

Digital Geometry Processing

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

Scotty 3D setup recitation

- **Today!**
- **Hunt Library Computer Lab**
- **3:30-5pm**

Last time part 1: overview of geometry

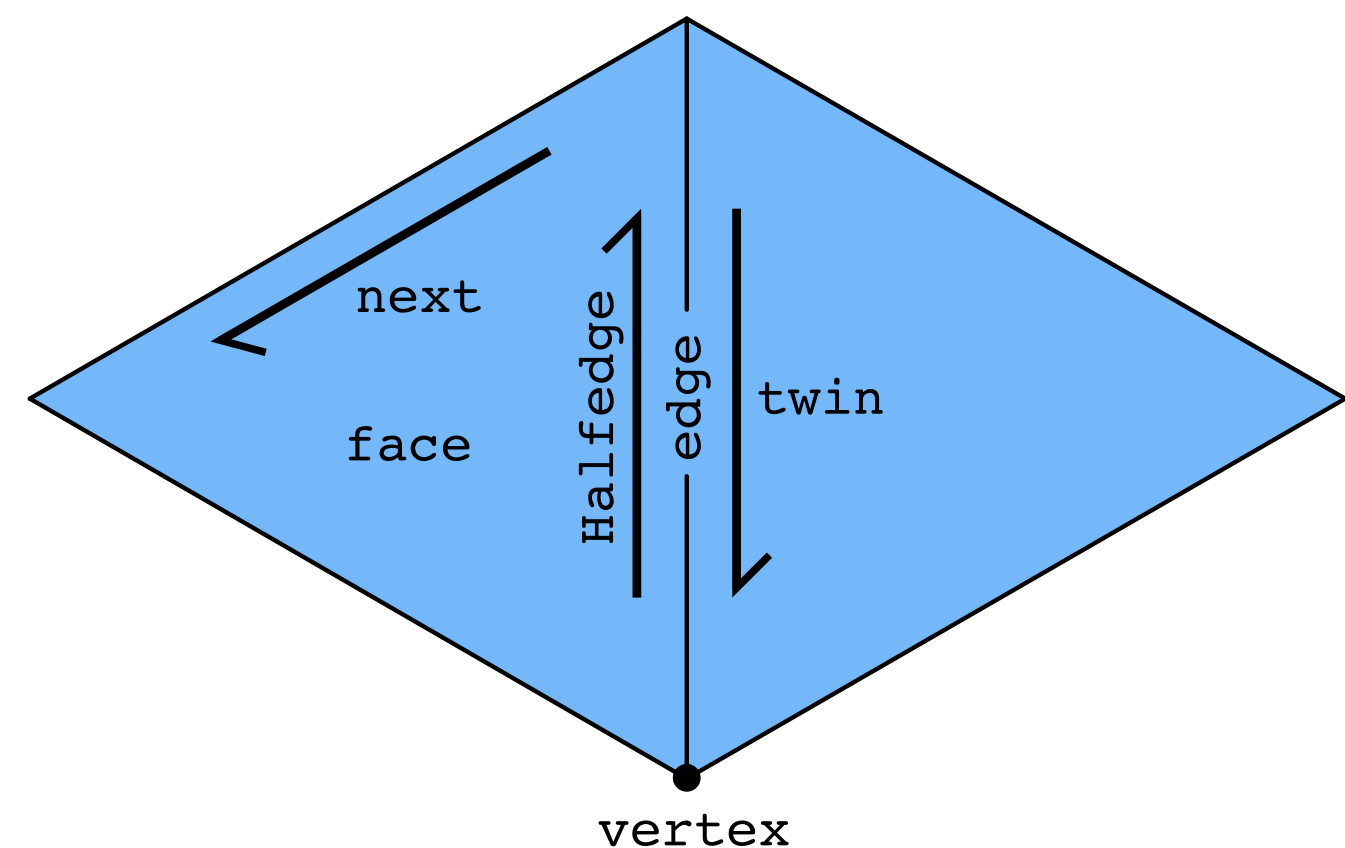
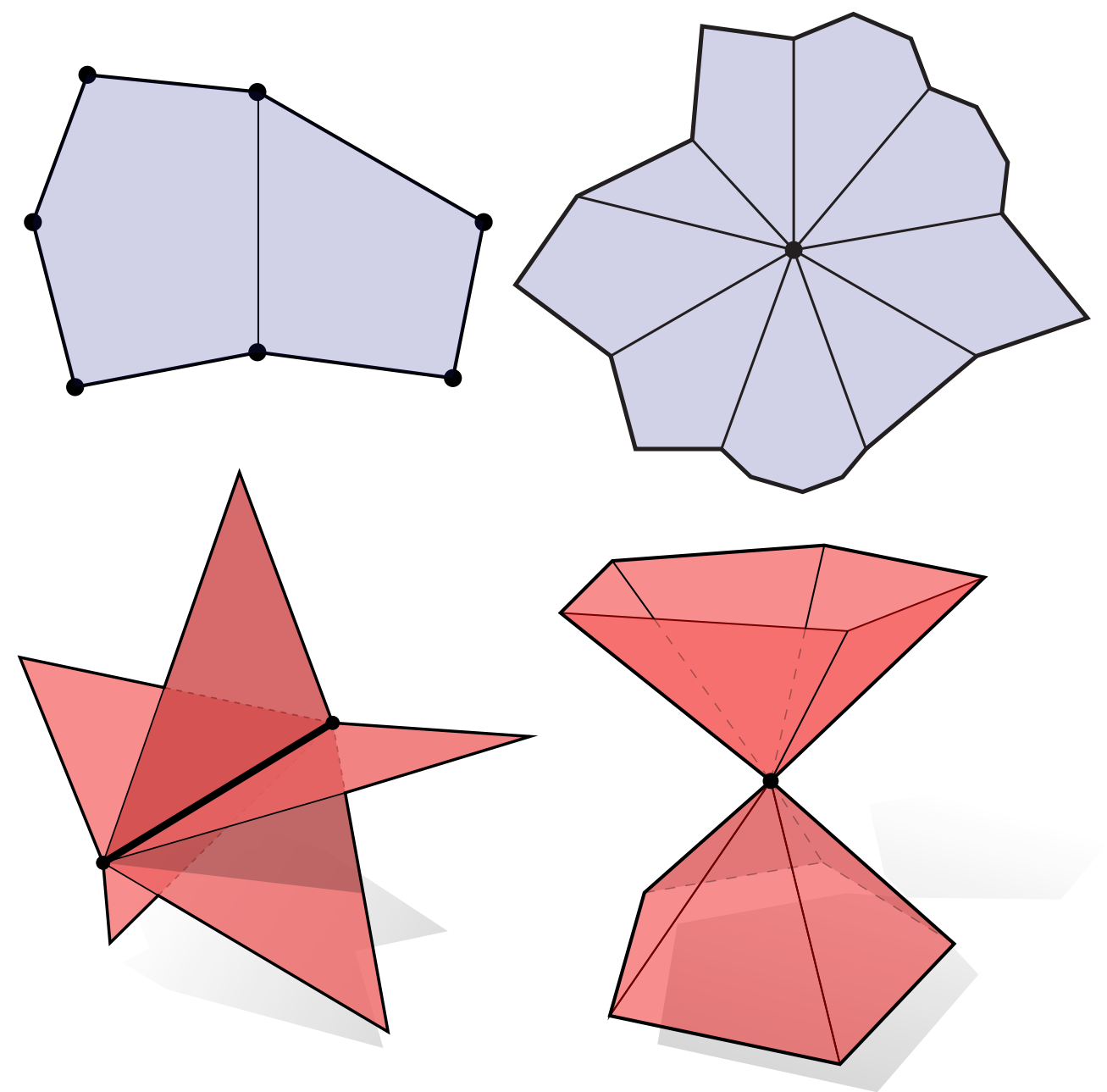
- Many types of geometry in nature
- Demand sophisticated representations
- Two major categories:
 - IMPLICIT - “tests” if a point is in shape
 - EXPLICIT - directly “lists” points
- Lots of representations for both

