

IFT 6113

APPLICATIONS OF

CURVATURE

<http://tiny.cc/ift6113>



Mikhail Bessmeltsev

MESH SIMPLIFICATION

Warning: Stone Age Tools
(but still useful)



Image from Wikipedia

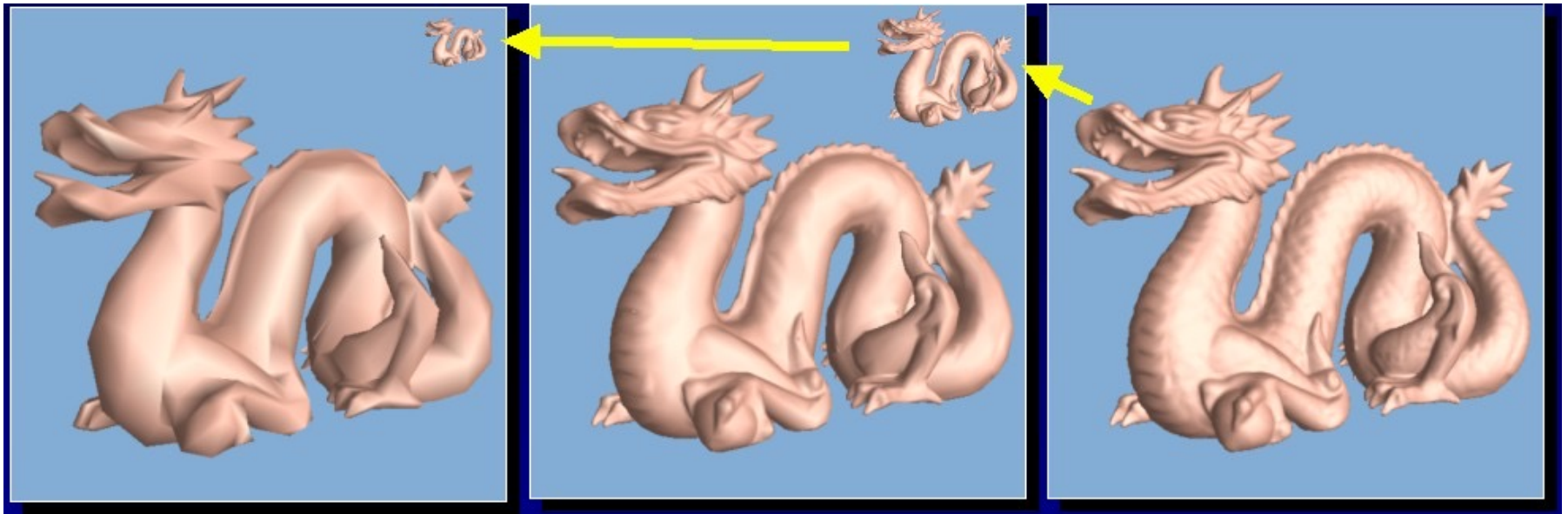


Why simplify meshes?

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

Why simplify meshes?

Faster rendering: Level of Detail

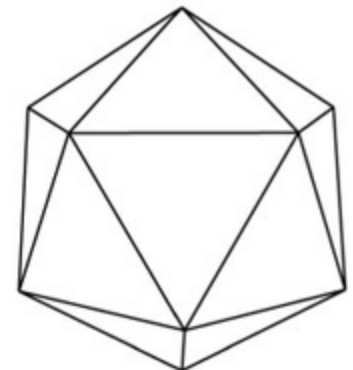
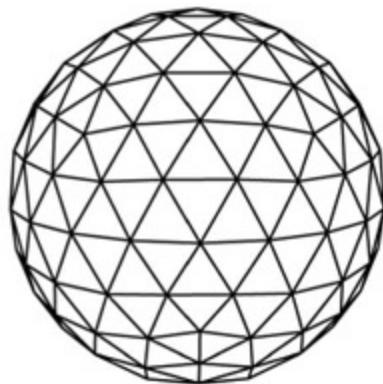
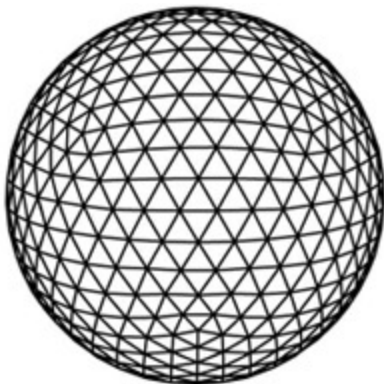


Rough, if the mesh is far

Fine, if the mesh is close

Requirements

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

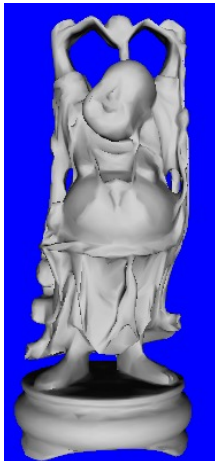


Complexities

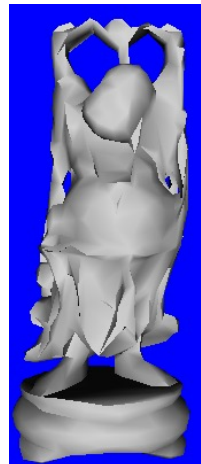
Holes!

Connected components!

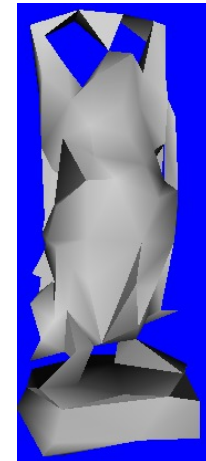
Boundaries!



