Introduction to Meshes and Geometry Processing

Shape tools that might be useful for geometric deep learning

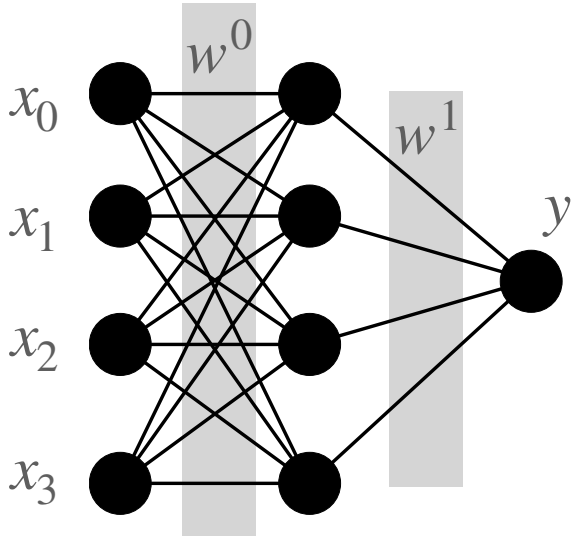
Andreas Bærentzen, Visual Computing, DTU Compute

Multi-Layer Perceptron

- An MLP computes the value of a function:

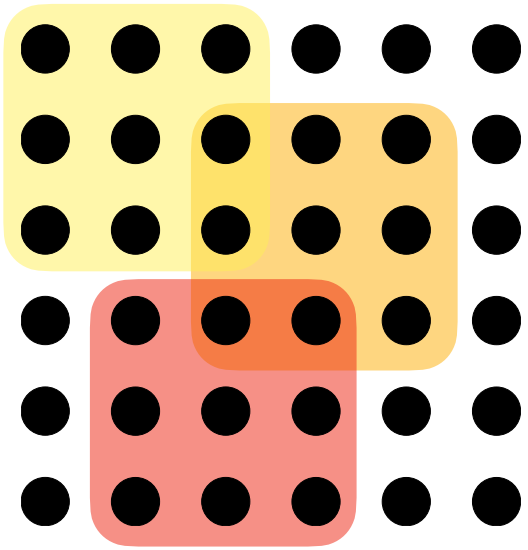
$$y = f(\mathbf{x}) = \phi\left(\sum_j w_j^1 \phi\left(\sum_i w_{i,j}^0 x_i\right)\right)$$

- Where ϕ is the non-linear activation function, and w are the learned weights
- Why is this **powerful**?



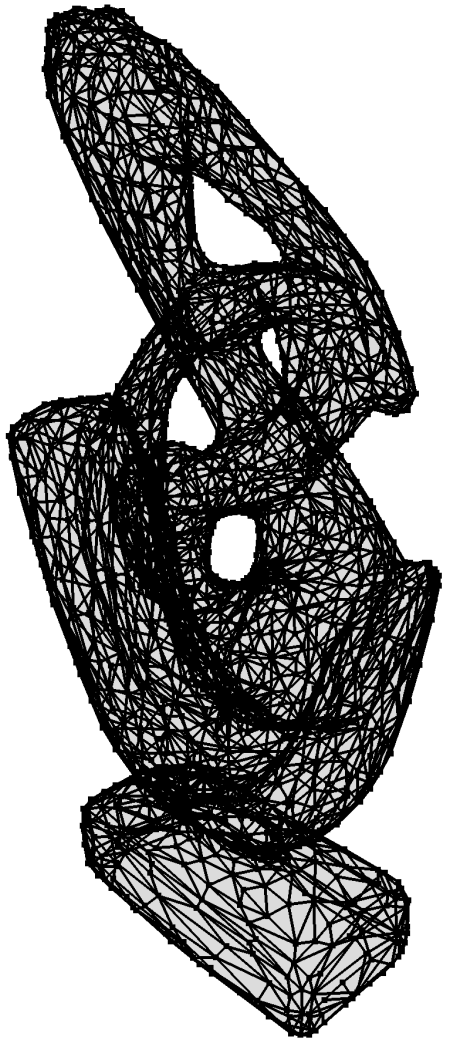
Convolutional Neural Network

- Now, groups of pixels form inputs to the MLP, turning it into a filter
- By design this filter has
  - *translational equivariance*
  - but not *rotational equivariance*
- Now, data is not always on a regular grid…



Geometry

- Now it is not so easy to define convolutions …
- Why do we care? We might
  - have a shape space: each shape is a point
    - Recognition, classification, synthesis
  - want to learn a function on the surface
    - segmentation
    - mapping between shapes
    - position of fiducial/annotation points.



Geometry

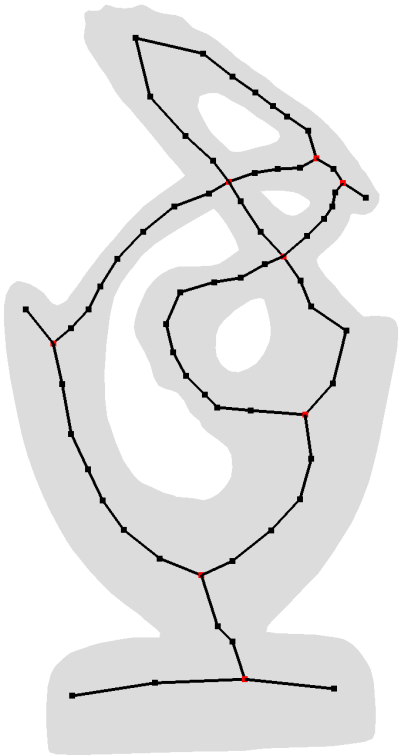
Can come in many guises

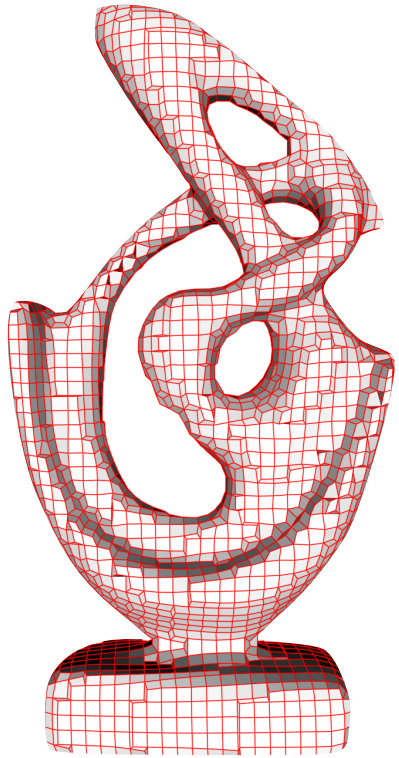
Surface/mesh

Image

Graph

Skeleton





Implicit/distance field/
Voxels

GEL and PyGEL

https://github.com/janba/GEL
https://pypi.org/project/PyGEL3D/
http://www2.compute.dtu.dk/projects/GEL/PyGEL

- GEL is a C++ library of **geometry processing tools** including (but not limited to)
  - a half-edge based polygonal mesh,
  - a graph data structure, and
  - various spatial data structures
- PyGEL
  - a set of Python bindings for core features in GEL
  - has its own viewer based on OpenGL
  - PyGEL can be used from Jupyter notebooks (Also Google Colab)

Who are you !?

The intended audience for this presentation

- Students with a good background in
  - engineering math
  - machine learning
  - computer programming
    - The bits that can be directly reproduced in the PyGEL framework are labelled as shown:

Works in PyGEL

- in search of a “swiss army knife” of methods for geometry processing that can be used for geometric deep learning.

Overview

Declaration of contents

- Part 1
  - Introduction to mesh data structures
  - Mesh simplification and optimization
  - Discrete log map and exp map
  - Linear functions on meshes and the Laplace Beltrami Operator
  - Smoothing, parametrization, spectral analysis, and functional maps
- Part 2
  - Skeletons and topology
  - Implicit functions and distance fields
  - Volumetric Reconstruction

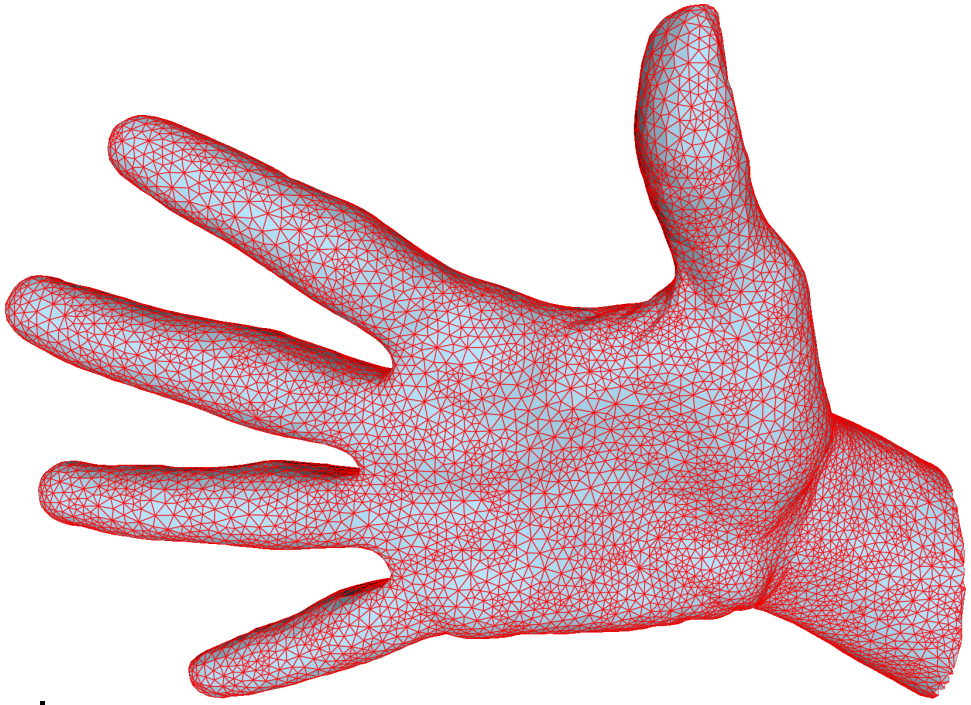
Part 1

Meshes, mostly, and the operators that work on them

Polygonal Meshes

Triangle meshes to be honest

- A common representation
- Efficient for computer graphics
- Usually the end result of optical acquisition processes
- Because acquired: noisy
- Mesh vertices can be seen as grid points and used to store function values
- Each mesh has its own connectivity

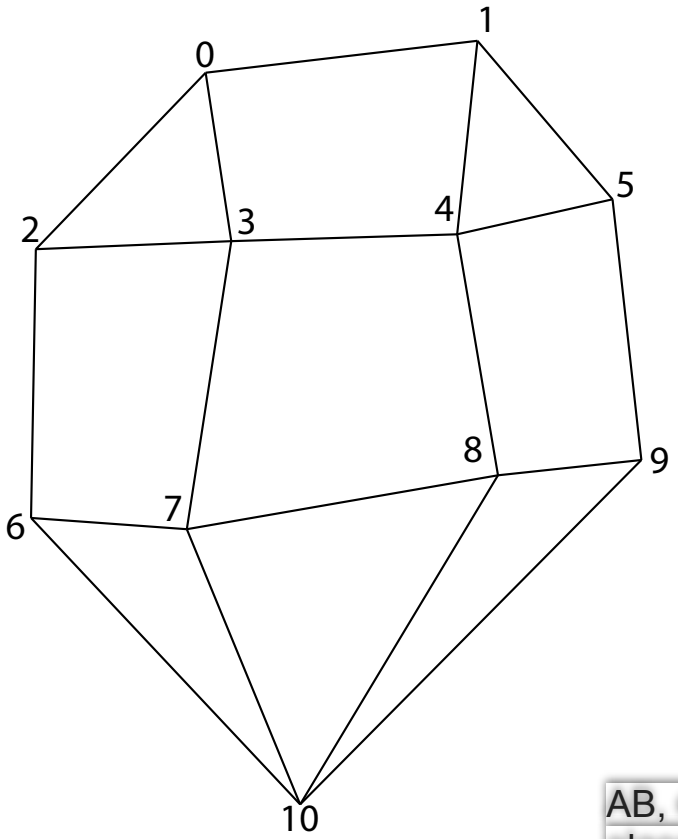


Representation of Polygonal Meshes

How do store a mesh in a computer system?!

- Indexed face Set
- Edge-based data structures

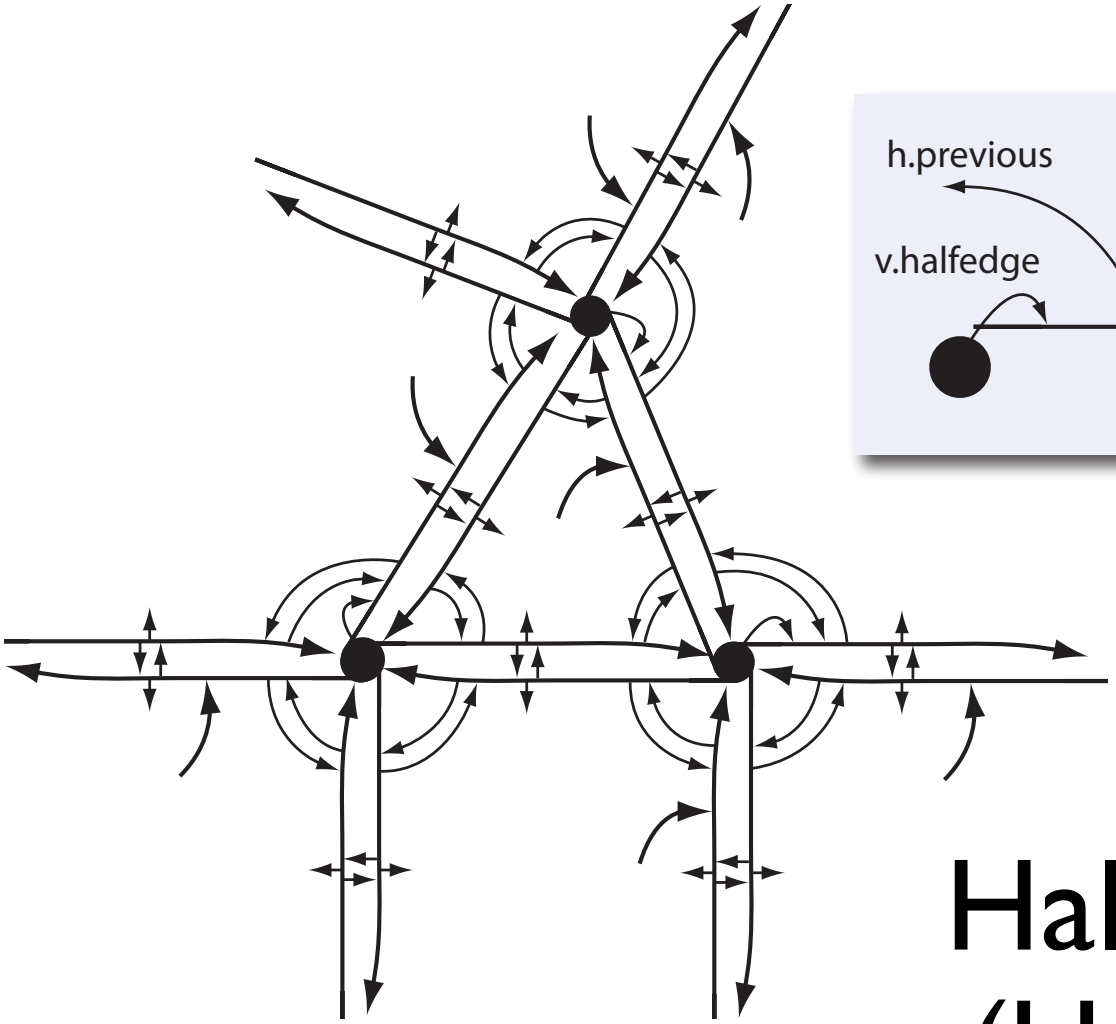
Indexed Face Set

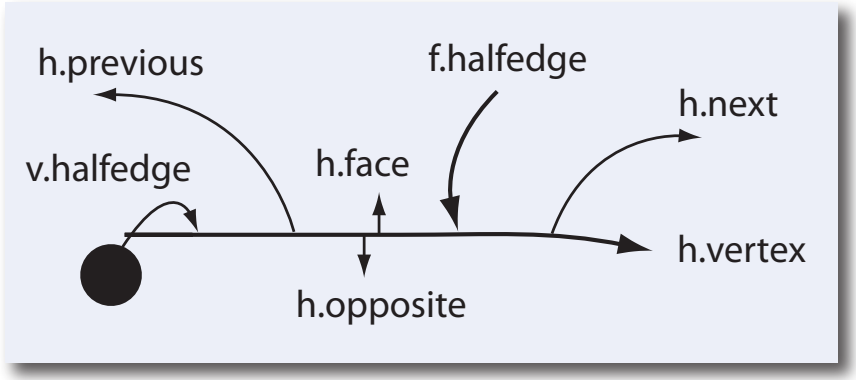


VERTICES
0: (-0.2, 1.5, 0)
1: (1.3, 1.7, 0)
2: (-1.1, 0.4, 0)
3: (0.0, 0.45, 1)
4: (1.1, 0.5, 1.2)
5: (2.1, 0.75, 0.2)
6: (-1.2, -1,0.01)
7: (-0.3, -1.2,2)
8: (1.3,-0.9, 3)
9: (2.0 -0.8,1.2)
10: (0.4, -2.1, -1.1)

FACES
0: 0,2,3
1: 0,3,4,1
2: 1,4,5
3: 2,6,7,3
4: 3,7,8,4
5: 4,8,9,5
6: 6,10,7
7: 7,10,8
8: 8,10,9

AB, et al. Guide to computational geometry processing: foundations, algorithms, and methods. Springer Science & Business Media, 2012.





Half Edge (Hmesh)

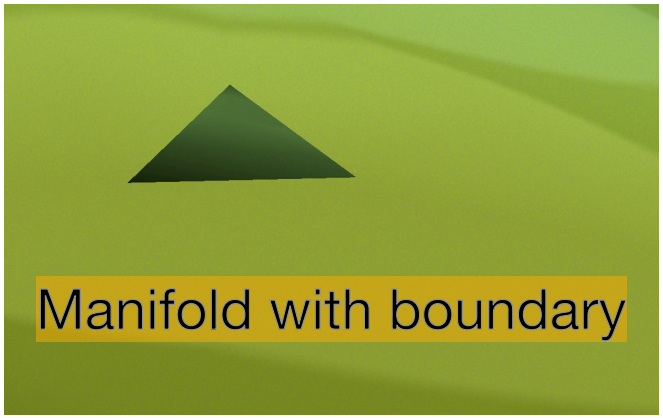
AB, et al. Guide to computational geometry processing: foundations, algorithms, and methods. Springer Science & Business Media, 2012.

Manifolds

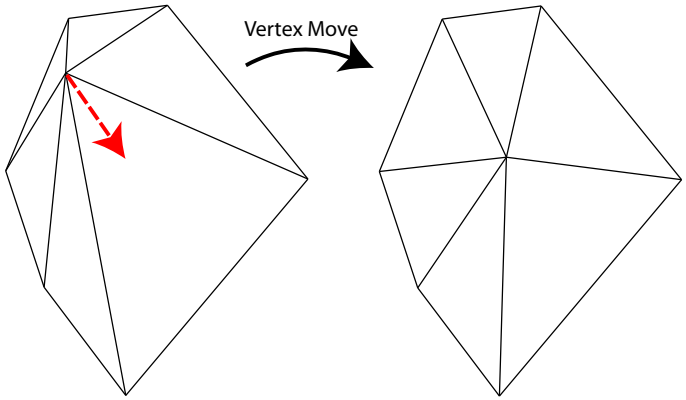
- An n-manifold is everywhere locally mappable to $\mathbb{R}^n$
- A manifold is covered by charts, $\phi : M \to U \subset \mathbb{R}^n$, forming an atlas, $\{\phi\}$ where ϕ is a homeomorphism, i.e.
  - one-to-one and onto
  - continuous with continuous inverse
- A manifold can be embedded in a space of higher dimension
  - When we think of surfaces, we often think of 2-manifolds embedded in $\mathbb{R}^3$
  - We often need maps from the 2-manifold into a 2D *parametrization* and back

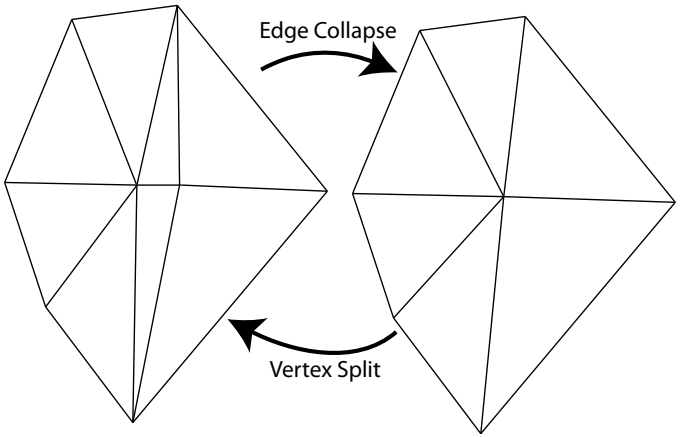
Half Edge Mesh and Manifoldness

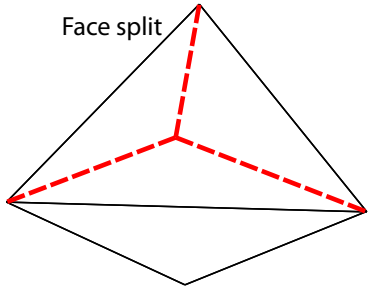
- A mesh is a 2-manifold if
  - Every edge is adjacent to two triangles
  - Triangles incident on a vertex form a single fan
- The half edge representation can only represent manifold meshes
- … along with meshes that are not manifold but close enough …

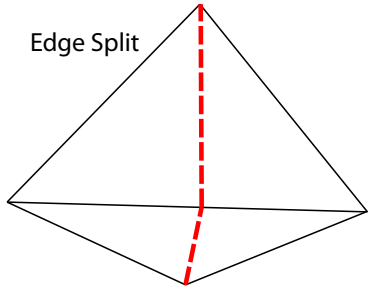


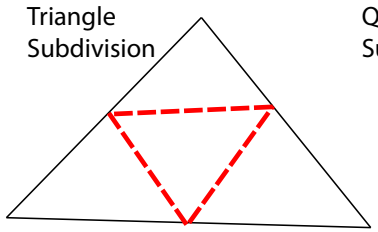
Things we can do with a mesh

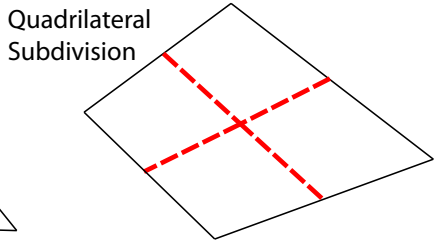


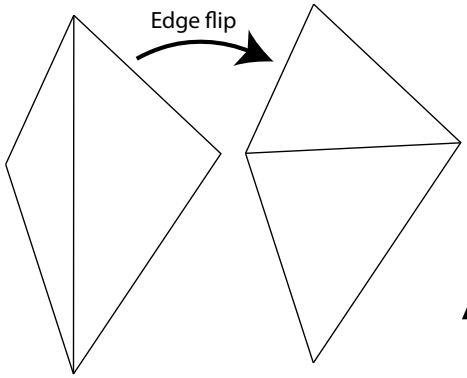






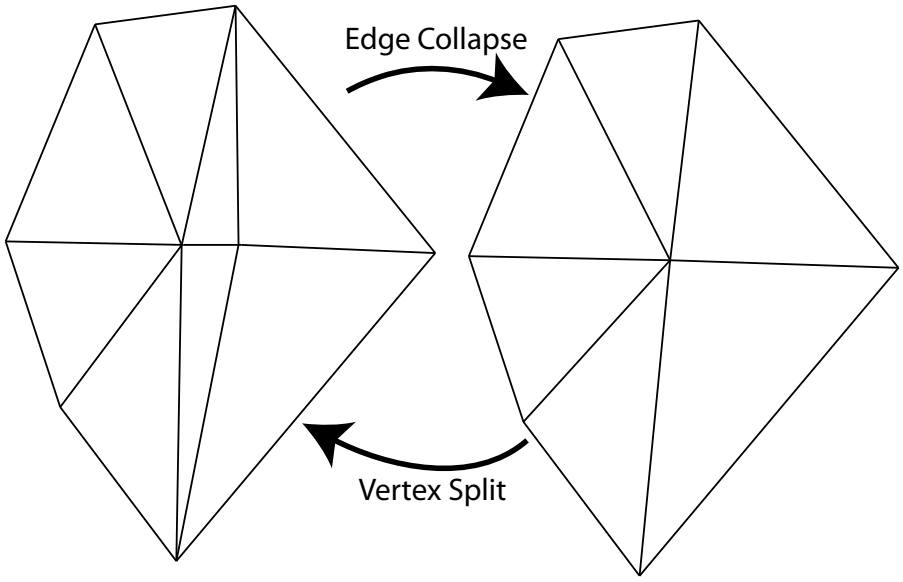


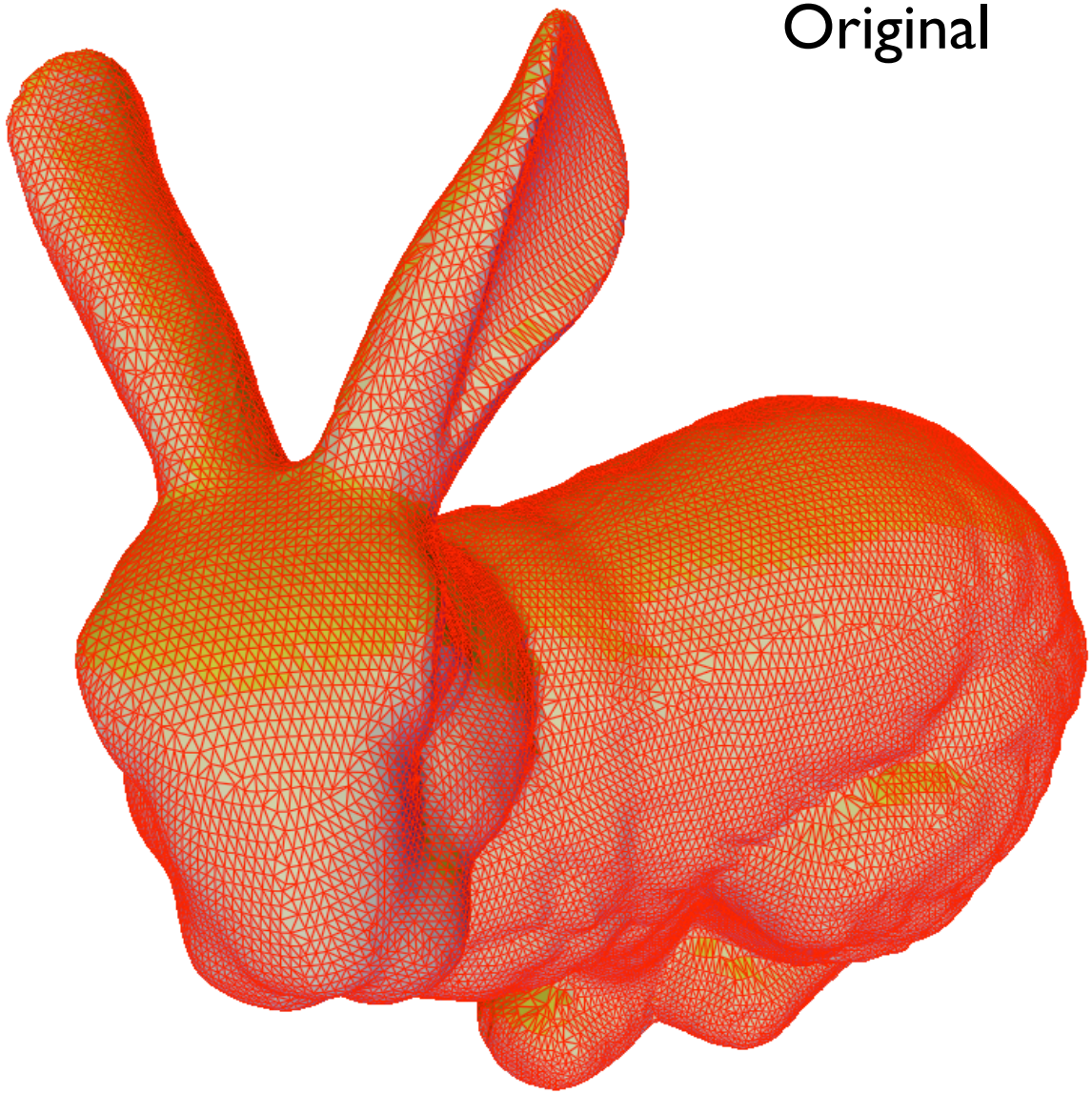




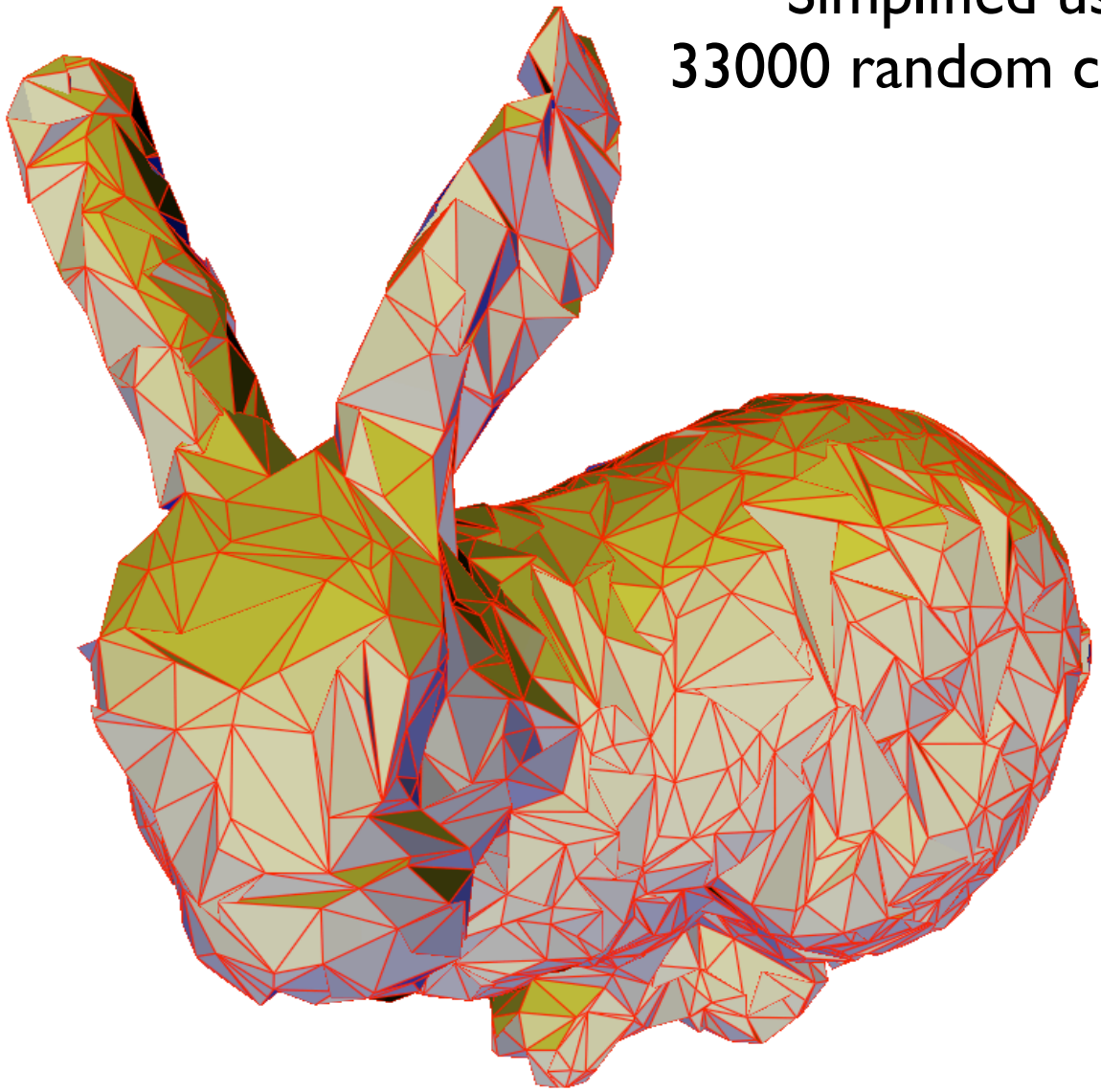
And more ...

Simplifying by Collapsing Edges

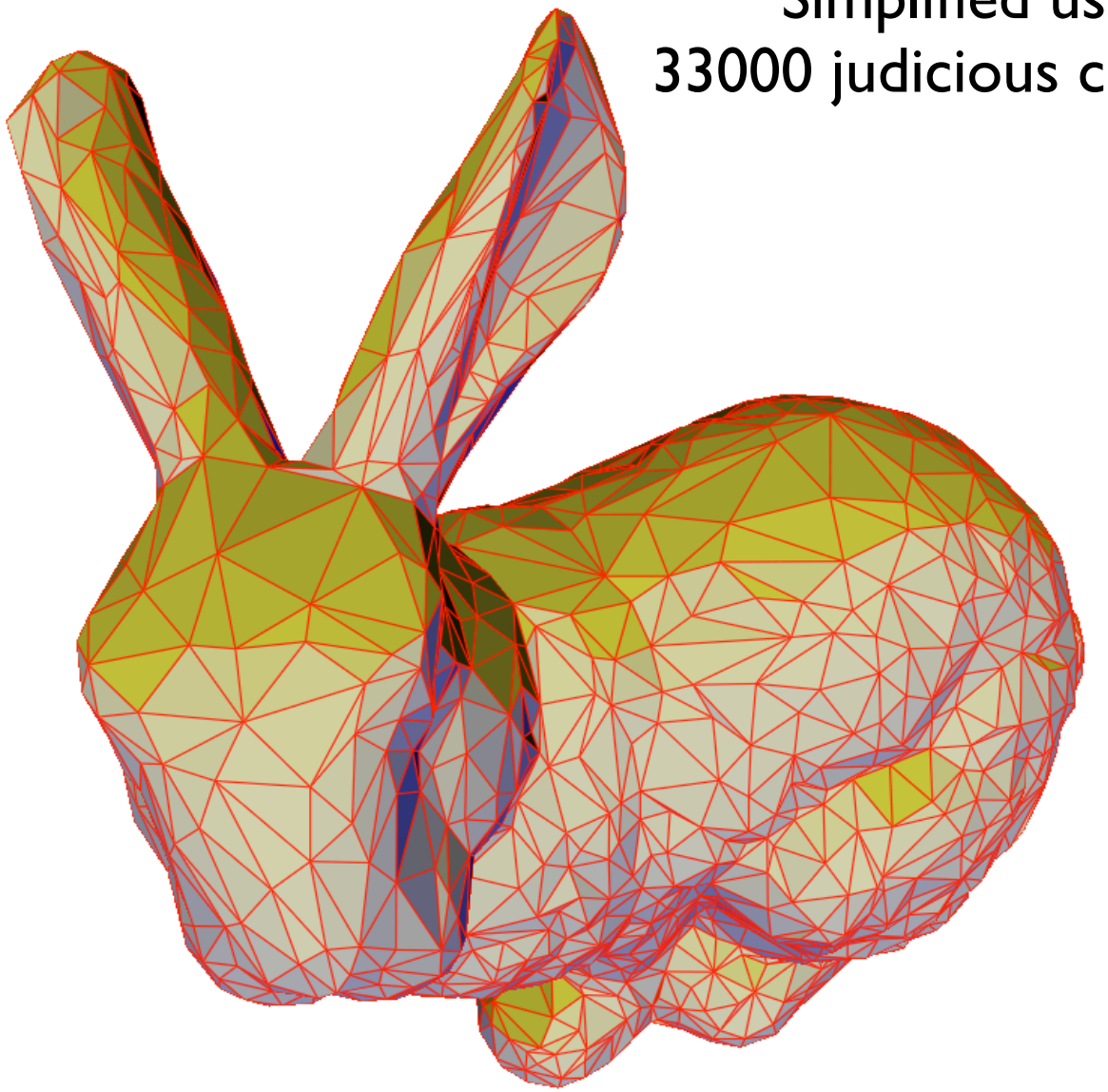



Original

Simplified using
33000 random collapses



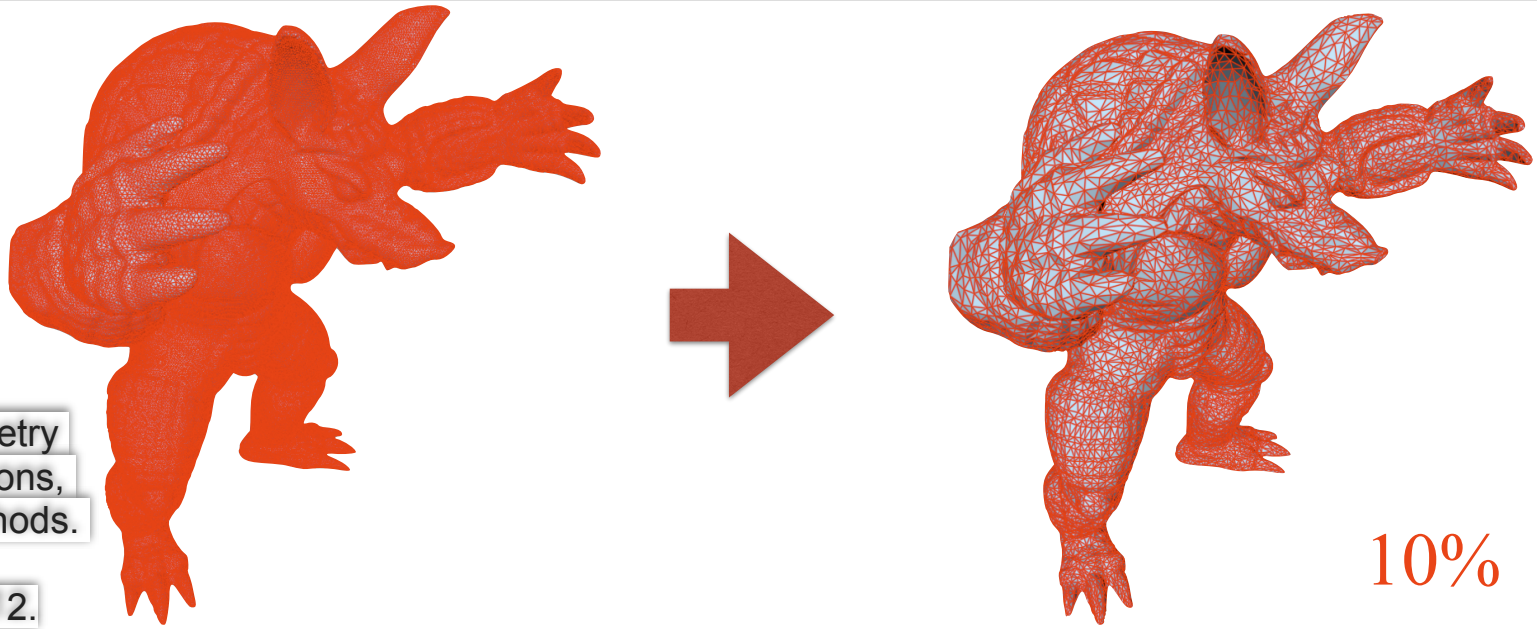
Simplified using
33000 judicious collapses



Edge Collapse Simplification

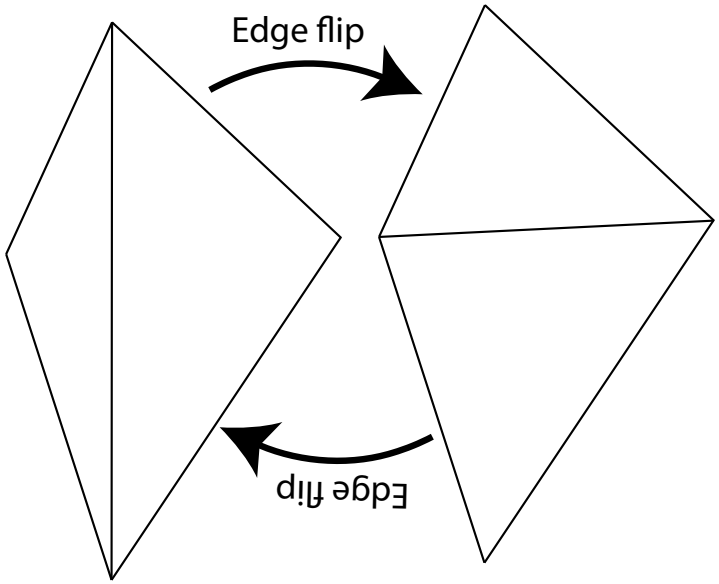
Works in PyGEL

1. For each edge we store the cost of a collapse in a priority queue.
2. Extract the collapse which is cheapest and perform it. This removes an edge along with its adjacent triangles. Its two vertices are merged into a single vertex, possibly with a new position.
3. Recompute the collapse cost for all edges affected by the collapse and update their position in the priority queue.
4. Until the stop condition is met, we go to 2.

