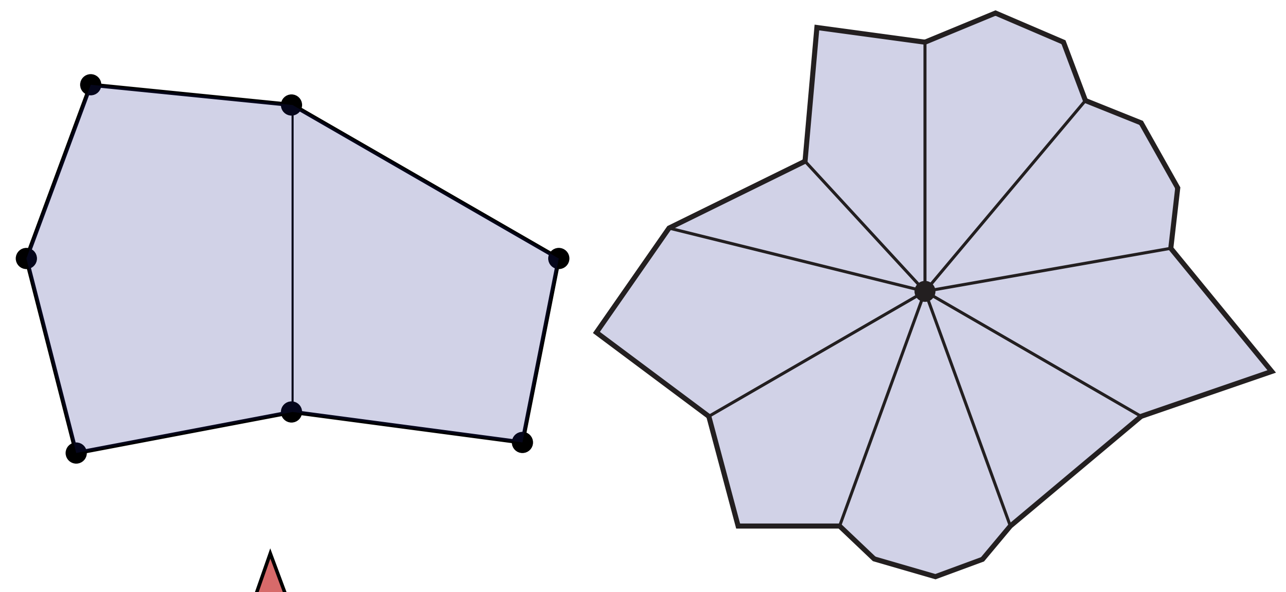
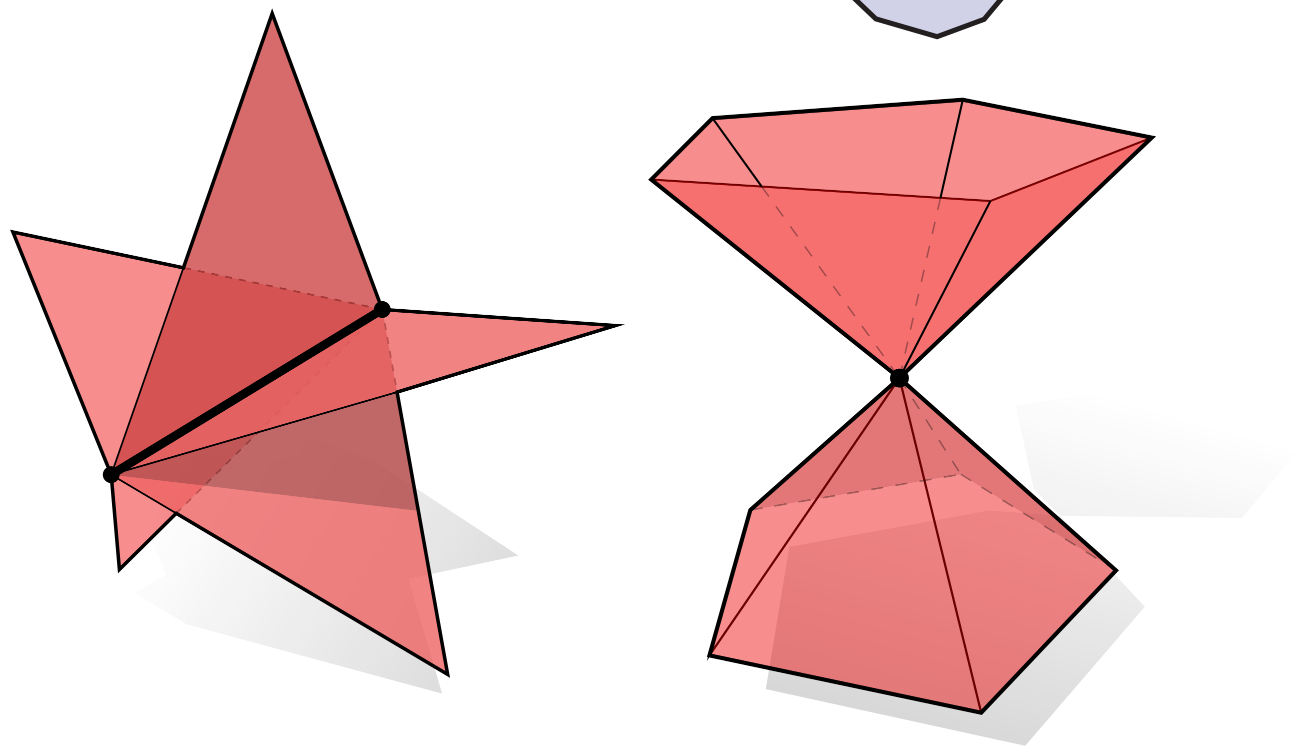
Digital Geometry Processing

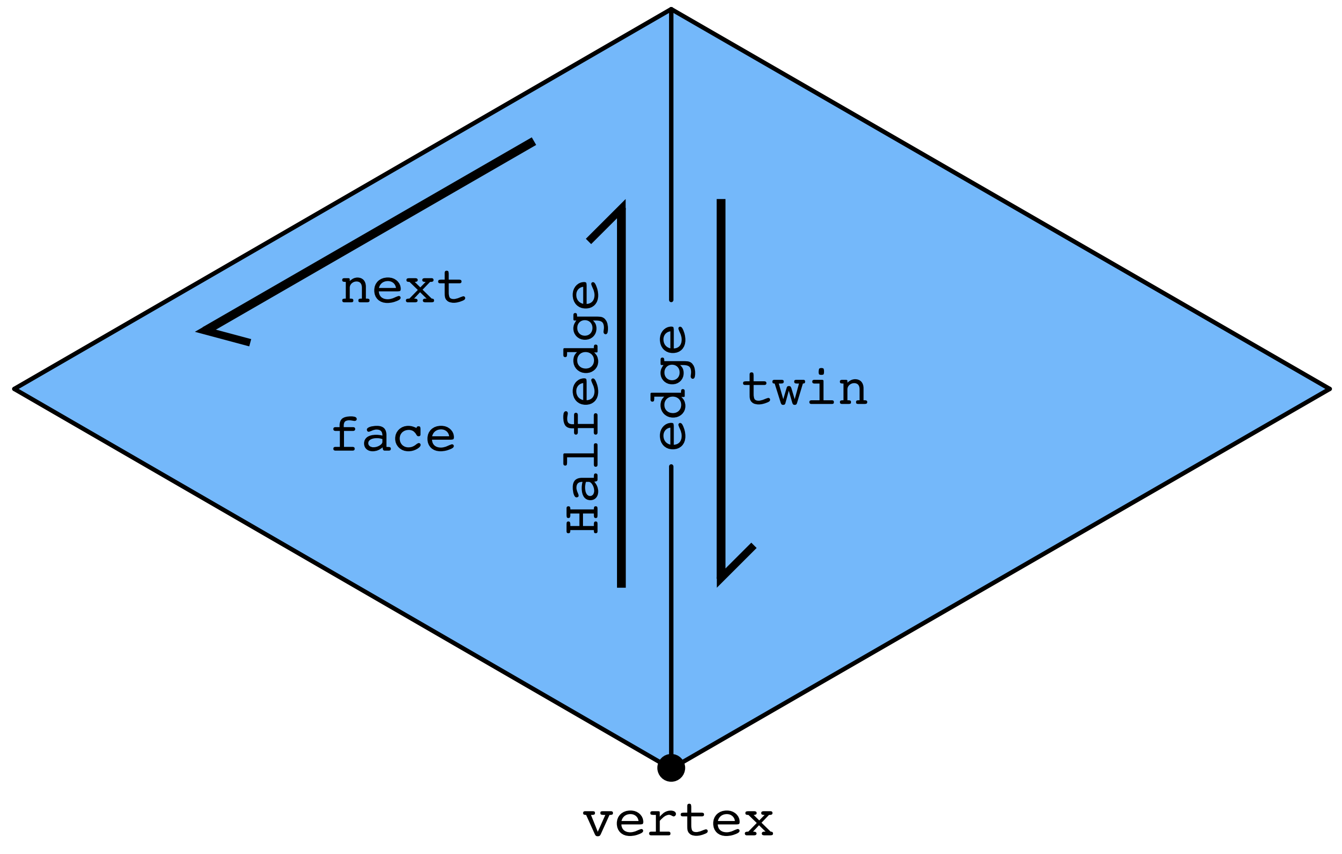
Computer Graphics
CMU 15-462/15-662

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

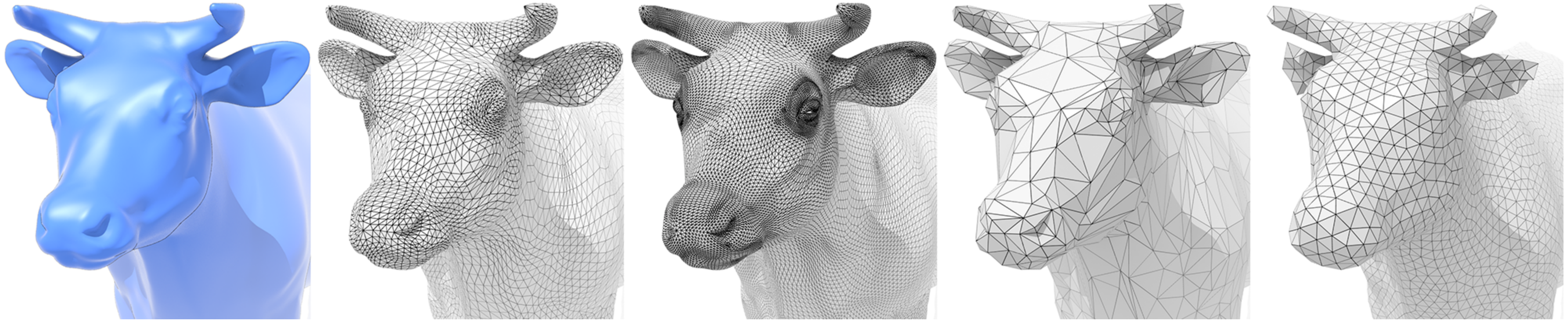






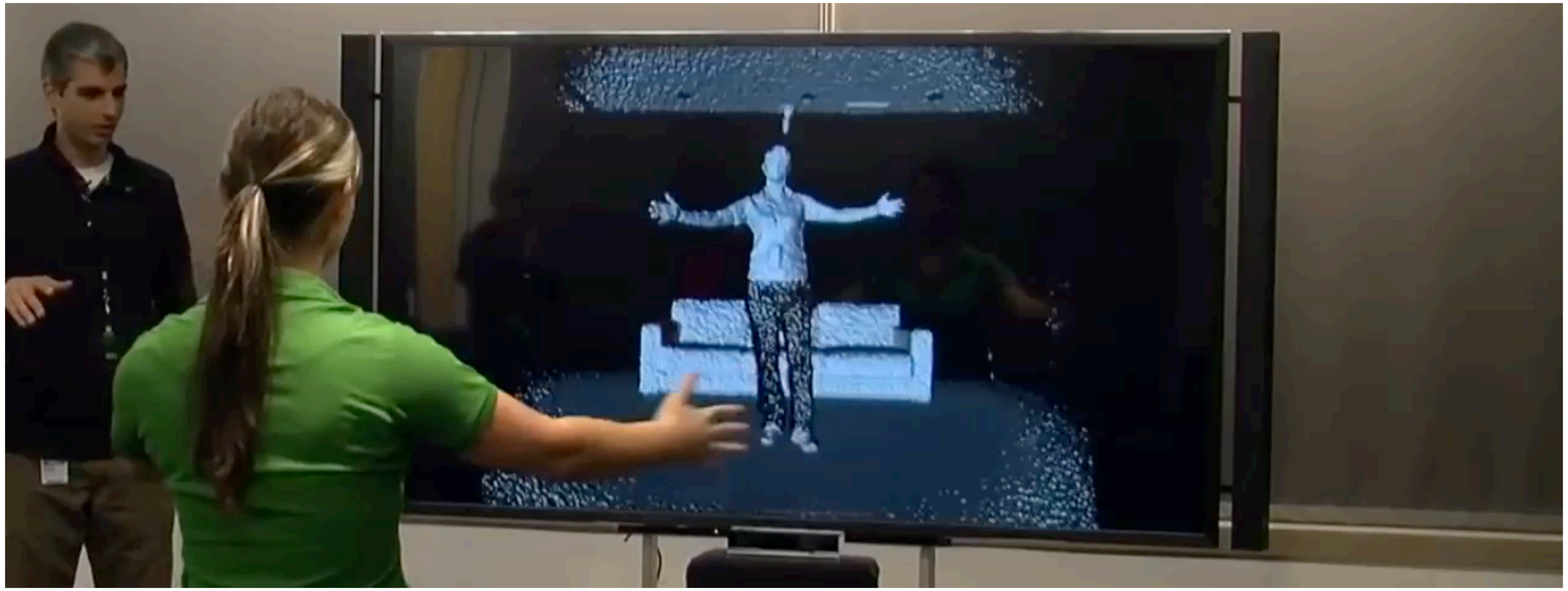
Today: Geometry Processing

- Extend traditional digital signal processing (audio, video, etc.) to deal with *geometric* signals:
  - upsampling / downsampling / resampling / filtering ...
  - aliasing (reconstructed surface gives "false impression")
- Beyond pure geometry, these are basic building blocks for many areas/algorithms in graphics (rendering, animation...)