12,000

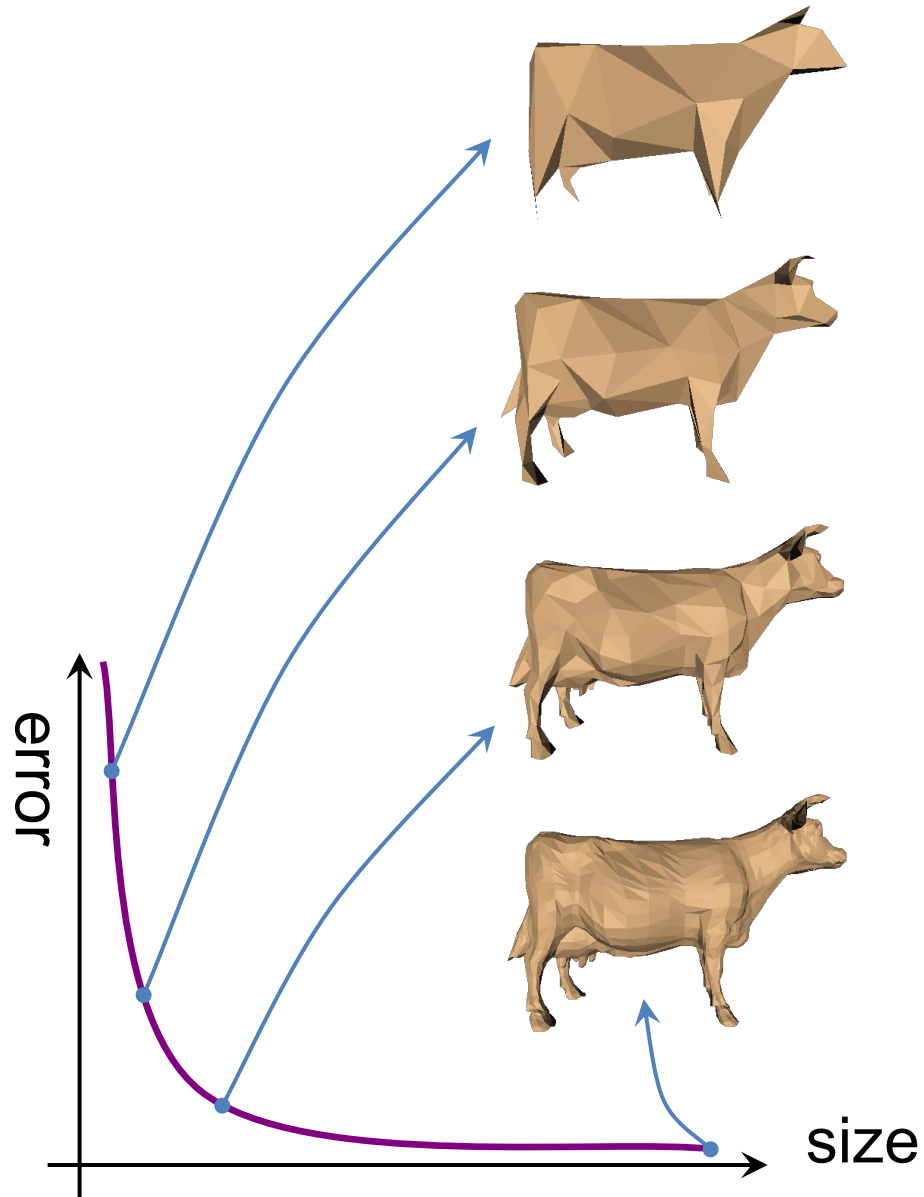


2,000



300

Approximation error



Approach: **Local operations**

Removing edges or vertices

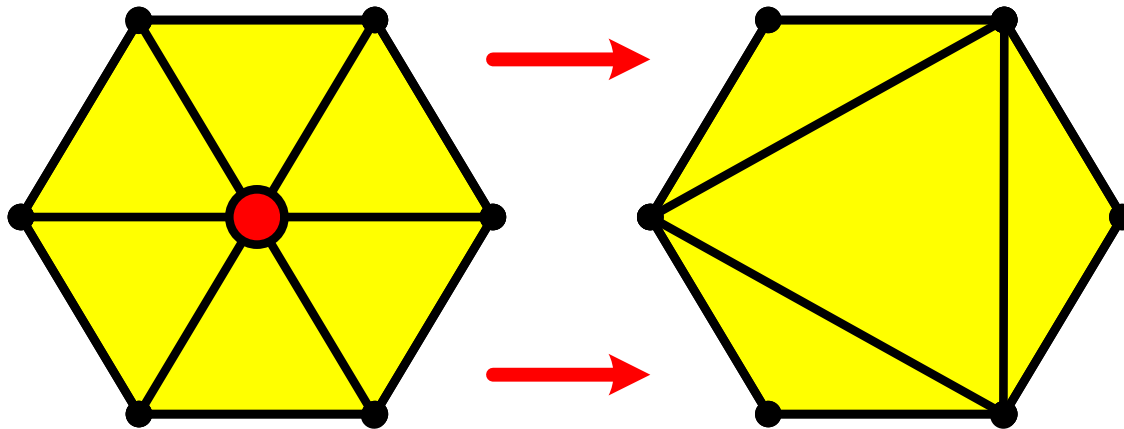
Keeping track of the geometry/topology

Keep score

how much will we lose if we remove a vertex?

Approach

- Each operation introduces error
- Quantify?

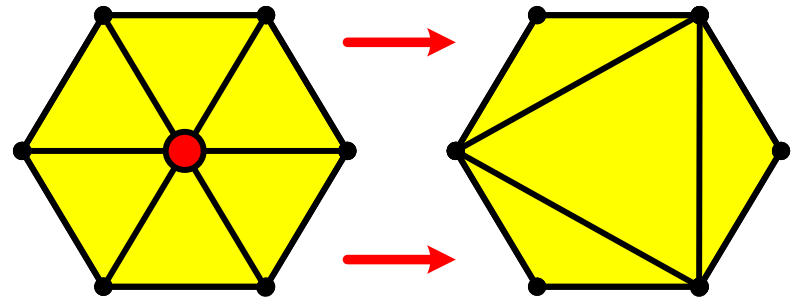


Decimation

- Vertex removal

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

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



Remove vertex \rightarrow hole \rightarrow triangulate

Decimation

- Vertex removal

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

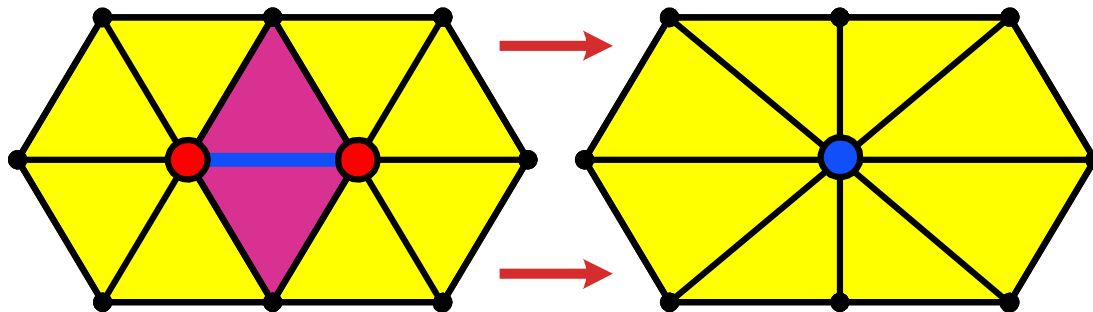
$$\#F \rightarrow \#F - 2$$

Decimation

- Edge collapse

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

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

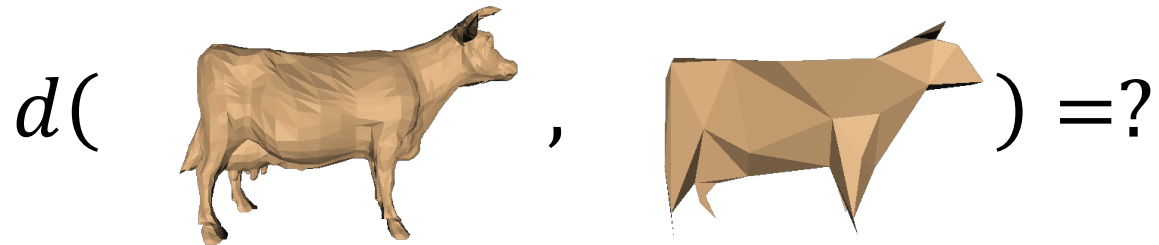


Measuring error

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

How much did we distort the shape?

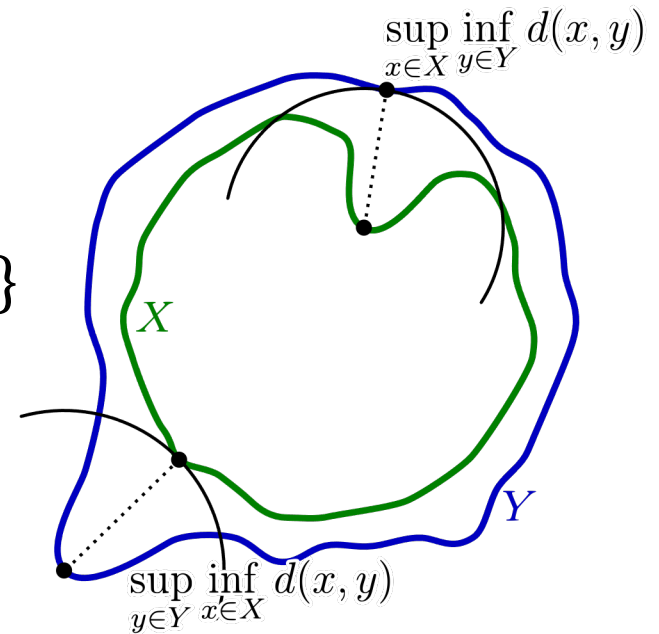
Measuring error



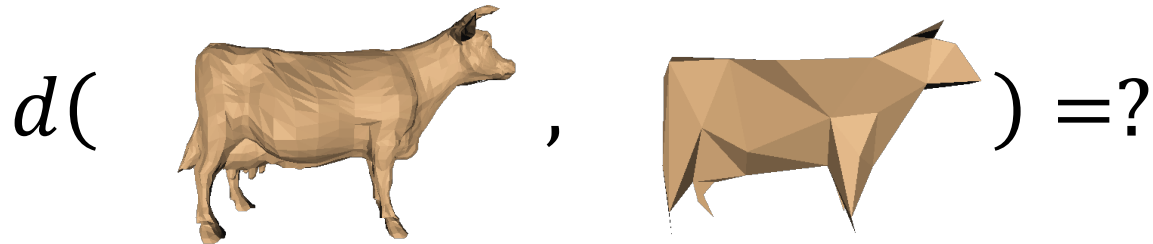
How much did we distort the shape?

