Lecture 8:

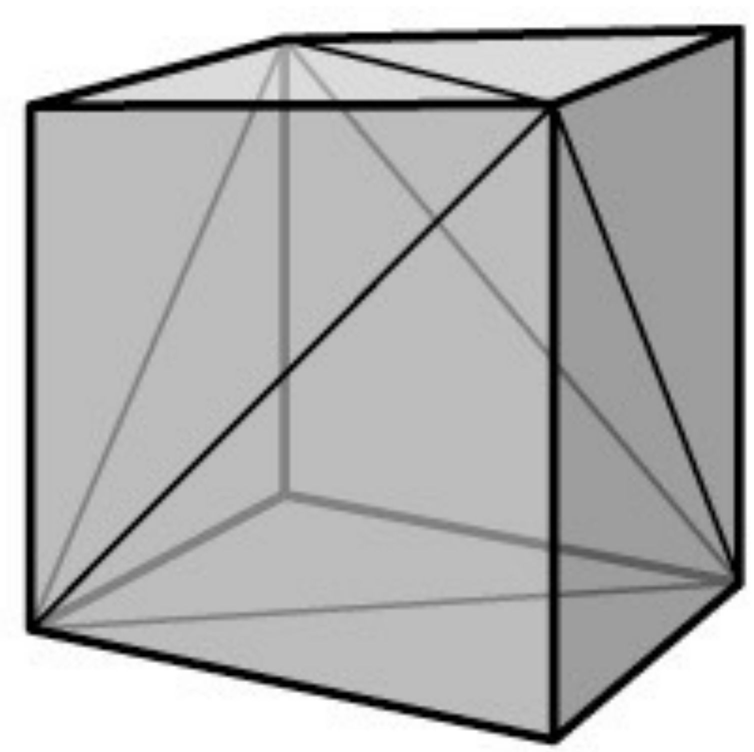
Mesh Representations & Geometry Processing

Computer Graphics and Imaging

UC Berkeley CS184/284A

Mesh Examples

A Small Triangle Mesh

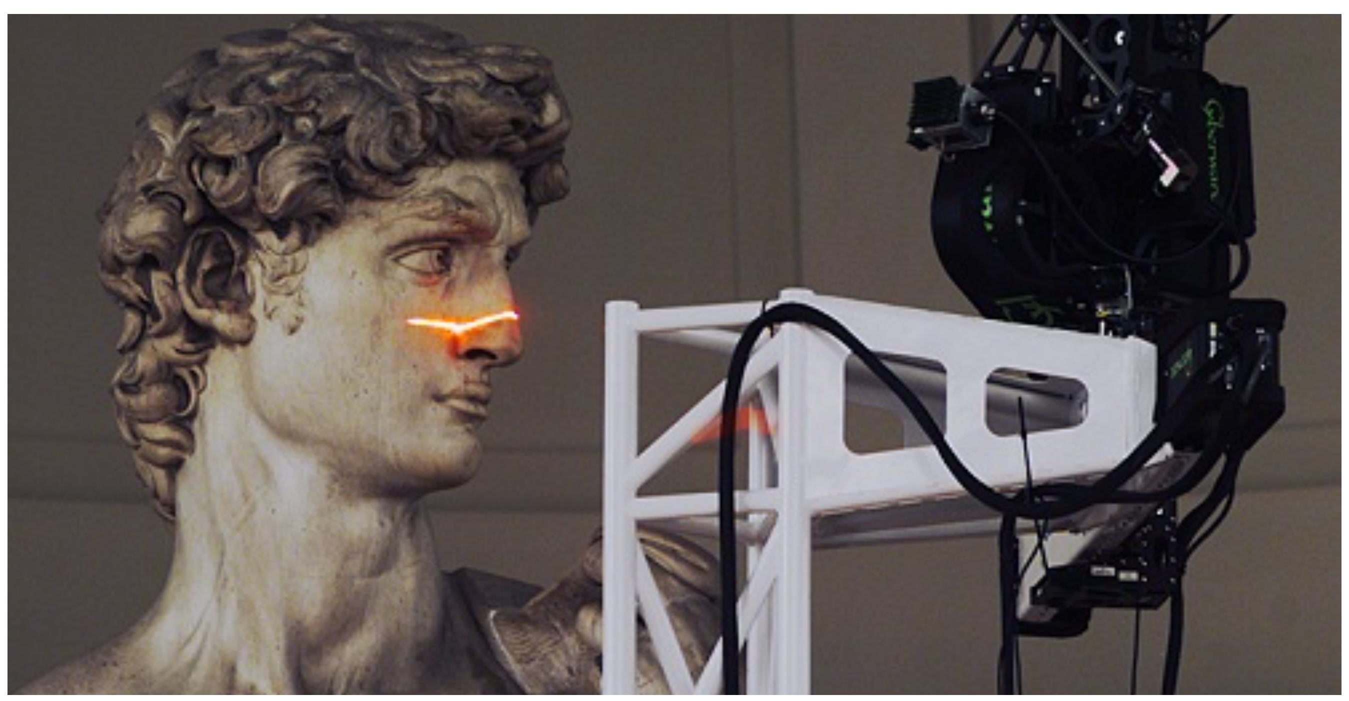


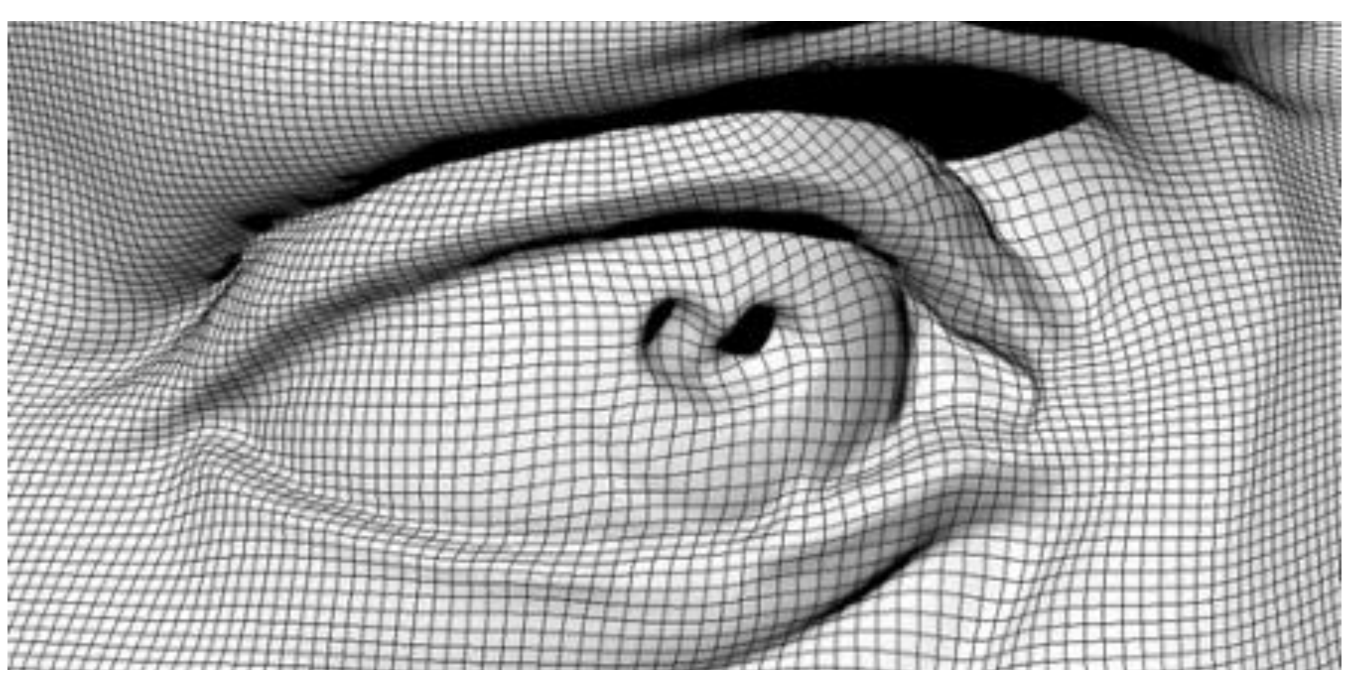
8 vertices, 12 triangles

A Large Triangle Mesh

David

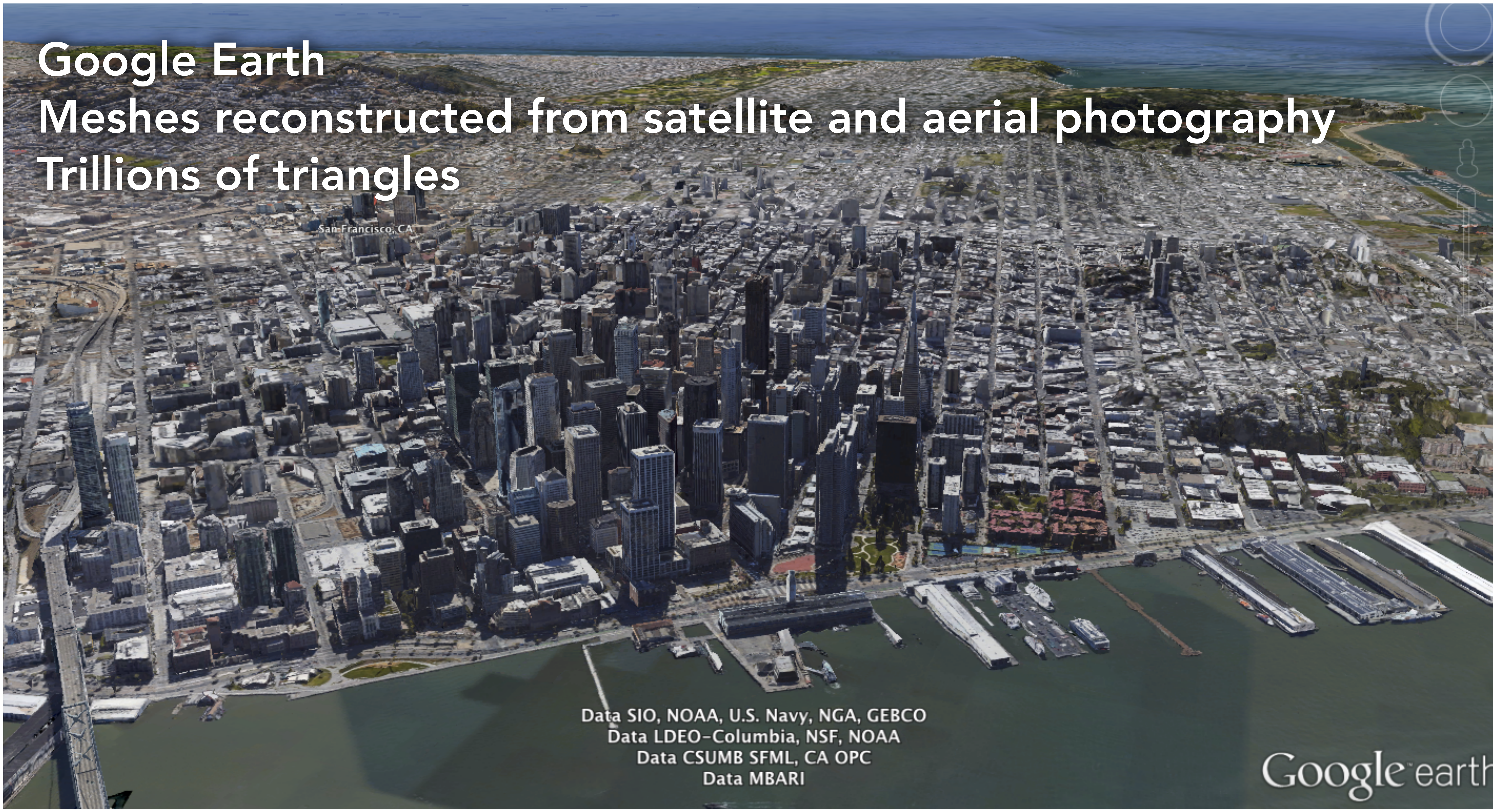
Digital Michelangelo Project
28,184,526 vertices
56,230,343 triangles



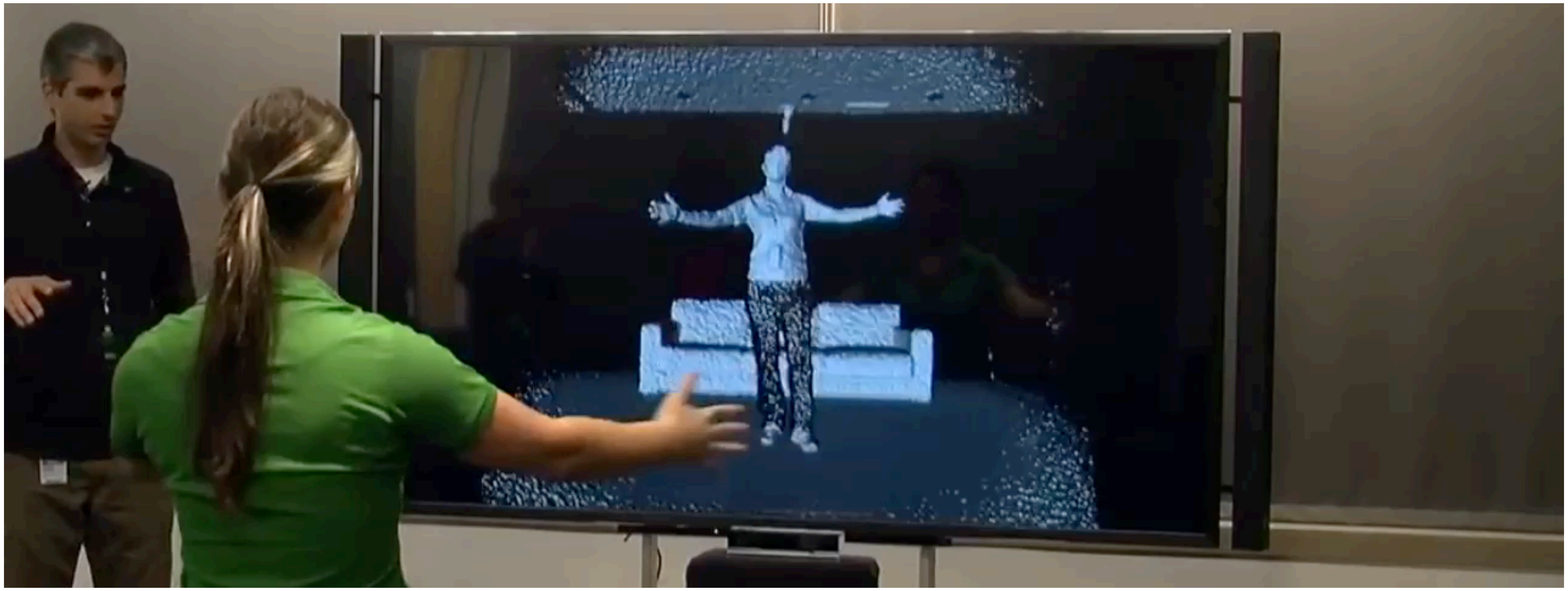


A Very Large Triangle Mesh

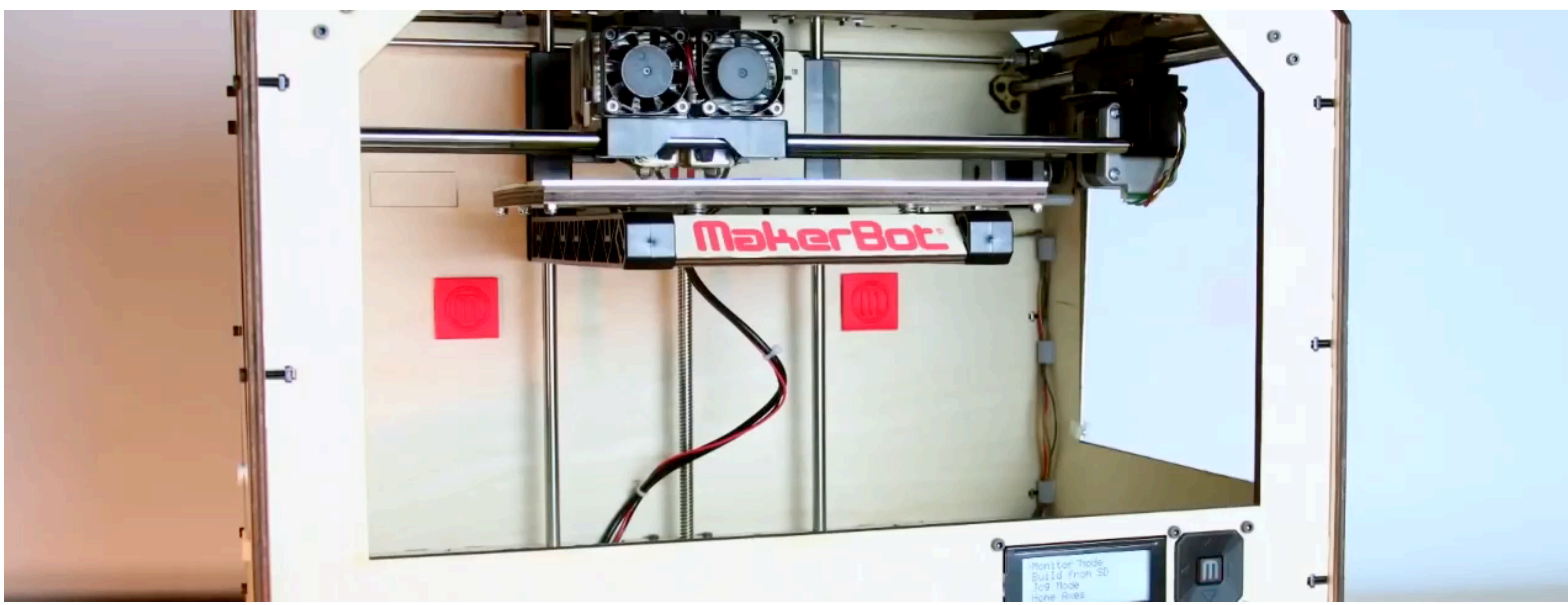
Google Earth
Meshes reconstructed from satellite and aerial photography
Trillions of triangles



