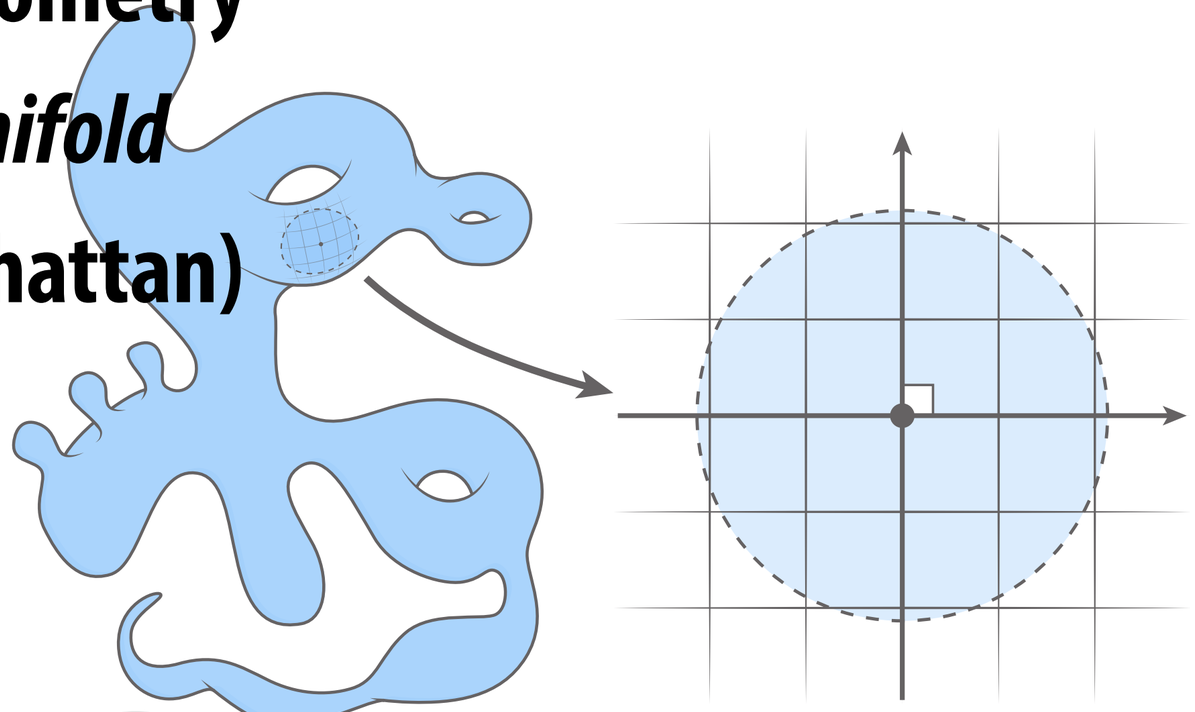
Lecture 7:

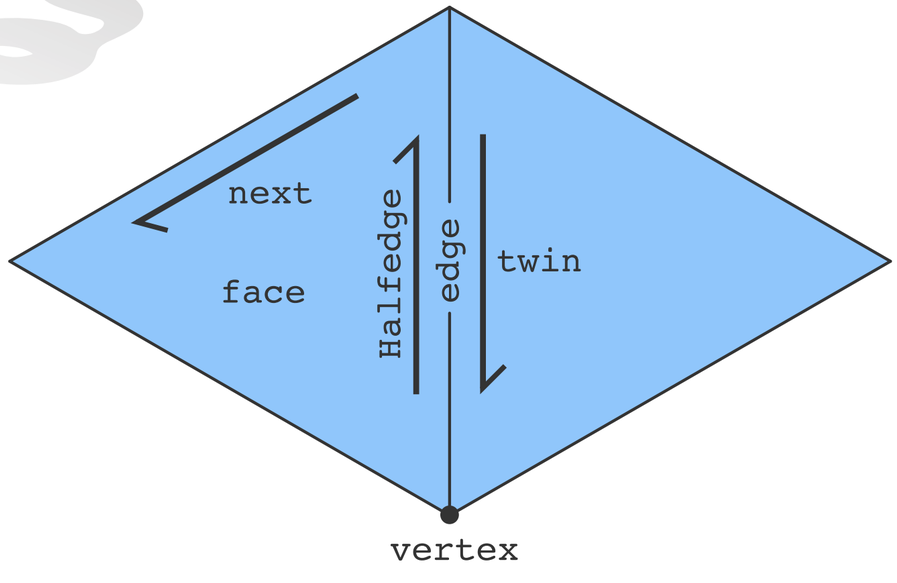
Geometry Processing

Computer Graphics
CMU 15-462/15-662, Fall 2015

Last time: Curves, Surfaces & Meshes

- Mathematical description of geometry
  - simplifying assumption: *manifold*
  - gives local coordinates (Manhattan)
- Data structures for surfaces
  - polygon soup
  - halfedge mesh
  - storage cost vs. access time, etc.
- Today:
  - how do we manipulate geometry?
  - geometry processing / resampling





Refresher: Polygon Soup

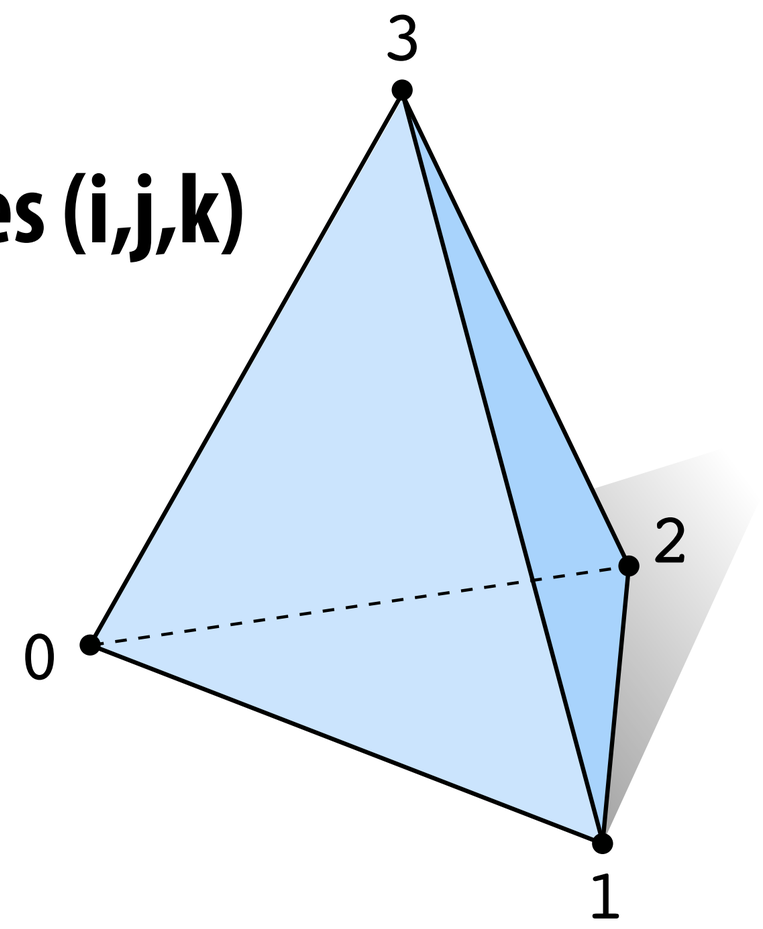
- Store triples of coordinates (x,y,z) and indices (i,j,k)
- E.g., tetrahedron:

VERTICES

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

TRIANGLES

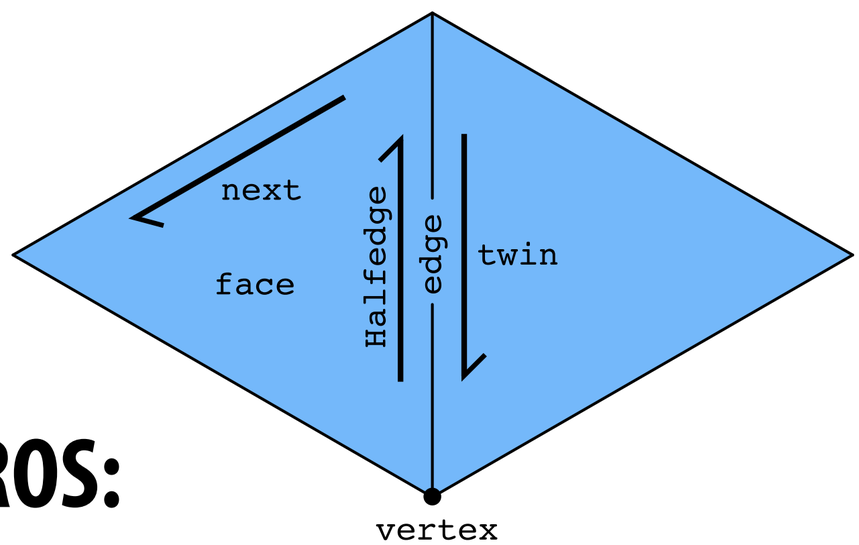
i	j	k
0	2	1
0	3	2
3	0	1
3	1	2

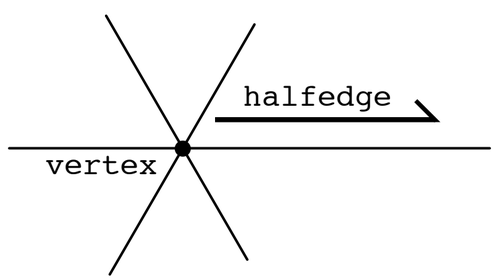


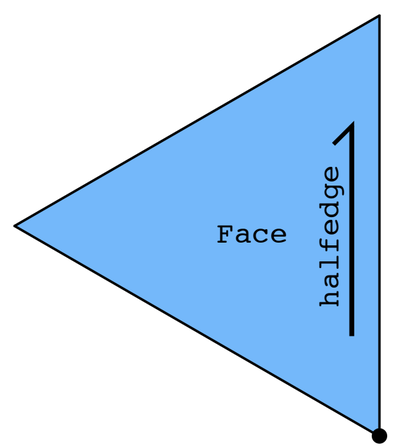
- PROS:
  - very low storage cost (how much?)
  - coherent memory access
  - can easily represent nonmanifold geometry
- CONS:
  - very expensive to access neighbors
  - very difficult to change connectivity (how?)

Refresher: Halfedge Mesh

- **Connectivity encoded via *halfedges***







- **PROS:**
  - **easy to access neighbors**
  - **easy to change connectivity**
- **CONS:**
  - **greater storage cost**
  - **less coherent memory access**
  - **can only represent manifold, orientable surfaces**

Food for thought: Incidence Matrices

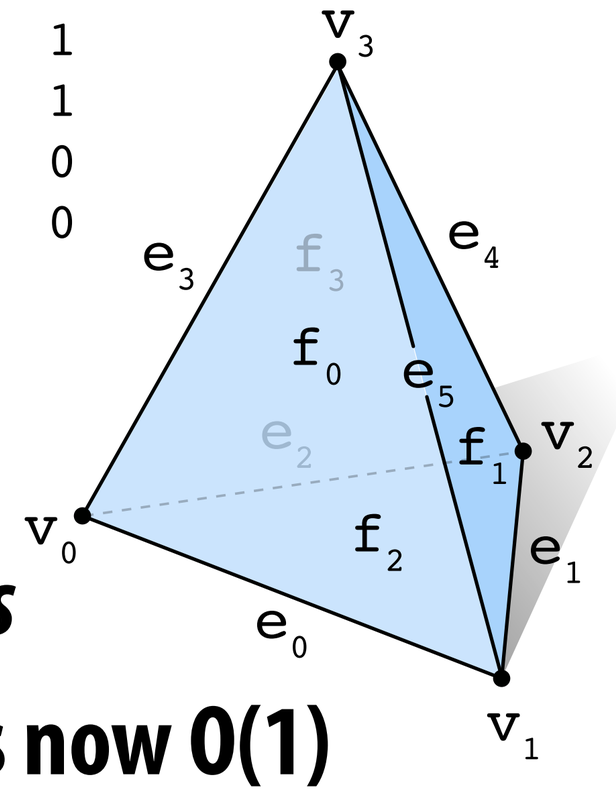
- If we want to answer neighborhood queries, why not simply store a list of neighbors?
- Can encode all neighbor information via *incidence matrices*
- E.g., tetrahedron:

VERTEX↔EDGE

	v0	v1	v2	v3
e0	1	1	0	0
e1	0	1	1	0
e2	1	0	1	0
e3	1	0	0	1
e4	0	0	1	1
e5	0	1	0	1

EDGE↔FACE

	e0	e1	e2	e3	e4	e5
f0	1	0	0	1	0	1
f1	0	1	0	0	1	1
f2	1	1	1	0	0	0
f3	0	0	1	1	1	0



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

Comparison of Polygon Mesh Data Strucutres

	Polygon Soup	Incidence Matrices	Halfedge Mesh
storage cost*	~3 x #vertices	~33 x #vertices	~36 x #vertices
constant-time neighborhood access?	NO	YES	YES
easy to add/remove mesh elements?	NO	NO	YES
nonmanifold geometry?	YES	YES	NO

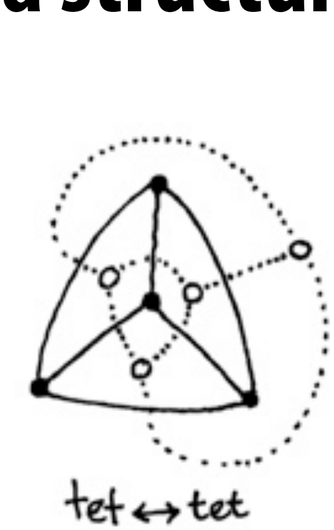
Conclusion: pick the right data structure for the job!

