10 Efficient mesh models

Follows chapter 16 in RTR 4e

Steve Marschner

CS5625 Spring 2019

Basics of efficiency for meshes

Use triangle or quad meshes

- general polygon meshes lead to too much complexity
- quad meshes are great for some applications but more constrained

Use shared-vertex triangle meshes for GPU applications

- major memory/bandwidth savings over separate triangles
- if you get separate triangles, merge them in a pre-process

Store most data at vertices

- there are ~half as many vertices as faces
- vertex data may be interpolated across faces
- in typical GPU mesh representation, vertices must be duplicated to create discontinuities

More sophistication in mesh storage

Optimizing vertex order

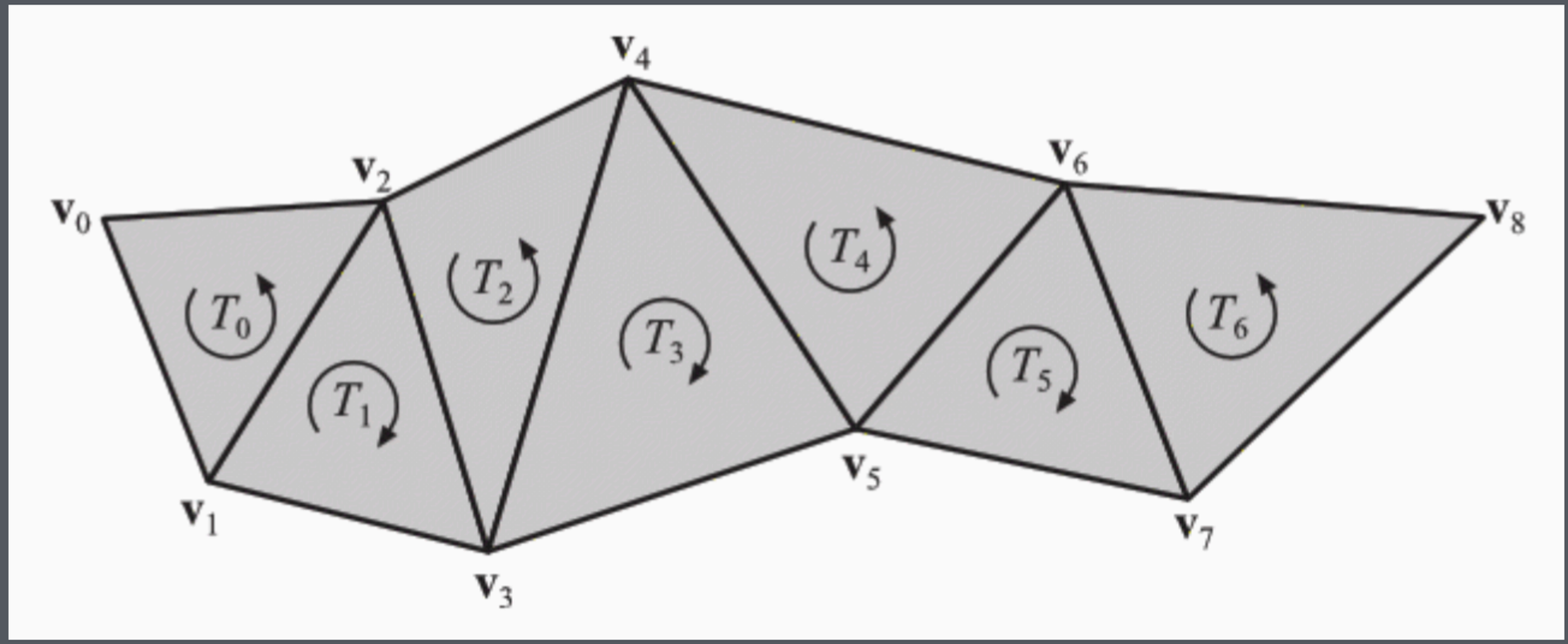
- strips and fans as examples (when per-frame bandwidth was the concern)
- modern systems don’t use these but optimize for hit rate in vertex cache

Reducing the number of triangles

- ultimately this is needed to save more time and space
- many *levels of detail* are useful
  - simpler meshes for faraway objects
  - simpler meshes for lower-resolution screens
  - simpler meshes for lower-performance hardware or networks

Vertex cache and mesh ordering

Triangle strips gain efficiency by caching the most recent two vertices



- we are essentially using a FIFO cache policy with a size of 2
- cache miss rate approaches 1 miss / triangle

