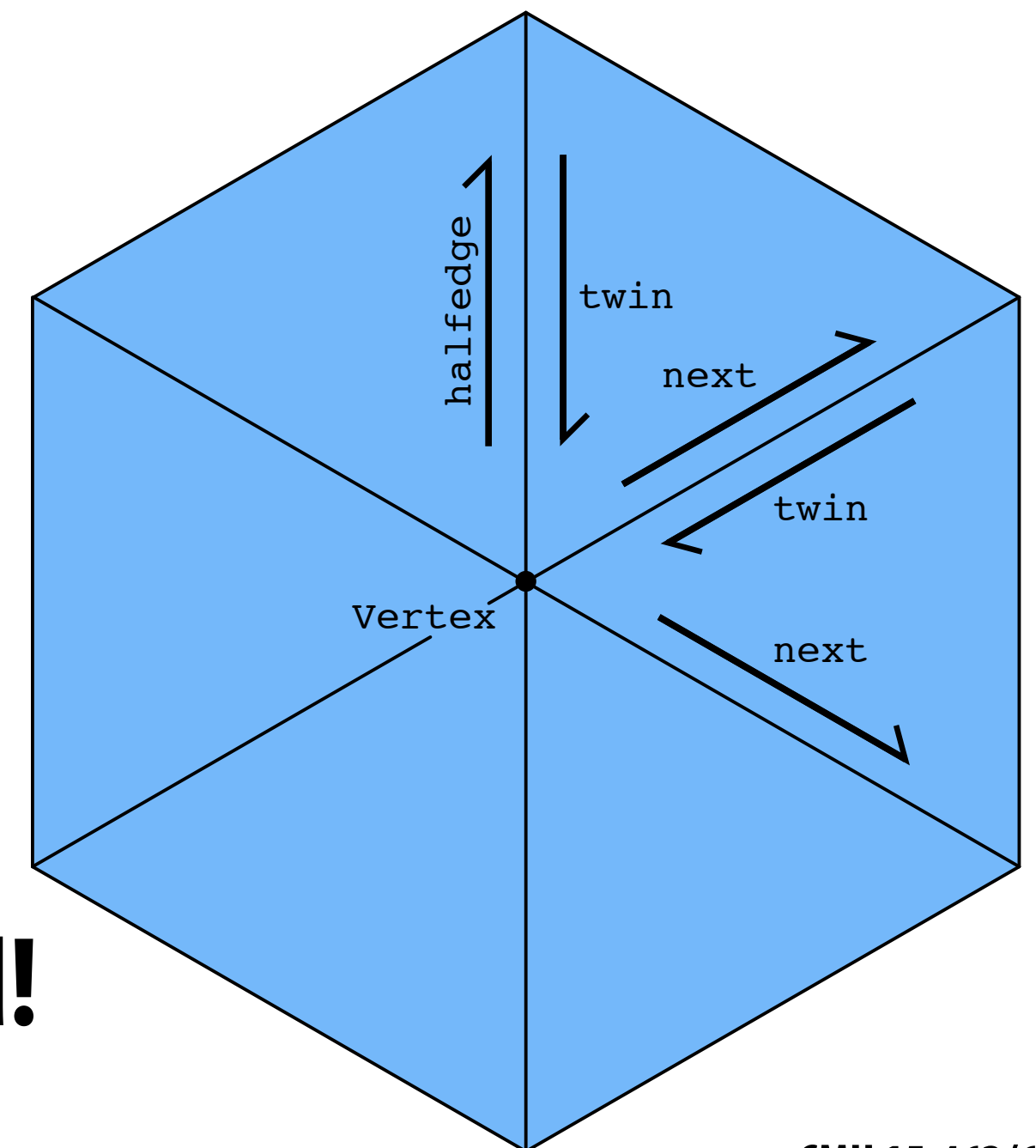


- Example: visit all neighbors of a vertex:

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

- [DEMO]

- Note: only makes sense if mesh is manifold!



Halfedge meshes are always manifold

- Consider simplified halfedge data structure
- Require only “common-sense” conditions

