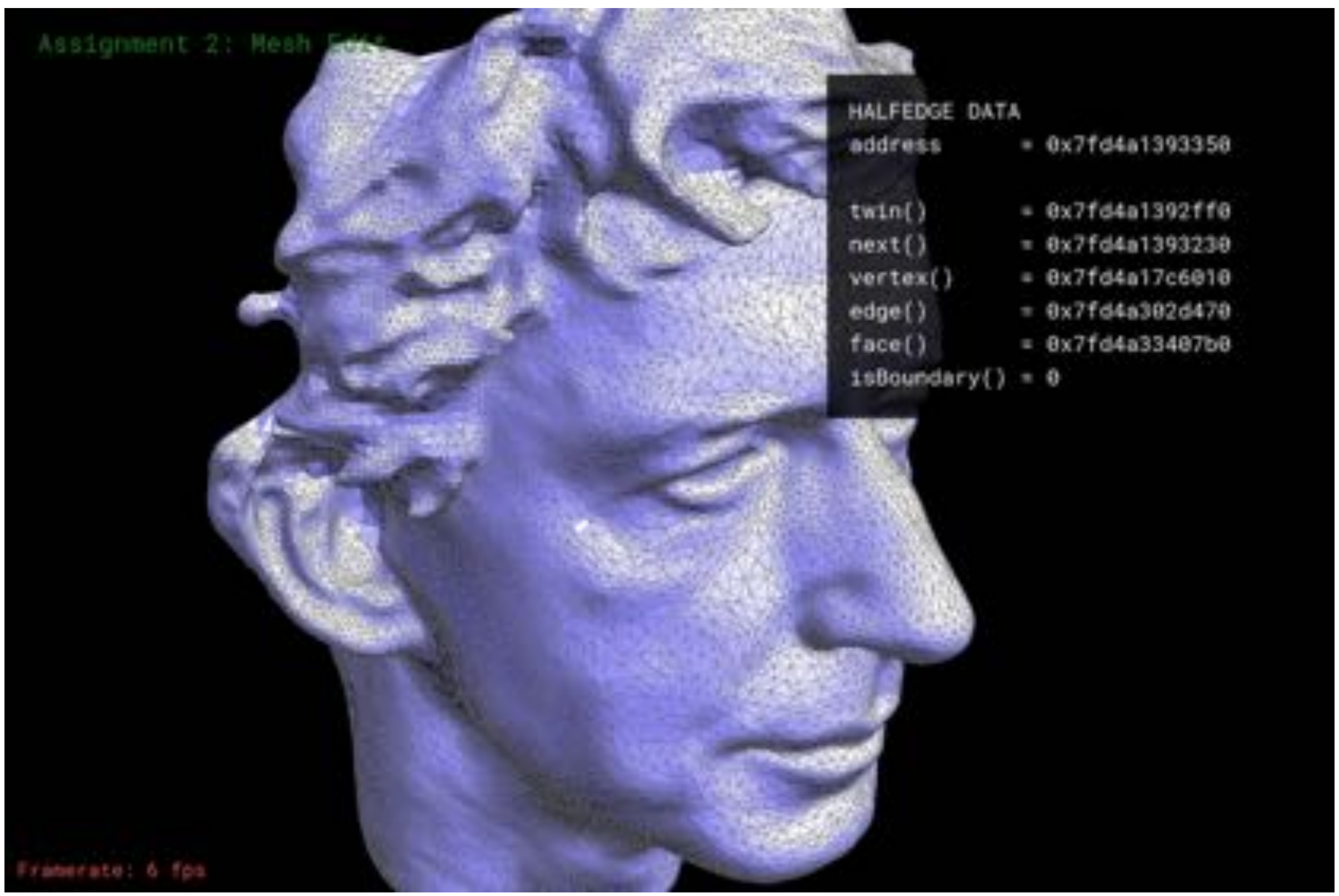
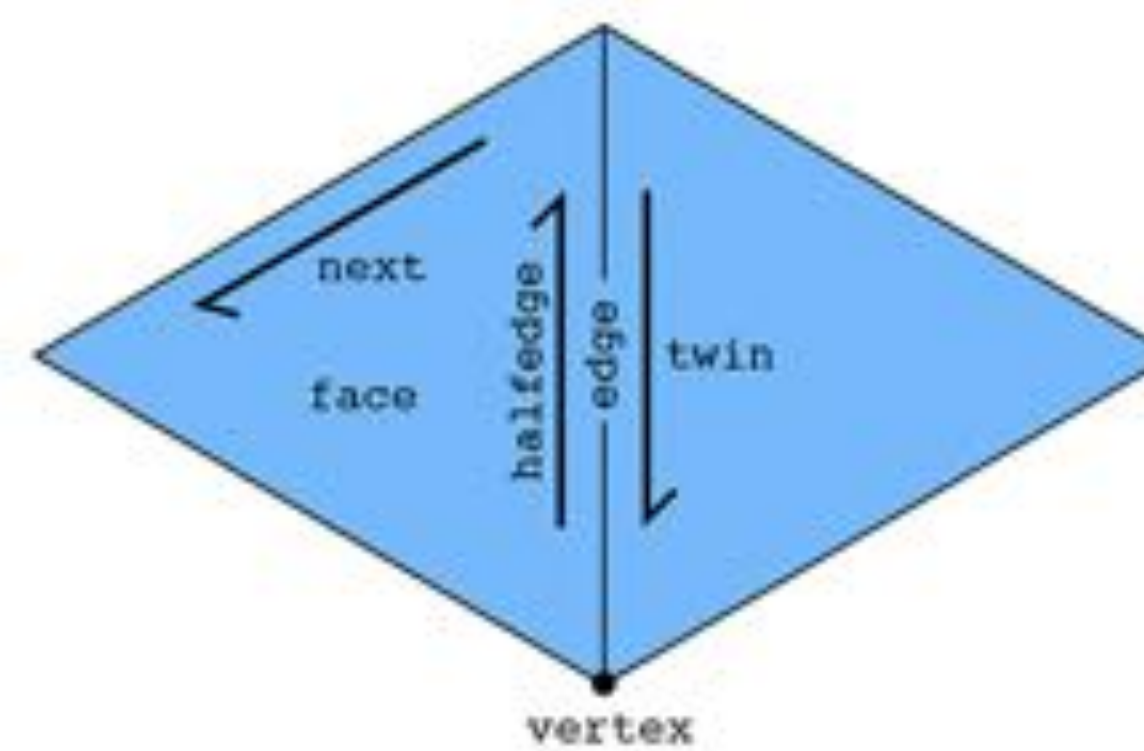
Lecture 10:

Curves, Surfaces & Meshes

Computer Graphics
CMU 15-462/15-662, Spring 2016

Assignment 2 is out!





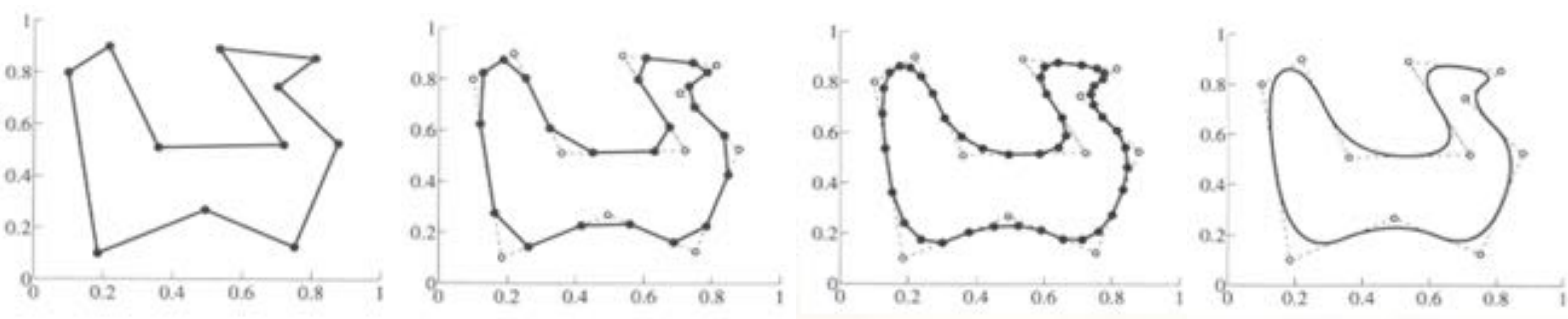
Last time: overview of geometry

- Many types of geometry in nature
- Demand sophisticated representations
- Two major categories:
  - IMPLICIT - "tests" if a point is in shape
  - EXPLICIT - directly "lists" points
- Lots of representations for both
- Today:
  - subdivision curves and surfaces (explicit)
  - what is a surface, anyway?
  - nuts & bolts of polygon meshes
  - geometry processing / resampling

Geometry

Subdivision (Explicit)

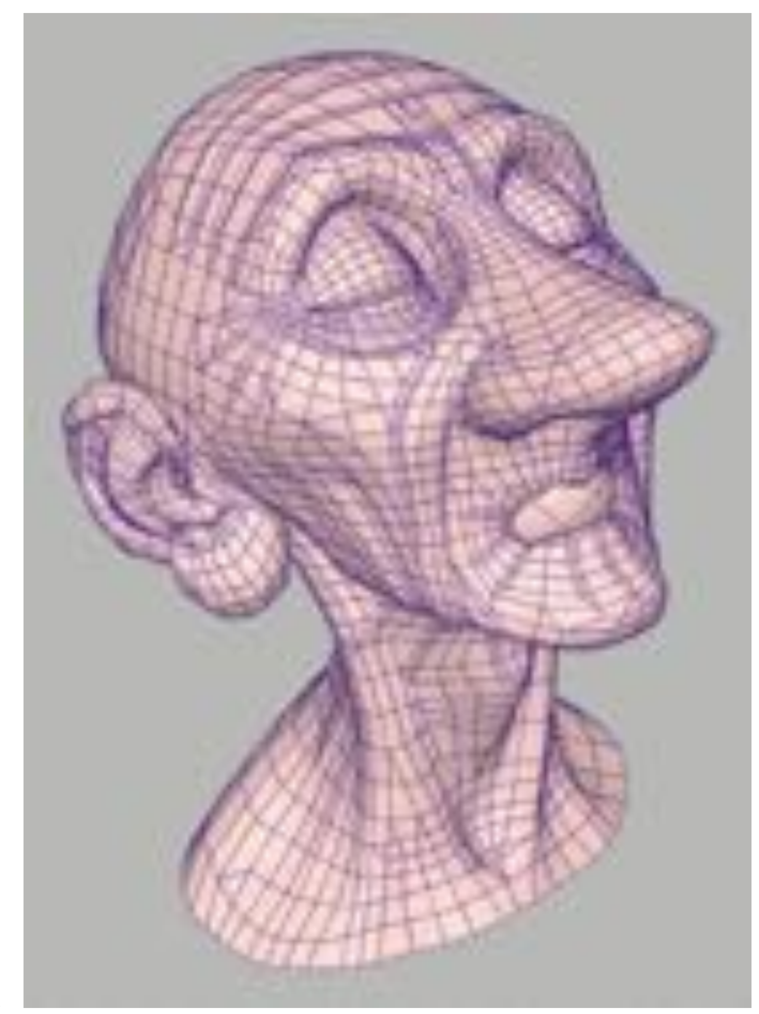
- Alternative starting point for B-spline curves: subdivision
- Start with control curve
- Insert new vertex at each edge midpoint
- Update vertex positions according to fixed rule
- For careful choice of averaging rule, yields smooth curve
    - Average with “next” neighbor (Chaikin): quadratic B-spline

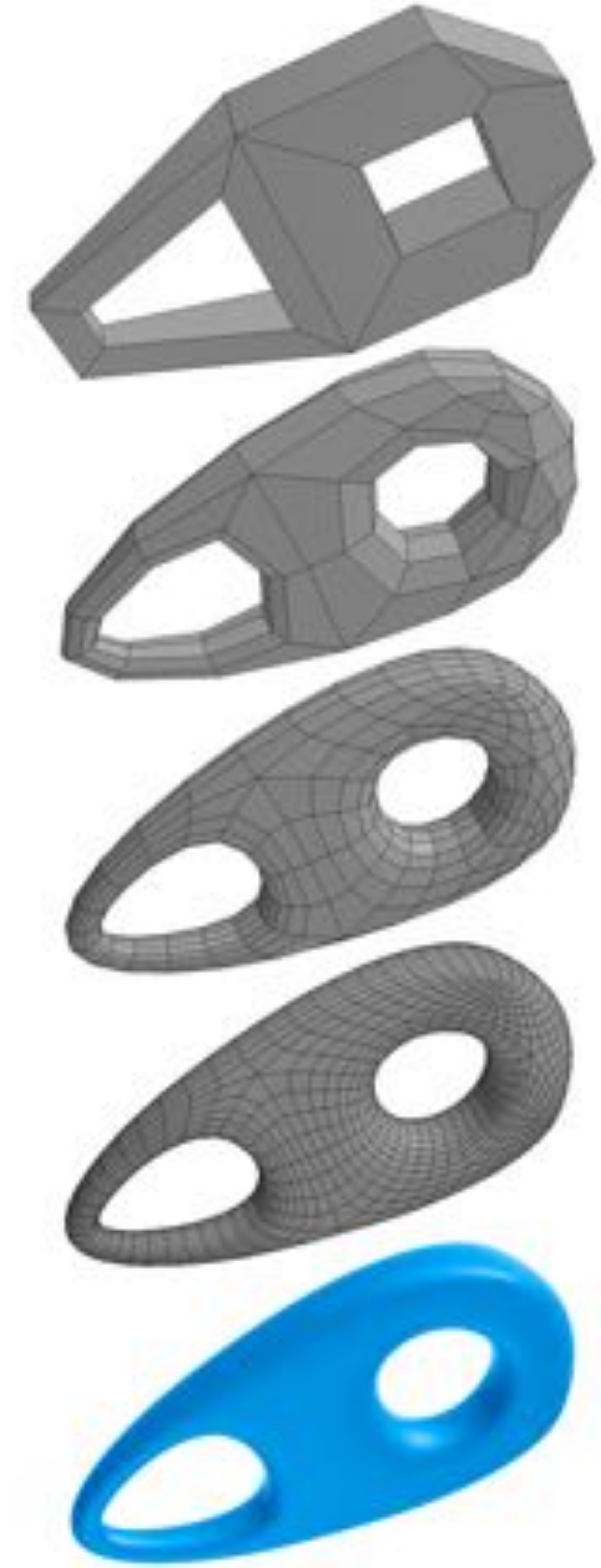


Slide cribbed from Don Fussell.

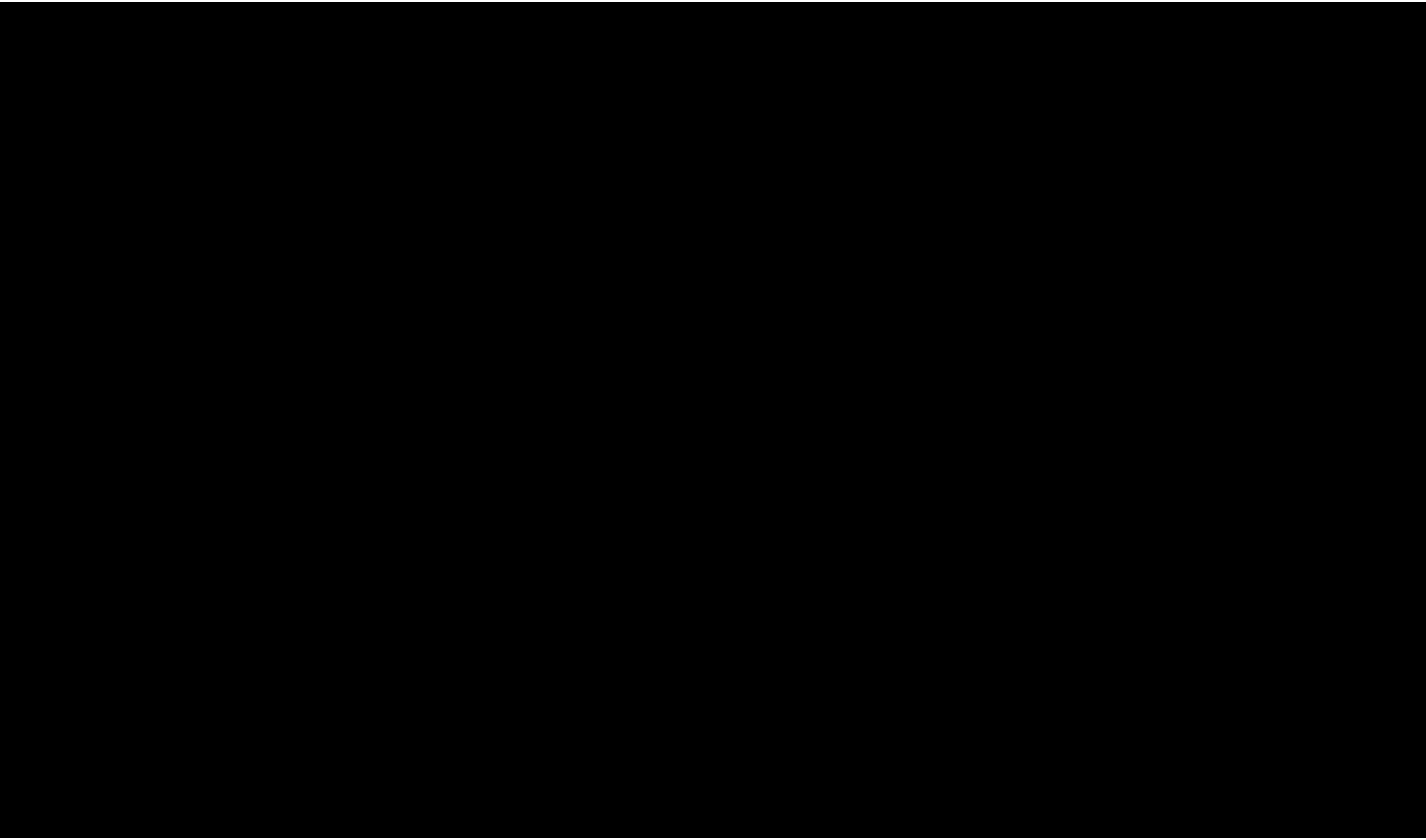
Subdivision Surfaces (Explicit)

- 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 NURBS for modeling; harder to guarantee continuity





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



Q: What is a "surface?"

A: Oh, it's a 2-dimensional manifold.

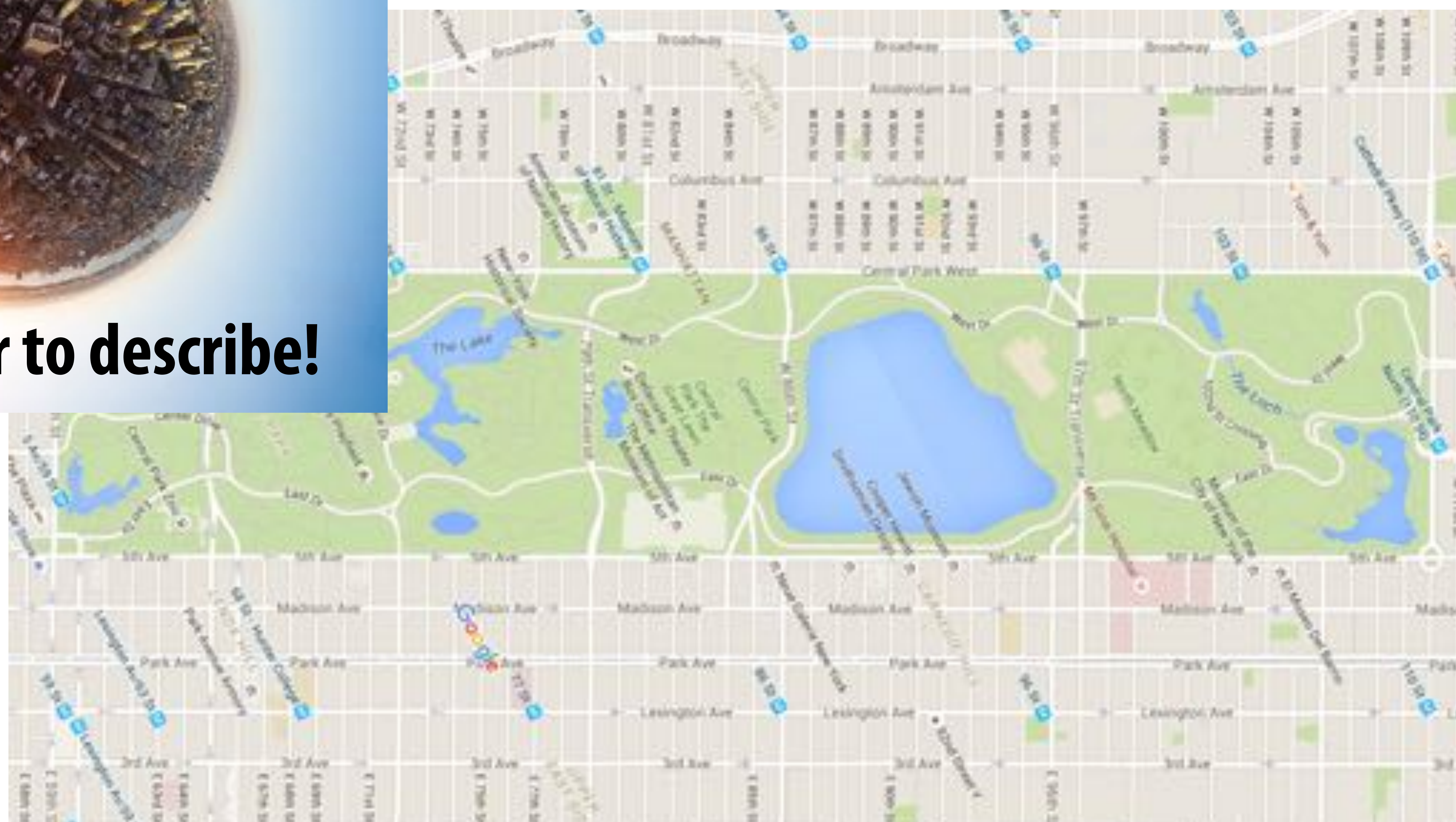
Q: Ok… but what the heck is a manifold?

The Earth looks flat, if you get close enough

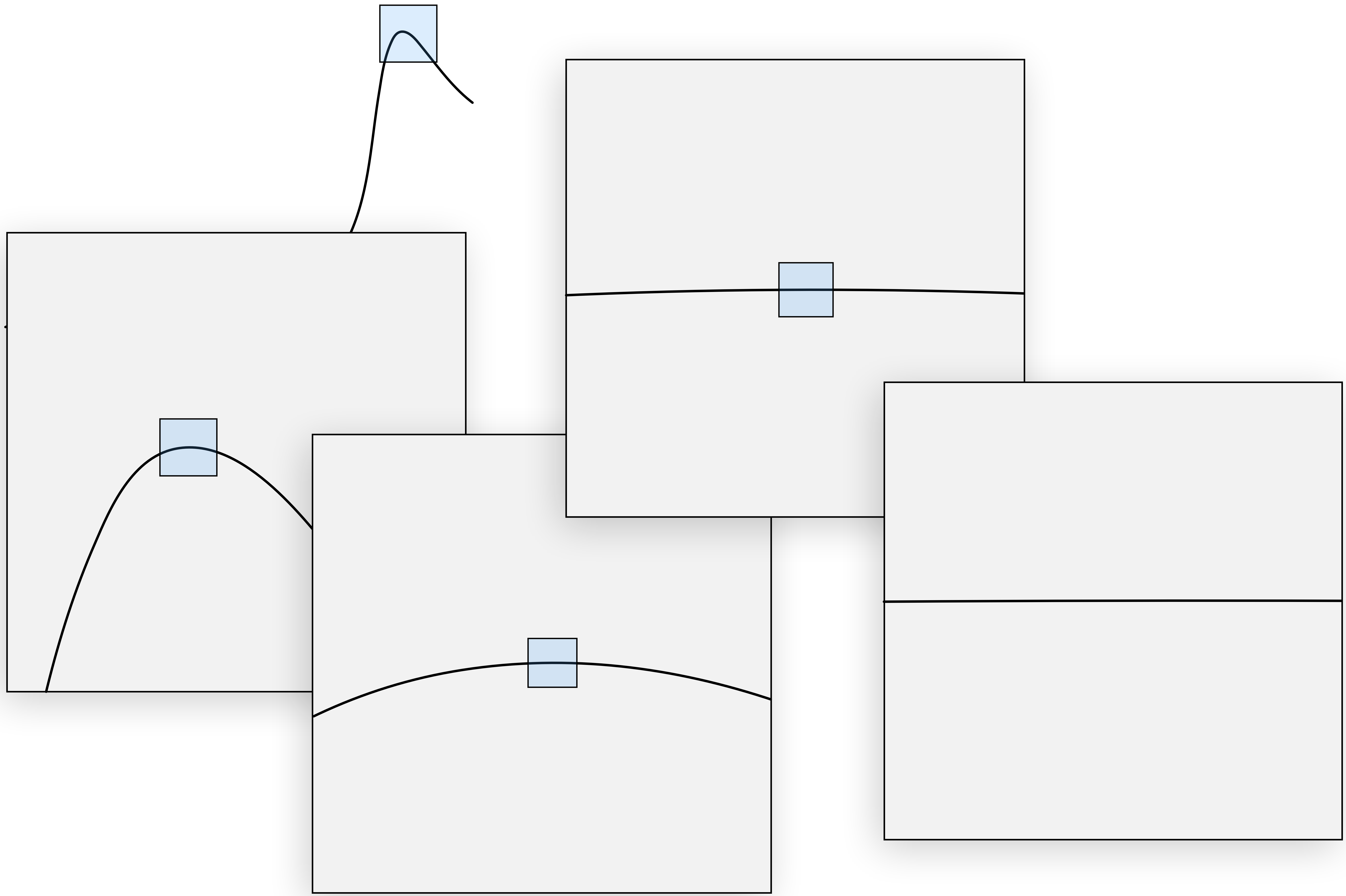
Can pretend we're on a grid:

The Earth looks flat, if you get close enough

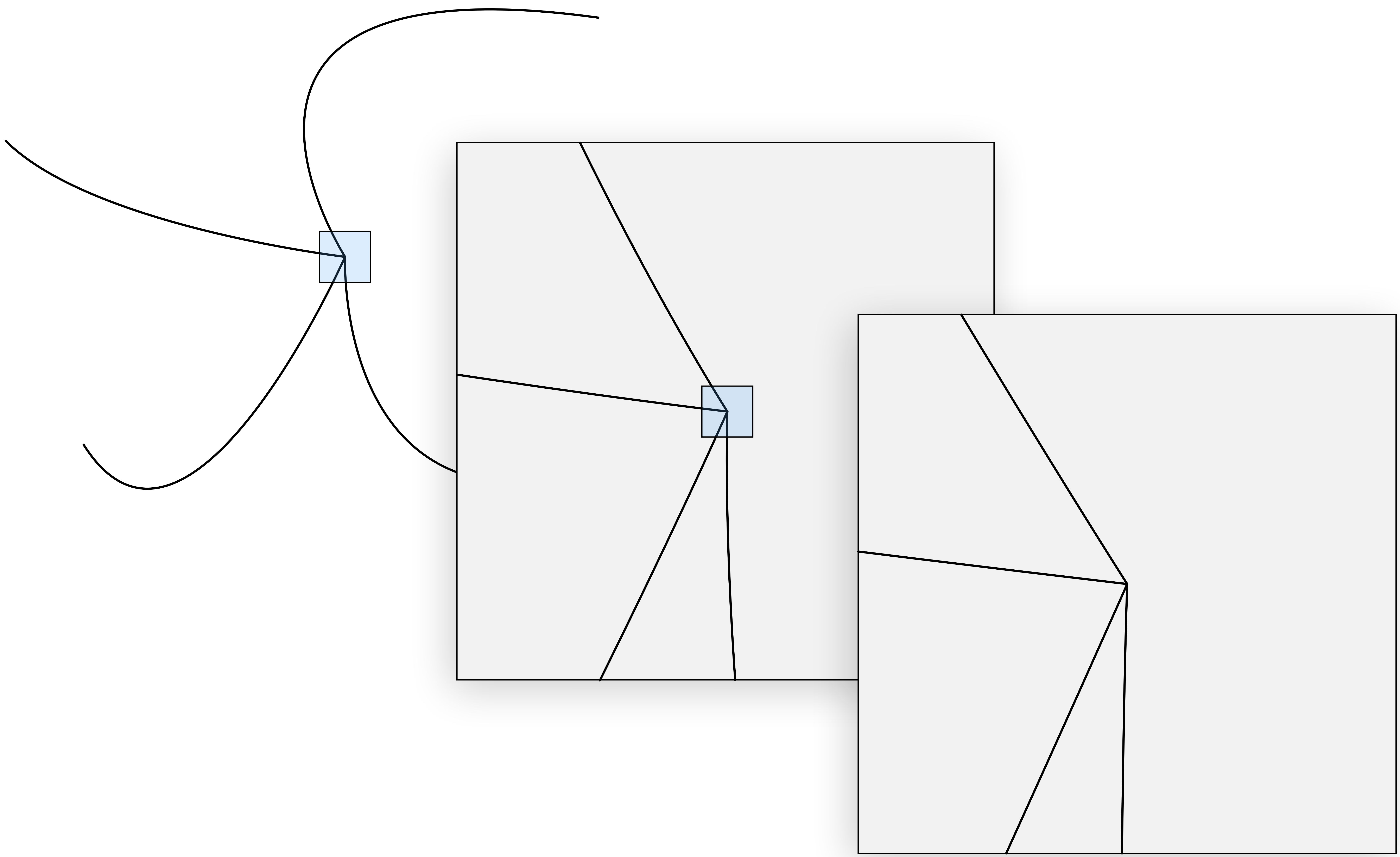
Can pretend we're on a grid:



A smooth manifold also looks flat close up

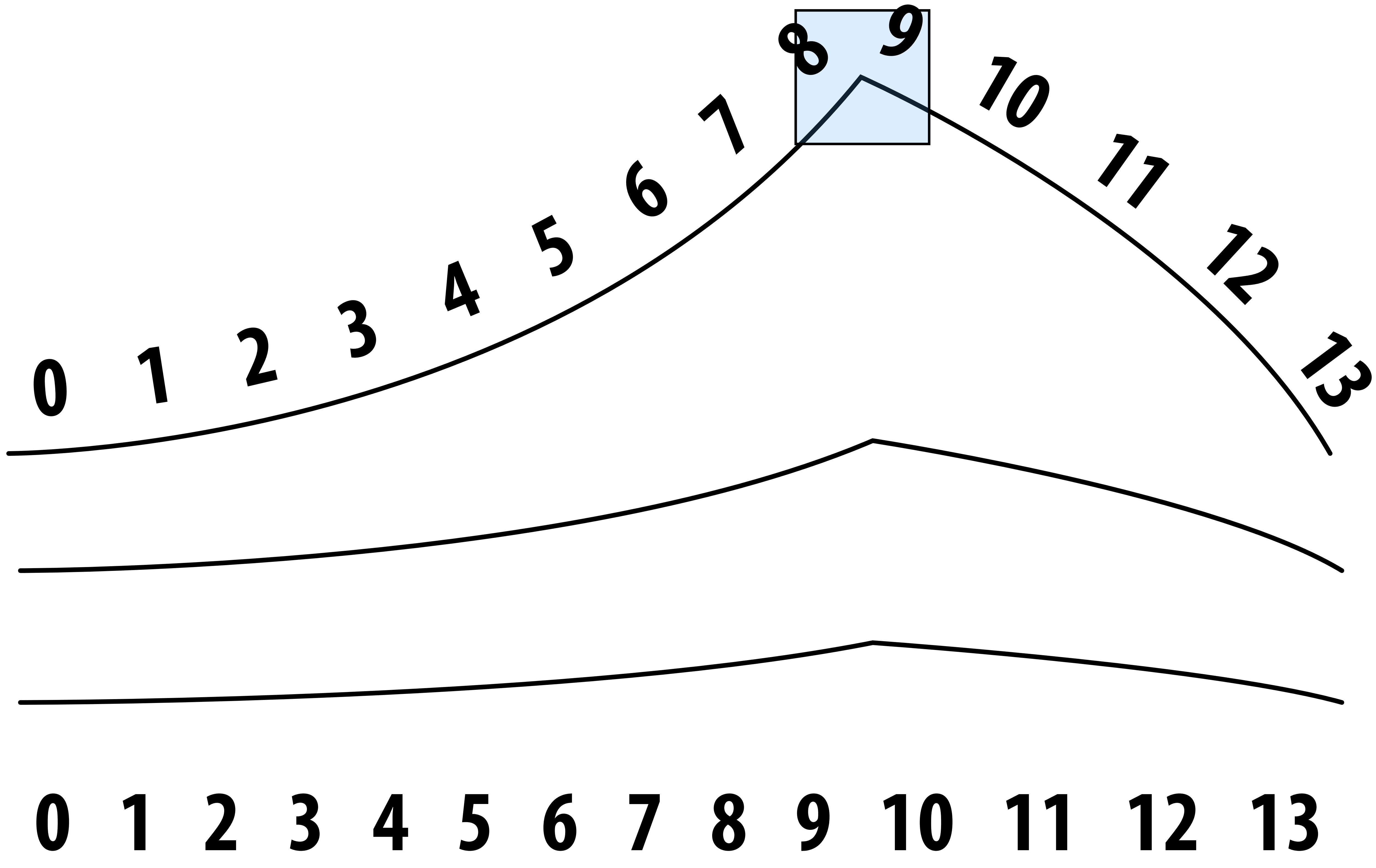


Not all curves are smooth manifolds



No matter how close we get, doesn't look like a single line!

What about sharp corners?



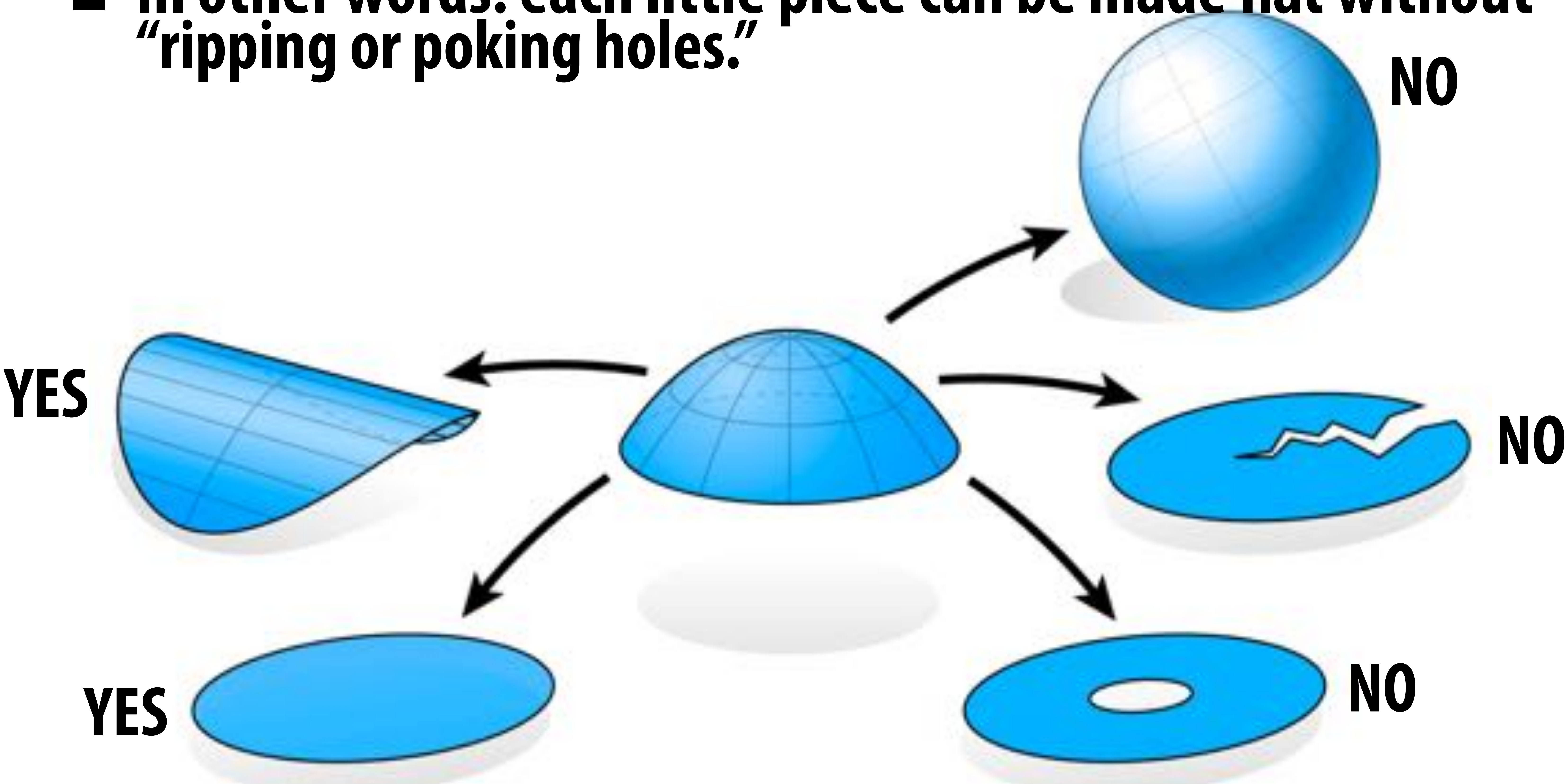
Can easily be flattened into a line.

Can still assign coordinates (just like Manhattan!)

...But is it a manifold?

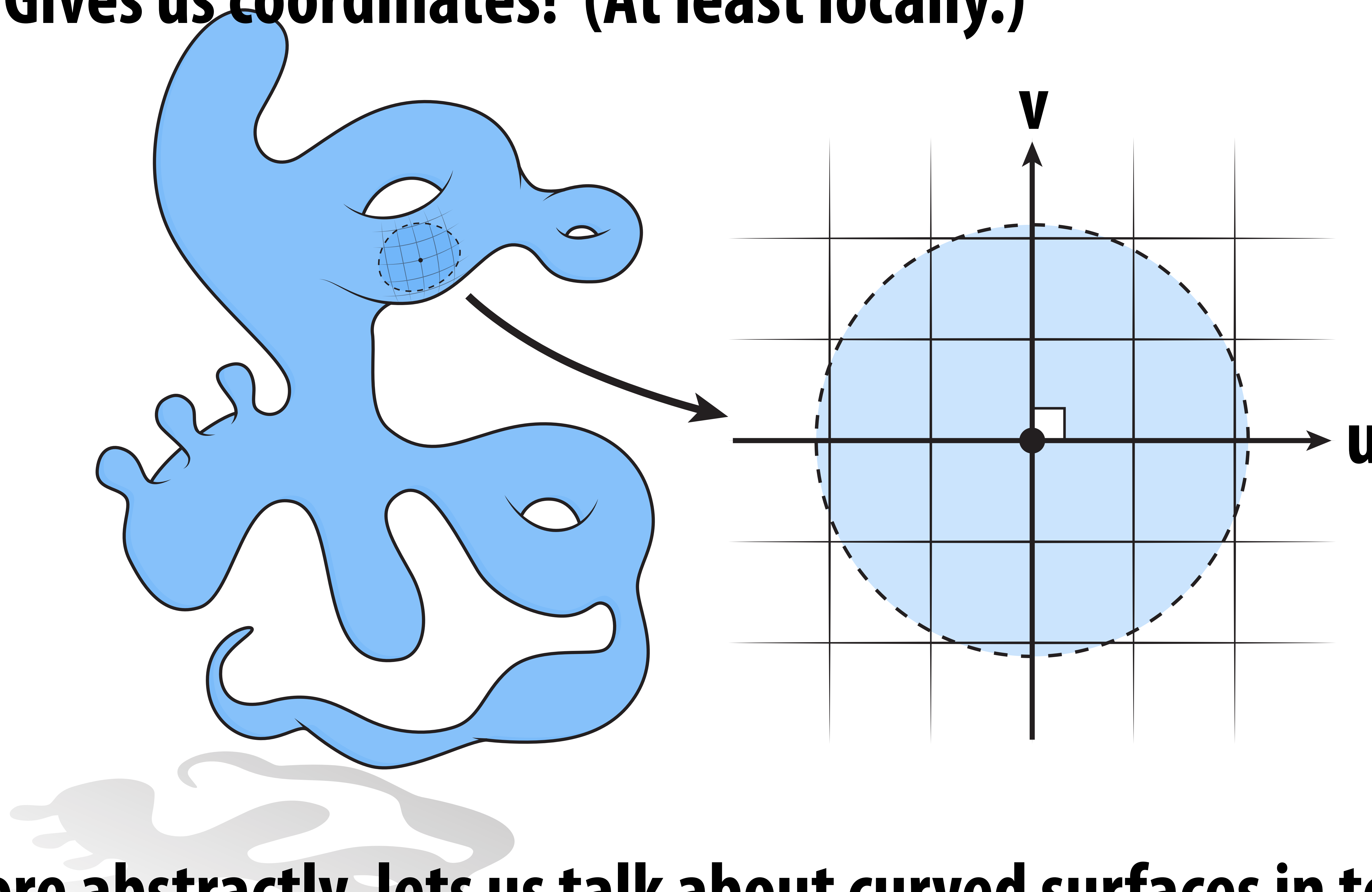
Definition of a manifold

- "A subset S of R^m is an n-manifold if every point p in S is contained in a neighborhood that can be mapped bijectively and continuously (both ways) to the open ball in R^n."
- In other words: each little piece can be made flat without "ripping or poking holes."



Why is the manifold property valuable?

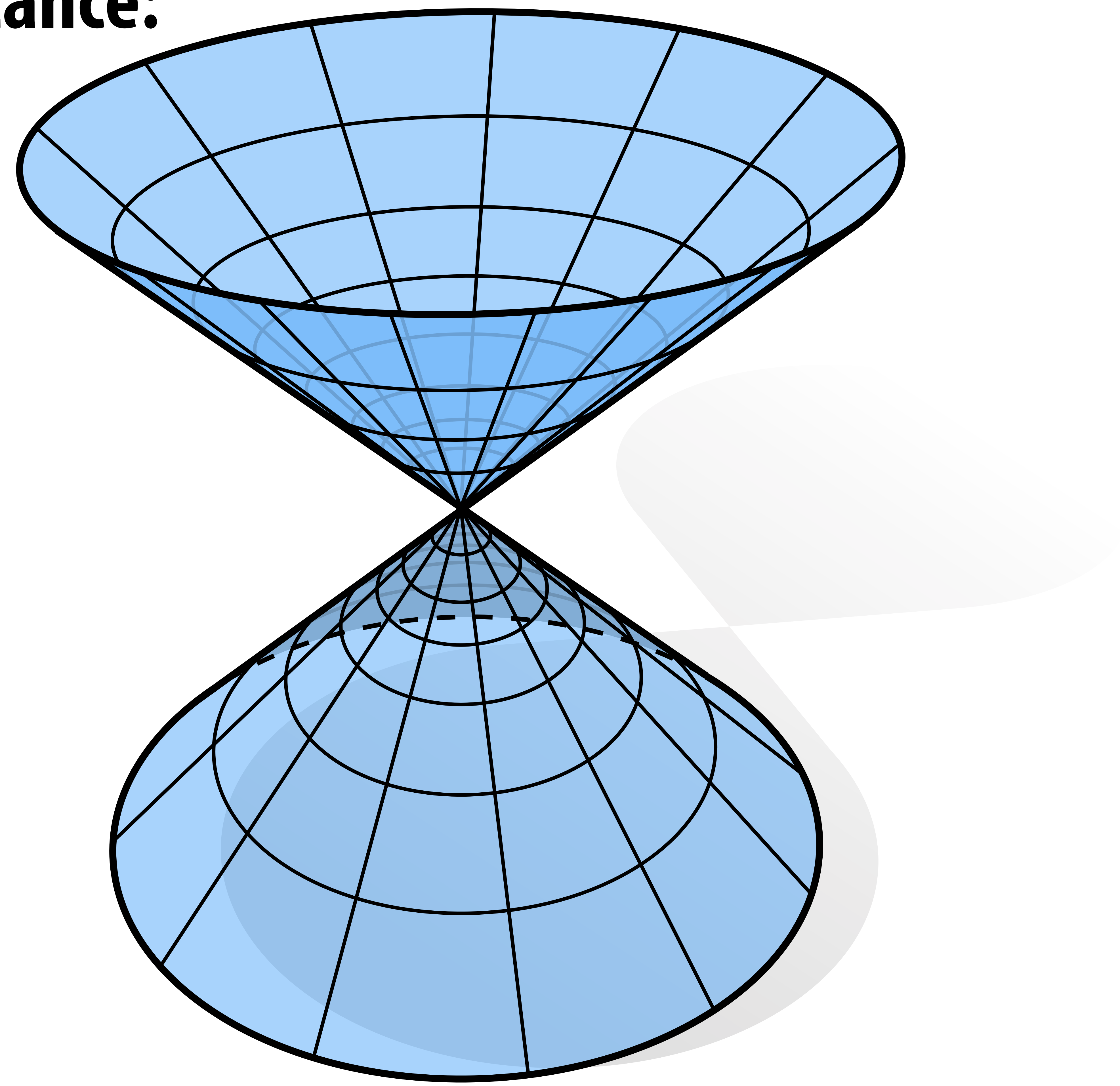
- Makes life simple: all surfaces look the same (at least locally).
- Gives us coordinates! (At least locally.)



- More abstractly, lets us talk about curved surfaces in terms of familiar tools: vector calculus & linear algebra.

Isn’t every shape manifold?

- No, for instance:

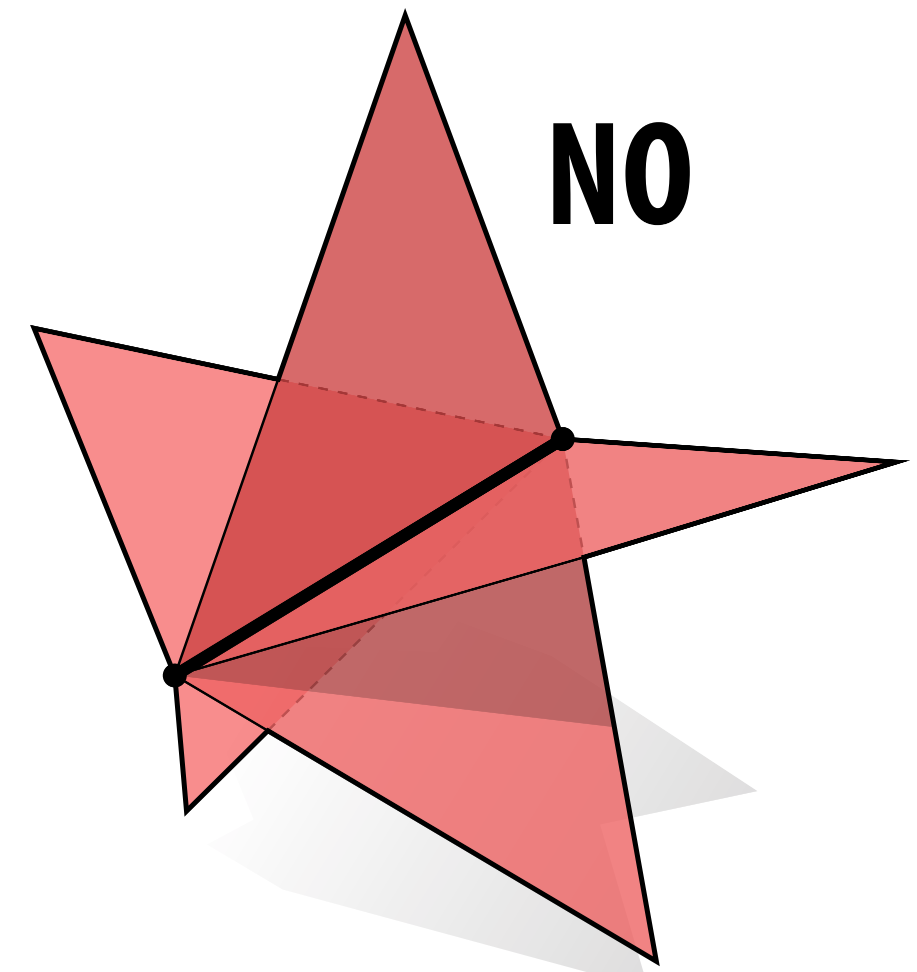


No way to put a (simple) coordinate system on the center point!

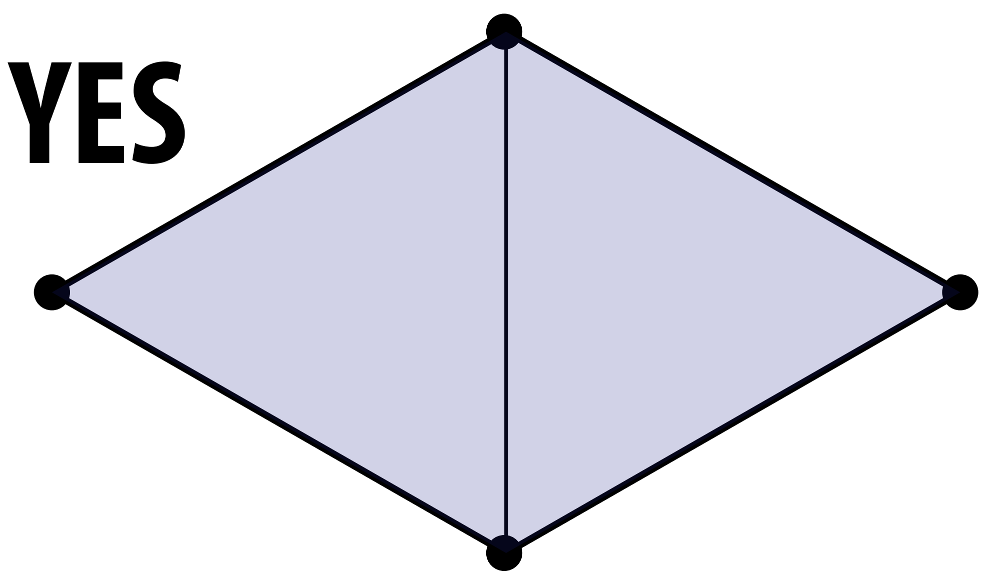
What about discrete surfaces?