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

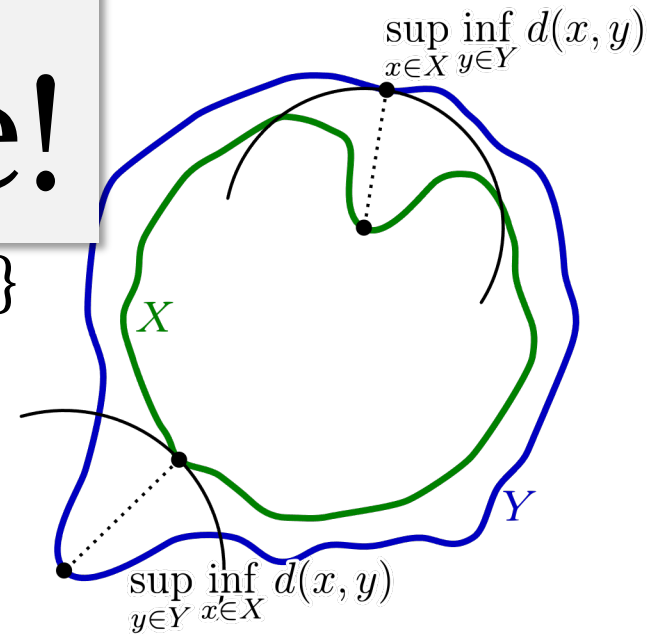


How to measure the shape?

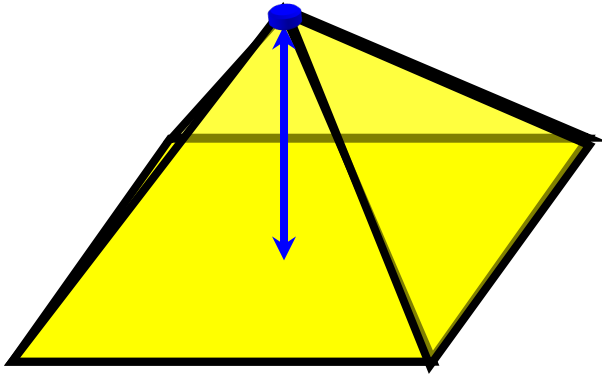
Option: Hausdorff

Hard to compute!

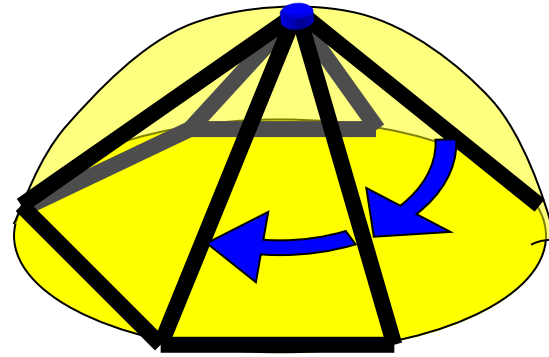
$$d_H(x, y) = \max\left\{\sup_{x \in X} \inf_{y \in Y} d(x, y), \sup_{y \in Y} \inf_{x \in X} d(x, y)\right\}$$



Local Error Control



Vertex-plane distance



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

Curvature

Algorithm

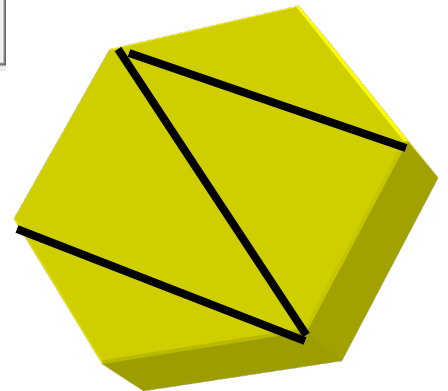
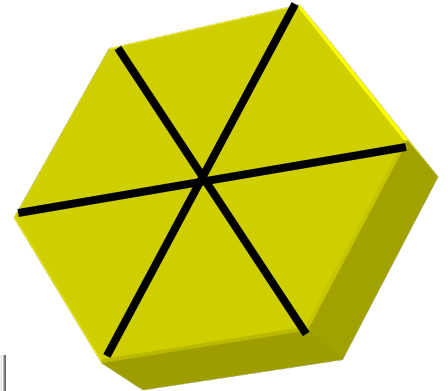
```
while (nVertices > nTarget)
  e = elementWithMinError()
  process(e) # remove/contract
  preserveStructure()
  updateErrors() # local/global
```

Data Structures

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

Vertex Removal Algorithm

```
1 while (nVertices > nTarget)
2   e = elementWithMinError()
3   process(e) # remove/contract
4   preserveStructure()
5   updateErrors() # local/global
```

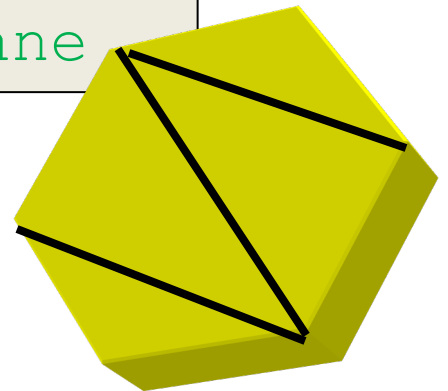


Vertex Removal Algorithm

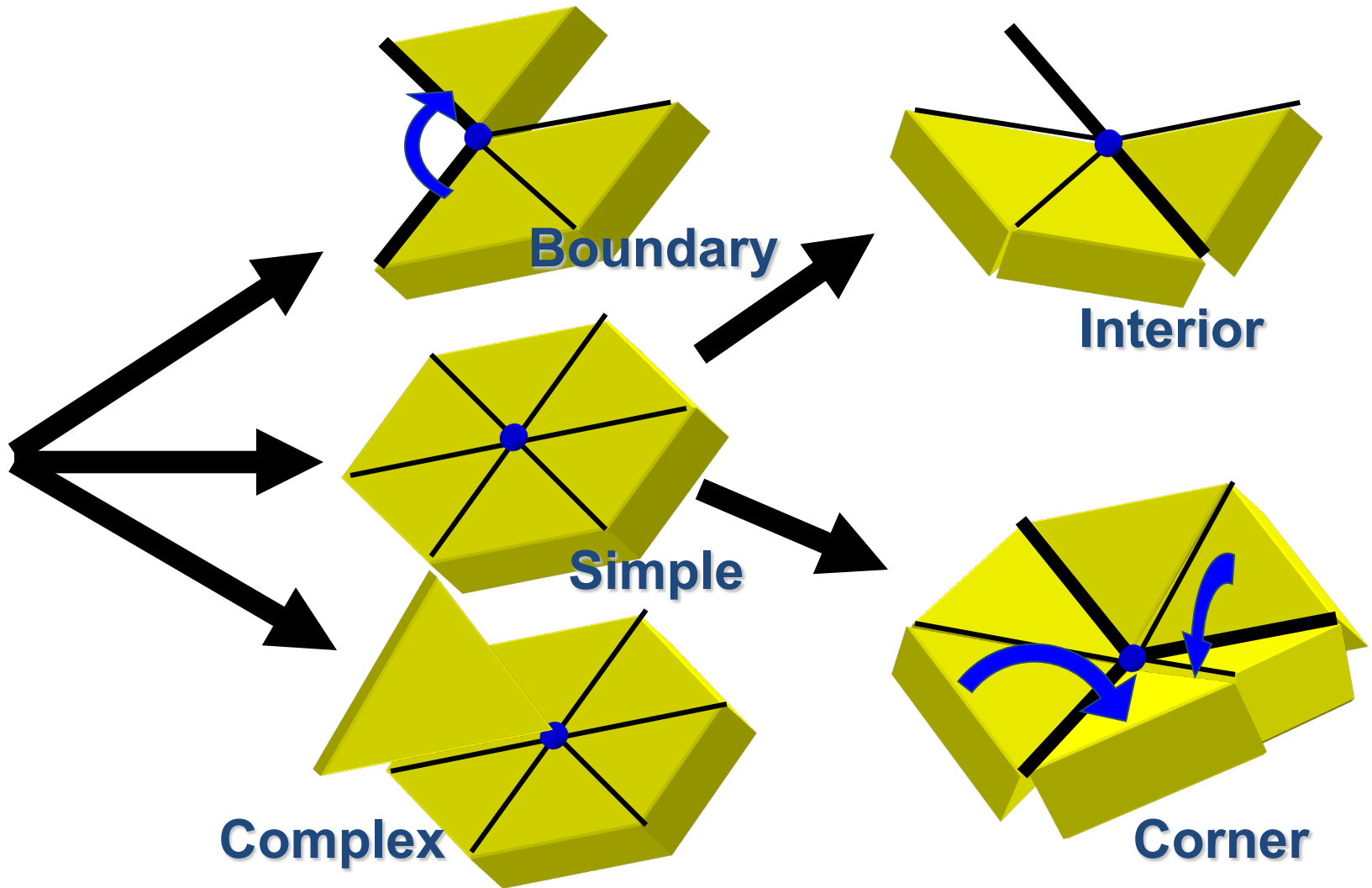
Characterize local topology/geometry

Classify vertices as removable or not

```
while (nVertices > nTarget)
  v = vertexWithMinDistance()
  remove(v)
  triangulateHole()
  updateDistances() #to average plane
```