Geometry



Last time part 2: 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...)**

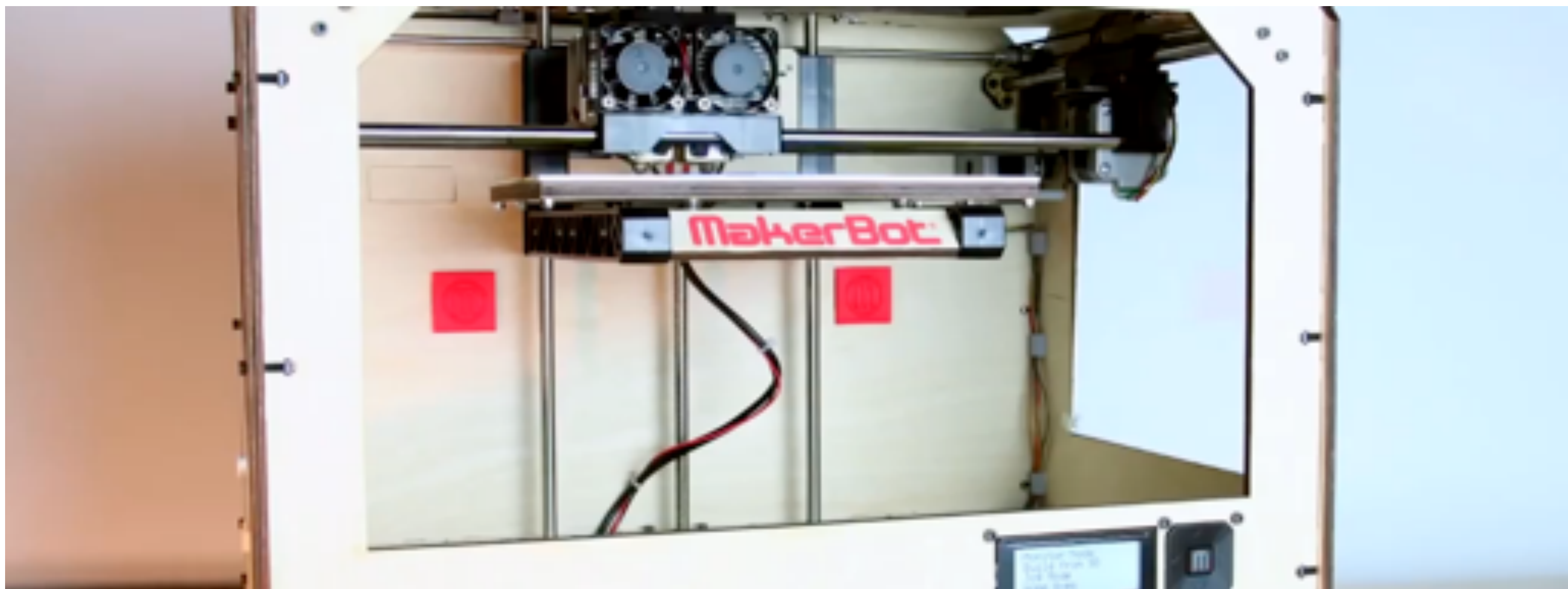


Digital Geometry Processing: Motivation

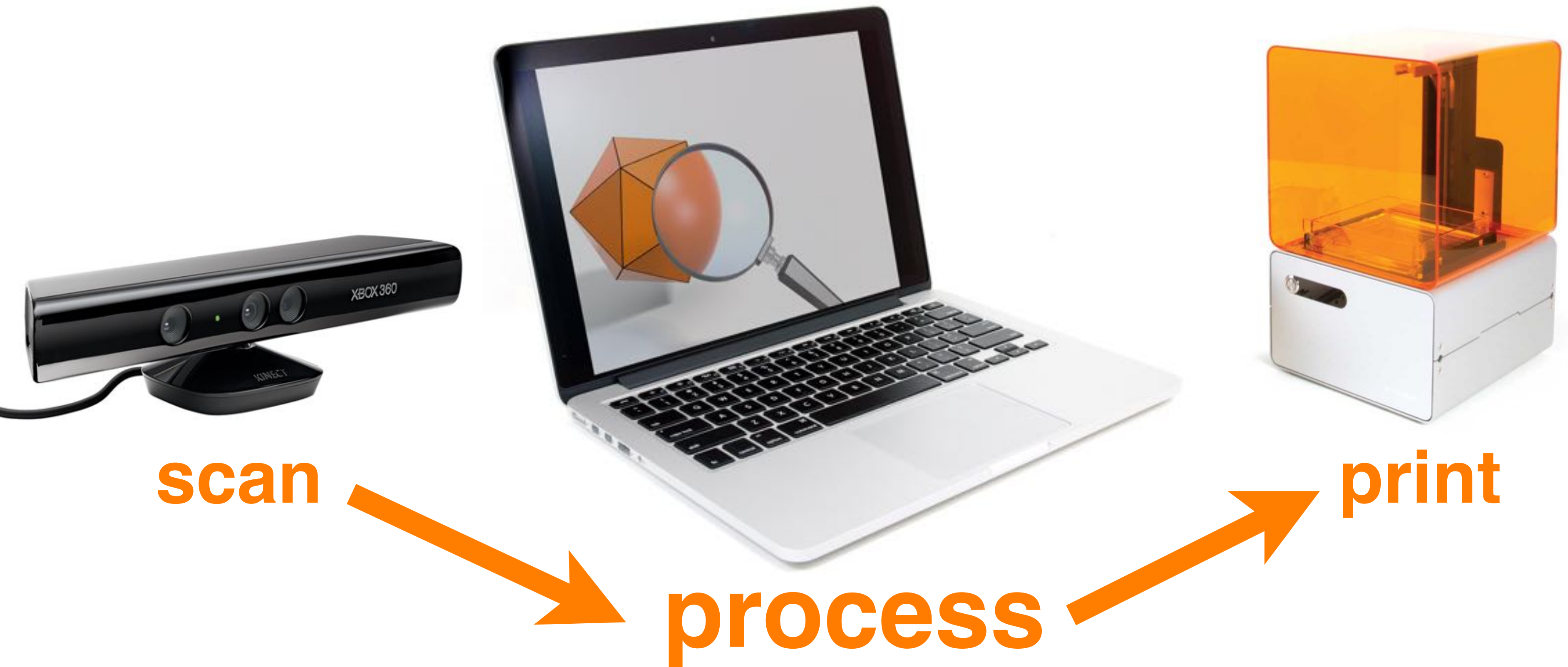
3D Scanning



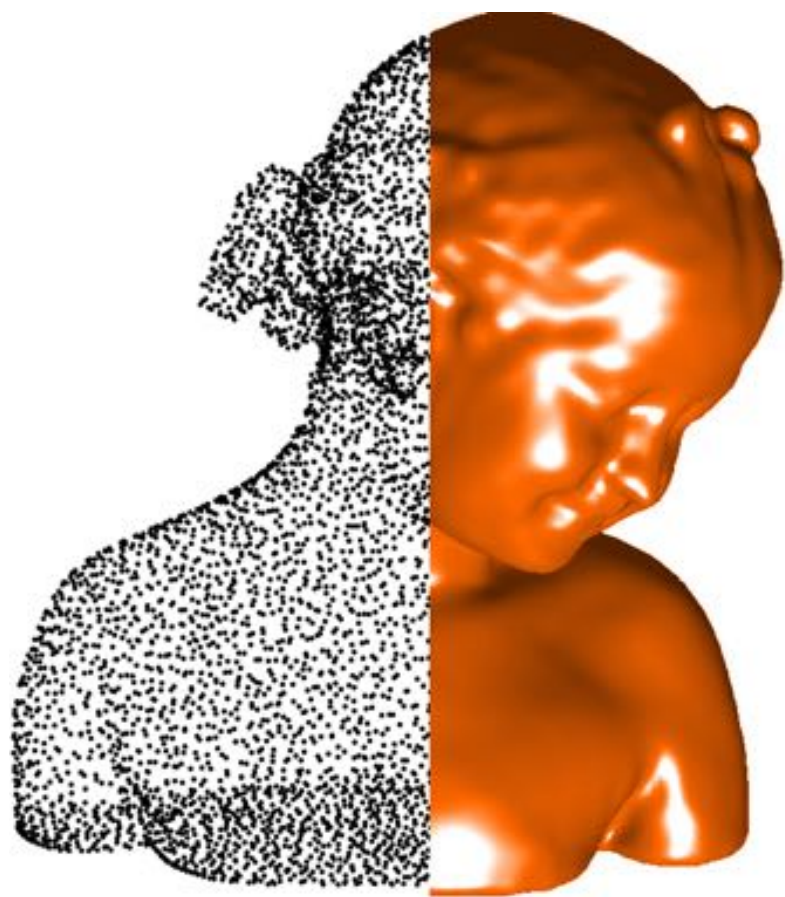
3D Printing



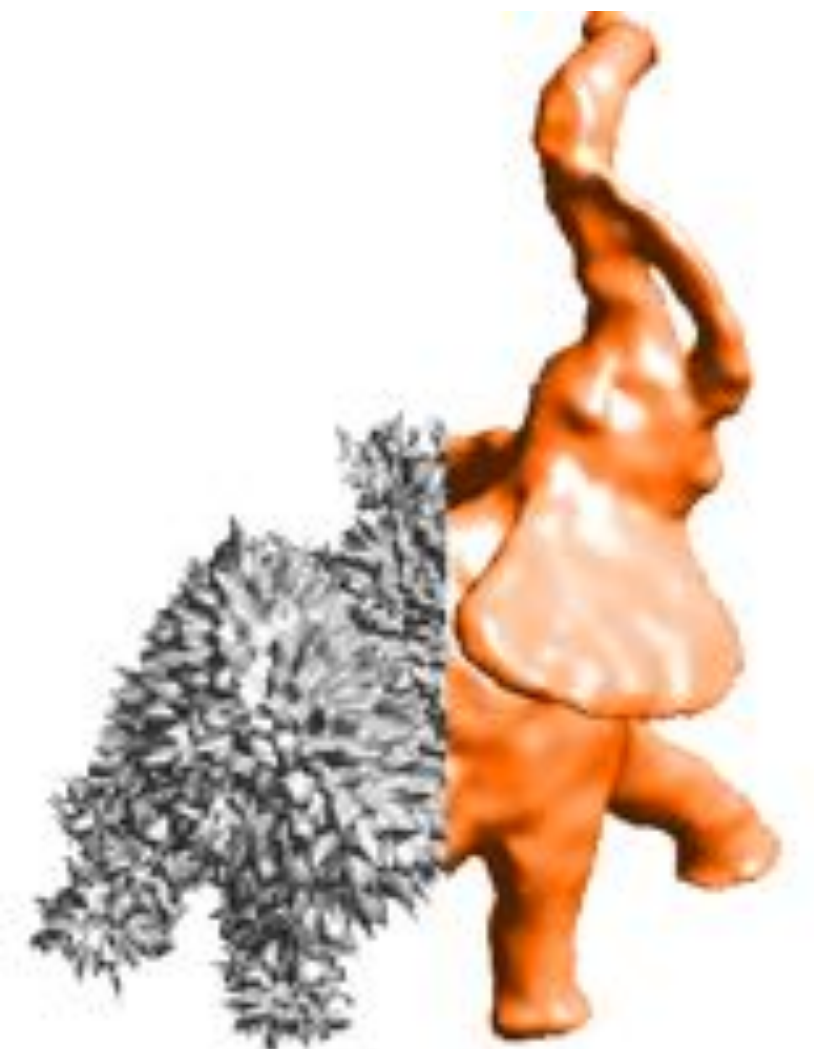
Geometry Processing Pipeline



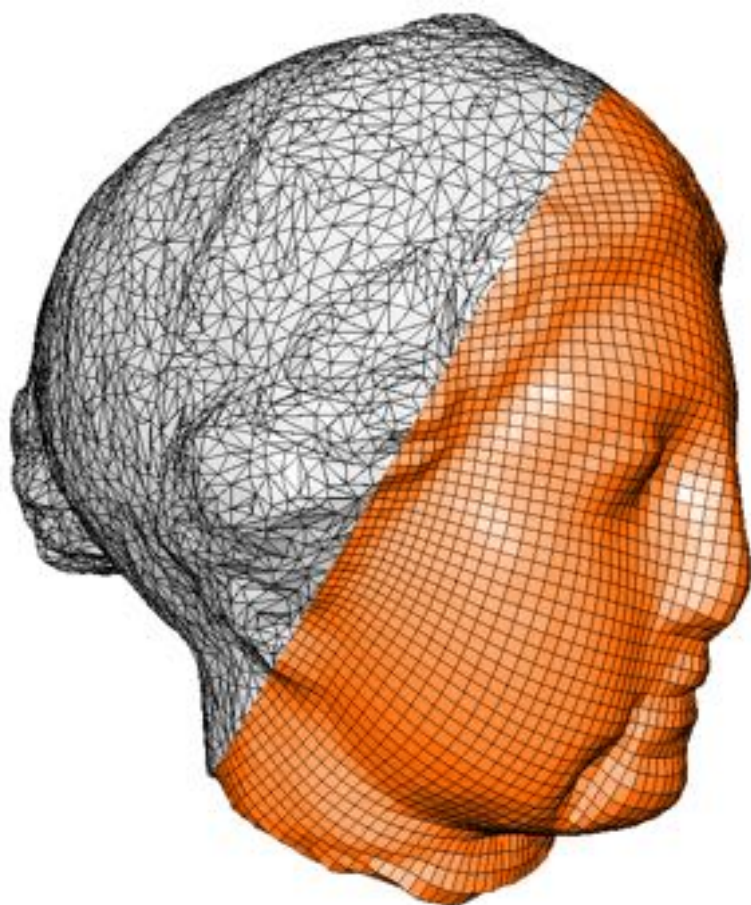
Geometry Processing Tasks



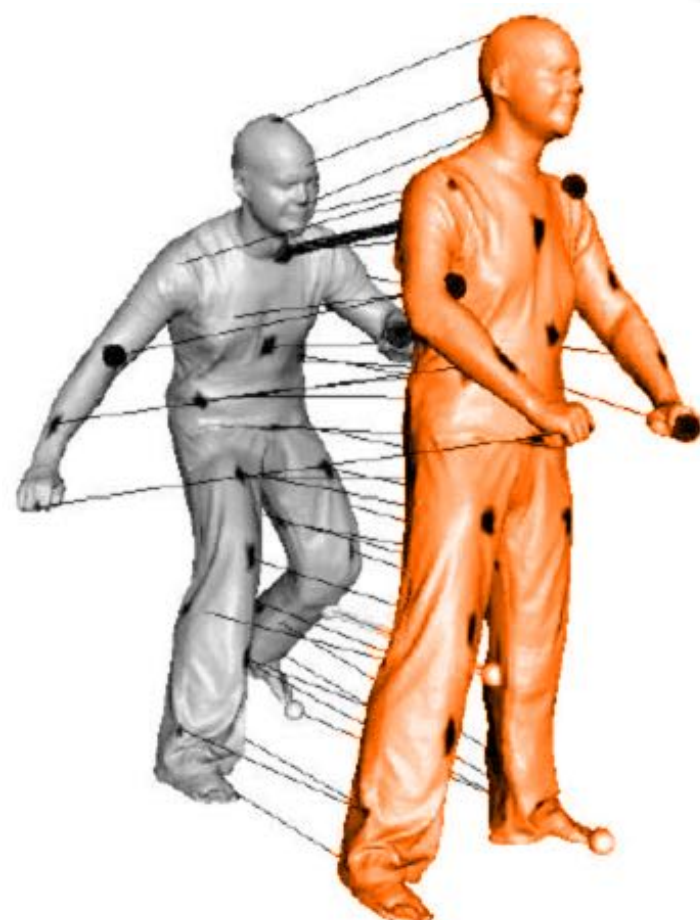
reconstruction



filtering



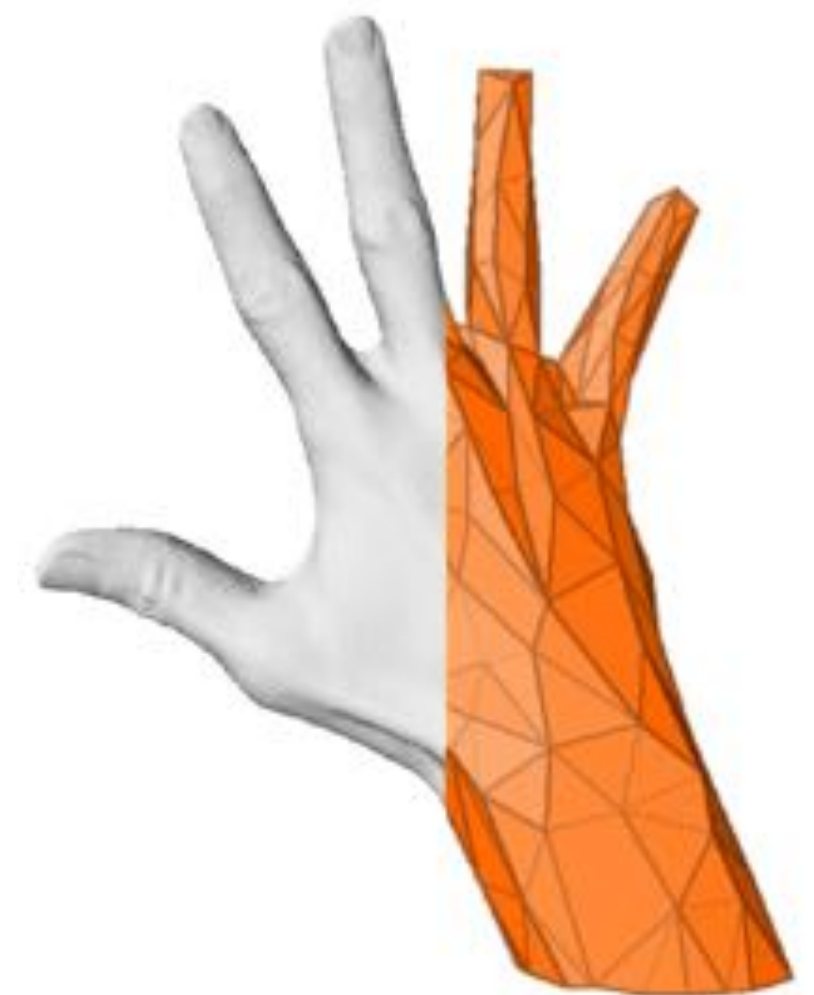
remeshing



shape analysis



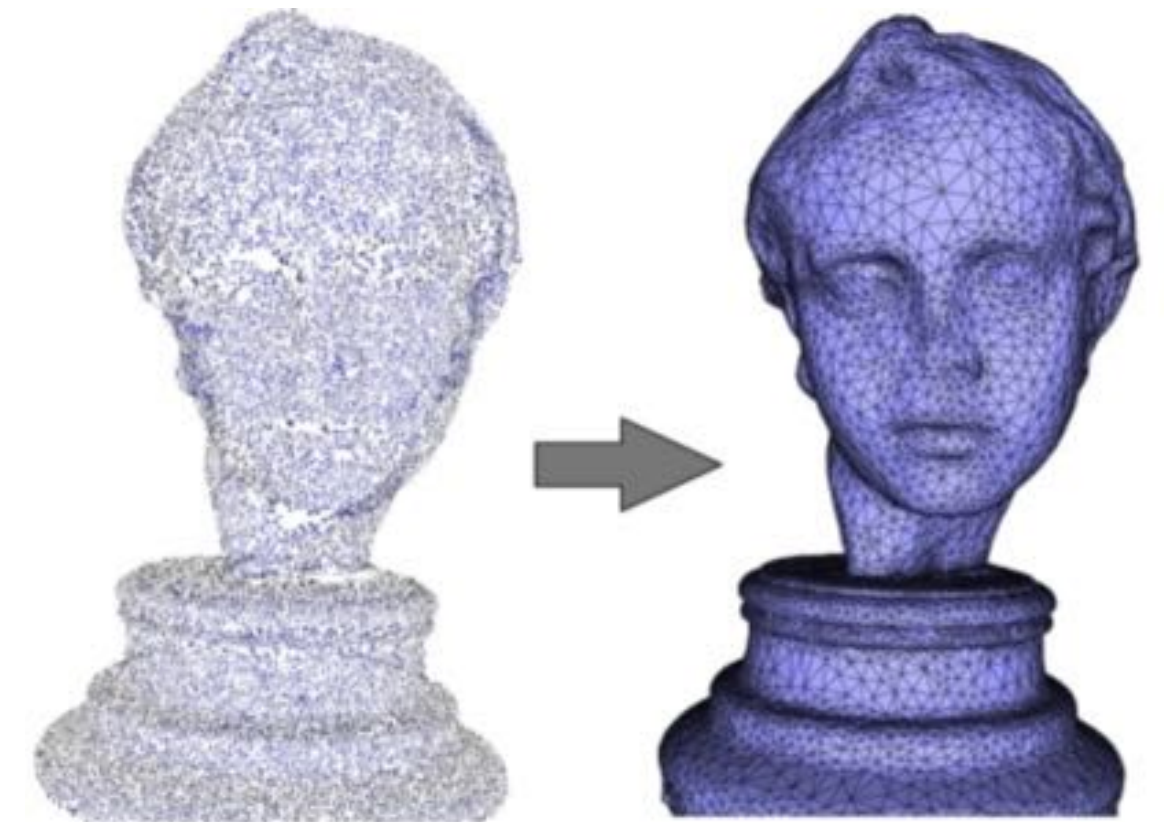
parameterization



compression

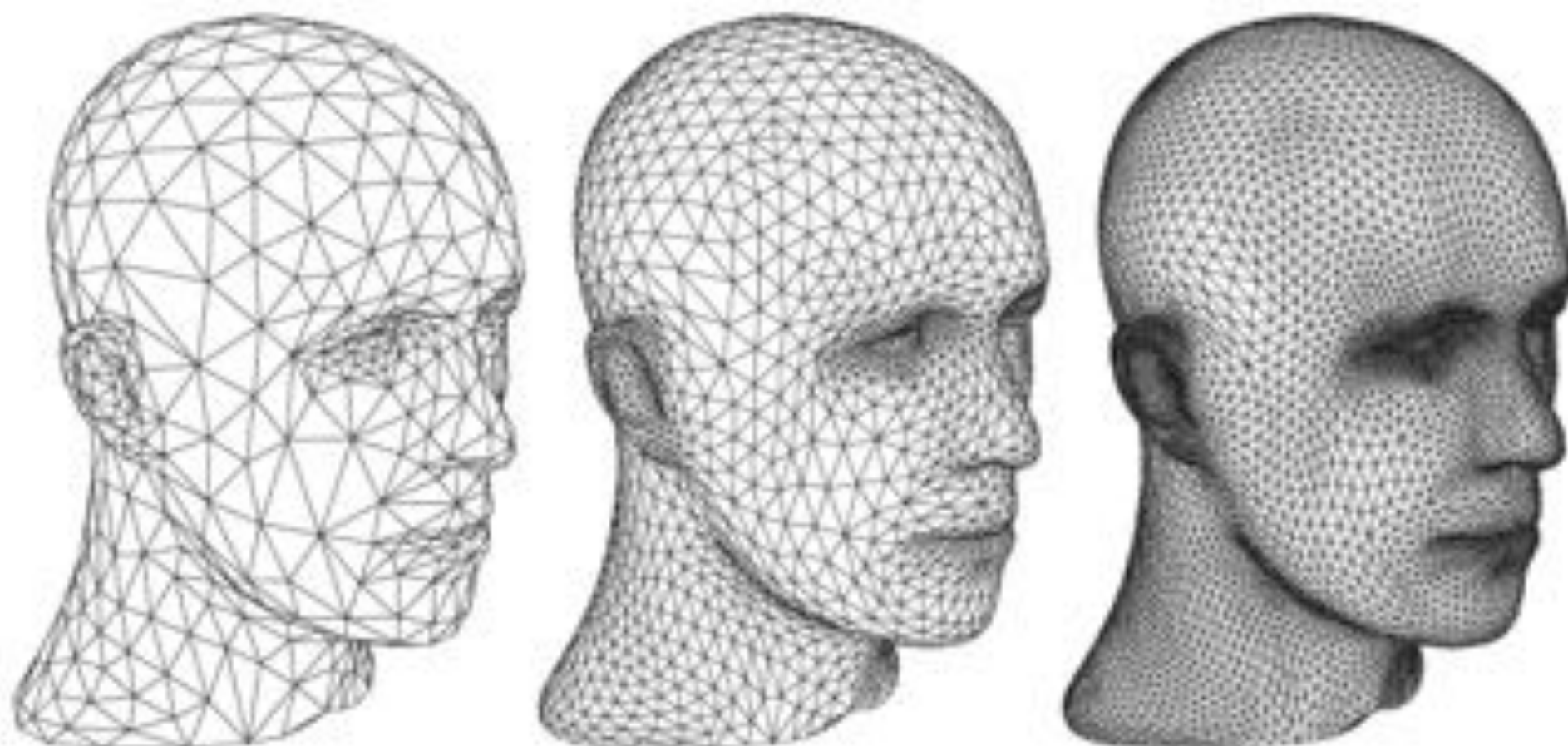
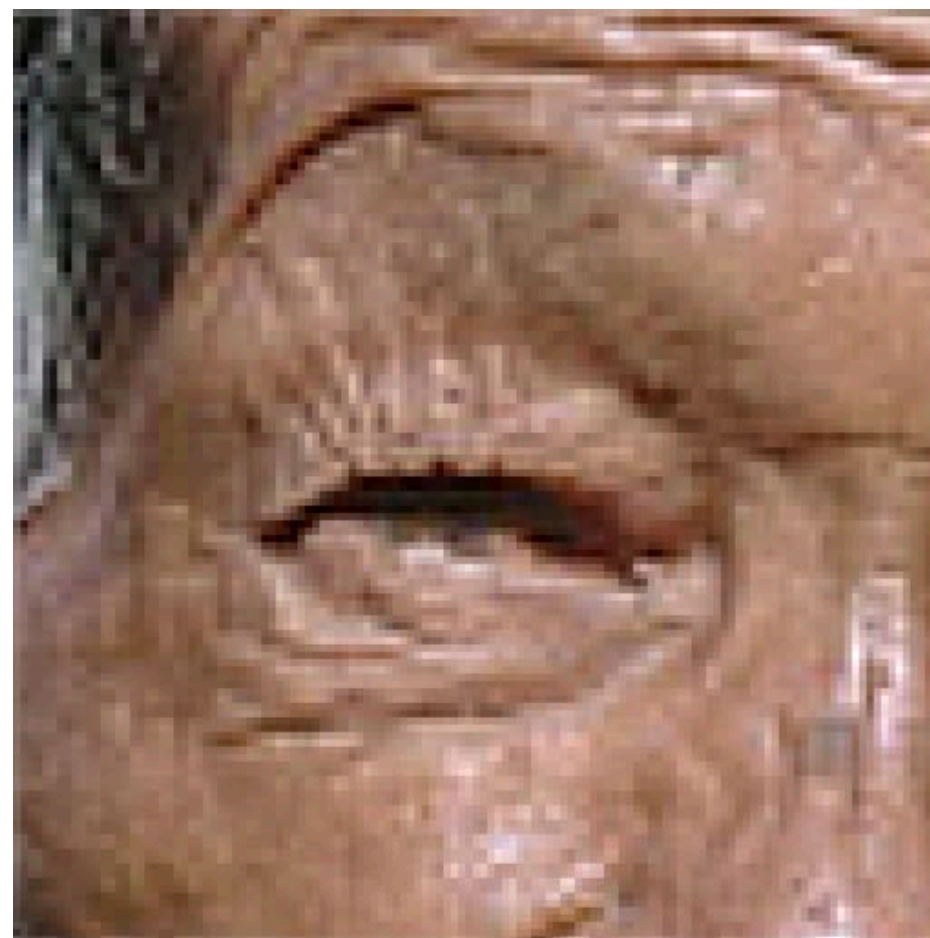
Geometry Processing: Reconstruction