- Surfaces made of, e.g., triangle are no longer smooth.
- But they can still be manifold:
  - two triangles per edge (no “fins”)
  - every vertex looks like a “fan”

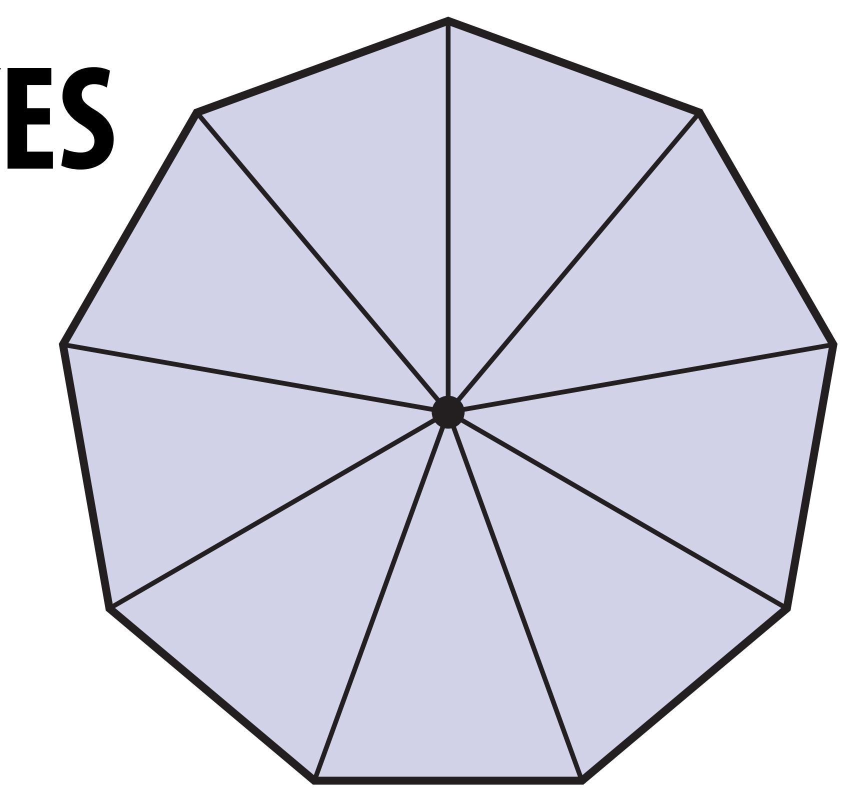
NO

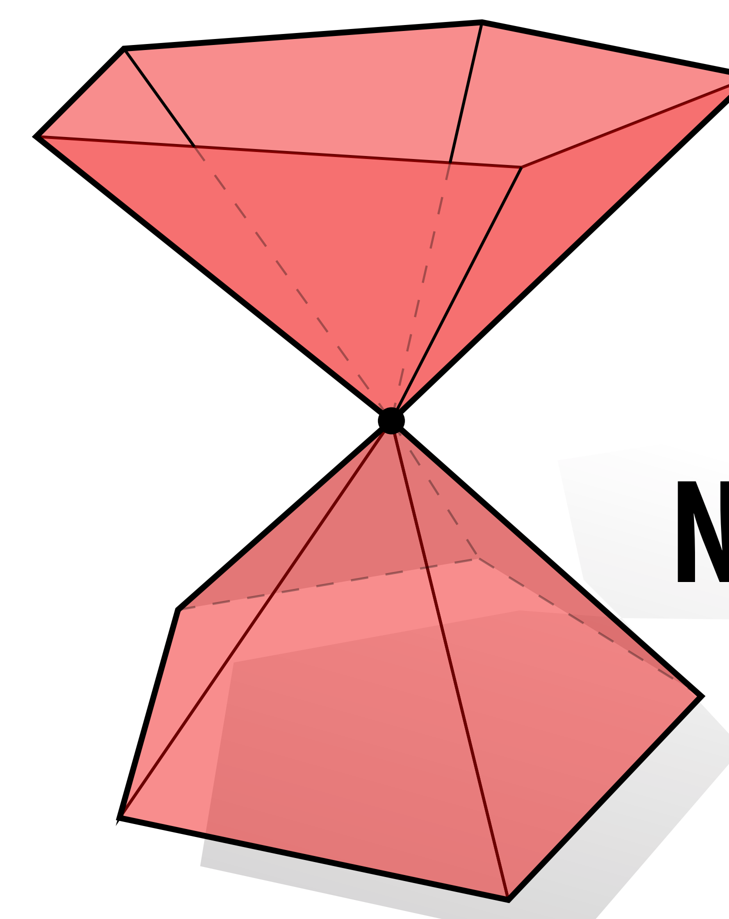


YES



YES



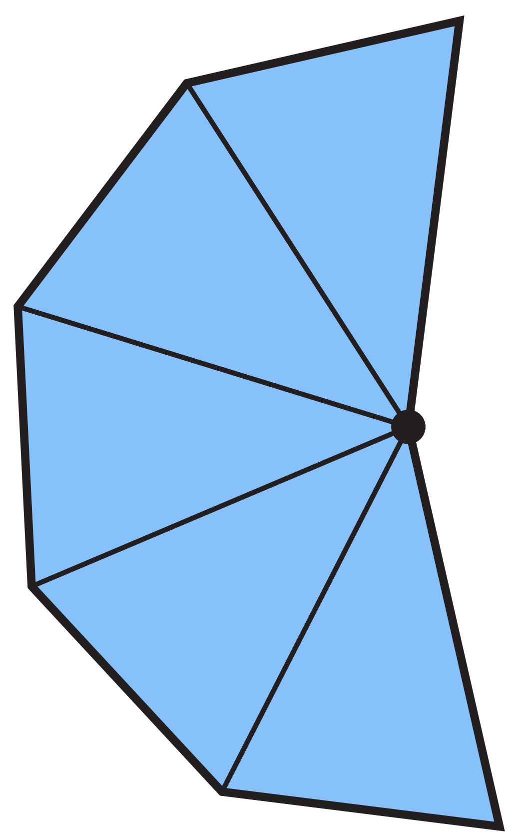


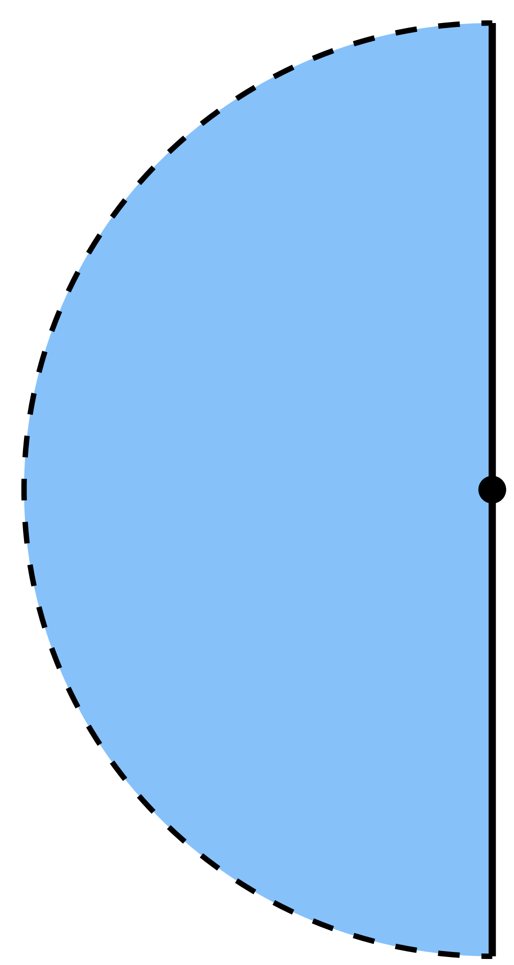
NO

- Why? Simplicity.
  - no special cases to handle
  - keeps data structures (reasonably) simple)

What about boundary?

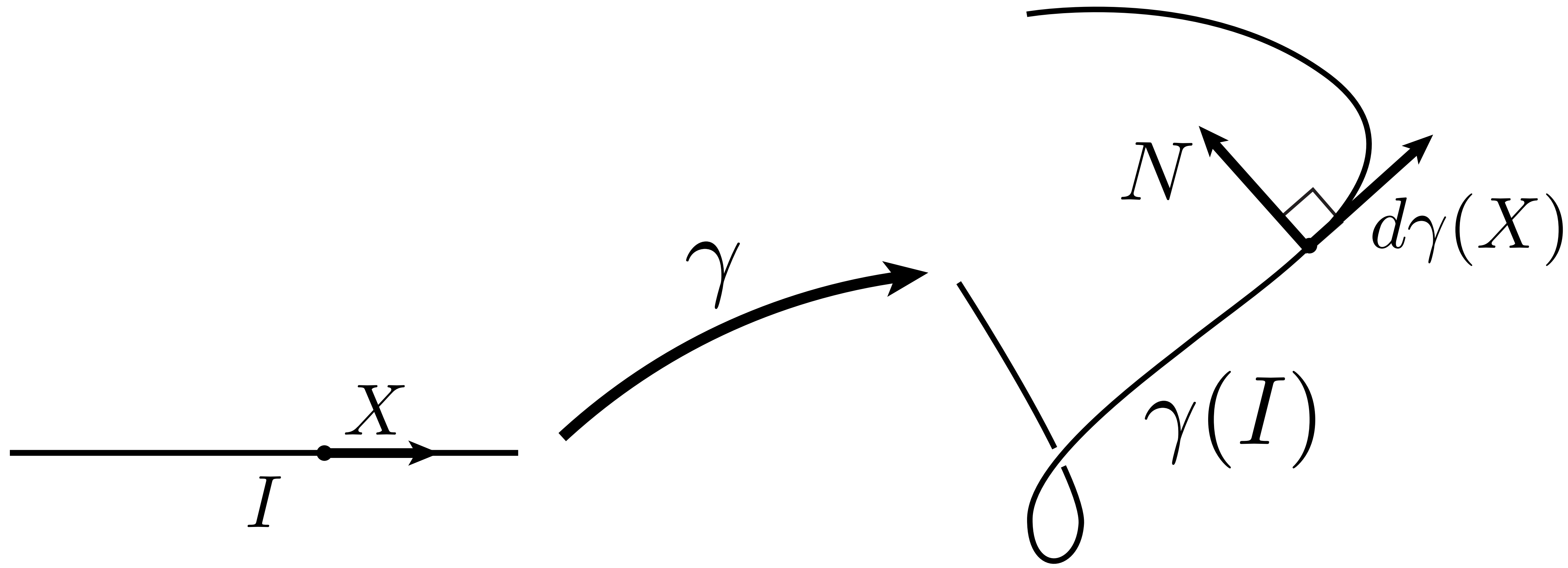
- The boundary is where the surface “ends.”
- E.g., waist & ankles on a pair of pants.
- Locally, looks like a half disk
- Globally, each boundary forms a loop

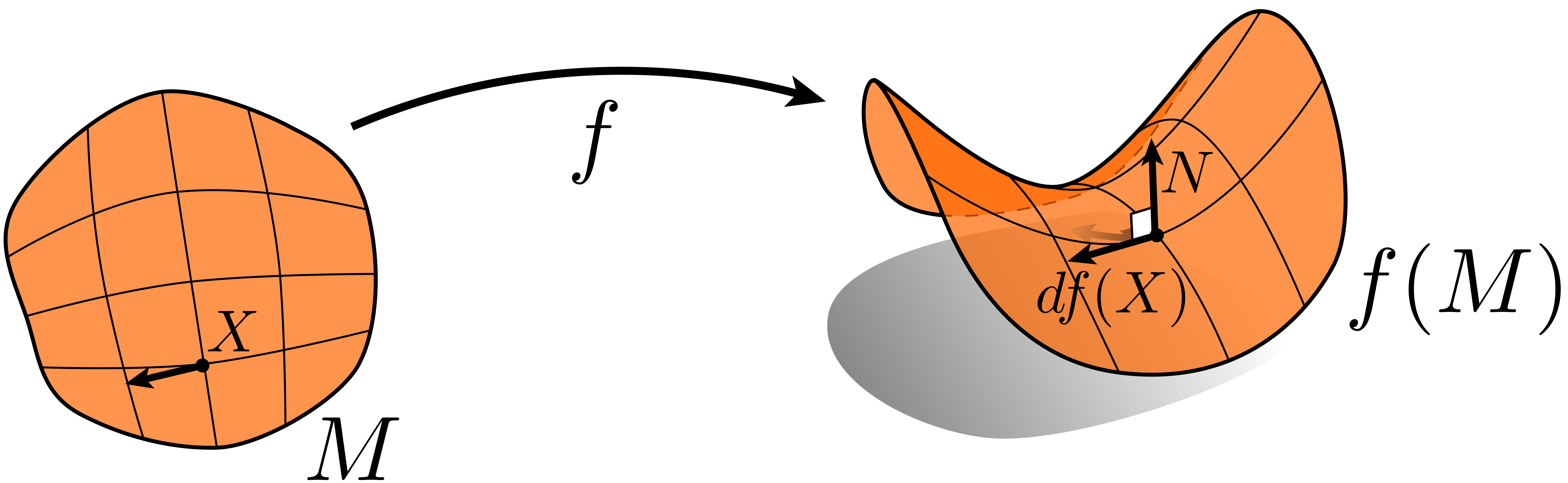




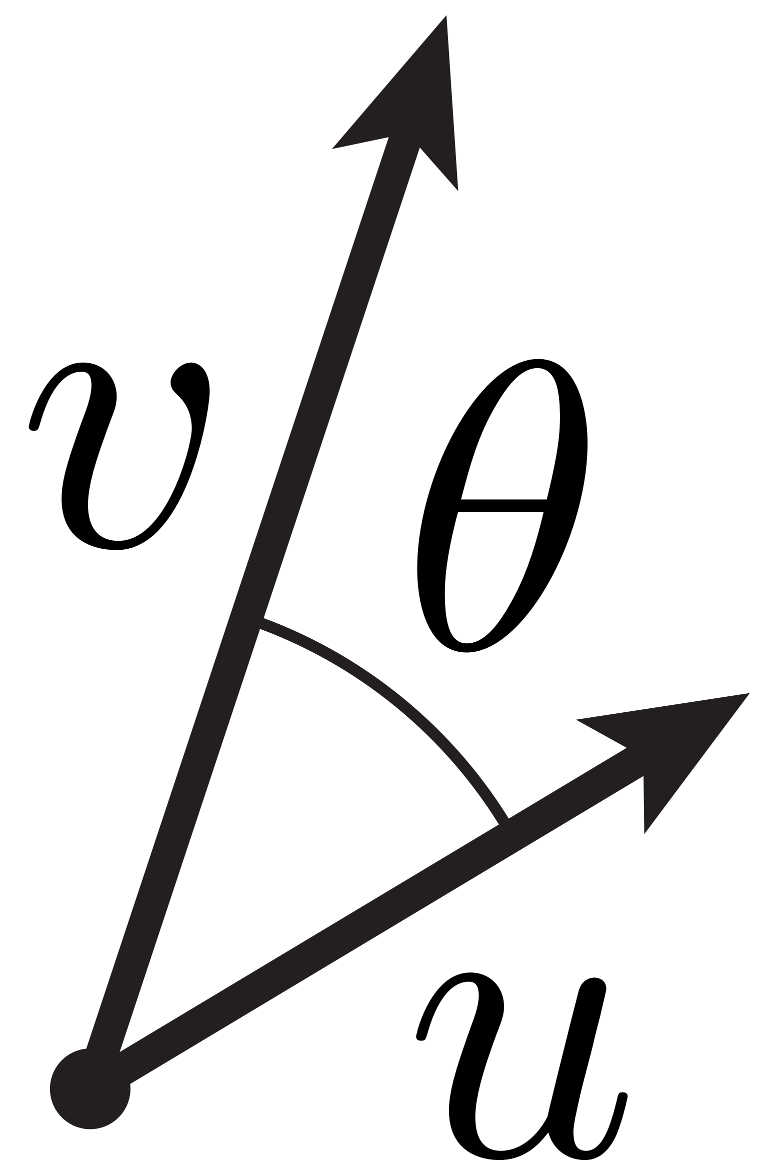
- Triangle mesh:
  - one triangle per boundary edge
  - boundary vertex looks like “pacman”

Anatomy of a manifold (in 2D and 3D)



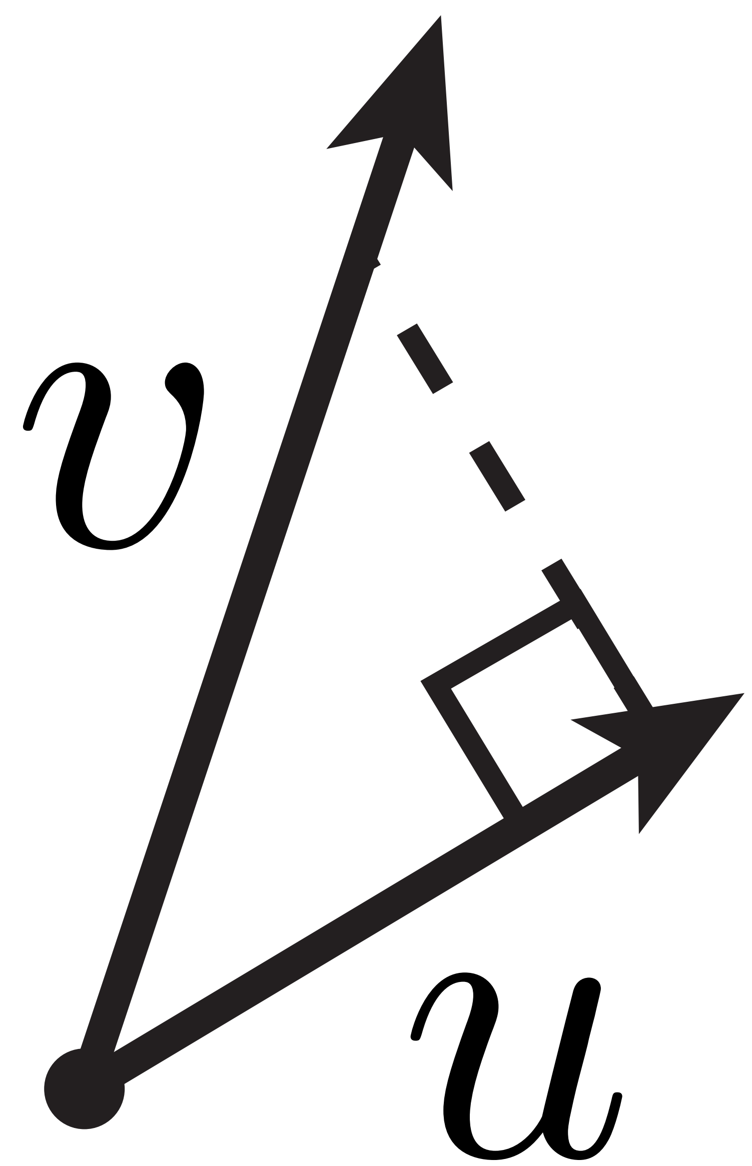


What can we measure about vectors?



$$u \cdot v = |u||v| \cos\theta$$

What can we measure about vectors?

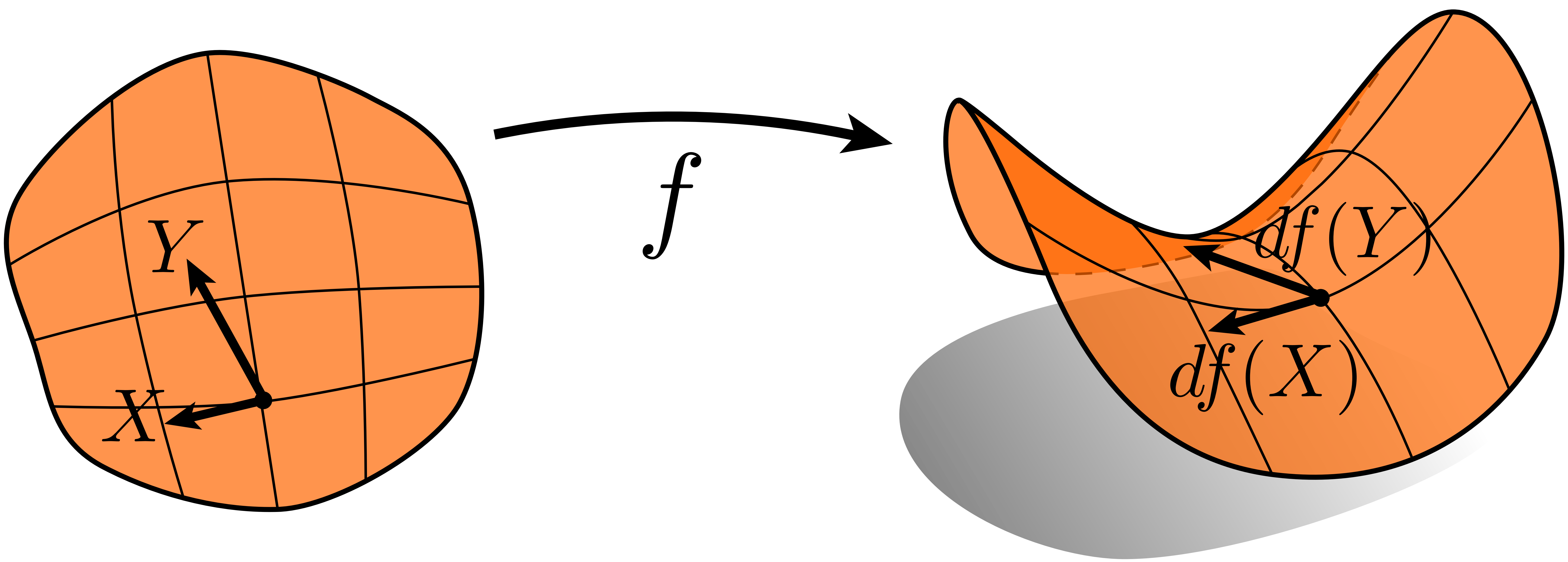


$$v \cdot (u/|u|)$$

Inner product of tangent vectors is

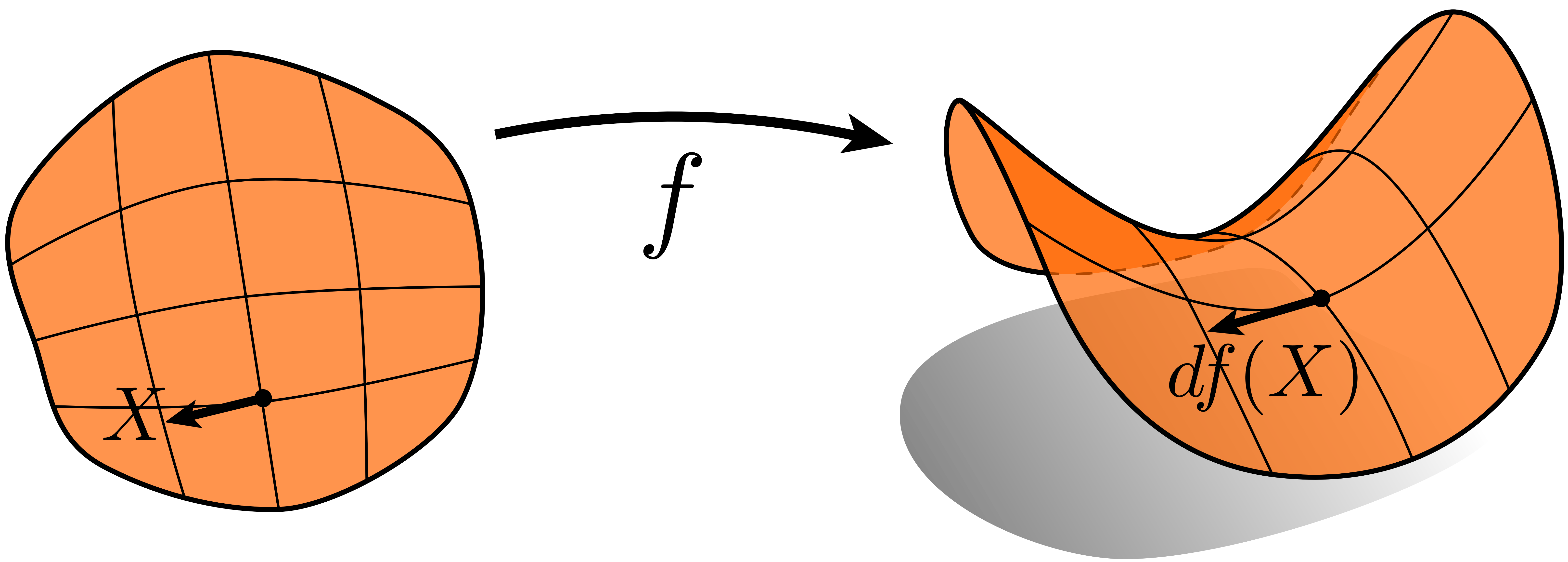
$$g(X,Y) = df(X) \cdot df(Y)$$

metric
("first fundamental form")



Q: What's the length of a tangent vector?

