

# IFT 6112

# APPLICATIONS OF

# CURVATURE

<http://www-labs.iro.umontreal.ca/~bmpix/teaching/6112/2018/>



Mikhail Bessmeltsev

# **MESH SIMPLIFICATION**



Stone Age Tools

Image from Wikipedia

# MESH SIMPLIFICATION

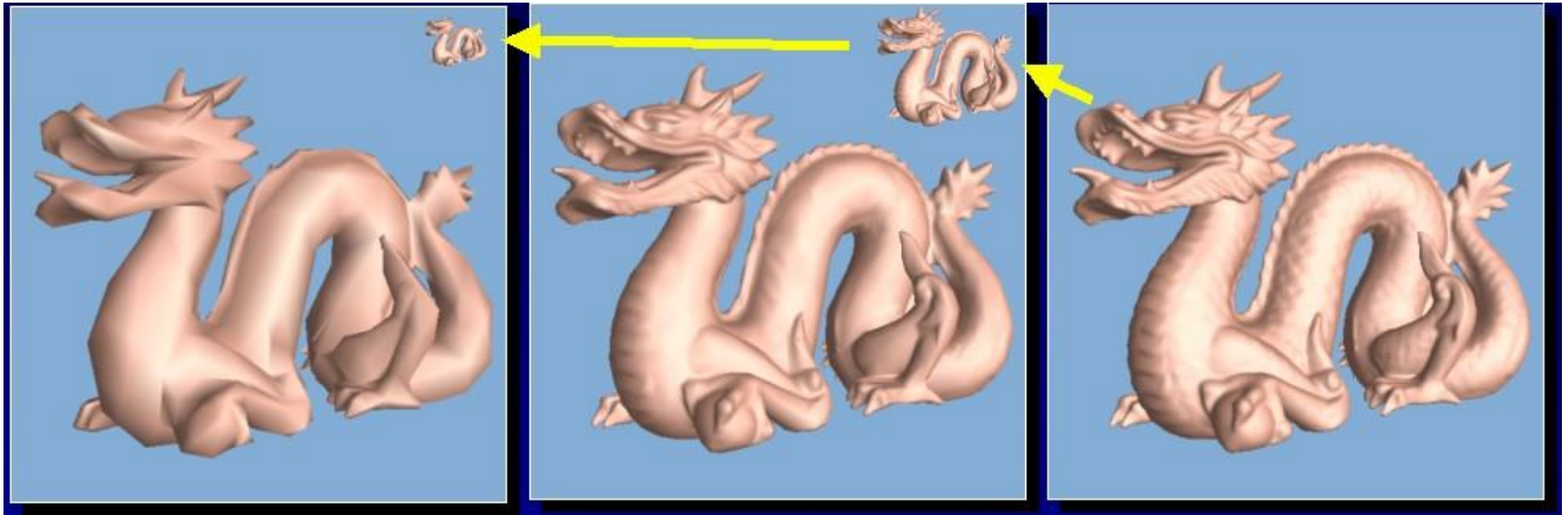


# Why simplify meshes?

- Faster rendering
- Faster collision detection
- Storage/transmission/etc.

# Why simplify meshes?

- Faster rendering: Level of Detail

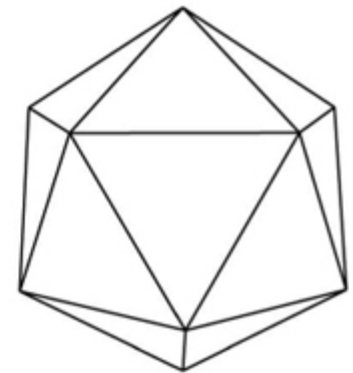
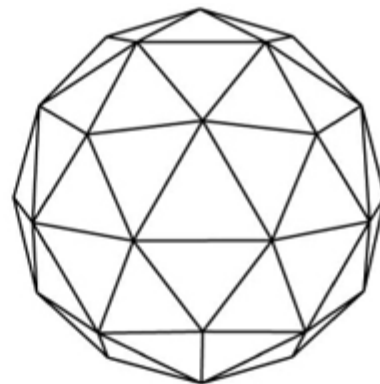
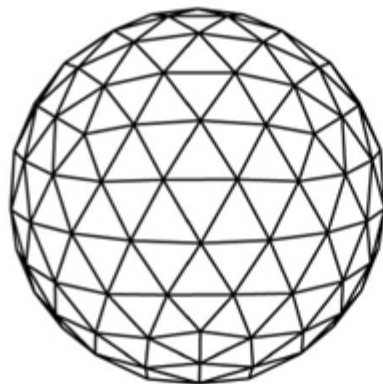
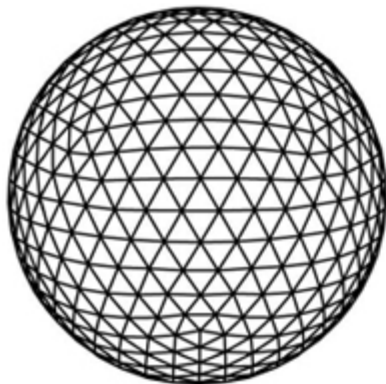


Rough, if the mesh is far

Fine, if the mesh is close

# Requirements

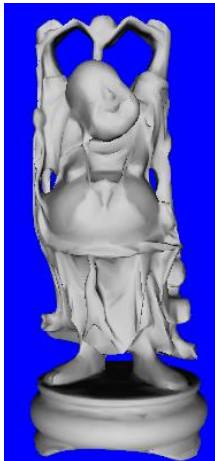
- Reduce # of polygons
- Preserve the shape
  - Geometry
    - Features
  - Topology
  - Other constraints?



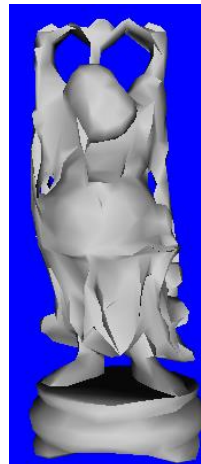
# Complexities

Orientable 2-manifolds with

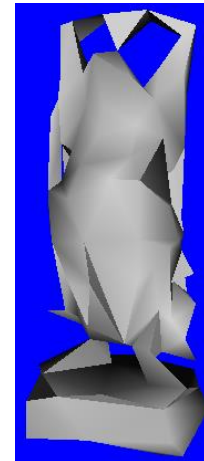
- Arbitrary genus
- Arbitrary # of connected components
- With/without boundaries