Characterizing structure



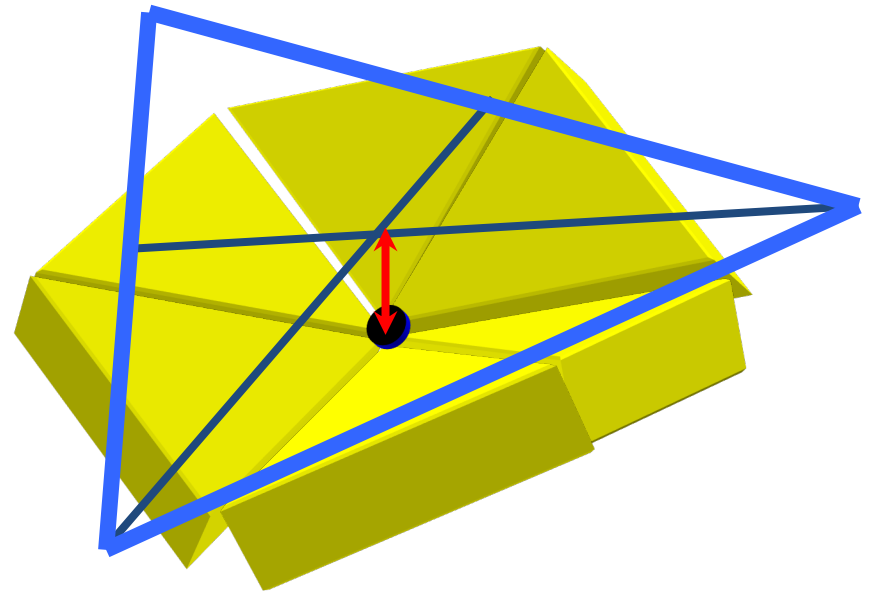
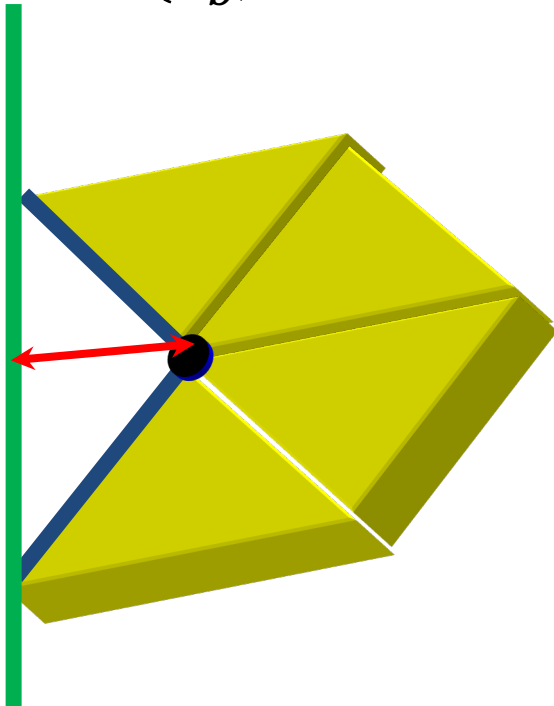
Decimation Criterion

Simple vertex v :

$$d(v, \text{face loop average plane}) < E_{MAX}$$

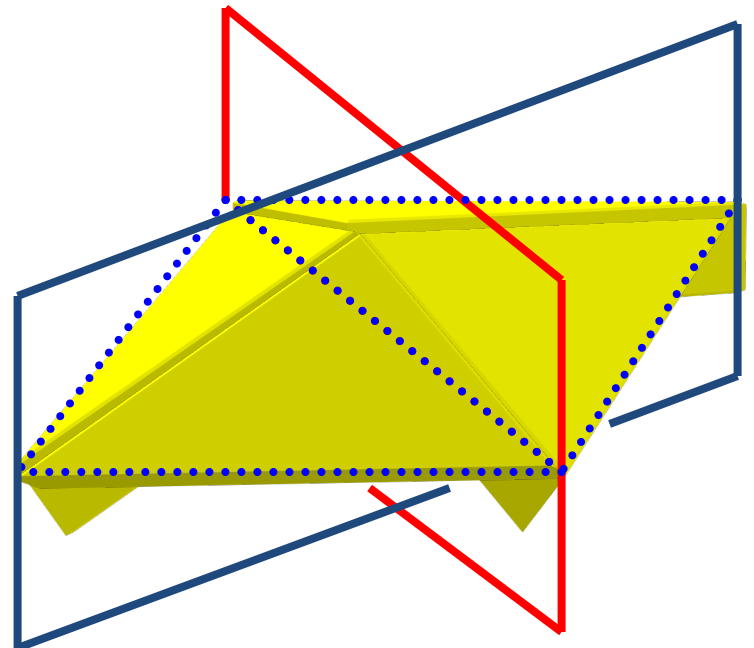
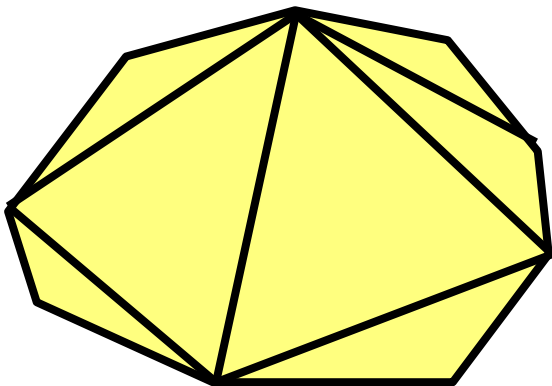
Boundary vertex v_b :

$$d(v_b, \text{new boundary edge}) < E_{MAX}$$

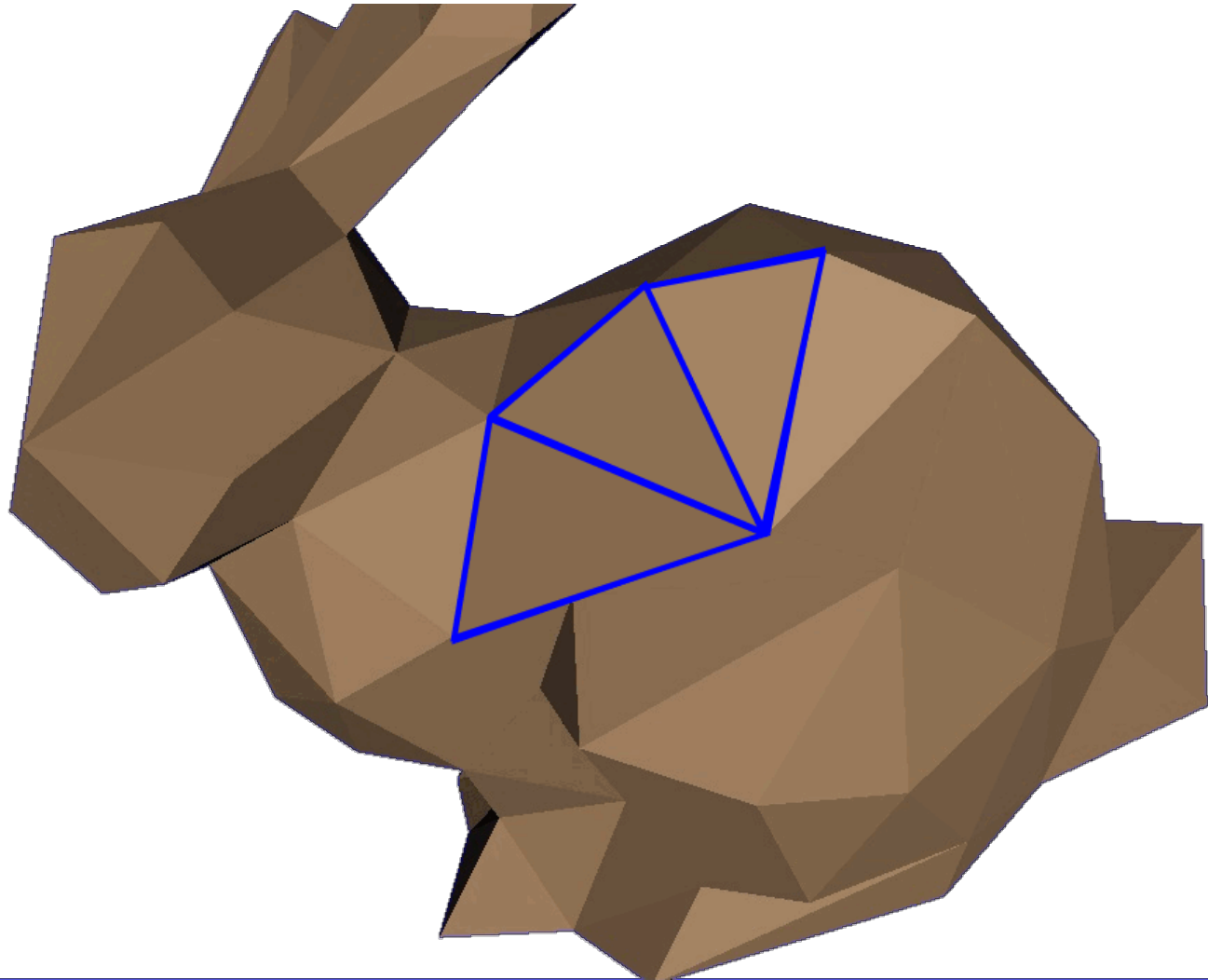


triangulateHole()