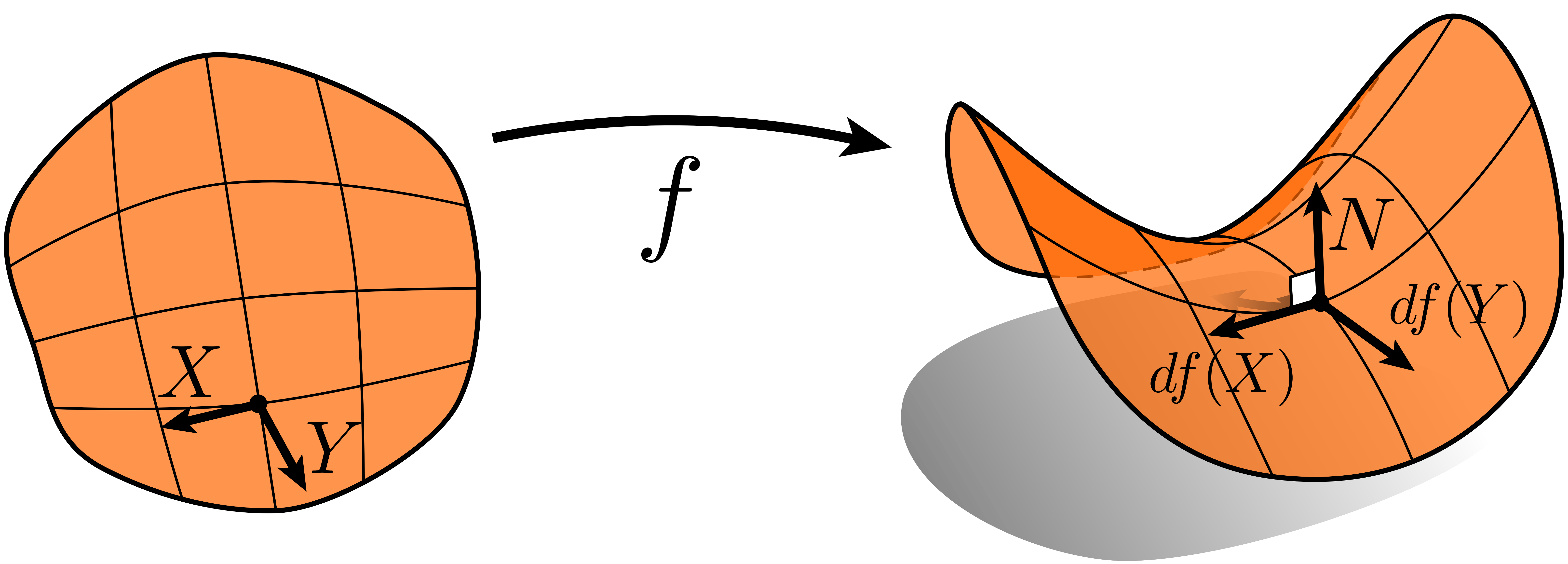
$$|X| = \sqrt{df(X) \cdot df(X)}$$
$$= \sqrt{g(X, X)}$$



Normal is vector orthogonal to all tangents

$$N$$

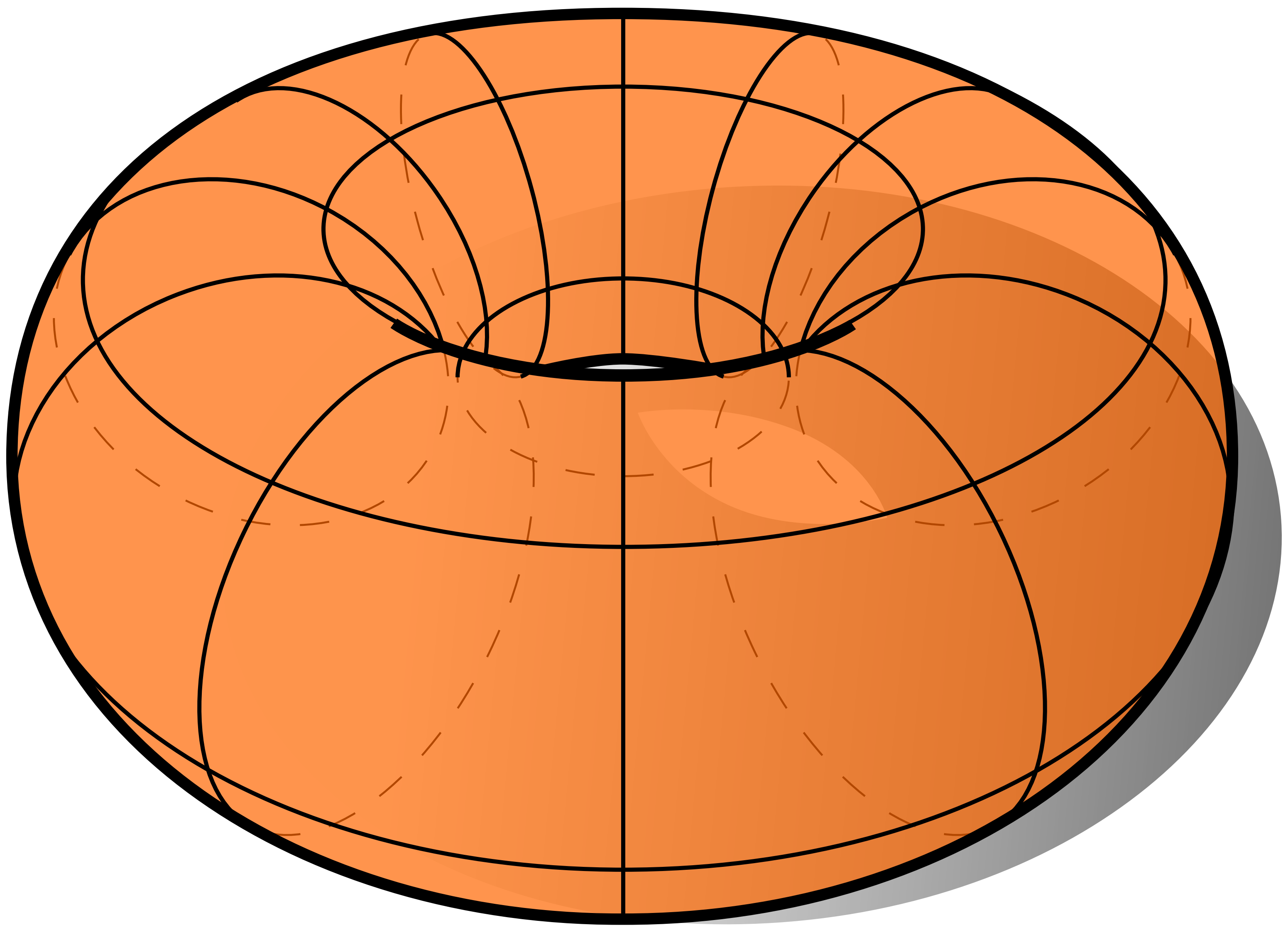
$$N \cdot df(X) = 0 \ \ \forall X$$



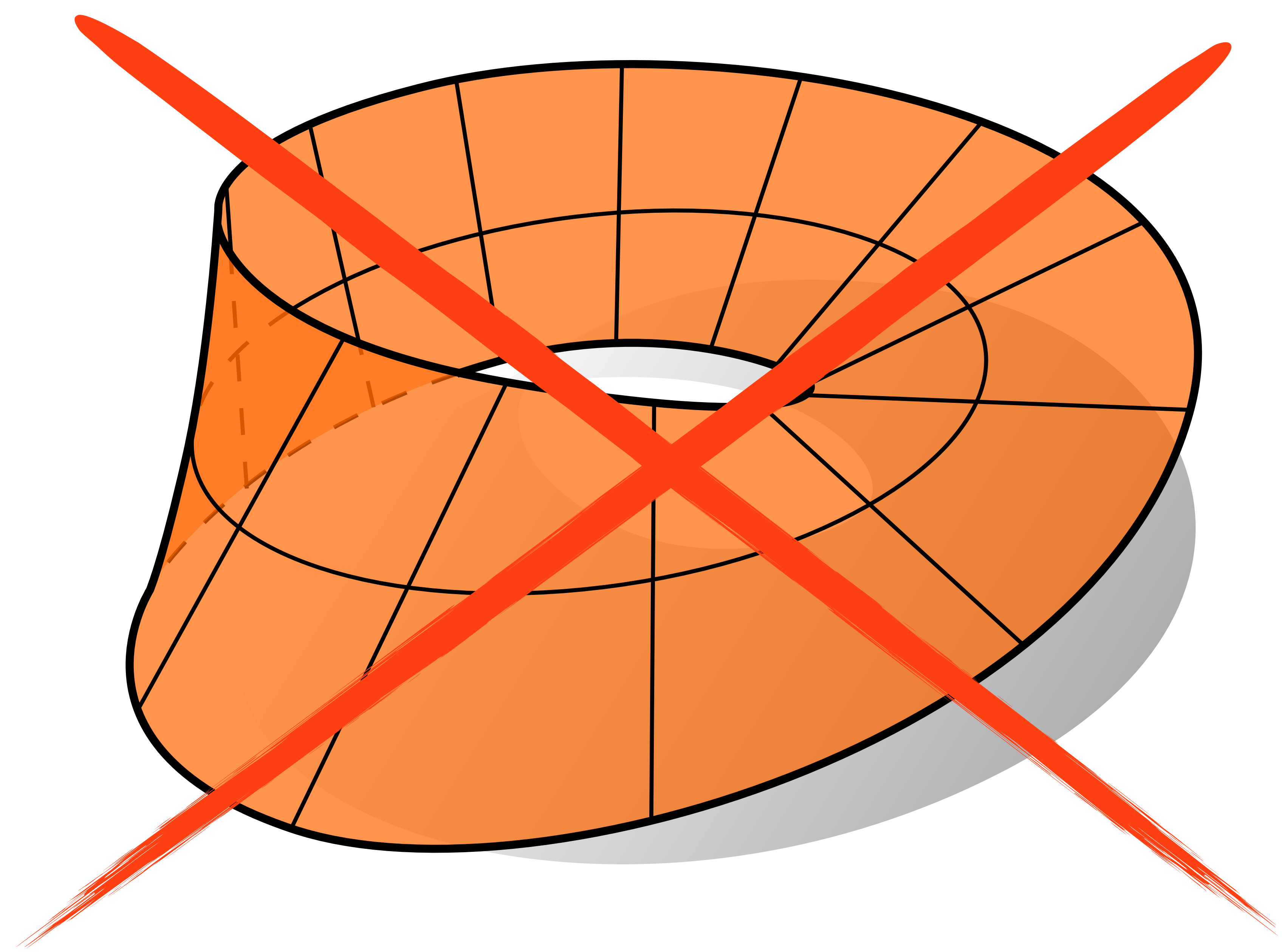
Which direction does the normal point?

$$|N| = 1$$

$$N \cdot df(X) = 0 \quad \forall X$$

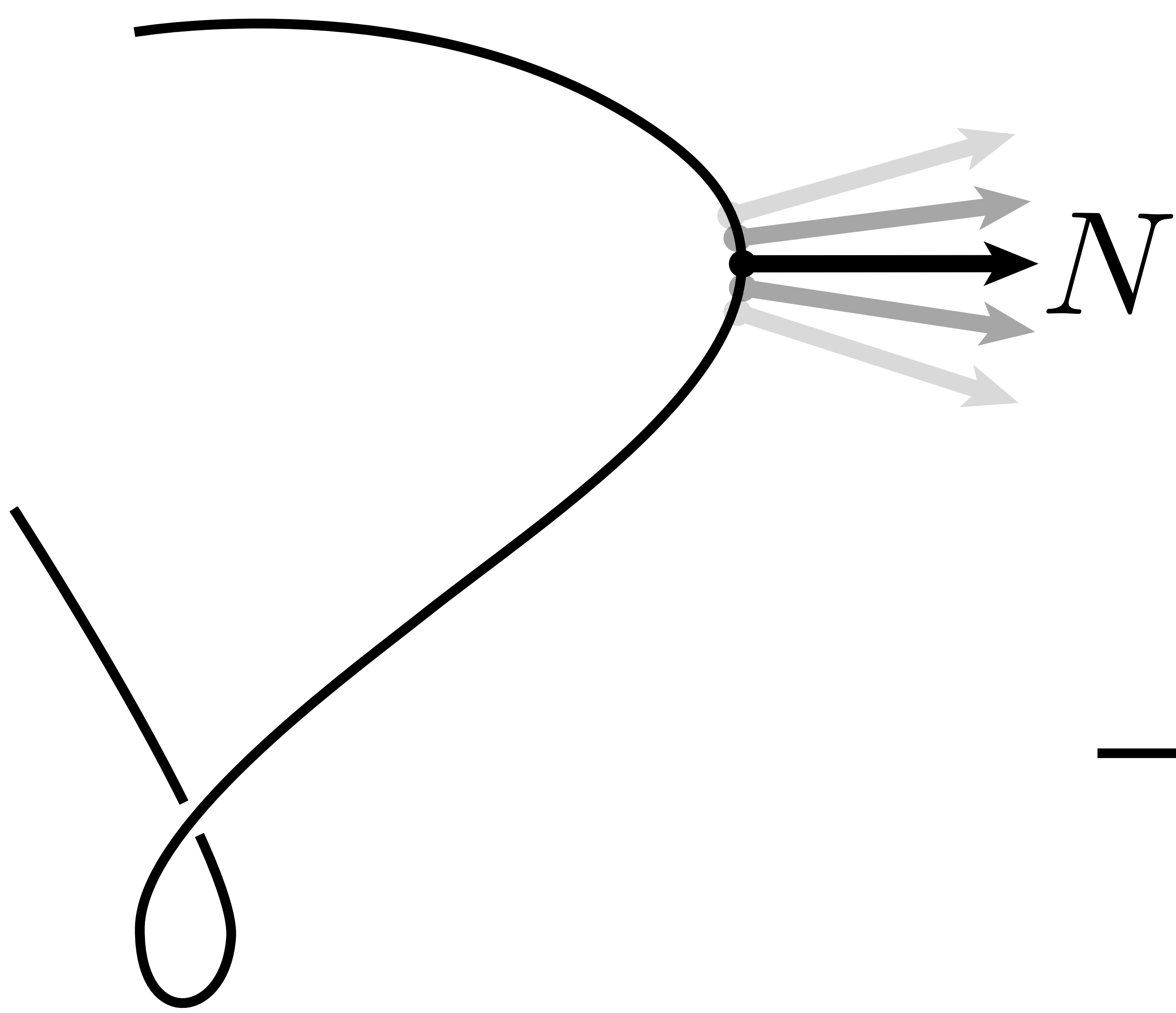


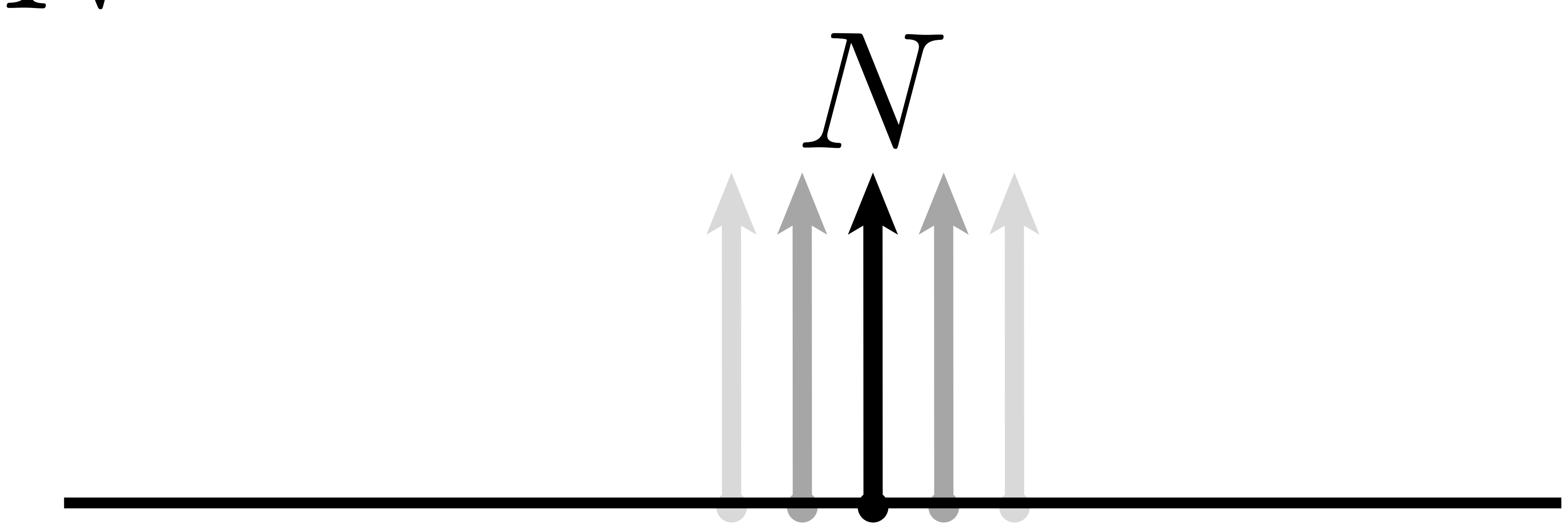
orientable



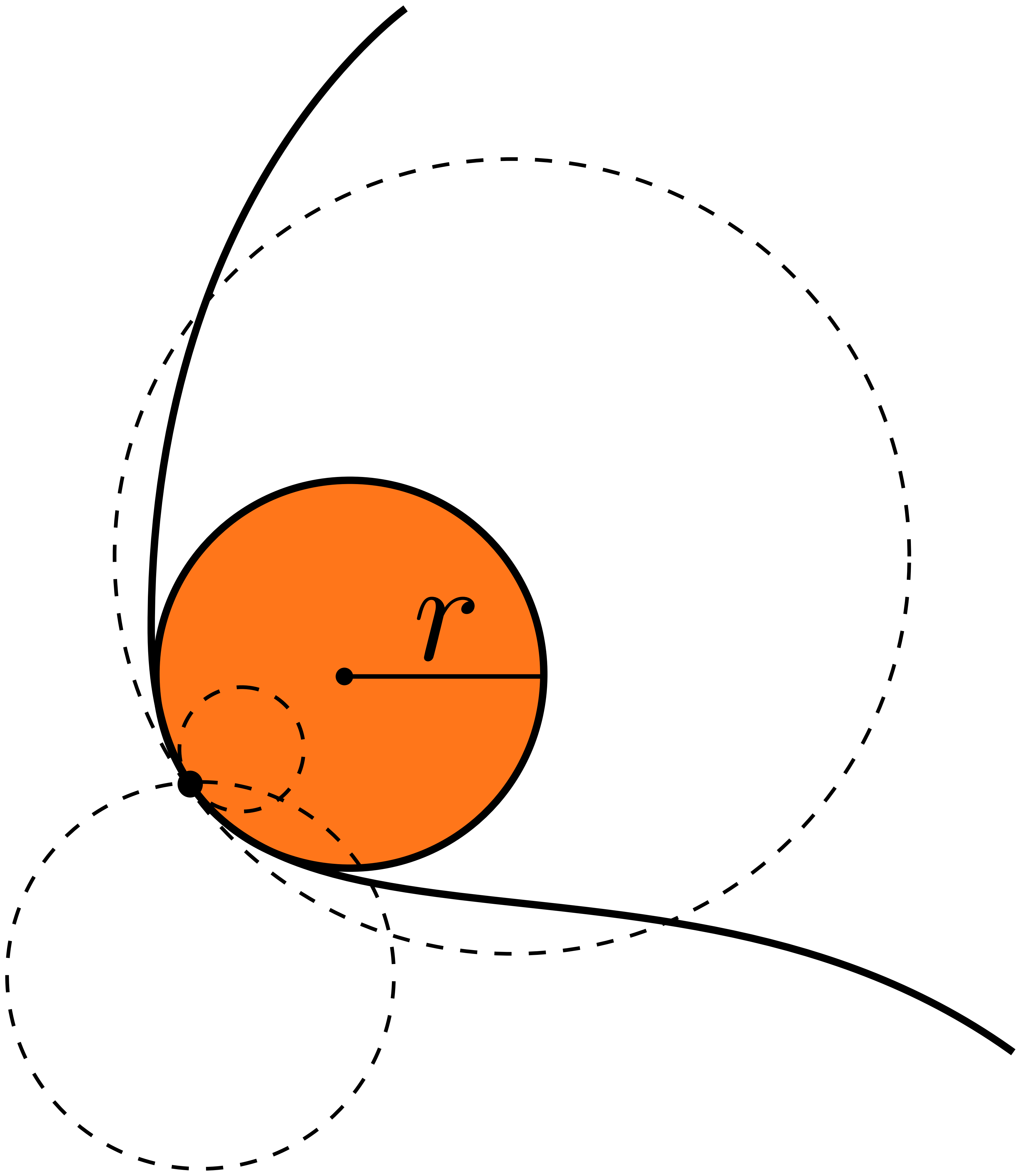
nonorientable

Curvature is change in normal





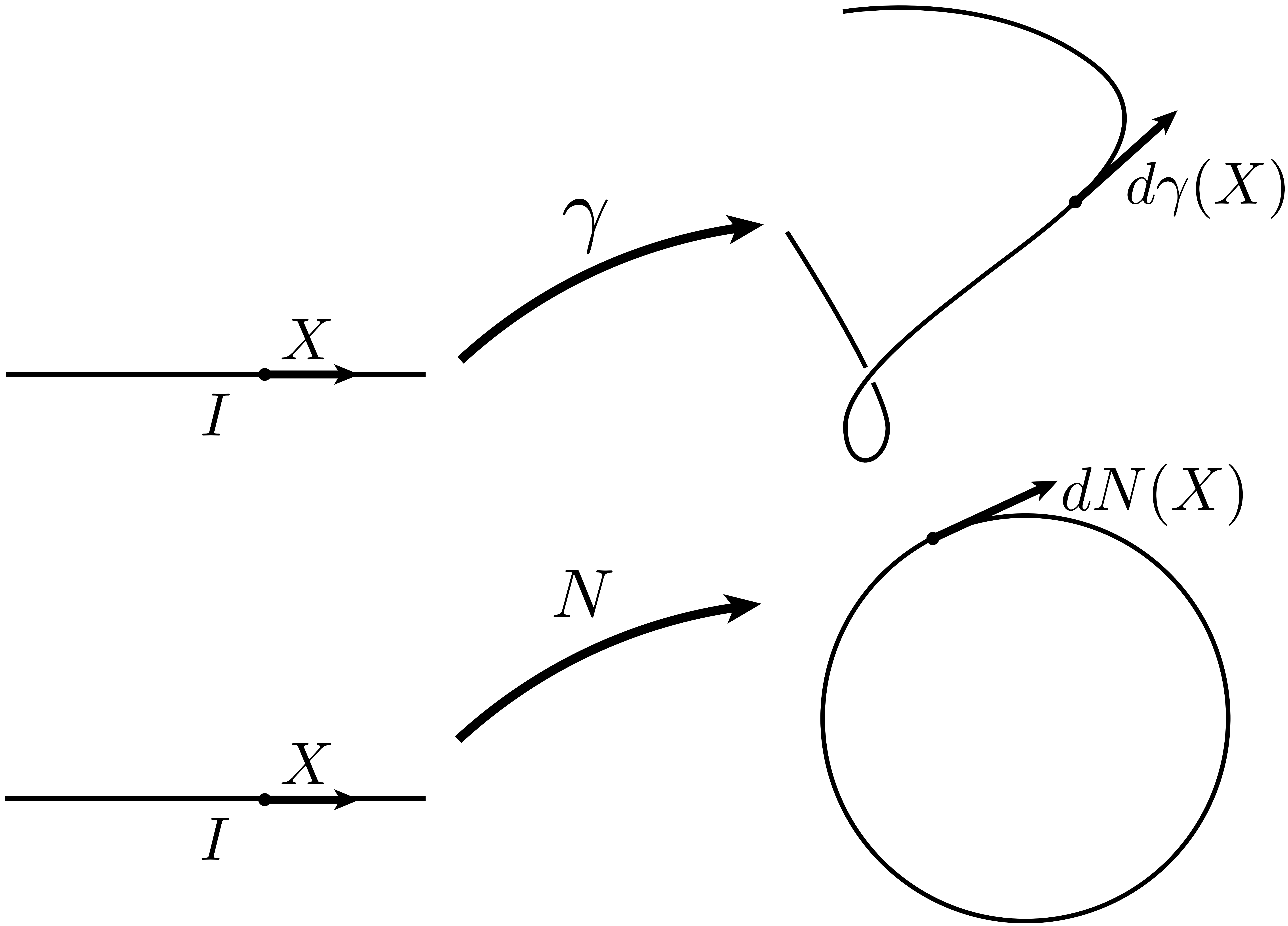
Standard definition: radius of curvature



$$\kappa = \frac{1}{r}$$

curvature

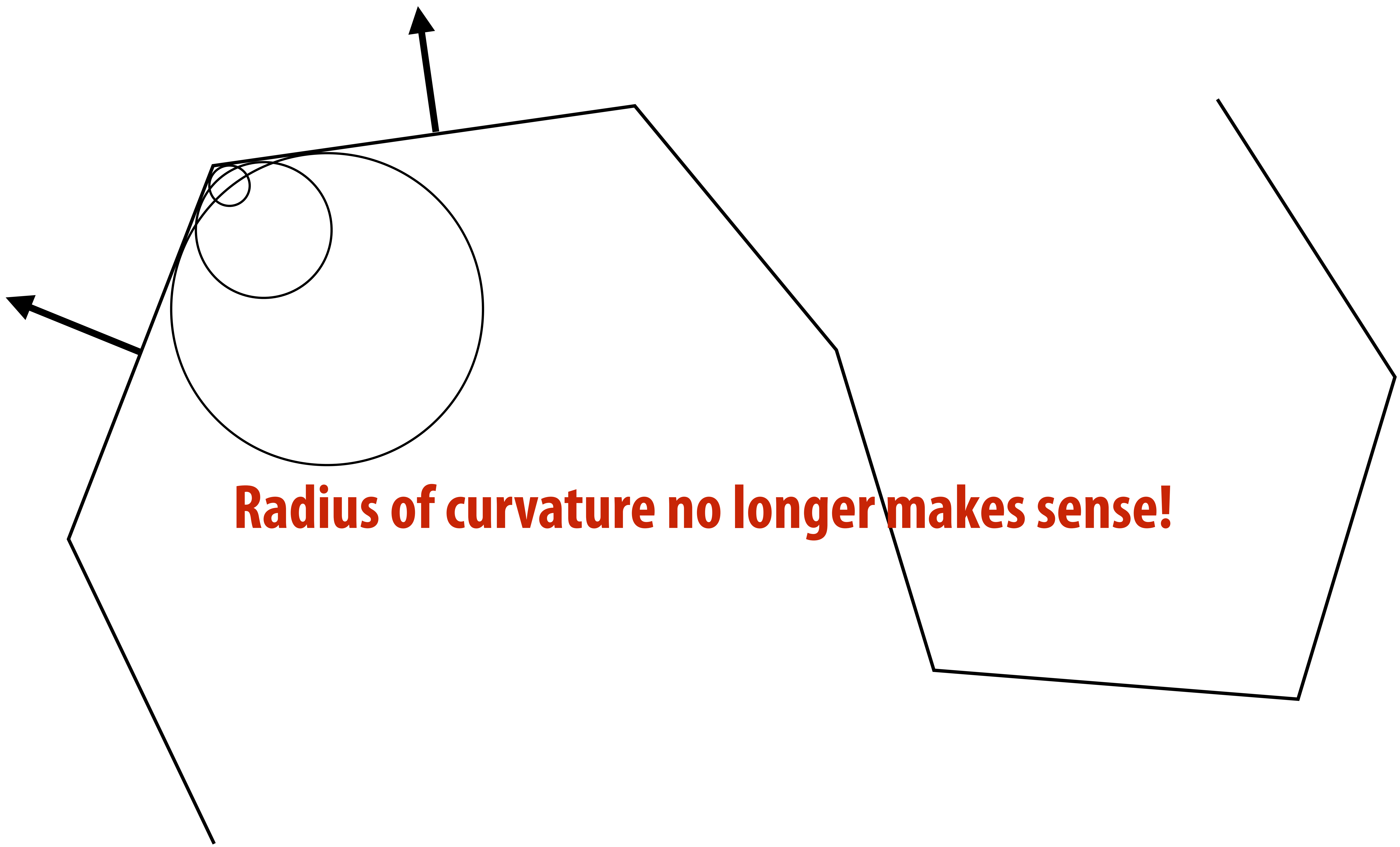
Alternative: normals as map to unit circle



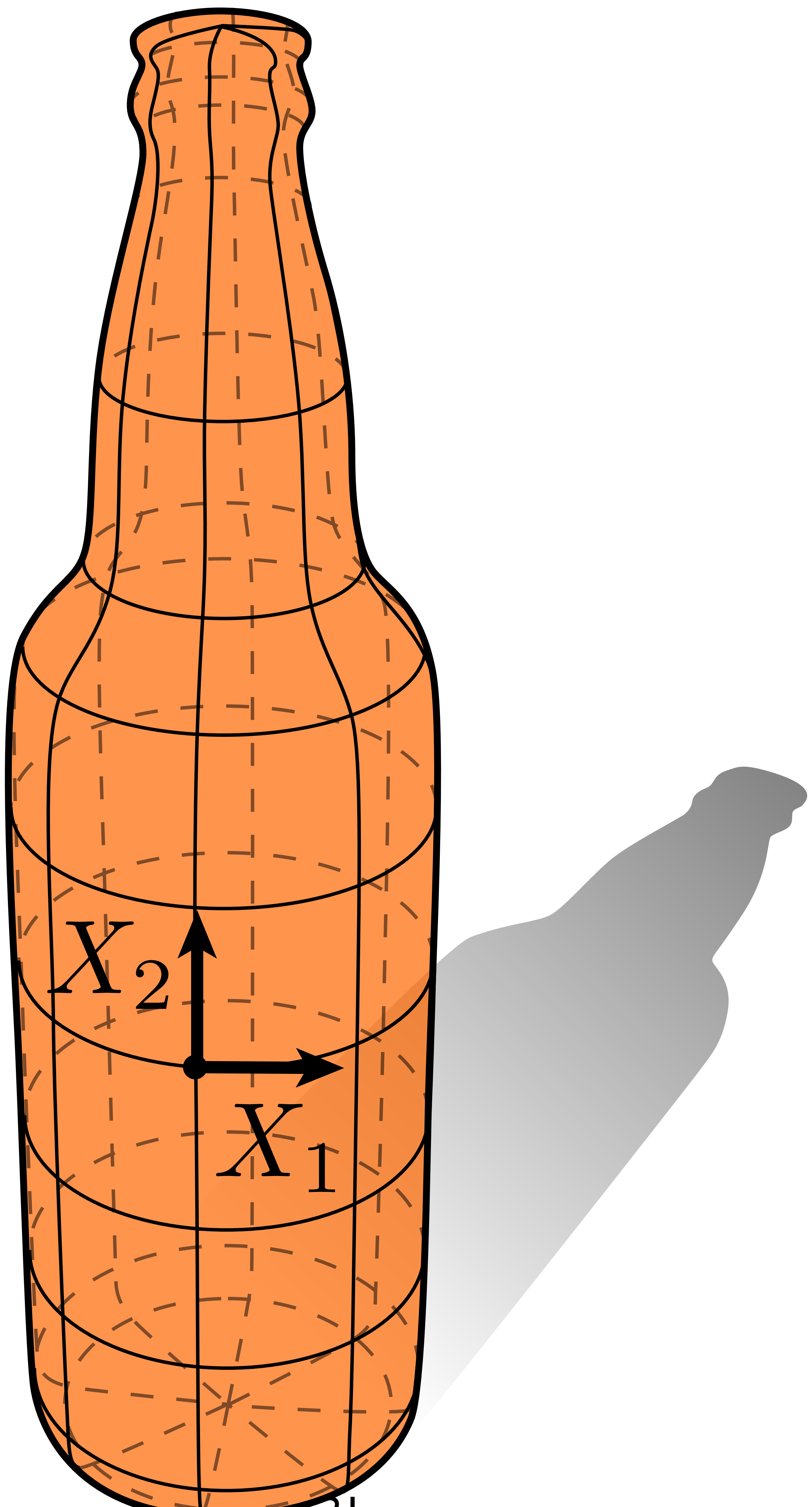
Key idea: size of swept-out piece gives total curvature.