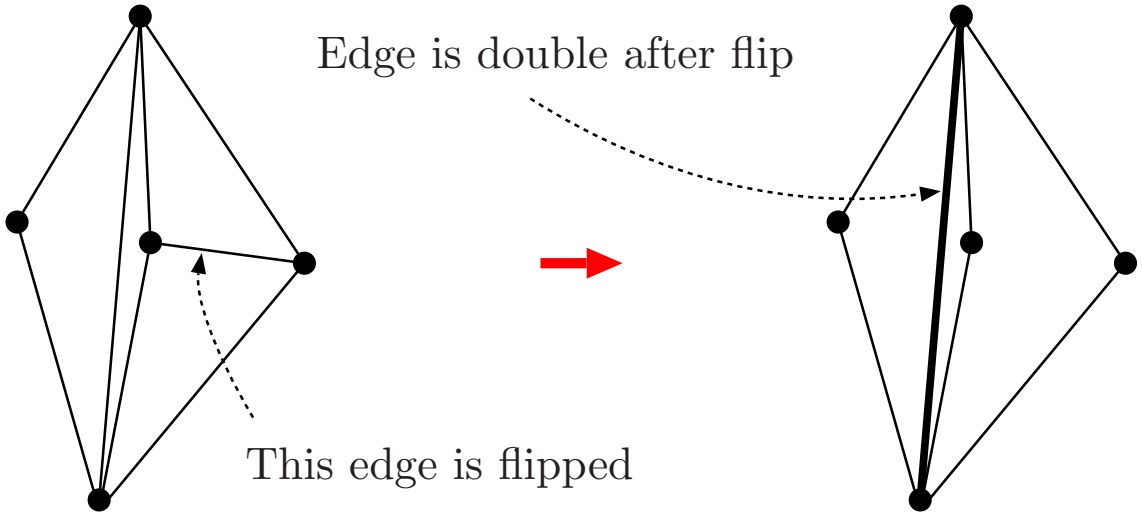


AB, et al. Guide to computational geometry processing: foundations, algorithms, and methods. Springer Science & Business Media, 2012.

Optimizing by Edge Flipping



Degeneracies can arise



We cannot flip an edge if either end-point is valence 3
(or 2 at the boundary)

Mesh Optimization

1. Initially, ΔF is computed for all edges, and for each edge e a pair $< \Delta F(e), e >$ is inserted into a priority queue if $\Delta F(e) < 0$.
2. The next step is a loop where we iteratively extract and remove the record with the ΔF corresponding to the greatest decrease in energy from the heap and flip the corresponding edge.
3. After an edge flip, $\Delta F(e')$ must be recomputed for any edge e' if its $\Delta F(e')$ has changed as the result of e being flipped.
4. The loop continues until the priority queue is empty.

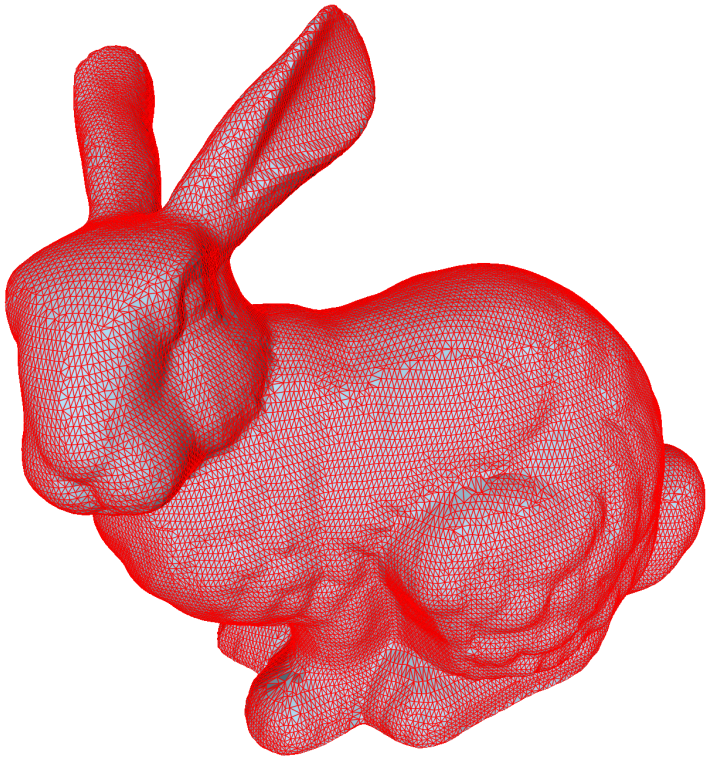
AB, et al. Guide to computational geometry processing: foundations, algorithms, and methods. Springer Science & Business Media, 2012.

Examples of Energy Functions

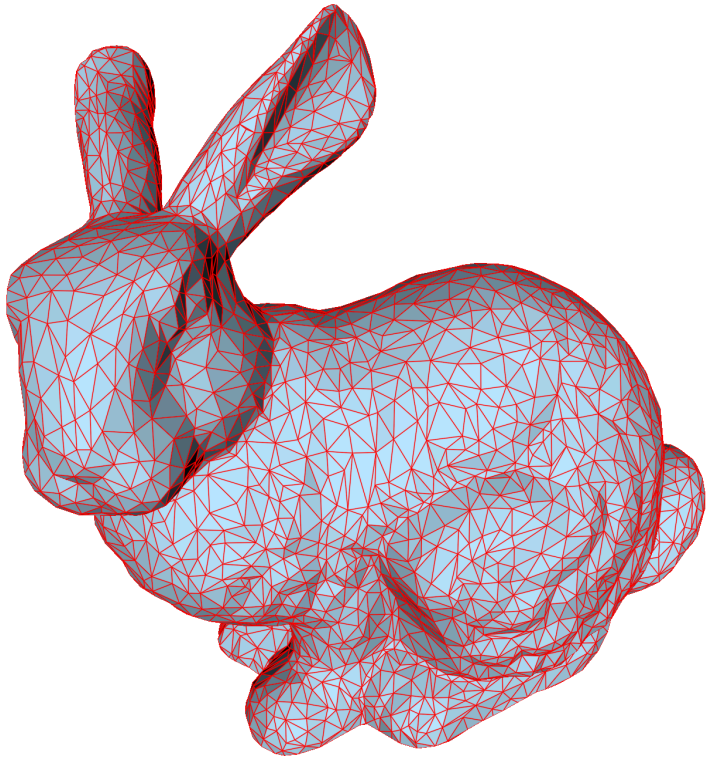
- Maximize minimum angle. Makes mesh Delaunay
- Minimize absolute dihedral angle (times length)
  - This is really curvature minimization
- Minimize dihedral angle raised to some power.

Mesh Optimization

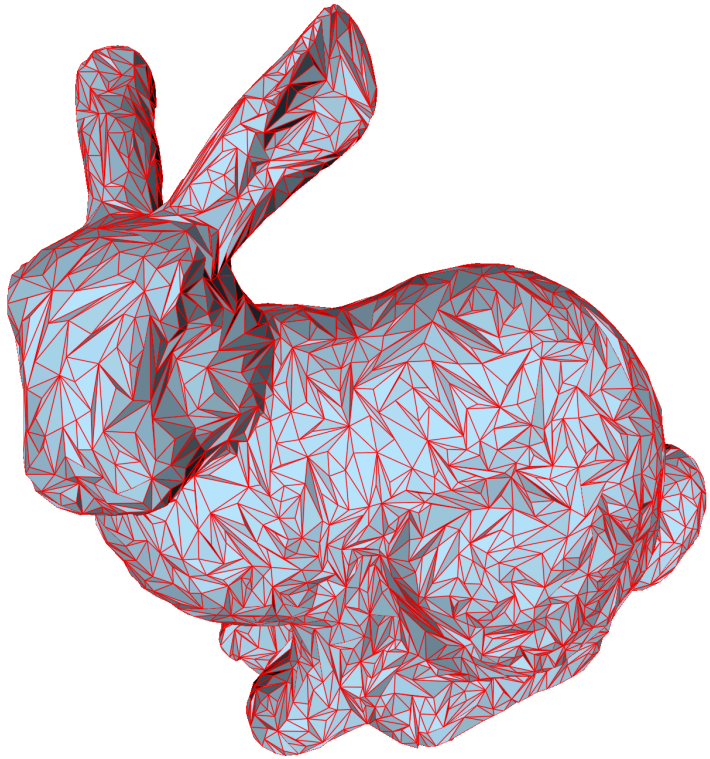
Works in PyGEL



Original



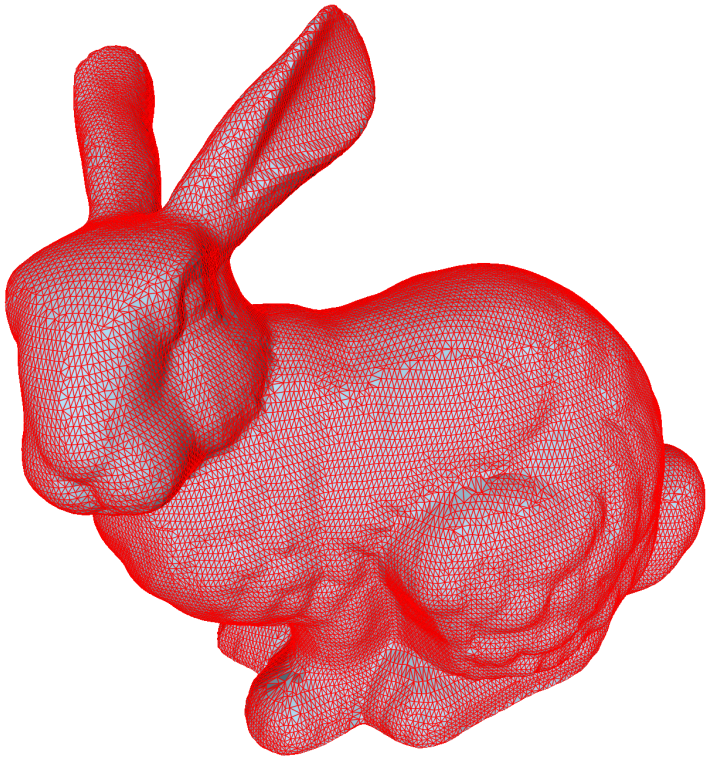
10%



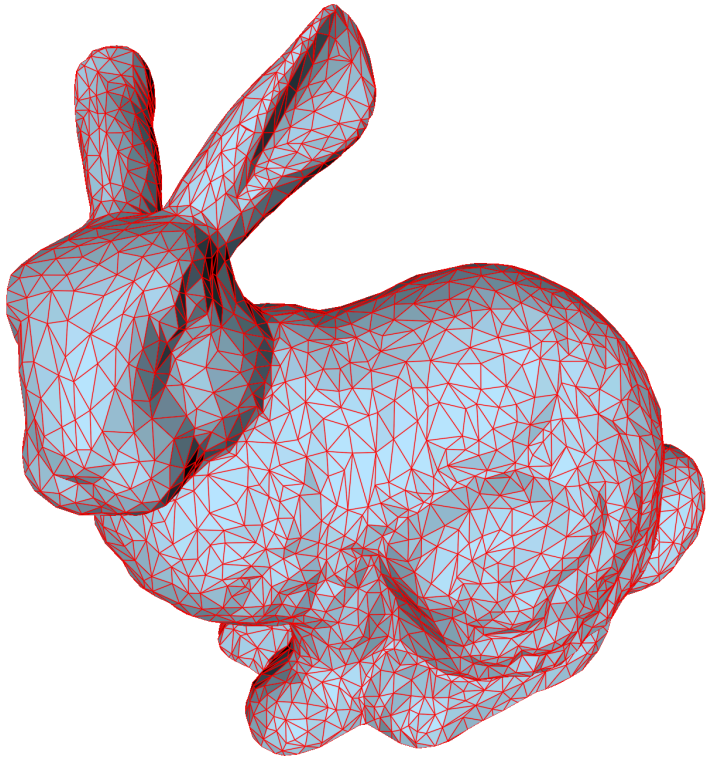
Connectivity perturbed

Mesh Optimization

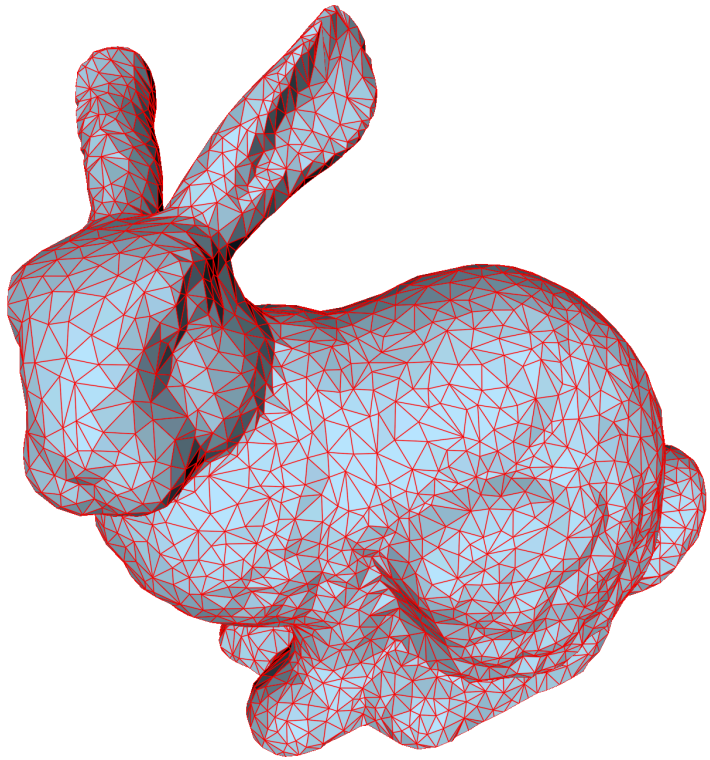
Works in PyGEL



Original



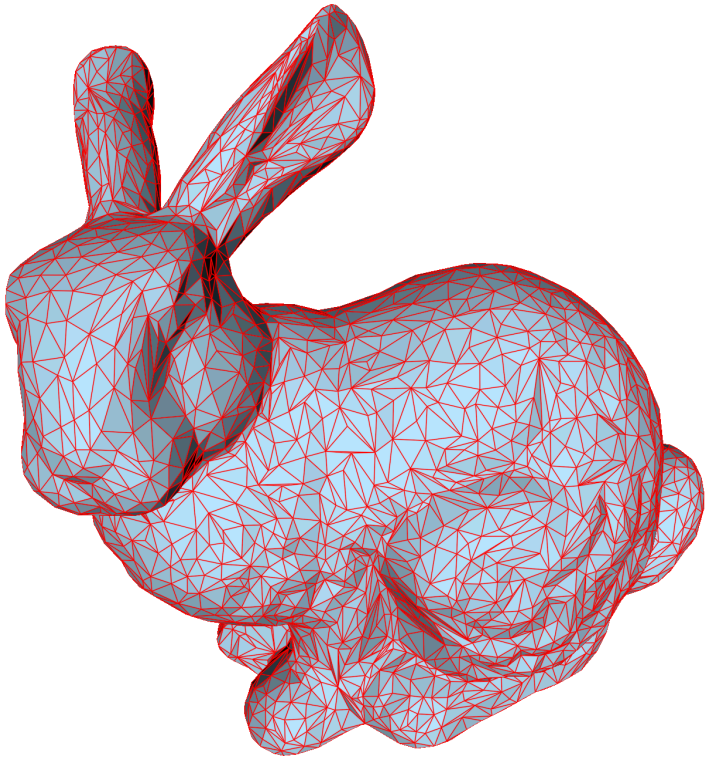
10%



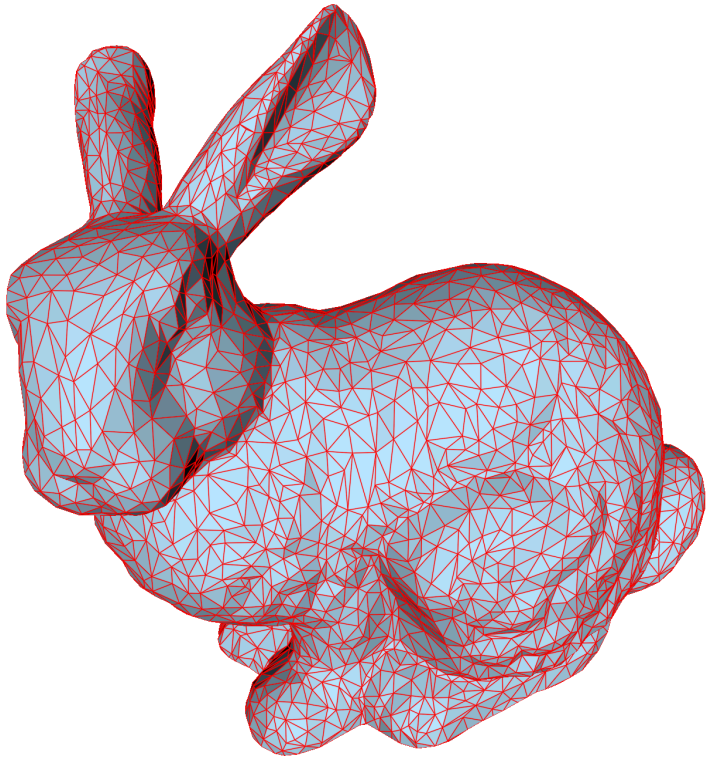
Maximized min angle

Mesh Optimization

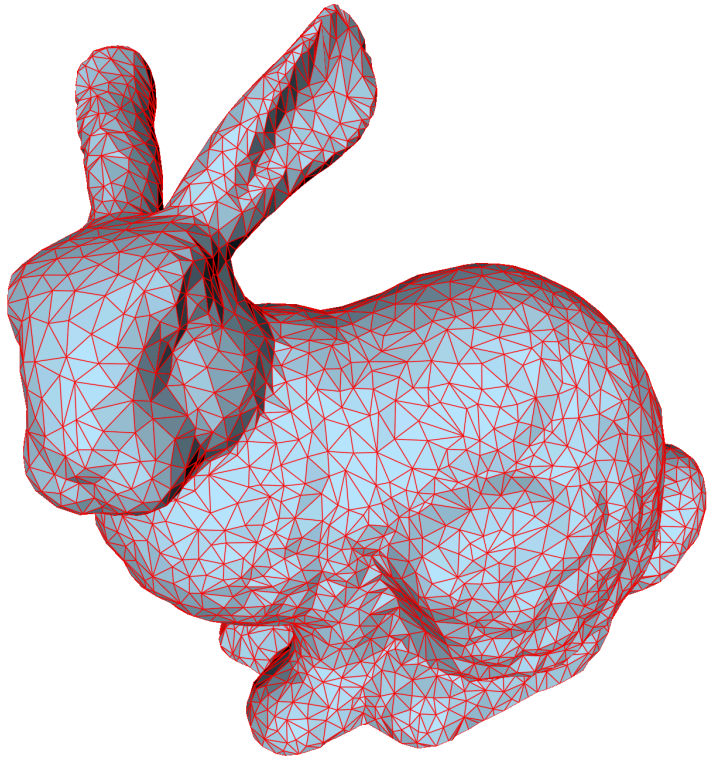
Works in PyGEL



Minimized dihedral angle



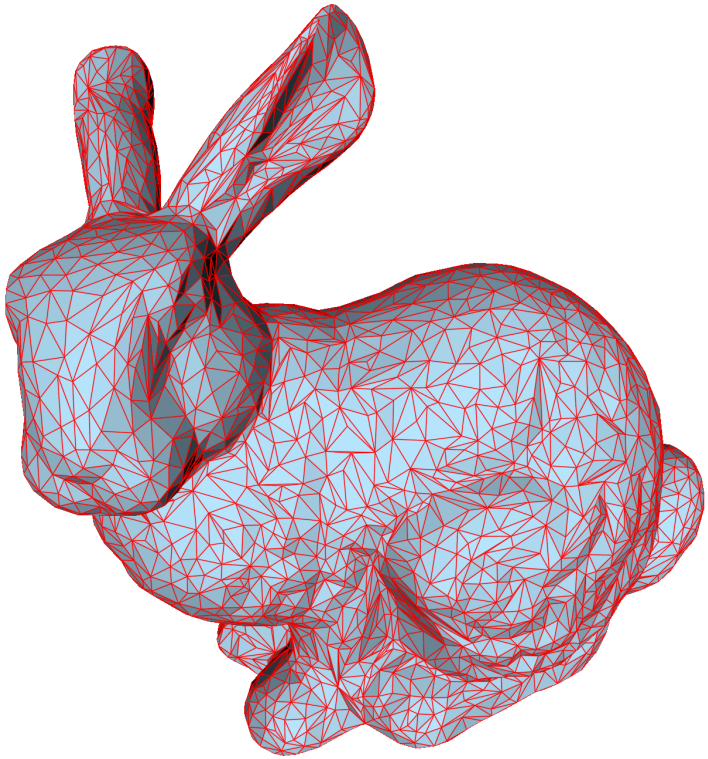
10%



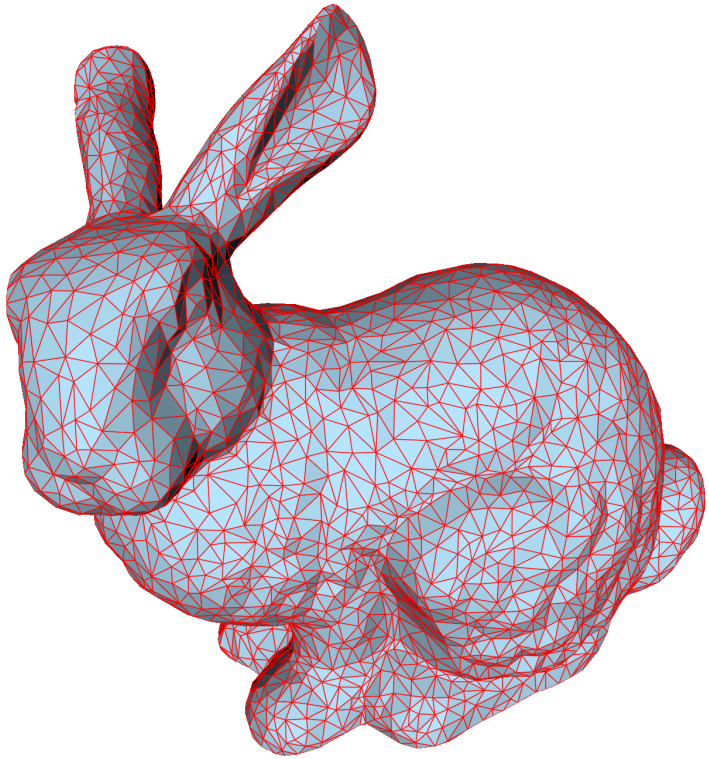
Maximized min angle

Mesh Optimization

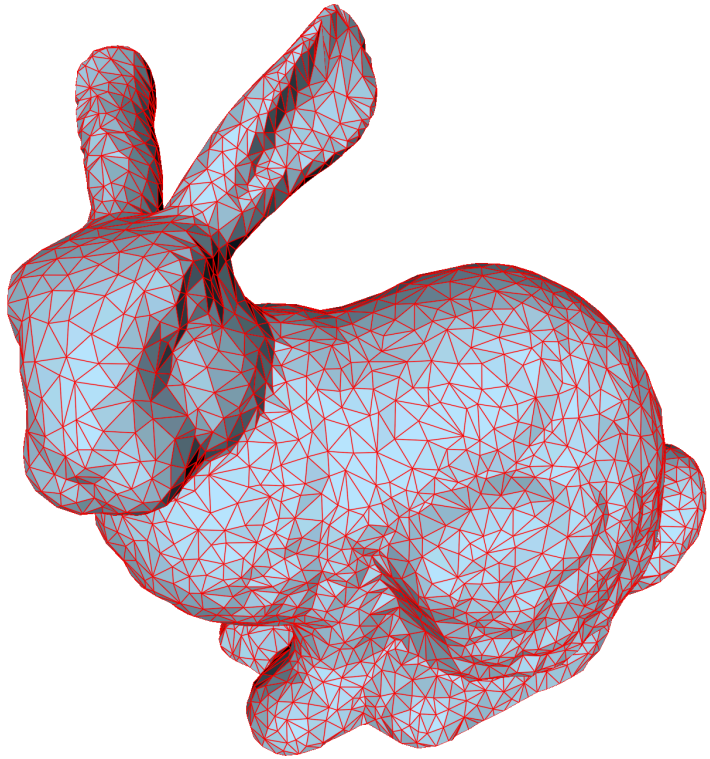
Works in PyGEL



Minimized dihedral angle



Both



Maximized min angle

Linear Functions on Triangle Meshes

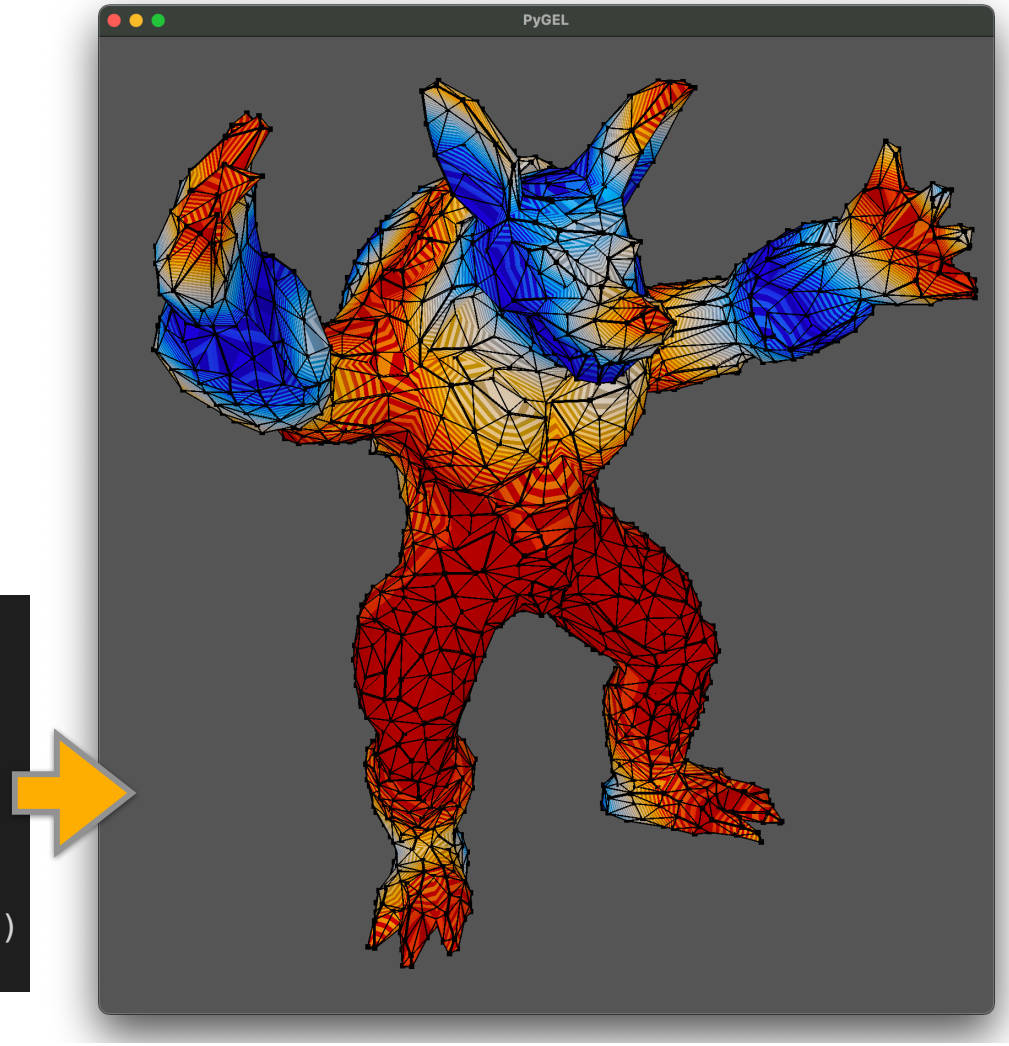
A practical example

Works in PyGEL

- f is stored as a value per vertex
- Linearly interpolated for visualization
- Down and dirty with PyGEL:

```
from pygel3d import hmesh, graph, gl_display as gl
from numpy import array
from math import cos

m = hmesh.load("armadillo-very-simple.obj")
g = graph.from_mesh(m)
P = m.positions()
V = gl.Viewer()
f = array([ cos(P[v][2]*P[v][1]/500) for v in m.vertices()])
V.display(m,g,mode='s',data=f)
```



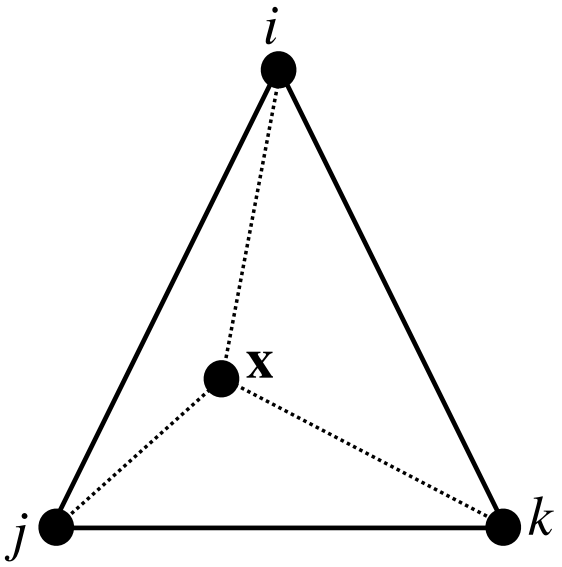
Linear Functions on Triangle Meshes

Barycentric coordinates

- Given a function, f, defined on the vertices of a triangle mesh,
- we use barycentric coordinates to extend it to any point, i.e.

  $f(\mathbf{x}) = b_i f_i + b_j f_j + b_k f_k$, where

  - $\mathbf{x} = b_i \mathbf{p}_i + b_j \mathbf{p}_j + b_k \mathbf{p}_k$ and
  - $b_i + b_j + b_k = 1$
- Barycentric coordinates
  - are easy to compute as a ratio of triangle areas
  - can be used to interpolate quantities stored at vertices
  - $b_i(\mathbf{x})$, $b_j(\mathbf{x})$, and $b_k(\mathbf{x})$ are functions of $\mathbf{x}$ and form a linear basis



Linear Functions on Triangle Meshes

The Umbrella Operator

- The umbrella applied to f is

  $(\Delta f)_i = \frac{1}{|N_i|} \sum_{j \in N_i} f_j - f_i$, where N_i are the neighbors of vertex i,

- as a matrix product,

  $(\Delta f)_i = (\mathbf{L} f)_i$, where $L_{ij} = \begin{cases} \frac{1}{|N_i|} & j \in N_i \\ -1 & j = i \\ 0 & \text{otherwise} \end{cases}$.

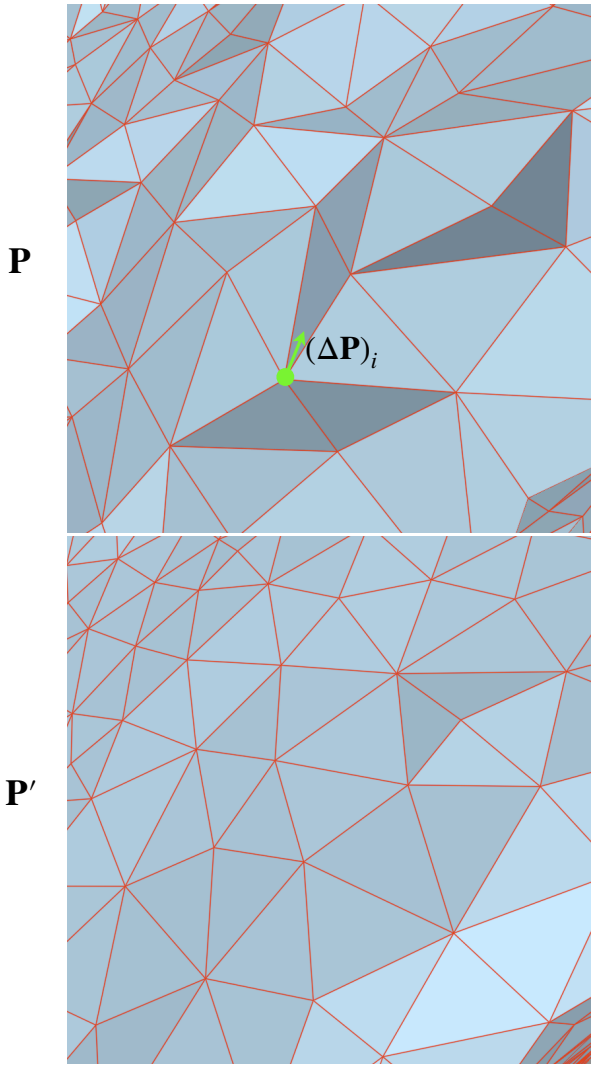
Linear Functions on Triangle Meshes

The Umbrella Operator

- Let $\mathbf{P}$ be an $n \times 3$ matrix of vertex positions
- We can compute new positions thusly:

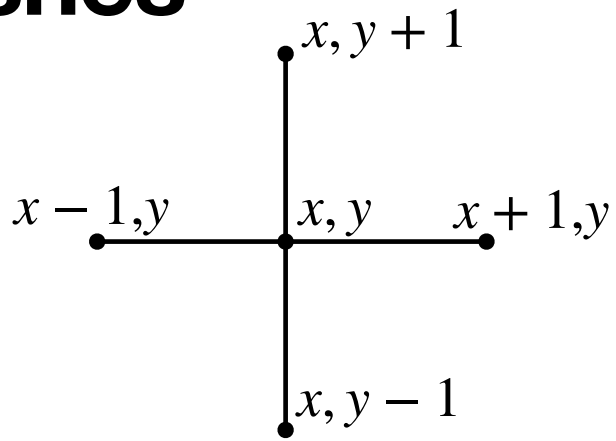
$$\mathbf{P}' = (\mathbf{I} + \mathbf{L})\mathbf{P}$$

- Umbrella applied to the vertex positions → smoothing
- Note: this is just replacing vertex with avg. of neighbors!
- Note: we could have applied this to *other functions* than vertex position!



Linear Functions on Triangle Meshes

The Umbrella Operator



- In 1D: $f''(0) \approx f(1) + f(-1) - 2f(0)$
- In 2D, $\Delta f \approx \dfrac{\partial^2 f}{\partial x^2} + \dfrac{\partial^2 f}{\partial y^2}$ and on a grid:

$$\Delta f(x,y) \approx f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 4f(x,y)$$

$$= \sum_{(a,b) \in N_{(x,y)}} f(a,b) - f(x,y)$$

- The umbrella operator is an approximation of the Laplacian

The Laplace Beltrami Operator

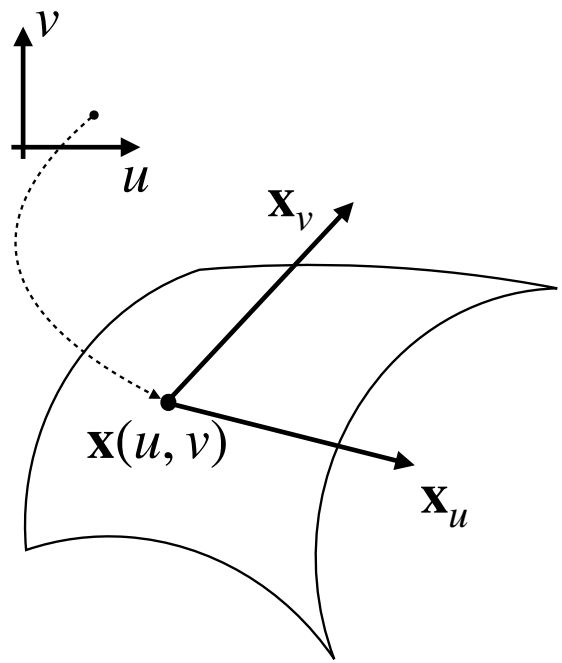
Enter the parametrization

- The umbrella operator assumes neighbors distributed evenly on unit disk in the parameter domain, but
- The *Laplace Beltrami Operator* includes the surface metric in its definition:

$$\Delta f = \nabla \cdot \nabla f = \frac{1}{\sqrt{|g_{ij}|}} \partial_i \left(\sqrt{|g_{ij}|}\, g^{ij} \partial_j f \right)$$

where $g_{ij} = \mathbf{x}_i \cdot \mathbf{x}_j$ is the surface metric

- If $\mathbf{x}$ is isometric, $g_{ij} = \mathbf{I}$



AB, et al. Guide to computational geometry processing: foundations, algorithms, and methods. Springer Science & Business Media, 2012.

The Laplace Beltrami Operator

Green's First Identity

- Consider the projection of Δf onto a basis function ψ (linear "tent" - one for each vertex)

$$\langle \Delta f, \psi_i \rangle = \sum_k \langle \Delta f, \psi_i \rangle_{T_k}$$

- Plugging into G.F.I.

$$\sum_k \langle \Delta f, \psi_i \rangle_{T_k} = \sum_k \langle \mathbf{N} \cdot \nabla f, \psi_i \rangle_{\partial T_k} - \sum_k \langle \nabla f, \nabla \psi_i \rangle_{T_k} = \boxed{- \sum_k \langle \nabla f, \nabla \psi_i \rangle_{T_k}}$$

- Note: inner products are

$$\langle f, g \rangle = \int_A fg \, \mathrm{d}A \text{ and } \langle \mathbf{X}, \mathbf{Y} \rangle = \int_A \mathbf{X}^T \mathbf{Y} \, \mathrm{d}A$$

Crane, Keenan. "Discrete differential geometry: An applied introduction." *Notices of the AMS, Communication* (2018): 1153-1159.

