# Computer Graphics
## -Subdivision & Simplification

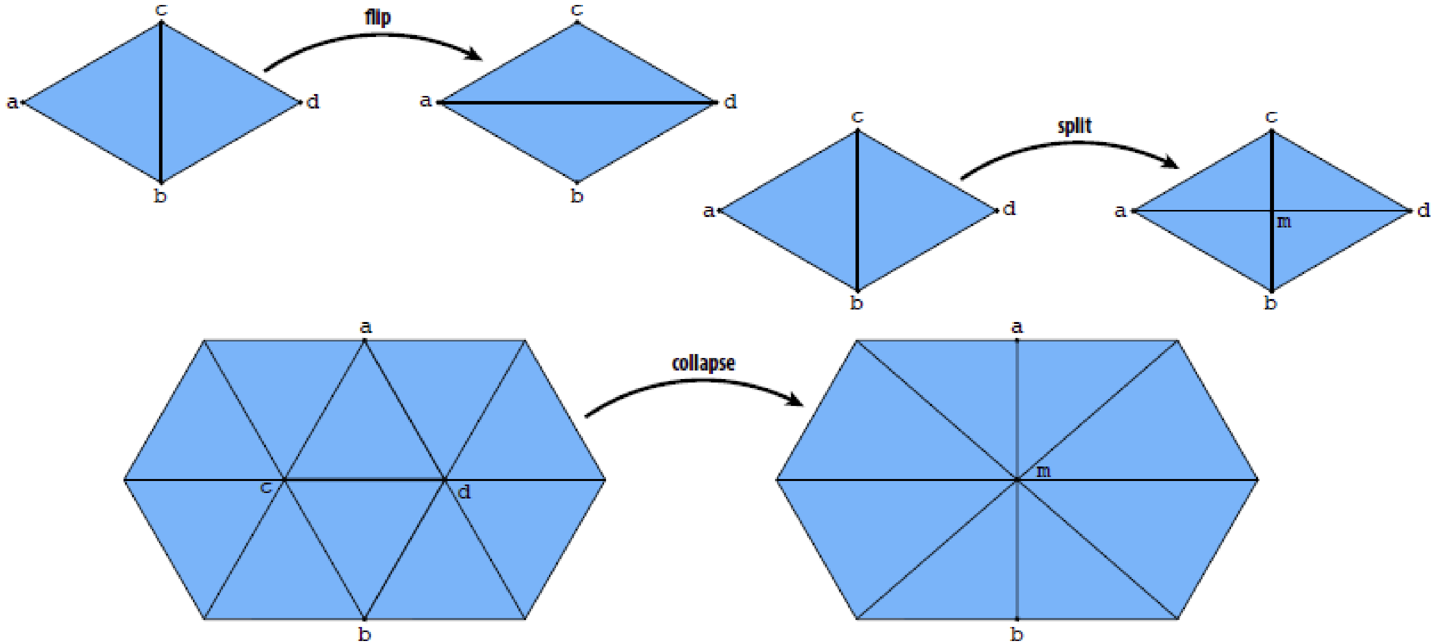
Junjie Cao @ DLUT

Spring 2017

http://jjcao.github.io/ComputerGraphics/

# Processing geometry with halfedges

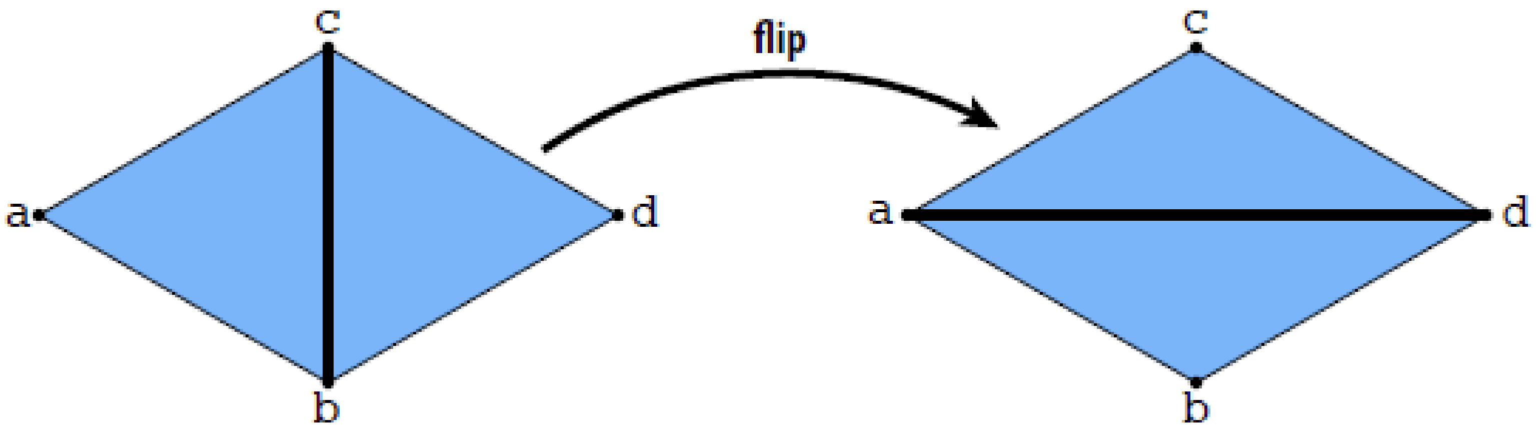
- Remember key feature of linked list: insert/delete elements
- Same story with halfedge mesh (“linked list on steroids”)
- Several atomic operations for triangle meshes:




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

# Edge Flip

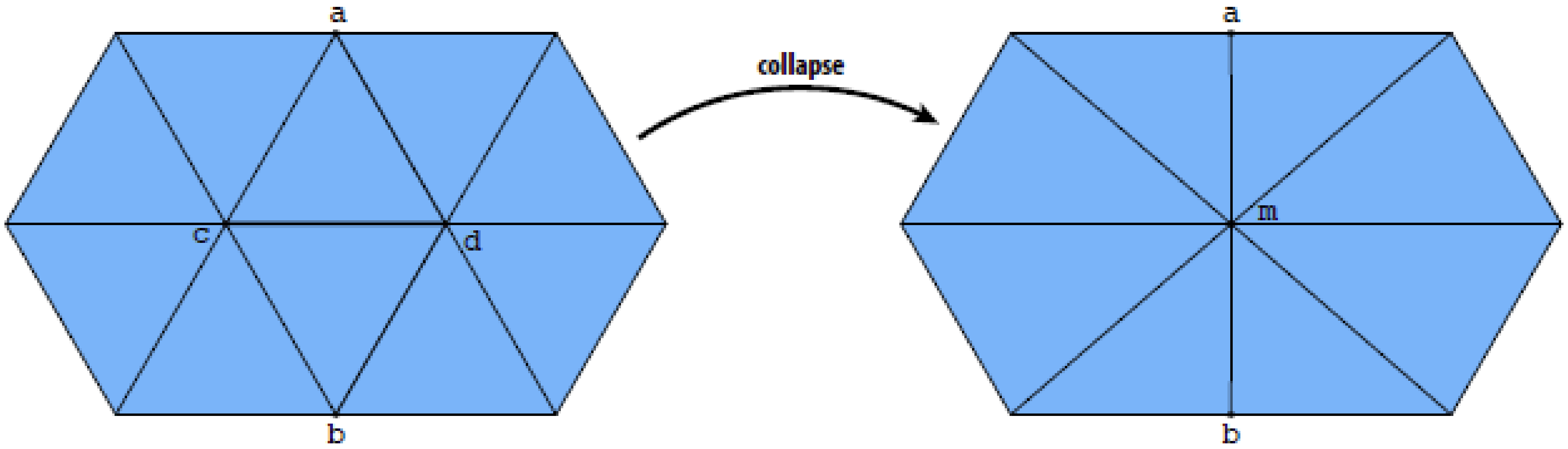
- Triangles (a,b,c), (b,d,c) become (a,d,c), (a,b,d):