12,000



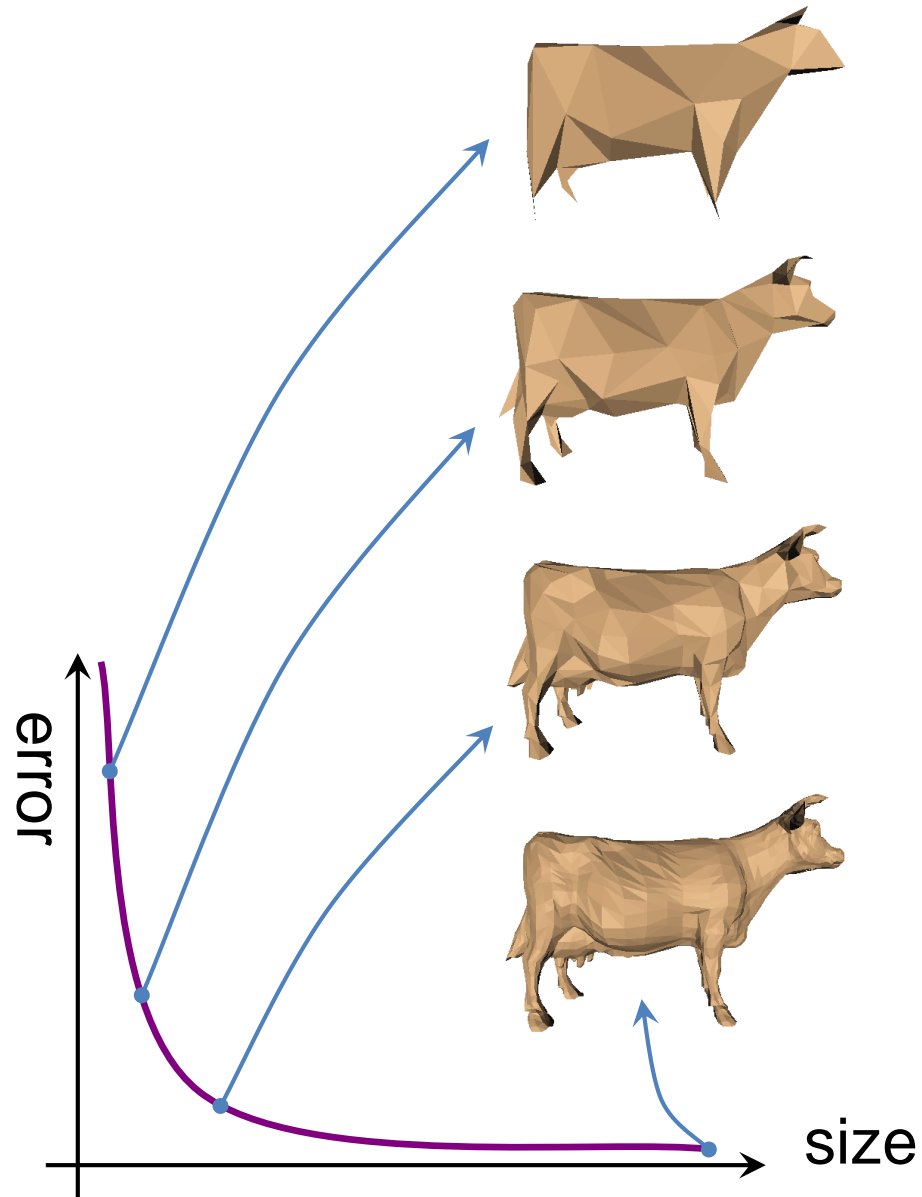
2,000



300



# Approximation error

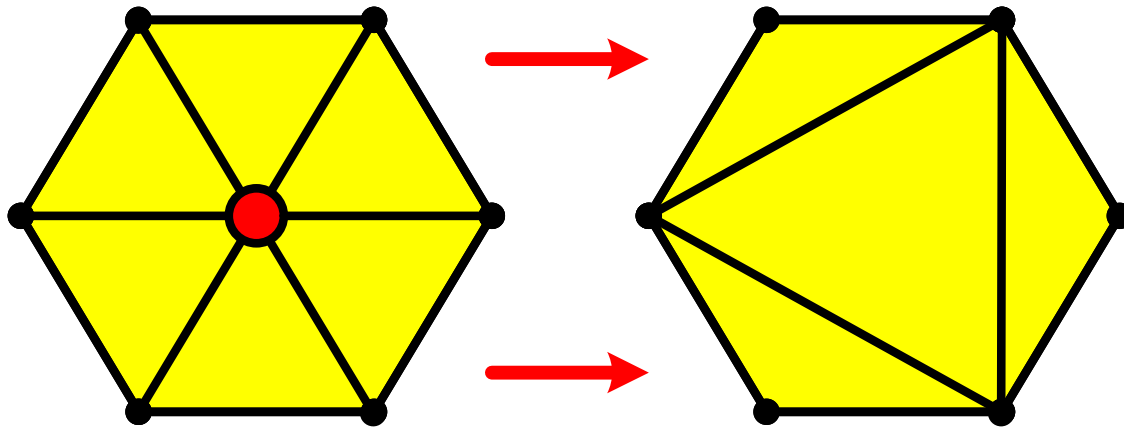


# Approach

- Local operations
  - Removing edges or vertices
  - Keeping track of the geometry/topology
  
- Keep score
  - how much will we lose if we remove a vertex?

# Approach

- Each operation introduces error
- Quantify?

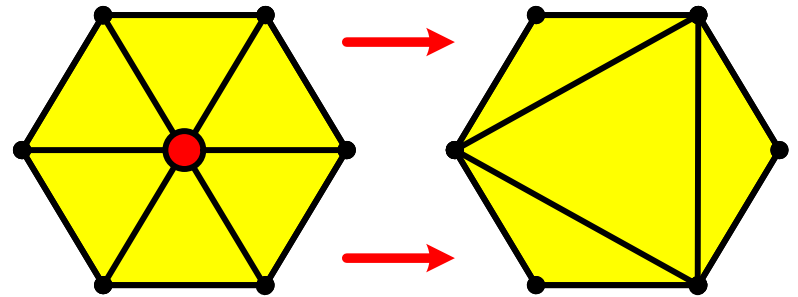


# Decimation

- Vertex removal

$$\#V \rightarrow \#V - 1$$

$$\#F \rightarrow \#F - 2$$



Remove vertex  $\rightarrow$  hole  $\rightarrow$  triangulate

# Decimation

- Vertex removal

$$\#V \rightarrow \#V - 1$$

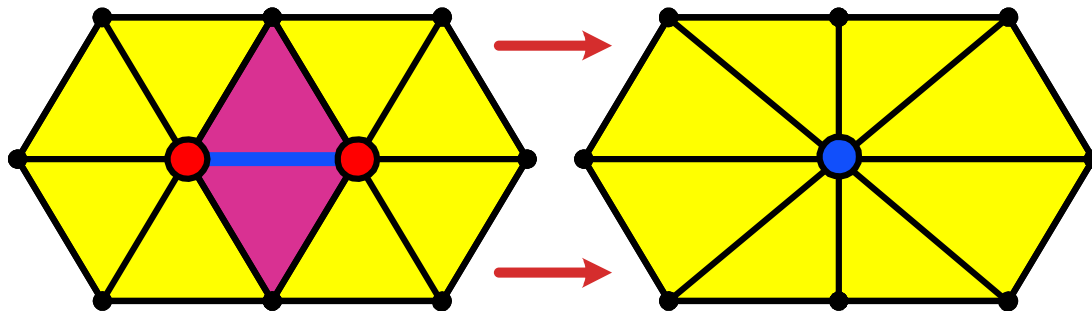
$$\#F \rightarrow \#F - 2$$

# Decimation

- Edge collapse

$$\#V \rightarrow \#V - 1$$

$$\#F \rightarrow \#F - 2$$

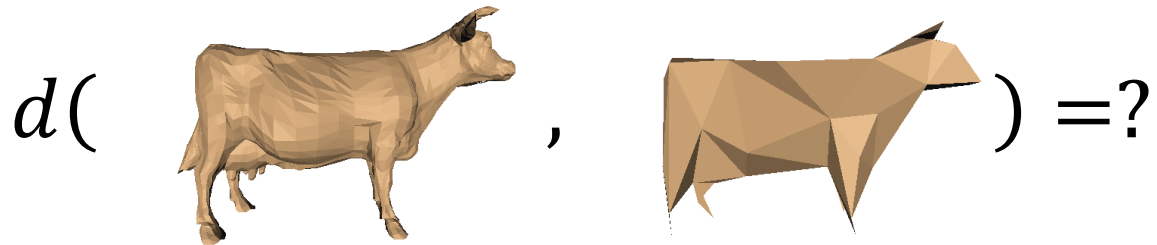


# Measuring error

$$d(\text{[smooth cow]}, \text{[low-poly cow]}) = ?$$

How much did we distort the shape?

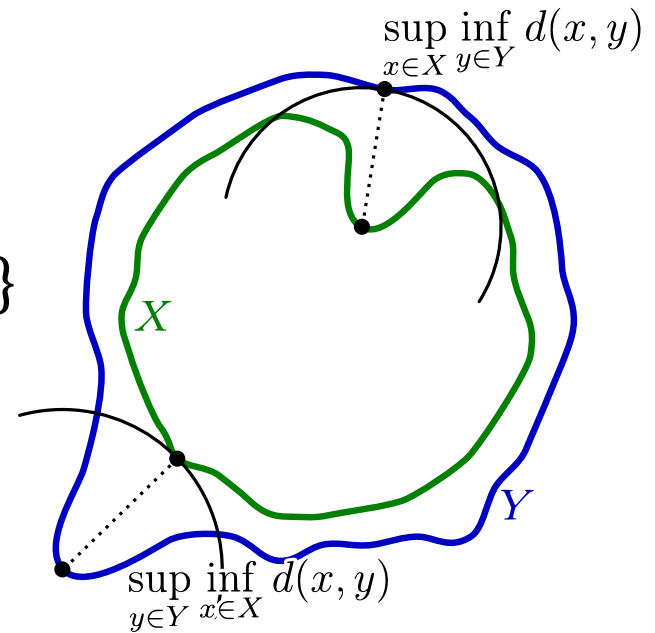
# Measuring error



How much did we distort the shape?