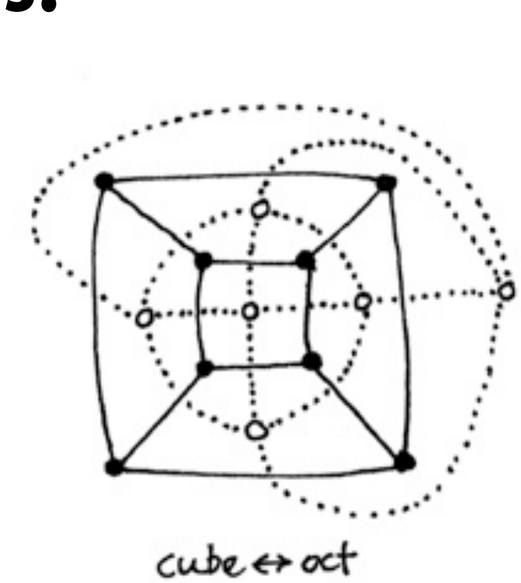
*number of integer values and/or pointers required to encode *connectivity*
(all data structures require same amount of storage for vertex positions)

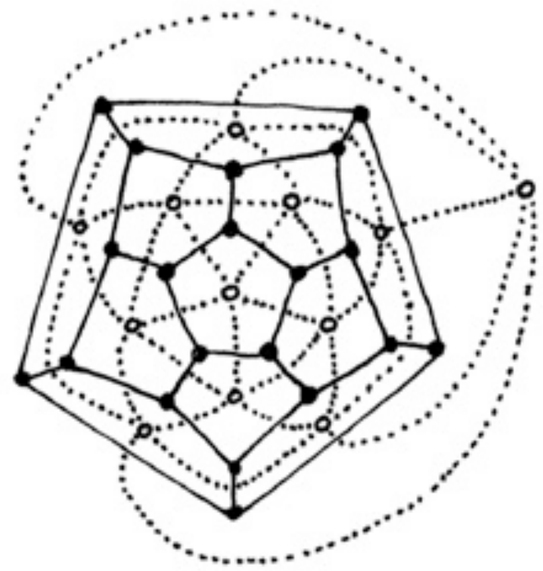
Even more alternatives to Halfedge

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

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





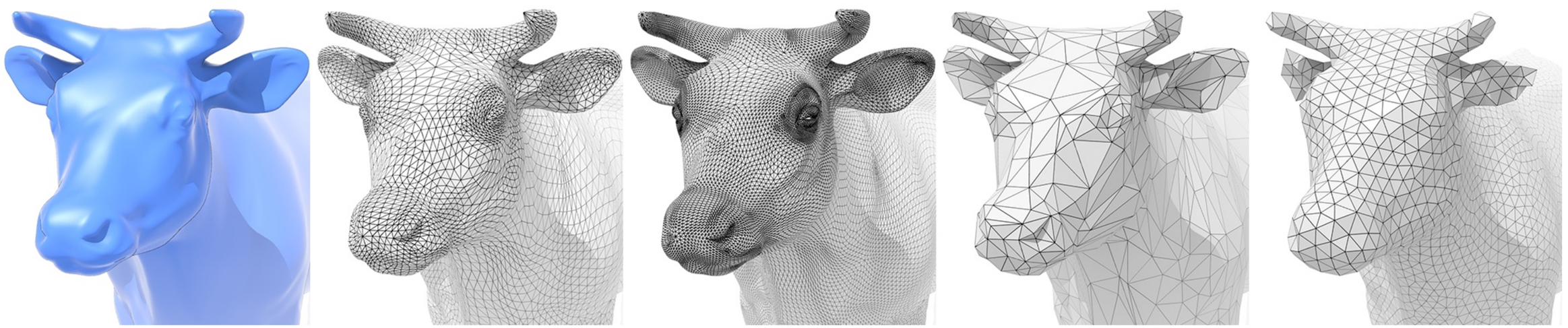


- Each stores local neighborhood information
- Similar tradeoffs relative to simple polygon list:
  - CONS: additional storage, incoherent memory access
  - PROS: better access time for individual elements, intuitive traversal of local neighborhoods
- (Food for thought: can you design a halfedge-like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

So, what's the job?