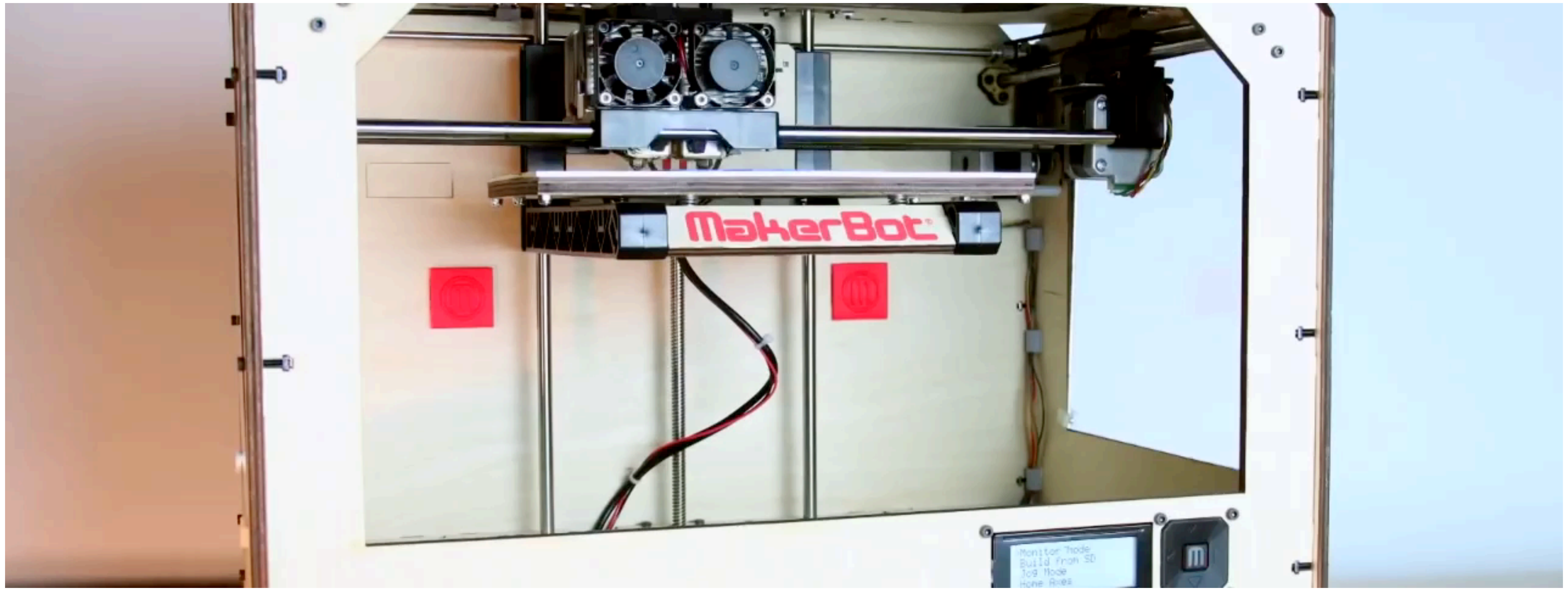


Digital Geometry Processing: Motivation

3D Scanning



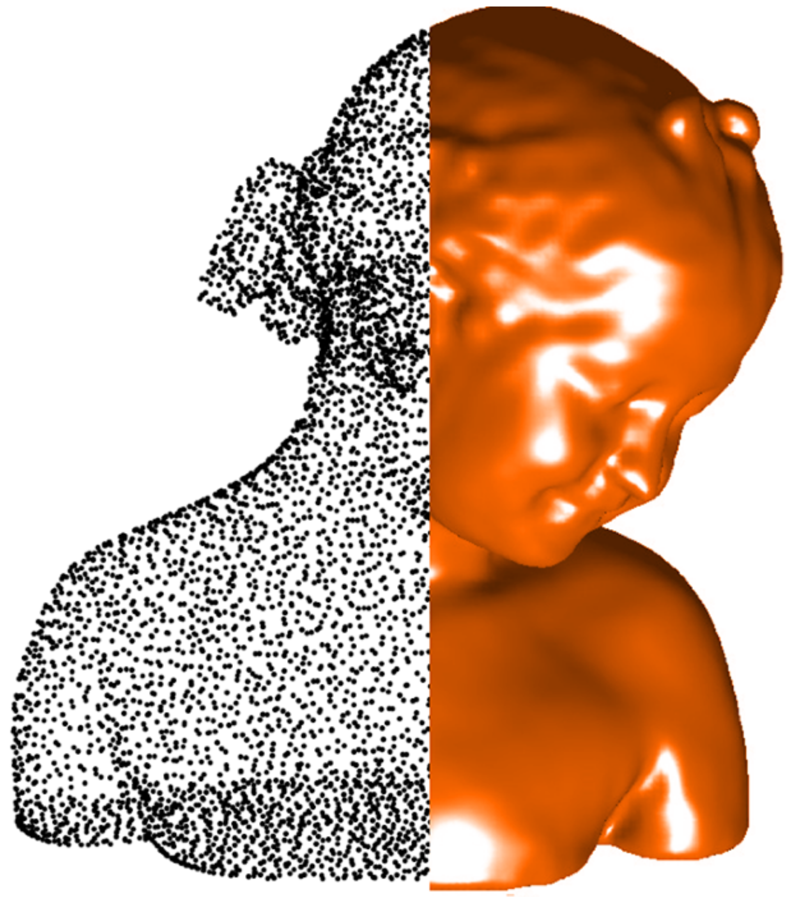
3D Printing



Geometry Processing Pipeline

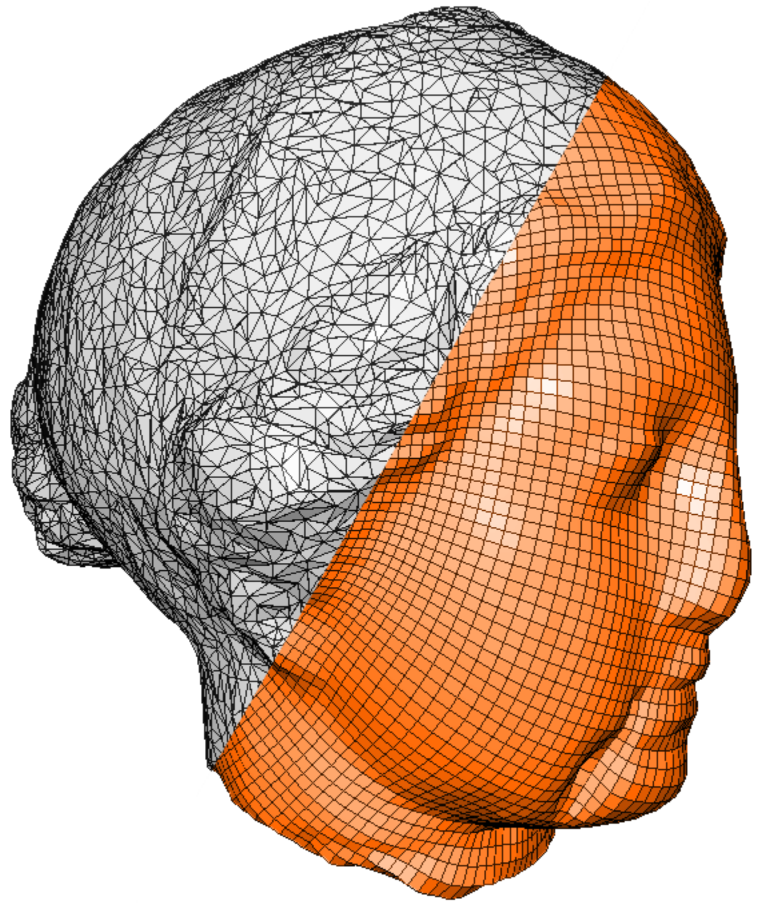
scan → process → print

Geometry Processing Tasks

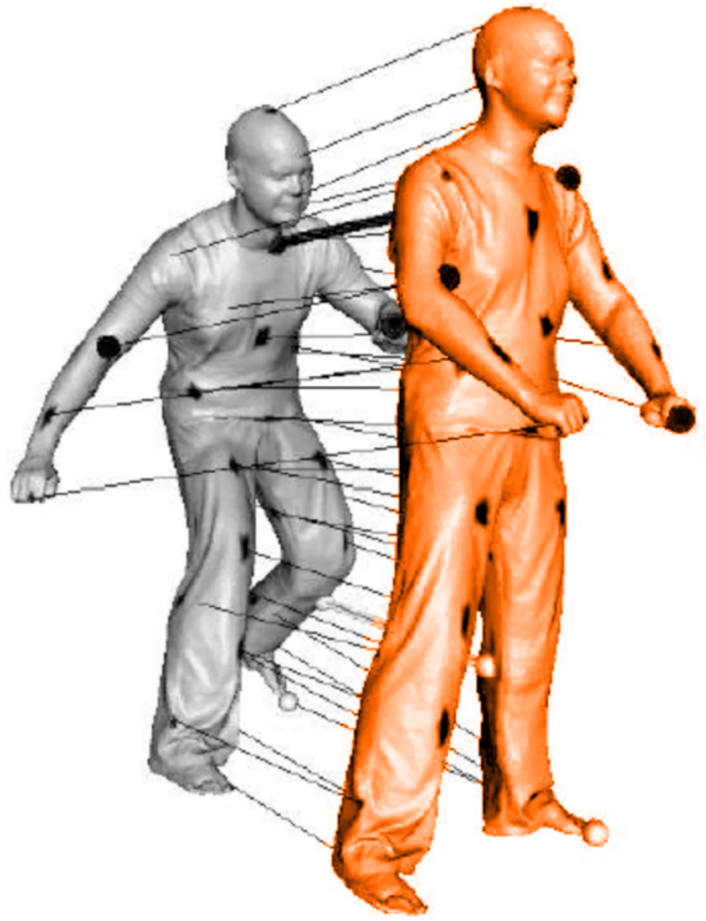


reconstruction

filtering

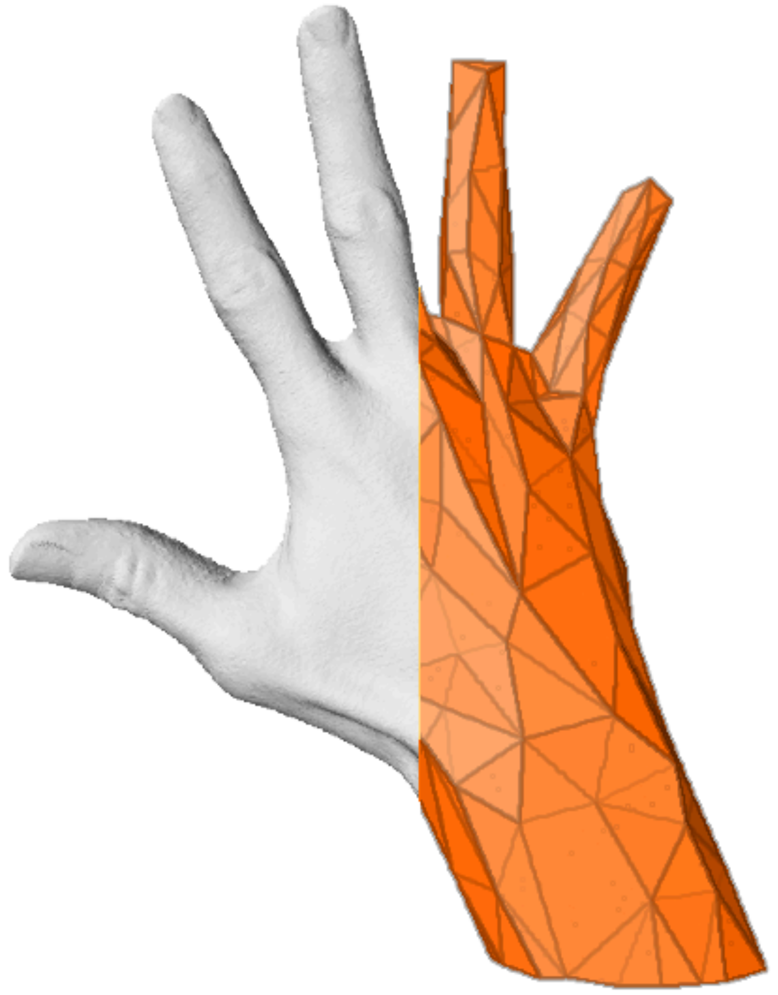


remeshing



shape analysis

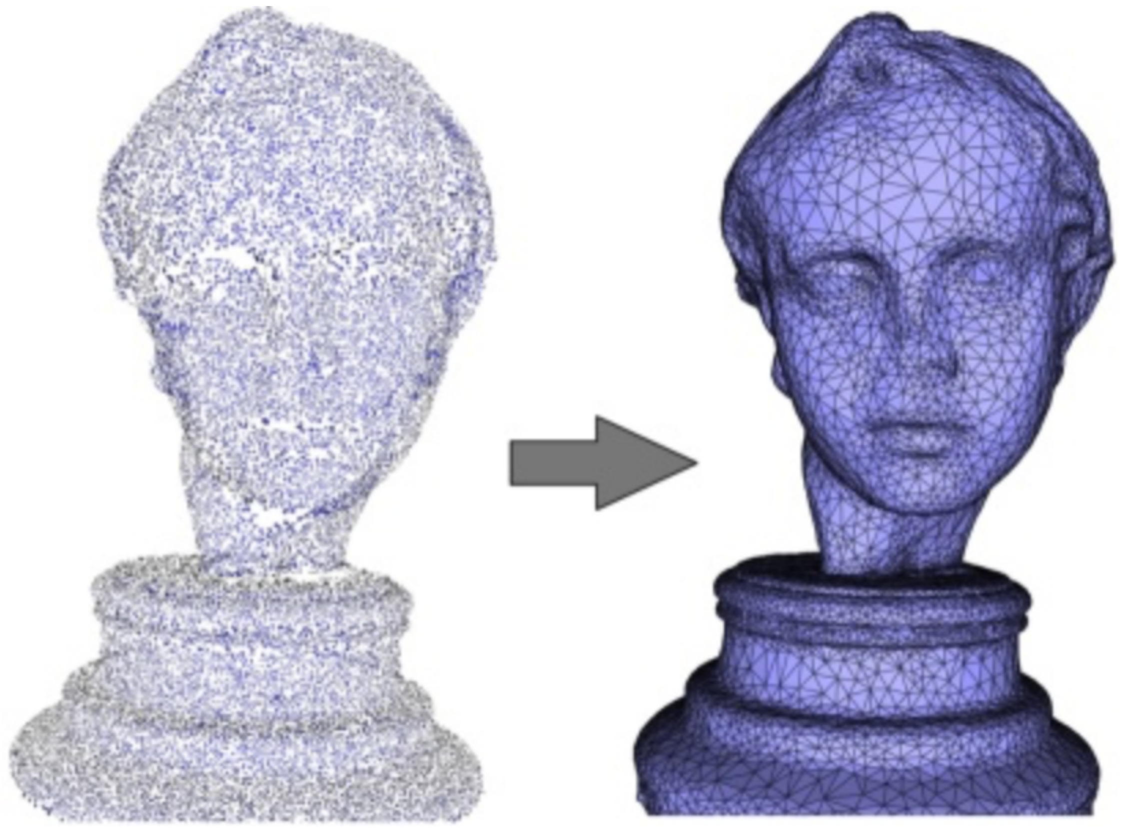
parameterization



compression

Geometry Processing: Reconstruction

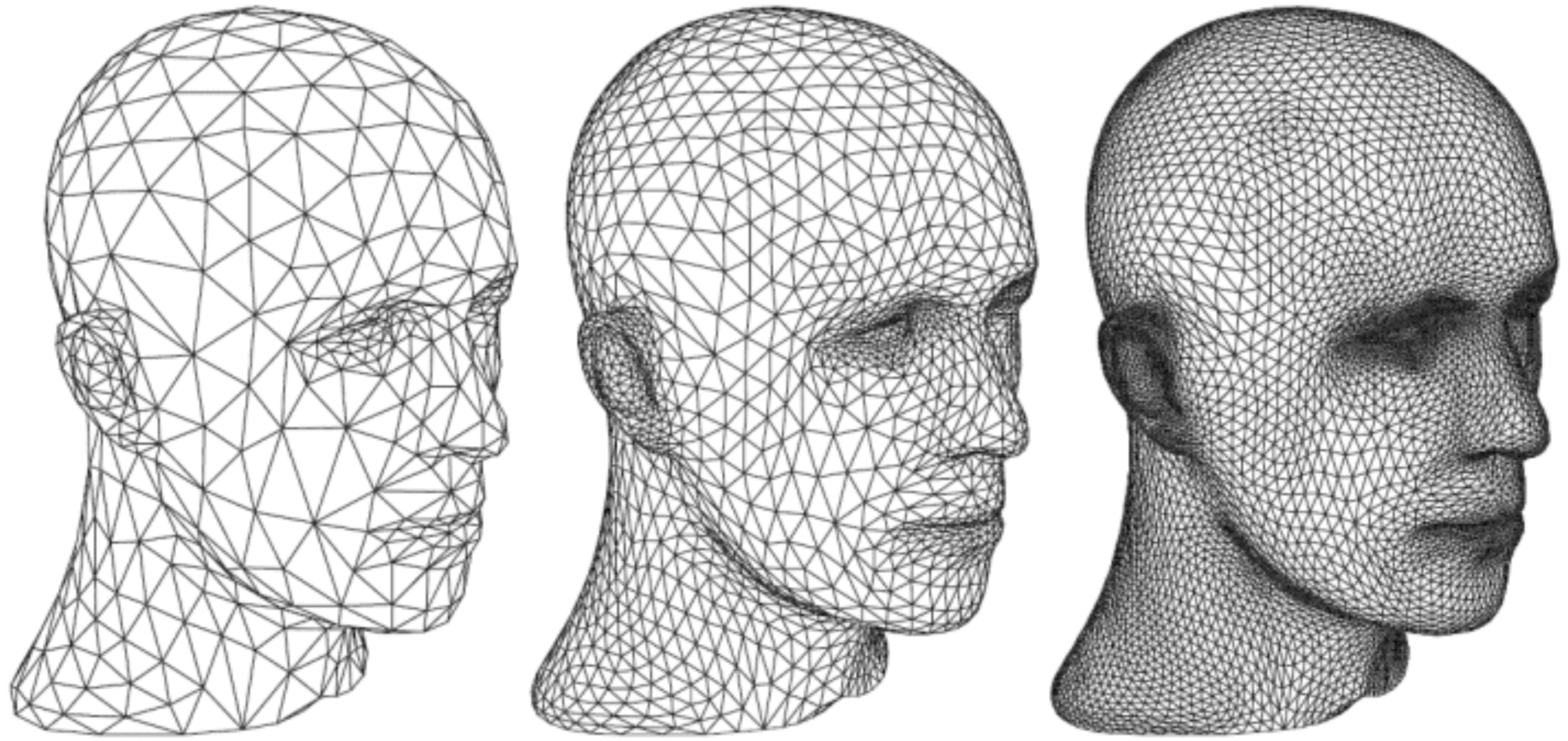
- Given samples of geometry, reconstruct surface
- What are "samples"? Many possibilities:
  - points, points & normals, ...
  - image pairs / sets (multi-view stereo)
  - line density integrals (MRI/CT scans)



- 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)

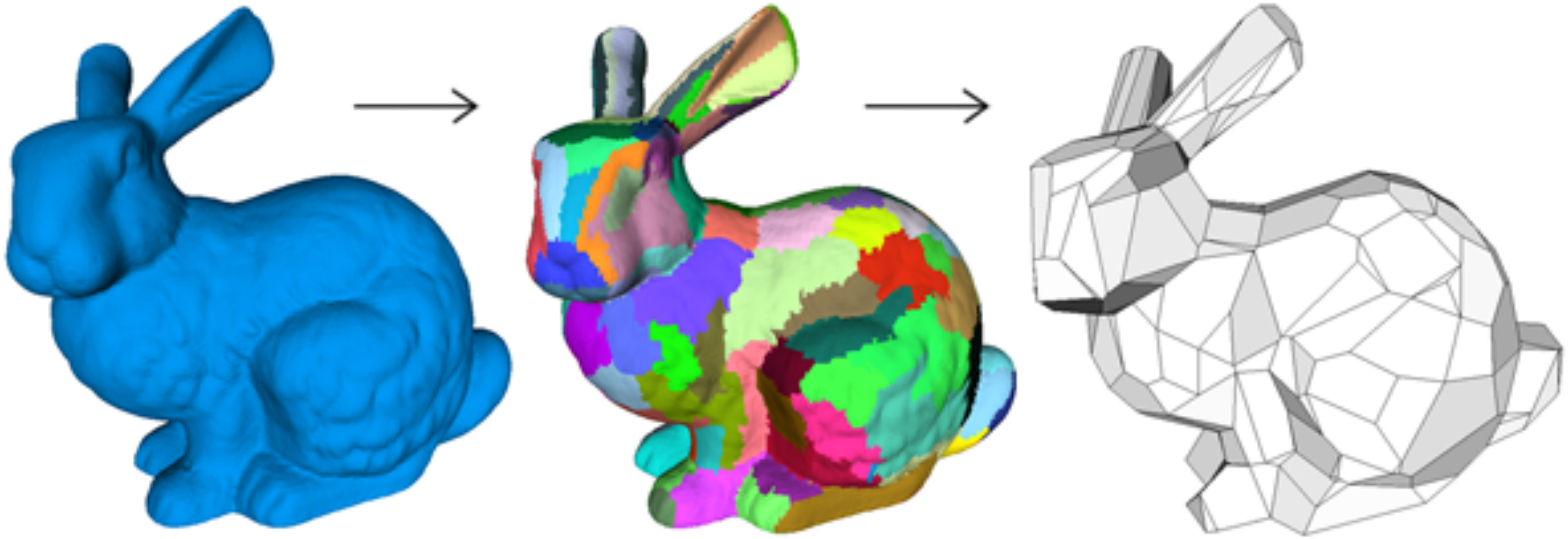
Geometry Processing: Upsampling

- 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, ...

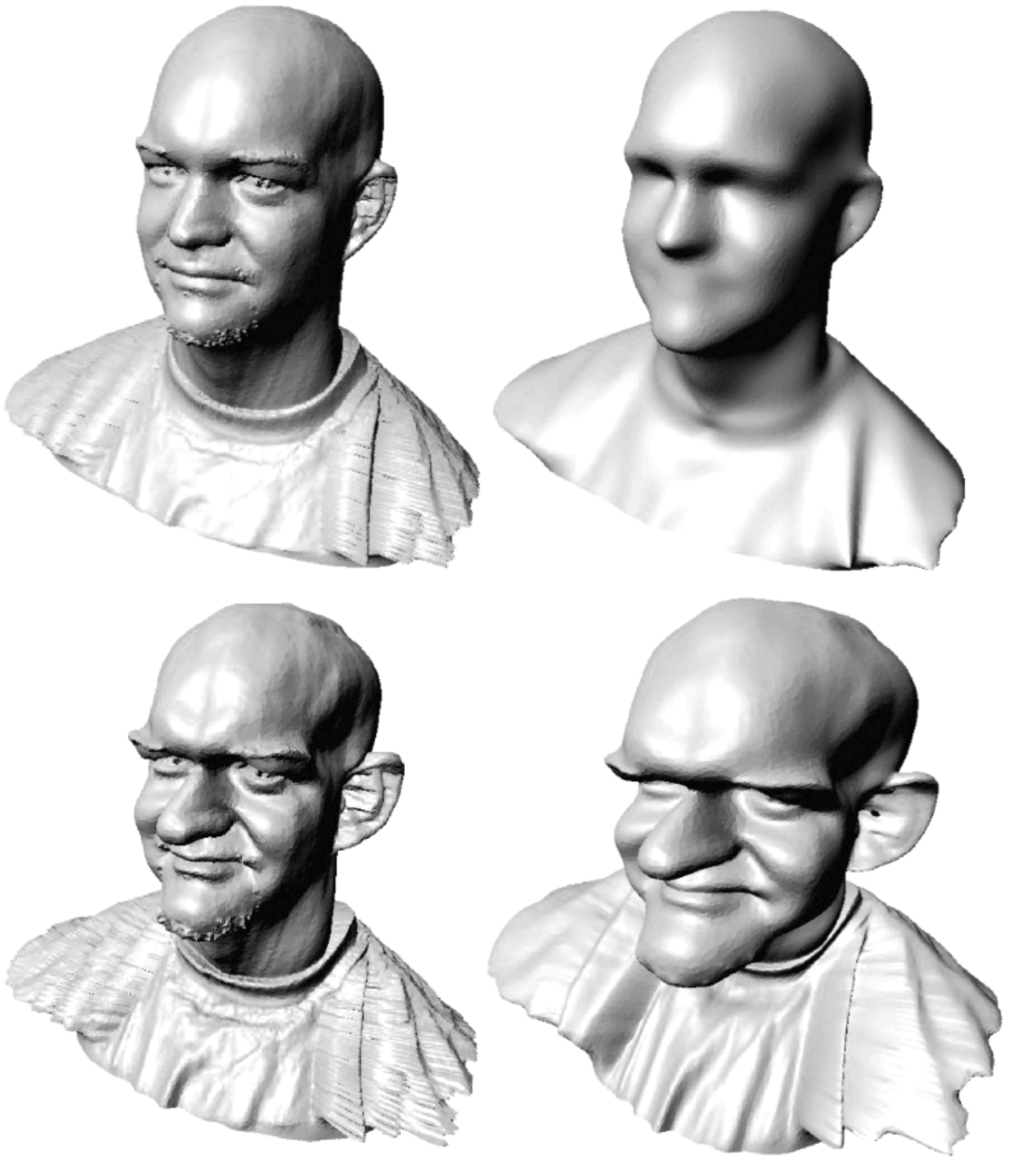


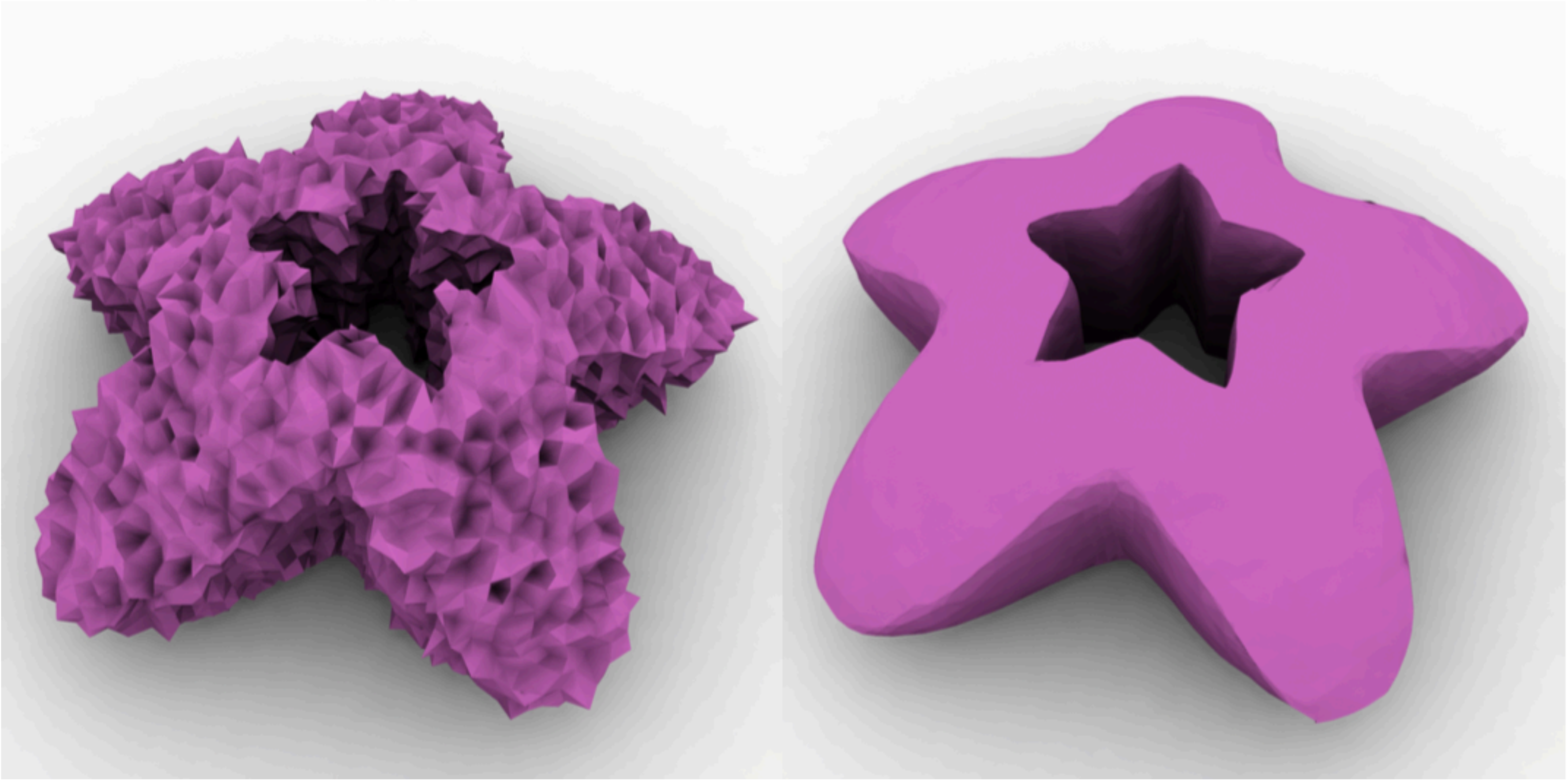
Geometry Processing: Resampling

- 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 equations

Geometry Processing: Filtering

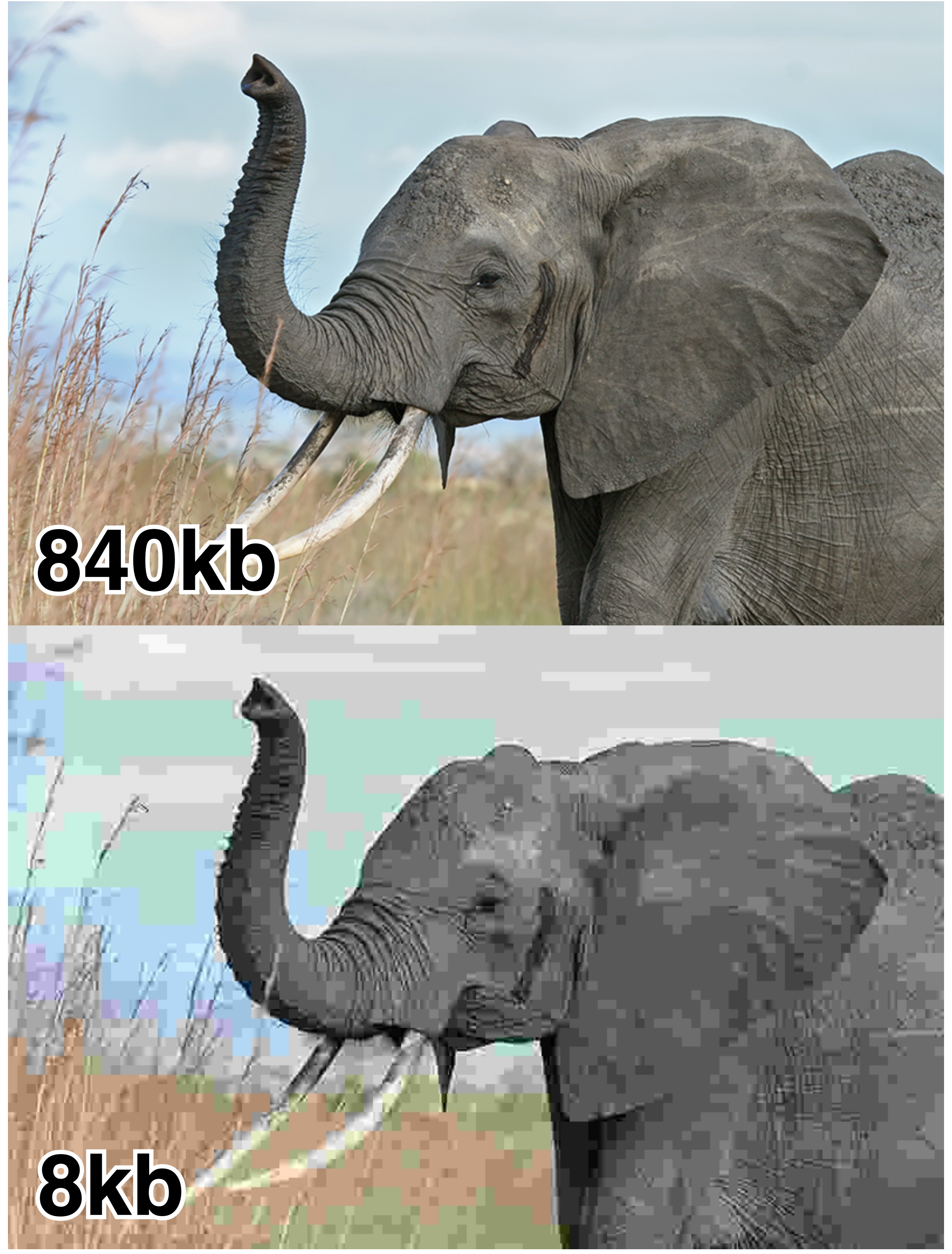
- Remove noise, or emphasize important features (e.g., edges)
- Images: blurring, bilateral filter, edge detection, ...
- Polygon meshes:
  - curvature flow
  - bilateral filter
  - spectral filter