- **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)**



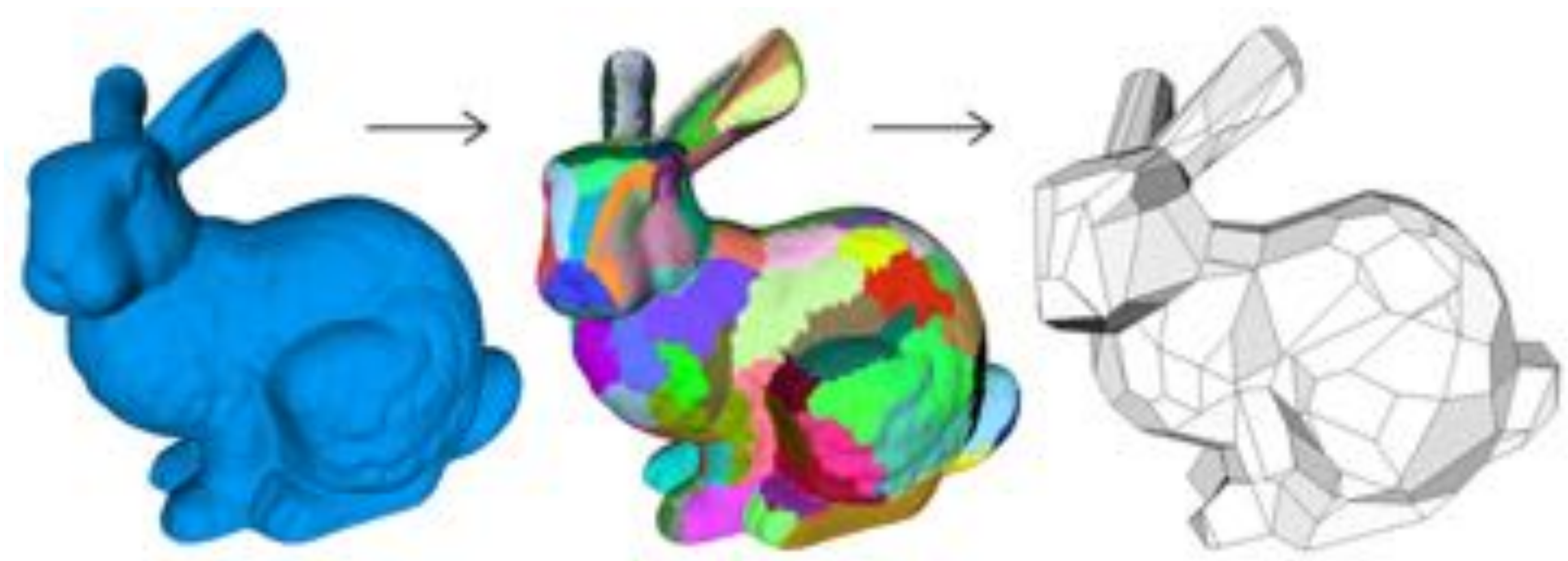
Geometry Processing: Upsampling

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



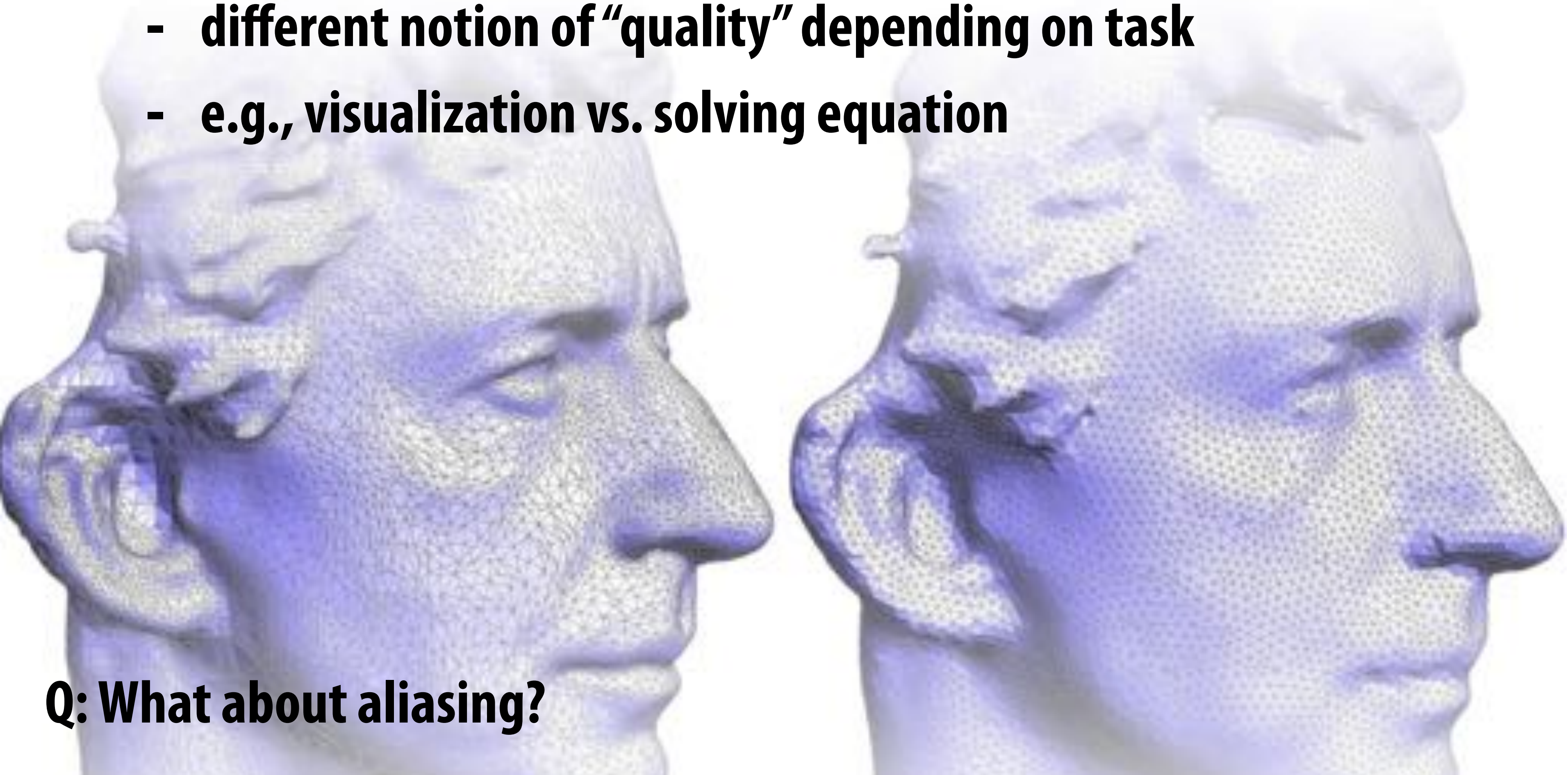
Geometry Processing: Downsampling

- 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

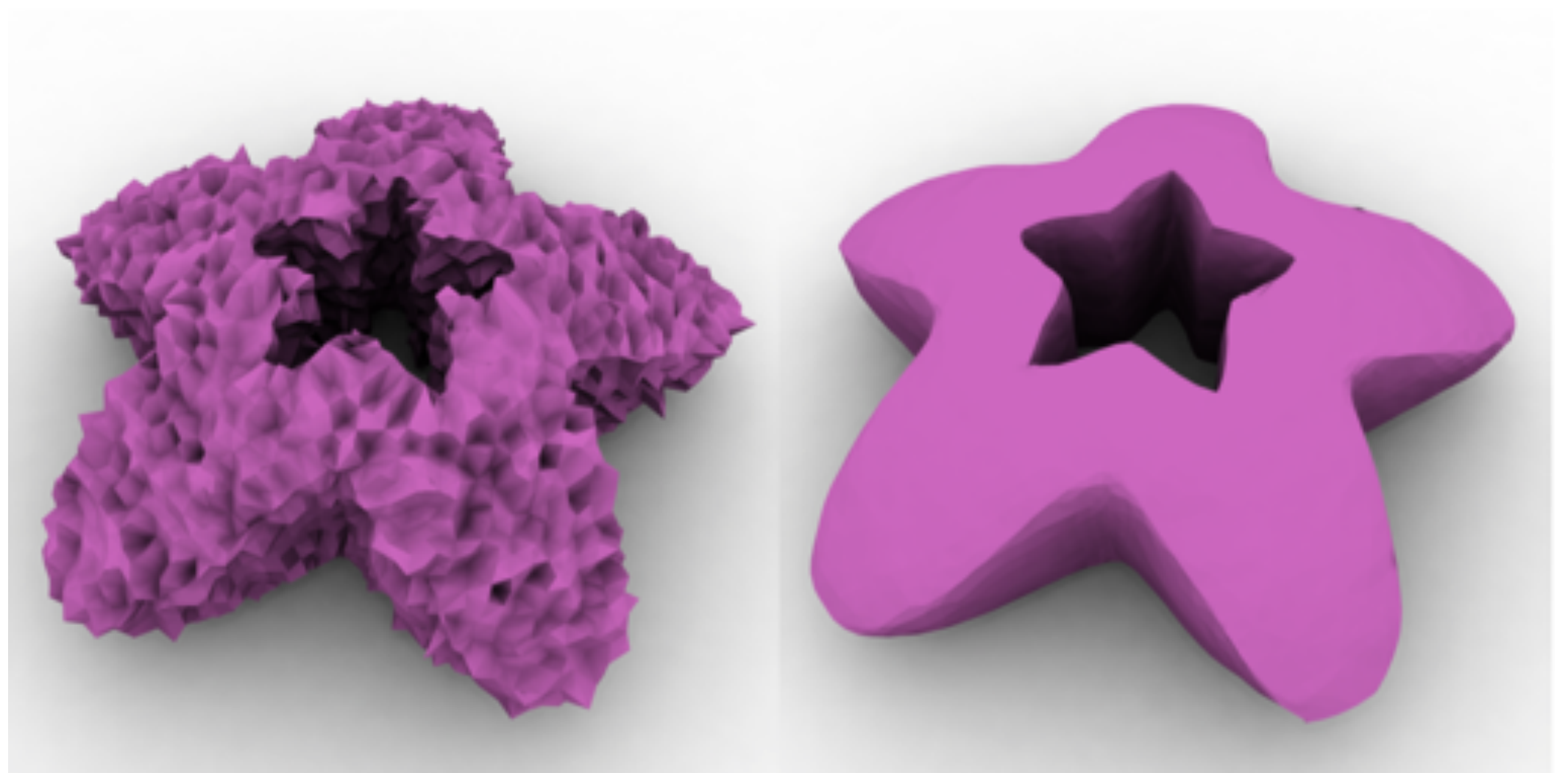
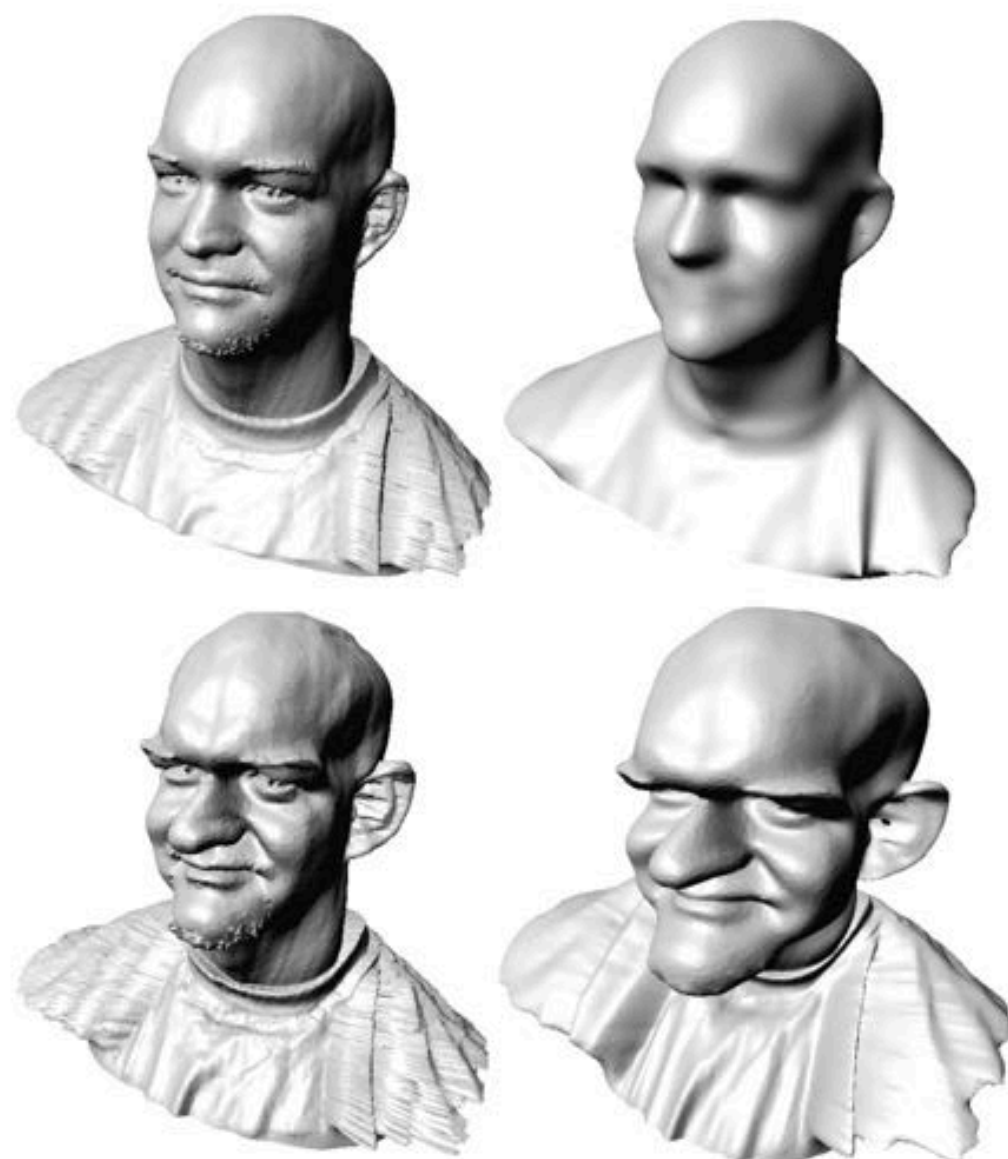
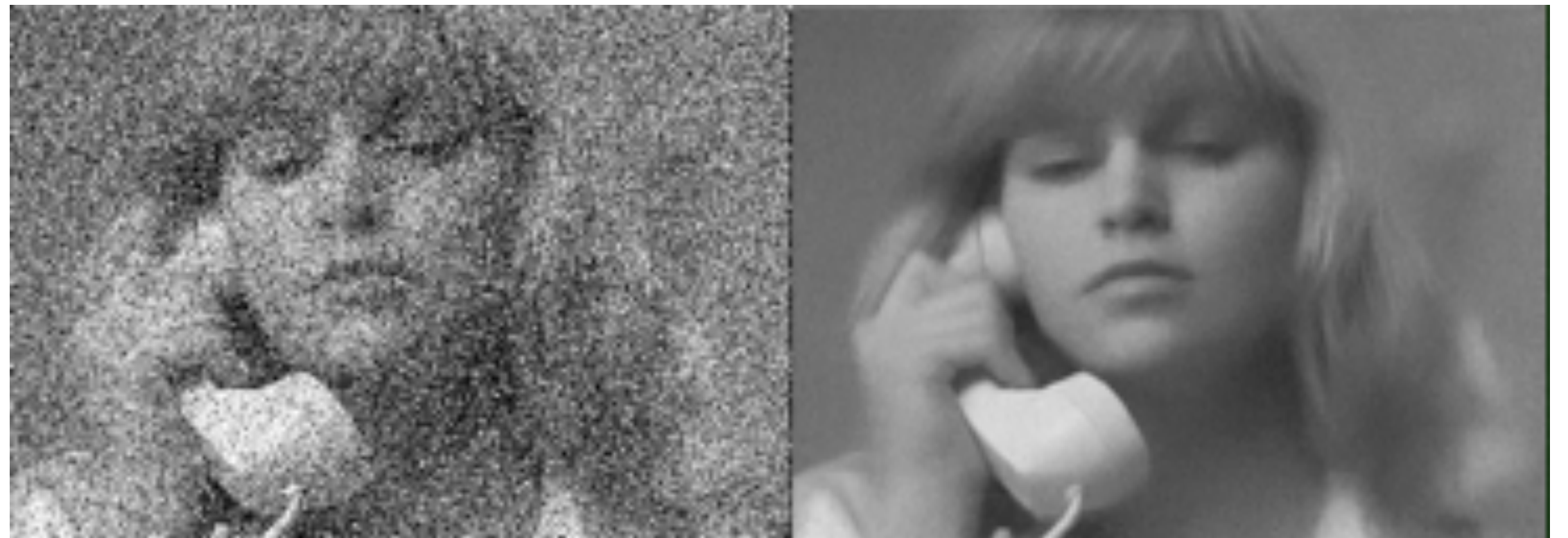
- **Modify sample distribution to improve quality**
- **Images: not an issue! (Pixels always stored on a regular grid)**
- **Meshes: shape of polygons is extremely important!**
 - **different notion of “quality” depending on task**
 - **e.g., visualization vs. solving equation**



Q: What about aliasing?

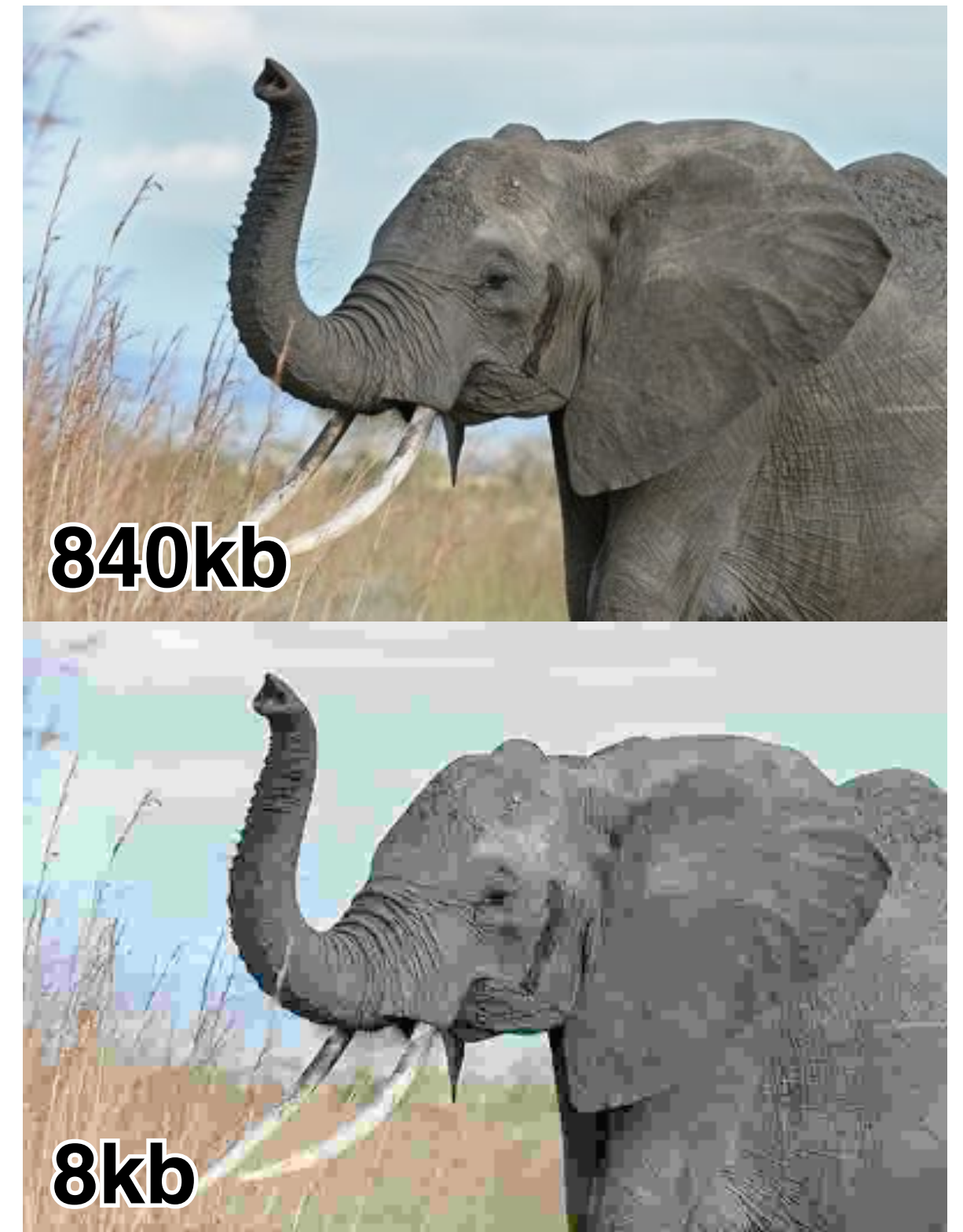
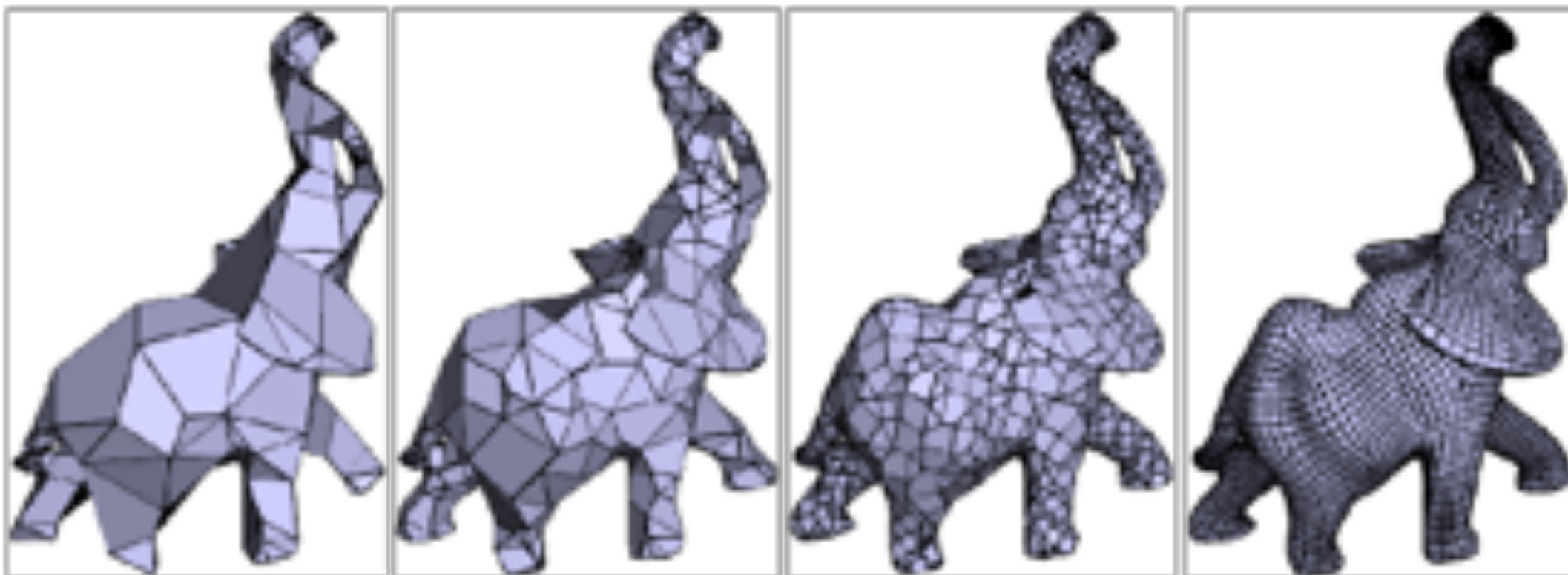
Geometry Processing: Filtering

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



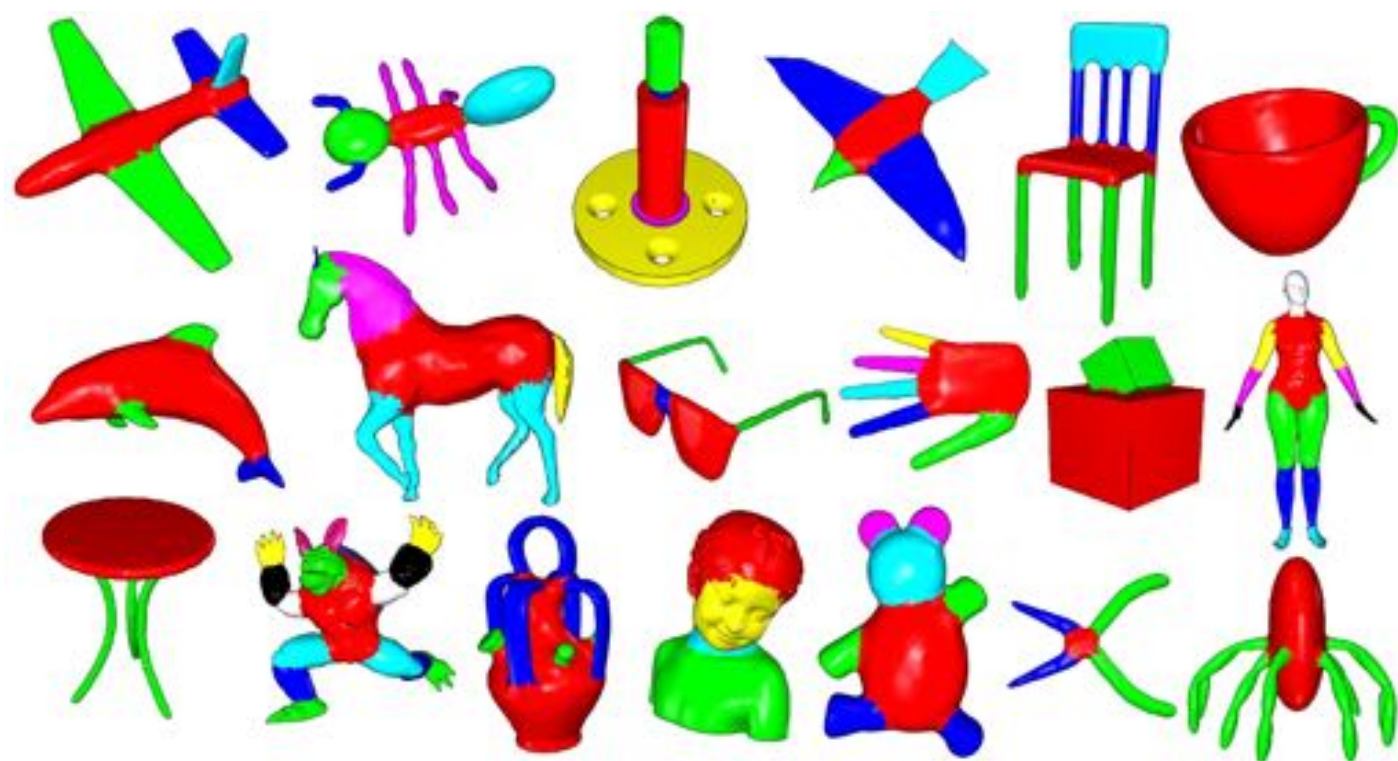
Geometry Processing: Compression

- 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



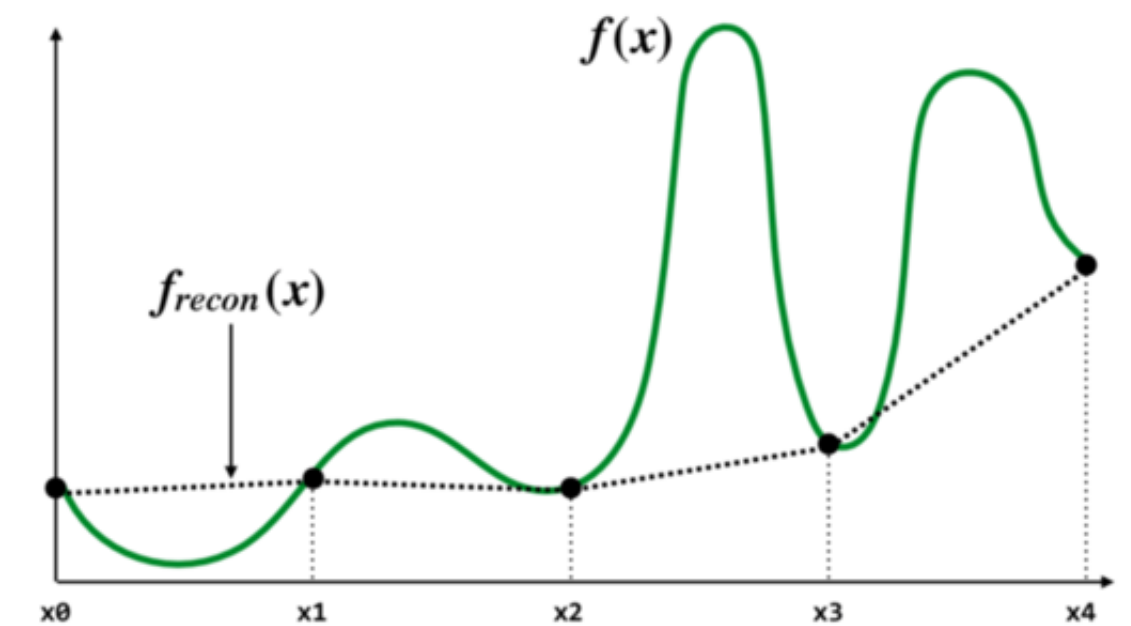
Intrinsic symmetry



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

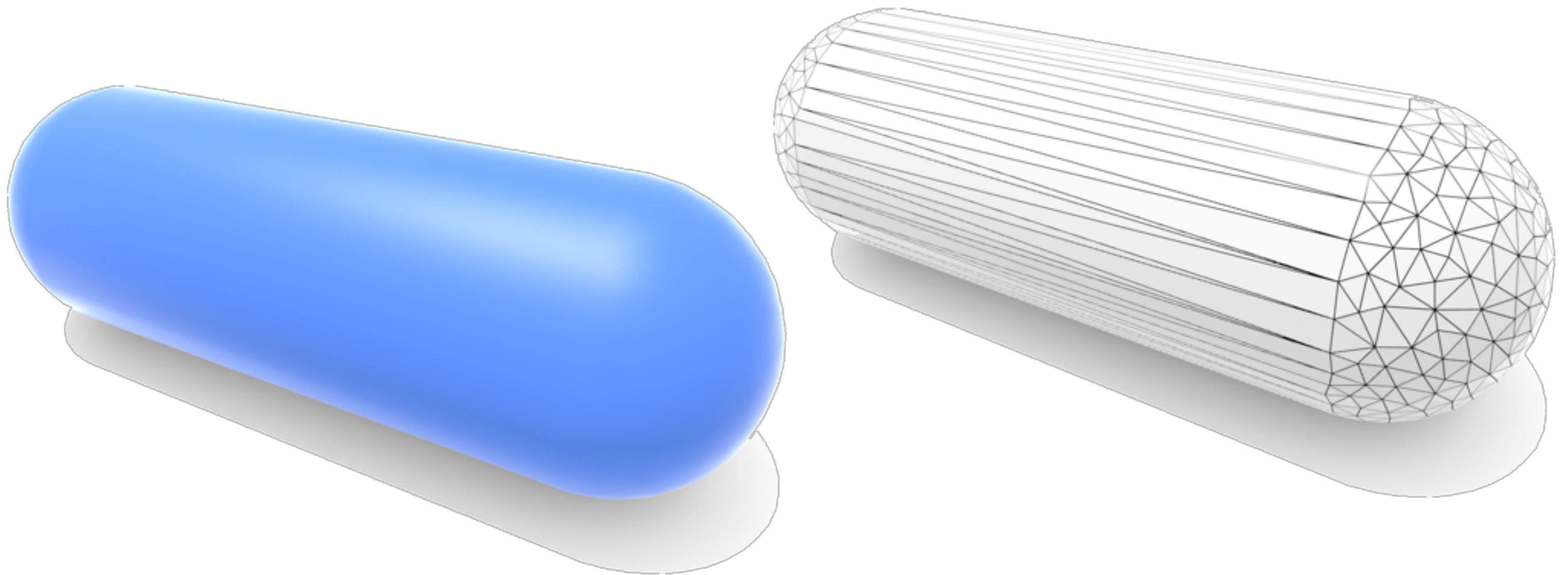
Remeshing as resampling

- 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 bad for performance



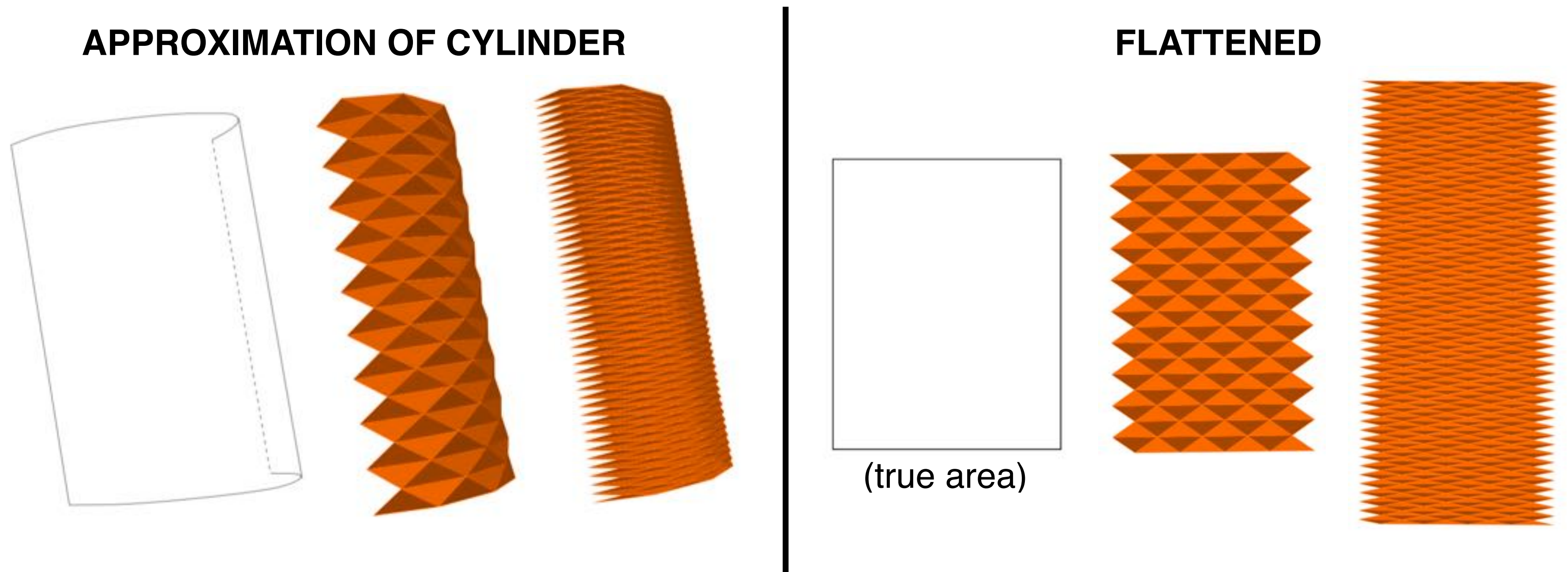
What makes a “good” mesh?

