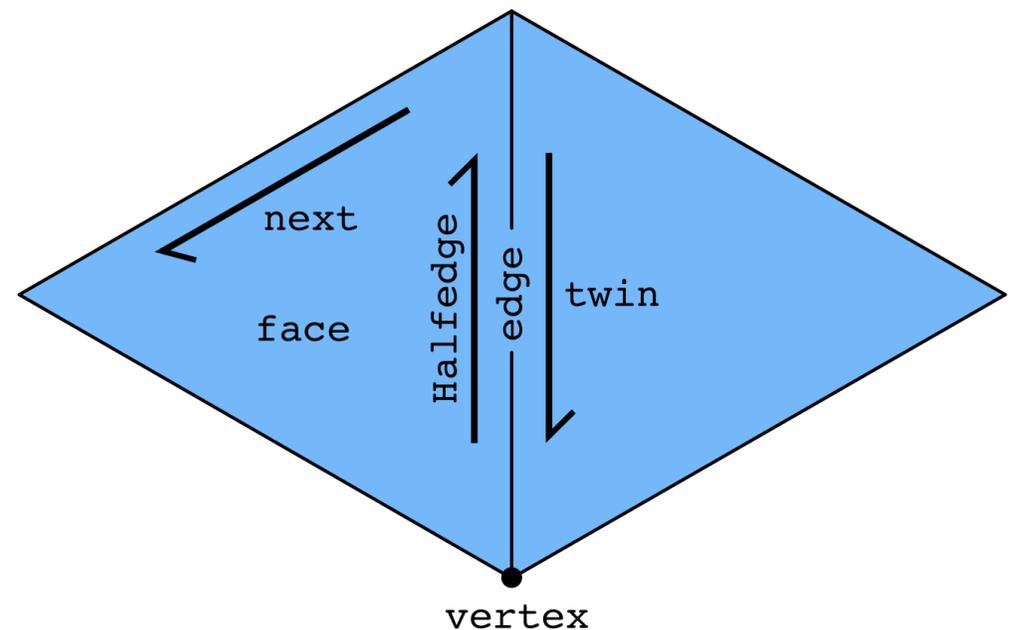
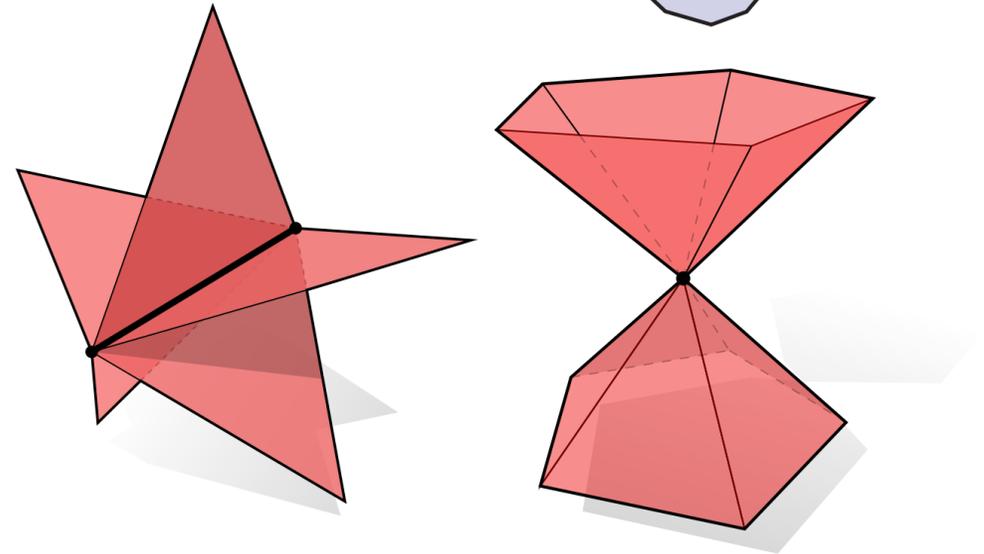
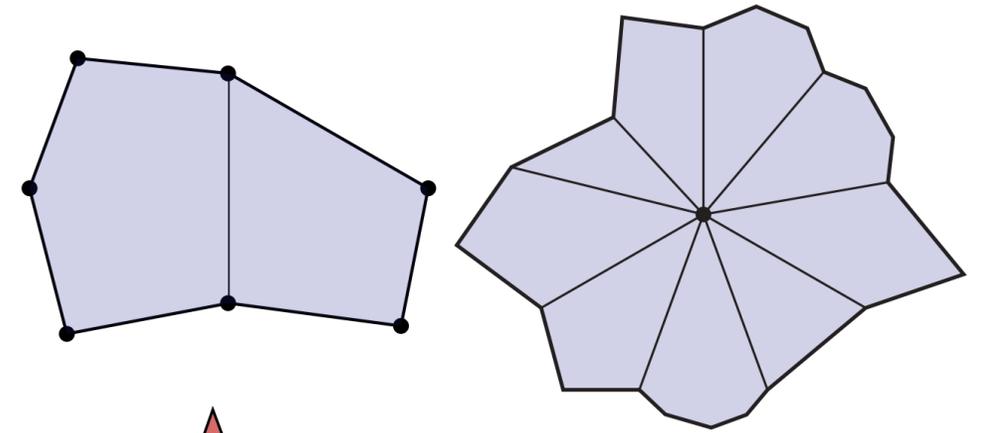


Geometry Processing & Geometric Queries

**Computer Graphics
CMU 15-462/15-662**

Last time: Meshes & Manifolds

- **Mathematical description of geometry**
 - simplifying assumption: *manifold*
 - for polygon meshes: “fans, not fins”
- **Data structures for surfaces**
 - polygon soup
 - halfedge mesh
 - storage cost vs. access time, etc.
- **Today:**
 - how do we manipulate geometry?
 - geometry processing / resampling



Today: Geometry Processing & Queries

- **Extend traditional digital signal processing (audio, video, etc.) to deal with *geometric* signals:**
 - **upsampling / downsampling / resampling / filtering ...**
 - **aliasing (reconstructed surface gives “false impression”)**
- **Also ask some basic questions about geometry:**
 - **What’s the closest point? Do two triangles intersect? Etc.**
- **Beyond pure geometry, these are basic building blocks for many algorithms in graphics (rendering, animation...)**

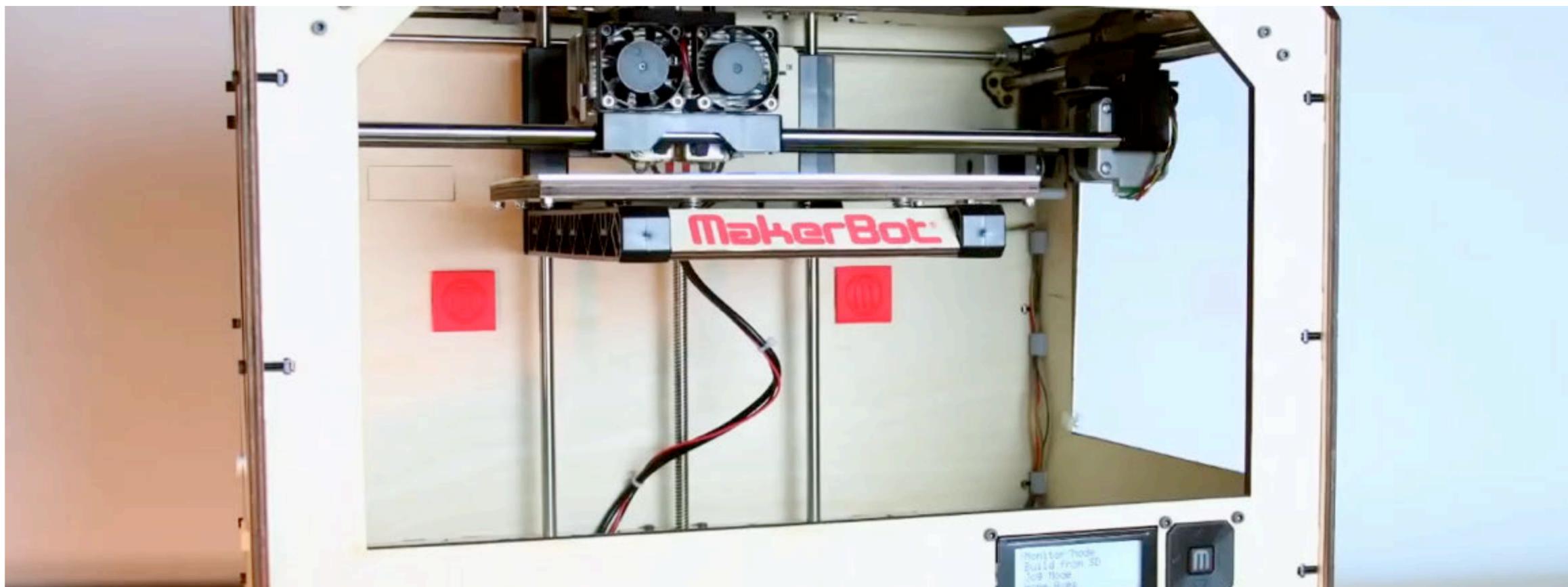


Digital Geometry Processing: Motivation

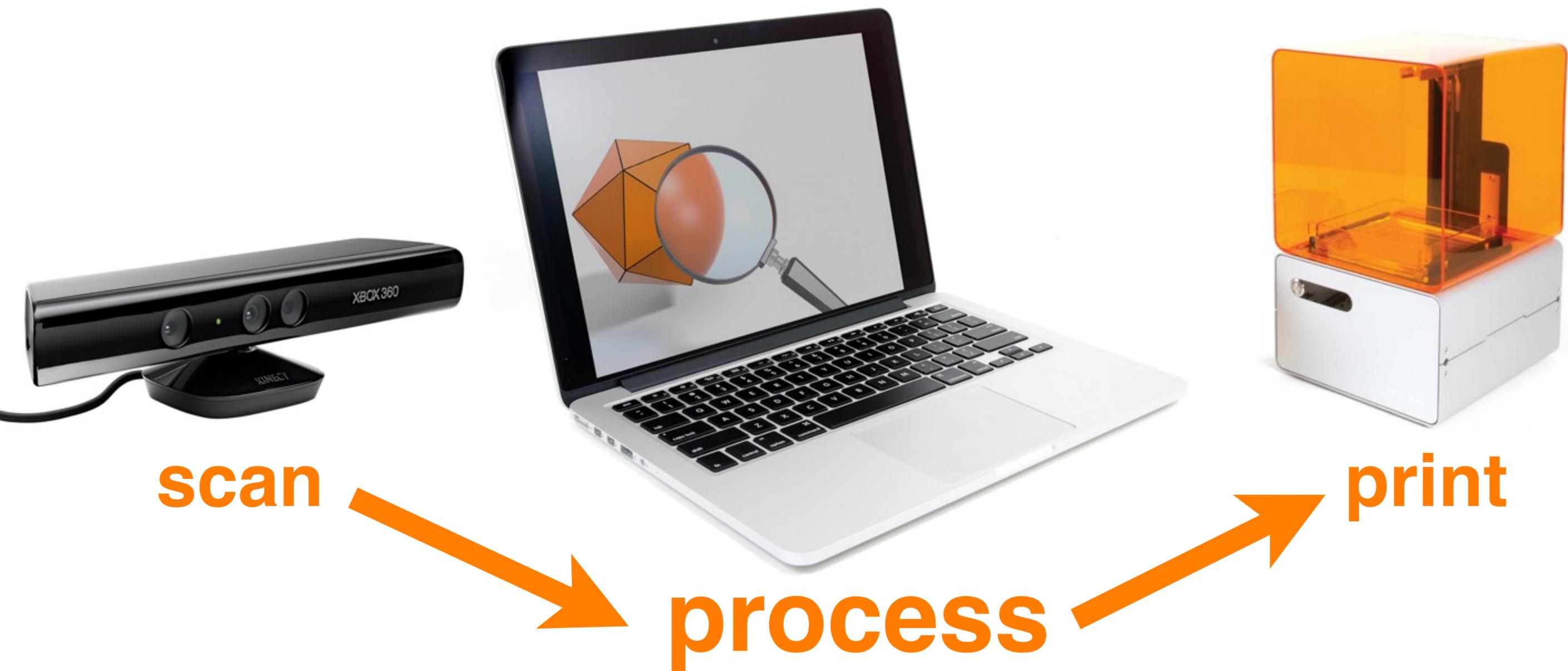
3D Scanning



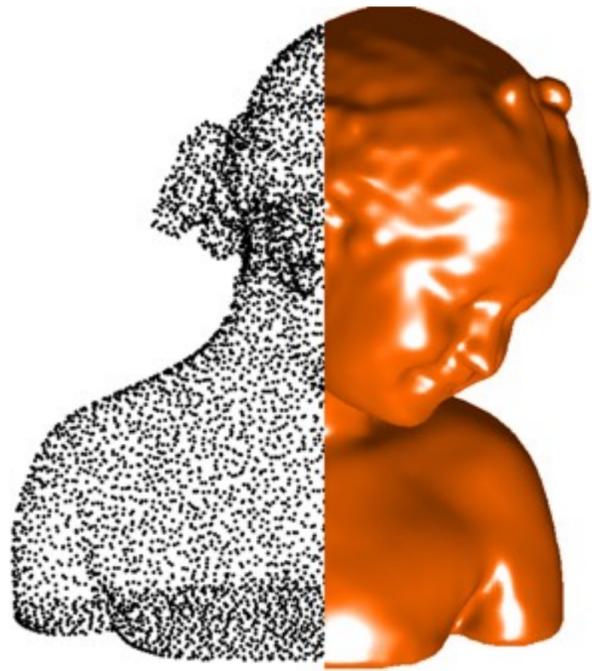
3D Printing



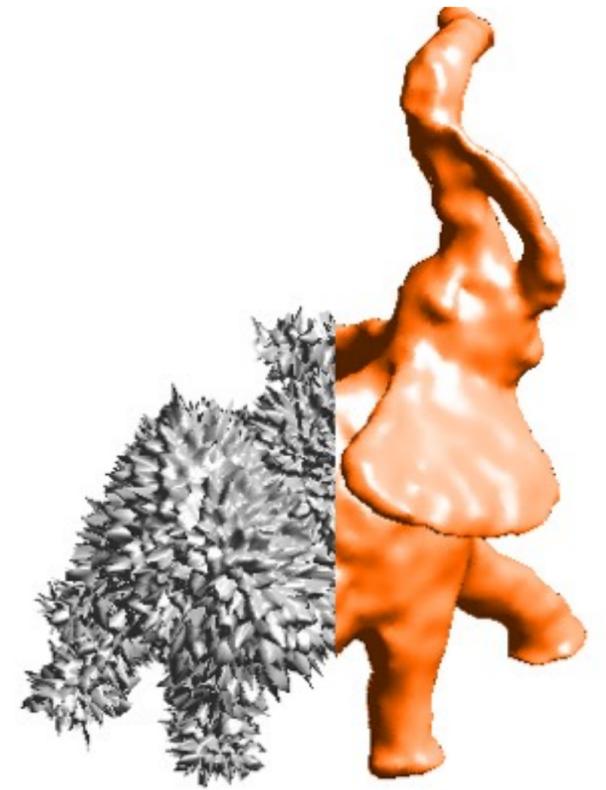
Geometry Processing Pipeline



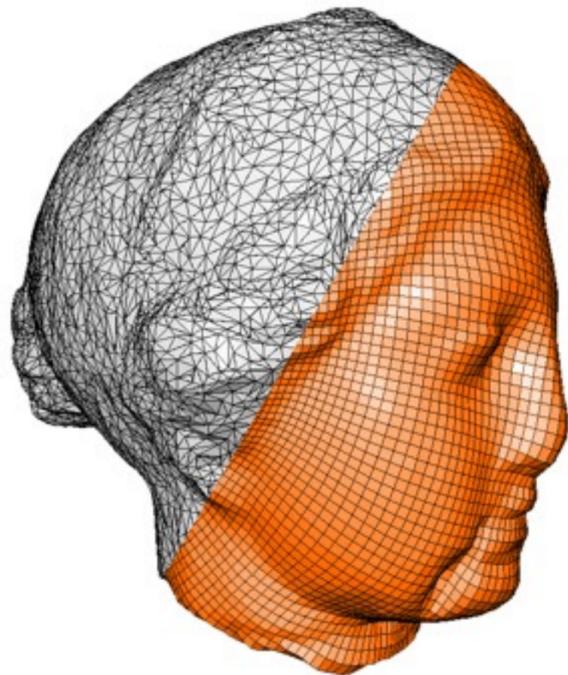
Geometry Processing Tasks



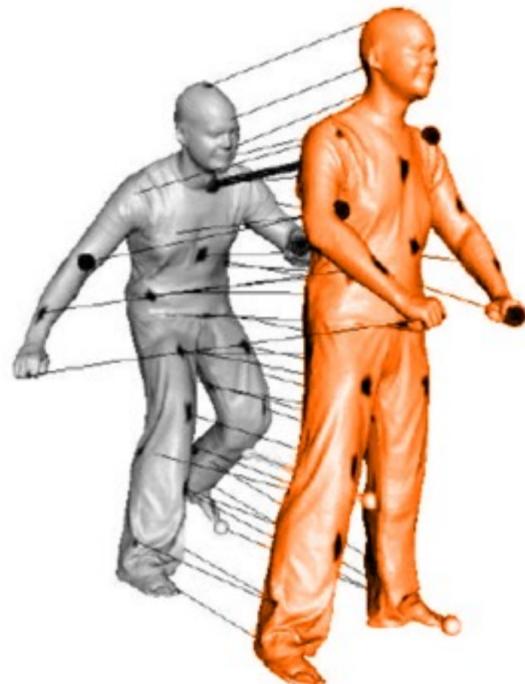
reconstruction



filtering



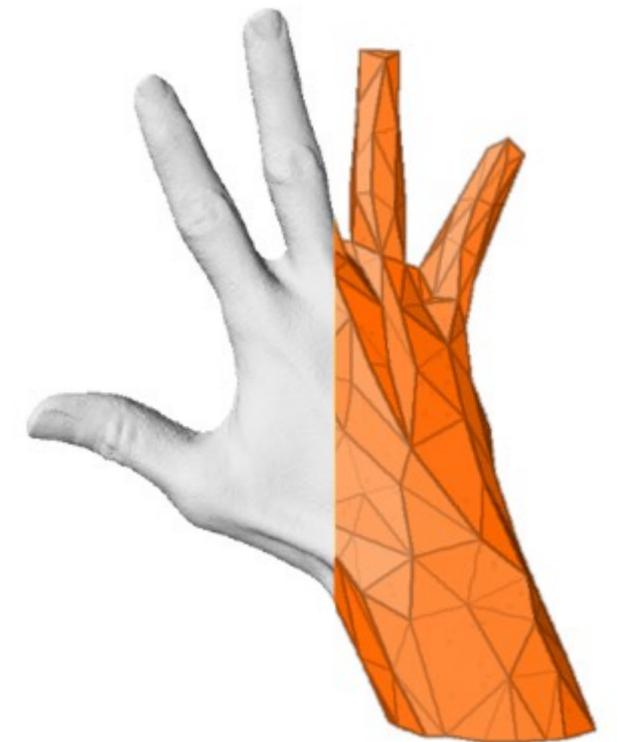
remeshing



shape analysis



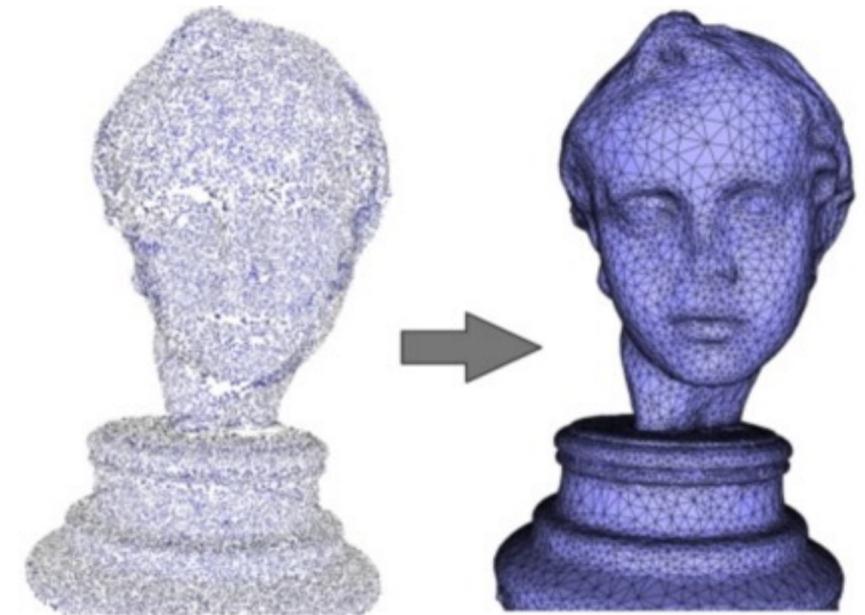
parameterization



compression

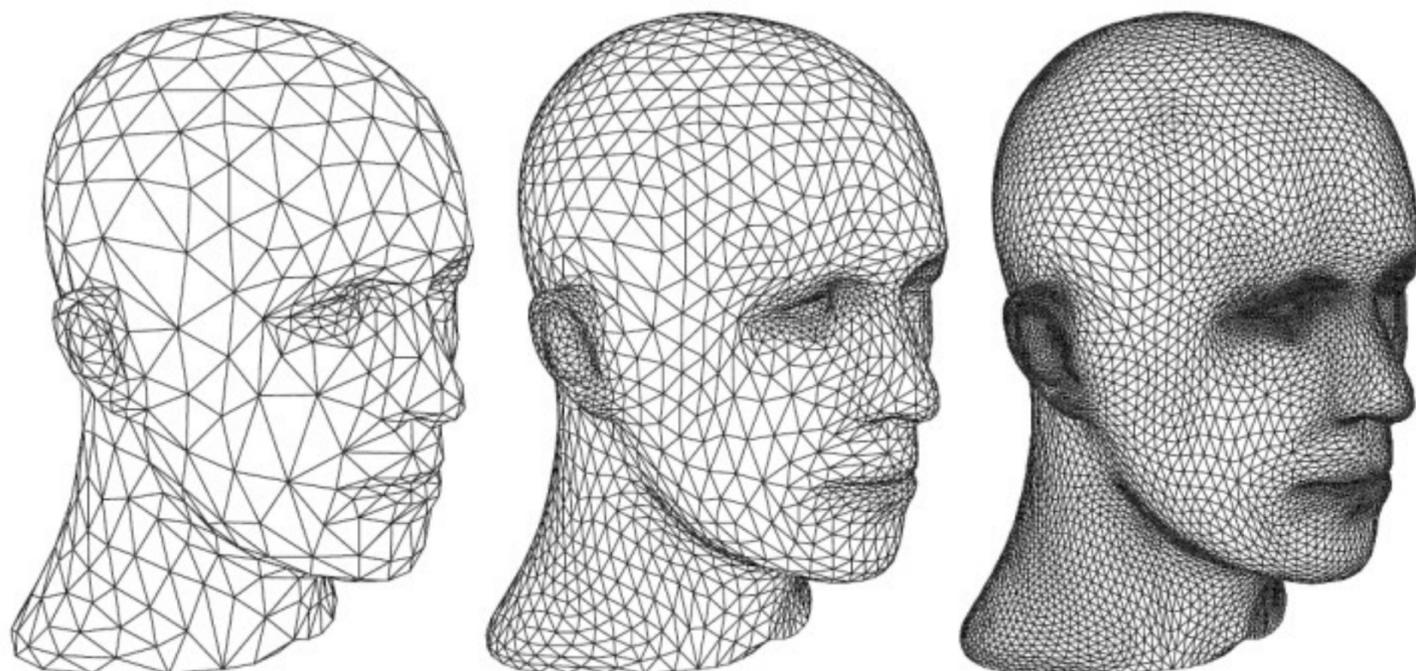
Geometry Processing: Reconstruction

- **Given samples of geometry, reconstruct surface**
- **What are “samples”? Many possibilities:**
 - **points, points & normals, ...**
 - **image pairs / sets (multi-view stereo)**
 - **line density integrals (MRI/CT scans)**
- **How do you get a surface? Many techniques:**
 - **silhouette-based (visual hull)**
 - **Voronoi-based (e.g., power crust)**
 - **PDE-based (e.g., Poisson reconstruction)**
 - **Radon transform / isosurfacing (marching cubes)**



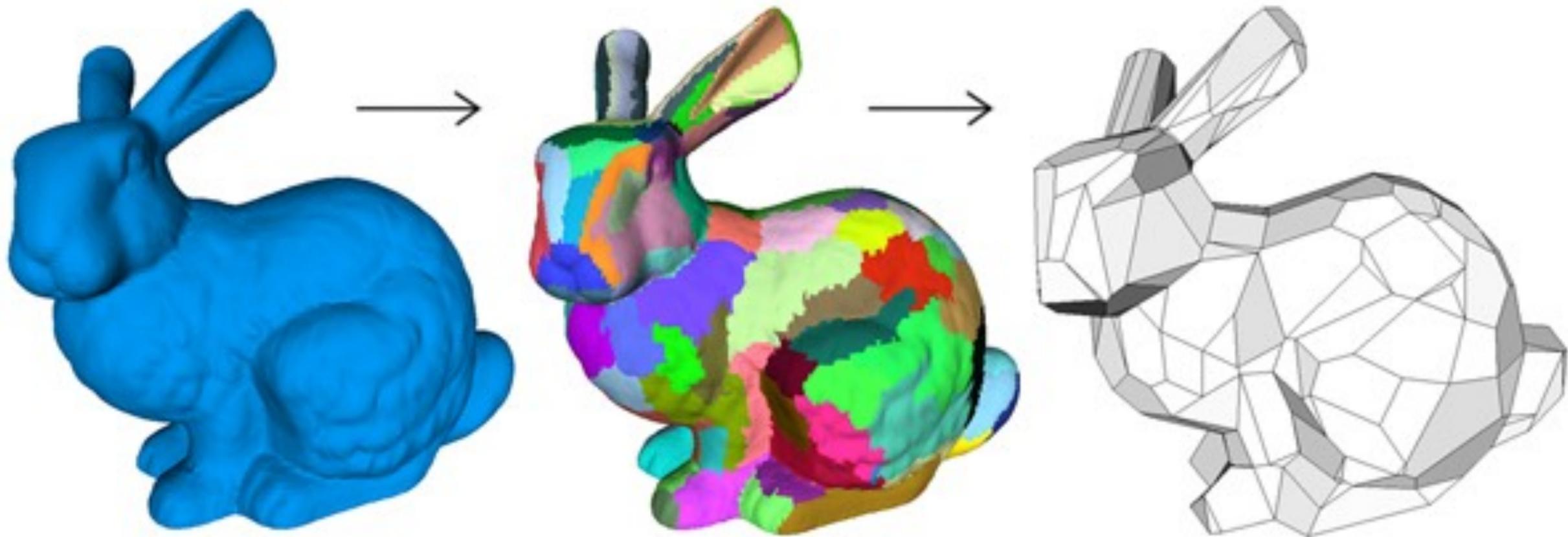
Geometry Processing: Upsampling

- Increase resolution via interpolation
- Images: e.g., bilinear, bicubic interpolation
- Polygon meshes:
 - subdivision
 - bilateral upsampling
 - ...