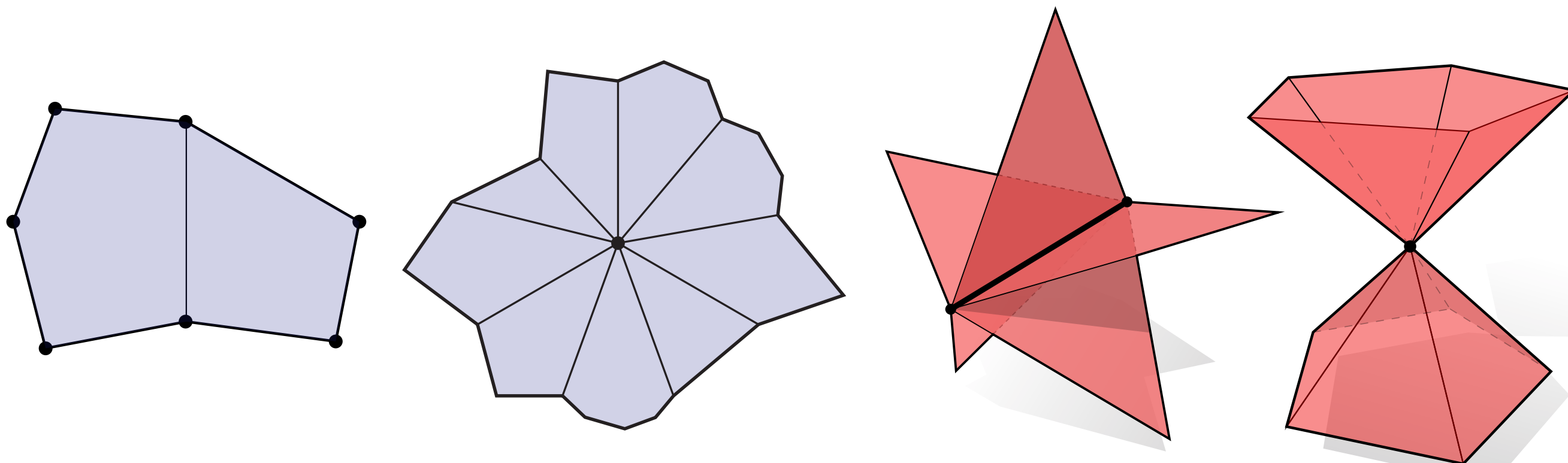
```
struct Halfedge {  
    Halfedge *next, *twin;  
};
```

```
twin->twin == this  
next != this  
twin != this
```

(pointer to yourself!)



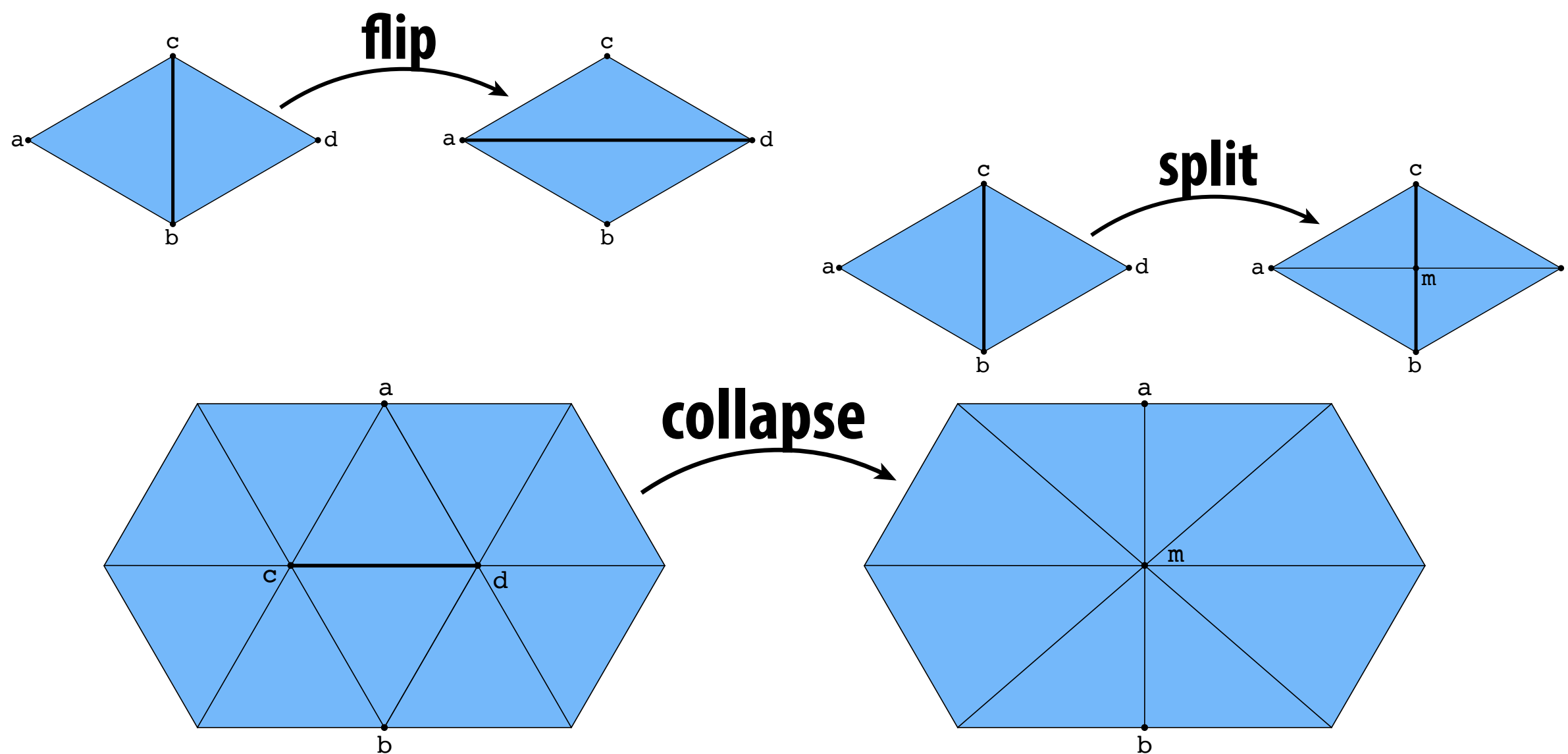
- Keep following `next`, and you'll get faces.
- Keep following `twin` and you'll get edges.
- Keep following `next->twin` and you'll get vertices.