Discrete curvature as change in normal

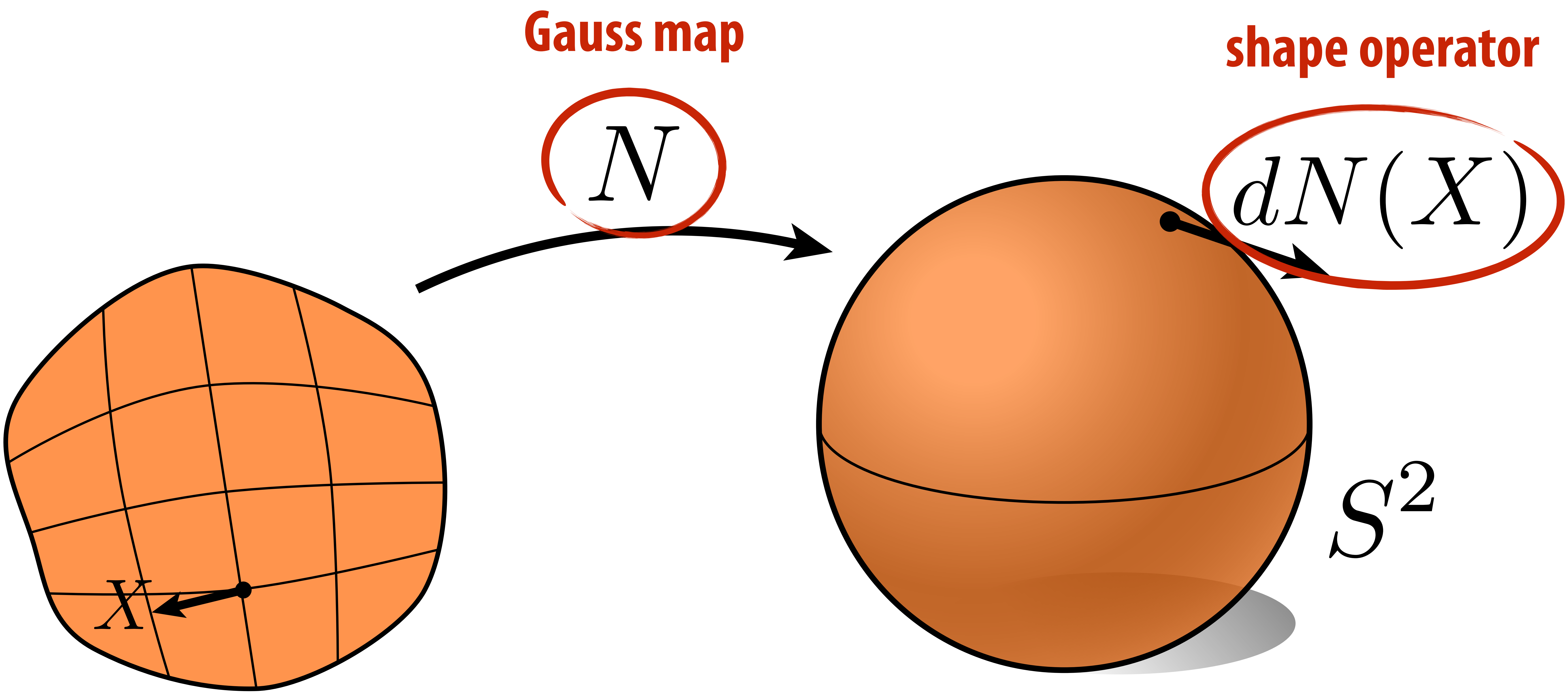
...can still talk about change in normal.



What about surfaces?



Normal is now map to the sphere

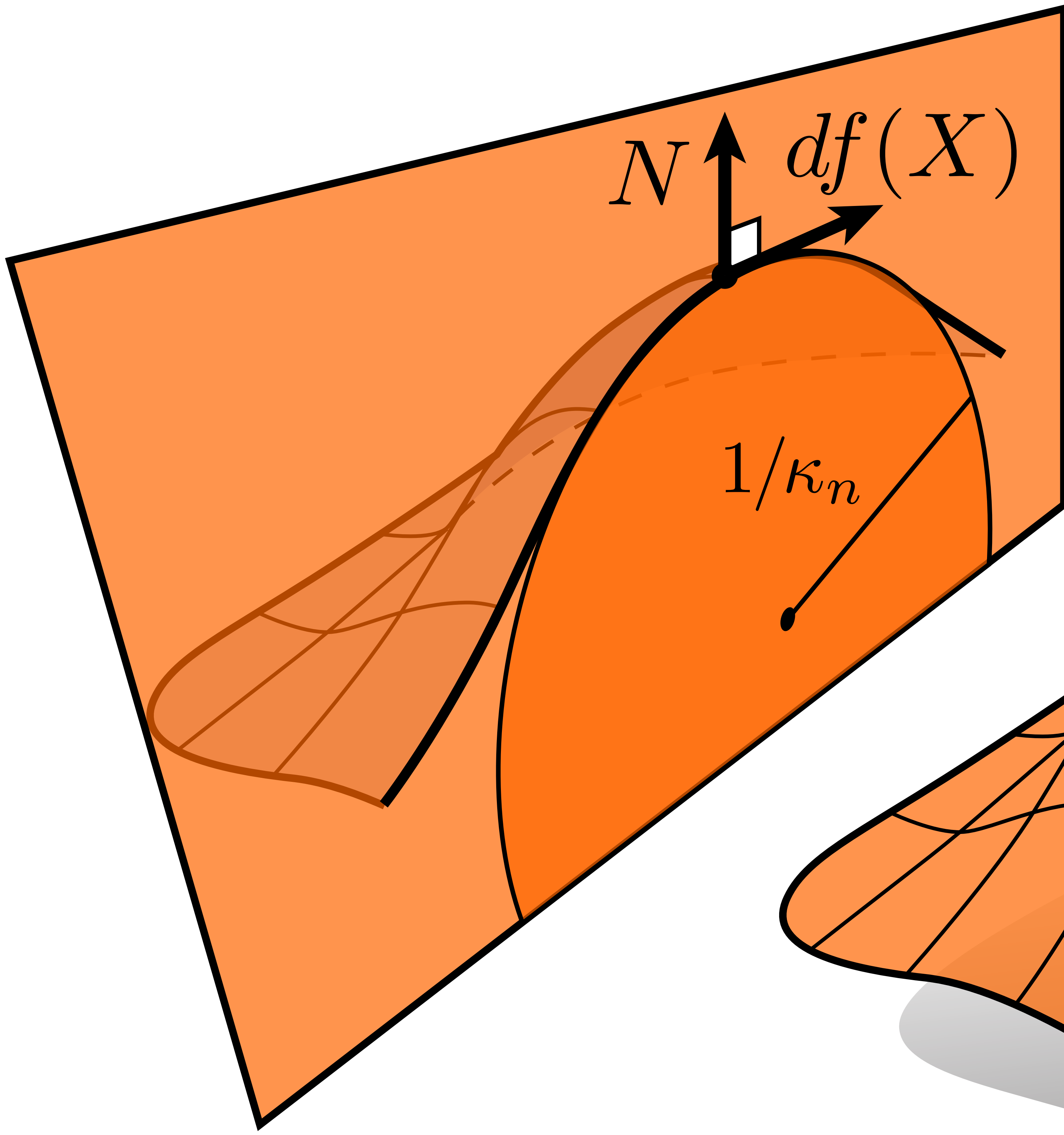


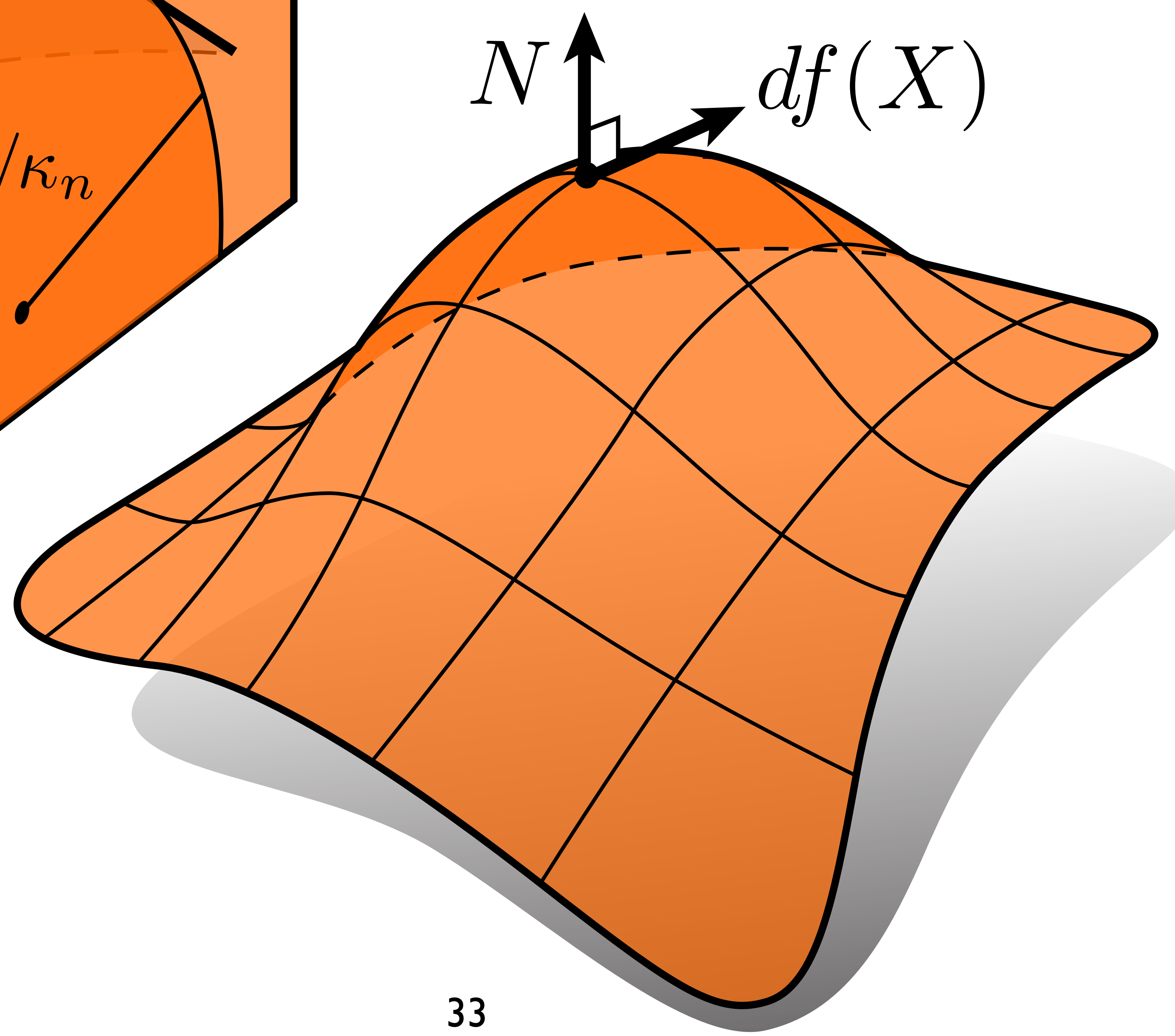
Normal curvature

$$\kappa_n(X) = -df(X) \cdot dN(X)$$

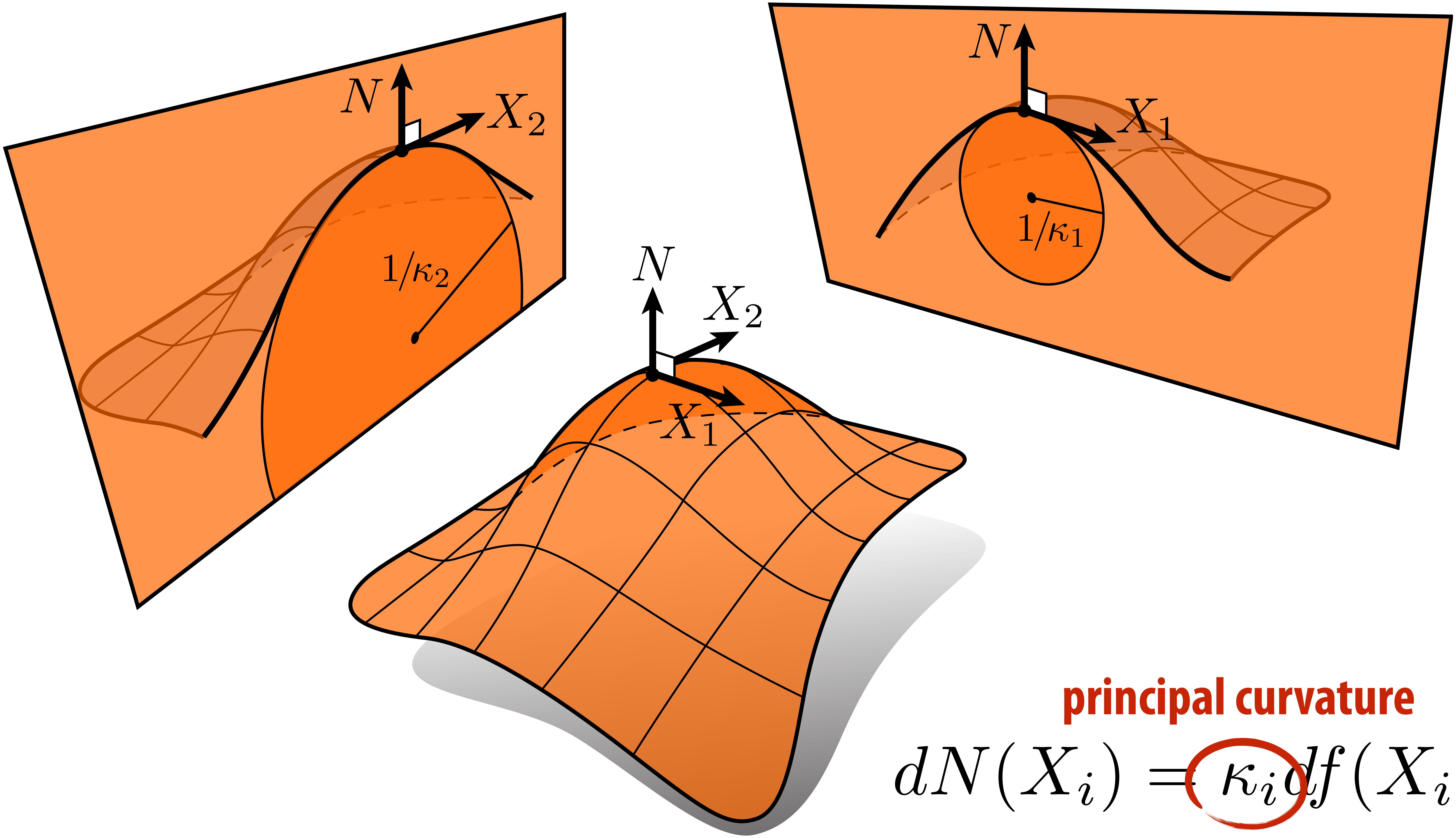
$$-df(X) \cdot dN(Y)$$

("second fundamental form")





Principal Curvatures

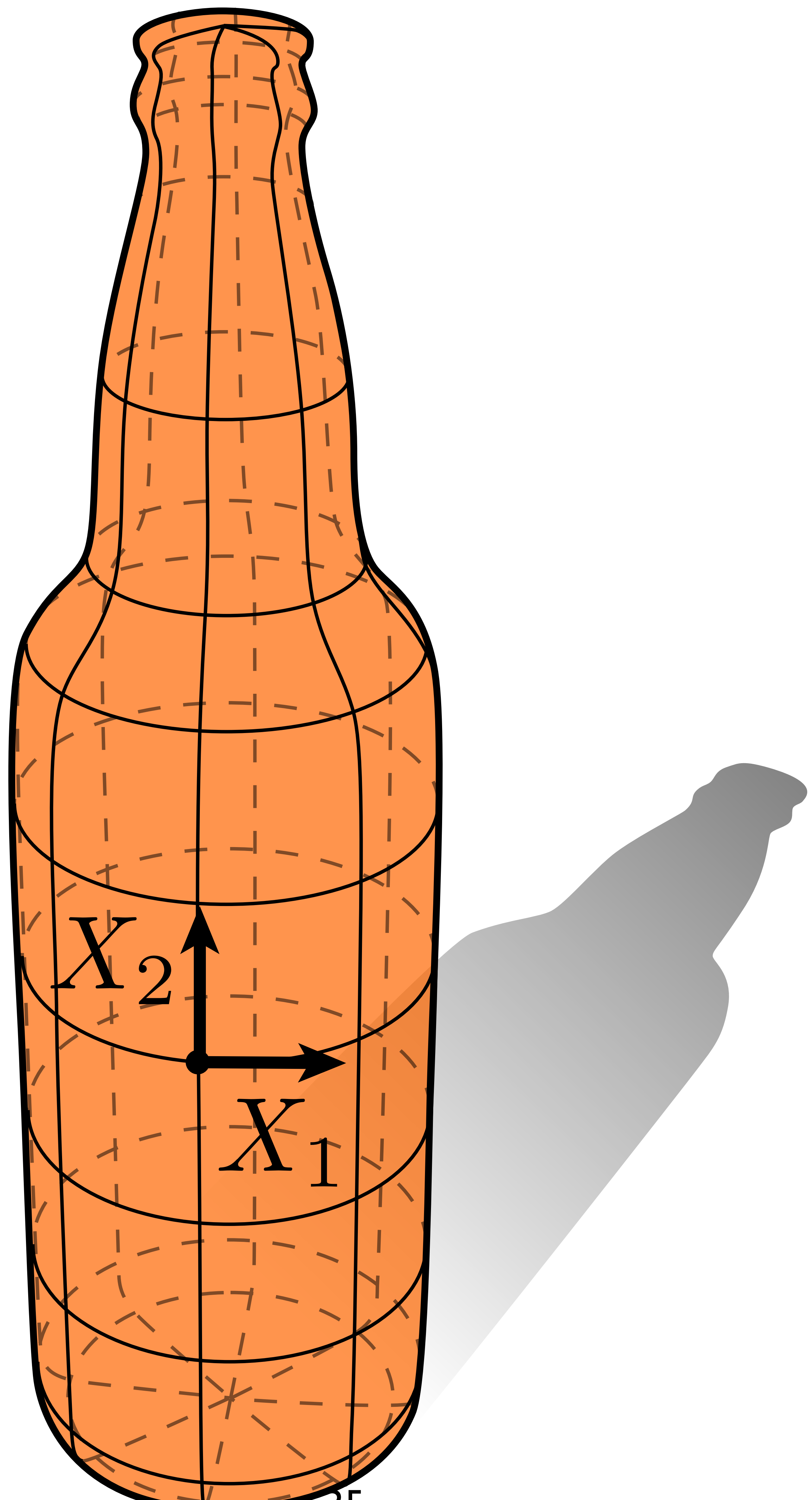


principal curvature

$$dN(X_i) = \kappa_i df(X_i)$$

Fact: principal curvature directions are orthogonal.

Q: What are the principal curvatures?