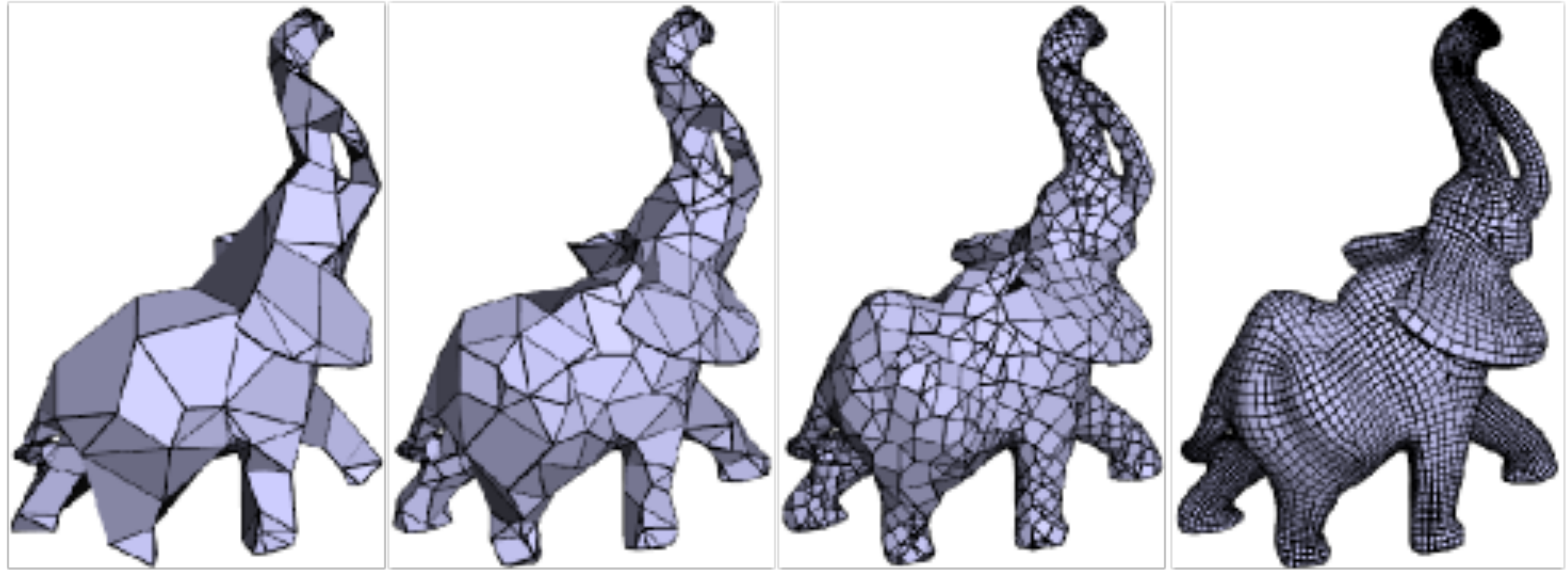




Geometry Processing: Compression

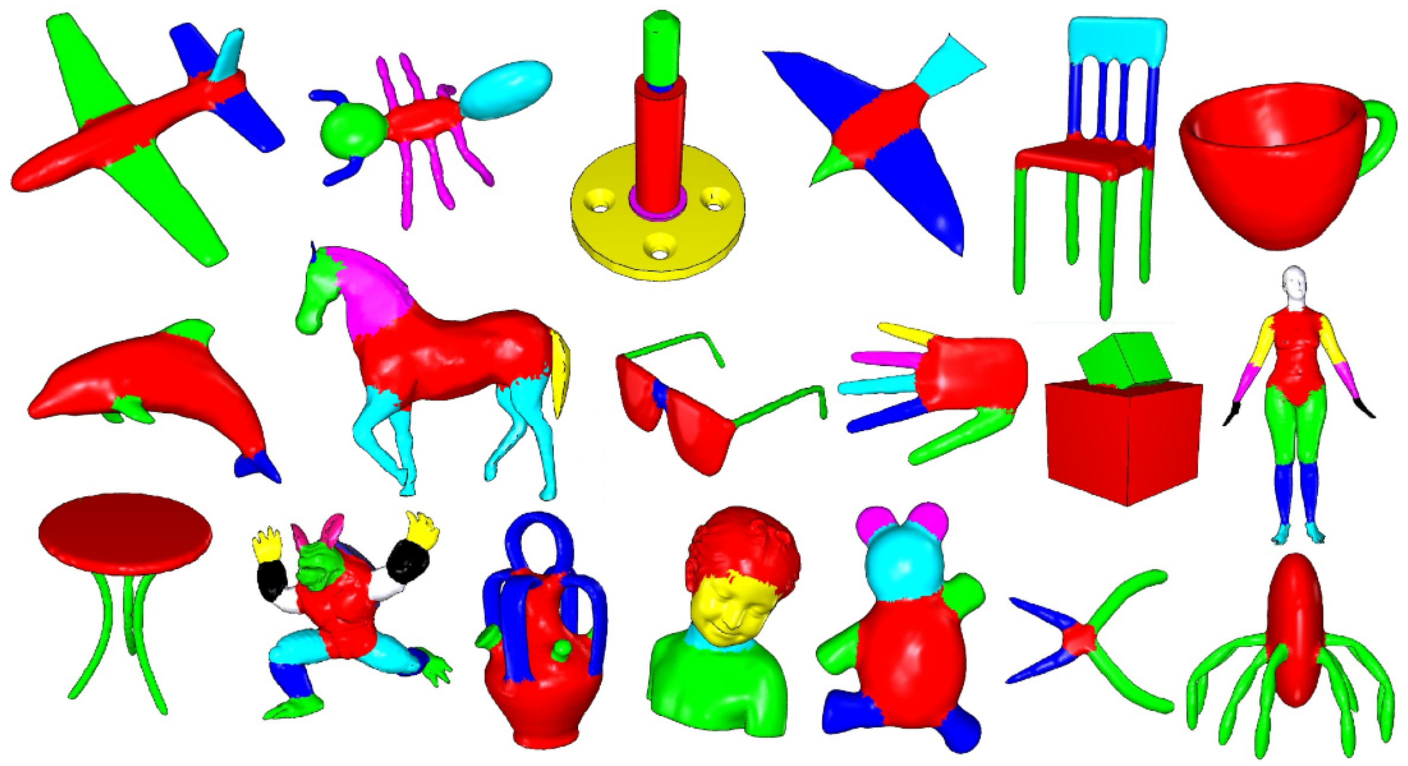
- Reduce storage size by eliminating redundant data/ approximating unimportant data
- Images:
    - run-length, Huffman coding - *lossless*
    - cosine/wavelet (JPEG/MPEG) - *lossy*
- Polygon meshes:
    - compress geometry *and* connectivity
    - many techniques (lossy & lossless)

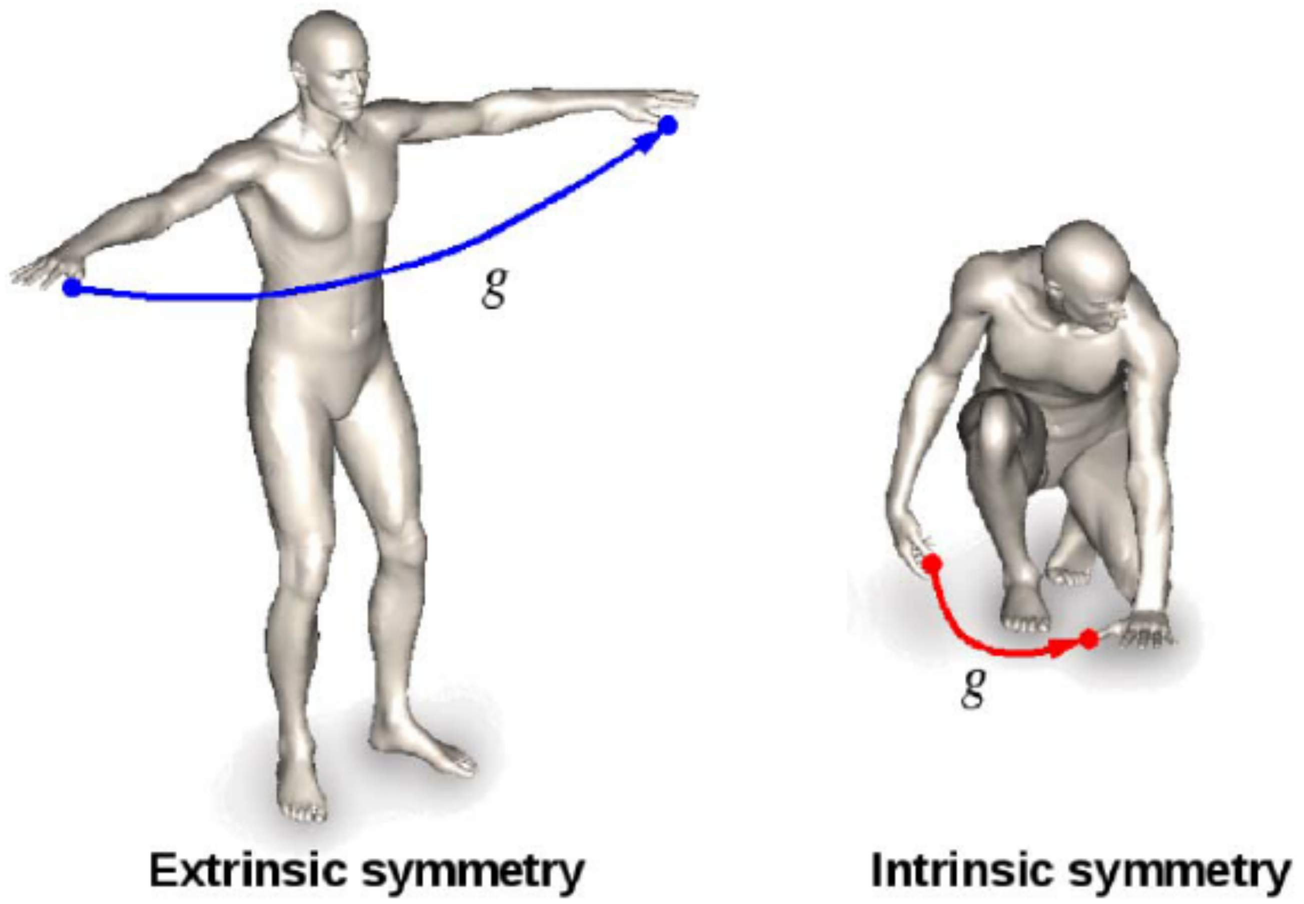


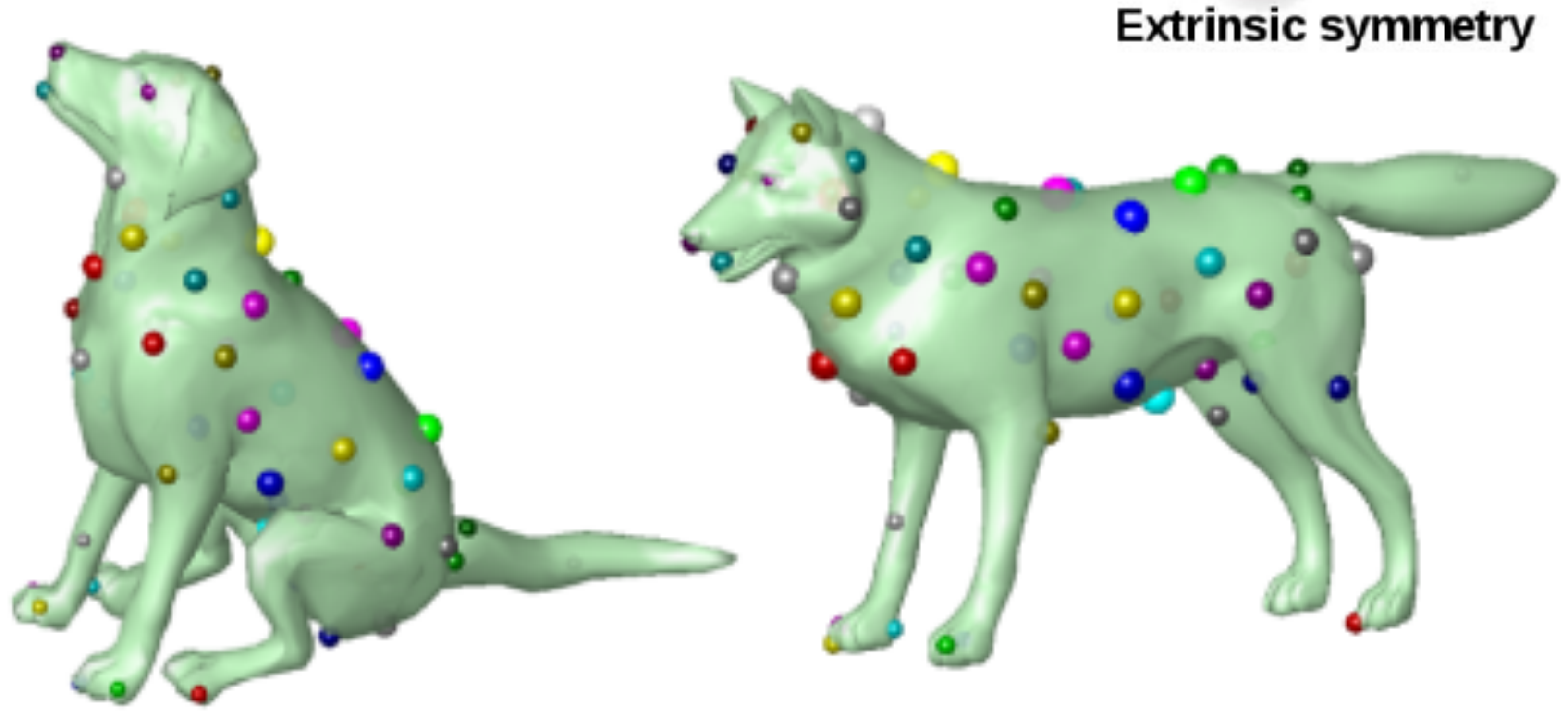


Geometry Processing: Shape Analysis

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



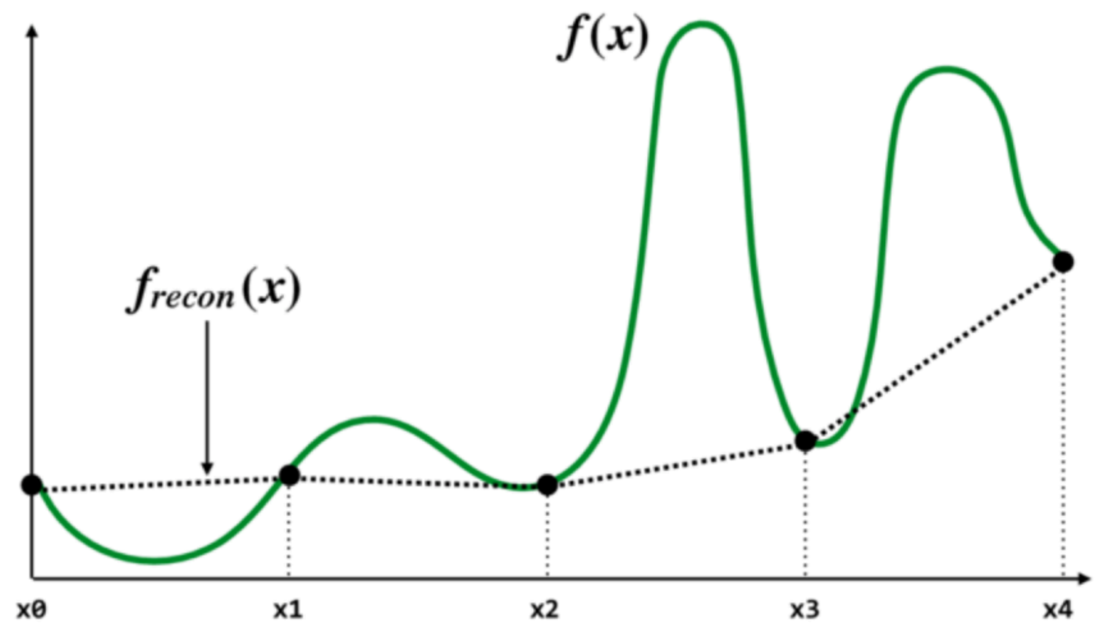


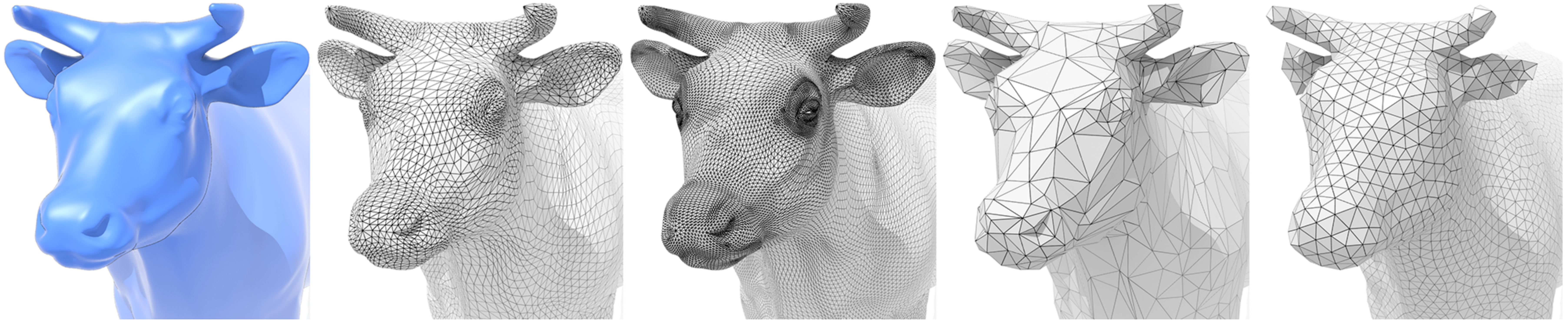


Enough overview—
Let's process some geometry!

Remeshing as resampling

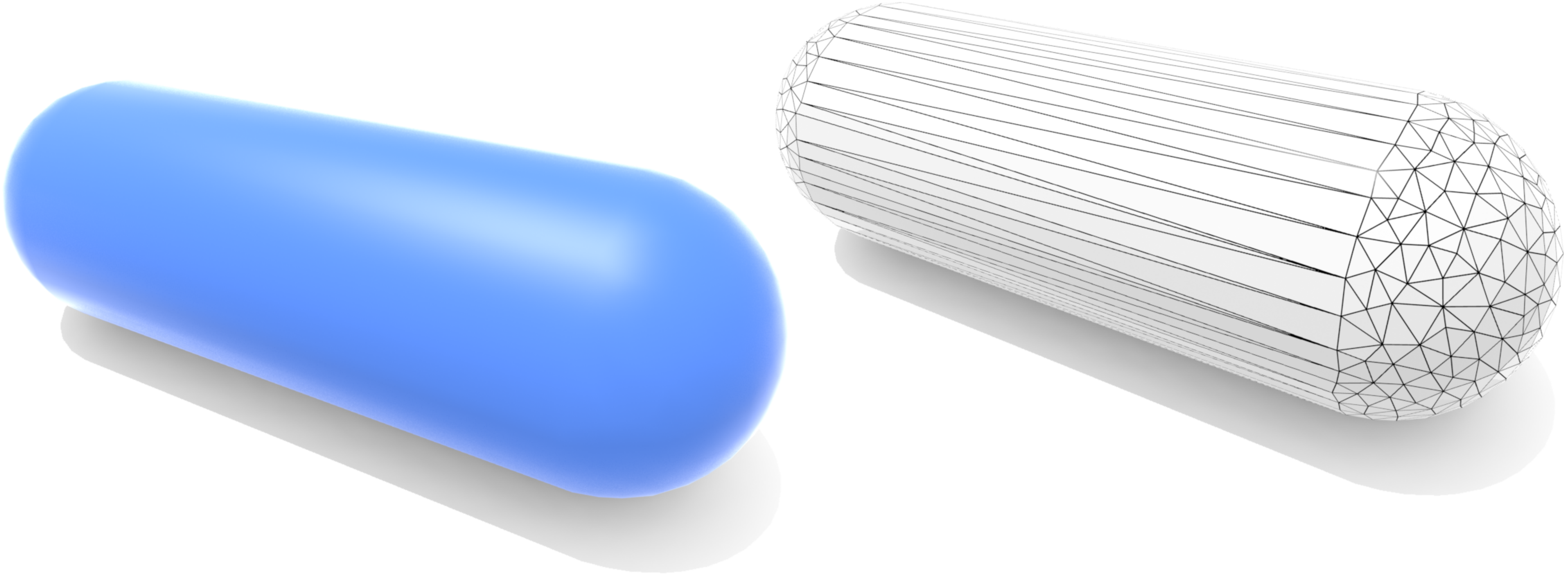
- Remember our discussion of *aliasing*
- Bad sampling makes signal appear different than it really is
- E.g., undersampled curve looks flat
- Geometry is no different!
  - undersampling destroys features
  - oversampling bad for performance





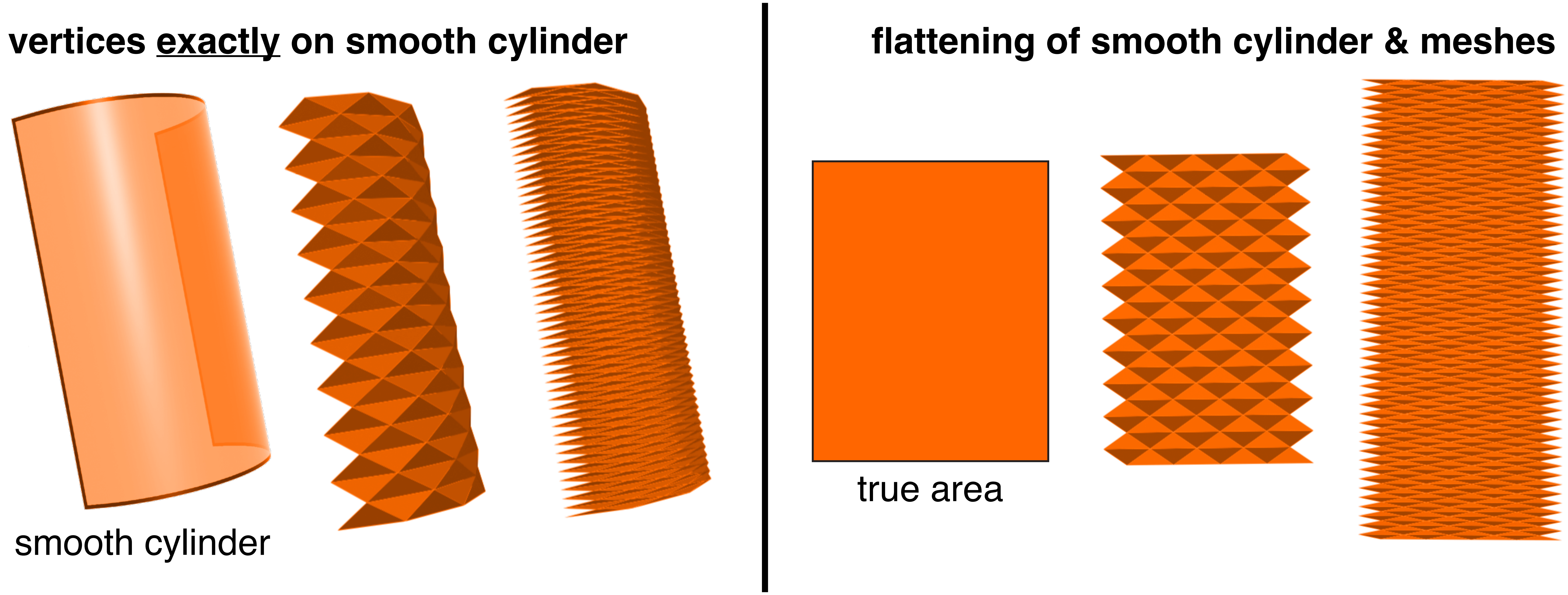
What makes a "good" mesh?

- 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!

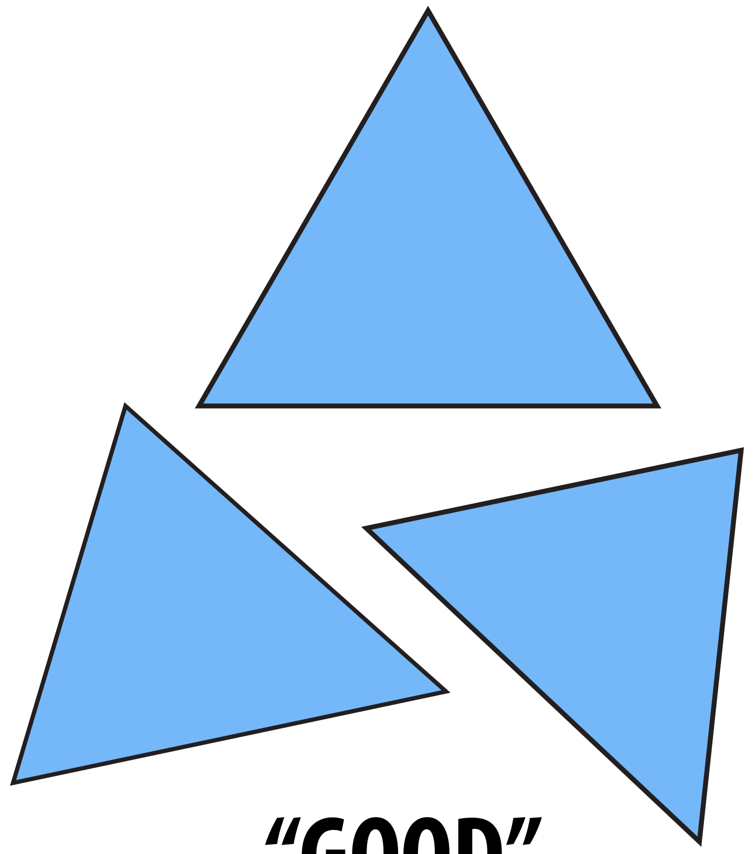
- Just because the vertices of a mesh are close to the surface it approximates *does not mean it's a good approximation!*
- Can still have wrong appearance, wrong area, wrong…
- Need to consider other factors*, e.g., close approximation of *surface normals*



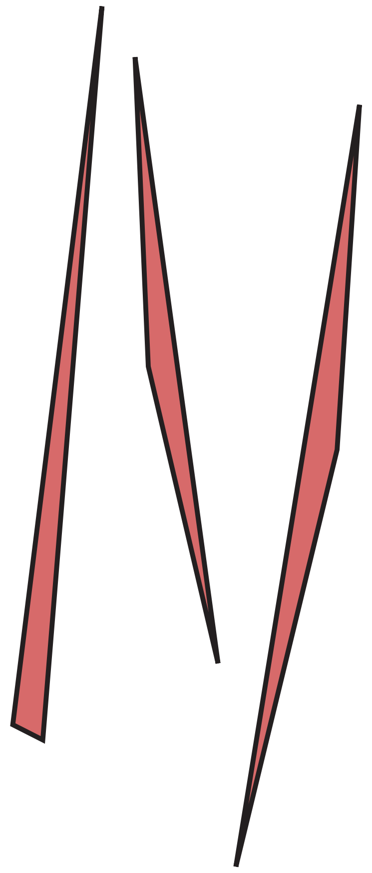
*See Hildebrandt et al (2007), "On the convergence of metric and geometric properties of polyhedral surfaces"

What else makes a “good” triangle mesh?