The Laplace Beltrami Operator

Expressing f in terms of the basis

- Requiring that

$$f = \sum_j f_j \psi_j$$

- We obtain:

$$\sum_k \langle \nabla f, \nabla \psi_i \rangle_{T_k} = \sum_k \left\langle \nabla \left(\sum_j f_j \psi_j \right), \nabla \psi_i \right\rangle_{T_k} = \sum_k \sum_j f_j \langle \nabla \psi_j, \nabla \psi_i \rangle_{T_k}$$

The Laplace Beltrami Operator

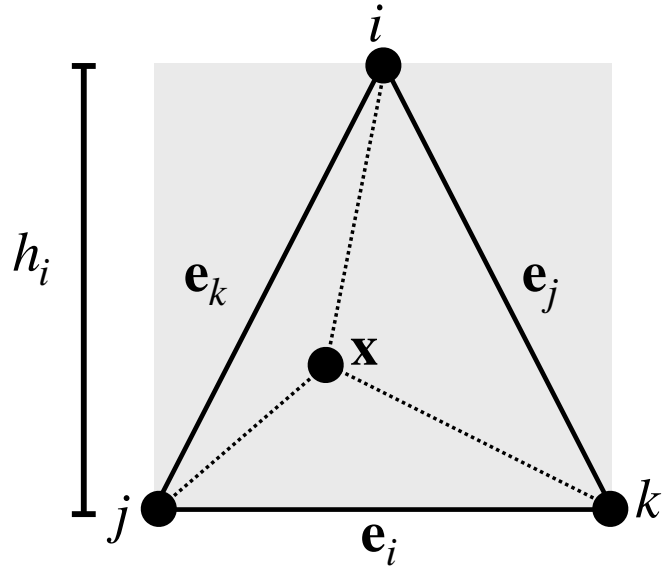
Gradients

- The gradient of a linearly interpolated f is constant over each triangle:

$$\nabla f(\mathbf{x}) = f_i \nabla b_i + f_j \nabla b_j + f_k \nabla b_k,$$

where $\nabla b_i = \mathbf{R}_{90} \dfrac{\mathbf{e}_i}{2A}$ and $2A = h_i \|\mathbf{e}_i\|$

Likewise for b_j and b_k



- Note:
$\mathbf{e}_i + \mathbf{e}_j + \mathbf{e}_k = 0 \implies \nabla b_i + \nabla b_j + \nabla b_k = 0$

The Laplace Beltrami Operator

Computing the inner products

- So what is $\langle \nabla\psi_j, \nabla\psi_i \rangle_{T_k}$?
- If $i \neq j$ but both i and j are in T_k

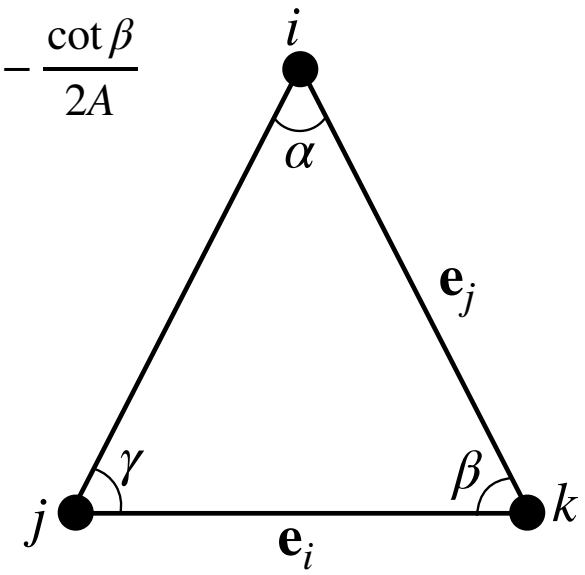
$$\nabla\psi_i \cdot \nabla\psi_j = \left(\mathbf{R}_{90}\frac{\mathbf{e}_i}{2A}\right) \cdot \left(\mathbf{R}_{90}\frac{\mathbf{e}_j}{2A}\right) = -\frac{\mathbf{e}_i \cdot -\mathbf{e}_j}{2A\|\mathbf{e}_i \times -\mathbf{e}_j\|} = -\frac{\cos\beta\|\mathbf{e}_i\|\|\mathbf{e}_j\|}{2A\sin\beta\|\mathbf{e}_i\|\|\mathbf{e}_j\|} = -\frac{\cot\beta}{2A}$$

- Hence, for each triangle

$$\langle \psi_i, \psi_j \rangle_{T_k} = \int_{T_k} -\frac{\cot\beta_{ij}}{2A}\mathrm{d}A = -\frac{1}{2}\cot\beta_{ij}$$

- And

$$\langle \nabla\psi_i, \nabla\psi_i \rangle_{T_k} = -\langle \nabla\psi_i, \nabla\psi_j + \nabla\psi_k \rangle_{T_k} = \frac{1}{2}(\cot\alpha_{jk} + \cot\gamma_{ki})$$



The Laplace Beltrami Operator

Assembling the operator

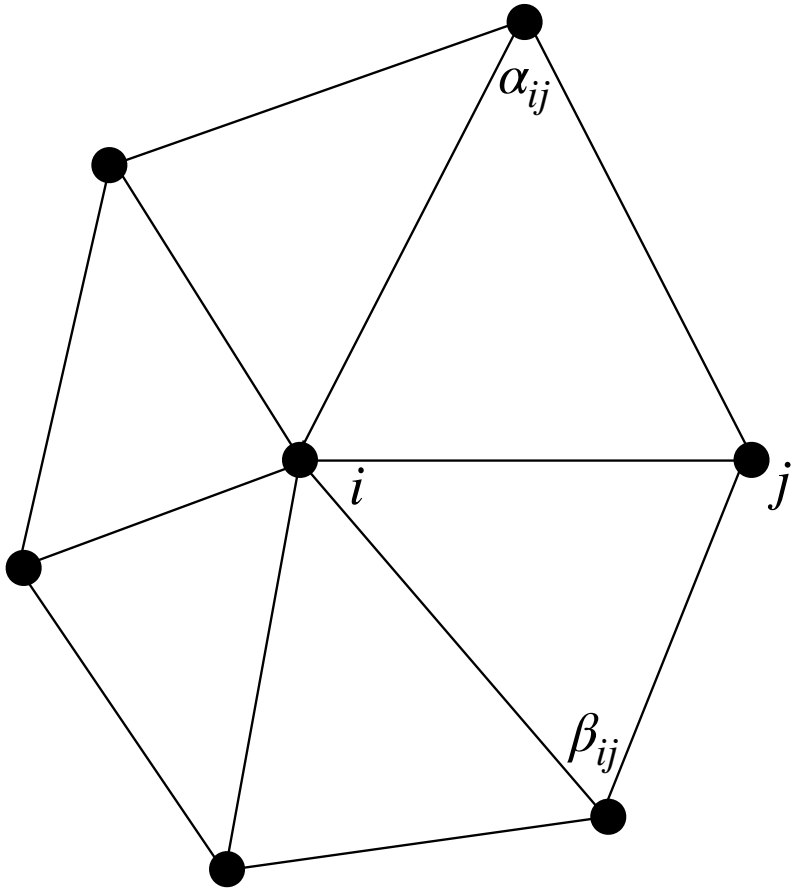
- For a vertex,

$$\langle \Delta f, \psi_i \rangle = \frac{1}{2} \sum_{j \in N_i} (\cot \alpha_{ij} + \cot \beta_{ij})(f_j - f_i)$$

- this is the well-known cotan formula due to Pinkall and Polthier [1993]. Normalizing, we obtain the discrete LBO

$$(\Delta f)_i \approx \frac{\langle \Delta f, \psi_i \rangle}{\langle 1, \psi_i \rangle} = \frac{1}{2A_i} \sum_{j \in N_i} (\cot \alpha_{ij} + \cot \beta_{ij})(f_j - f_i)$$

where $A_i = \frac{1}{3} \sum_{k \in T_i} A_k$, and A_k is the area of triangle k incident on i

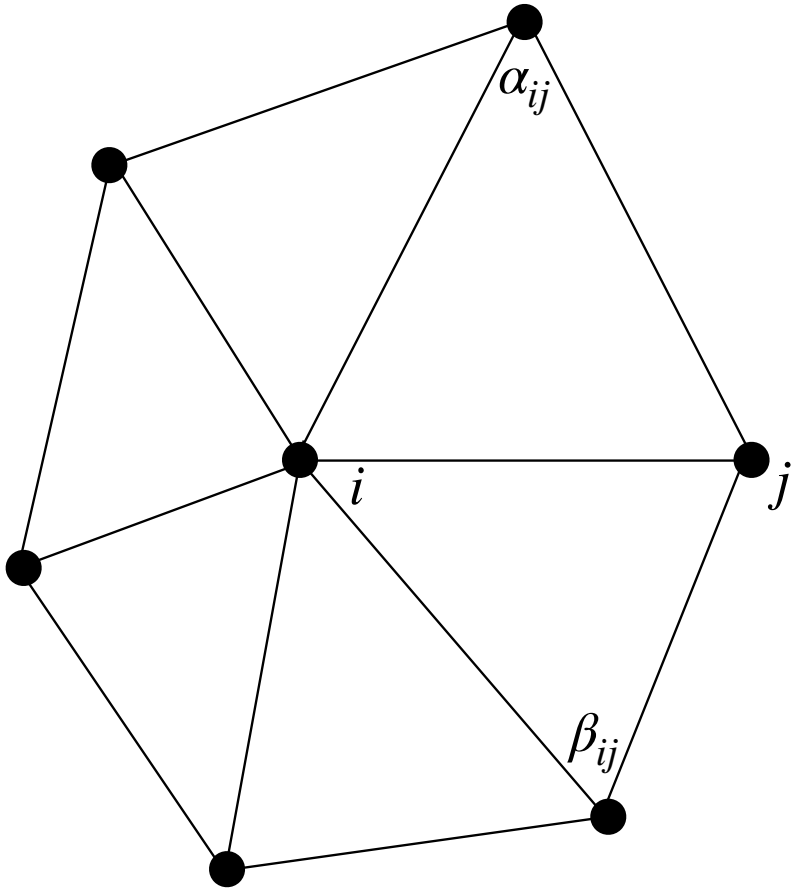


The Laplace Beltrami Operator

Now as a matrix

- The not-so-symmetric cot based LBO, $\mathbf{L}$, is

$$L_{ij} = \begin{cases} \dfrac{\cot(\alpha_{ij}) + \cot(\beta_{ij})}{2A_i} & \text{for } j \in N_i \\ -\sum_{k \in N_i} L_{ik} & \text{for } j = i \\ 0 & \text{otherwise} \end{cases}$$

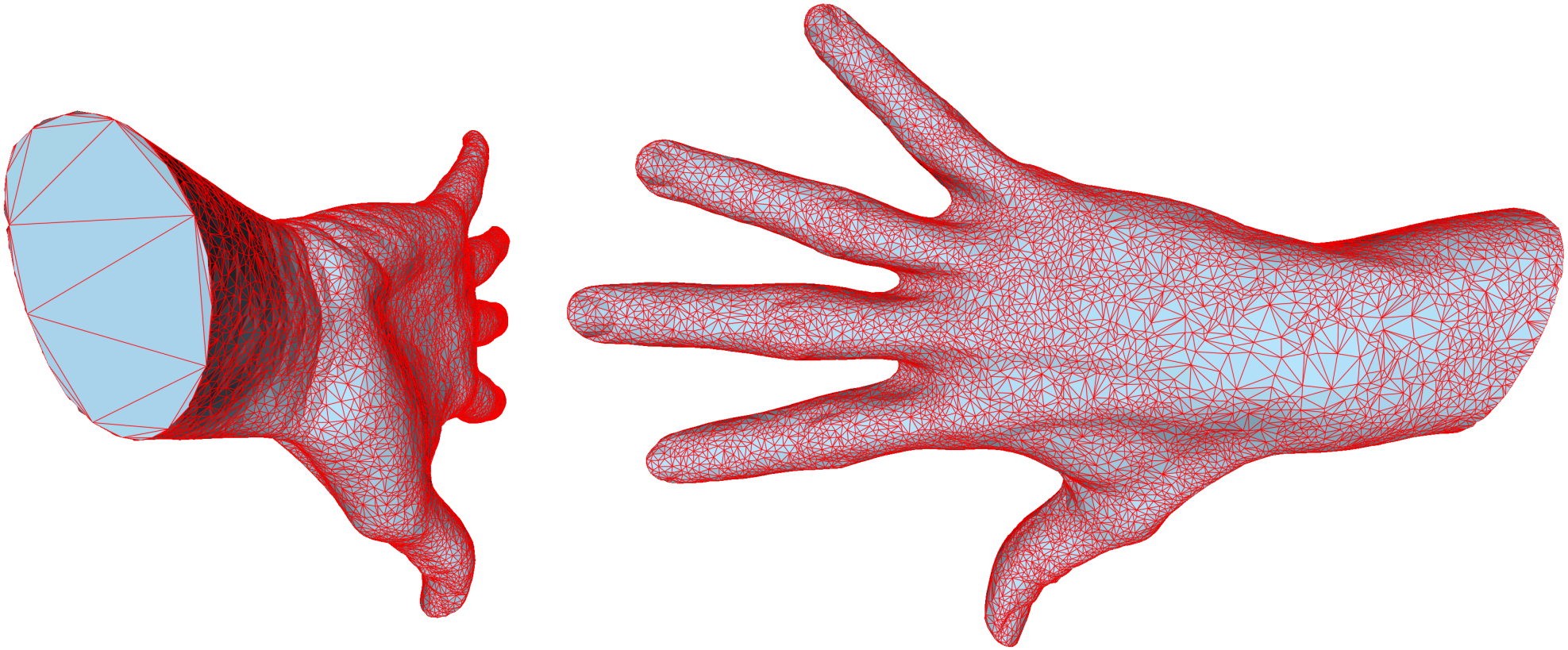


Smoothing

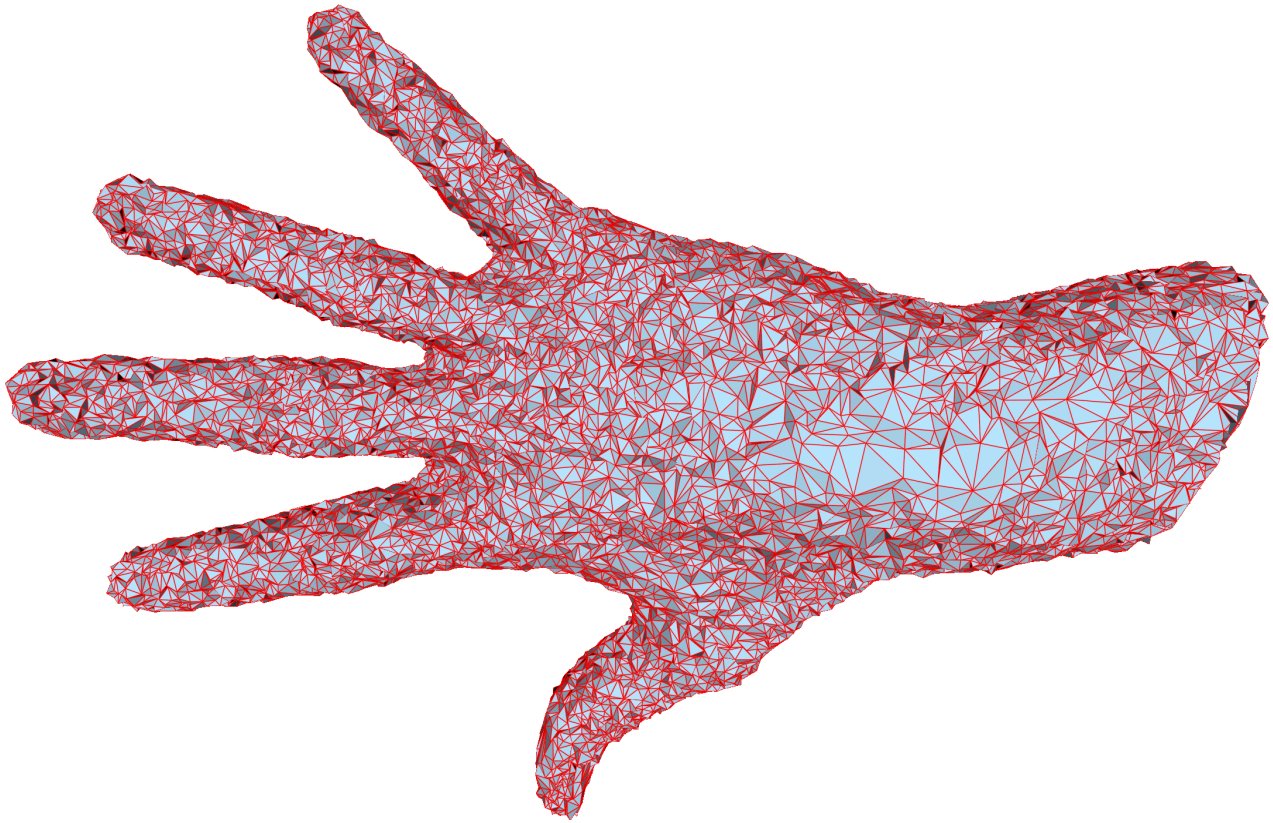
A shootout between several methods!

- We now have two discrete Laplace operators and many ways to use them
  - Explicit smoothing: $\mathbf{P}' = (\mathbf{I} + \lambda\mathbf{L})\mathbf{P}$
  - Implicit smoothing: $(\mathbf{I} - \lambda\mathbf{L})\mathbf{P}' = \mathbf{P}$
  - Taubin's method: $\mathbf{P}' = (\mathbf{I} - \mu\mathbf{L})(\mathbf{I} + \lambda\mathbf{L})\mathbf{P}$
  - Constrained Laplacian smoothing: smooth only in tangent plane
  - Geodesic smoothing: smooth in a $\log$ map
  - Feature preserving smoothing: very different, other trade-offs

Original Mesh

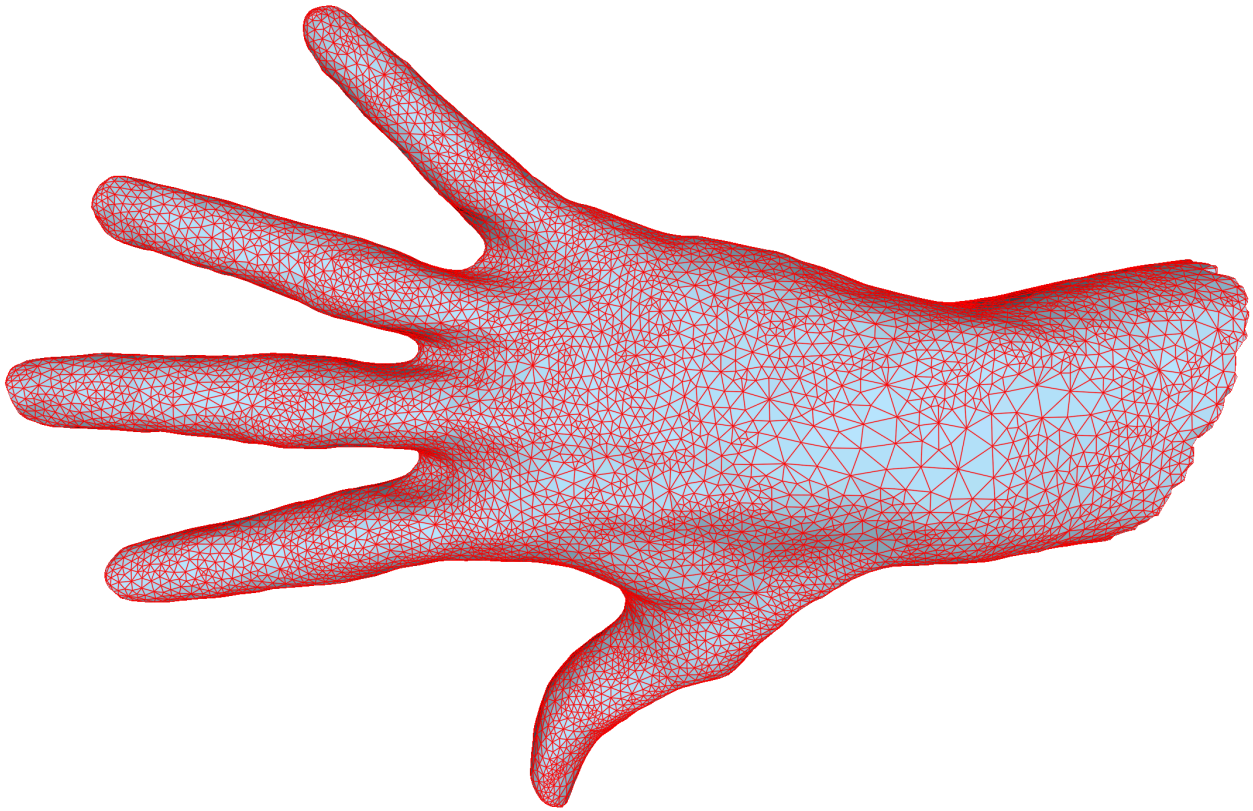


Noisy



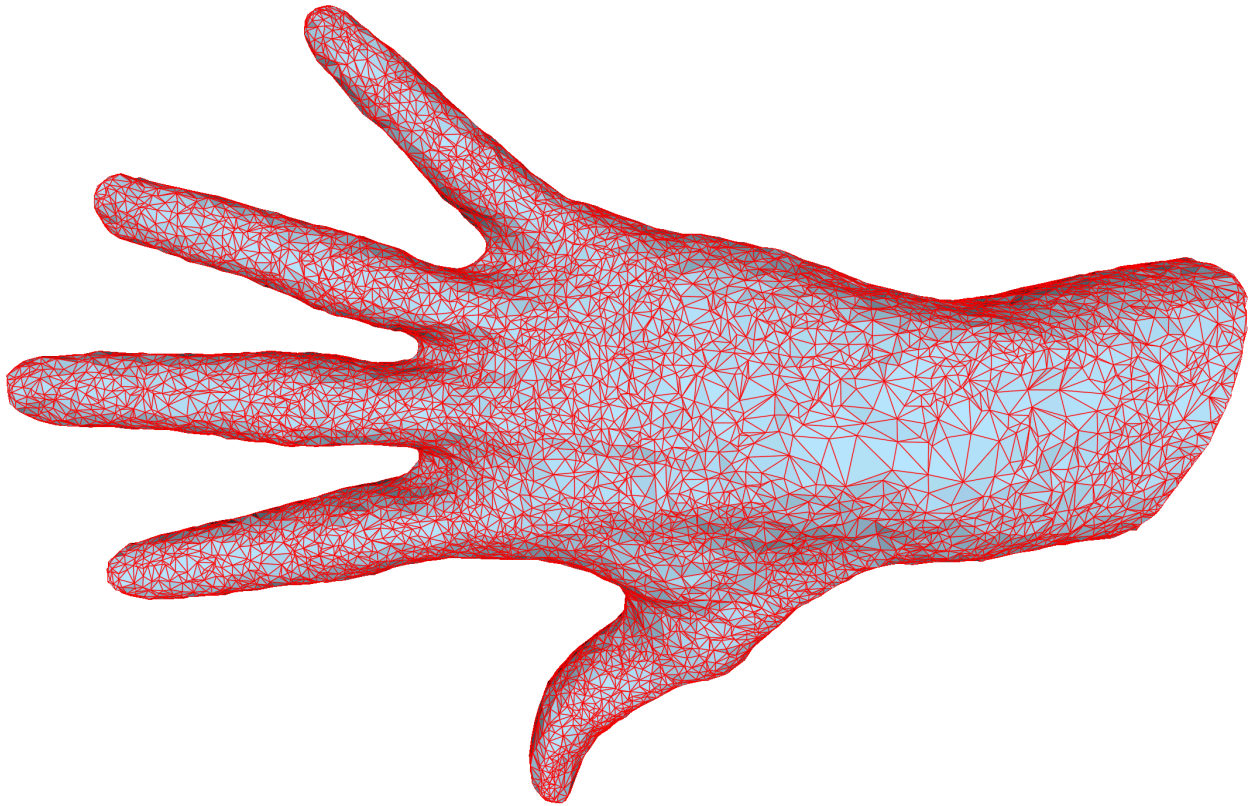
Laplacian Smoothing

$\lambda = 0.25$, iterations = 10, time < 0.009 seconds



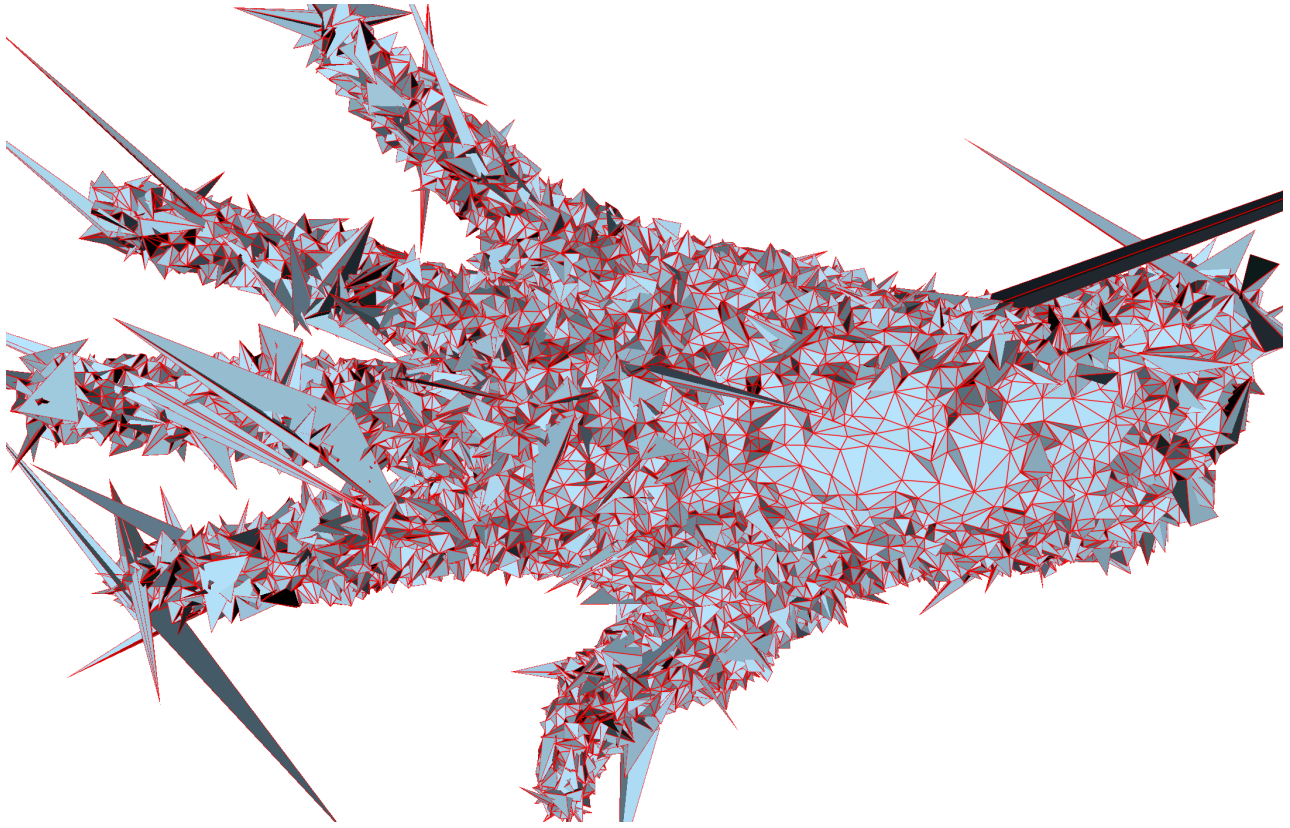
Smoothing with the LBO (mean curvature)

$\lambda = 3$, iterations = 1 (implicit), time = 2.1 seconds



Smoothing with the LBO (mean curvature)

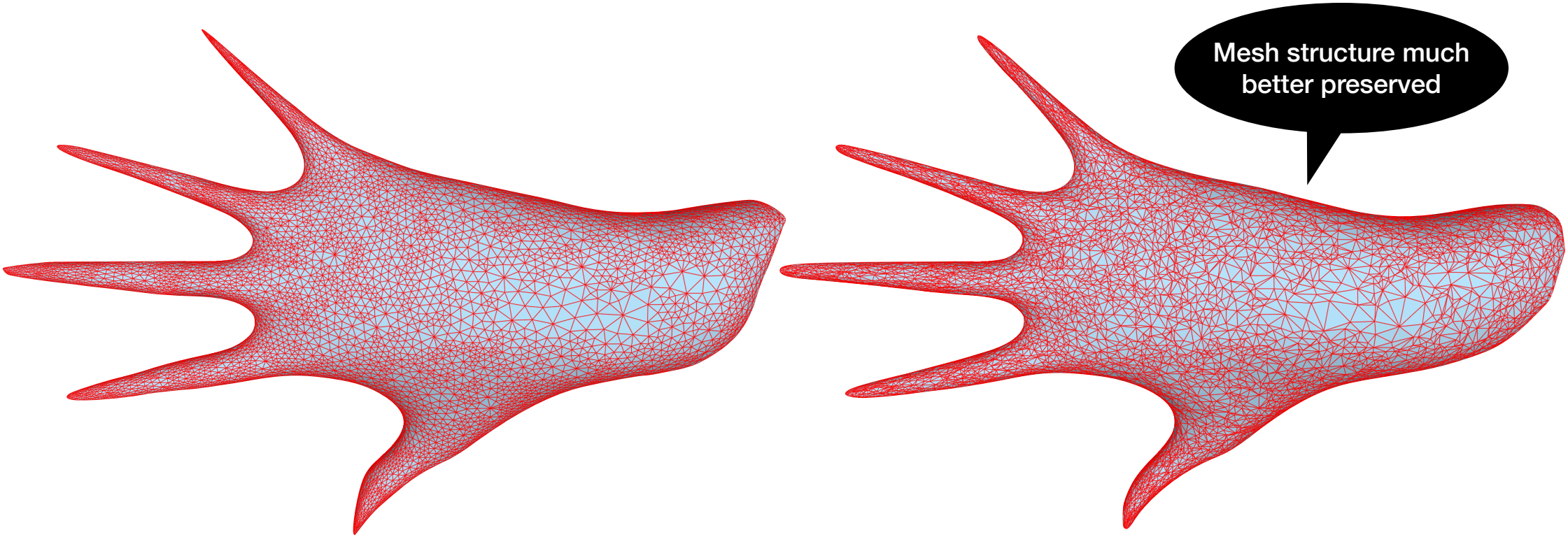
$\lambda = 3$, iterations = 1 (explicit), time = 2.1 seconds



Umbrella vs LBO

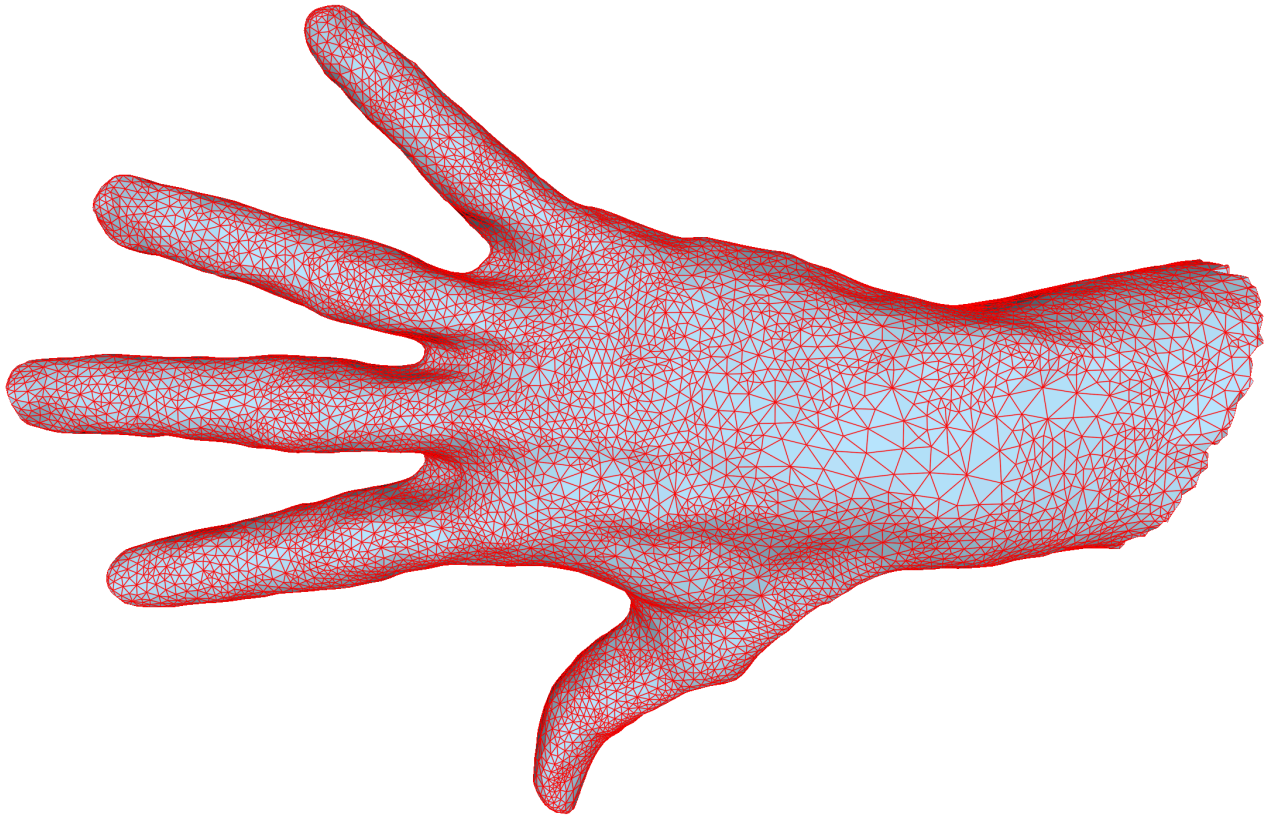
Left: Umbrella, $\lambda = 0.25$, iter=250, time = 0.11 seconds

Right: LBO, $\lambda = 100$, iter=1 (implicit), time = 2.1 seconds



Taubin Smoothing (w. Umbrella)

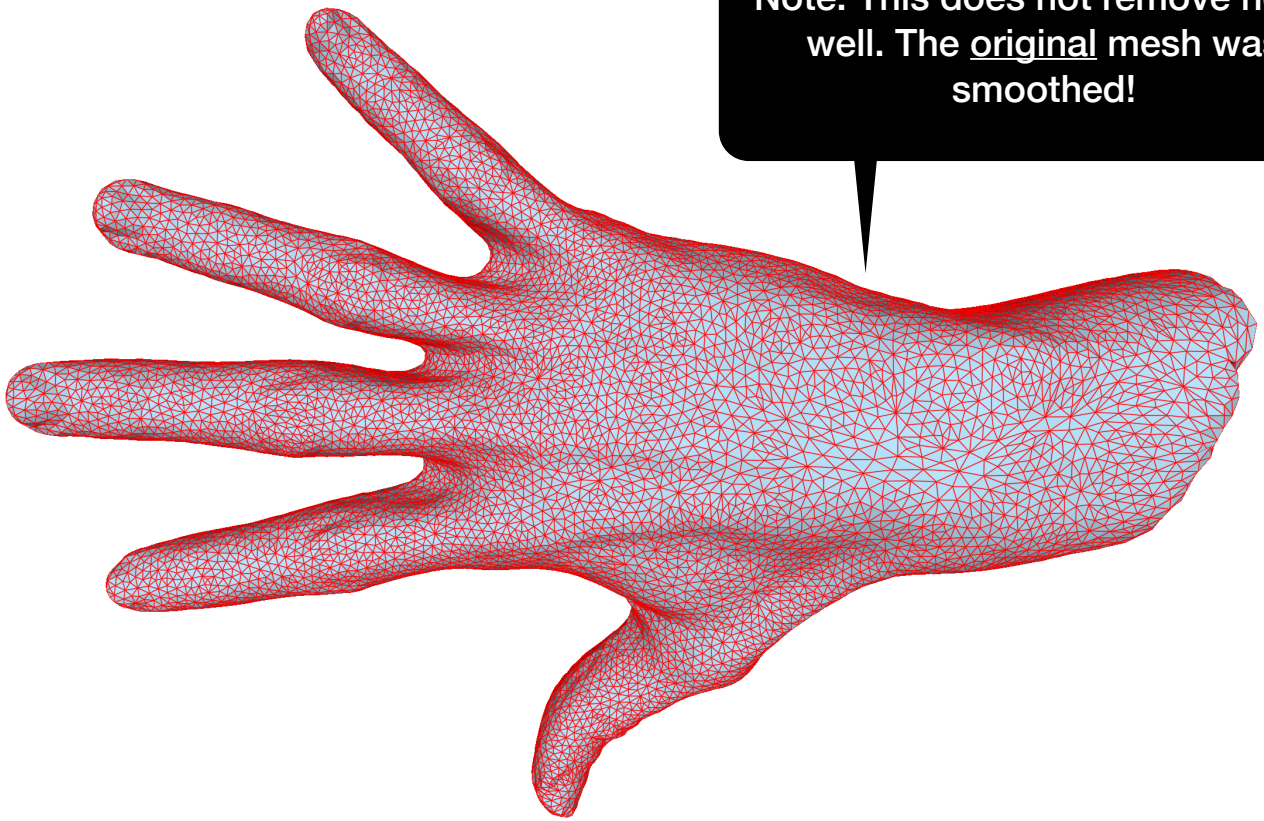
$\lambda = 0.5$, $\mu = -0.52$, iterations = 20, time = 0.08 seconds



Tangential Area-Weighted Laplacian Smoothing

$\lambda = 1$, iterations = 20, time = 0.32 seconds

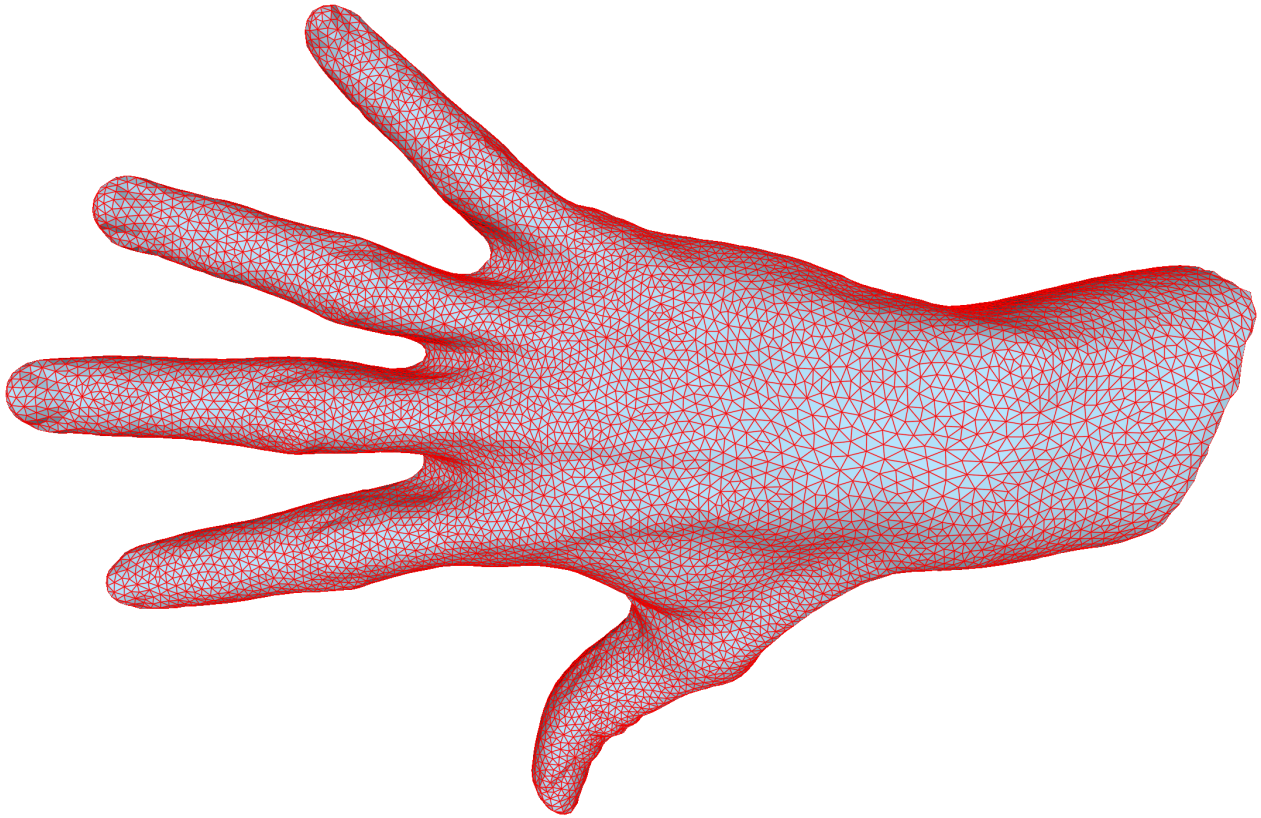
- Projects Laplacian into orthogonal complement of surface normal
- Weights neighboring vertices with area
- Improves triangle size and shape while not changing geometry (much)



TAL Smoothing + Optimization

$\lambda = 1$, iterations = 20+ (convergence)