Optimizing for larger caches

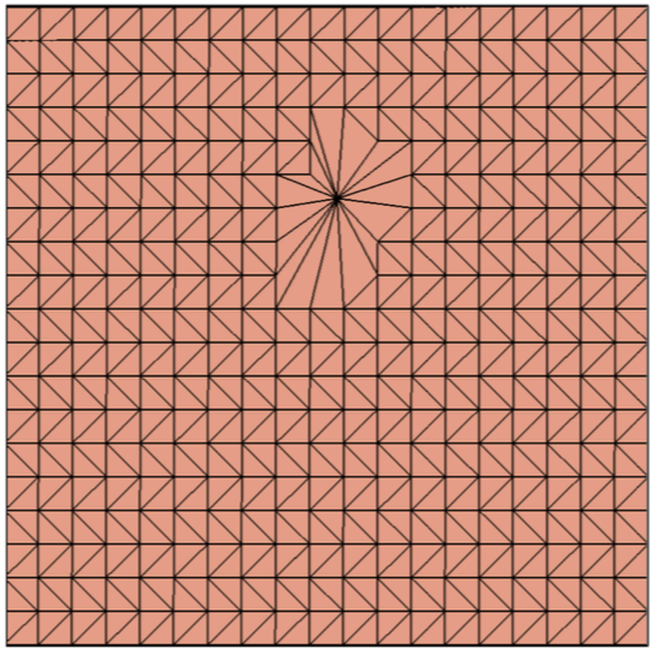
With indexed meshes, saving indices is less important

- we store lots of data at vertices; ~6 indices is the least of our worries
- just putting meshes in triangle-strip order gives you the same vertex caching behavior ("transparent" vertex caching)

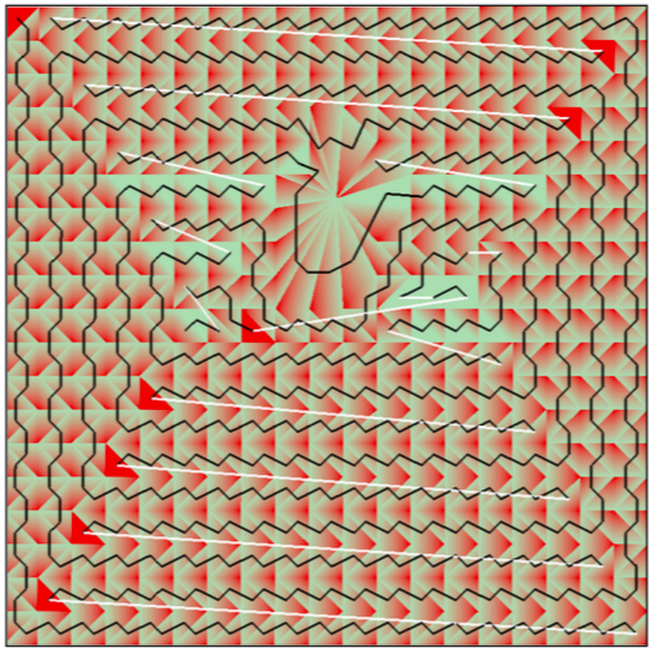
Newer systems are built with post-transform vertex caches

- cache the results of the vertex processing stage
- cache hits can save substantial computation

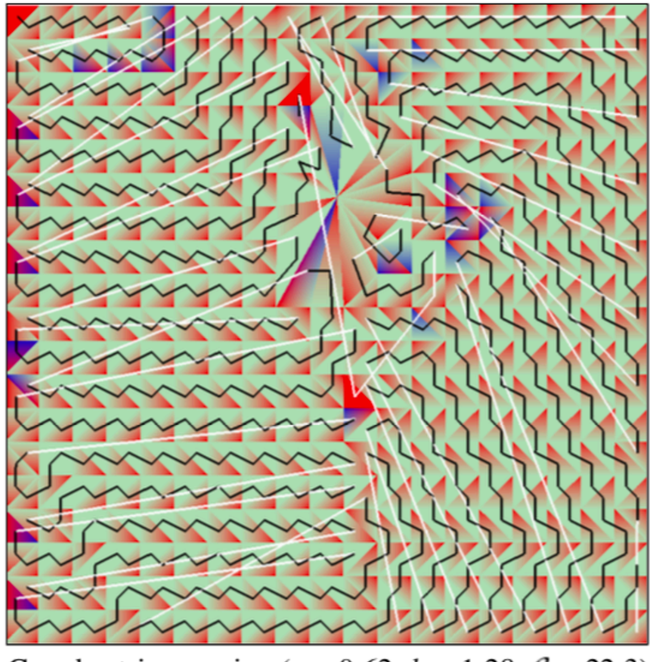
As with other applications of caches, now order of data access matters