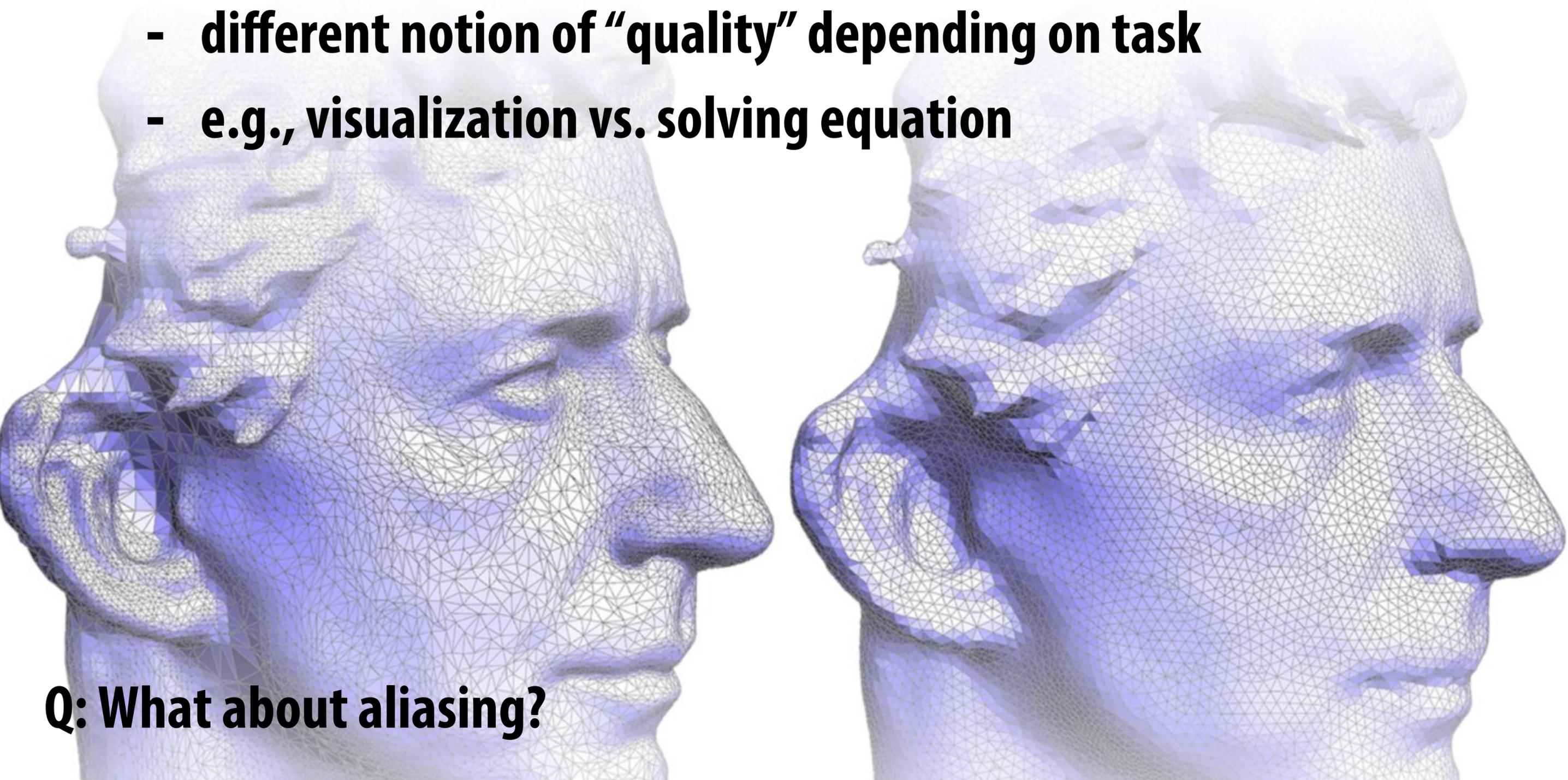
Geometry Processing: Downsampling

- **Decrease resolution; try to preserve shape/appearance**
- **Images: nearest-neighbor, bilinear, bicubic interpolation**
- **Point clouds: subsampling (just take fewer points!)**
- **Polygon meshes:**
 - **iterative decimation, variational shape approximation, ...**



Geometry Processing: Resampling

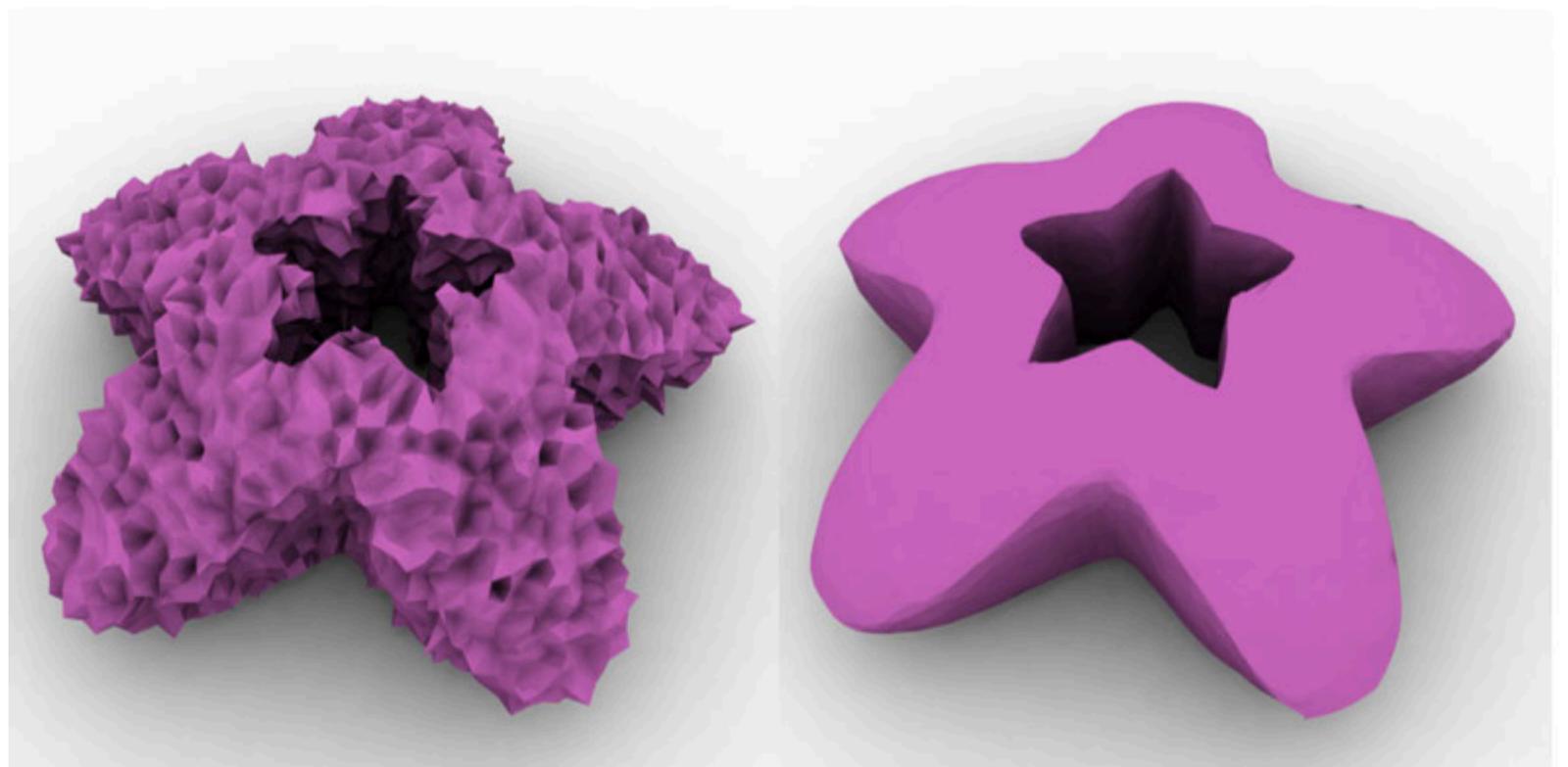
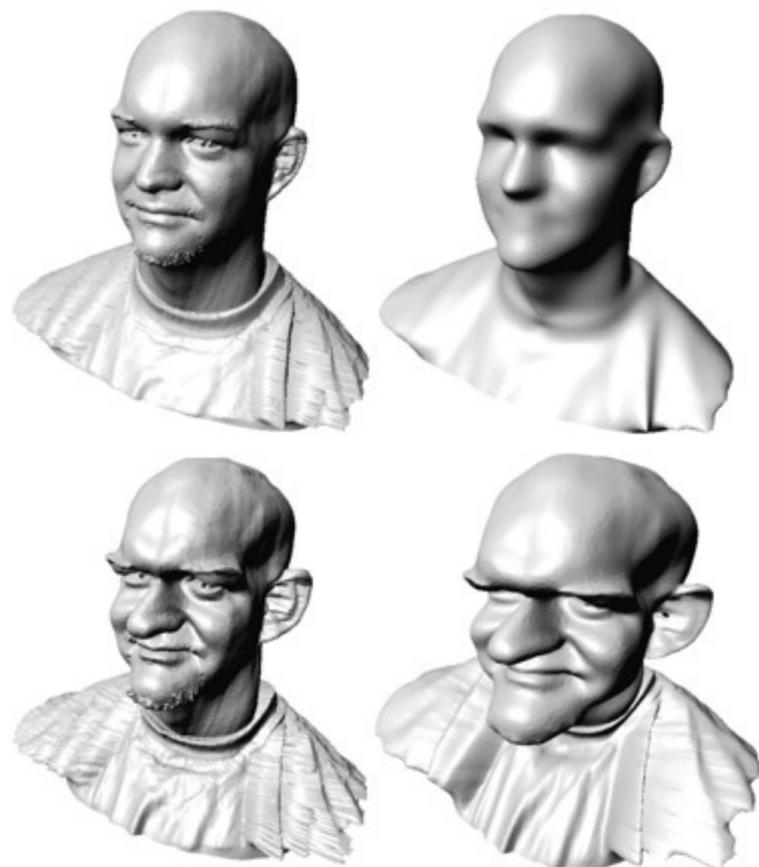
- **Modify sample distribution to improve quality**
- **Images: not an issue! (Pixels always stored on a regular grid)**
- **Meshes: *shape* of polygons is extremely important!**
 - **different notion of “quality” depending on task**
 - **e.g., visualization vs. solving equation**



Q: What about aliasing?

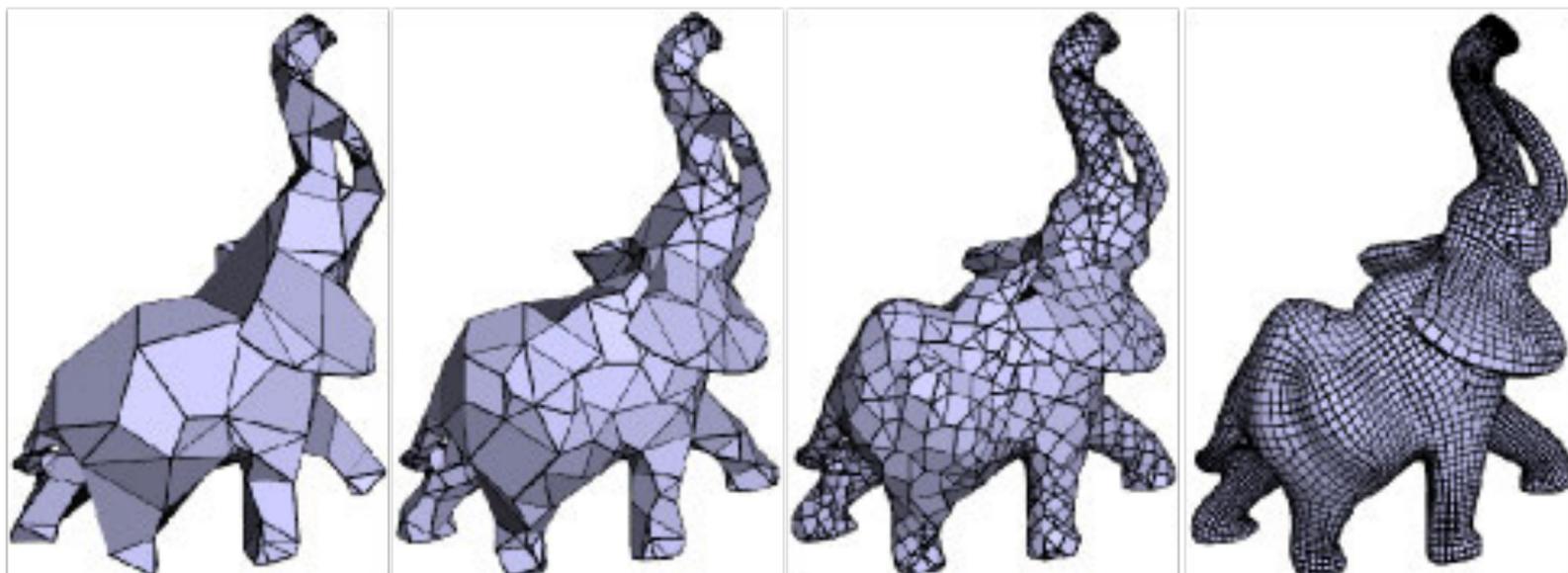
Geometry Processing: Filtering

- Remove noise, or emphasize important features (e.g., edges)
- Images: blurring, bilateral filter, edge detection, ...
- Polygon meshes:
 - curvature flow
 - bilateral filter
 - spectral filter



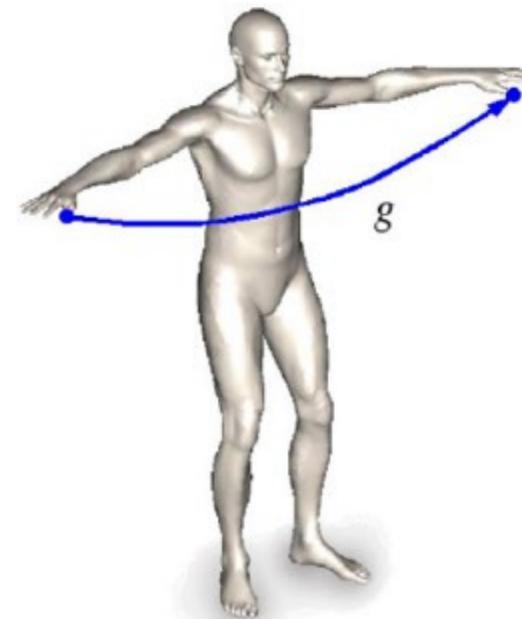
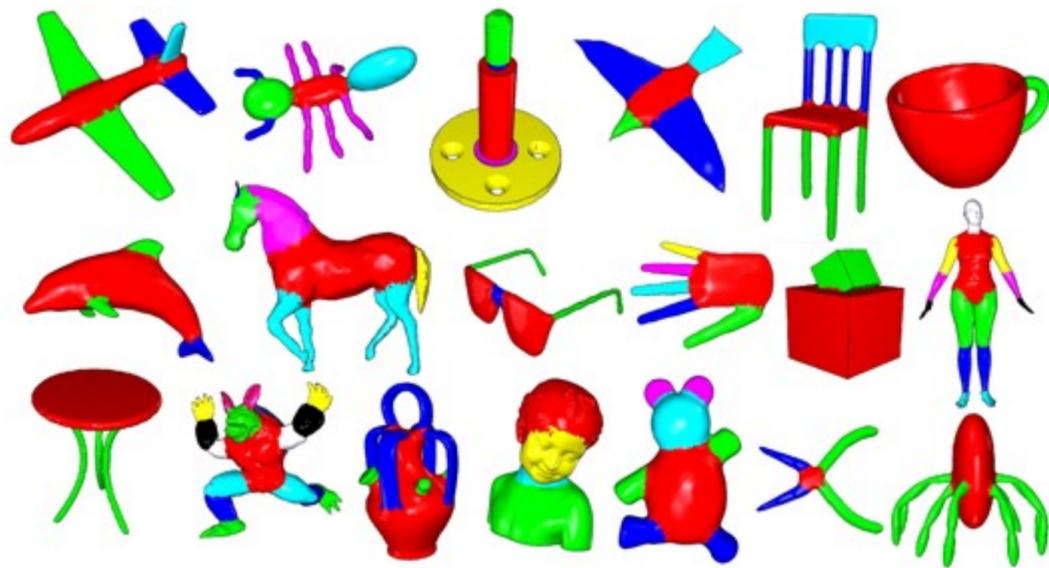
Geometry Processing: Compression

- Reduce storage size by eliminating redundant data/
approximating unimportant data
- Images:
 - run-length, Huffman coding - *lossless*
 - cosine/wavelet (JPEG/MPEG) - *lossy*
- Polygon meshes:
 - compress geometry *and* connectivity
 - many techniques (lossy & lossless)



Geometry Processing: Shape Analysis

- Identify/understand important semantic features
- Images: computer vision, segmentation, face detection, ...
- Polygon meshes:
 - segmentation, correspondence, symmetry detection, ...



Extrinsic symmetry



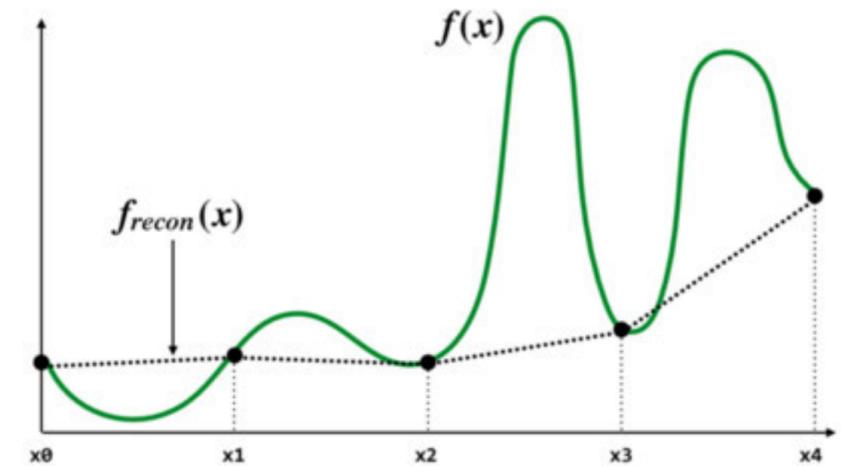
Intrinsic symmetry



**Enough overview—
Let's process some geometry!**

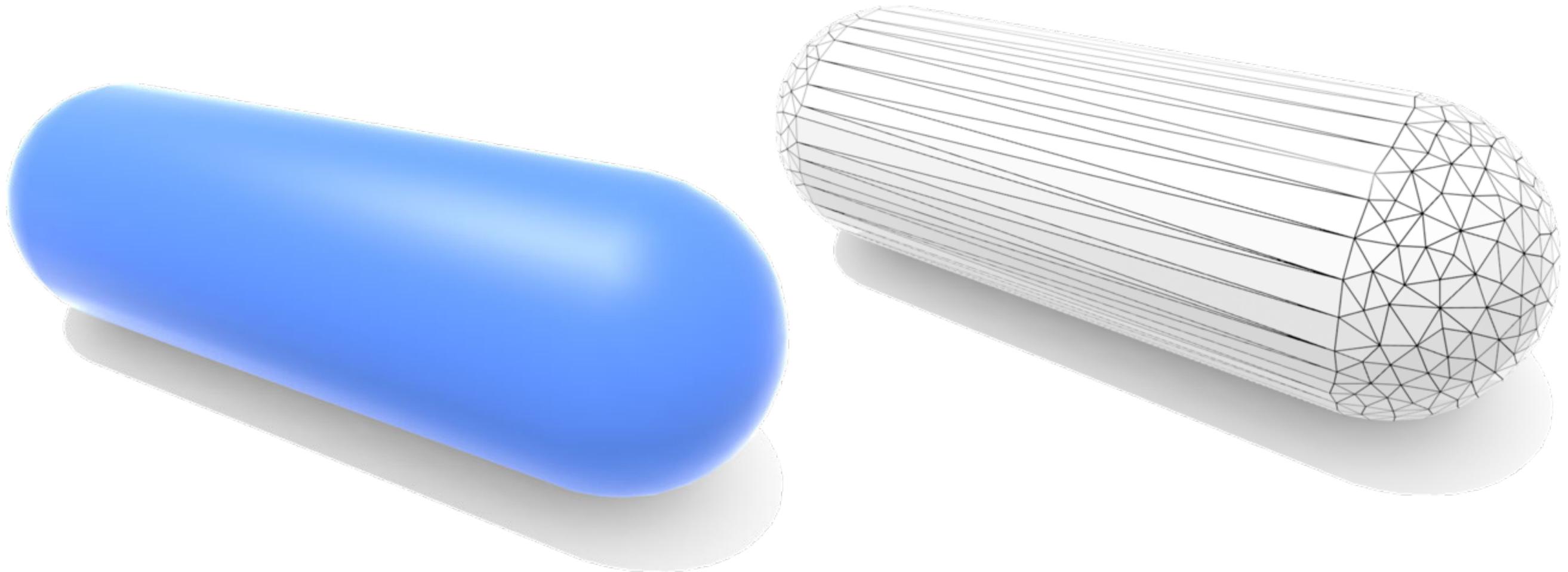
Remeshing as resampling

- 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



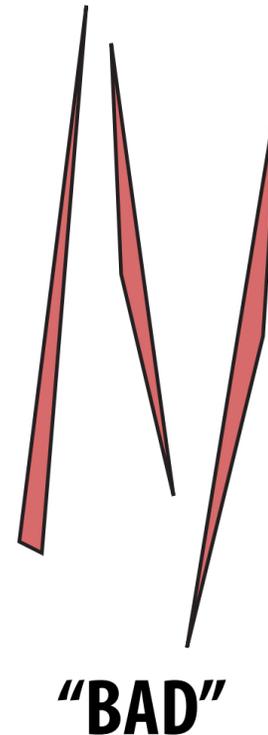
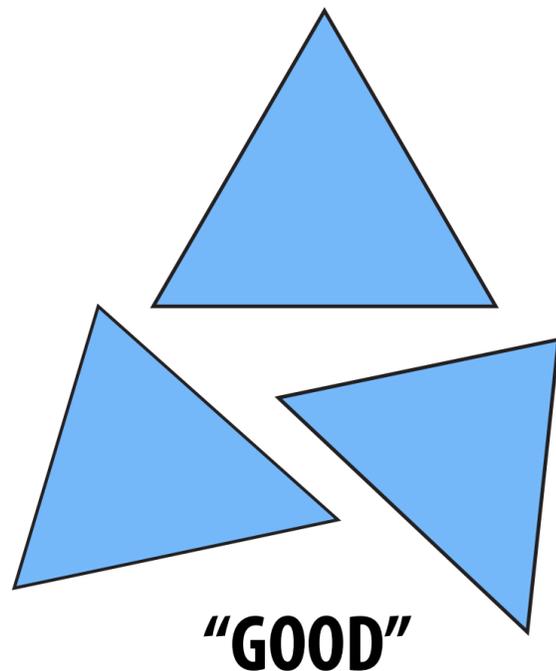
What makes a “good” mesh?