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



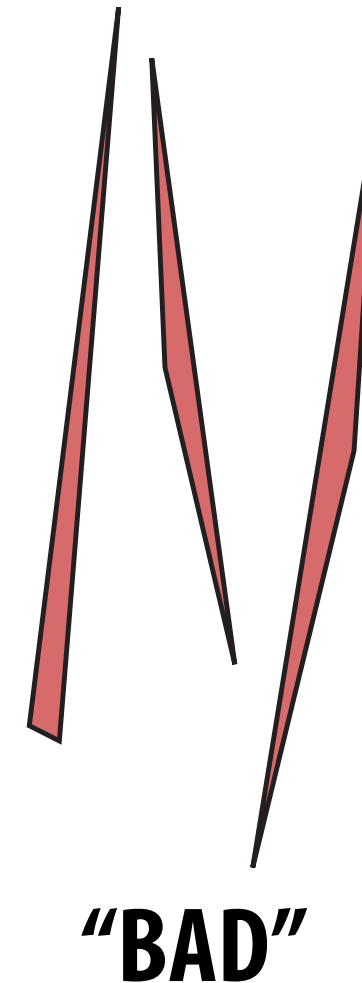
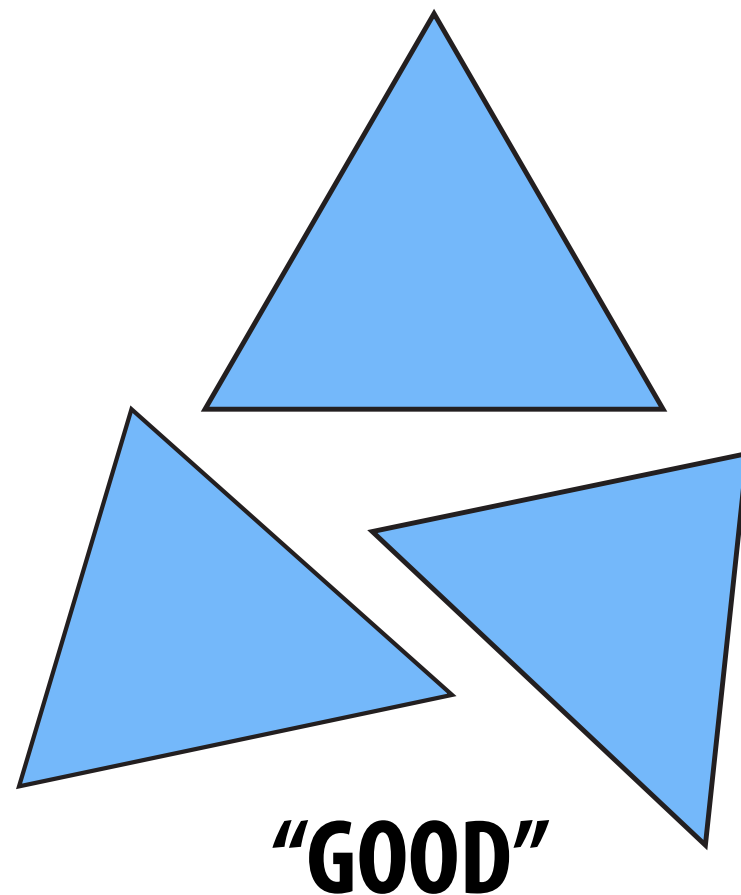
Approximation of position is not enough!

- Just because the vertices of a mesh are very close to the surface it approximates does not mean it's a good approximation!
- Need to consider other factors, e.g., close approximation of surface normals
- Otherwise, can have wrong appearance, wrong area, wrong...

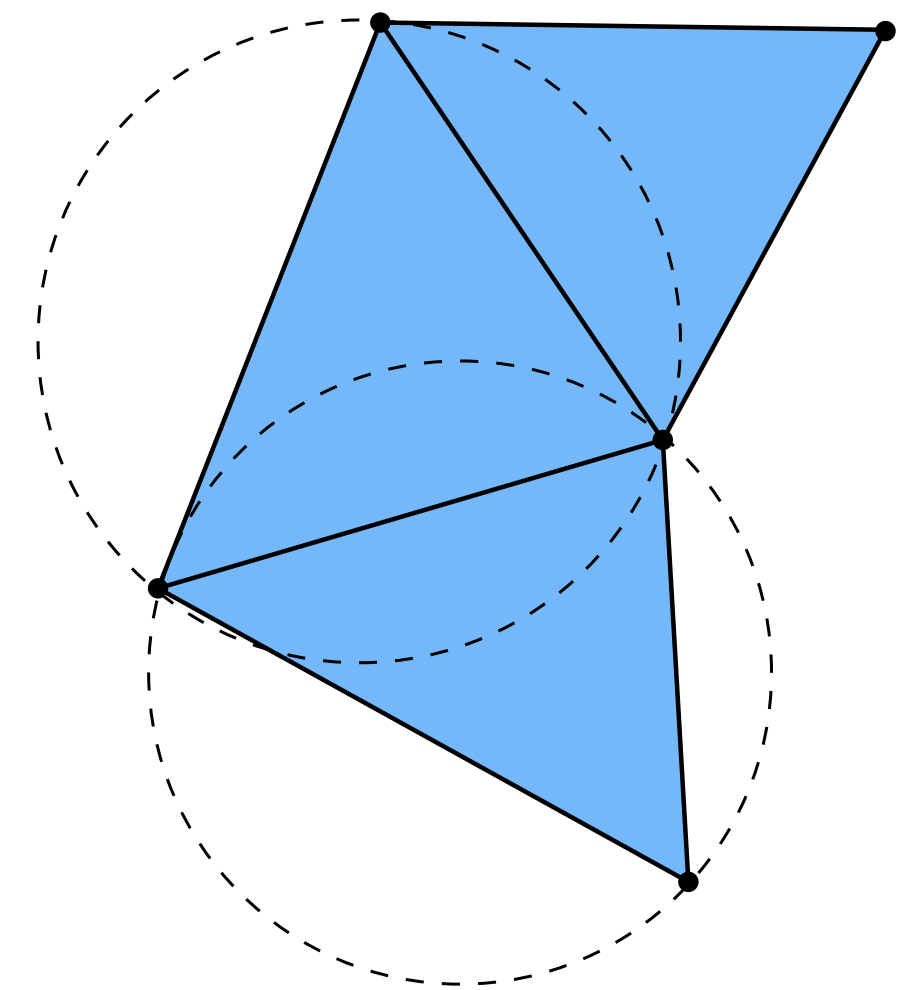


What else makes a “good” triangle mesh?

- Another rule of thumb: triangle



DELAUNAY

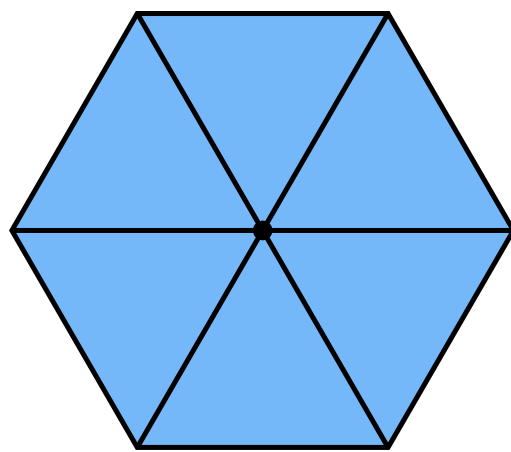


- E.g., all angles close to 60 degrees
- More sophisticated condition: Delaunay
- Can help w/ numerical accuracy/stability
- Tradeoffs w/ good geometric approximation*
 - e.g., long & skinny might be “more efficient”

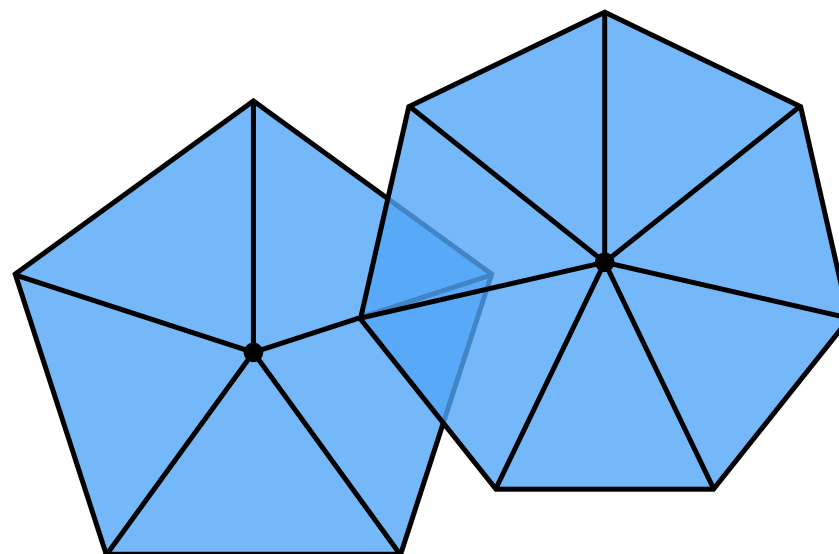
*See Shewchuk, “What is a Good Linear Element”

What else constitutes a good mesh?

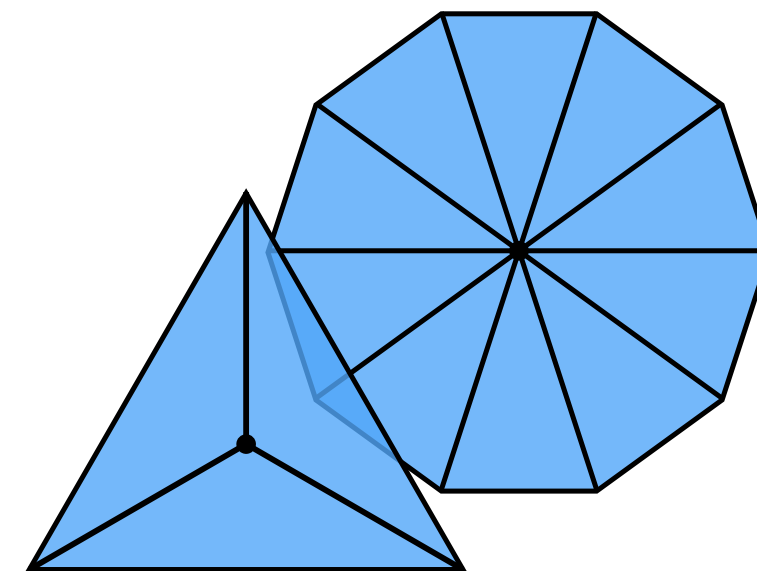
- Another rule of thumb: regular vertex degree
- E.g., valence 6 for triangle meshes (equilateral)



"GOOD"

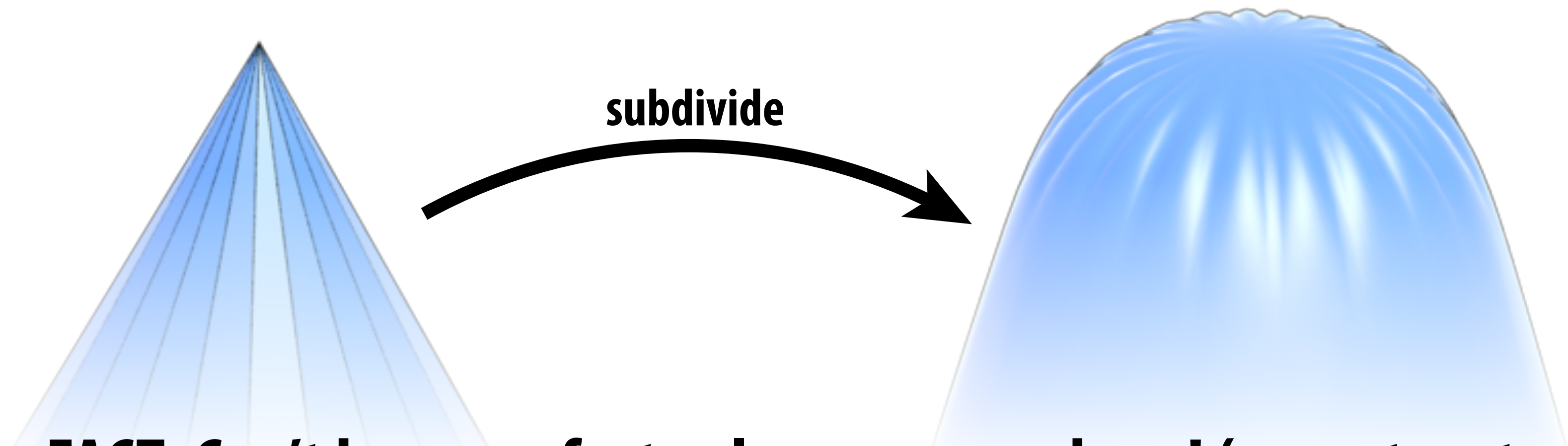


"OK"



"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

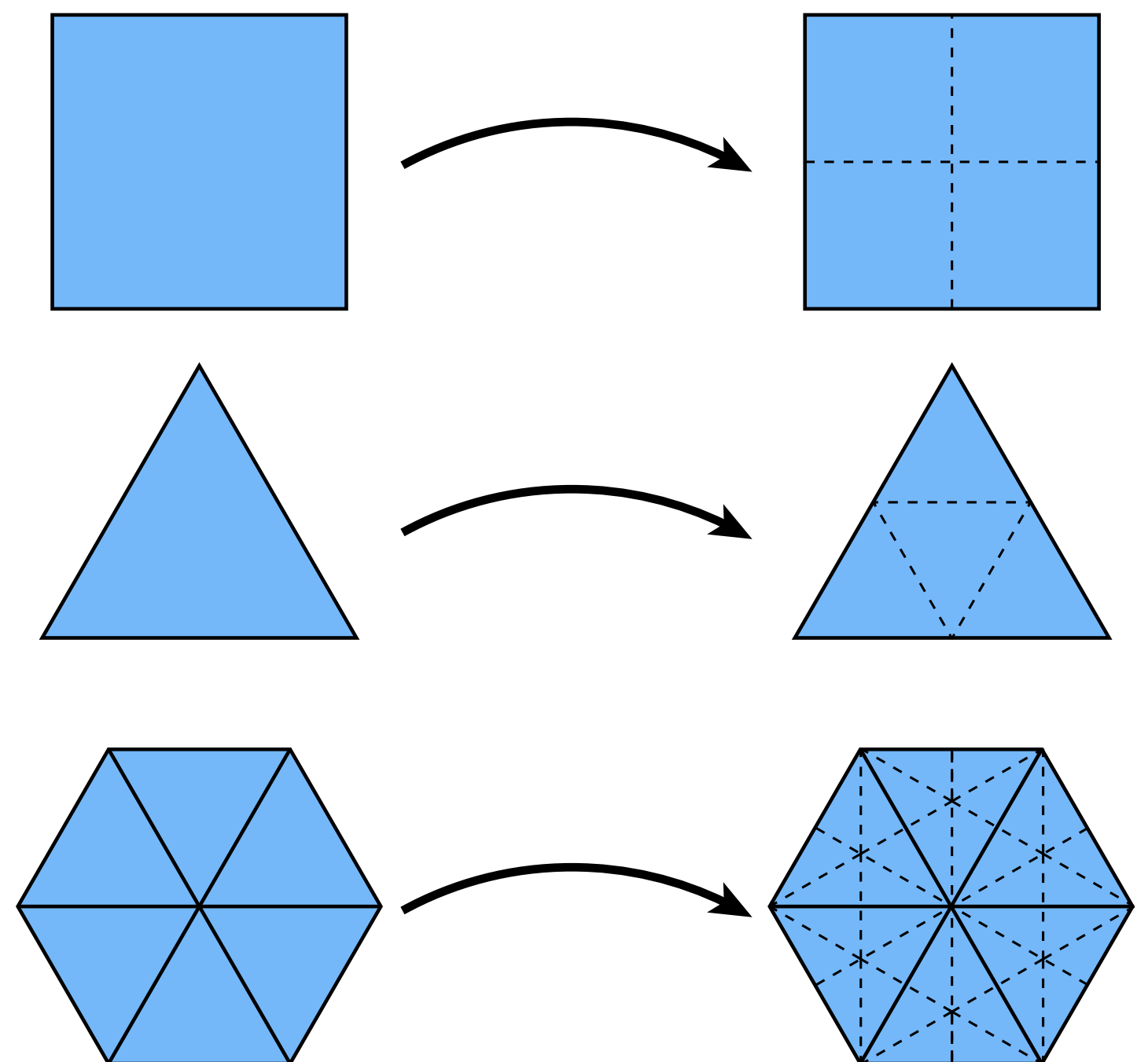
- Repeatedly split each element into smaller pieces
- Replace vertex positions with weighted average of neighbors

- Main considerations:

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

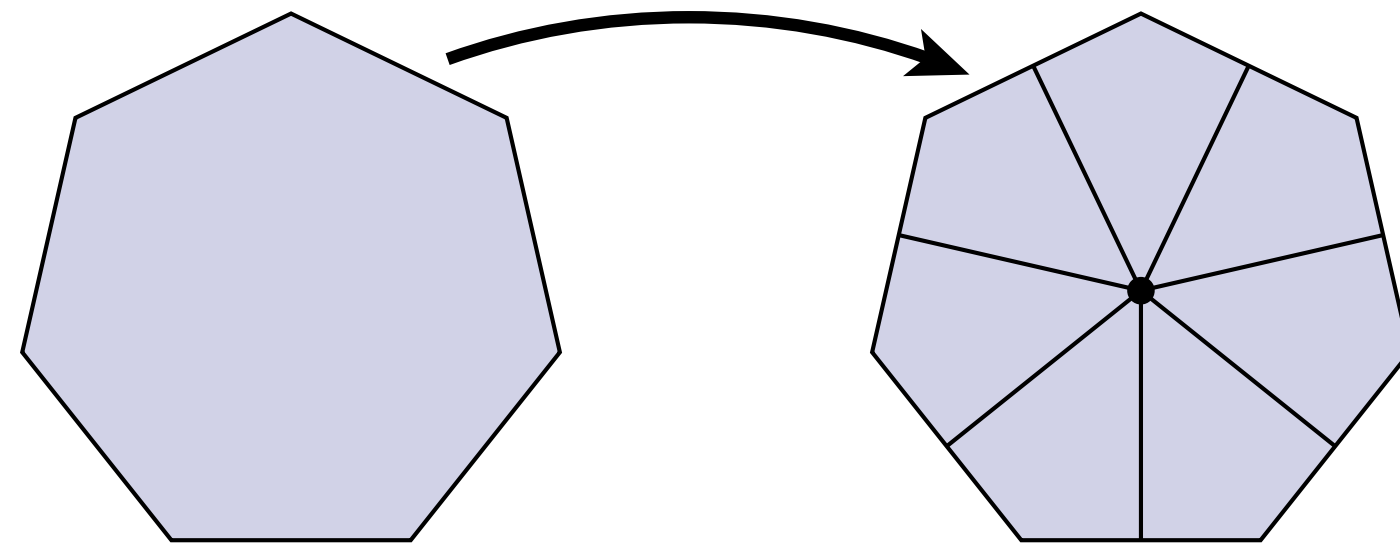
- Many options:

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

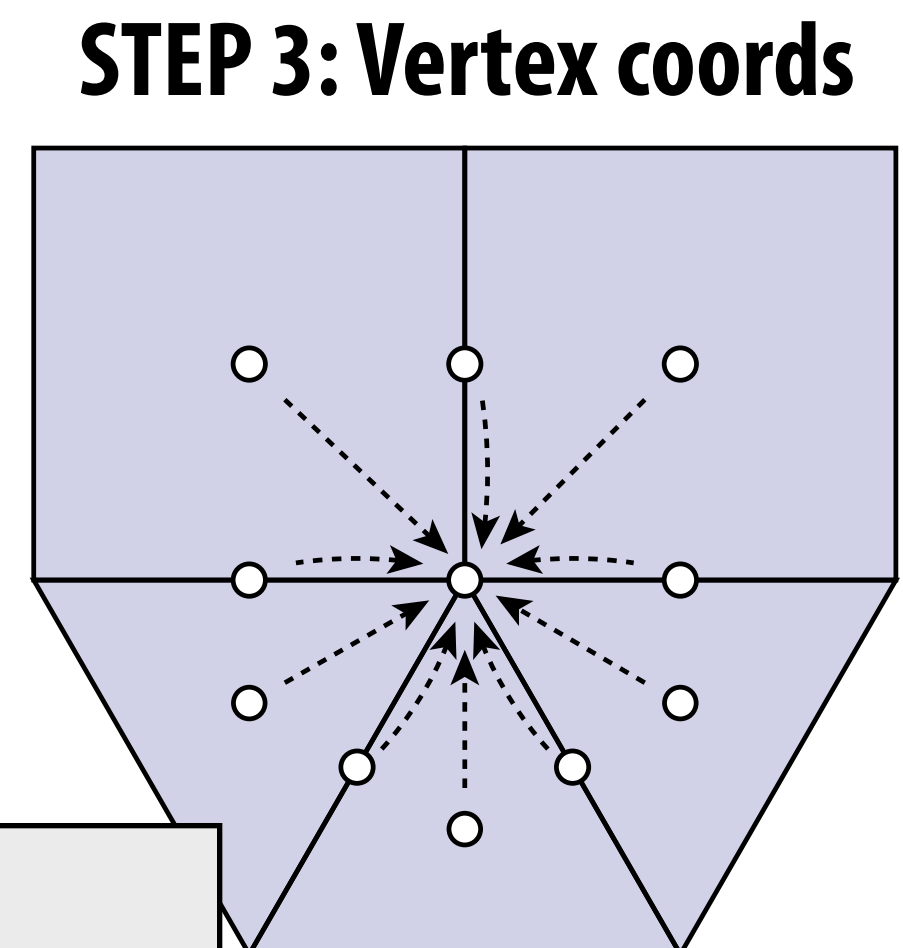
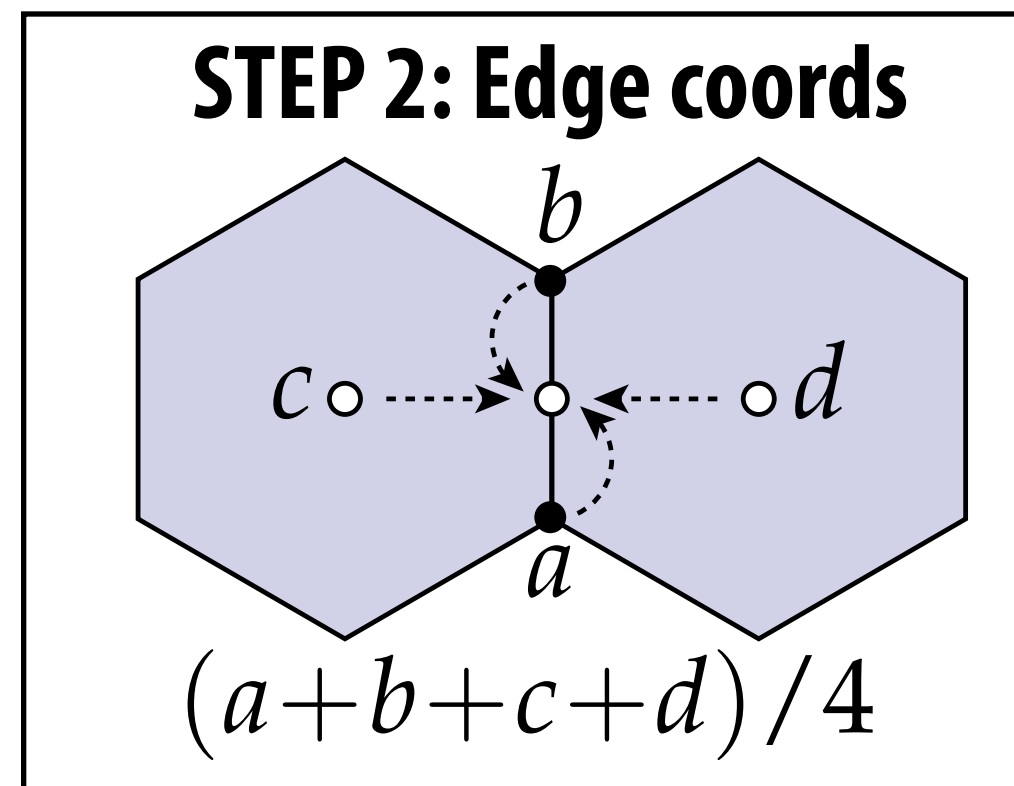
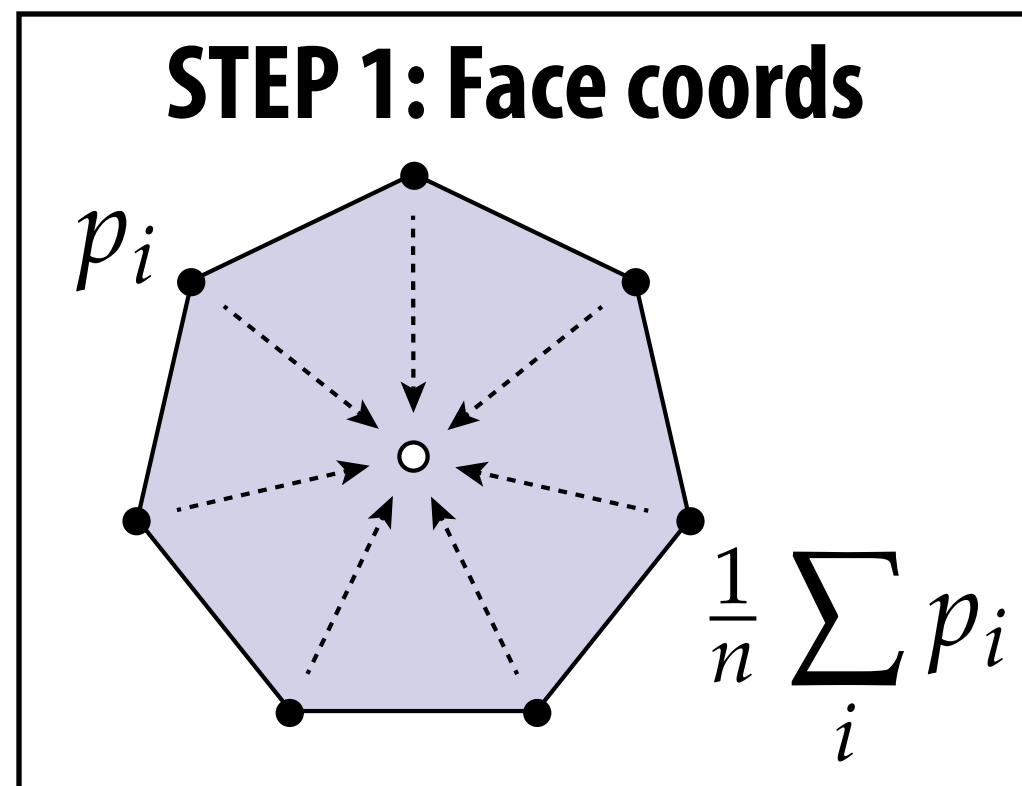


Catmull-Clark Subdivision

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



- New vertex positions are weighted combination of old ones:



New vertex coords:

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

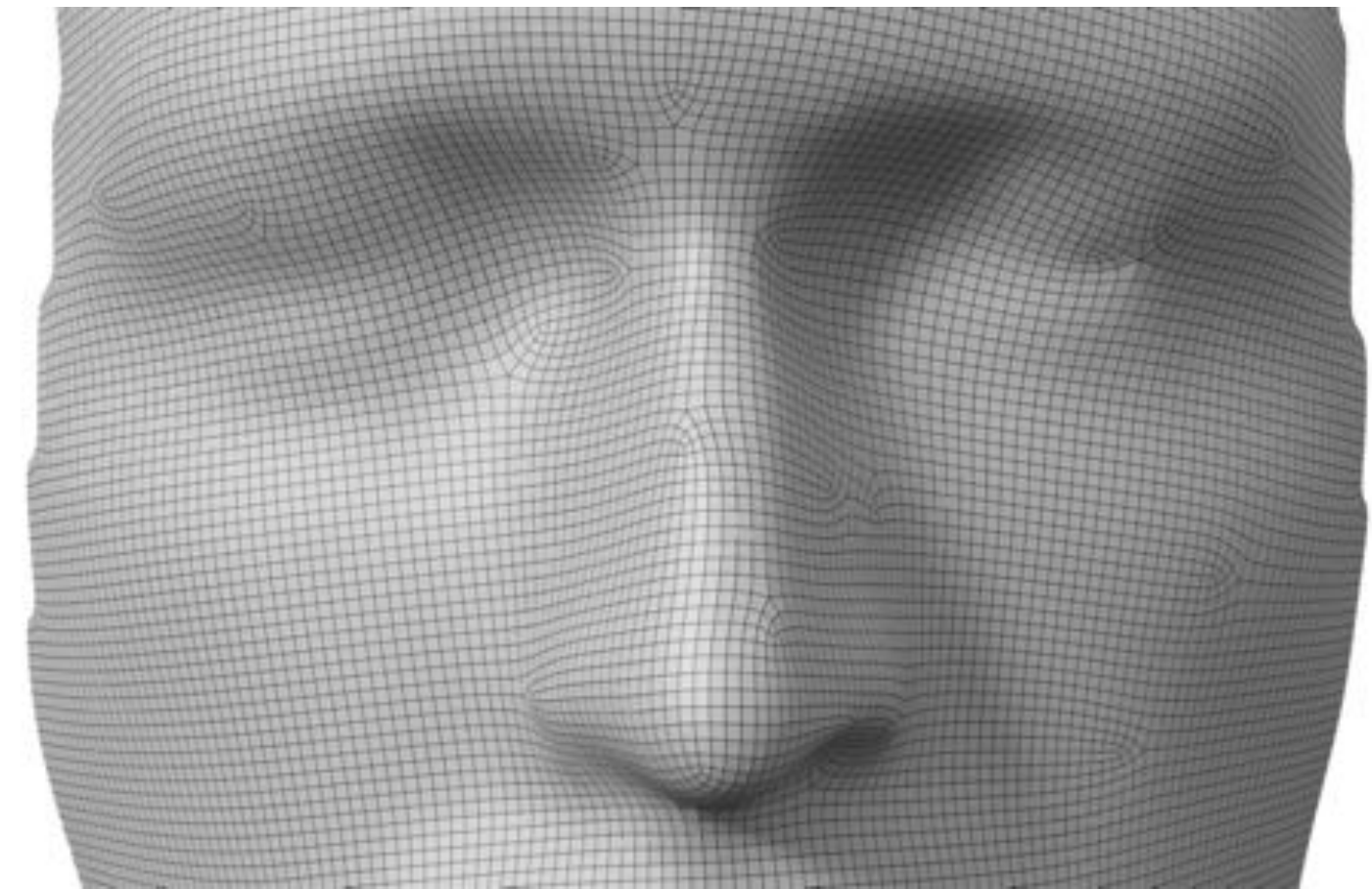
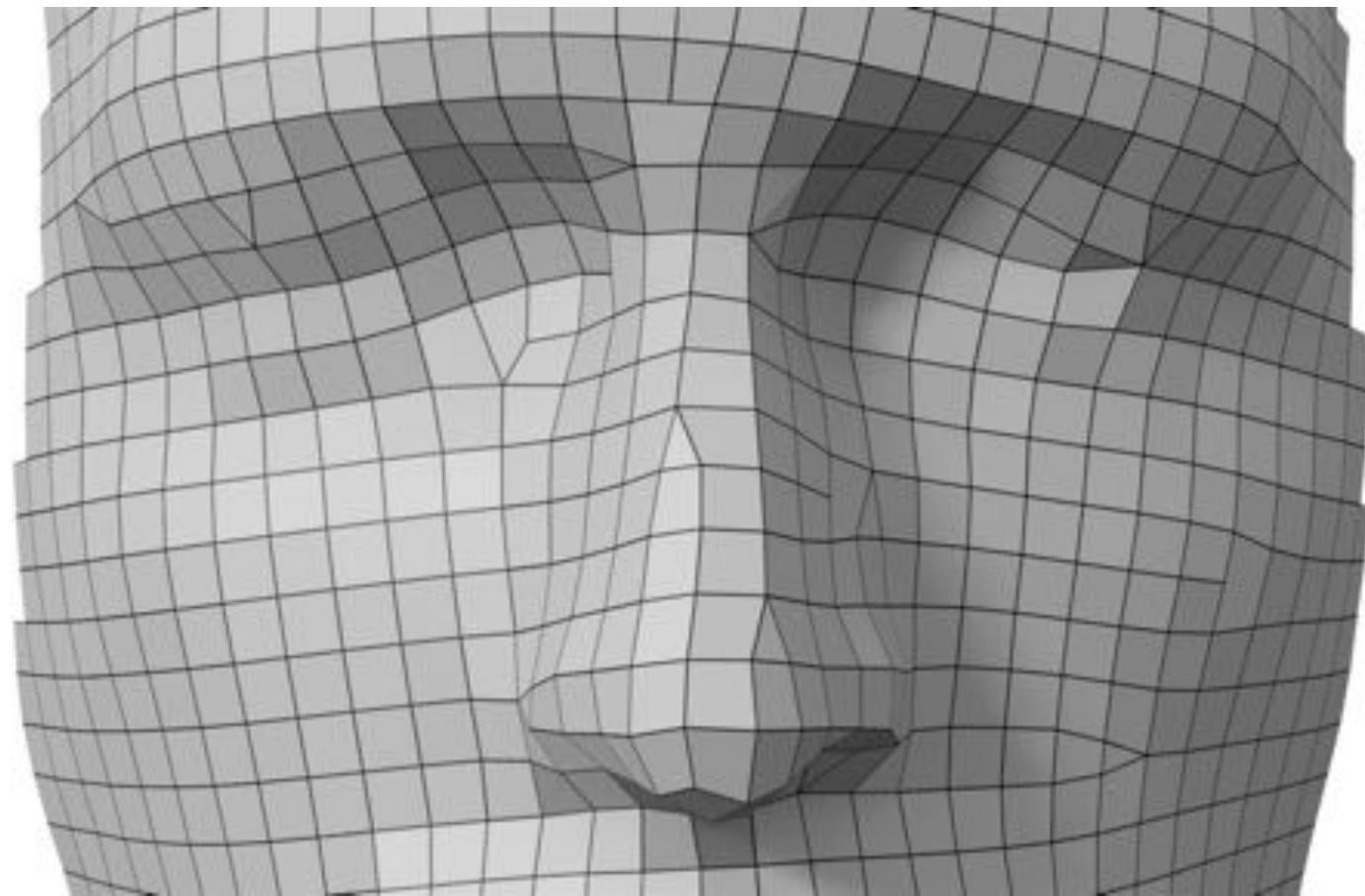
n – vertex degree

Q – average of face coords around vertex

R – average of edge coords around vertex

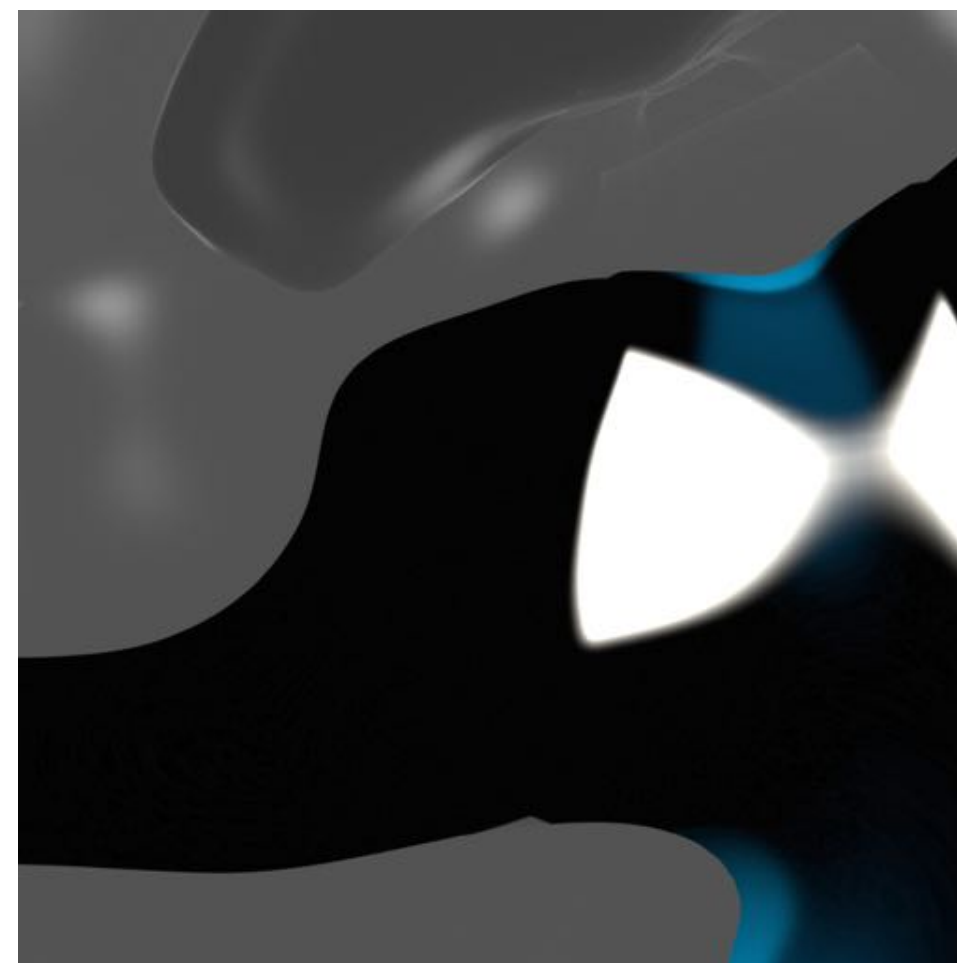
S – original vertex position

Catmull-Clark on quad mesh

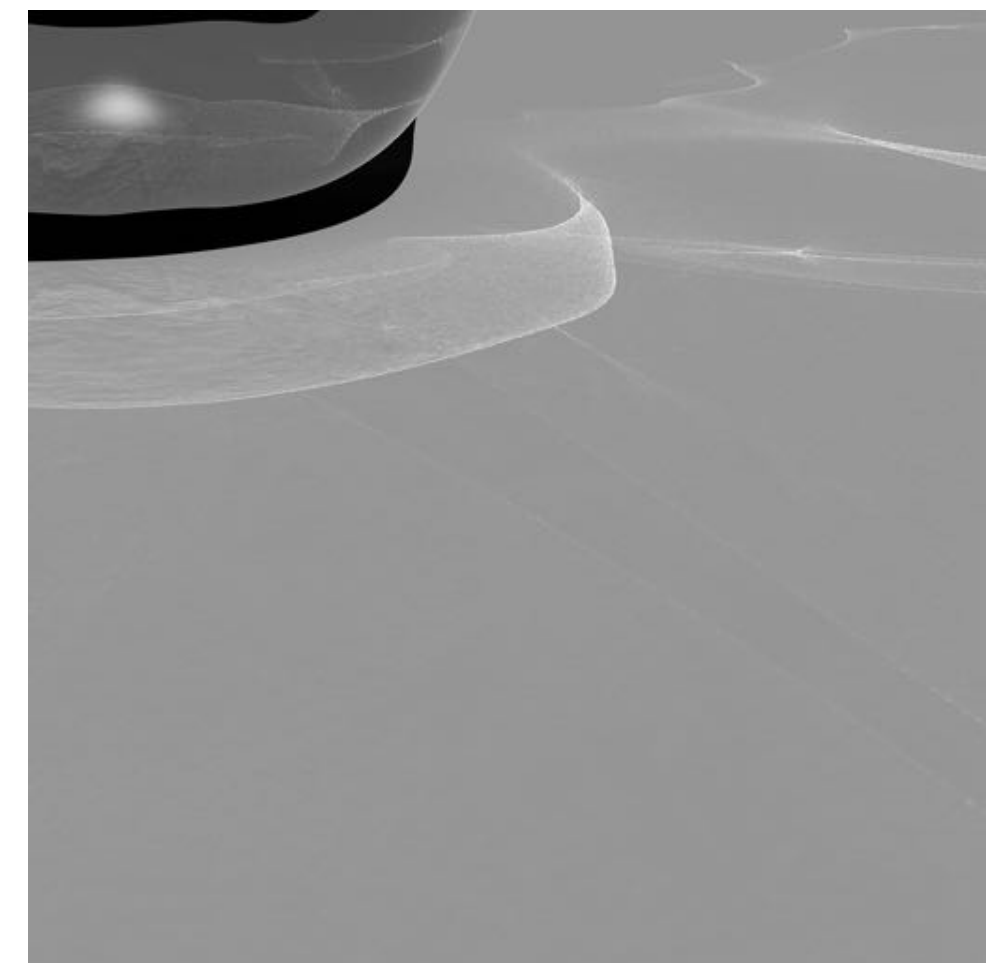


(very few irregular vertices)

Good normal approximation almost everywhere:

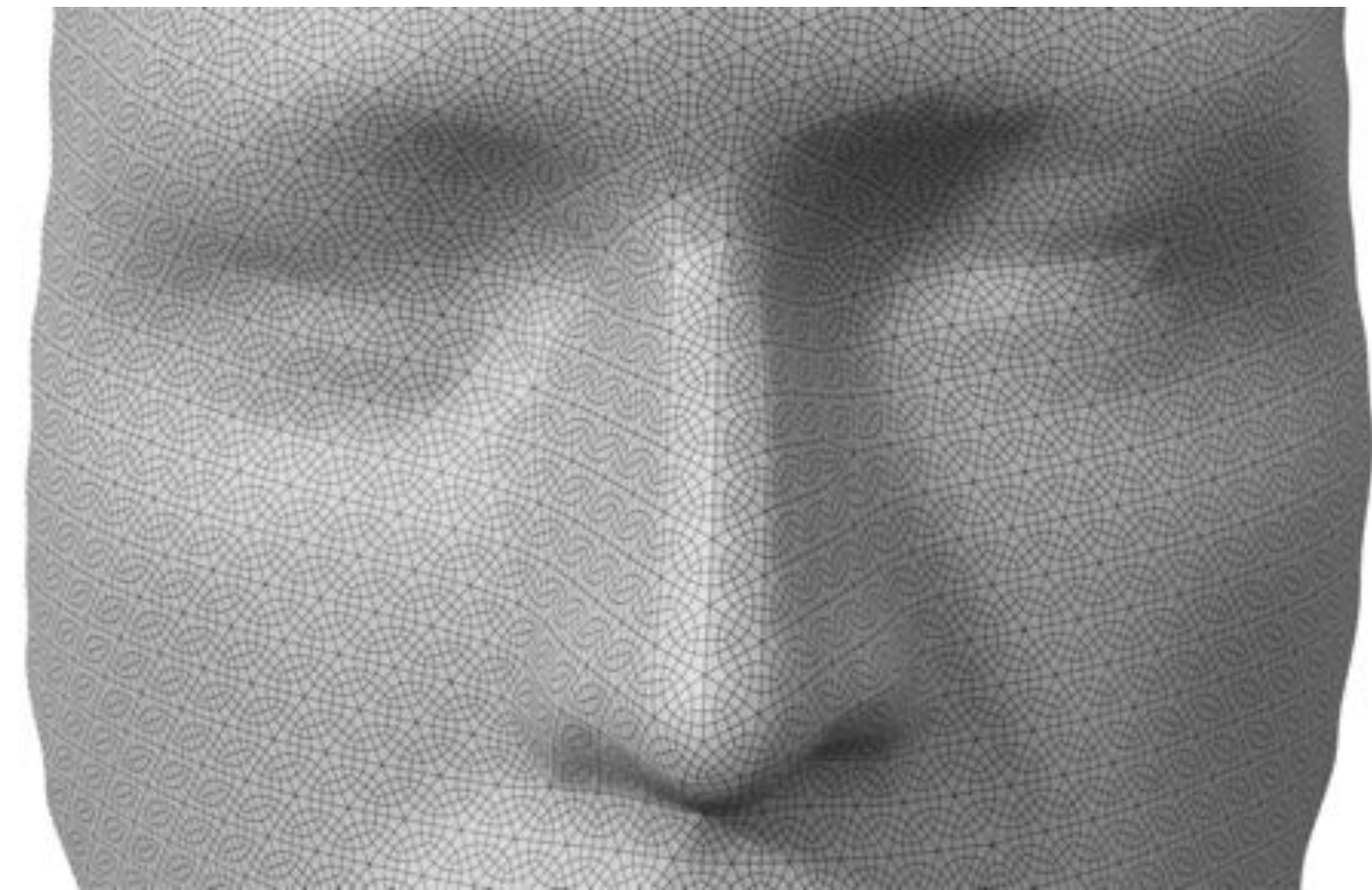
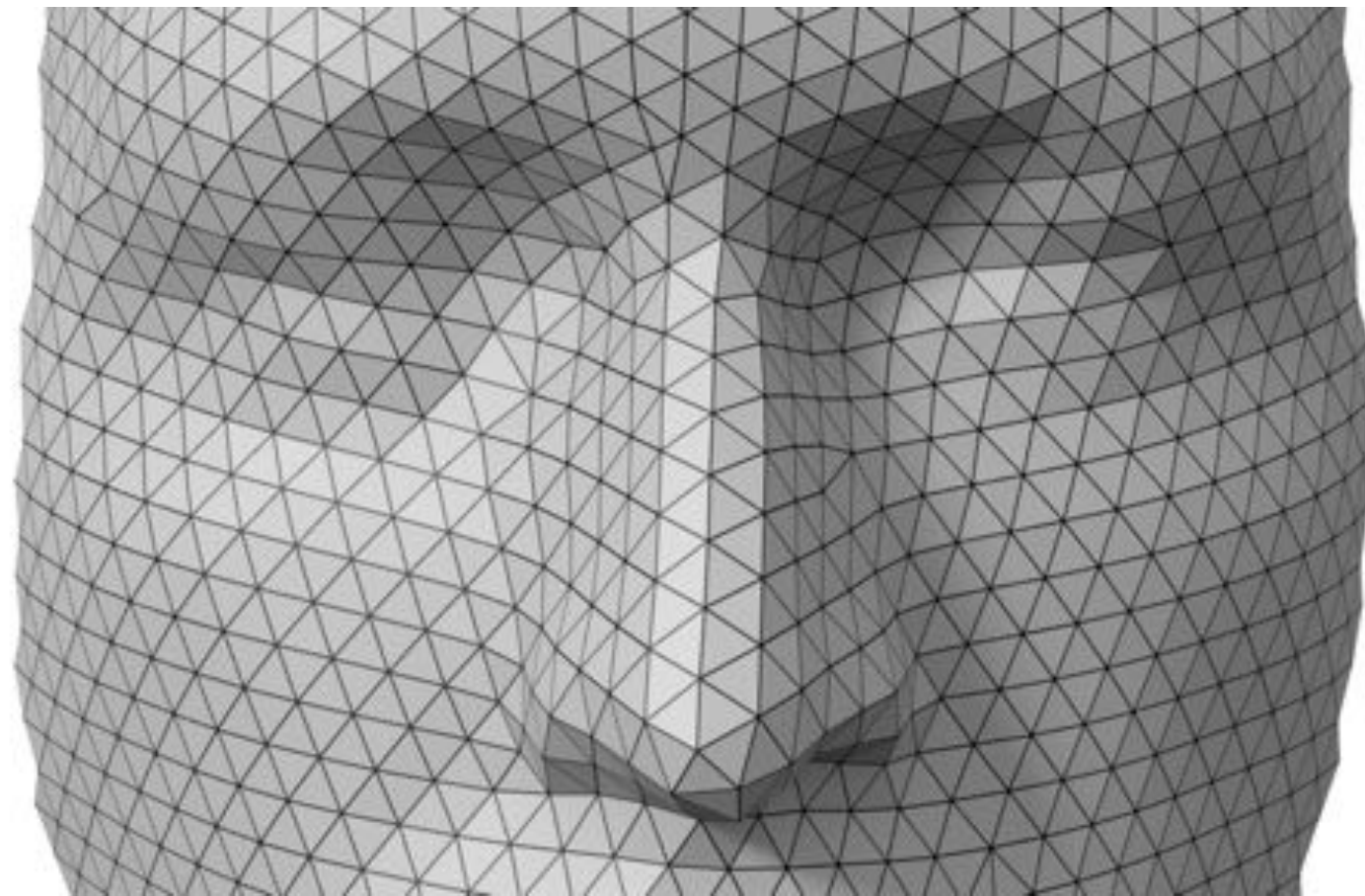


**smooth
reflection lines**



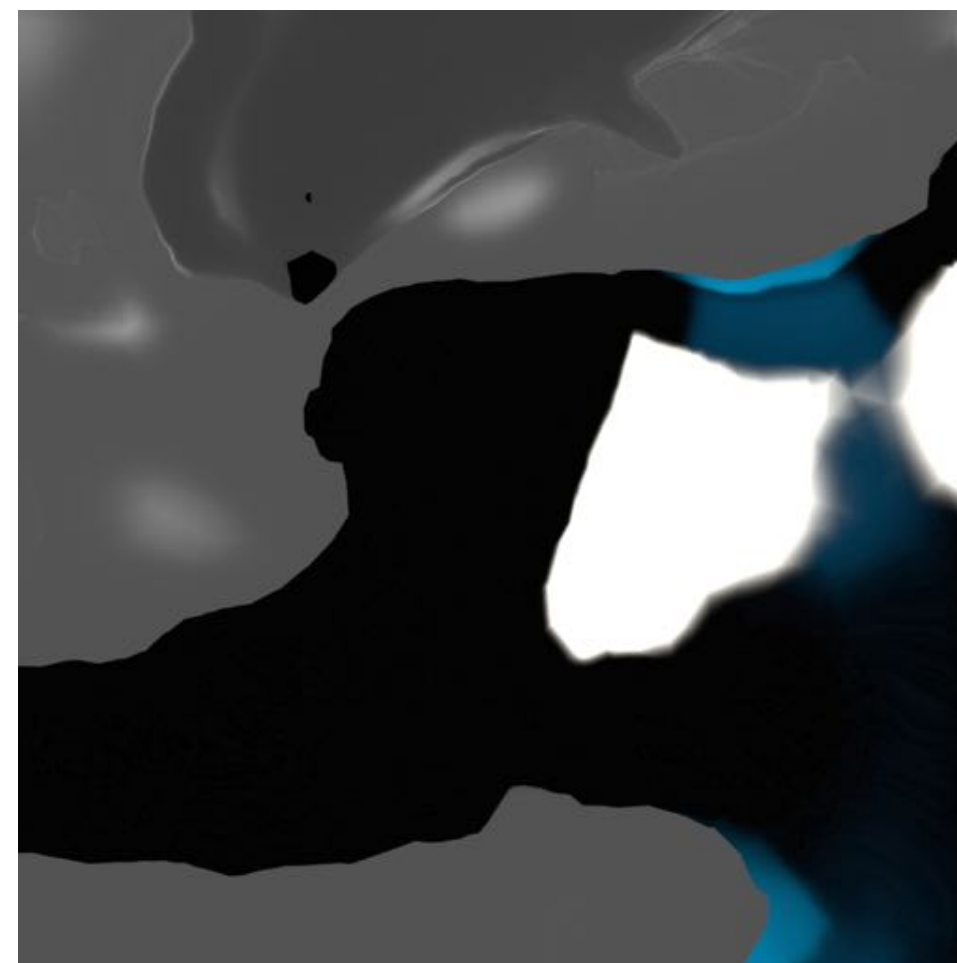
**smooth
caustics**

Catmull-Clark on triangle mesh

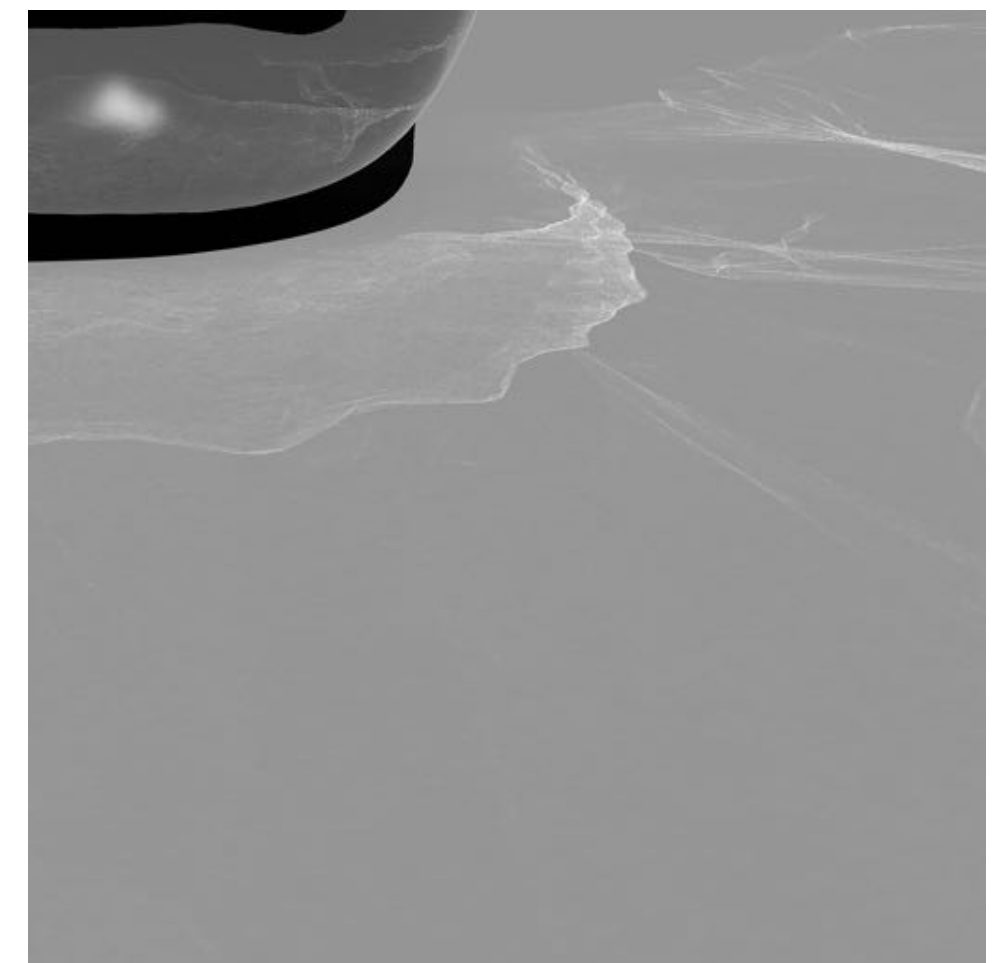


(huge number of irregular vertices!)

Poor normal approximation almost everywhere:



jagged
reflection lines

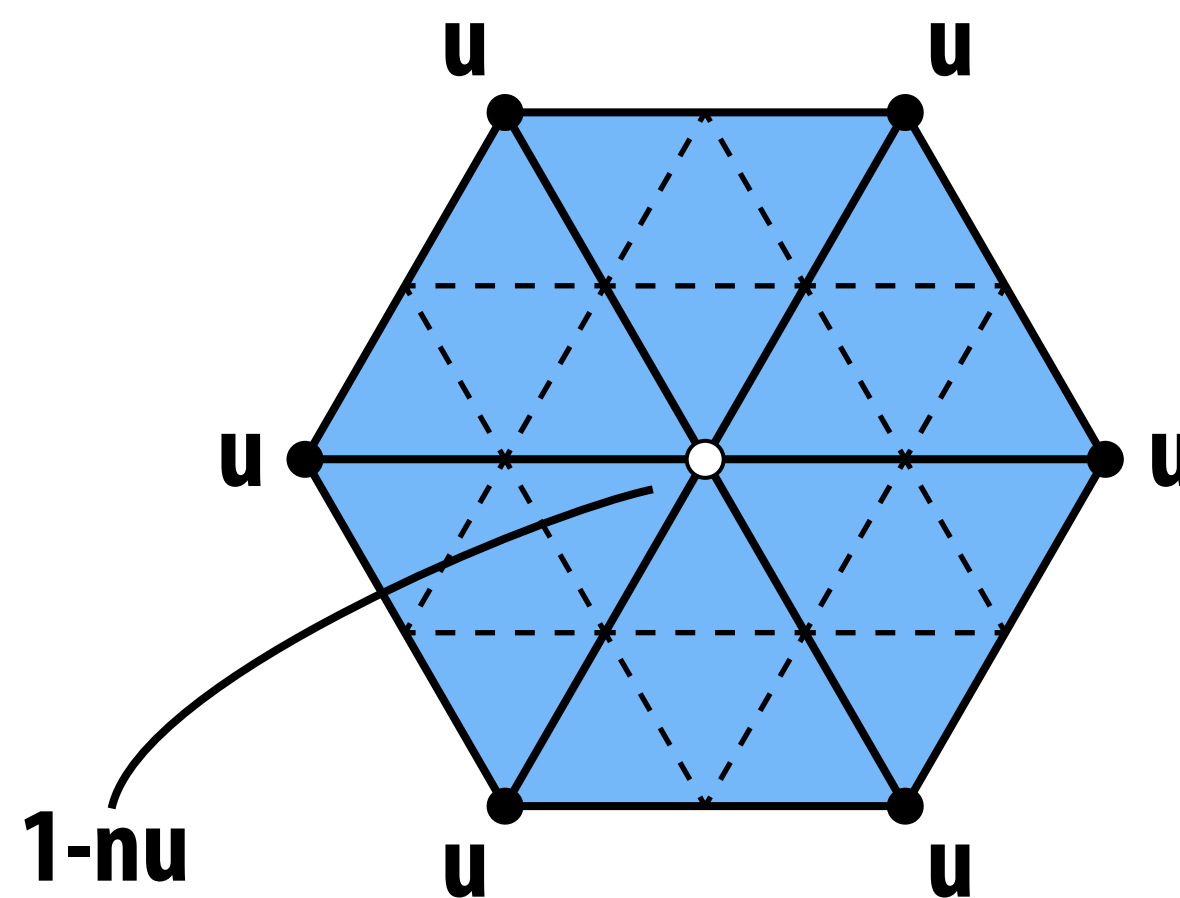
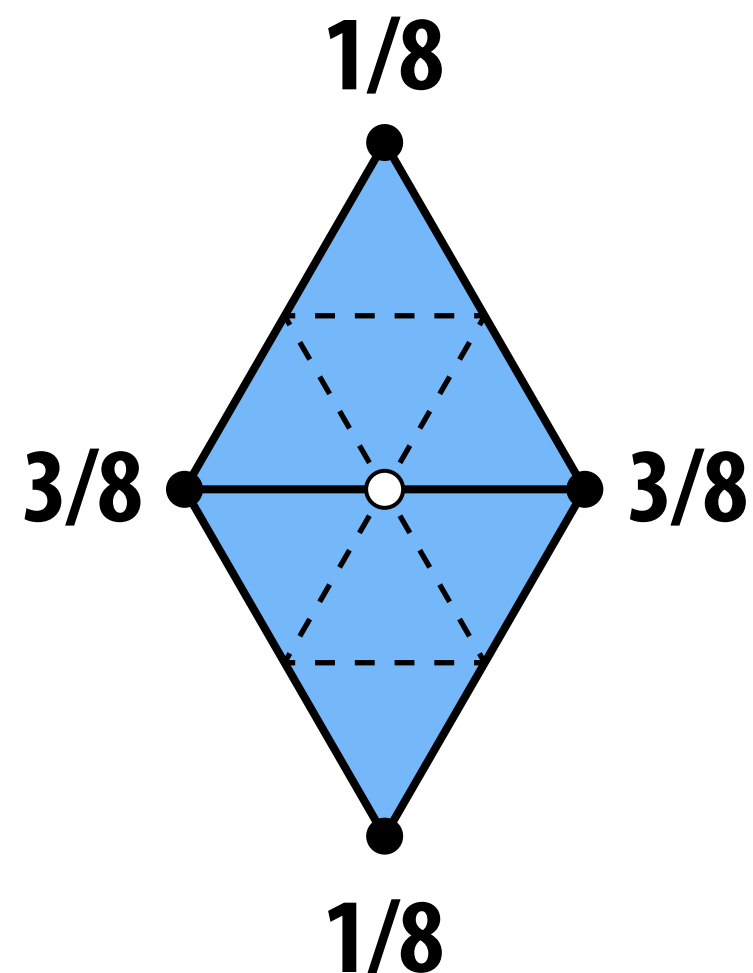
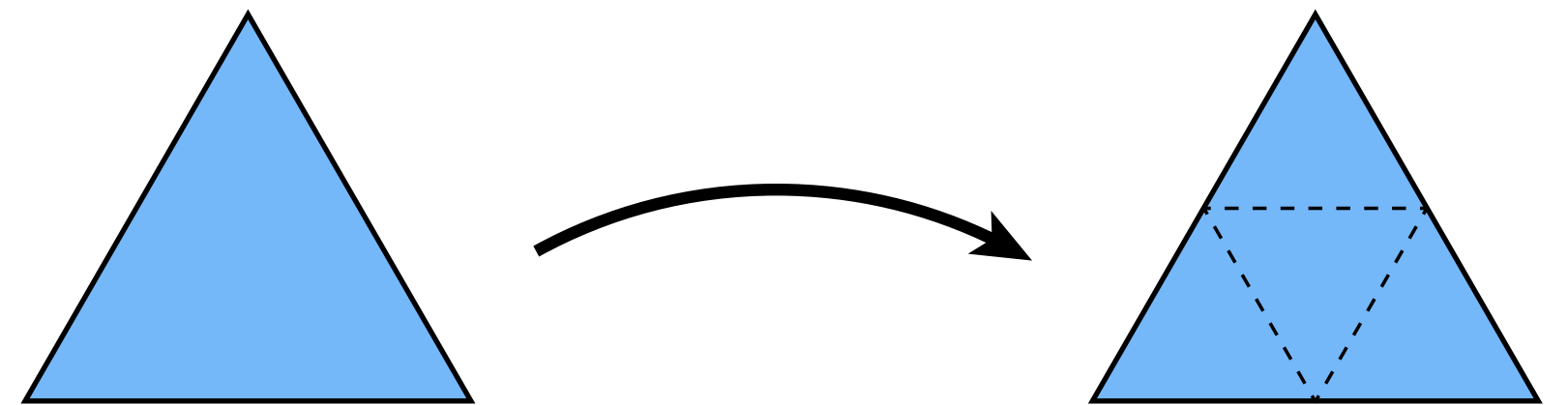


jagged
caustics

ALIASING!

Loop Subdivision

- Alternative subdivision scheme for triangle meshes
- Curvature is continuous away from irregular vertices (“C²”)
- Algorithm:
 - Split each triangle into four
 - Assign new vertex positions according to weights:

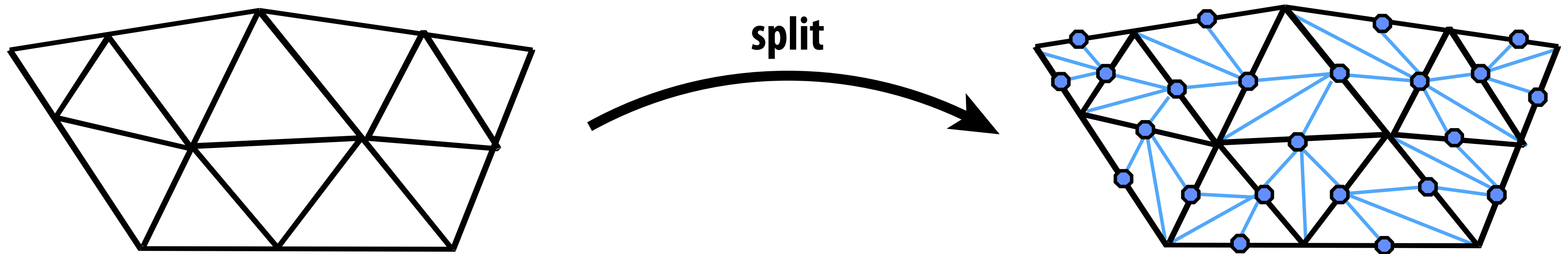


n : vertex degree

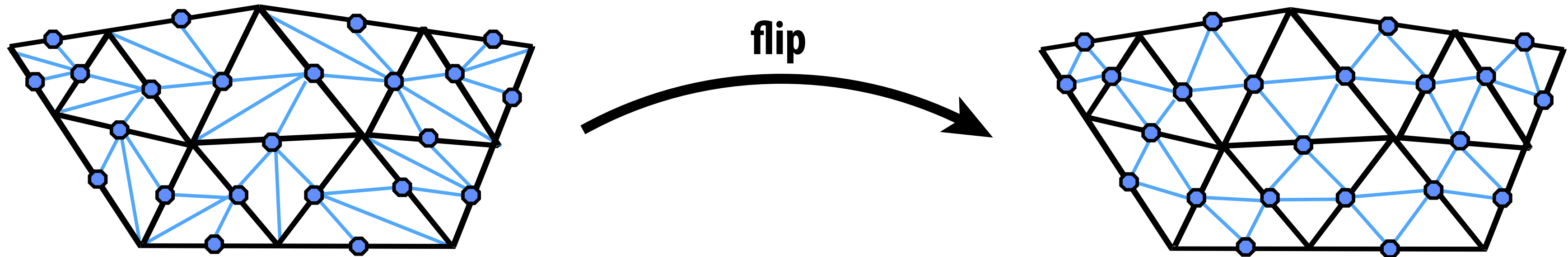
u : $3/16$ if $n=3$, $3/(8n)$ otherwise

Loop Subdivision via Edge Operations

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



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

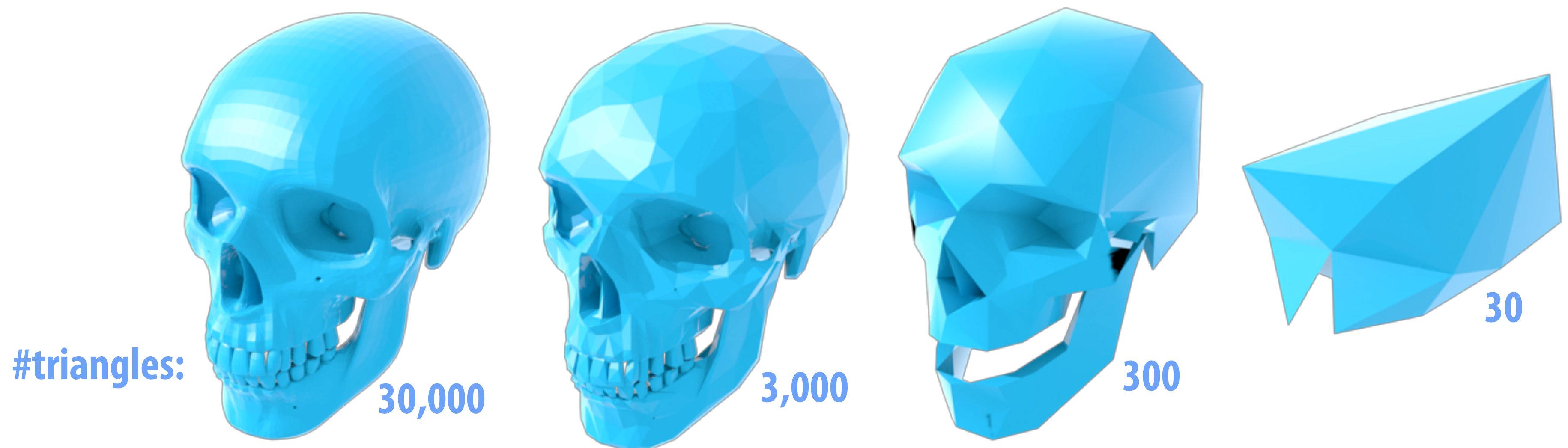


(Don't forget to update vertex positions!)