- 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 Collapse

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

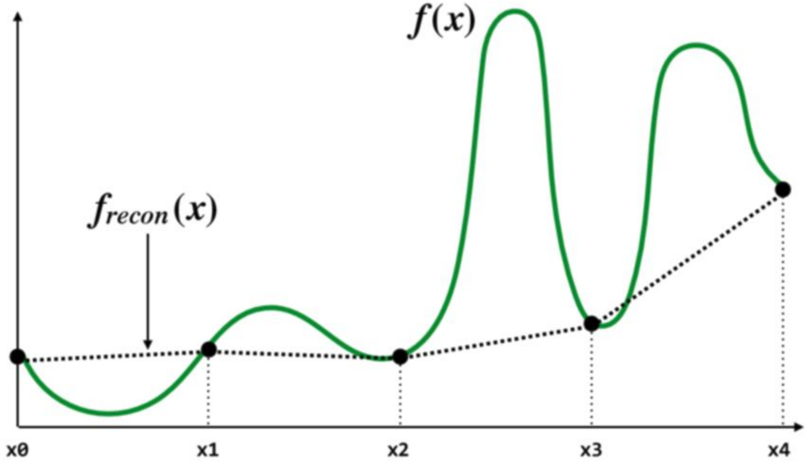



- **Now have to *delete* elements.**
- **Still lots of pointer assignments!**
- **Q: How would we implement this with a polygon soup?**
- **Any other good way to do it? (E.g., different data structure?)**

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

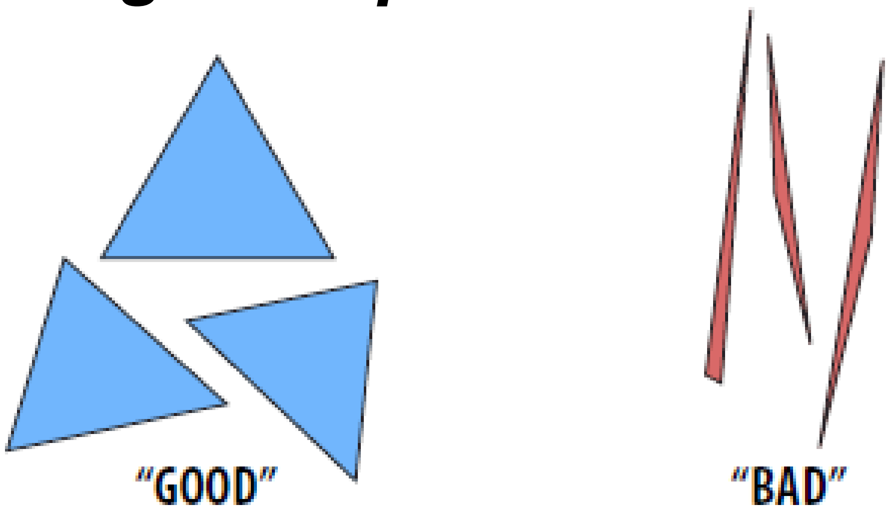
# Remeshing as resampling

- 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

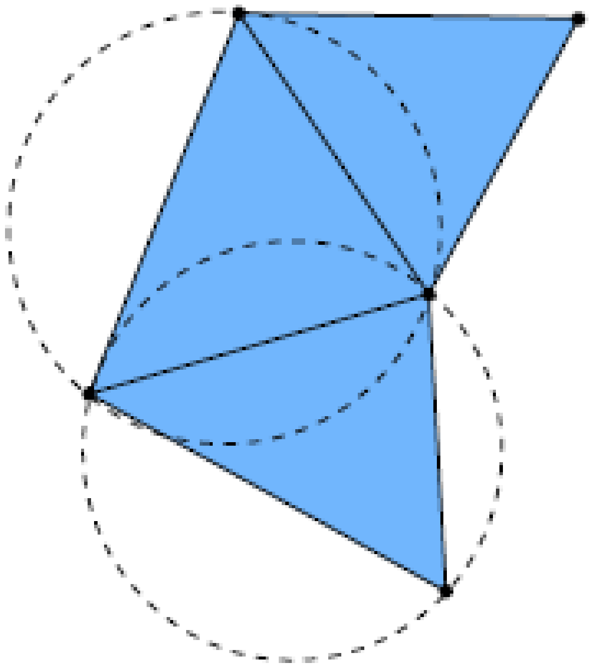



# What makes a “good” triangle mesh?

- One rule of thumb: *triangle shape*

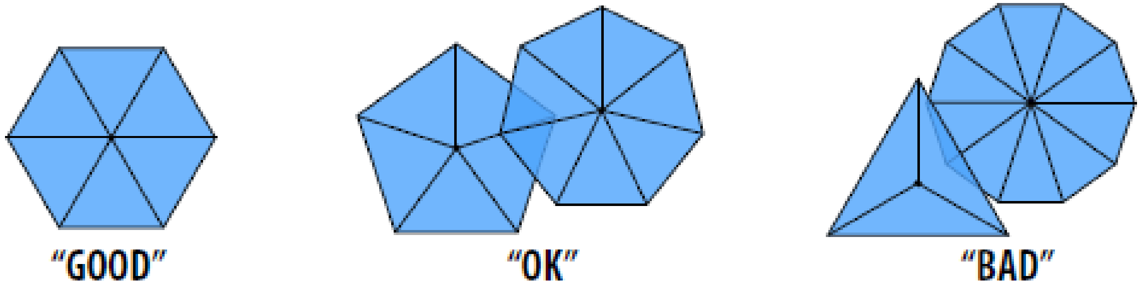



- More specific condition: *Delaunay*
- “Circumcircle interiors contain no vertices.”
- Not *always* a good condition, but often*
- especially important for simulation
- for approximation, long triangles may be better