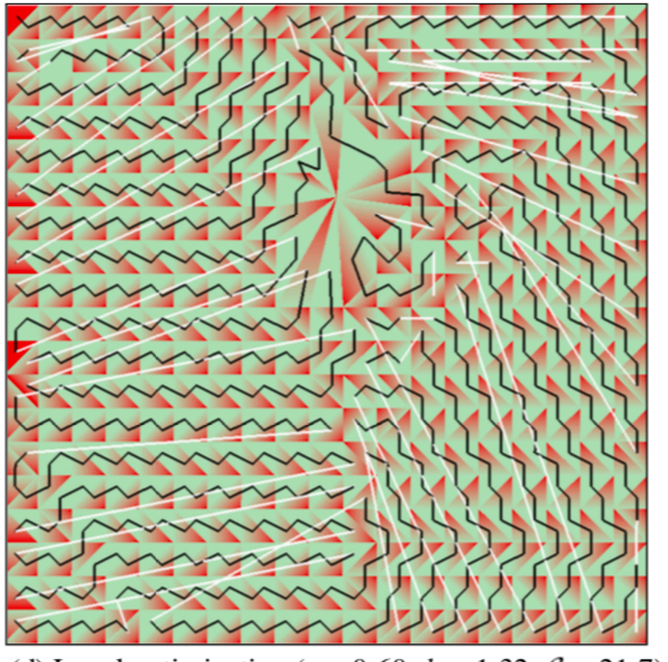
(a) Original mesh (704 faces)



(b) Traditional strips ($r = 0.99$; $b = 1.29$; $\mathcal{C} = 34.3$)



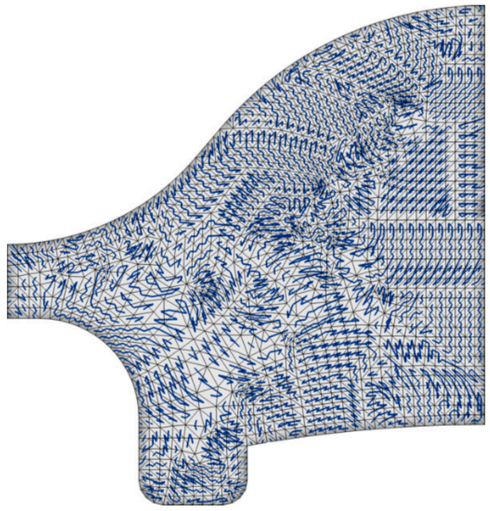
(c) Greedy strip-growing ($r = 0.62$; $b = 1.28$; $\mathcal{C} = 22.3$)



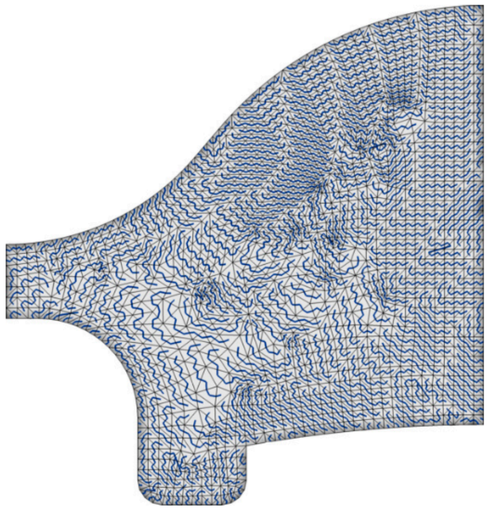
(d) Local optimization ($r = 0.60$; $b = 1.32$; $\mathcal{C} = 21.7$)

r is the average cache miss rate

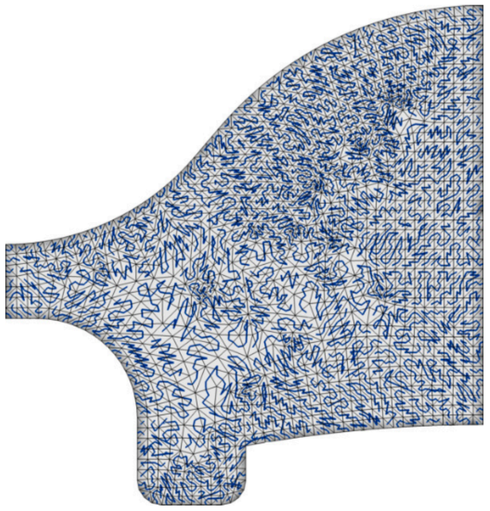
[Hoppe 1999]



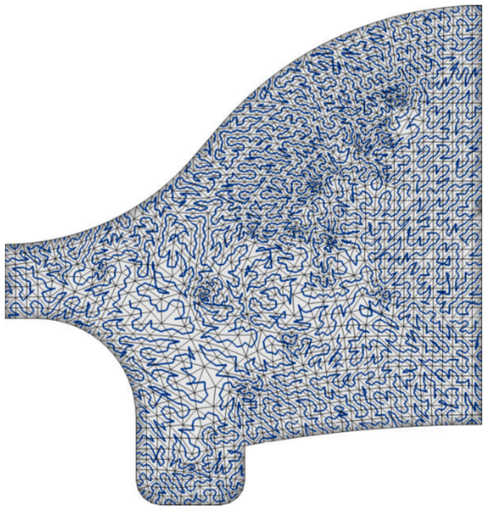
(a) K-Cache-Reorder
Lin and Yu [2006]