Digital Geometry Processing



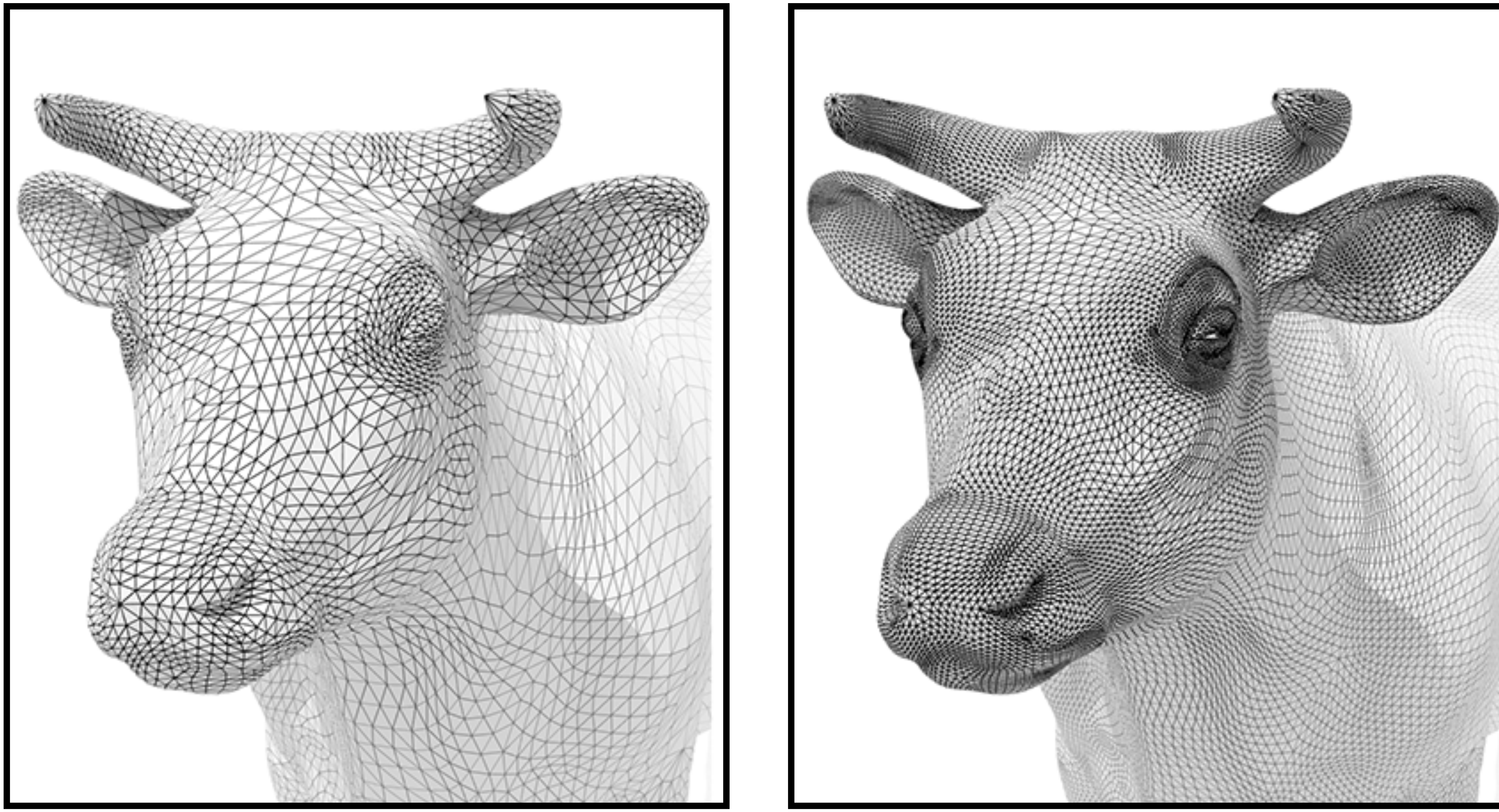
3D Scanning



3D Printing

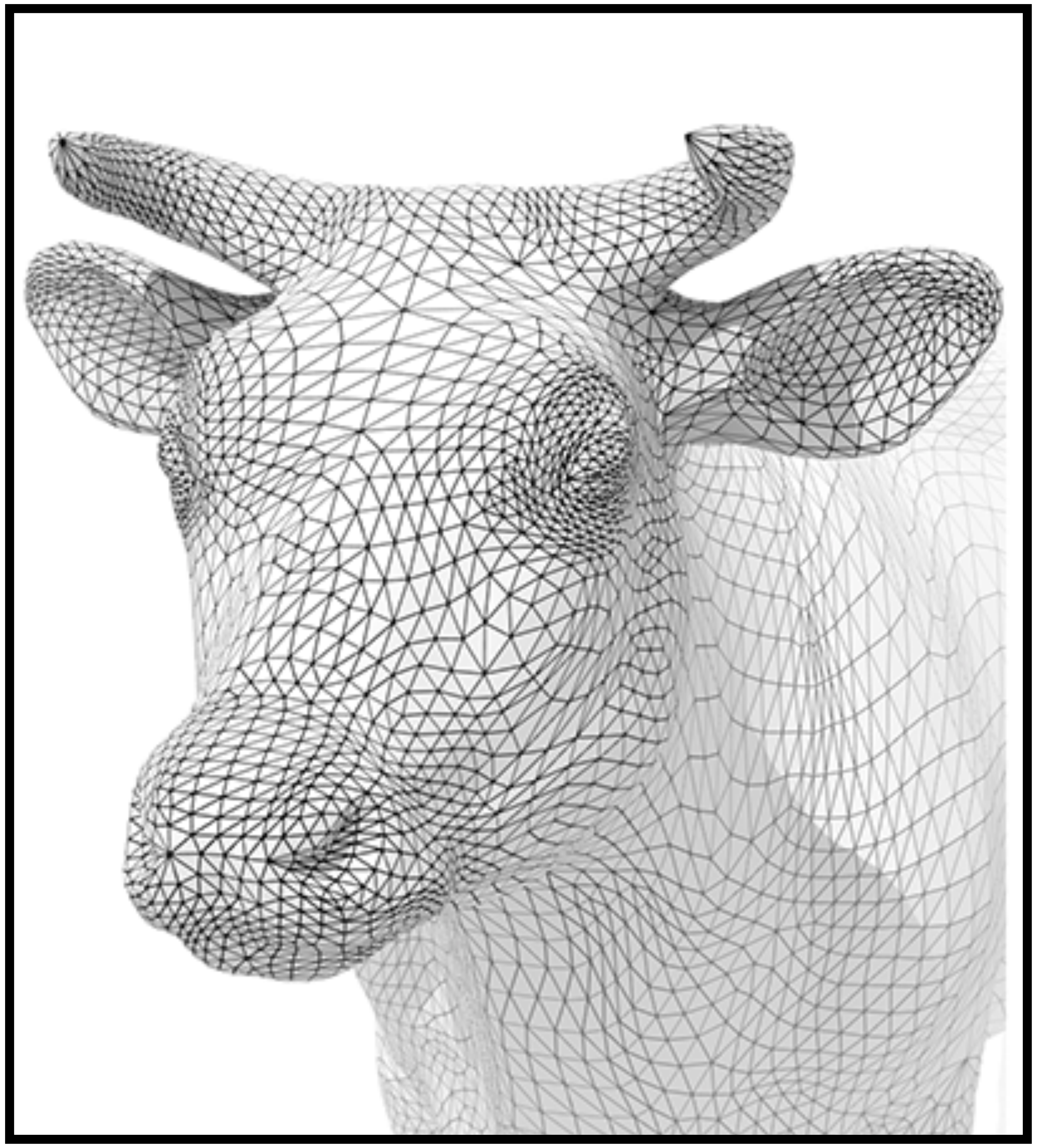
Geometry Processing Tasks: 3 Examples

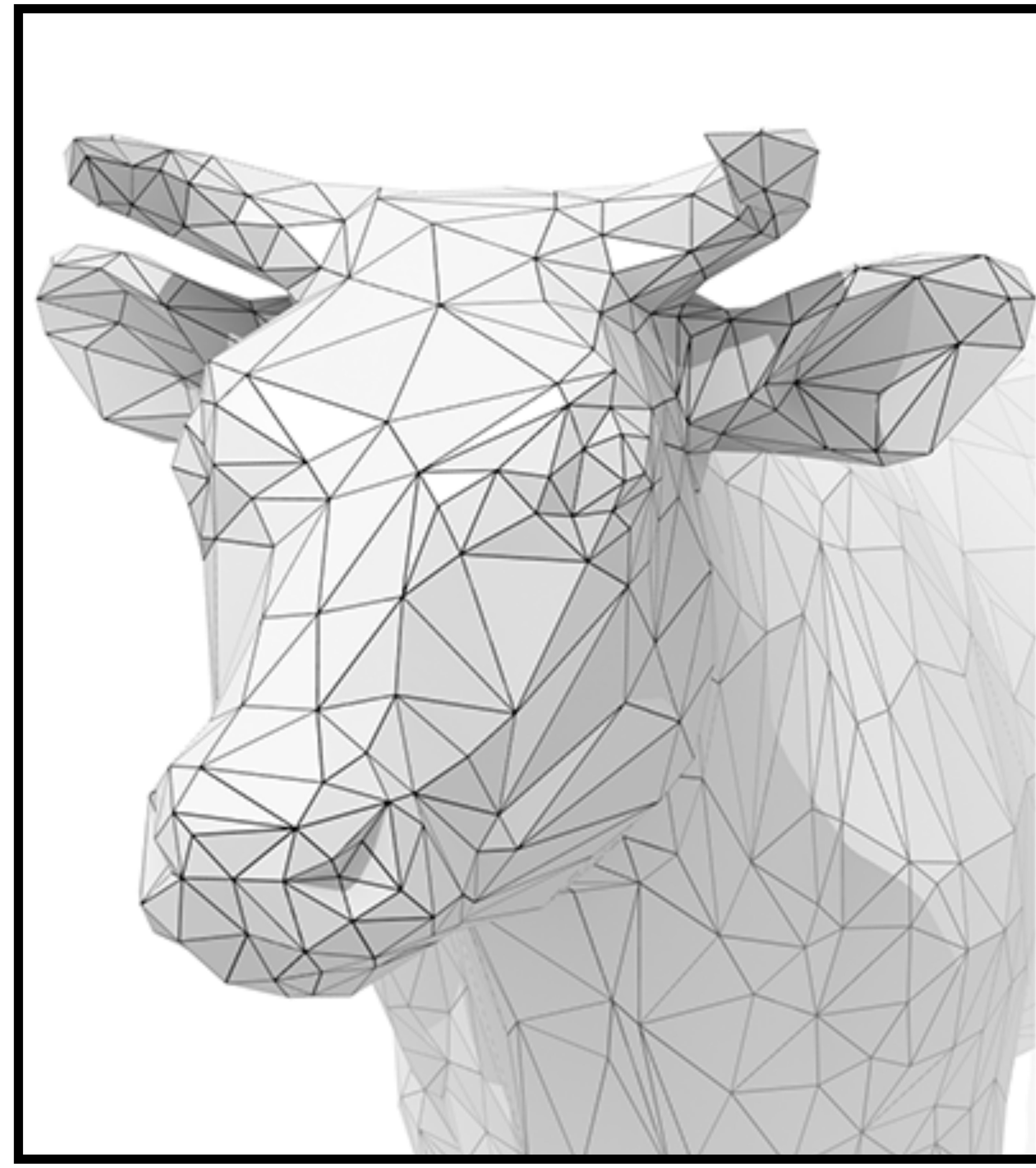
Mesh Upsampling – Subdivision



Increase resolution via subdivision

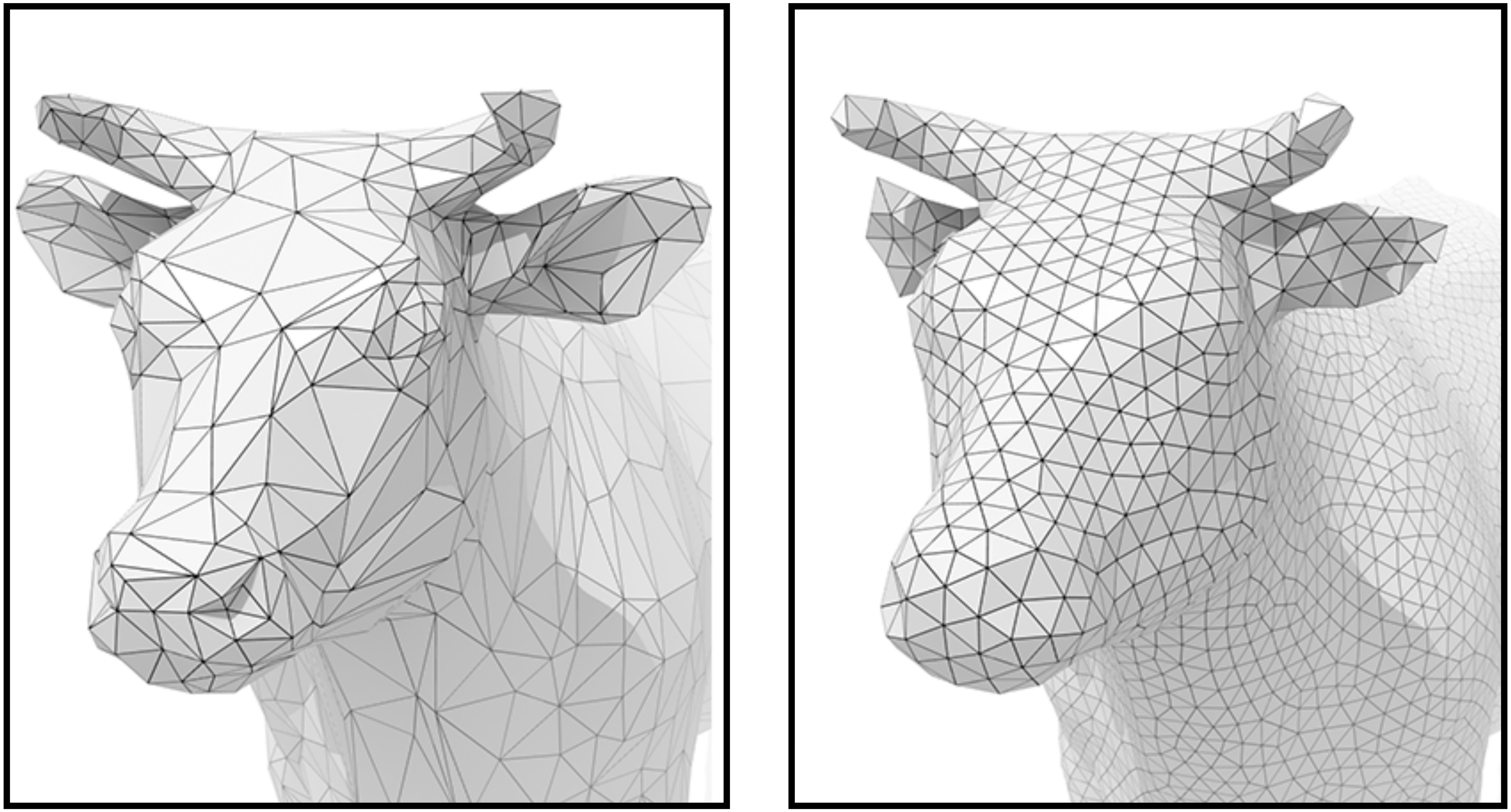
Mesh Downsampling – Simplification





Decrease resolution; try to preserve shape/appearance

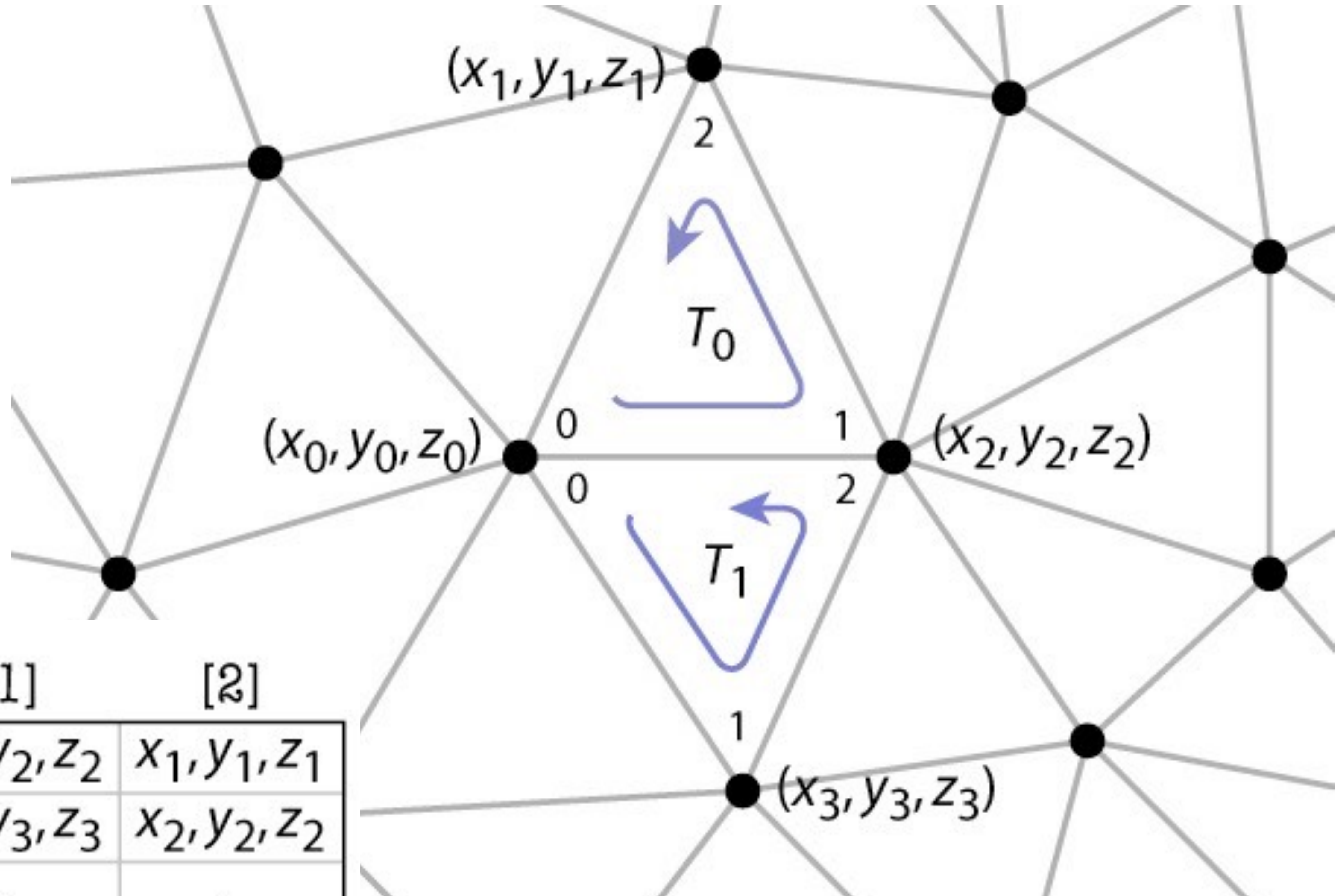
Mesh Regularization



Modify sample distribution to improve quality

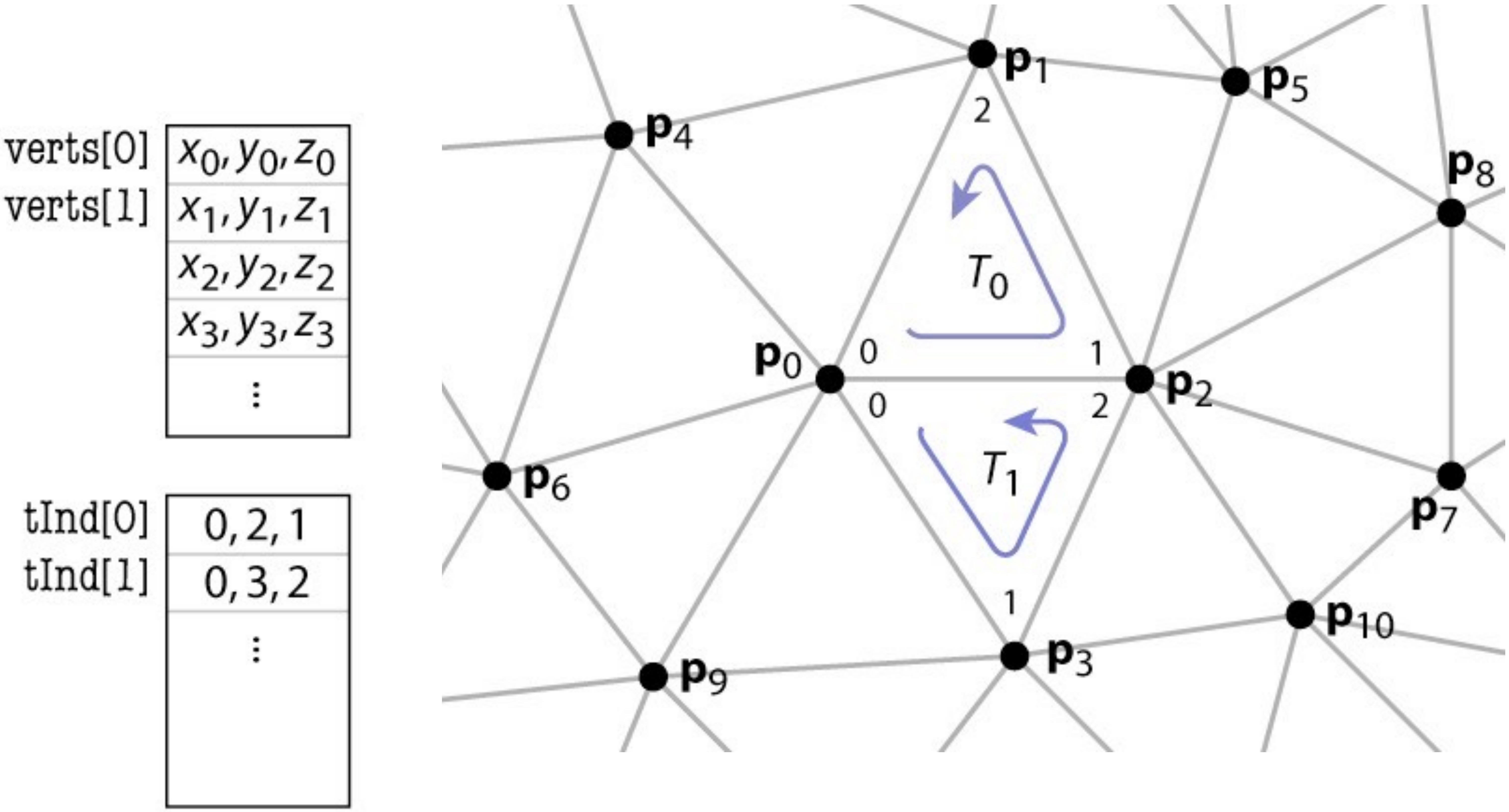
Mesh Representations

List of Triangles



	[0]	[1]	[2]
tris[0]	x_0, y_0, z_0	x_2, y_2, z_2	x_1, y_1, z_1
tris[1]	x_0, y_0, z_0	x_3, y_3, z_3	x_2, y_2, z_2
	⋮	⋮	⋮

Lists of Points / Indexed Triangle



Comparison

Triangles

+ Simple

– Redundant information

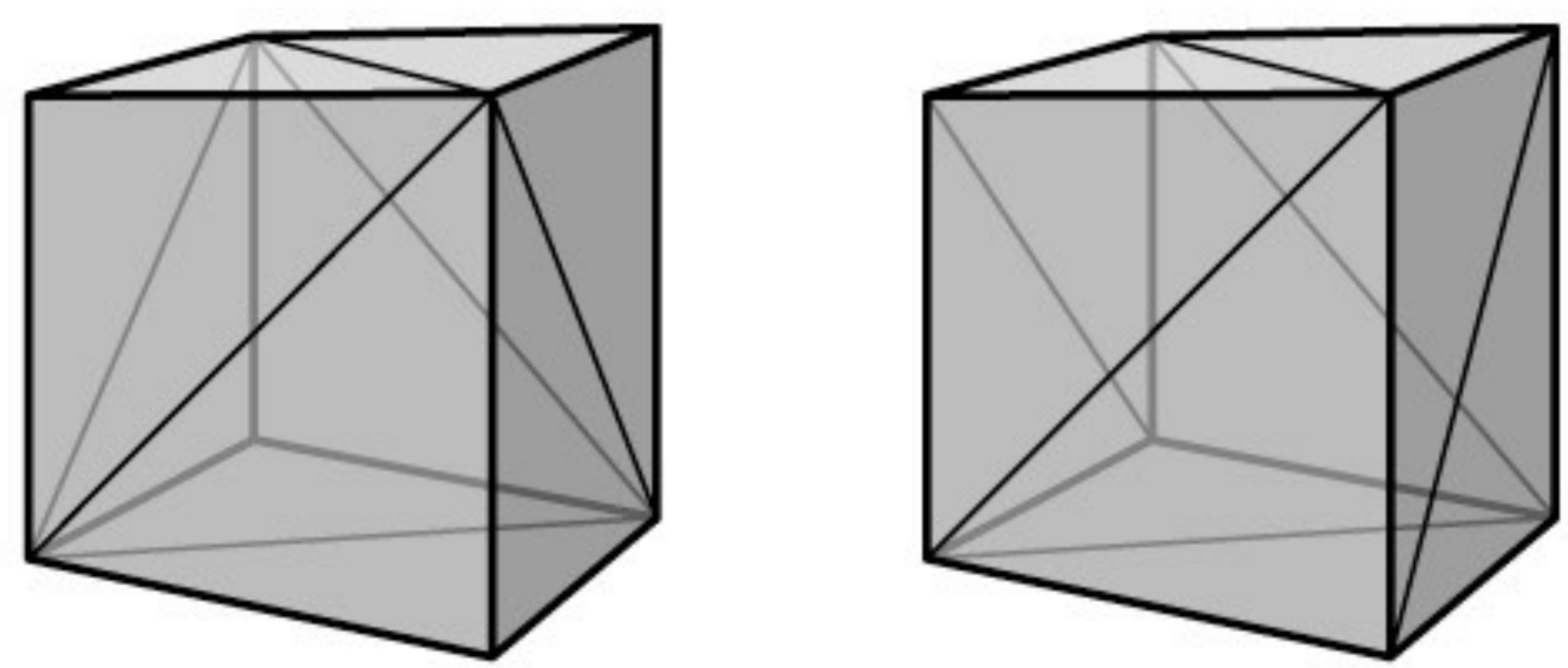
Points + Triangles

+ Sharing vertices reduces memory usage

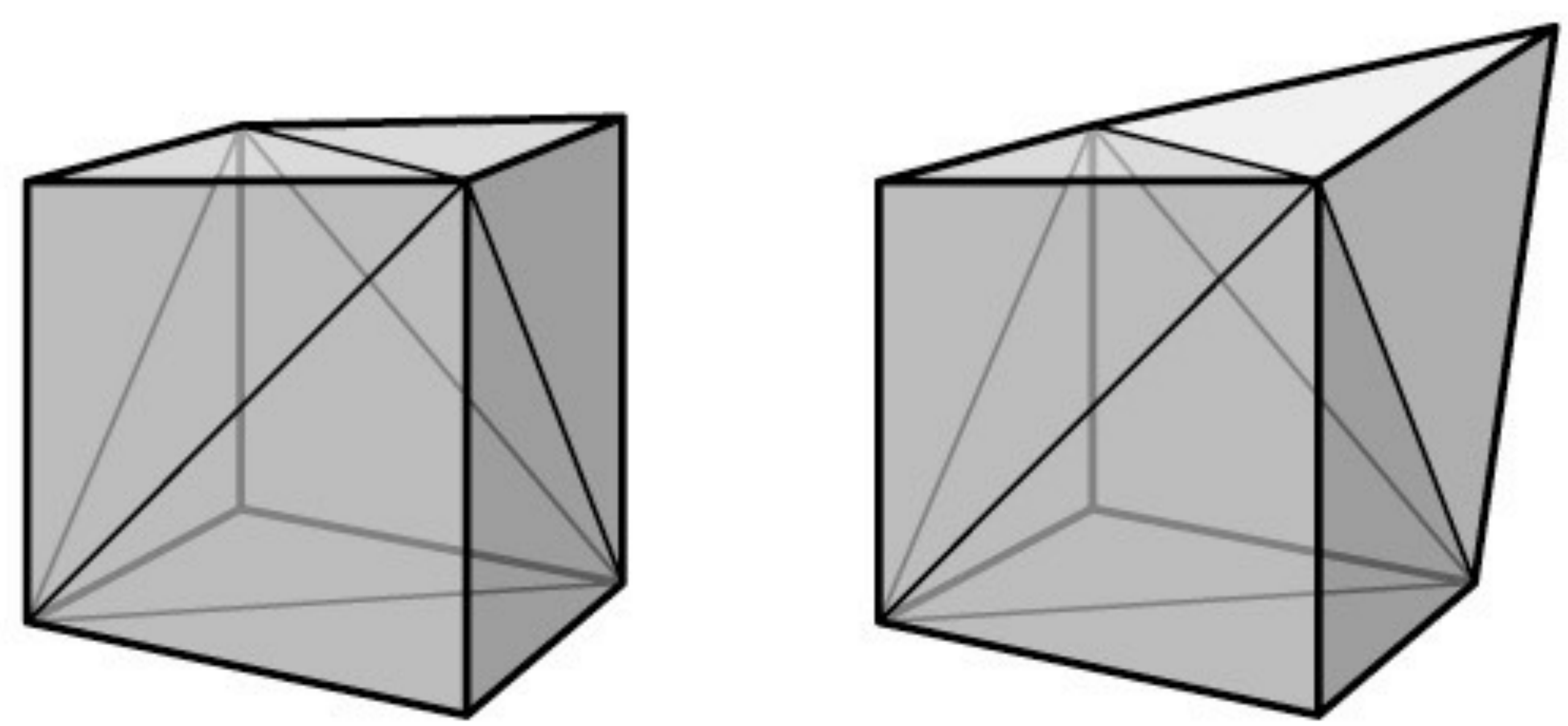
+ Ensure integrity of the mesh (moving a vertex causes that vertex in all the polygons to move)

Topology vs Geometry

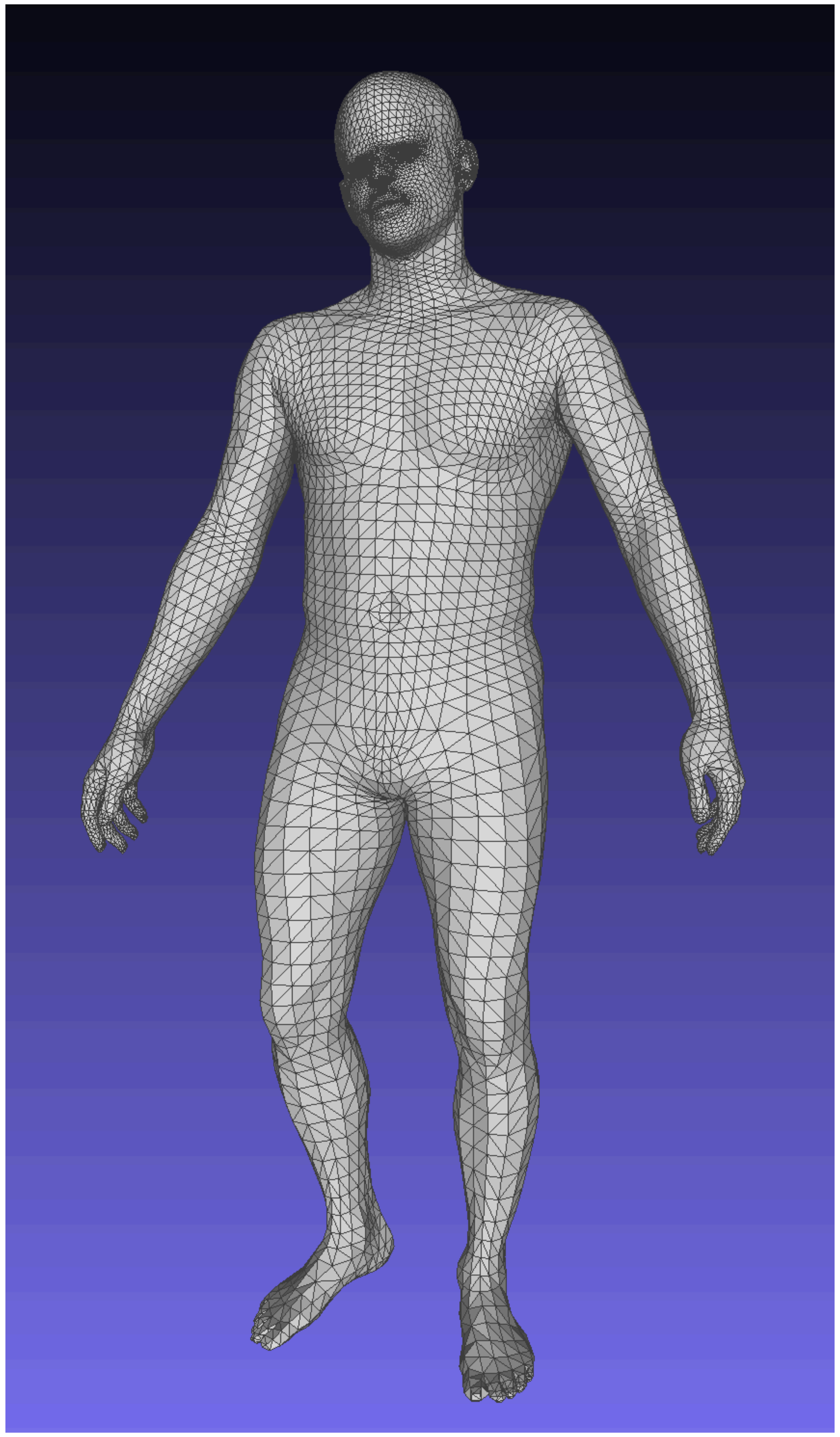
Same geometry, different mesh topology

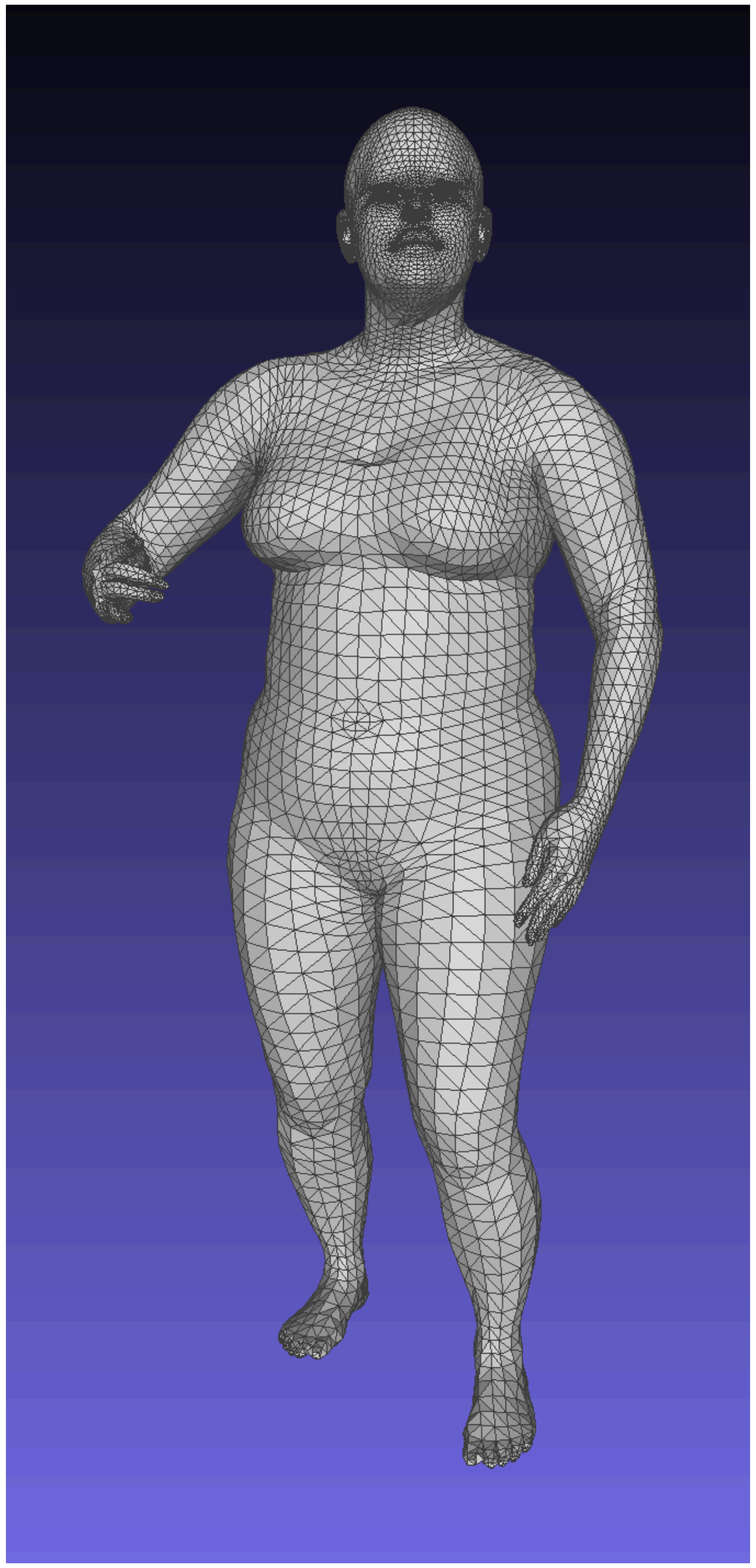


Same mesh topology, different geometry



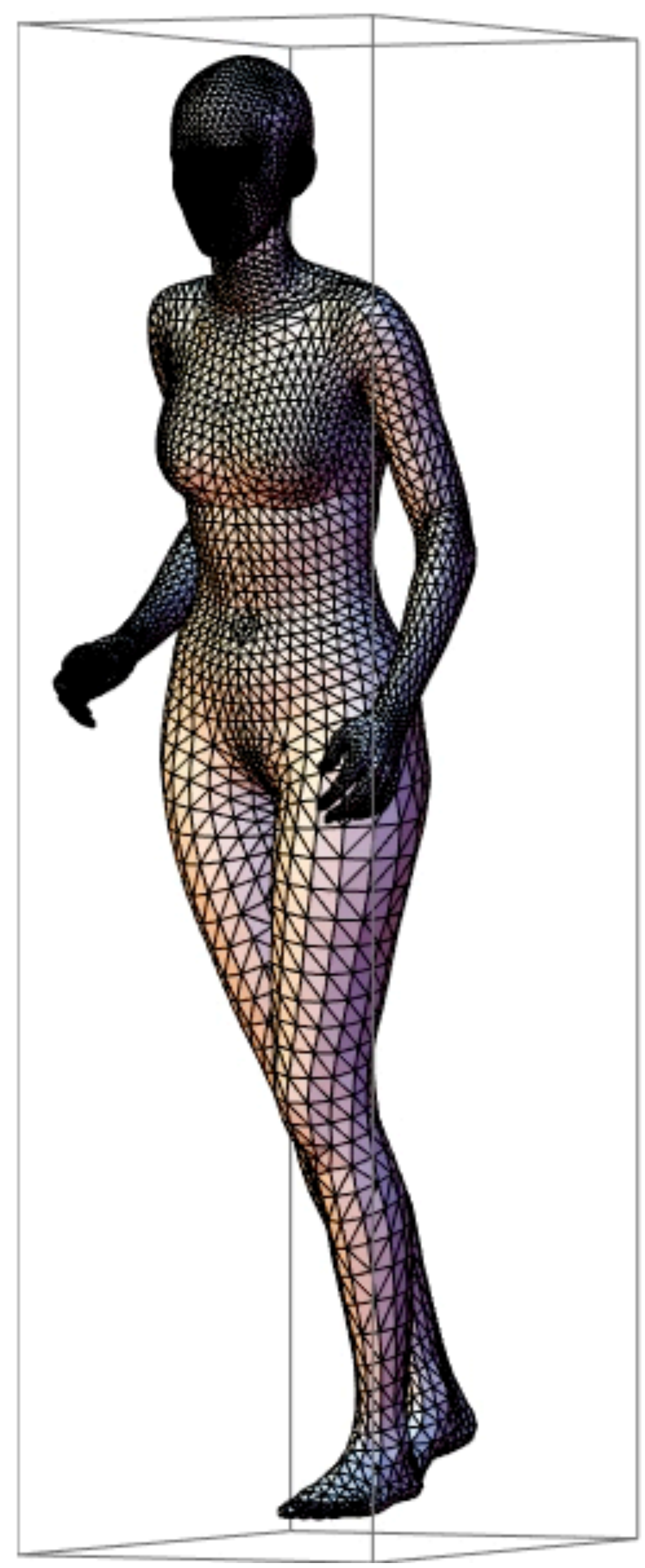
Topology vs Geometry





Same topology / Different Geometry

Topology vs Geometry

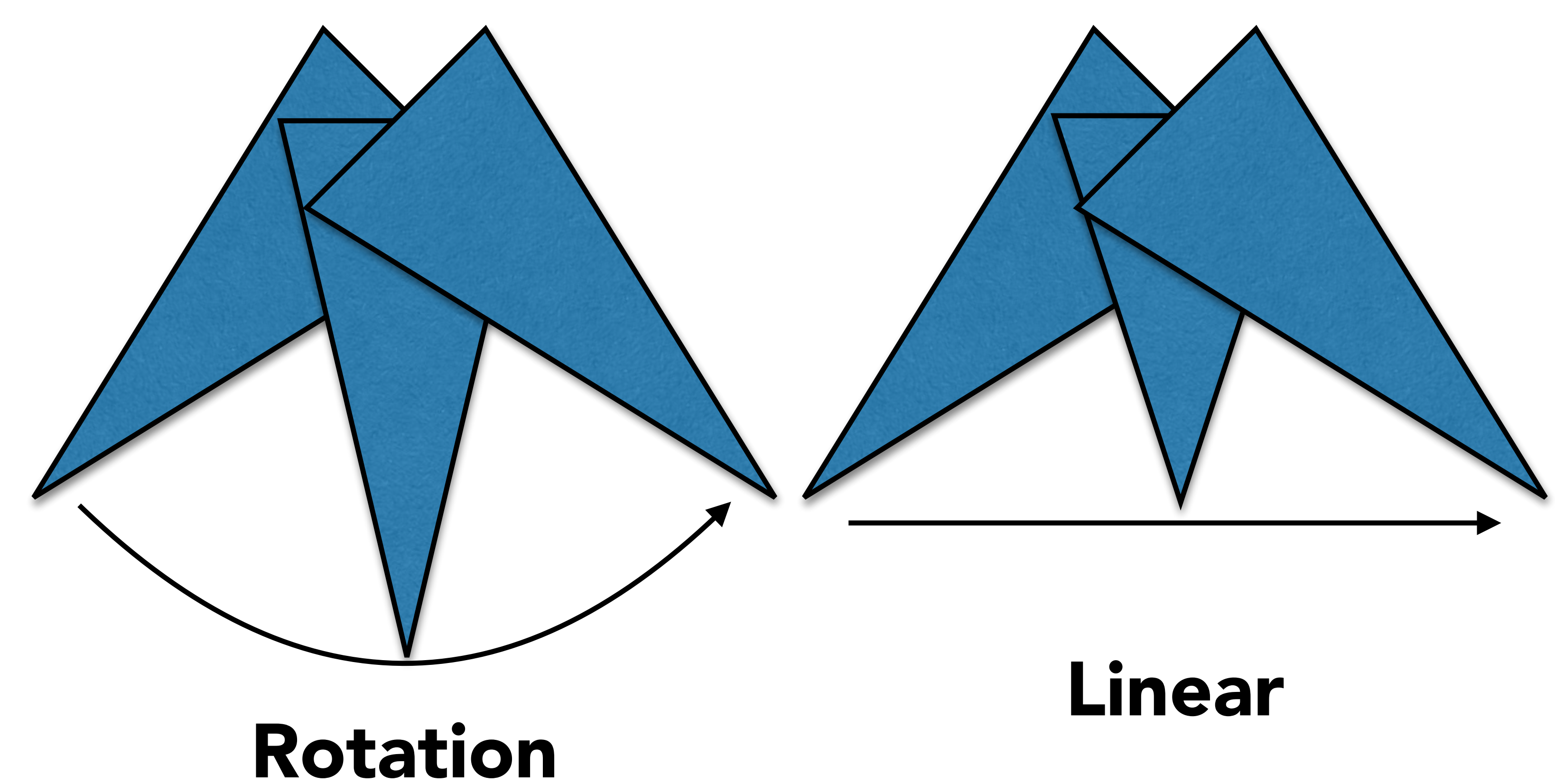


Same topology / Different Geometry

Meshes with same topology allow easy interpolation.

$$V_{\textbf{new}} = \alpha V_1 + (1-\alpha) V_2$$

Note that basic linear interpolation may not be semantically correct.