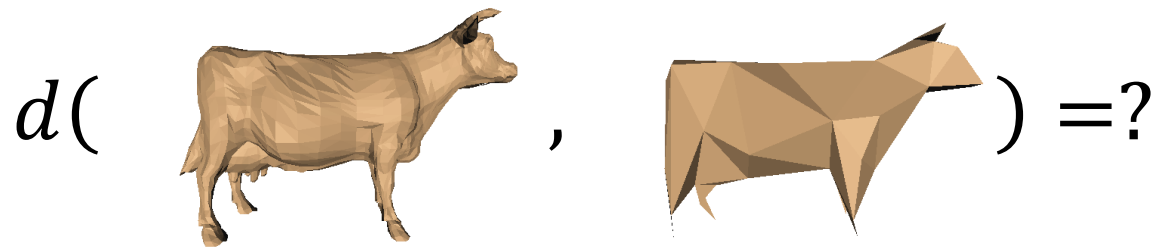
Option: Hausdorff distance

$$d_H(x,y) = \max\left\{\sup_{x \in X} \inf_{y \in Y} d(x,y), \sup_{y \in Y} \inf_{x \in X} d(x,y)\right\}$$





# Measuring error

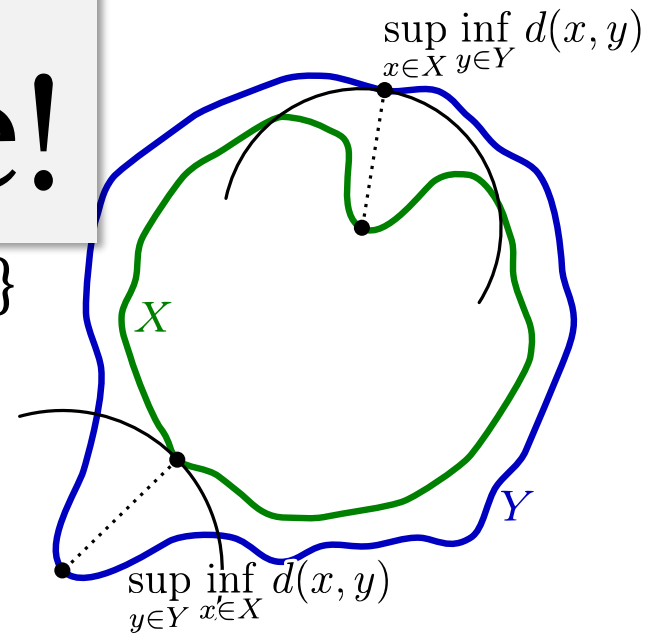


How to compare the shape?

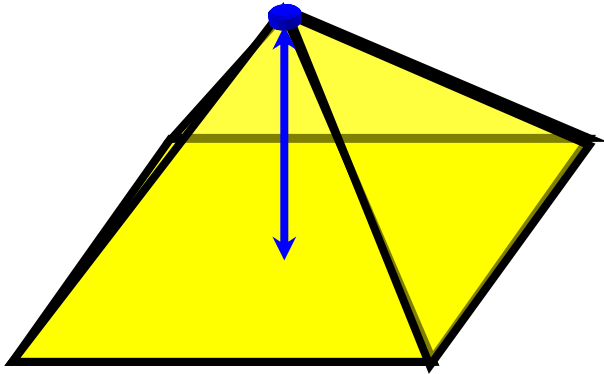
Option: Hausdorff

Hard to compute!

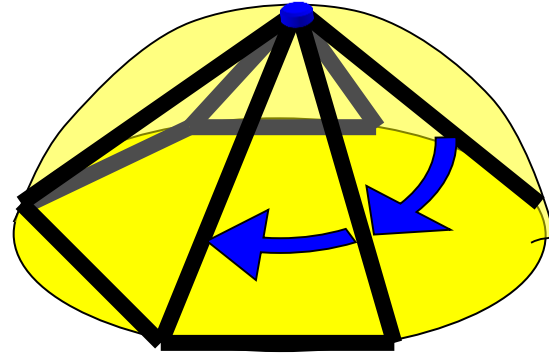
$$d_H(x, y) = \max\left\{\sup_{x \in X} \inf_{y \in Y} d(x, y), \sup_{y \in Y} \inf_{x \in X} d(x, y)\right\}$$



# Local Error Control



Vertex-plane distance



$$2\pi - \sum \alpha_i$$

Curvature

# Algorithm

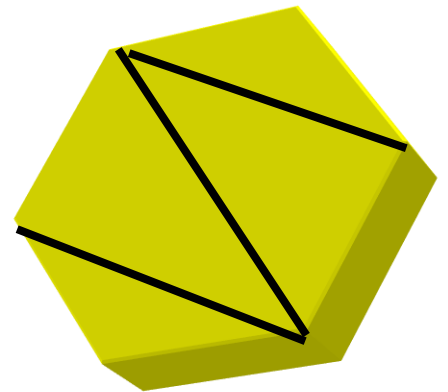
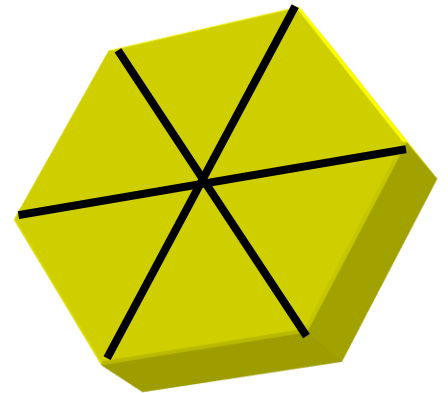
- Repeat
  - Select the element with minimal error
  - Perform simplification operation (remove/contract)
  - Update error (local/global)
  - Preserve structure
- Until mesh size / quality is achieved

# Data Structures

- Easy access to neighbors
  - Filling holes
  - Computing cost
- Priority queue/heap
  - Quick access to the cheapest element

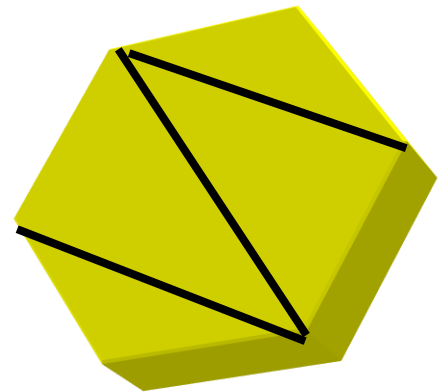
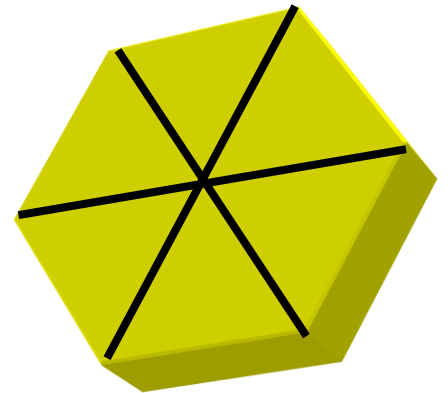
# Vertex Removal Algorithm

- Simplification operation: Vertex removal
- Error metric: Distance to average plane
- May preserve mesh features (creases)