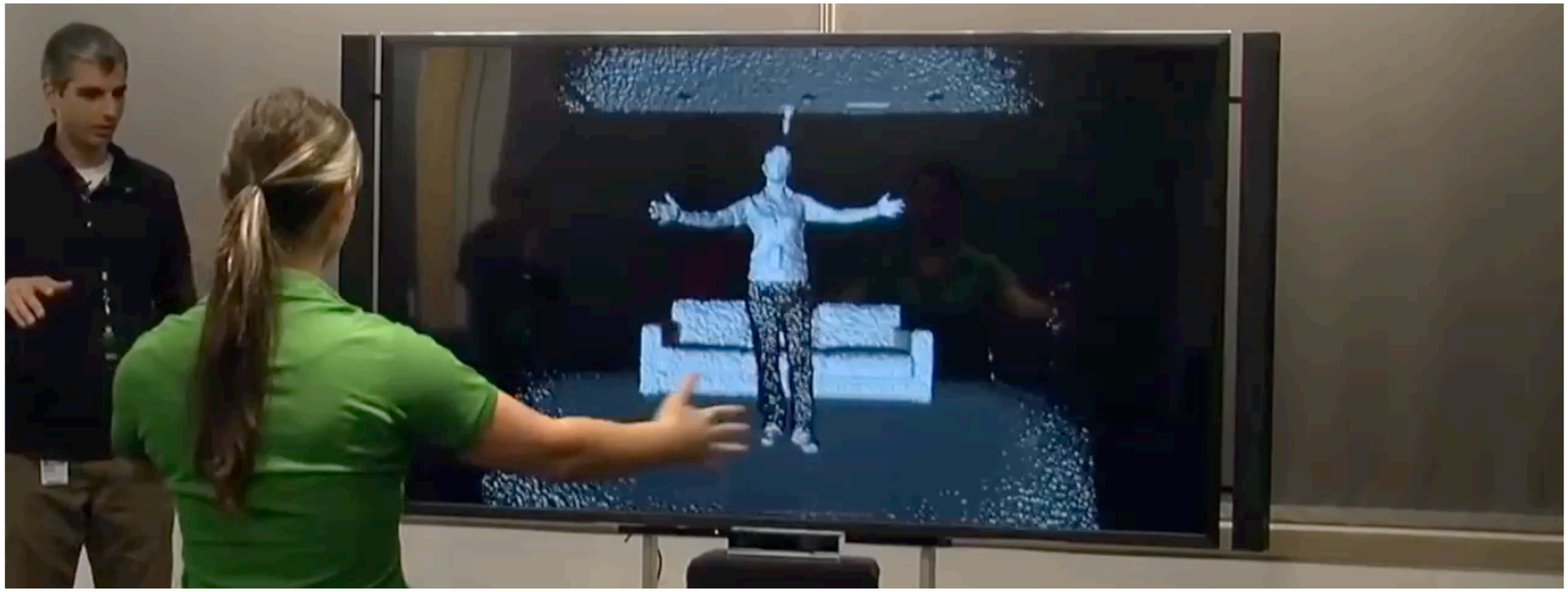
Digital Geometry Processing

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

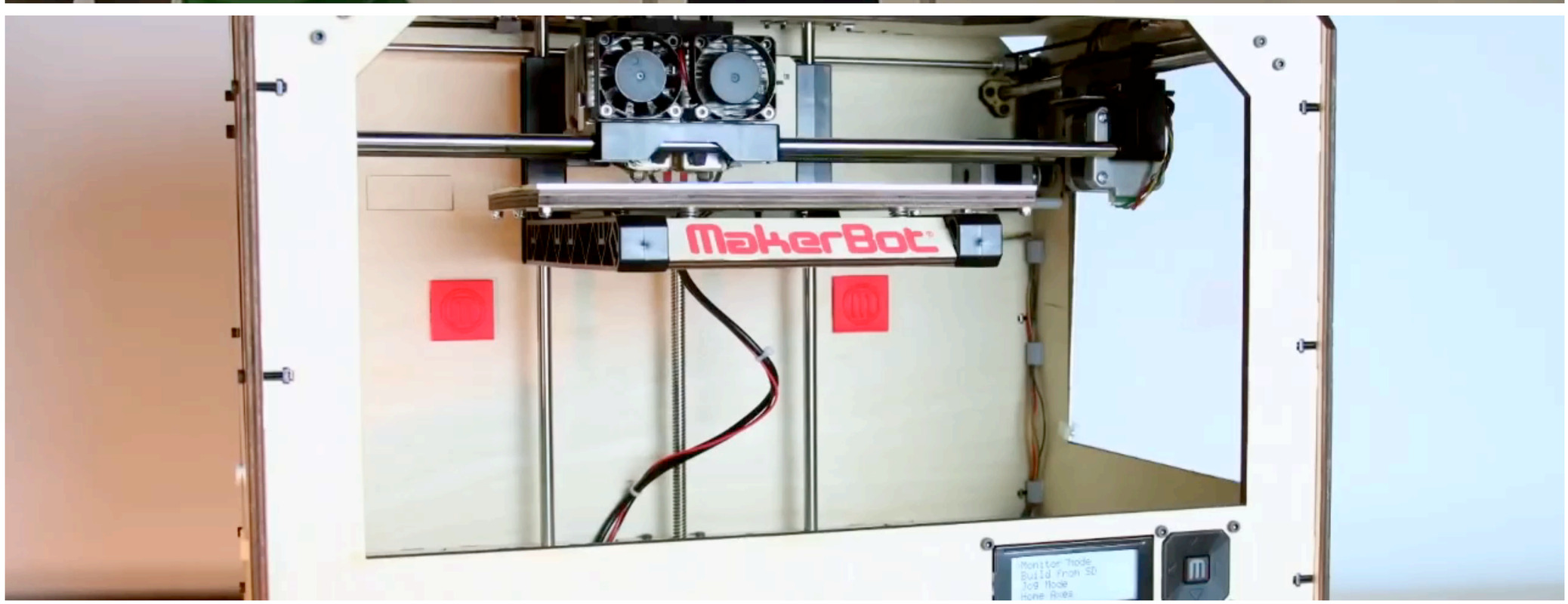


Digital Geometry Processing: Motivation

3D Scanning



3D Printing



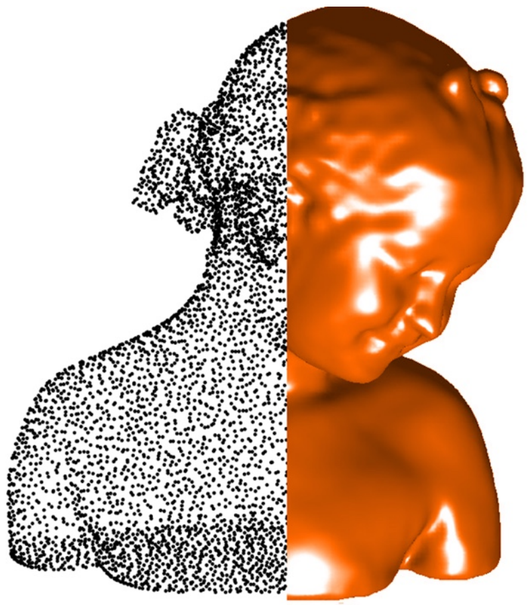
Geometry Processing Pipeline

scan

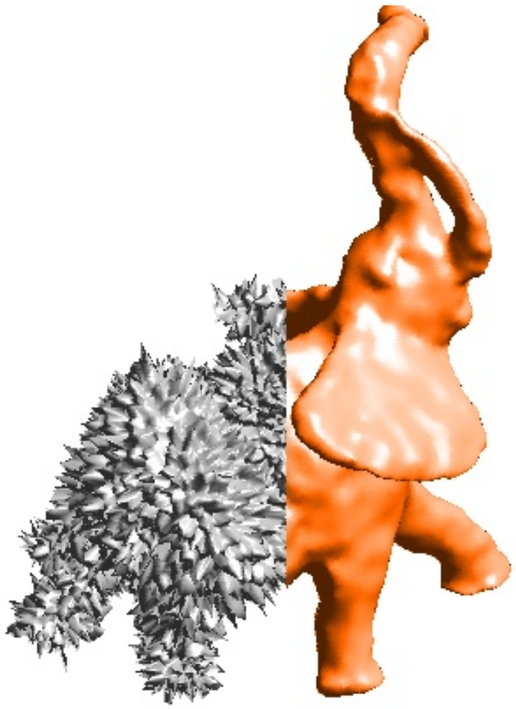
process

print

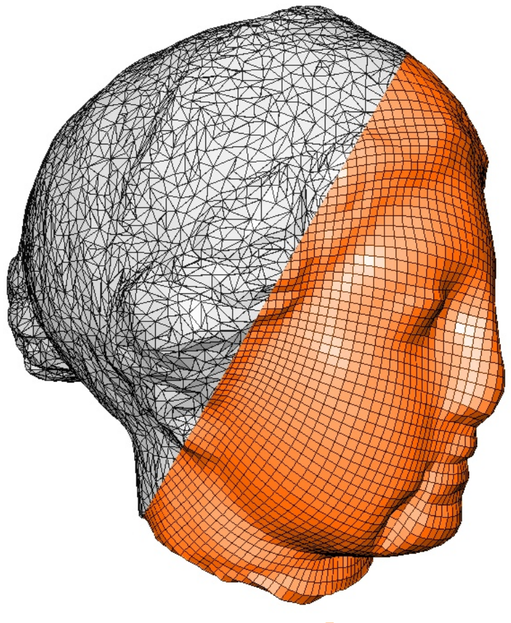
Geometry Processing Tasks



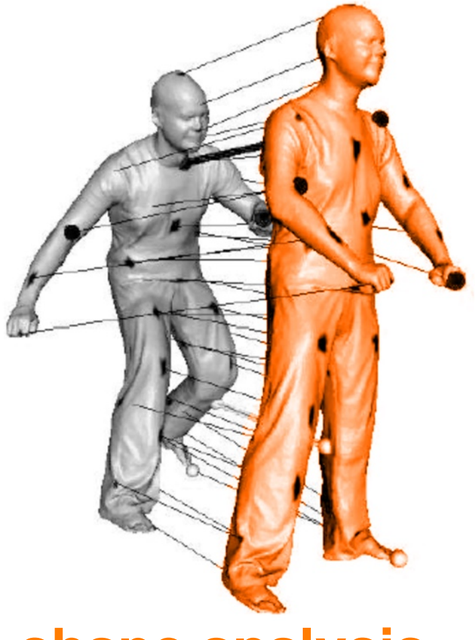
reconstruction



filtering

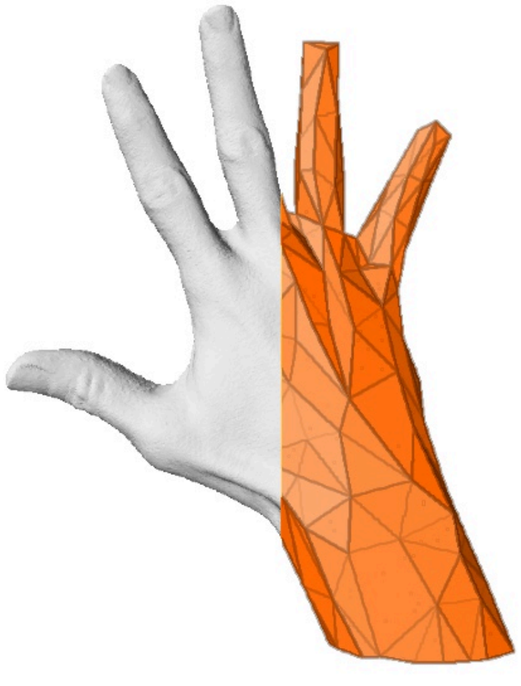


remeshing



shape analysis

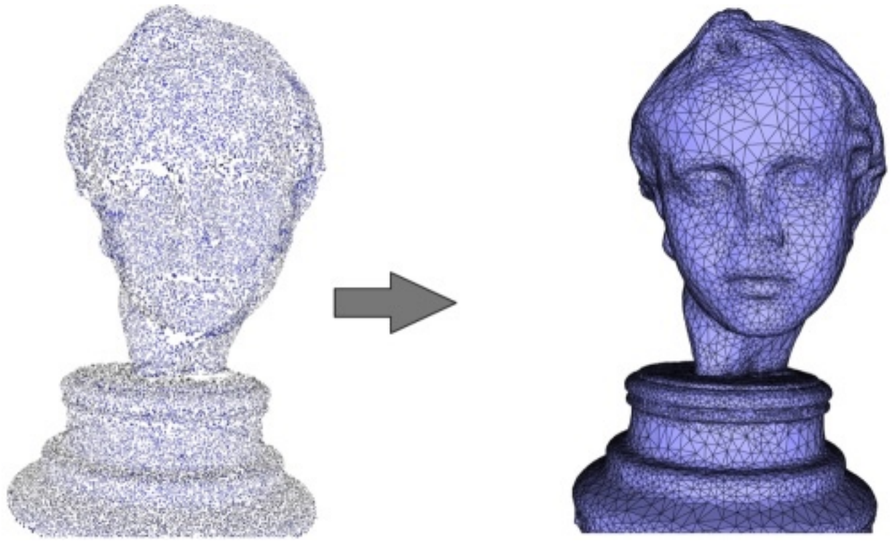
parameterization



compression

Geometry Processing: Reconstruction

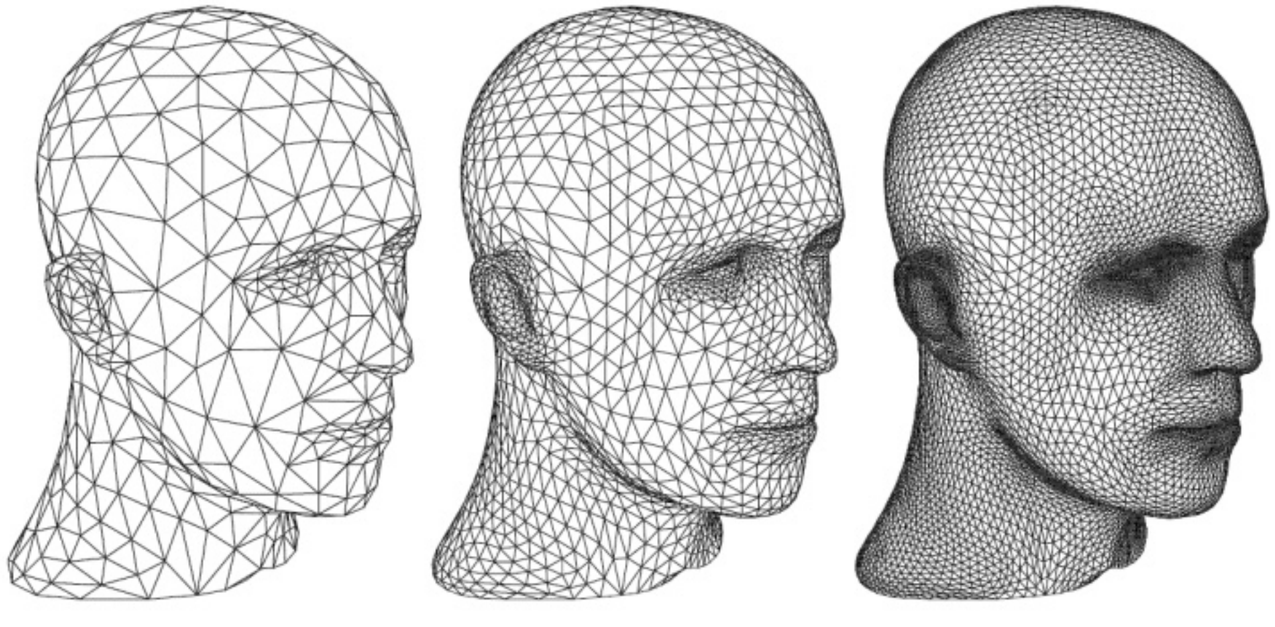
- Given samples of geometry, reconstruct continuous surface
- What are "samples"? Many possibilities:
  - points, points & normals, ...
  - image pairs / sets (multi-view stereo)
  - line density integrals (MRI/CT scans)



- How do you get a surface? Many techniques:
  - PDE-based (e.g., Poisson reconstruction)
  - Voronoi-based (e.g., power crust)
  - silhouette-based (visual hull)
  - Radon transform / isosurfacing (marching cubes)

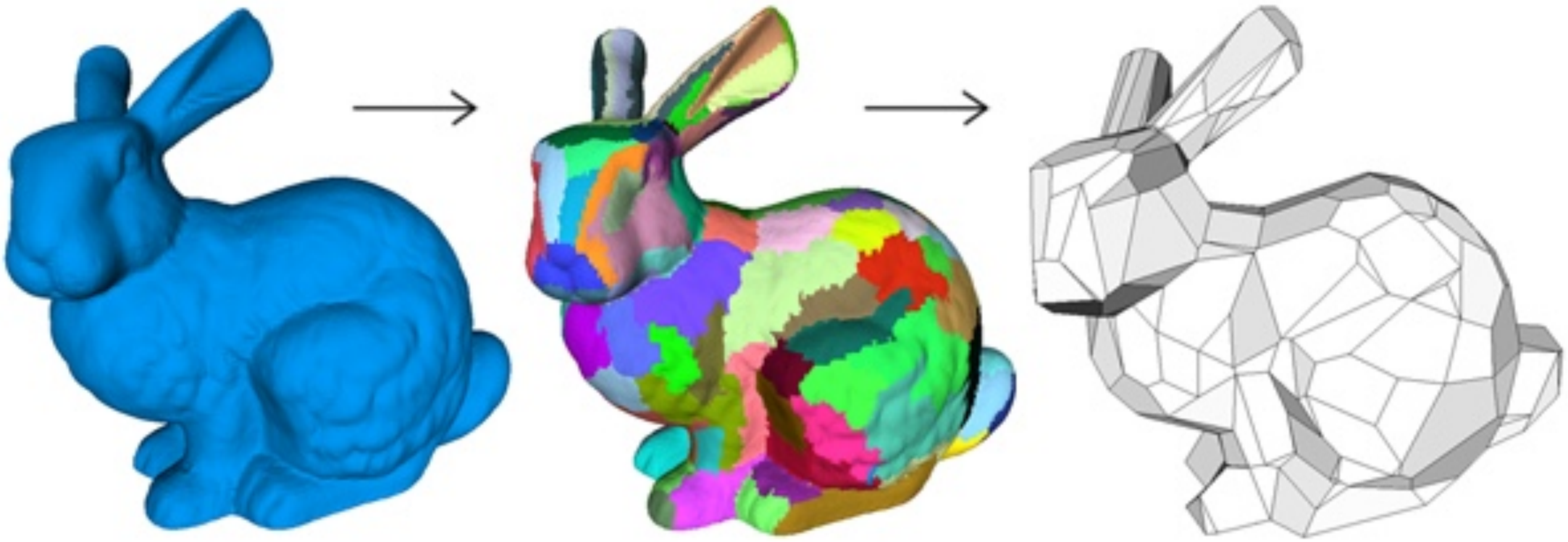
Geometry Processing: Upsampling

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



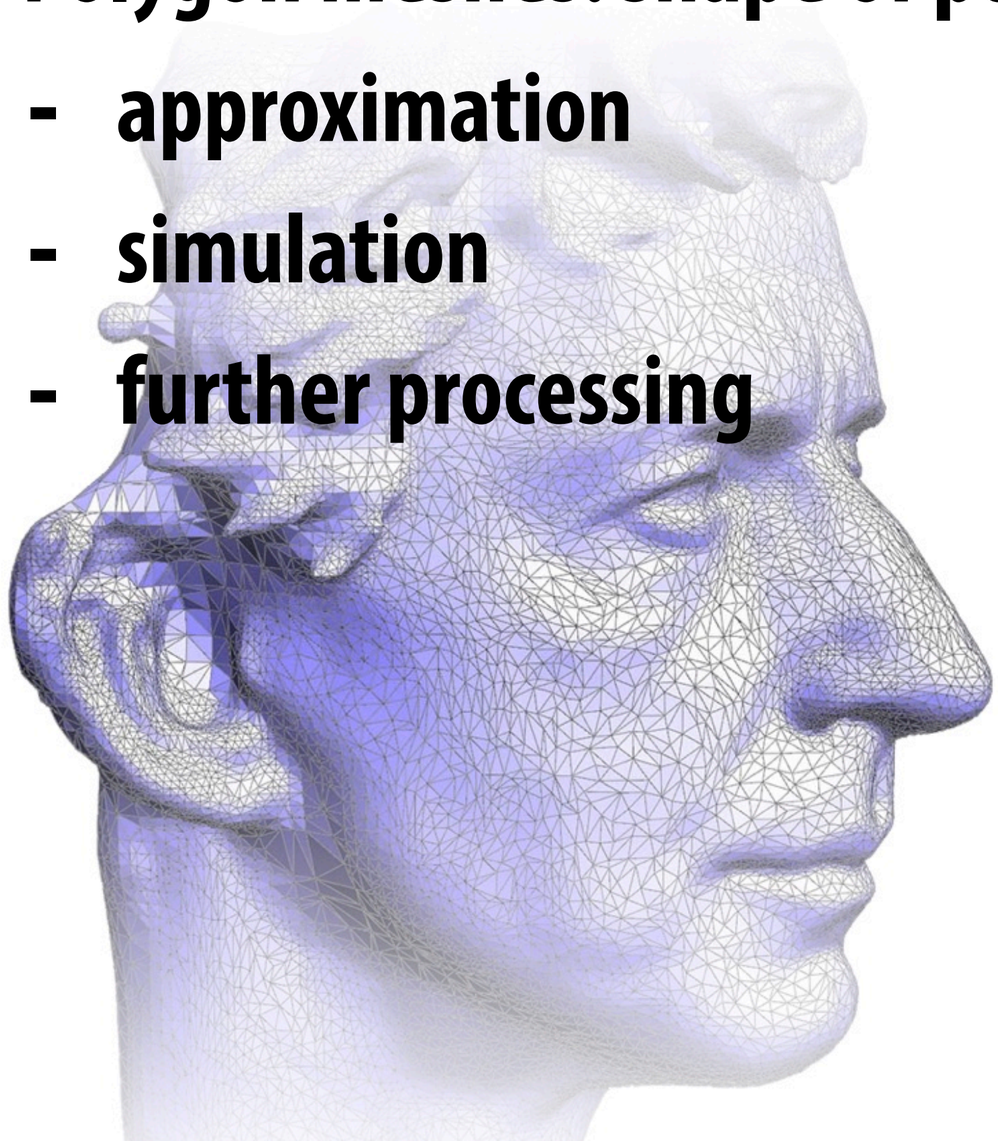
Geometry Processing: Downsampling

- Decrease resolution; try to preserve shape/appearance
- Images: again, bilinear, bicubic interpolation (again)
- Polygon meshes:
  - iterative decimation
  - variational shape approximation
  - ...



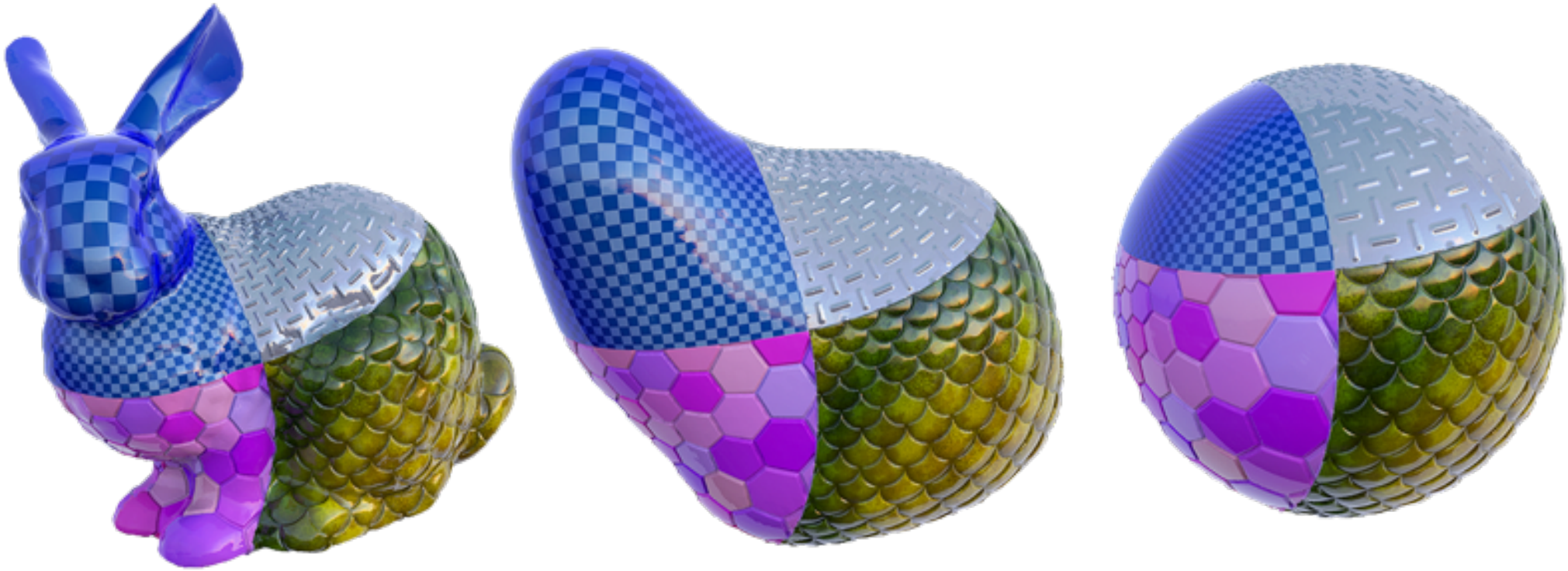
Geometry Processing: Resampling

- Modify sample distribution to improve quality
- Images: ...not usually an issue!
  - pixels are always stored on a regular grid
- Polygon meshes: shape of polygons extremely important!
  - approximation
  - simulation
  - further processing



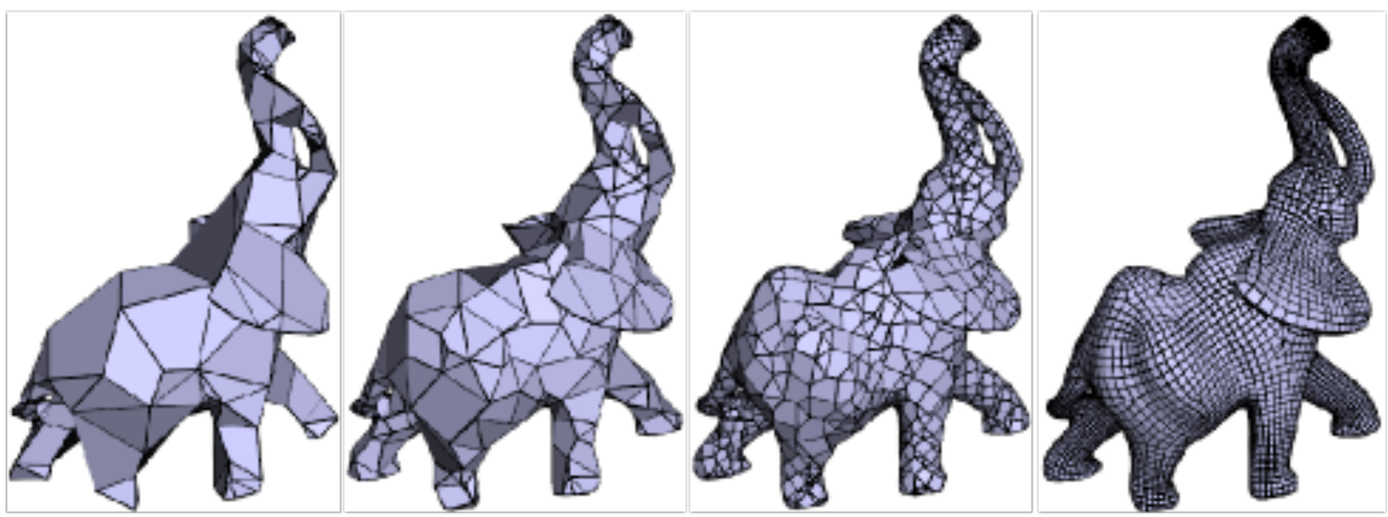
Geometry Processing: Filtering

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



Geometry Processing: Compression

- Reduce storage size by eliminating redundant data/ approximating unimportant data
- Images:
  - run-length encoding (RLE) - no loss of information
  - spectral/wavelet encoding (e.g., JPEG/MPEG) - lossy
- Polygon meshes:
  - have to compress geometry *and* connectivity
  - many techniques (spectral, diffusion, ...)