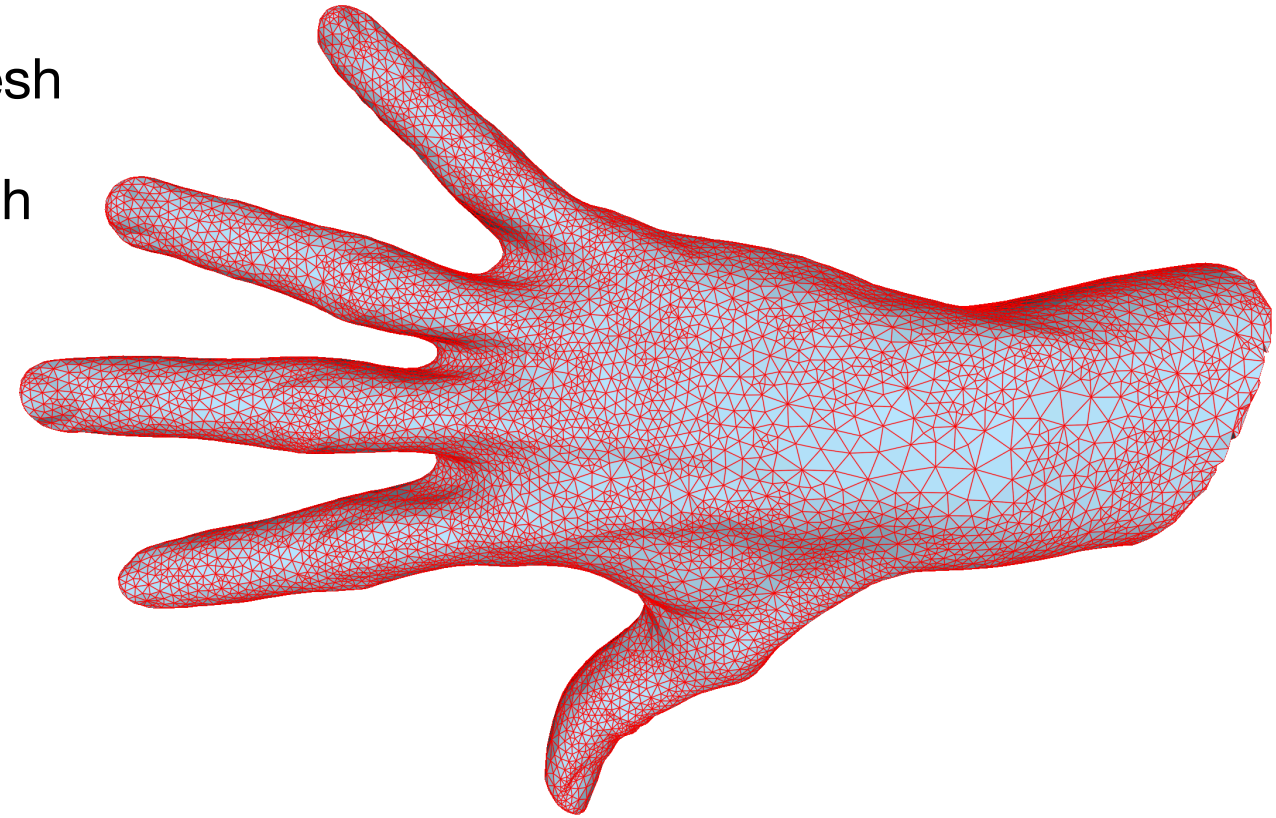
- Here TAL smoothing was combined with maximization of minimum angle



Geodesic Smoothing

$\lambda = 0.5$, iterations = 20, time = 20.9 seconds

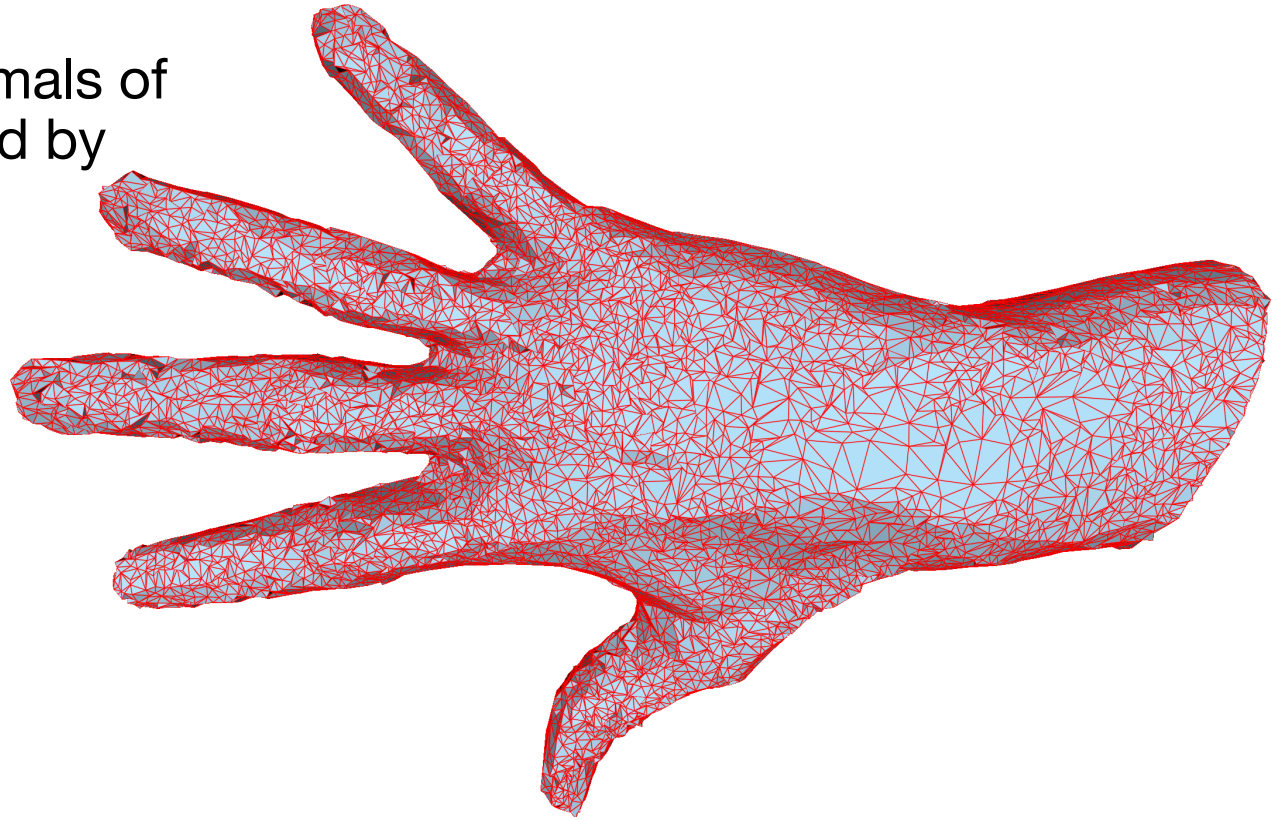
- Smoothing of noisy mesh but constrained to original (not noisy) mesh



Bilateral Normal Filtering

iterations = 50, time = 21.3 seconds

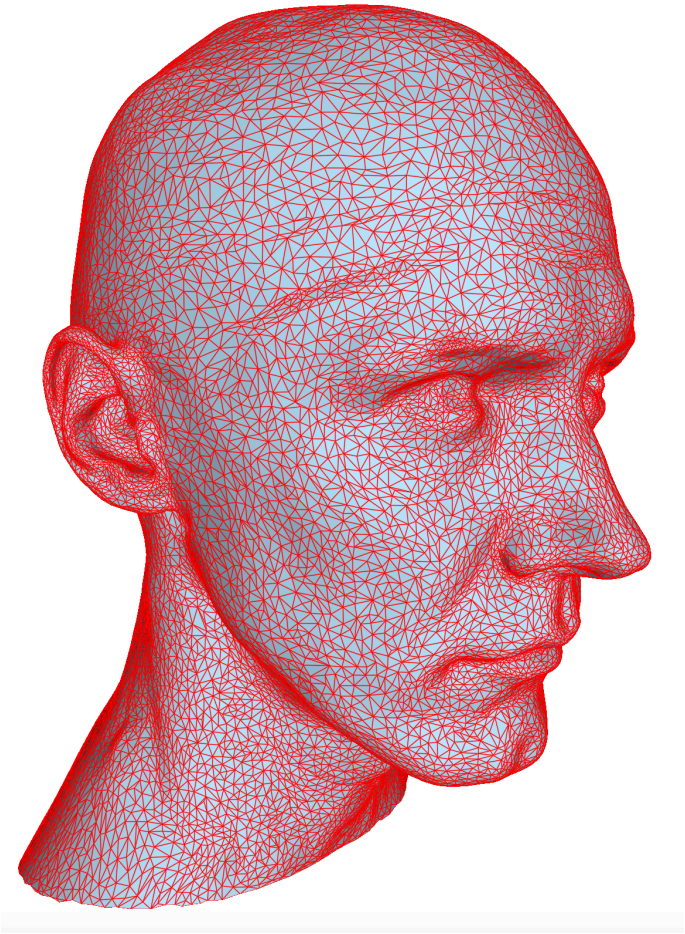
- Bilateral filtering of normals of adjacent faces, followed by vertex refitting
- Note that this preserves edges …
- Perhaps too well …



Mesh Parametrization

What if I want a global map of a shape?

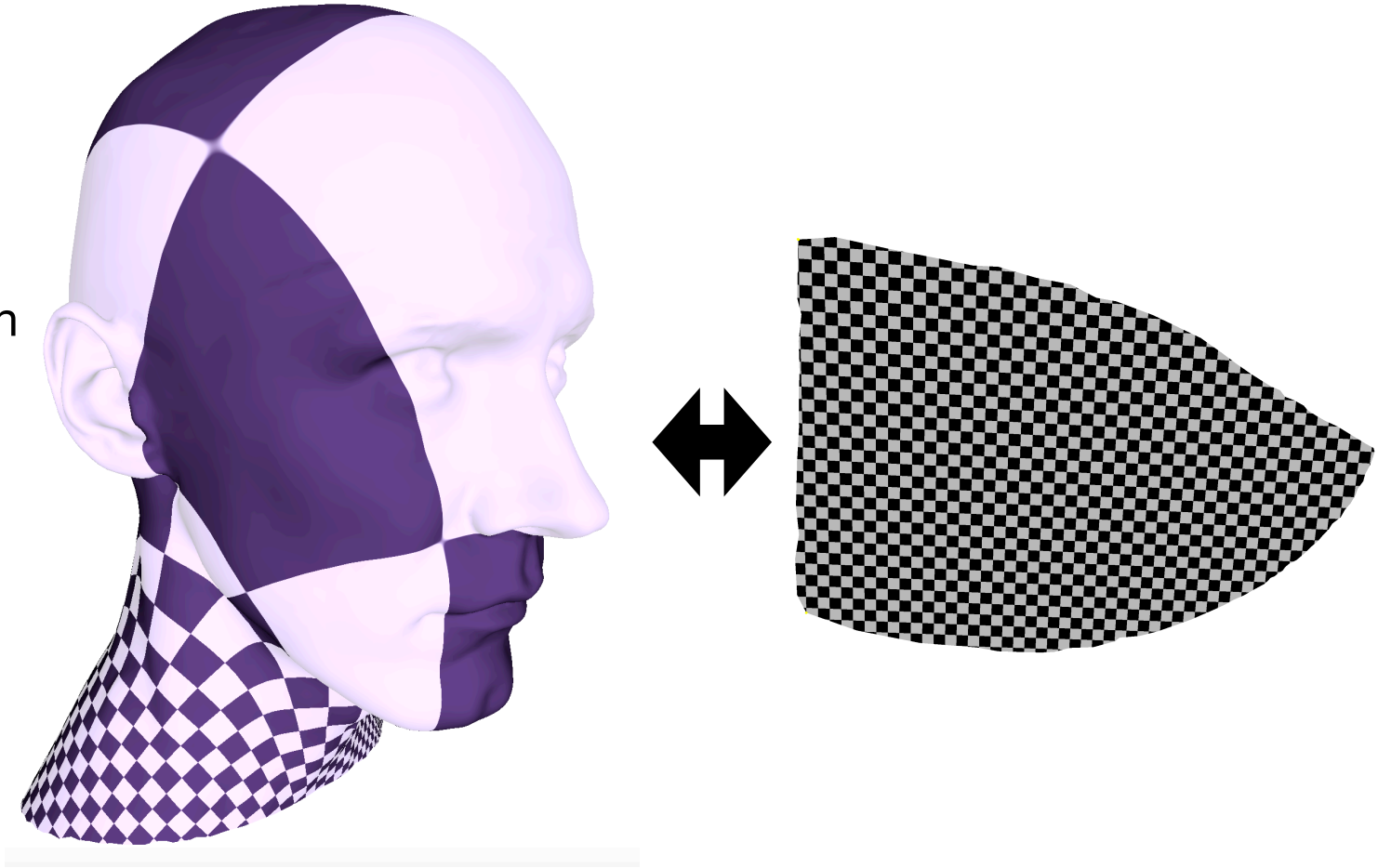
- We assume a shape with disc topology



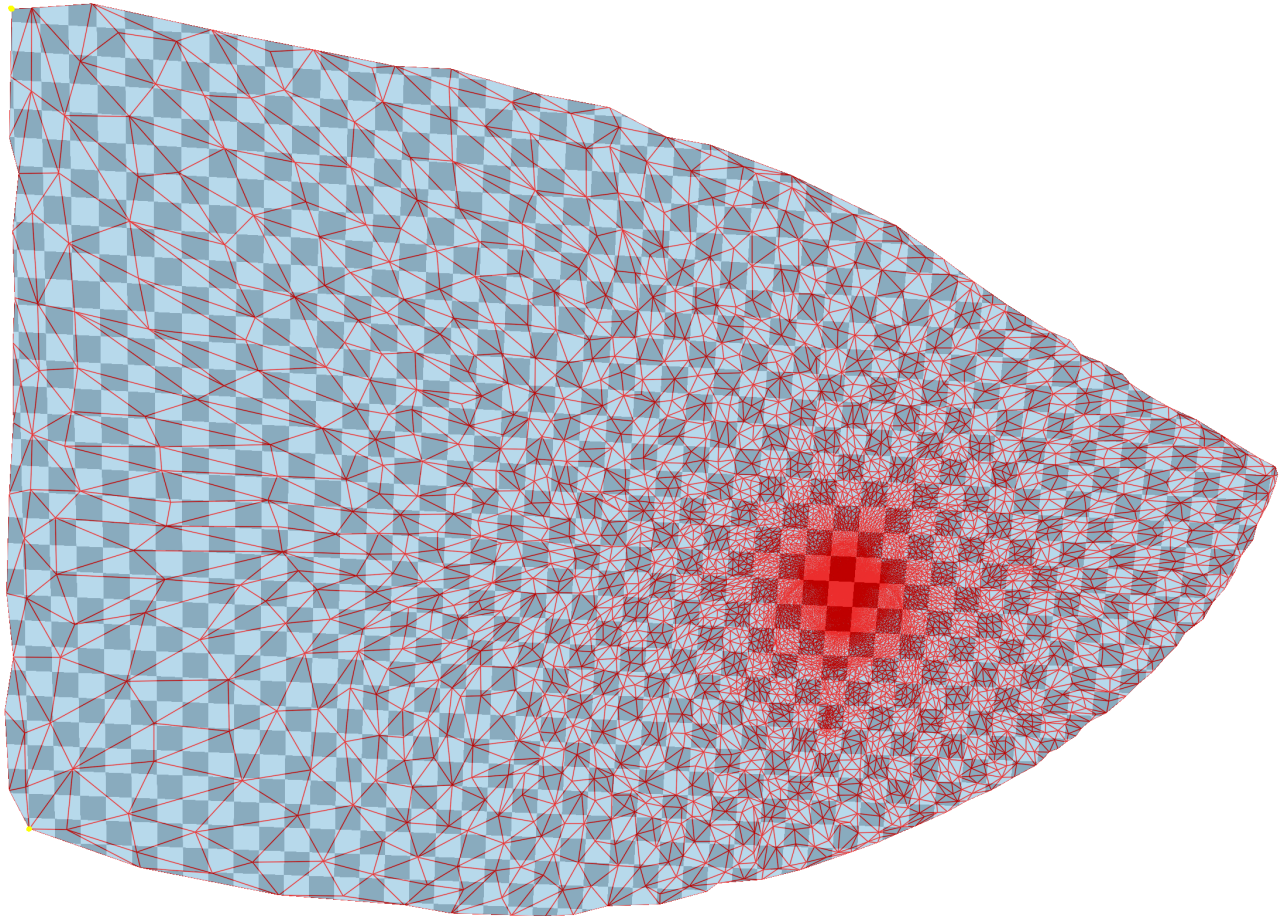
Mesh Parametrization

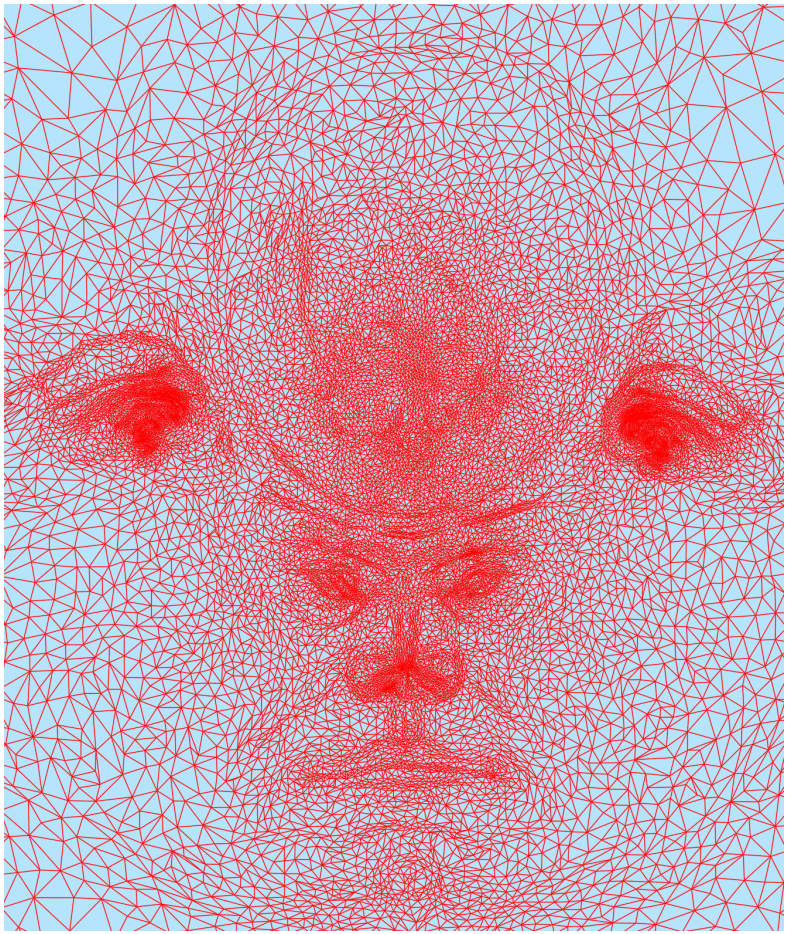
Such as this!

- The mapping should establish an invertible 1-1 map
- We also want preservation of certain properties
  - angles
  - areas
- Can you guess what is <u>not</u> preserved here?



Mesh Parametrization



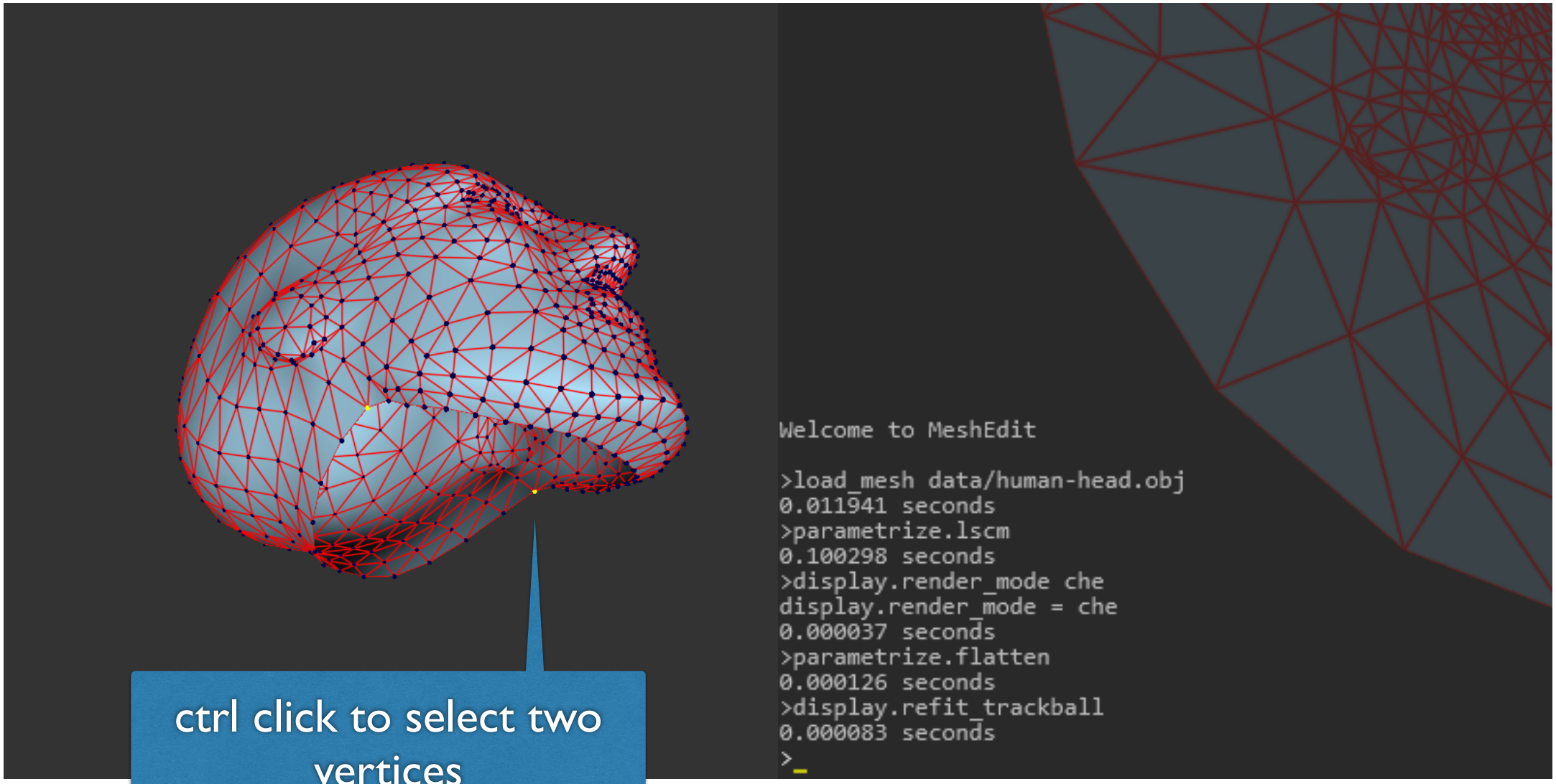


Least Squares Conformal Maps

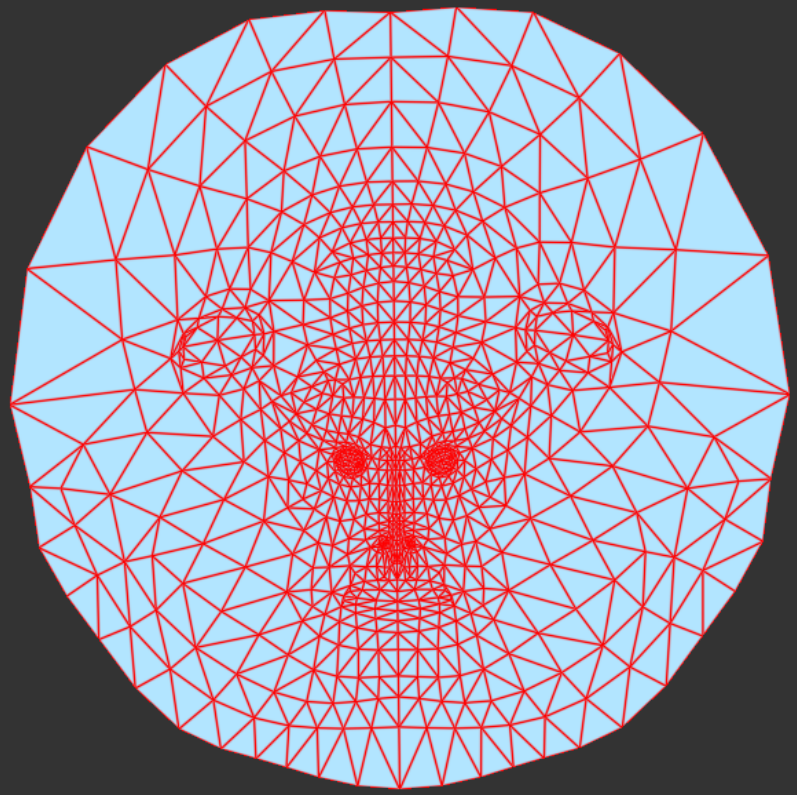
- Angles are preserved (in the LS sense) by the mapping to 2D (u,v)
- We can express the conformal energy

$$E_C(\mathbf{u}) = E_D(\mathbf{u}) - A(\mathbf{u})$$

- Where $E_D = \frac{1}{2}\int_S \|\nabla \mathbf{u}\| dA$ is the Dirichlet energy and A is the area of the map.
- E_D is minimized by harmonic functions $\Delta \mathbf{u} = 0$
- Gradient of triangle areas imposed as boundary constraints
- Two vertices must be fixed


Welcome to MeshEdit
>load_mesh data/human-head.obj
0.011941 seconds
>parametrize.lscm
0.100298 seconds
>display.render_mode che
display.render_mode = che
0.000037 seconds
>parametrize.flatten
0.000126 seconds
>display.refit_trackball
0.000083 seconds
>_
ctrl click to select two vertices

Results

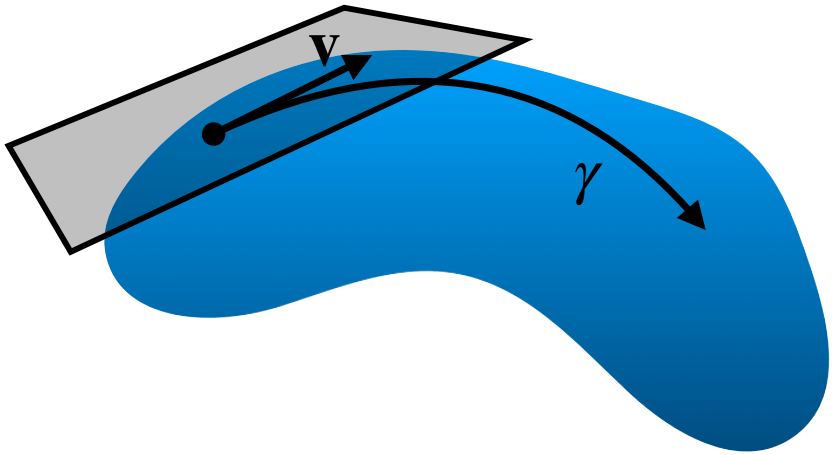


$\exp_R : \mathrm{T}M \to M$

- The Riemannian exponential,

  $\exp_R(t\mathbf{v}) = \gamma(t)$, where $\|\mathbf{v}\| = 1$

  maps a tangent vector in direction $\mathbf{v}$ onto the same length, t , of a geodesic, γ



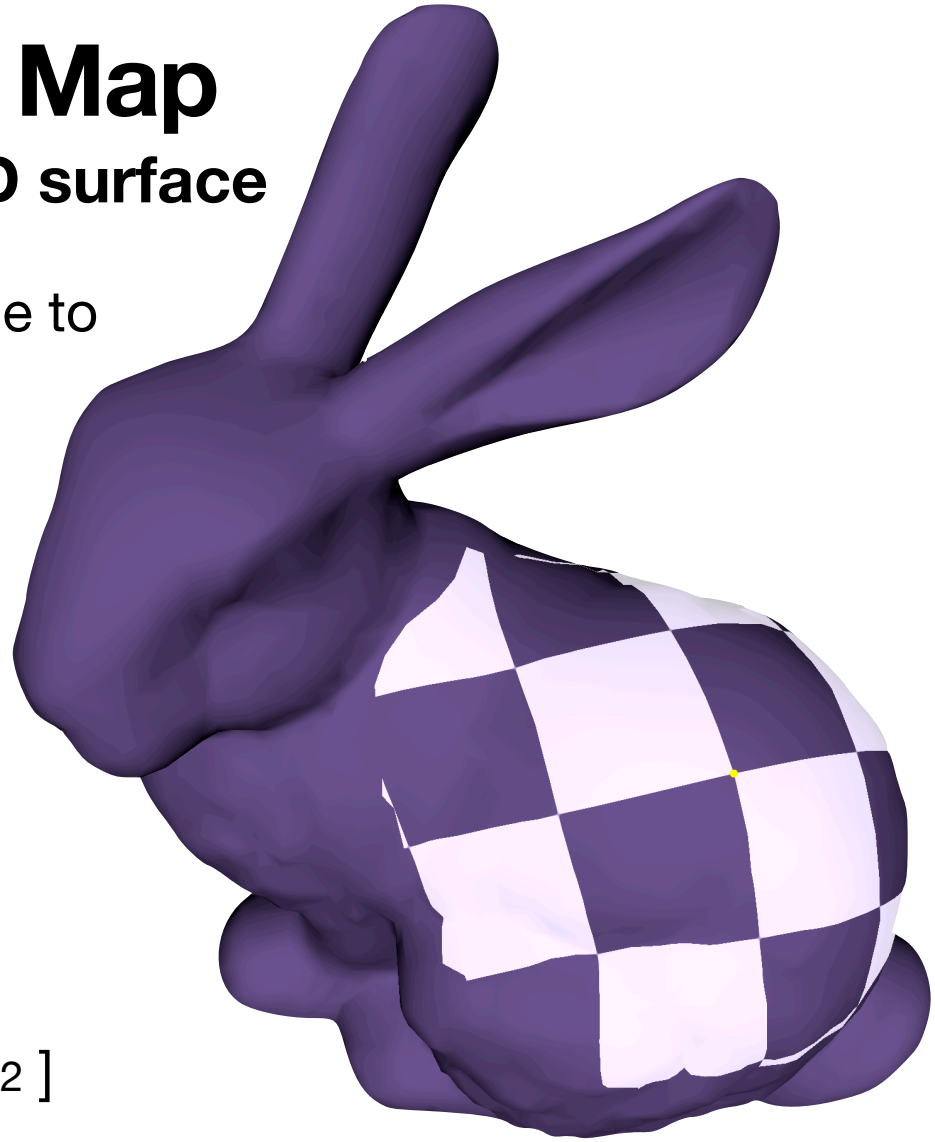
A geodesic is the generalization of a straight line to a curved surface: the curve of constant bearing

The shortest path between two points are always geodesics, but two points can be connected by multiple geodesics

The Discrete Exponential Map

A Map between a 2D domain and a 3D surface

- The discrete $\exp$ maps from a tangent plane to the surface
- The discrete $\log$ map is the inverse:
  - used to map a checkerboard texture from the tangent plane to the surface
- This provides a way to sample a surface consistently up to rotation
- Computation similar to Dijkstra's algorithm: [Melvær and Reimers. Geodesic polar coordinates on polygonal meshes. Computer Graphics Forum, 31(8), 2012]

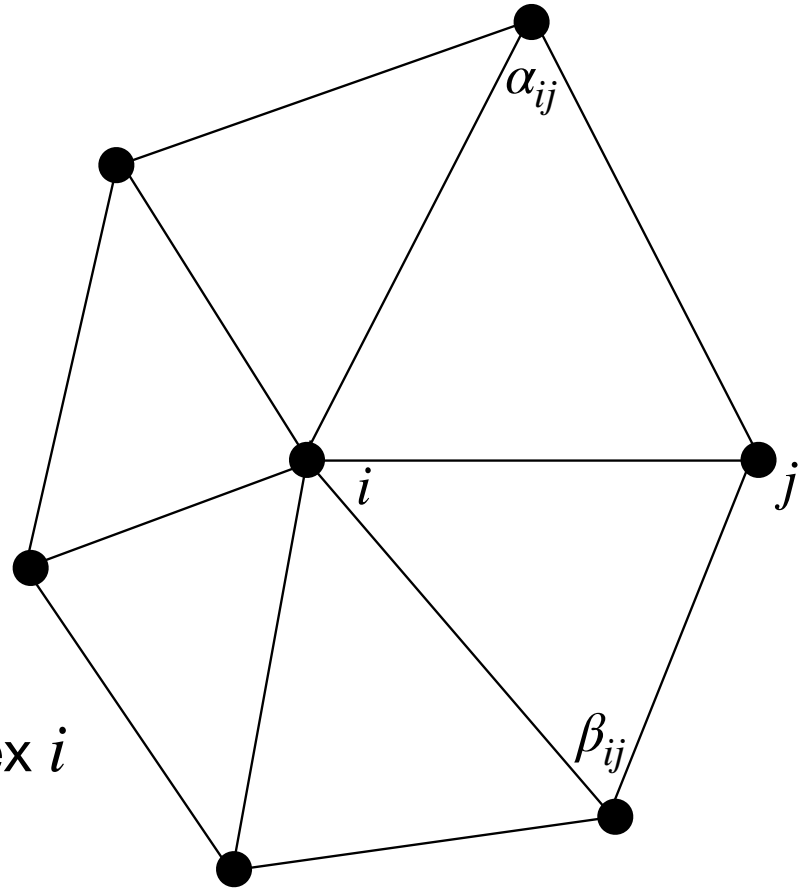


LBO Matrix

- The symmetric cot based LBO:

$$L_{ij} = \begin{cases} \dfrac{\cot(\alpha_{ij}) + \cot(\beta_{ij})}{2\sqrt{A_i A_j}} & \text{for } j \in N_i \\ -\sum_{k \in N_i} L_{ik} & \text{for } j = i \\ 0 & \text{otherwise} \end{cases},$$