- Non-planar hole!
 - Split loop recursively
 - Split plane orthogonal to the average plane
- Control aspect ratio
- May fail
 - Vertex is not removed



Example



Simplifier

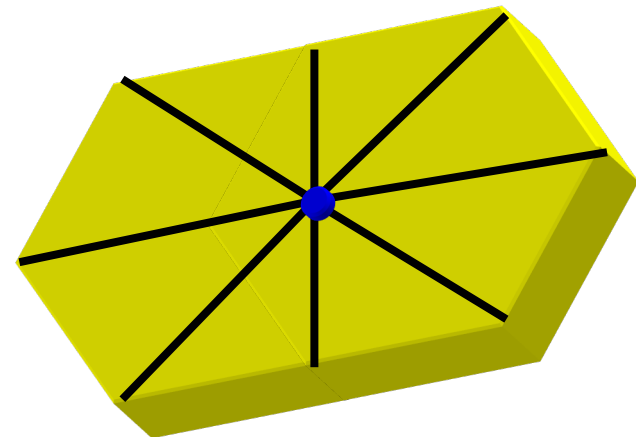
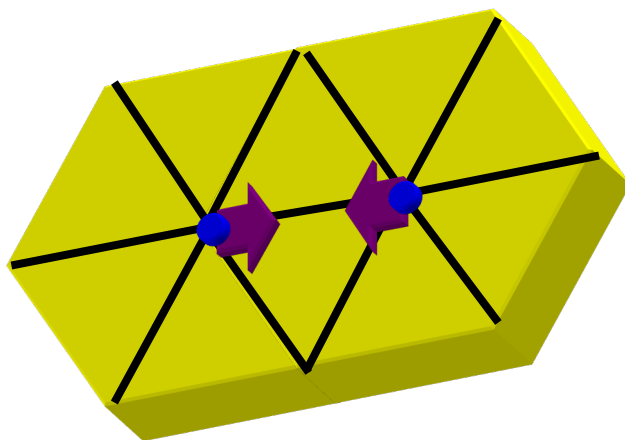
Pros and Cons

- 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

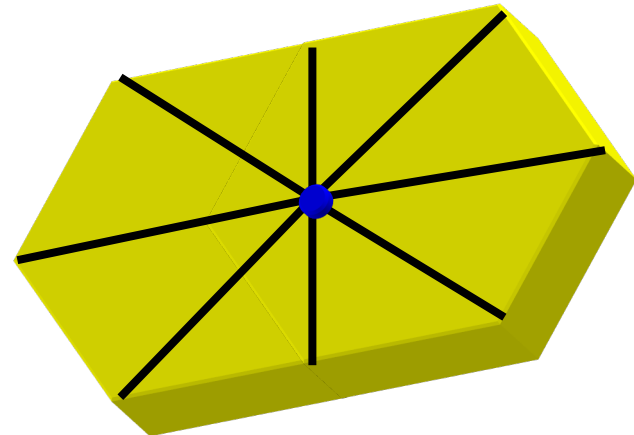
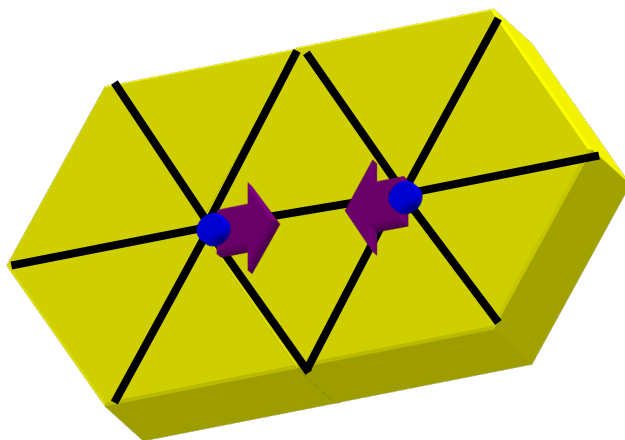
```
1 while (nVertices > nTarget)
2   e = elementWithMinError()
3   process(e) # remove/contract
4   preserveStructure()
5   updateErrors() # local/global
```

I



Edge Collapse Algorithm

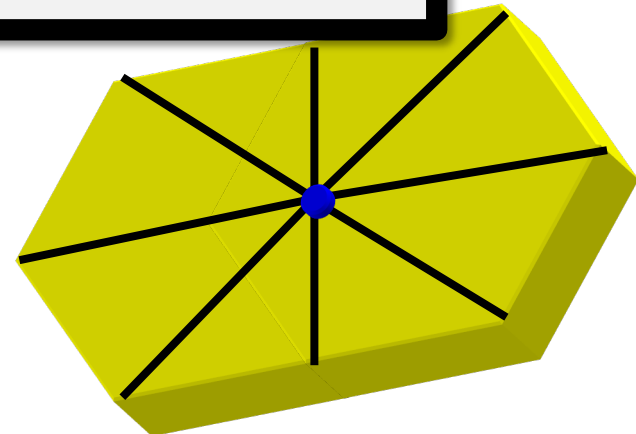
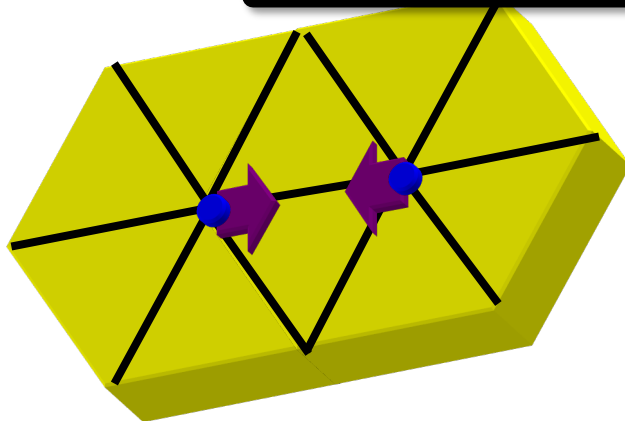
```
while (nVertices > nTarget)
  e = edgeWithMinError()
  v = collapse(e) #returns a vertex
  place(v)
  updateErrors() # local/global
```



Edge Collapse Algorithm

```
while (nVertices > nTarget)
  e = edgeWithMinError()
  v = collapse(e) #returns a vertex
  place(v)
  updateErrors() # local/global
```

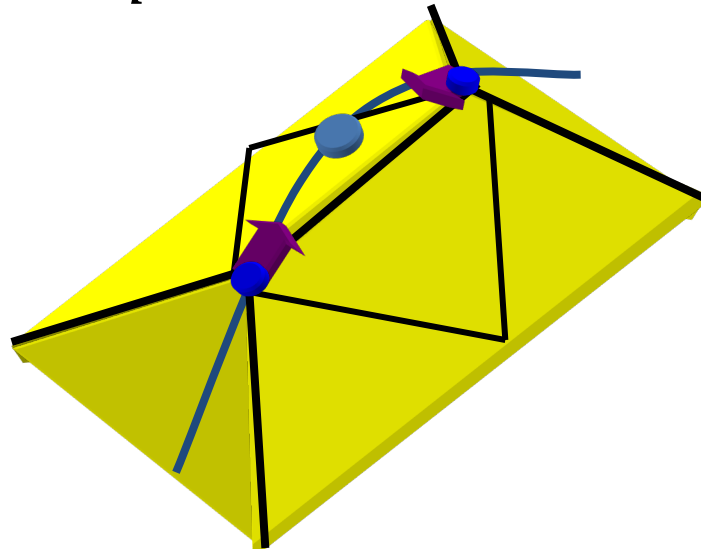
Where should we
place the vertex?