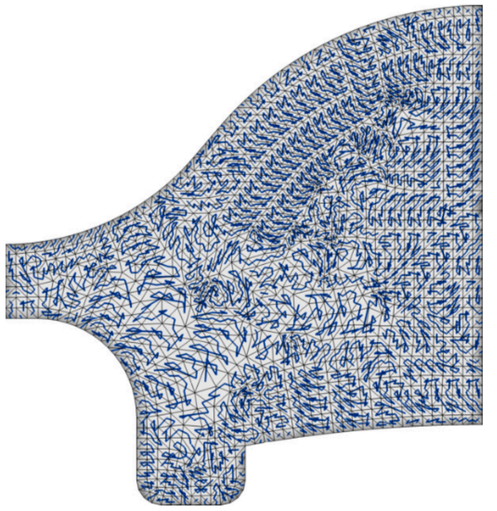
(b) D3DXMesh
based on [Hoppe 1999]



(c) OpenCCL
Yoon and Lindstrom [2006]



(d) dfsrendseq
Bogomjakov et al. [2001]



(e) Our work

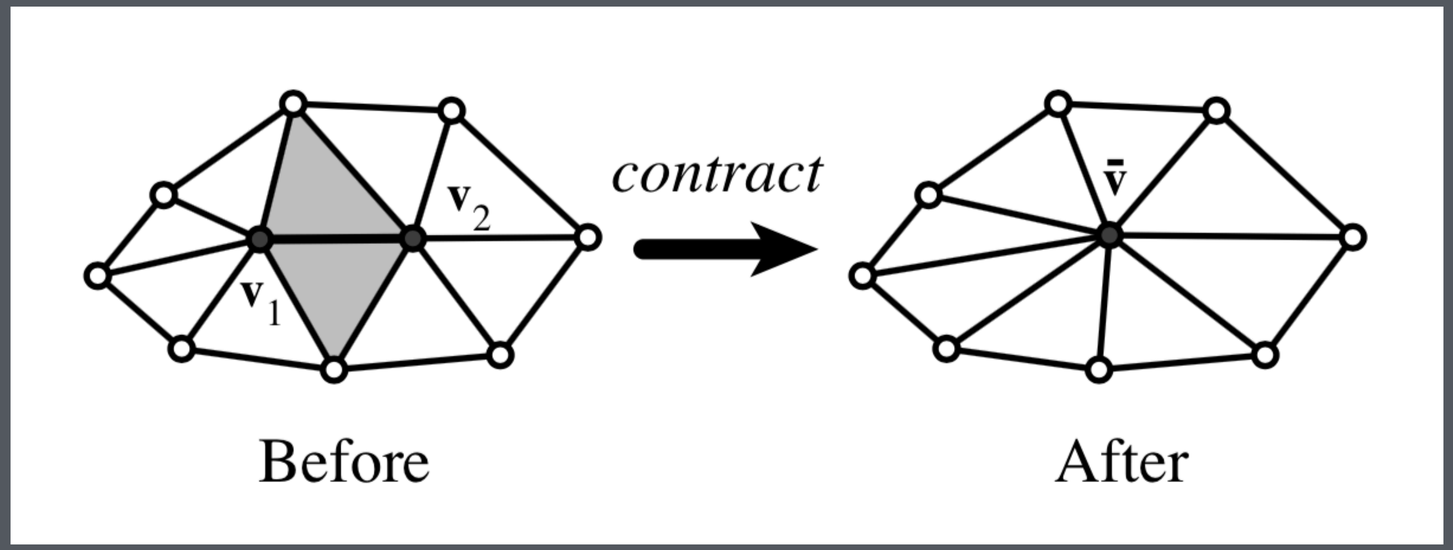
[Sander et al. 2007 “Fast Triangle Reordering for Vertex Locality and Reduced Overdraw”]

Mesh simplification

Many ways to simplify meshes

- remove chunks, retriangulate hole
- quantize vertices to centers of voxels

Particularly simple and effective is edge collapses, or edge contractions:



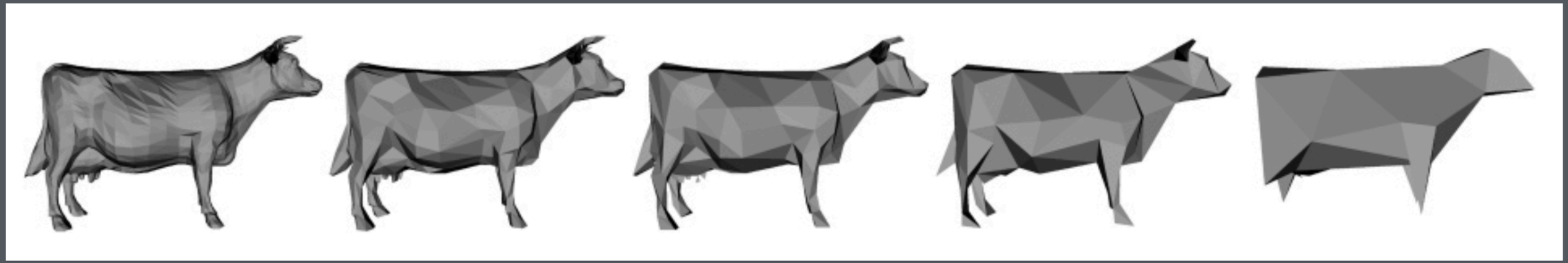
Quadric Error Metric

Edge-collapse simplification produces a sequence of meshes

- each mesh has one fewer face
- each is derived from the previous by a single edge collapse

Key question: where to put the vertex after the collapse?

- at first vertex? at second? at midpoint?
- can choose location as the solution to an optimization



Where to put the new vertex?

It depends on the mesh geometry:

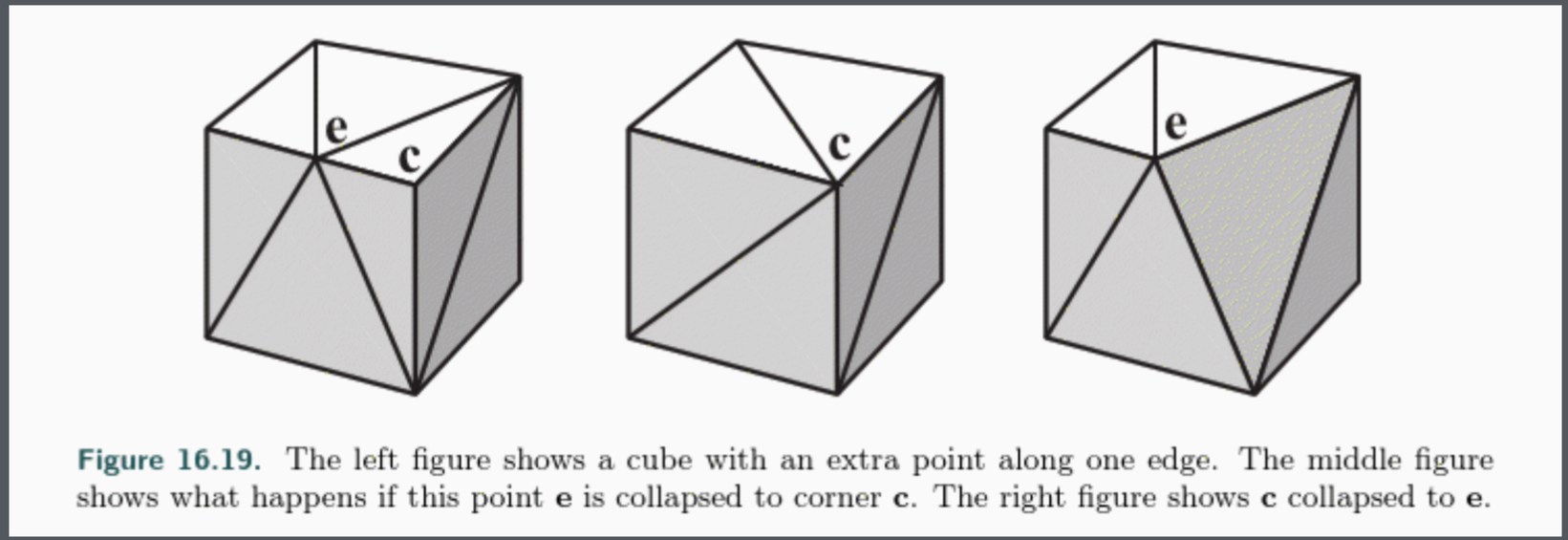


Figure 16.19. The left figure shows a cube with an extra point along one edge. The middle figure shows what happens if this point **e** is collapsed to corner **c**. The right figure shows **c** collapsed to **e**.