- One idea: good approximation of original shape!
- Keep only elements that contribute *information* about shape
- Add additional information where, e.g., *curvature* is large

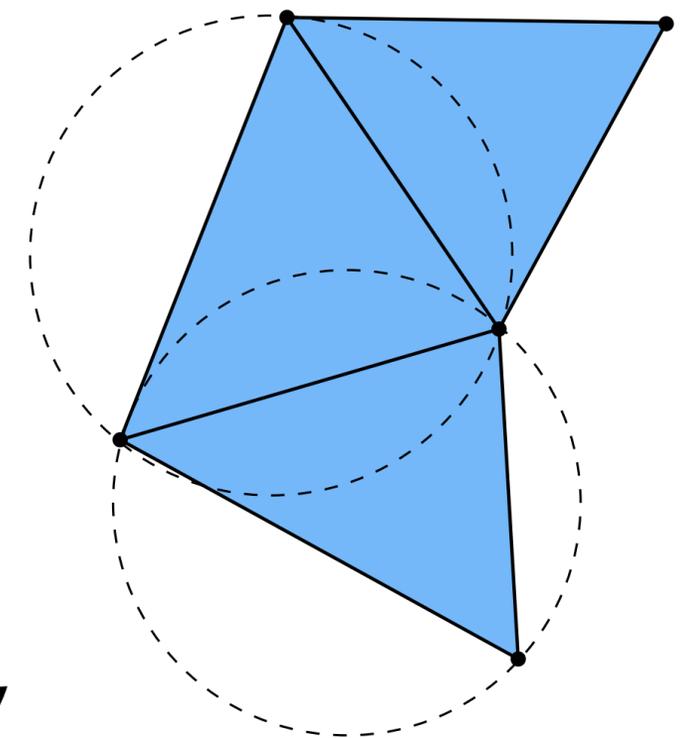


What else makes a “good” triangle mesh?

- Another rule of thumb: *triangle shape*



DELAUNAY

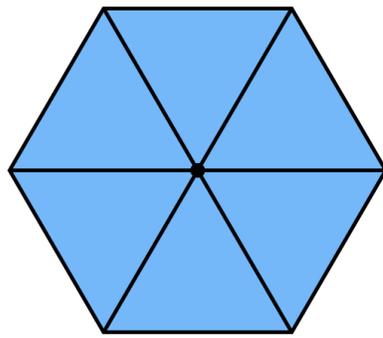


- E.g., all angles close to 60 degrees
- More sophisticated condition: *Delaunay*
- Can help w/ numerical accuracy/stability
- Tradeoffs w/ good geometric approximation*
 - e.g., long & skinny might be “more efficient”

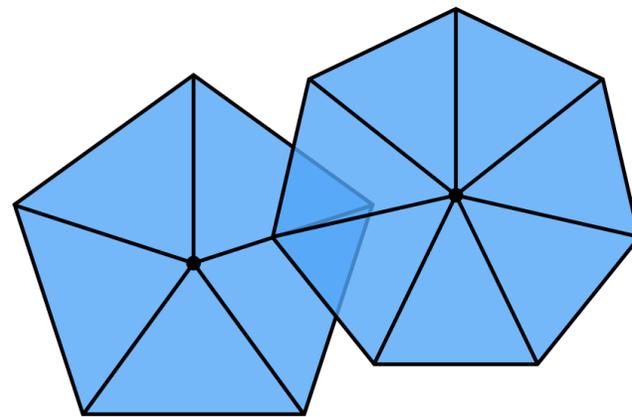
*See Shewchuk, “What is a Good Linear Element”

What else constitutes a good mesh?

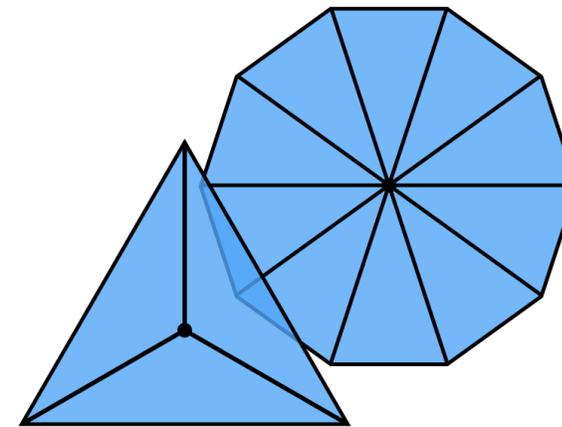
- Another rule of thumb: *regular vertex degree*
- E.g., valence 6 for triangle meshes (equilateral)



"GOOD"

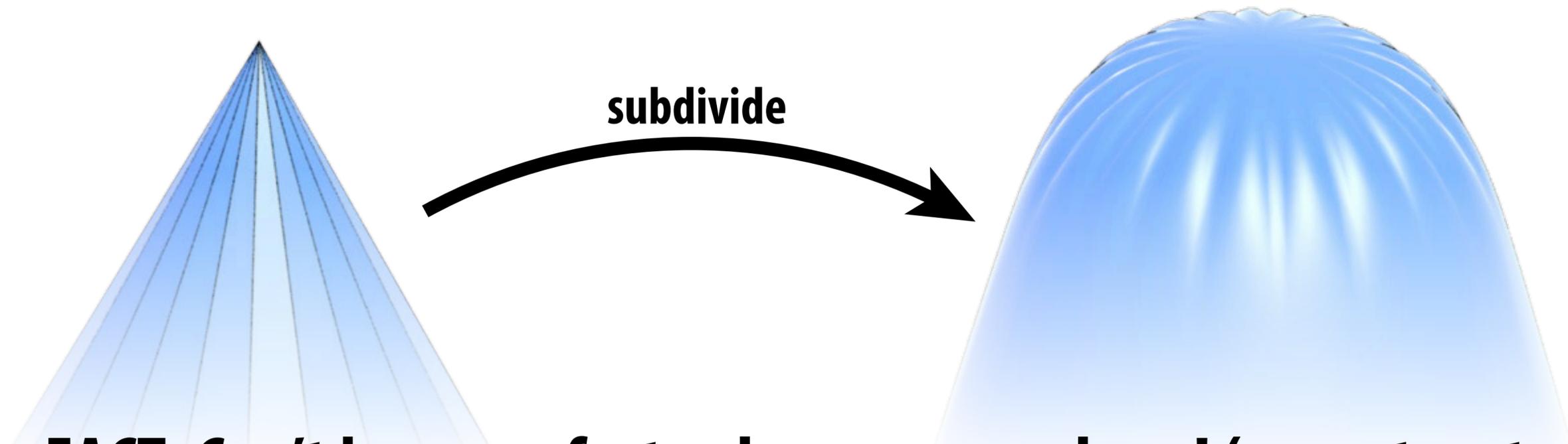


"OK"



"BAD"

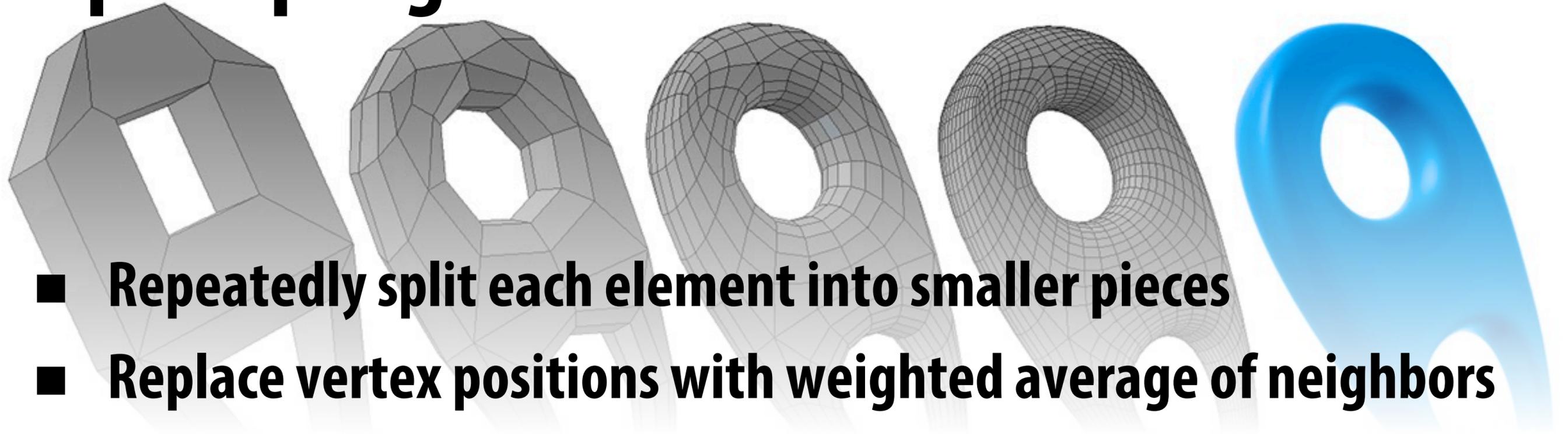
- Why? Better polygon shape, important for (e.g.) subdivision:



- **FACT: Can't have perfect valence everywhere! (except on torus)**

How do we upsample a mesh?

Upsampling via Subdivision



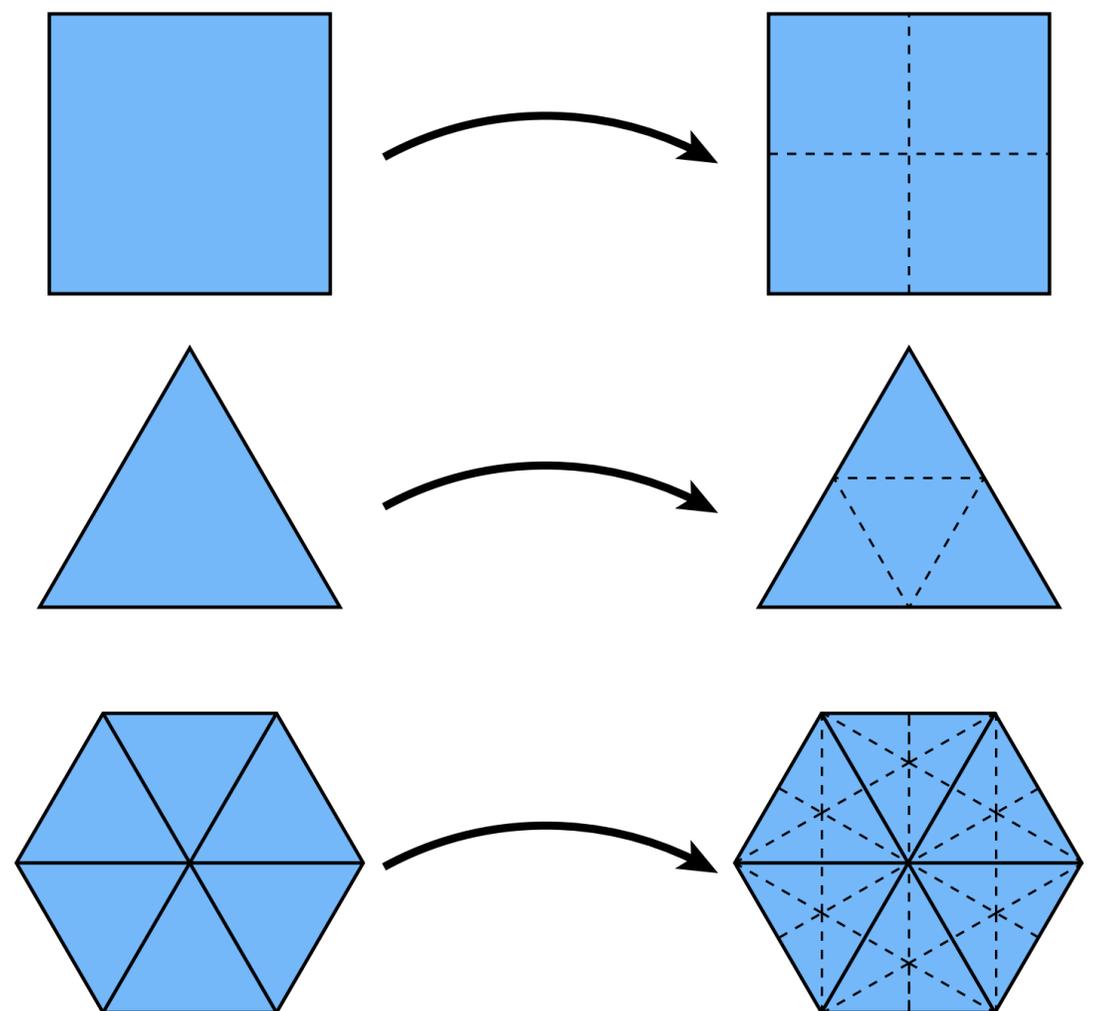
- Repeatedly split each element into smaller pieces
- Replace vertex positions with weighted average of neighbors

- Main considerations:

- interpolating vs. approximating
- limit surface continuity (C^1, C^2, \dots)
- behavior at irregular vertices

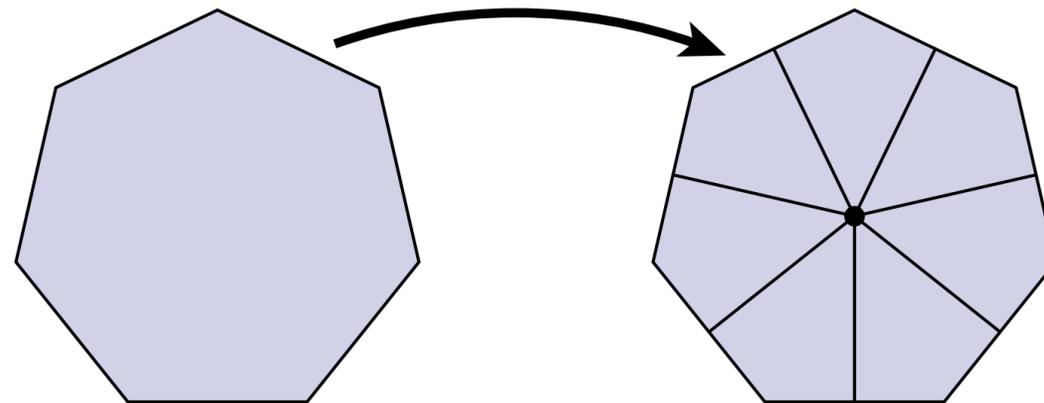
- Many options:

- Quad: Catmull-Clark
- Triangle: Loop, Butterfly, Sqrt(3)

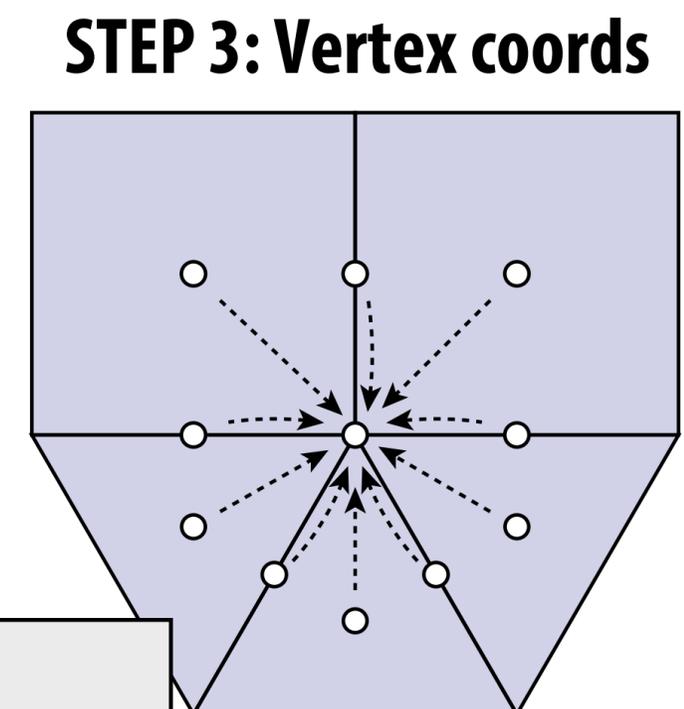
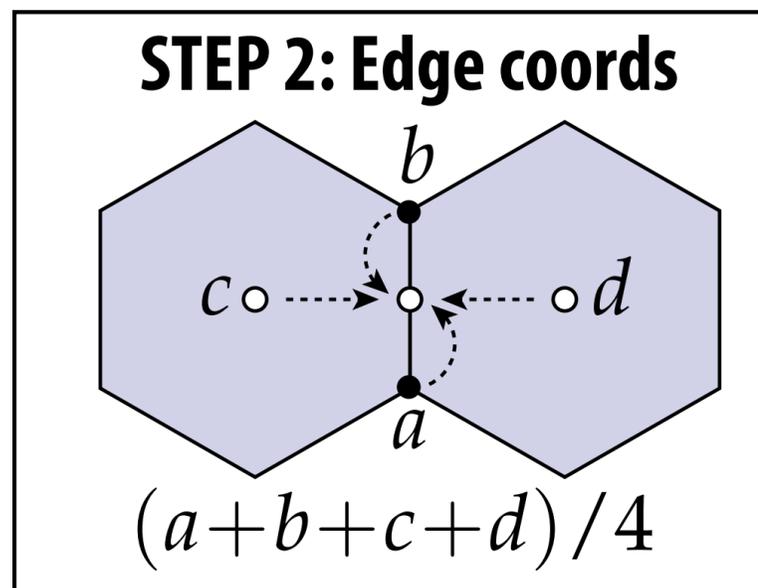
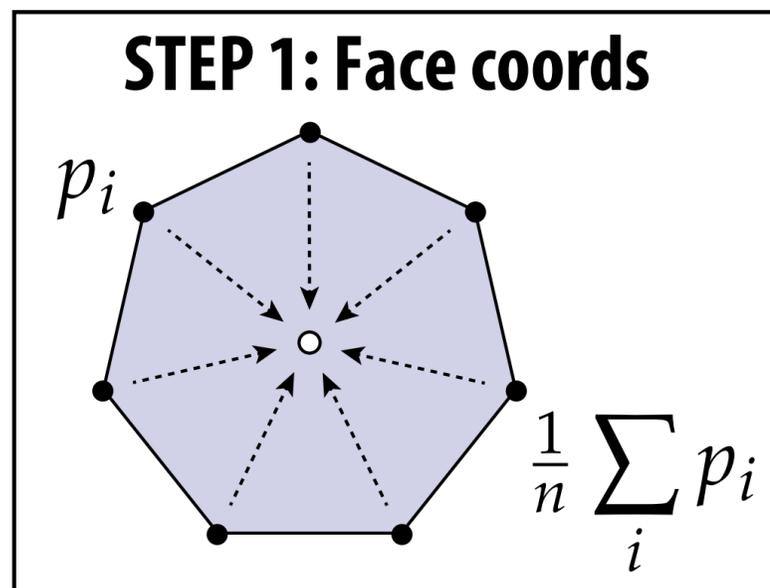


Catmull-Clark Subdivision

- **Step 0: split every polygon (any # of sides) into quadrilaterals:**



- **New vertex positions are weighted combination of old ones:**



New vertex coords:

$$\frac{Q + 2R + (n - 3)S}{n}$$

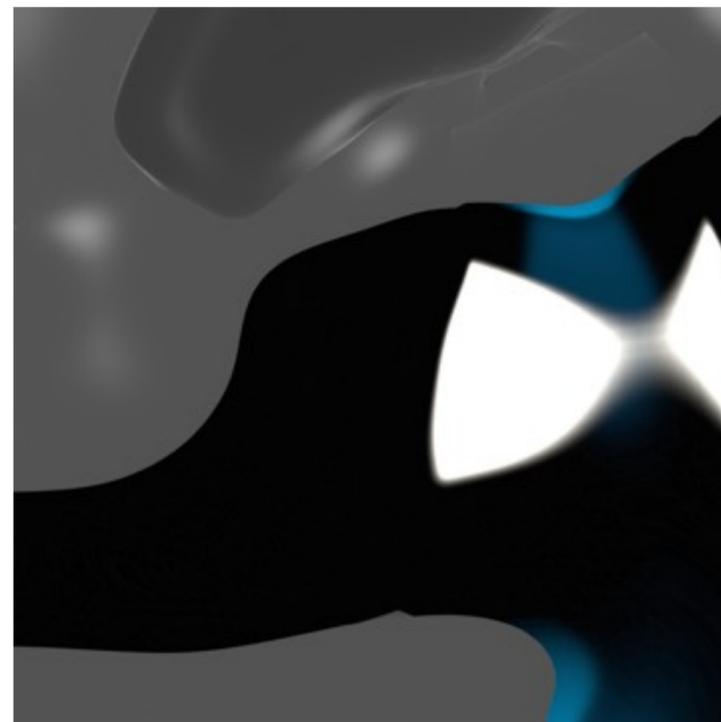
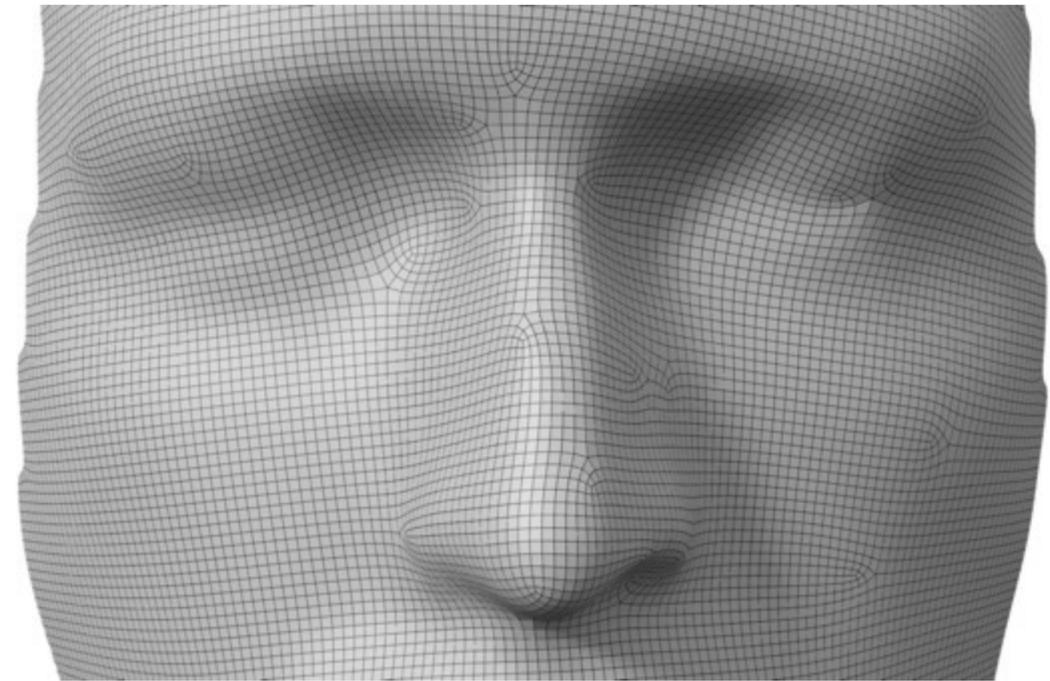
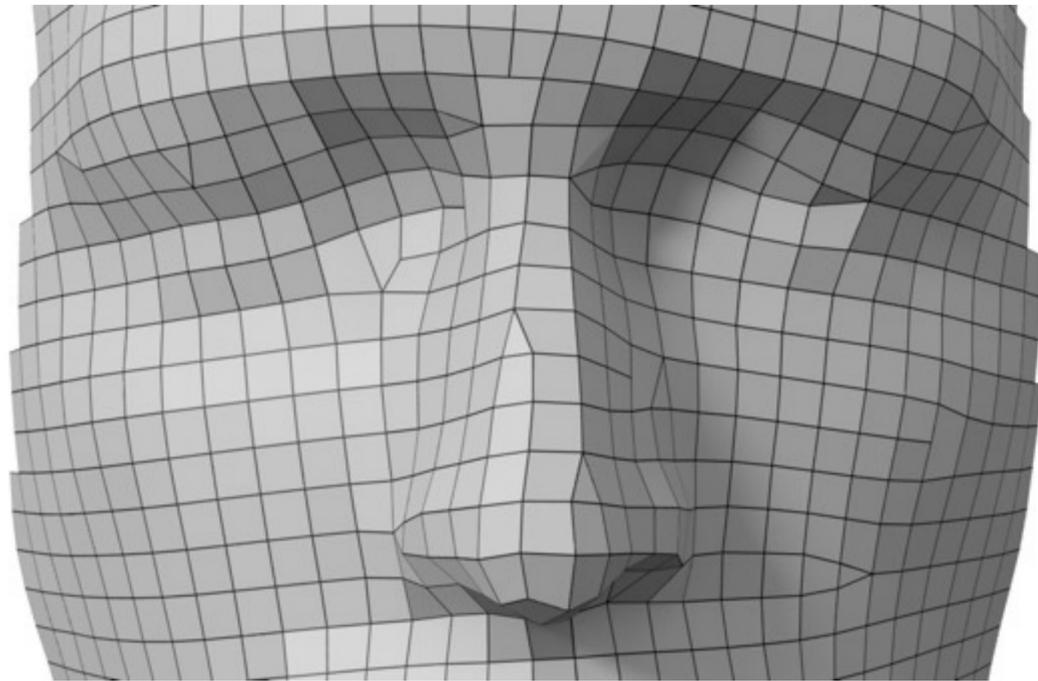
n - vertex degree