- **Another rule of thumb: *triangle shape***



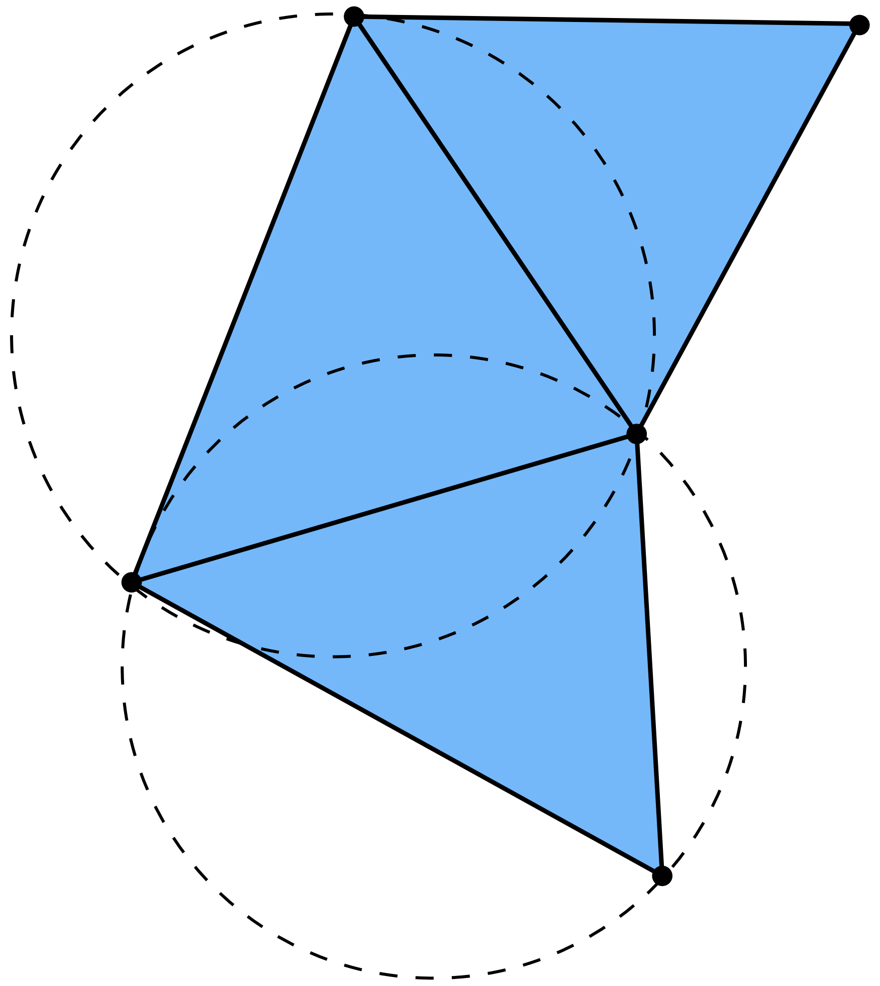
“GOOD”



“BAD”

pronunciation:

DELAUNAY

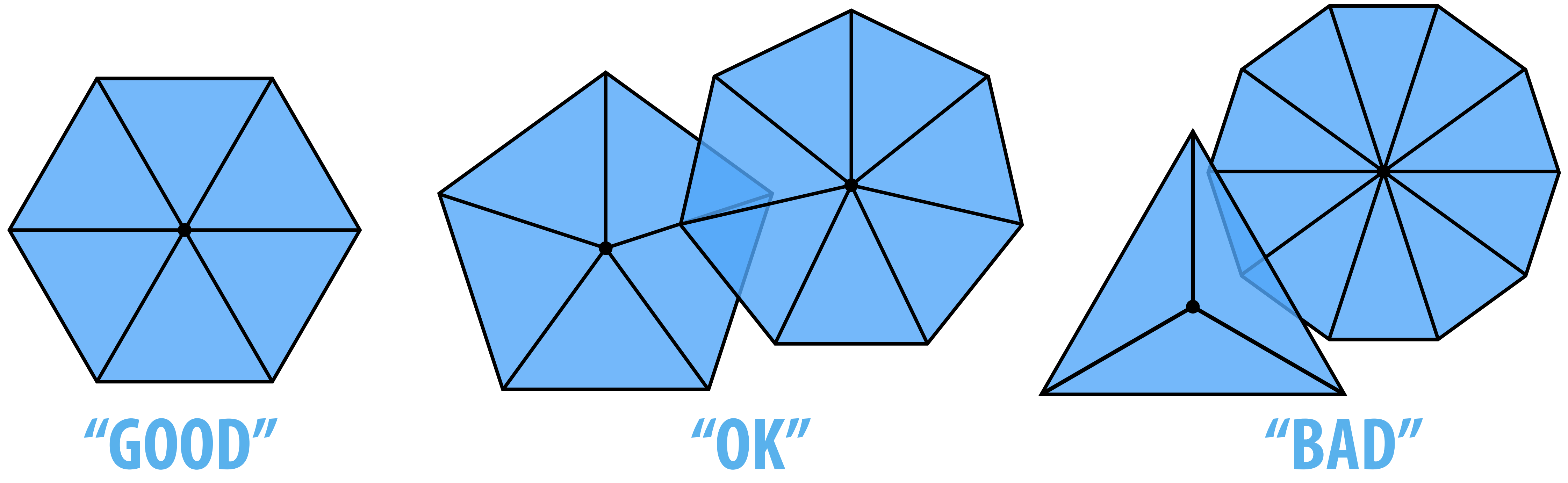


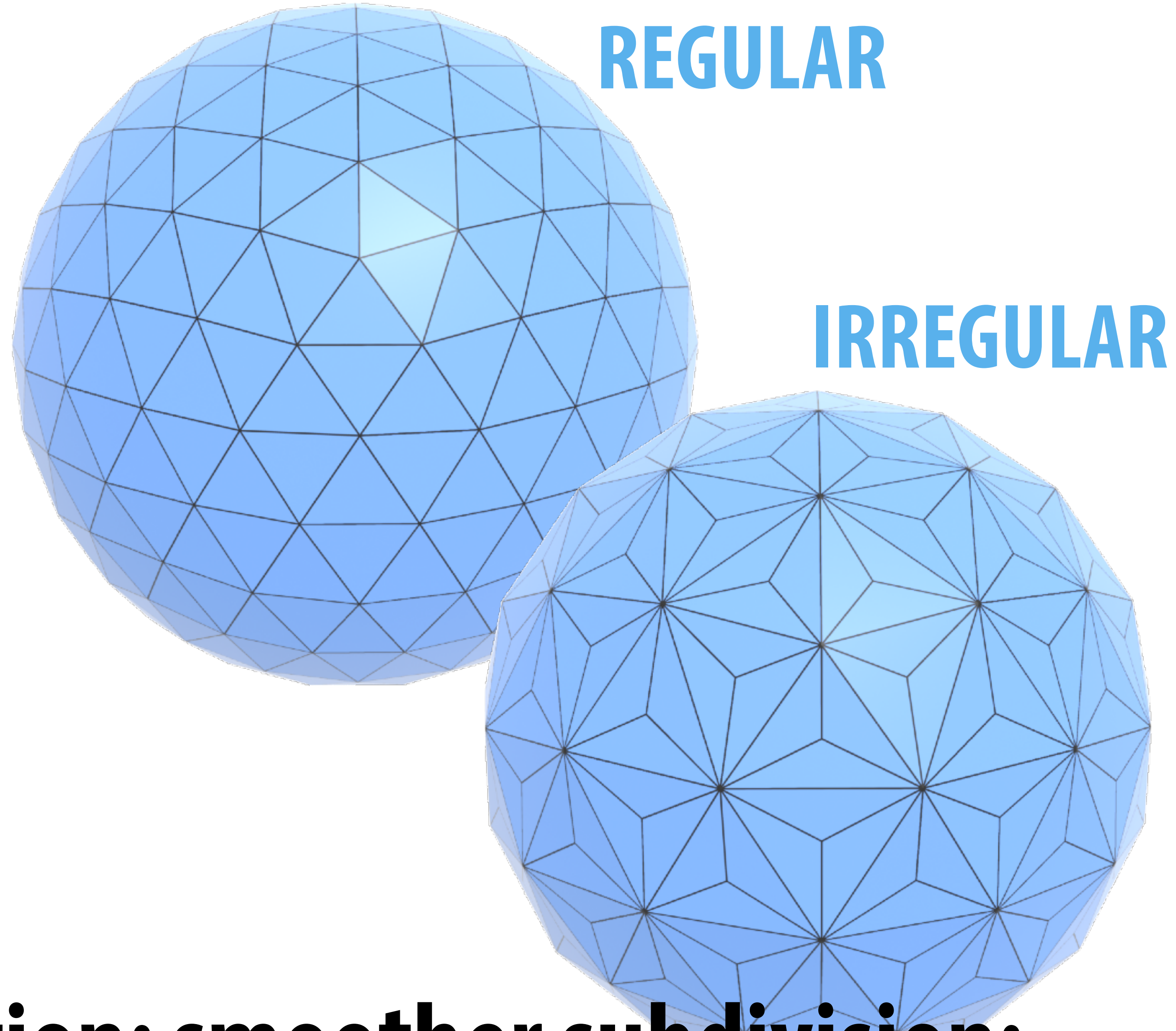
- E.g., all angles close to 60 degrees
- More sophisticated condition: *Delaunay* (empty circumcircles)
  – often helps with numerical accuracy/stability
  – coincides with <u>shockingly</u> many other desirable properties
  (maximizes minimum angle, provides smoothest interpolation, guarantees maximum principle…)
- Tradeoffs w/ good geometric approximation*
  –e.g., long & skinny might be “more efficient”

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

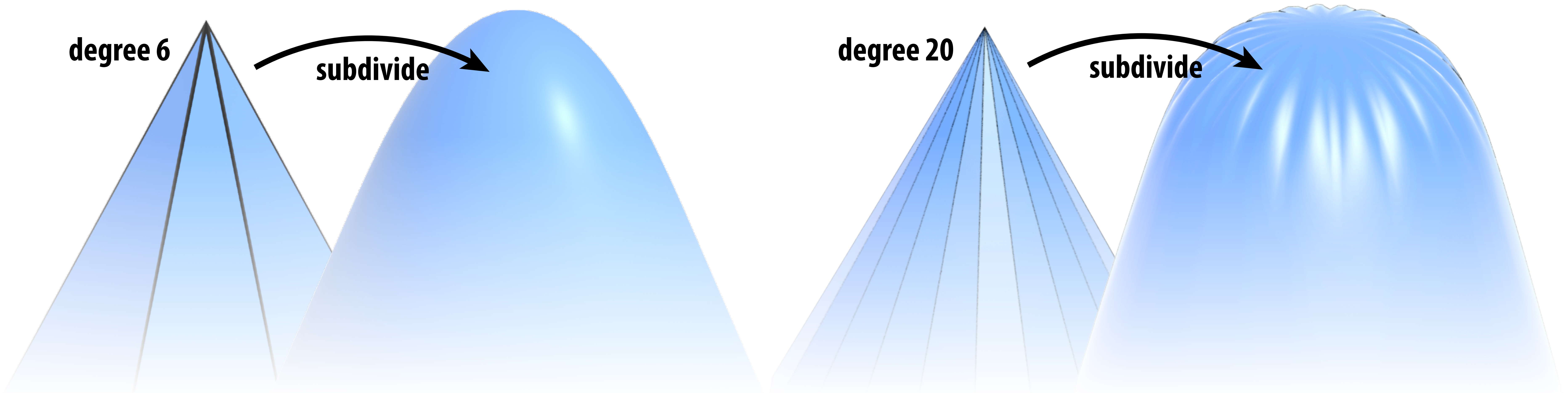
What else constitutes a “good” mesh?

- Another rule of thumb: *regular vertex degree*
- Degree 6 for triangle mesh, 4 for quad mesh





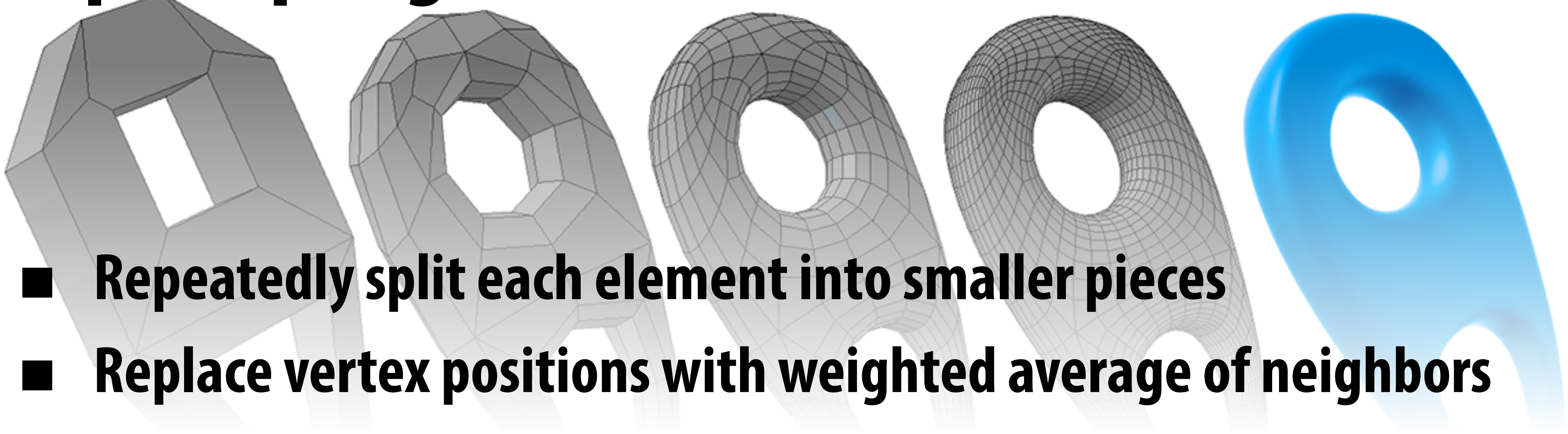
Why? Better polygon shape; more regular computation; smoother subdivision:



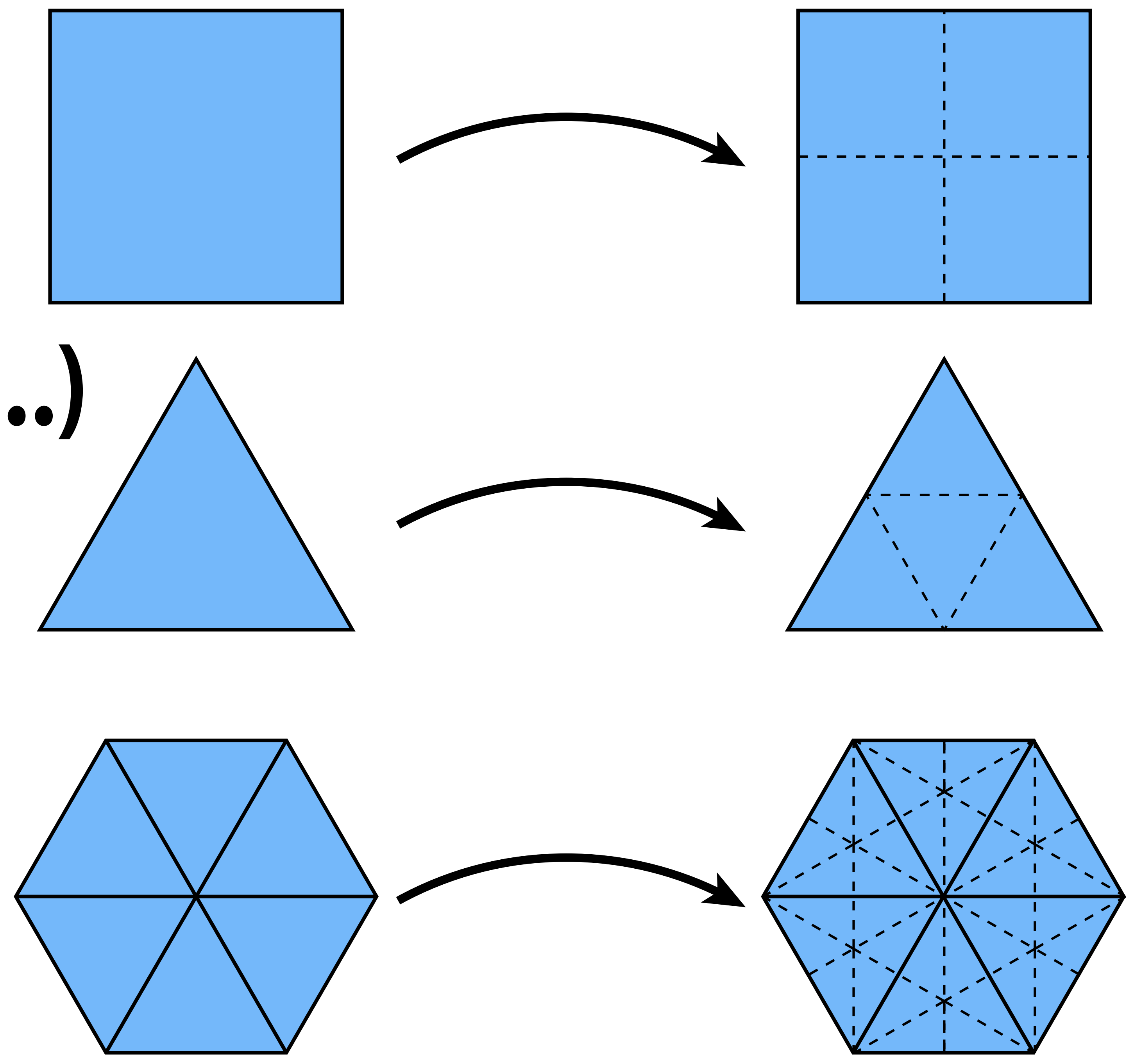
<u>Fact</u>: in general, can’t have regular vertex degree everywhere!

How do we upsample a mesh?

Upsampling via Subdivision

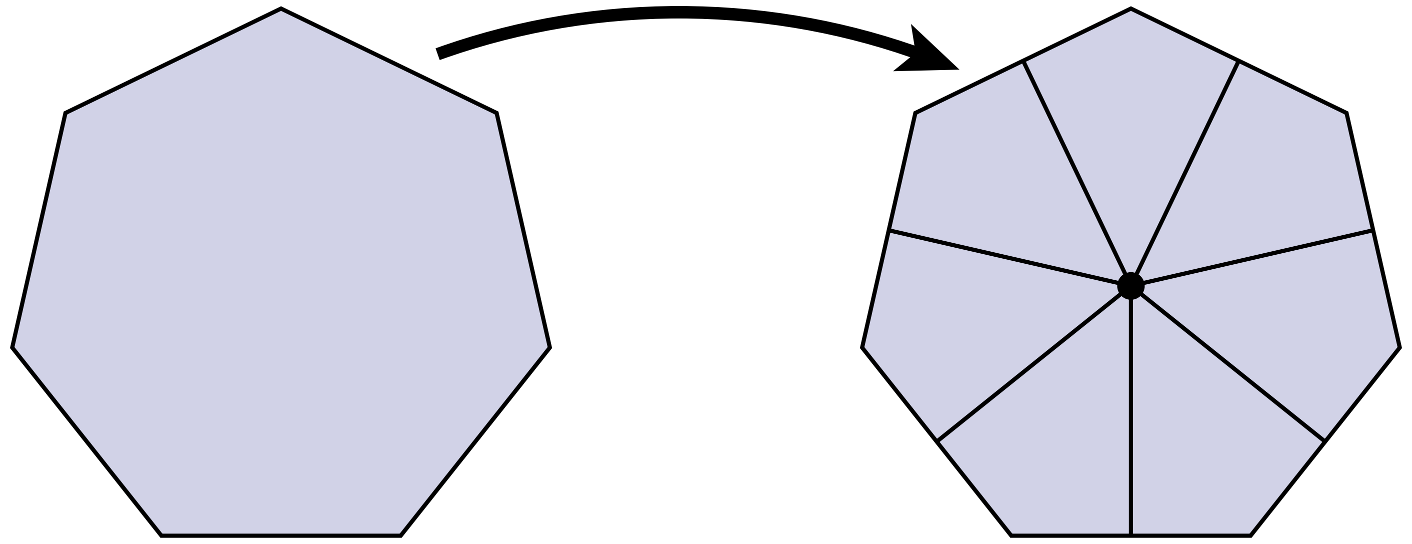


- Repeatedly split each element into smaller pieces
- Replace vertex positions with weighted average of neighbors
- Main considerations:
  - interpolating vs. approximating
  - limit surface continuity ($C^1, C^2, \ldots$)
  - behavior at irregular vertices
- Many options:
  - Quad: Catmull-Clark
  - Triangle: Loop, Butterfly, Sqrt(3)



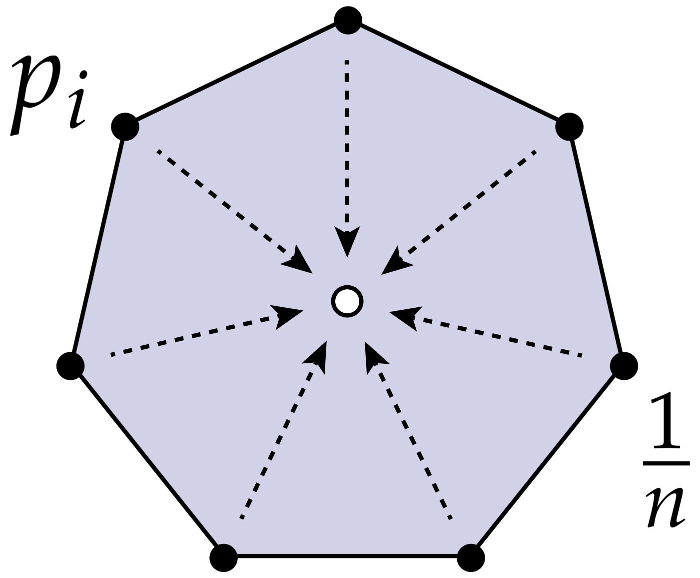
Catmull-Clark Subdivision

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



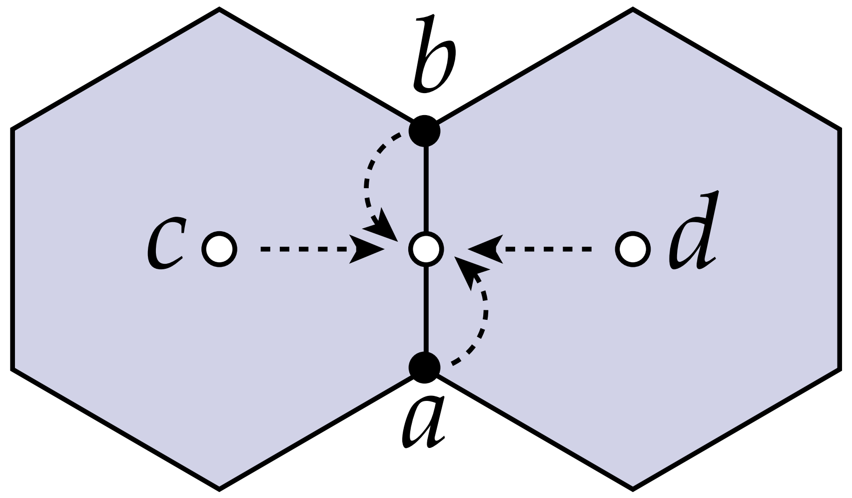
- **New vertex positions are weighted combination of old ones:**

STEP 1: Face coords



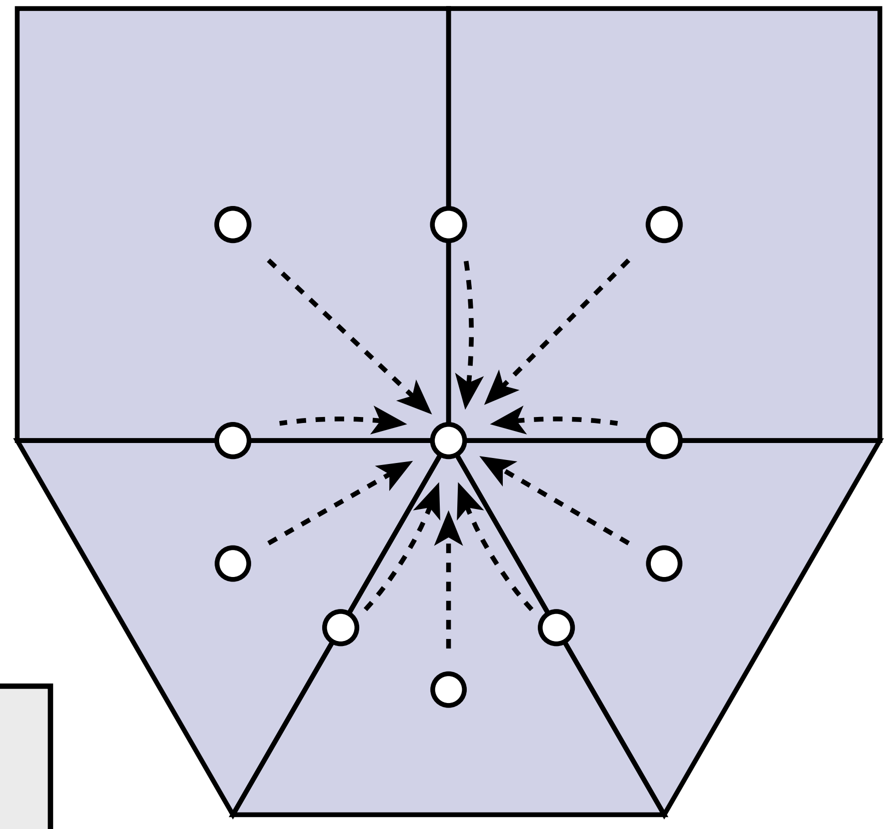
$\frac{1}{n}\sum_i p_i$

STEP 2: Edge coords



$(a+b+c+d)/4$

STEP 3: Vertex coords



New vertex coords:

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

n – vertex degree

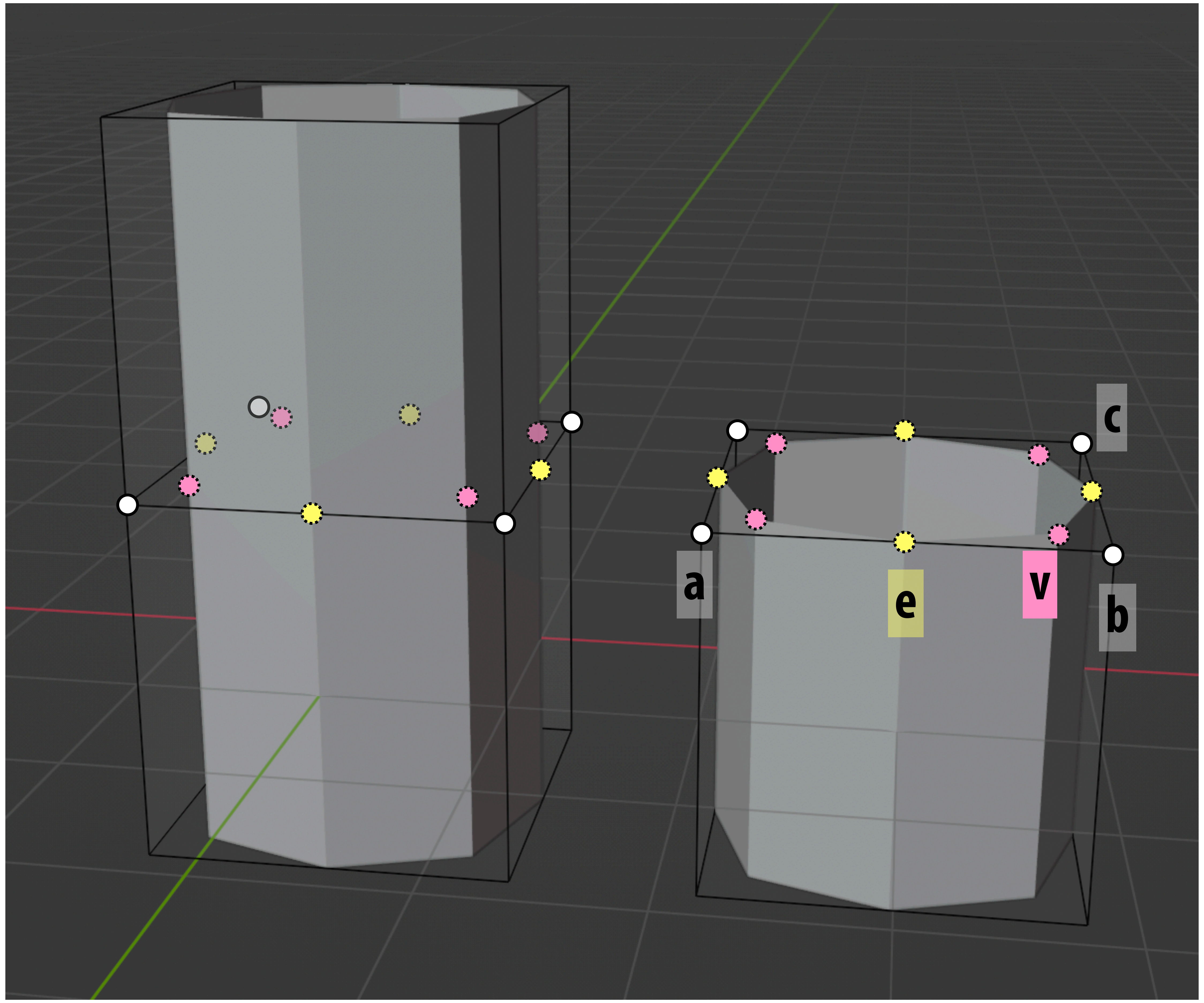
Q – average of face coords around vertex

R – average of edge midpoints around vertex

S – original vertex position

Catmull-Clark: Boundaries

- How to handle boundaries?