Distance Metric: Quadrics

Find the closest point to
the set of adjacent ~~triangles~~ planes

$$v = \underset{p}{\operatorname{argmin}} d(p, \text{planes}(v))$$



$$v = \operatorname{argmin}_p d(p, \text{planes}(v))$$

Plane equation

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

Squared distance to a plane

$$d(p, \text{plane}) = (Ax + By + Cz + D)^2$$

assuming

$$\|\vec{n}\|^2 = A^2 + B^2 + C^2 = 1$$

Squared distance to a plane

$$d(p, \text{plane}) = (Ax + By + Cz + D)^2$$

Squared distance to a plane

$$\begin{aligned}d(\mathbf{x}, \text{plane}) &= (Ax + By + Cz + D)^2 \\ &= (\mathbf{x}^T \mathbf{p})^2 = \mathbf{x}^T \mathbf{p} \mathbf{p}^T \mathbf{x}\end{aligned}$$

$$\mathbf{p} = \begin{pmatrix} A \\ B \\ C \\ D \end{pmatrix}$$

Squared distance to a plane

$$\begin{aligned}d(\mathbf{x}, \text{plane}) &= (Ax + By + Cz + D)^2 \\ &= (\mathbf{x}^T \mathbf{p})^2 = \mathbf{x}^T \mathbf{p} \mathbf{p}^T \mathbf{x} = \mathbf{x}^T \mathbf{K} \mathbf{x}\end{aligned}$$

$$\mathbf{K} = \mathbf{p} \mathbf{p}^T = \begin{pmatrix} A^2 & AB & AC & AD \\ AB & B^2 & BC & BD \\ AC & BC & C^2 & CD \\ AD & BD & CD & D^2 \end{pmatrix} \succeq 0$$

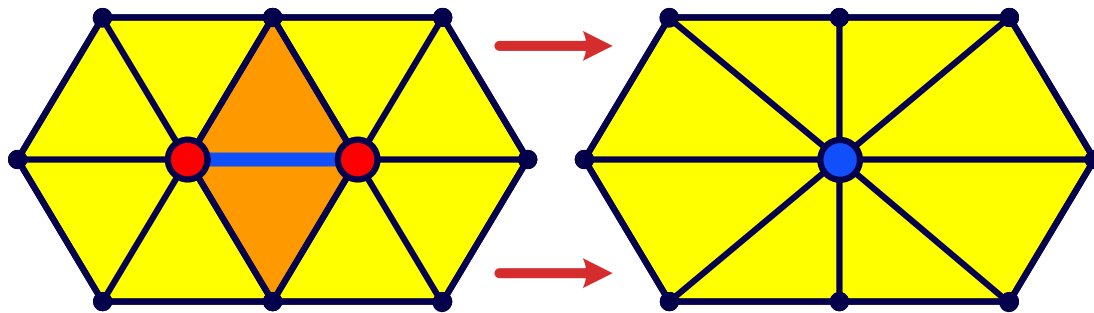
Squared distance to a set of planes

$$\begin{aligned}d(x, \mathit{planes}(v)) &= \sum_i d(x, \mathit{plane}_i) = \\&= \sum_i \mathbf{x}^T K_i \mathbf{x} = \mathbf{x}^T \left(\sum_i K_i \right) \mathbf{x} = \\&= \mathbf{x}^T Q_v \mathbf{x}\end{aligned}$$

place (v)

$$v = \operatorname{argmin}_x \mathbf{x}^T Q_v \mathbf{x}$$

$Q_v = Q_{v_1} + Q_{v_2}$ #assuming we collapsed edge between v_1 and v_2



$$v = \underset{x}{\operatorname{argmin}} \mathbf{x}^T Q_v \mathbf{x}$$

$$\text{s. t. } \mathbf{x} = (x, y, z, 1)^T$$

$$\Rightarrow$$

$$\begin{pmatrix} q_{11} & q_{12} & q_{13} & q_{14} \\ q_{12} & q_{22} & q_{23} & q_{24} \\ q_{13} & q_{23} & q_{33} & q_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix} \mathbf{x} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

Algorithm

Compute Q_v for all the mesh vertices

for each valid pair

 Compute optimal vertex position

 Compute its error

Store all valid pairs in a priority queue
(sorted by d)

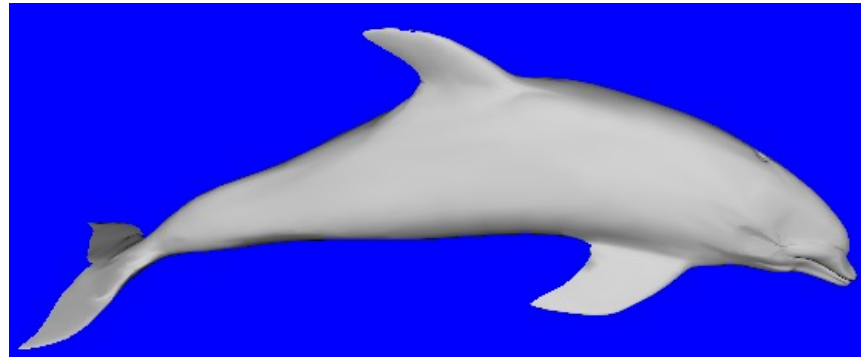
while reduction goal not met

 Take an edge from the queue, contract

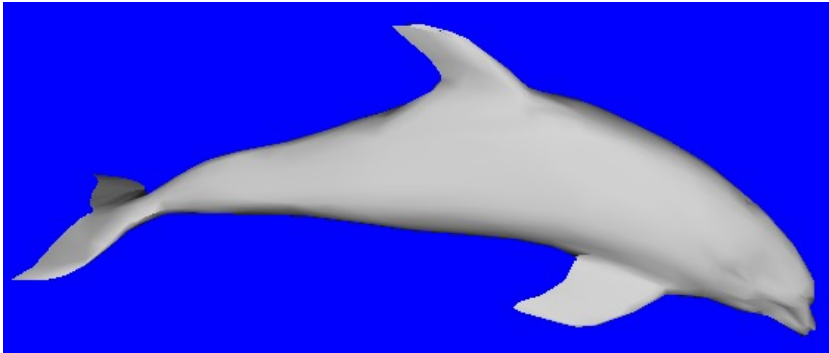
 Update the priority queue with new valid pairs

Examples

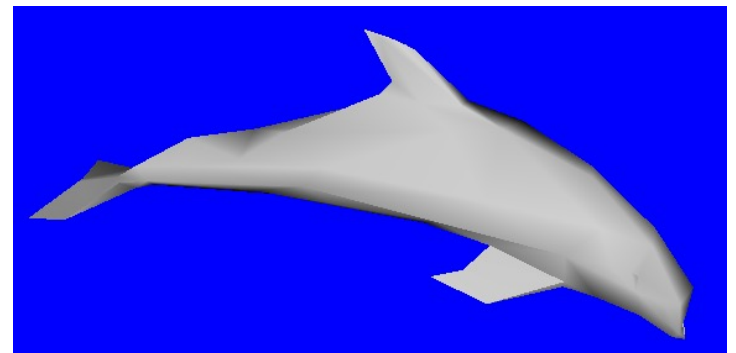
Dolphin (Flipper)



Original - 12,337 faces



2,000 faces



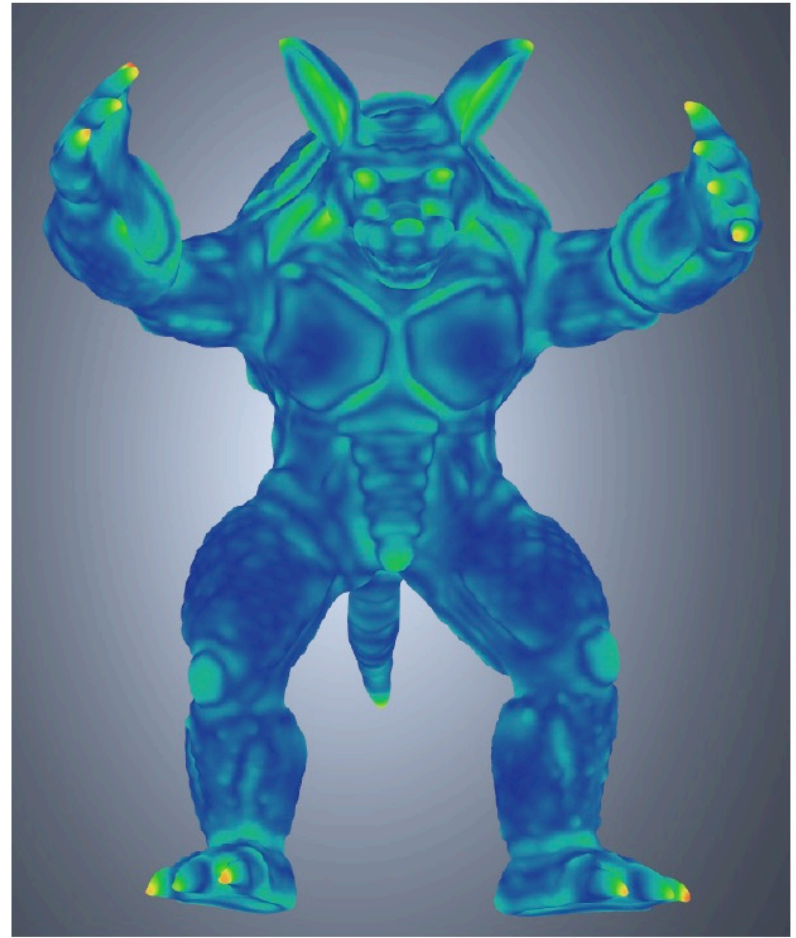
300 faces (142 vertices)