Topological Mesh Information

Applications:

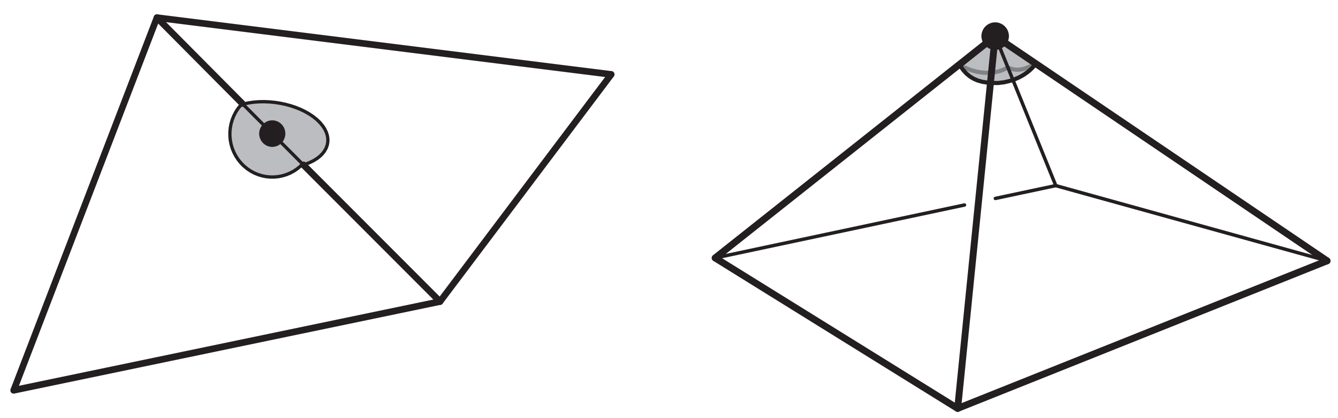
- Constant time access to neighbors
  e.g. surface normal calculation, subdivision
- Editing the geometry
  e.g. adding/removing vertices, faces, edges, etc.

Solution: Topological data structures

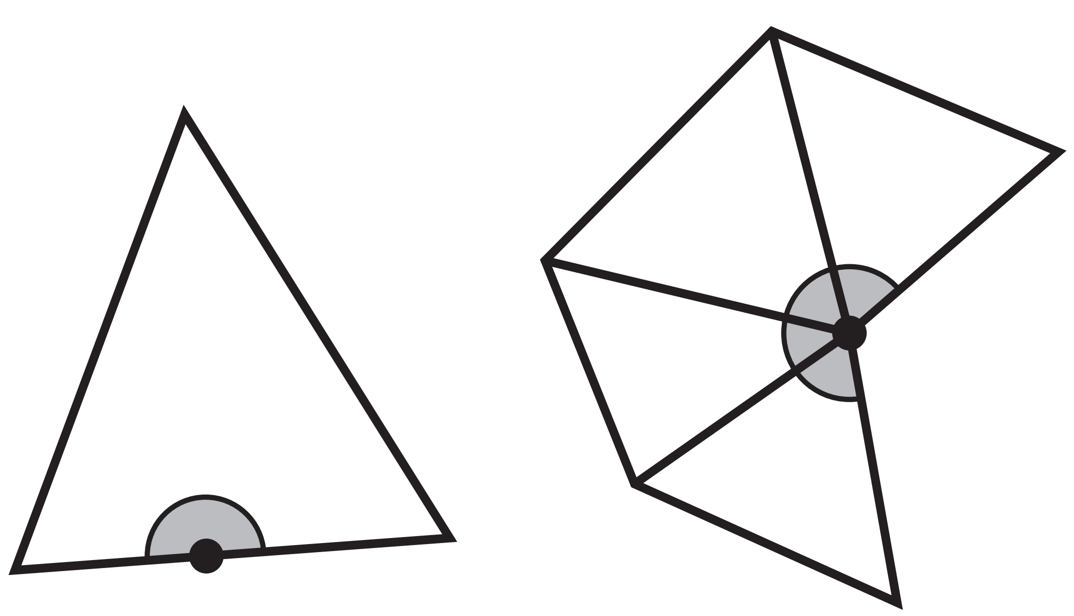
Topological Validity: Manifold

Definition: a 2D manifold is a surface that when cut with a small sphere always yields a disk.

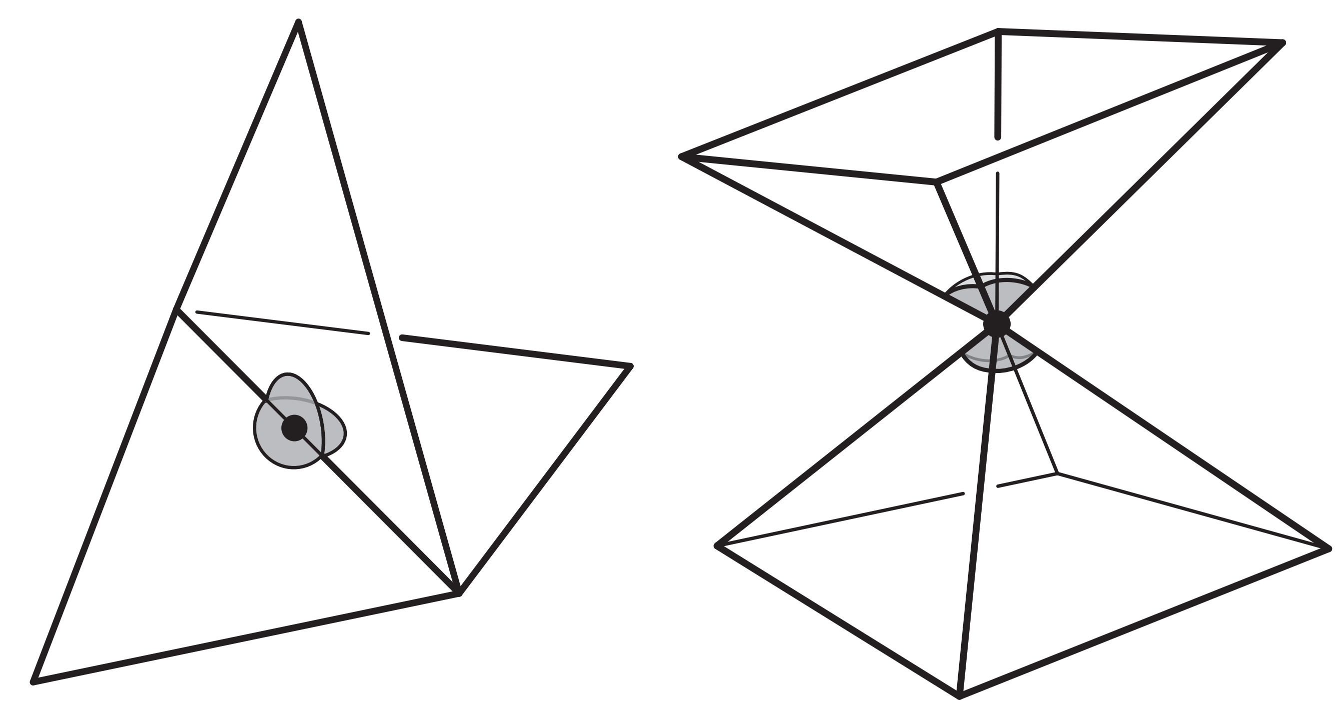
Manifold



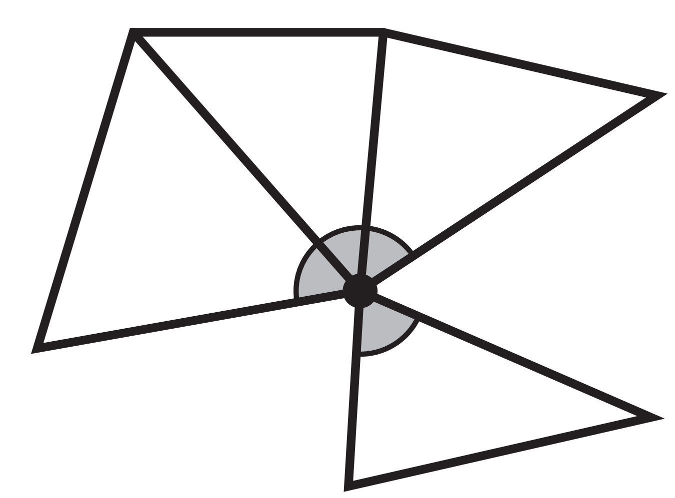
With border



Not manifold



With border



Topological Validity: Manifold

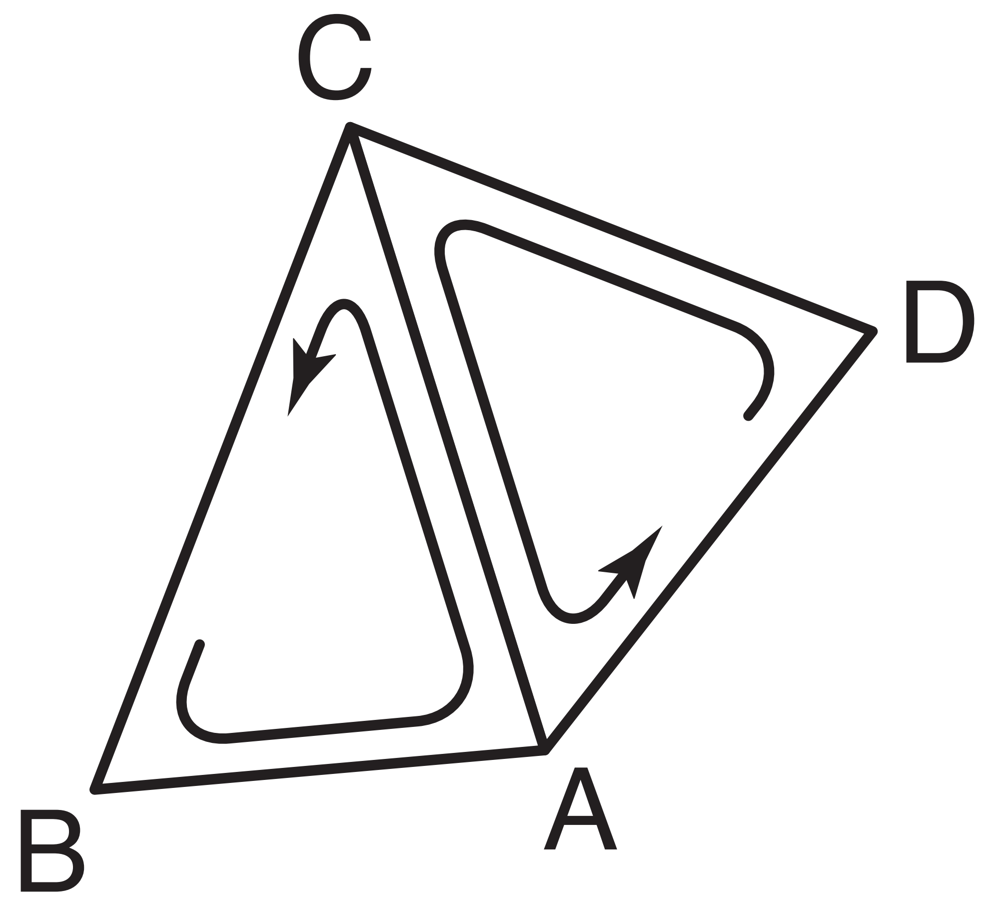
Definition: a 2D manifold is a surface that when cut with a small sphere always yields a “disk”.

If a mesh is manifold we can rely on these useful properties:

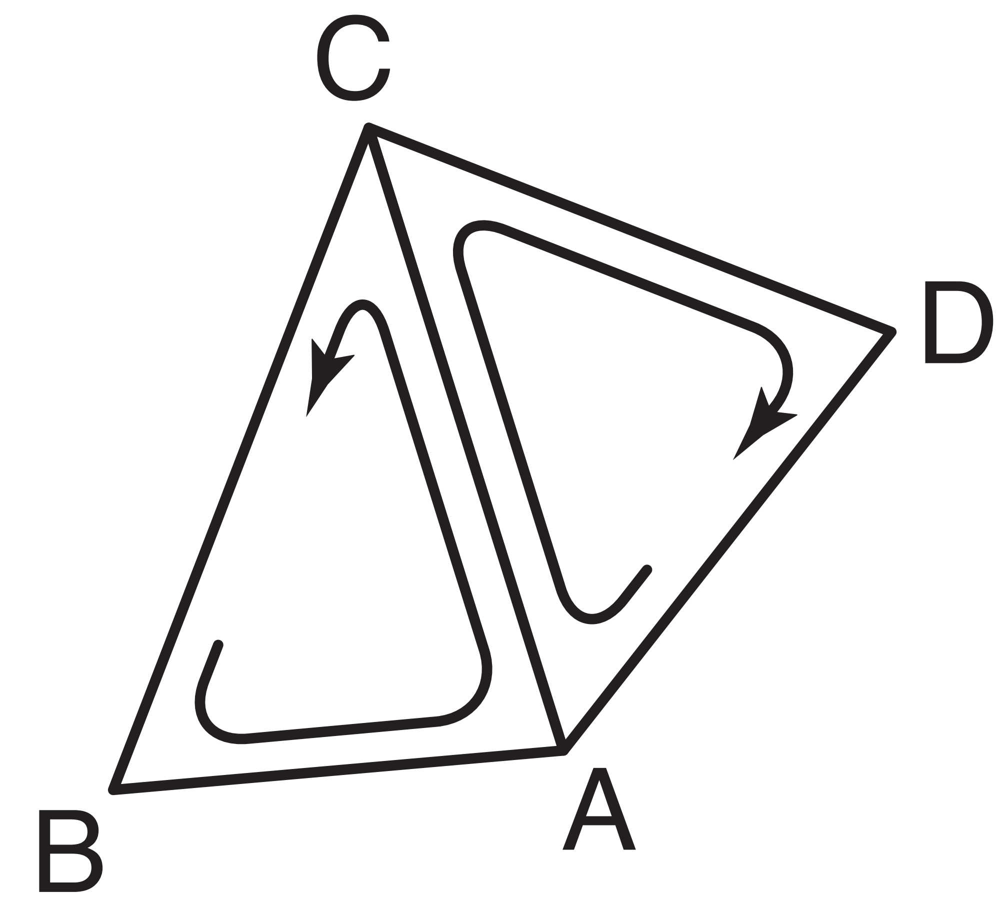
- An edge connects exactly two faces
- An edge connects exactly two vertices
- A face consists of a ring of edges and vertices
- A vertex consists of a ring of edges and faces
- Euler’s polyhedron formula holds: #f – #e + #v = 2
  (for a surface topologically equivalent to a sphere)
  (Check for a cube: 6 – 12 + 8 = 2)

Topological Validity: Orientation Consistency

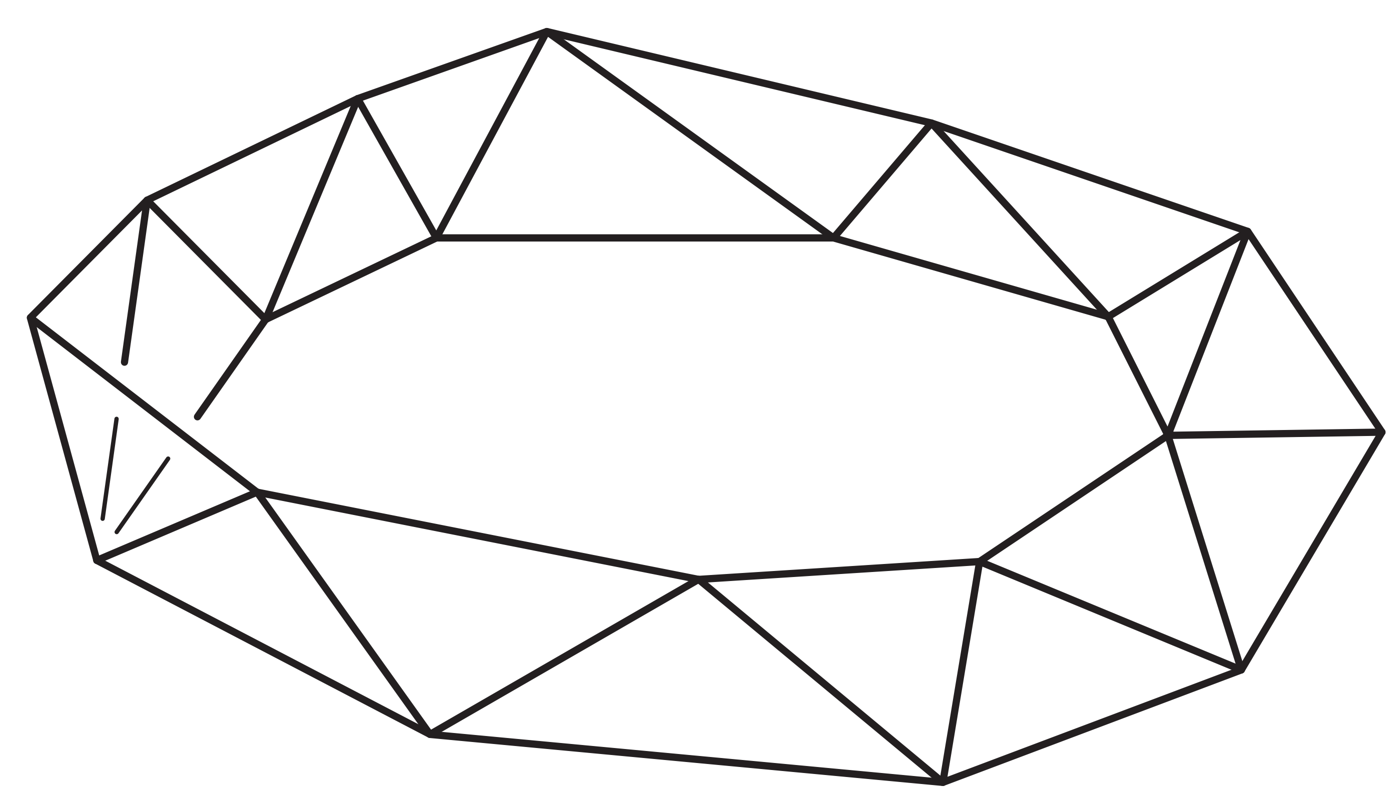
Both facing front



Inconsistent orientations



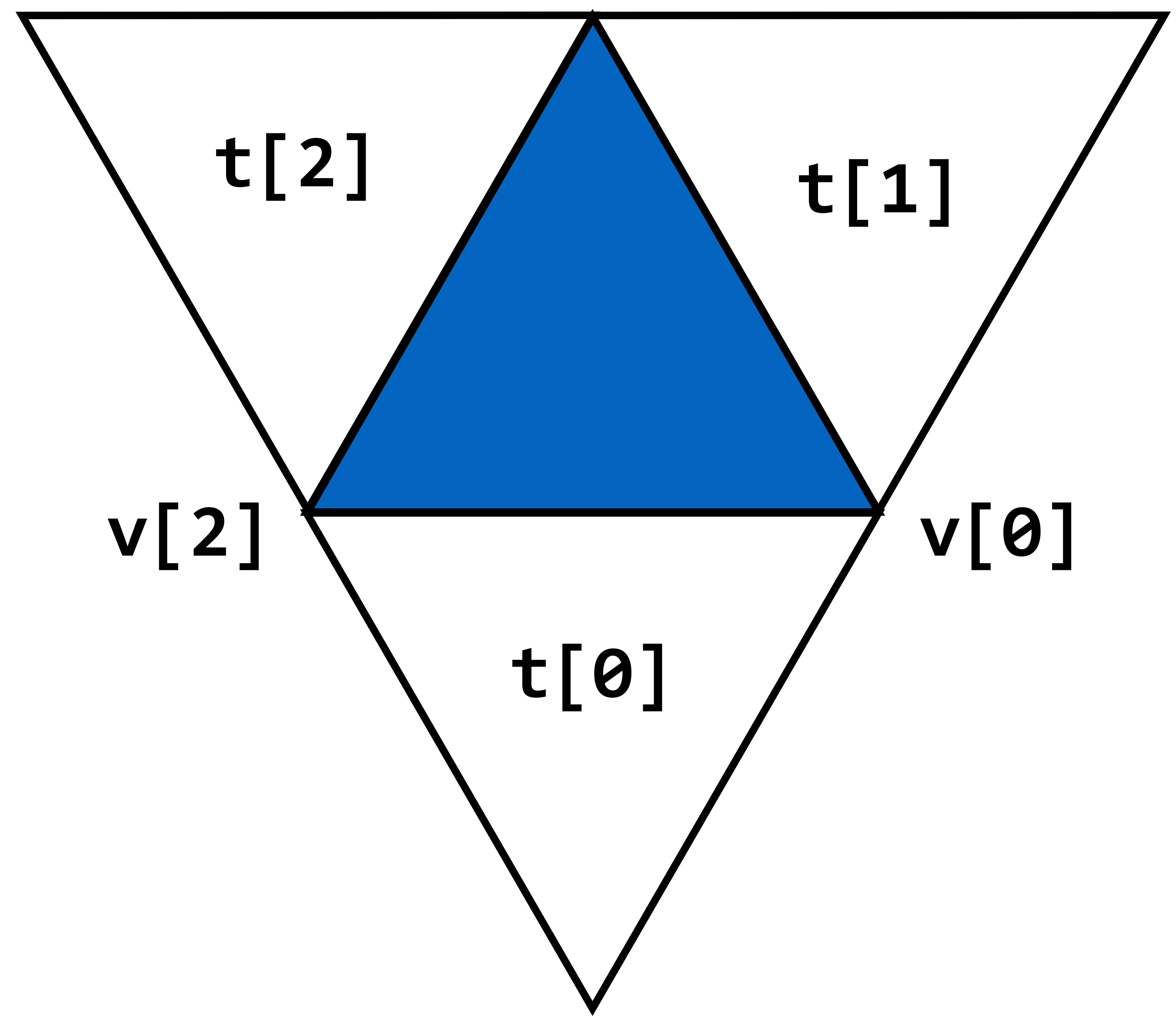
Non-orientable



Mesh Data Structures

Triangle-Neighbor Data Structure

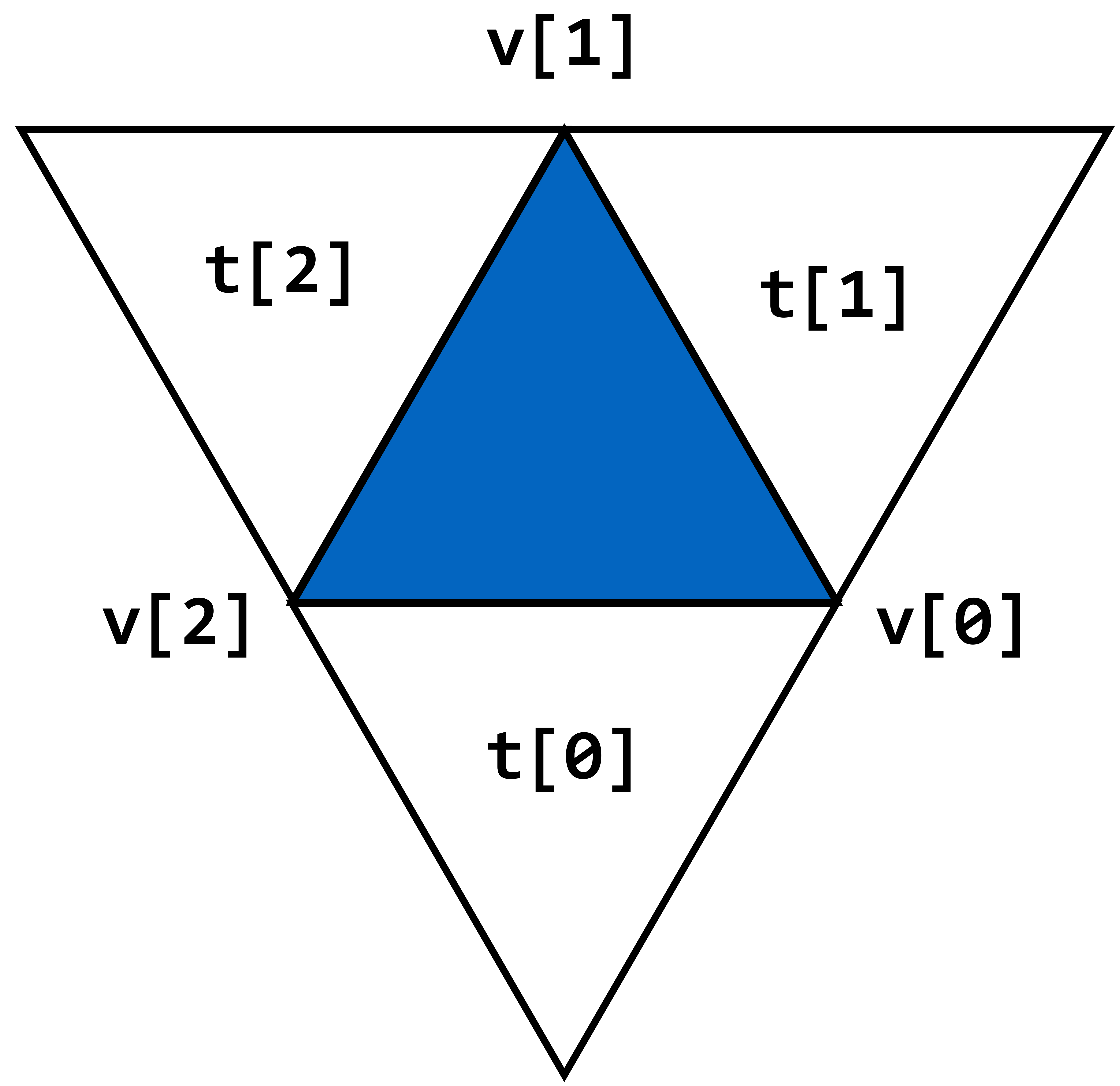
```
struct Tri {
   Vert   * v[3];
   Tri    * t[3];
}

struct Vert {
   Point  pt;
   Tri    *t;
}
```



Triangle-Neighbor – Mesh Traversal

Find next triangle counter-clockwise around vertex v from triangle t

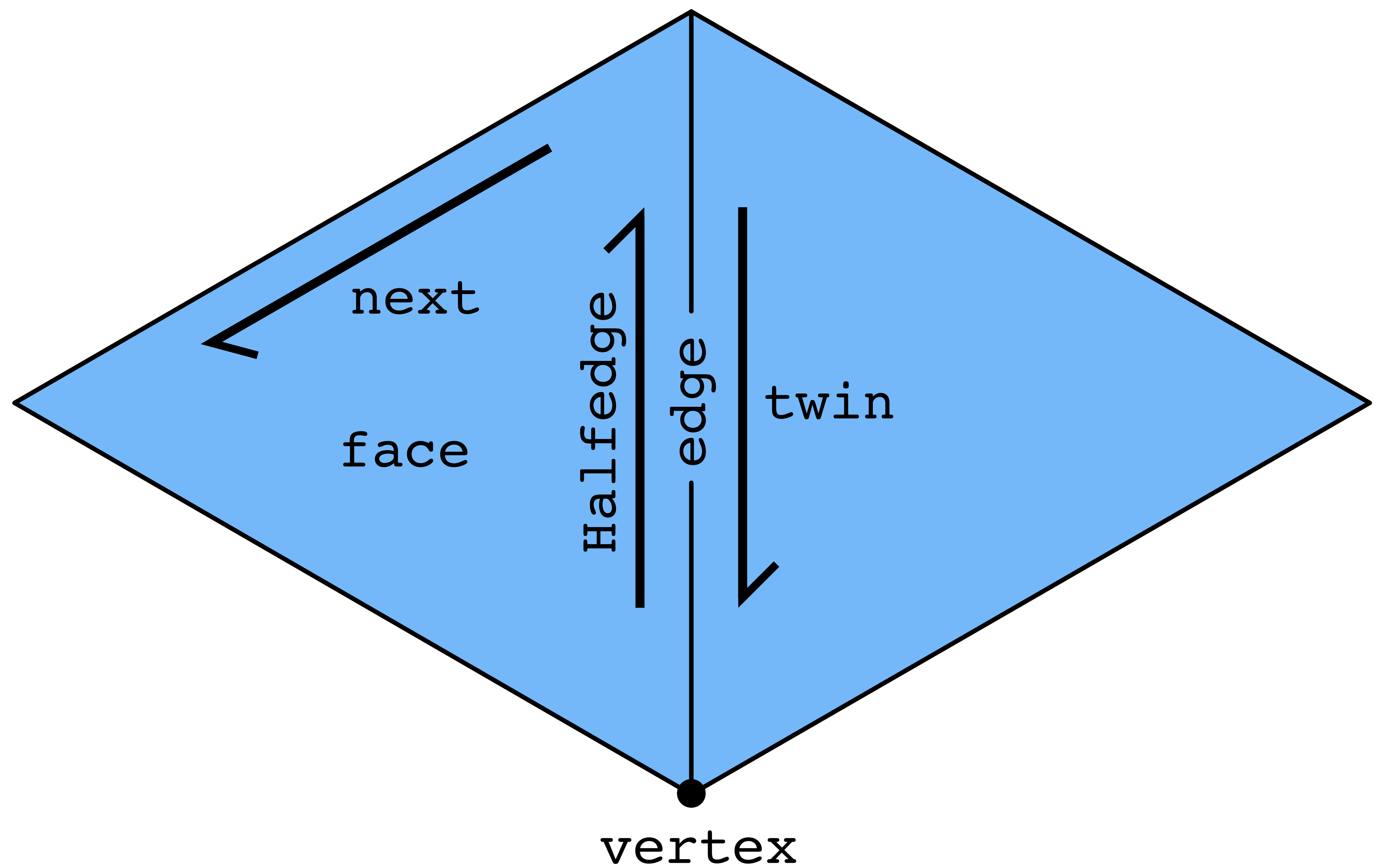
```
Tri *tccwvt(Vert *v, Tri *t)
{
   if (v == t->v[0])
      return t->t[0];
   if (v == t->v[1])
      return t->t[1];
   if (v == t->v[2])
      return t->t[2];
}
```