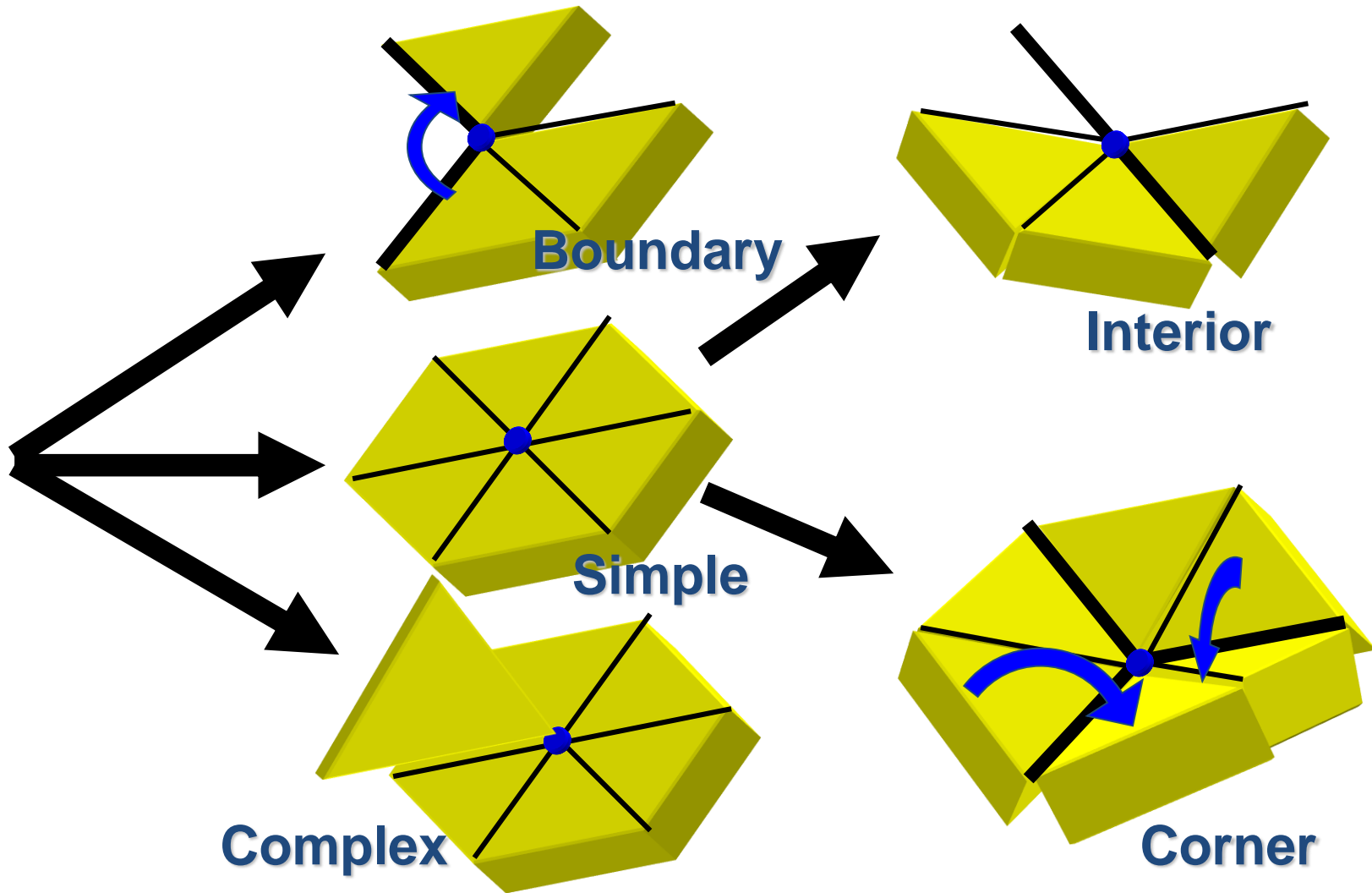


# Vertex Removal Algorithm

- Characterize local topology/geometry
- Classify vertices as removable or not
- Repeat
  - Remove vertex
  - Triangulate resulting hole
  - Update error of affected vertices
- Until reduction goal is met

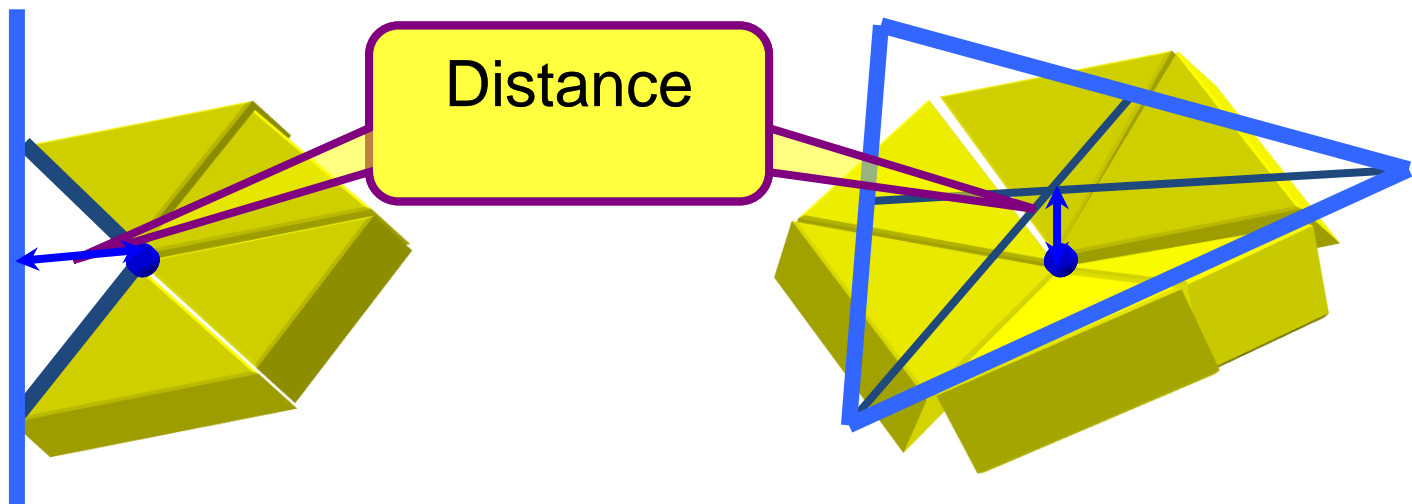


# Characterizing structure



# Decimation Criterion

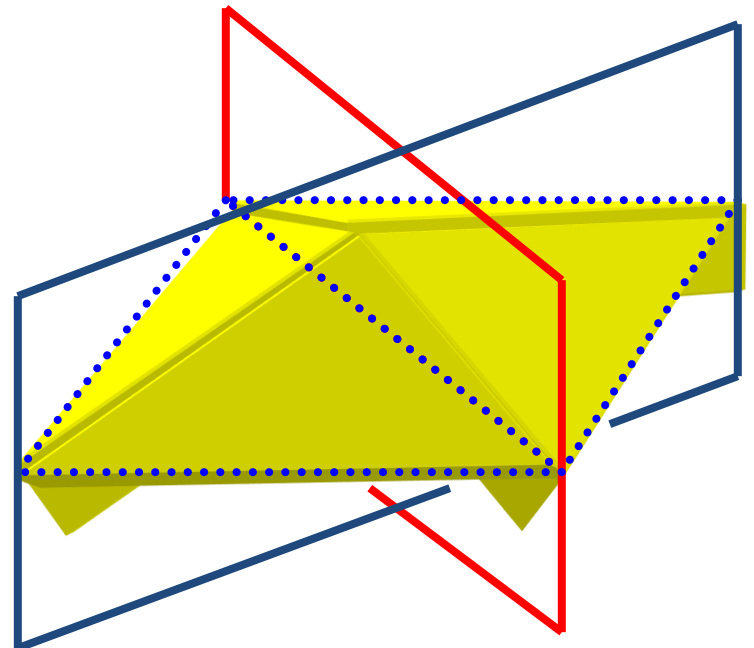
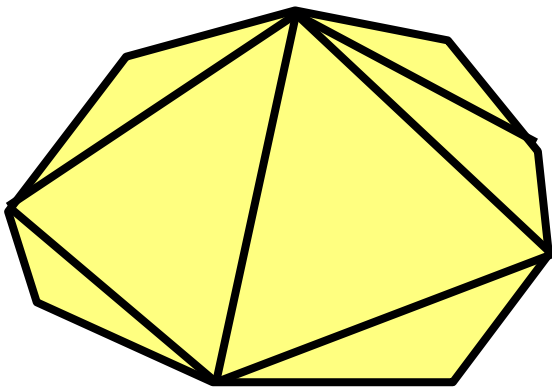
- $E_{MAX}$  – user defined parameter
- Simple vertex:
  - Distance of vertex to the face loop average plane  $< E_{MAX}$
- Boundary vertices:
  - Distance of the vertex to the new boundary edge  $< E_{MAX}$