where A_i is the area of triangles incident on vertex i



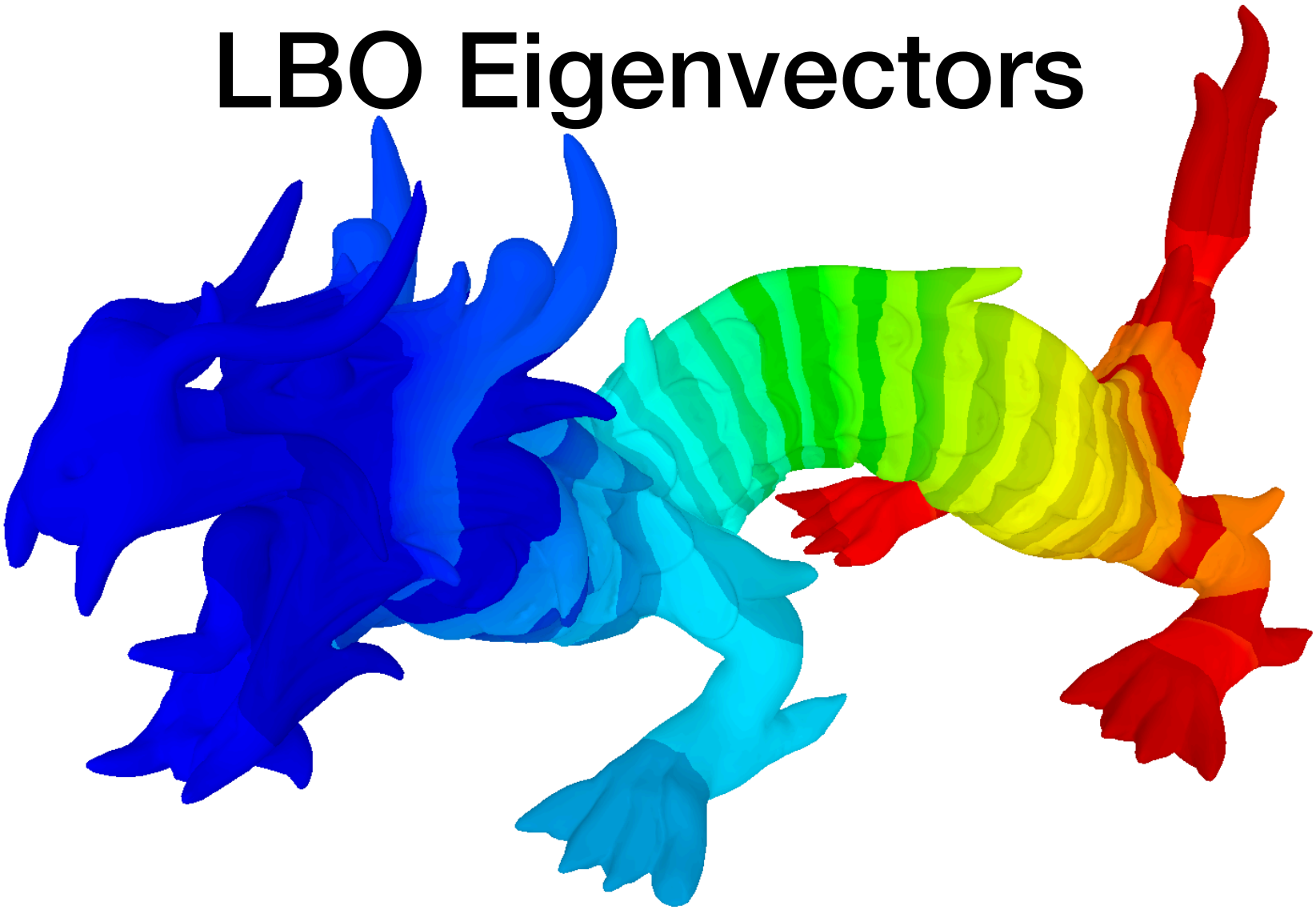
LBO Eigenvectors

Example Shape

The Asian Dragon

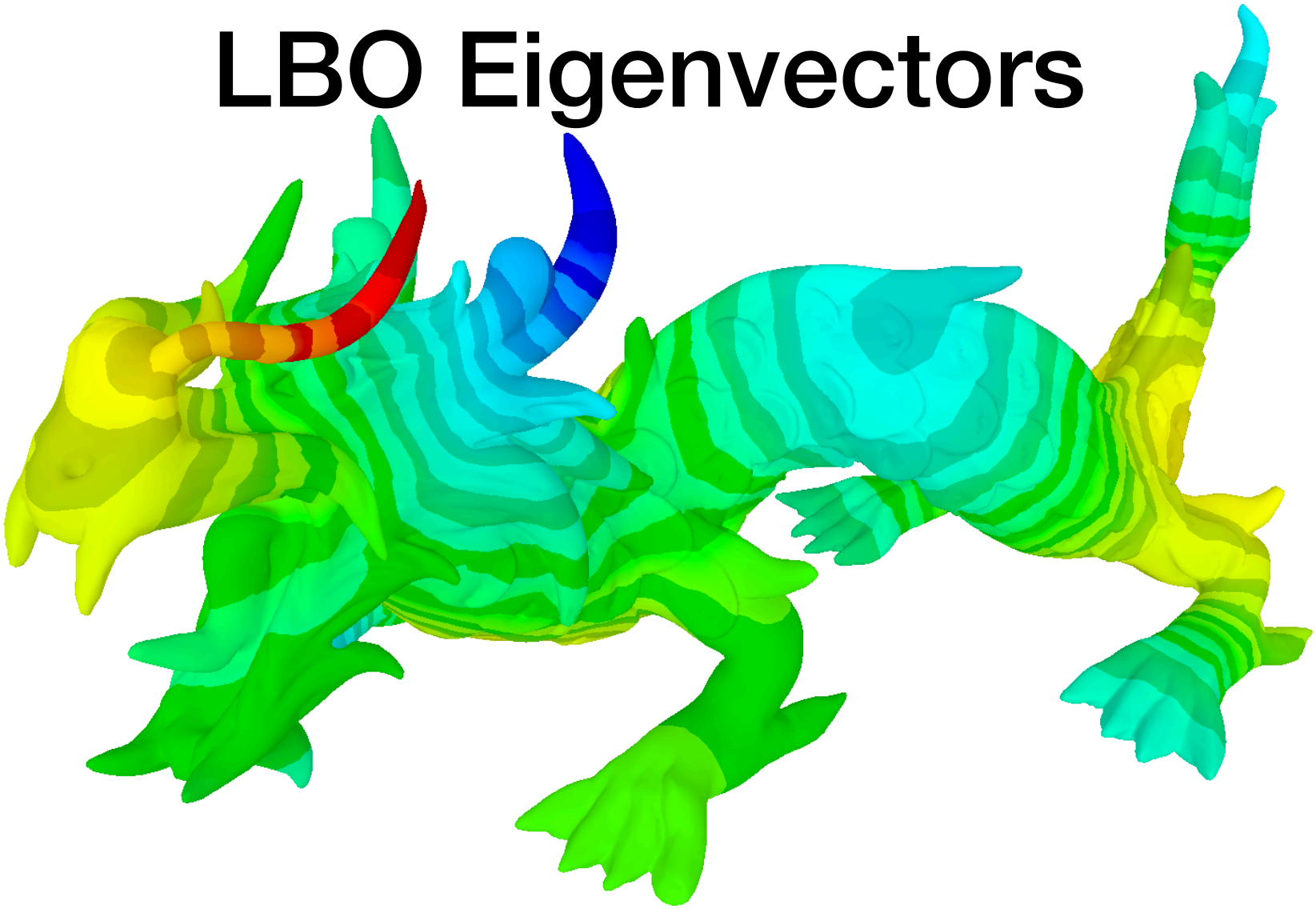
eigenvector : 1

LBO Eigenvectors



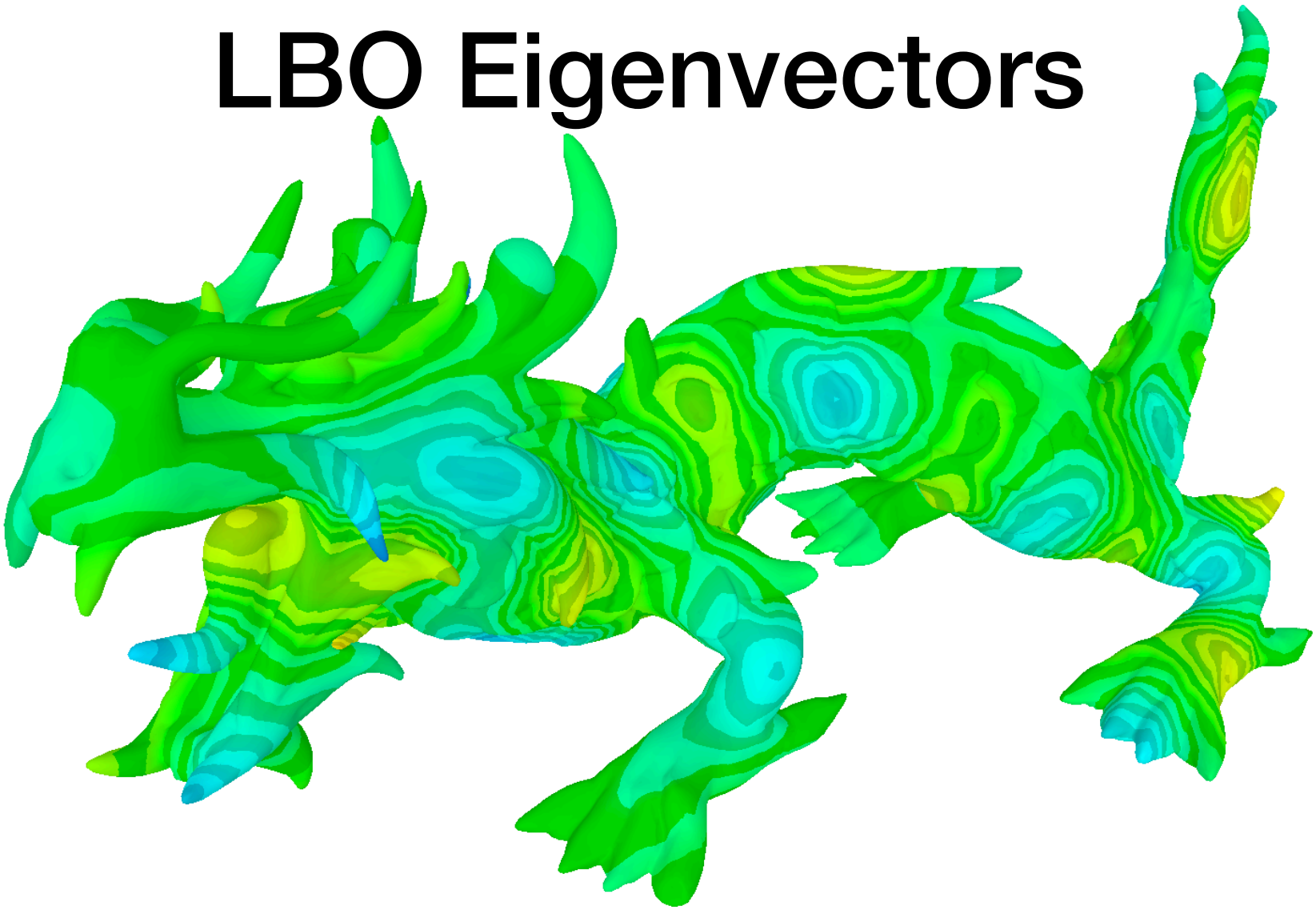
eigenvector : 10

LBO Eigenvectors



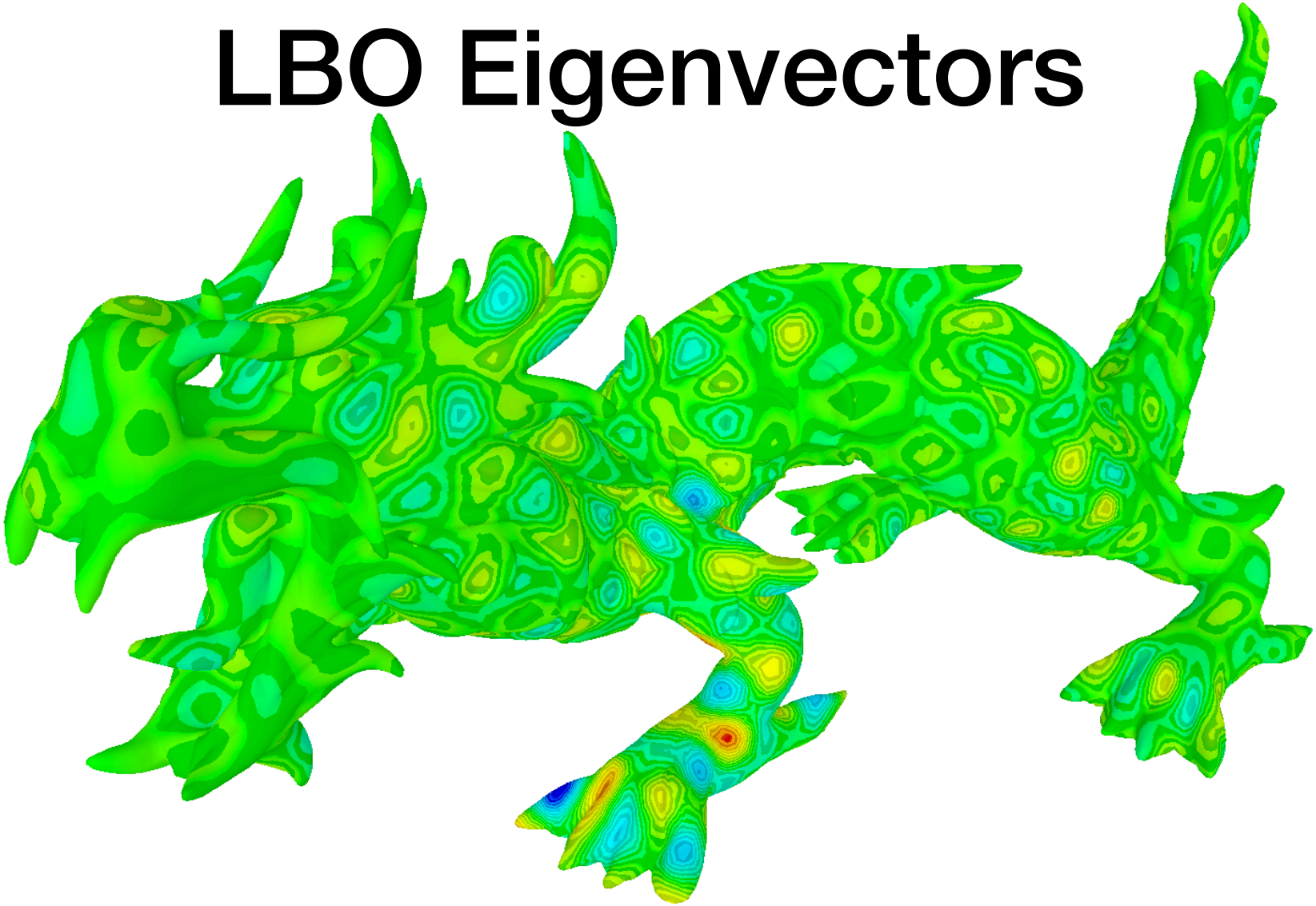
eigenvector : 100

LBO Eigenvectors



eigenvector : 999

LBO Eigenvectors



LBO eigenvectors

Technicalities

- The LBO is a huge matrix, but:
  - Typically, we need only a subset of the eigenvectors
    - Power methods can find a subset for the largest eigenvalues
    - We need the smallest, though …
    - Solution: compute eigenvalues and eigenvectors for $\mathbf{L}^{-1}$
    - The Spectra library provides a modern Eigen compatible interface

Spectral Analysis

The Fourier Transform but for Meshes

- Let Φ be a basis of LBO eigenfunctions for a given mesh, i.e. $\mathbf{L} = \Phi \Lambda \Phi^T$, where Λ is a diagonal matrix of eigenvalues
- We can now transform a function

  $$\hat{f} = \Phi^T f$$

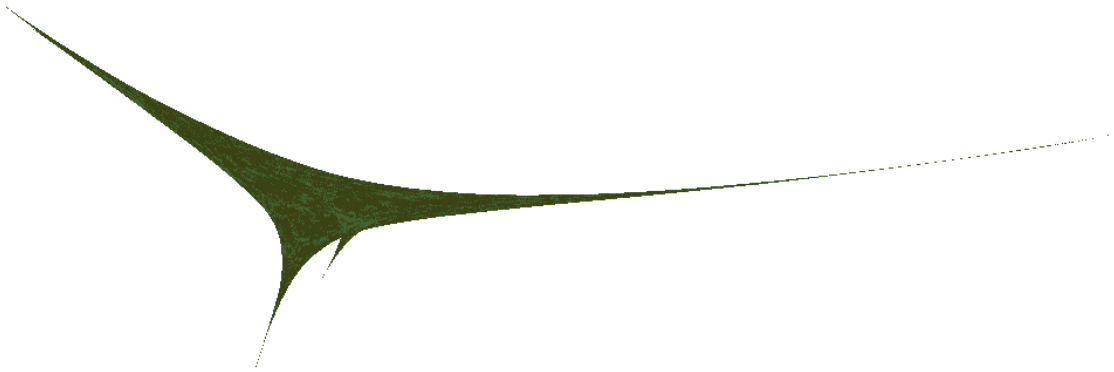
- And get it back

  $$f = \Phi \hat{f}$$

- Note: we can set entries of $\hat{f}$ to zero / perform other manipulations

4 eigenvectors

Spectral Analysis and Reconstruction



31 eigenvectors

Spectral Analysis and Reconstruction

301 eigenvectors

Spectral Analysis and Reconstruction

1000 eigenvectors

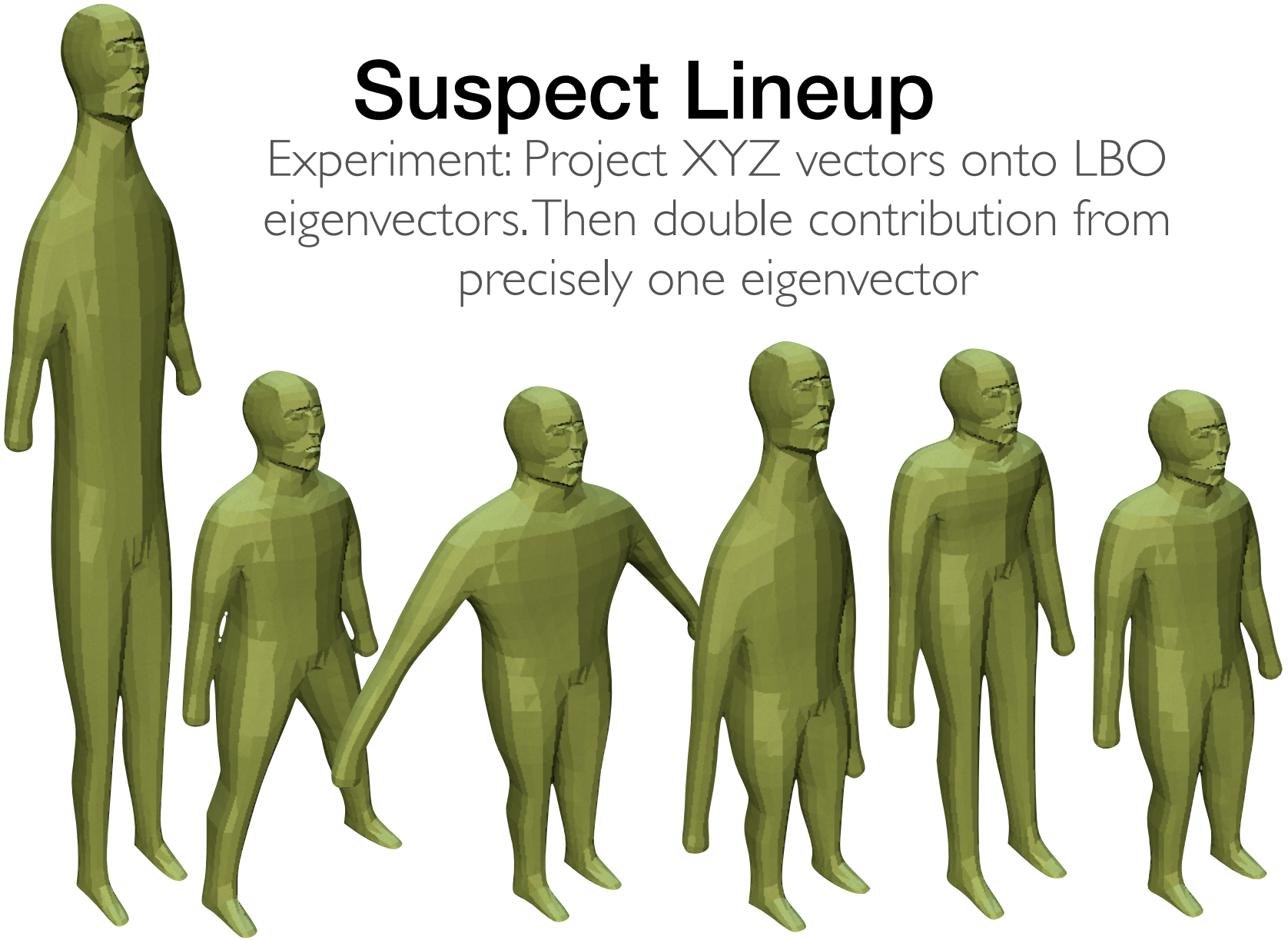
Spectral Analysis and Reconstruction

Example Shape

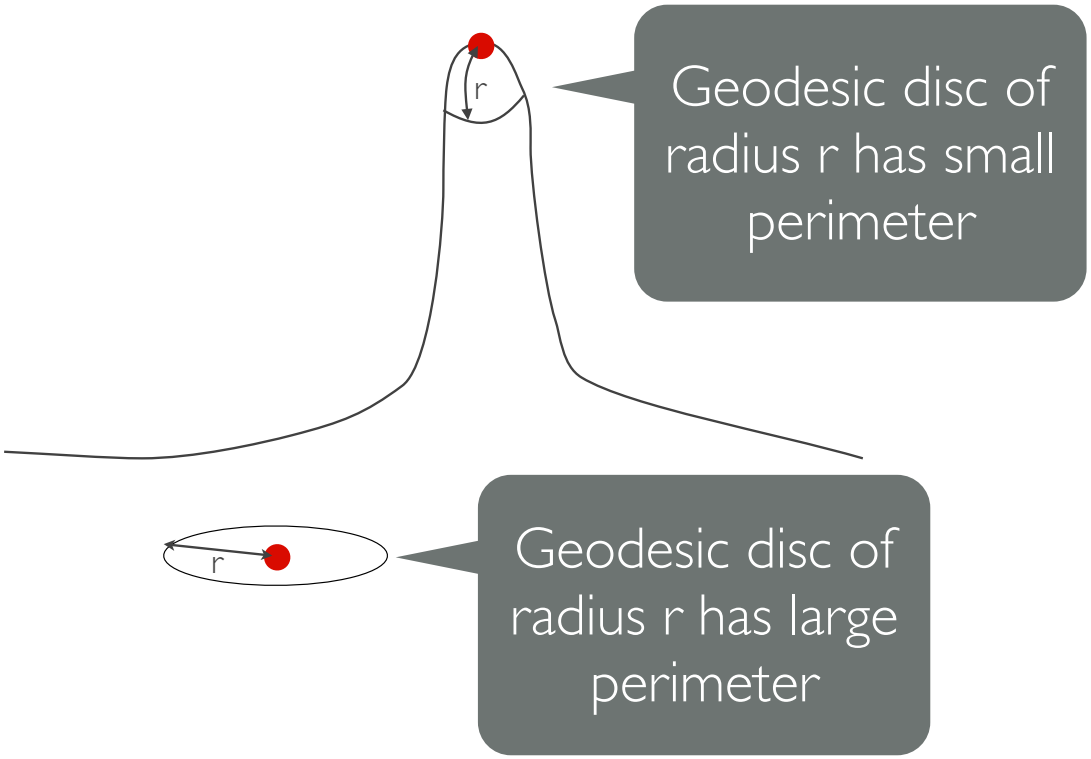
Spectral Analysis and Reconstruction

Suspect Lineup

Experiment: Project XYZ vectors onto LBO eigenvectors. Then double contribution from precisely one eigenvector



Shape Analysis Using the Auto Diffusion Function



the Auto diffusion function

From the heat kernel,

$$h(t,x,y) = \sum_{i=0}^{N} e^{-t\lambda_i} \Phi_{xi} \Phi_{yi}$$

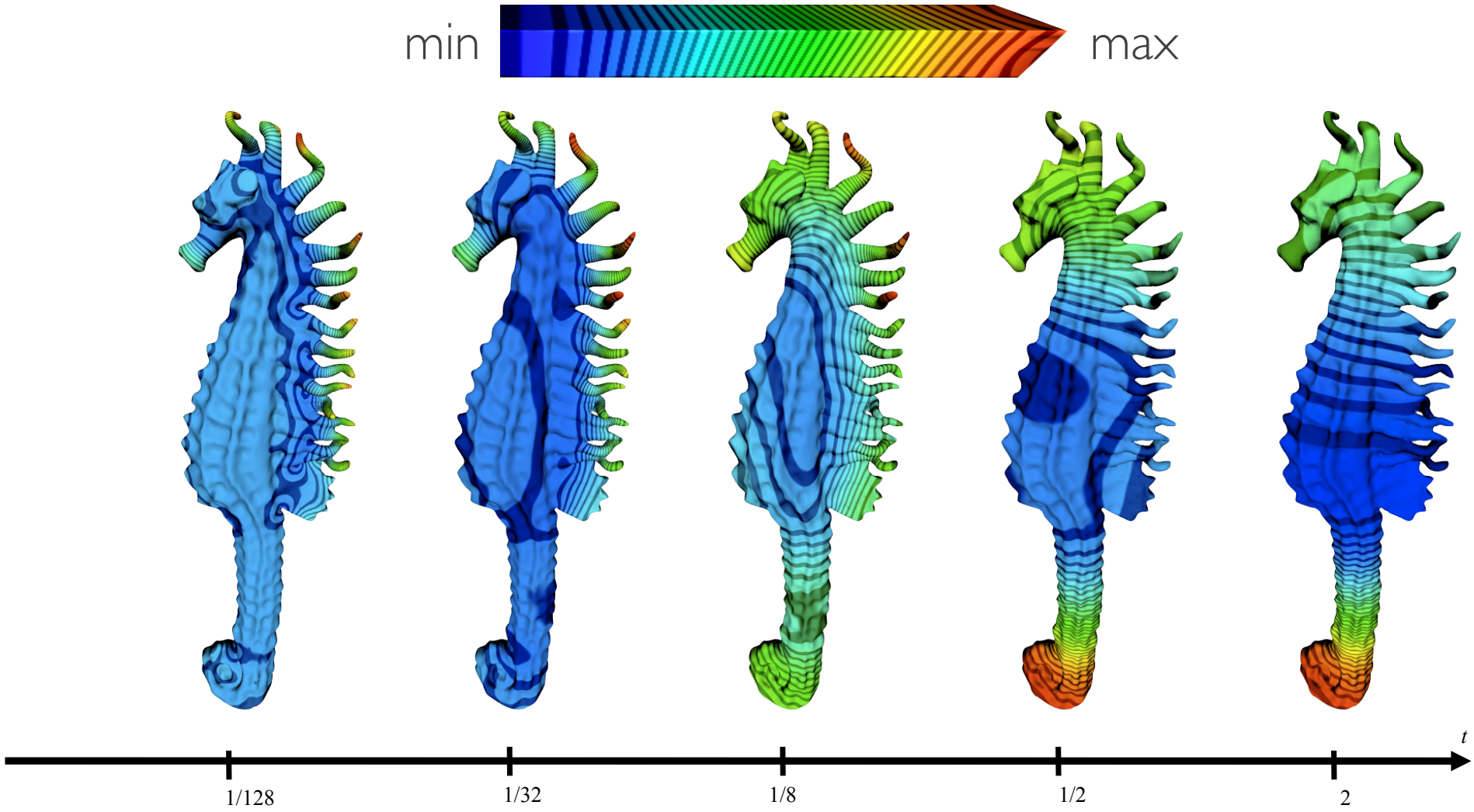
we now define

$$\mathrm{ADF}_t(x) = \sum_{i=0}^{N} e^{-t\lambda_i/\lambda_1} \Phi_{xi}^2$$

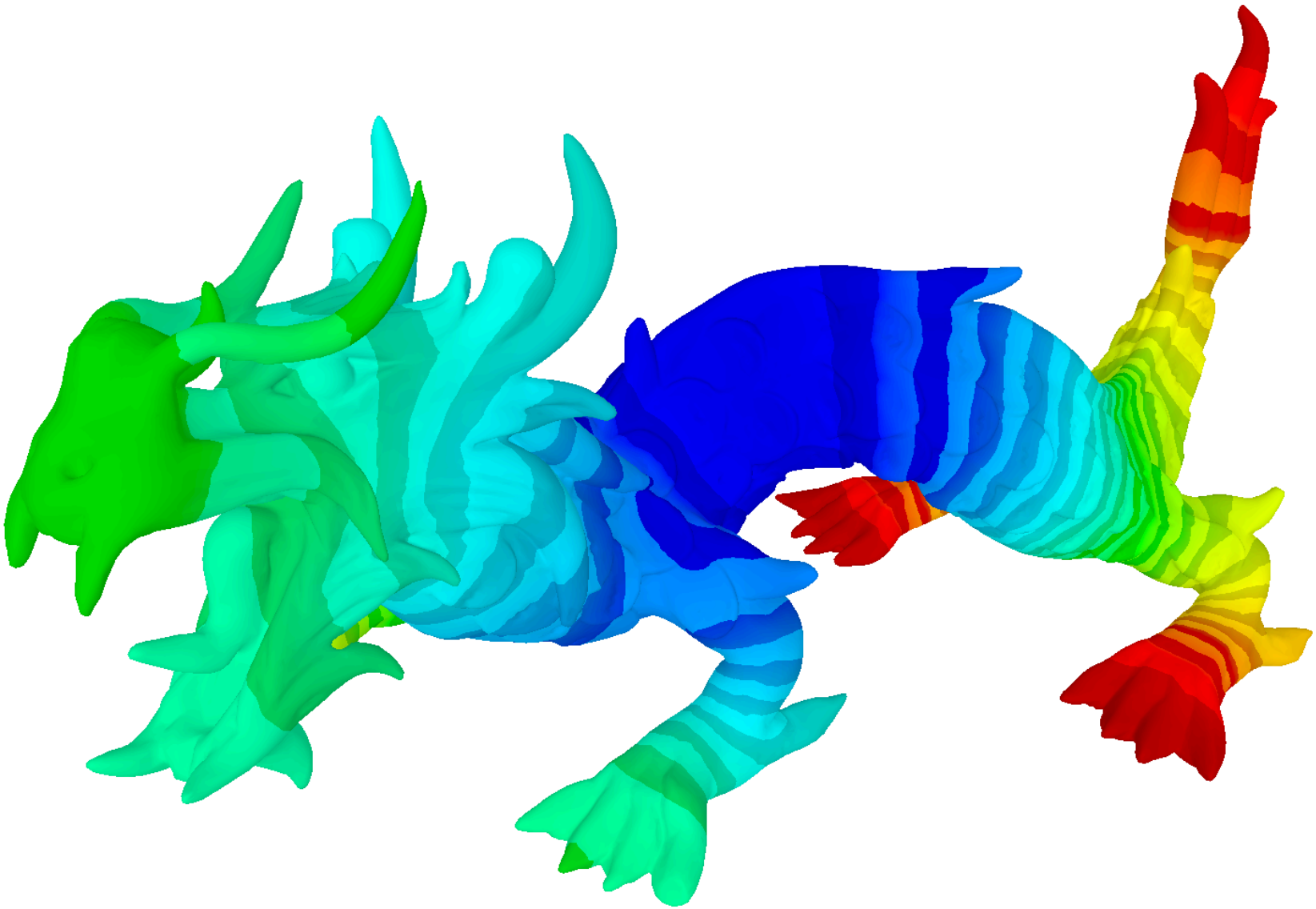
which is simply a function that describes how much [quantity subj. to diffusion] is left at x at time t

Gębal, K., et al. "Shape analysis using the auto diffusion function." Computer Graphics Forum. Vol. 28. No. 5.

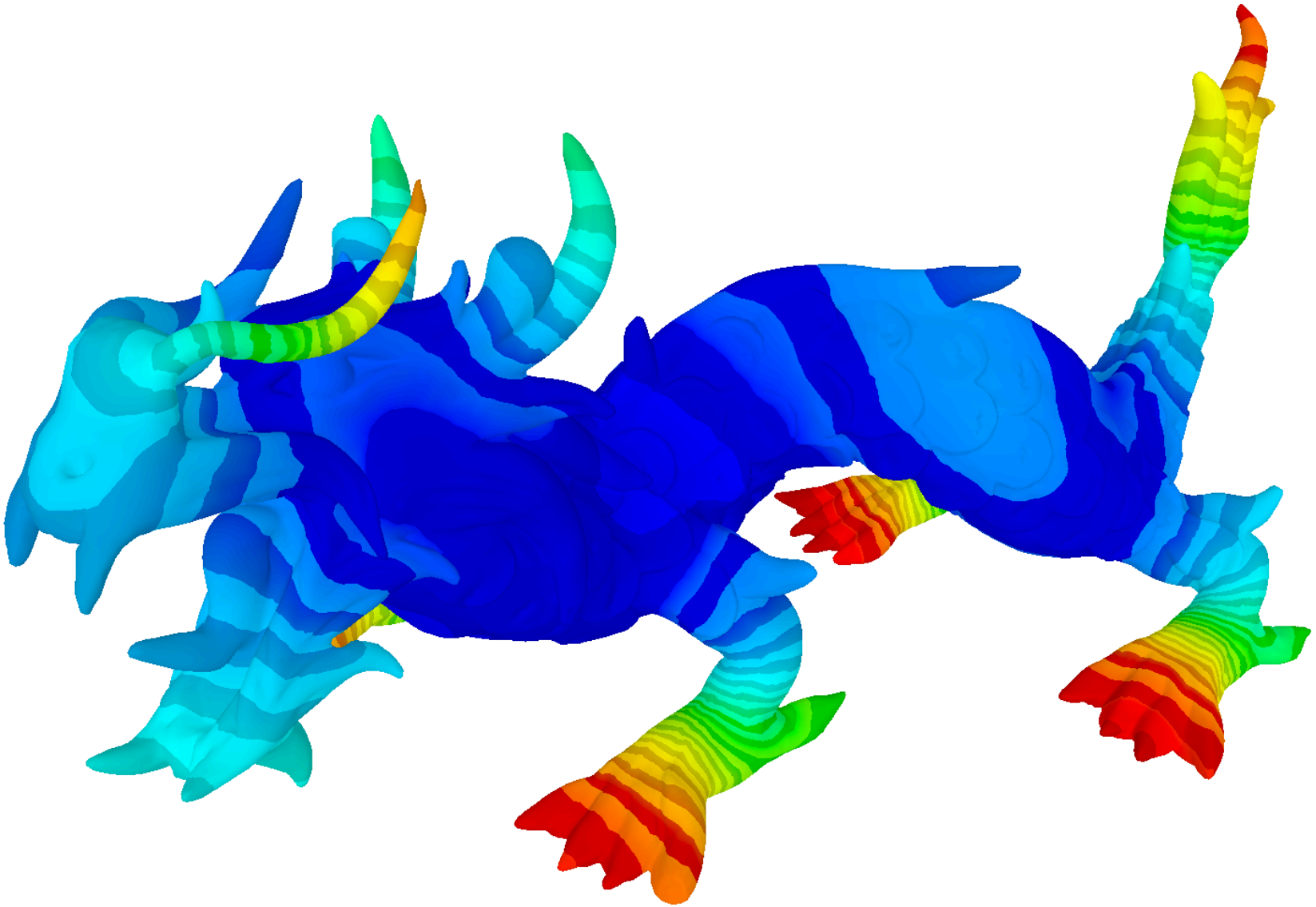
The Auto Diffusion function



t=1

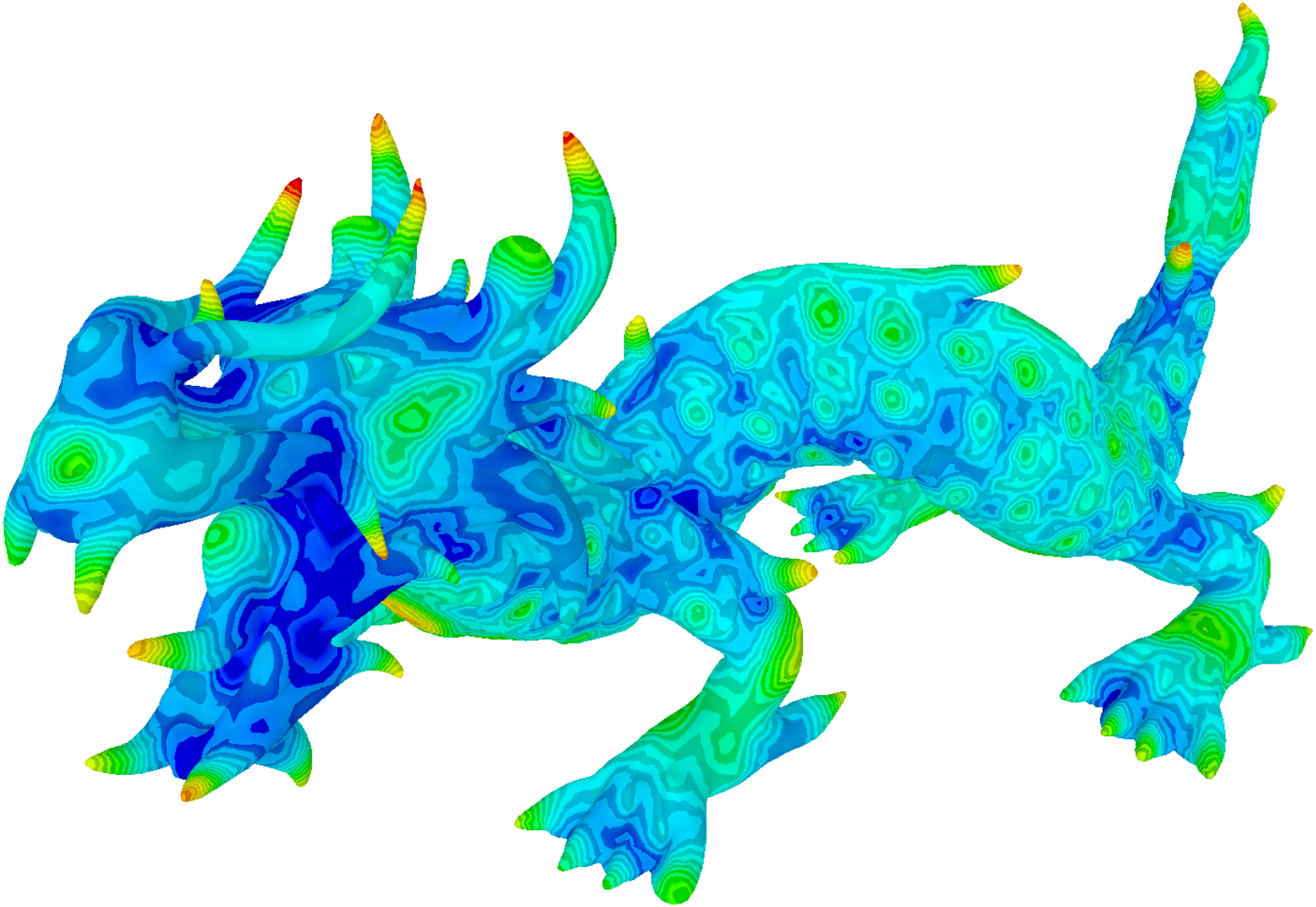


t=0.1

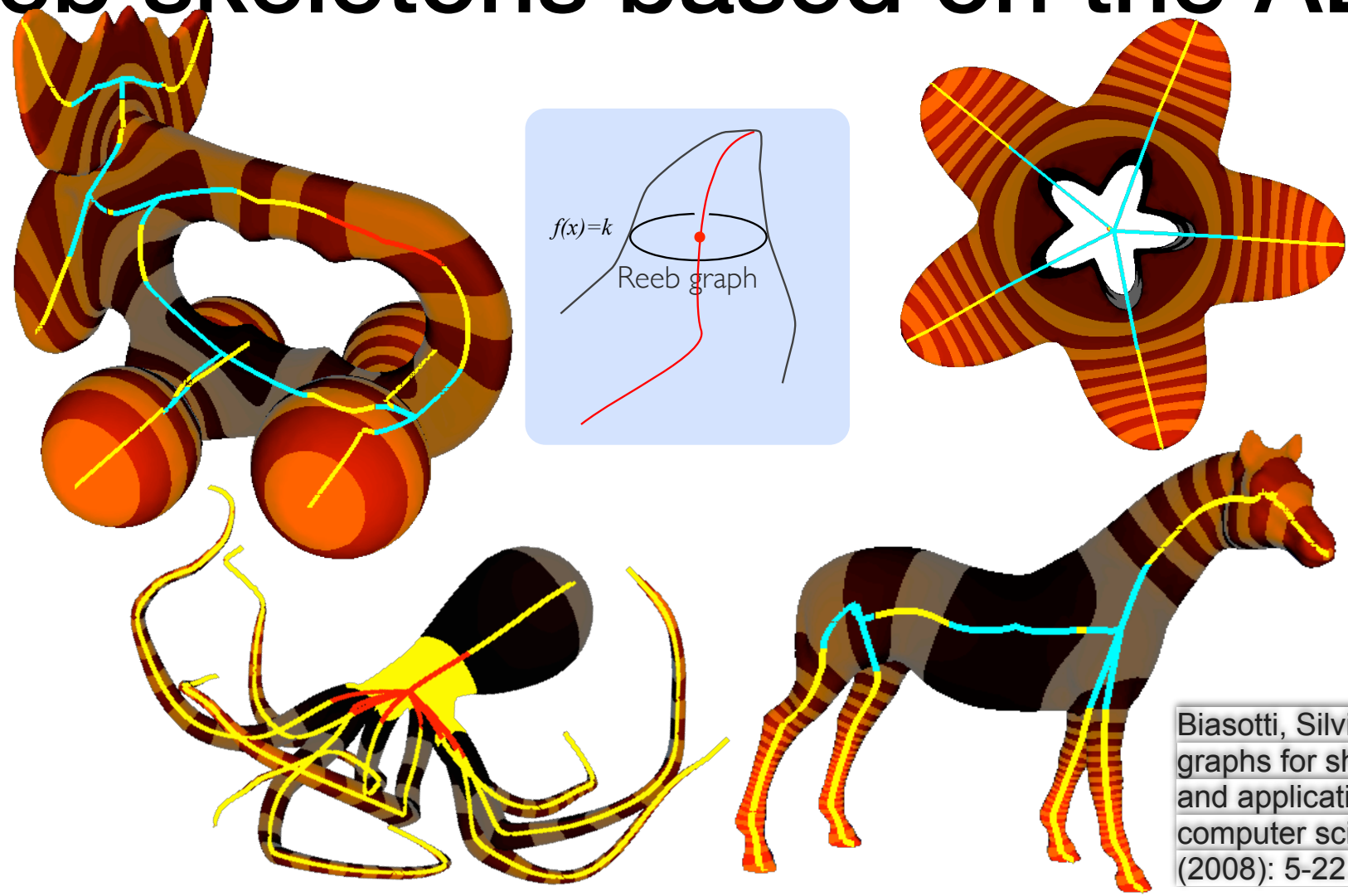


t=0.01

t=0.002

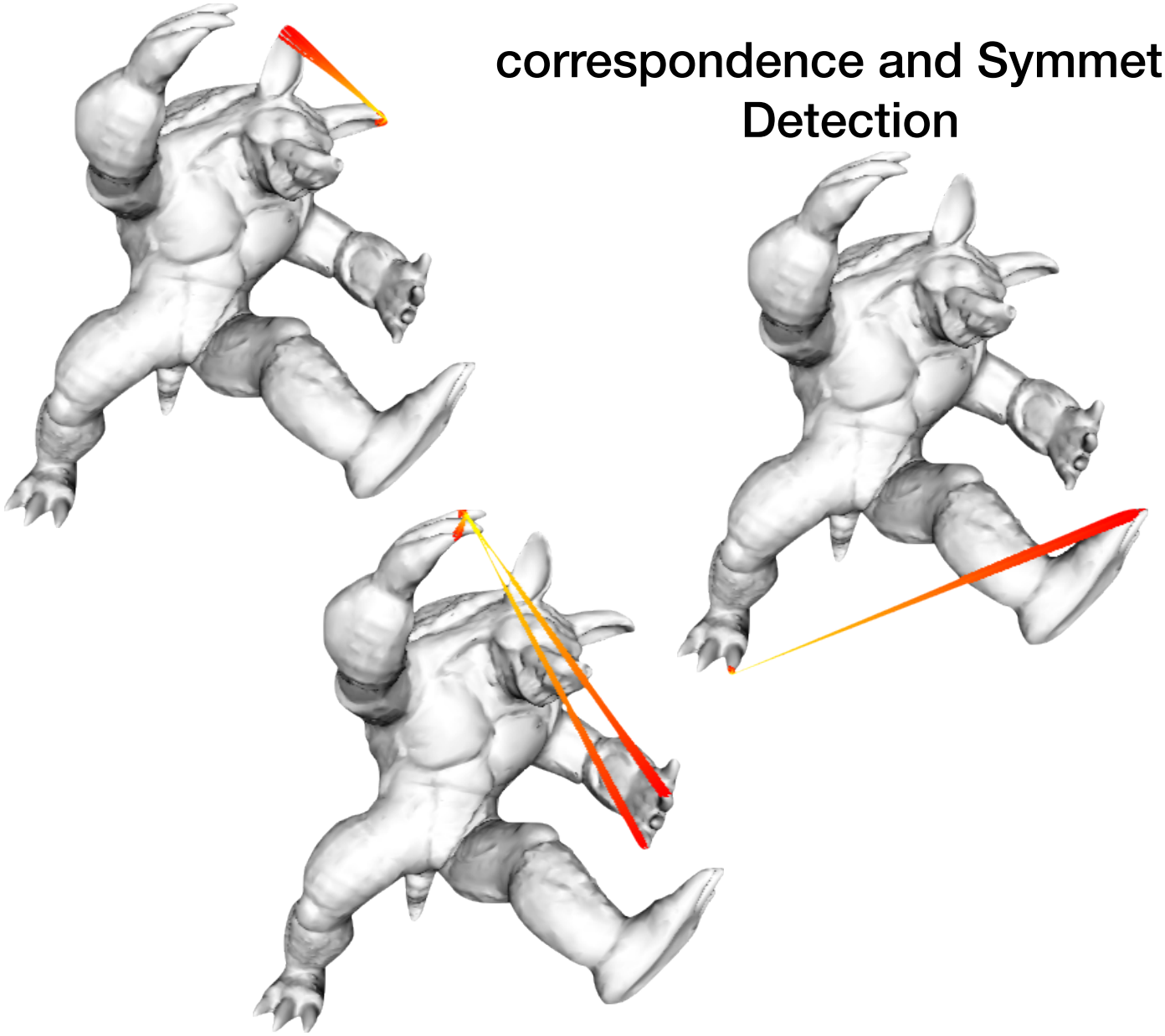


Reeb skeletons based on the ADF



Biasotti, Silvia, et al. "Reeb graphs for shape analysis and applications." Theoretical computer science 392.1-3 (2008): 5-22.

correspondence and Symmetry Detection



Related to

Sun, Jian, Maks Ovsjanikov, and Leonidas Guibas. "A concise and provably informative multi-scale signature based on heat diffusion." Computer graphics forum. Vol. 28. No. 5.