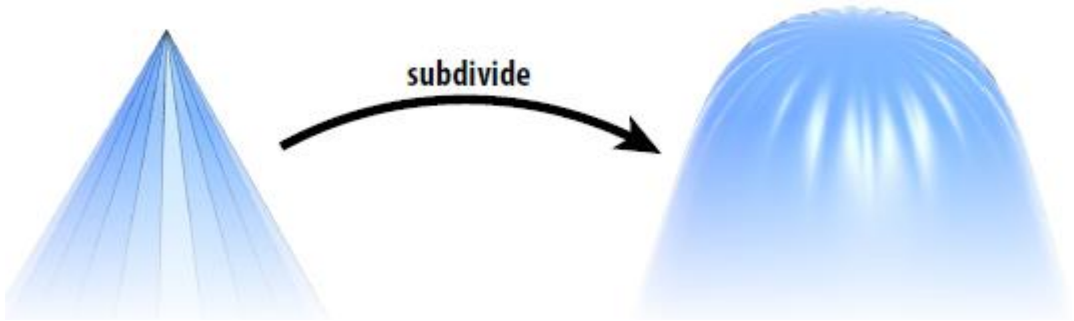


# What else constitutes a good mesh?

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



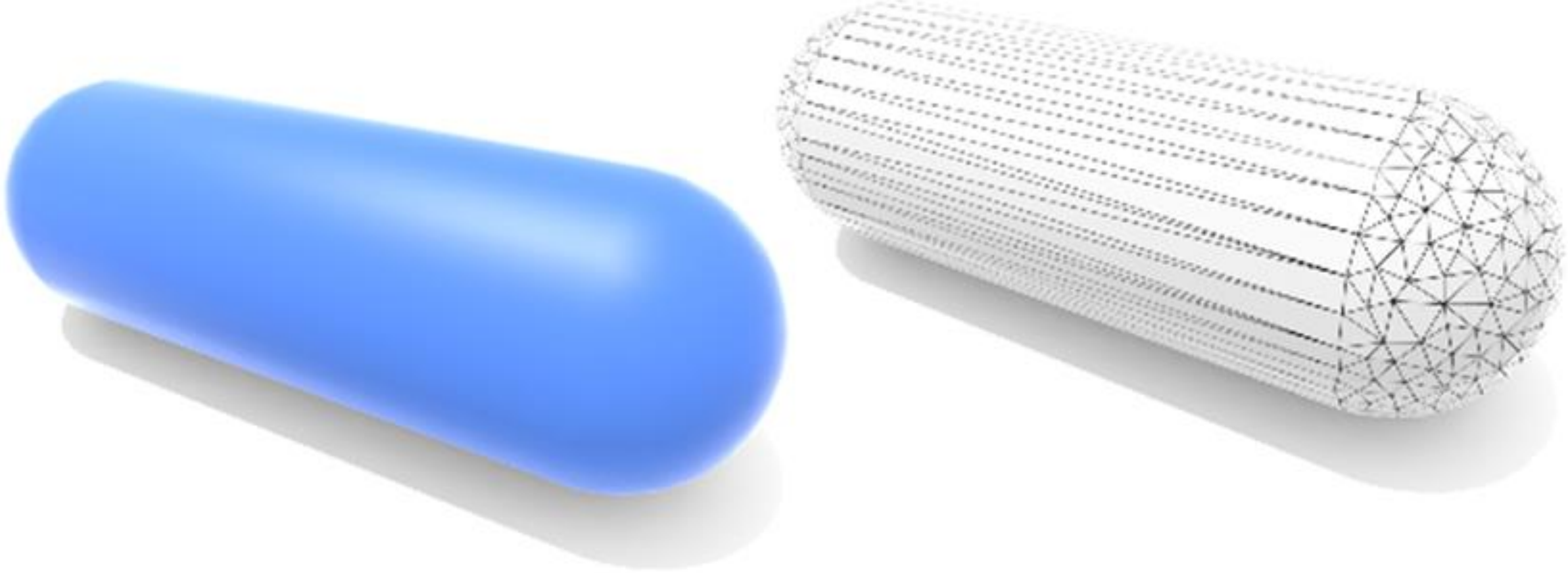

- Why? Better triangle shape, important for (e.g.) subdivision:




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

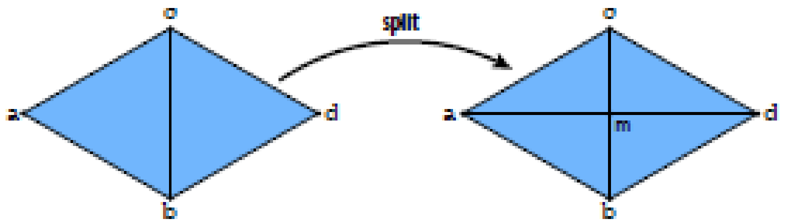
# What *else* makes a “good” mesh?

- 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.