Half-Edge Data Structure

```
struct Halfedge {
    Halfedge *twin,
    Halfedge *next;
    Vertex *vertex;
    Edge *edge;
    Face *face;
}
struct Vertex {
    Point pt;
    Halfedge *halfedge;
}
struct Edge {
    Halfedge *halfedge;
}
struct Face {
    Halfedge *halfedge;
}
```

Key idea: two half-edges act as "glue" between mesh elements



Each vertex, edge and face points to one of its half edges

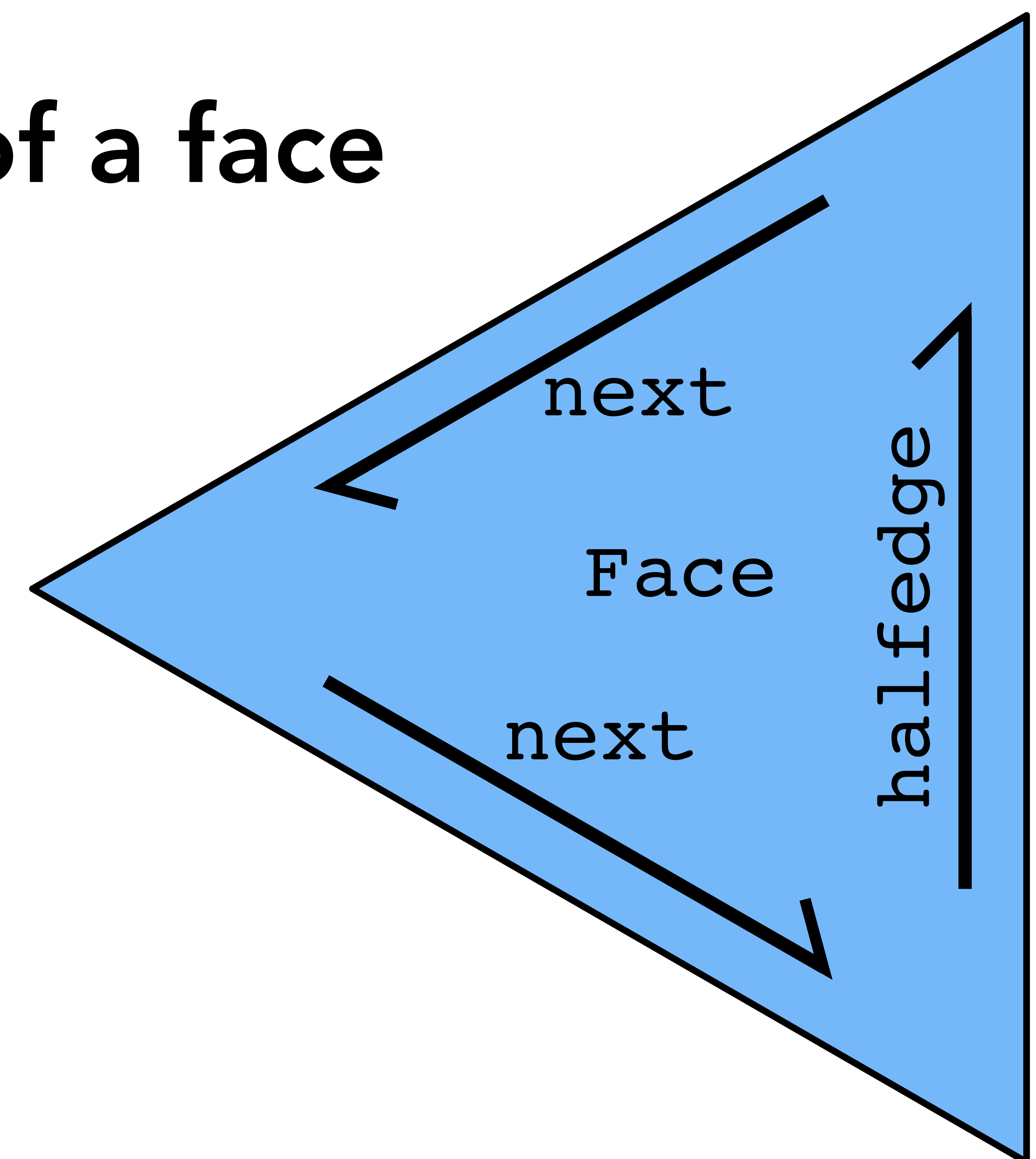
Half-Edge Facilitates Mesh Traversal

Use twin and next pointers to move around mesh

Process vertex, edge and/or face pointers

Example 1: process all vertices of a face

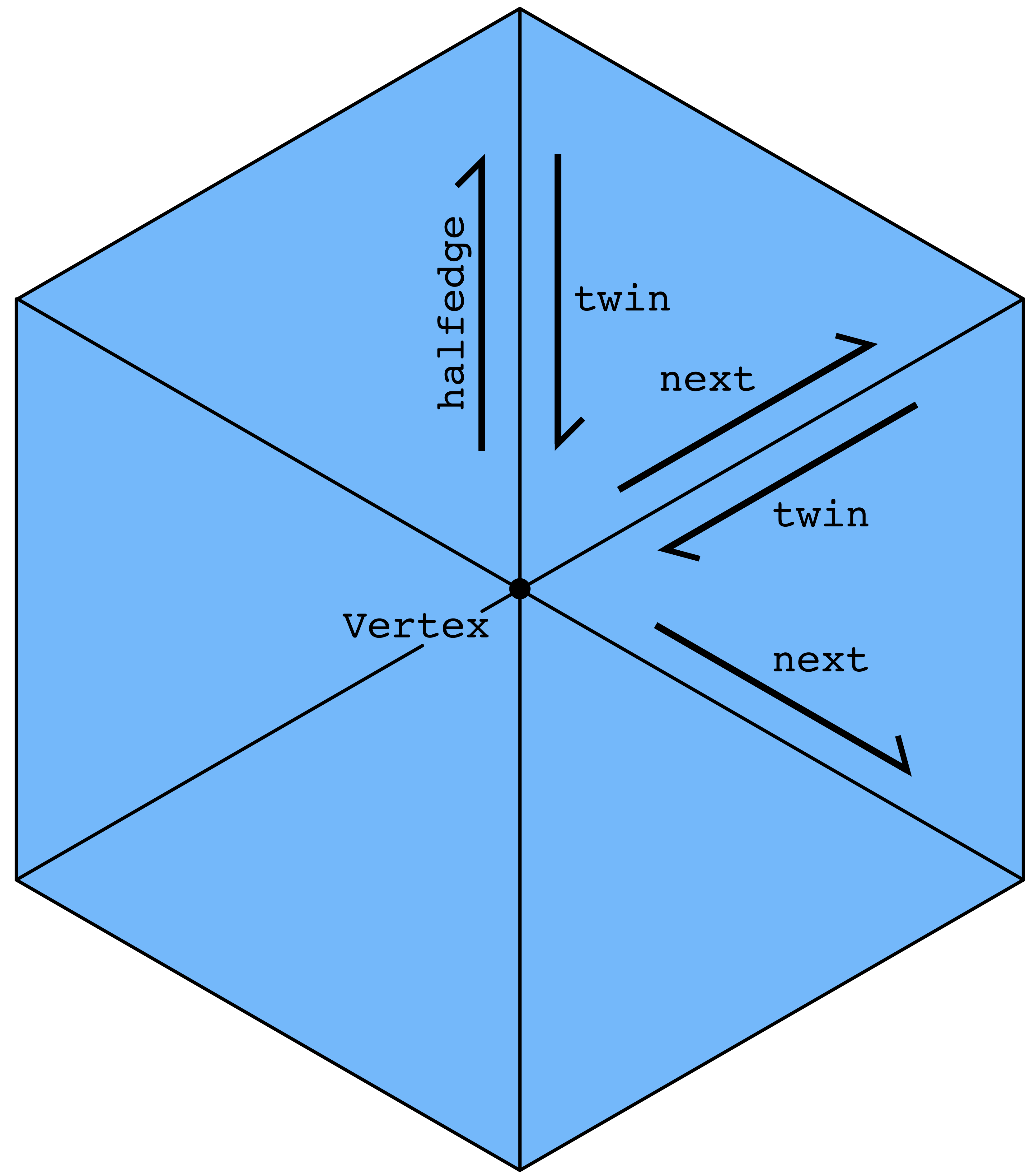
```
Halfedge* h = f->halfedge;
do {
   process(h->vertex);
   h = h->next;
}
while( h != f->halfedge );
```



Half-Edge Facilitates Mesh Traversal

Example 2: process all edges around a vertex

```
Halfedge* h = v->halfedge;
do {
   process(h->edge);
   h = h->twin->next;
}
while( h != v->halfedge );
```

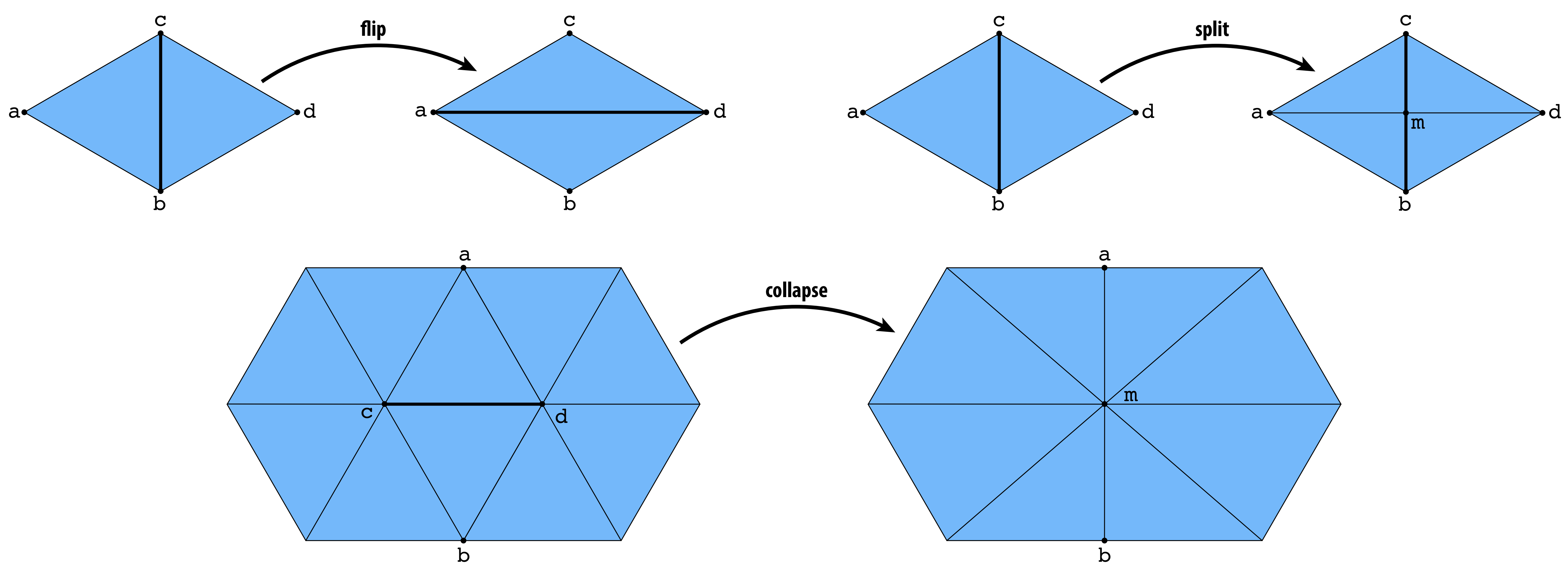


Local Mesh Operations

Half-Edge – Local Mesh Editing

Basic operations for linked list: insert, delete

Basic ops for half-edge mesh: <u>flip</u>, <u>split</u>, <u>collapse</u> edges

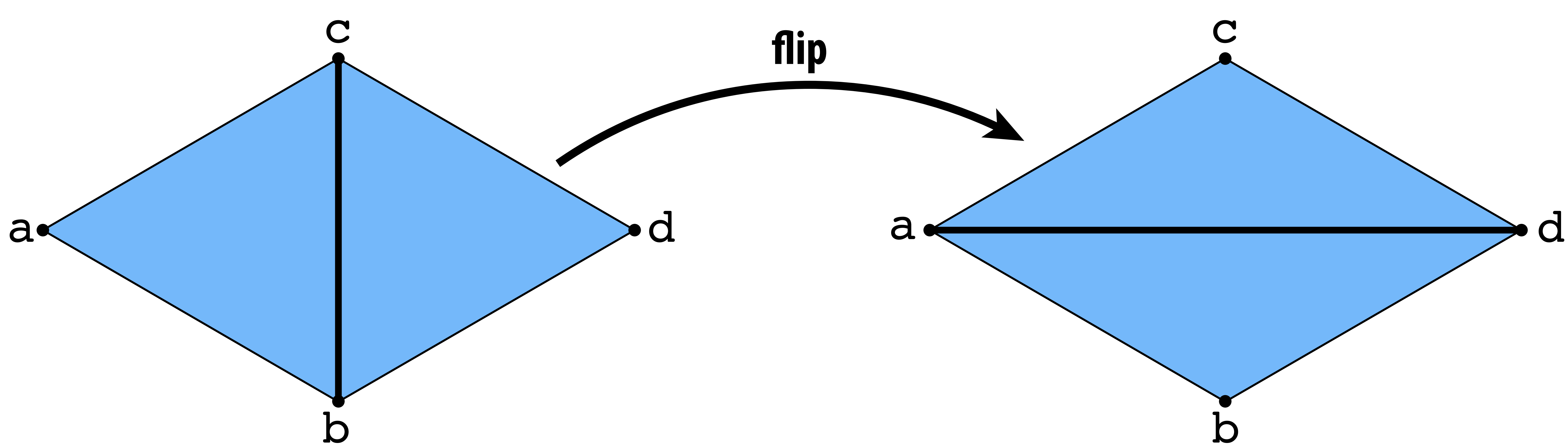


Allocate / delete elements; reassign pointers

(Care needed to preserve mesh manifold property)

Half-Edge – Edge Flip

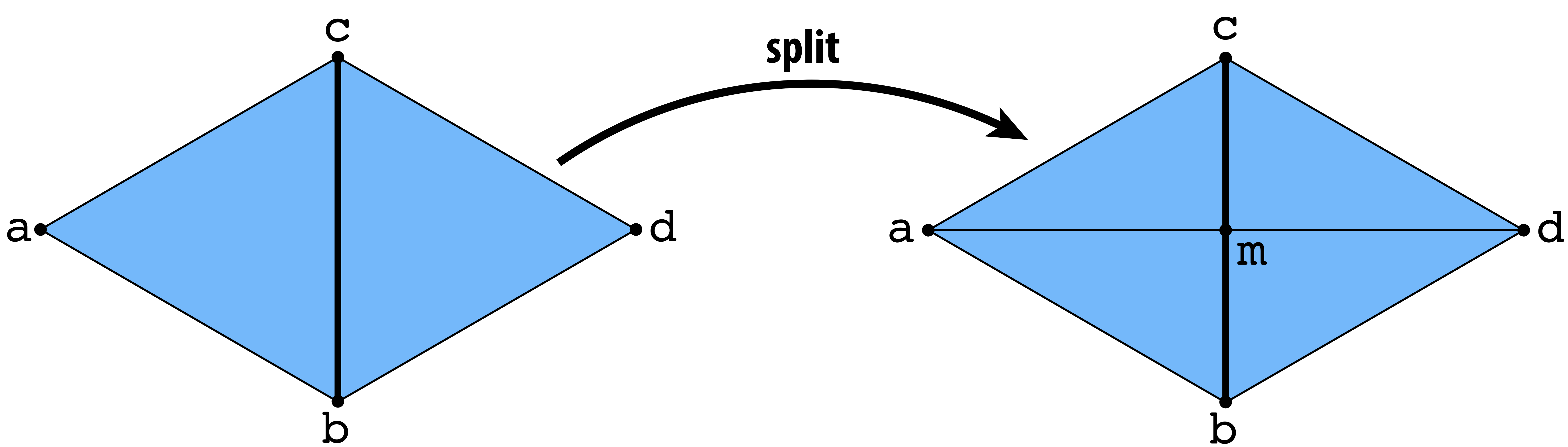
- Triangles (a,b,c), (b,d,c) become (a,d,c), (a,b,d):



- Long list of pointer reassignments
- However, no elements created/destroyed.

Half-Edge – Edge Split

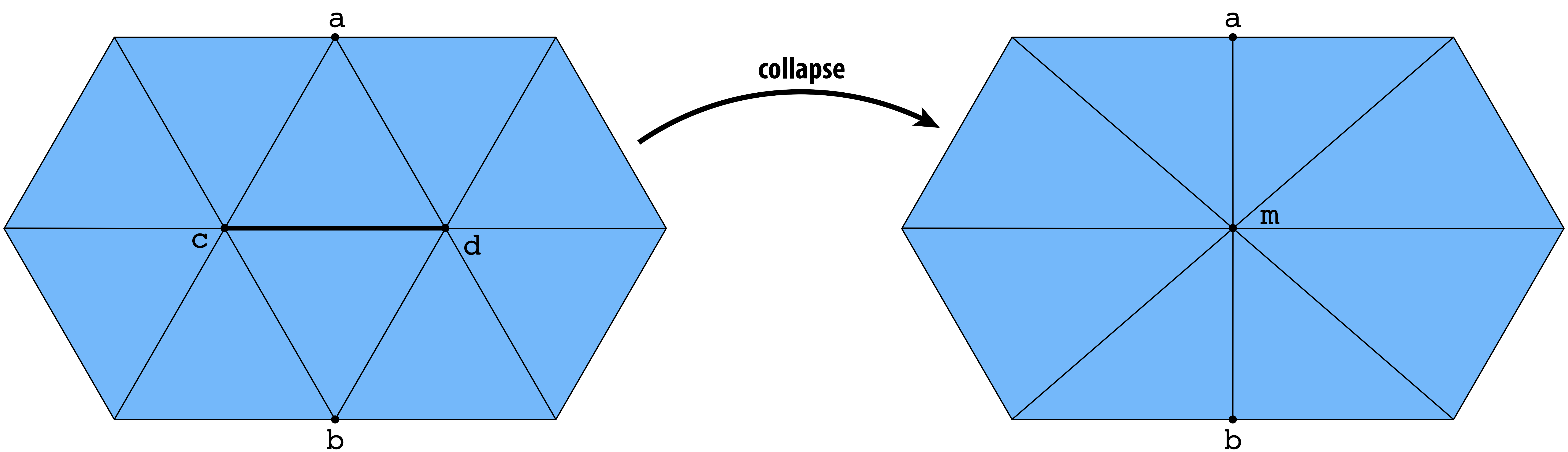
- Insert midpoint m of edge (c,b), connect to get four triangles:



- This time have to add elements
- Again, many pointer reassignments

Half-Edge – Edge Collapse

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

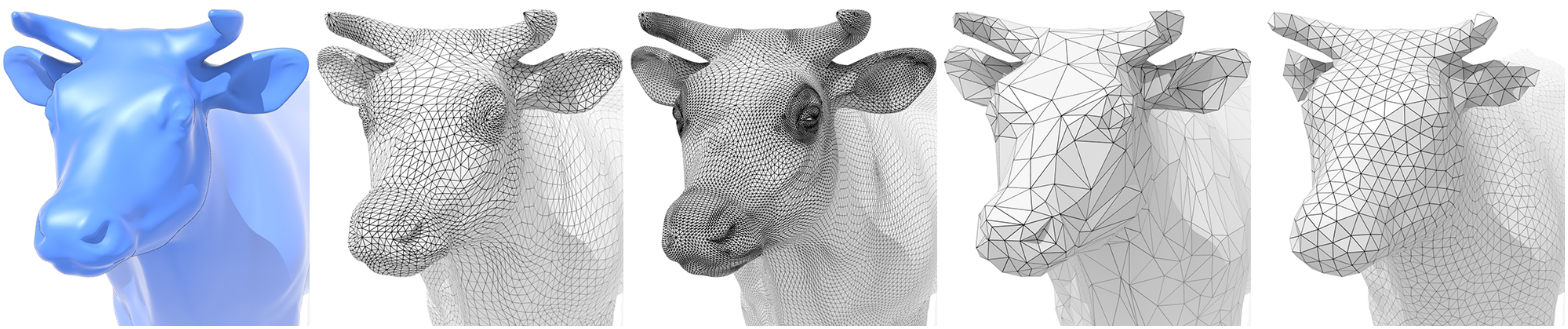


- This time have to delete elements
- Again, many pointer reassignments

Global Mesh Operations

Global Mesh Operations: Geometry Processing

- Mesh subdivision
- Mesh simplification
- Mesh regularization

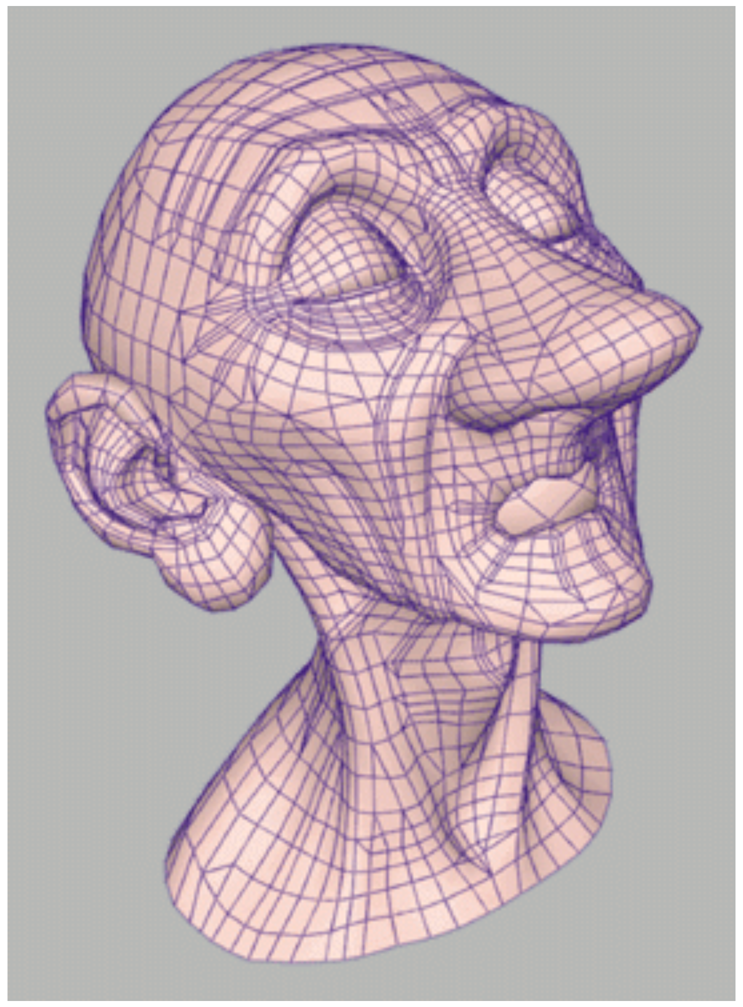


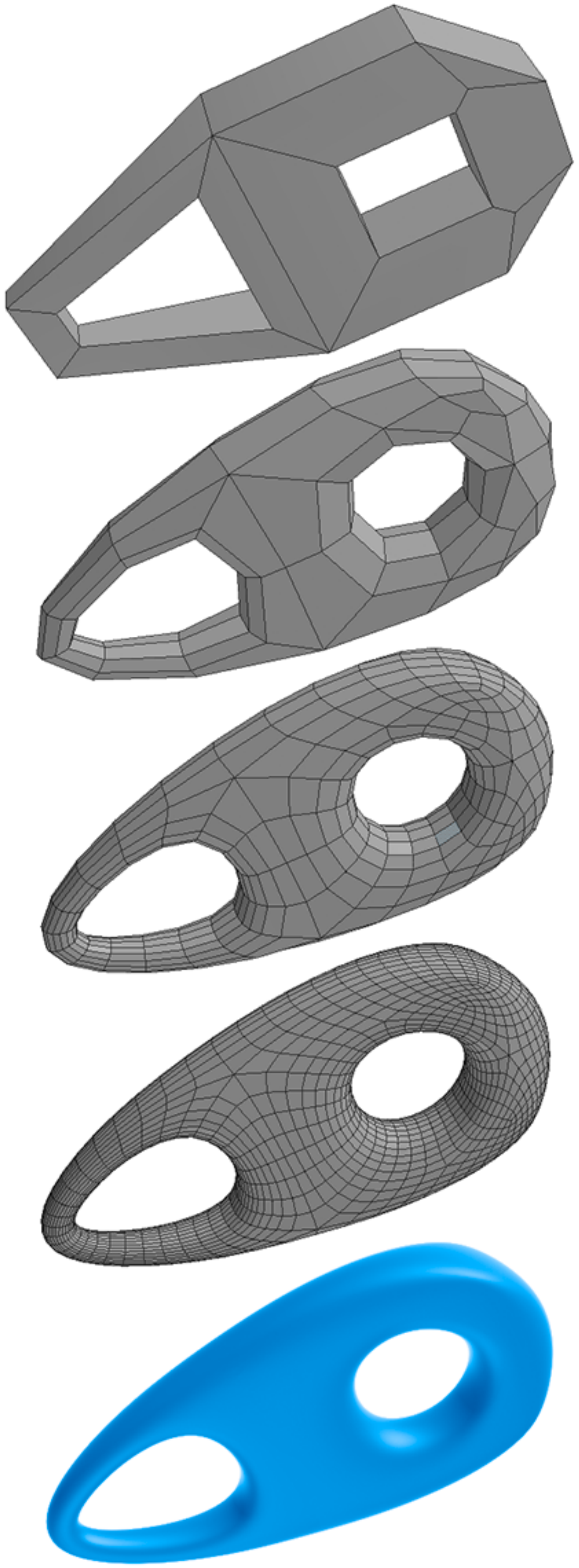
Subdivision Surfaces

Subdivision Surfaces

Start with coarse polygon mesh ("control cage")

- Subdivide each element
- Update vertices via local averaging





Many possible rules:

- Catmull-Clark (quads)
- Loop (triangles)
- ...

Common issues:

- interpolating or approximating?
- continuity at vertices?

Relatively easy for modeling; harder to guarantee continuity

Core Idea: Let Subdivision Define The Surface

In Bezier curves, we saw:

- Evaluation by subdivision (de Casteljau algorithm)
- Or evaluation by algebra (Bernstein polynomials)

Insight that leads to subdivision surfaces:

- Free ourselves from the algebraic evaluation
- Let subdivision fully define the surface

Many possible subdivision rules – different surfaces

- Technical challenge shifts to designing rules and proving properties (e.g. convergence and continuity)
- Applying rules to compute surface is procedural