Idea: want the subdivision to "behave the same" if tube gets cut in half

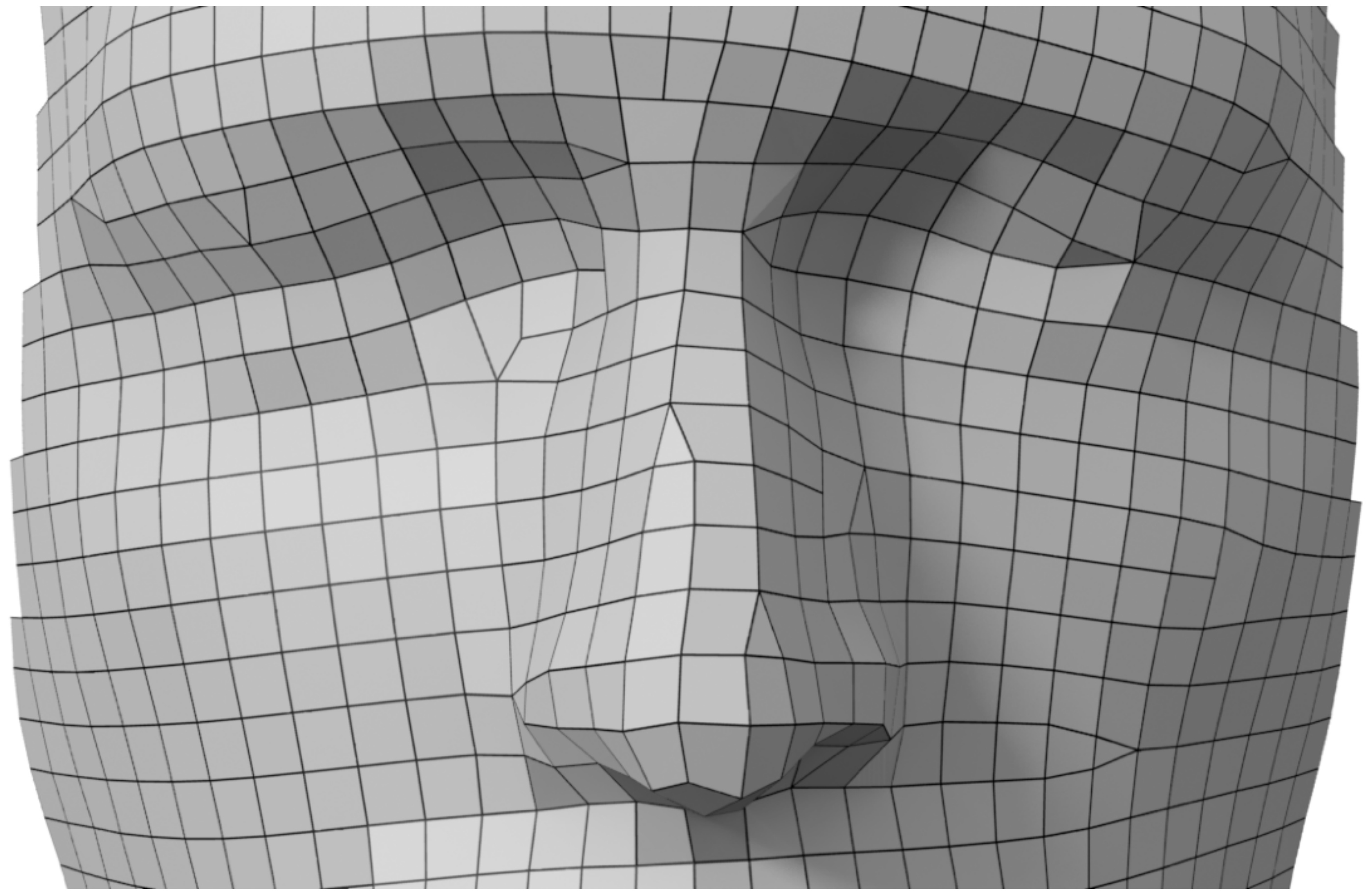
=> edge points: midpoints

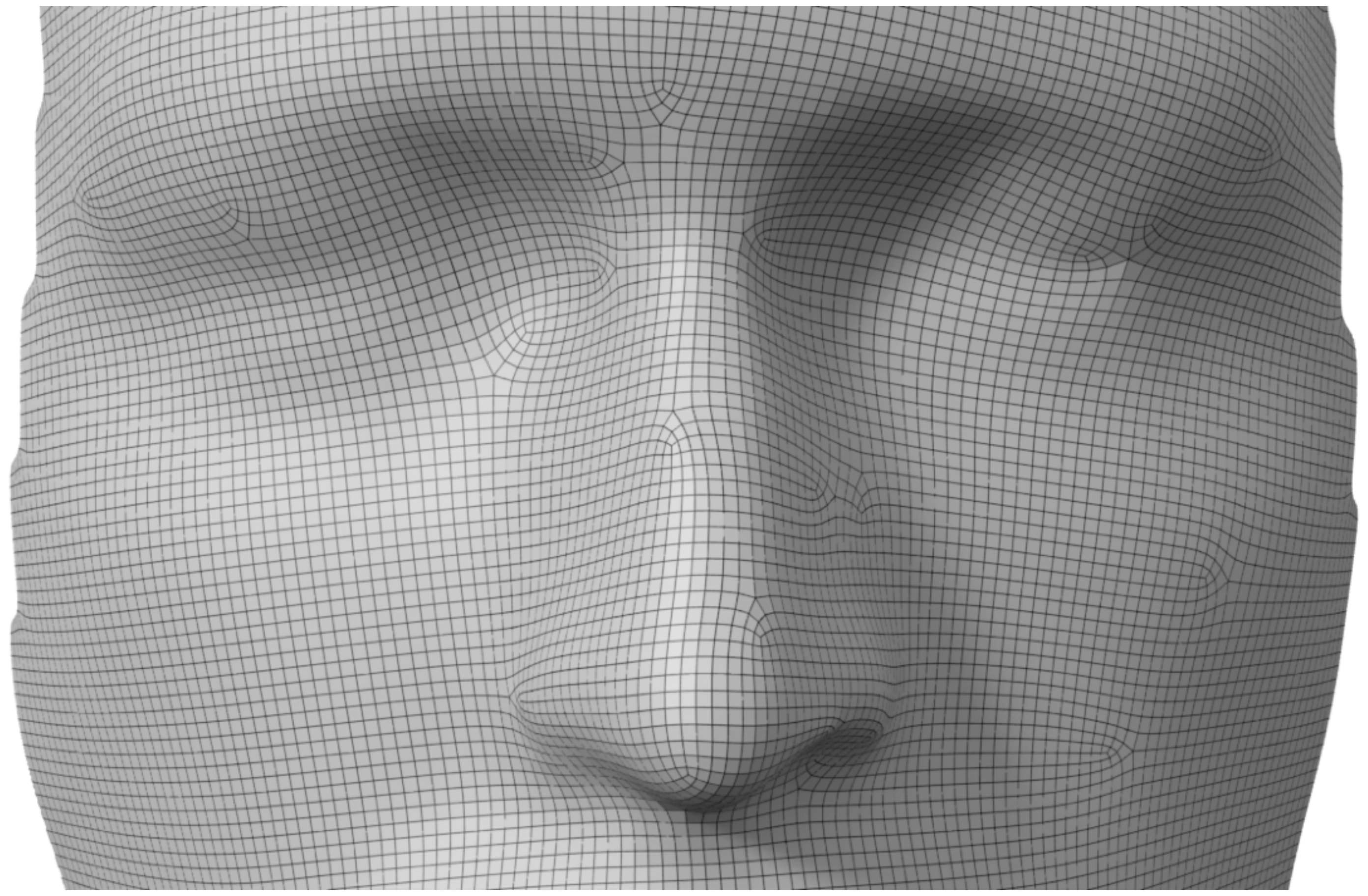
$e = (a + b) / 2$

=> vertex positions:

$v = 1/8a + 1/8c + 3/4\,b$

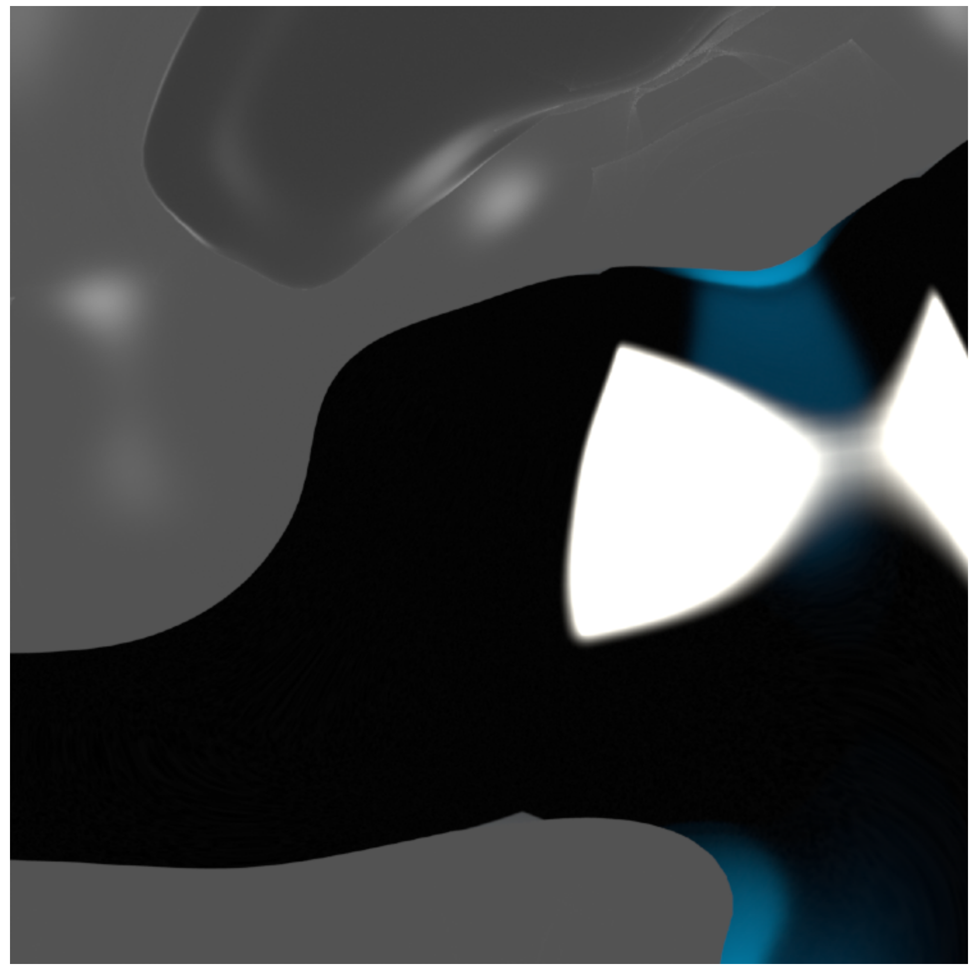
Catmull-Clark on quad mesh



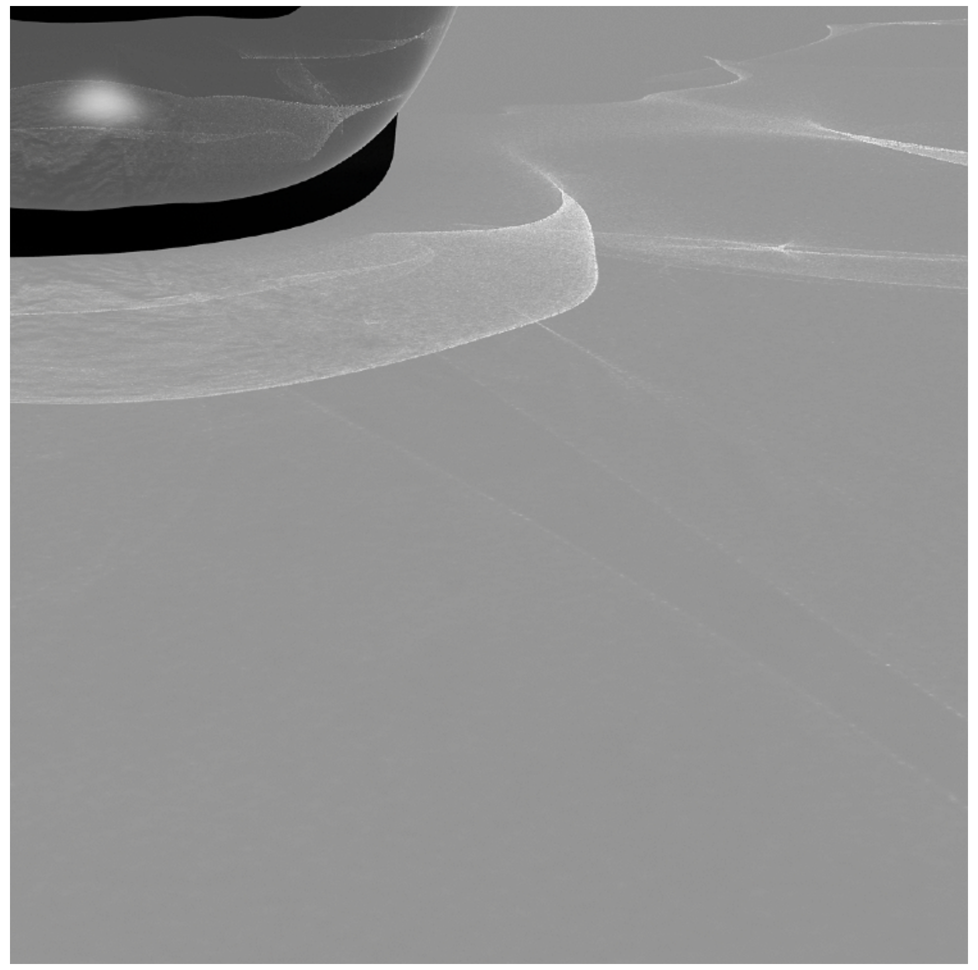


few irregular vertices

$\Longrightarrow$ smoothly-varying surface normals

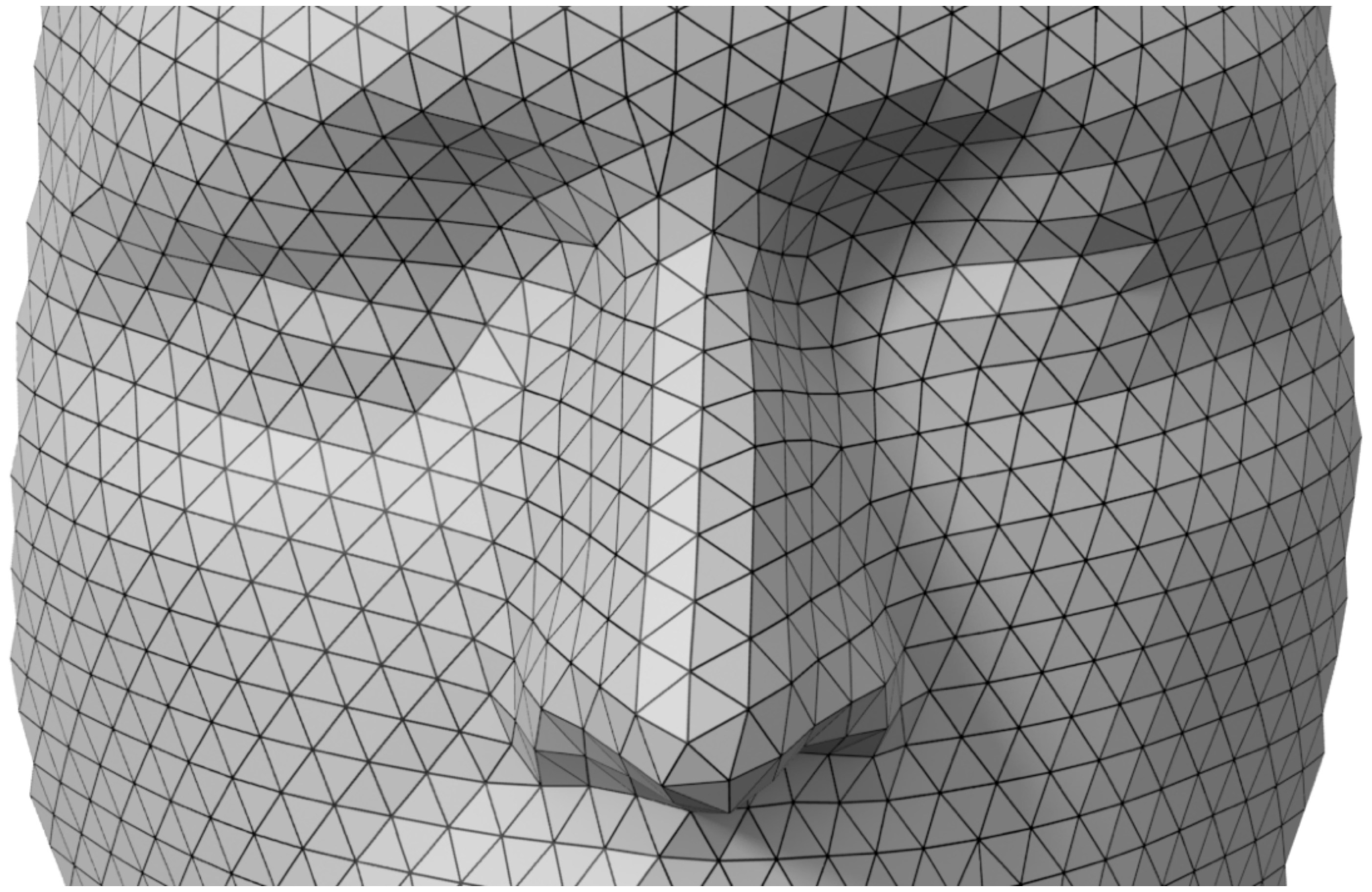


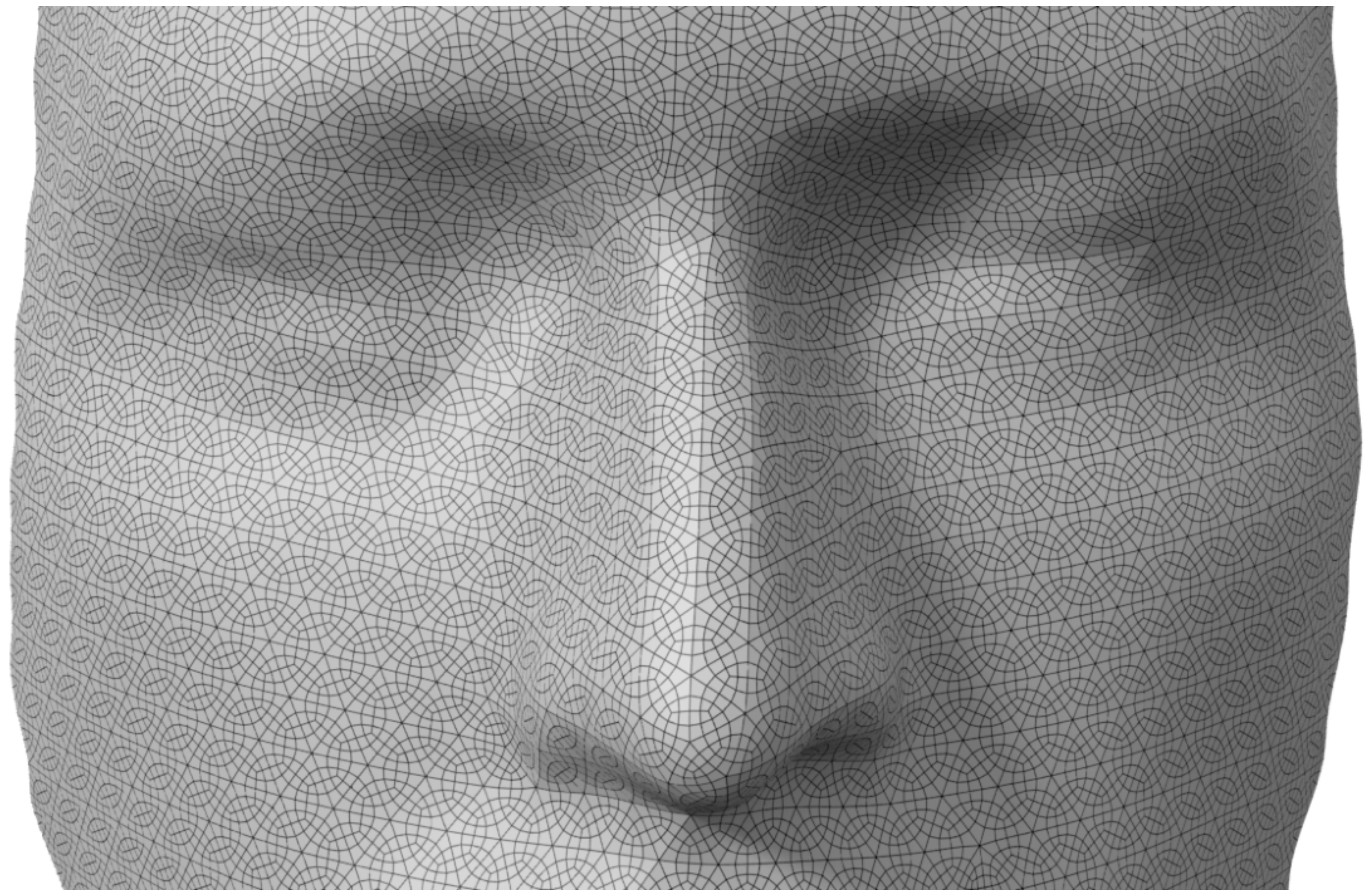
smooth
reflection lines



smooth
caustics

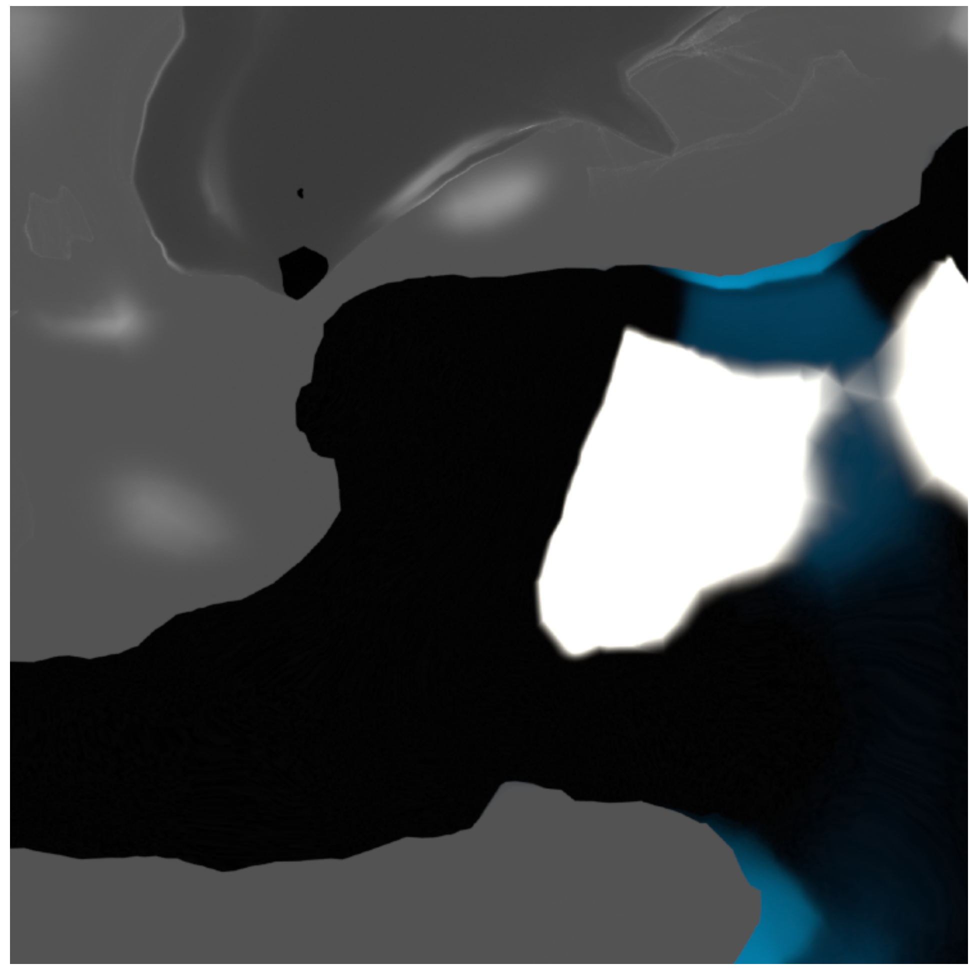
Catmull-Clark on triangle mesh



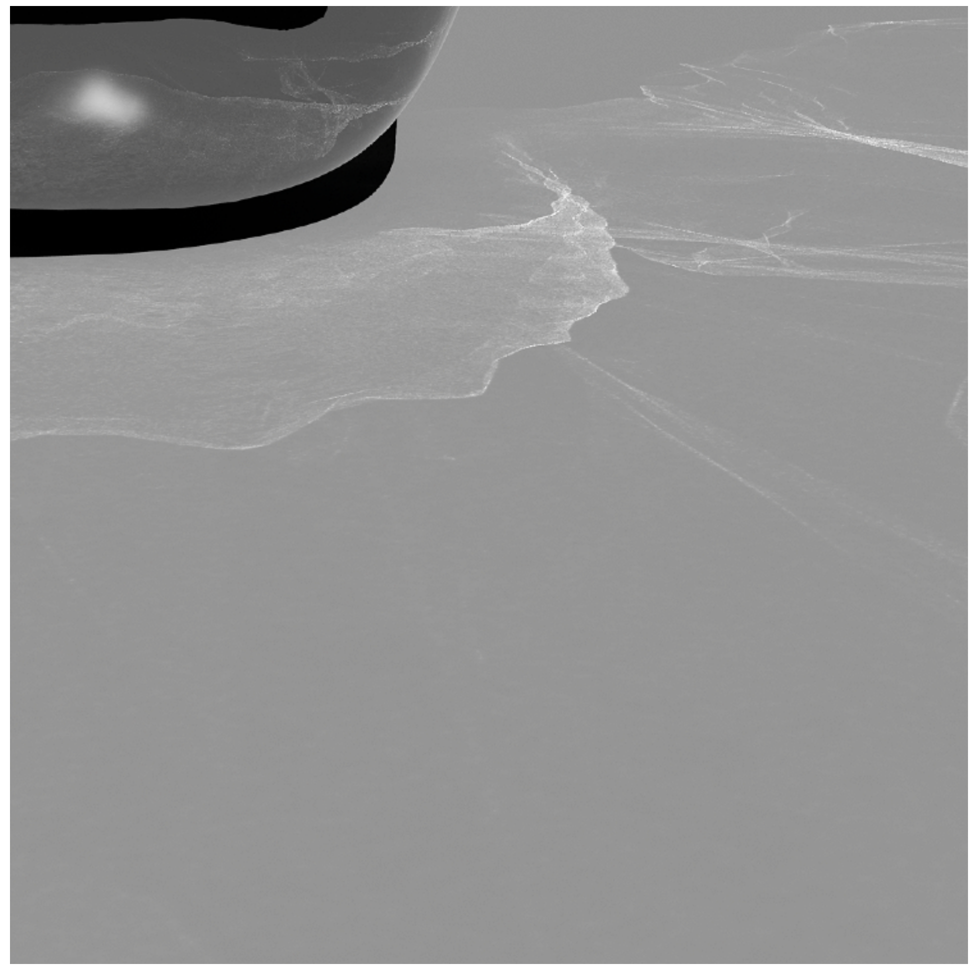


many irregular vertices

$\Longrightarrow$ erratic surface normals



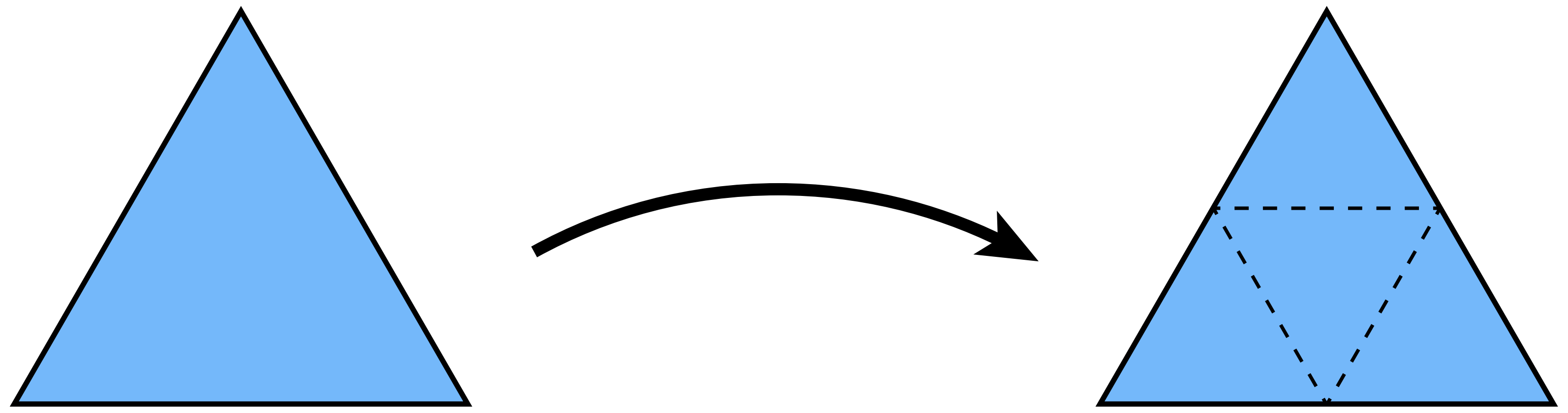
jagged
reflection lines



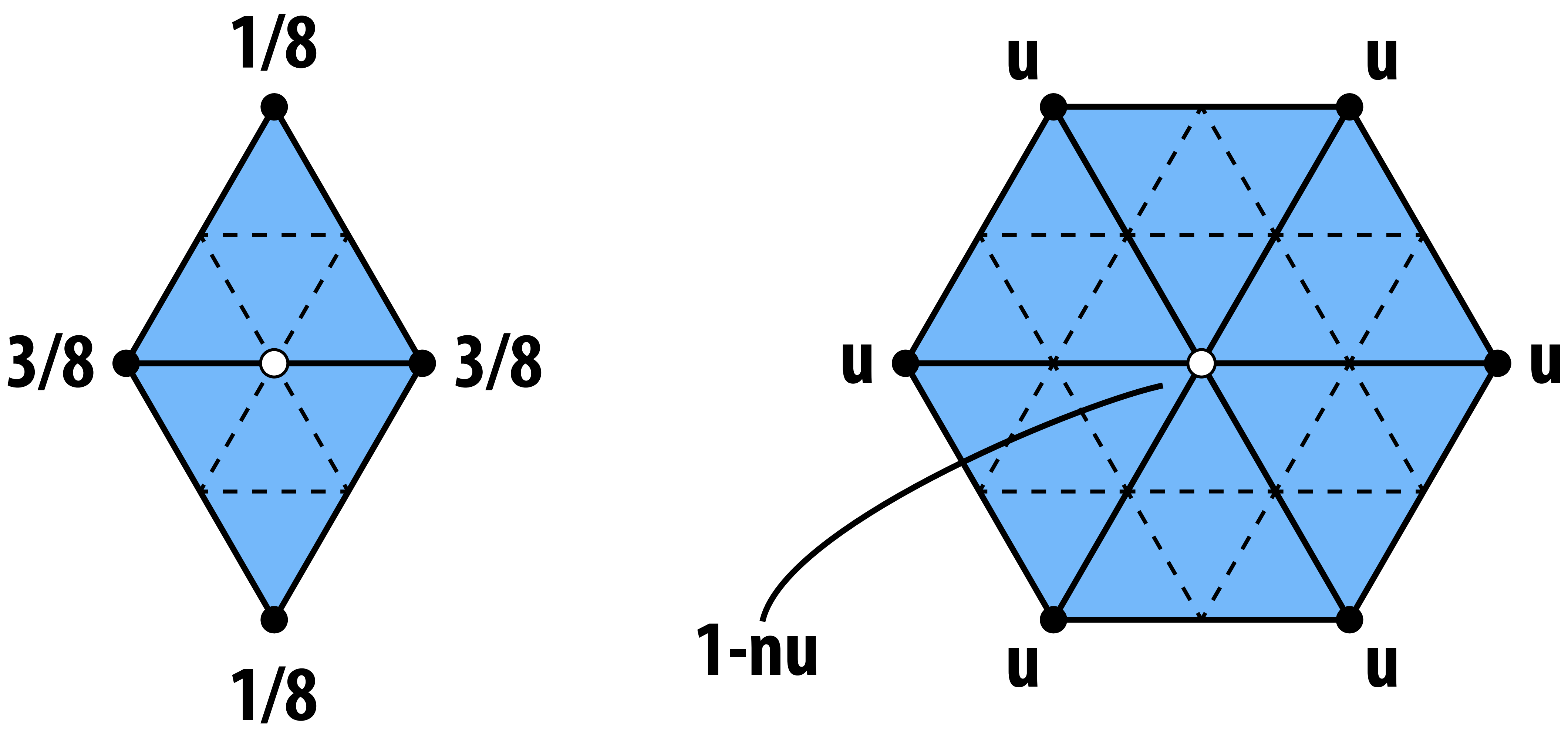
jagged
caustics

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:

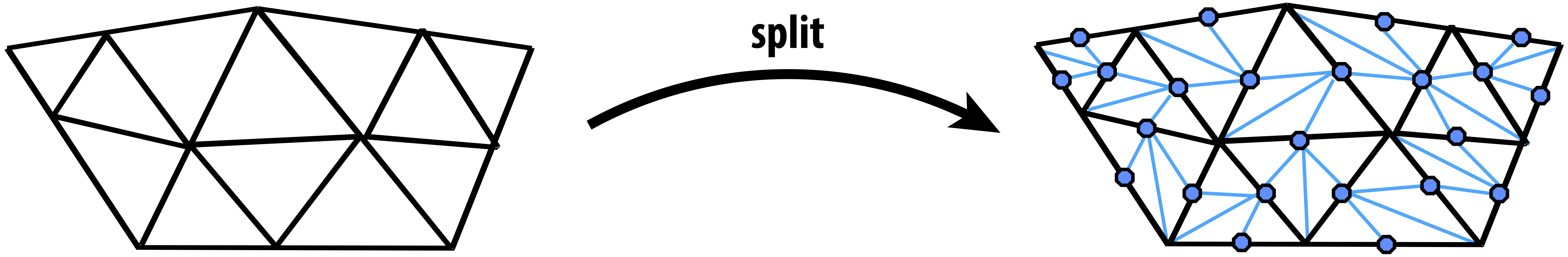


n: vertex degree

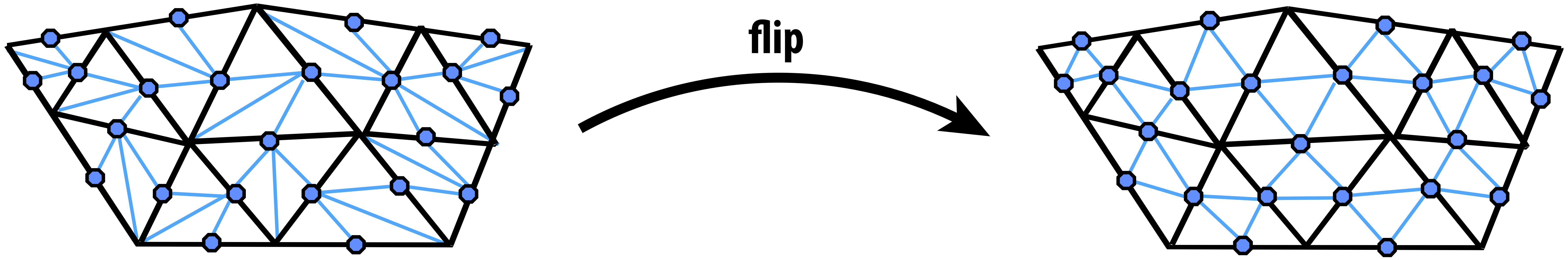
u: 3/16 if n=3, 3/(8n) otherwise

Loop Subdivision via Edge Operations

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



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



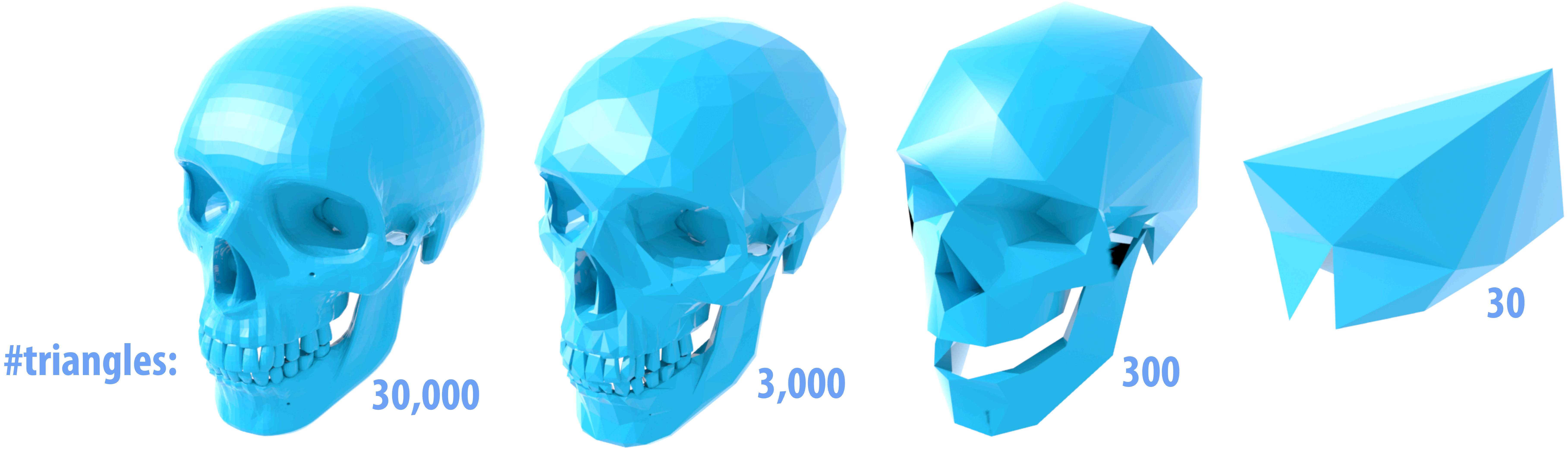
(Don't forget to update vertex positions!)

Images cribbed from Denis Zorin.

What if we want *fewer* triangles?

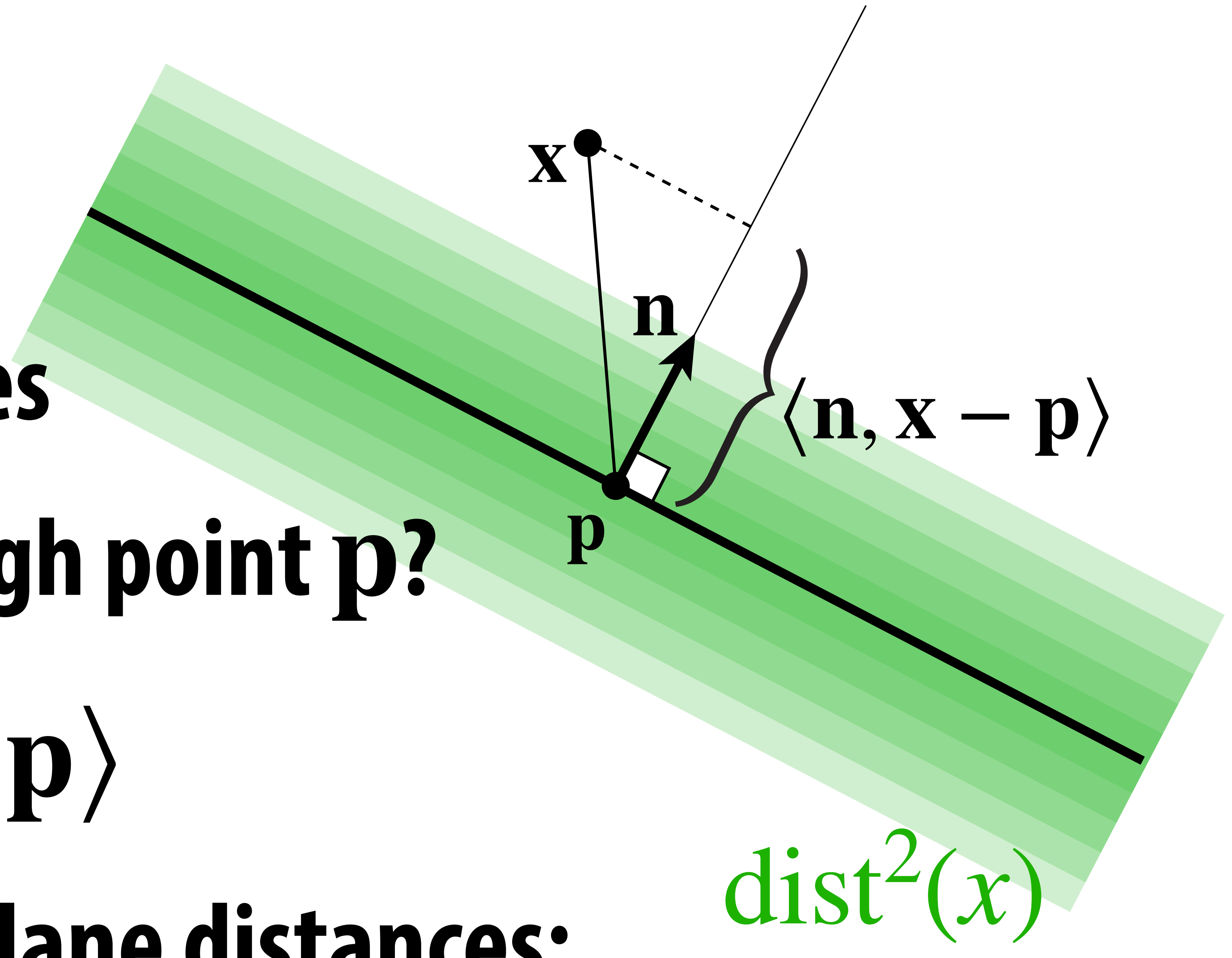
Simplification via Edge Collapse

- 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**

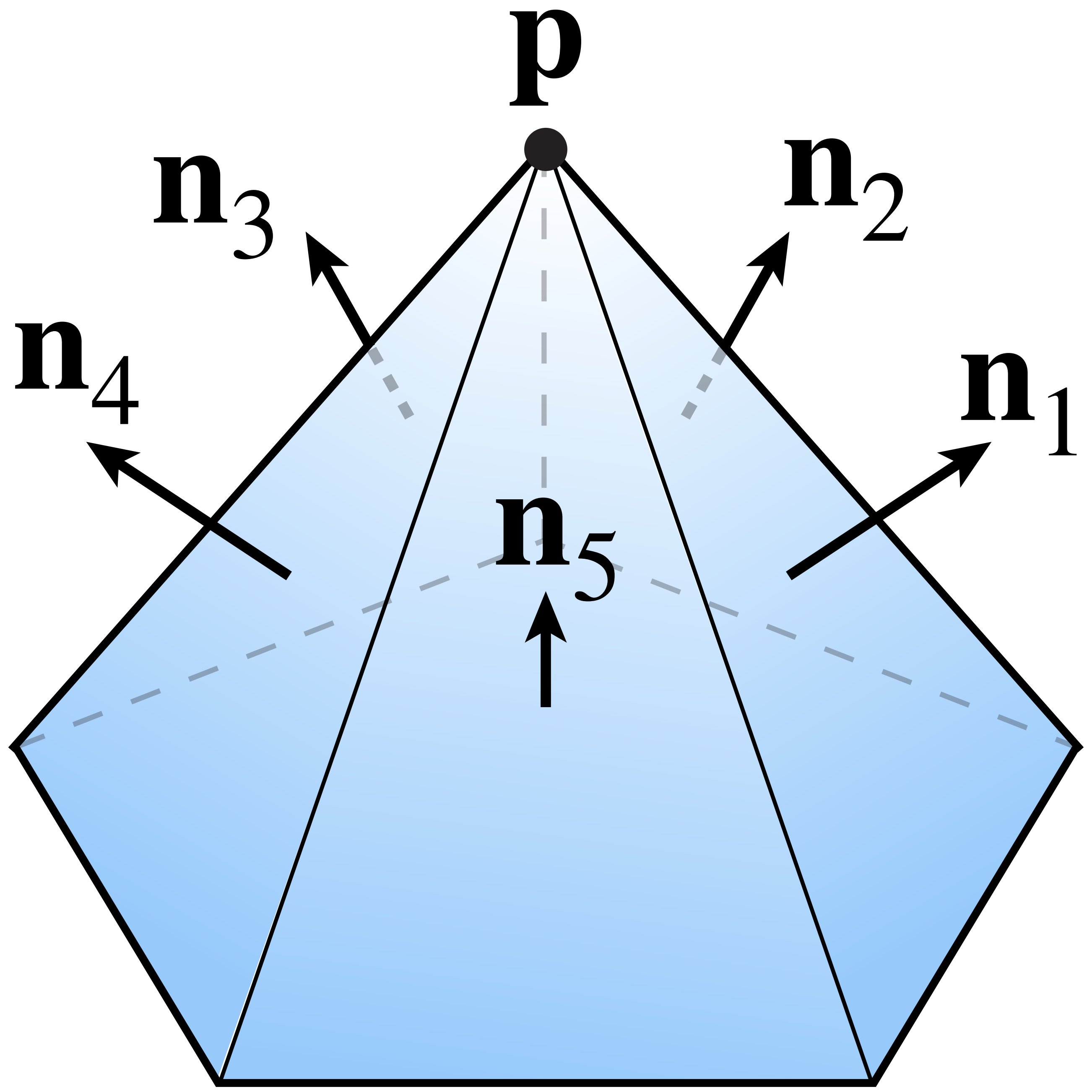


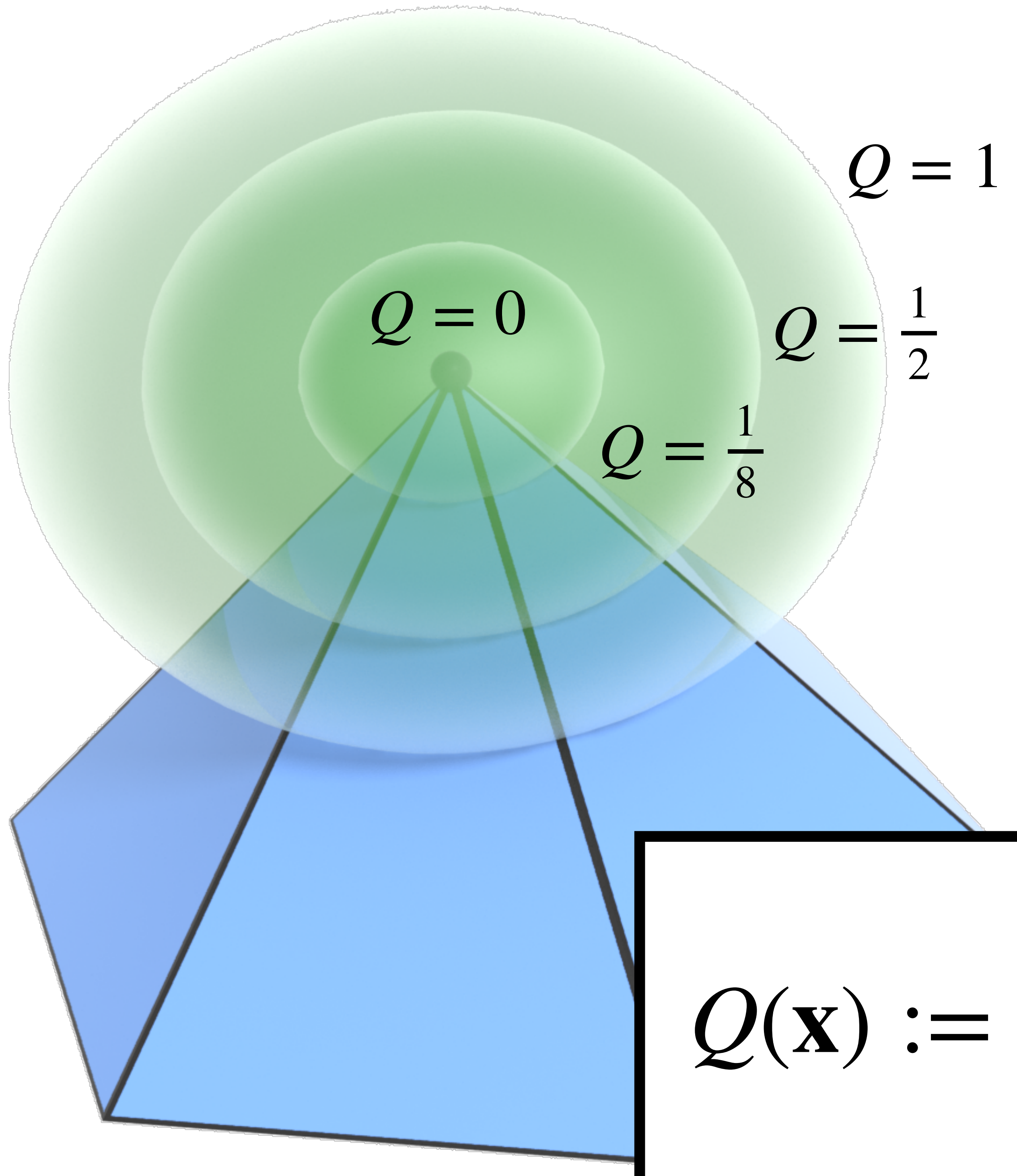
*invented at CMU (Garland & Heckbert 1997)