Q - average of face coords around vertex

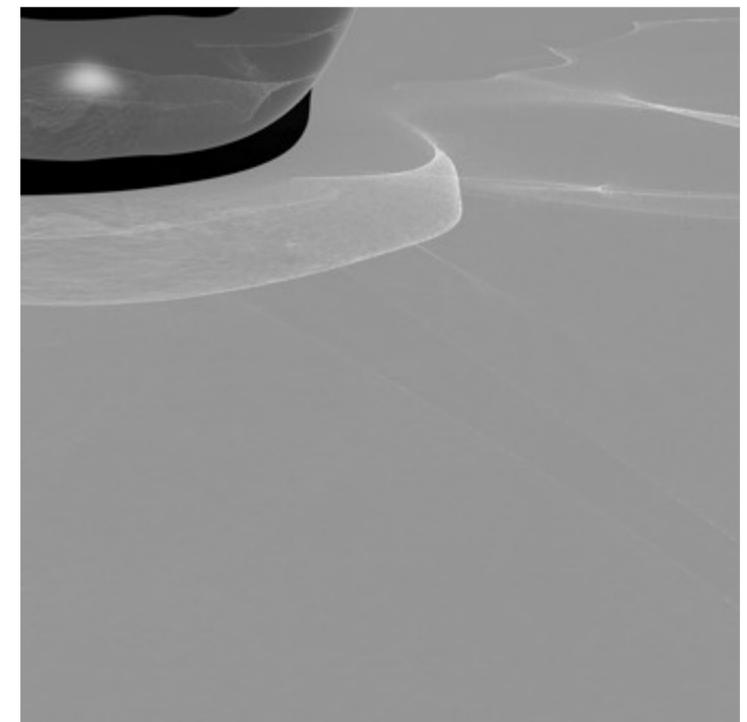
R - average of edge coords around vertex

S - original vertex position

Catmull-Clark on quad mesh

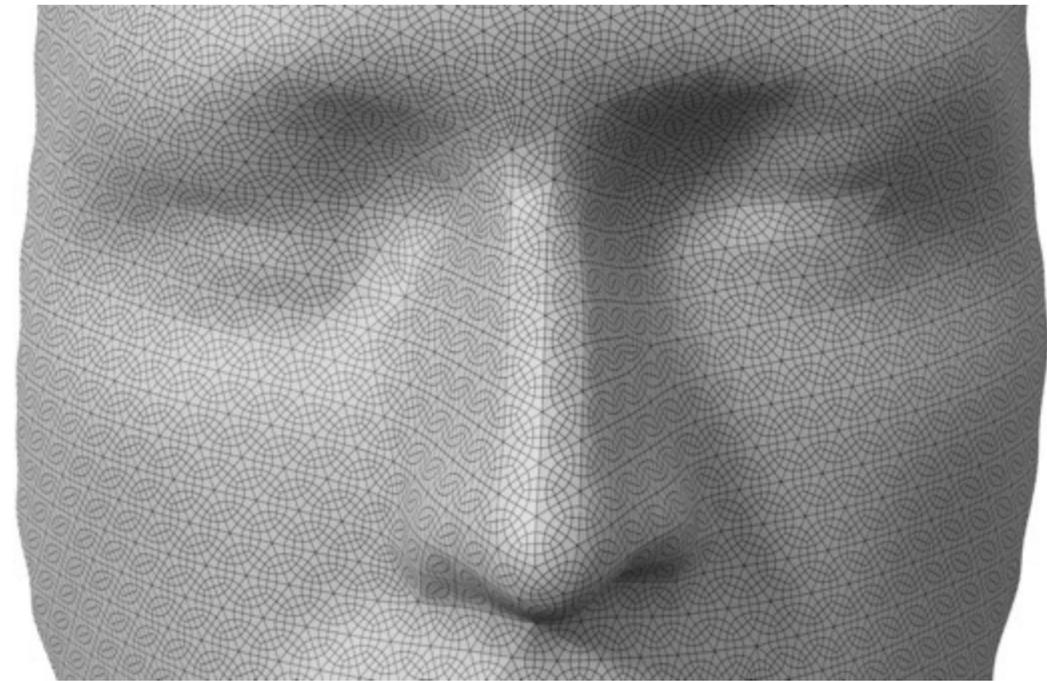
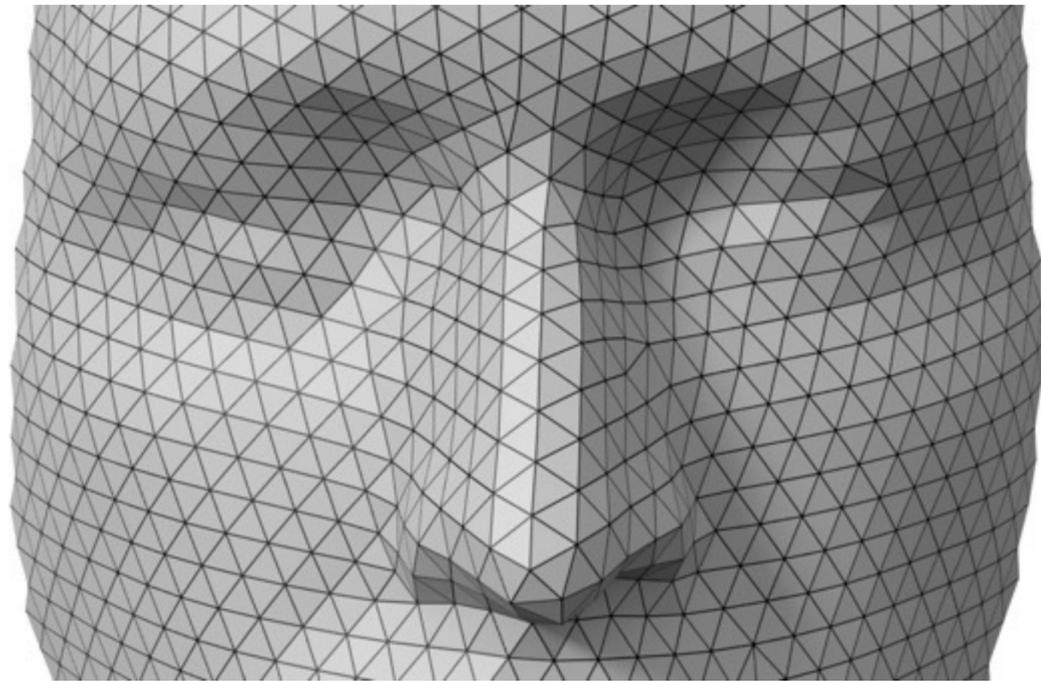


**smooth
reflection lines**

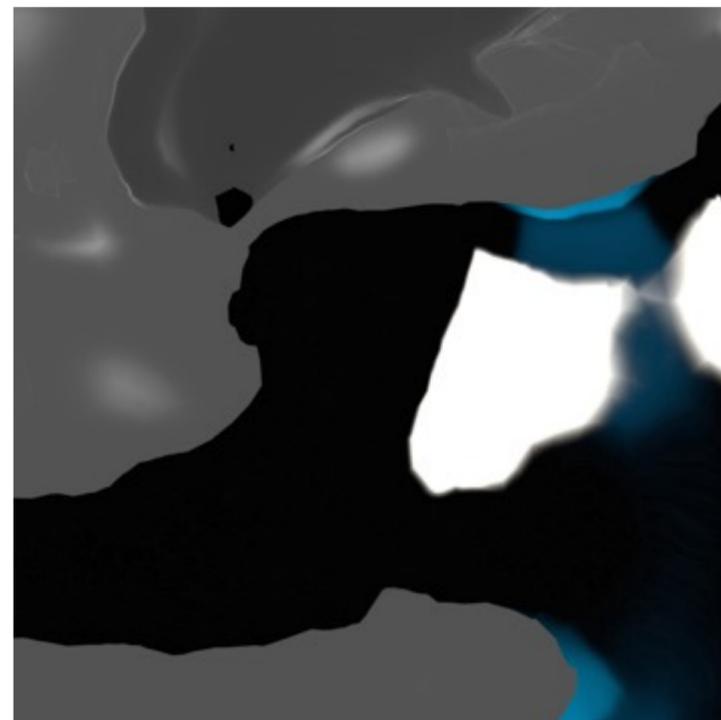


**smooth
caustics**

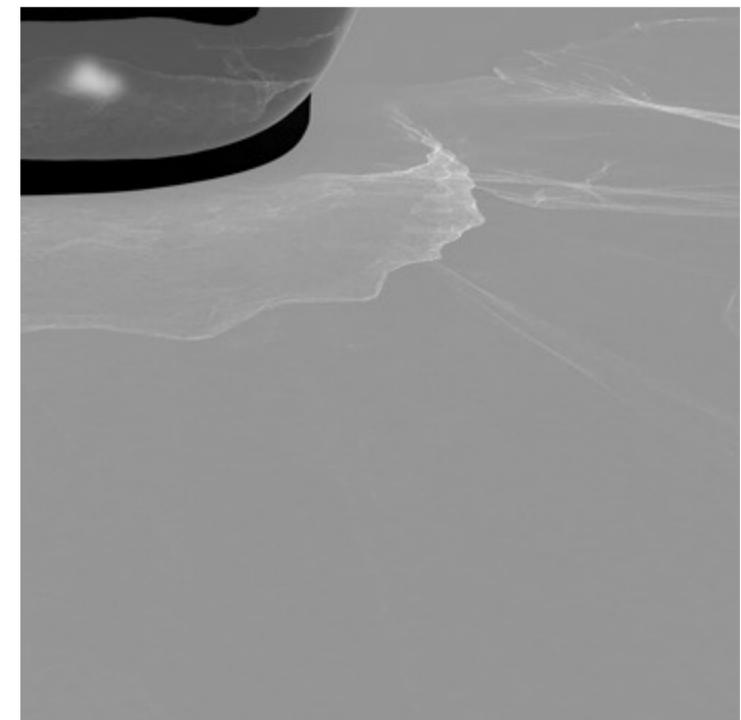
Catmull-Clark on triangle mesh



(huge number of irregular vertices!)



**jagged
reflection lines**



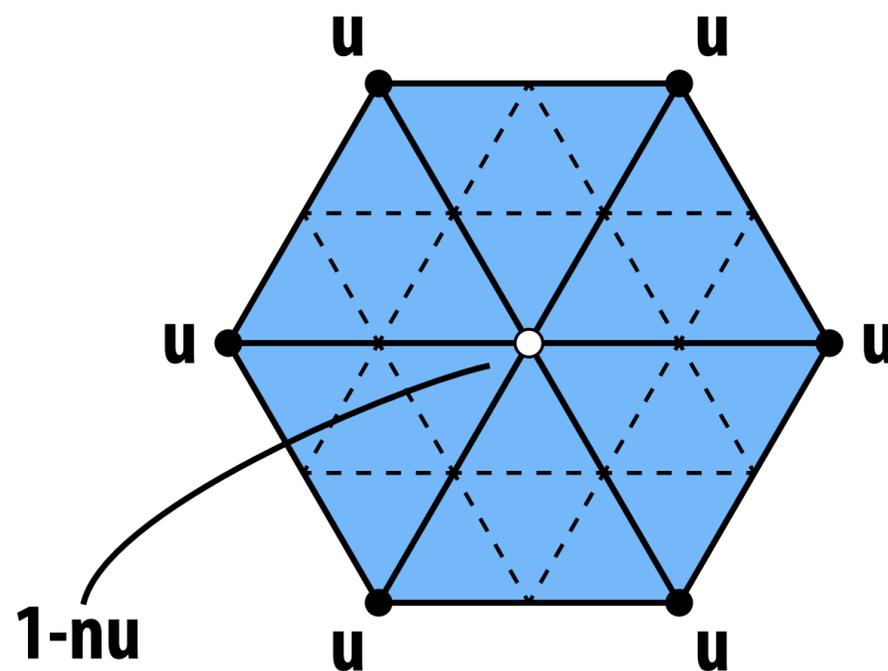
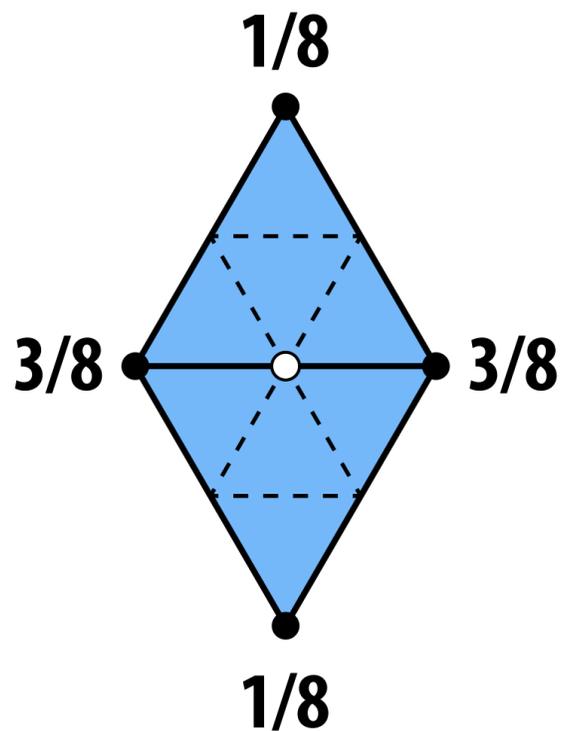
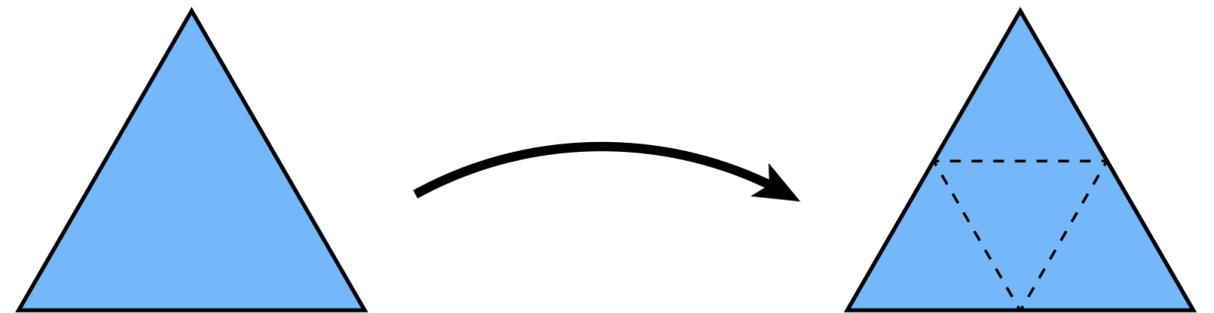
**jagged
caustics**

Loop Subdivision

- Alternative subdivision scheme for triangle meshes
- Curvature is continuous away from irregular vertices (" C^2 ")

- Algorithm:

- Split each triangle into four
- Assign new vertex positions according to weights:

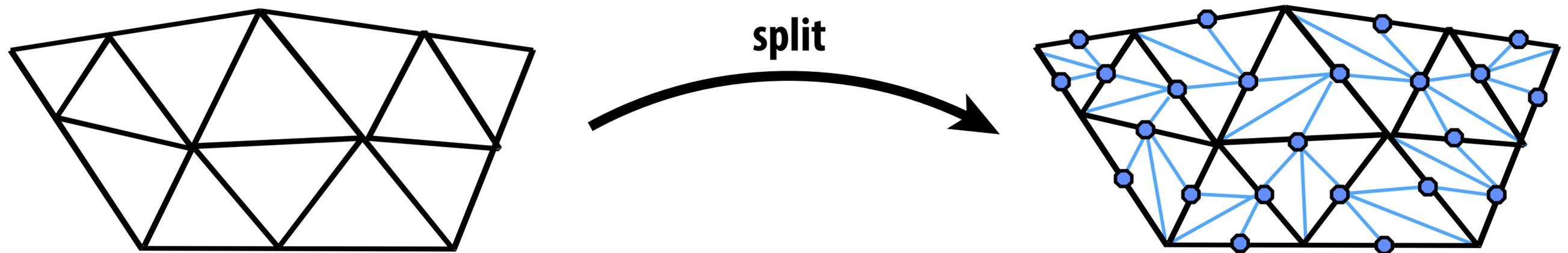


n : vertex degree

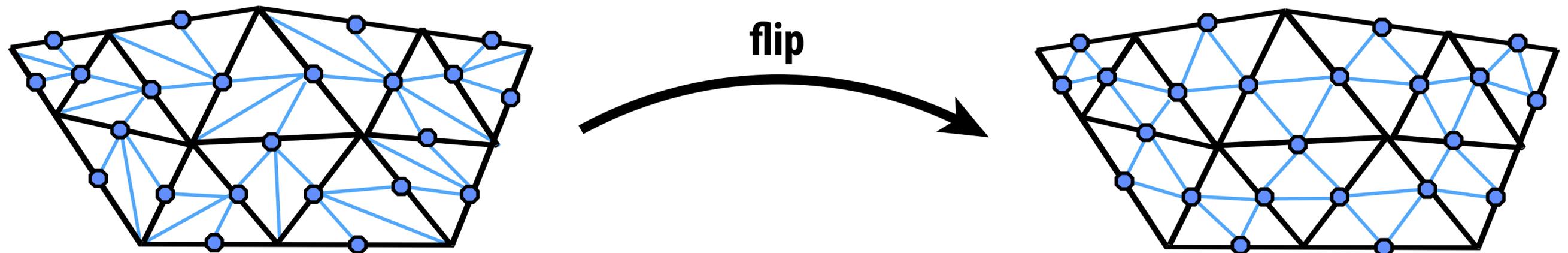
u : $3/16$ if $n=3$, $3/(8n)$ otherwise

Loop Subdivision via Edge Operations

- First, split edges of original mesh in *any* order:



- Next, flip new edges that touch a new & old vertex:



(Don't forget to update vertex positions!)

What if we want *fewer* triangles?

Simplification via Edge Collapse

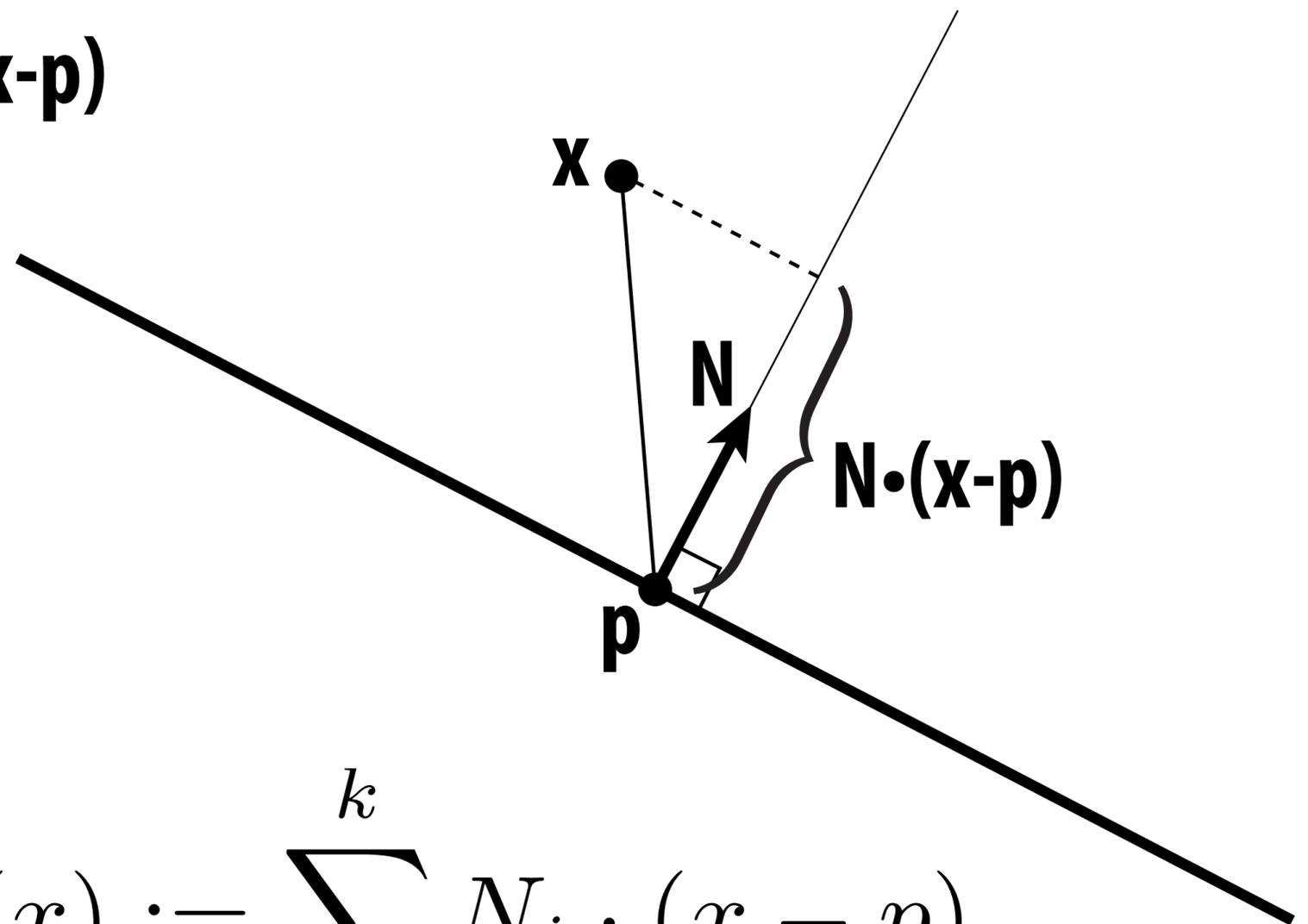
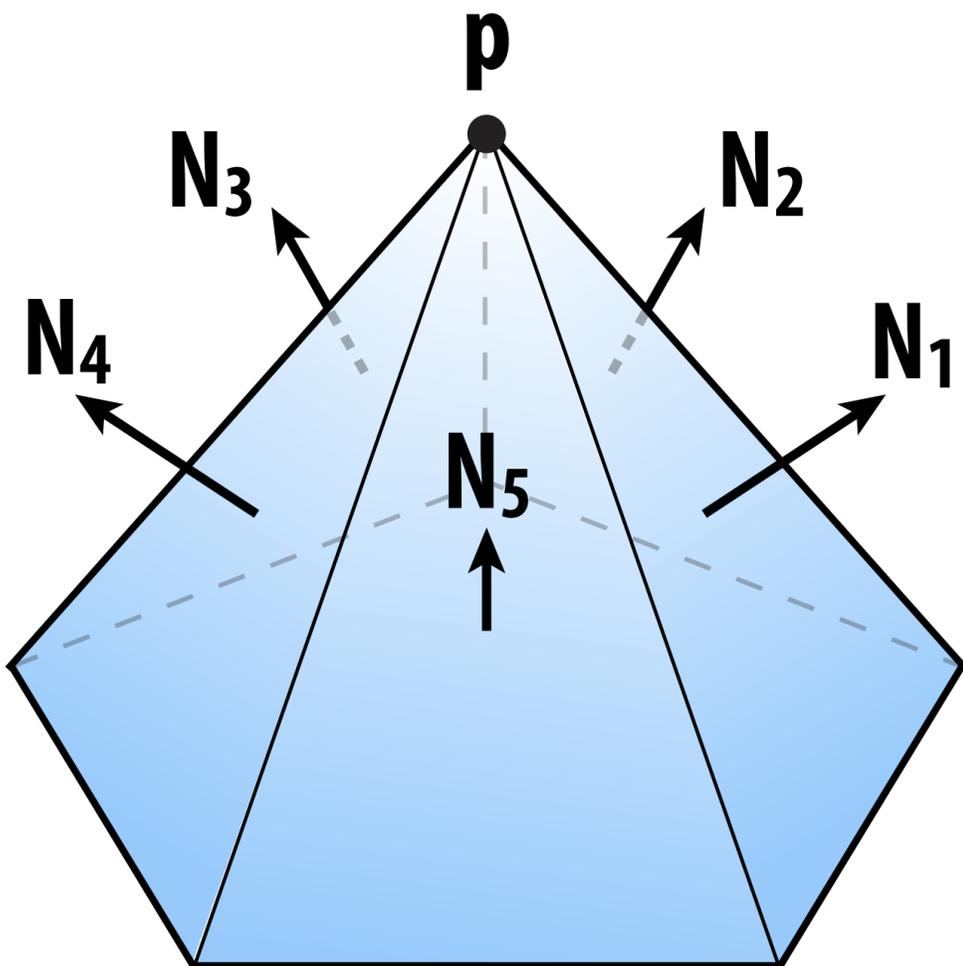
- One popular scheme: iteratively collapse edges
- Greedy algorithm:
 - assign each edge a cost
 - collapse edge with least cost
 - repeat until target number of elements is reached
- Particularly effective cost function: *quadric error metric**