Functional Maps

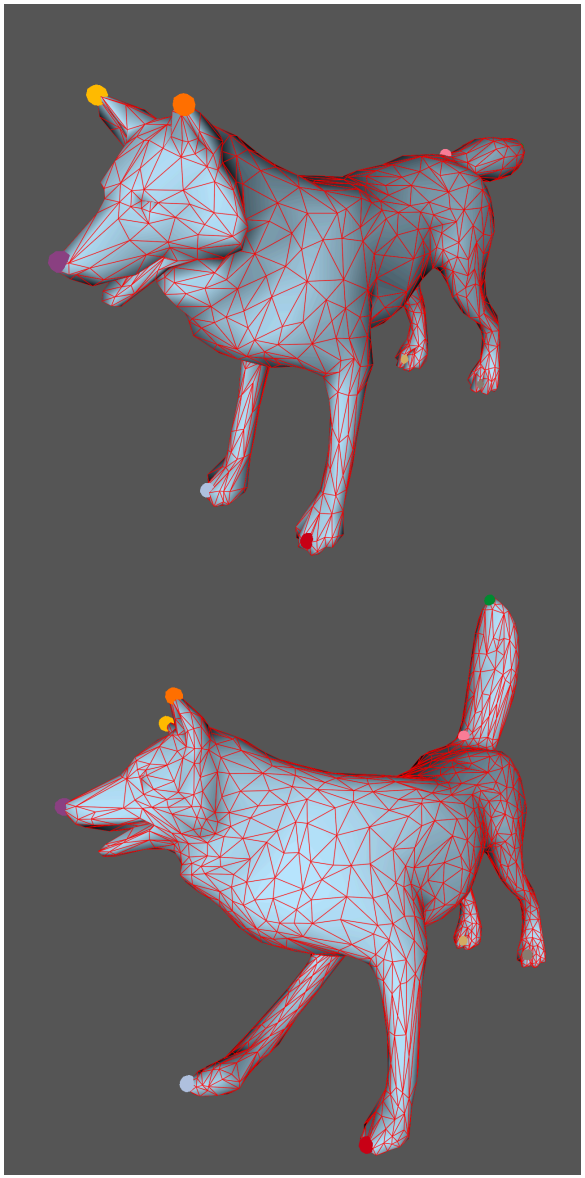
- Given corresponding functions f^i, g^i on meshes M_1 and M_2, respectively, and a
- basis of eigenvectors Φ and Γ s.t.

  $f^i = \Phi \hat{f}^i$ and $g^i = \Gamma \hat{g}^i$
- We seek $\mathbf{C}$ such that $\hat{g}^i = \mathbf{C}\hat{f}^i$, so

  $\mathbf{G} = \Gamma \mathbf{C} \Phi^T \mathbf{F}$,

  where $F_{\cdot i} = f^i$ and $G_{\cdot i} = g^i$

Ovsjanikov, Maks, et al. "Functional maps: a flexible representation of maps between shapes." ACM Transactions on Graphics (TOG) 31.4 (2012): 1-11.

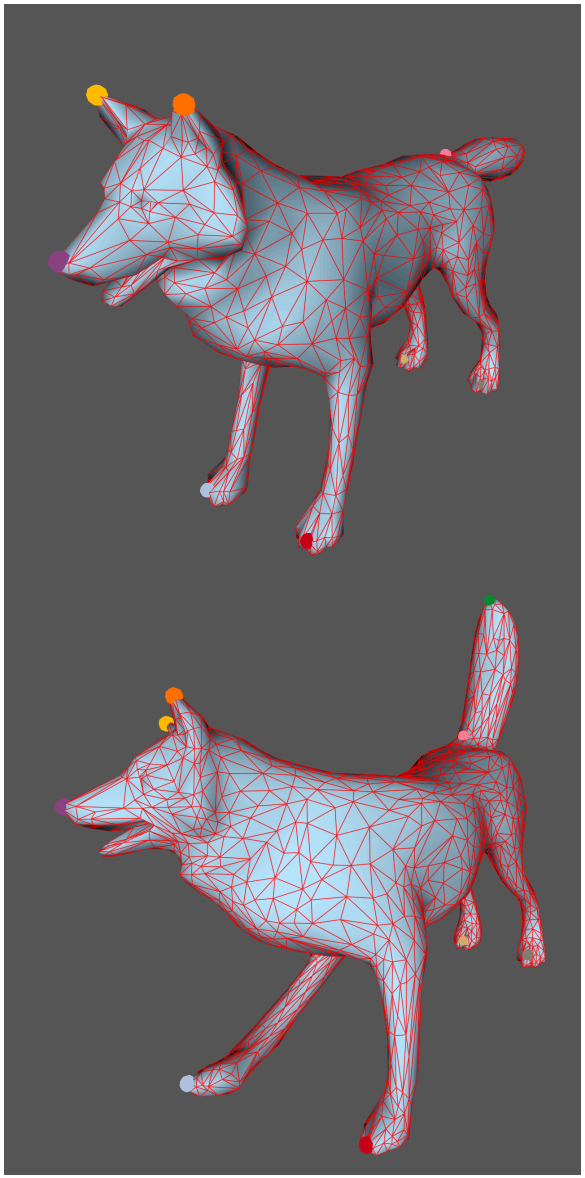


M_1

M_2

Functional Maps

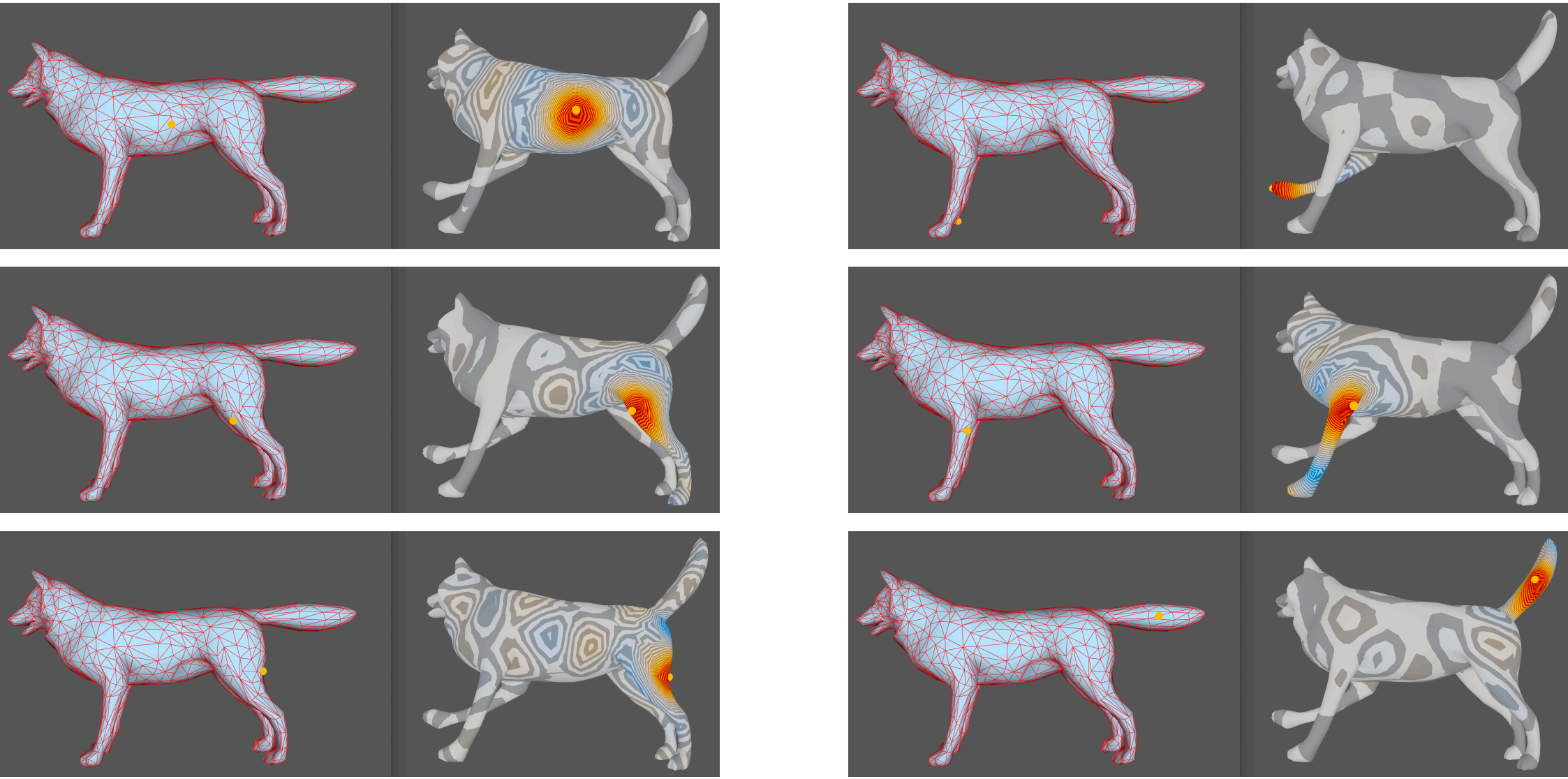
- We find f^i, g^i by
  - placing corresponding landmarks on M_1 and M_2
  - associating a set of heat kernels with each landmark
- Secret sauce:
  - the delta functions should match
  - Use ICP in eigen-coefficient space
  - Use close to isometric shapes



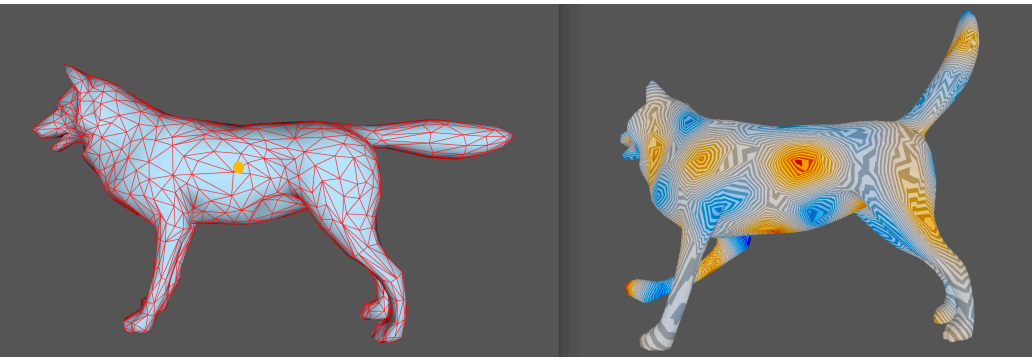
M_1

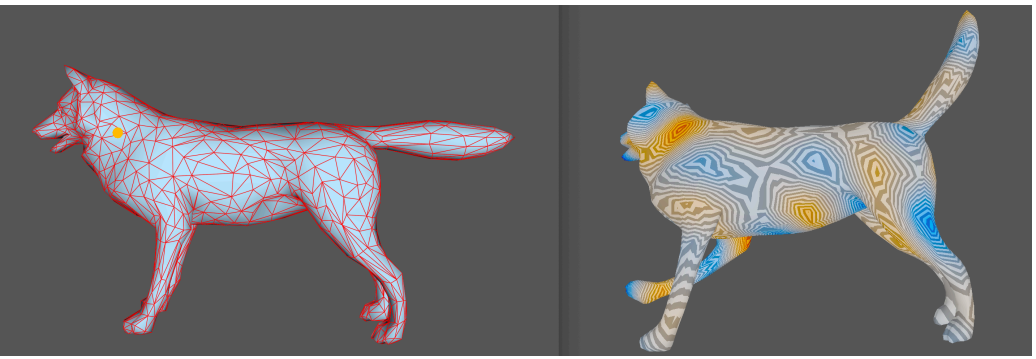
M_2

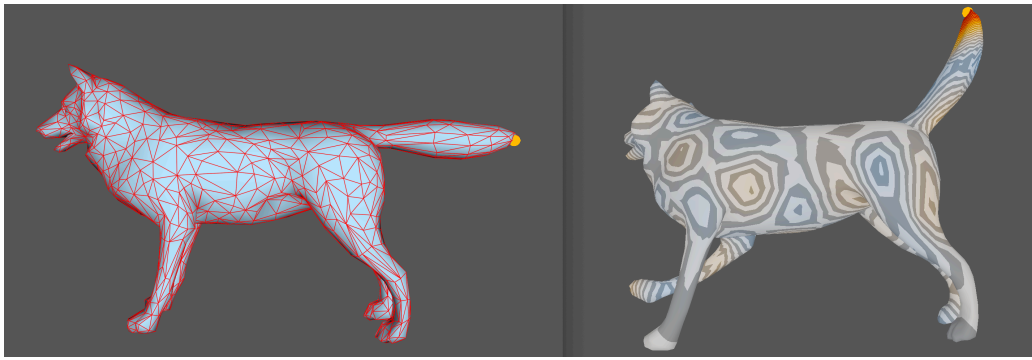
Optimized

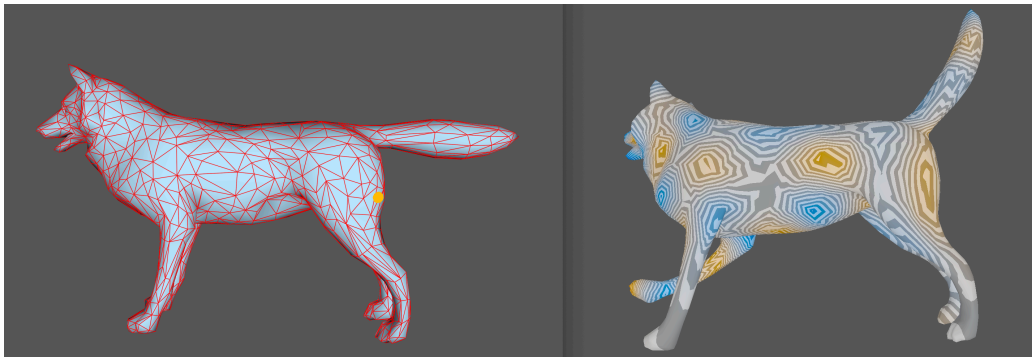


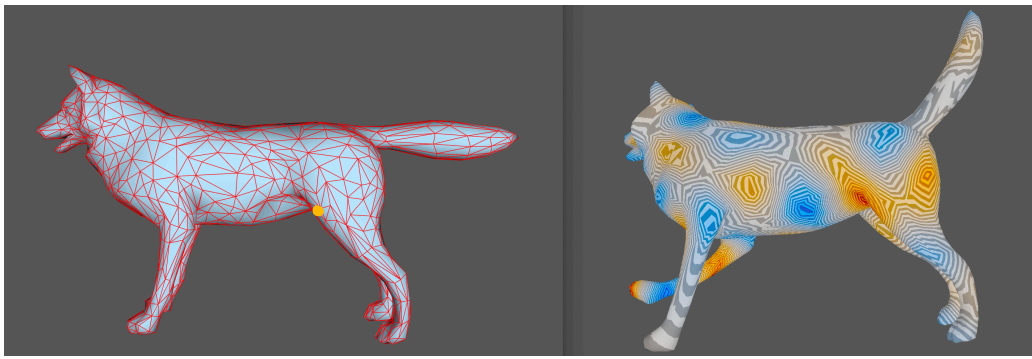
Un-Optimized







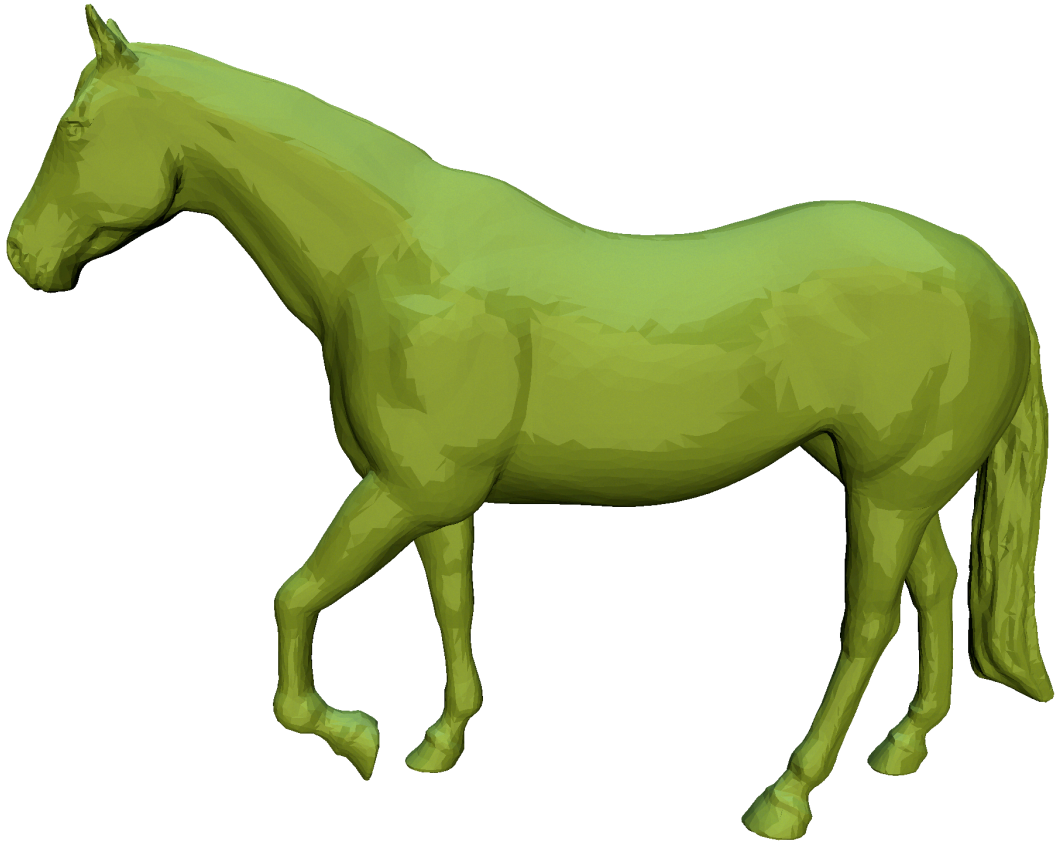




Part 2

In which we encounter skeletons, implicits, and distance fields

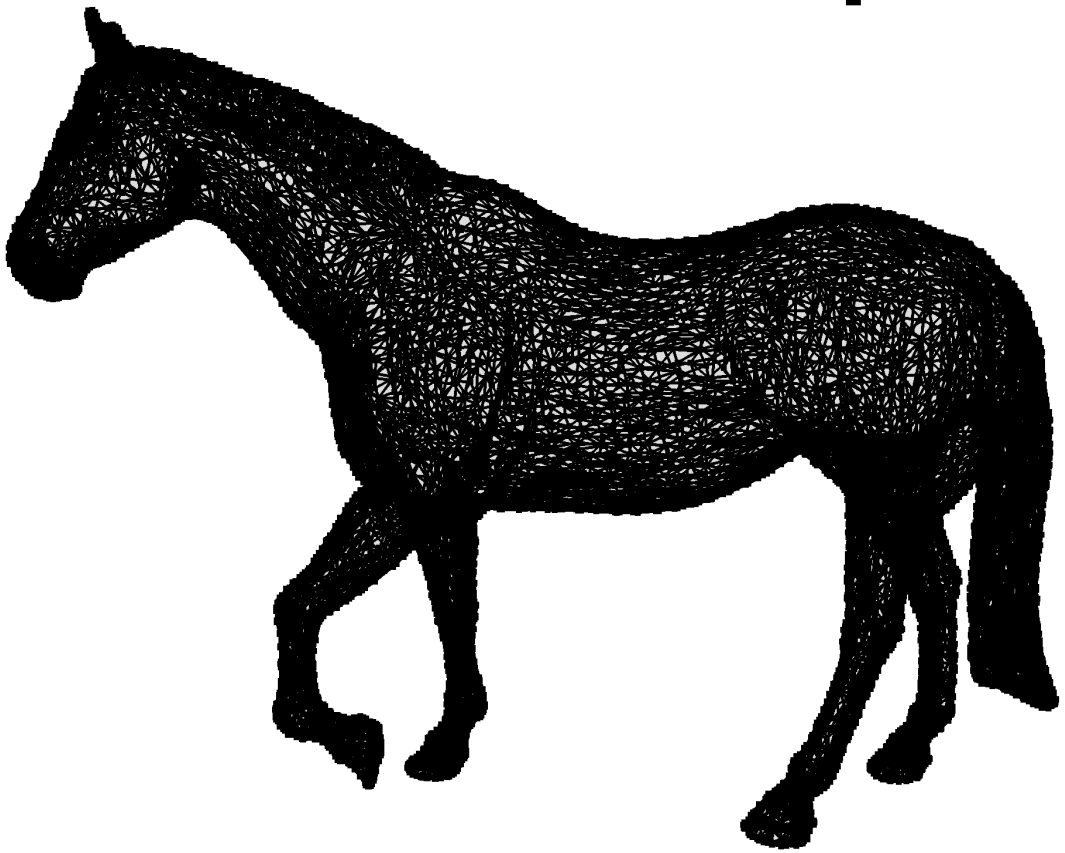
Skeletonization of 3D Meshes



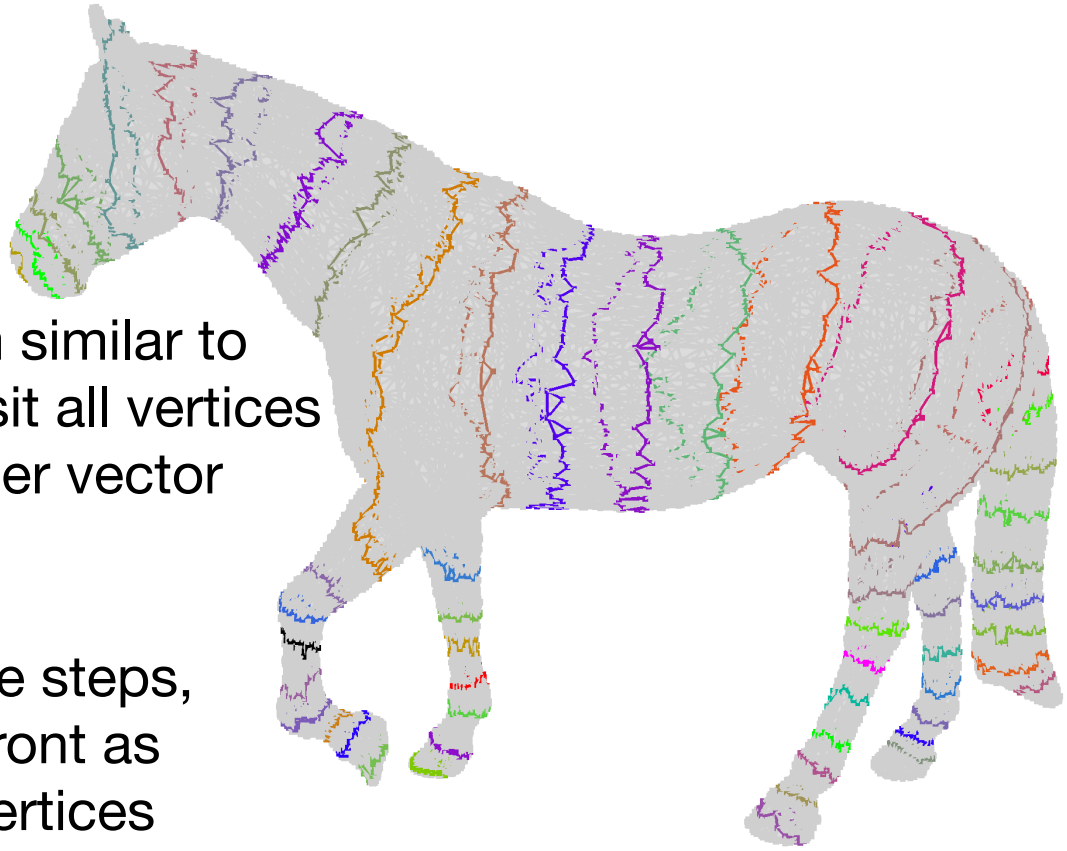
Fiedler Vector

Aka first non-constant eigenvector of the Laplace-Beltrami Operator

Mesh as Graph

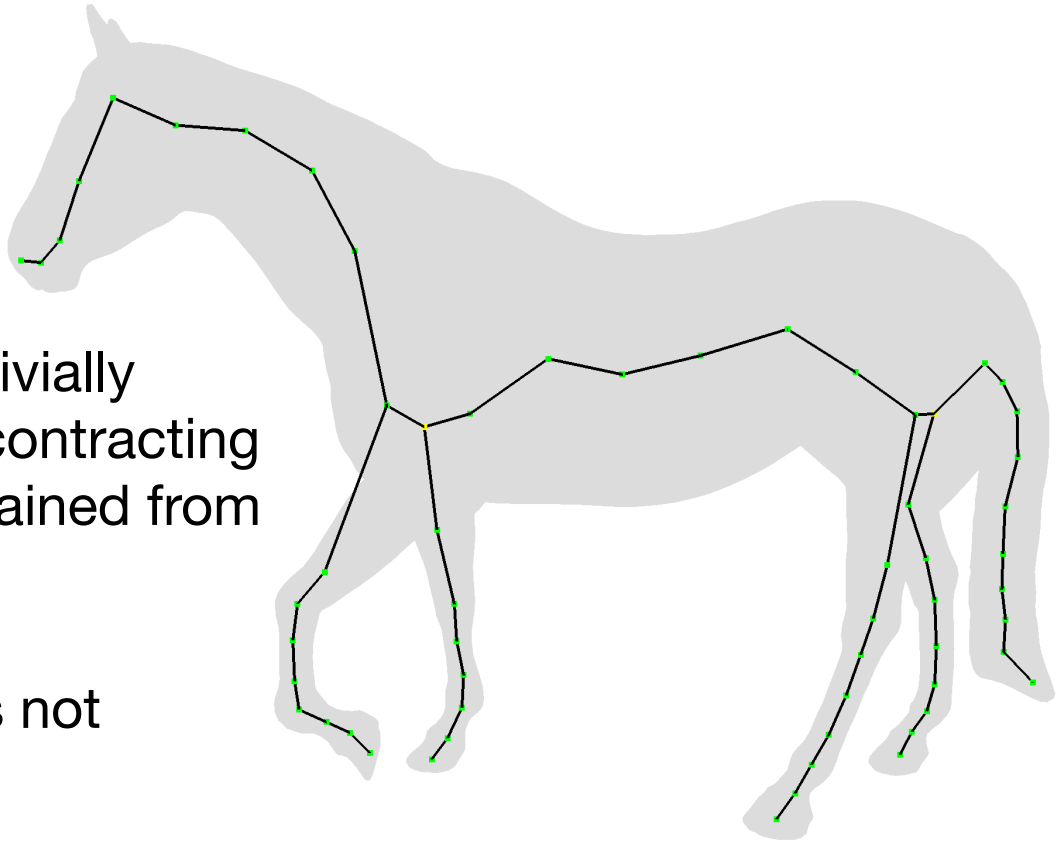


Separators from Fiedler vector



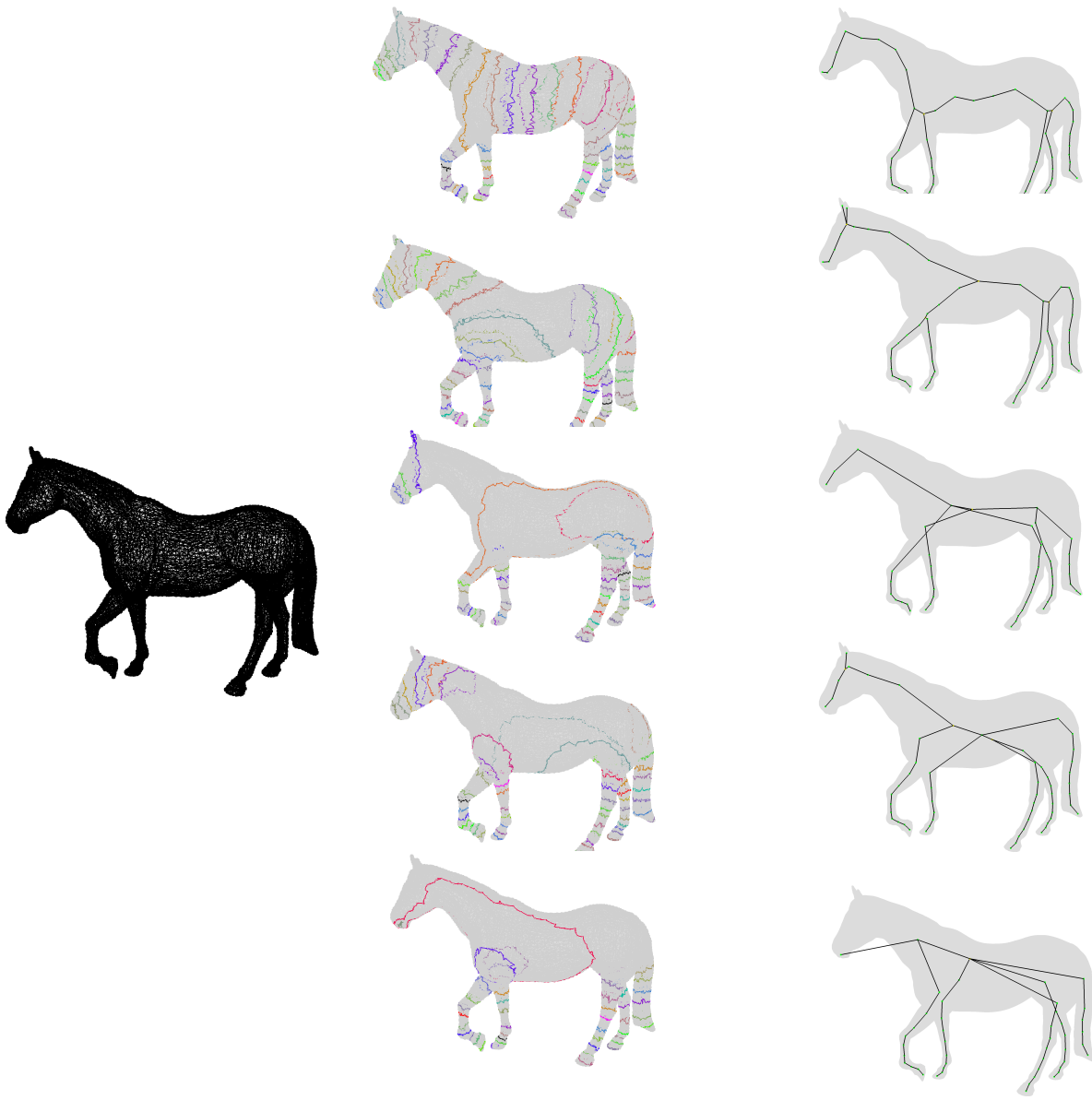
- Using algorithm similar to Dijkstra's we visit all vertices in order of Fiedler vector value
- For specific time steps, we output the front as a selection of vertices (color coded)

Skeleton

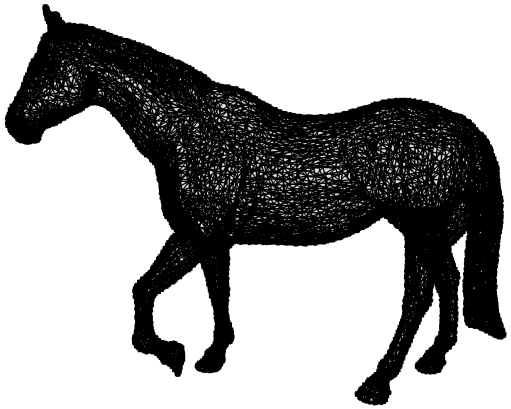


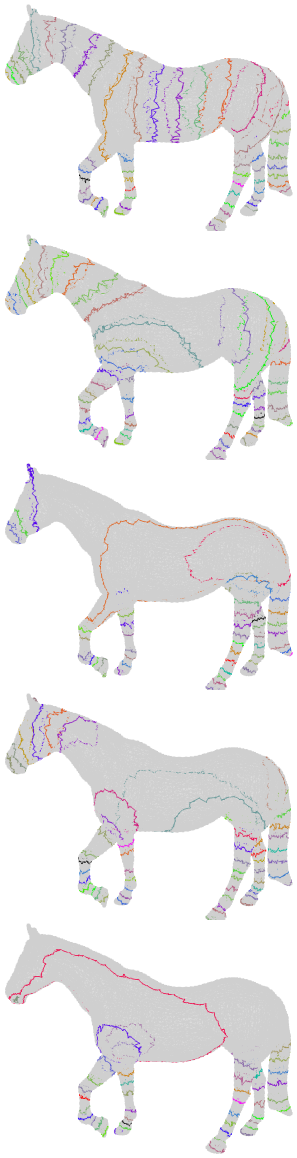
- A skeleton is trivially computed by contracting separators obtained from front sets.

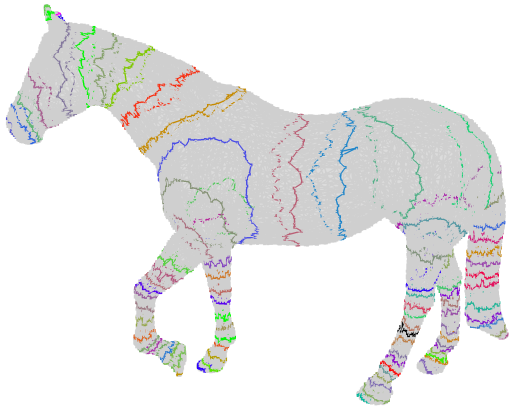
- The skeleton is not satisfactory

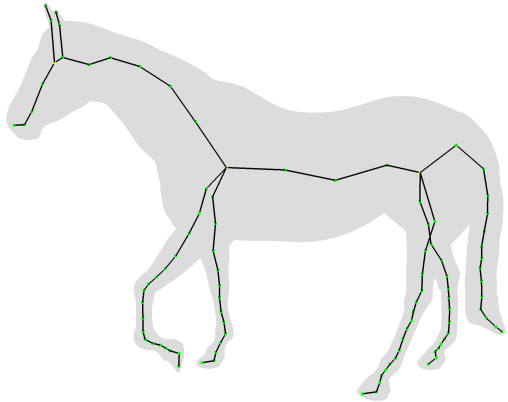


- Repeat the process for several eigenvectors of the Laplace-Beltrami eigenvector
- Results are increasingly hopeless …



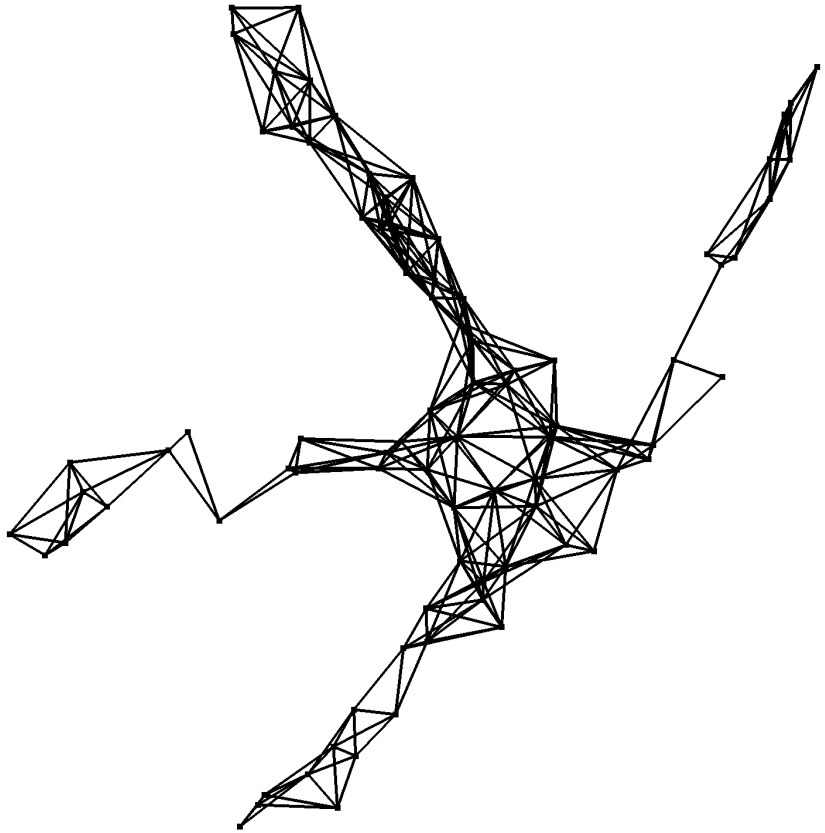




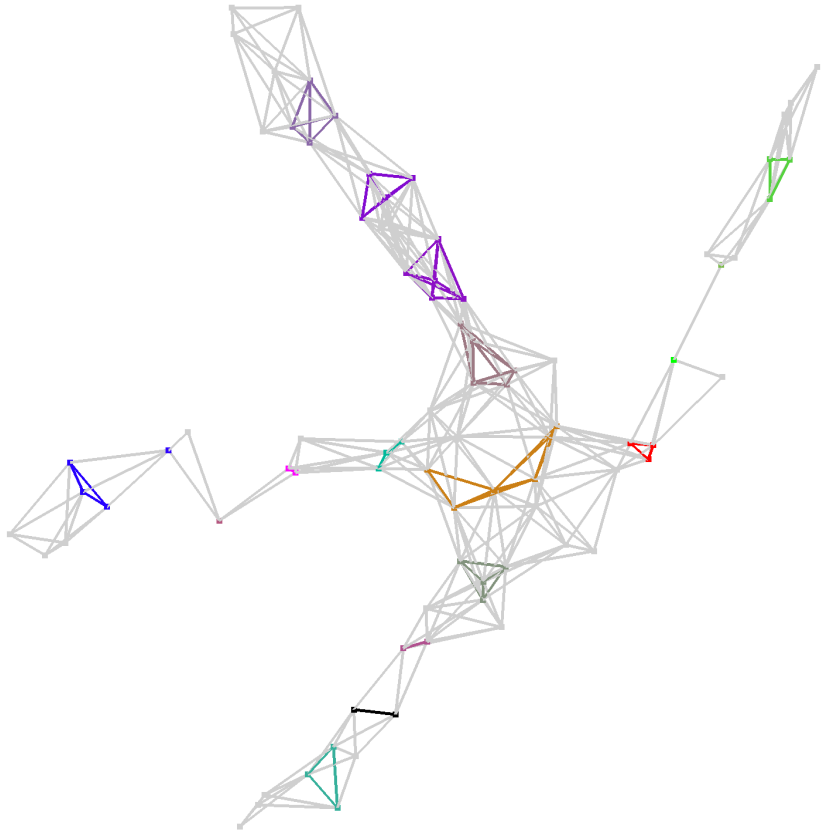


- We can significantly improve the skeleton by packing separators from a variety of eigenvectors