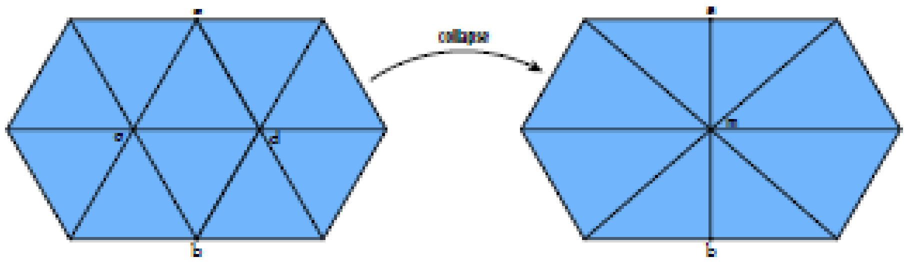


# How do we resample? Already know how!

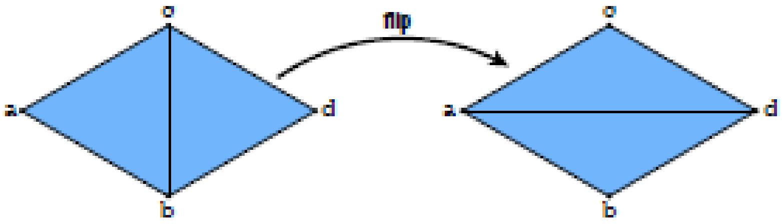
- **Edge split is (local) upsampling:**




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




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

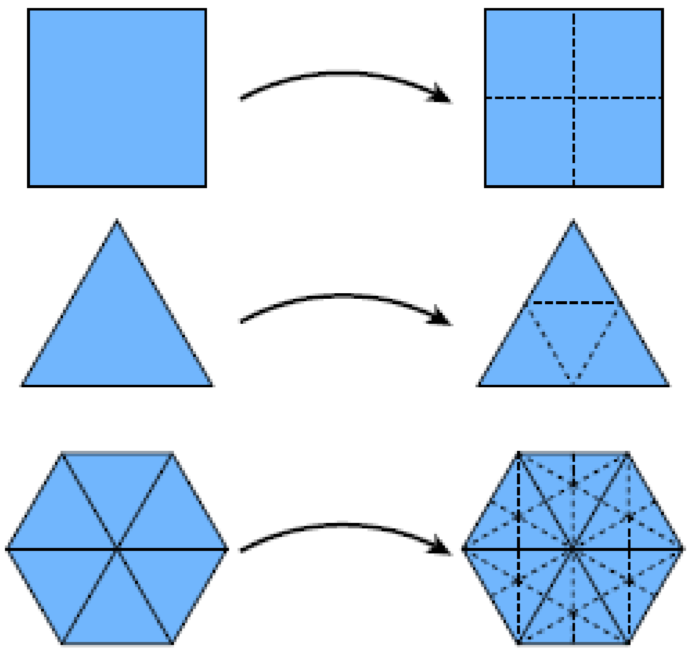


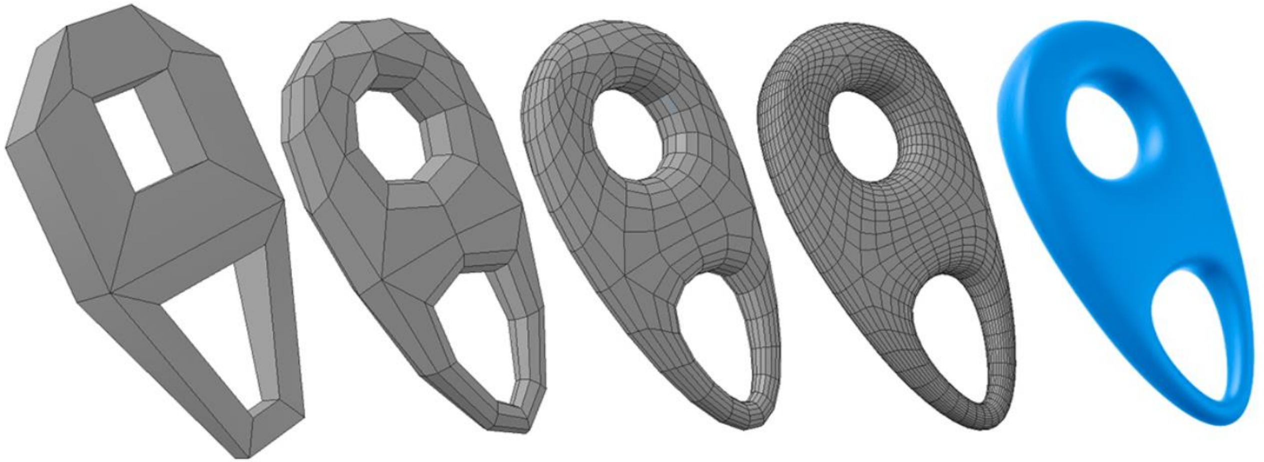

- **Still need to intelligently decide *which* edges to modify!**

# Which edges should we split to upsample the whole mesh?

# Subdivision as Upsampling

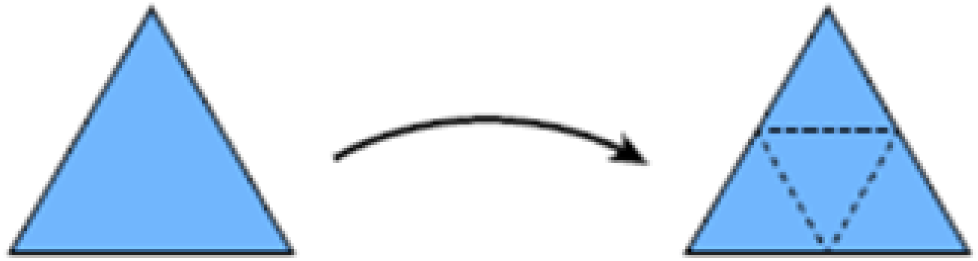
- Repeatedly split each element into smaller pieces
- Replace vertex positions with weighted average of neighbors
- Main considerations:
  - interpolating vs. approximating
  - limit surface continuity (*C1*, *C2*, ...)
  - behavior at irregular vertices
- Many options:
  - Quad: Catmull-Clark
  - Triangle: Loop, Butterfly, Sqrt(3)

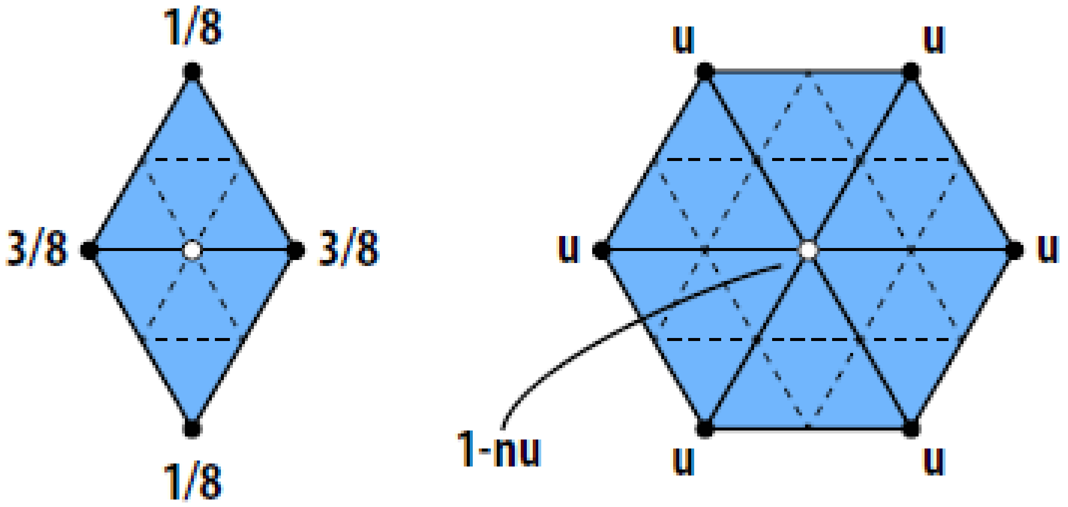




# Loop Subdivision

- **Fairly common subdivision rule for triangles**
- **Curvature is continuous away from irregular vertices ("*C2*")**
- ***Approximating*, not interpolating**
- **Algorithm:**
  - **Split each triangle into four**
  - **Assign new vertex positions according to weights:**

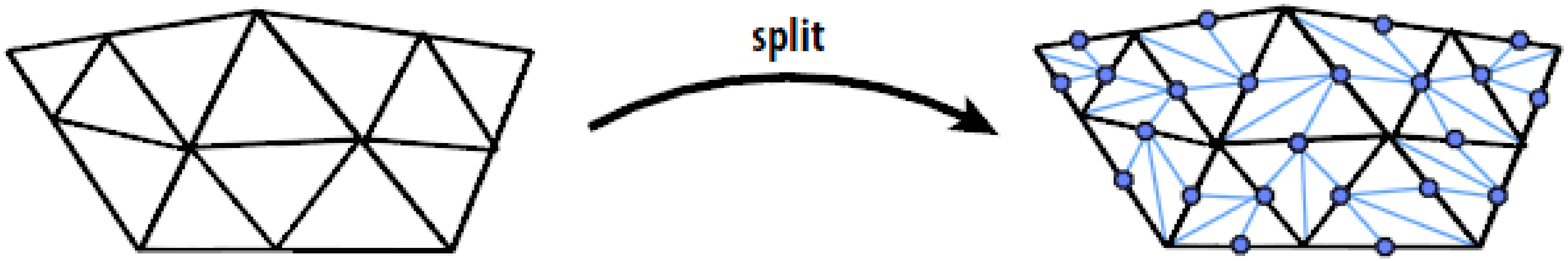




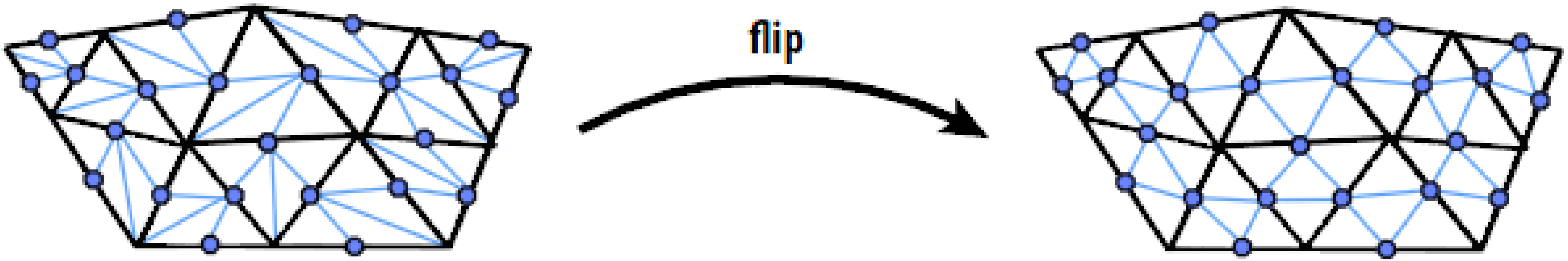

n: vertex degree
u: 3/16 if n=3, 3/(8n) otherwise

# Loop Subdivision via Edge Operations

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




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



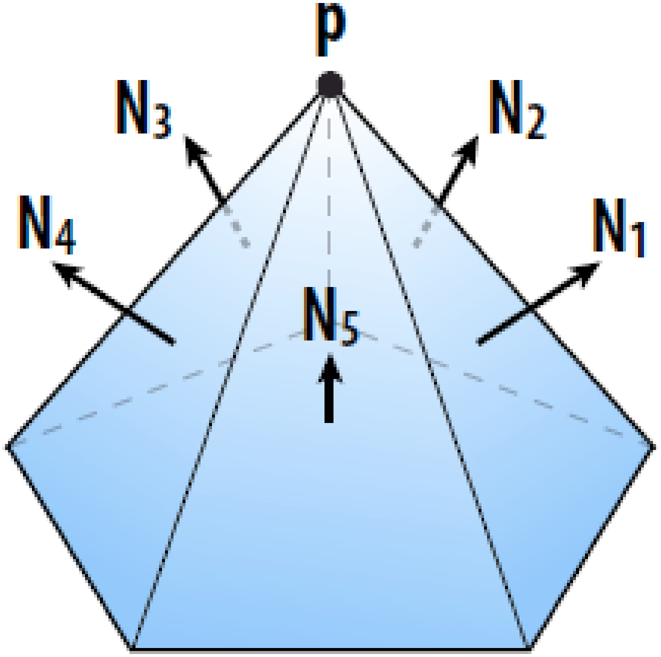
# What if we want fewer triangles?