Quadric Error Metric



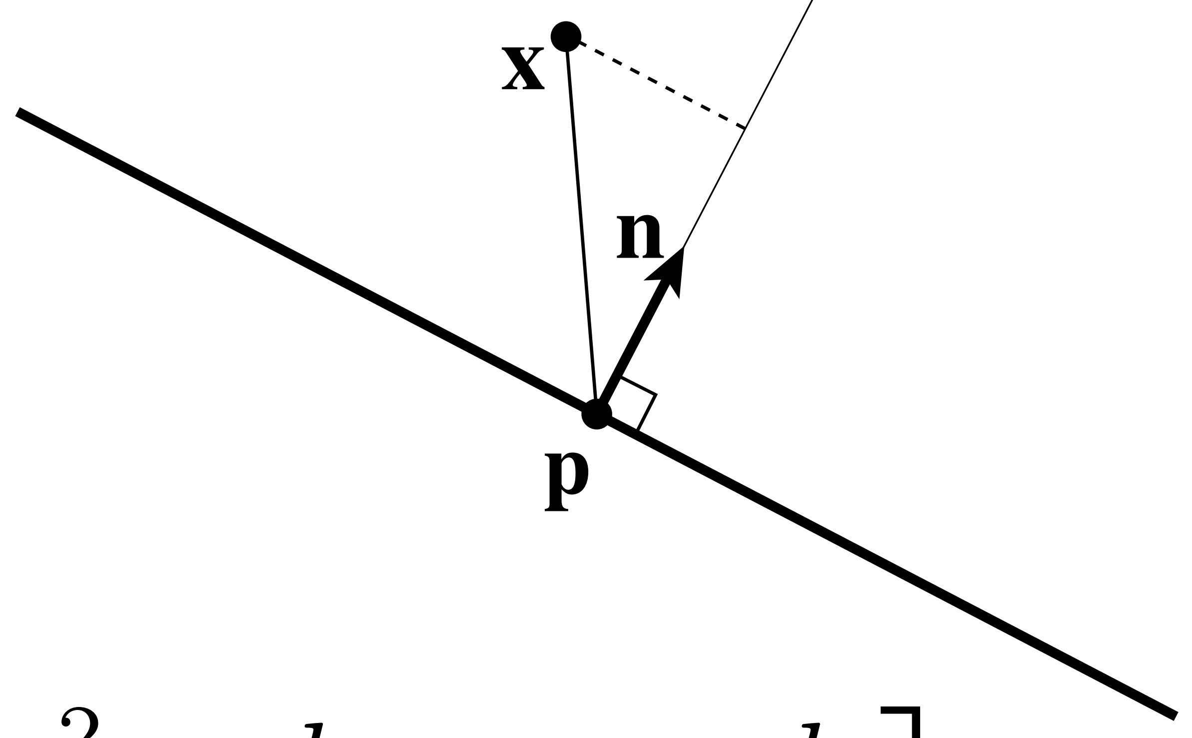
- Approximate distance to a collection of triangles
- Q: Distance to plane w/ normal $\mathbf{n}$ passing through point $\mathbf{p}$?
- A: $\text{dist}(\mathbf{x}) = \langle \mathbf{n}, \mathbf{x} \rangle - \langle \mathbf{n}, \mathbf{p} \rangle = \langle \mathbf{n}, \mathbf{x} - \mathbf{p} \rangle$
- Quadric error is then sum of squared point-to-plane distances:





$$Q(\mathbf{x}) := \sum_{i=1}^{k} \langle \mathbf{n}_i, \mathbf{x} - \mathbf{p} \rangle^2$$

Quadric Error - Homogeneous Coordinates



- Suppose in coordinates we have
  - a query point $\mathbf{x} = (x, y, z)$
  - a normal $\mathbf{n} = (a, b, c)$
  - an offset $d := \langle \mathbf{n}, \mathbf{p} \rangle$
- In homogeneous coordinates, let
  - $\mathbf{u} := (x, y, z, 1)$
  - $\mathbf{v} := (a, b, c, d)$

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

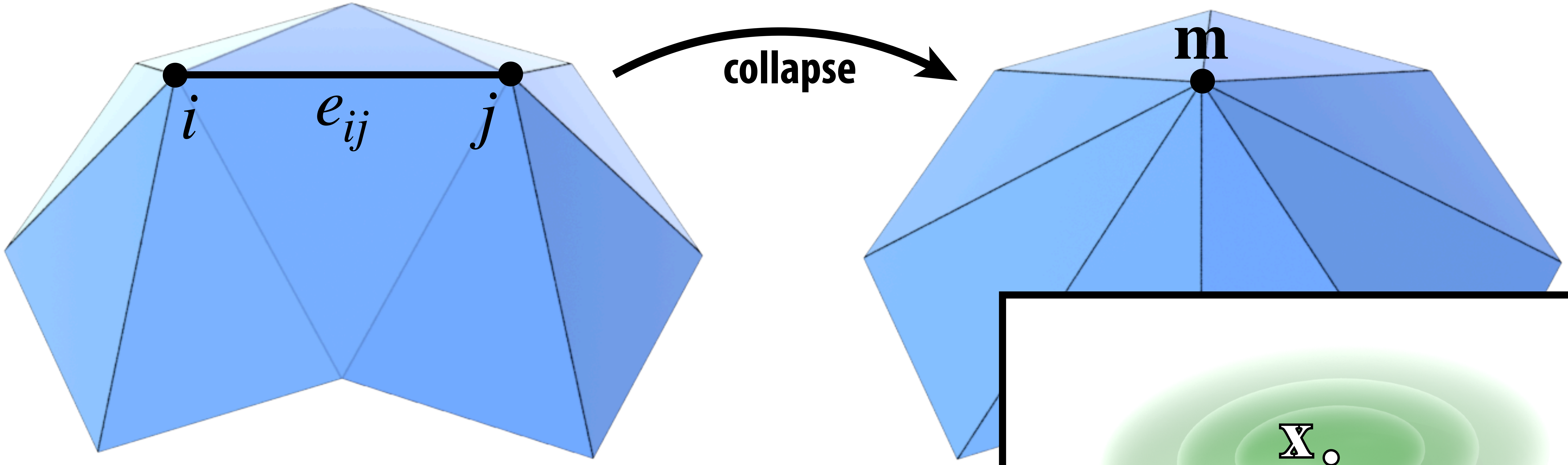
- Signed distance to plane is then just $\langle \mathbf{u}, \mathbf{v} \rangle = ax + by + cz + d$
- Squared distance is $\langle \mathbf{u}, \mathbf{v} \rangle^2 = \mathbf{u}^\mathsf{T}(\mathbf{v}\mathbf{v}^\mathsf{T})\mathbf{u} =: \mathbf{u}^\mathsf{T} K \mathbf{u}$
- Matrix $K = \mathbf{v}\mathbf{v}^T$ encodes squared distance to plane

Key idea: sum of matrices $K \iff$ distance to union of planes

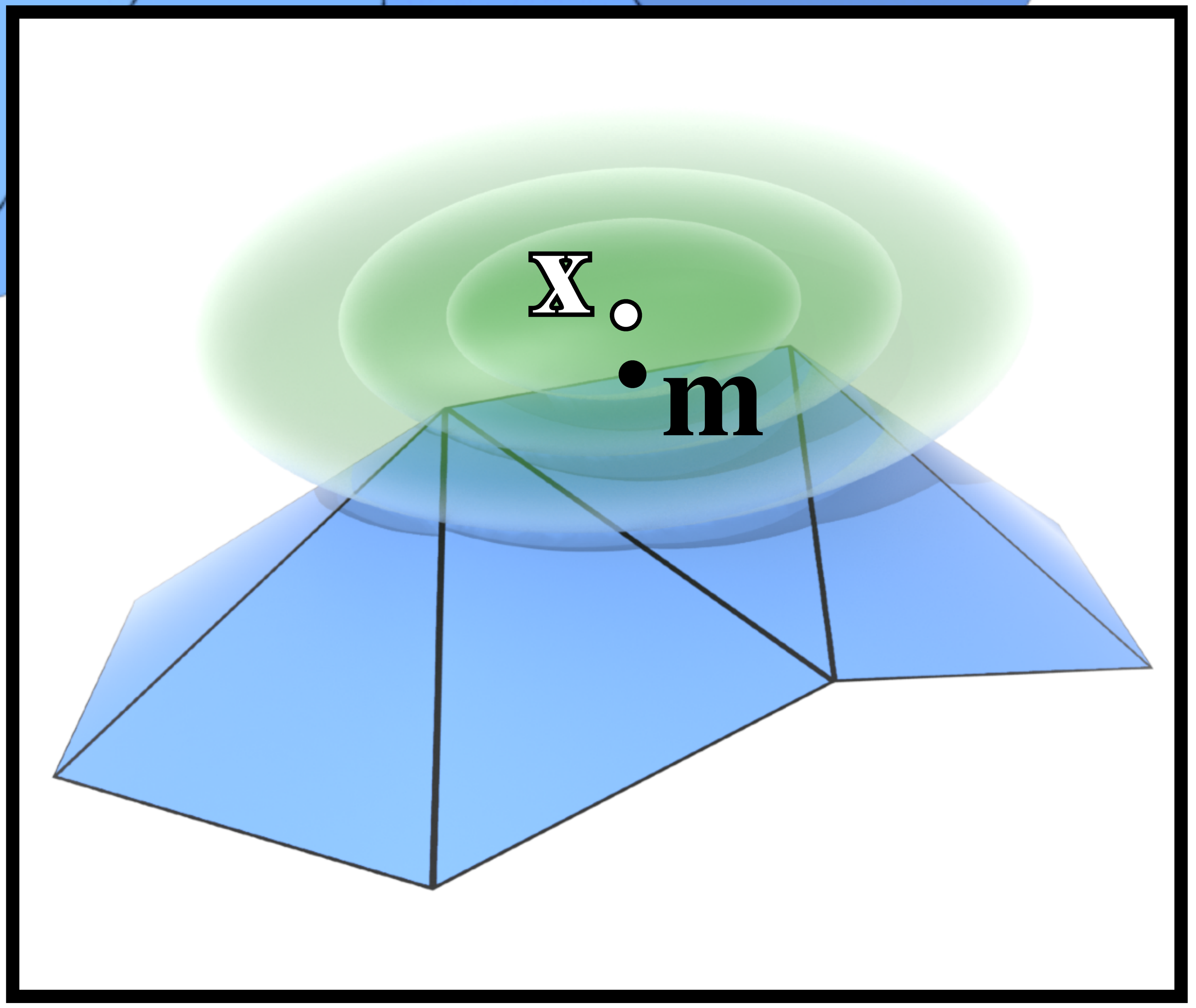
$$\mathbf{u}^\mathsf{T} K_1 \mathbf{u} + \mathbf{u}^\mathsf{T} K_2 \mathbf{u} = \mathbf{u}^\mathsf{T}(K_1 + K_2)\mathbf{u}$$

Quadric Error of Edge Collapse

- How much does it cost to collapse an edge e_{ij}?
- Idea: compute midpoint $\mathbf{m}$, measure error $Q(\mathbf{m}) = \mathbf{m}^\mathsf{T}(K_i + K_j)\mathbf{m}$
- Error becomes "score" for e_{ij}, determining priority



- Better idea: find point $\mathbf{x}$ that *minimizes error*!
- Ok, but how do we minimize quadric error?



Review: Minimizing a Quadratic Function

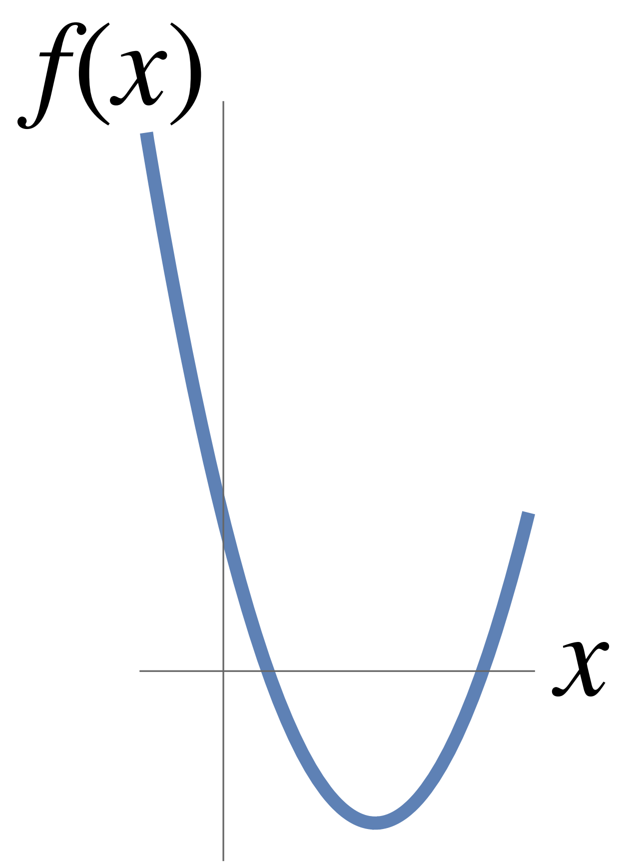
- Suppose you have a function $f(x) = ax^2 + bx + c$
- Q: What does the graph of this function look like?
- Could also look like this!
- Q: How do we find the *minimum?*
- A: Find where the function looks "flat" if we zoom in really close
- I.e., find point x where 1st derivative vanishes:

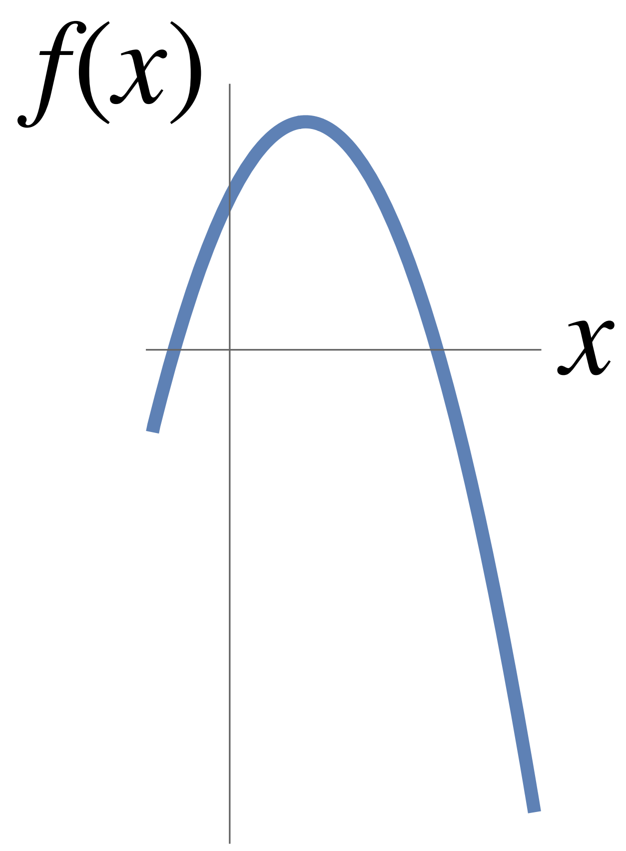
$$f'(x) = 0$$

$$2ax + b = 0$$

$$x = -b/2a$$

(What does x describe for the second function?)





Minimizing Quadratic Polynomial

- Not much harder to minimize a quadratic polynomial in n variables
- Can always write in terms of a symmetric matrix A
- E.g., in 2D: $f(x,y) = ax^2 + bxy + cy^2 + dx + ey + g$

$$\mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix} \qquad A = \begin{bmatrix} a & b/2 \\ b/2 & c \end{bmatrix} \qquad \mathbf{u} = \begin{bmatrix} d \\ e \end{bmatrix}$$

$$f(x,y) = \mathbf{x}^\mathsf{T} A \mathbf{x} + \mathbf{u}^\mathsf{T}\mathbf{x} + g$$

(will have this same form for any n)

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

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

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

(compare with our 1D solution)

$$x = -b/2a$$

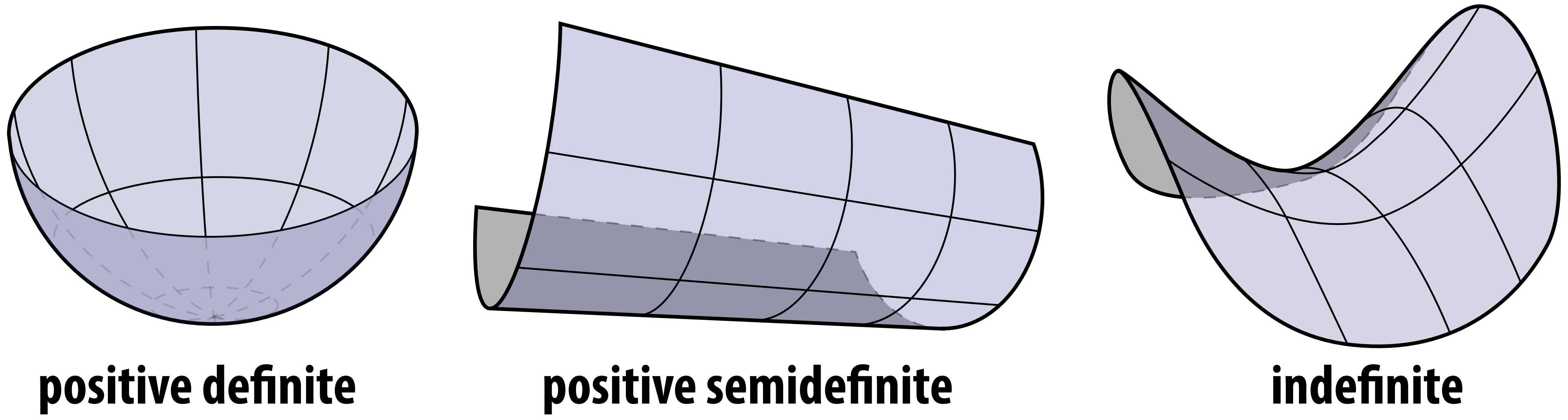
(Can you show this is true, at least in 2D?)

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:***

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

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



Positive-definiteness ***extremely important*** in computer graphics: means we can find minimizers by solving <u>linear</u> equations. Starting point for many algorithms (geometry processing, simulation, ...)

Minimizing Quadric Error

- Find "best" point for edge collapse by minimizing quadratic form

$$\min_{\mathbf{u}\in\mathbb{R}^4} \mathbf{u}^T K \mathbf{u}$$

- Already know fourth (homogeneous) coordinate for a point is 1
- So, break up our quadratic function into two pieces:

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

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

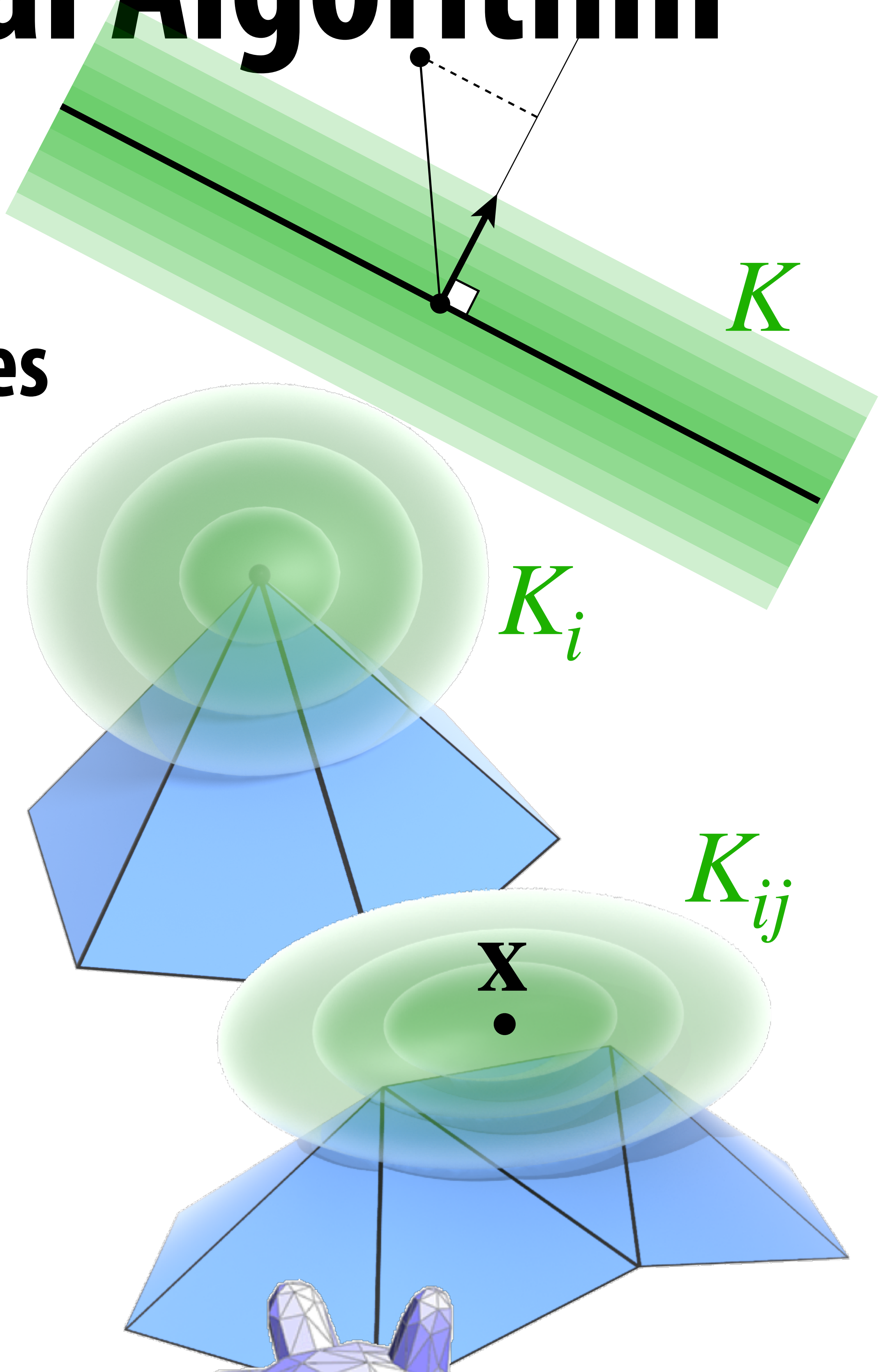
- Now we have a quadratic polynomial in the unknown position $\mathbf{x} \in \mathbb{R}^3$
- Can minimize as before:

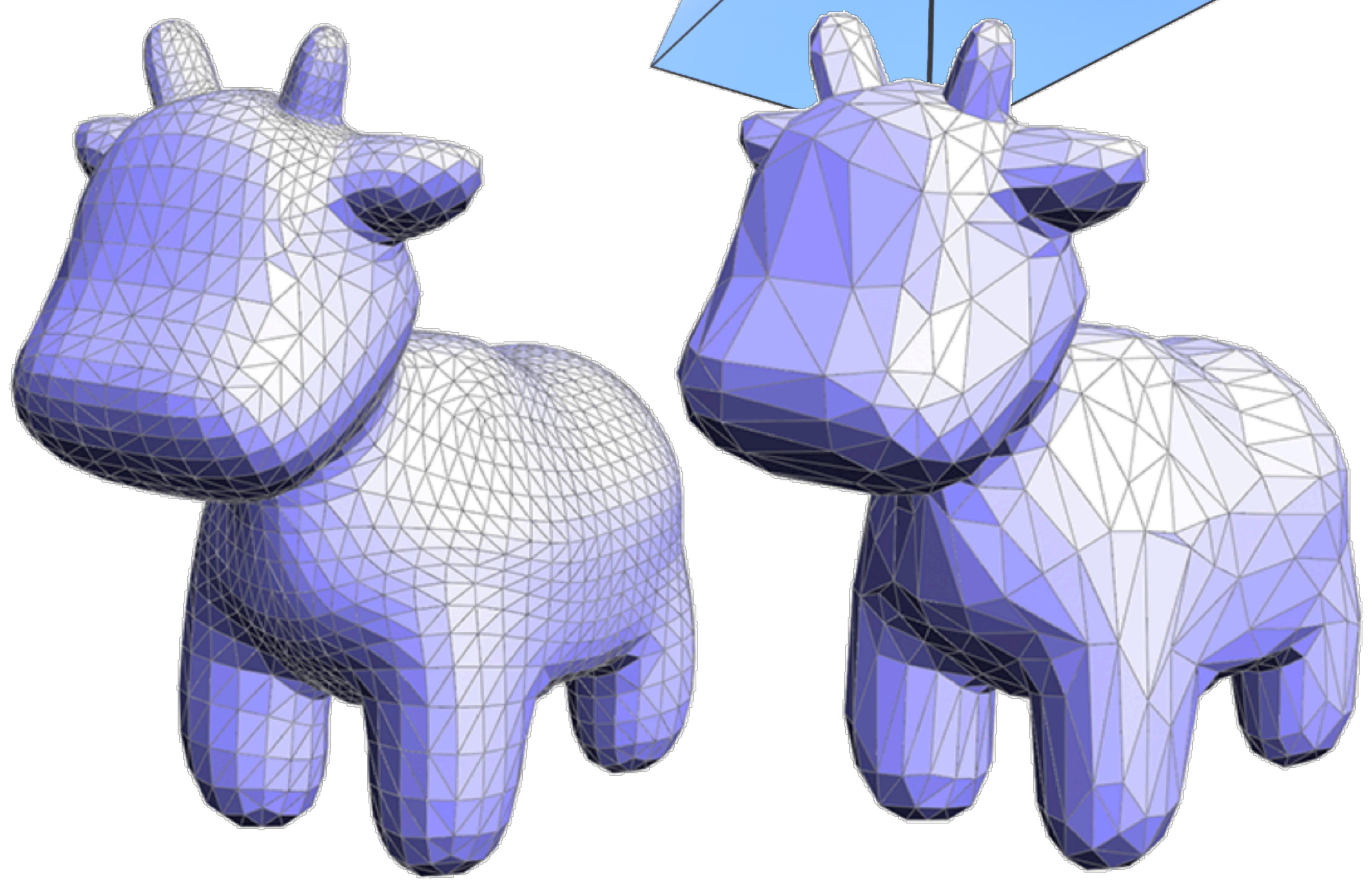
$$2B\mathbf{x} + 2\mathbf{w} = 0 \qquad \iff \qquad \mathbf{x} = -B^{-1}\mathbf{w}$$

Q: Why should B be positive-definite?