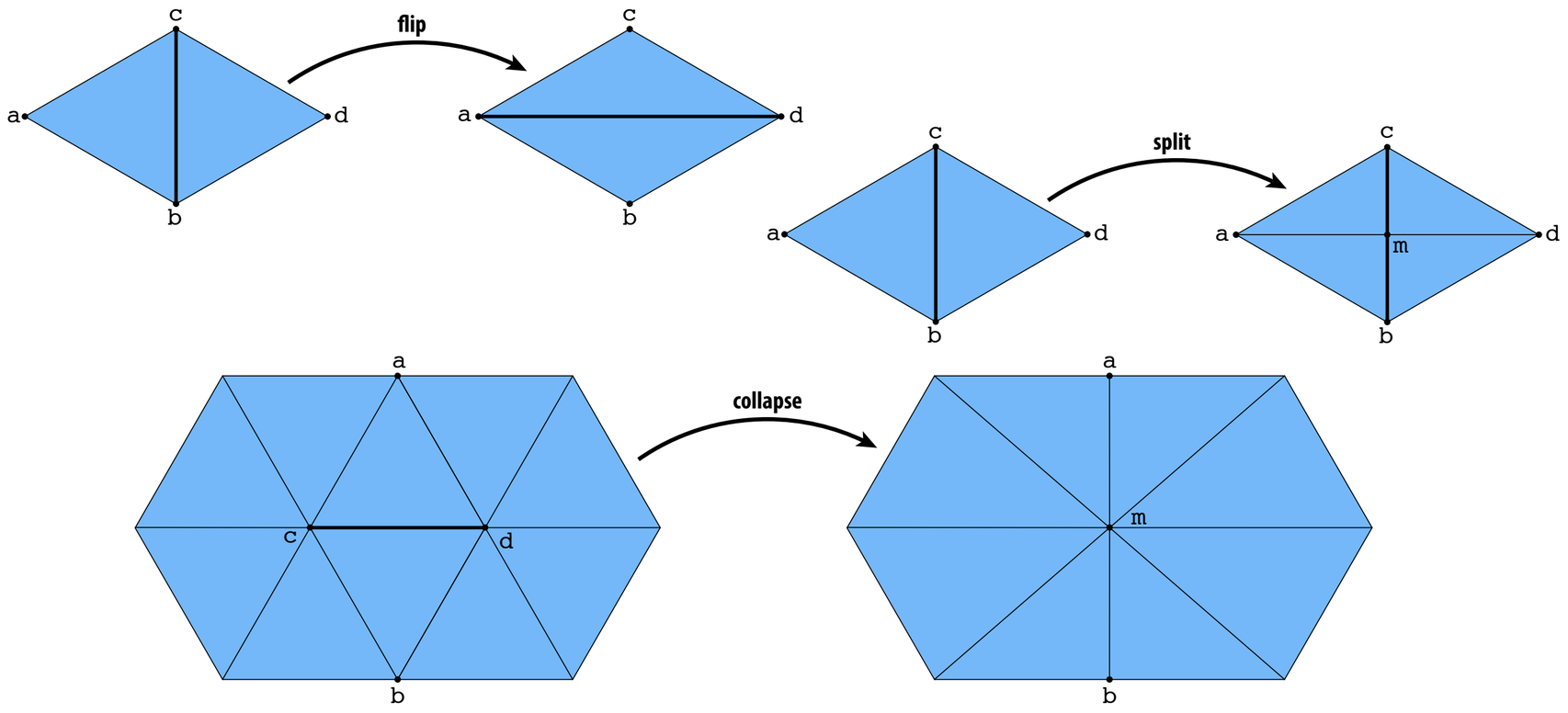
Geometry Processing: Shape Analysis

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

Ok, enough! Let's process some geometry!

Processing geometry with halfedges

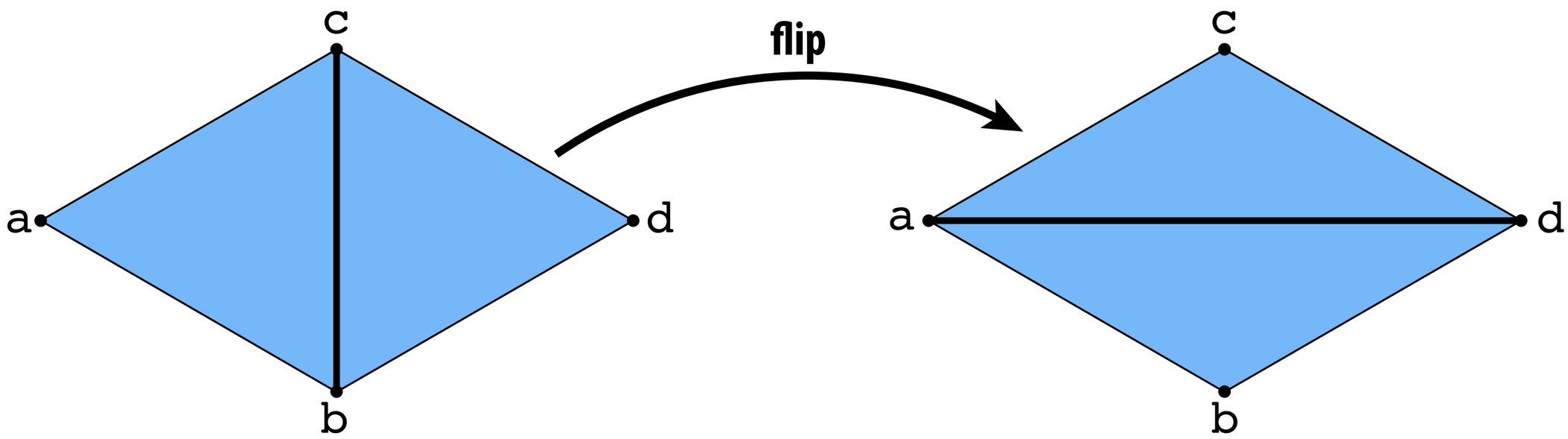
- Remember key feature of linked list: insert/delete elements
- Same story with halfedge mesh ("linked list on steroids")
- Several atomic operations for triangle meshes:



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

Edge Flip

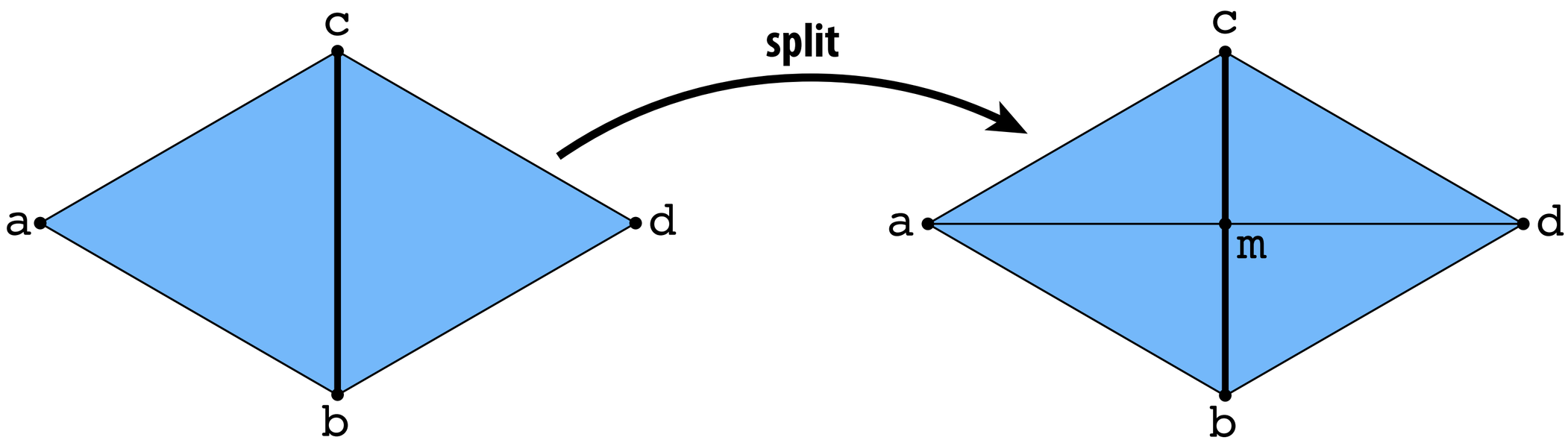
- Triangles (a,b,c), (b,d,c) become (a,d,c), (a,b,d):



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

Edge Split

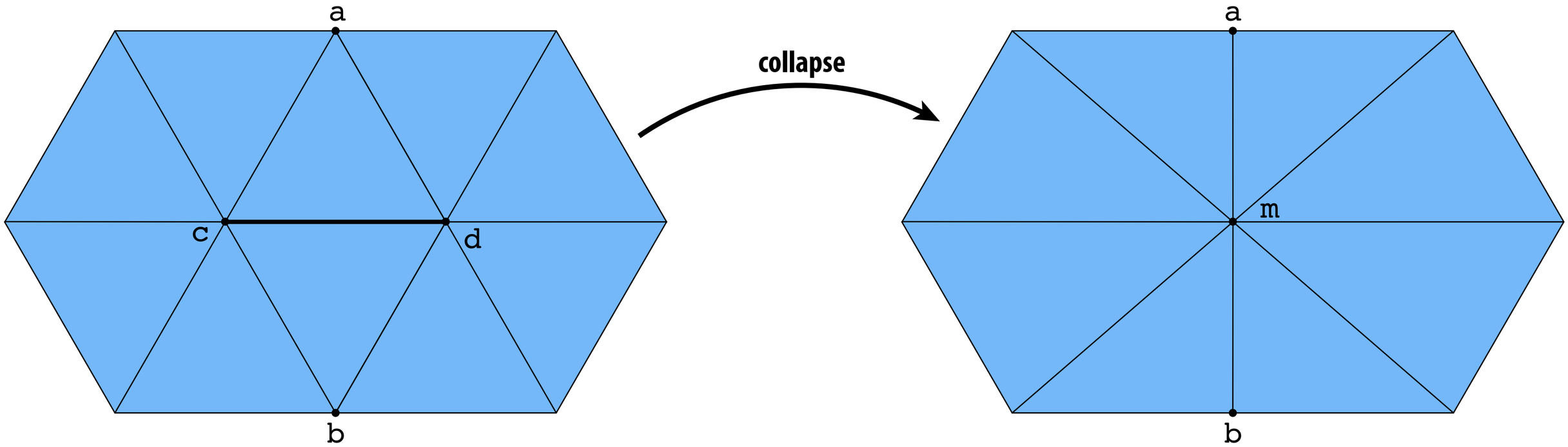
- Insert midpoint m of edge (c,b), connect to get four triangles:



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

Edge Collapse

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

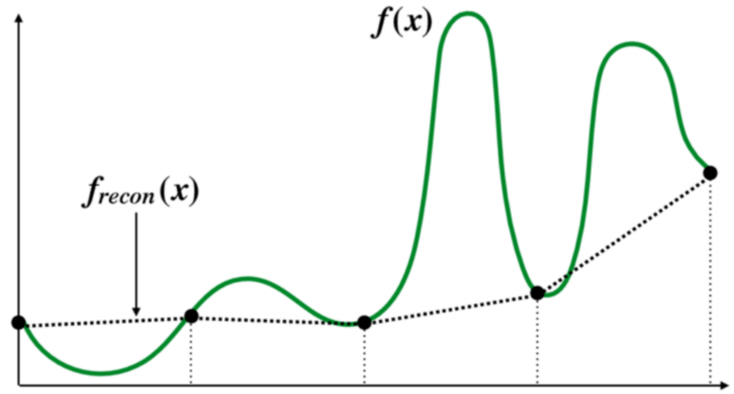


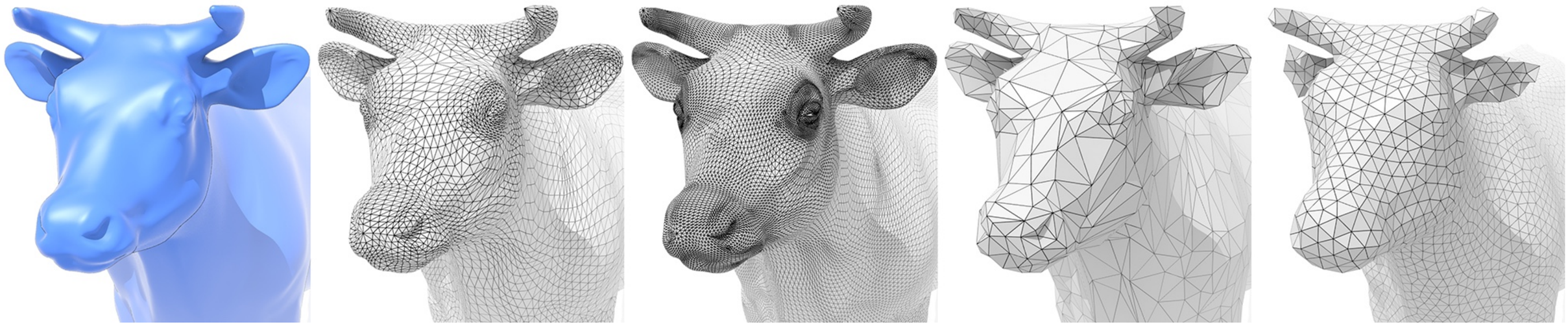
- Now have to *delete* elements.
- Still lots of pointer assignments!
- Q: How would we implement this with a polygon soup?
- Any other good way to do it? (E.g., different data structure?)

Ok, but what can we actually *do* with these operations?