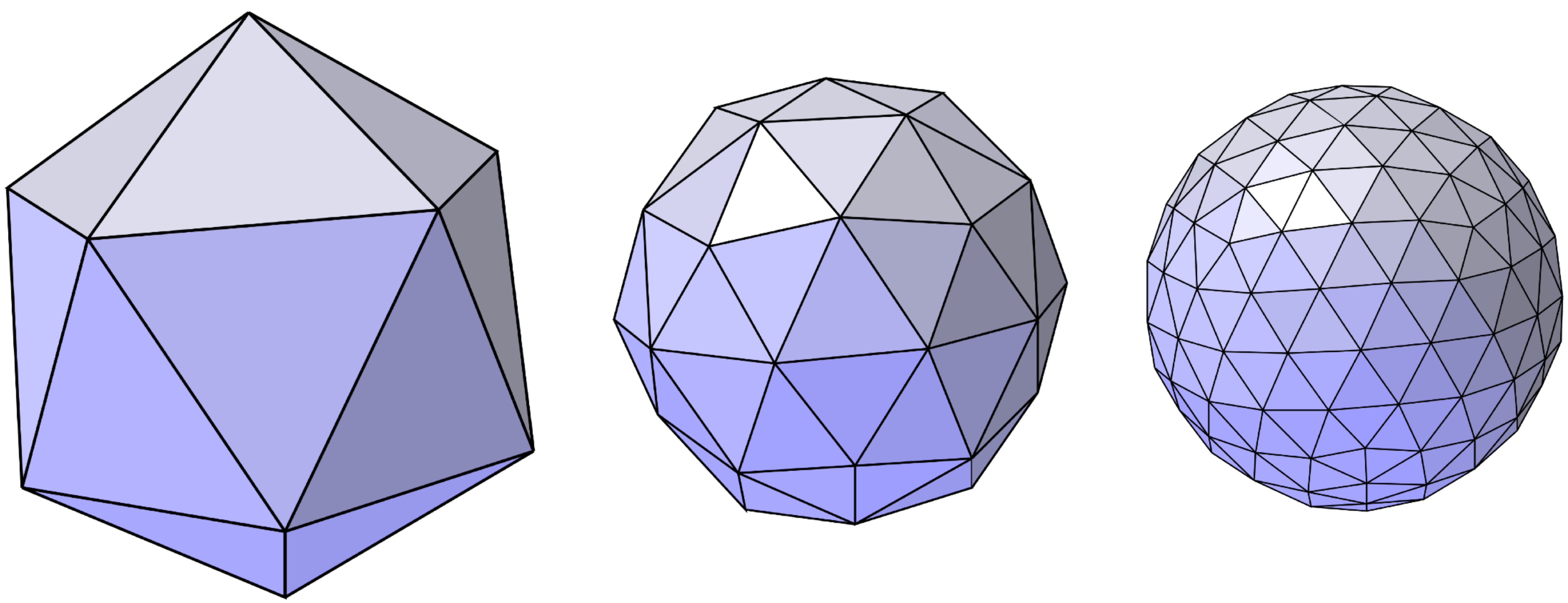
Loop Subdivision

Loop Subdivision

Common subdivision rule for triangle meshes

"C2" smoothness away from irregular vertices

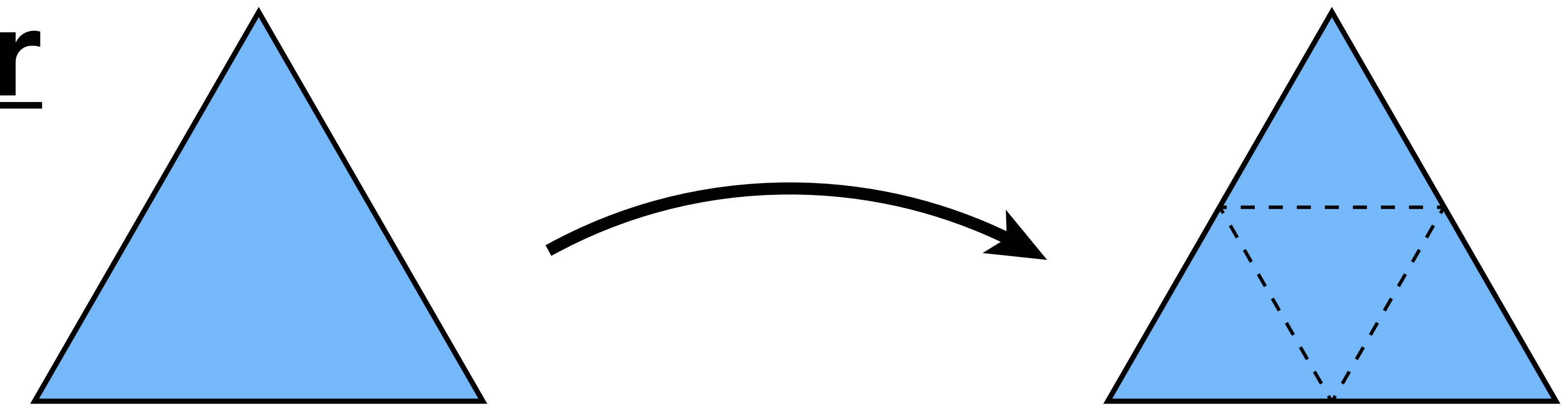
Approximating, not interpolating



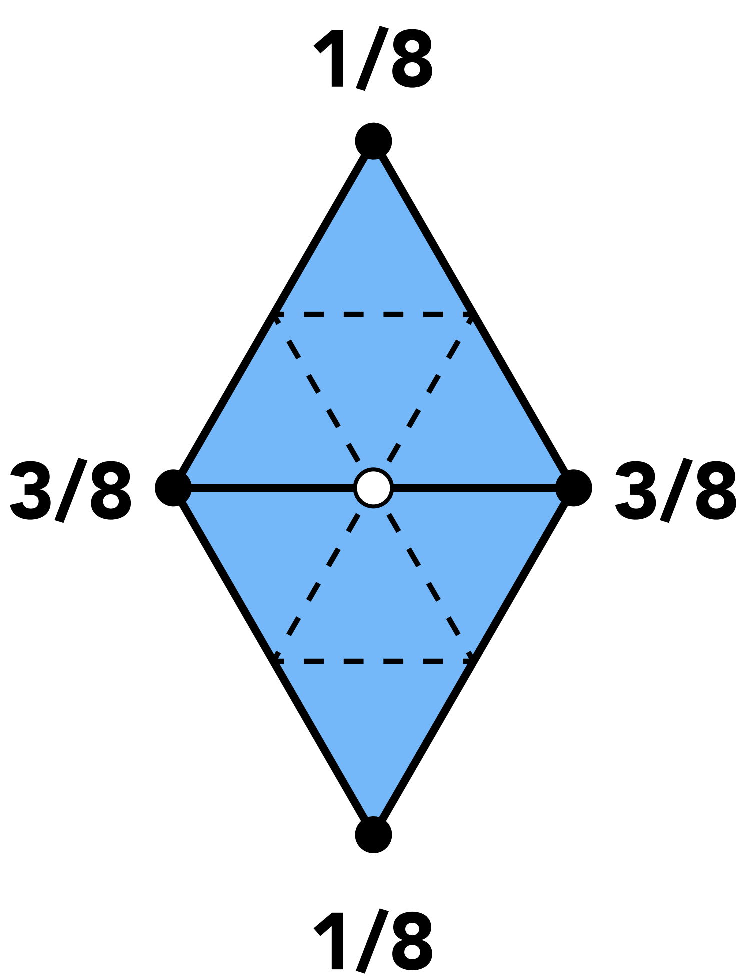
Simon Fuhrman

Loop Subdivision Algorithm

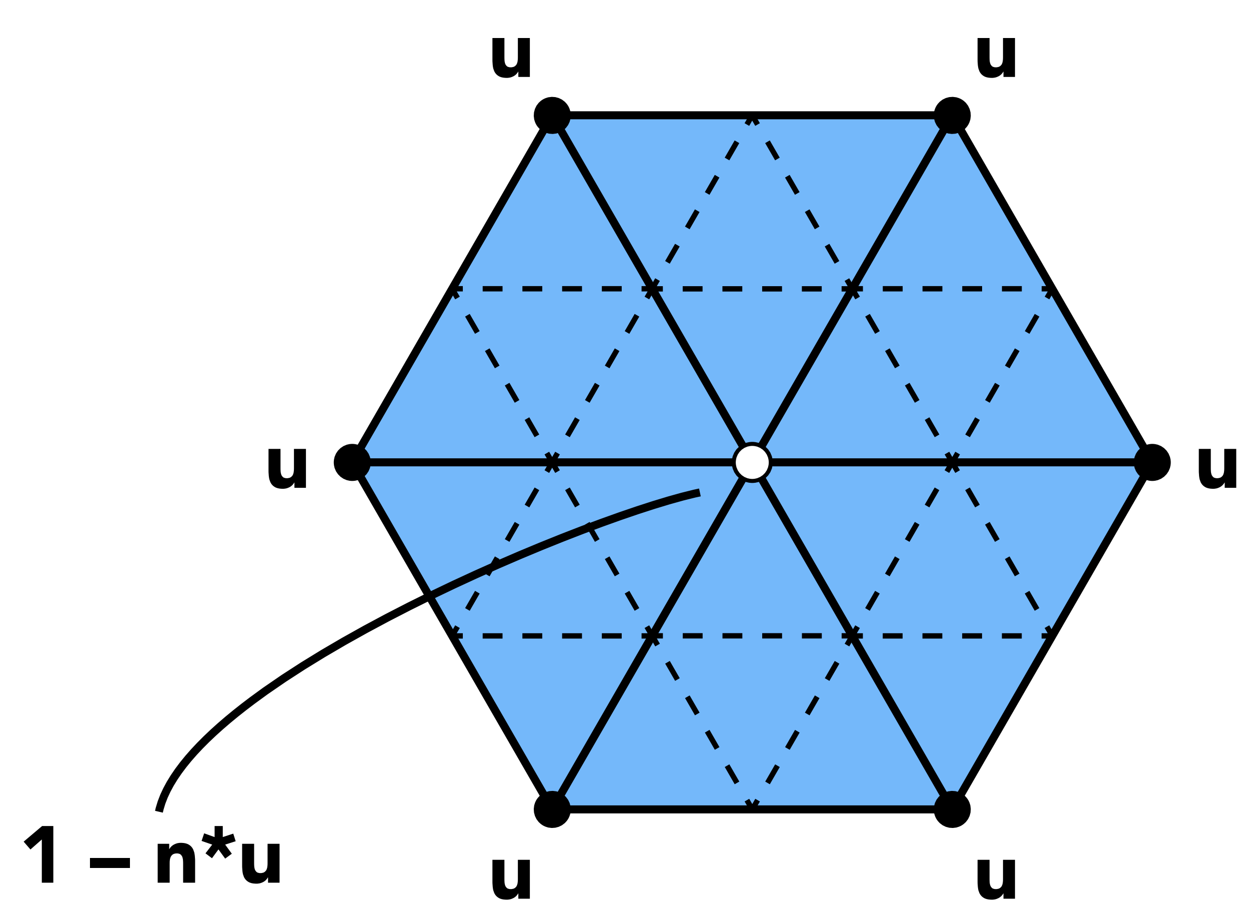
- Split each triangle into four



- Assign new vertex positions according to weights:



New vertices



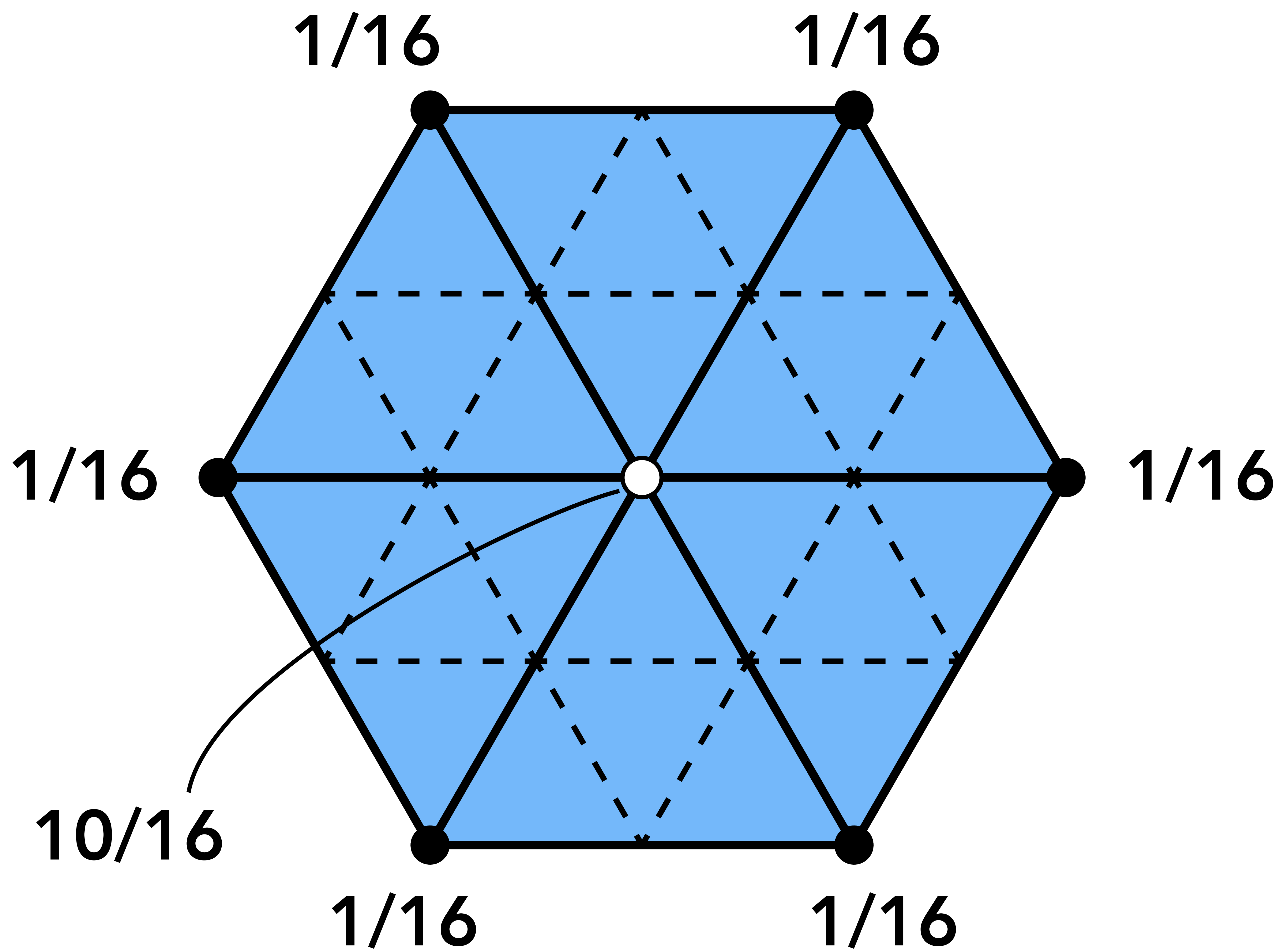
Old vertices

n: vertex degree

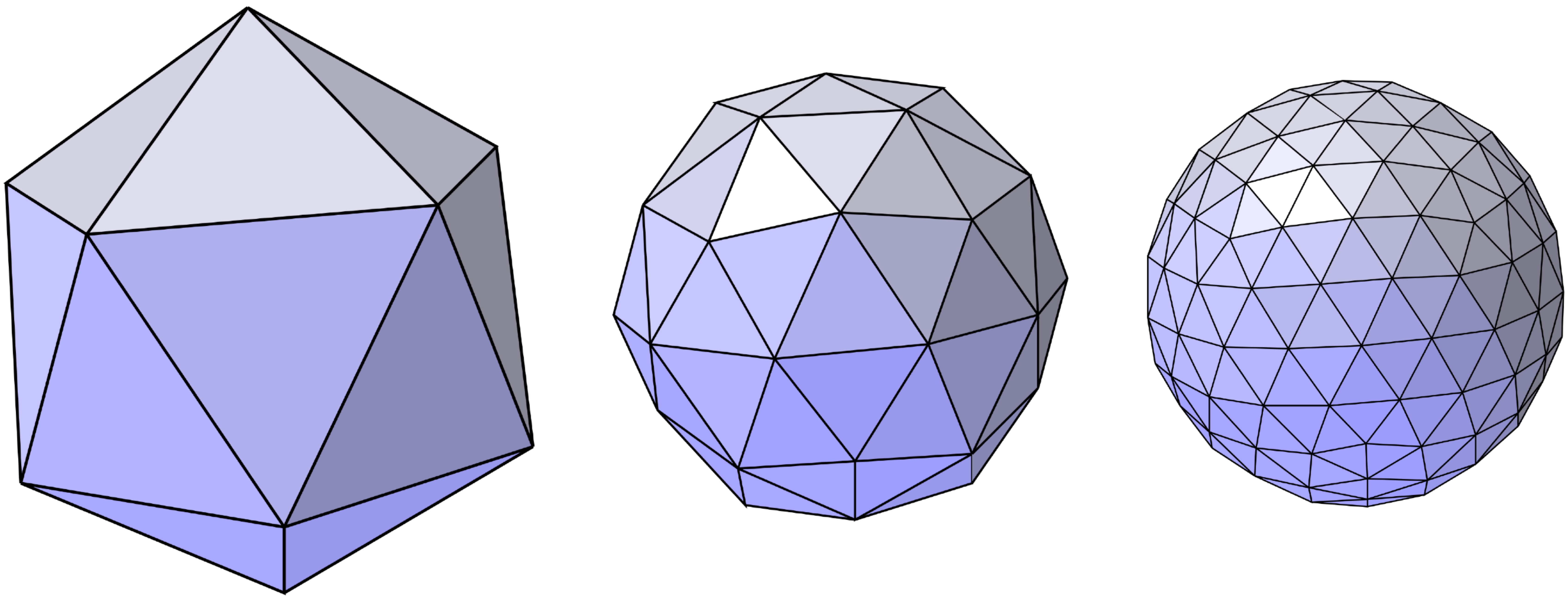
u: 3/16 if n=3, 3/(8n) otherwise

Loop Subdivision Algorithm

Example, for degree 6 vertices



Loop Subdivision Algorithm



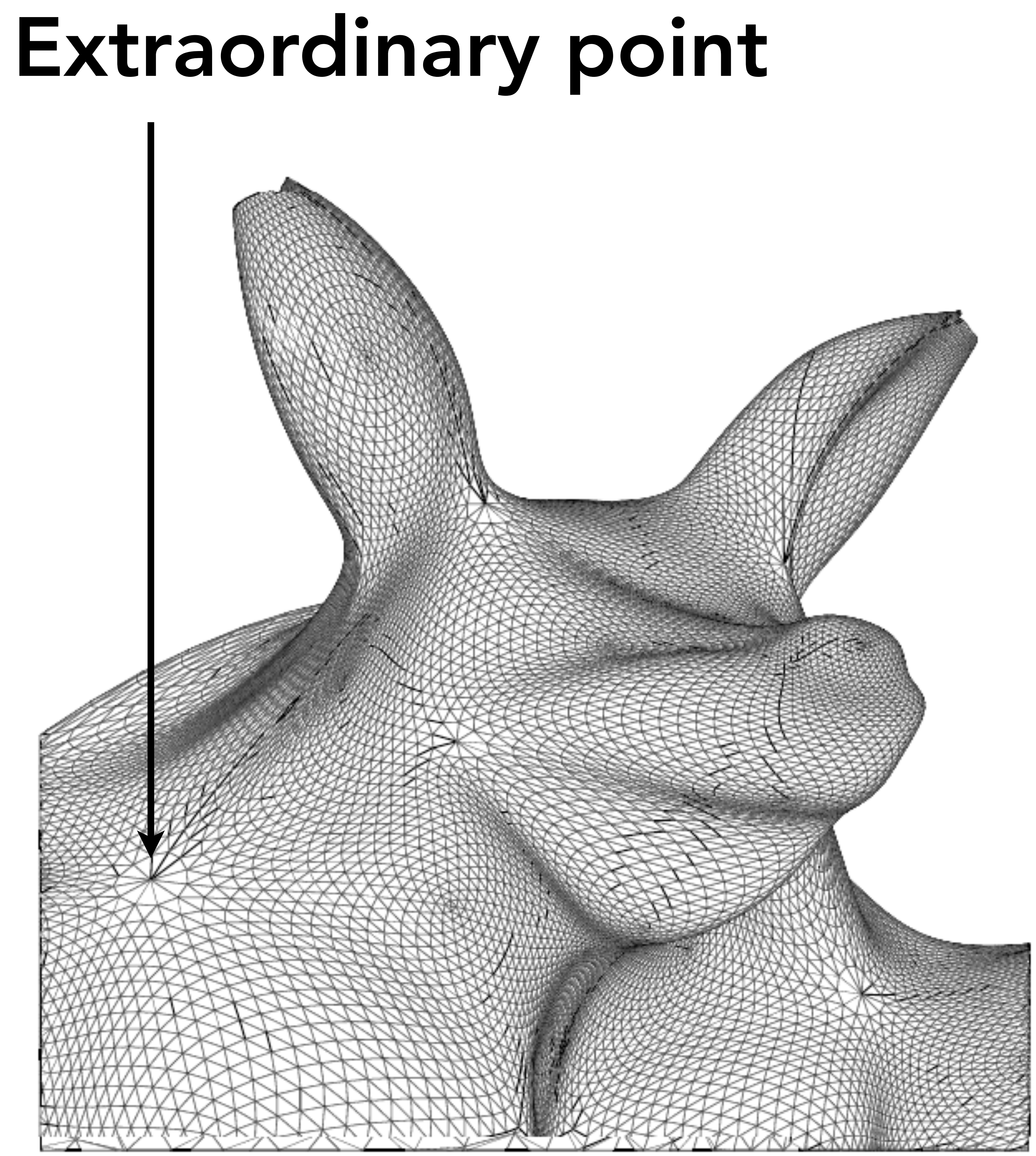
Simon Fuhrman

Semi-Regular Meshes

Most of the mesh has vertices with degree 6

But if the mesh is topologically equivalent to a sphere, then not all the vertices can have degree 6

Must have a few extraordinary points (degree not equal to 6)



Proof: Always an Extraordinary Vertex

Our mesh (topologically equivalent to sphere) has V vertices, E edges, and T triangles

$E = 3/2\ T$

- There are 3 edges per triangle, and each edge is part of 2 triangles
- Therefore $E = 3/2T$

$T = 2V - 4$

- Euler Convex Polyhedron Formula: $T - E + V = 2$
- => $V = 3/2\ T - T + 2$ => $T = 2V - 4$

If all vertices had 6 triangles, $T = 2V$

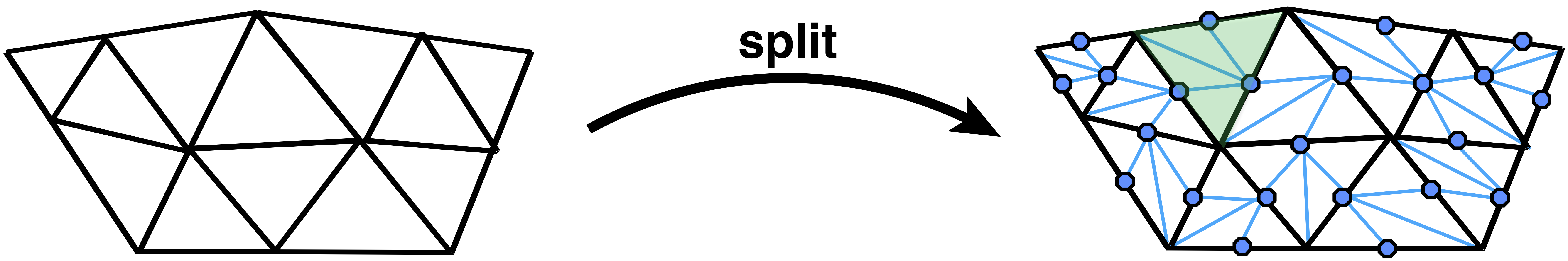
- There are 6 edges per vertex, and every edge connects 2 vertices
- Therefore, $E = 6/2V$ => $3/2T = 6/2V$ => $T = 2V$

T cannot equal both $2V - 4$ and $2V$, a contradiction

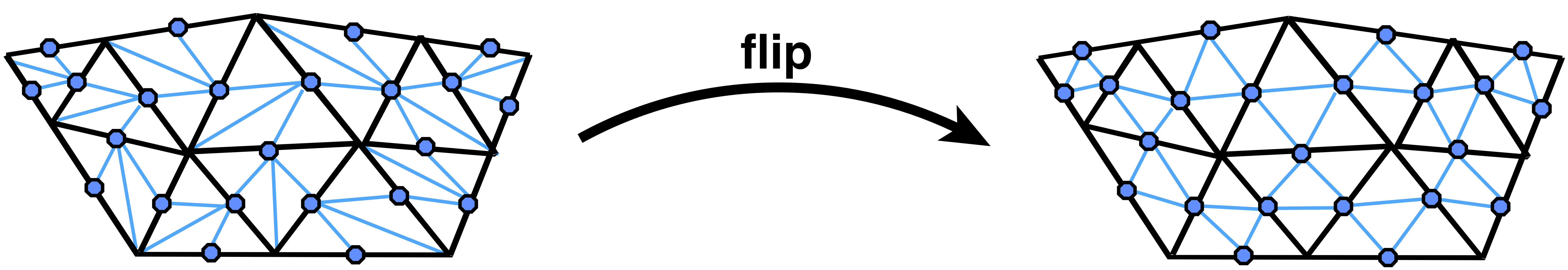
- Therefore, the mesh cannot have 6 triangles for every vertex

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 Keenan Crane, cribbed from Denis Zorin

Continuity of Loop Subdivision Surface

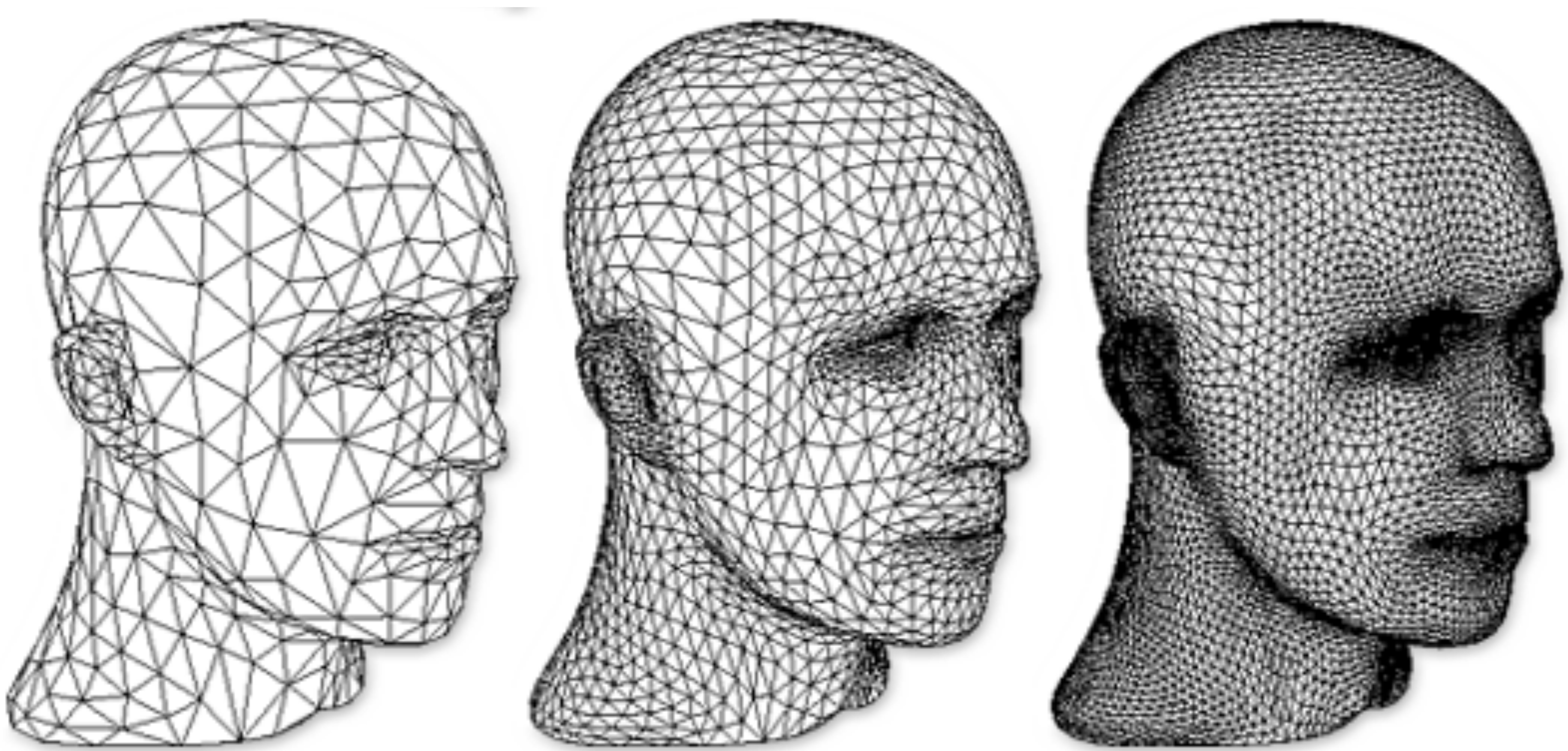
At extraordinary points

- Surface is at least C^1 continuous

Everywhere else (“ordinary” regions)