*invented here at CMU! (Garland & Heckbert 1997)

Quadric Error Metric

- Approximate distance to a collection of triangles
- Distance is sum of point-to-plane distances
 - Q: Distance to plane w/ normal N passing through point p ?
 - A: $d(x) = N \cdot x - N \cdot p = N \cdot (x - p)$
- Sum of distances:



$$d(x) := \sum_{i=1}^k N_i \cdot (x - p)$$

Quadric Error - Homogeneous Coordinates

- Suppose in coordinates we have

- a query point (x,y,z)
- a normal (a,b,c)
- an offset $d := -(p,q,r) \cdot (a,b,c)$

$$K = \begin{bmatrix} a^2 & ab & ac & ad \\ ab & b^2 & bc & bd \\ ac & bc & c^2 & cd \\ ad & bd & cd & d^2 \end{bmatrix}$$

- Then in homogeneous coordinates, let

- $u := (x,y,z,1)$
- $v := (a,b,c,d)$

- *Signed* distance to plane is then just $u \cdot v = ax+by+cz+d$

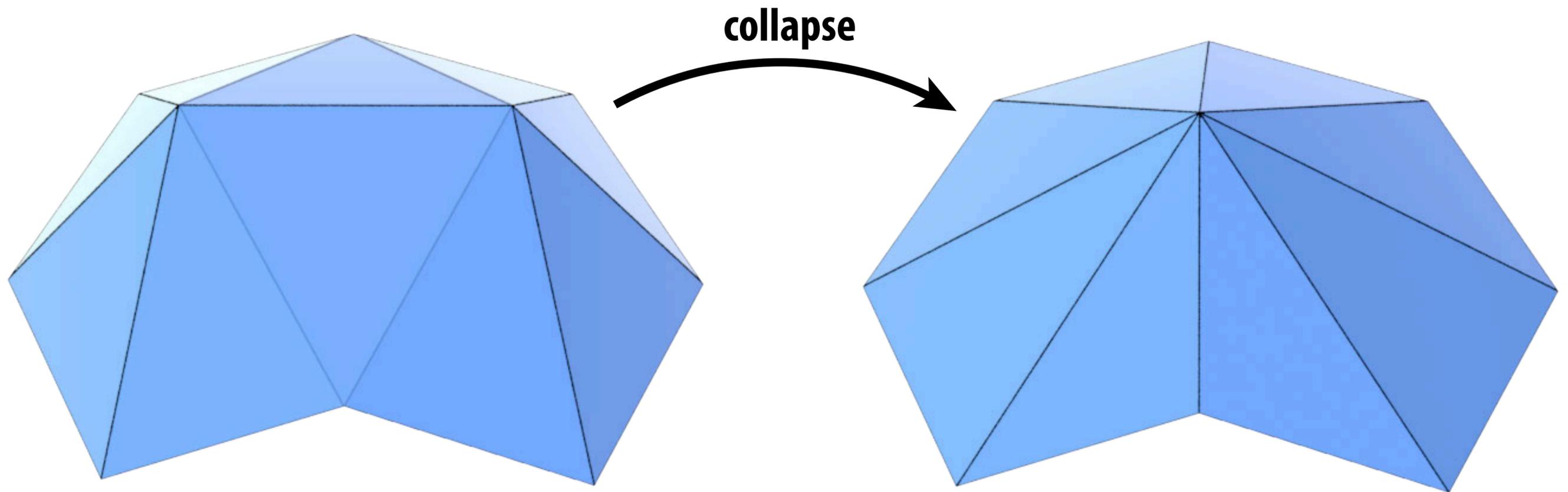
- *Squared* distance is $(u^T v)^2 = u^T (v v^T) u =: u^T K u$

- Key idea: *matrix K encodes distance to plane*

- K is symmetric, contains 10 unique coefficients (small storage)

Quadric Error of Edge Collapse

- How much does it cost to collapse an edge?
- Idea: compute edge midpoint, measure quadric error



- Better idea: use point that *minimizes quadric error* as new point!
- Q: Ok, but how do we minimize quadric error?

Review: Minimizing a Quadratic Function

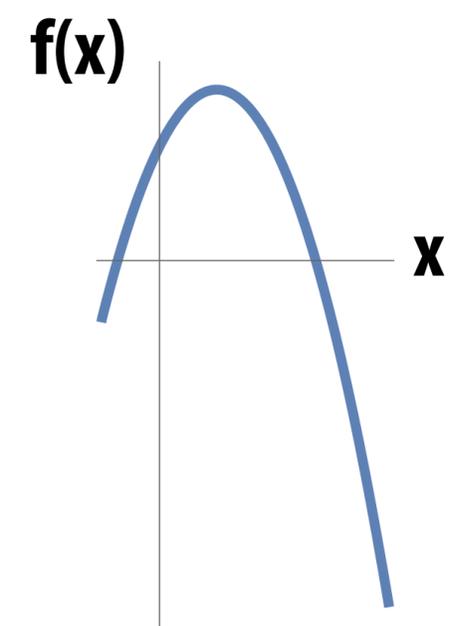
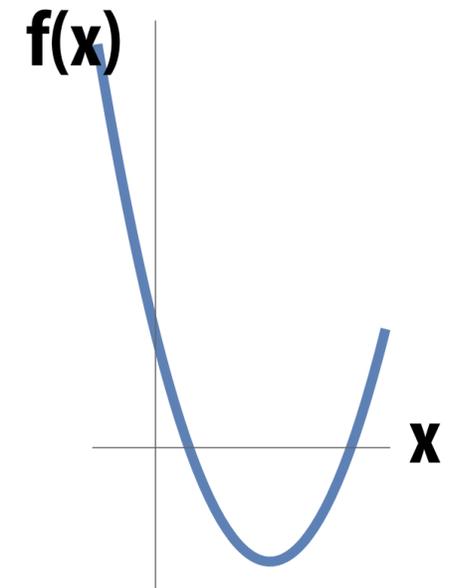
- Suppose I give you a function $f(x) = ax^2 + bx + c$
- Q: What does the graph of this function look like?
- Could also look like this!
- Q: How do we find the *minimum*?
- A: Look for the point where the function isn't changing (if we look "up close")
- I.e., find the point where the *derivative vanishes*

$$f'(x) = 0$$

$$2ax + b = 0$$

$$x = -b/2a$$

(What about our second example?)



Minimizing a Quadratic Form

- A *quadratic form* is just a generalization of our quadratic polynomial from 1D to nD
- E.g., in 2D: $f(x,y) = ax^2 + bxy + cy^2 + dx + ey + g$
- Can always (always!) write quadratic polynomial using a *symmetric* matrix (and a vector, and a constant):

$$f(x, y) = \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} a & b/2 \\ b/2 & c \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} d & e \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + g$$
$$= \mathbf{x}^T A \mathbf{x} + \mathbf{u}^T \mathbf{x} + g \quad \text{(this expression works for any } n\text{!)}$$

- Q: How do we find a critical point (min/max/saddle)?
- A: Set derivative to zero!

$$2A\mathbf{x} + \mathbf{u} = 0$$

$$\mathbf{x} = -\frac{1}{2}A^{-1}\mathbf{u}$$

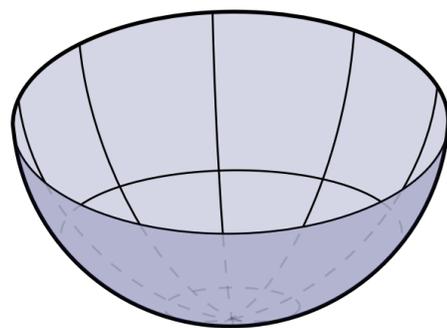
(Can you show this is true, at least in 2D?)

Positive Definite Quadratic Form

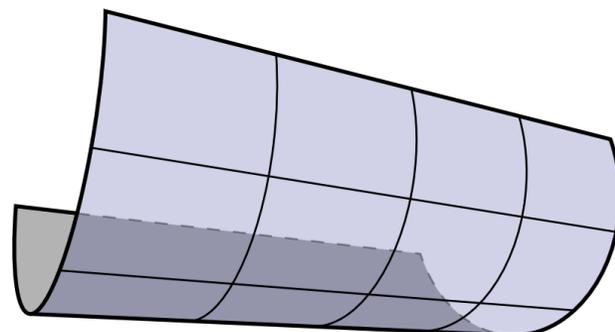
- Just like our 1D parabola, critical point is *not* always a min!
- Q: In 2D, 3D, nD, when do we get a *minimum*?
- A: When matrix A is *positive-definite*:

$$\mathbf{x}^T A \mathbf{x} > 0 \quad \forall \mathbf{x}$$

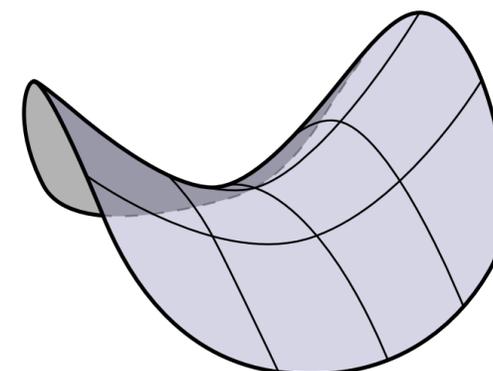
- 1D: Must have $xax = ax^2 > 0$. In other words: a is positive!
- 2D: Graph of function looks like a “bowl”:



positive definite



positive semidefinite



indefinite

- Positive-definiteness is ***extremely important*** in computer graphics: it means we can find a minimum by solving linear equations. Basis of many, many modern algorithms (geometry processing, simulation, ...).

Minimizing Quadratic Error

- Find “best” point for edge collapse by minimizing quad. form

$$\min_u \mathbf{u}^\top K \mathbf{u}$$

- Already know fourth (homogeneous) coordinate is 1!

- So, break up our quadratic function into two pieces:

$$\begin{bmatrix} \mathbf{x}^\top & 1 \end{bmatrix} \begin{bmatrix} B & \mathbf{w} \\ \mathbf{w} & d^2 \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}$$

$$= \mathbf{x}^\top B \mathbf{x} + 2\mathbf{w}^\top \mathbf{x} + d^2$$

- Now we have a quadratic form in the 3D position \mathbf{x} .

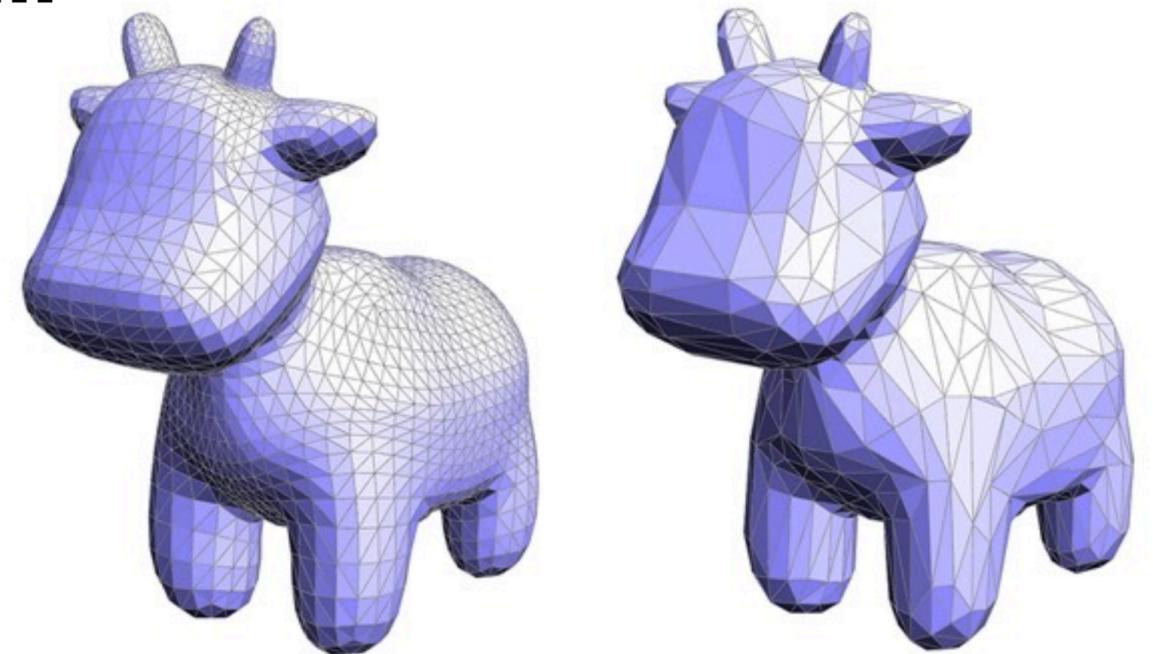
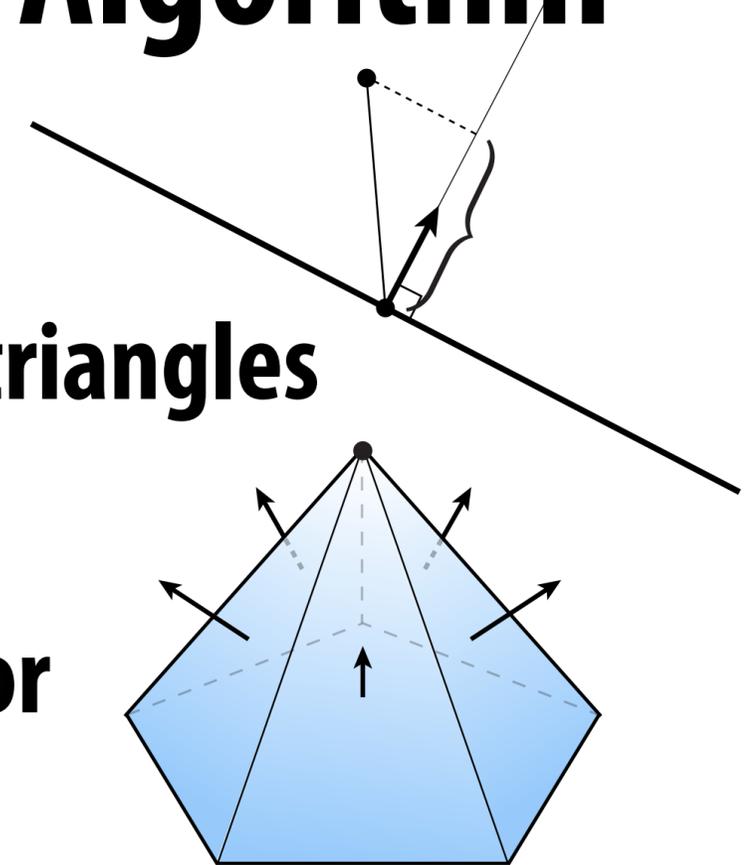
- Can minimize as before:

$$2B\mathbf{x} + 2\mathbf{w} = 0 \quad \iff \quad \mathbf{x} = -B^{-1}\mathbf{w}$$

(Q: Why should B be positive-definite?)

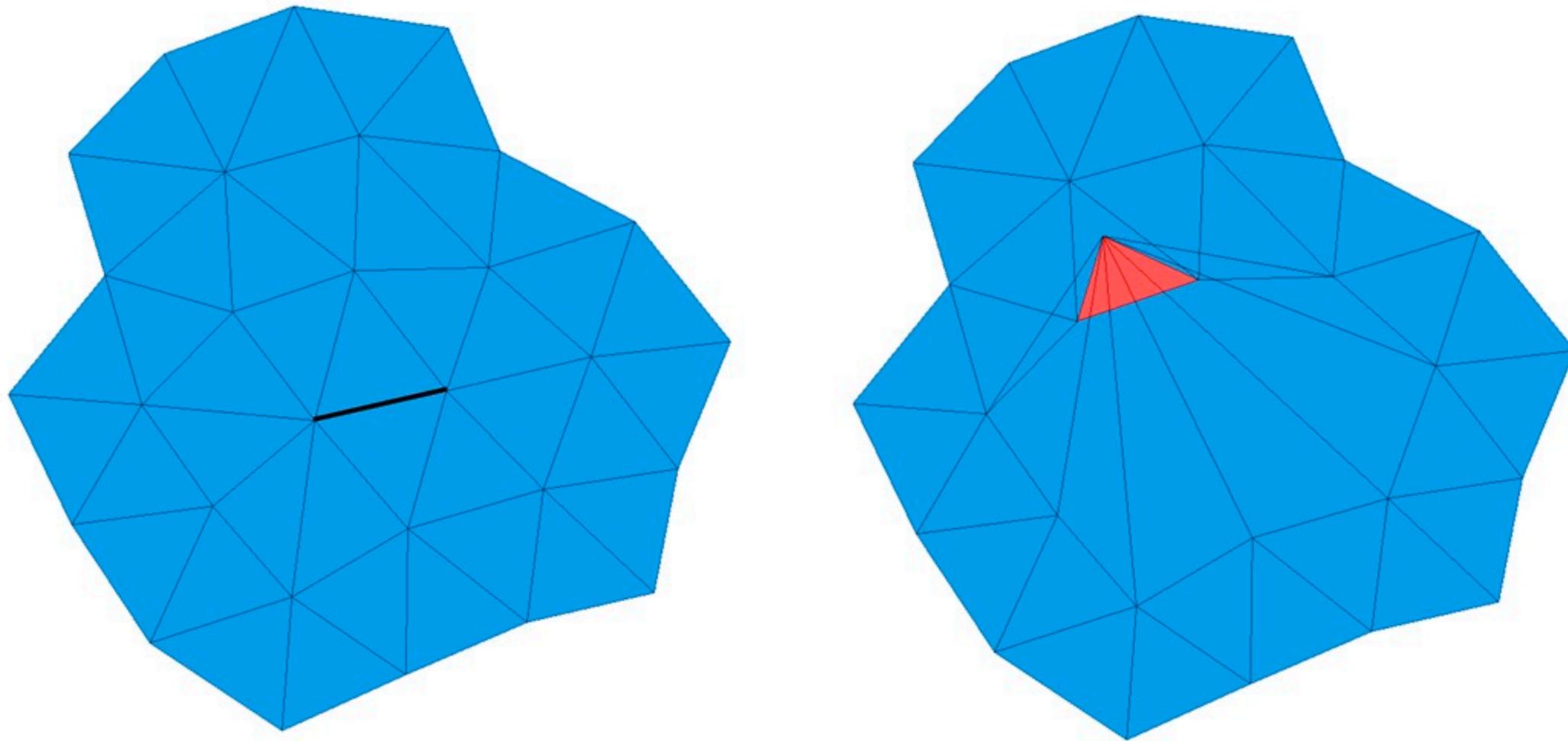
Quadric Error Simplification: Final Algorithm

- Compute K for each triangle (distance to plane)
- Set K at each vertex to sum of K s from incident triangles
- Set K at each edge to sum of K s at endpoints
- Find point at each edge minimizing quadric error
- Until we reach target # of triangles:
 - collapse edge (i,j) with smallest cost to get new vertex m
 - add K_i and K_j to get quadric K_m at m
 - update cost of edges touching m
- *More details in assignment writeup!*



Quadric Simplification—Flipped Triangles

- Depending on where we put the new vertex, one of the new triangles might be “flipped” (normal points in instead of out):

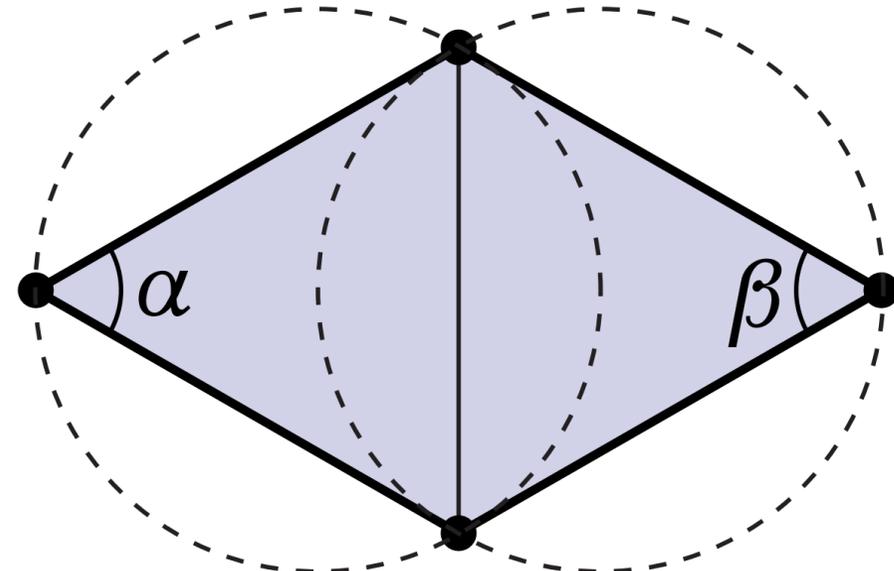
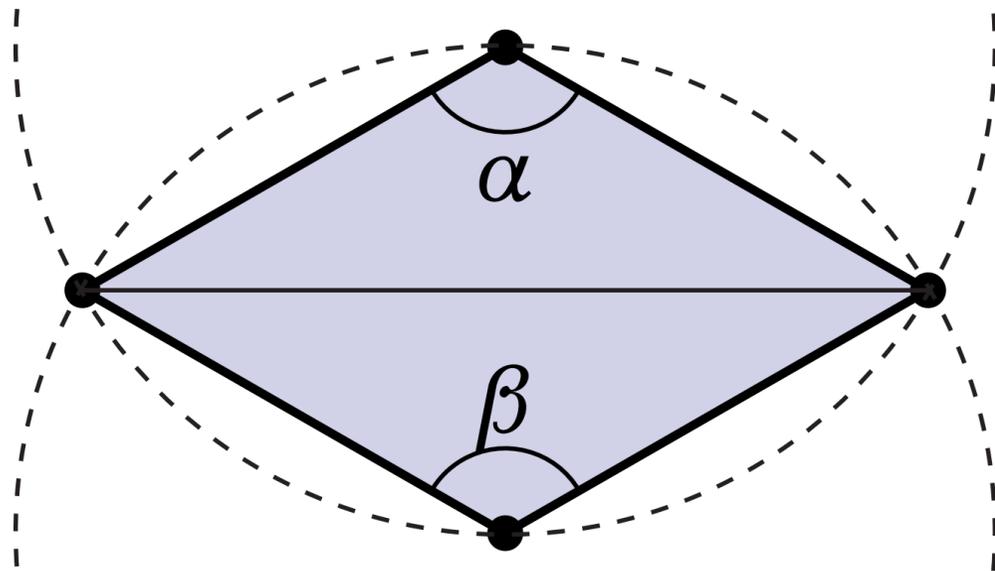


- Easy solution: check dot product between normals across edge
- If negative, don't collapse this edge!

What if we're happy with the *number* of triangles, but want to improve *quality*?

How do we make a mesh “more Delaunay”?