Q: Why, therefore, is it impossible to encode the red figures?

Halfedge meshes are easy to edit

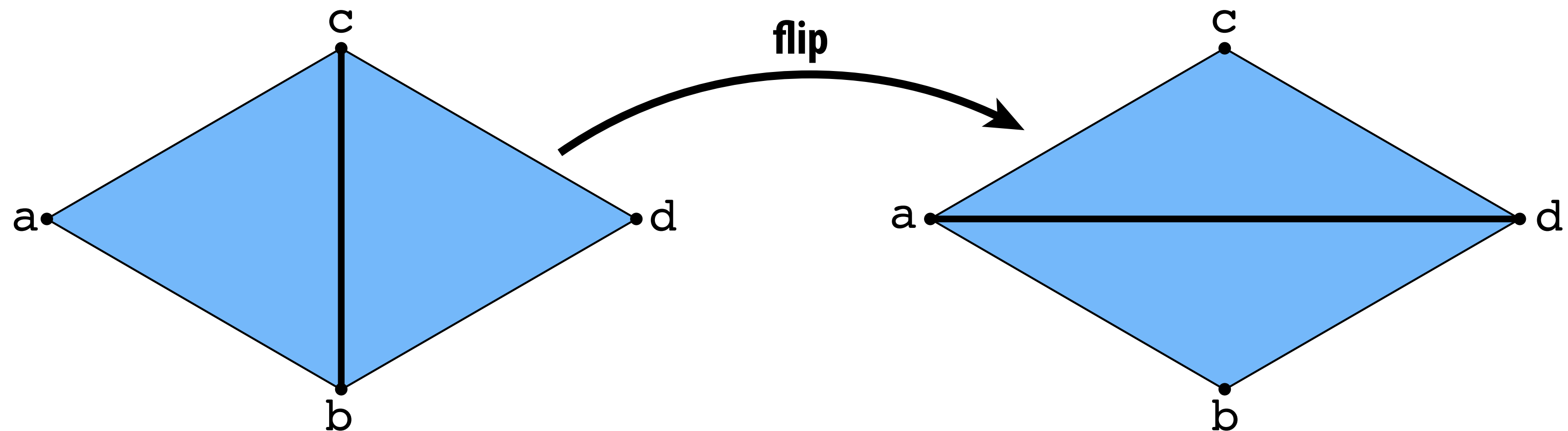
- Remember key feature of linked list: insert/delete elements
- Same story with halfedge mesh (“linked list on steroids”)
- E.g., for triangle meshes, several atomic operations:



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

Edge Flip (Triangles)

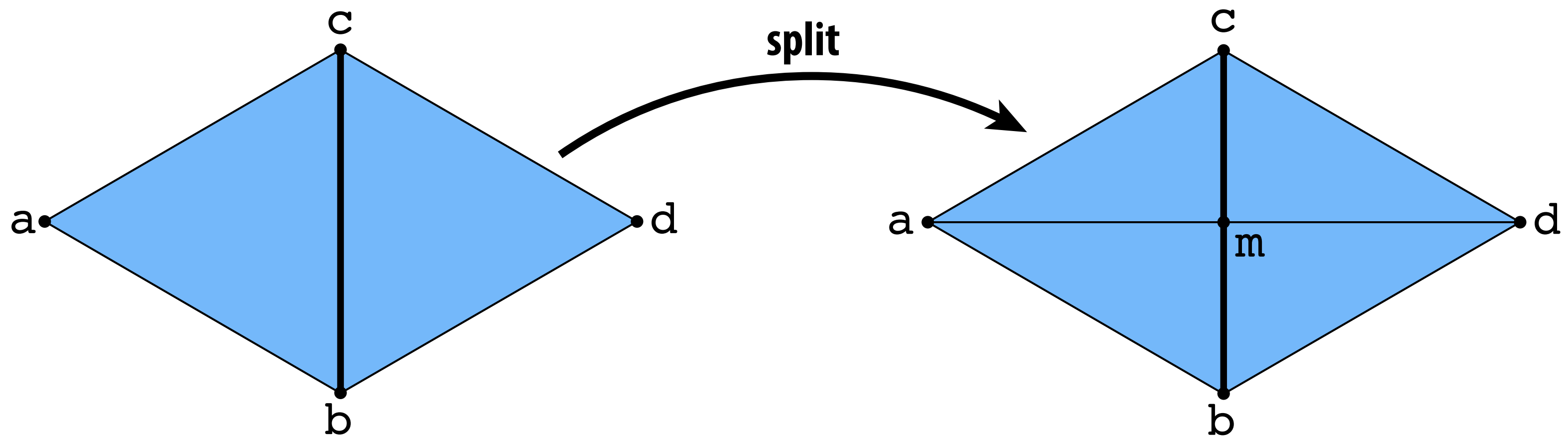
- Triangles (a,b,c) , (b,d,c) become (a,d,c) , (a,b,d) :



- Long list of pointer reassignments (`edge->halfedge = ...`)
- However, no elements created/destroyed.
- Q: What happens if we flip twice?
- Challenge: can you implement edge flip such that pointers are unchanged after two flips?