Remeshing as resampling

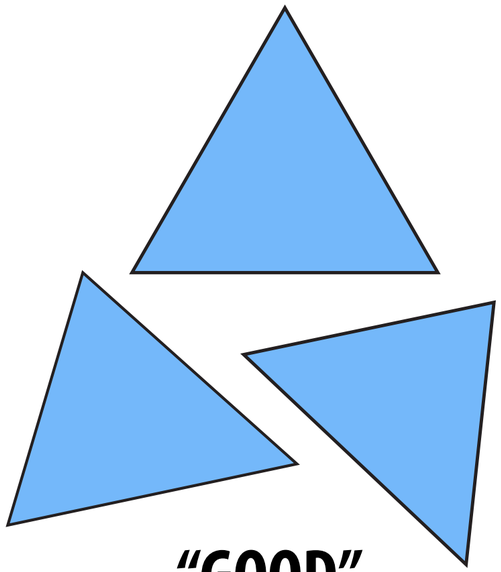
- Remember our discussion of *aliasing*
- Bad sampling makes signal appear different than it really is
- E.g., undersampled curve looks flat
- Geometry is no different!
  - undersampling destroys features
  - oversampling destroys performance



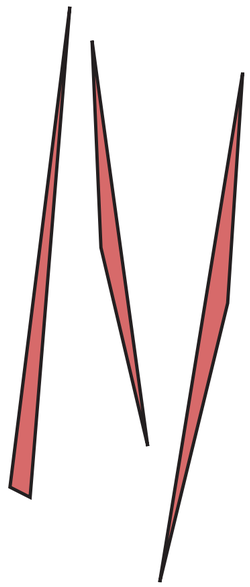


What makes a "good" triangle mesh?

- One rule of thumb: *triangle shape*

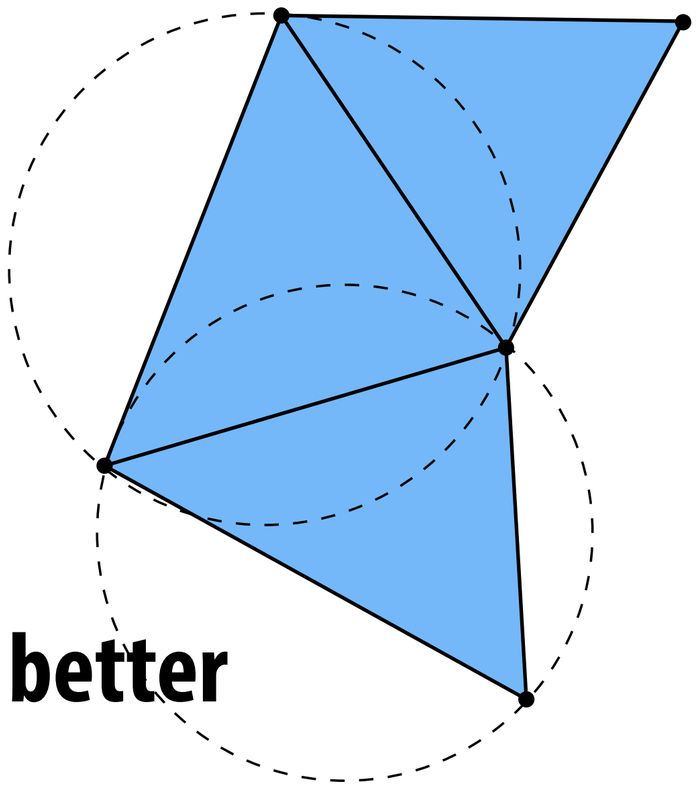


"GOOD"



"BAD"

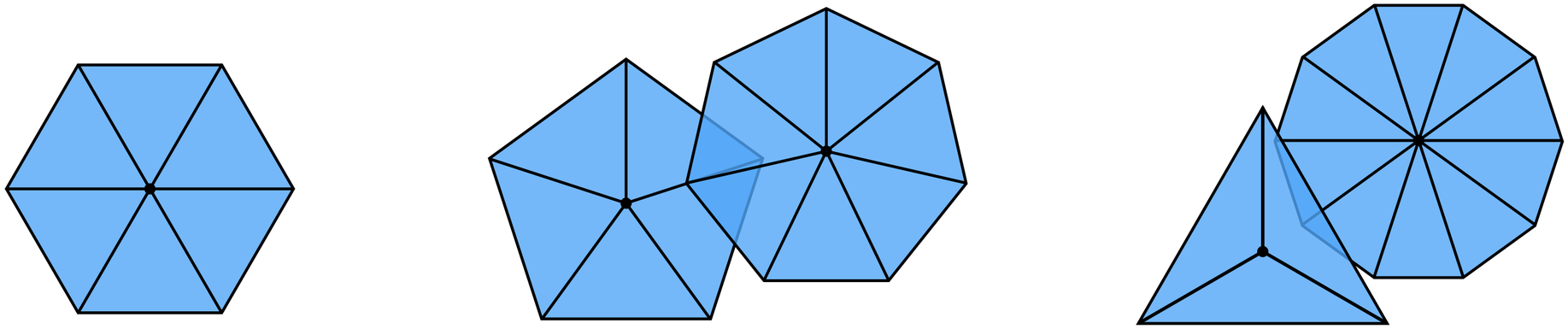
- More specific condition: *Delaunay*
- "Circumcircle interiors contain no vertices."
- Not *always* a good condition, but often*
  - especially important for simulation
  - for approximation, long triangles may be better



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

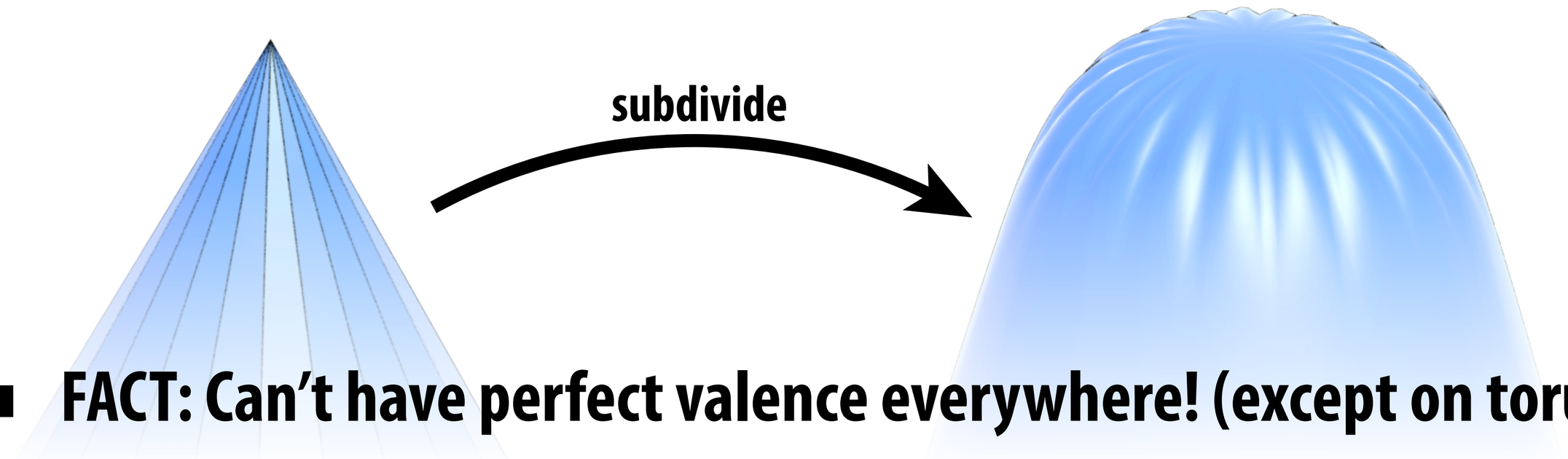
What else constitutes a good mesh?

- Another rule of thumb: *regular vertex degree*
- Ideal for triangle meshes: make every vertex valence 6:



"GOOD" "OK" "BAD"

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

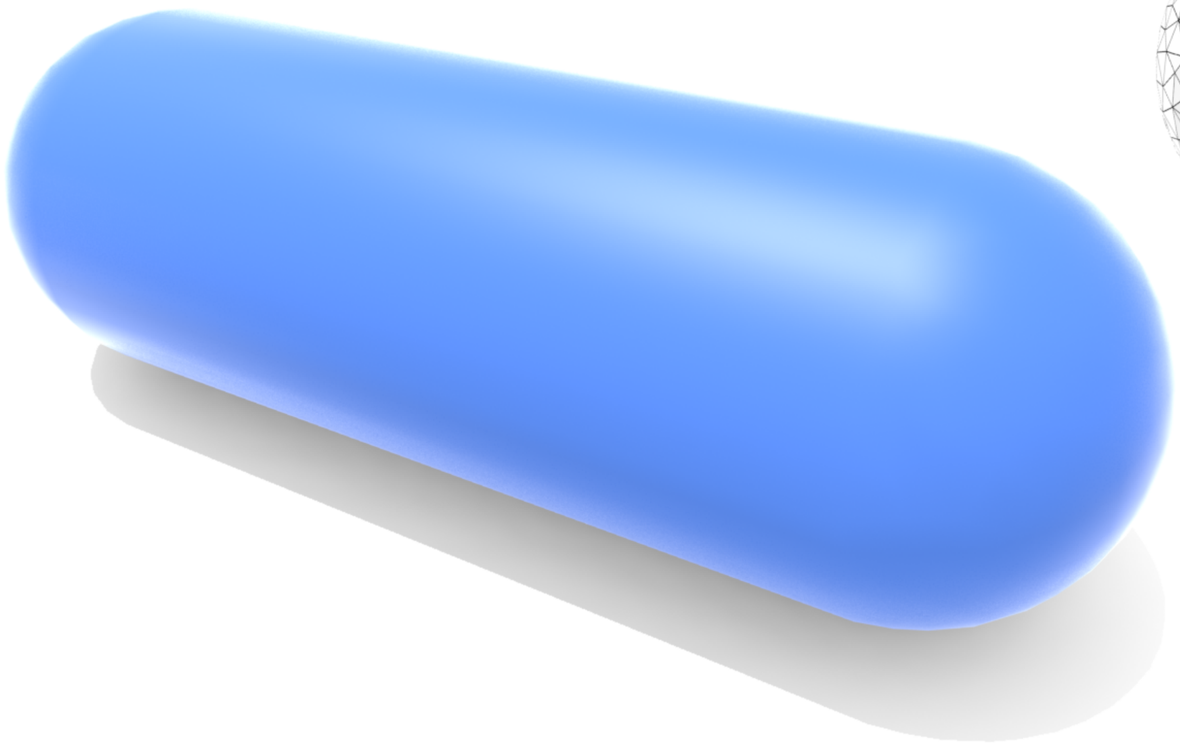


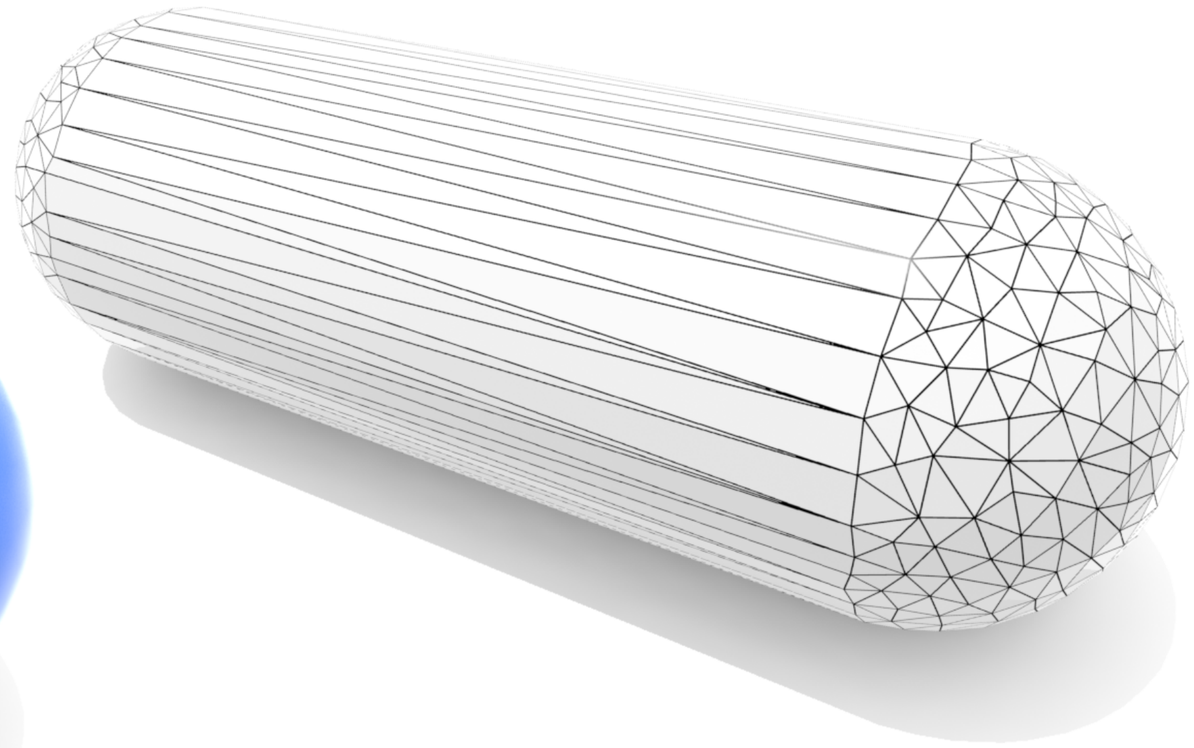
- FACT: Can't have perfect valence everywhere! (except on torus)

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

What *else* makes a "good" mesh?

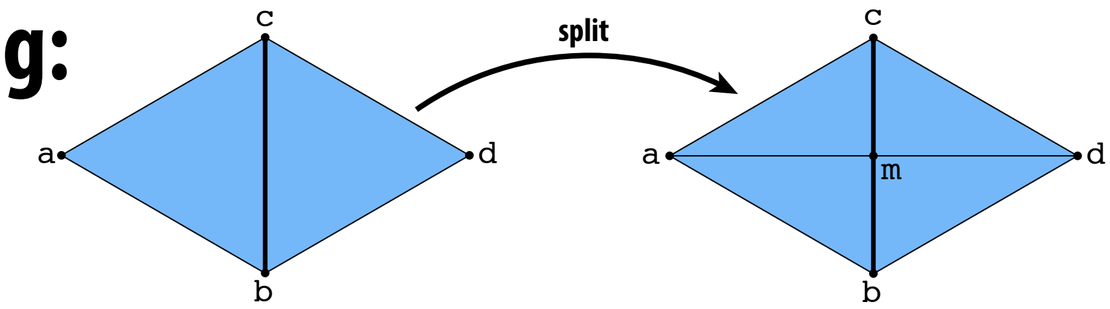
- Good approximation of original shape!
- Keep only elements that contribute *information* about shape
- Add additional information where, e.g., *curvature* is large
- Balance with element quality
  - Delaunay, "round" triangles, regular degree, etc.