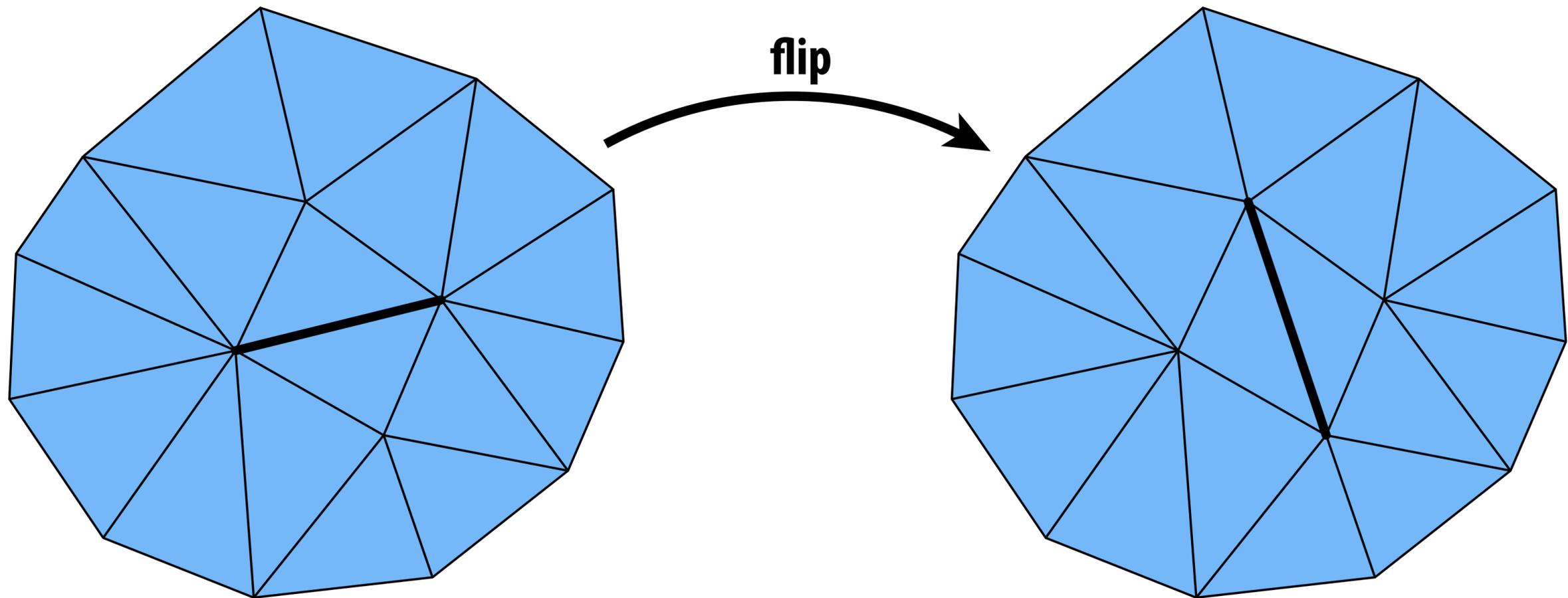
- Already have a good tool: edge flips!
- If $\alpha + \beta > \pi$, flip it!



- **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

Alternatively: how do we improve degree?

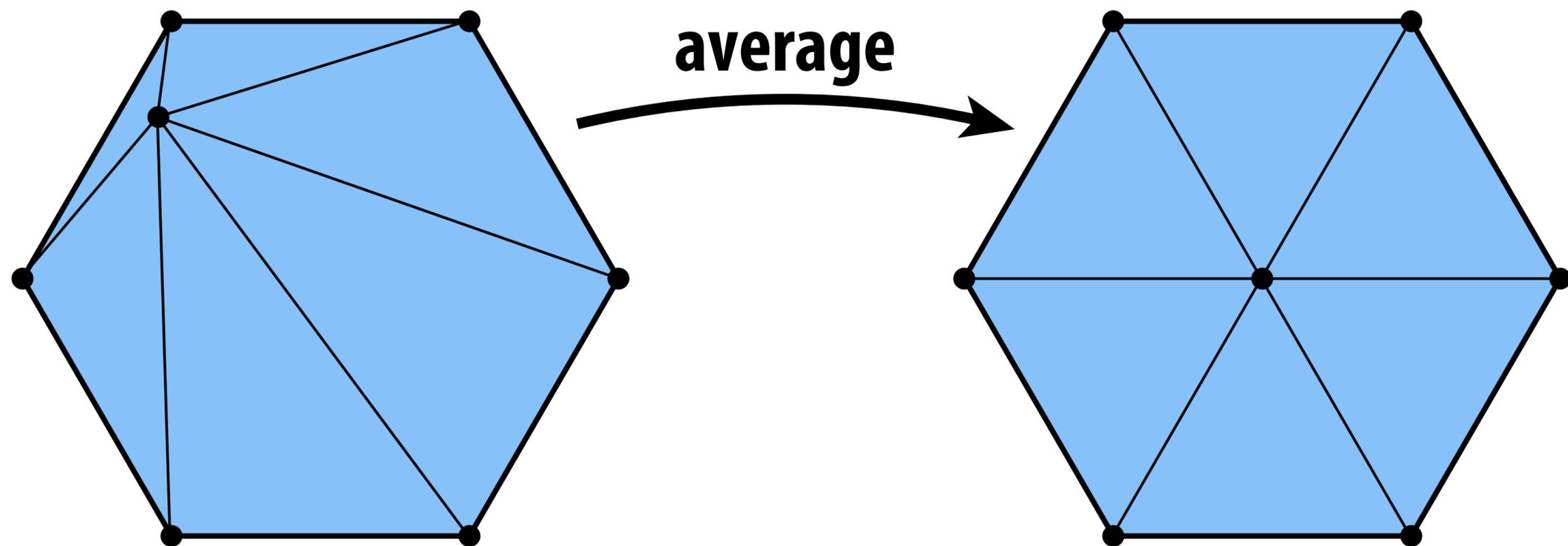
- Same tool: edge flips!
- If total deviation from degree-6 gets smaller, flip it!



- **FACT: average valence of any triangle mesh is 6**
- Iterative edge flipping acts like “discrete diffusion” of degree
- Again, no (known) guarantees; works well in practice

How do we make a triangles “more round”?

- Delaunay doesn't mean triangles are “round” (angles near 60°)
- Can often improve shape by centering vertices:

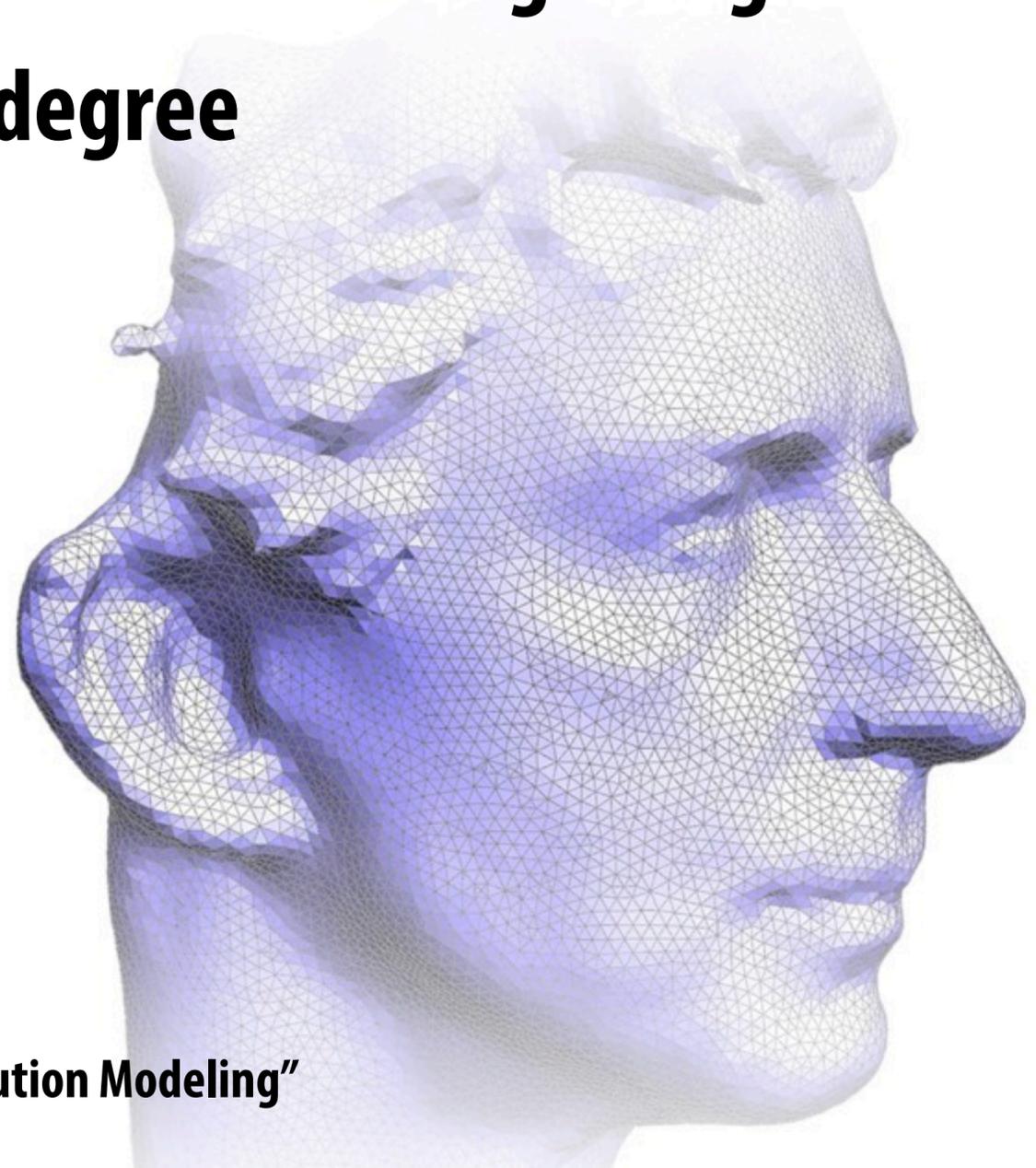
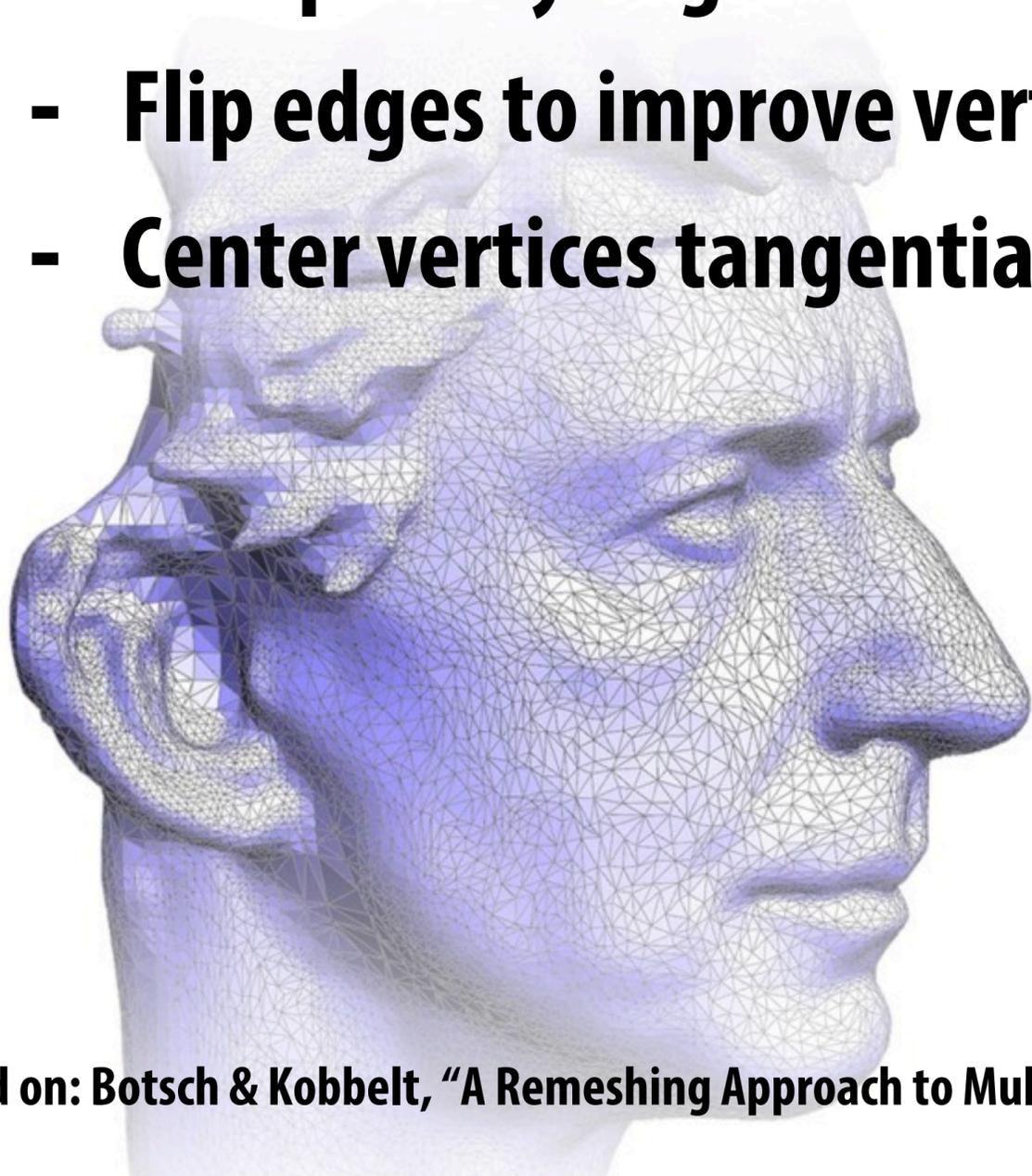


- Simple version of technique called “Laplacian smoothing”.*
- On surface: move only in *tangent* direction
- How? Remove normal component from update vector.

*See Crane, “Digital Geometry Processing with Discrete Exterior Calculus” <http://keenan.is/ddg>

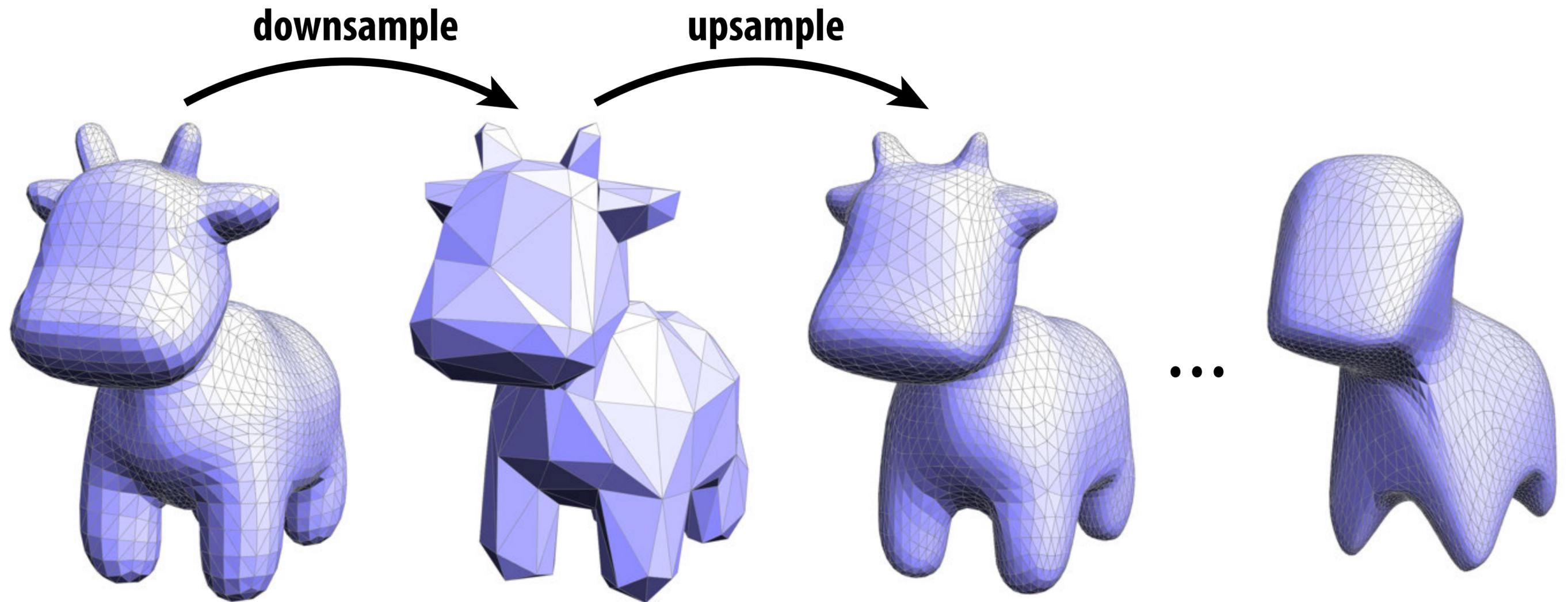
Isotropic Remeshing Algorithm

- Try to make triangles uniform shape & size
- Repeat four steps:
 - Split any edge over $\frac{4}{3}$ ths mean edge length
 - Collapse any edge less than $\frac{4}{5}$ ths mean edge length
 - Flip edges to improve vertex degree
 - Center vertices tangentially



**What can go wrong when
you resample a signal?**

Danger of Resampling

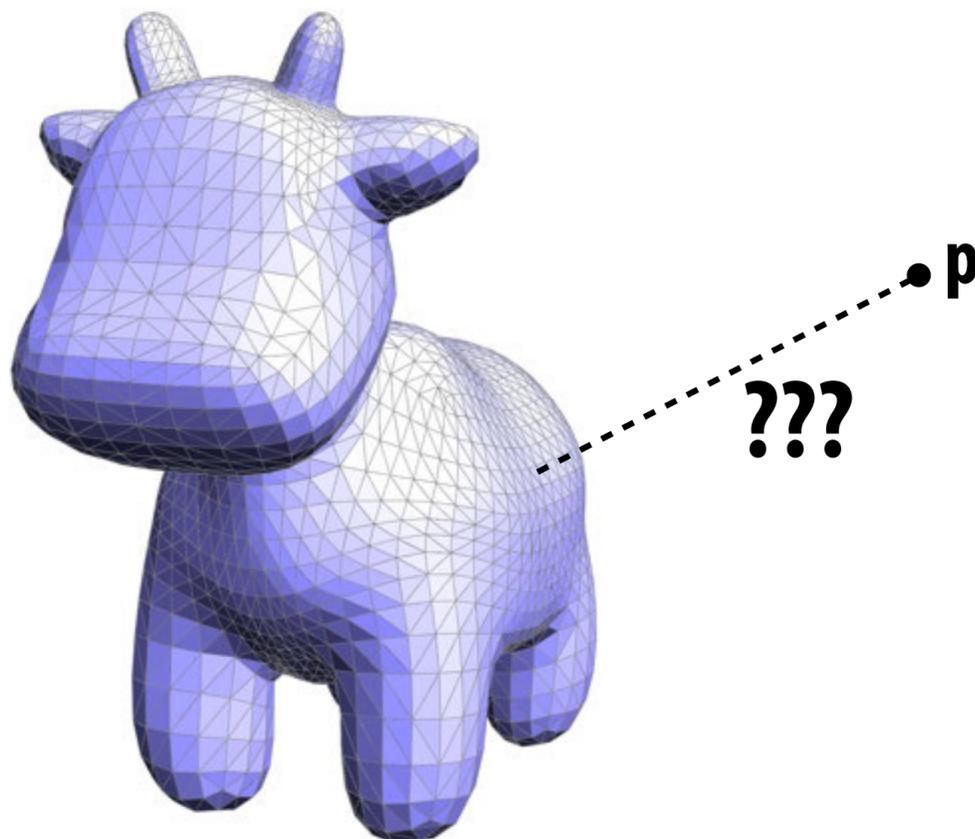


(Q: What happens with an image?)

**But wait: we have the original mesh.
Why not just project each new sample point
onto the closest point of the original mesh?**

Geometric Queries

- **Q: Given a point, in space (e.g., a new sample point), how do we find the closest point on a given surface?**
- **Q: Does implicit/explicit representation make this easier?**
- **Q: Does our halfedge data structure help?**
- **Q: What's the cost of the naïve algorithm?**
- **Q: How do we find the distance to a single triangle anyway?**
- **So many questions!**

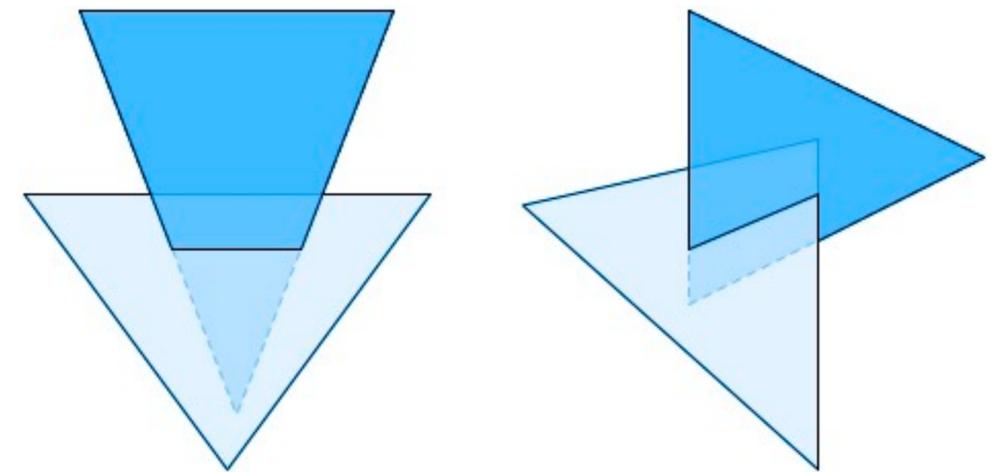
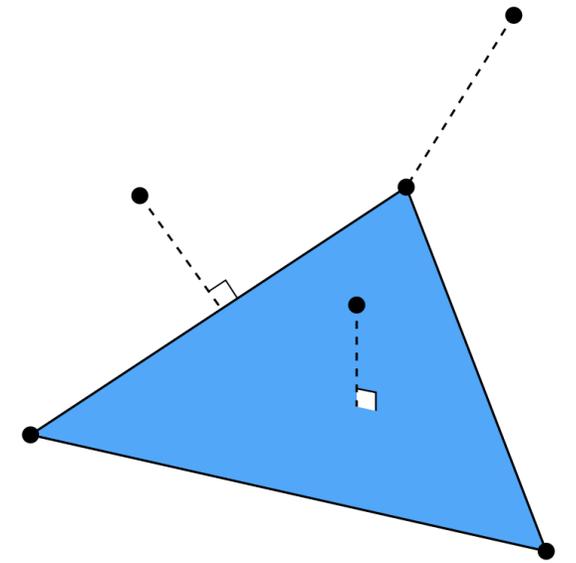


Many types of geometric queries

- Already identified need for “closest point” query

- Plenty of other things we might like to know:

- Do two triangles intersect?
- Are we inside or outside an object?
- Does one object contain another?
- ...



- Data structures we've seen so far not really designed for this...

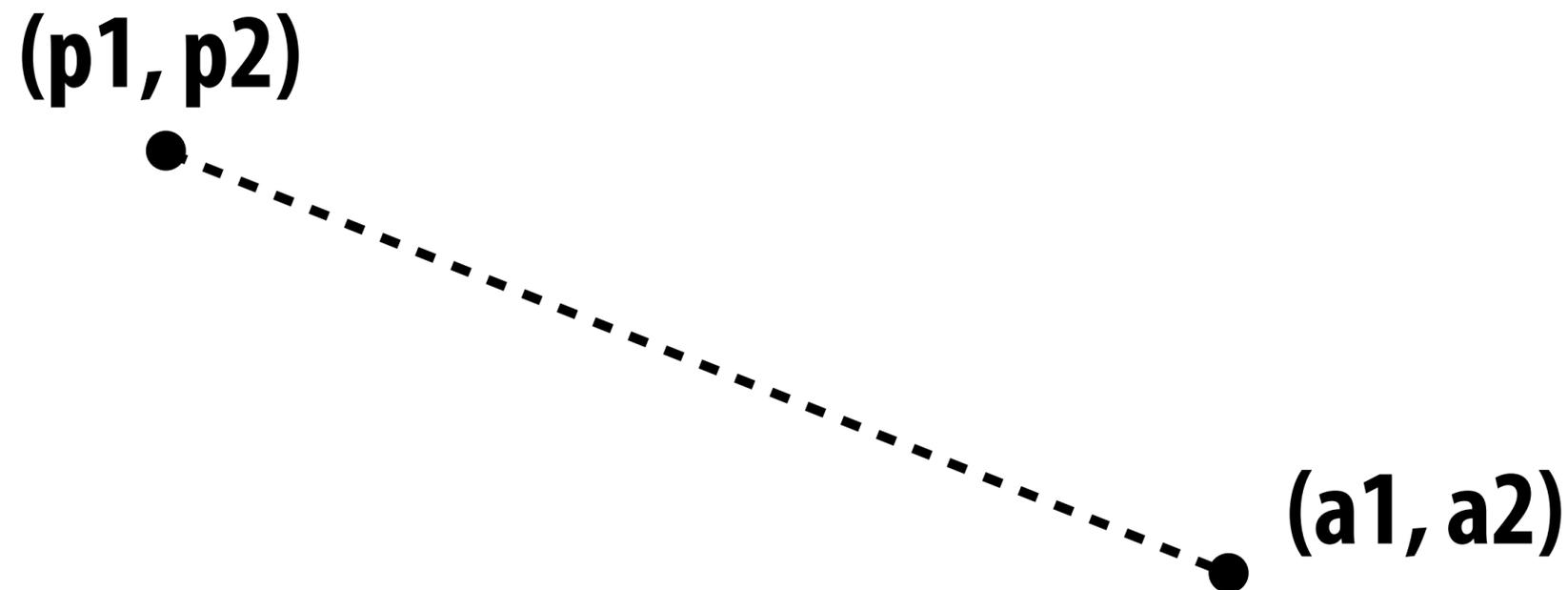
- Need some new ideas!