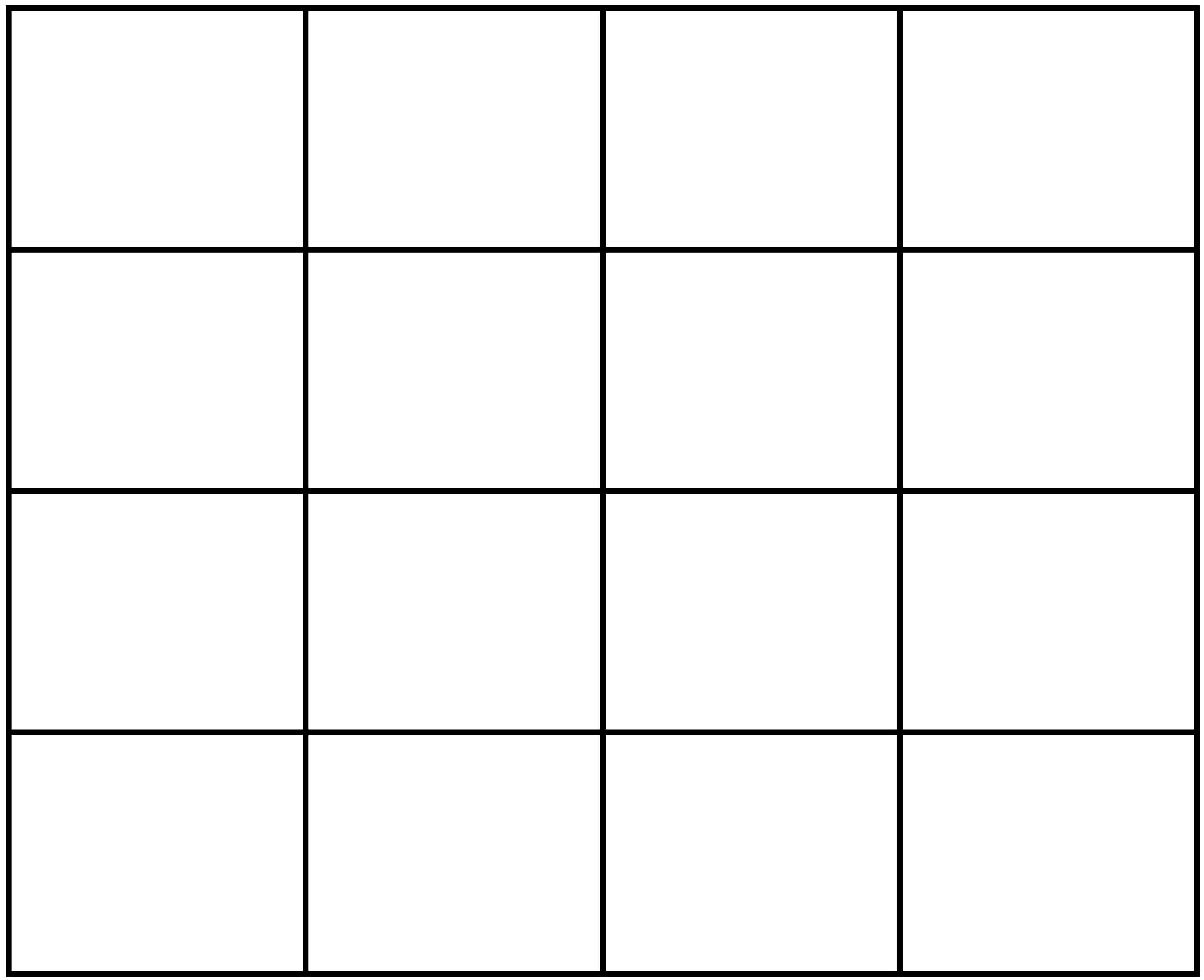
- Surface is C^2 continuous

Loop Subdivision Results

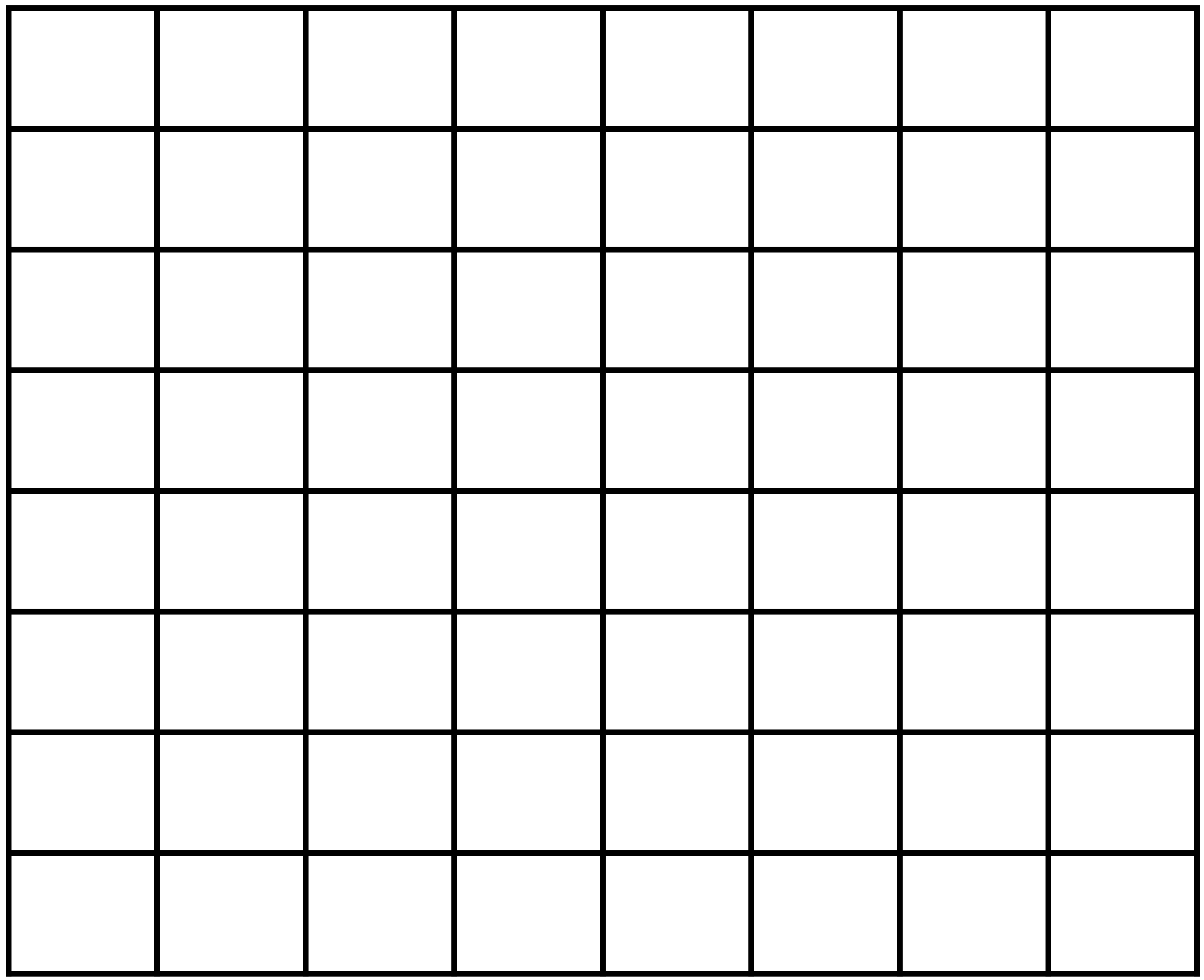


Catmull-Clark Subdivision

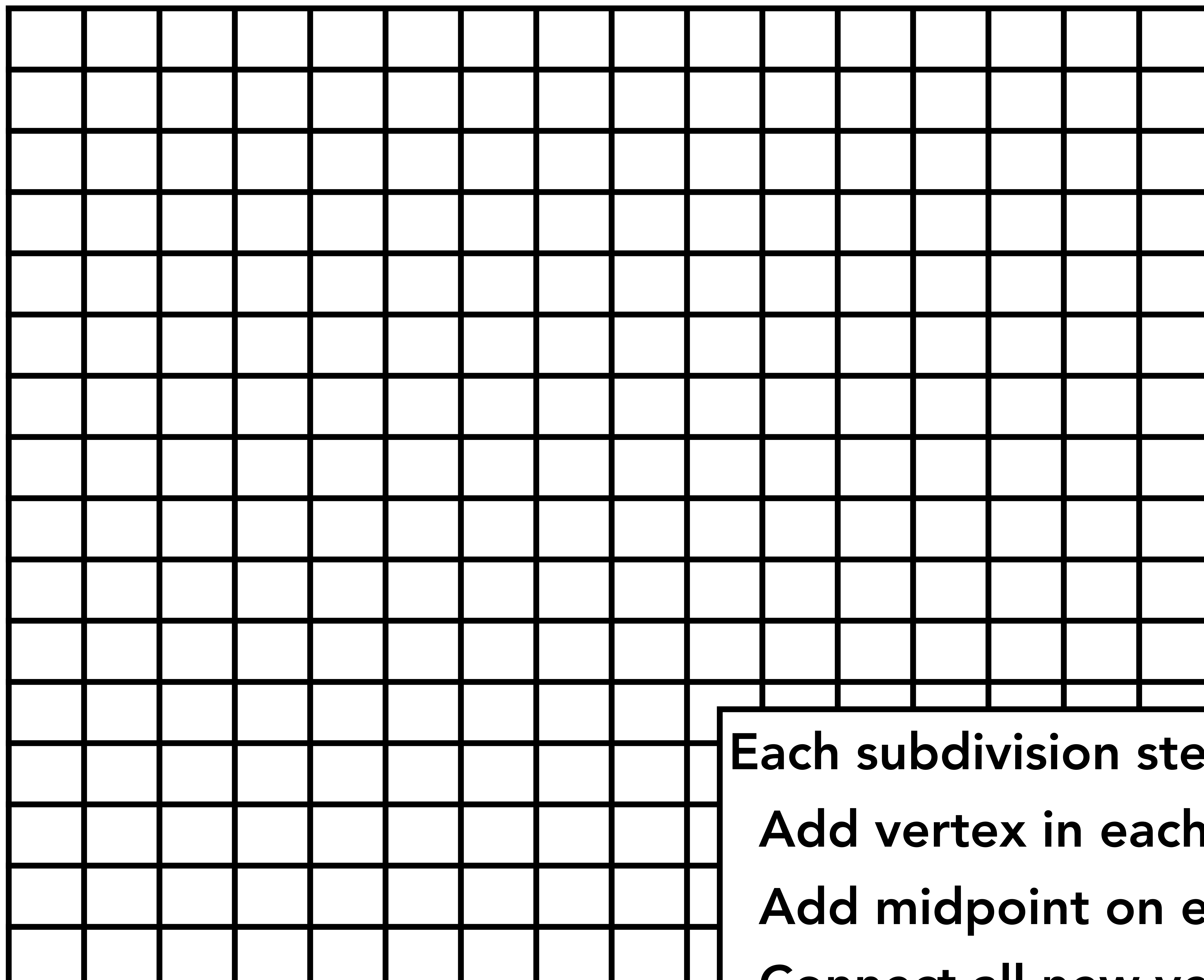
Catmull-Clark Subdivision (Regular Quad Mesh)



Catmull-Clark Subdivision (Regular Quad Mesh)



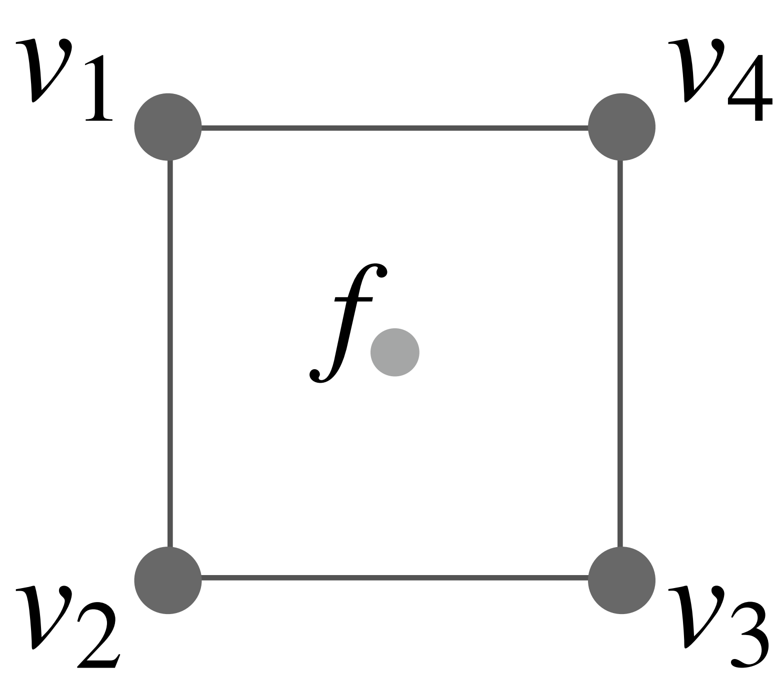
Catmull-Clark Subdivision (Regular Quad Mesh)



Each subdivision step:
  Add vertex in each face
  Add midpoint on each edge
  Connect all new vertices

Catmull-Clark Vertex Update Rules (Quad Mesh)

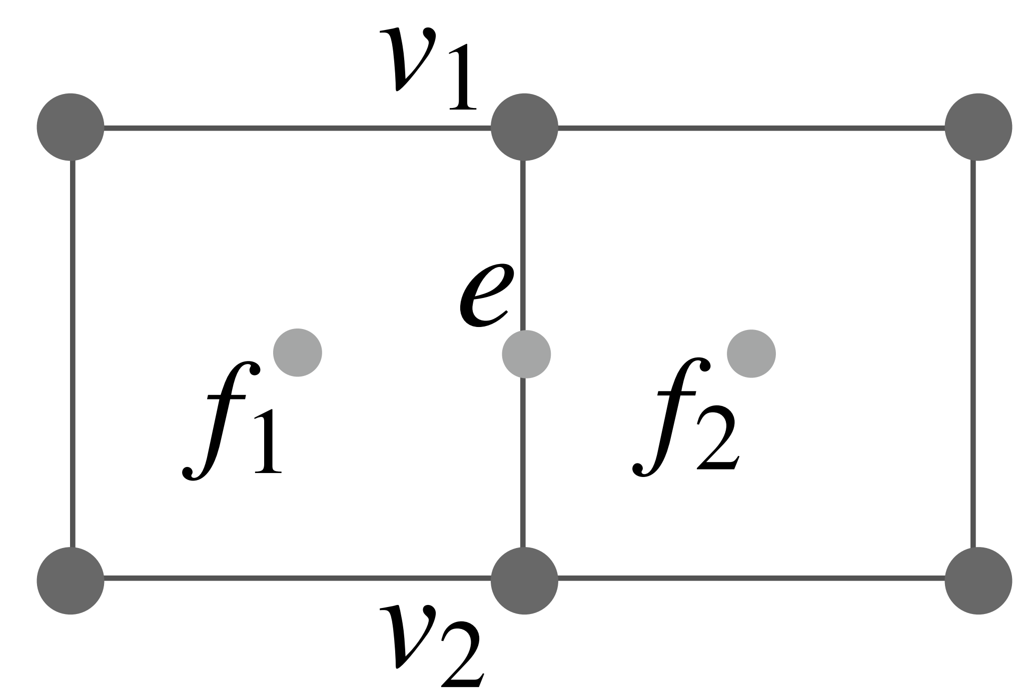
Face point

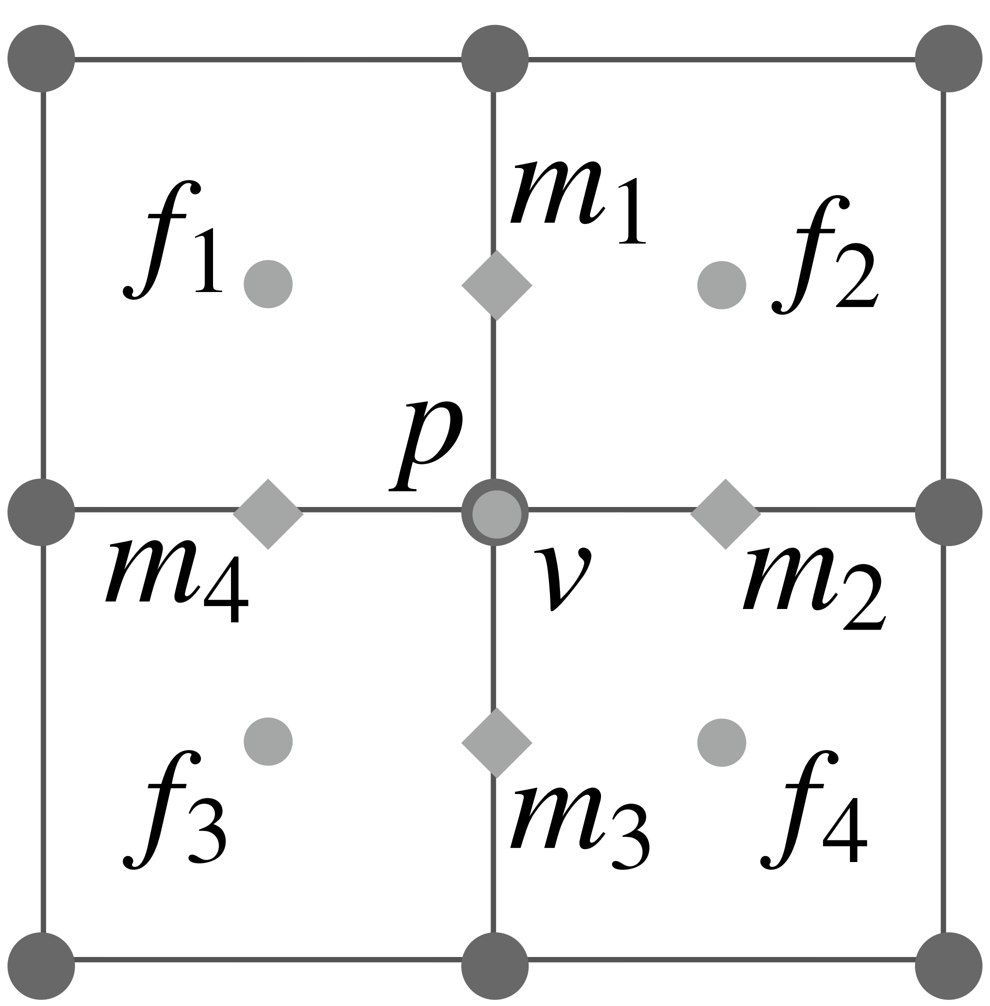


$$f = \frac{v_1 + v_2 + v_3 + v_4}{4}$$

$$e = \frac{v_1 + v_2 + f_1 + f_2}{4}$$

Edge point





Vertex point

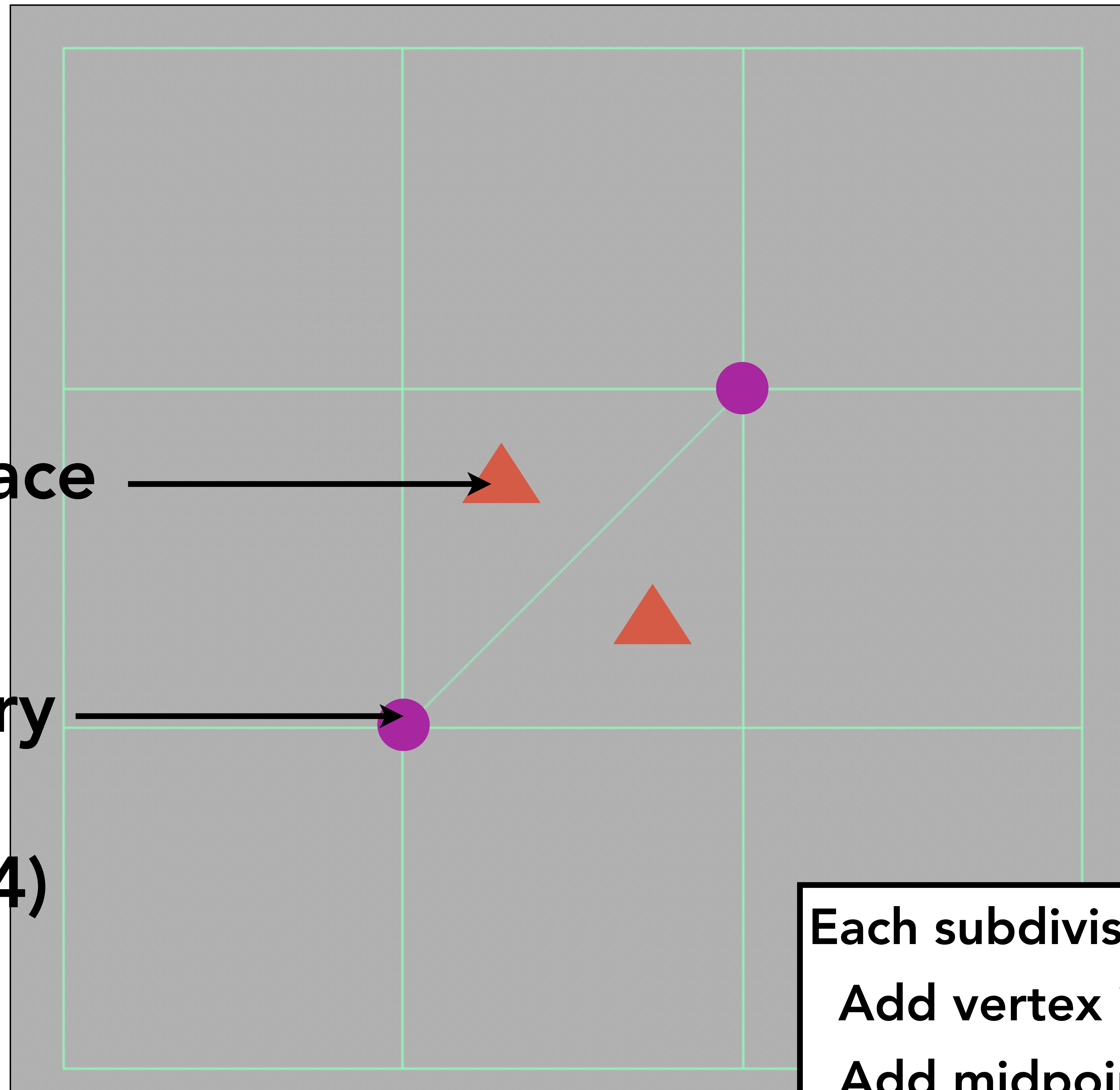
$$v = \frac{f_1 + f_2 + f_3 + f_4 + 2(m_1 + m_2 + m_3 + m_4) + 4p}{16}$$

m midpoint of edge, not “edge point”
p old “vertex point”

Catmull-Clark Subdivision (General Mesh)

Non-quad face

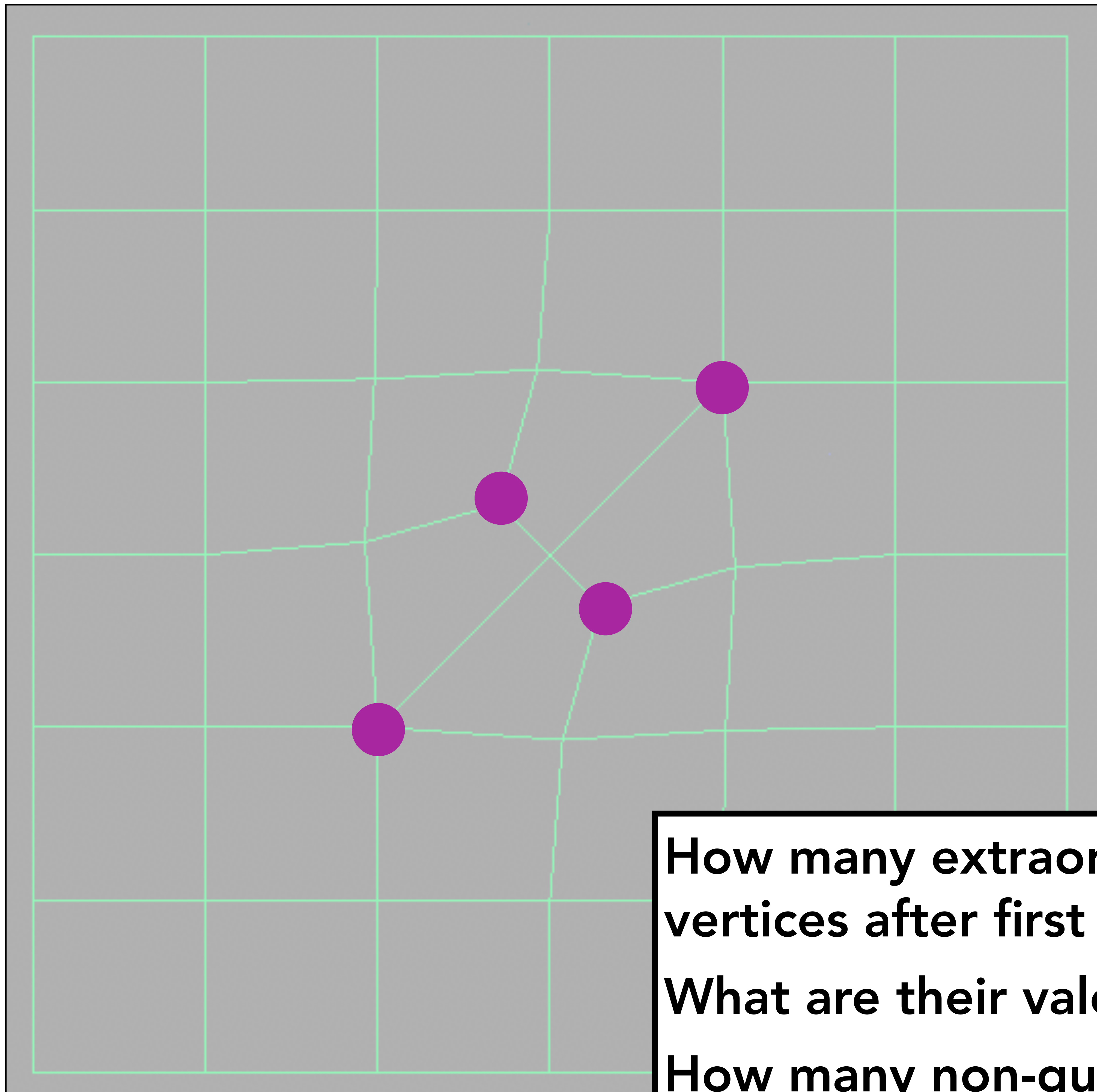
Extraordinary vertex (valence != 4)



Each subdivision step:

- Add vertex in each face
- Add midpoint on each edge
- Connect all new vertices

Catmull-Clark Subdivision (General Mesh)

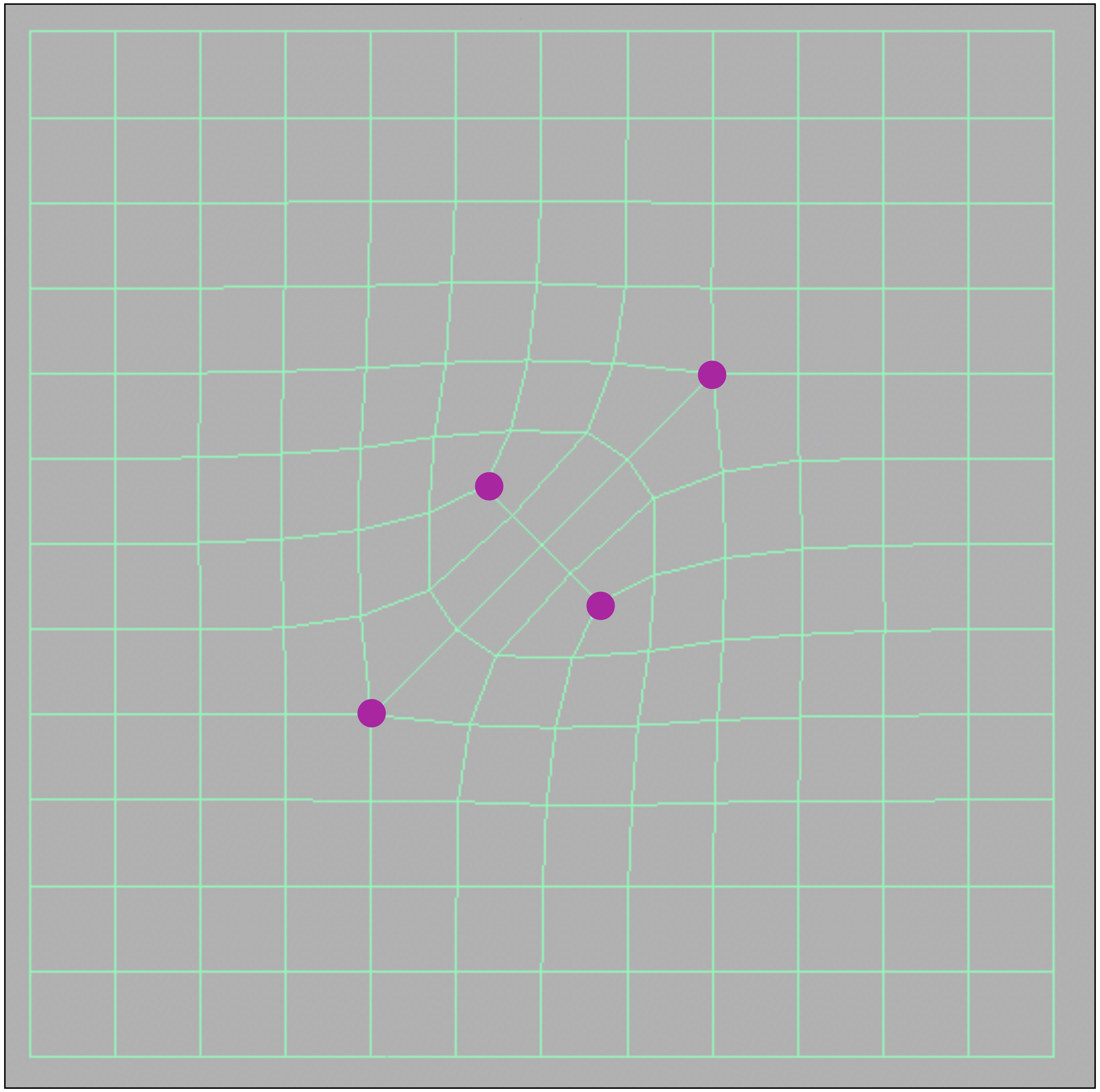


How many extraordinary vertices after first subdivision?

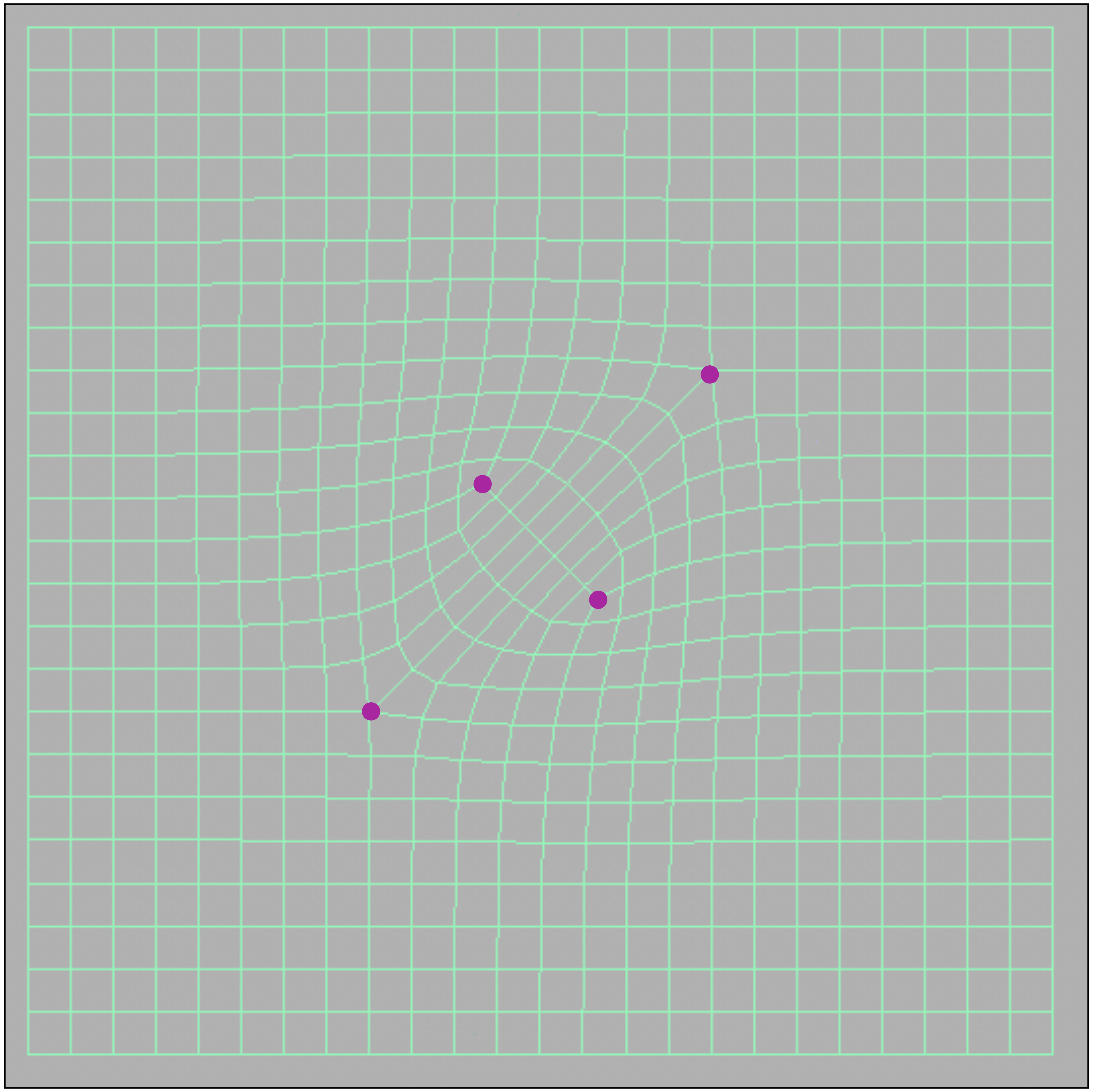
What are their valences?

How many non-quad faces?

Catmull-Clark Subdivision (General Mesh)



Catmull-Clark Subdivision (General Mesh)



Catmull-Clark Vertex Update Rules (General Mesh)

$$f = \text{average of surrounding vertices}$$

$$e = \frac{f_1 + f_2 + v_1 + v_2}{4}$$

$$v = \frac{\bar{f}}{n} + \frac{2\bar{m}}{n} + \frac{p(n-3)}{n}$$

These rules reduce to earlier quad rules for ordinary vertices / faces

$$\bar{m} = \text{average of adjacent midpoints}$$
$$\bar{f} = \text{average of adjacent face points}$$
$$n = \text{valence of vertex}$$
$$p = \text{old ''vertex'' point}$$

Continuity of Catmull-Clark Surface

At extraordinary points

- Surface is at least C^1 continuous

Everywhere else ("ordinary" regions)

- Surface is C^2 continuous

What About Sharp Creases?

From Pixar Short, "Geri's Game"
Hand is modeled as a Catmull Clark surface with creases between skin and fingernail

What About Sharp Creases?