Edge Split (Triangles)

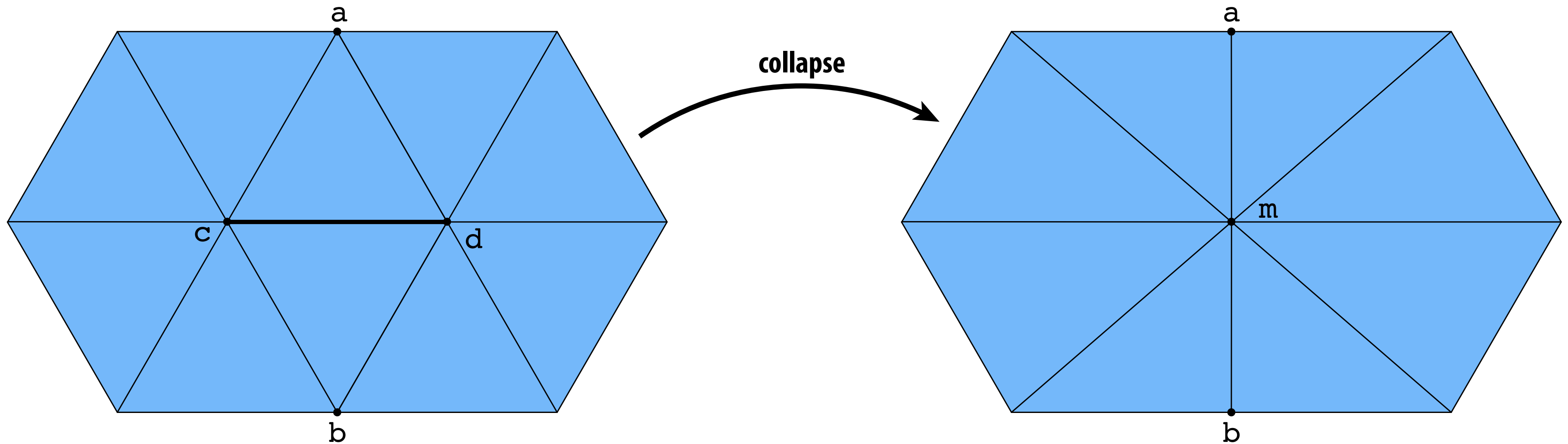
- Insert midpoint m of edge (c,b) , connect to get four triangles:



- This time, have to add new elements.
- Lots of pointer reassignments.
- Q: Can we “reverse” this operation?

Edge Collapse (Triangles)

- Replace edge (b,c) with a single vertex m:



- Now have to delete elements.
- Still lots of pointer assignments!
- Q: How would we implement this with an adjacency list?
- Any other good way to do it? (E.g., different data structure?)

Comparison of Polygon Mesh Data Structures

Case study: triangles.	Adjacency List	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!