- TODAY: come up with simple (read: slow) algorithms.

- NEXT TIME: intelligent ways to accelerate geometric queries.

Warm up: closest point on point

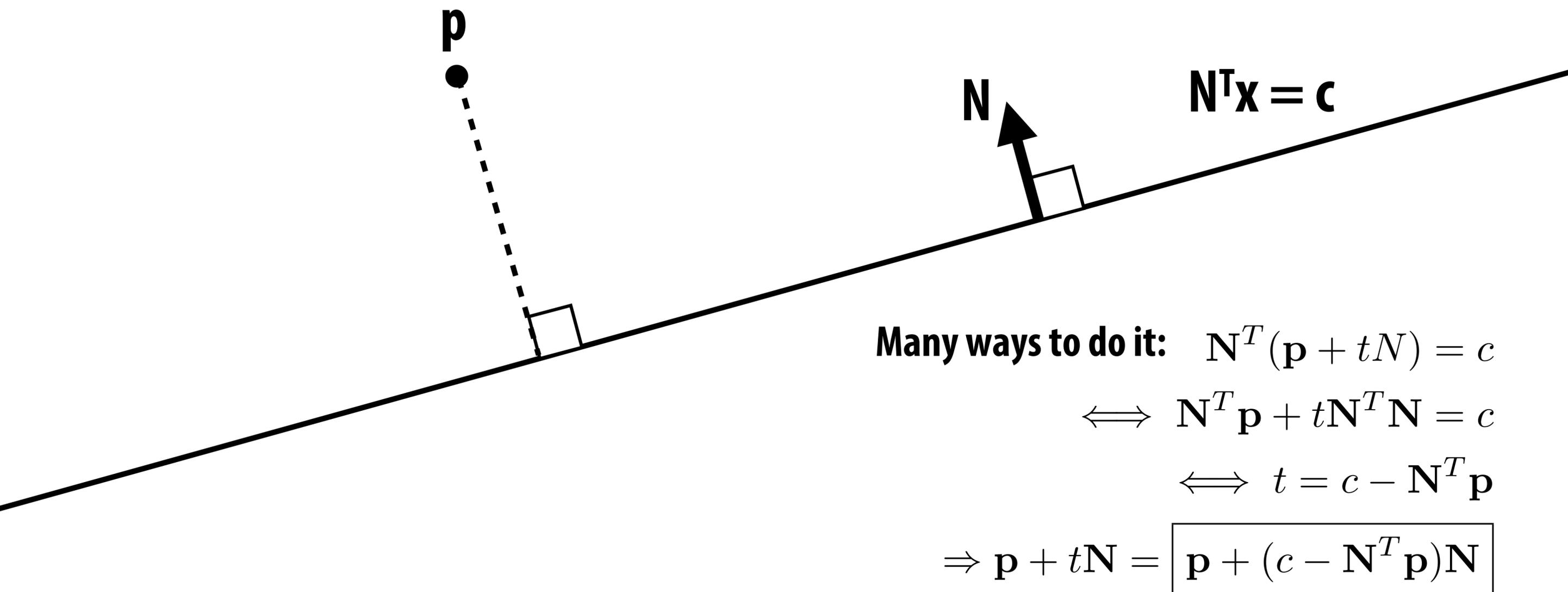
- Goal is to find the point on a mesh closest to a given point.
- *Much* simpler question: given a query point (p_1, p_2) , how do we find the closest point on the point (a_1, a_2) ?



Bonus question: what's the distance?

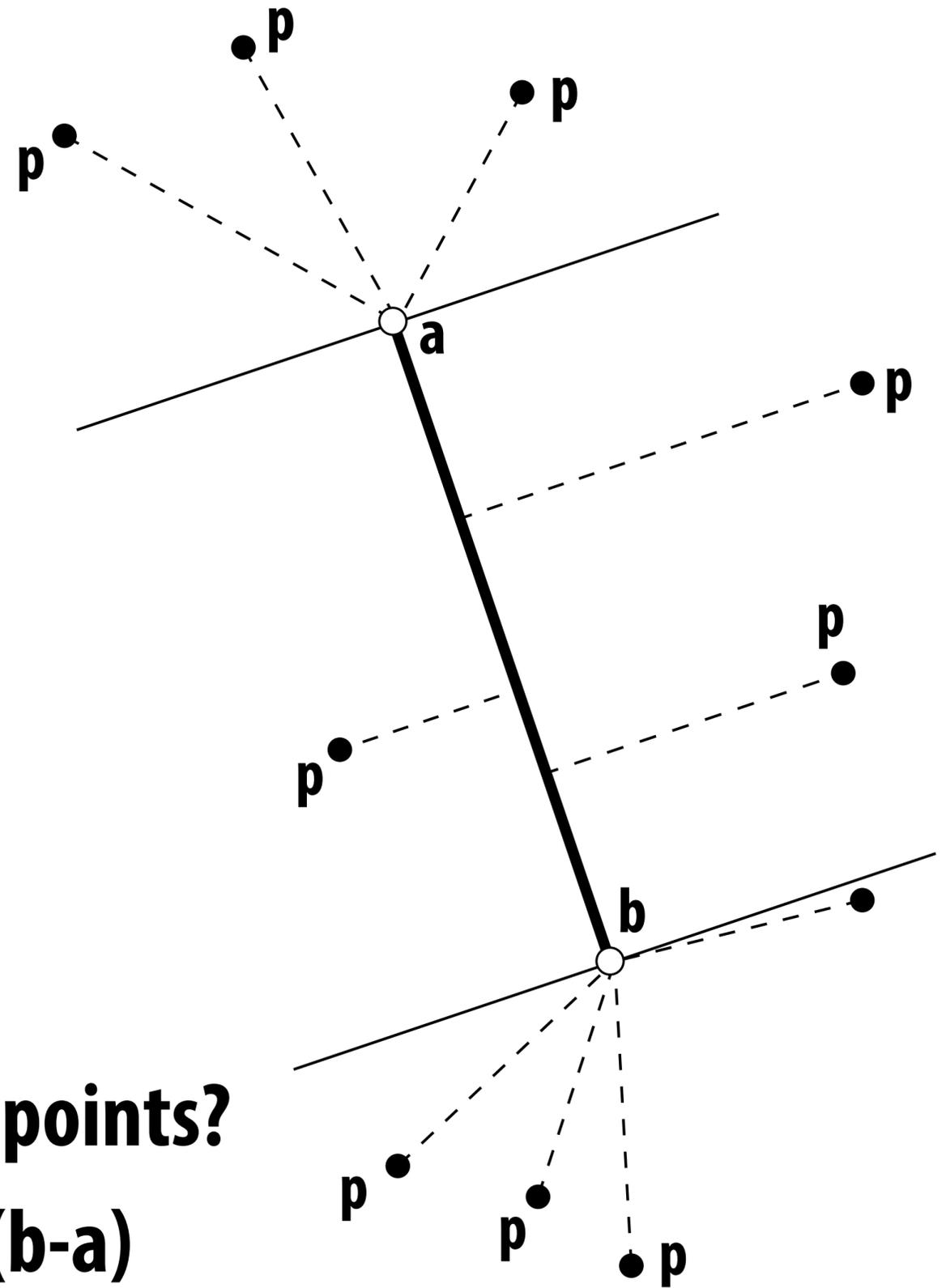
Slightly harder: closest point on line

- Now suppose I have a line $\mathbf{N}^T \mathbf{x} = c$, where \mathbf{N} is the unit normal
- How do I find the point closest to my query point \mathbf{p} ?



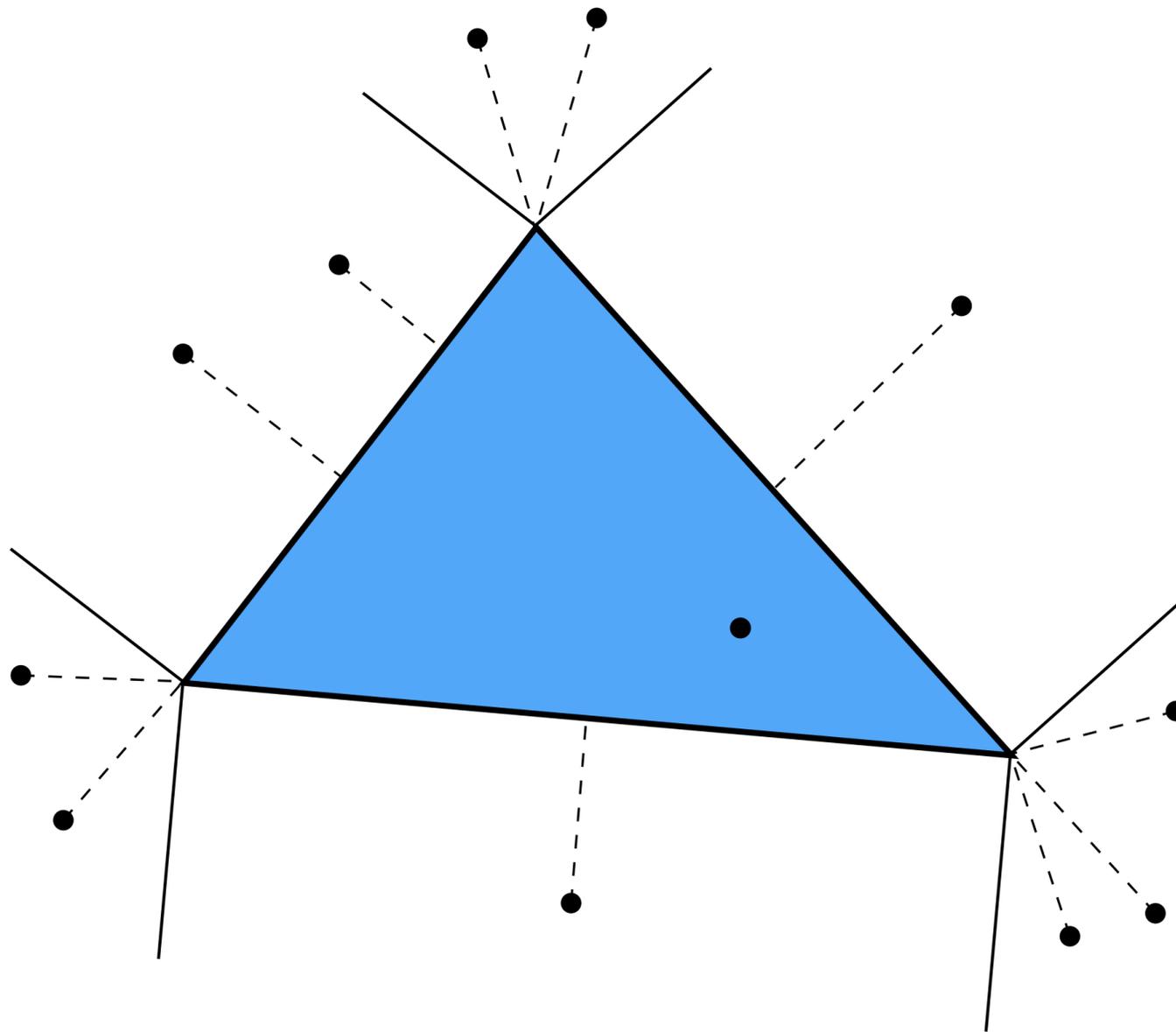
Harder: closest point on line segment

- Two cases: endpoint or interior
- Already have basic components:
 - point-to-point
 - point-to-line
- Algorithm?
 - find closest point on line
 - check if it's between endpoints
 - if not, take closest endpoint
- How do we know if it's between endpoints?
 - write closest point on line as $a+t(b-a)$
 - if t is between 0 and 1, it's inside the segment!



Even harder: closest point on triangle

- What are all the possibilities for the closest point?
- Almost just minimum distance to three segments:



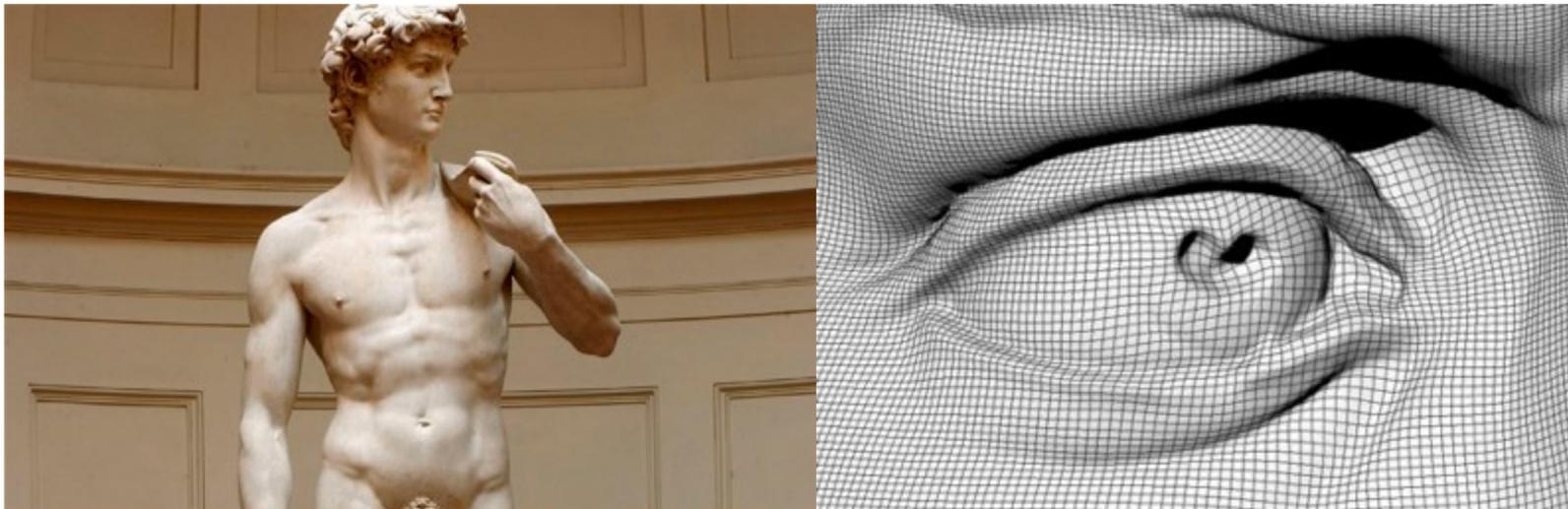
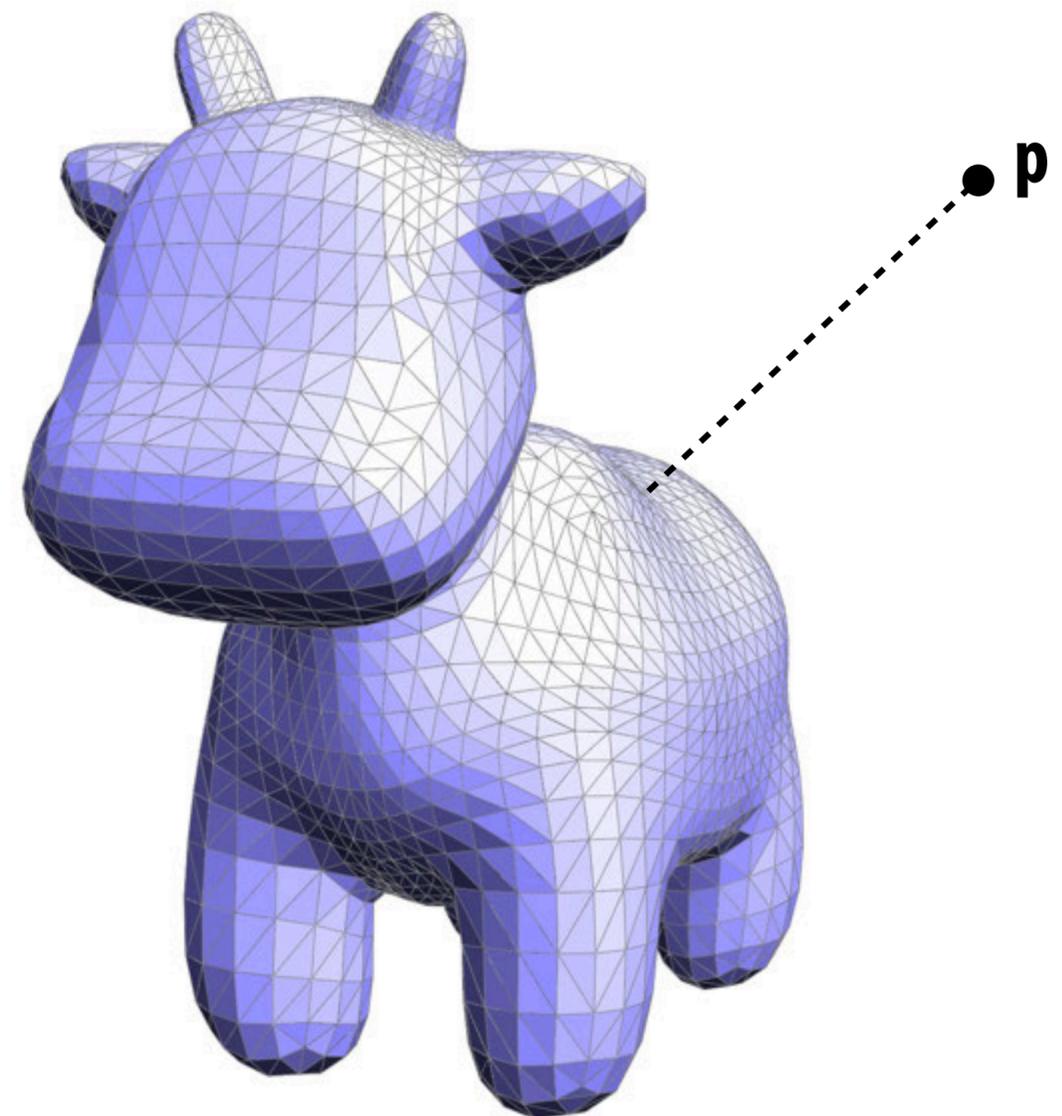
Q: What about a point inside the triangle?

Closest point on triangle in 3D

- Not so different from 2D case
- Algorithm?
 - project onto plane of triangle
 - use *half-space* tests to classify point (vs. half plane)
 - if inside the triangle, we're done!
 - otherwise, find closest point on associated vertex or edge
- By the way, how do we find closest point on plane?
- Same expression as closest point on a line!
- E.g., $p + (c - N^T p) N$

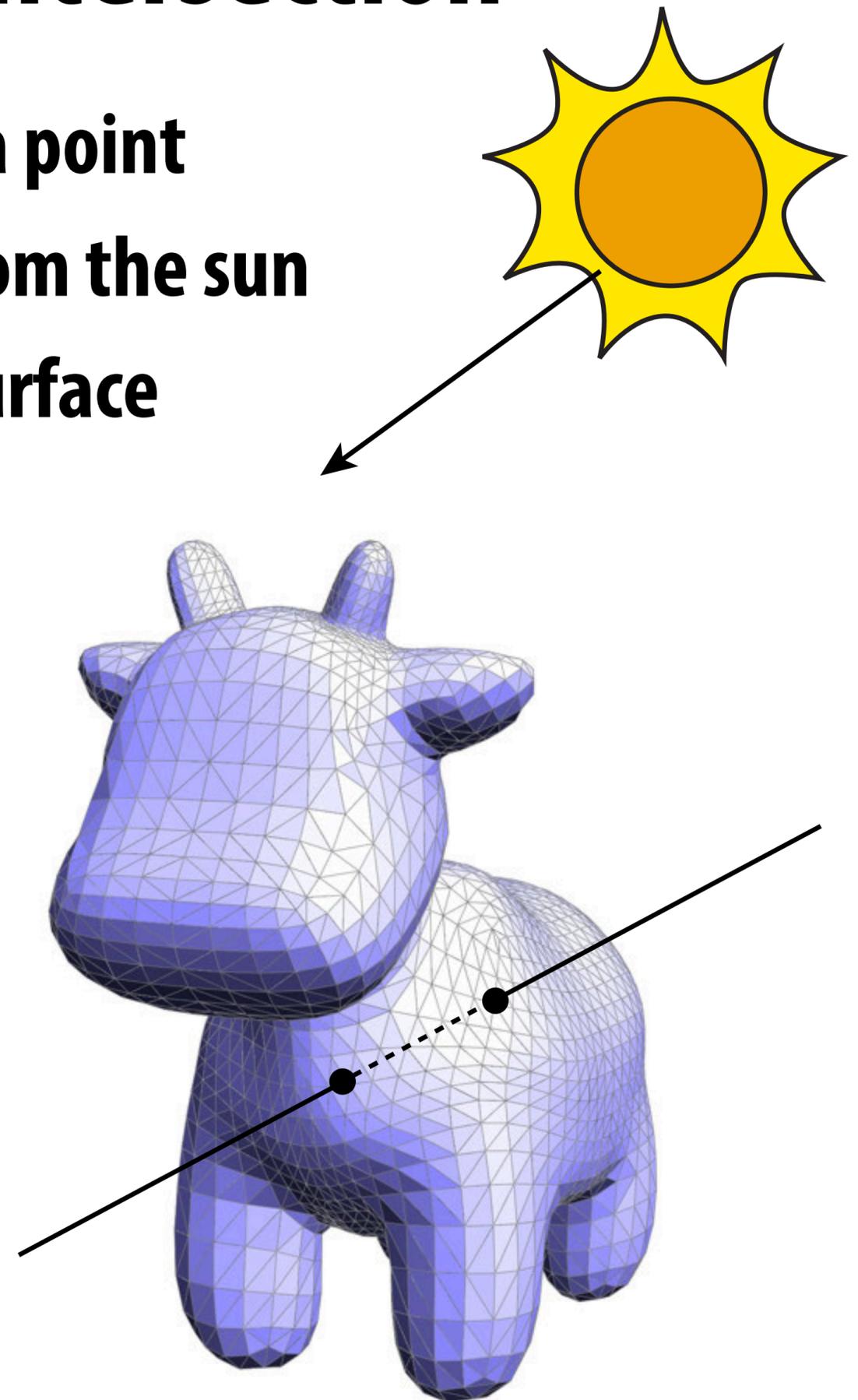
Closest point on triangle *mesh* in 3D?