Computing Local Separators

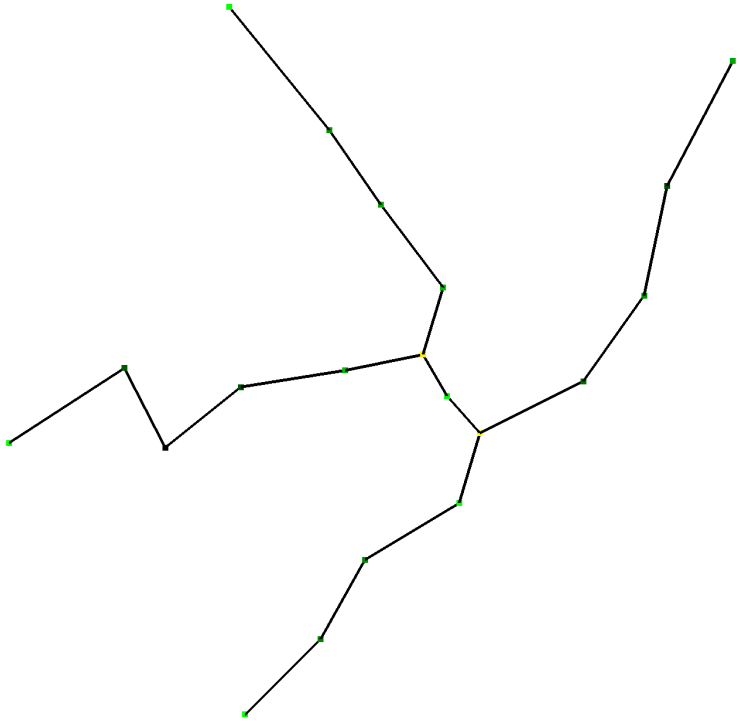


Computing Local Separators

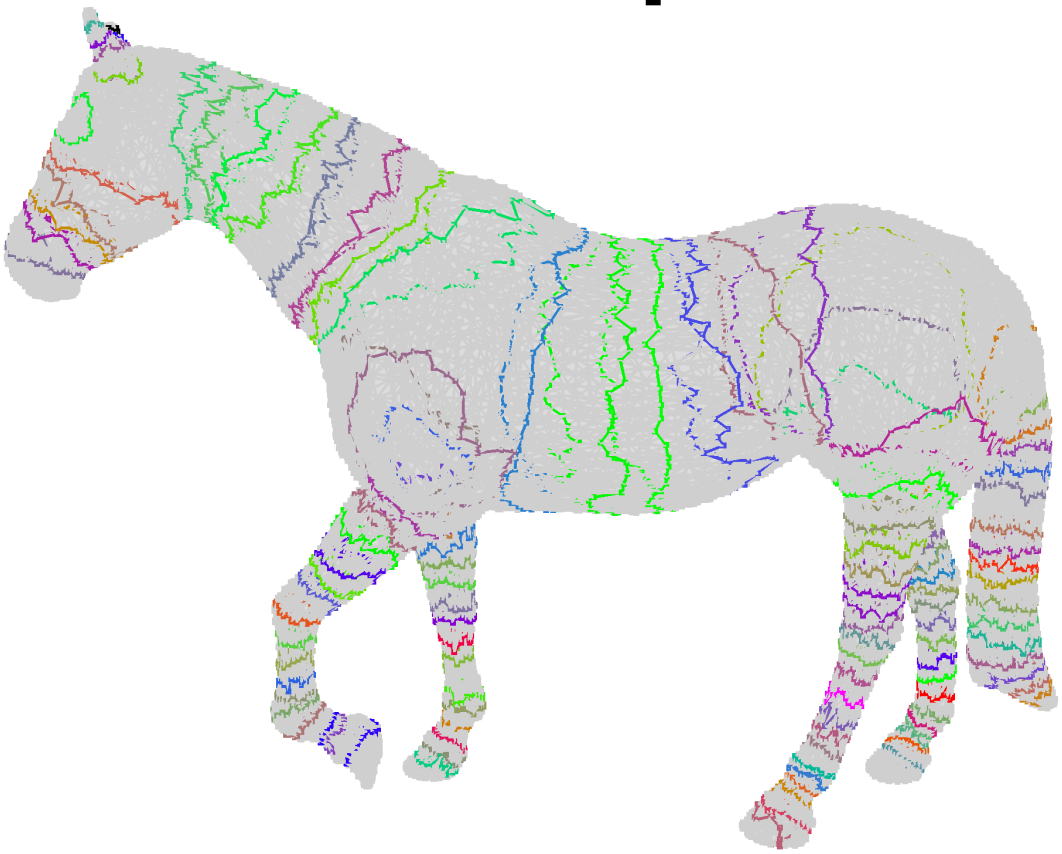


Local separators are separators of a subgraph. In practice, we grow a cluster of vertices and a separator is found when the front breaks into two components

Skeleton from Local Separators

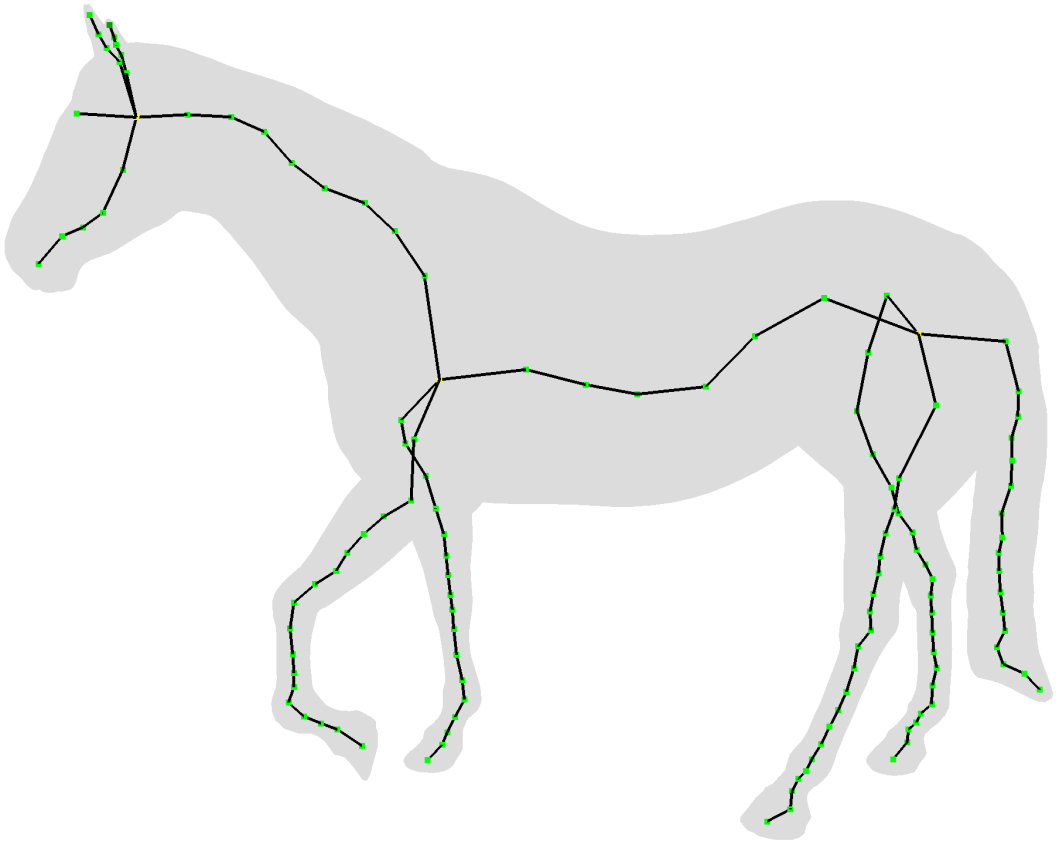


Local Separators



Works in PyGEL

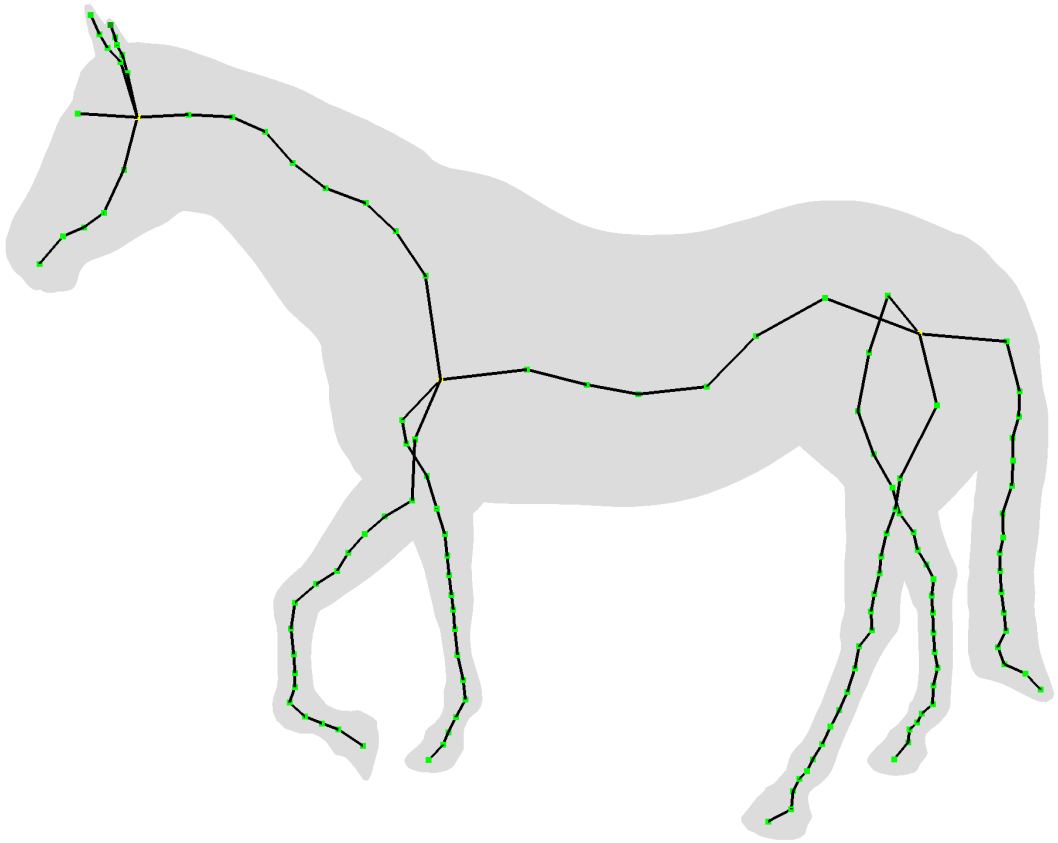
Skeleton from Local Separators



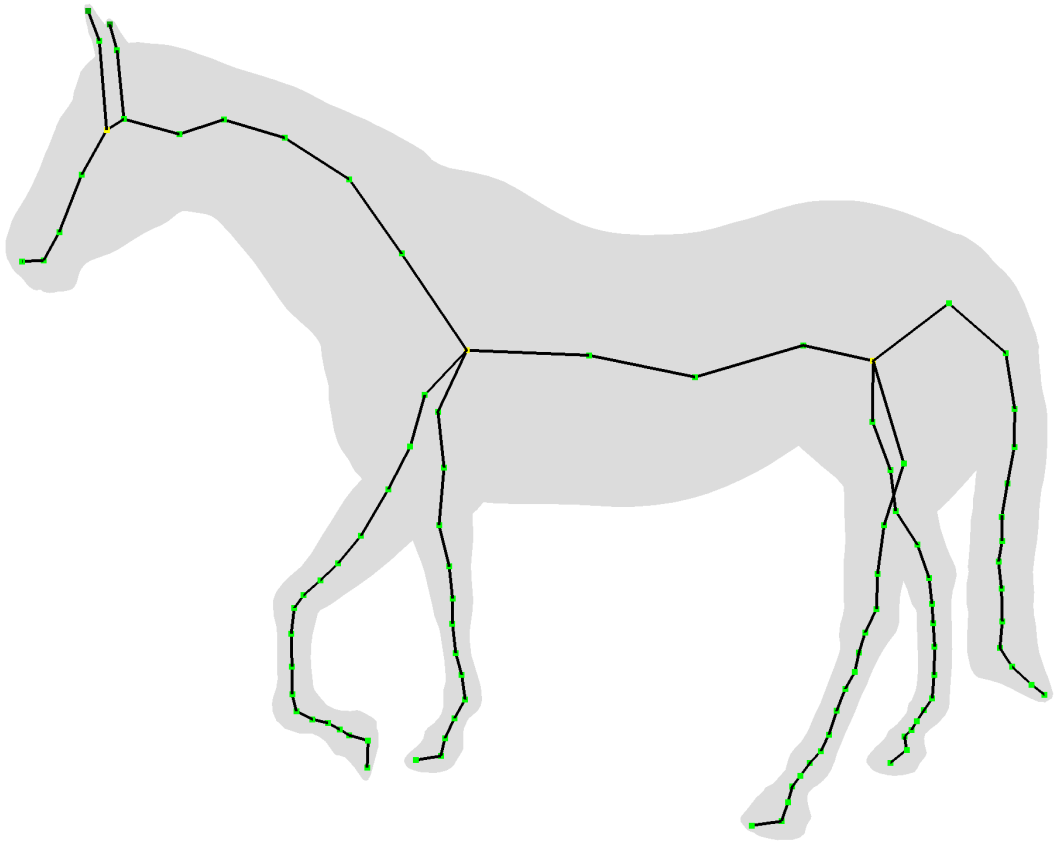
Works in PyGEL

AB and Eva Rotenberg. “Skeletonization via Local Separators”. In: Transactions on Graphics (2021). Accepted for publication.

Skeletons Compared



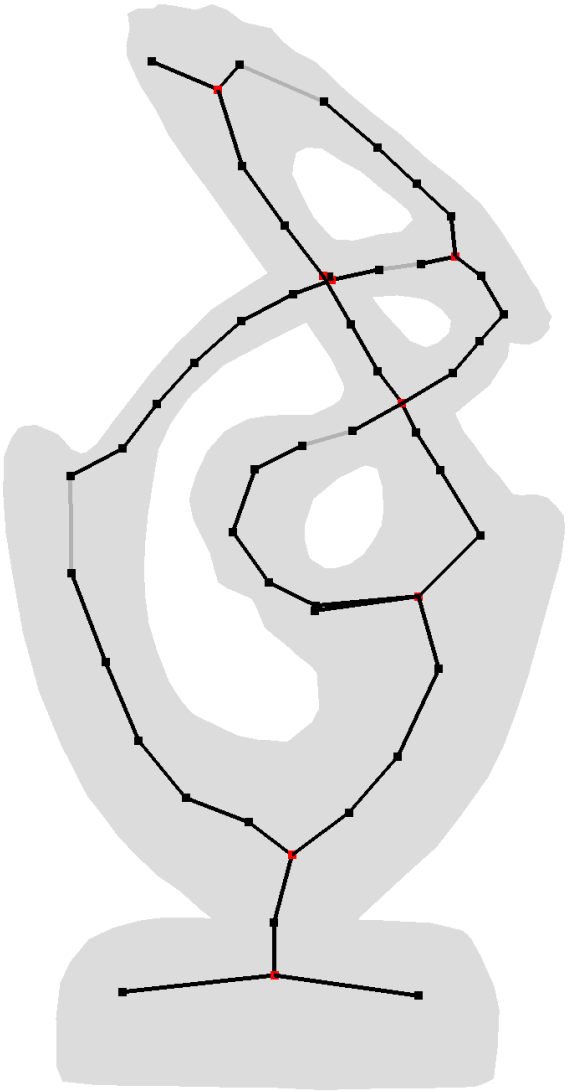
Local separators



LBO Eigen vector skeleton

Topology

- Each edge not in a spanning tree corresponds to a loop in the graph
- Thus, the skeleton provides both geometric and topological information

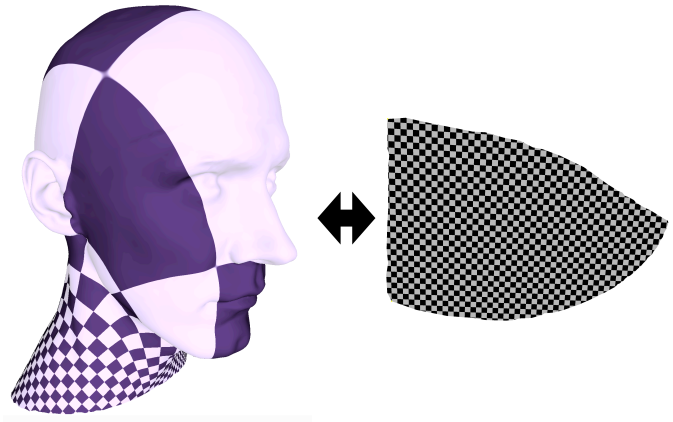


Topology

Three ways to describe topology

Homeomorphism

There is a 1-1, onto, invertible Mapping between homeomorphic shapes



Edelsbrunner, Herbert, and John Harer. Computational topology: an introduction. American Mathematical Soc., 2010.

Homotopy equivalence

Two shapes are homotopy equivalent if one can be smoothly deformed into the other

Homology

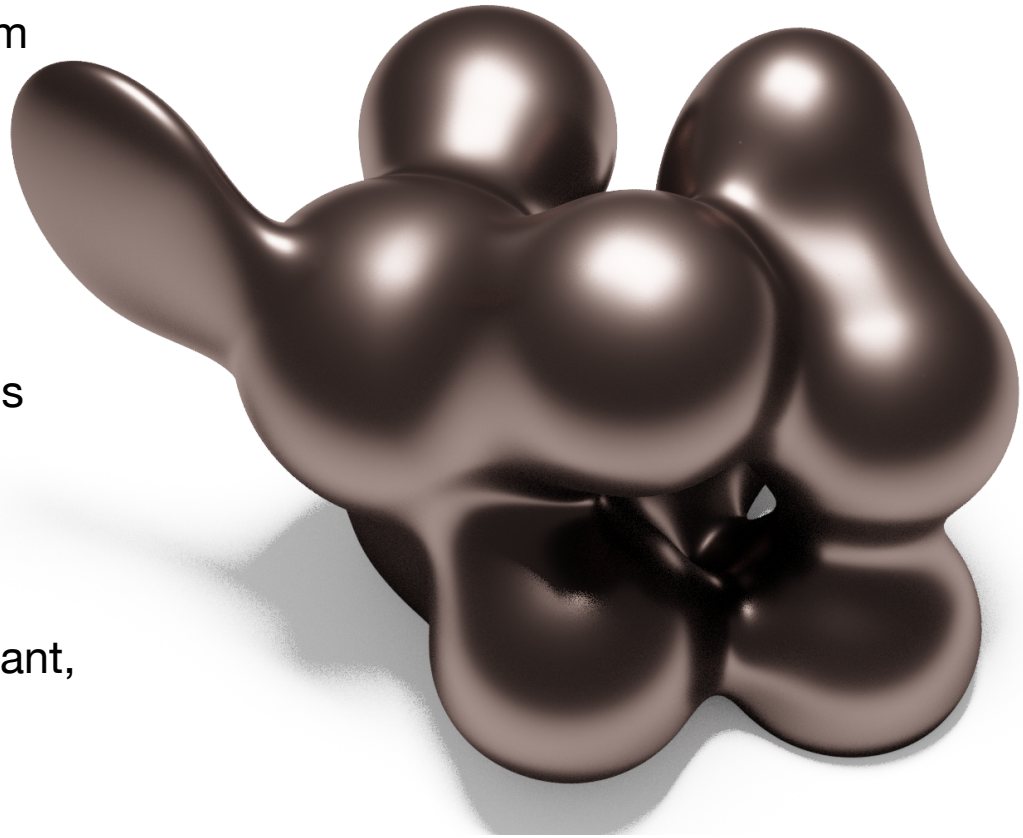
Establishes ways to distinguish classes of cycles

An Implicit Surface

- Implicit surfaces are simply functions of the form $f : \mathbb{R}^3 \rightarrow \mathbb{R}$
- The surface is $S = \left\{ \mathbf{x} \,|\, f(\mathbf{x}) = 0 \right\}$
- We ask that $\nabla f \neq 0$ on S
- It is easy to compose implicits as a sum of basis functions

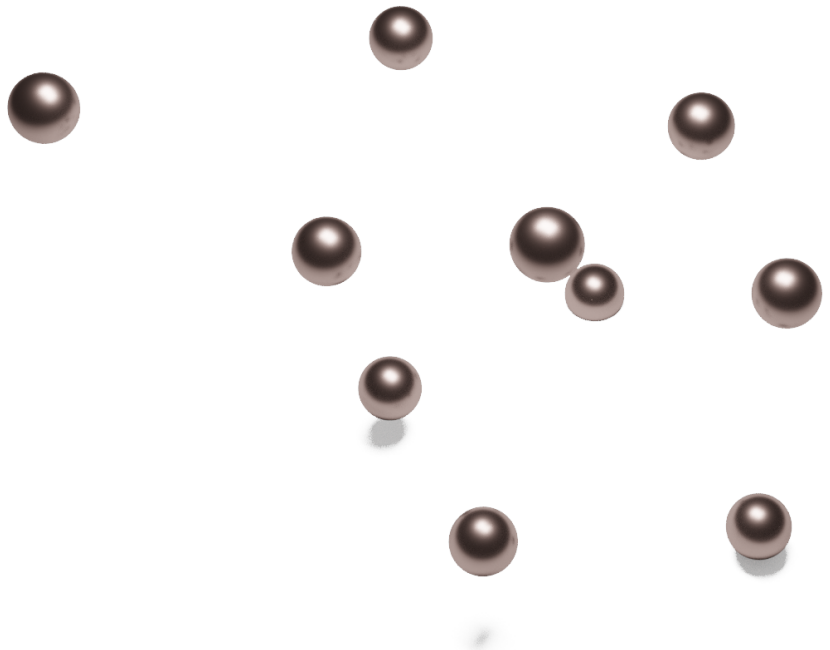
$$f(\mathbf{x}) = \sum_i \theta_i(\mathbf{x}) \,,$$

where e.g. $\theta_i(\mathbf{x}) = k_i \exp(\mathbf{x}^T \mathbf{M}_i \mathbf{x})$, k_i is a constant, and $\mathbf{M}_i$ is a positive definite 3×3 matrix



An Implicit Surface

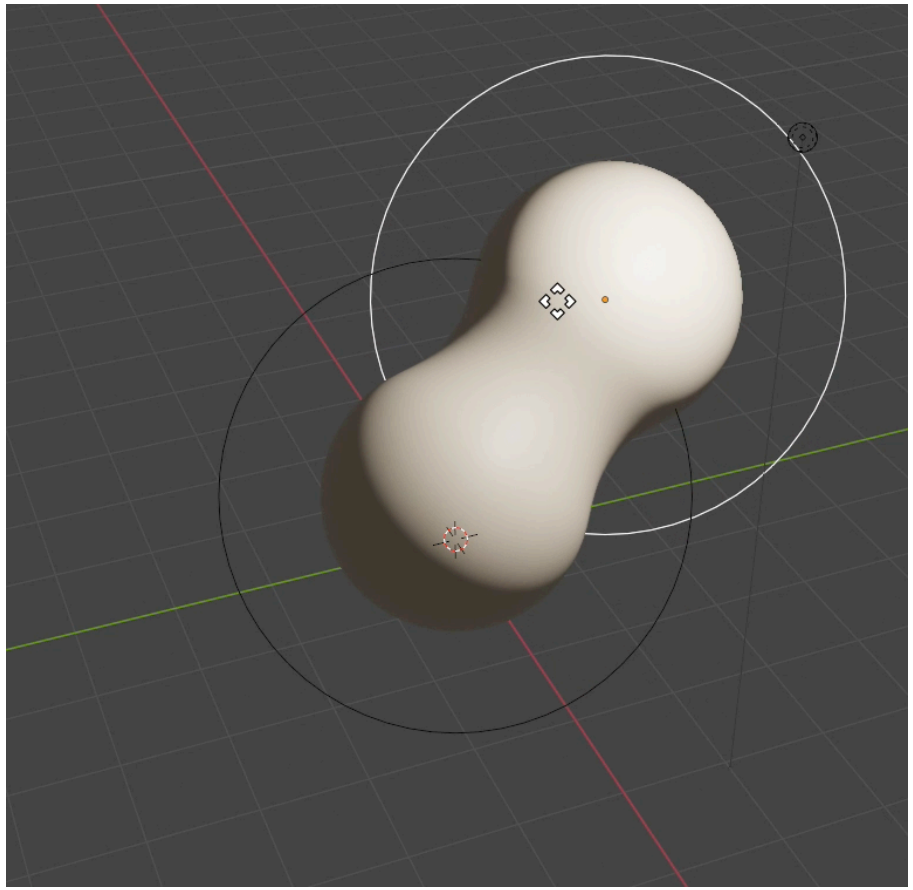
- Setting $\mathbf{M} = \mathbf{I}$, we get result on the right
- This is done in Blender
- They are called “Metaballs” in Blender



Blinn, James F. "A generalization of algebraic surface drawing." ACM transactions on graphics (TOG) 1.3 (1982): 235-256.

An Implicit Surface

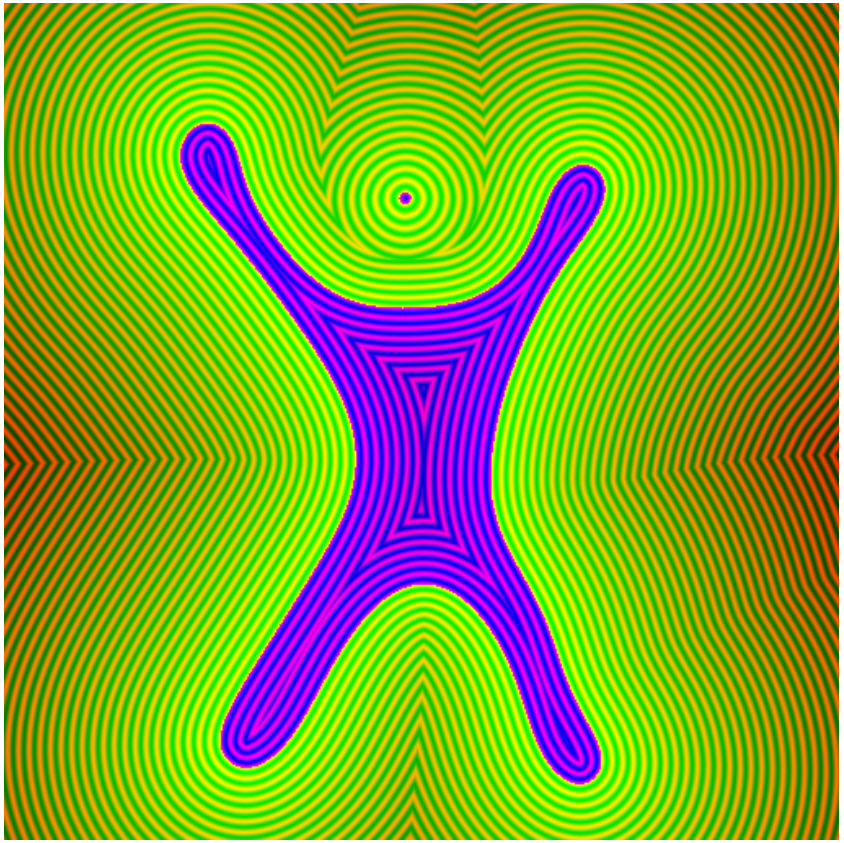
- Now with two implicits and one moving around.
- The blending shows clearly
- No - I don’t really know that they are Gaussian’s but that is what Jim Blinn used.



Blinn, James F. "A generalization of algebraic surface drawing." ACM transactions on graphics (TOG) 1.3 (1982): 235-256.

Discrete Distance Field

- Given a surface, S, a distance field of that surface is a function: $d_S(x,y,z)$
- $d_S(x,y,z) = 0$ on surface
- $d_S(x,y,z) < 0$ inside
- $d_S(x,y,z) > 0$ outside
- d_S is a discrete distance field if it is sampled



Properties

- Defining Propery: The gradient is unit length

$$\|\nabla d_S\| = 1$$

- The Mean Curvature (of an isocontour) is simply the divergence of the gradient/Laplacian/trace of the Hessian

$$H = \nabla \cdot \nabla d_S = \Delta d_S = \frac{\partial^2 d_S}{\partial x^2} + \frac{\partial^2 d_S}{\partial y^2} + \frac{\partial^2 d_S}{\partial z^2}$$

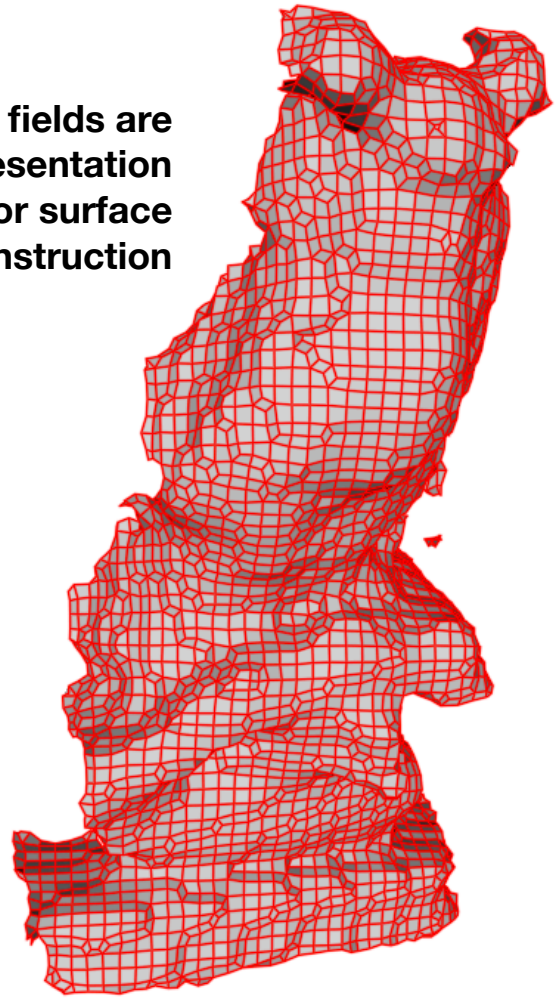
Jones, Mark W., J. Andreas Baerentzen, and Milos Sramek. "3D distance fields: A survey of techniques and applications." IEEE Transactions on visualization and Computer Graphics 12.4 (2006): 581-599.

Distance Field Variations

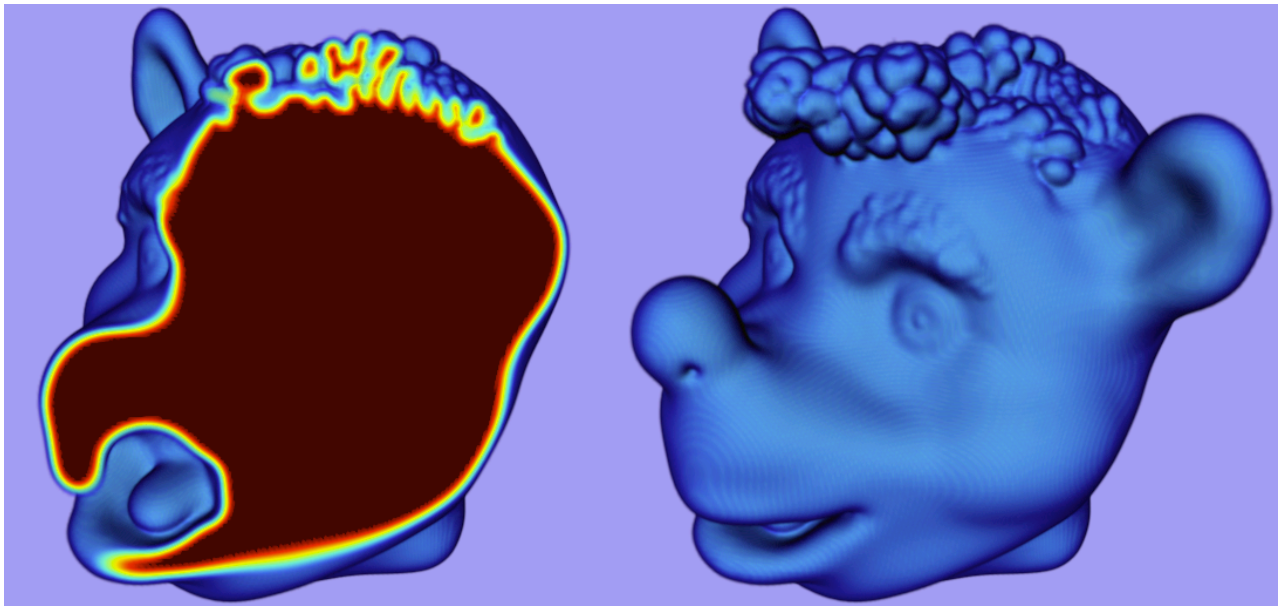
- **Scalar field**. Sometimes our distance field does not contain actual distance values: we can usually fix that.
- **Signed vs unsigned**. An unsigned distance field provides no inside-outside information
- **TSDF.** Truncated Signed Distance Fields only inform us of the distance in a narrow band around the surface

Applications

Distance fields are generally the representation of choice for surface reconstruction



Distance Fields can be used for dynamic surfaces, i.e. level set representation



Computing Distance Fields

- From triangle meshes
- From a set of voxels
- From an existing scalar field

Triangle Mesh to Distance Field

- Input:
  - Triangle Mesh M, Bounding hierarchy B(M), Voxel grid G
- Output: distance field D
- For each voxel v in G:
  - Locate closest triangle: t = B.closest(v)
  - compute D[v] = t.dist(v)

Works in PyGEL

From Scalar Field to Distance Field

- Given a scalar field, Φ, we iteratively solve

$$\frac{\mathrm{d}\Phi}{\mathrm{d}t} + s(\Phi_0)(\|\nabla\Phi\| - 1) = 0$$

- where

$$s(x) = \frac{x}{\sqrt{x^2 + \epsilon^2}}$$

- Note: it is import to compute the gradient in the upwind direction.

Jones, Mark W., J. Andreas Baerentzen, and Milos Sramek. "3D distance fields: A survey of techniques and applications." IEEE Transactions on visualization and Computer Graphics 12.4 (2006): 581-599.

From Binary Voxels to Distance Field

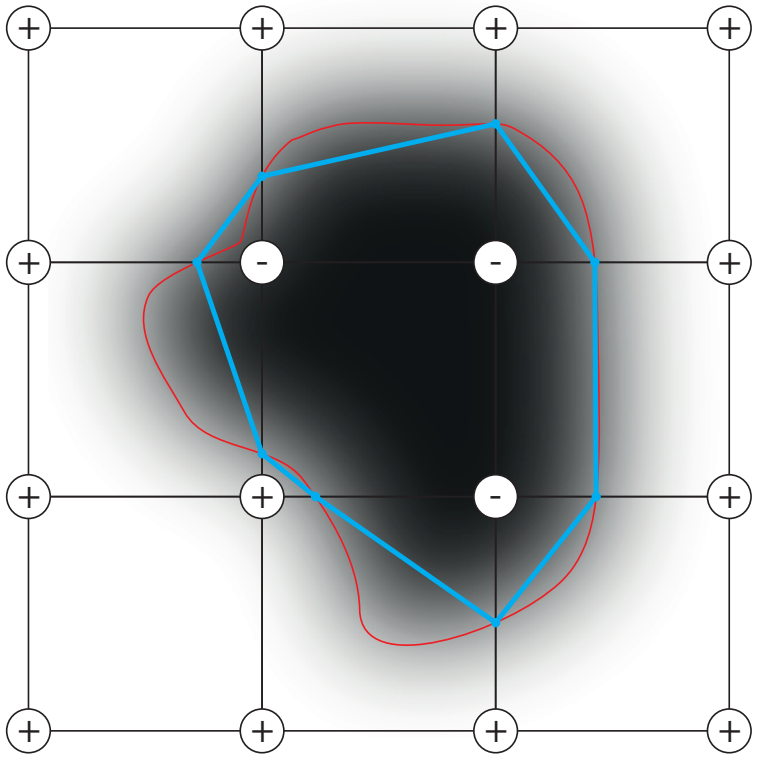
- Starting from a binary voxel grid, the reinitialization method might be slowish.
  - Try the Fast Marching Method (FMM)
- The FMM is based on Dijkstra's shortest path algorithm…
  - to be precise it is Dijkstra!

Techniques for Polygonization

- Find 2D contours and connect
- Grow triangles on the surface
- Divide space into cells, approximate surface in each cell
  - Primal vs dual methods

Isosurface Polygonization

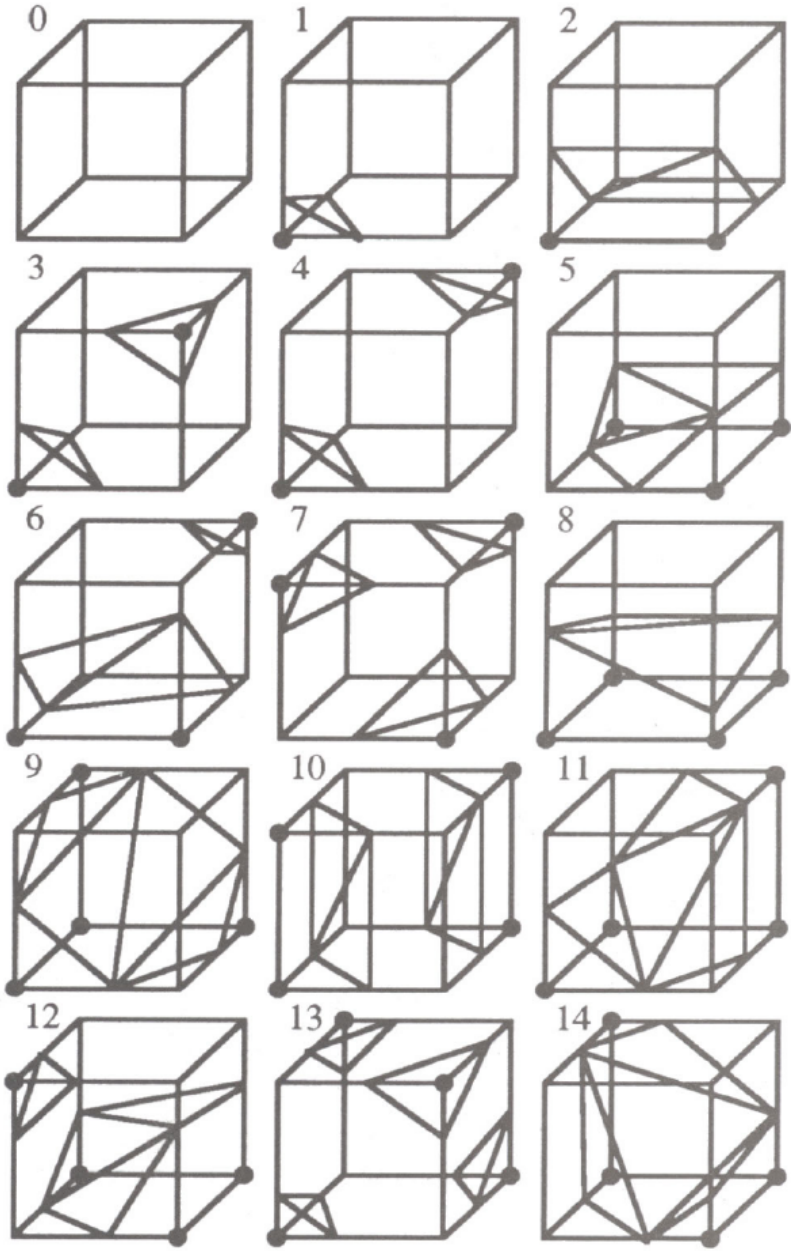
- marching cubes is the most common method.
- Polygonization cell is a cube of eight voxels
- Pro: manifold
- Con: Ugly triangles



Marching Cubes

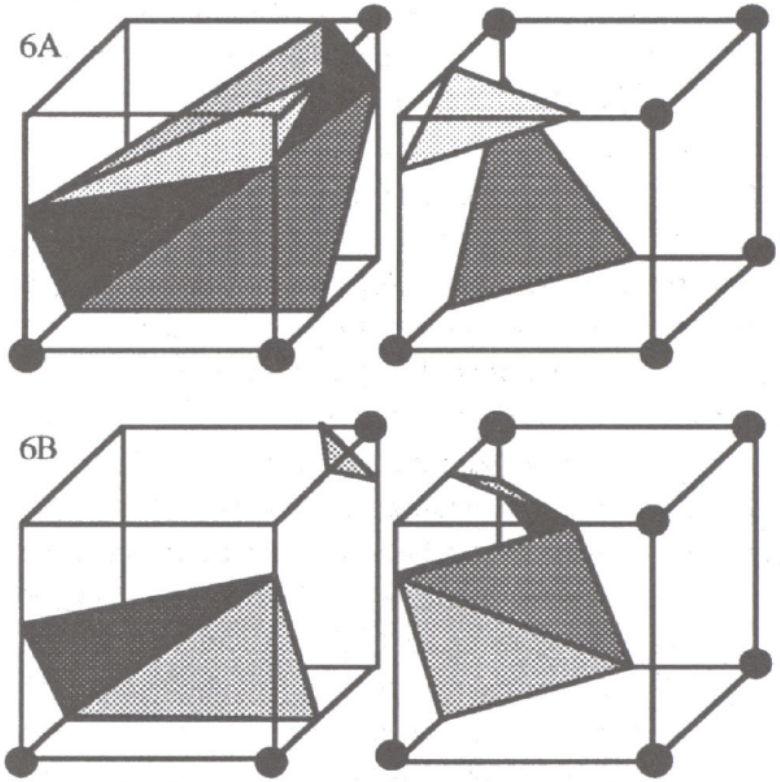
- For each cube:
  - Compute table index
  - Pick triangle configuration
  - Place vertices on isosurface!
  - Optional: Interpolate gradients to get surface normal

Lorensen, William E., and Harvey E. Cline. "Marching cubes: A high resolution 3D surface construction algorithm." ACM siggraph computer graphics 21.4 (1987): 163-169.