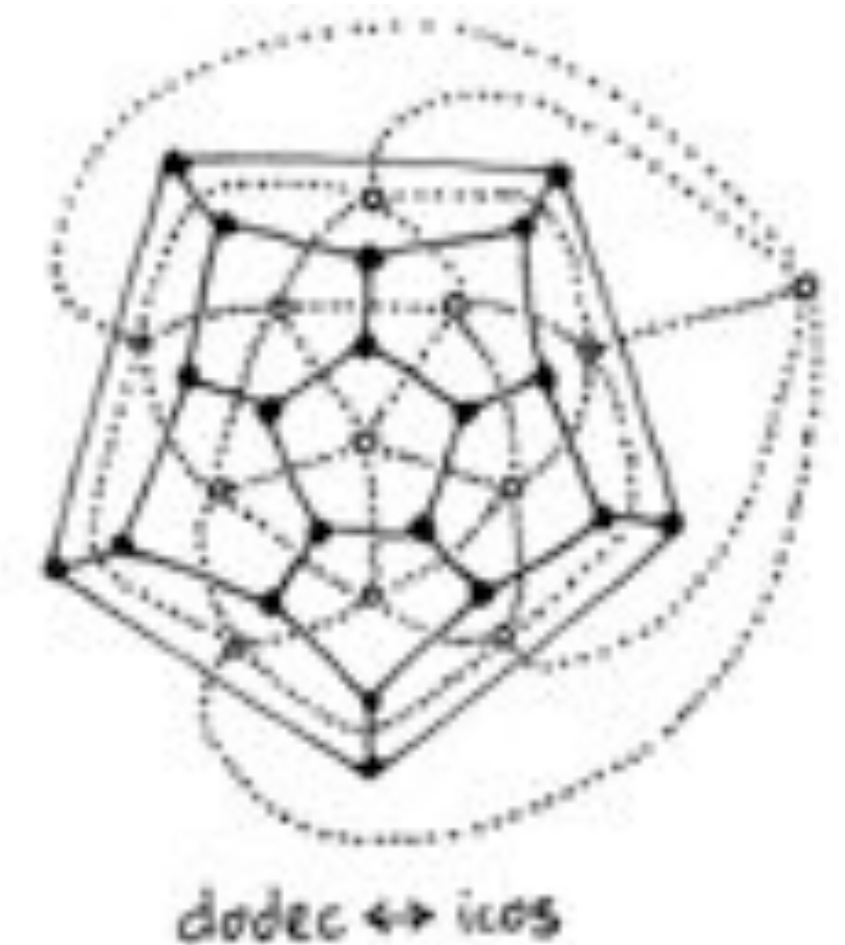
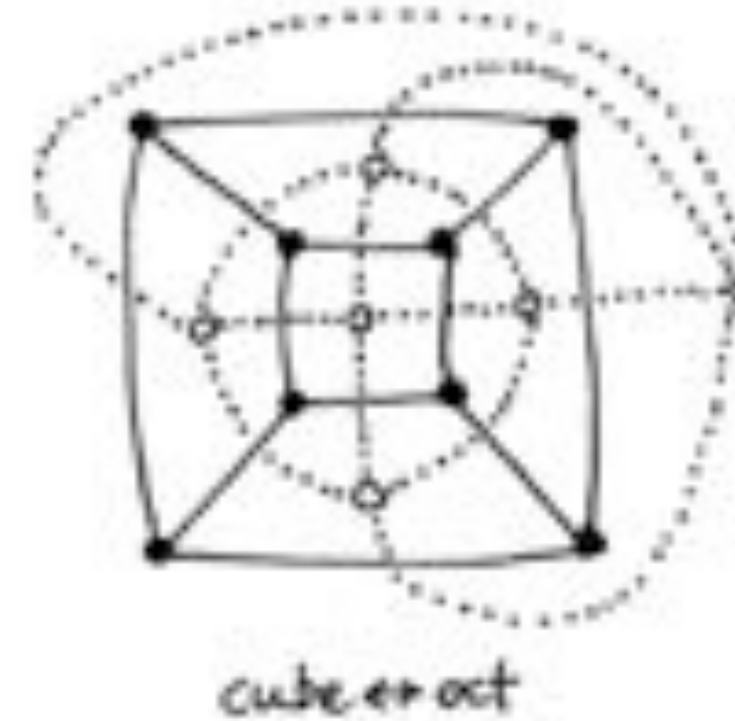
***number of integer values and/or pointers required to encode connectivity
(all data structures require same amount of storage for vertex positions)**

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

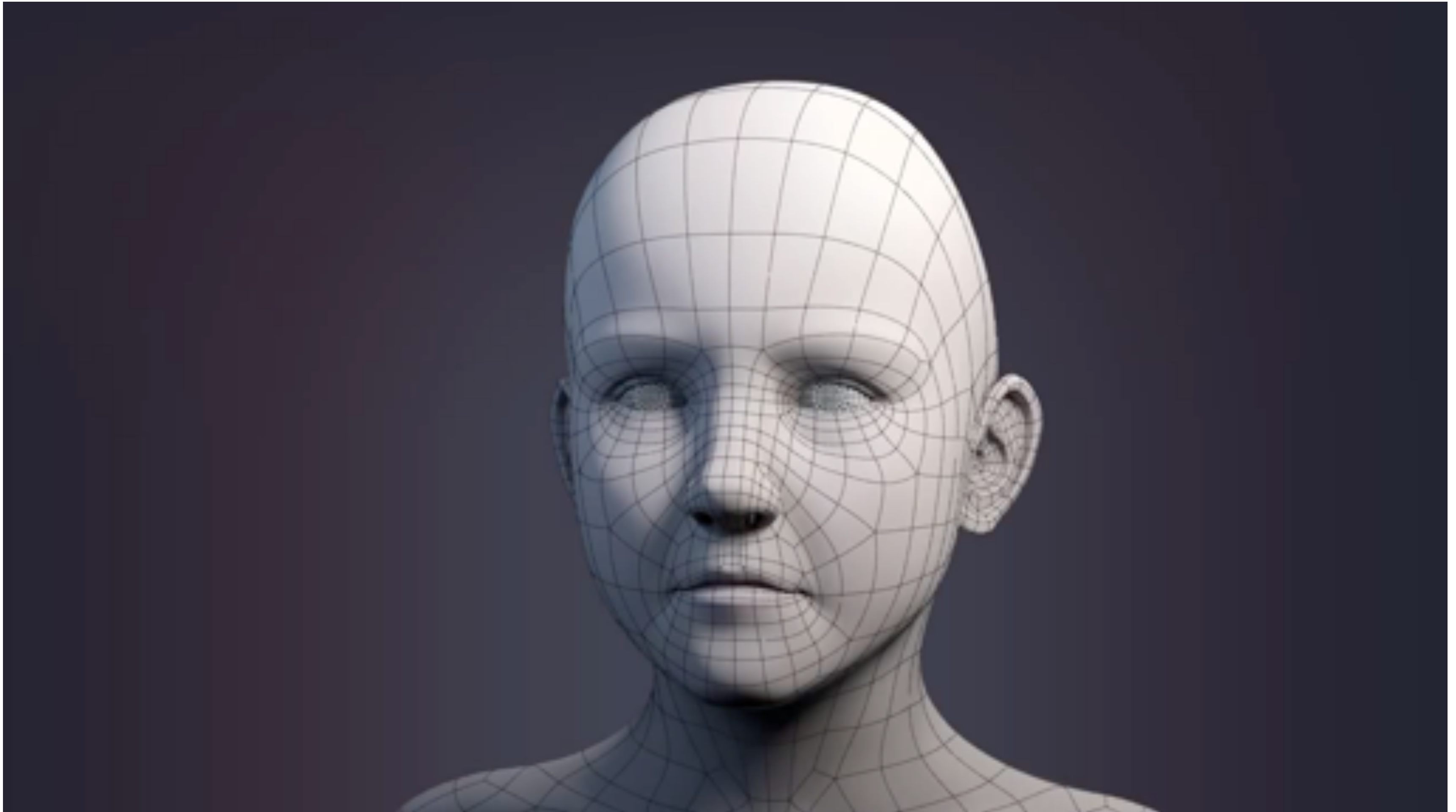
■ Similar tradeoffs relative to simple polygon list:

- **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?)

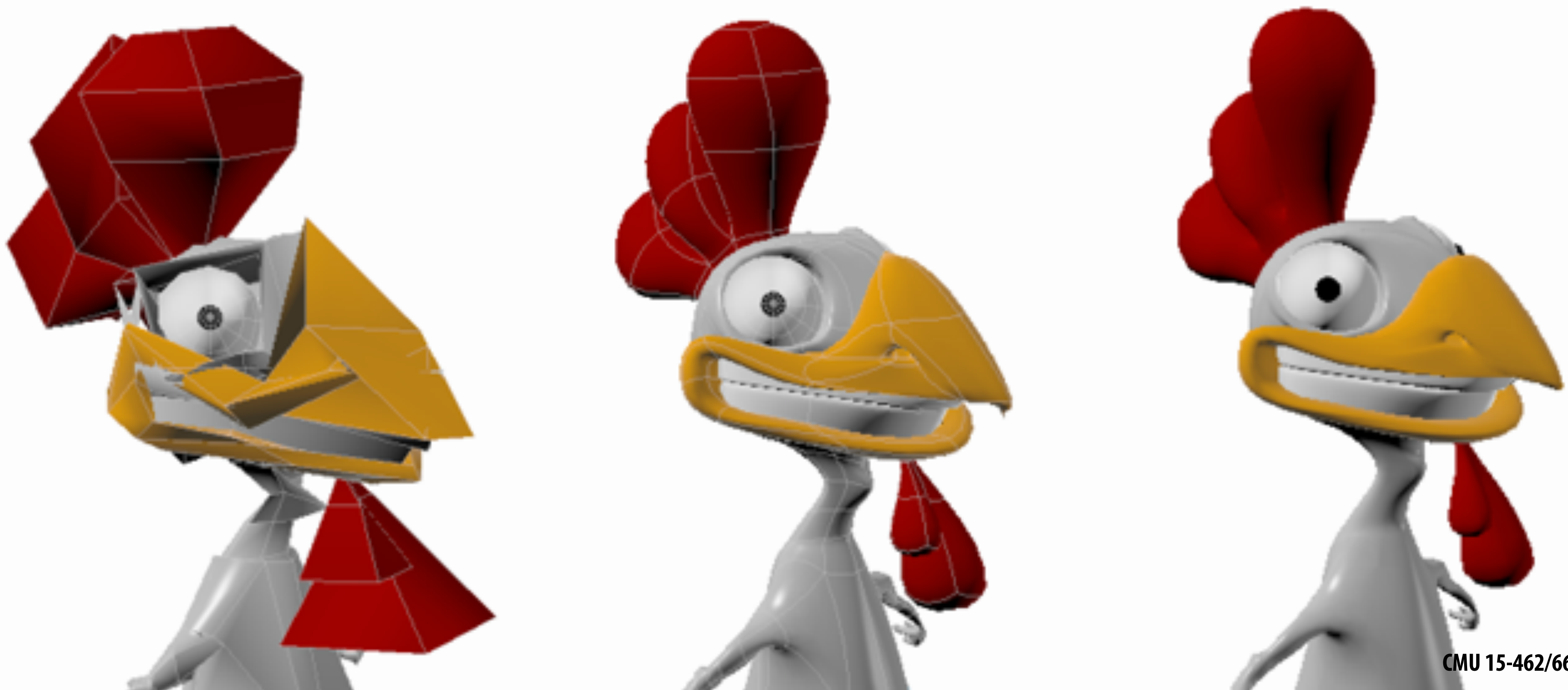
**Ok, but what can we actually do with our
fancy new data structure?**