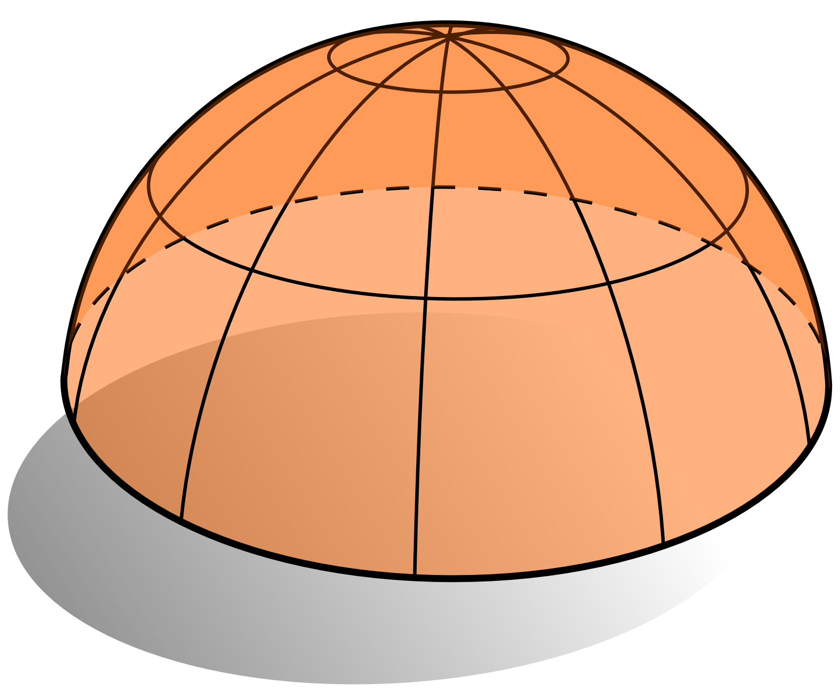


Mean & Gaussian Curvature

mean $$H = \frac{1}{2}(\kappa_1 + \kappa_2)$$

Gaussian $$K = \kappa_1 \kappa_2$$

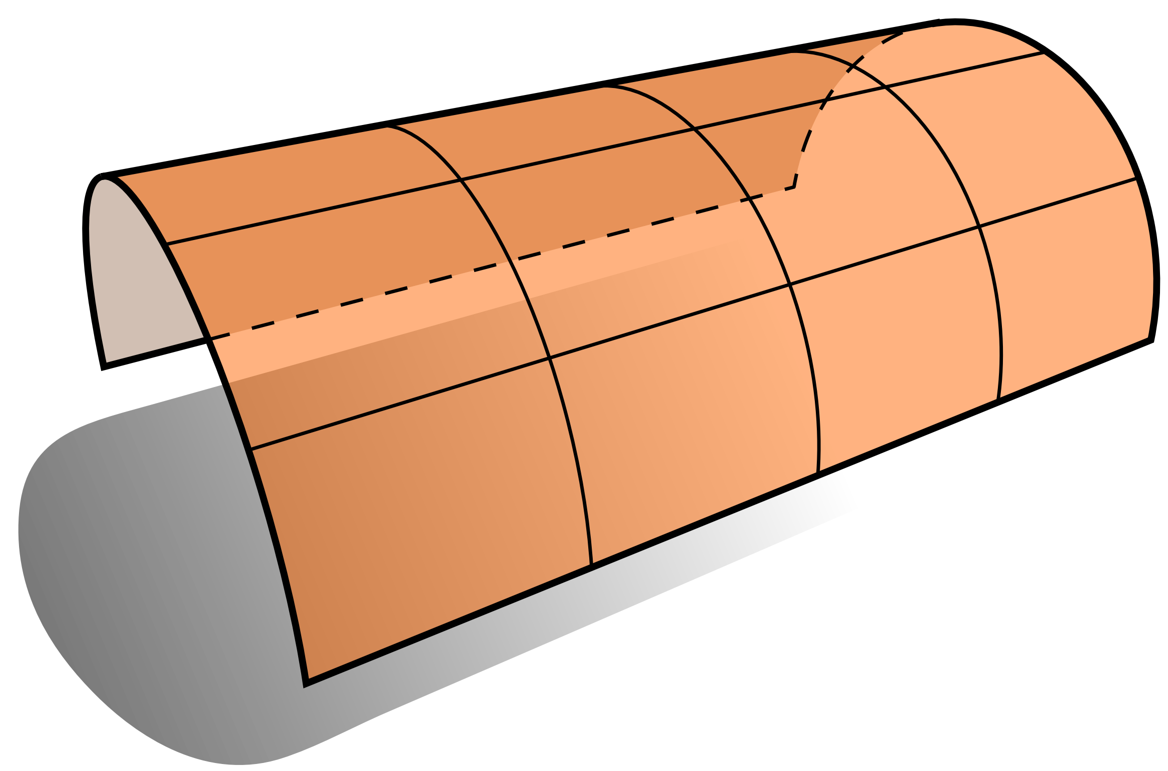
$\kappa_1 > 0, \kappa_2 > 0$



$$H > 0$$

$$K > 0$$

$\kappa_1 > 0, \kappa_2 = 0$

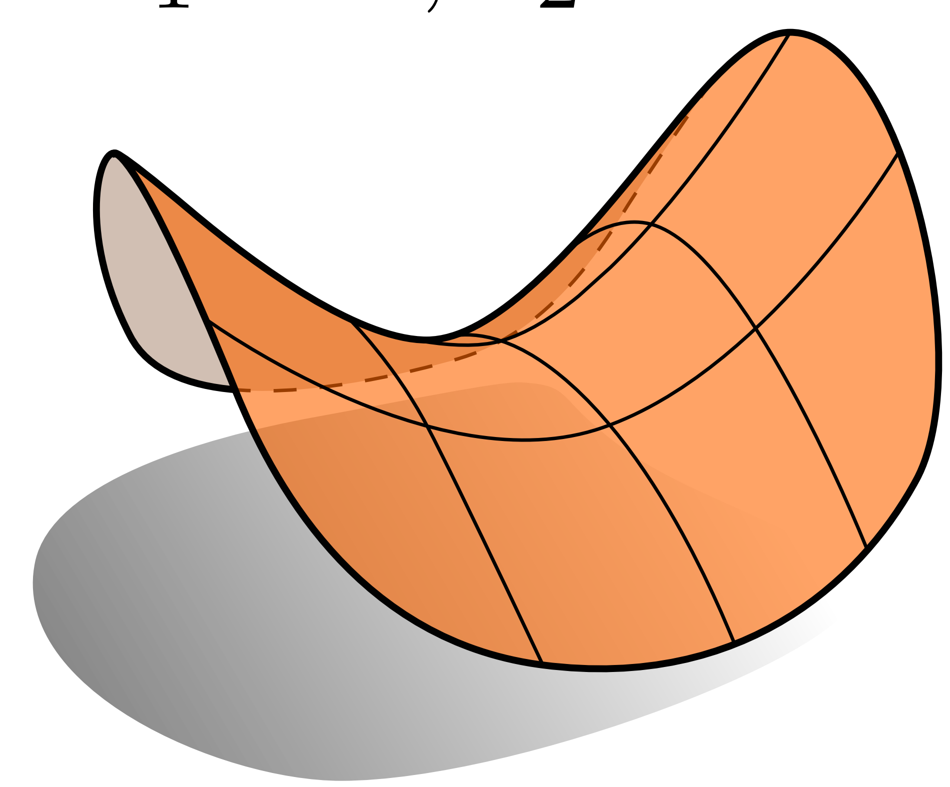


$$H \neq 0$$

$$K = 0$$

developable

$\kappa_1 = 1, \kappa_2 = -1$



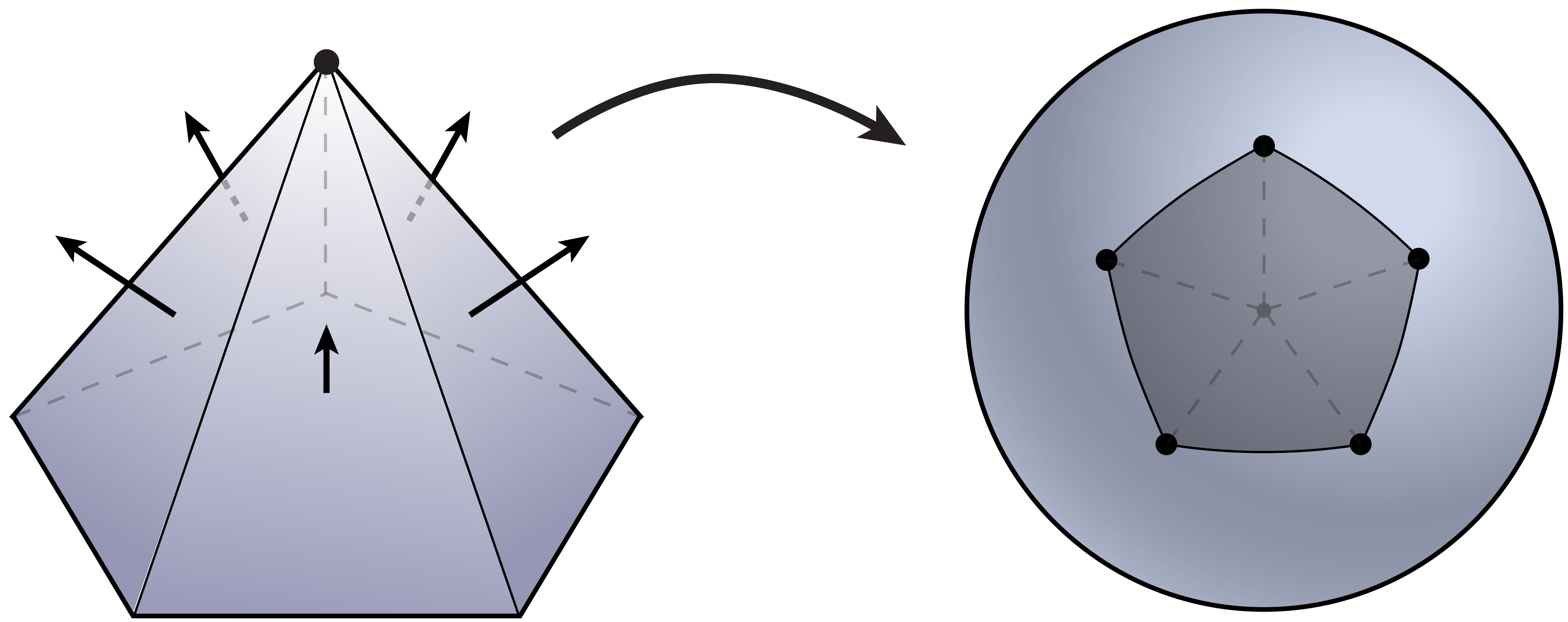
minimal

$$H = 0$$

$$K < 0$$

Discrete Gaussian Curvature?

- **Once again, use area on Gauss sphere:**



A lot can be done with this representation!
See http://keenan.is/dgpdec for more.

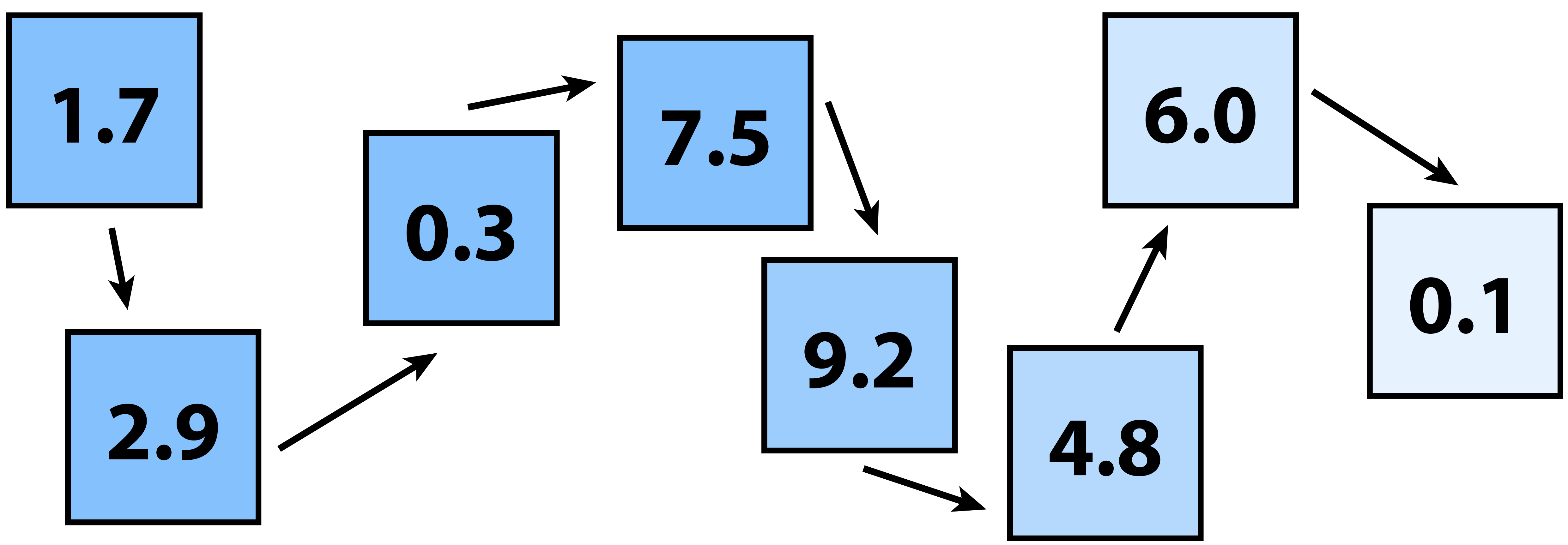
How do we actually encode all this data?

Warm up: arrays vs. linked lists

- Want to store a list of numbers
- One idea: use an array (constant time lookup, coherent access)

1.7	2.9	0.3	7.5	9.2	4.8	6.0	0.1

- 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...

Polygon soup, revisited

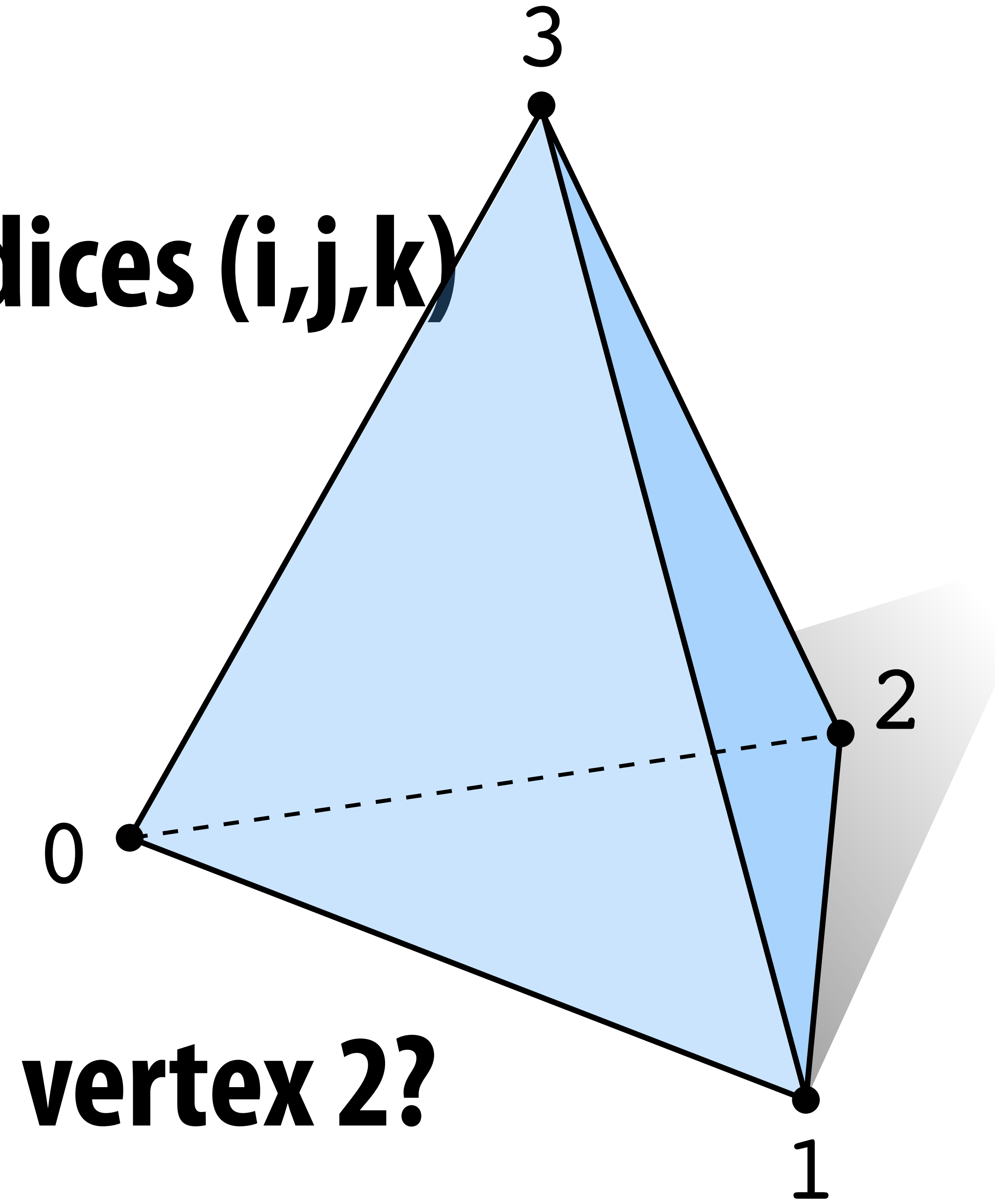
- Store triples of coordinates (x,y,z) and indices (i,j,k)
- E.g., tetrahedron:

VERTICES

	x	y	z
0:	–1	–1	–1
1:	1	–1	1
2:	1	1	–1
3:	–1	1	1

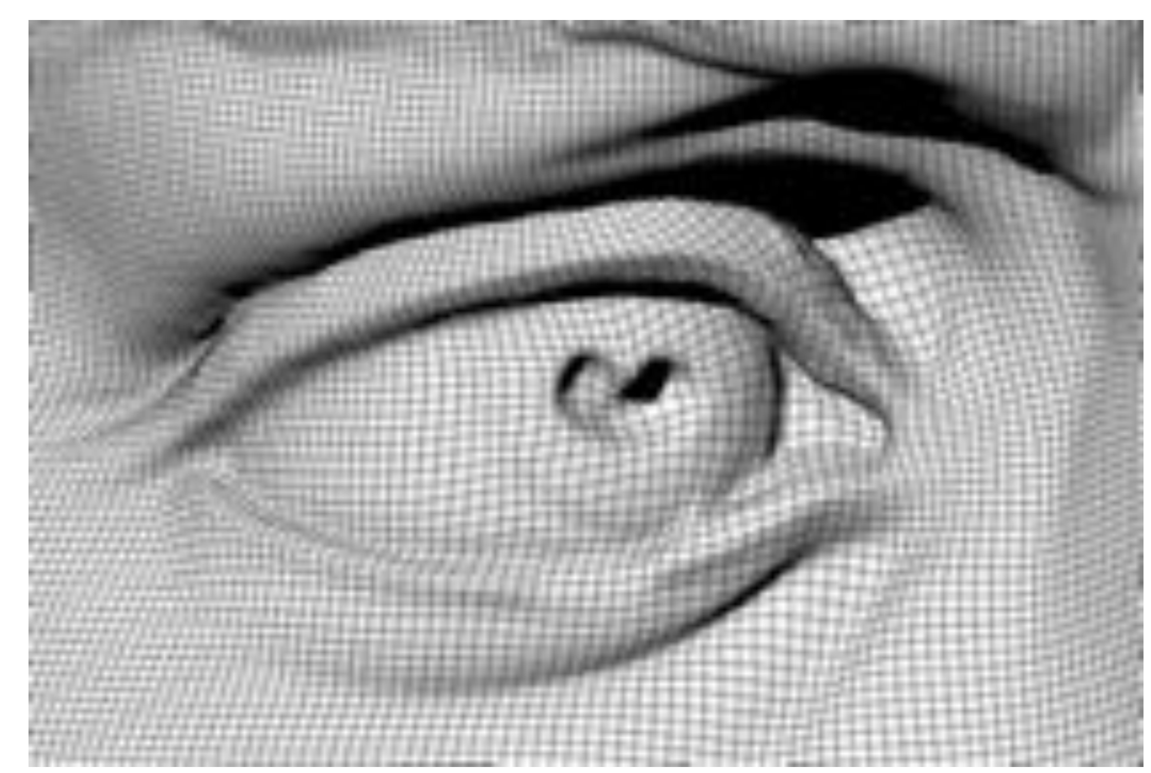
TRIANGLES

i	j	k
0	2	1
0	3	2
3	0	1
3	1	2



- Q: How do we find all the triangles touching vertex 2?
- Ok, now consider a more complicated mesh:

~1 billion polygons



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

Alternative: Incidence Matrices

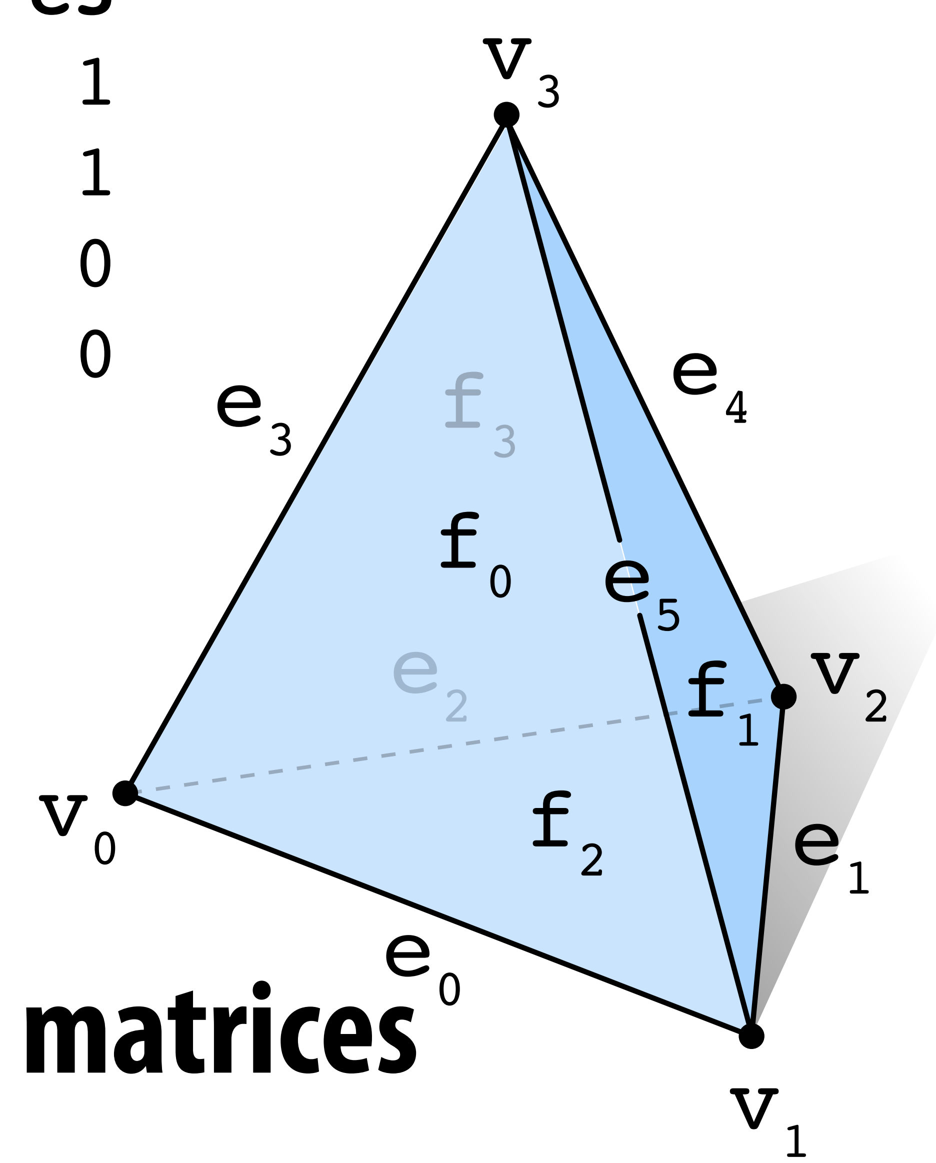
- If we want to answer neighborhood queries, why not simply store a list of neighbors?
- Can encode all neighbor information via incidence matrices
- E.g., tetrahedron:

VERTEX ↔ EDGE

	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

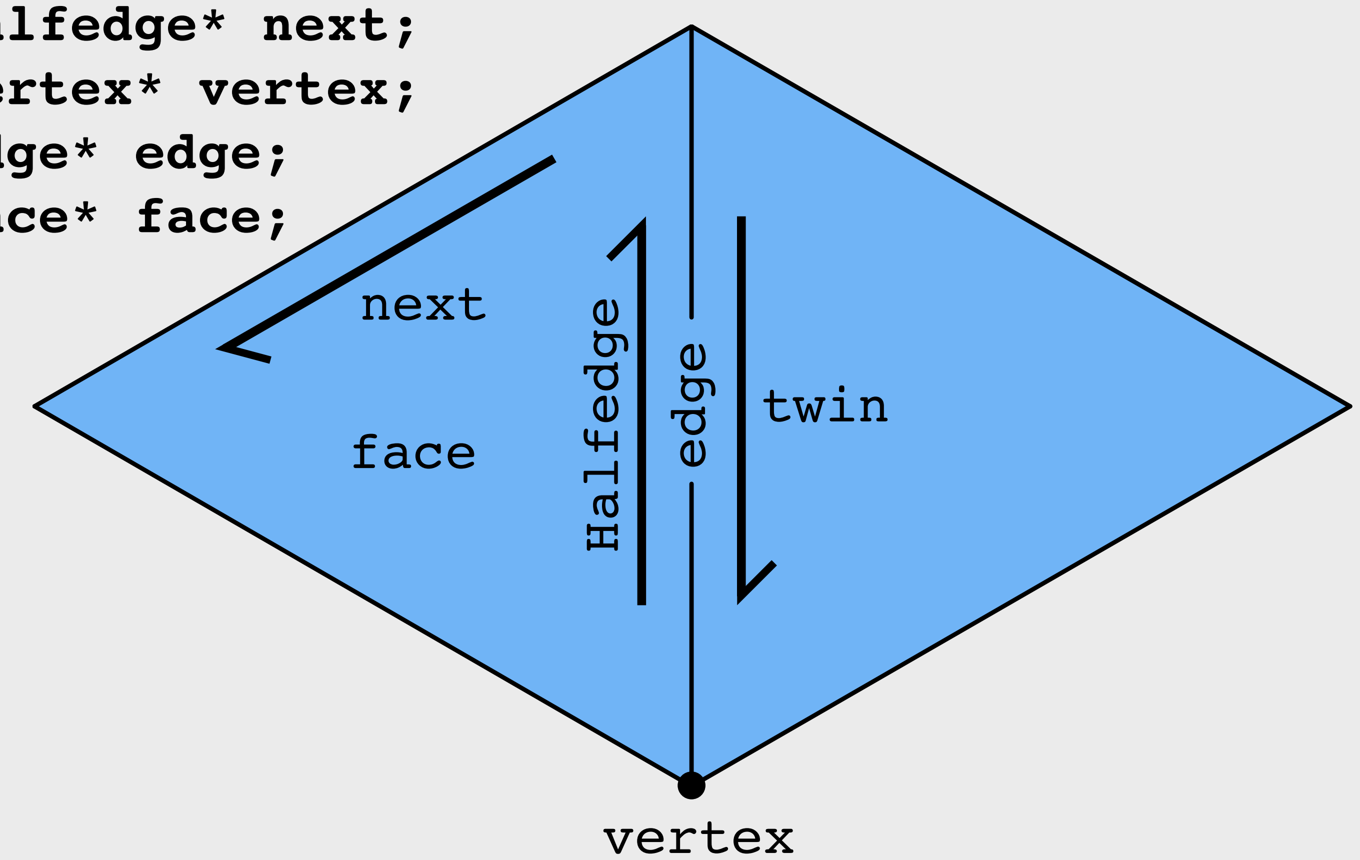


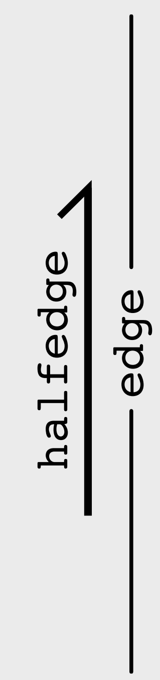
- 1 means "touches"; 0 means "does not touch"
- For large meshes, most entries will be zero!
- Can dramatically reduce storage cost using sparse matrices
- Still large storage cost, but finding neighbors is now O(1)
- (Bonus feature: mesh does not have to be manifold)

Alternative: Halfedge Data Structure

- 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:

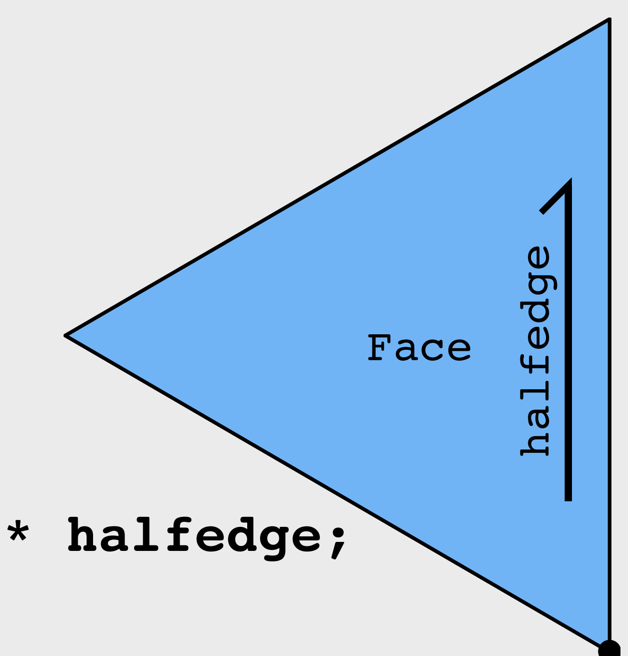
```
struct Halfedge
{
   Halfedge* twin;
   Halfedge* next;
   Vertex* vertex;
   Edge* edge;
   Face* face;
};
```

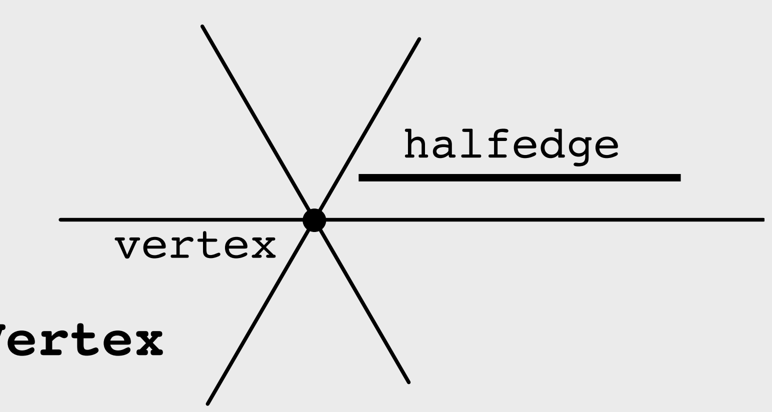





```
struct Edge
{
   Halfedge* halfedge;
};
```

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





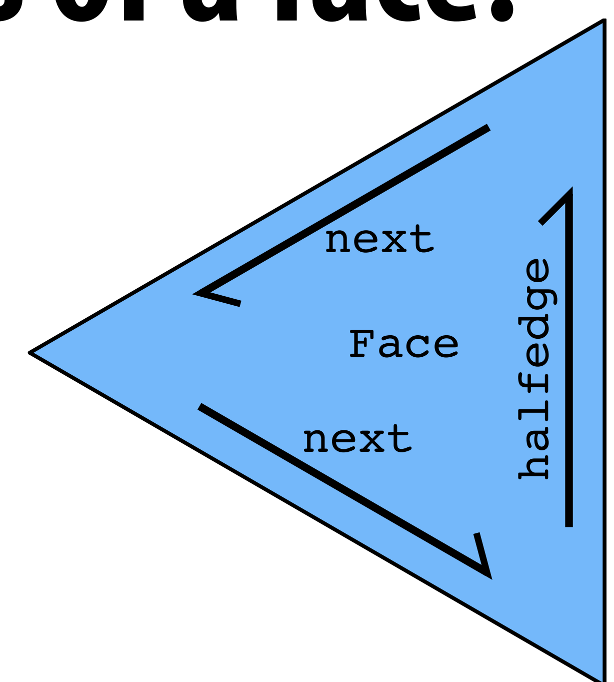

```
struct Vertex
{
   Halfedge* halfedge;
};
```

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

Halfedge makes mesh traversal easy

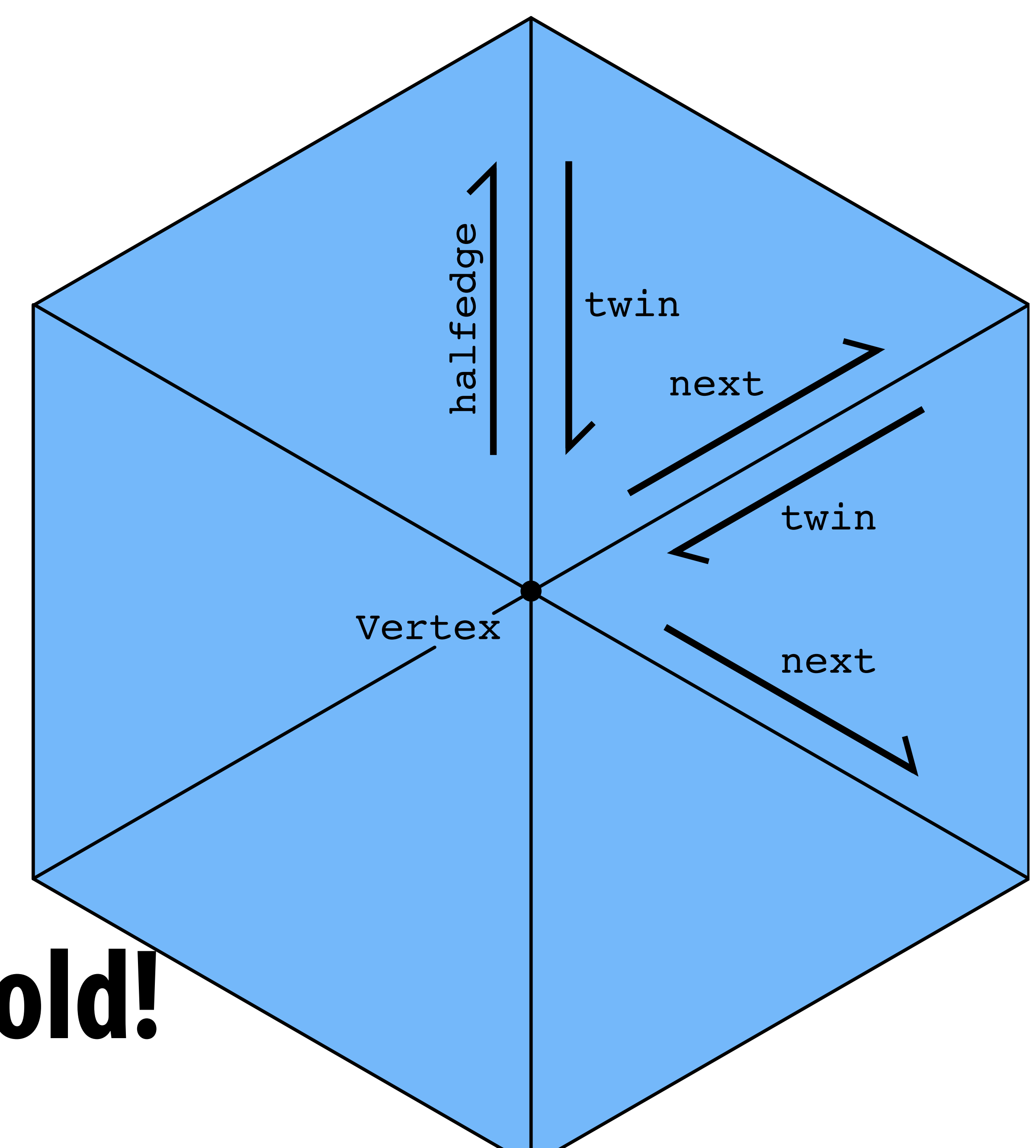
- Use "twin" and "next" pointers to move around mesh
- Use "vertex", "edge", and "face" pointers to grab element
- Example: visit all vertices of a face:

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



- Example: visit all neighbors of a vertex:

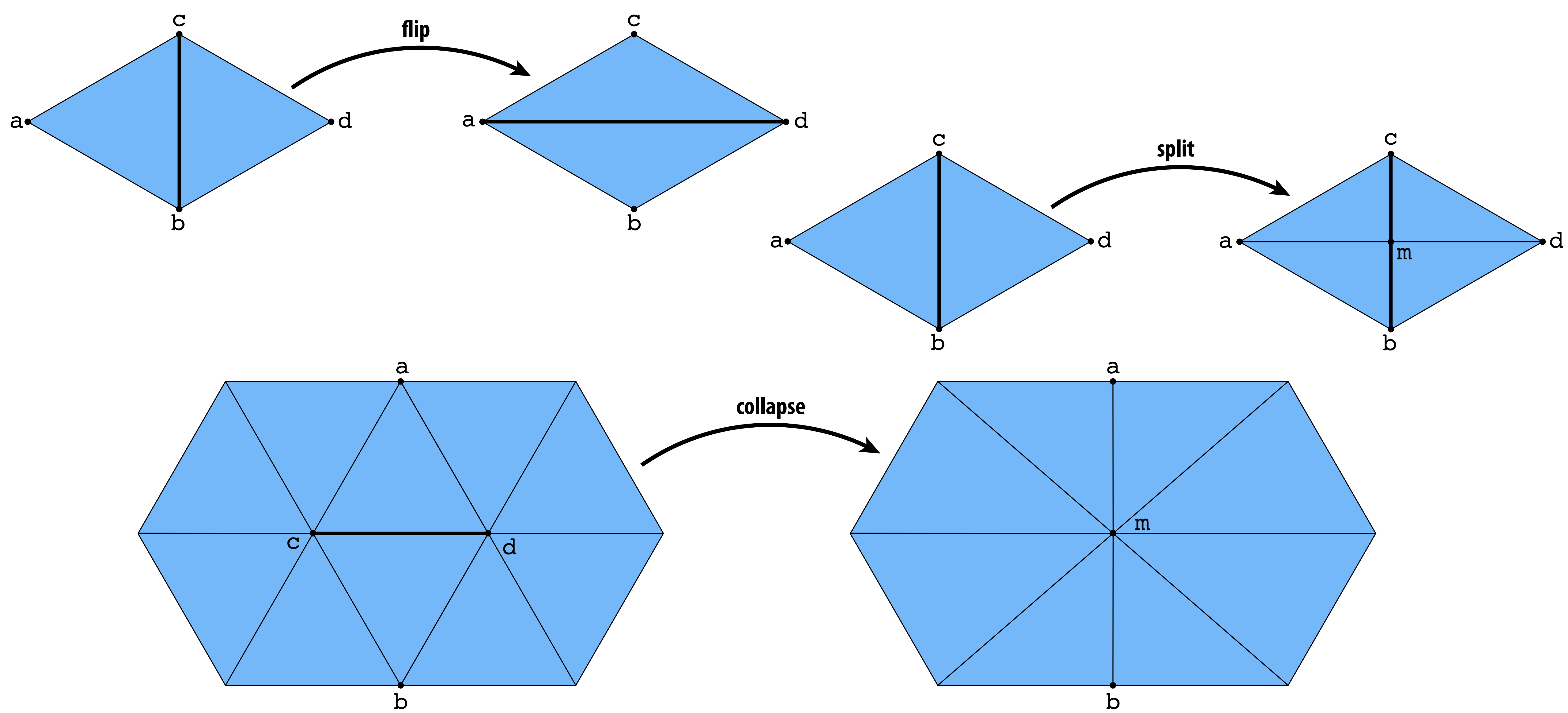
```
Halfedge* h = v->halfedge;
do {
   h = h->twin->next;
}
while( h != v->halfedge );
```



- Note: only makes sense if mesh is manifold!

Halfedge also easy to edit

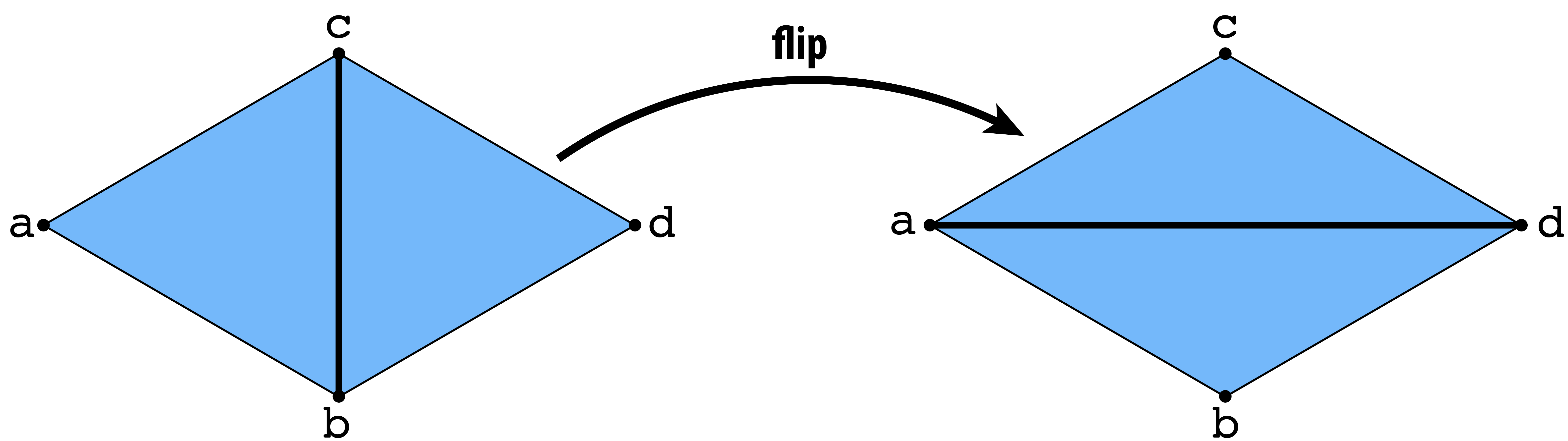
- Remember key feature of linked list: insert/delete elements
- Same story with halfedge mesh (“linked list on steroids”)
- Several atomic operations for triangle meshes:



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

Edge Flip

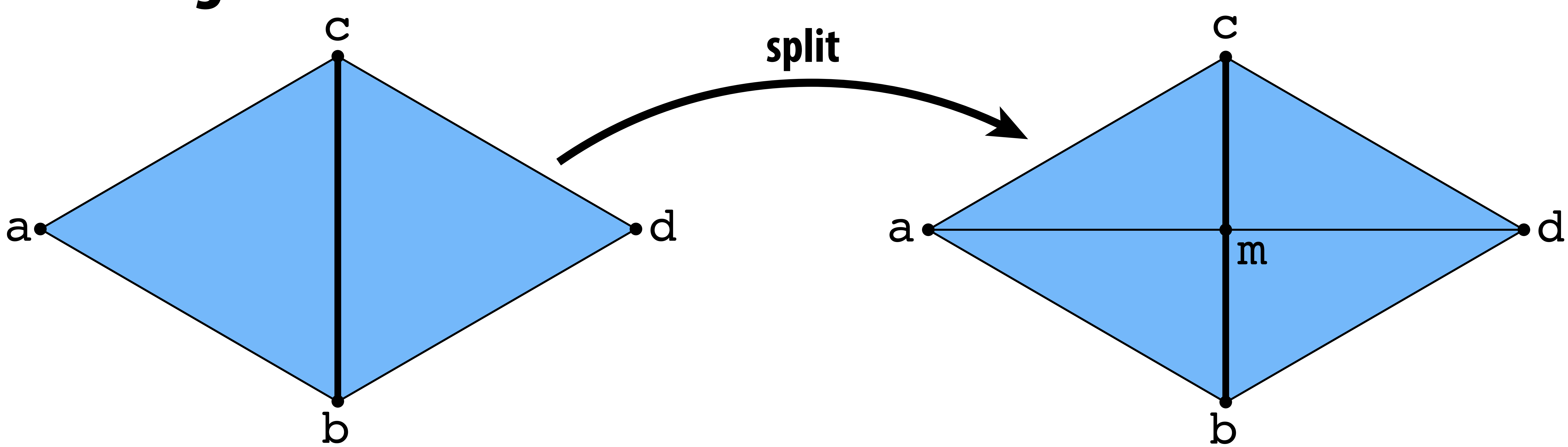
- **Triangles (a,b,c), (b,d,c) become (a,d,c), (a,b,d):**



- **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

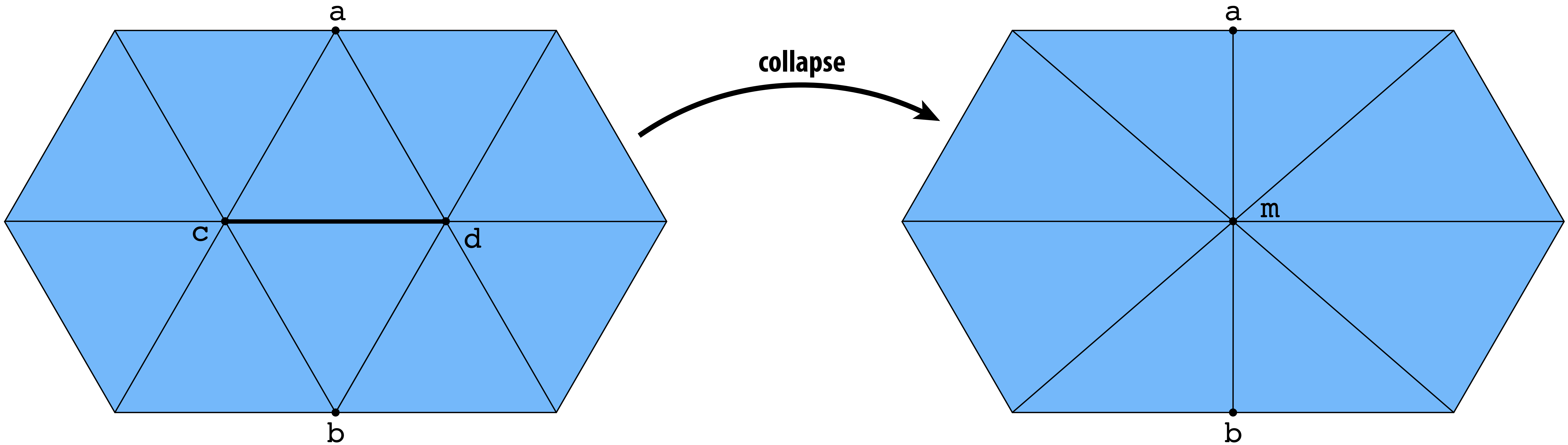
- Insert midpoint m of edge (c,b), connect to get four triangles:



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

Edge Collapse

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

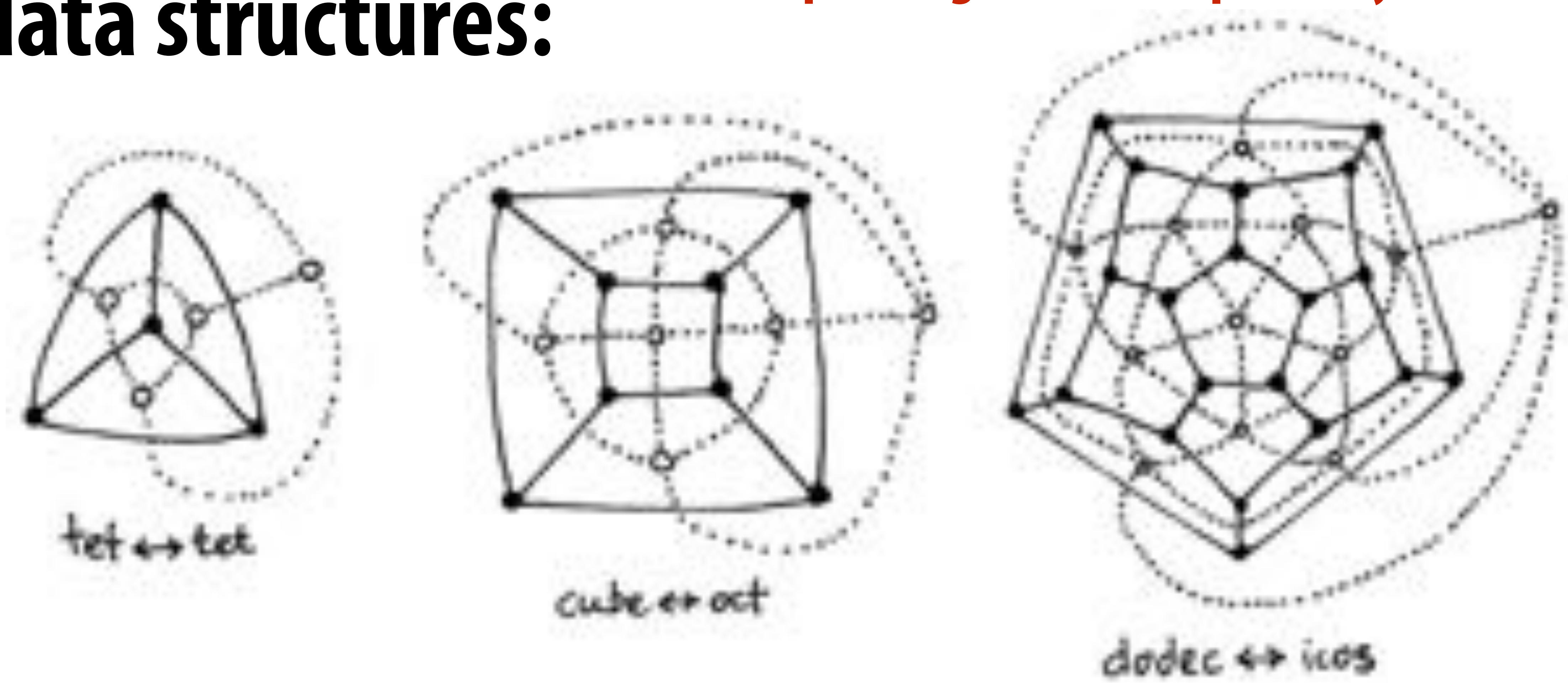


- Now have to delete elements.
- Still lots of pointer assignments!
- Q: How would we implement this with a polygon soup?
- Any other good way to do it? (E.g., different data structure?)

Alternatives to Halfedge

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

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

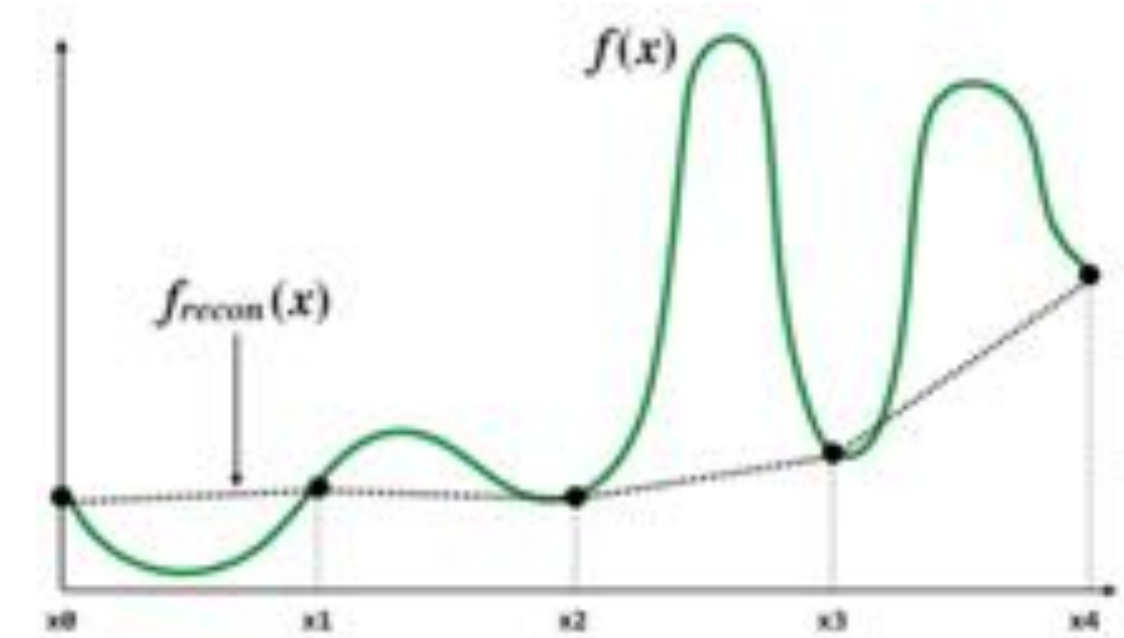


- **Each stores local neighborhood information**
- **Similar tradeoffs relative to simple polygon list:**
  - CONS: additional storage, incoherent memory access
  - PROS: better access time for individual elements, intuitive traversal of local neighborhoods
- **(Food for thought: can you design a halfedge-like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

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

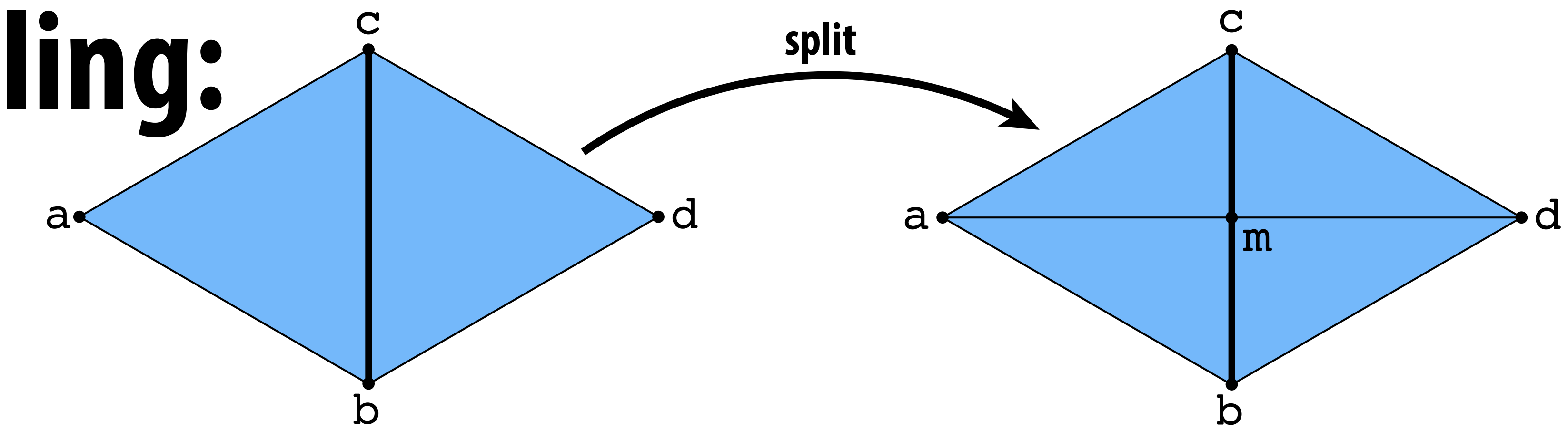
Remeshing as resampling

- Remember our discussion of aliasing
- Bad sampling makes signal appear different than it really is
- E.g., undersampled curve looks flat
- Geometry is no different!
  - undersampling destroys features
  - oversampling destroys performance
- How do we resample a geometric signal?