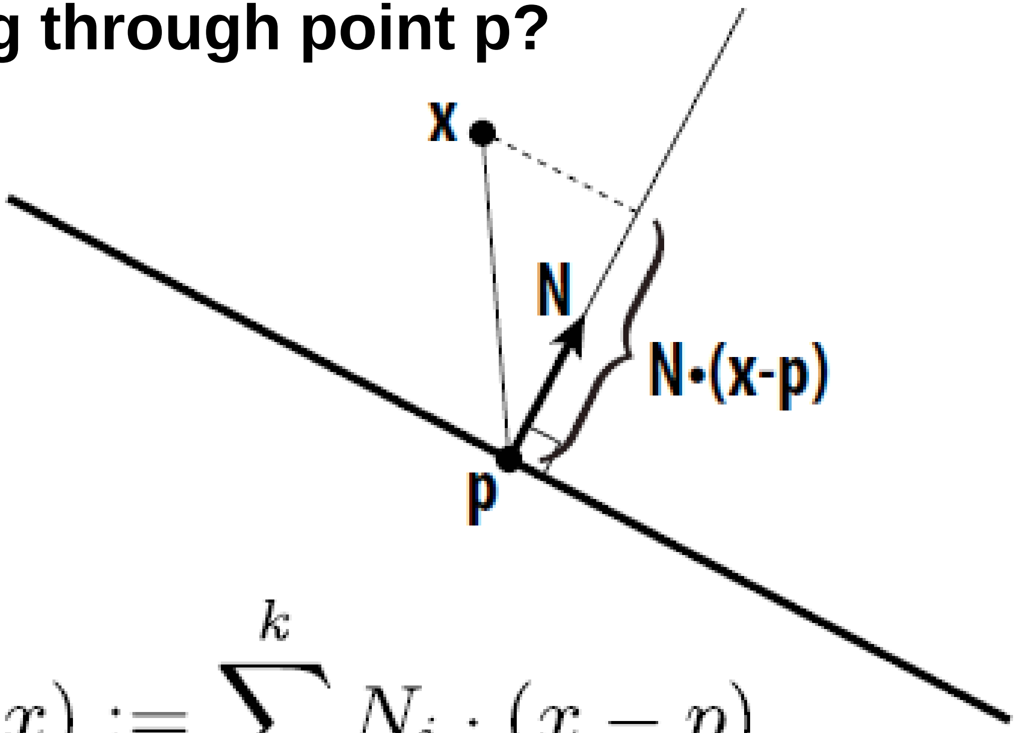
# Simplification via Quadric Error Metric

- **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!**

# 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•(x-p)**
- **Sum of distances:**




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

# Quadric Error - Homogeneous Coordinates

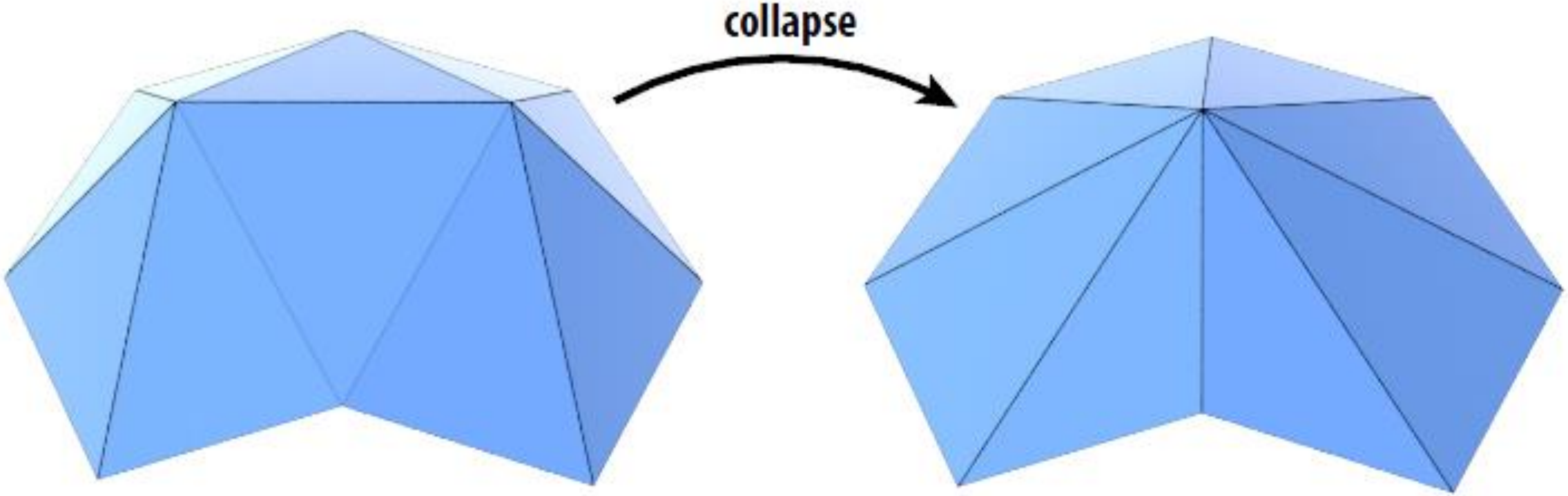
- **Suppose in coordinates we have**
  - **a query point (x,y,z)**
  - **a normal (a,b,c)**
  - **an offset d := -(x,y,z) • (a,b,c)**
- **Then in homogeneous coordinates, let**
  - **u := (x,y,z,1)**
  - **v := (a,b,c,d)**

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

- ***Signed* distance to plane is then just u•v = ax+by+cz+d**
- ***Squared* distance is (u’v)^2 = u’(vv’)u =: u’Qu**
- **Key idea: *matrix Q encodes distance to plane***
- **Q is symmetric, contains 10 unique coefficients (small storage)**