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

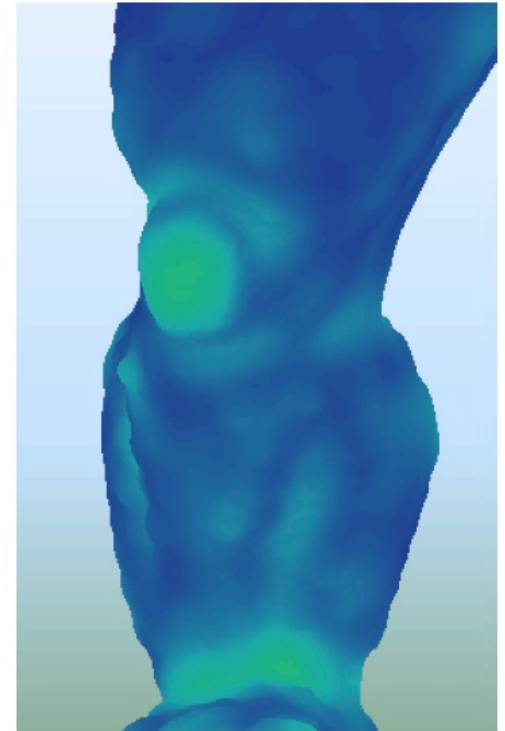
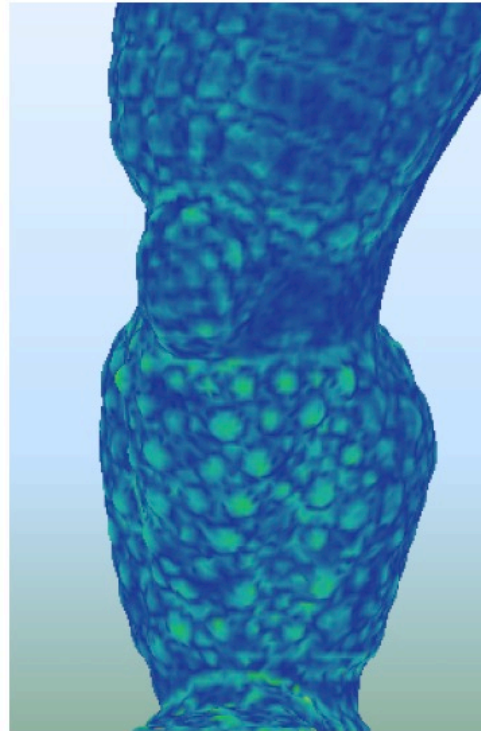
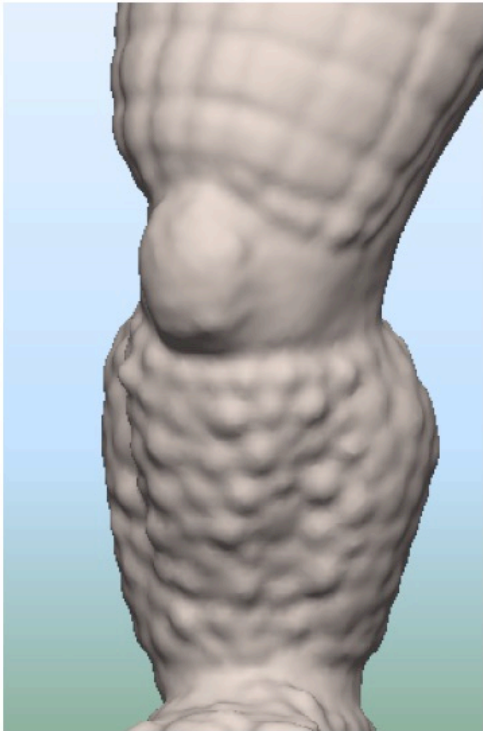
How to choose a viewpoint to show a mesh?



Saliency

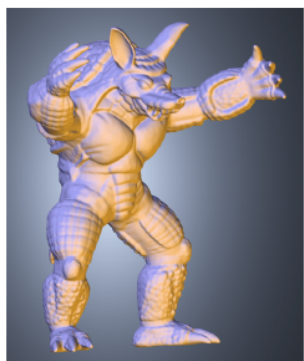
- Gaussian curvature
- Mean curvature
- Silhouette complexity (?)

Curvature alone is not enough!



Idea

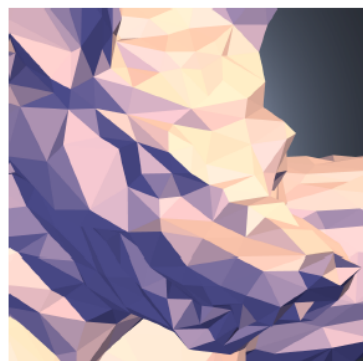
- k_σ = Average mean curvature over neighborhood of radius σ
- Compute $|k_\sigma - k_{2\sigma}|$



Original
(346K triangles)

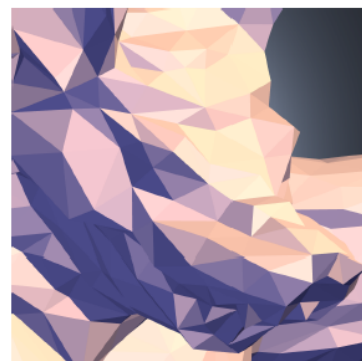


99% simplification
(3.5K triangles)

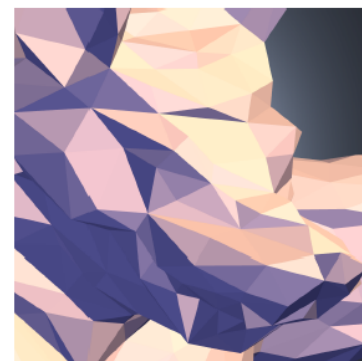


98% simplification
(6.9K triangles)

(a) Simplification by Qslim



98.5% simplification
(5.2K triangles)



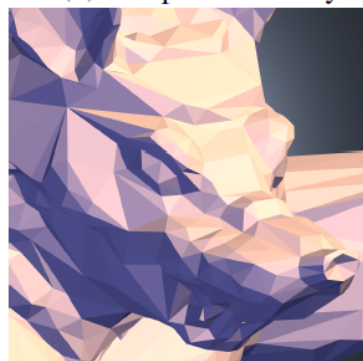
99% simplification
(3.5K triangles)



Saliency

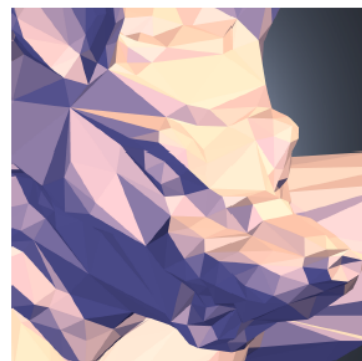


99% simplification
(3.5K triangles)



98% simplification
(6.9K triangles)