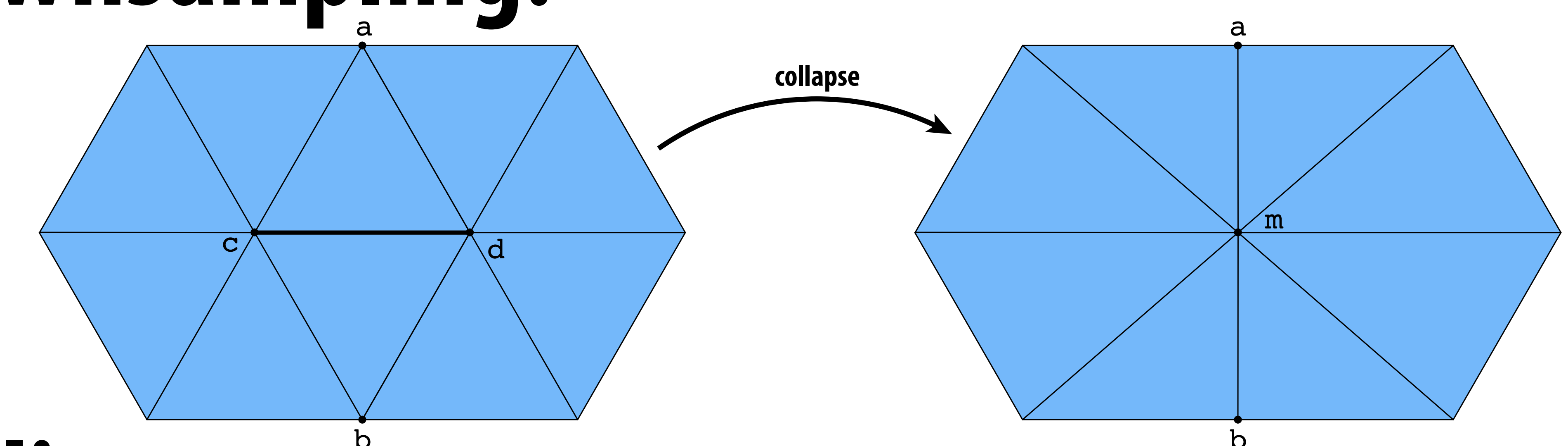


Already know how to resample!

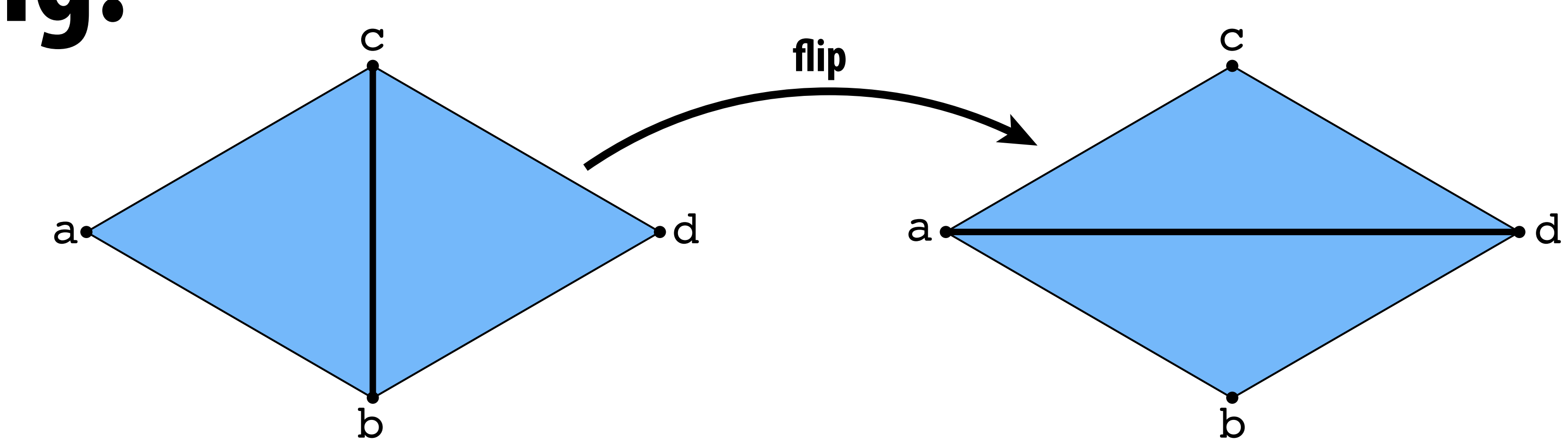
- Edge split is (local) upsampling:



- Edge collapse is (local) downsampling:



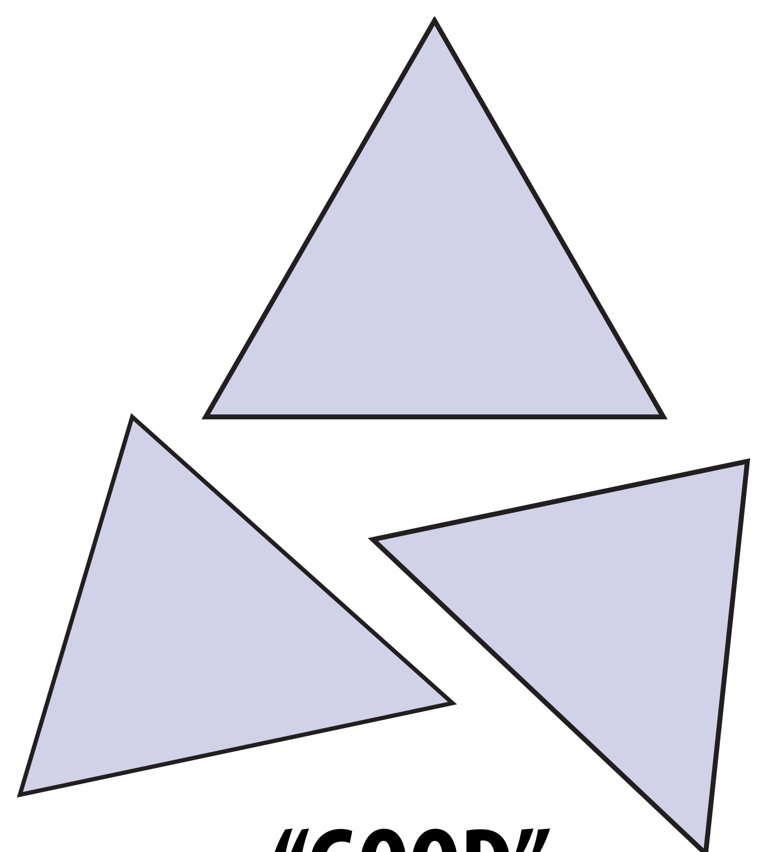
- Edge flip is (local) resampling:



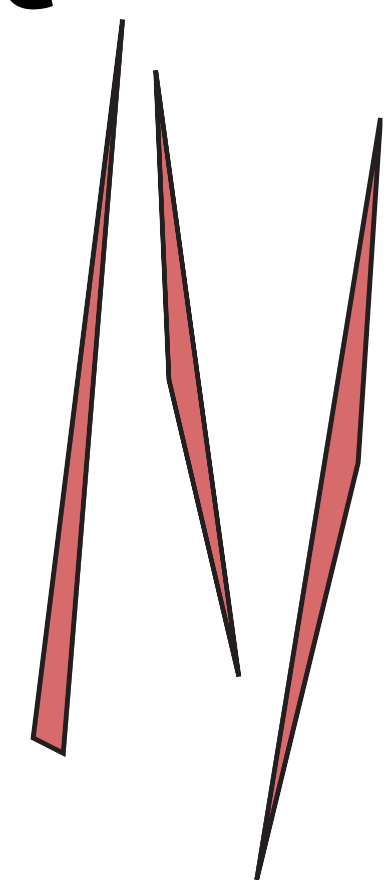
- Still need to intelligently decide which edges to modify!

What makes a "good" geometric signal?

- One rule of thumb: triangle shape

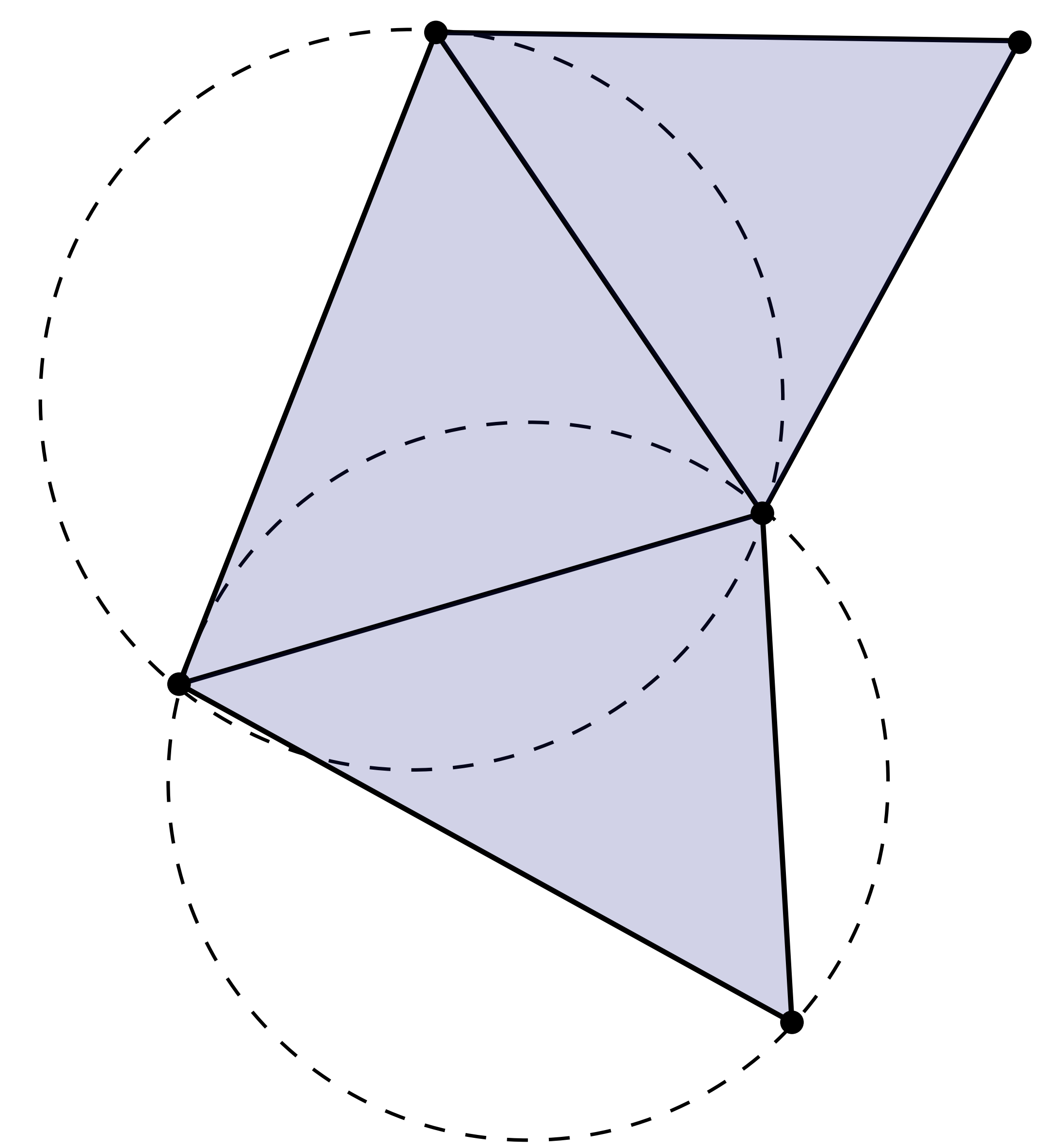


"GOOD"



"BAD"

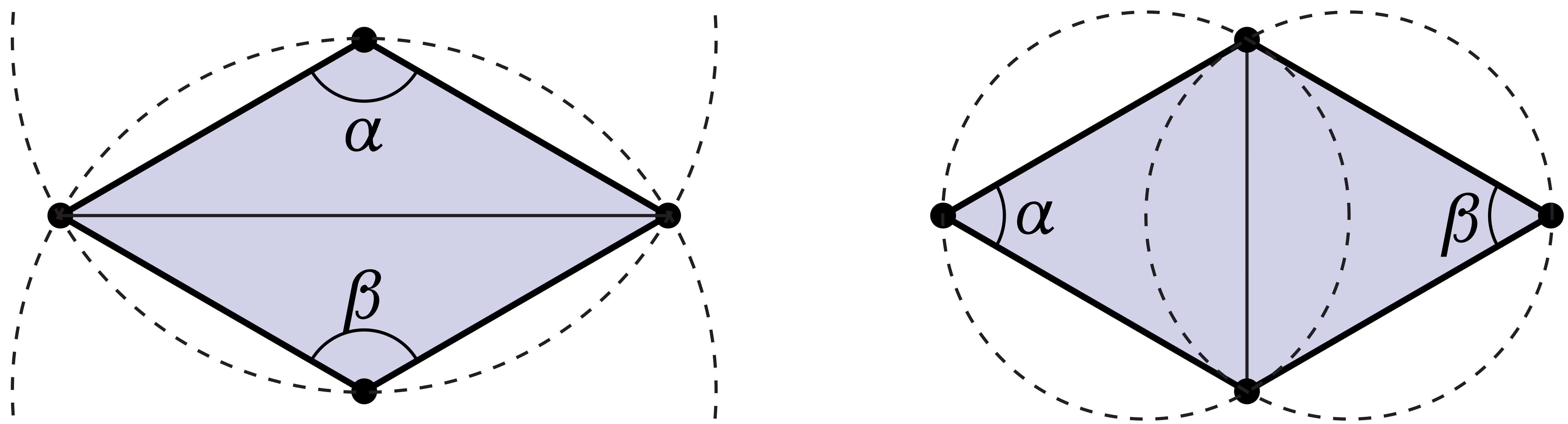
- More specific condition: Delaunay
- "Circumcircle interiors contain no vertices."
- Not always a good condition, but often*.



*See Shewchuk, "What is a Good Linear Element"

How do we make a mesh "more Delaunay"?

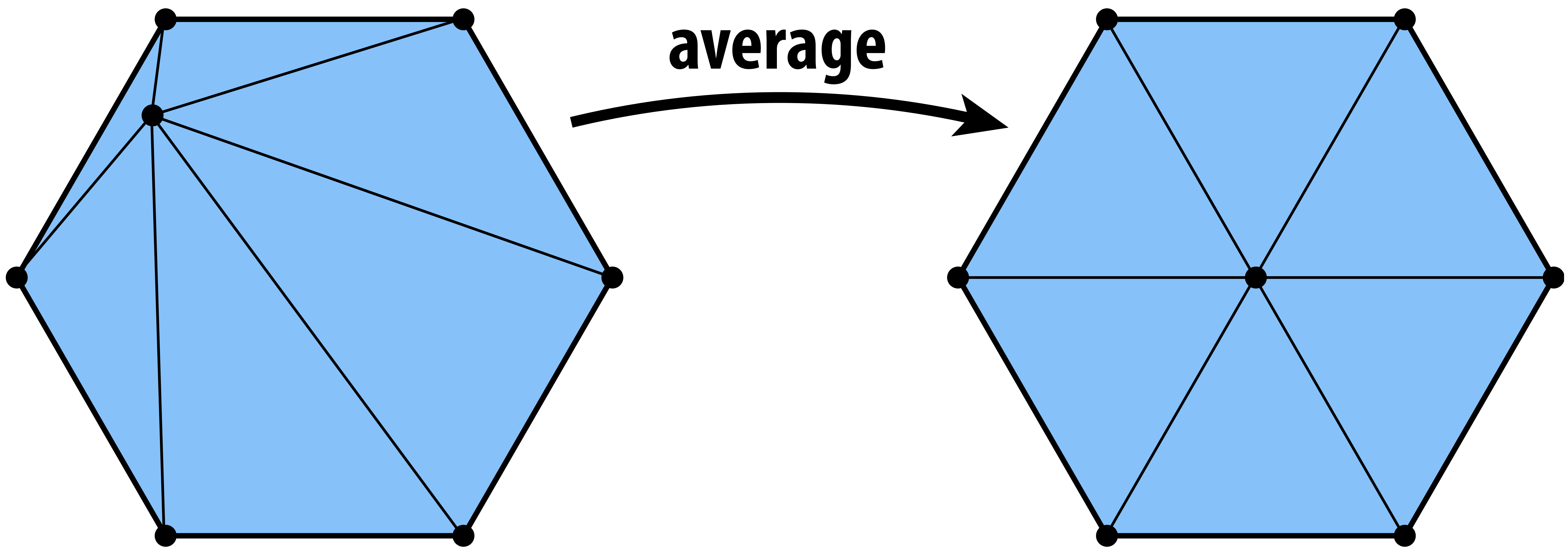
- **Already have a good tool: edge flips!**
- **If $\alpha+\beta > \pi$, flip it!**



- **FACT: in 2D, flipping edges eventually yields Delaunay mesh**
- **Theory: worst case $O(n^2)$; no longer true for surfaces in 3D.**
- **Practice: simple, effective way to improve mesh quality**

How do we make a triangles "more round"?

- Delaunay doesn't mean triangles are "round" (angles near 60°)

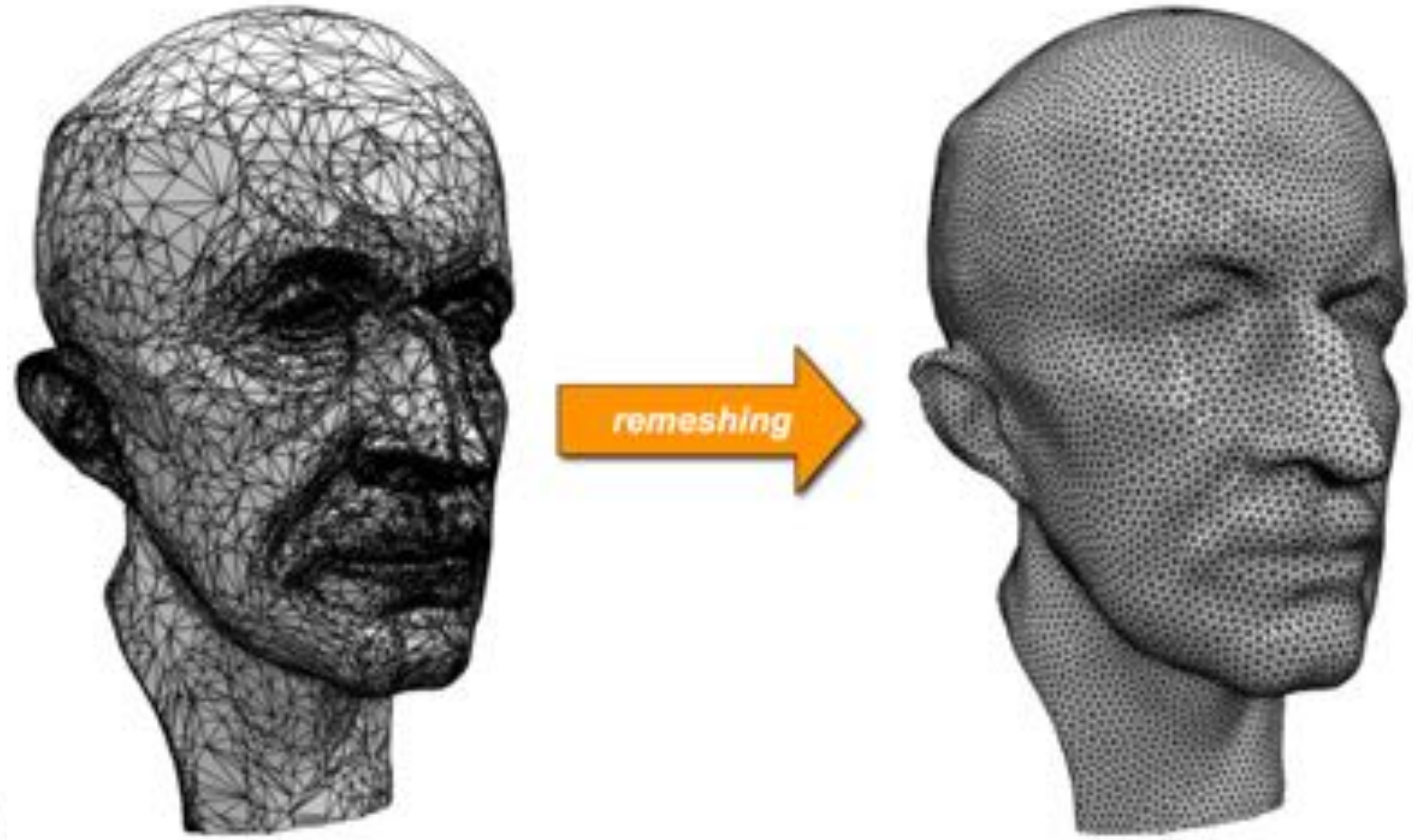


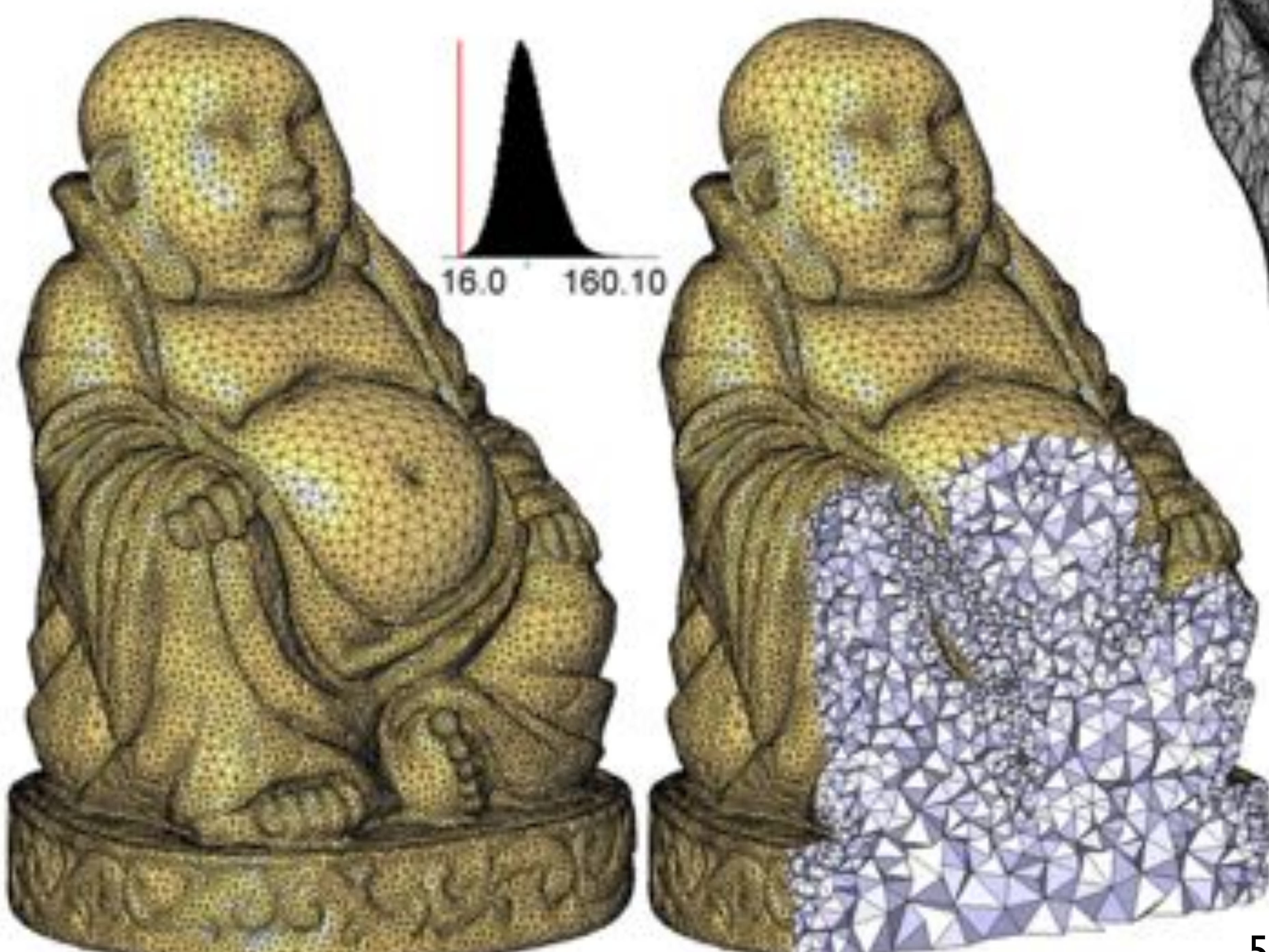
- Simple version of technique called "Laplacian smoothing".*

*See Crane, "Digital Geometry Processing with Discrete Exterior Calculus" http://keenan.is/dgpdec

Combine Smoothing + Refinement

- **Current best techniques do both**





What else makes a "good" geometric signal?

- Good approximation of original signal!
- Keep only elements that contribute information about shape.
  - simplification (e.g., quadric error metric)
- Add additional information where curvature is large.
  - subdivision (e.g., Loop, Catmull-Clark, etc.)
- Will see more of this in your assignment...!

What you should know:

- How to use split and average operations to do subdivision
- What is a manifold surface?
- Distinguish manifold from non-manifold surfaces
- Can a manifold surface have a boundary?   Give an example.
- Explain the idea of surface curvature with a diagram.
- Give an example of a surface where one of the principal curvatures is zero
- What do you need to store in a halfedge data structure?
- How can you find all vertices in a face with this data structure?
- How can you find all faces that contain a vertex with this data structure?
- Be able to perform edge flips, edge splits, and edge collapse with this data structure.
- BONUS: Think of an algorithm to traverse every face in a manifold using this data structure.