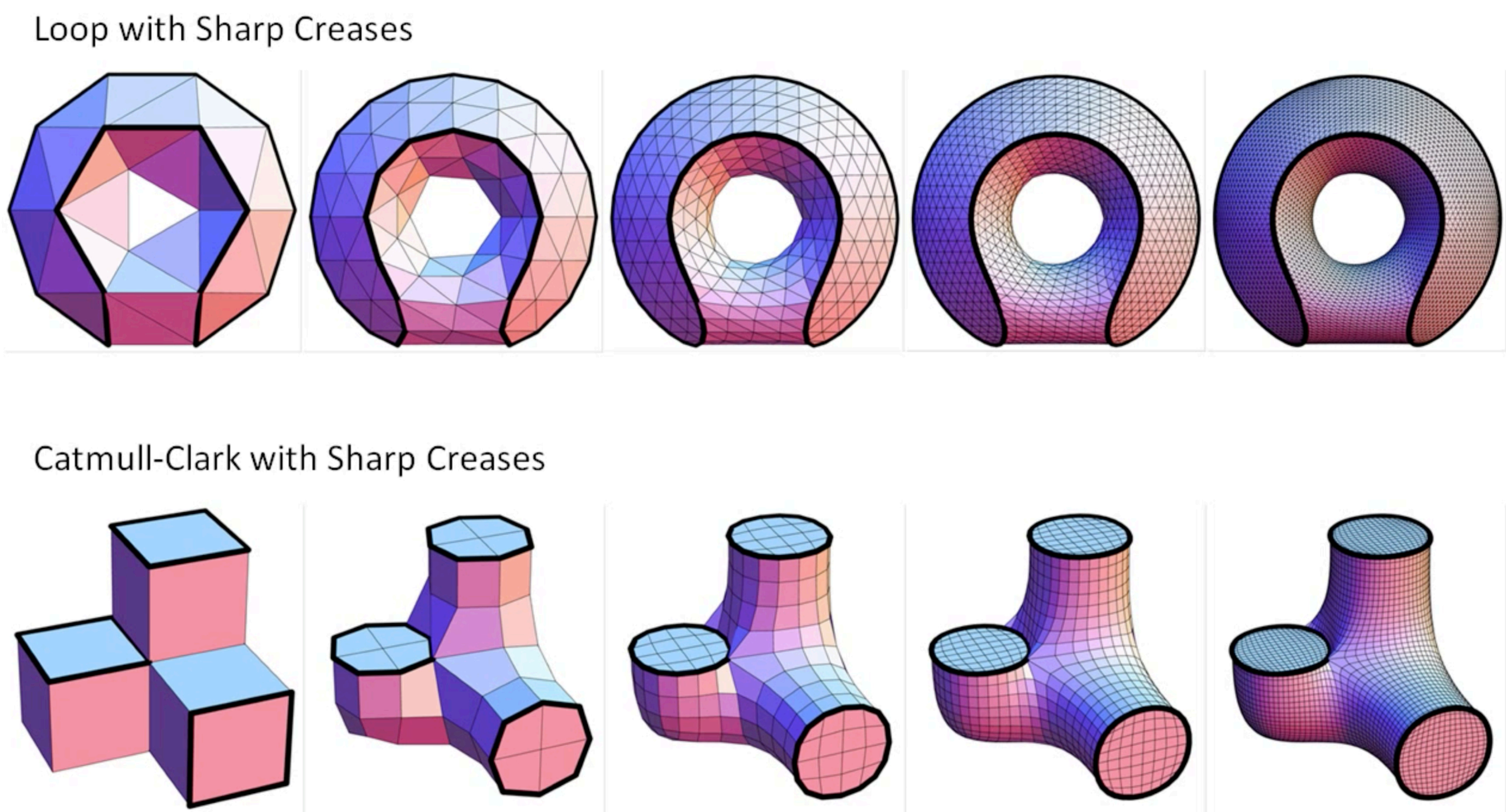
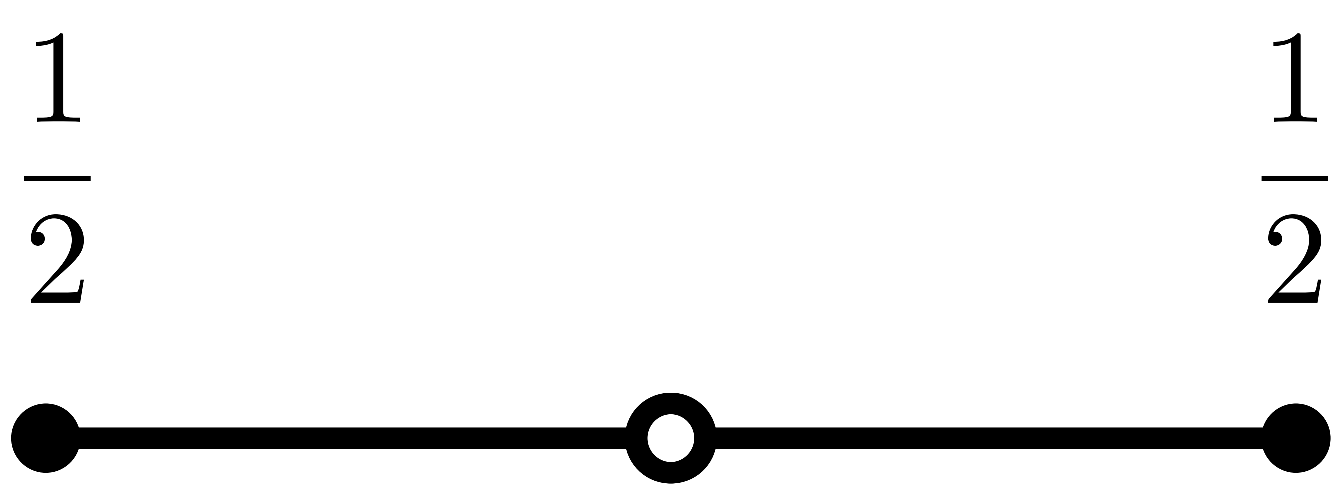


Figure from: Hakenberg et al. Volume Enclosed by Subdivision Surfaces with Sharp Creases

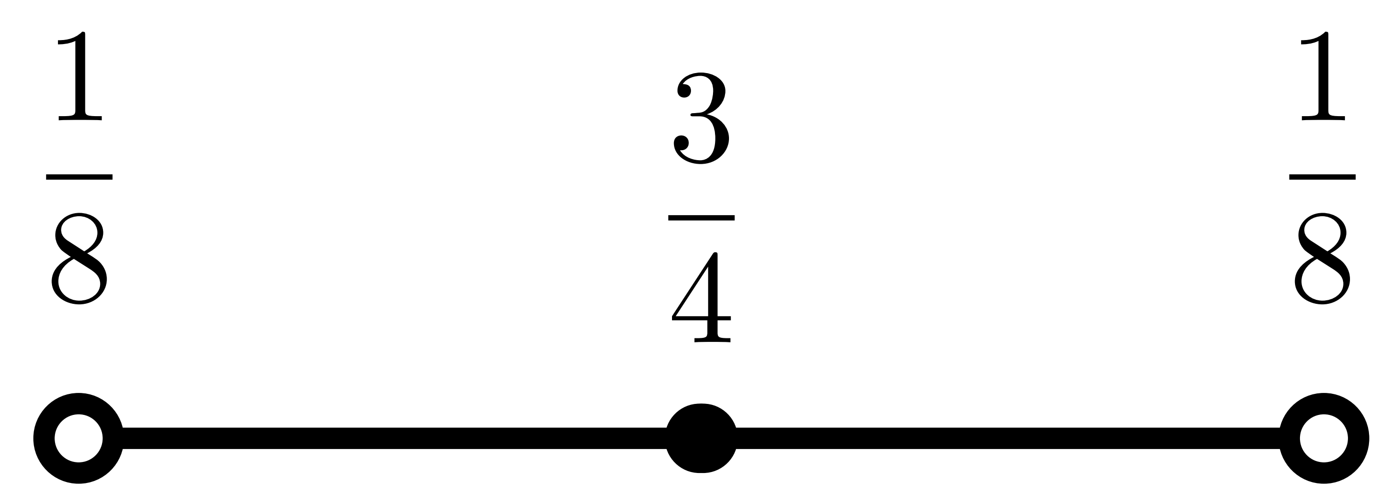
Creases + Boundaries

Can create creases in subdivision surfaces by marking certain edges as “sharp”. Boundary edges can be handled the same way

- Use different subdivision rules for vertices along these “sharp” edges



Insert new midpoint vertex, weights as shown



Update existing vertices, weights as shown

Subdivision in Action ("Geri's Game", Pixar)

Subdivision used for entire character:

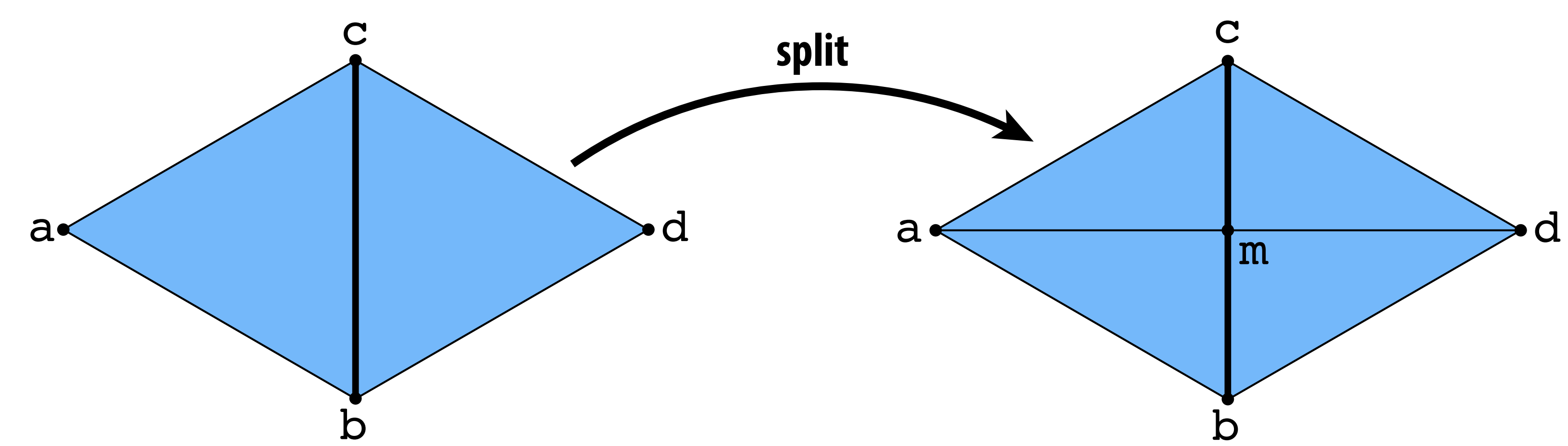
- Hands and head
- Clothing, tie, shoes

Subdivision in Action (Pixar's "Geri's Game")

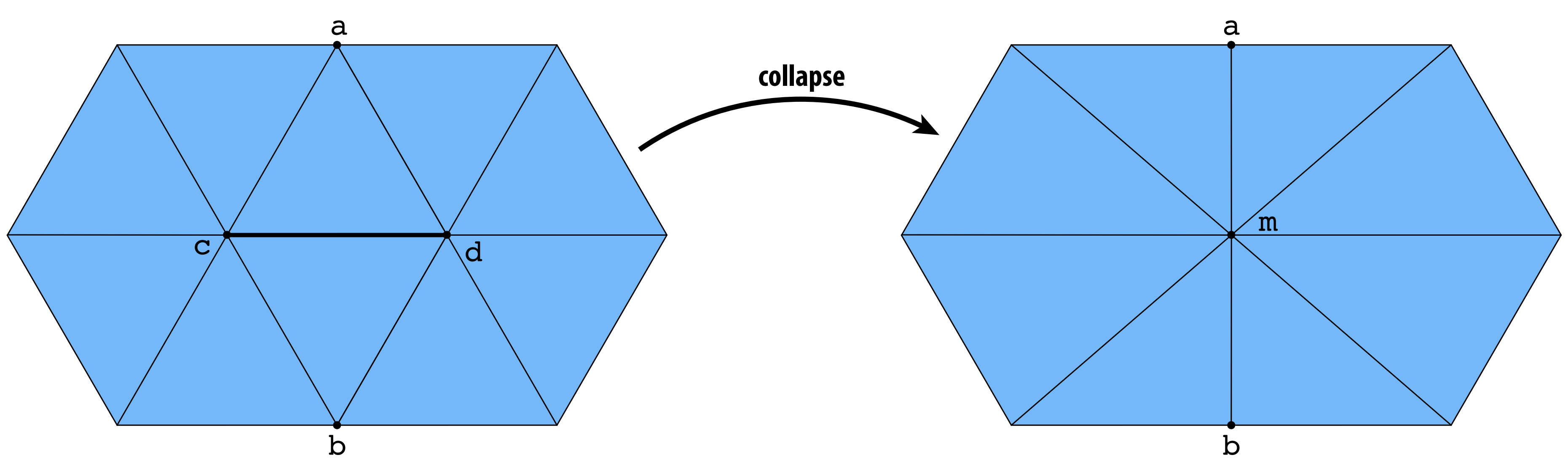
Mesh Simplification

How Do We Resample Meshes? (Reminder)

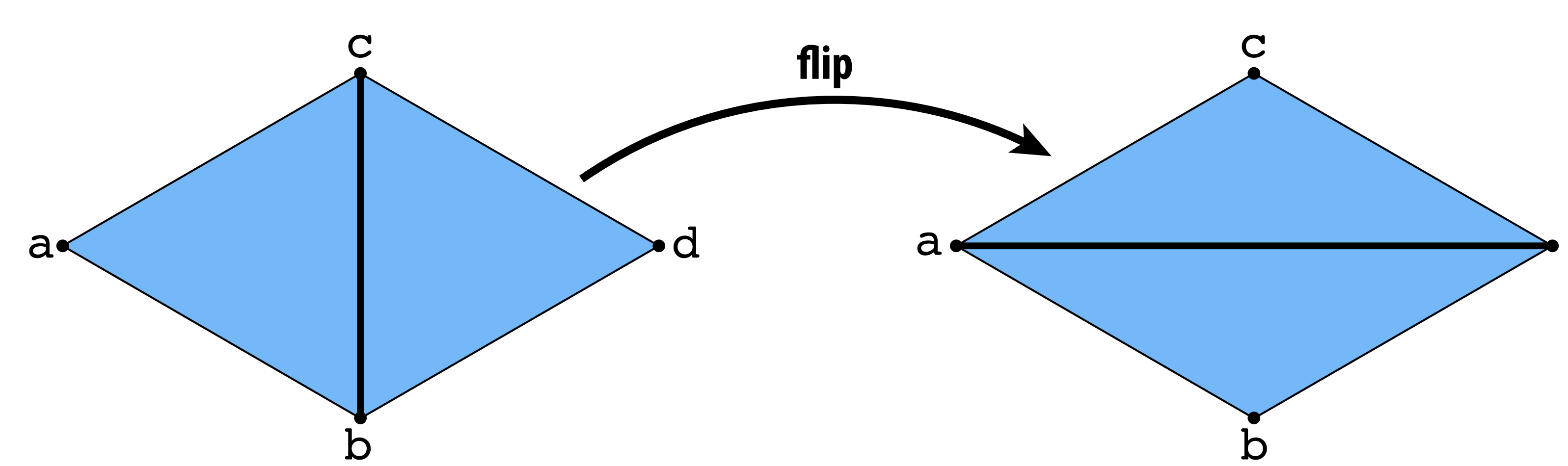
Edge split is (local) upsampling:



Edge collapse is (local) downsampling:



Edge flip is (local) resampling:



Still need to intelligently decide which edges to modify!

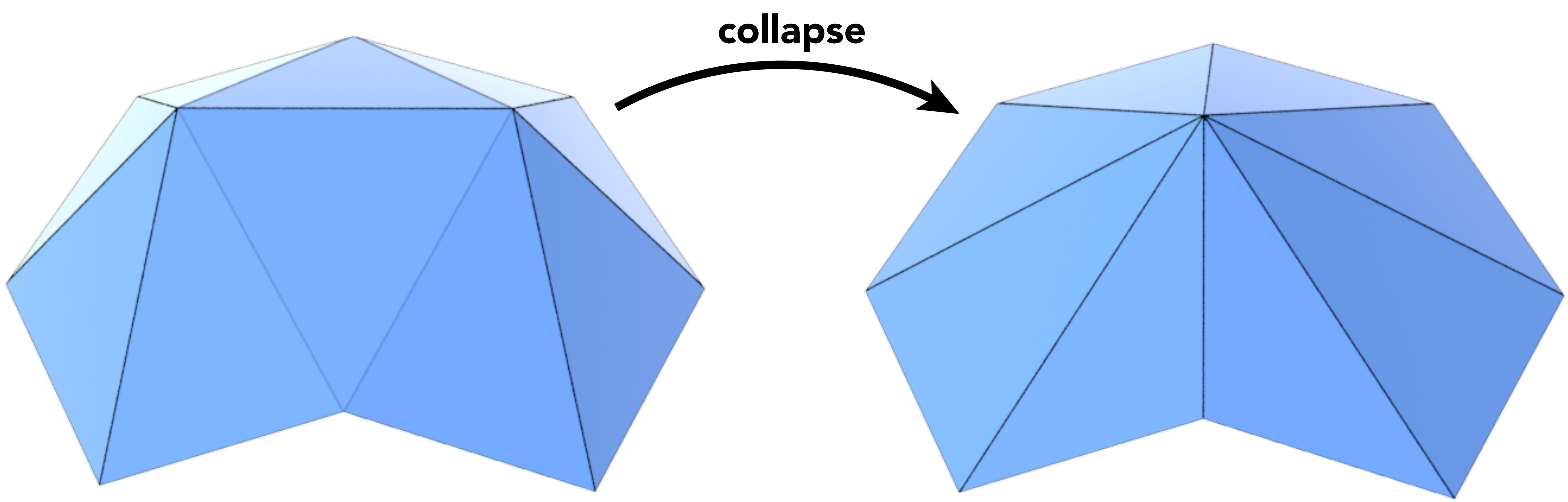
Mesh Simplification

Goal: reduce number of mesh elements while maintaining overall shape

How to compute?

Estimate: Error Introduced by Collapsing An Edge?

- How much geometric error for collapsing an edge?



Sketch of Quadric Error Mesh Simplification

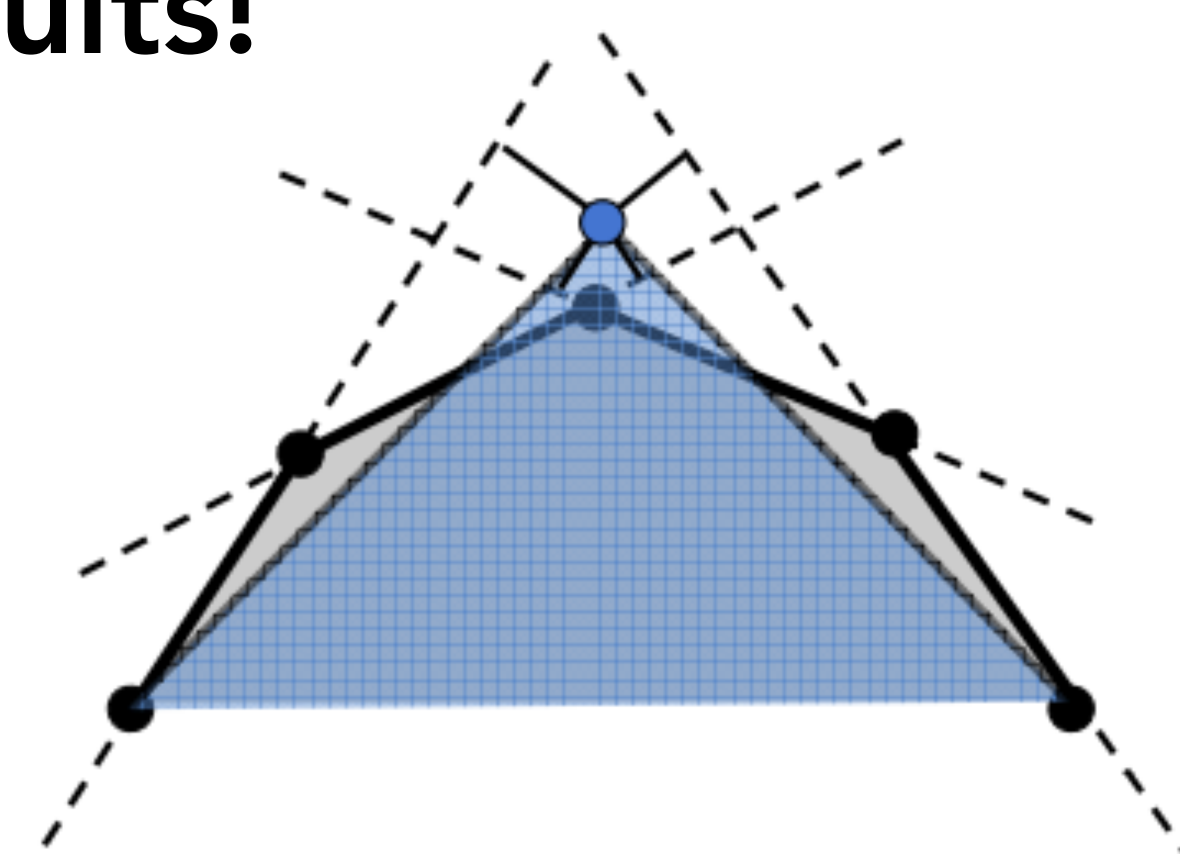
Simplification via Quadric Error

Iteratively collapse edges

Which edges? Assign score with quadric error metric*

- approximate distance to surface as sum of distances to planes containing triangles
- iteratively collapse edge with smallest score
- greedy algorithm... great results!

* (Garland & Heckbert 1997)



Quadric Error Matrix

Key idea:

- 4x4 (“quadric”) symmetric matrix encodes distance to plane

For plane ax + by + cz + d = 0

- Distance of query point (x, y, z) from plane is given by $u^T Q u$:
  - $u := (x, y, z, 1)^T$ is the query point in homogeneous coordinates
  - And Q is a symmetric matrix as follows:

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

  - Q contains 10 unique coefficients (small storage)

Quadric Error Matrix: Derivation

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

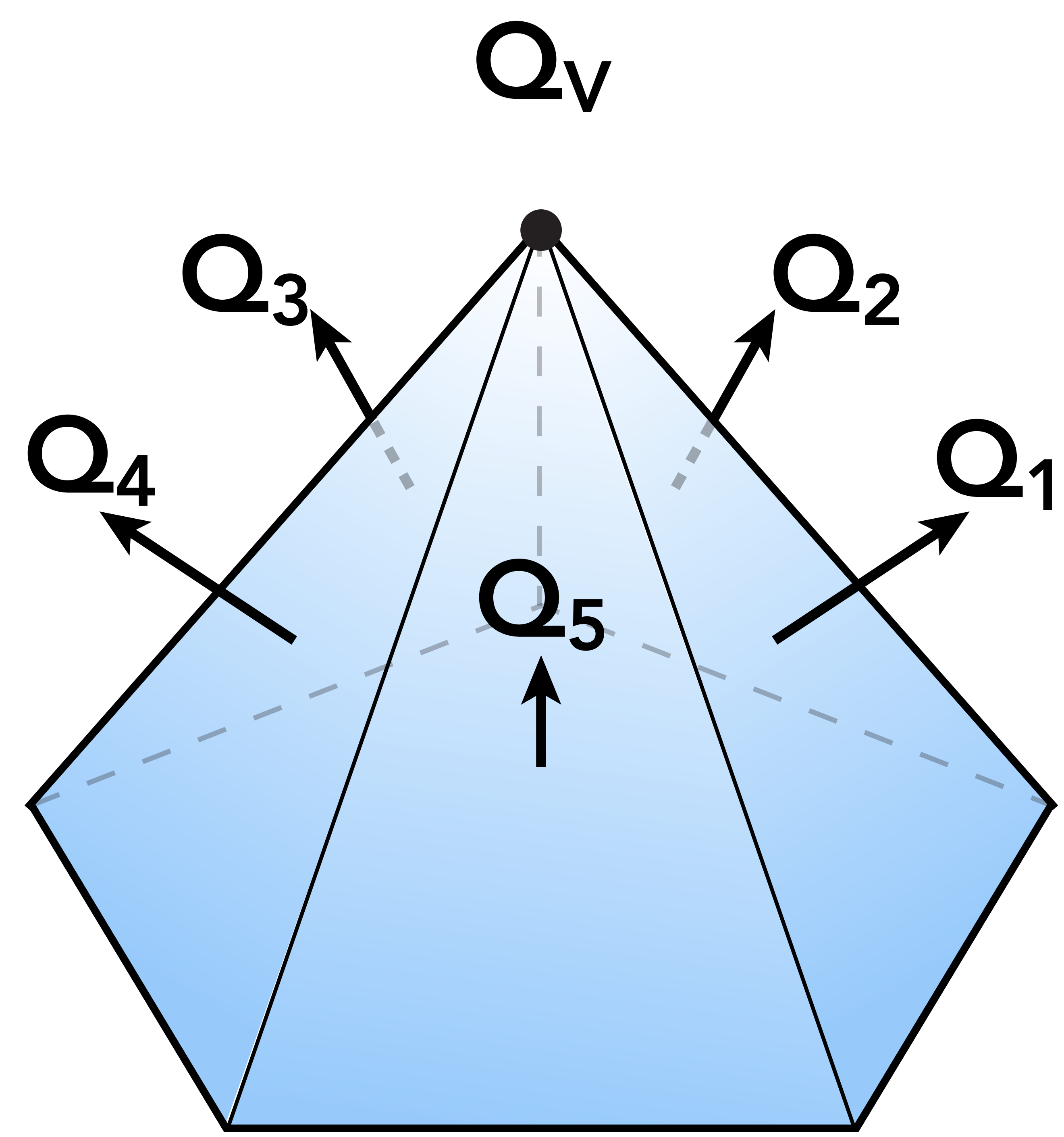
$$Q = \begin{bmatrix} a^2 & ab & ac & ad \\ ab & b^2 & bc & bd \\ ac & bc & c^2 & cd \\ ad & bd & cd & d^2 \end{bmatrix}$$

- Then in homogeneous coordinates, let
  - u := (x,y,z,1)
  - v := (a,b,c,d)
- Signed distance to plane is then
  $D = uv^T = vu^T = ax+by+cz+d$
- Squared distance is $D^2 = (uv^T)(vu^T) = u\,(v^Tv)\,u^T := u^TQu$

Quadric Error At Vertex

Approximate distance to vertex's triangles as sum of distances to each triangle's plane.

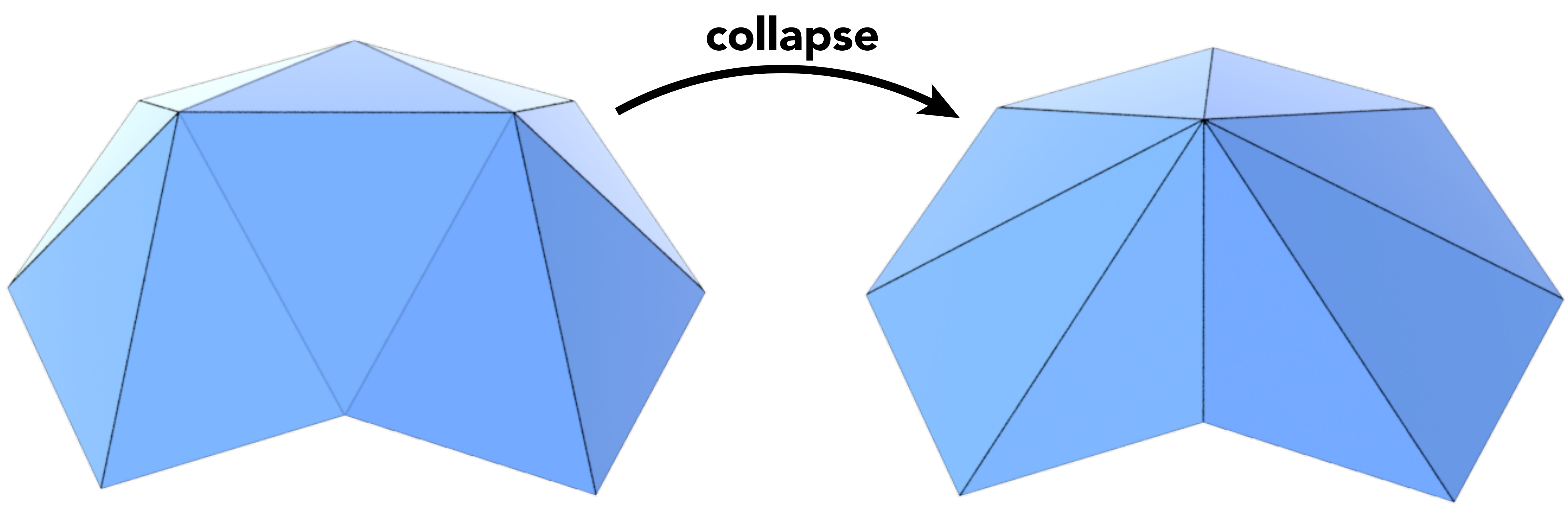
Encode this as a single quadric matrix for the vertex that is the sum of quadric error matrices for all triangles



$$Q_V = \sum_{i=1}^{N} Q_i$$

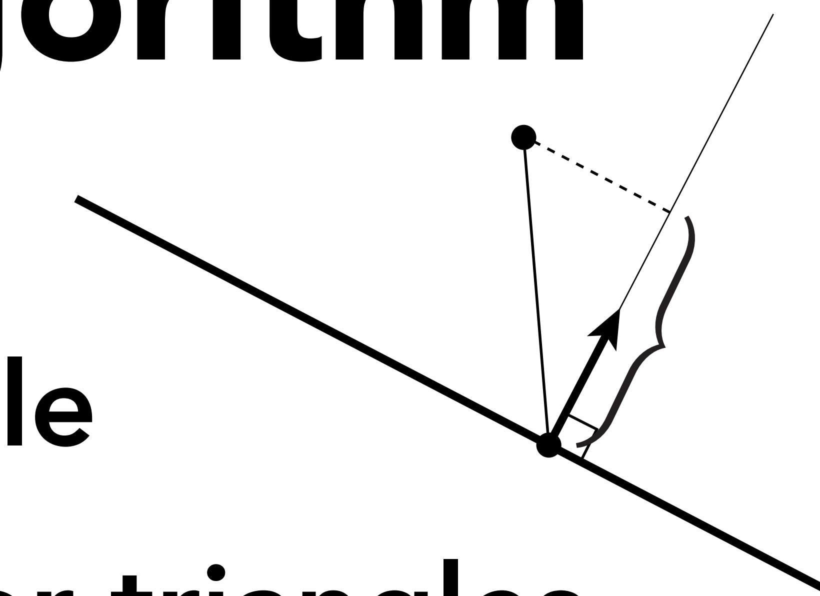
Quadric Error of Edge Collapse

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

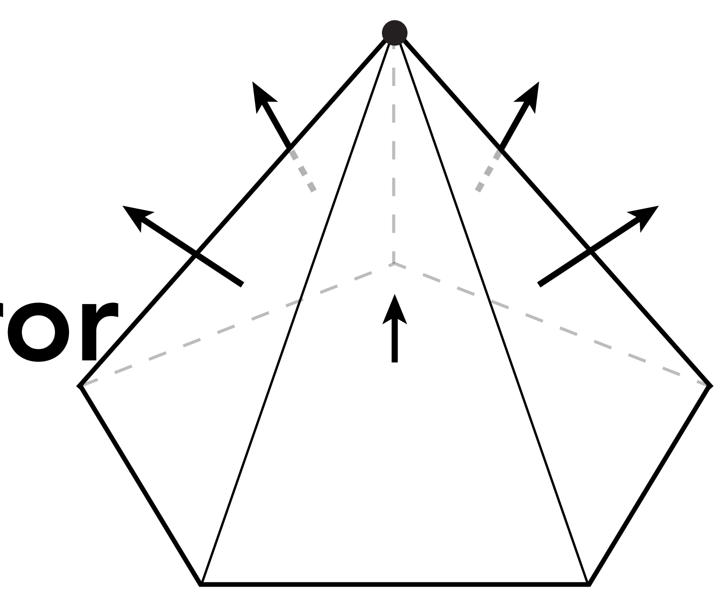


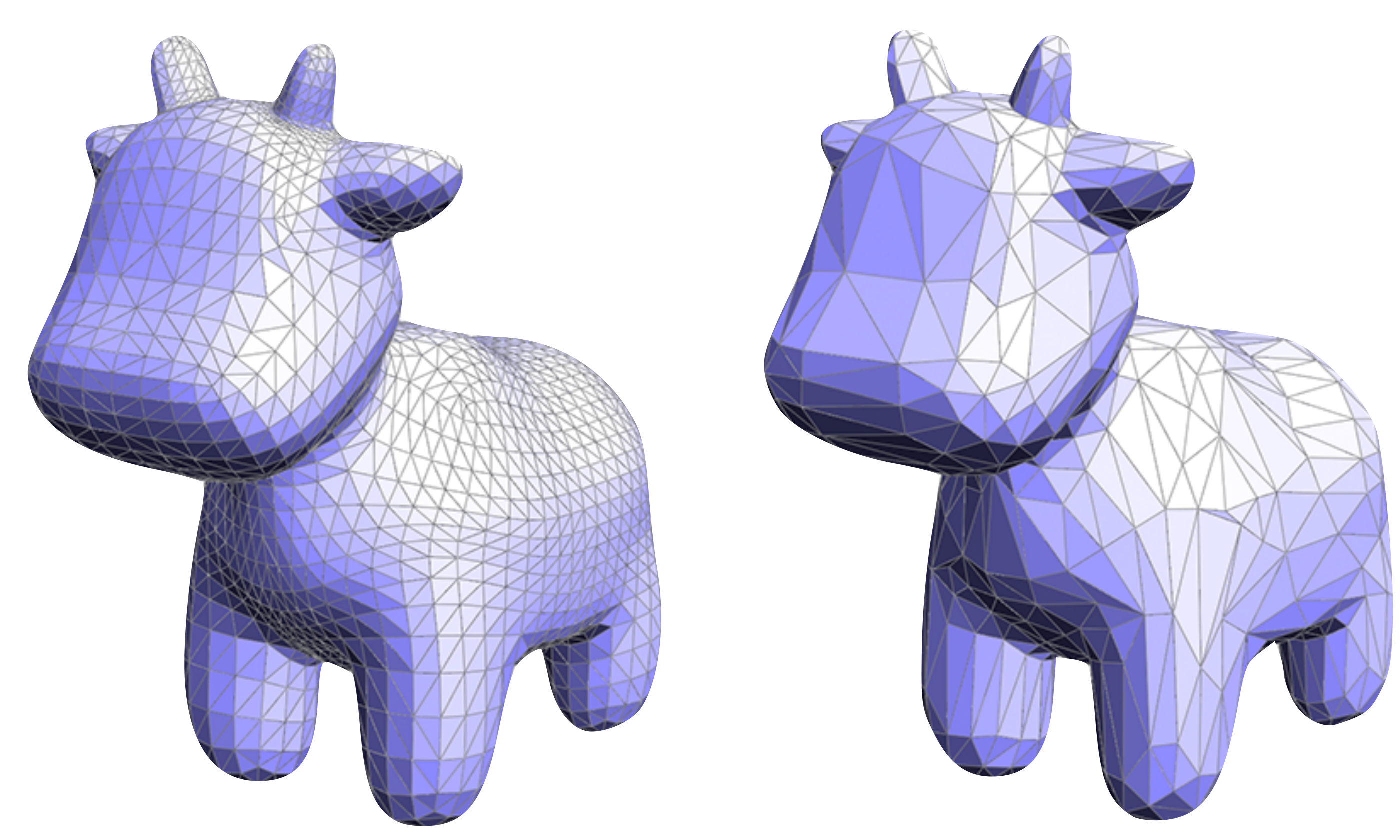
- Better idea: choose point that minimizes quadric error
- More details: Garland & Heckbert 1997.