(b) Simplification guided by saliency



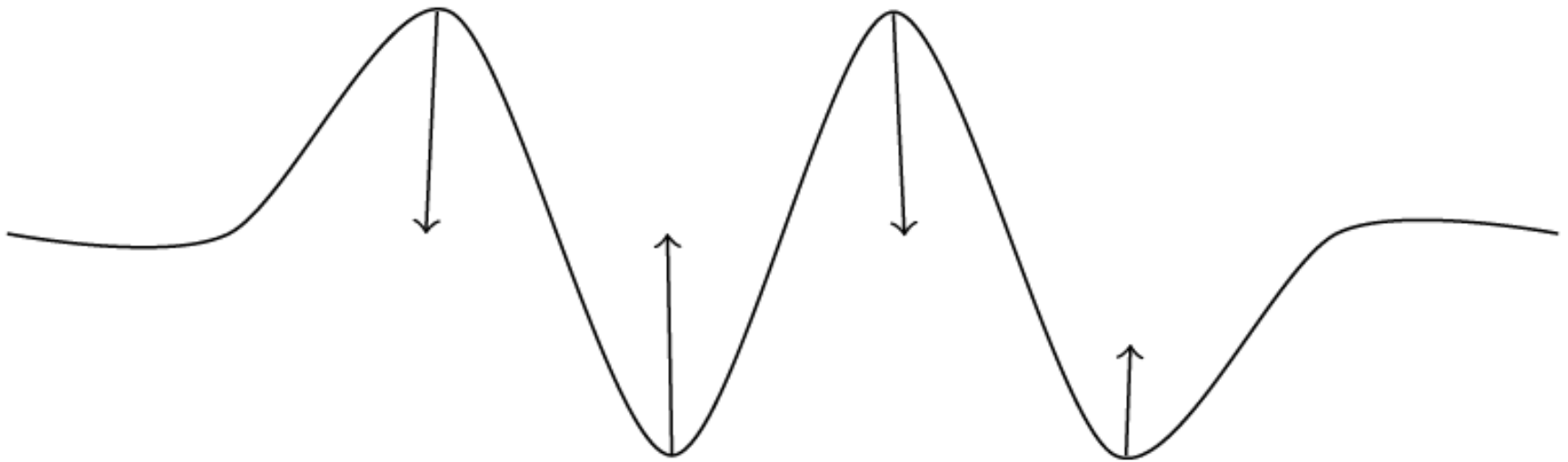
98.5% simplification
(5.2K triangles)



99% simplification
(3.5K triangles)

MEAN CURVATURE FLOW

What will happen to the curve?



Smoothing!

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

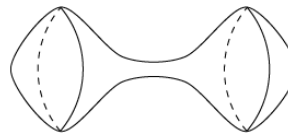


FIGURE 6. The (non-convex) dumbbell will have a “neck-pinch” before the entire surface shrinks away

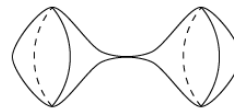


FIGURE 7. At the time of the neck-pinch, most of the surface will remain non-singular, but the diameter of the neck will have shrunk away

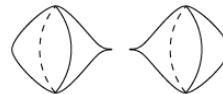
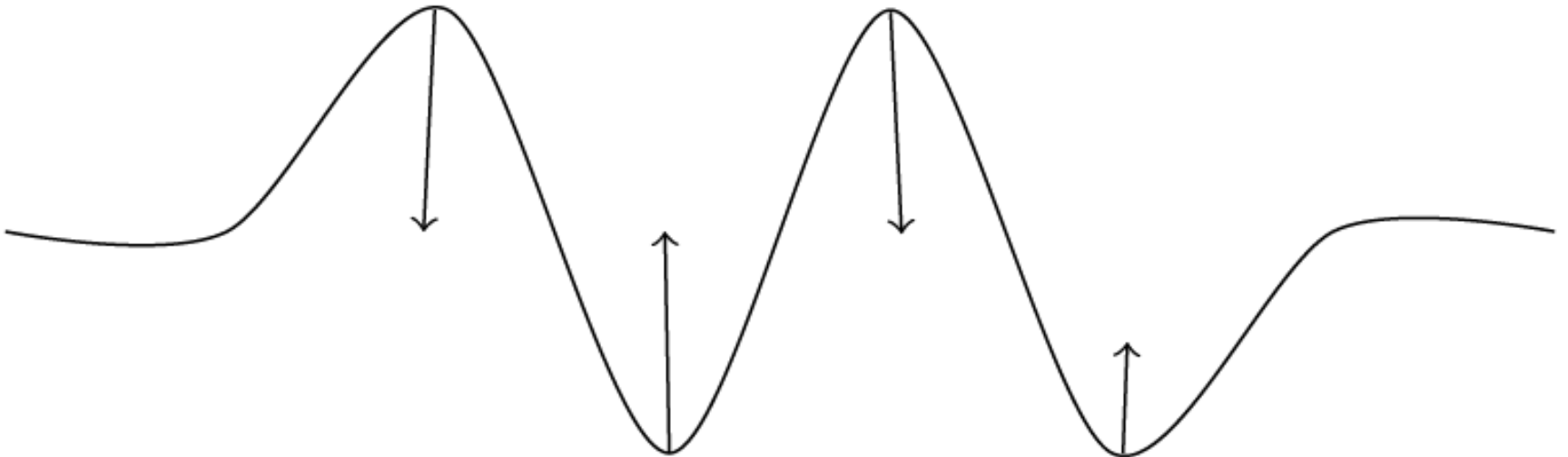


FIGURE 8. It is possible to “continue the flow through the singularity.” It will become smooth immediately after, and then both component will shrink away.

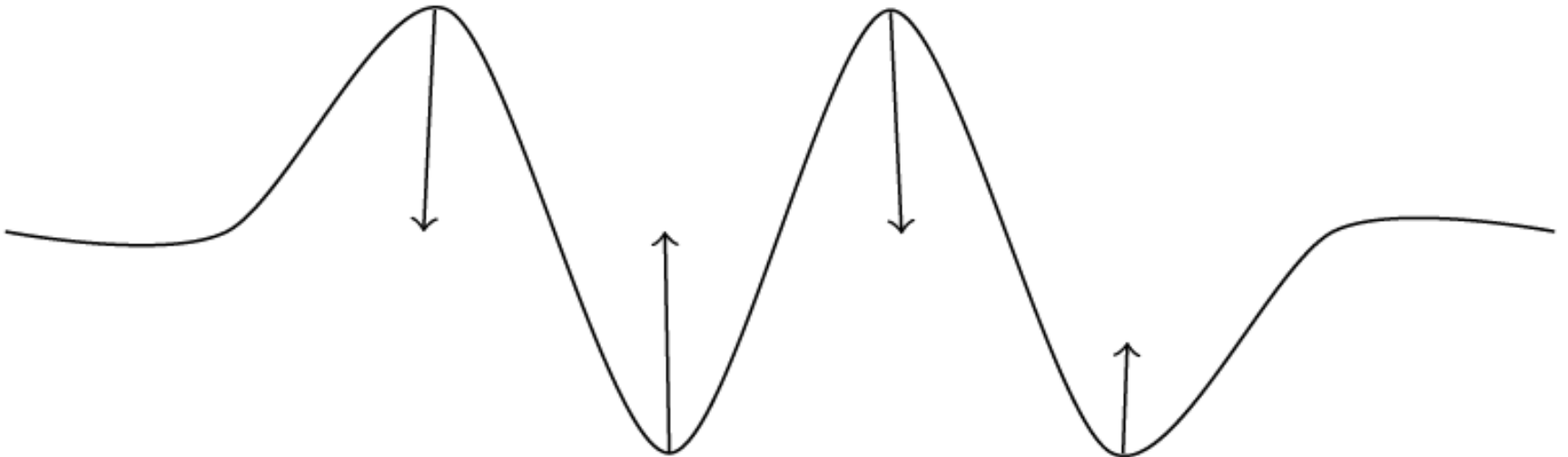
Formalization

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



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

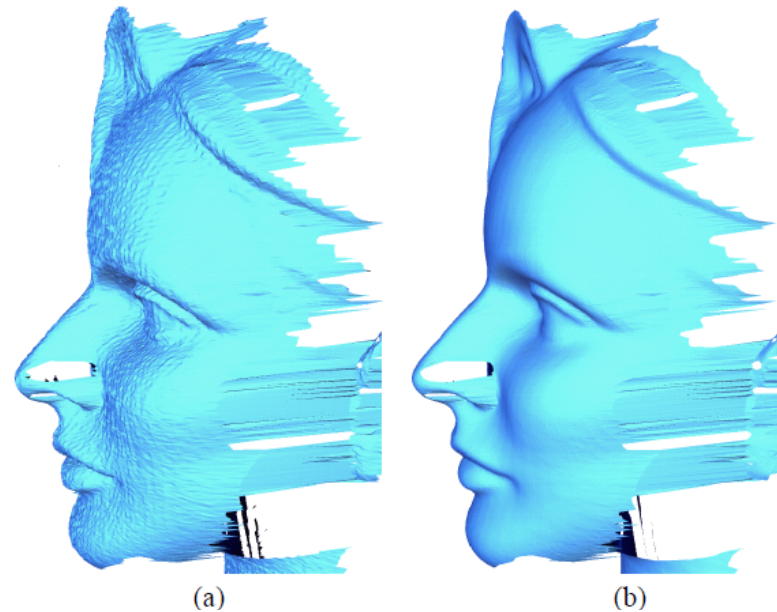
Caltech*

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 appear-



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