- **Conceptually easy:**
 - loop over all triangles
 - compute closest point to current triangle
 - keep globally closest point
- **Q: What's the cost?**
- **What if we have *billions* of faces?**
- **NEXT TIME: Better data structures!**



Different query: ray-mesh intersection

- A “ray” is an oriented line starting at a point
- Think about a ray of light traveling from the sun
- Want to know where a ray pierces a surface
- Why?
 - **GEOMETRY: inside-outside test**
 - **RENDERING: visibility, ray tracing**
 - **SIMULATION: collision detection**
- **Might pierce surface in many places!**



Ray equation

- Can express ray as

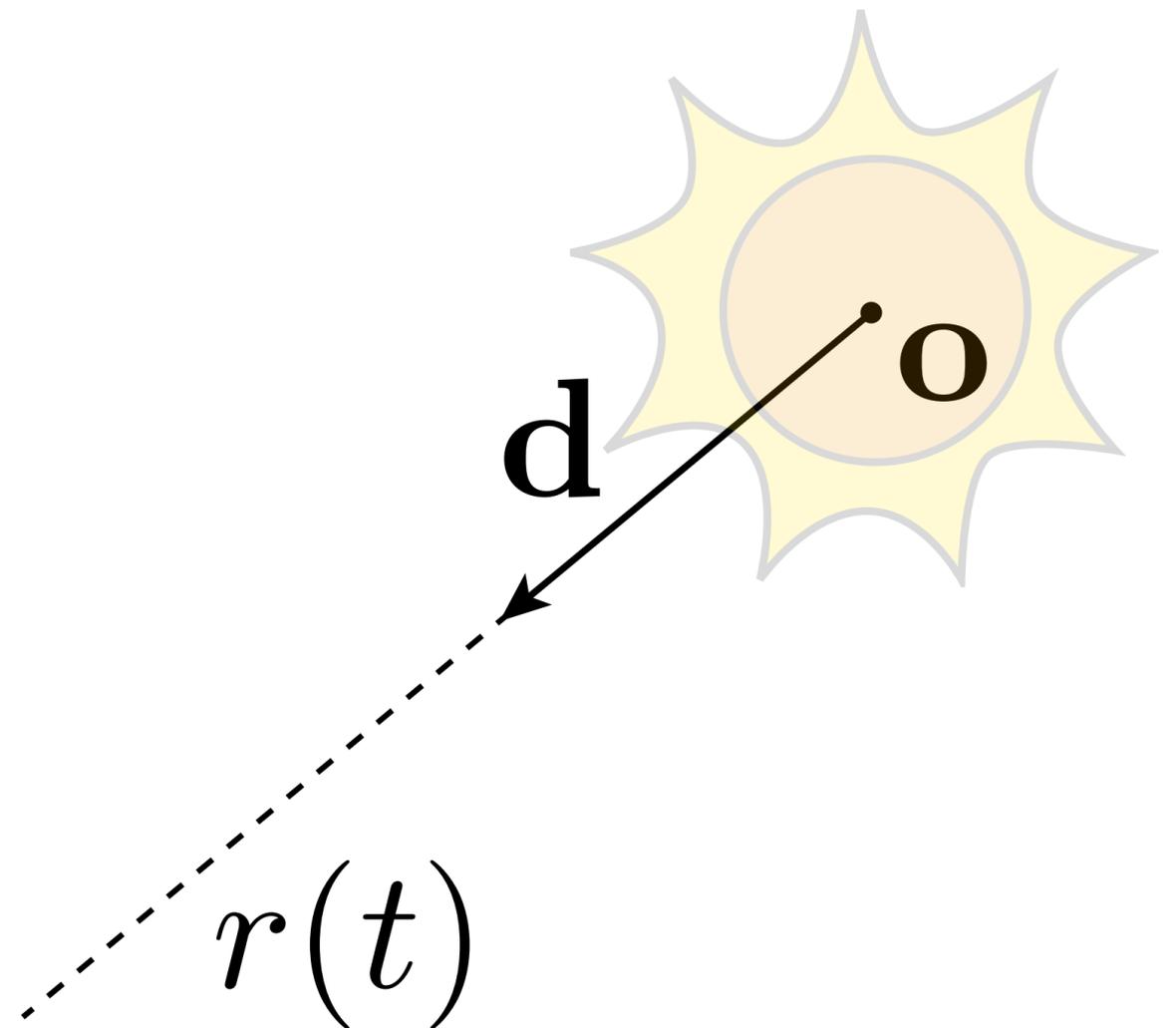
$$\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$$

point along ray

origin

unit direction

"time"



Intersecting a ray with an implicit surface

- Recall implicit surfaces: all points \mathbf{x} such that $f(\mathbf{x}) = 0$
- Q: How do we find points where a ray pierces this surface?
- Well, we know all points along the ray: $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$
- Idea: replace “ \mathbf{x} ” with “ \mathbf{r} ” in 1st equation, and solve for t
- Example: unit sphere

$$f(\mathbf{x}) = |\mathbf{x}|^2 - 1$$

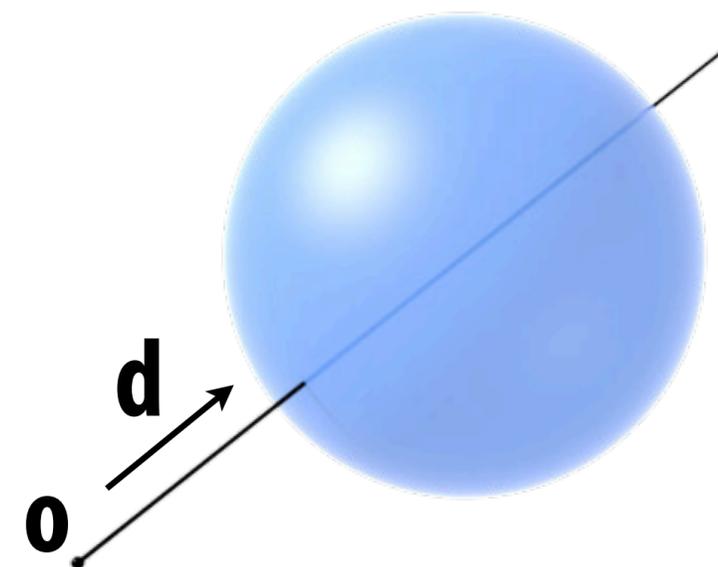
$$\Rightarrow f(\mathbf{r}(t)) = |\mathbf{o} + t\mathbf{d}|^2 - 1$$

$$\underbrace{|\mathbf{d}|^2}_{a} t^2 + \underbrace{2(\mathbf{o} \cdot \mathbf{d})}_{b} t + \underbrace{|\mathbf{o}|^2 - 1}_{c} = 0$$

$$t = \boxed{-\mathbf{o} \cdot \mathbf{d} \pm \sqrt{(\mathbf{o} \cdot \mathbf{d})^2 - |\mathbf{o}|^2 + 1}}$$

quadratic formula:

$$t = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

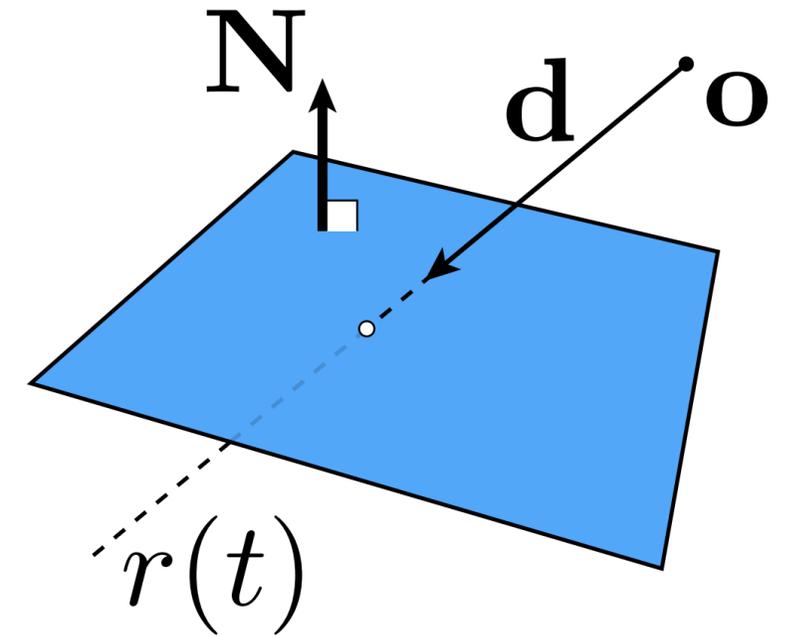


Why two solutions?

Ray-plane intersection

- Suppose we have a plane $\mathbf{N}^T \mathbf{x} = c$

- \mathbf{N} - unit normal
- c - offset



- How do we find intersection with ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$?

- *Key idea:* again, replace the point \mathbf{x} with the ray equation t :

$$\mathbf{N}^T \mathbf{r}(t) = c$$

- Now solve for t :

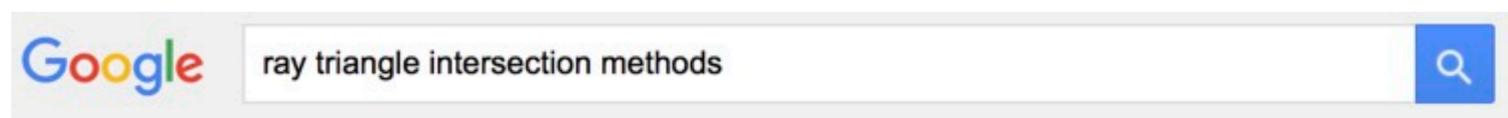
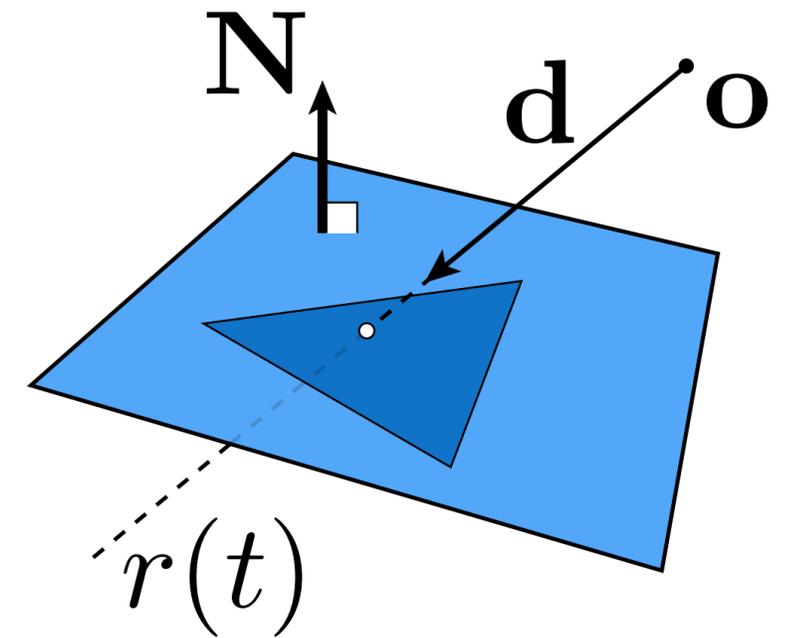
$$\mathbf{N}^T (\mathbf{o} + t\mathbf{d}) = c \quad \Rightarrow \quad t = \frac{c - \mathbf{N}^T \mathbf{o}}{\mathbf{N}^T \mathbf{d}}$$

- And plug t back into ray equation:

$$\mathbf{r}(t) = \mathbf{o} + \frac{c - \mathbf{N}^T \mathbf{o}}{\mathbf{N}^T \mathbf{d}} \mathbf{d}$$

Ray-triangle intersection

- Triangle is in a plane...
- Not much more to say!
 - Compute ray-plane intersection
 - Q: What do we do now?
 - A: Why not compute barycentric coordinates of hit point?
 - If barycentric coordinates are all positive, point in triangle
- Actually, a *lot* more to say... if you care about performance!



Web Shopping Videos News Images More Search tools

About 443,000 results (0.44 seconds)

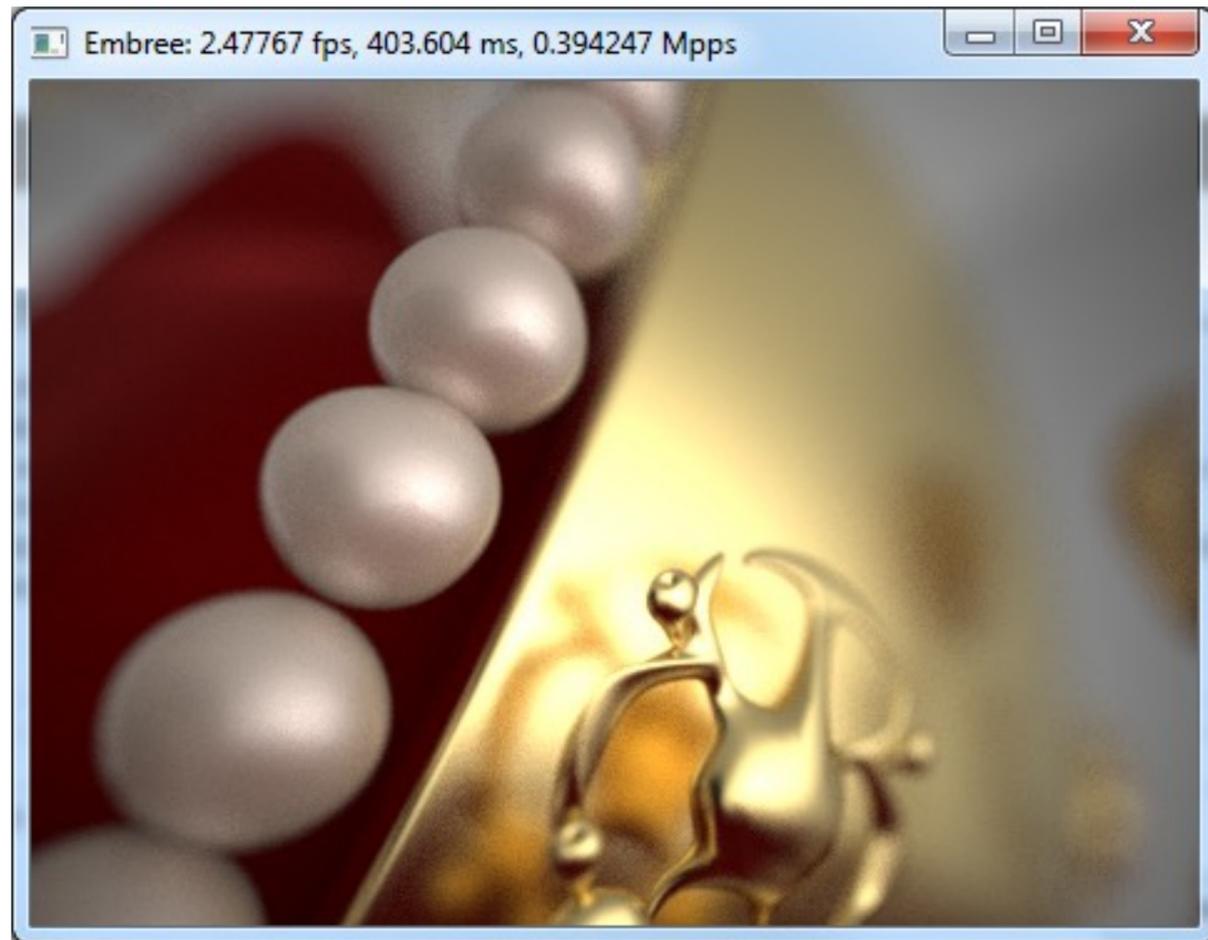
[Möller–Trumbore intersection algorithm - Wikipedia, the free ...](https://en.wikipedia.org/.../Möller-Trumbore_intersection_alg...)
https://en.wikipedia.org/.../Möller-Trumbore_intersection_alg... Wikipedia
The Möller–Trumbore ray-triangle intersection algorithm, named after its inventors Tomas Möller and Ben Trumbore, is a fast method for calculating the ...

[\[PDF\] Fast Minimum Storage Ray-Triangle Intersection.pdf](https://www.cs.virginia.edu/.../Fast%20MinimumSt...)
<https://www.cs.virginia.edu/.../Fast%20MinimumSt...> University of Virginia
by PC AB - Cited by 650 - Related articles
We present a clean algorithm for determining whether a ray intersects a triangle. ... ble in speed to previous methods, we believe it is the fastest ray/triangle.

[\[PDF\] Optimizing Ray-Triangle Intersection via Automated Search](http://www.cs.utah.edu/~aek/research/triangle.pdf)
www.cs.utah.edu/~aek/research/triangle.pdf University of Utah
by A Kensler - Cited by 33 - Related articles
method is used to further optimize the code produced via the fitness function. ... For these 3D methods we optimize ray-triangle intersection in two different ways.

[\[PDF\] Comparative Study of Ray-Triangle Intersection Algorithms](http://www.graphicon.ru/html/proceedings/2012/.../gc2012Shumskiy.pdf)
www.graphicon.ru/html/proceedings/2012/.../gc2012Shumskiy.pdf
by V Shumskiy - Cited by 1 - Related articles
optimized SIMD ray-triangle intersection method evaluated on. GPU for path- tracing

Why care about performance?



Intel Embree



NVIDIA OptiX

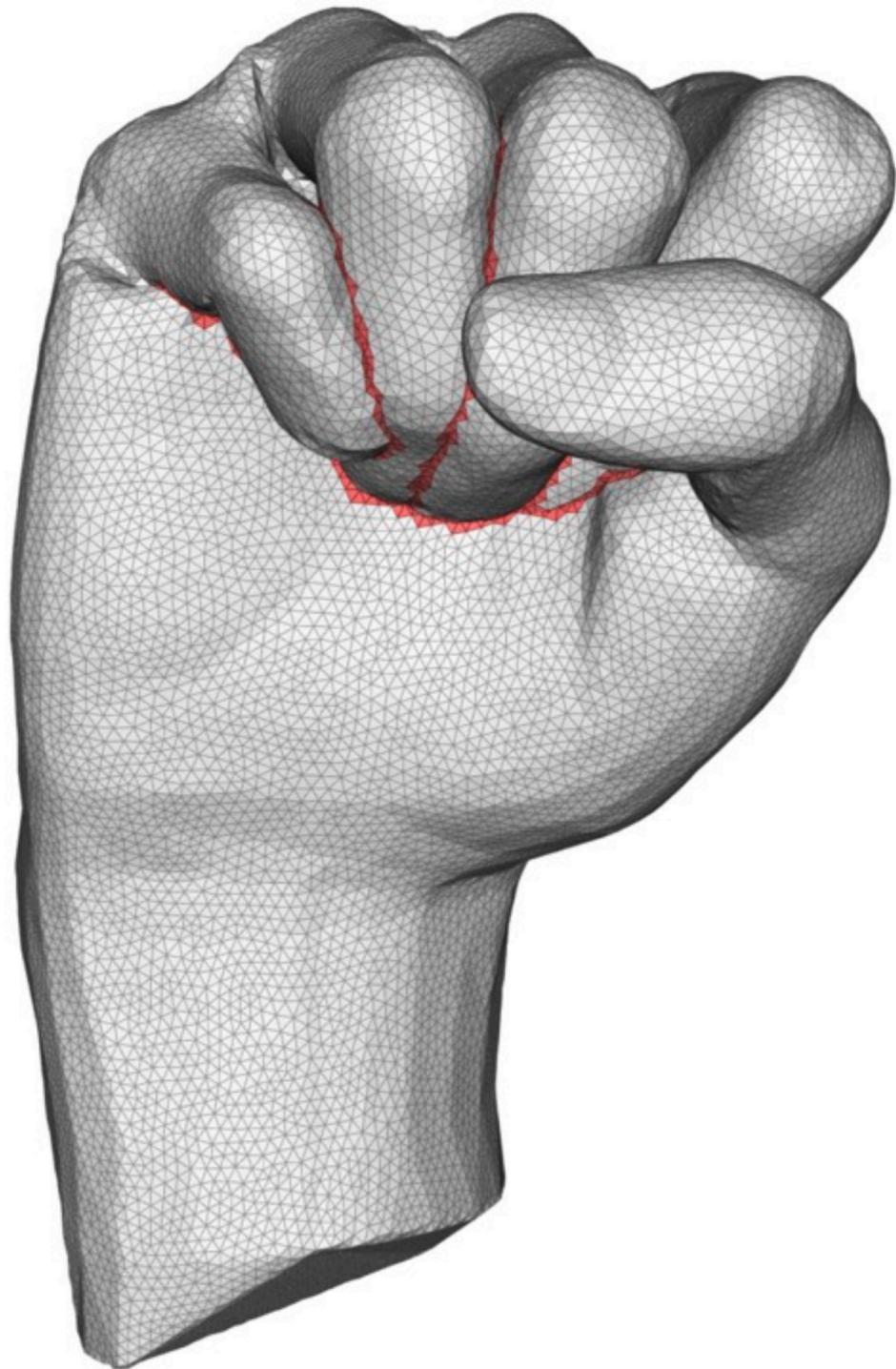
Why care about performance?



“Brigade 3” real time path tracing demo

One more query: mesh-mesh intersection

- **GEOMETRY:** How do we know if a mesh intersects itself?
- **ANIMATION:** How do we know if a collision occurred?



Warm up: point-point intersection

- Q: How do we know if p intersects a ?
- A: ...check if they're the same point!

$(p1, p2)$
●

● $(a1, a2)$

Sadly, life is not always so easy.

Slightly harder: point-line intersection

- Q: How do we know if a point intersects a given line?
- A: ...plug it into the line equation!

p
●

$$N^T x = c$$

I promise, life isn't always so easy.

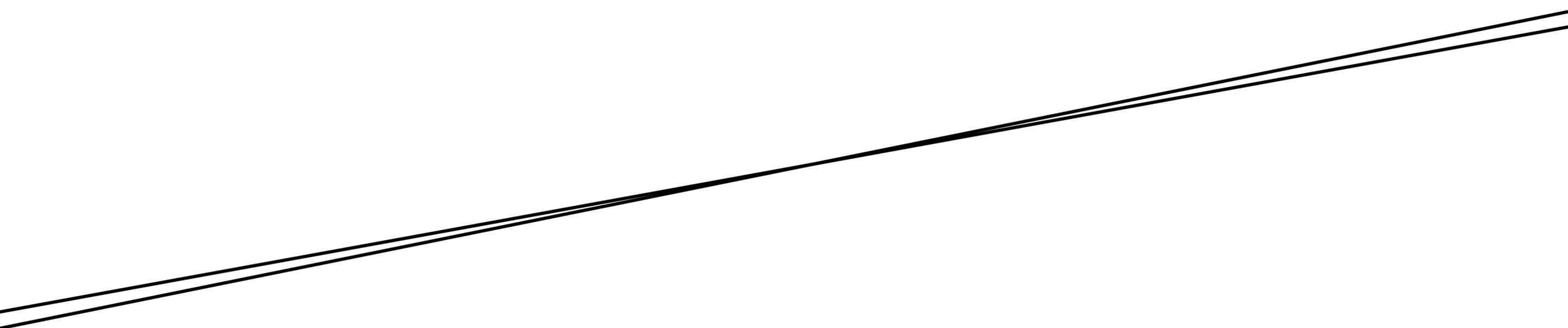
Finally interesting: line-line intersection

- Two lines: $ax=b$ and $cx=d$
- Q: How do we find the intersection?
- A: See if there is a simultaneous solution

- Leads to linear system:
$$\begin{bmatrix} a_1 & a_2 \\ c_1 & c_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b \\ d \end{bmatrix}$$

Degenerate line-line intersection?

- **What if lines are almost parallel?**
- **Small change in normal can lead to big change in intersection!**
- **Instability very common, very important with geometric predicates. Demands special care (e.g., analysis of matrix).**



Triangle-Triangle Intersection?

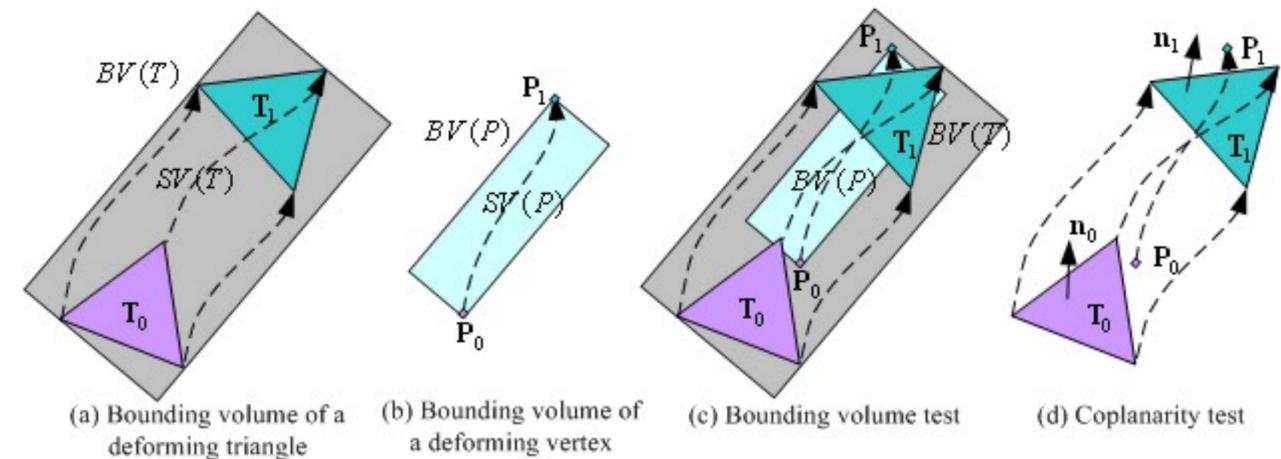
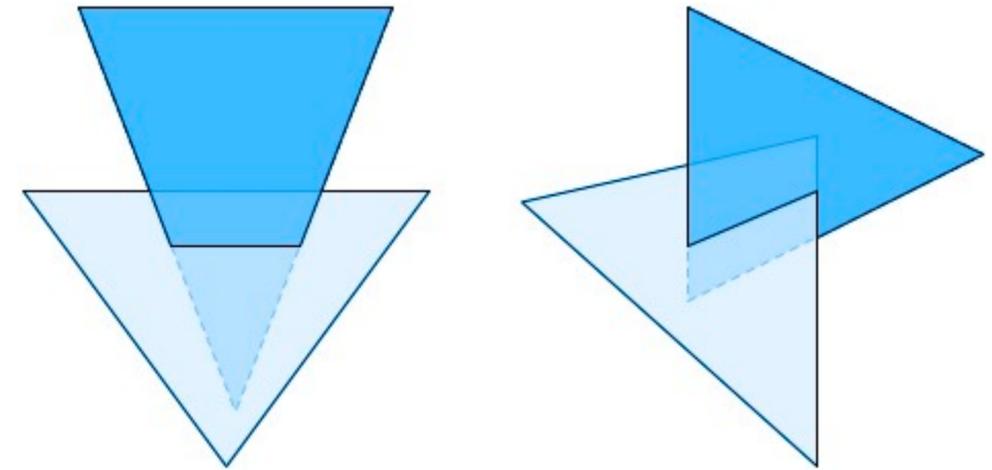
- Lots of ways to do it

- Basic idea:

- Q: Any ideas?
- One way: reduce to edge-triangle intersection
- Check if each line passes through plane
- Then do interval test

- What if triangle is *moving*?

- Important case for animation
- Can think of triangles as *prisms* in time
- Will say more when we talk about animation!



Up Next: Spatial Acceleration Data Structures

- Testing every element is *slow!*
- E.g., linearly scanning through a list vs. binary search
- Can apply this same kind of thinking to geometric queries