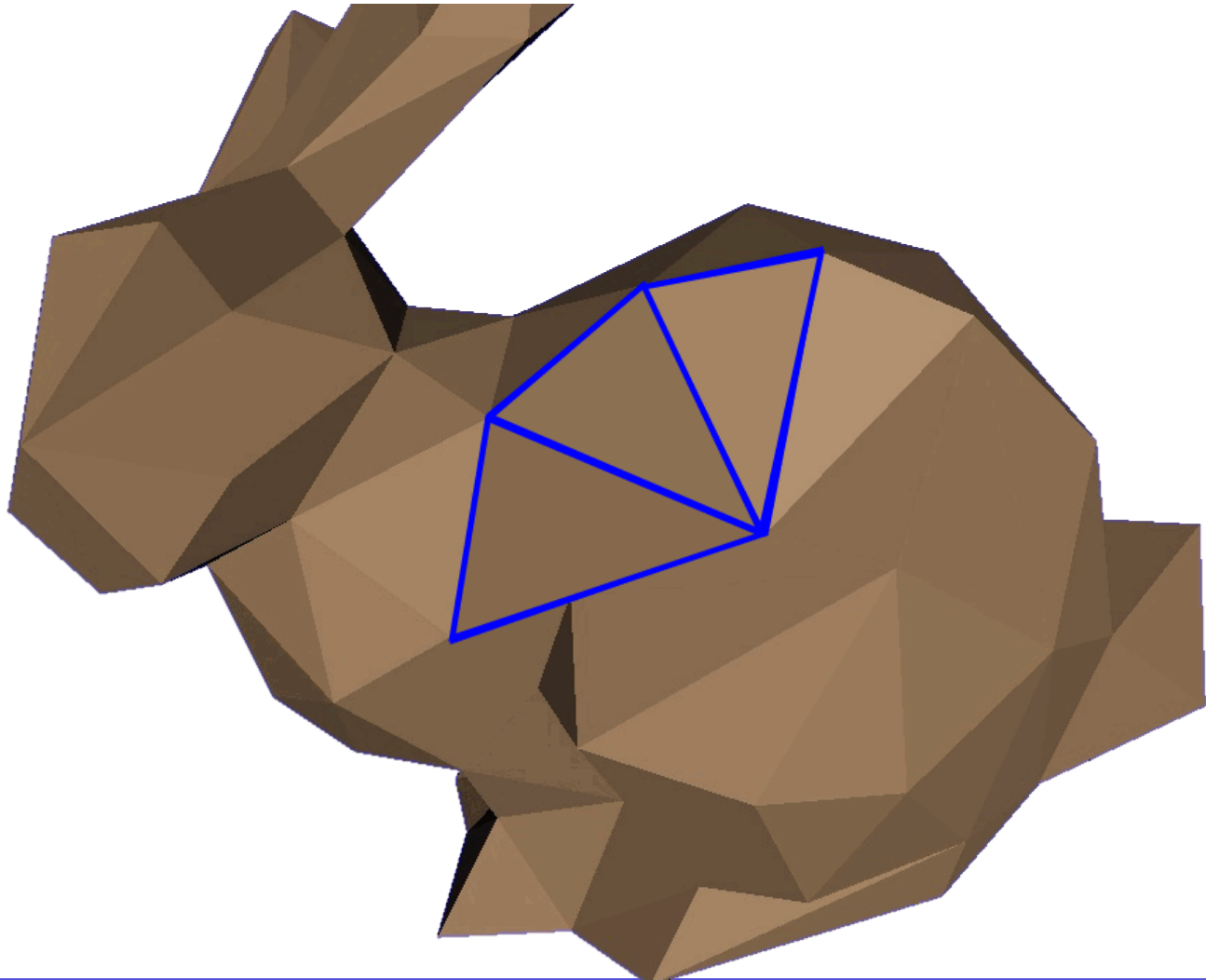


# Triangulating the Hole

- Vertex removal produces non-planar loop
  - Split loop recursively
  - Split plane orthogonal to the average plane
- Control aspect ratio
- Triangulation may fail
  - Vertex is not removed



# Example



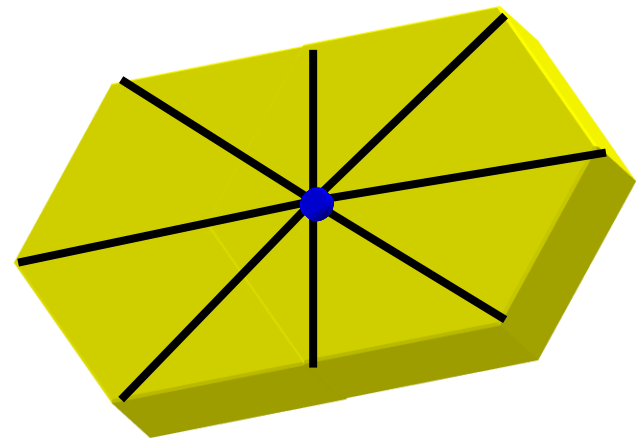
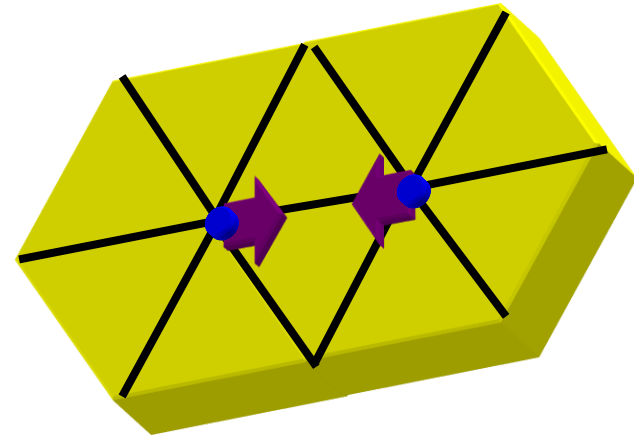
Simplifier

# Pros and Cons

- Pros:
  - Efficient
  - Simple to implement and use
    - Few input parameters to control quality
  - Reasonable approximation
  - Works on very large meshes
  - Preserves topology
  - Vertices are a subset of the original mesh
- Cons:
  - Error is not bounded
    - Local error evaluation causes error to accumulate

# Edge Collapse Algorithm

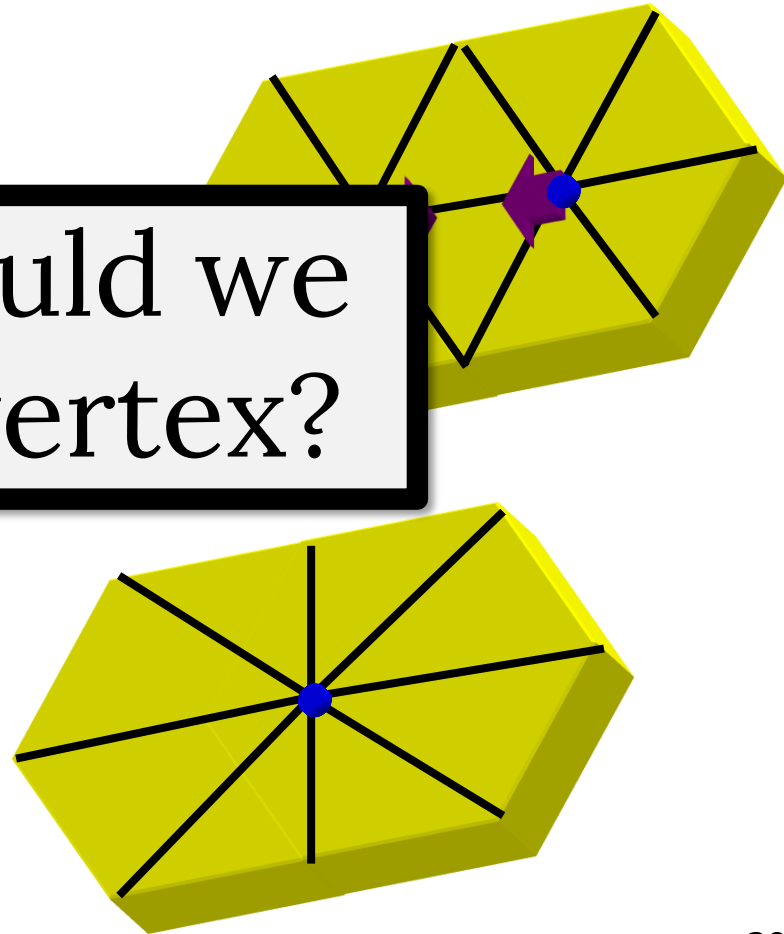
- Simplification operation:
  - Edge collapse (pair contraction)
- Error metric:  
distance, pseudo-global



# Edge Collapse Algorithm

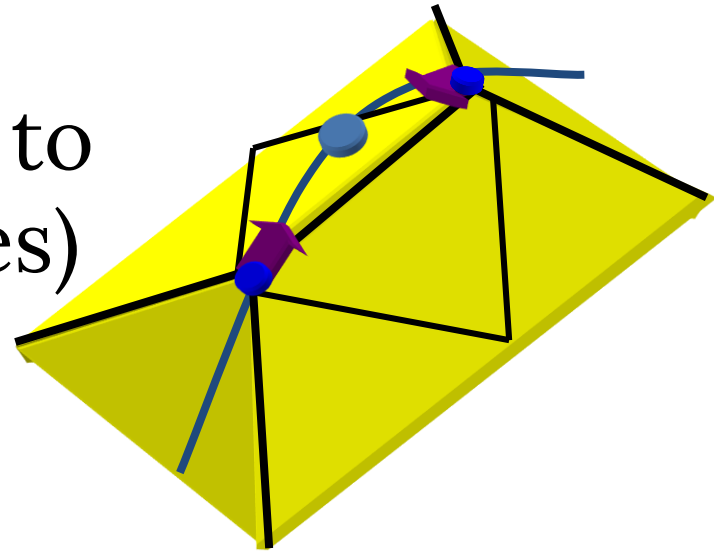
- Simplification operation:
  - Edge collapse
- Error metric: distance, pseudo-global

Where should we place the vertex?