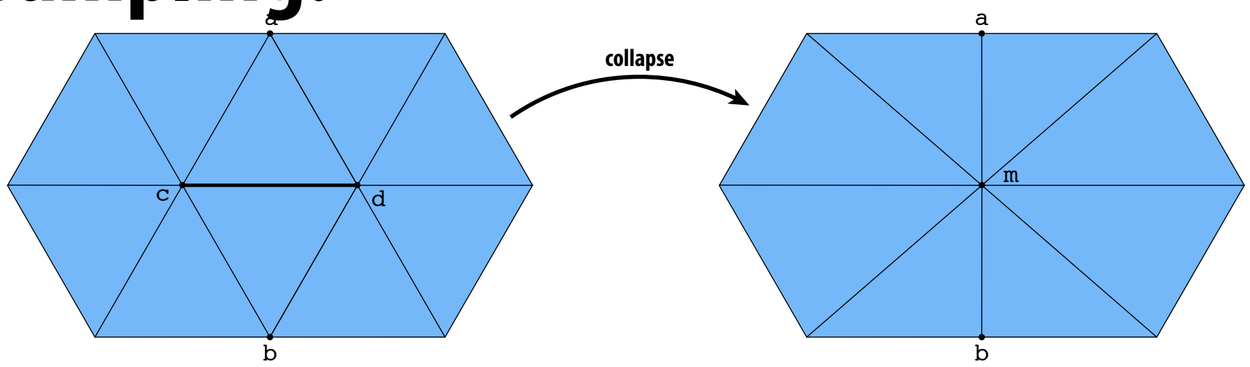


How do we resample? Already know how!

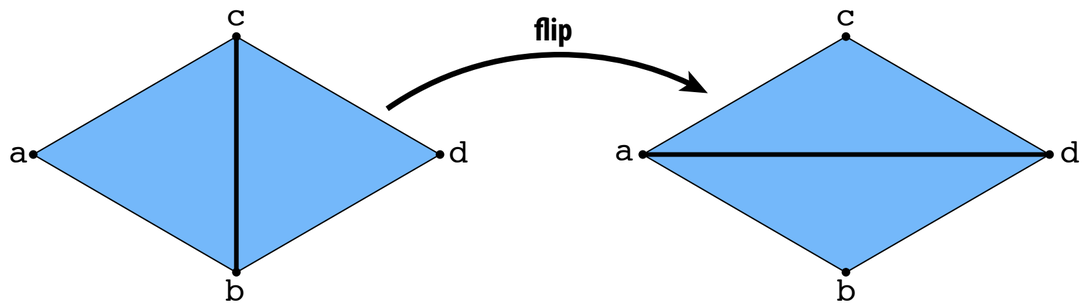
- **Edge split is (local) upsampling:**



- **Edge collapse is (local) downsampling:**



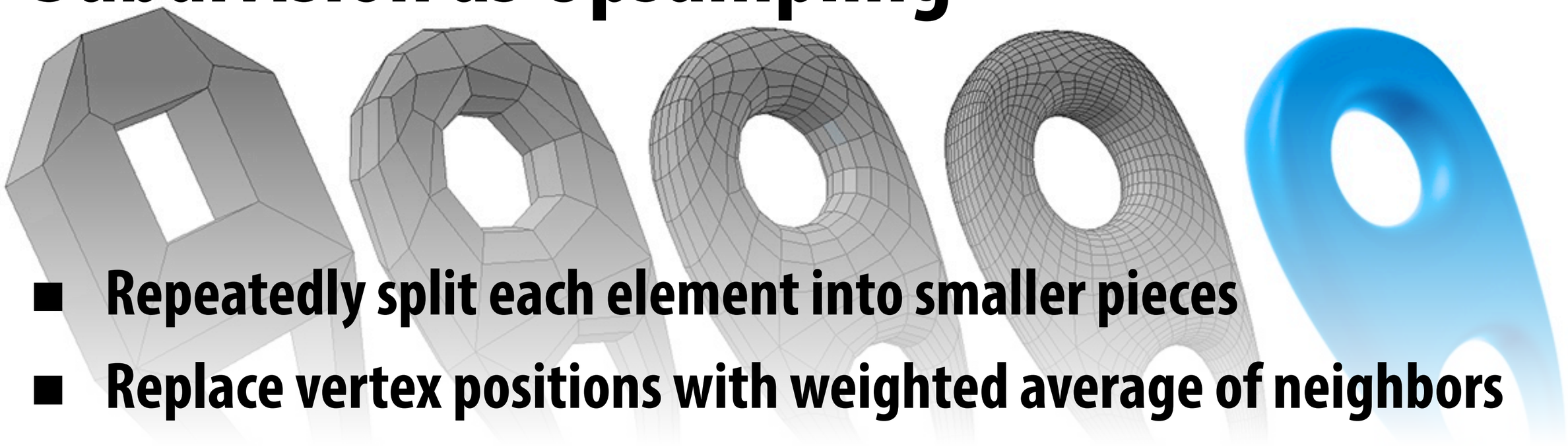
- **Edge flip is (local) resampling:**



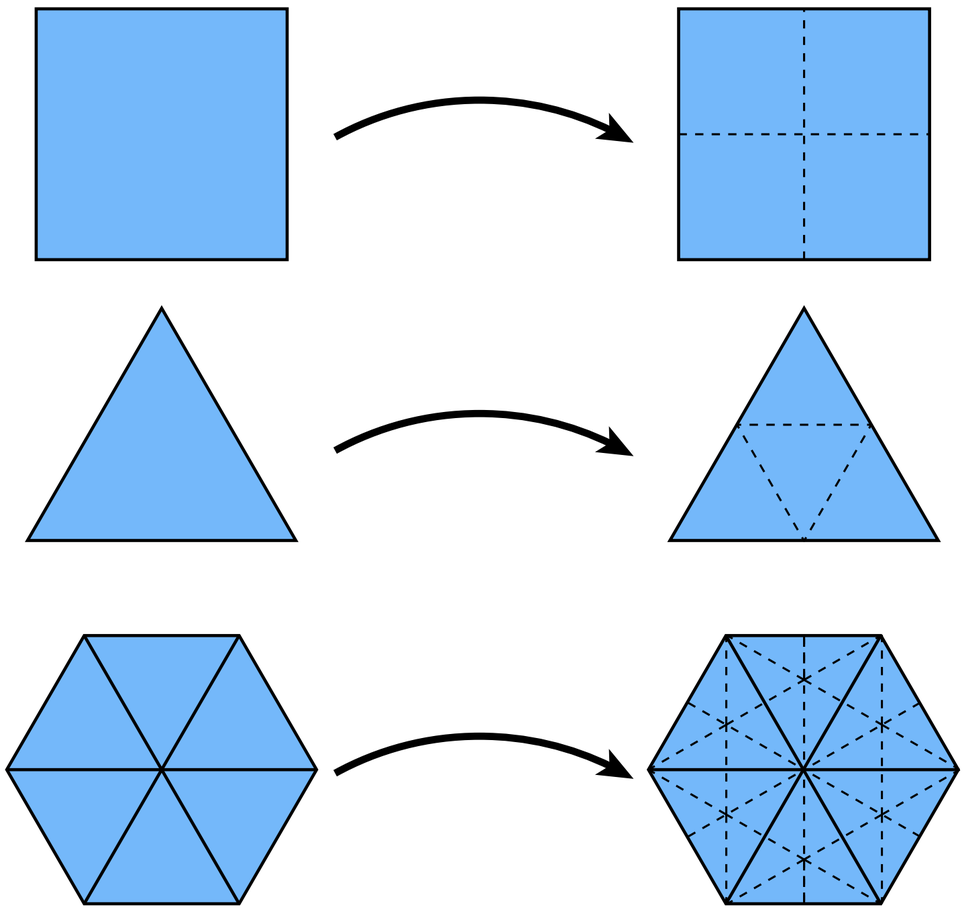
- **Still need to intelligently decide *which* edges to modify!**

Which edges should we split to upsample the whole mesh?

Subdivision as Upsampling

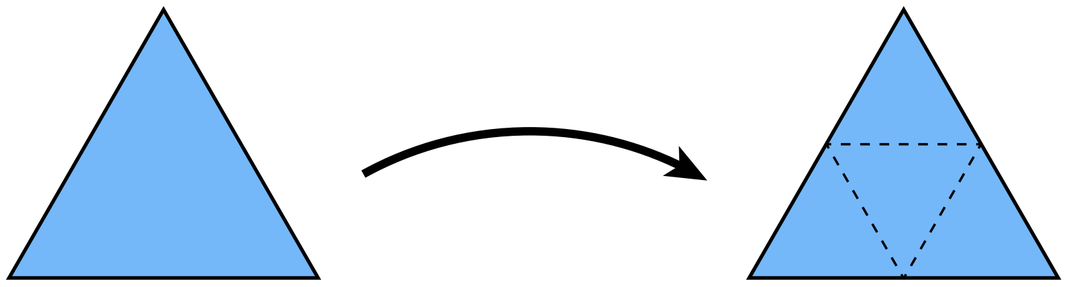


- Repeatedly split each element into smaller pieces
- Replace vertex positions with weighted average of neighbors
- Main considerations:
  - interpolating vs. approximating
  - limit surface continuity (C^1, C^2, ...)
  - behavior at irregular vertices
- Many options:
  - Quad: Catmull-Clark
  - Triangle: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

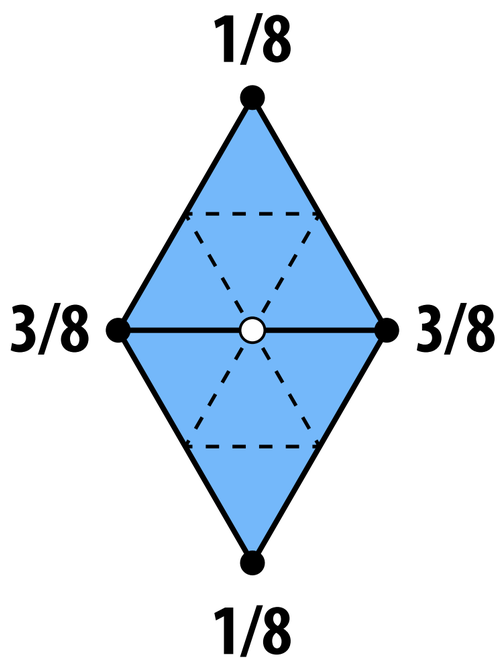


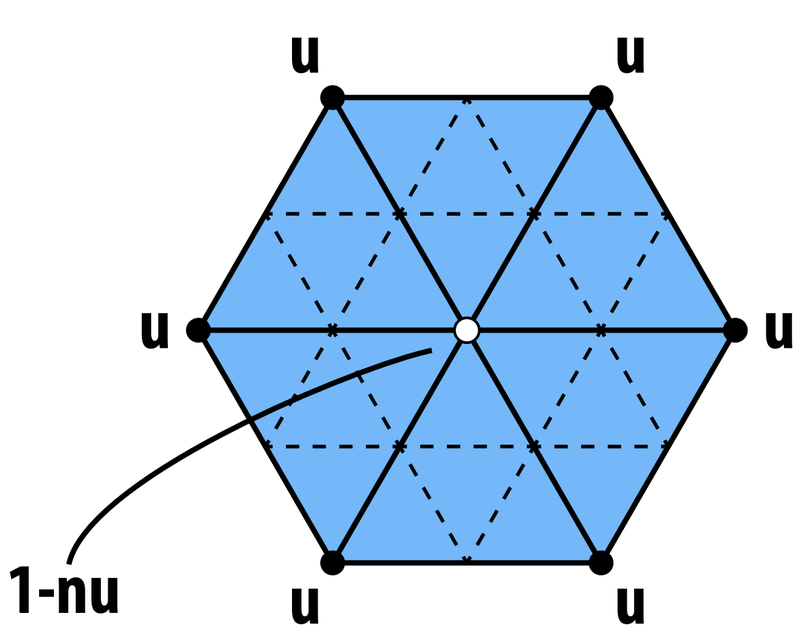
Loop Subdivision

- Fairly common subdivision rule for triangles
- Curvature is continuous away from irregular vertices ("C^2")
- *Approximating*, not interpolating
- Algorithm:
  - Split each triangle into four



  - Assign new vertex positions according to weights:



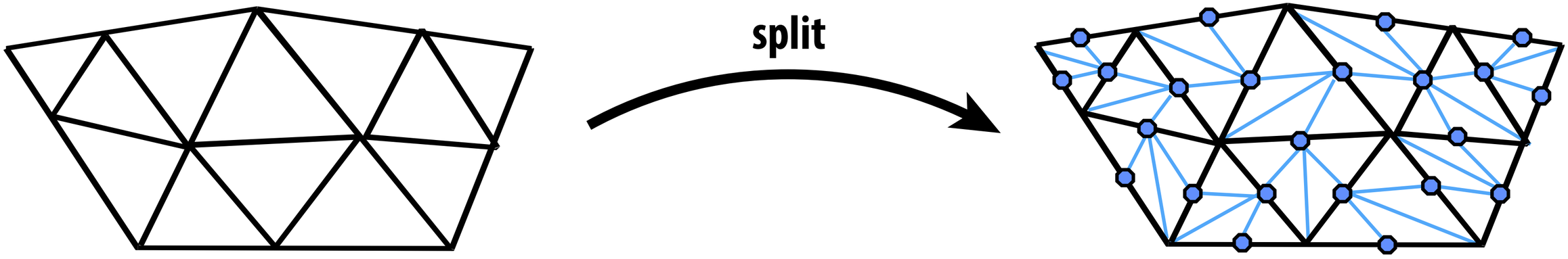


n: vertex degree

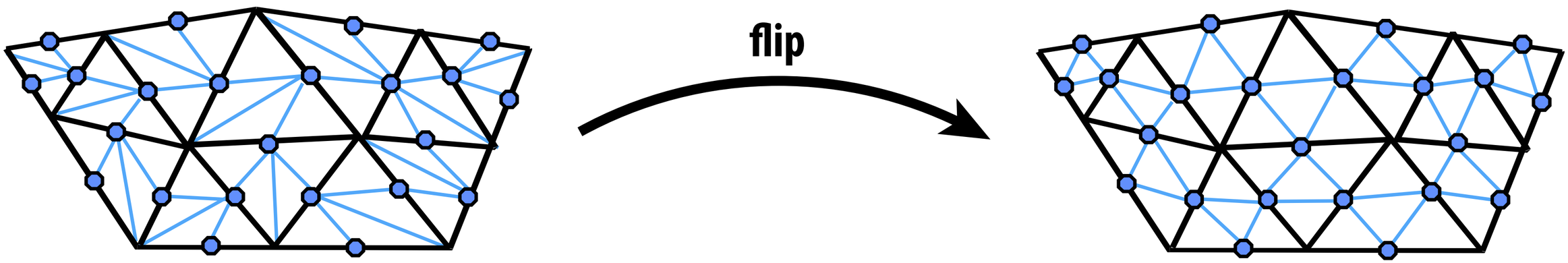
u: 3/16 if n=3, 3/(8n) otherwise

Loop Subdivision via Edge Operations

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



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



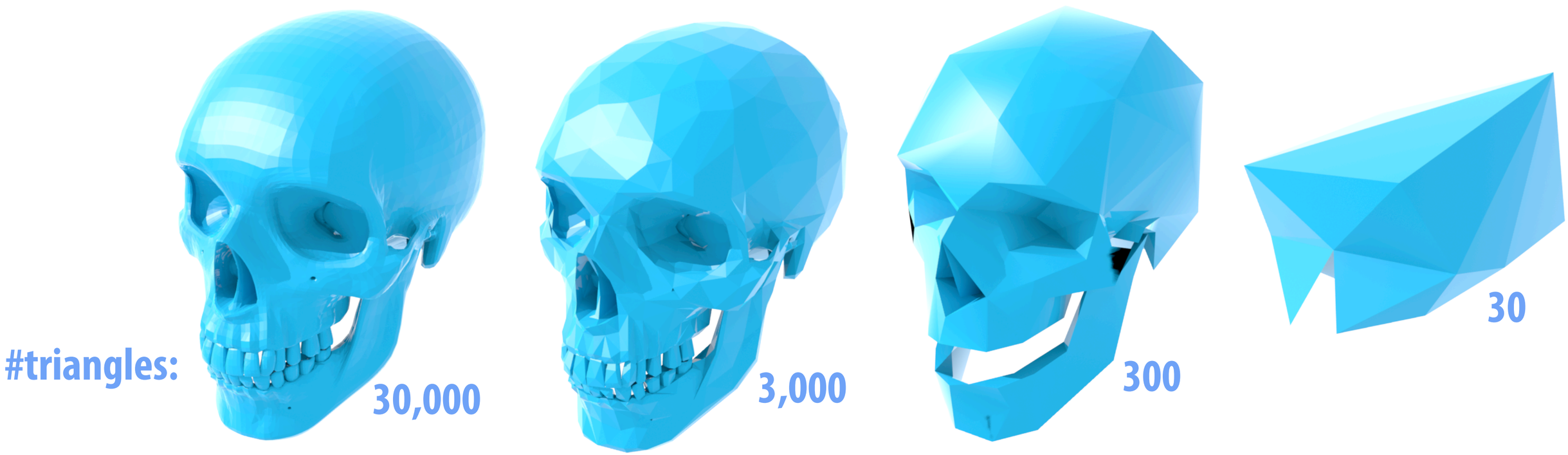
(Don’t forget to update vertex positions!)

Images cribbed from Denis Zorin.

What if we want *fewer* triangles?

Simplification via Quadric Error Metric

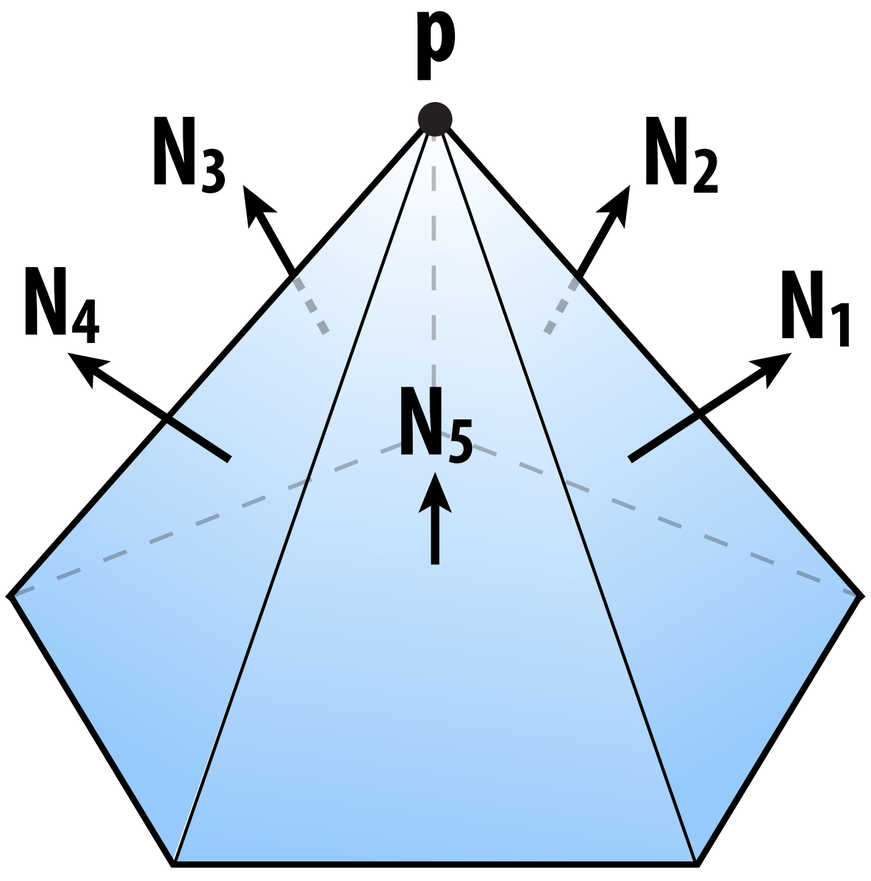
- One popular scheme: iteratively collapse edges
- Which edges? Assign score with *quadric error metric**
  - approximate distance to surface as sum of distance to aggregated triangles
  - iteratively collapse edge with smallest score
  - greedy algorithm… great results!

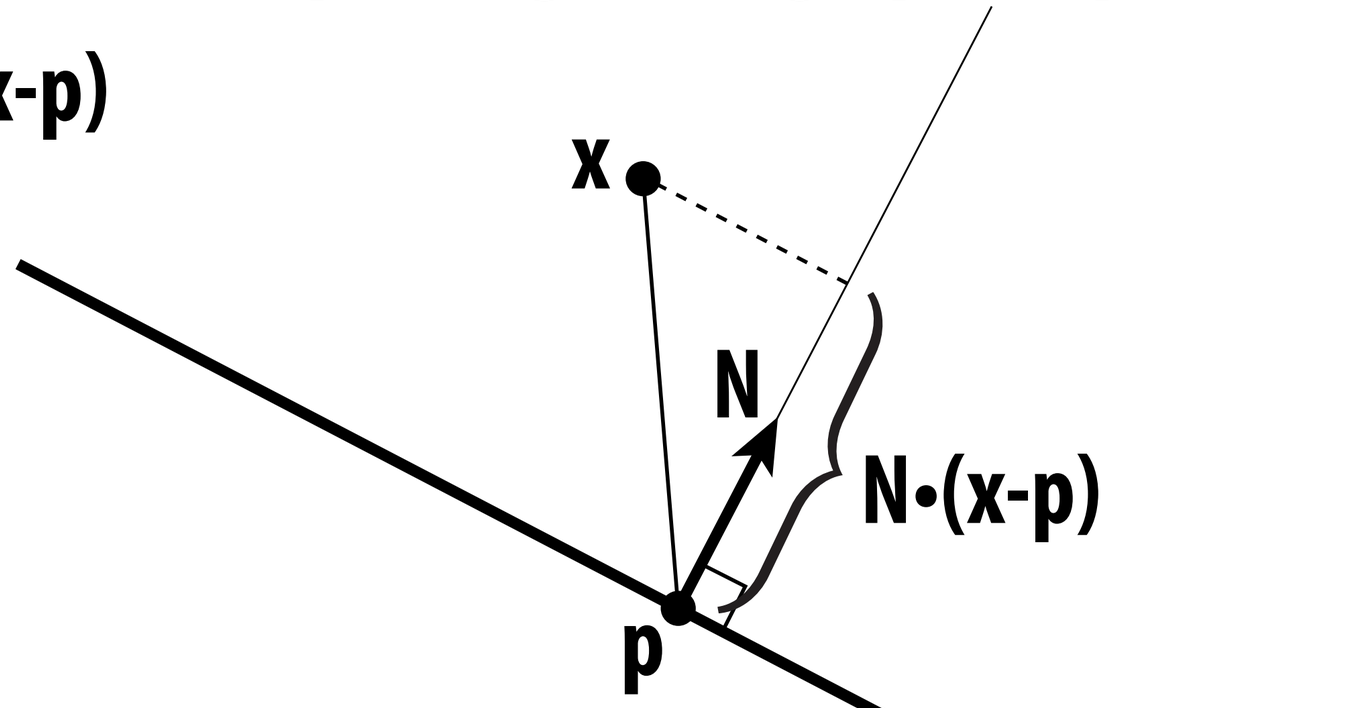


*invented here at CMU! (Garland & Heckbert 1997)

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: $d(x) = N \bullet x - N \bullet p = N \bullet (x-p)$
- Sum of distances:





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

Quadric Error - Homogeneous Coordinates

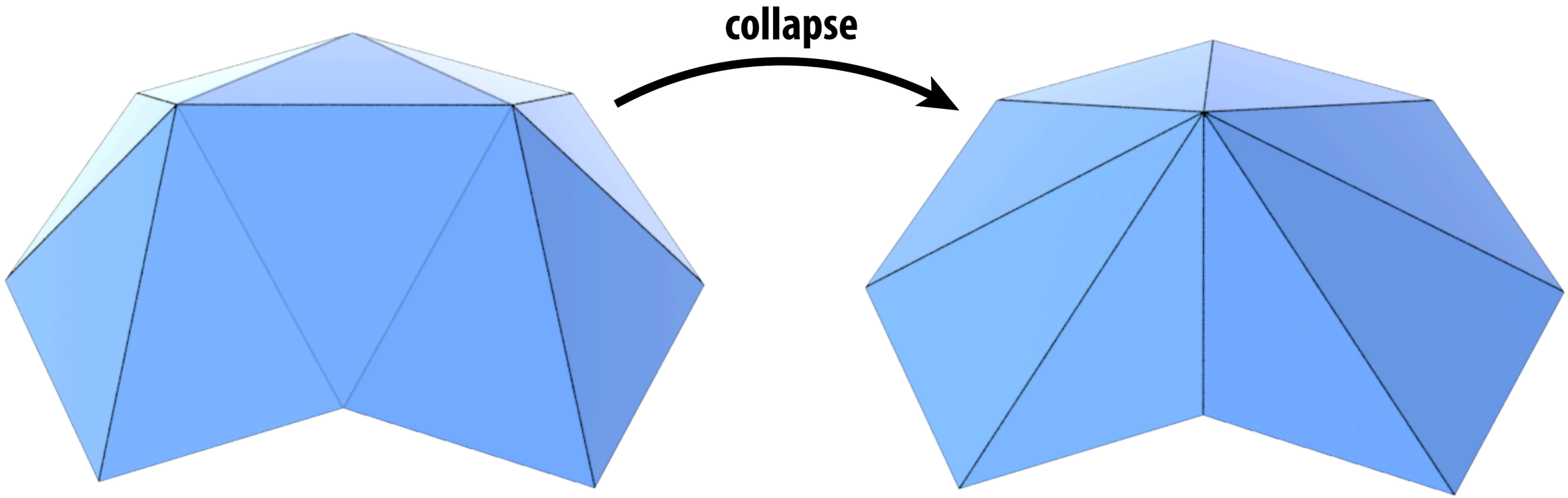
- Suppose in coordinates we have
  - a query point (x,y,z)
  - a normal (a,b,c)
  - an offset d := -(x,y,z) • (a,b,c)

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

- 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 $(u^T v)^2 = u^T(vv^T)u =: u^T Q u$
- Key idea: *matrix Q encodes distance to plane*
- Q is symmetric, contains 10 unique coefficients (small storage)

Quadric Error of Edge Collapse

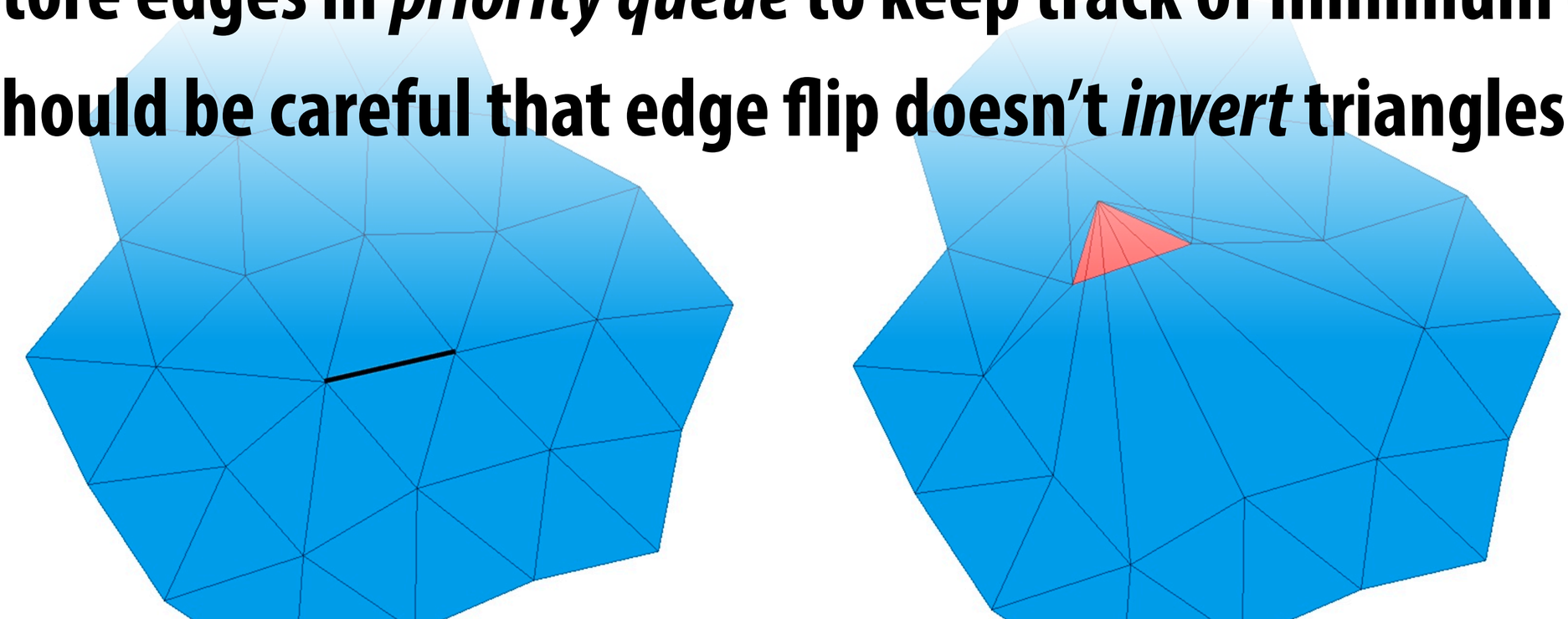
- How much does it cost to collapse an edge?
- Idea: compute edge midpoint, measure quadric error



- Better idea: use point that minimizes quadric error as new point!
- (More details in assignment; see also Garland & Heckbert 1997.)