# Quadric Error of Edge Collapse

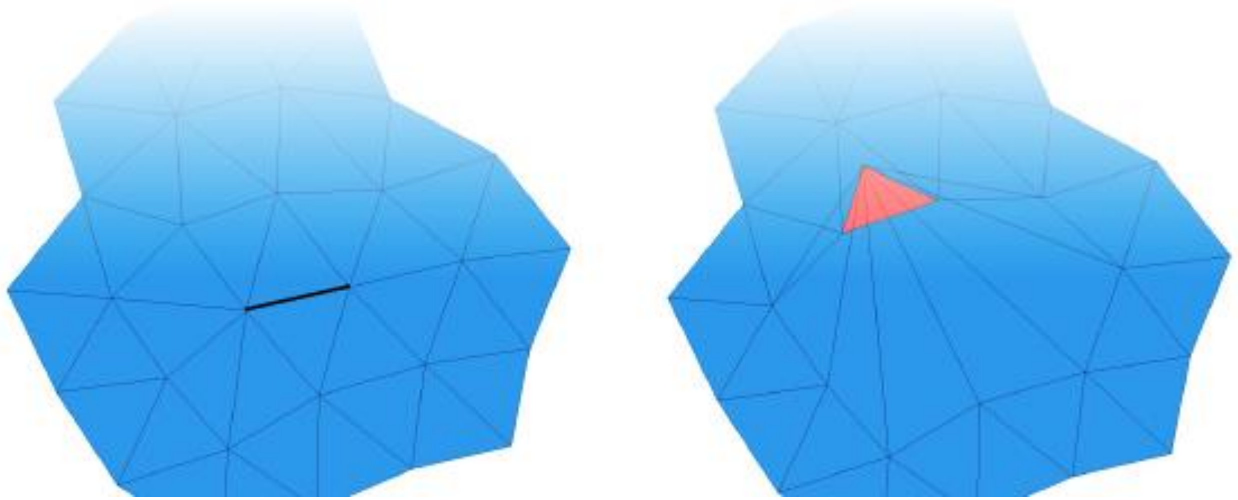
- How much does it cost to collapse an edge?
- Idea: compute edge midpoint, measure quadric error




- 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 Qi and Qj to get new quadric Qk**
  - **update cost of any edge touching new vertex k**
- **Store edges in *priority queue* to keep track of minimum cost**



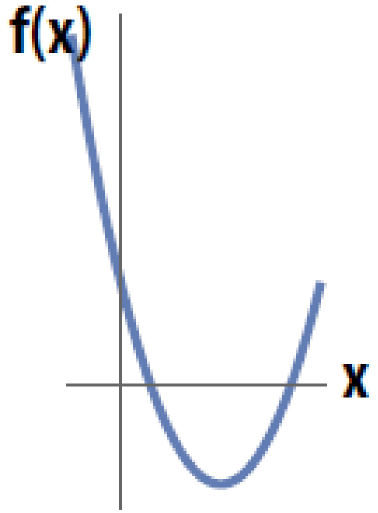
# Review: Minimizing a Quadratic Function

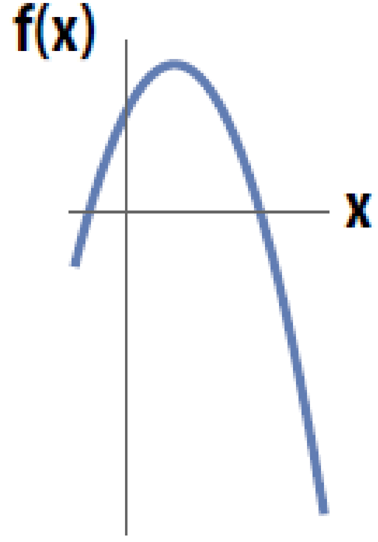
- Suppose I give you a function f(x) = ax2+bx+c
- Q: What does the graph of this function look like?
- Could also look like this!
- Q: How do we find the *minimum?*
- A: Look for the point where the function isn't
- changing (if we look "up close")
- I.e., find the point where the *derivative* vanishes

$$f'(x) = 0$$

$$2ax + b = 0$$

$$x = -b/2a$$



# 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) = ax2 + bxy + cy2 + dx + ey + g
- Can always (always!) write quadratic polynomial using a
- *symmetric* matrix (and a vector, and a constant):

$$f(x,y) = \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} a & b/2 \\ b/2 & c \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} d & e \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + g$$

$$= \mathbf{x}^\mathsf{T} A \mathbf{x} + \mathbf{u}^\mathsf{T} x + g$$

(this expression works for *any* n!)

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

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

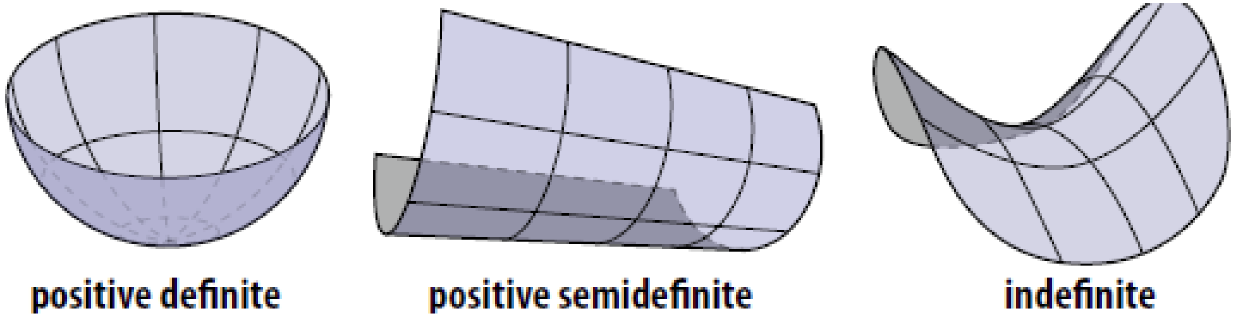
$$\mathbf{x} = -\tfrac{1}{2} A^{-1} \mathbf{u}$$

# Positive Definite Quadratic Form

- Just like our 1D parabola, critcal 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 = ax2 > 0. In other words: a is positive!
- 2D: Graph of function looks like a “bowl”:




- Positive-definiteness is *extremely important* in CG: 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

$$\min_u \mathbf{u}^\mathsf{T} K \mathbf{u}$$

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

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

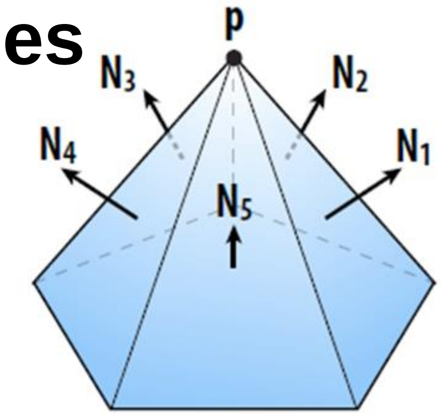
$$= \mathbf{x}^\mathsf{T} B \mathbf{x} + 2\mathbf{w}^\mathsf{T}\mathbf{x} + d^2$$

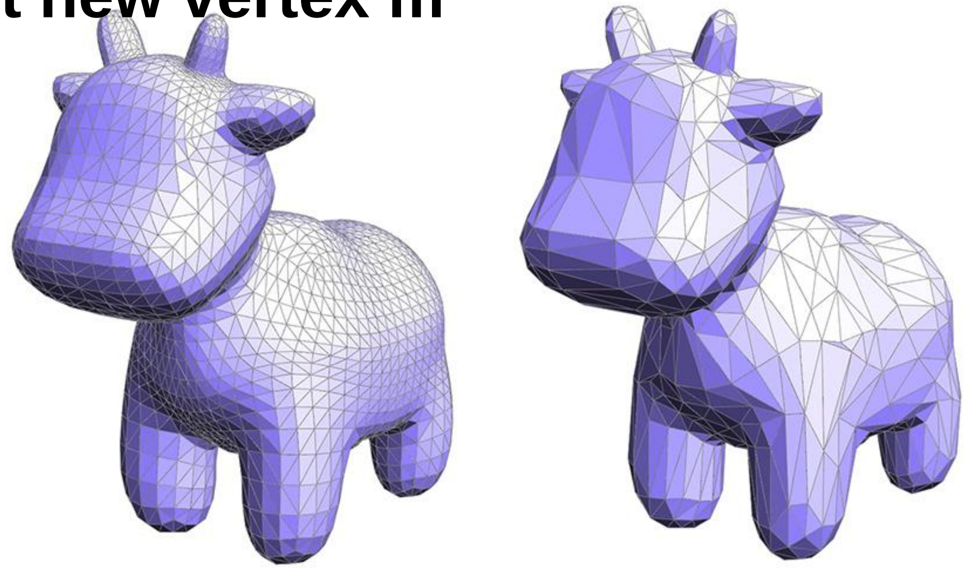
- Now we have a quadratic form in the 3D position x.
- Can minimize as before:

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

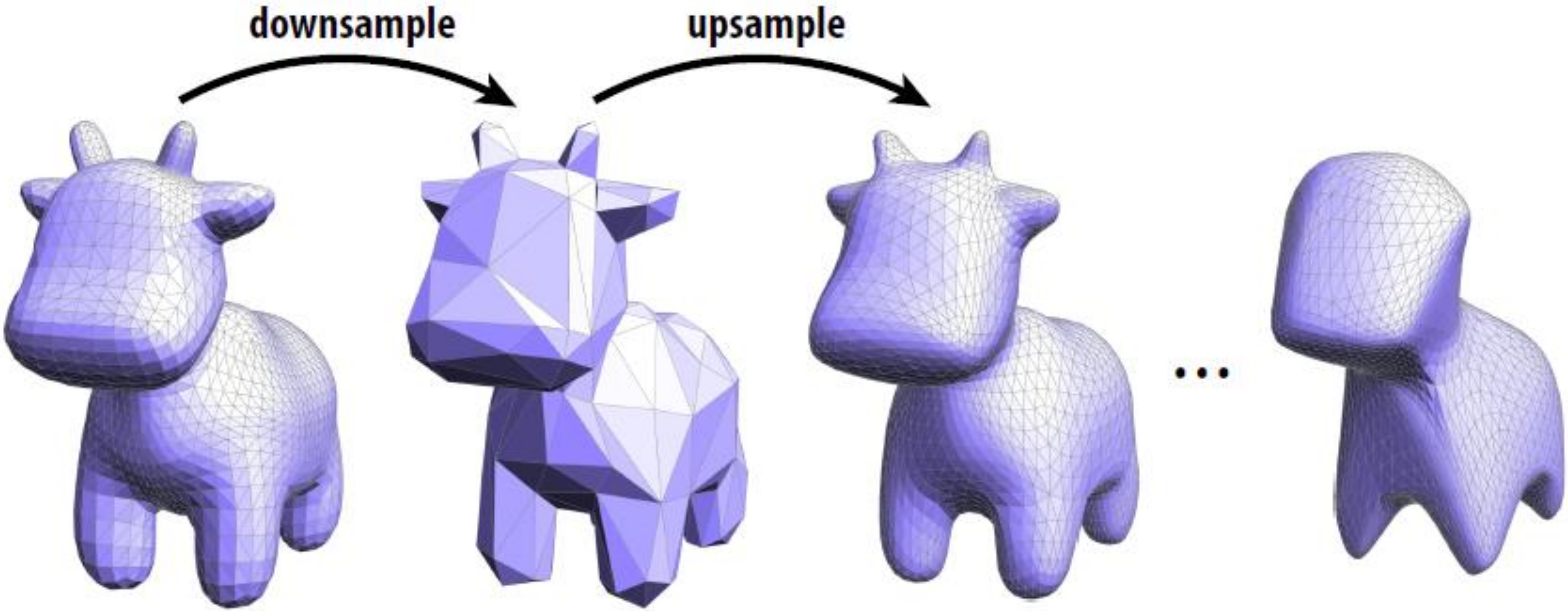
# Quadric Error Simplification: Final Algorithm

- Compute K for each triangle (distance to plane)
- Set K at each vertex to sum of Ks from incident triangles
- Set K at each edge to sum of Ks 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 Ki and Kj to get quadric Km at m
  - update cost of edges touching m
- *More details in assignment writeup!*



# Demo: Danger of Resampling

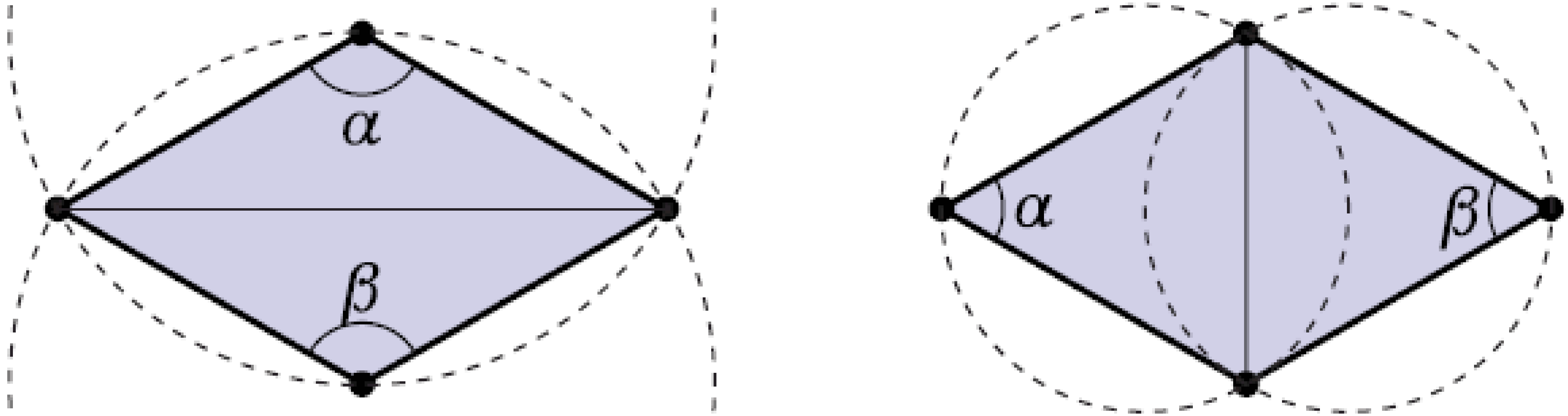



(Q: What happens with an image?)

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

# How do we make a mesh “more Delaunay”?

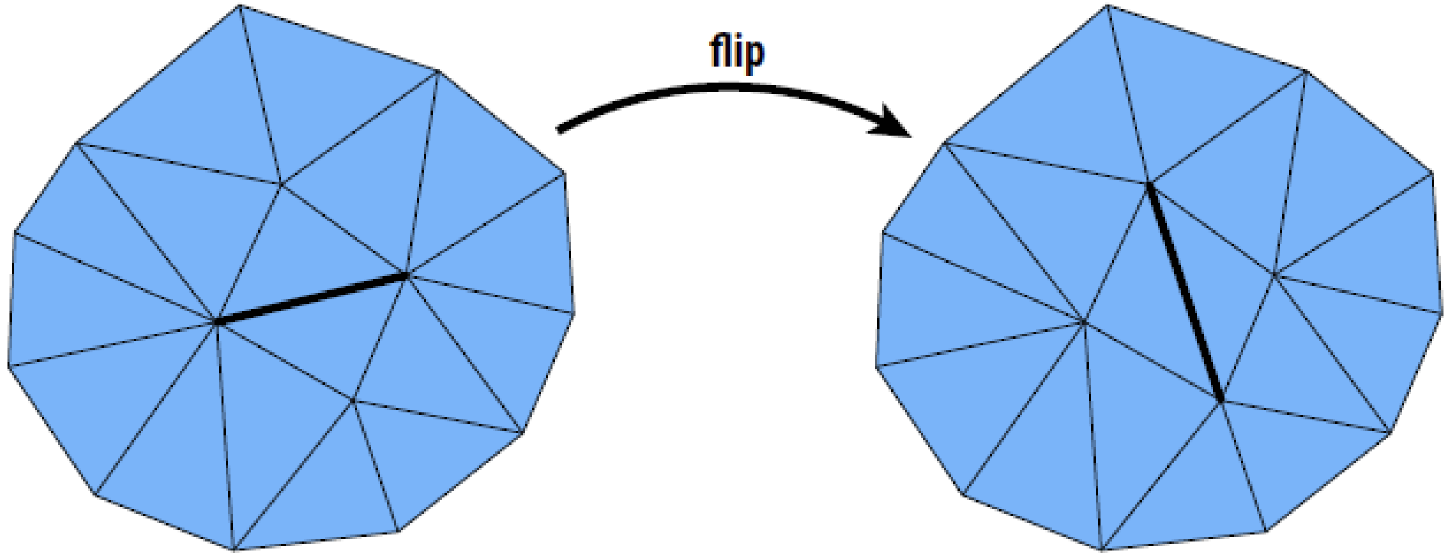
- Already have a good tool: edge flips!
- If $\alpha+\beta > \pi$, flip it!