- one way to formalize: the new vertex should be close to the planes of the triangles around it before the edge collapse

Garland & Heckbert QEM

A particularly convenient error metric: sum of squared distances to planes

- each plane has an equation, can be represented as a 4-vector (*a*, *b*, *c*, *d*)
  with (*a*, *b*, *c*) components normalized
- distance of a vertex **v** from the plane **p** is then the inner product $\mathbf{p}^T\mathbf{v}$
- squared distance from plane is in the form $\mathbf{v}^T\mathbf{M}\mathbf{v}$ for a 4x4 **M** (a quadric)

$$\begin{aligned}\Delta(\mathbf{v}) &= \sum_{\mathbf{p}\in\text{planes}(\mathbf{v})} (\mathbf{v}^T\mathbf{p})(\mathbf{p}^T\mathbf{v}) \\ &= \sum_{\mathbf{p}\in\text{planes}(\mathbf{v})} \mathbf{v}^T(\mathbf{p}\mathbf{p}^T)\mathbf{v} \\ &= \mathbf{v}^T\left(\sum_{\mathbf{p}\in\text{planes}(\mathbf{v})} \mathbf{K}_\mathbf{p}\right)\mathbf{v}\end{aligned}$$

- and better yet, the sum-squared distance from several planes is still in the form $\mathbf{v}^T\mathbf{Q}\mathbf{v}$

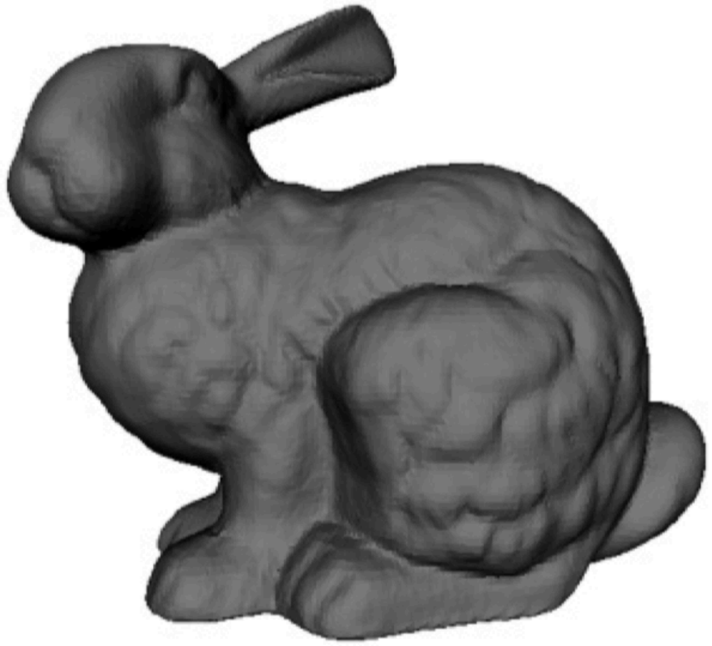
QEM simplification

With the error in the form of a quadric per vertex:

- the matrix is easy to compute from the surrounding triangles
- the error is easy to optimize. Given $\mathbf{Q}_1$ and $\mathbf{Q}_2$ belonging to a pair of vertices $\mathbf{v}_1$ and $\mathbf{v}_2$, we simply sum the errors of the two vertices:

$$\begin{aligned}\Delta(\mathbf{v}) &= \Delta_1(\mathbf{v}) + \Delta_2(\mathbf{v}) \\ &= \mathbf{v}^T\mathbf{Q}_1\mathbf{v} + \mathbf{v}^T\mathbf{Q}_2\mathbf{v} \\ &= \mathbf{v}^T(\mathbf{Q}_1 + \mathbf{Q}_2)\mathbf{v}\end{aligned}$$

- minimizing this error is a 4x4 linear system—very fast
- algorithm
  - 0. compute **Q**s for all vertices, compute errors for all potential edge collapses.
  - 1. use priority queue to find smallest-error edge. Collapse it; update the neighboring **Q**s.
  - 2. repeat until mesh is small enough!



69k faces

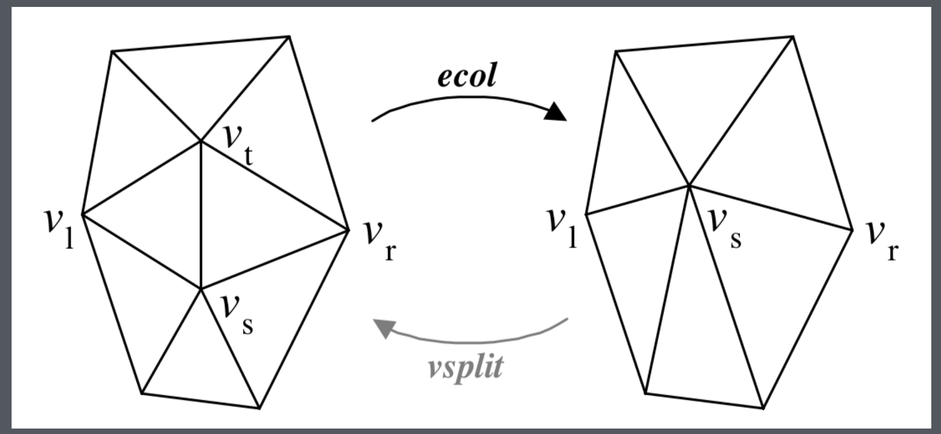
1k faces

surfaces of constant cost for reducing to 999 faces

[Garland & Heckbert 1997 “Surface Simplification Using Quadric Error Metrics”]