Quadric Error Simplification

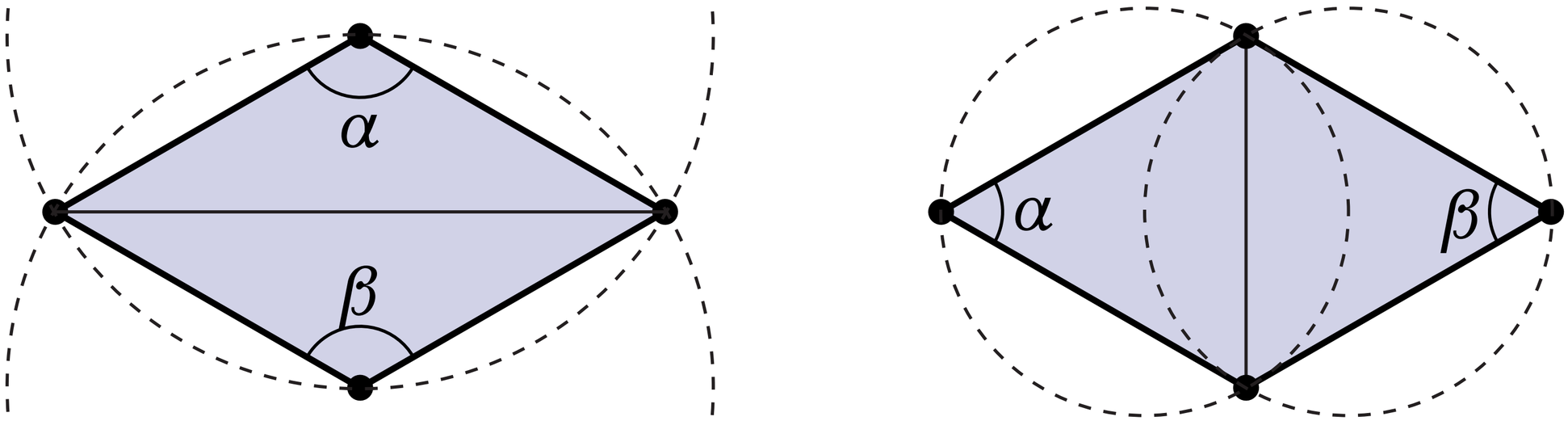
- Compute Q for each triangle
- Set Q at each vertex to sum of Qs from incident triangles
- Until we reach target # of triangles:
  - collapse edge (i,j) with smallest cost to get new vertex k
  - add Q_i and Q_j to get new quadric Q_k
  - update cost of any edge touching new vertex k
- Store edges in *priority queue* to keep track of minimum cost
- Should be careful that edge flip doesn't *invert* triangles:



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

How do we make a mesh “more Delaunay”?

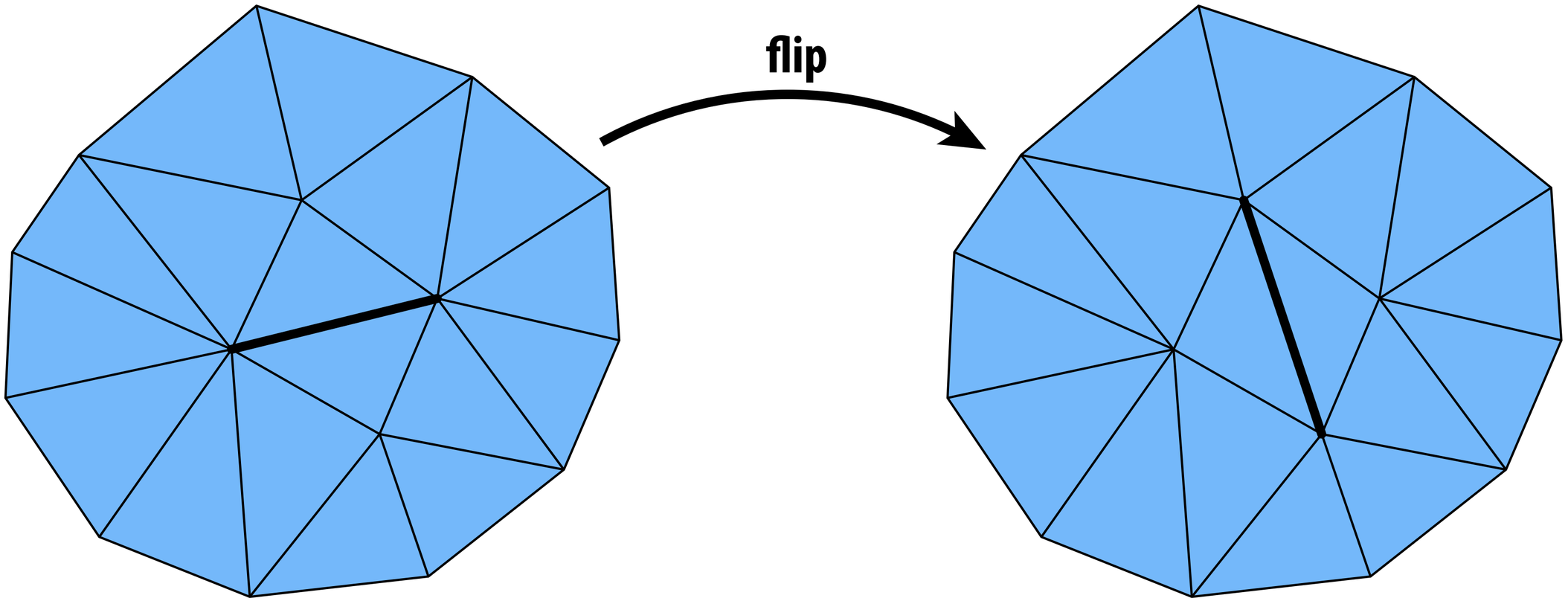
- Already have a good tool: edge flips!
- If $\alpha+\beta > \pi$, flip it!



- 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?

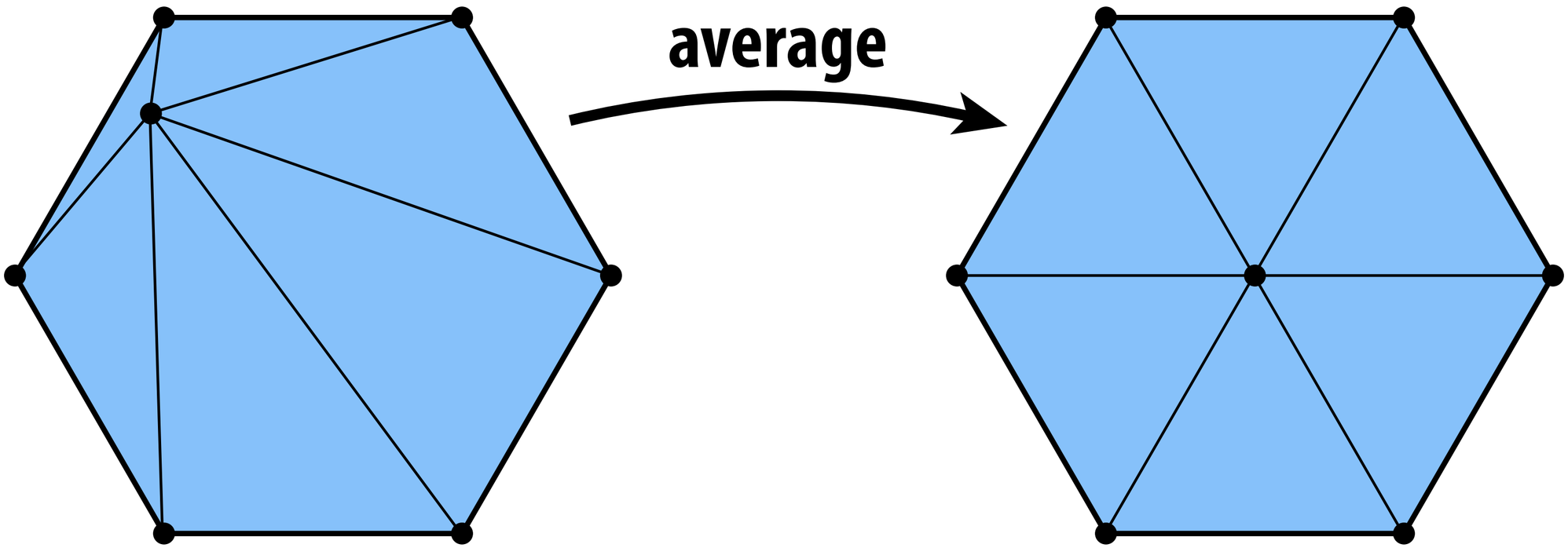
- Same tool: edge flips!
- If total deviation from degree-6 gets smaller, flip it!



- 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:

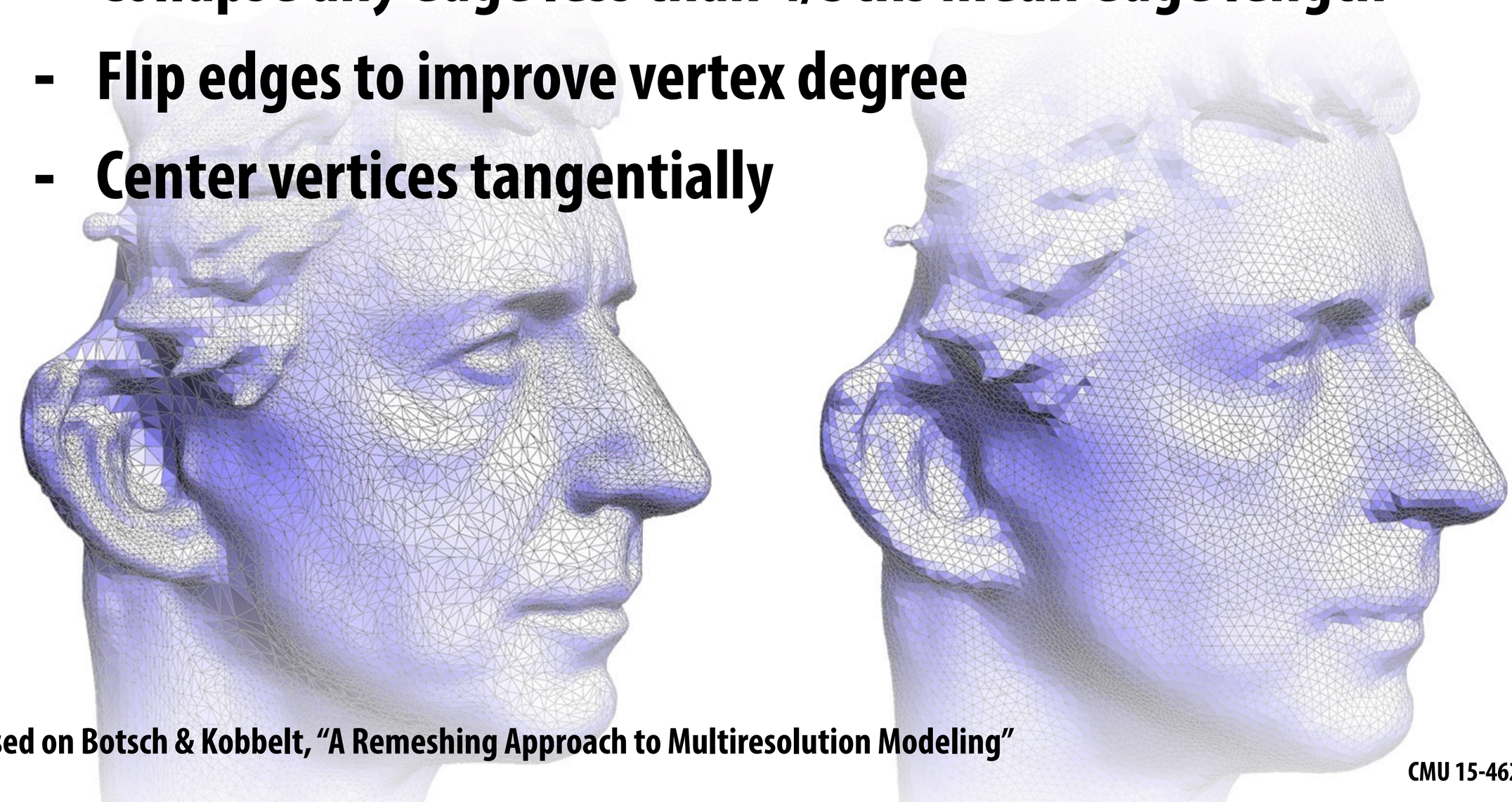


- Simple version of technique called “Laplacian smoothing”.*
- On surface: move only in *tangent* direction
- How? Remove normal component from update vector.

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

Isotropic Remeshing Algorithm*

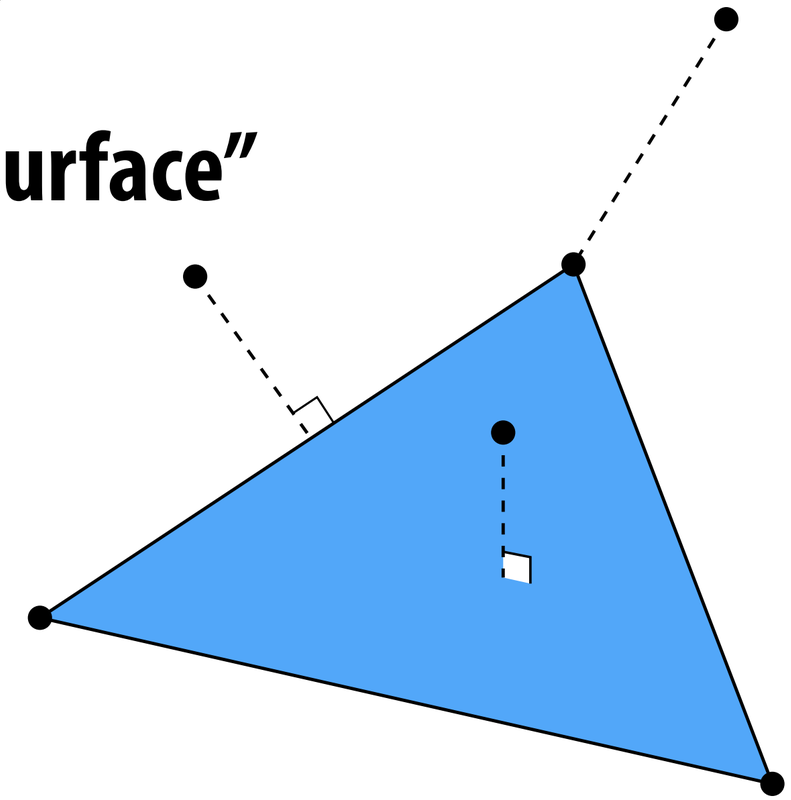
- Try to make triangles uniform shape & size
- Repeat four steps:
  - Split any edge over 4/3rds mean edge legth
  - Collapse any edge less than 4/5ths mean edge length
  - Flip edges to improve vertex degree
  - Center vertices tangentially



*Based on Botsch & Kobbelt, “A Remeshing Approach to Multiresolution Modeling”

Next time: Geometric Queries

- Not all queries can be answered "on the surface"
- E.g.,
  - distance from point to surface?
  - intersection between two surfaces?
  - where does a ray pierce the surface?



- Need totally different data structures/algorithms