What if we want fewer triangles?

Simplification via Edge Collapse

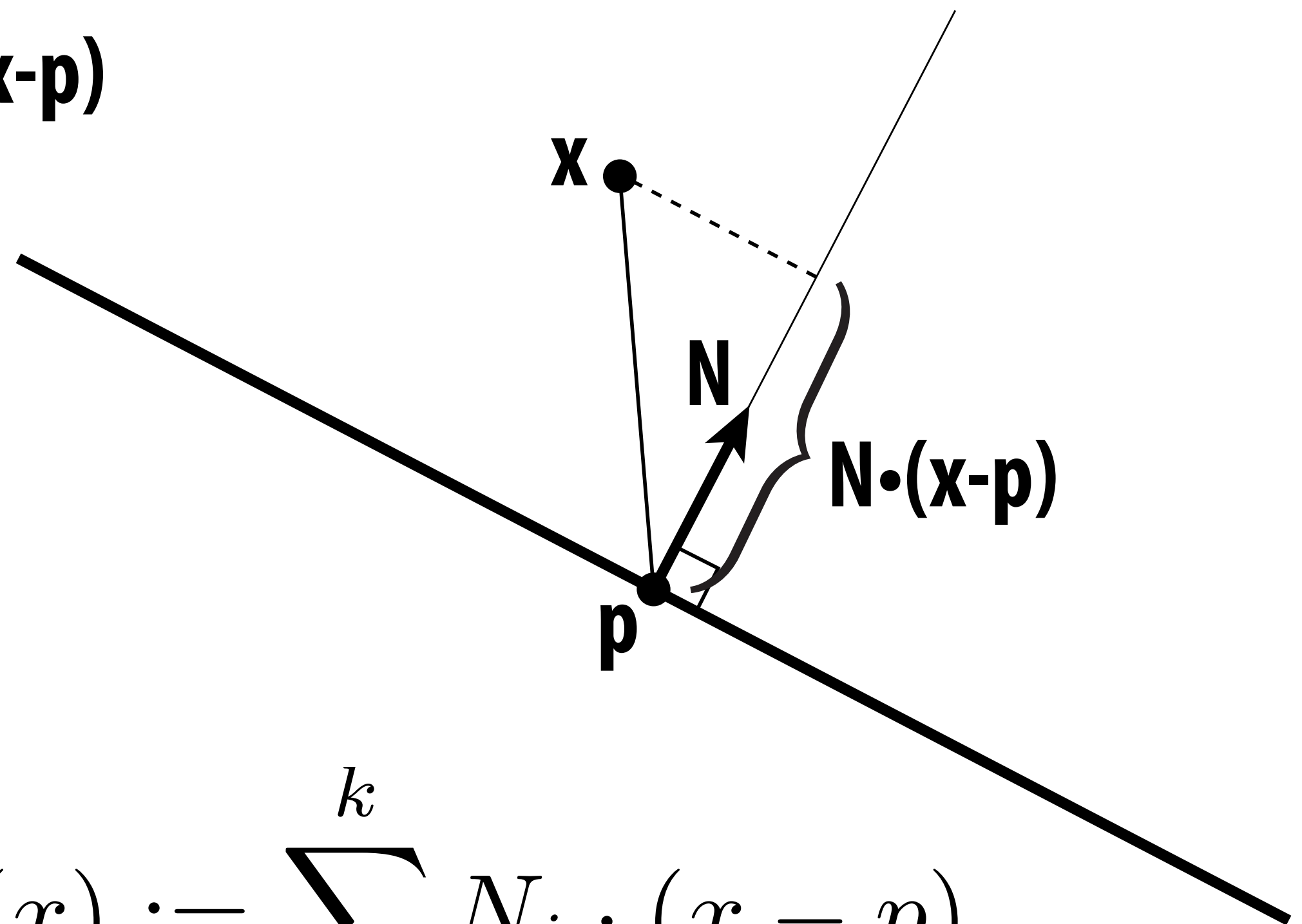
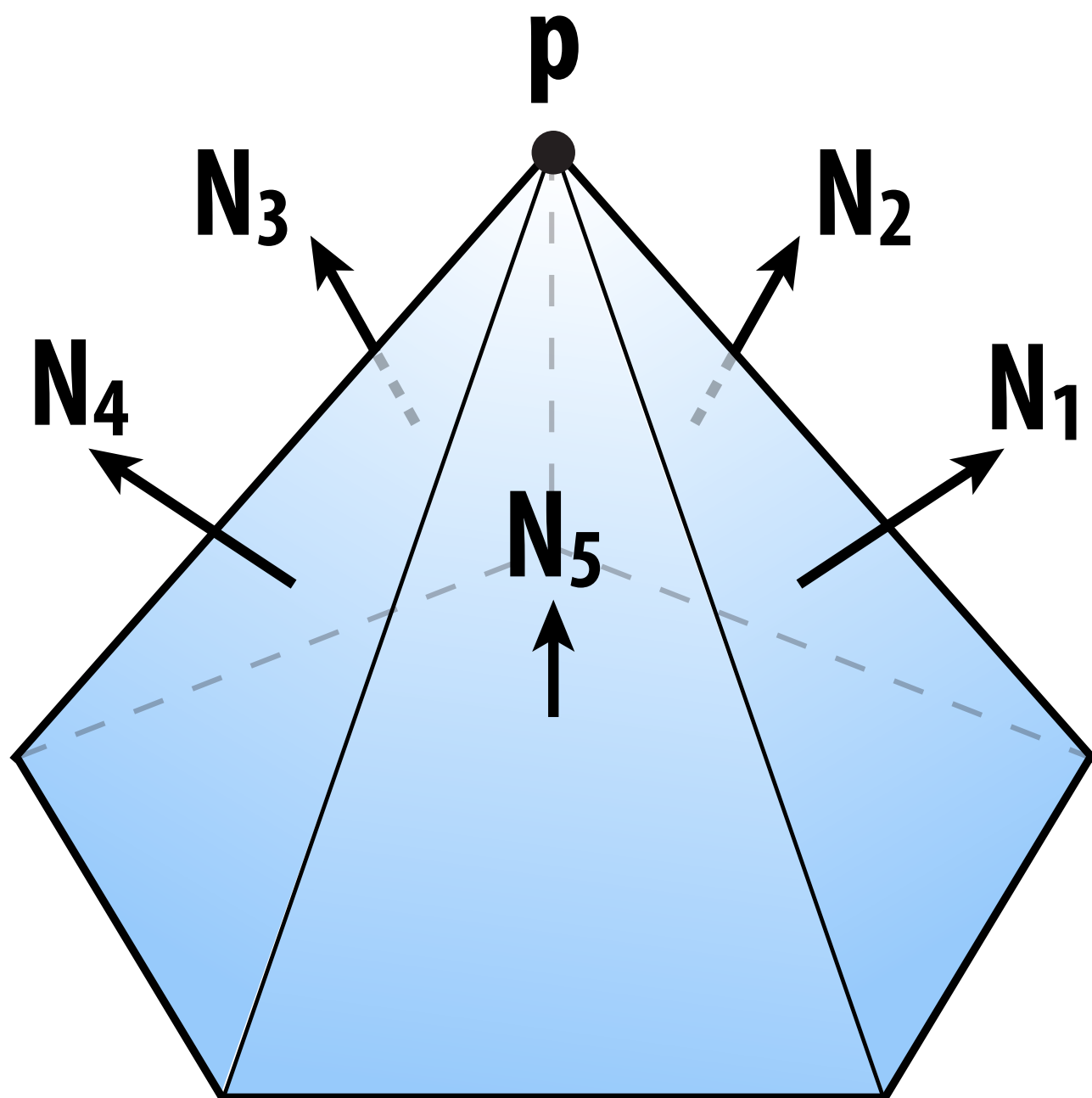
- One popular scheme: iteratively collapse edges
- Greedy algorithm:
 - assign each edge a cost
 - collapse edge with least cost
 - repeat until target number of elements is reached
- Particularly effective cost function: quadric error metric*



*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 $u \cdot v = ax+by+cz+d$

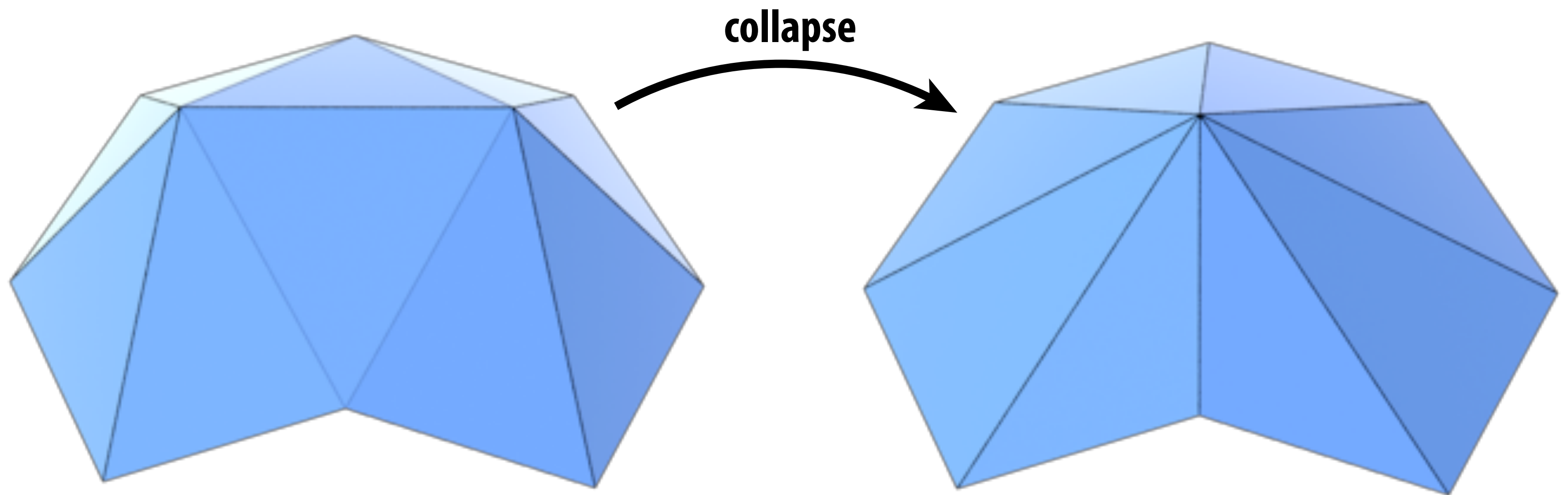
■ Squared distance is $(u^T v)^2 = u^T (v v^T) u =: u^T K u$

■ 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

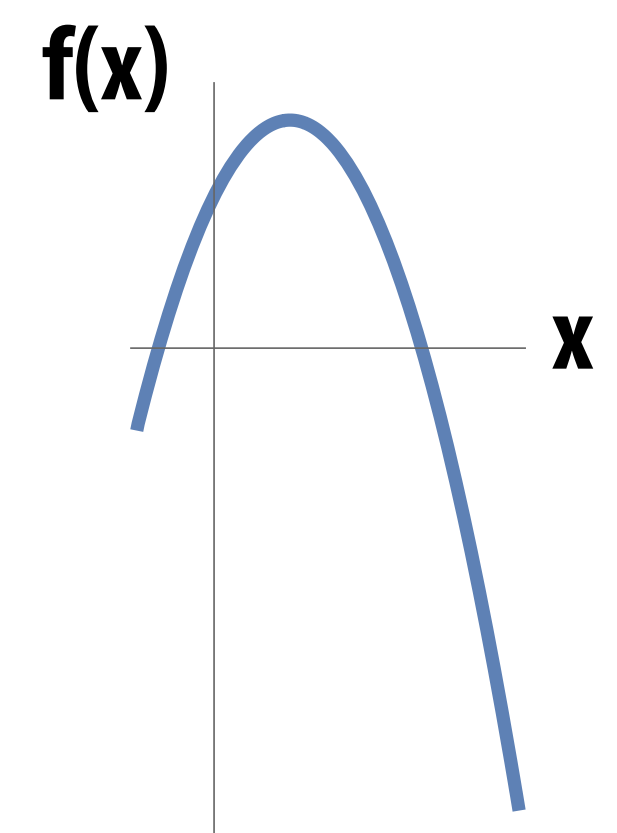
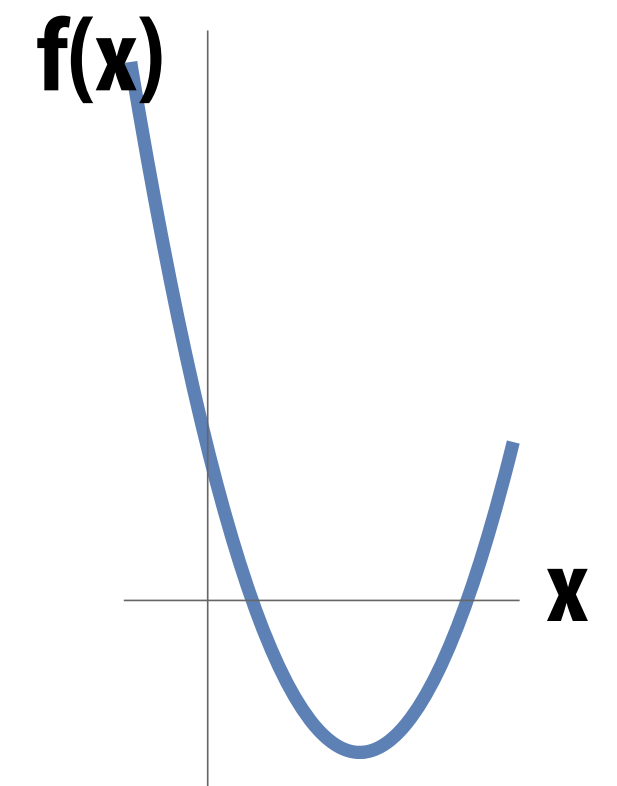
- Suppose I give you a function $f(x) = ax^2 + bx + c$
- Q: What does the graph of this function look like?
- Could also look like this!
- Q: How do we find the minimum?
- A: Look for the point where the function isn't changing (if we look "up close")
- I.e., find the point where the derivative vanishes

$$f'(x) = 0$$

$$2ax + b = 0$$

$$x = -b/2a$$

(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):

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

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

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

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

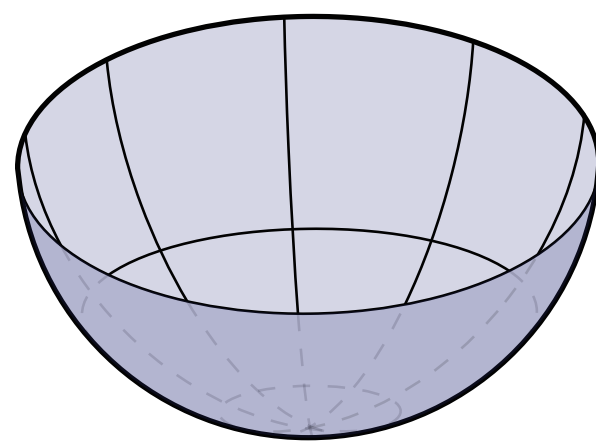
(Can you show this is true, at least in 2D?)

Positive Definite Quadratic Form

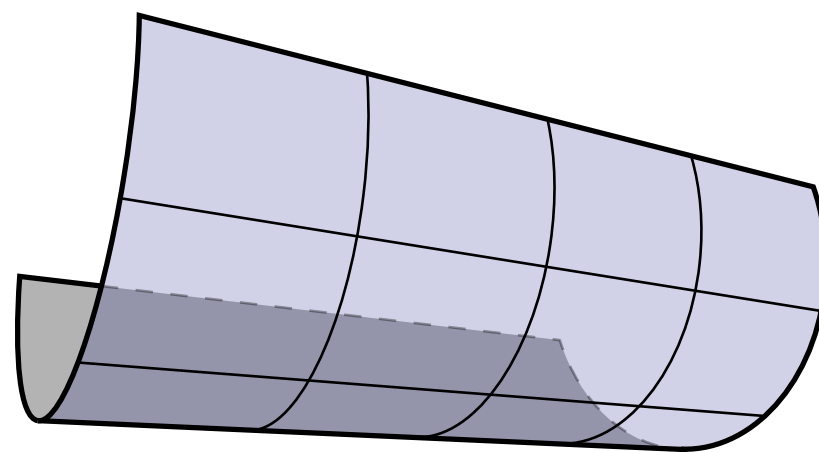
- Just like our 1D parabola, critical point is not always a min!
- Q: In 2D, 3D, nD, when do we get a minimum?
- A: When matrix A is positive-definite:

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

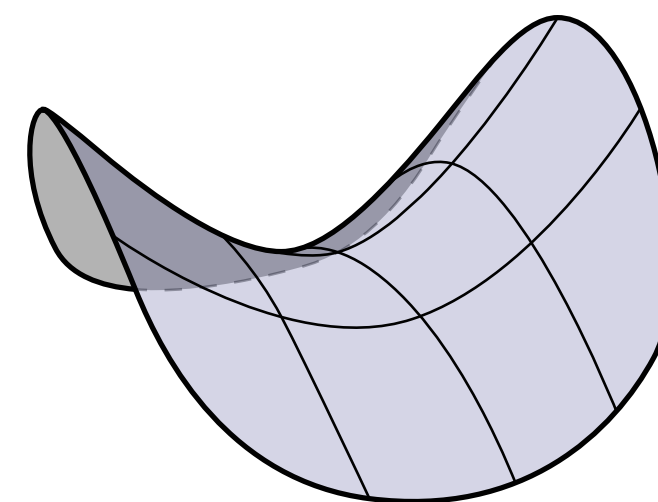
- 1D: Must have $xax = ax^2 > 0$. In other words: a is positive!
- 2D: Graph of function looks like a “bowl”:



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:

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

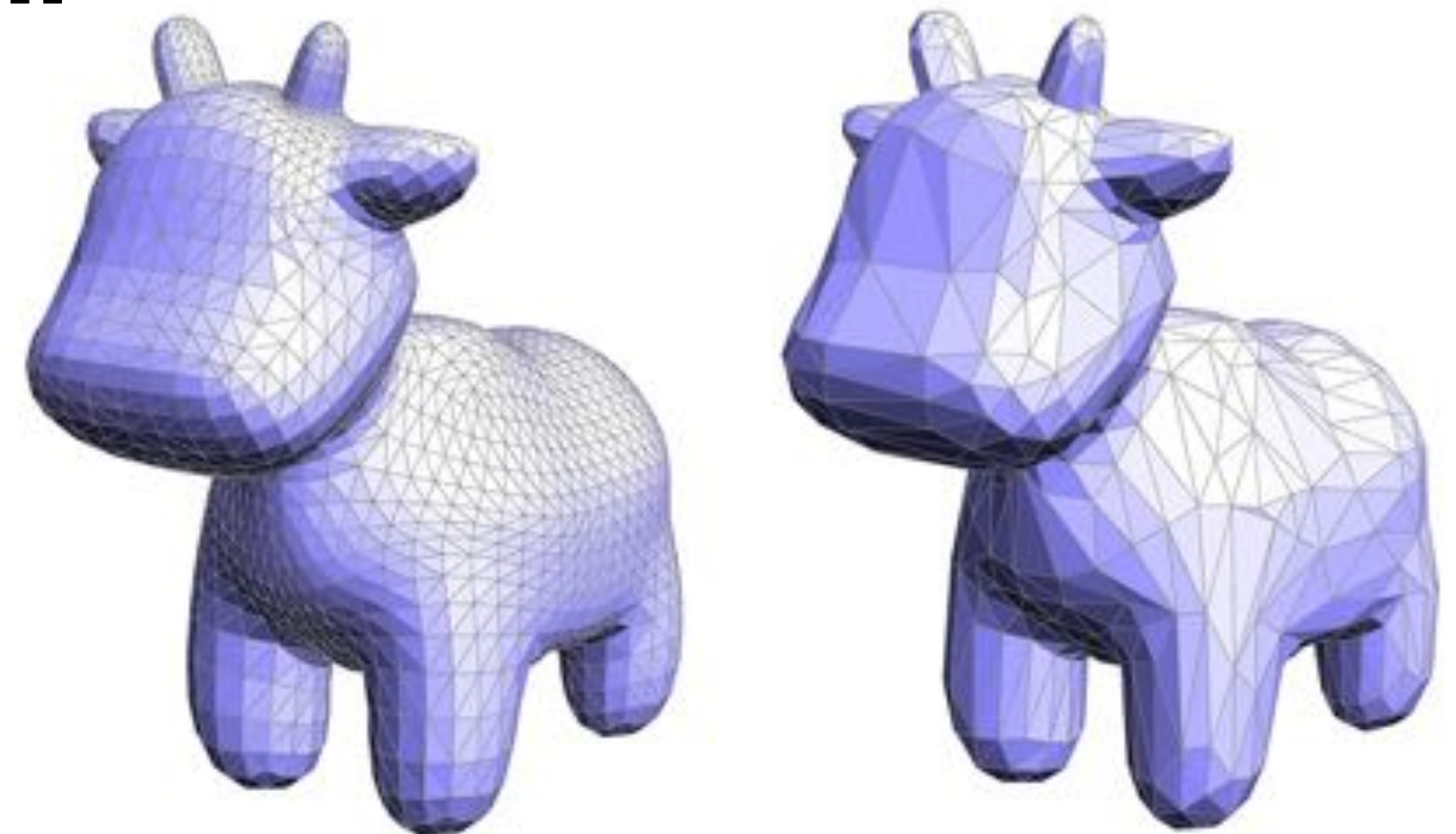
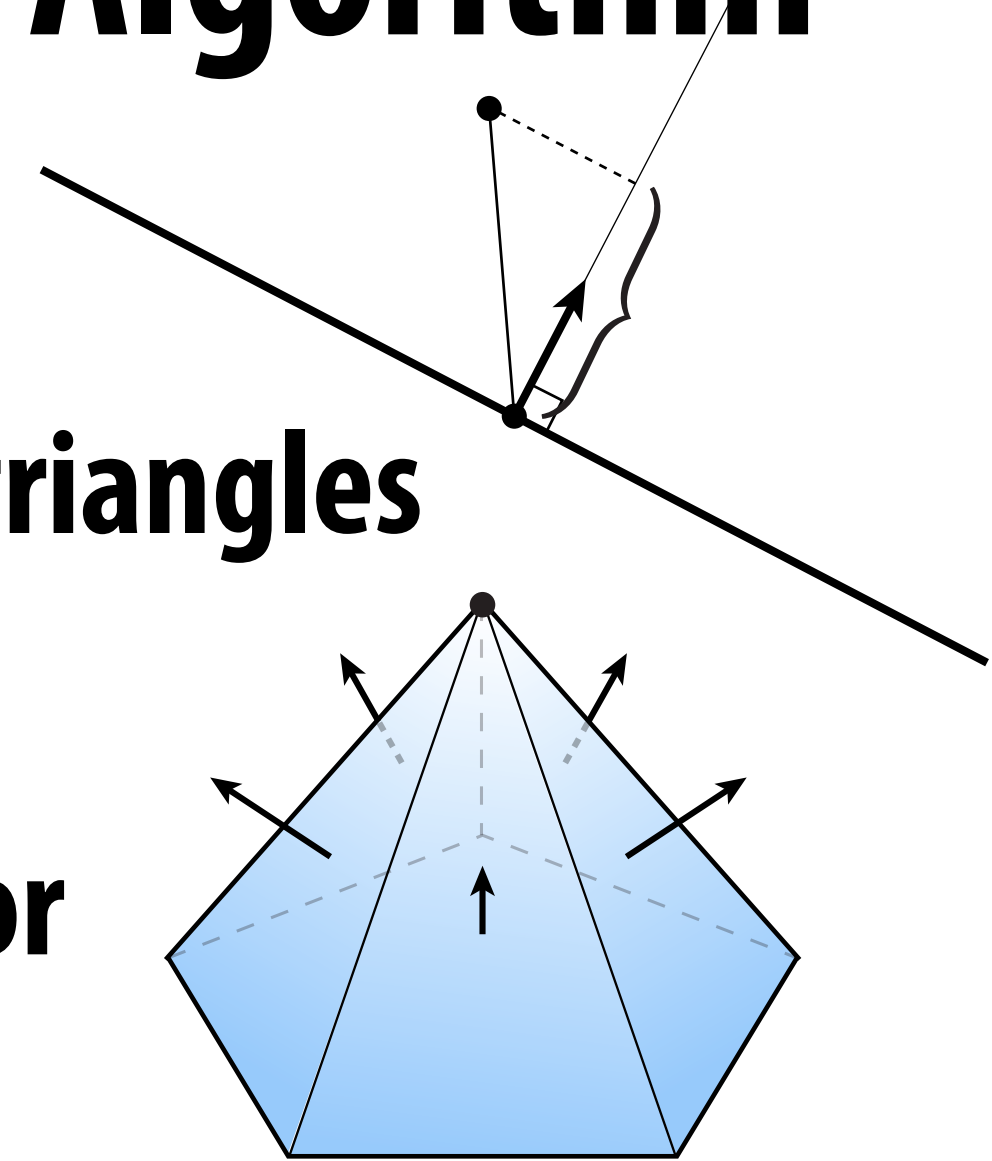
- Now we have a quadratic form in the 3D position \mathbf{x} .
- Can minimize as before:

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

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

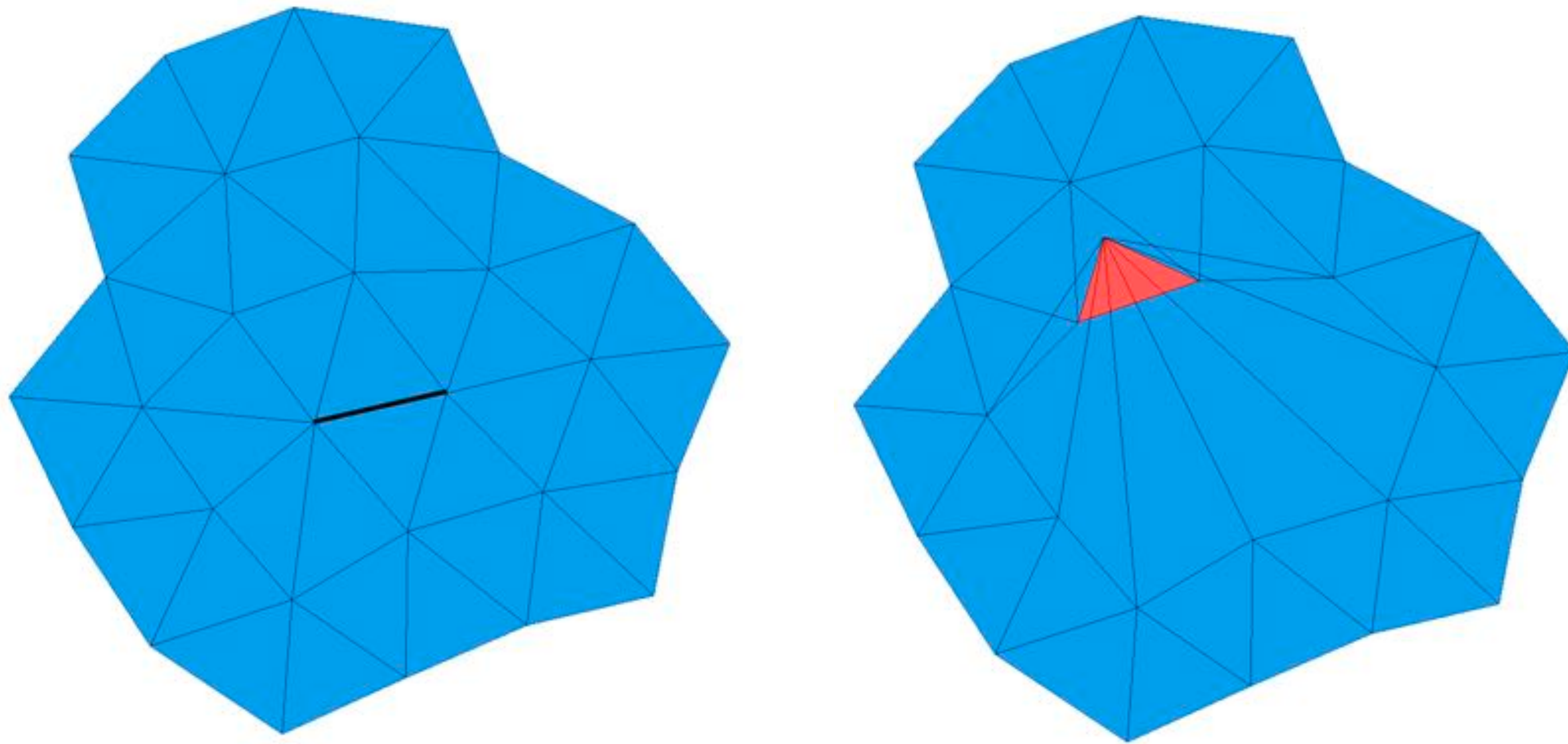
Quadric Error Simplification: Final Algorithm

- 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):

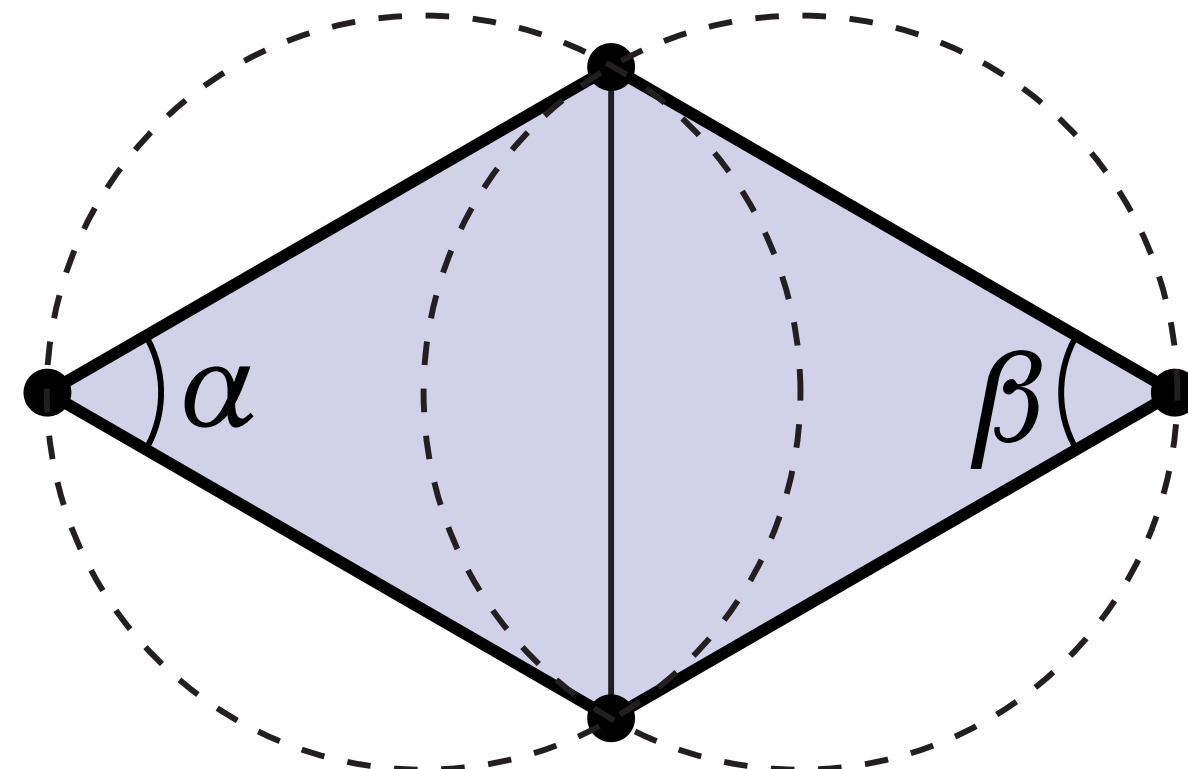
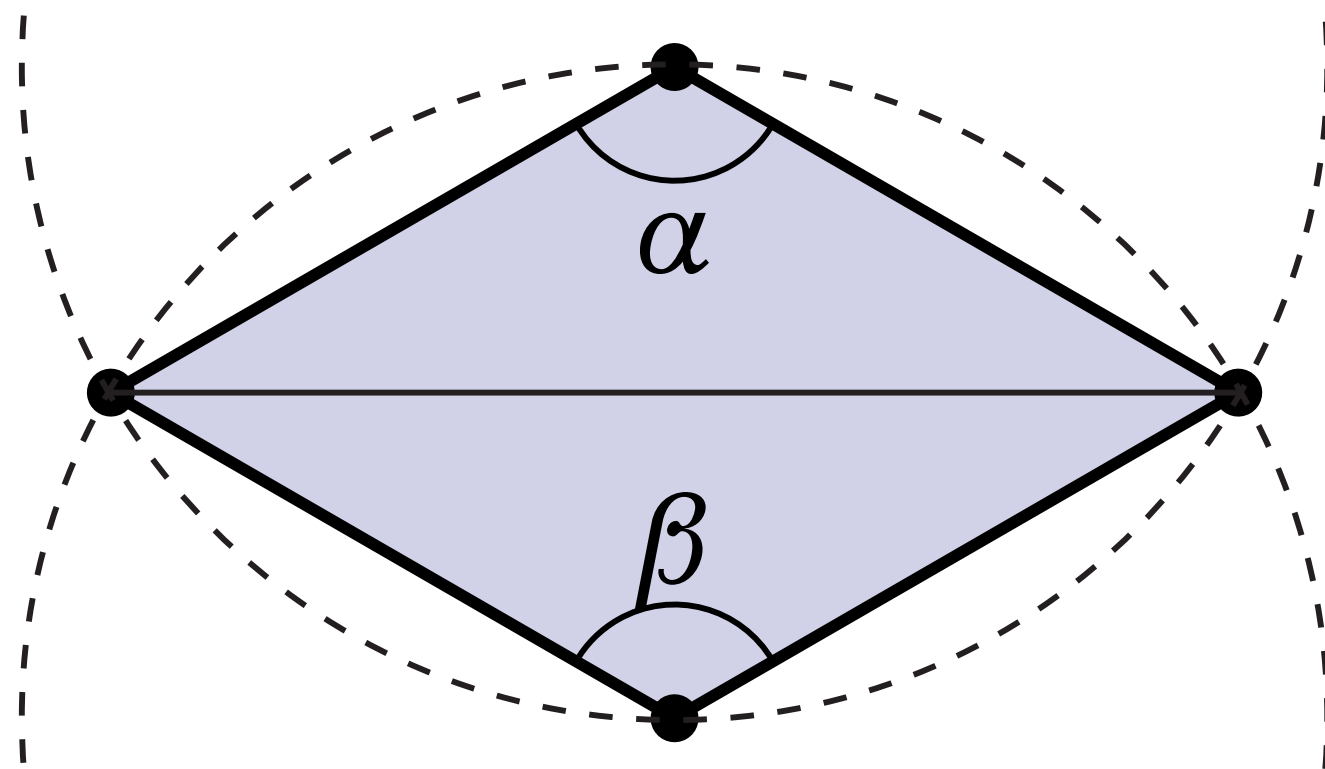


- Easy solution: check dot product between normals across edge
- If negative, don't collapse this edge!

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

How do we make a mesh “more Delaunay”?

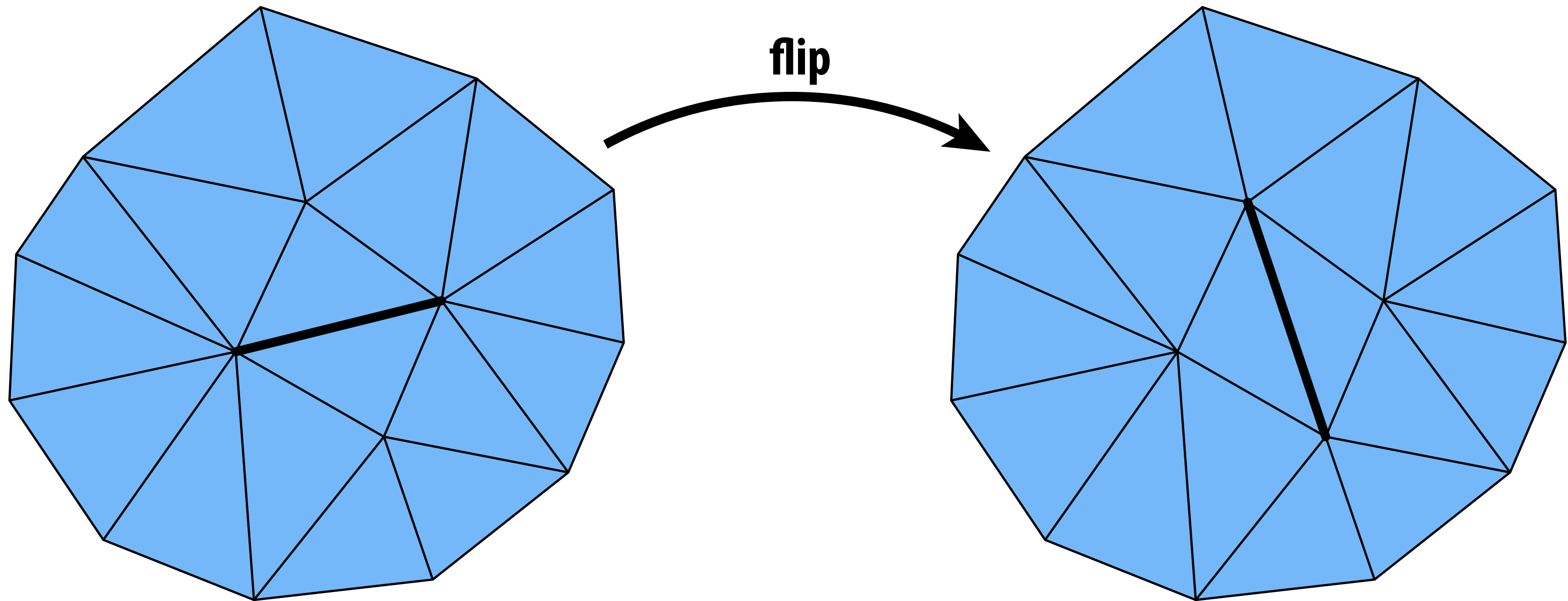
- Already have a good tool: edge flips!
- If $\alpha + \beta > \pi$, flip it!



- **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?

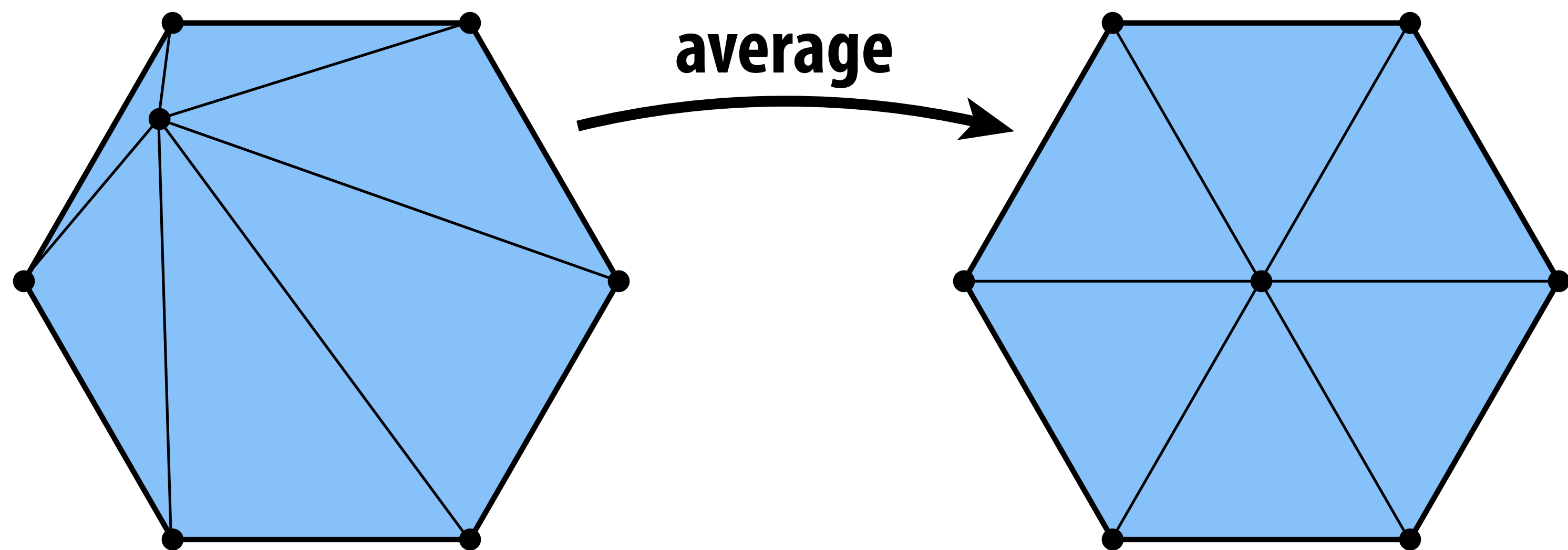
- Same tool: edge flips!
- If total deviation from degree-6 gets smaller, flip it!



- **FACT:** average vertex degree is 6 as number of elements increases
- 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>

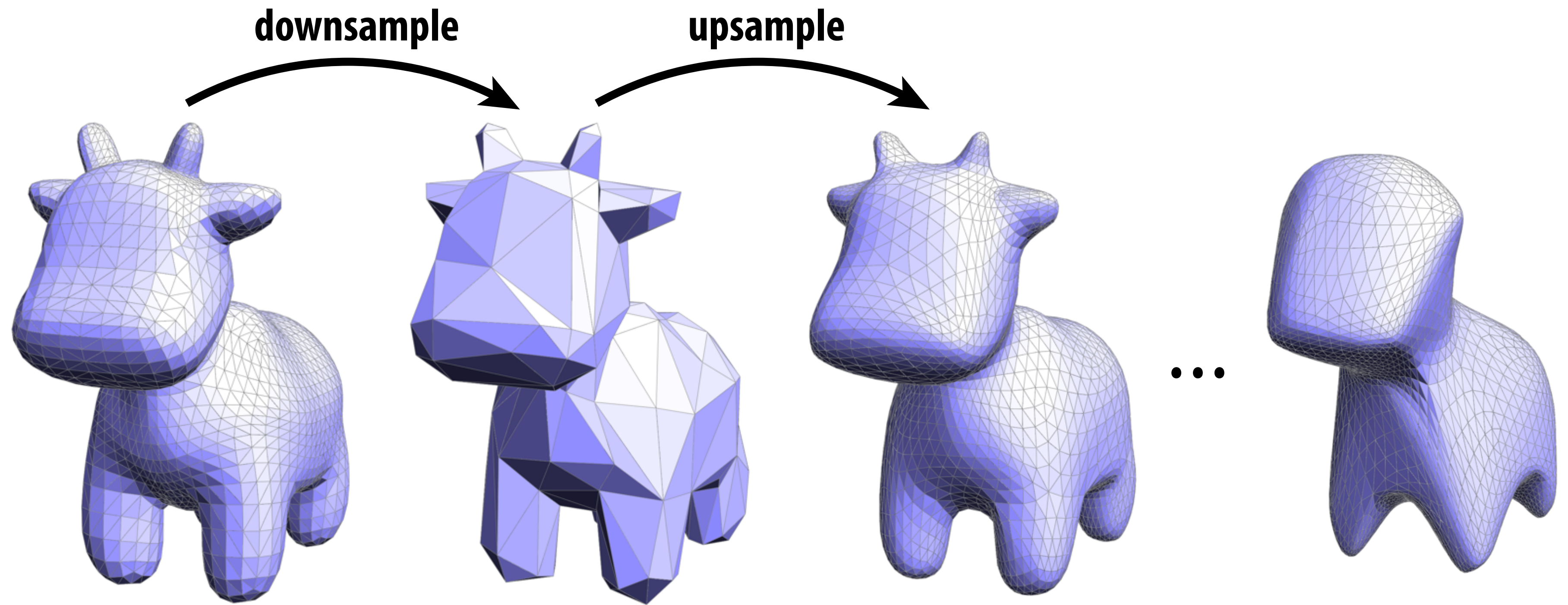
Isotropic Remeshing Algorithm

- 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?**

Next Time: Geometric Queries

- **Q: Given a point, in space, how do we find the closest point on a surface? Are we inside or outside the surface? How do we find intersection of two triangles? Etc.**
- **Q: Do implicit/explicit representations make such tasks easier?**
- **Q: What's the cost of the naïve algorithm, and how do we accelerate such queries for large meshes?**
- **So many questions!**