# Distance Metric: Quadrics

- Choose point closest to set of planes (triangles)



Sum of squared distances to set of planes is quadratic  $\Rightarrow$  convex

# Quadrics

- Plane

$$Ax + By + Cz + D = 0$$

$$p = [A, B, C, D], v = [x, y, z, 1], v p^T = 0$$

- Quadratic distance between  $v$  and  $p$ :

$$d_p(v) = (v p^T)^2$$

$$= (v p^T) (p v^T) = v (p^T p) v^T$$

$$= v K_P v^T$$

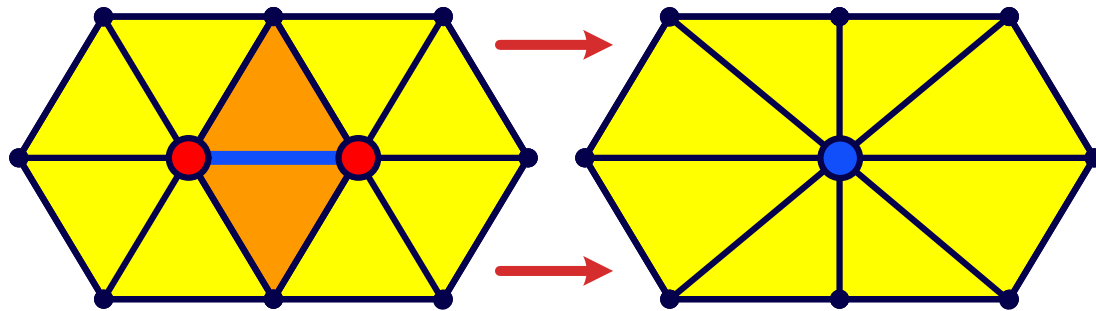
$$K_P = \begin{bmatrix} A^2 & AB & AC & AD \\ AB & B^2 & BC & BD \\ AC & BC & C^2 & CD \\ AD & BD & CD & D^2 \end{bmatrix}$$

# Distance to Set of Planes

$$d_M(v) = \sum_{p \in \text{planes}(v)} d_p(v) = \sum_{p \in \text{planes}(v)} v^T K_p v = v^T \left( \sum_{p \in \text{planes}(v)} K_p \right) v = v^T Q_v v^T$$

After contracting edge between  $v_1$  and  $v_2$

- place vertex at the minimum of  $d_M(v)$
- set  $Q_v = Q_{v_1} + Q_{v_2}$





# Contracting Two Vertices

- Goal: Given edge  $e = (v_1, v_2)$ , find contracted

$$v = (x, y, z, 1) \text{ that minimizes } d(v):$$
$$\nabla_v d = 0$$

- Solve system of linear normal equations:

$$\begin{bmatrix} q_{11} & q_{12} & q_{13} & q_{14} \\ q_{21} & q_{22} & q_{23} & q_{24} \\ q_{31} & q_{32} & q_{33} & q_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} v = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

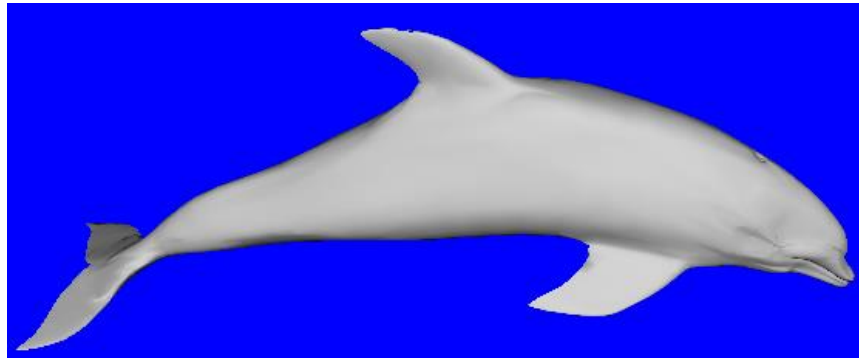
- If no solution - select the edge midpoint

# Algorithm

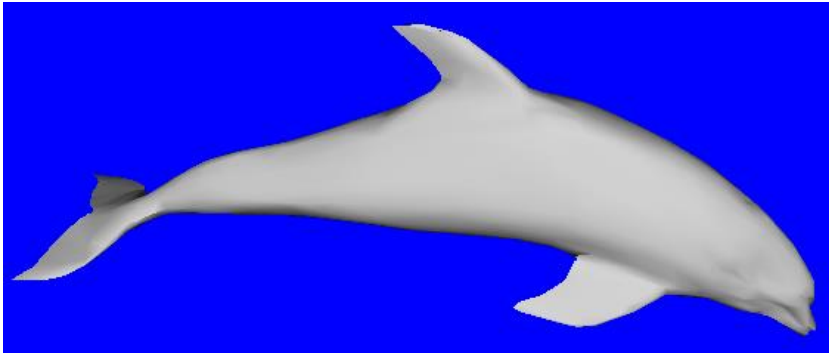
- Compute  $Q_V$  for all the mesh vertices
- For each valid pair, compute
  - optimal vertex position
  - its error
- Store all valid pairs in a priority queue (sorted by  $d$ )
- While reduction goal not met
  - Take an edge from the queue, contract
  - Update the priority queue with new valid pairs

# Examples

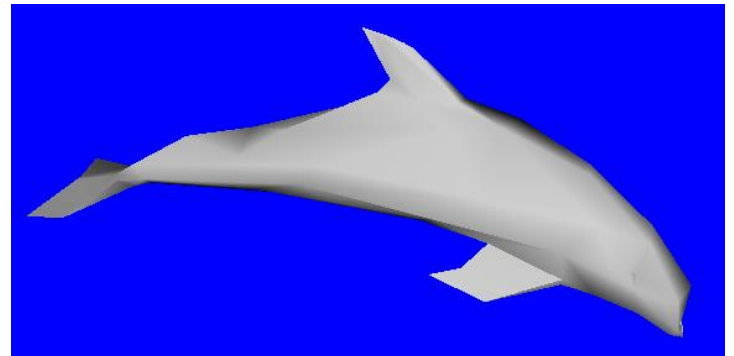
Dolphin (Flipper)



Original - 12,337 faces



2,000 faces



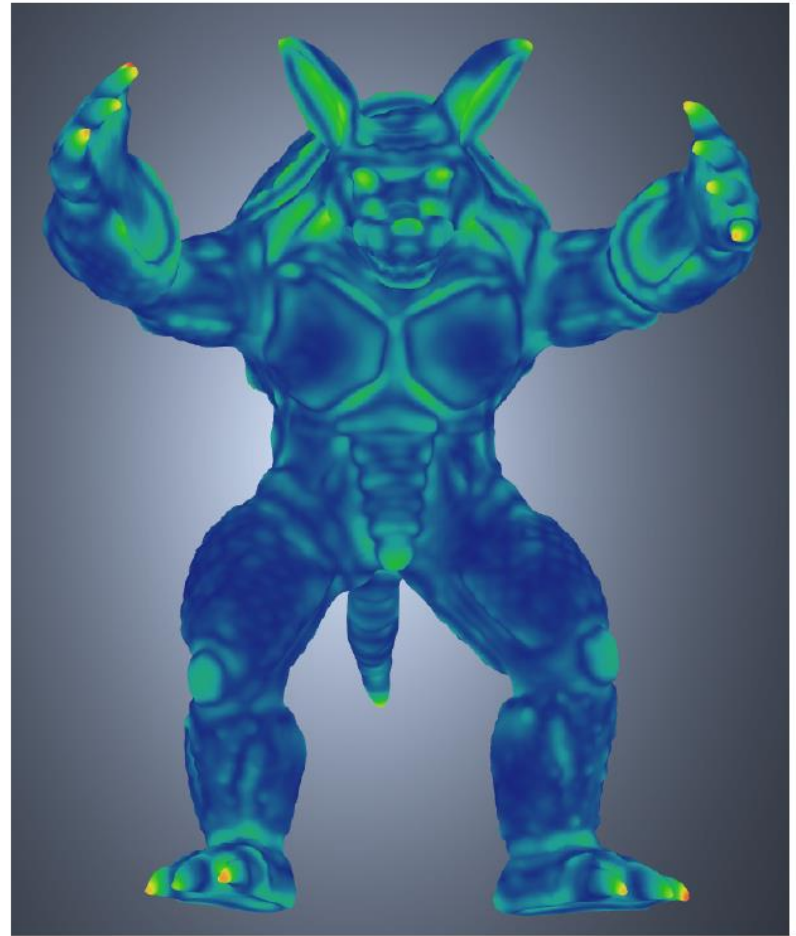
300 faces (142 vertices)

# Pros and Cons

- Pros
  - Error is bounded
  - Allows topology simplification
  - High quality result
  - Quite efficient
- Cons
  - Difficulties along boundaries
  - Difficulties with coplanar planes
  - Introduces new vertices not present in the original mesh