Quadric Error Simplification: Algorithm



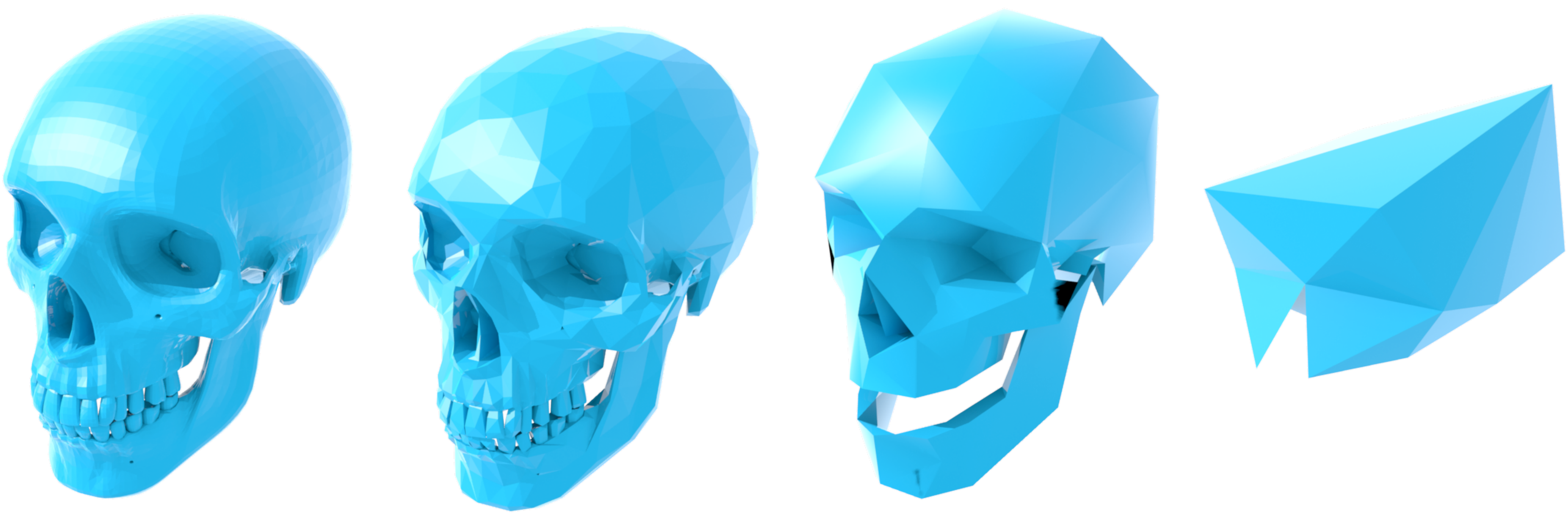
- Compute quadric error matrix Q for each triangle
- Set Q at each vertex to sum of Qs from neighbor triangles
- Set Q at each edge to sum of Qs at endpoints
- Find point at each edge minimizing quadric error
- Until we reach target # of triangles:
  - collapse edge (i,j) with smallest cost to get new vertex m
  - add Q_i and Q_j to get quadric Q_m at vertex m
  - update cost of edges touching vertex m





Quadric Error Mesh Simplification

5,804 994 532 248 64



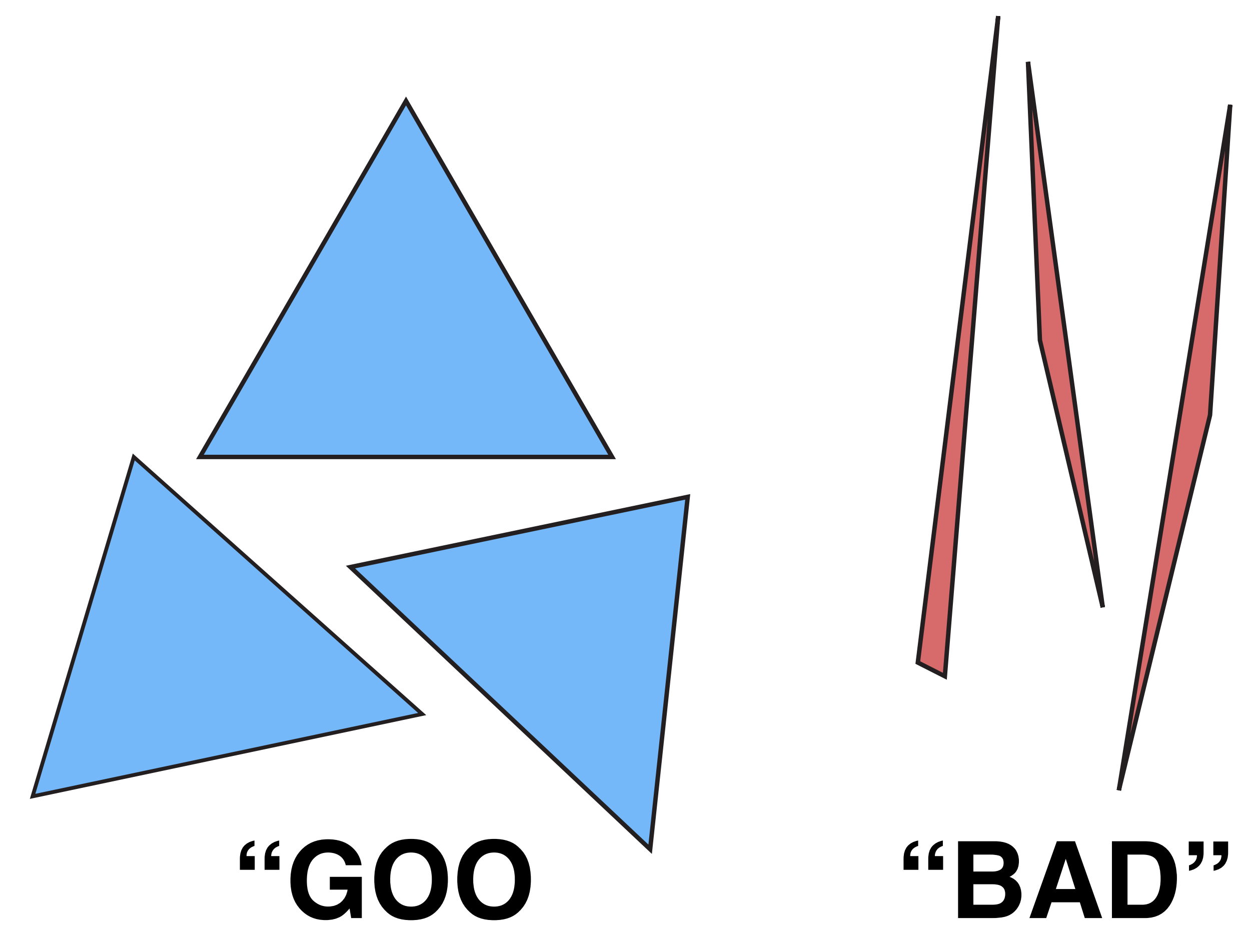
30,000 triangles 3,000 300 30

Garland and Heckbert '97

Mesh Regularization

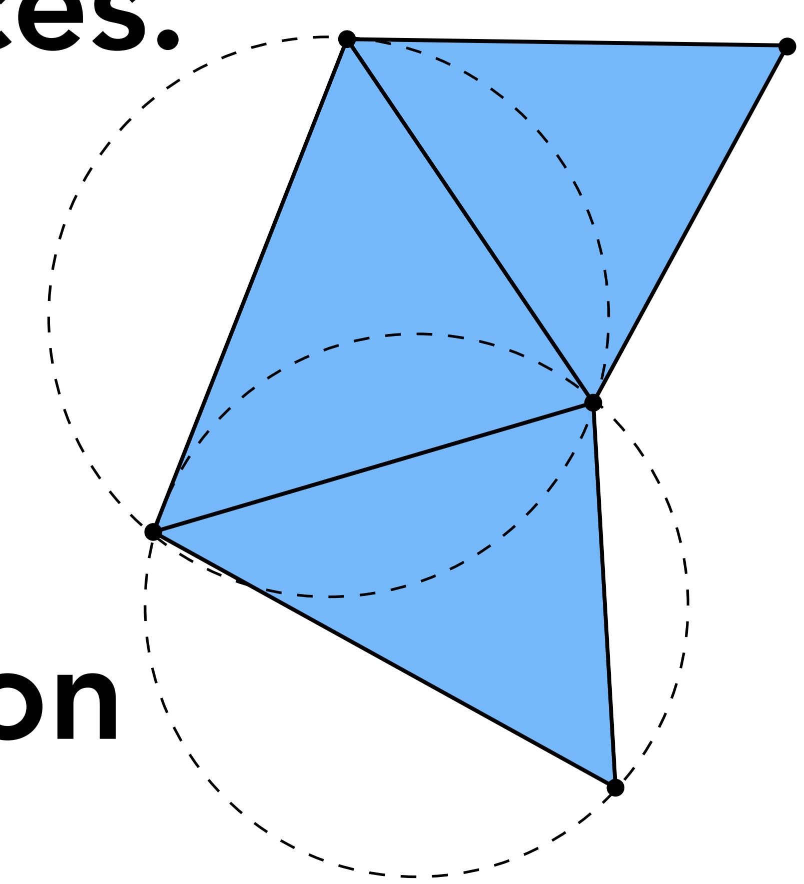
What Makes a "Good" Triangle Mesh?

One rule of thumb: triangle shape



More specific condition: Delaunay

- "Circumcircle interiors contain no vertices."



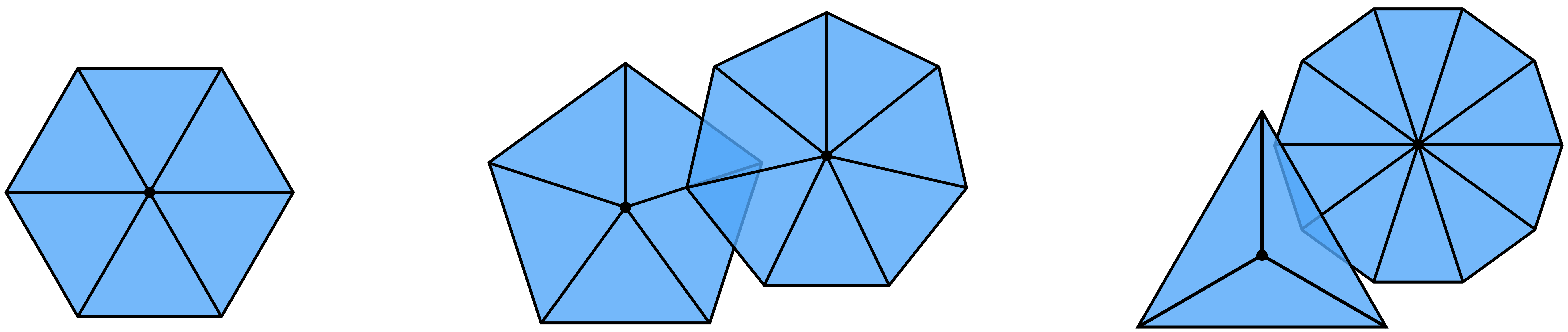
Not always a good condition, but often*

- Good for simulation
- Not always best for shape approximation

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

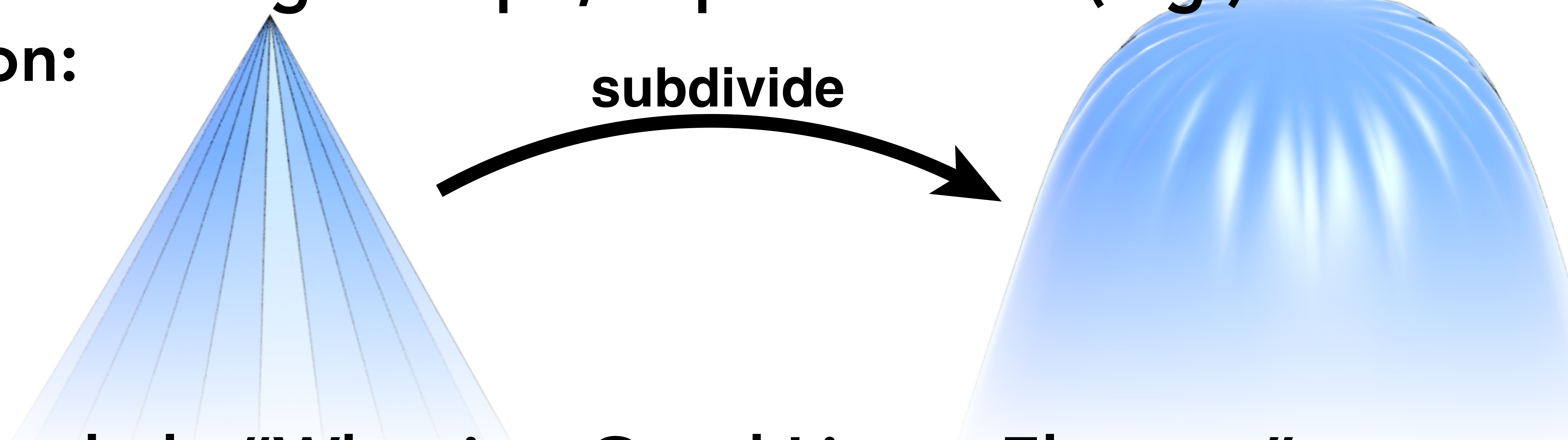
What Else Constitutes a Good Mesh?

Rule of thumb: regular vertex degree
Triangle meshes: ideal is every vertex with valence 6:



“GOOD” “OK” “BAD”

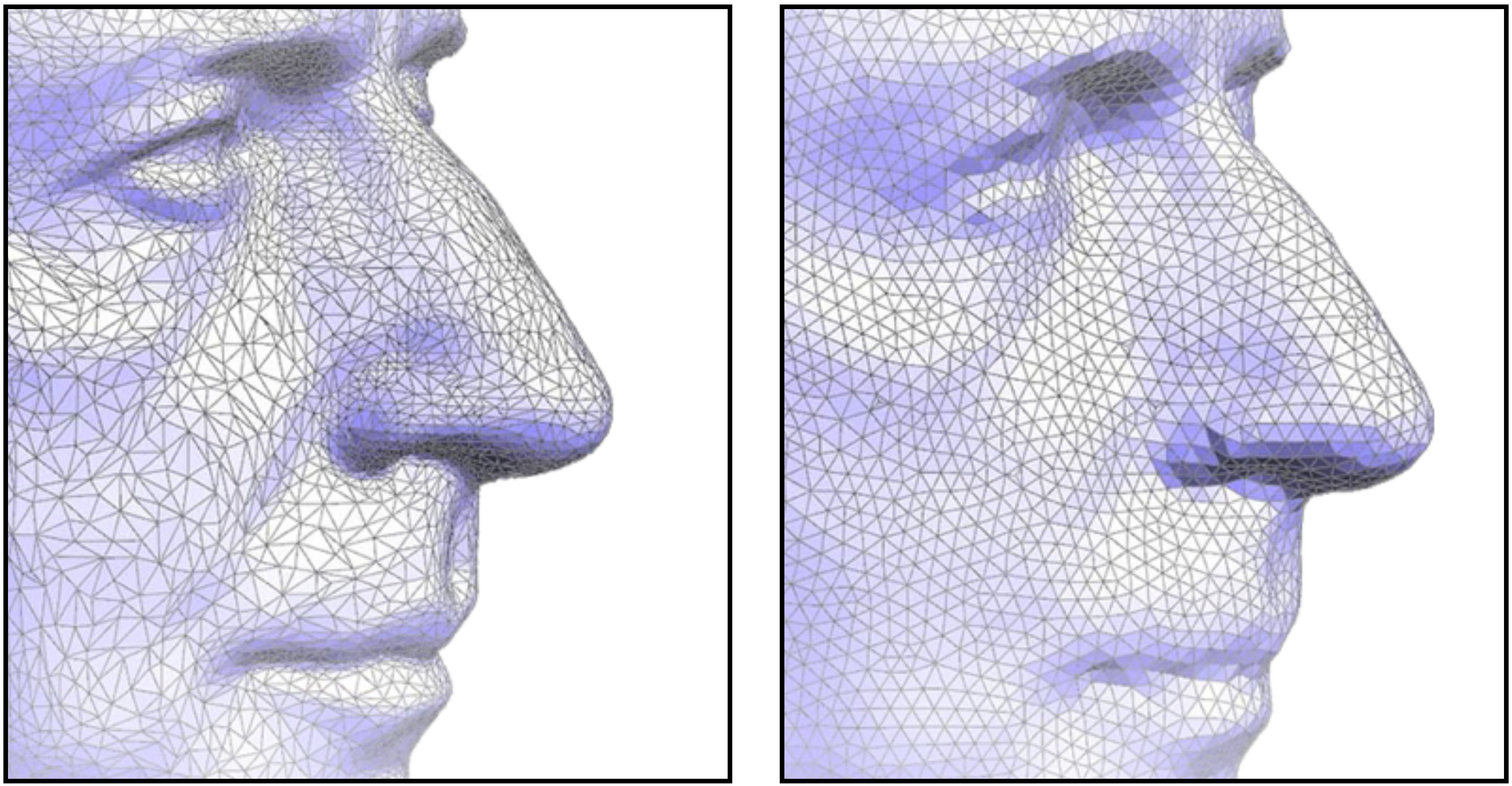
Why? Better triangle shape, important for (e.g.) subdivision:



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

Isotropic Remeshing

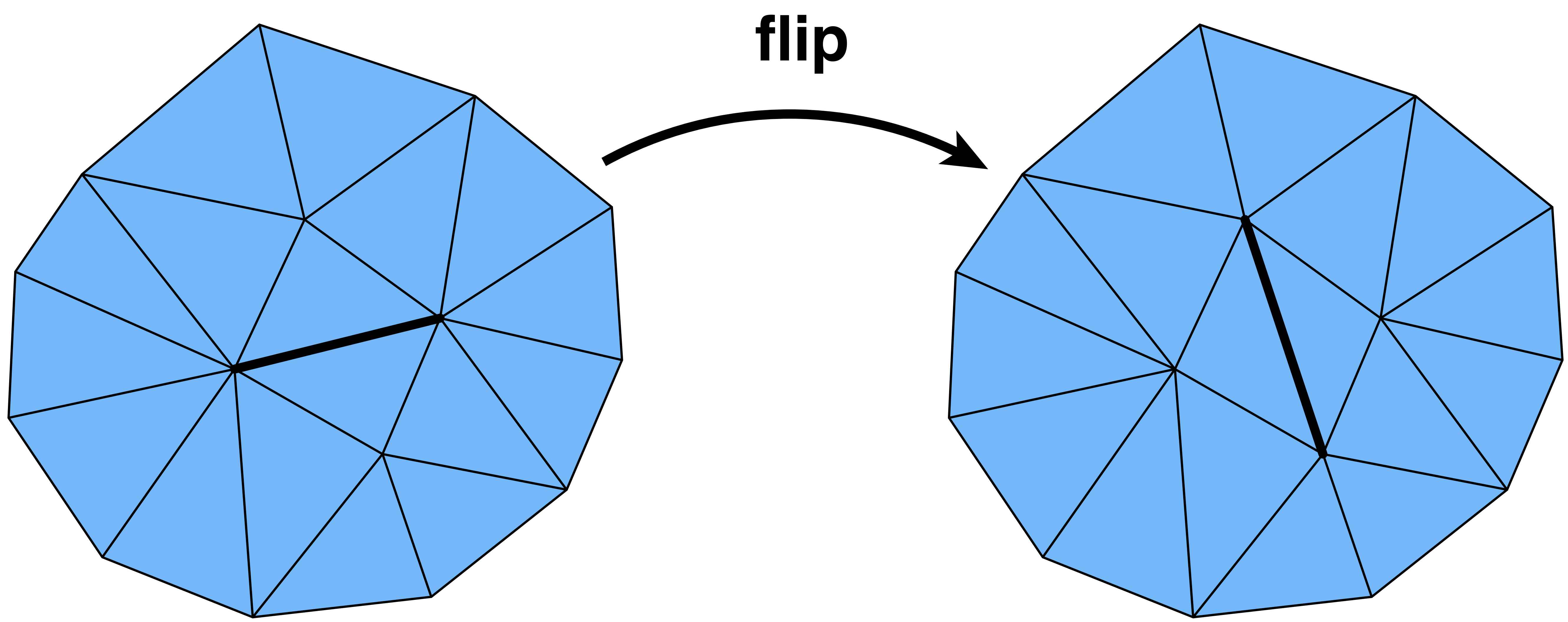
Try to make triangles uniform in shape and size



How Do We Improve Degree?

Edge flips!

If total deviation from degree 6 gets smaller, flip it!



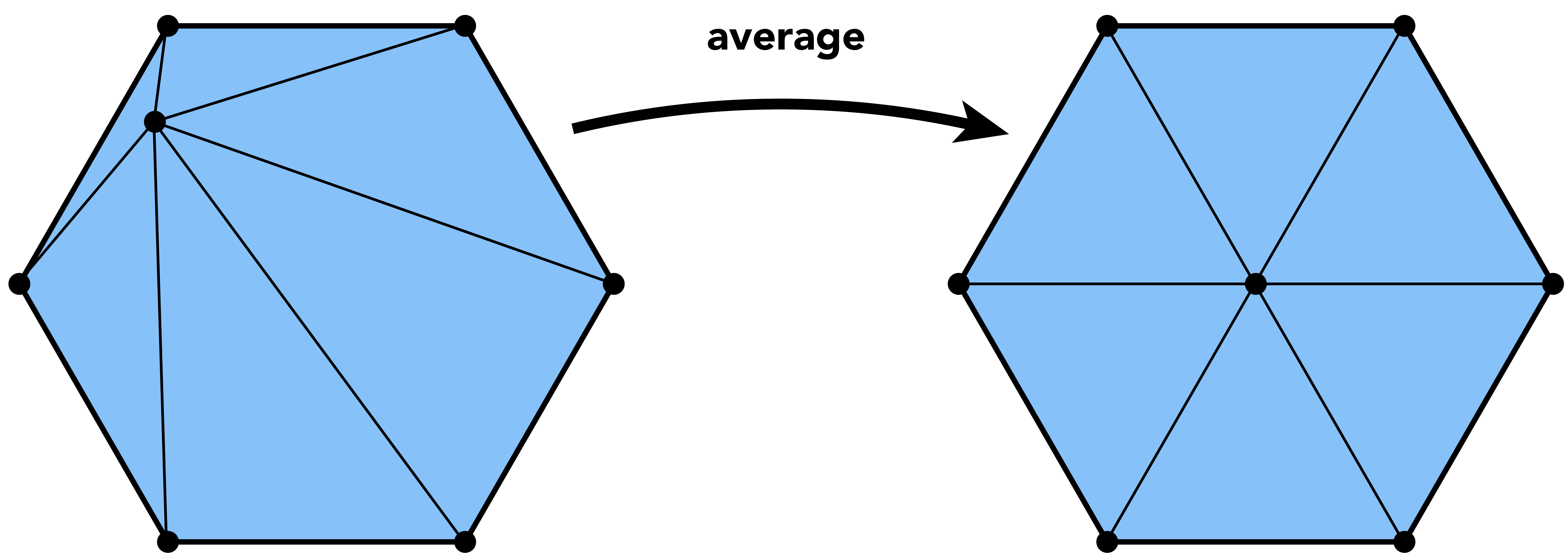
Iterative edge flipping acts like "discrete diffusion" of degree

No (known) guarantees; works well in practice

How Do We Make Triangles "More Round"?

Delaunay doesn't mean equilateral triangles

Can often improve shape by centering vertices:

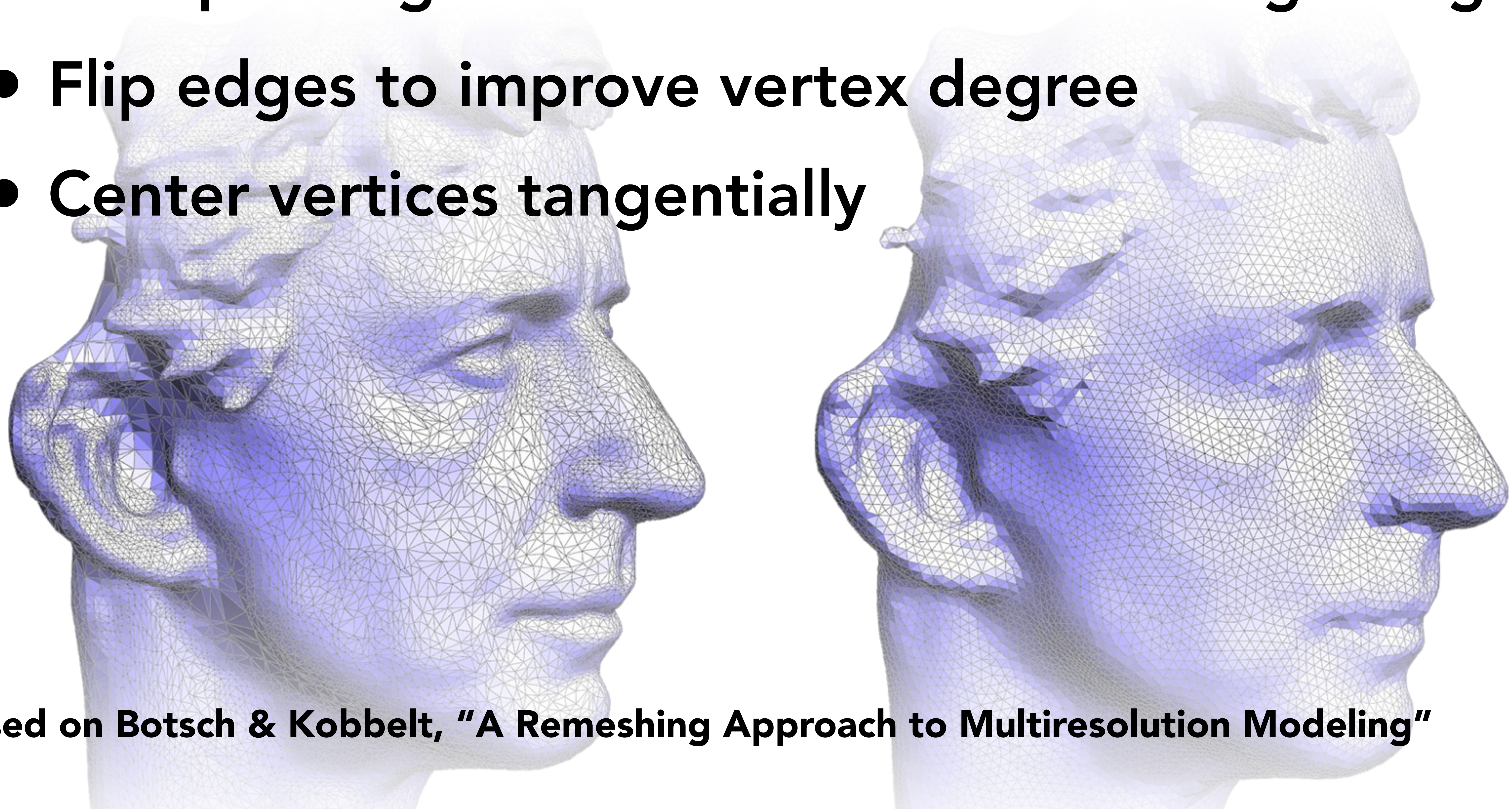


[Crane, "Digital Geometry Processing with Discrete Exterior Calculus"]

Isotropic Remeshing Algorithm*

Repeat four steps:

- Split edges over 4/3rds mean edge legth
- Collapse edges 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”

Things to Remember

Triangle mesh representations

- Triangles vs points+triangles
- Half-edge structure for mesh traversal and editing

Geometry processing basics

- Local operations: flip, split, and collapse edges
- Upsampling by subdivision (Loop, Catmull-Clark)
- Downsampling by simplification (Quadric error)
- Regularization by isotropic remeshing

Acknowledgments

This slide set contain contributions from:

- Kayvon Fatahalian
- David Forsyth
- Pat Hanrahan
- Angjoo Kanazawa
- Steve Marschner
- Ren Ng
- James O'Brien
- Mark Pauly