Continuous level-of-detail: Progressive Meshes

Key observation: edge collapse is invertible



- just need to store (offsets to) the locations of the two new vertices

Thus a sequence of edge collapses, reversed, is a representation for a mesh

$$(\hat{M}=M^n) \xrightarrow{ecol_{n-1}} \dots \xrightarrow{ecol_1} M^1 \xrightarrow{ecol_0} M^0 .$$

$$M^0 \xrightarrow{vsplit_0} M^1 \xrightarrow{vsplit_1} \dots \xrightarrow{vsplit_{n-1}} (M^n=\hat{M})$$

[Hoppe 1996
“Progressive Meshes”]

Progressive Meshes

Store full representation, load various levels of detail

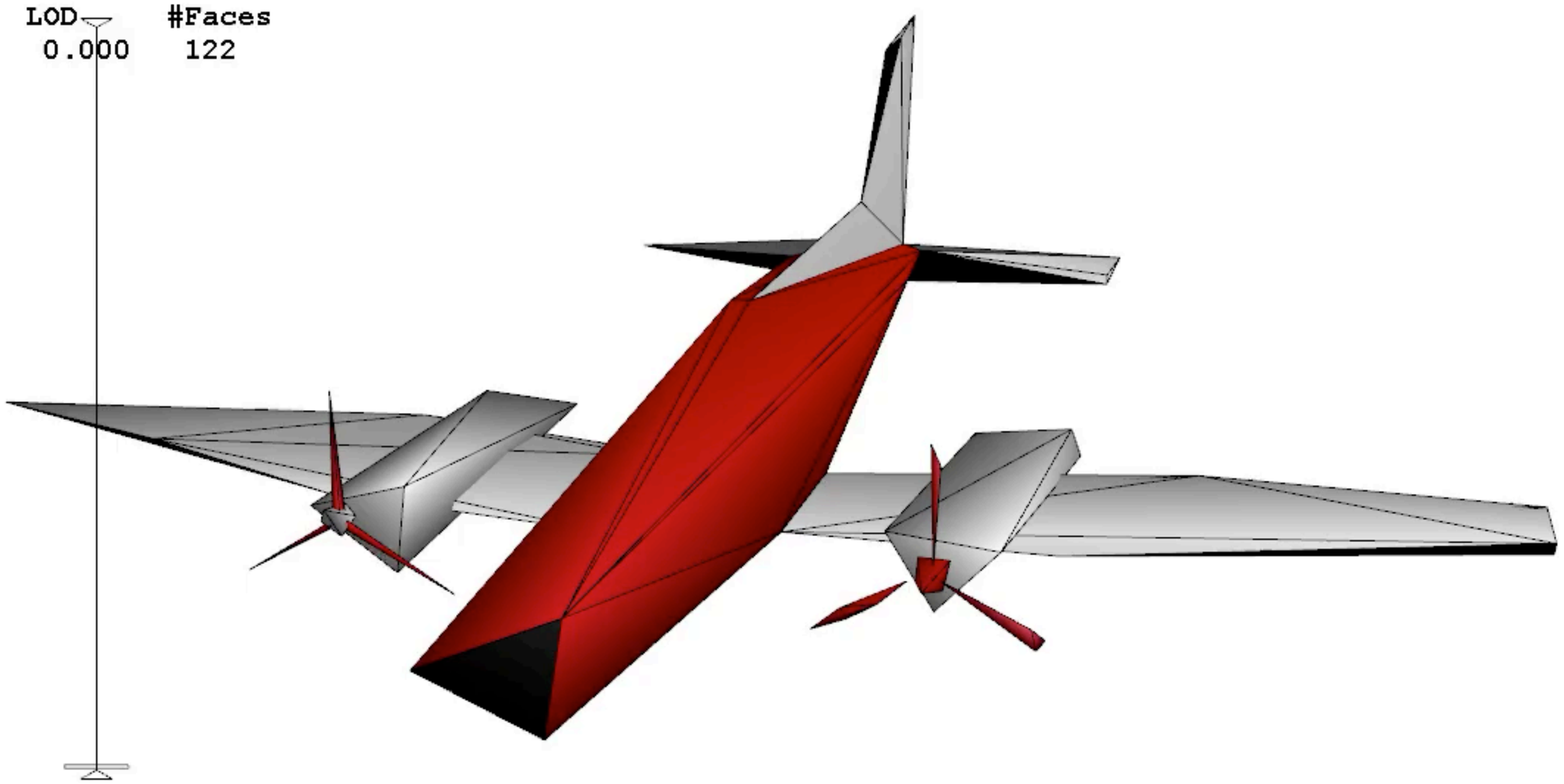
- just load or transmit a prefix of the list of edge splits
- can change level of detail smoothly depending on size/distance/salience/etc.