# **MESH SALIENCY**

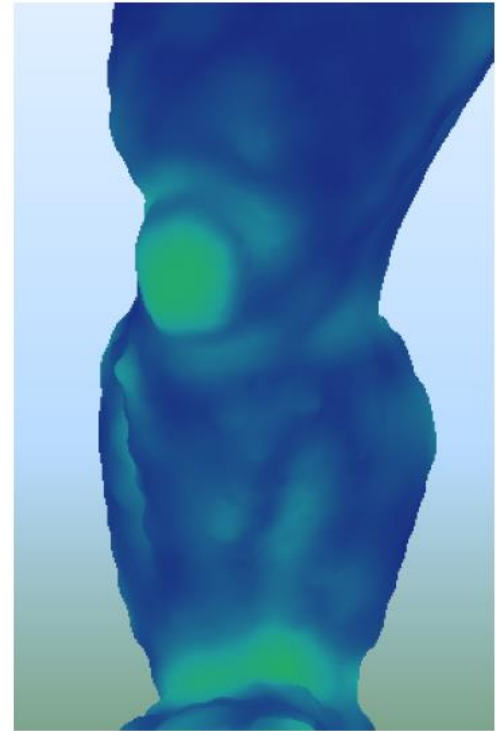
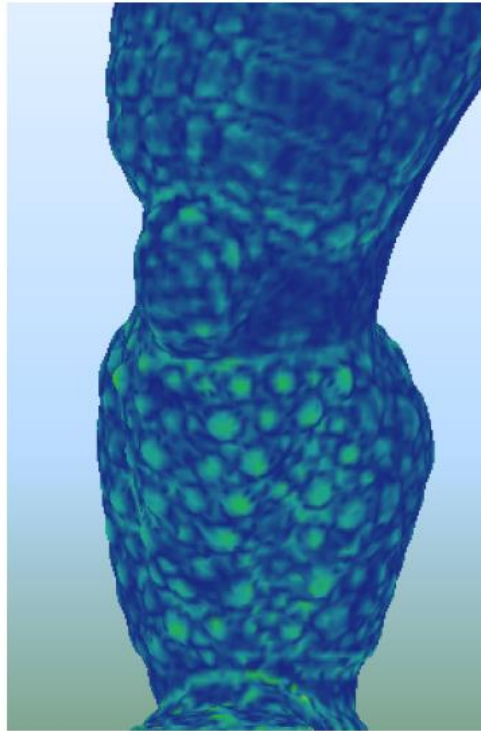
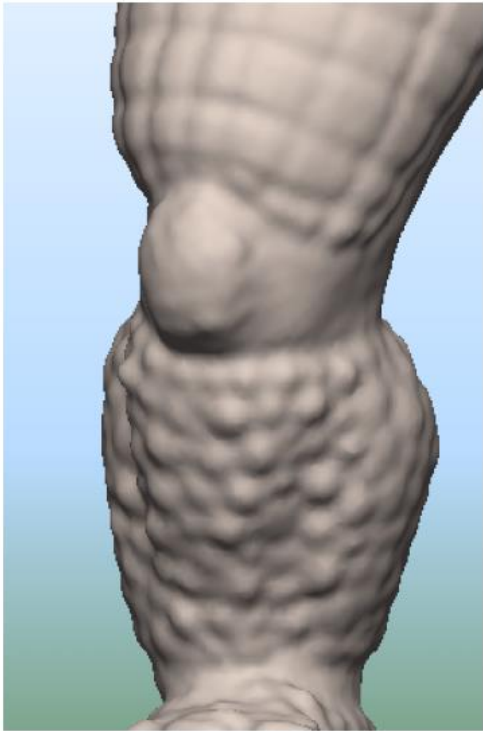
# How to choose a viewpoint to show a mesh?



# Saliency

- Gaussian curvature
- Mean curvature
- Silhouette complexity (?)

# Curvature alone is not enough!





# Idea

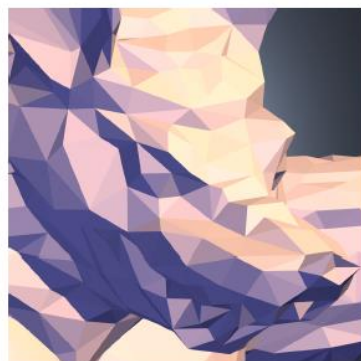
- $k_\sigma$  = Average mean curvature over neighborhood of radius  $\sigma$
- Compute  $|k_\sigma - k_{2\sigma}|$



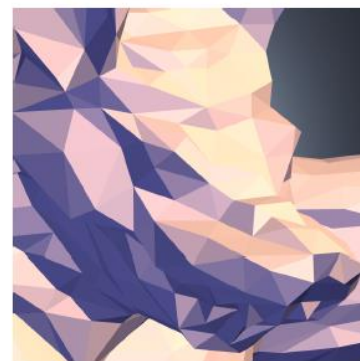
Original  
(346K triangles)



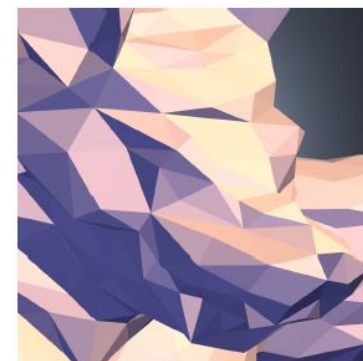
99% simplification  
(3.5K triangles)



98% simplification  
(6.9K triangles)



98.5% simplification  
(5.2K triangles)



99% simplification  
(3.5K triangles)

(a) Simplification by Qslim



Saliency



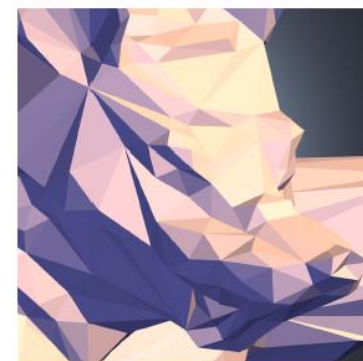
99% simplification  
(3.5K triangles)



98% simplification  
(6.9K triangles)



98.5% simplification  
(5.2K triangles)

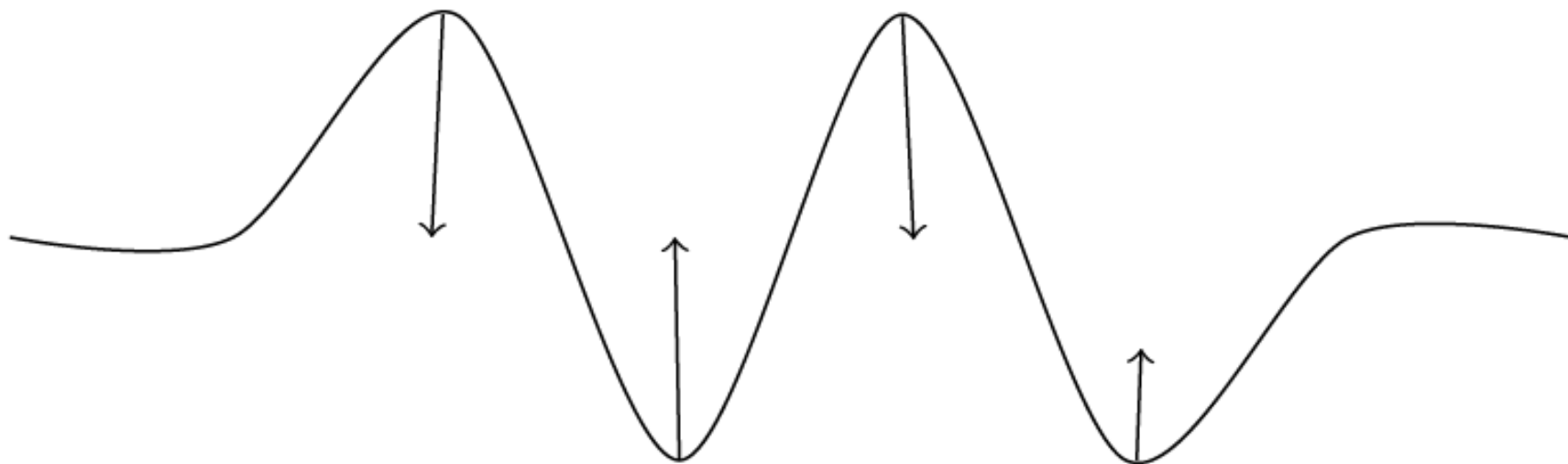


99% simplification  
(3.5K triangles)

(b) Simplification guided by saliency

# **MEAN CURVATURE FLOW**

What will happen to the curve?



# Smoothing!

- aka Curve Shortening Flow
- (eventually) Produces convex curves!