- FACT: in 2D, flipping edges eventually yields Delaunay mesh
- Theory: worst case O(n2); no longer true for surfaces in 3D.
- Practice: simple, effective way to improve mesh quality

# Alternatively: how do we improve degree?

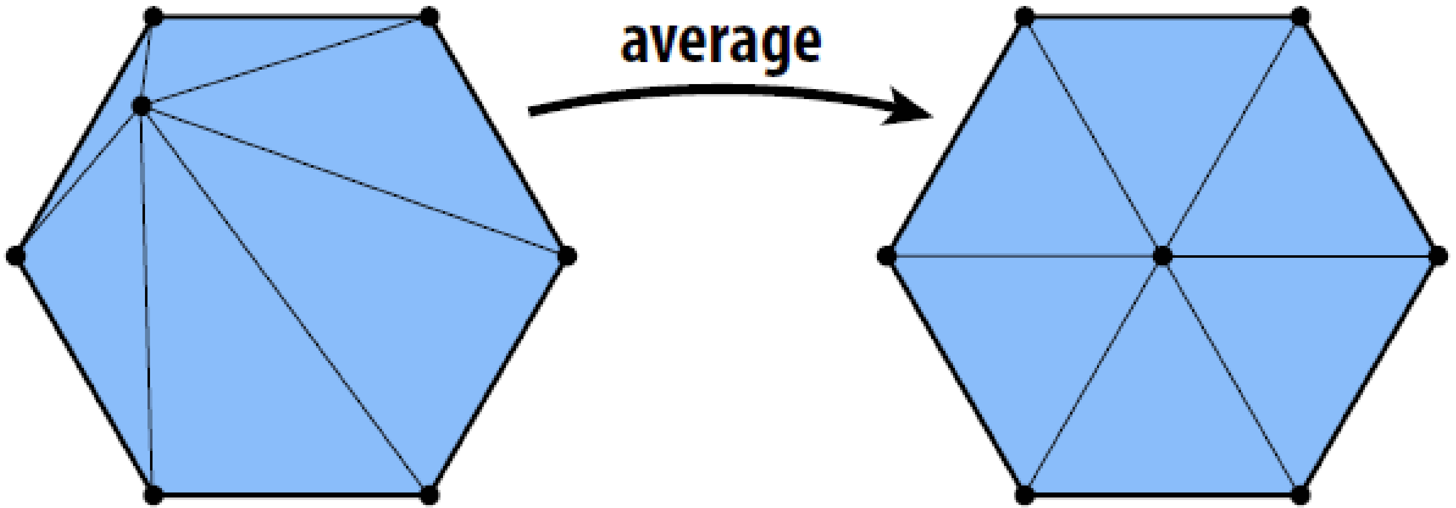
- Same tool: edge flips!
- If total deviation from degree-6 gets smaller, flip it!




- 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”?

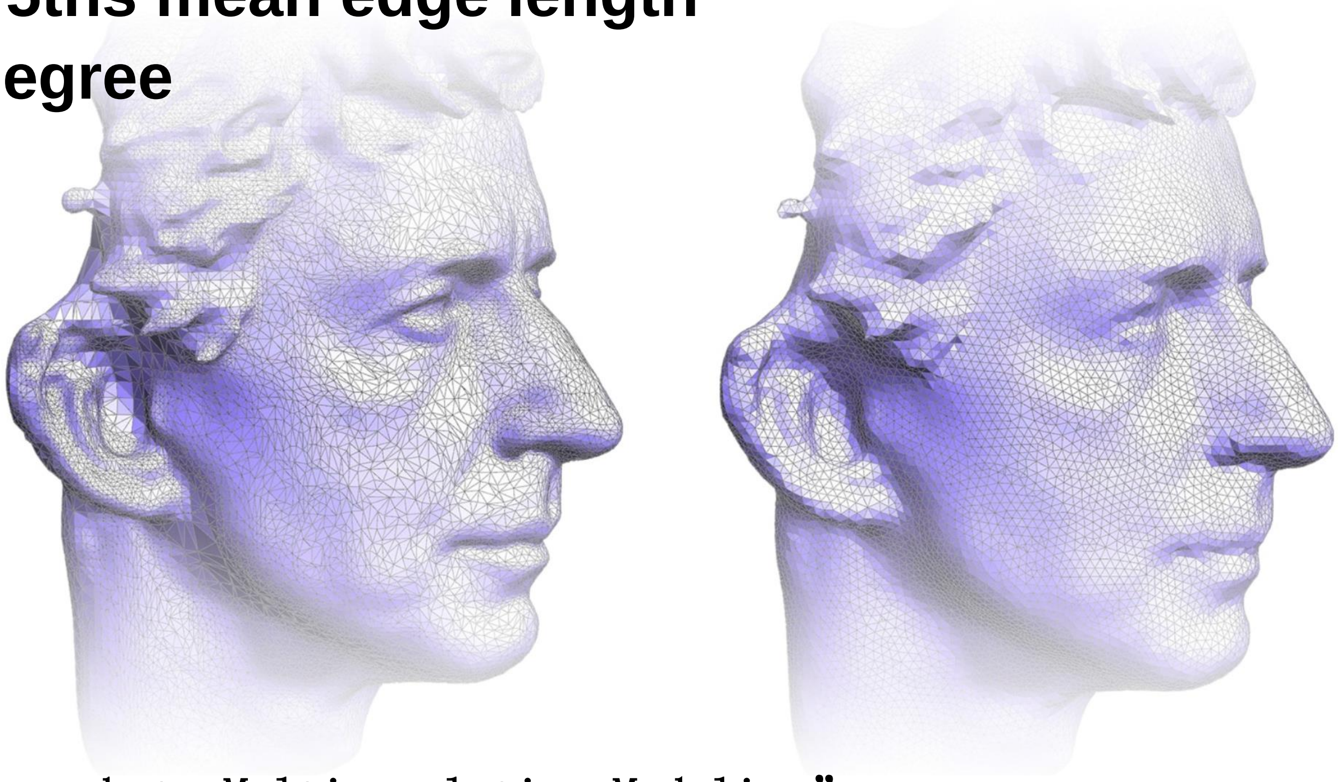
- Delaunay doesn’t mean triangles are “round” (angles near 60° )
- Can often improve shape by centering vertices:




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

# 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”

# Summary

- **Extend signal processing to curved shapes**
  - **encounter familiar issues (sampling, aliasing, etc.)**
  - **some new challenges (irregular sampling, no FFT, etc.)**
- **Focused on resampling triangle meshes**
  - **local: edge flip, split, collapse**
  - **global: subdivision, quadric error, isotropic remeshing**

# Thank you