Quadric Error Simplification: Final Algorithm

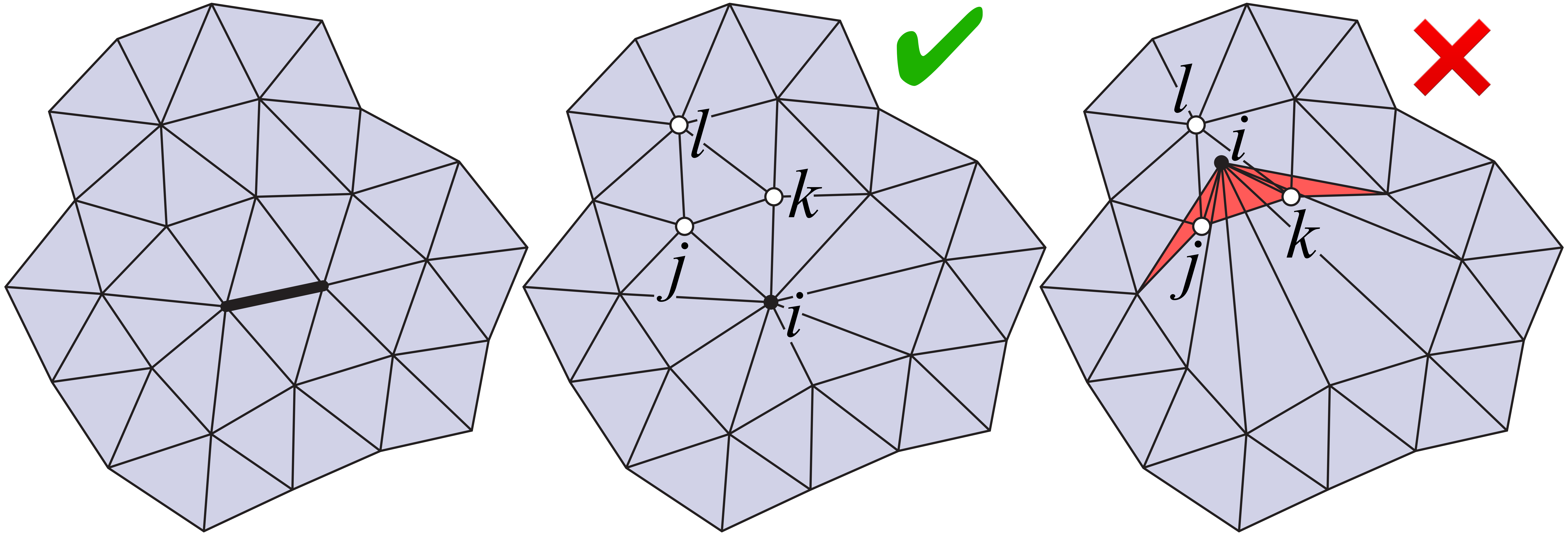
- Compute K for each triangle (squared distance to plane)
- Set K_i at each vertex to sum of Ks from incident triangles
- For each edge e_{ij}:
  - set $K_{ij} = K_i + K_j$
  - find point $\mathbf{x}$ minimizing error, set cost to $K_{ij}(\mathbf{x})$
- Until we reach target number of triangles:
  - collapse edge e_{ij} with smallest cost to optimal point $\mathbf{x}$
  - set quadric at new vertex to K_{ij}
  - update cost of edges touching new vertex
- 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):

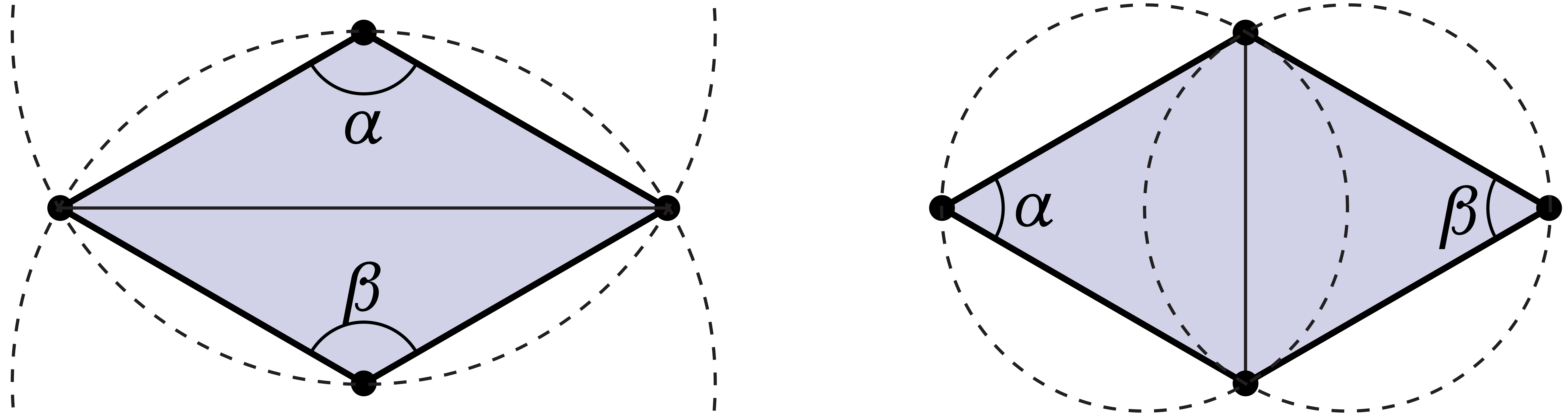


- Easy solution: for each triangle ijk touching collapsed vertex i, consider normals N_{ijk} and N_{kjl} (where kjl is other triangle containing edge jk)
- If $\langle N_{ijk}, N_{kjl} \rangle$ is negative, don't collapse this edge!

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

How do we make a mesh “more Delaunay”?

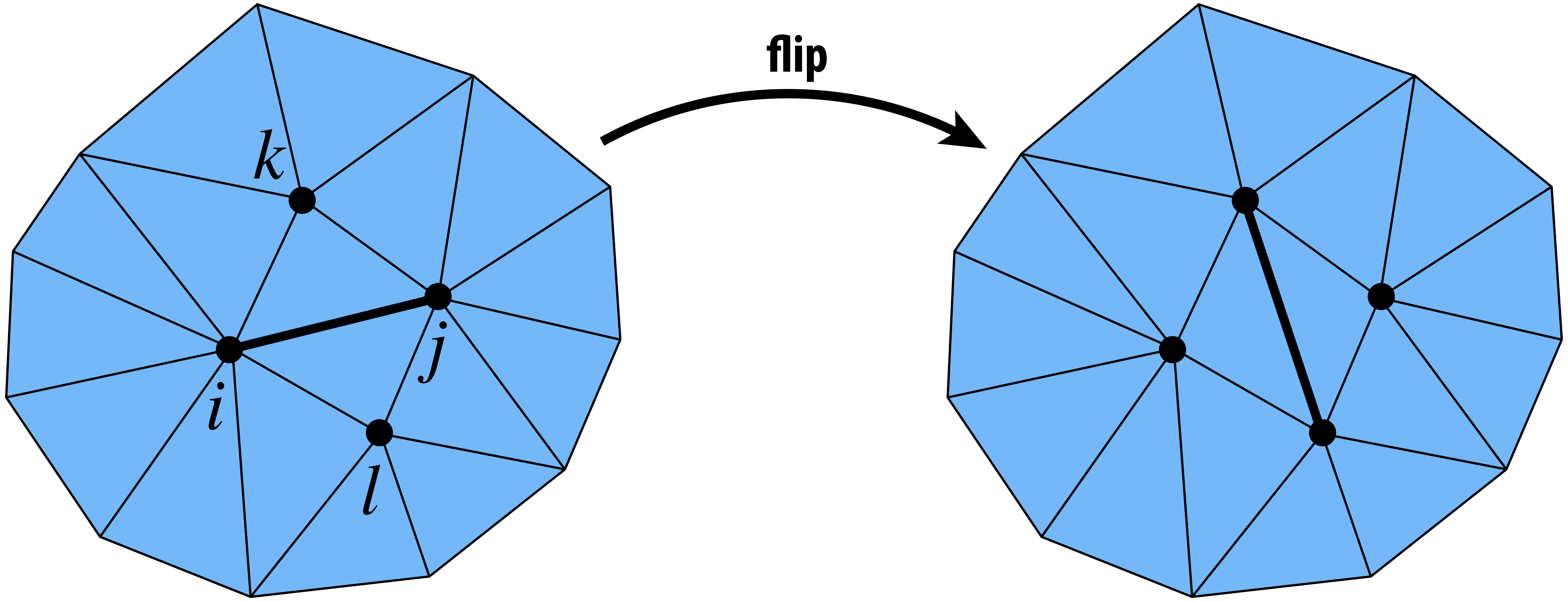
- **Already have a good tool: edge flips!**
- **If $\alpha + \beta > \pi$, flip it!**



- **FACT: in 2D, flipping edges eventually yields Delaunay mesh**
- **Theory: worst case $O(n^2)$; doesn’t always work 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!

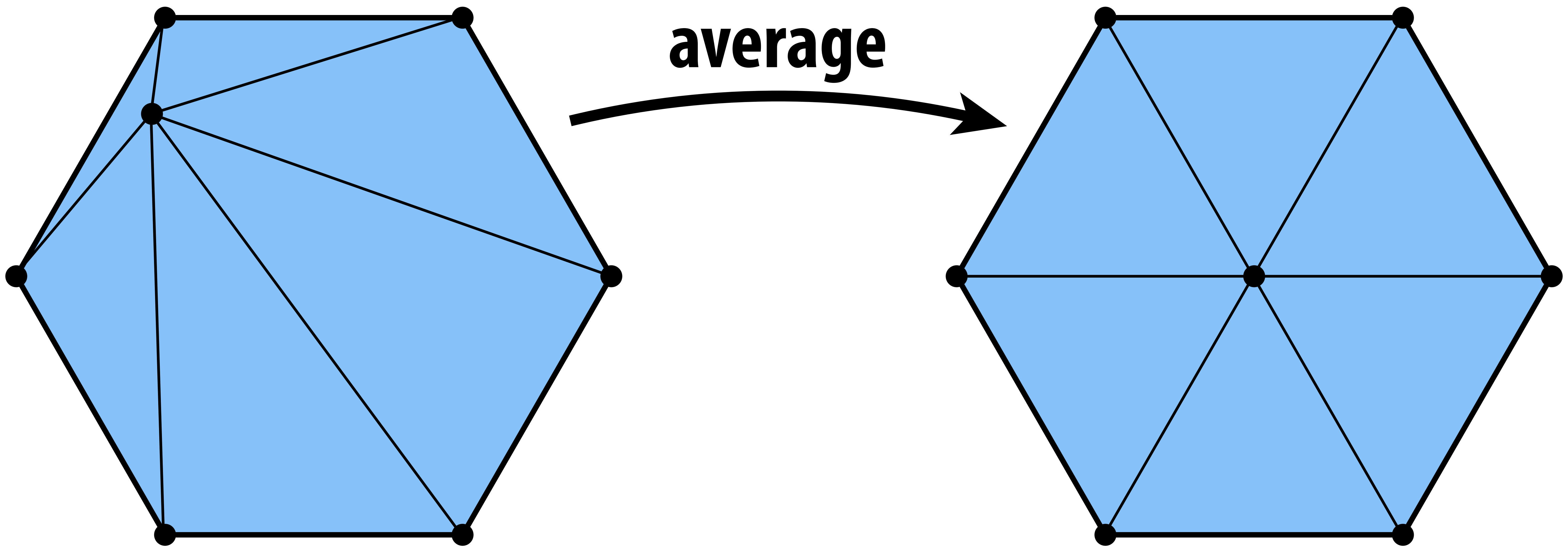


total deviation: $|d_i - 6| + |d_j - 6| + |d_k - 6| + |d_l - 6|$

- FACT: average degree approaches 6 as number of elements increases
- Iterative edge flipping acts like "discrete diffusion" of degree
- No (known) guarantees; works well in practice

How do we make a triangles “more round”?

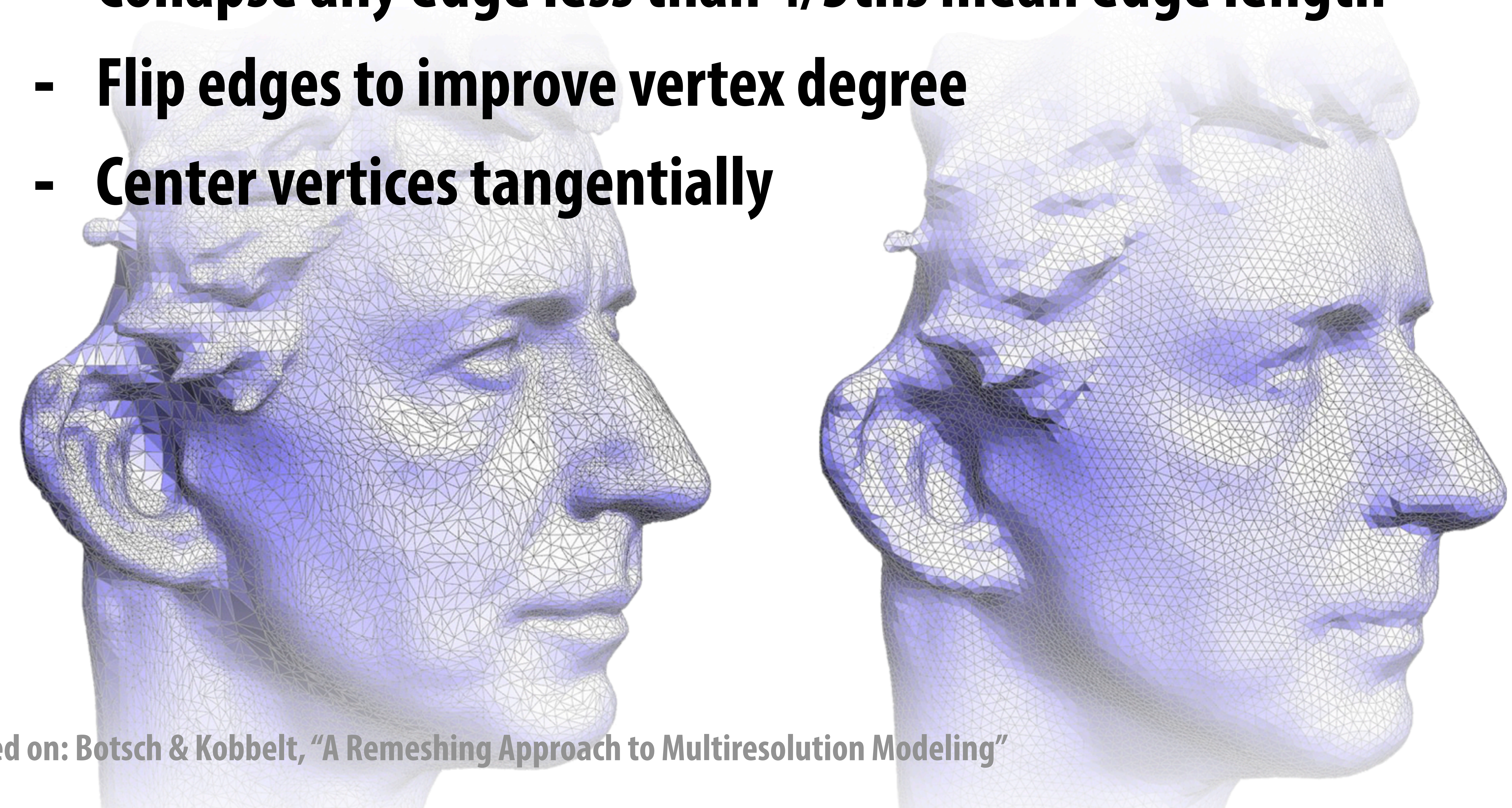
- Delaunay doesn’t guarantee triangles are “round” (angles near 60°)
- Can often improve shape by centering vertices:



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

Isotropic Remeshing Algorithm

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

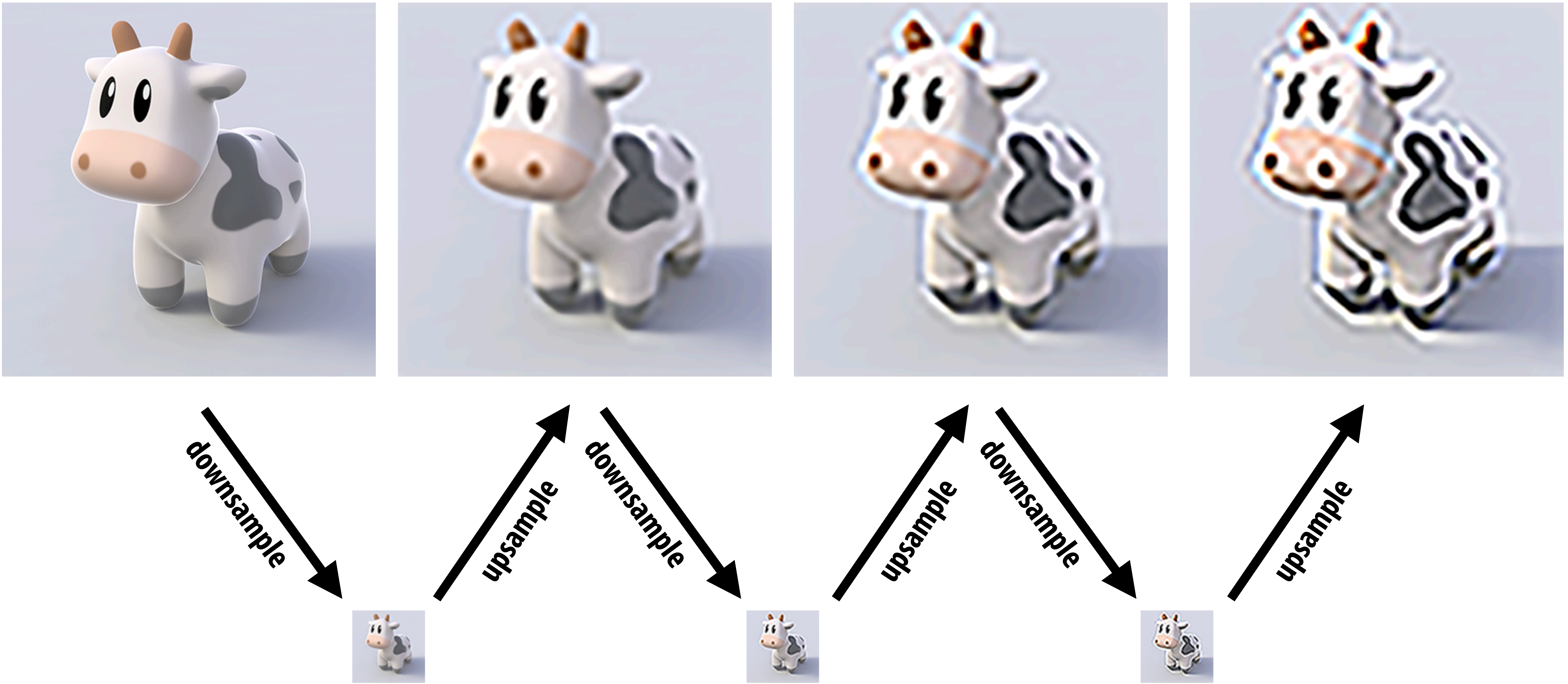


Based on: Botsch & Kobbelt, "A Remeshing Approach to Multiresolution Modeling"

What can go wrong when you resample a signal?

Danger of Resampling

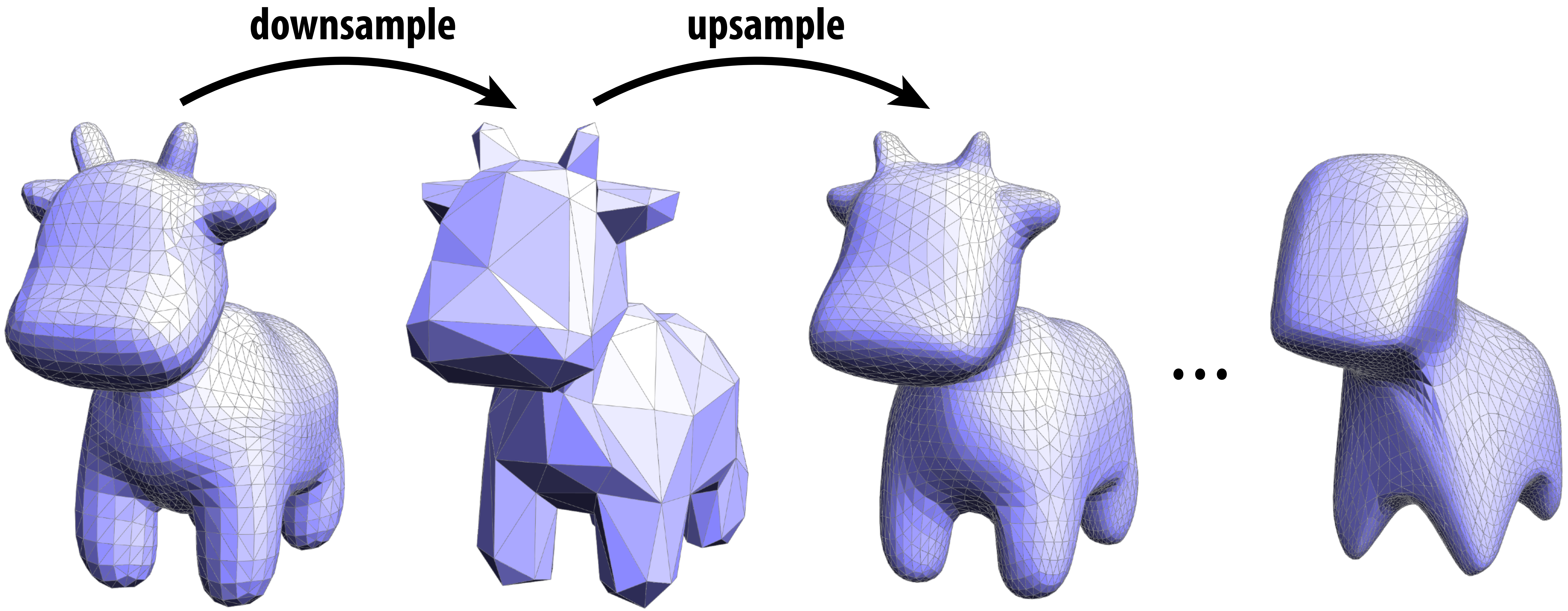
Q: What happens if we repeatedly resample an image?



A: Signal quality degrades!

Danger of Resampling

Q: What happens if we repeatedly resample a mesh?



A: Signal also degrades!

But wait: we have the original signal (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.
- Do implicit/explicit representations make such tasks easier?
- What's the cost of the naïve algorithm, and how do we <u>accelerate</u> such queries for large meshes?
- So many questions!