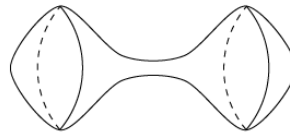


FIGURE 6. The (non-convex) dumbbell will have a “neck-pinch” before the entire surface shrinks away

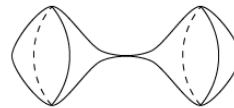


FIGURE 7. At the time of the neck-pinch, most of the surface will remain non-singular, but the diameter of the neck will have shrunk away

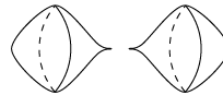
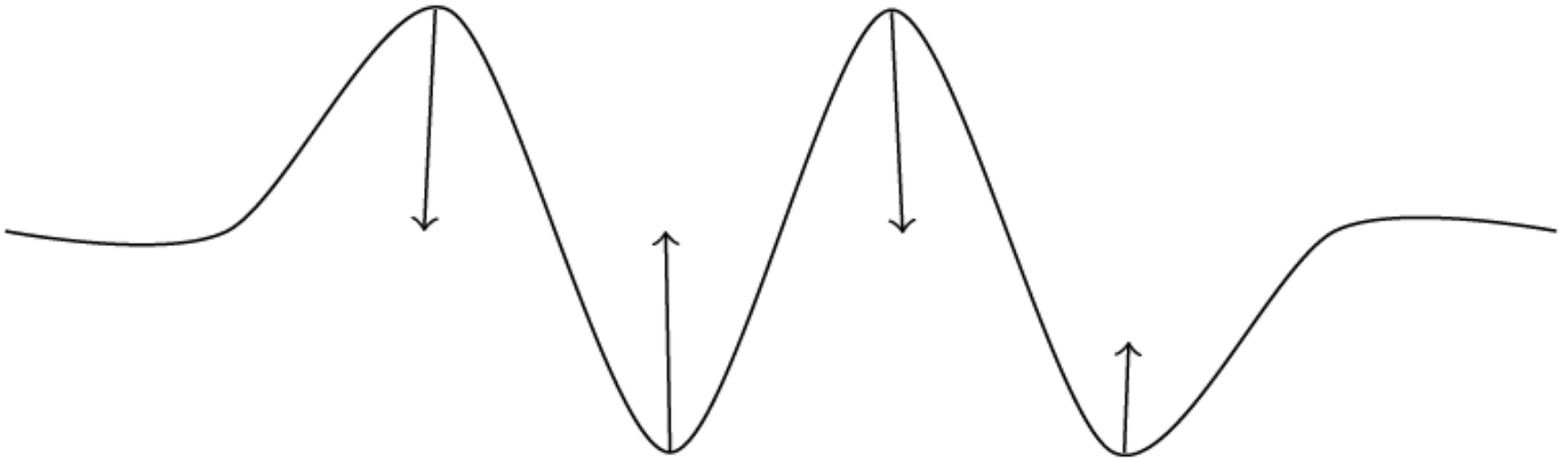


FIGURE 8. It is possible to “continue the flow through the singularity.” It will become smooth immediately after, and then both component will shrink away.

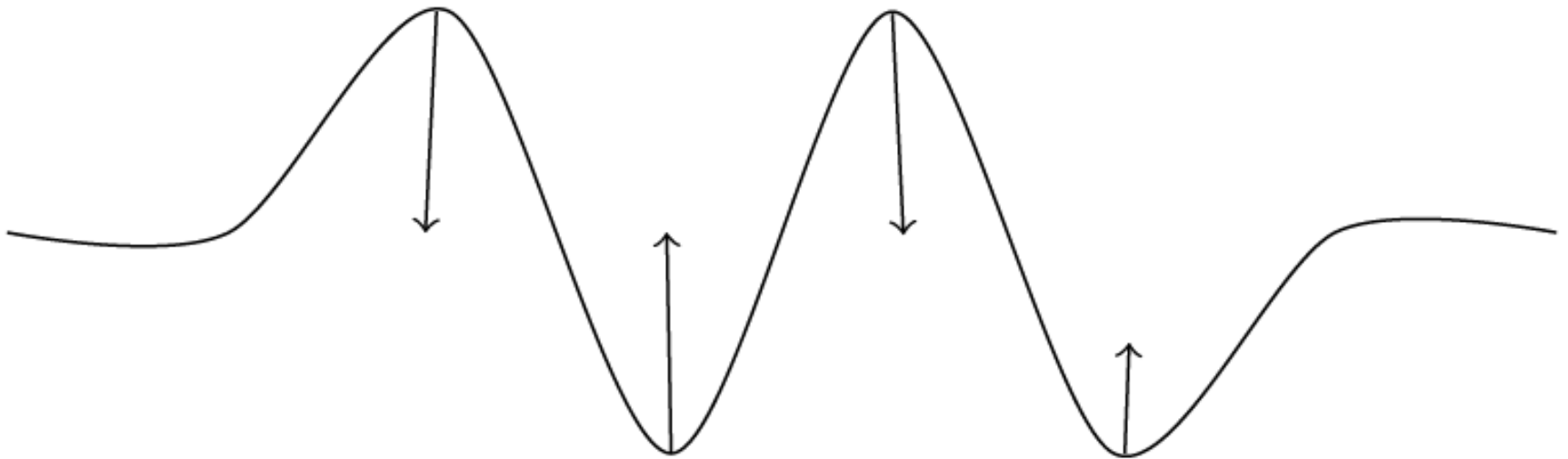
# Formalization

$$\frac{\partial \gamma}{\partial T} = \frac{\partial^2 \gamma}{\partial s^2}$$



# Formalization

$$\frac{\partial \gamma}{\partial T} = \frac{\partial^2 \gamma}{\partial s^2} = k \vec{n}$$



# 3D

## Implicit Fairing of Irregular Meshes using Diffusion and Curvature Flow

Mathieu Desbrun

Mark Meyer

Peter Schröder

Alan H. Barr

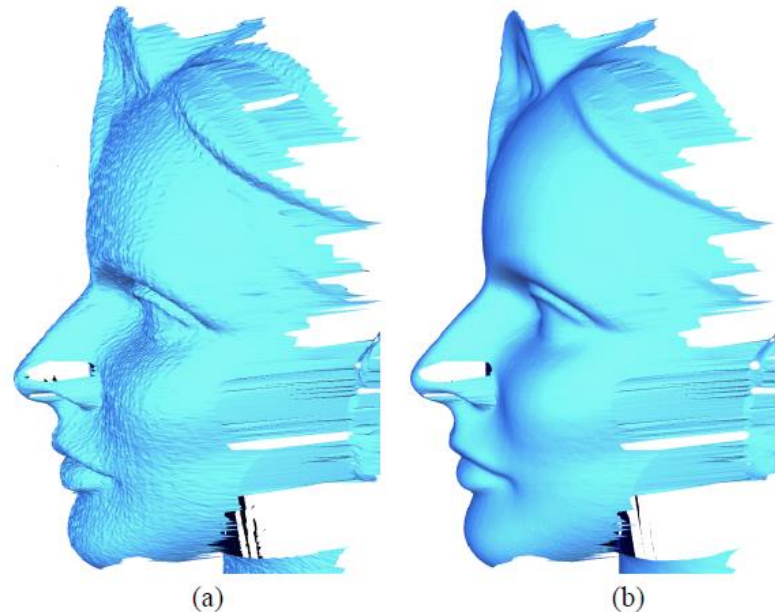
Caltech\*

### Abstract

In this paper, we develop methods to rapidly remove rough features from irregularly triangulated data intended to portray a smooth surface. The main task is to remove undesirable noise and uneven edges while retaining desirable geometric features. The problem arises mainly when creating high-fidelity computer graphics objects using imperfectly-measured data from the real world.

Our approach contains three novel features: an *implicit integration* method to achieve efficiency, stability, and large time-steps; a scale-dependent Laplacian operator to improve the diffusion process; and finally, a robust curvature flow operator that achieves a smoothing of the shape itself, distinct from any parameterization. Additional features of the algorithm include automatic exact volume preservation, and hard and soft constraints on the positions of the points in the mesh.

We compare our method to previous operators and related algorithms, and prove that our curvature and Laplacian operators have several mathematically-desirable qualities that improve the appearance of the resulting surface. To demonstrate the effectiveness of





# Laplacian smoothing

$$\frac{\partial x_i}{\partial t} = -\bar{\kappa}_i \mathbf{n}_i$$

$$-\bar{\kappa} \mathbf{n} = \frac{1}{4A} \sum_{j \in N_1(i)} (\cot \alpha_j + \cot \beta_j) (x_j - x_i)$$

# Laplacian Smoothing