Can interpolate (“geomorphs”)

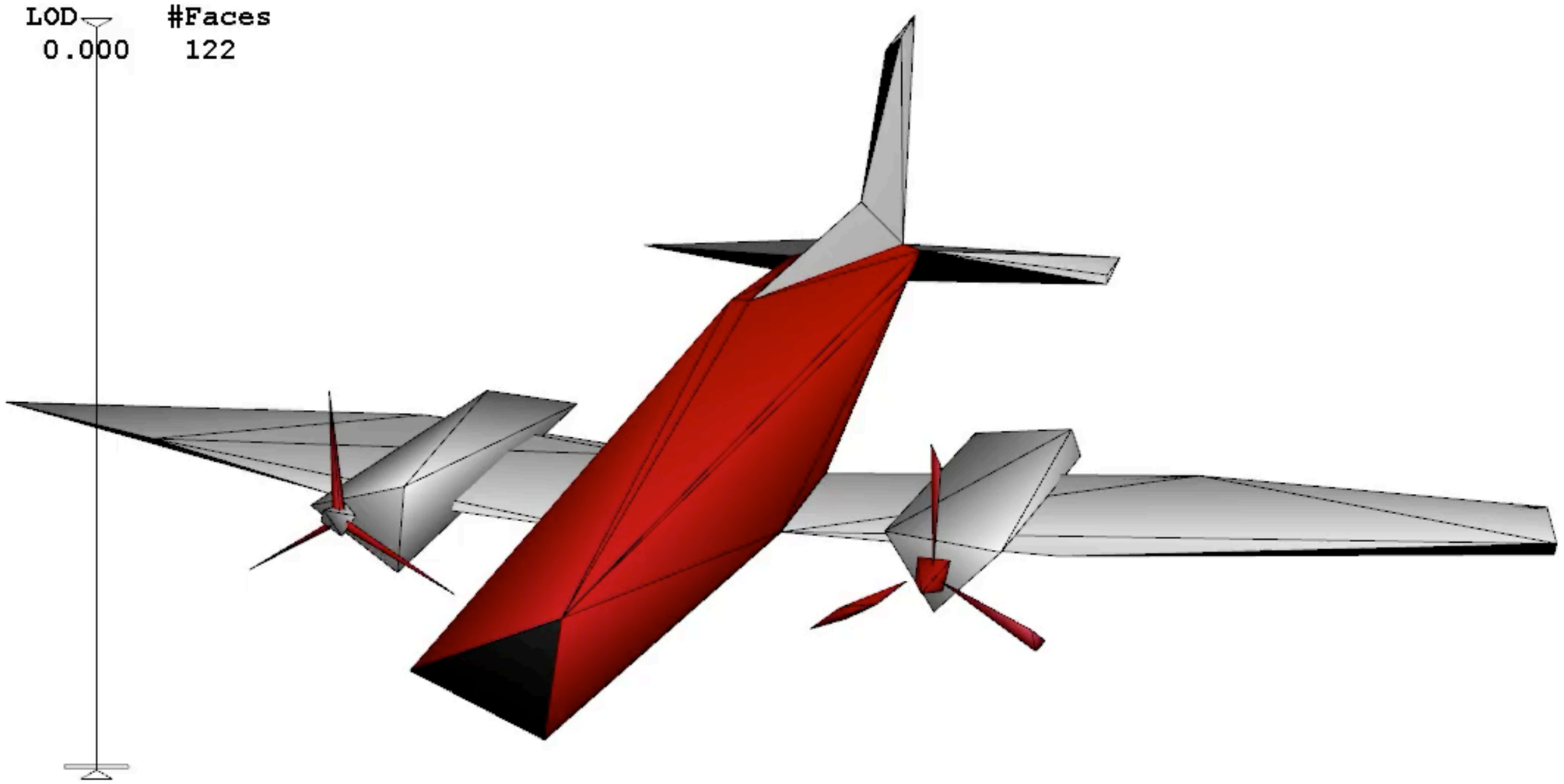
- sudden edge splits/collapses are jarring
- interpolate new vertices from merged position to new positions
- leads to truly continuous LoD

Extra details (of QEM and PM)

- boundaries, creases—want to preserve them
- merging of small pieces—otherwise can’t simplify enough
- maintenance of additional attributes—throw them in the metric too



[Hoppe 1996 “Progressive Meshes”]



[Hoppe 1996 “Progressive Meshes”]