Subdivision Modeling



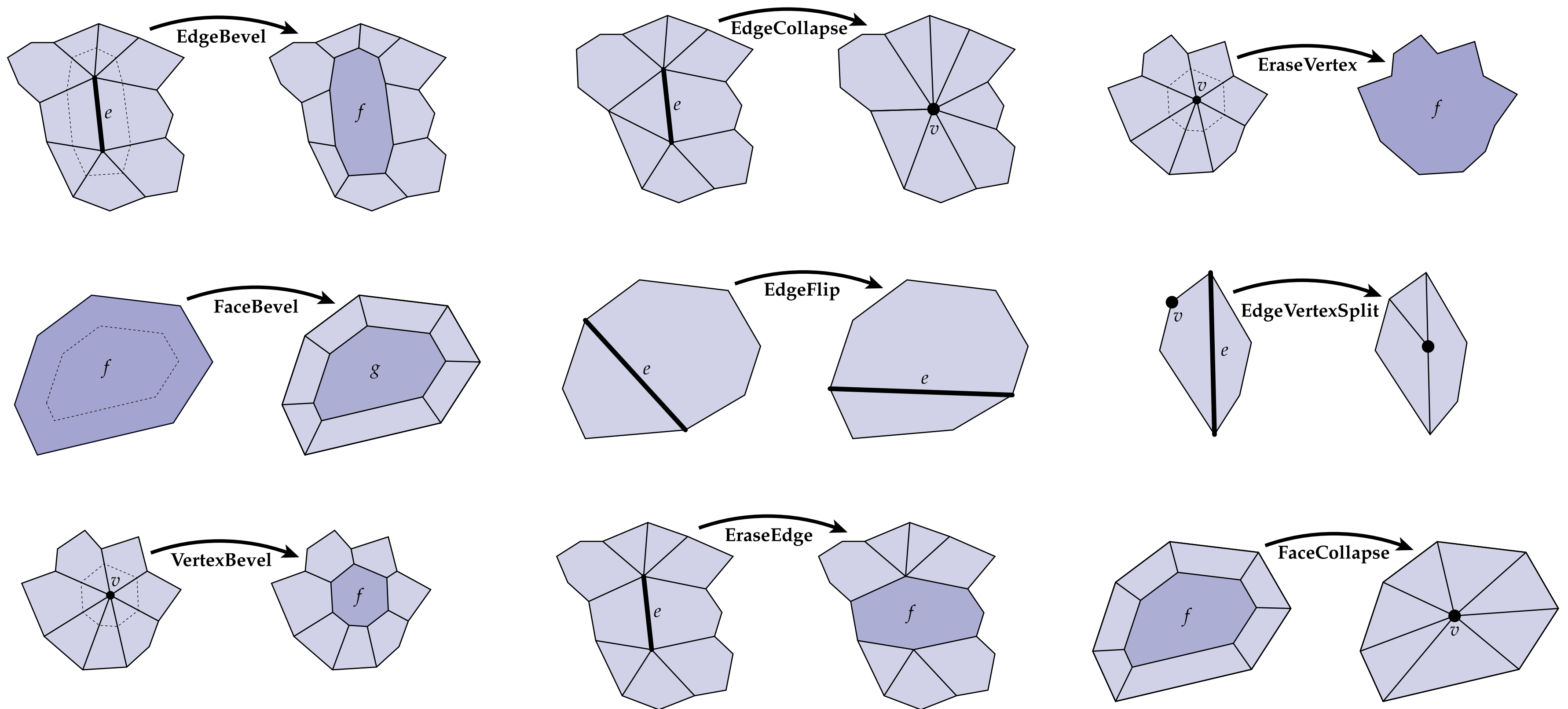
Subdivision Modeling

- **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

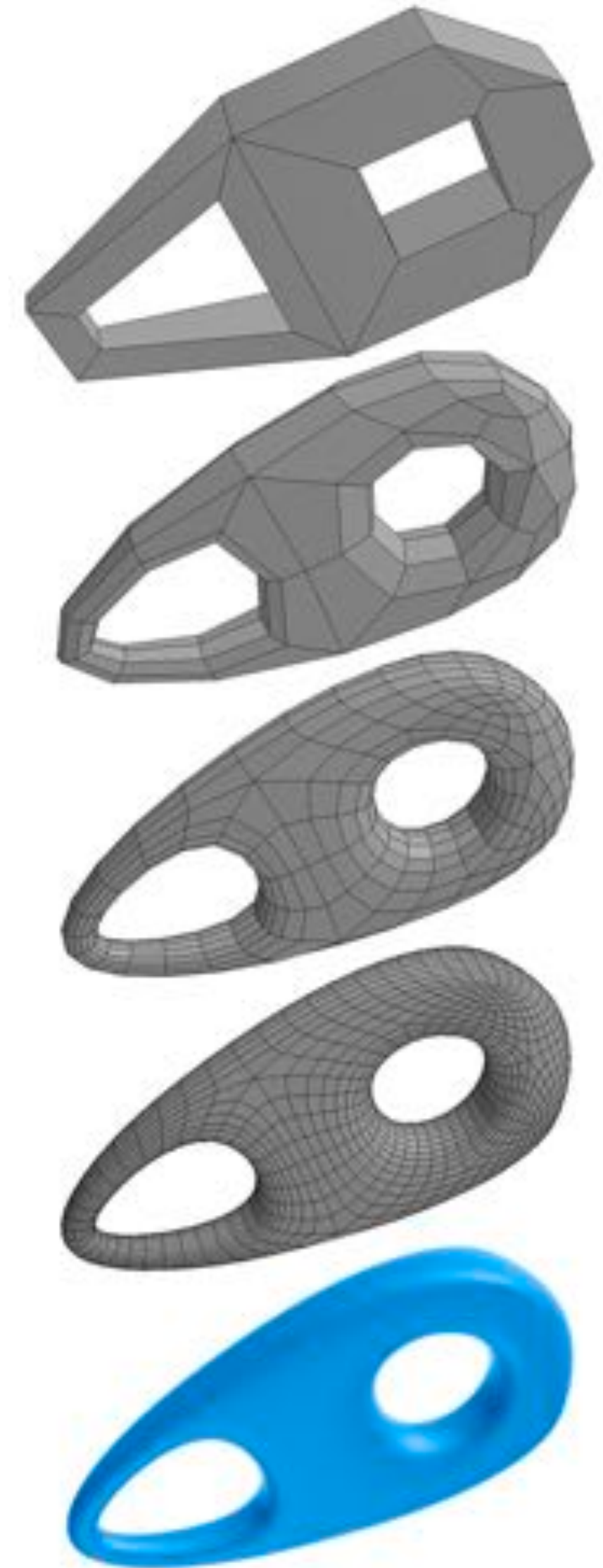
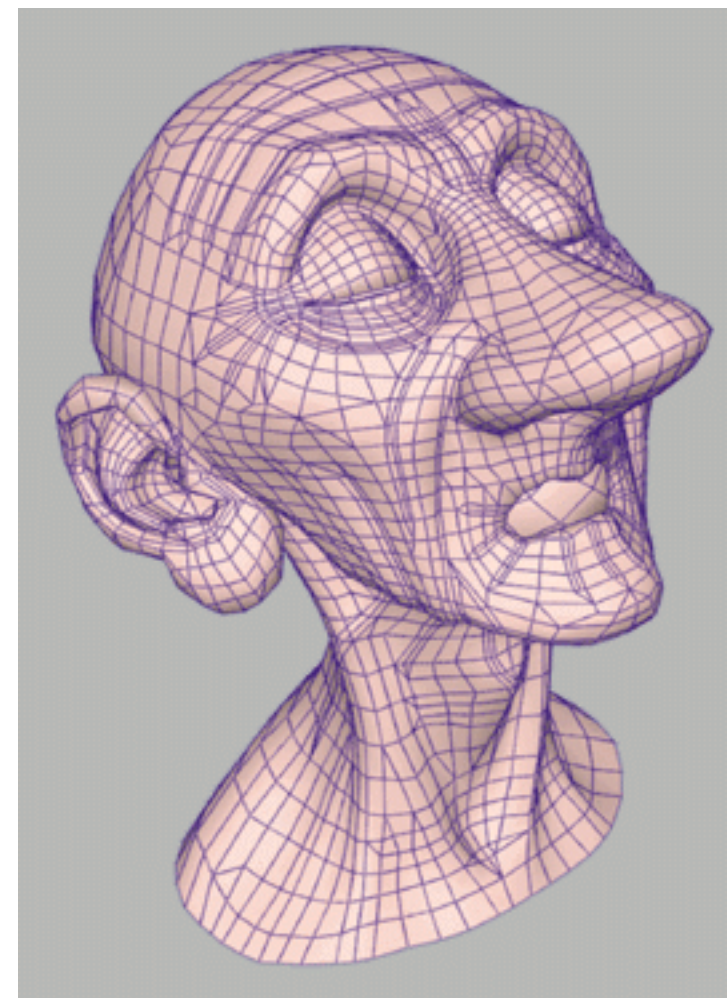
- For general polygon meshes, we can dream up lots of local mesh operations that might be useful for modeling:



...and many, many more!

Global Subdivision

- Start with coarse polygon mesh (“control cage”)
- Subdivide each element
- Update vertices via local averaging
- Many possible rule:
 - Catmull-Clark (quads)
 - Loop (triangles)
 - ...
- Common issues:
 - interpolating or approximating?
 - continuity at vertices?
- Easier than splines for modeling; harder to evaluate pointwise



Next Time: Digital Geometry Processing

- **Extend traditional digital signal processing (audio, video, etc.) to deal with geometric signals:**
 - **upsampling / downsampling / resampling / filtering ...**
 - **aliasing (reconstructed surface gives “false impression”)**
- **Also some new challenges (very recent field!):**
 - **over which domain is a geometric signal expressed?**
 - **no terrific sampling theory, no fast Fourier transform, ...**
- **Often need new data structures & new algorithms**