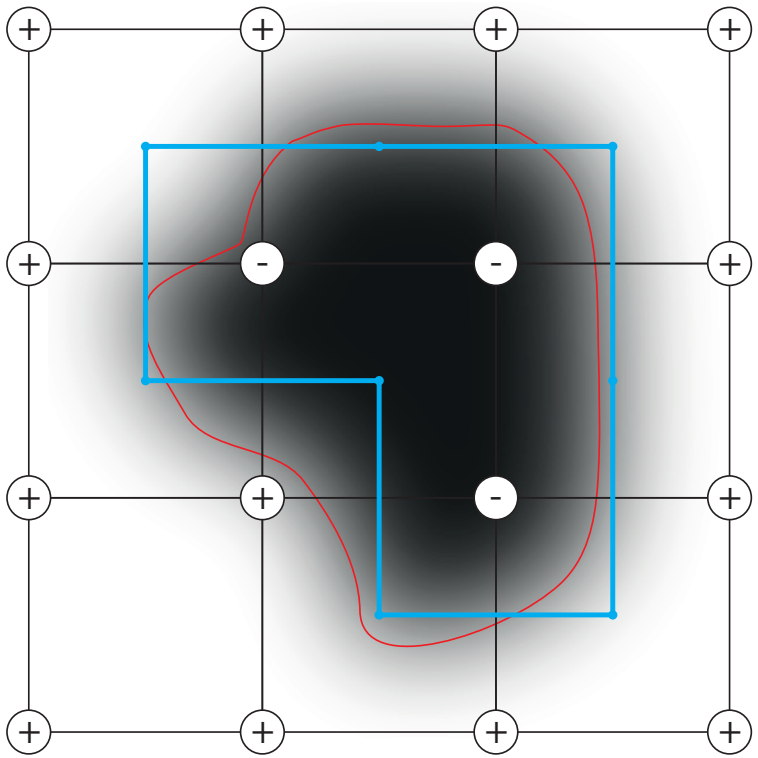
Marching Cubes ambiguity

- Some configurations are ambiguous
- We must select consistent tables



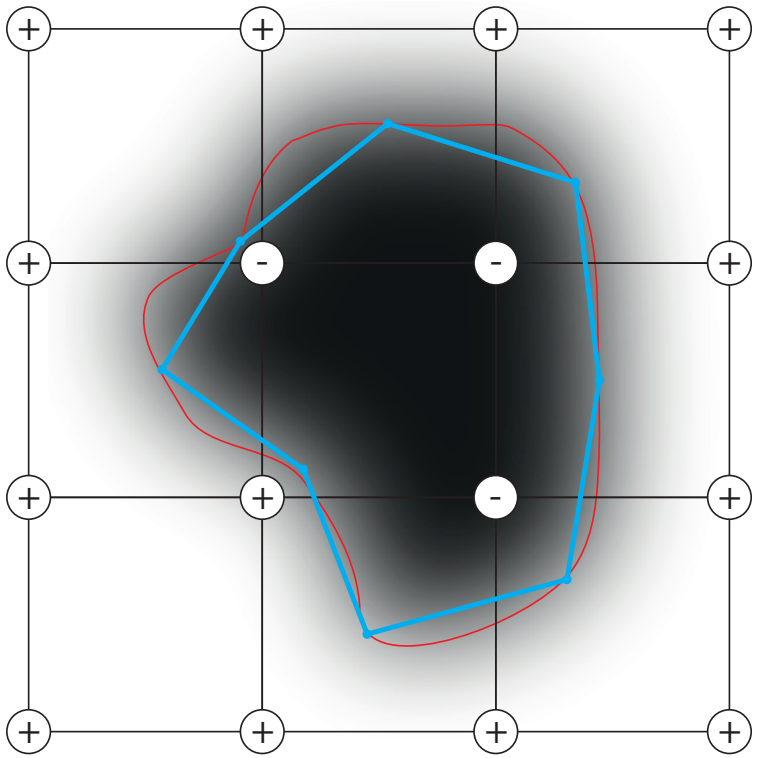
Dual Contouring

- In dual contouring, we make a box around each voxel.
- If neighboring voxel has opposite sign, we emit shared face
- Then we project onto the isosurface
- and split quads into triangles



Dual Contouring

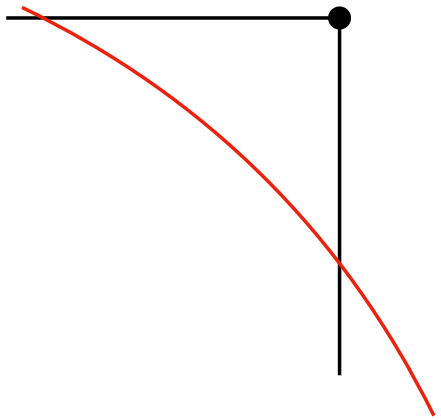
- In dual contouring, we make a box around each voxel.
- If neighboring voxel has opposite sign, we emit shared face
- Then we project onto the isosurface
- and split quads into triangles



Dual Contouring

Placing vertices

- In DC vertices are not constrained to edges
  - Additional freedom avoids most poorly shaped triangles

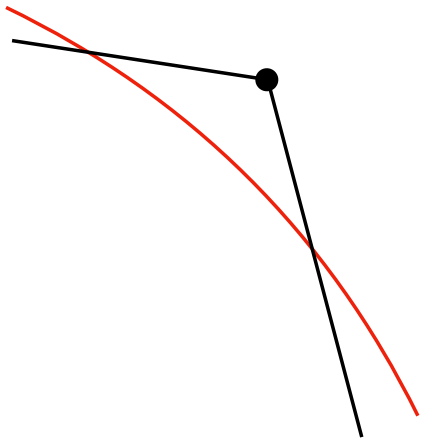


Dual Contouring

Placing vertices

- In DC vertices are not constrained to edges
  - Additional freedom avoids most poorly shaped triangles
  - Placing vertices:

1. smooth mesh

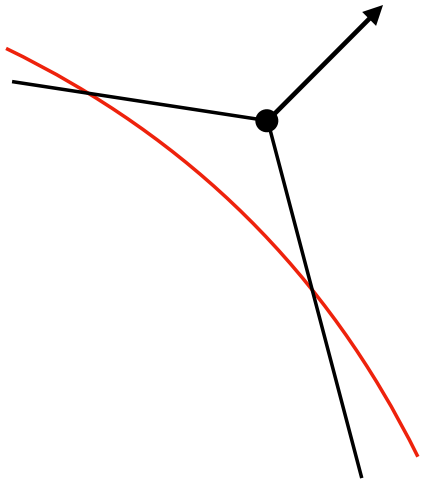


Dual Contouring

Placing vertices

- In DC vertices are not constrained to edges
  - Additional freedom avoids most poorly shaped triangles
  - Placing vertices:

1. smooth mesh
2. compute normal for each vertex

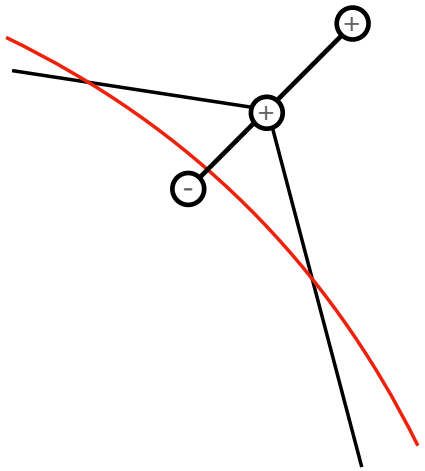


Dual Contouring

Placing vertices

- In DC vertices are not constrained to edges
  - Additional freedom avoids most poorly shaped triangles
  - Placing vertices:

1. smooth mesh
2. compute normal for each vertex
3. sample both directions along normal

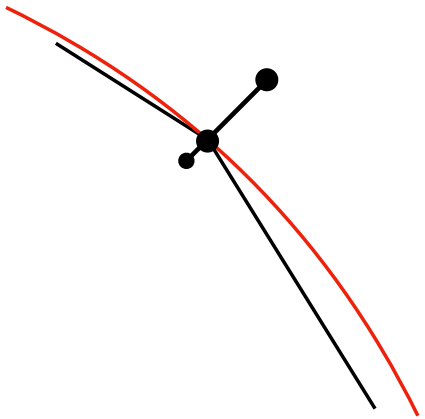


Dual Contouring

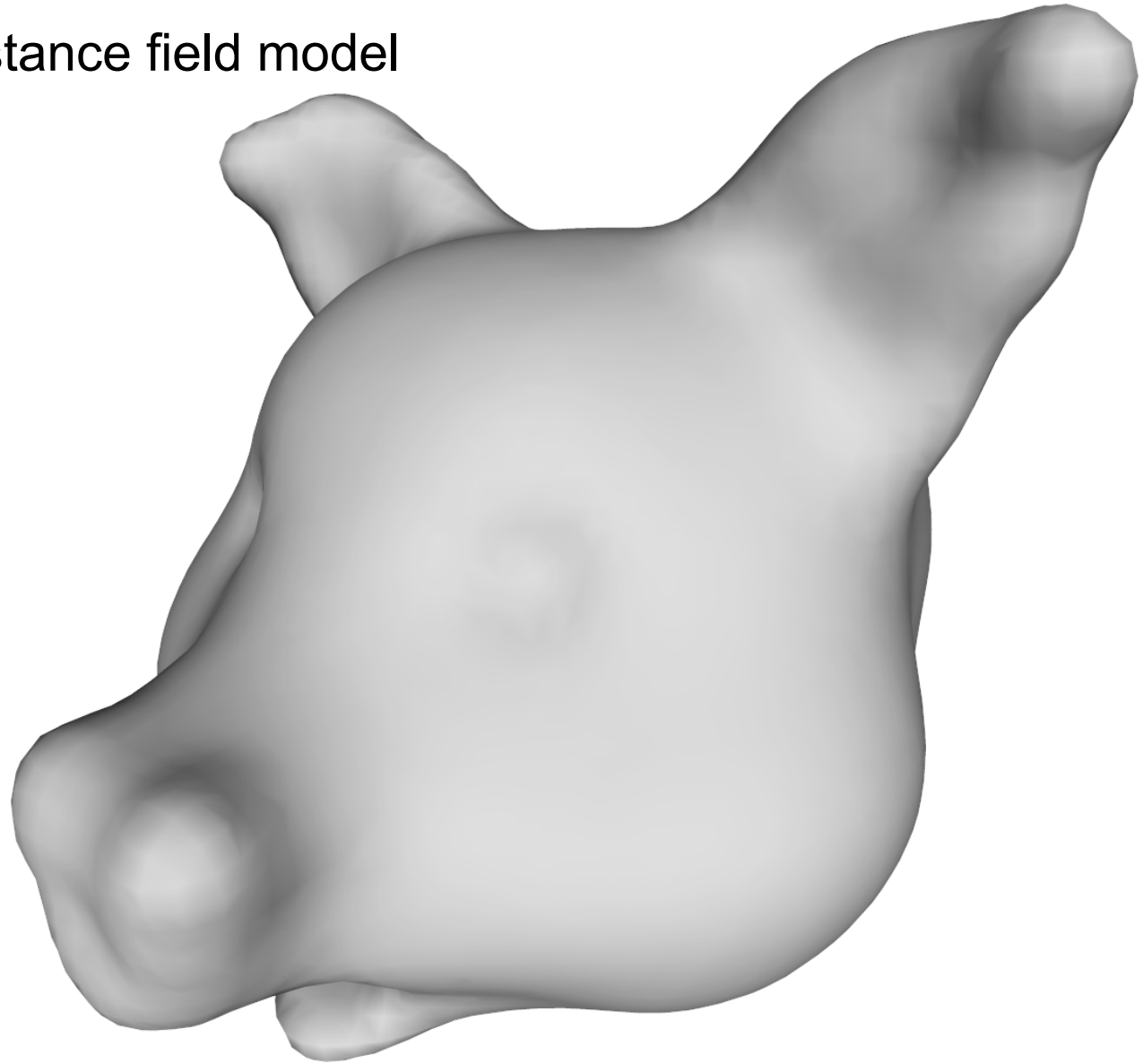
Placing vertices

- In DC vertices are not constrained to edges
  - Additional freedom avoids most poorly shaped triangles
  - Placing vertices:

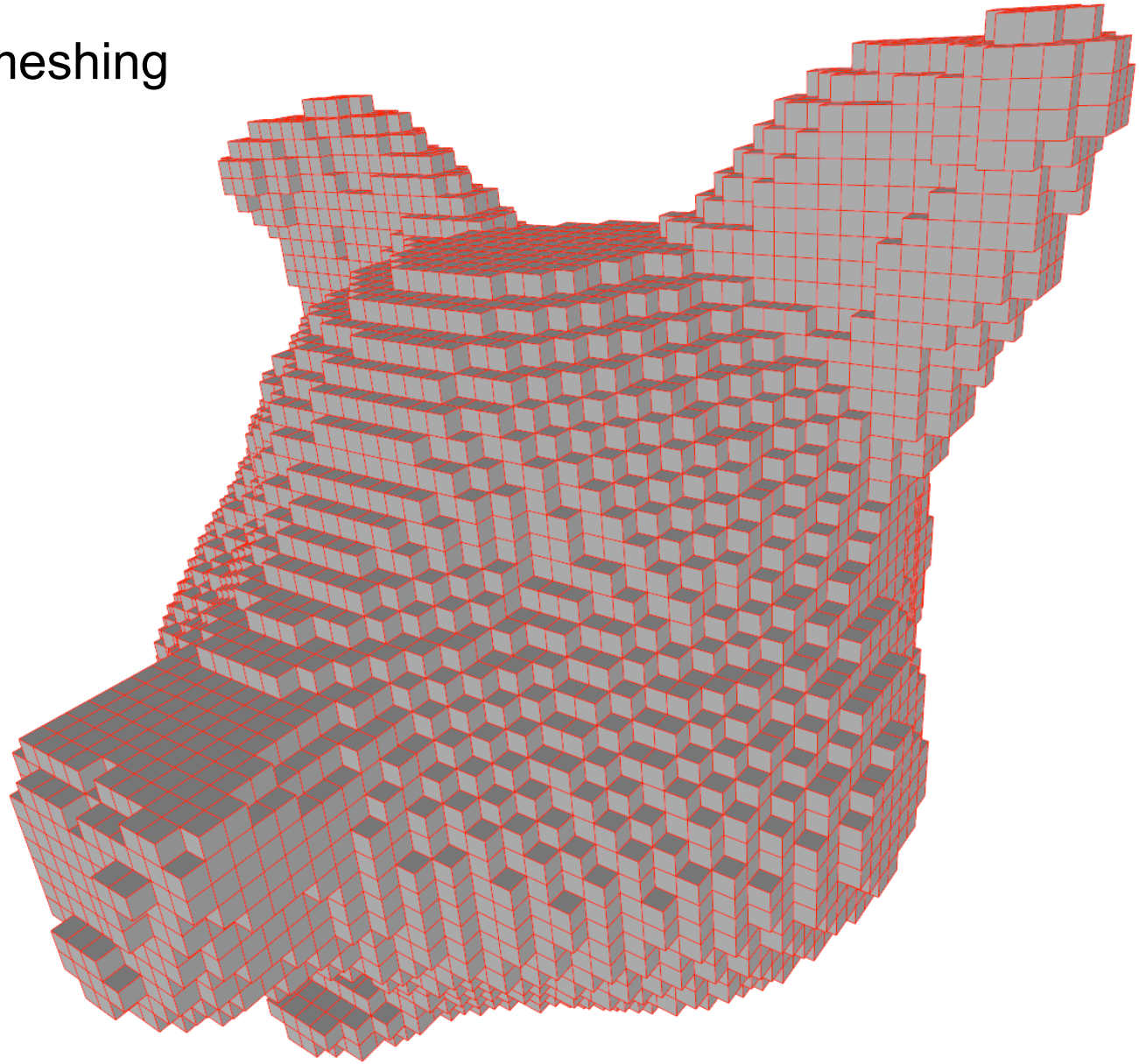
1. smooth mesh
2. compute normal for each vertex
3. sample both directions along normal
4. move vertex to intersection



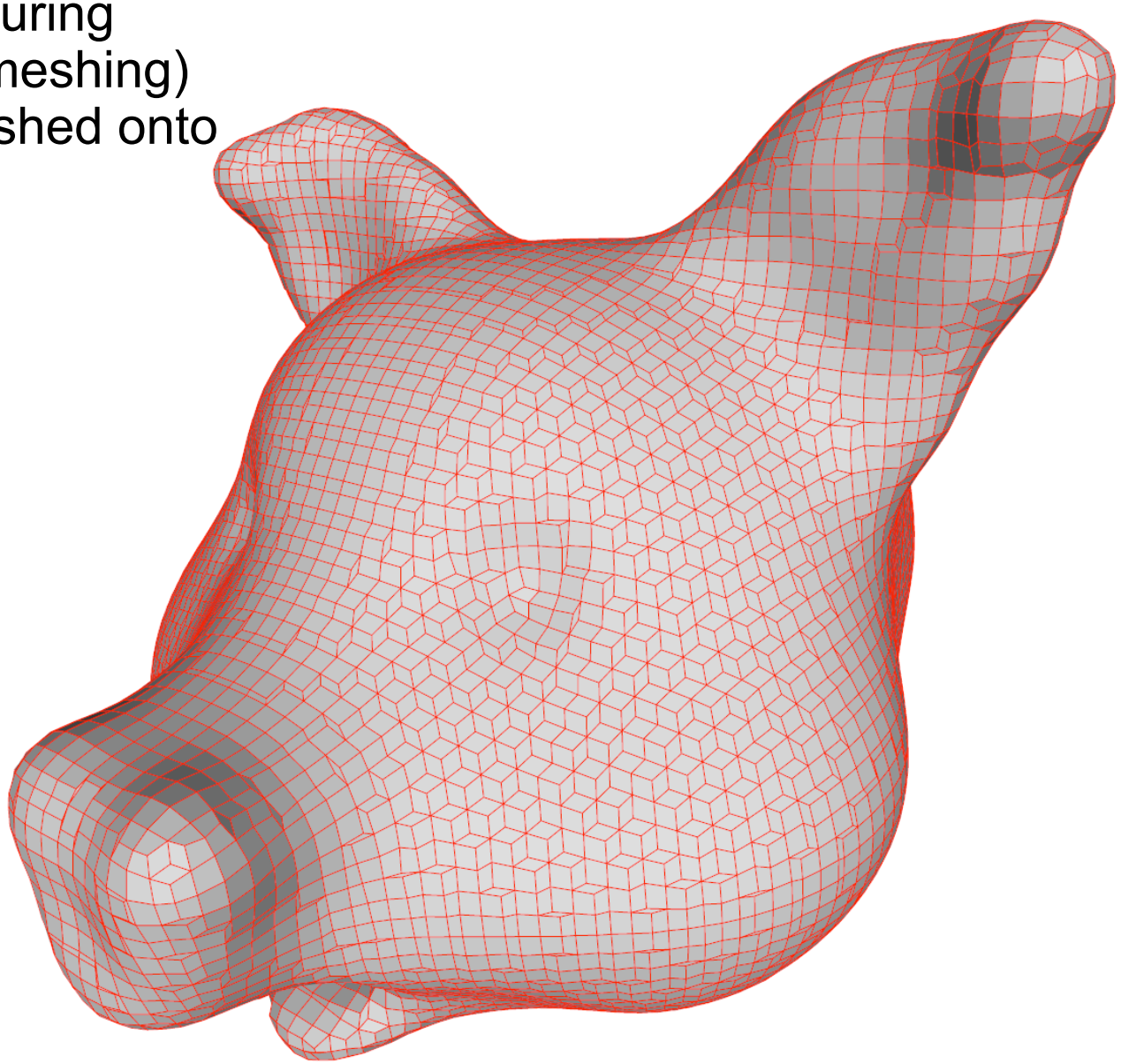
Original distance field model



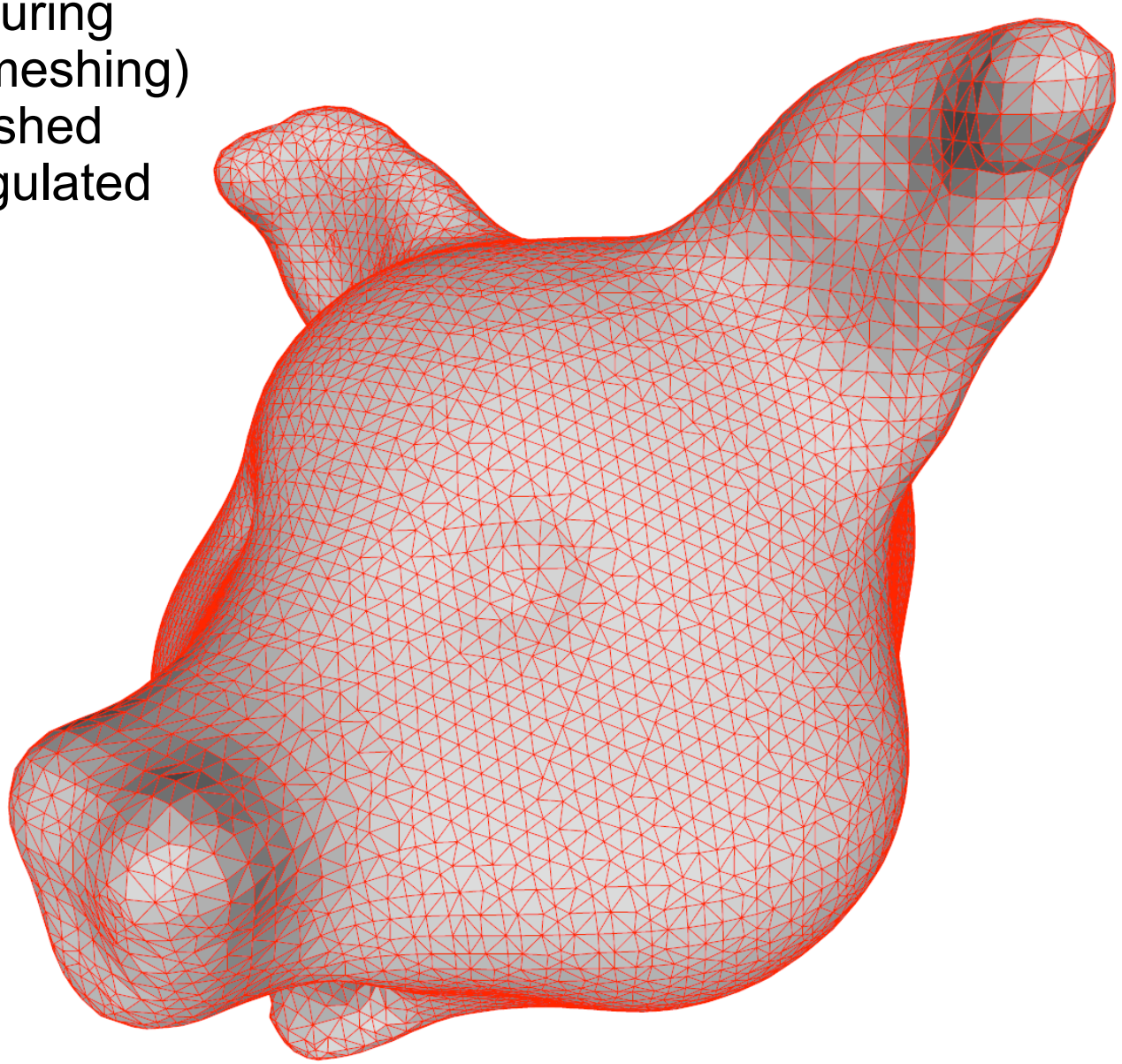
Cuberille meshing



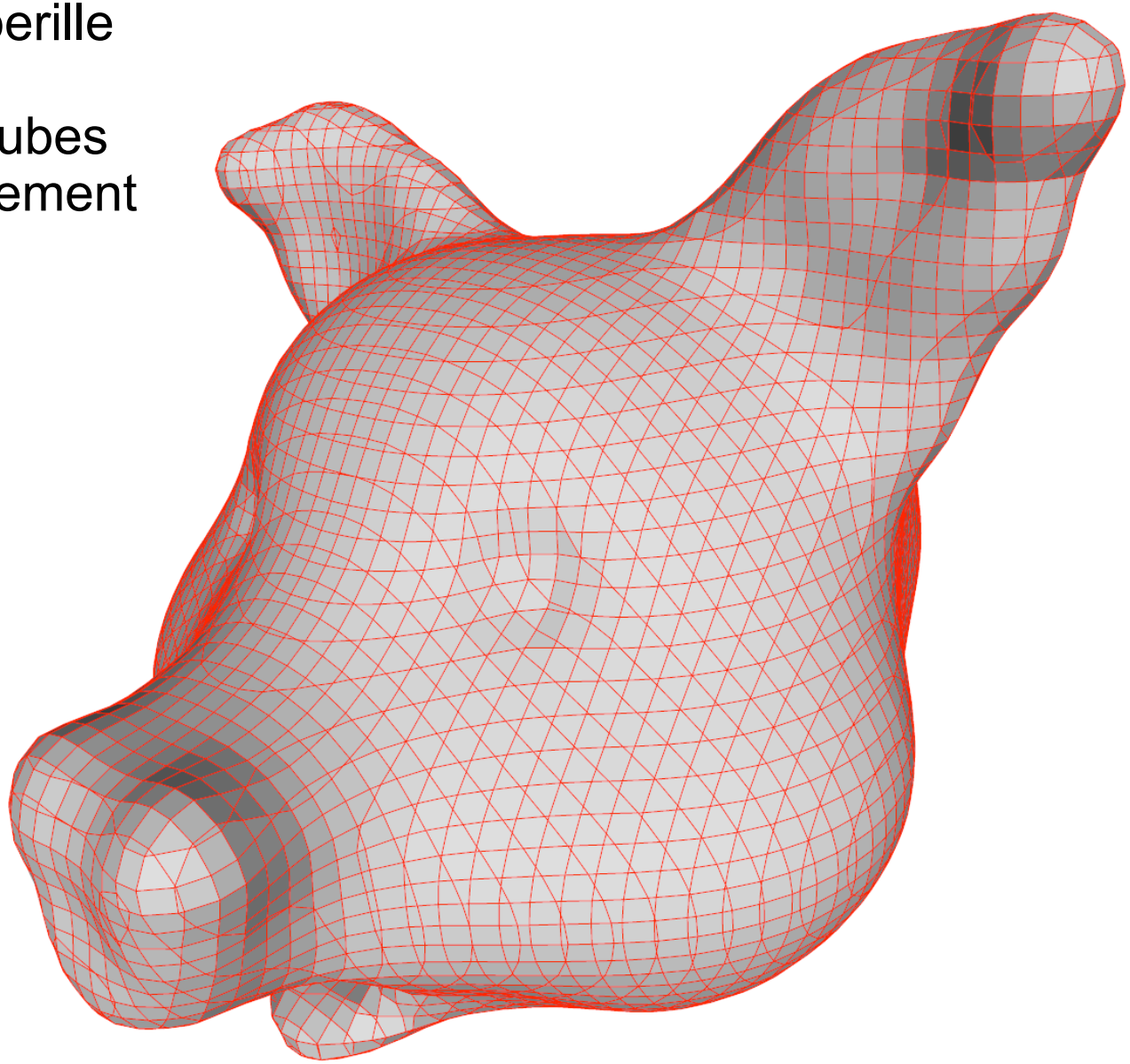
Dual Contouring
(Cuberille meshing)
vertices pushed onto
surface



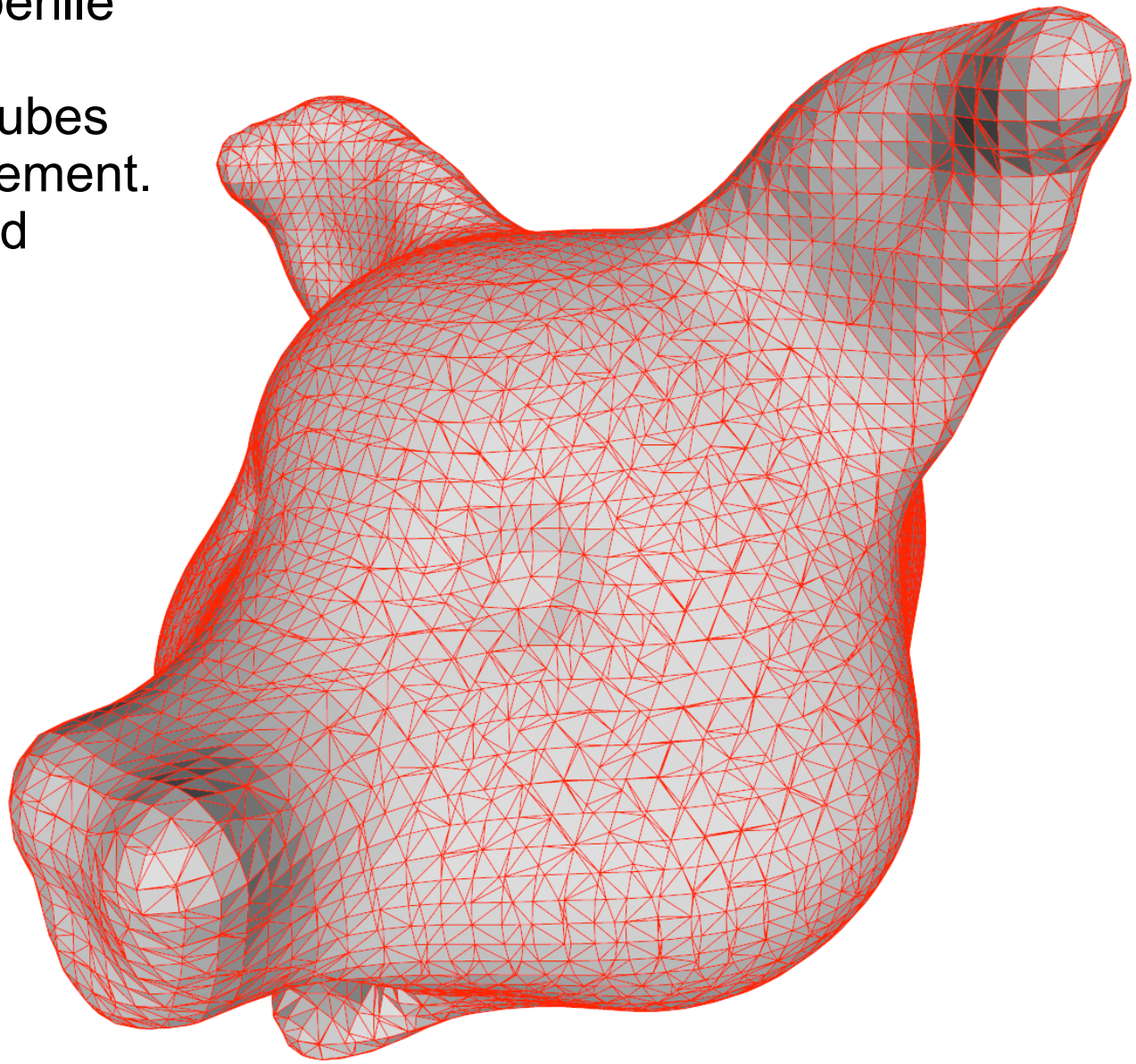
Dual Contouring
(Cuberille meshing)
vertices pushed
mesh triangulated



Dual of cuberille meshing.
Marching cubes vertex placement



Dual of cuberille meshing.
Marching cubes vertex placement.
Triangulated



Basis Function Summation

How do we reconstruct from a point cloud already?!

- Robust and easy:
  - Create a distance field by summing a basis function for each point
  - Run dual contouring to get a mesh
- Or more likely: use the widely available Poisson reconstruction code

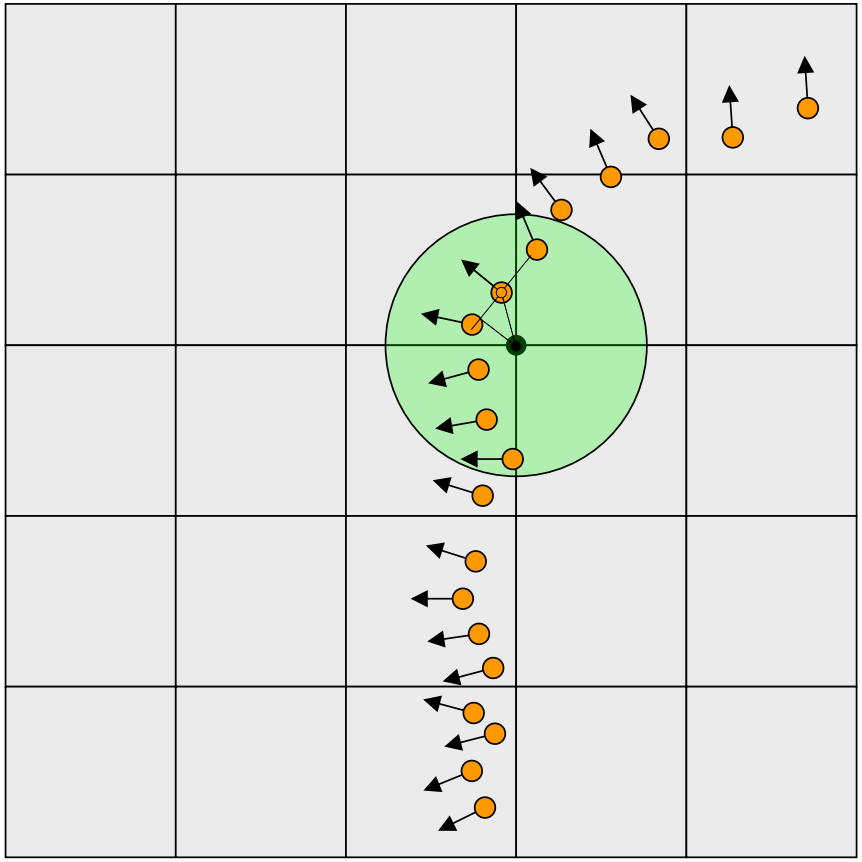
Basis Function Summation

- For each voxel, $\mathbf{v}$, locate points $\mathbf{p}_i$ with normal $\mathbf{n}_i$ within sphere of given radius
- Compute plane distance and square Euclidean distance for each point:

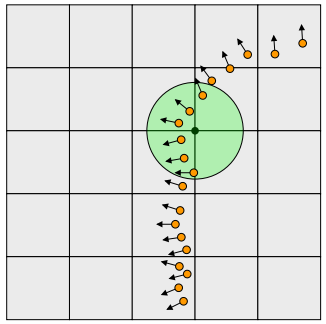
$$d_{\mathbf{p},\mathbf{n}}(\mathbf{x}) = \mathbf{n}^T(\mathbf{x} - \mathbf{p})$$
$$w_{\mathbf{p}}(\mathbf{x}) = \exp(-\alpha\|\mathbf{x} - \mathbf{p}\|^2)$$

- Compute sum of Gaussian weighted plane distances for each voxel:

$$\Phi(\mathbf{x}) = \frac{\sum_i w_{\mathbf{p}_i}(\mathbf{x})\, d_{\mathbf{p}_i,\mathbf{n}_i}(\mathbf{x})}{\sum_i w_{\mathbf{p}_i}(\mathbf{x})}$$

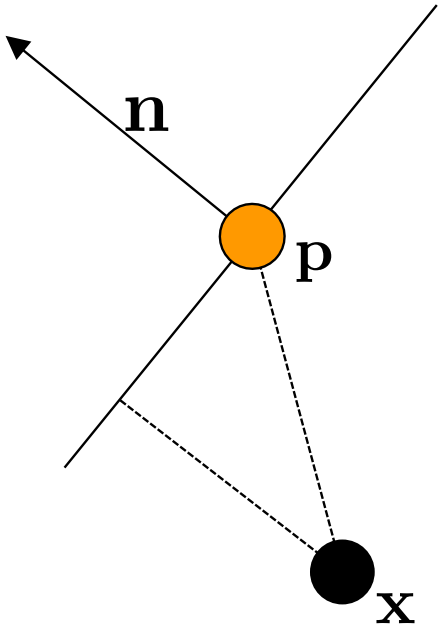


Basis Function Summation



- For each voxel, $\mathbf{v}$, locate points $\mathbf{p}_i$ with normal $\mathbf{n}_i$ within sphere of given radius
- Compute plane distance and square Euclidean distance for each point:

$$d_{\mathbf{p},\mathbf{n}}(\mathbf{x}) = \mathbf{n}^T(\mathbf{x} - \mathbf{p})$$
$$w_{\mathbf{p}}(\mathbf{x}) = \exp(-\alpha\|\mathbf{x} - \mathbf{p}\|^2)$$



- Compute sum of Gaussian weighted plane distances for each voxel:

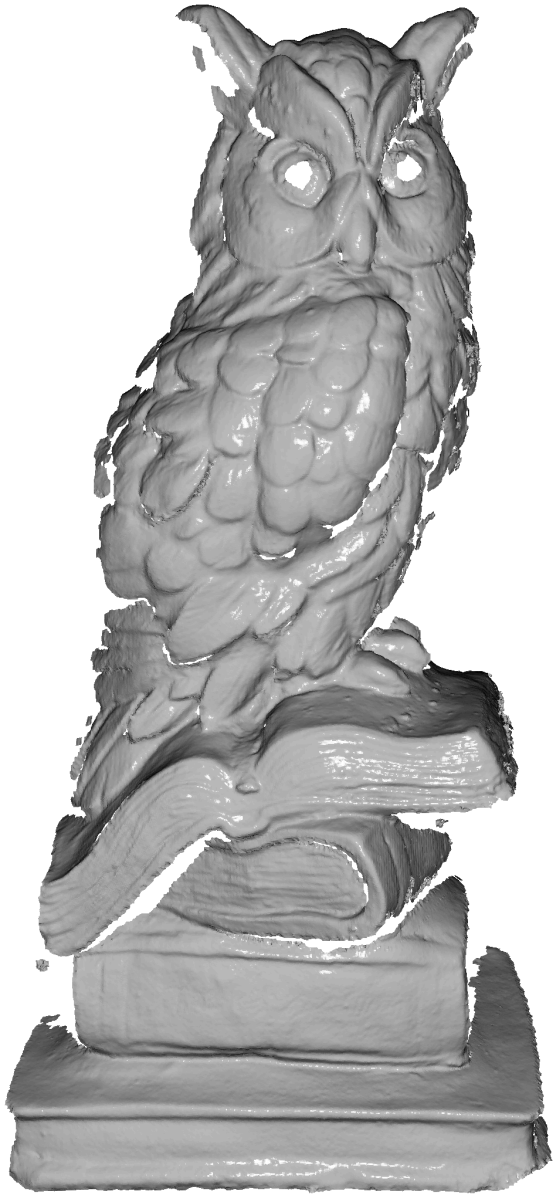
$$\Phi(\mathbf{x}) = \frac{\sum_i w_{\mathbf{p}_i}(\mathbf{x})\, d_{\mathbf{p}_i,\mathbf{n}_i}(\mathbf{x})}{\sum_i w_{\mathbf{p}_i}(\mathbf{x})}$$

BFS: Implementation

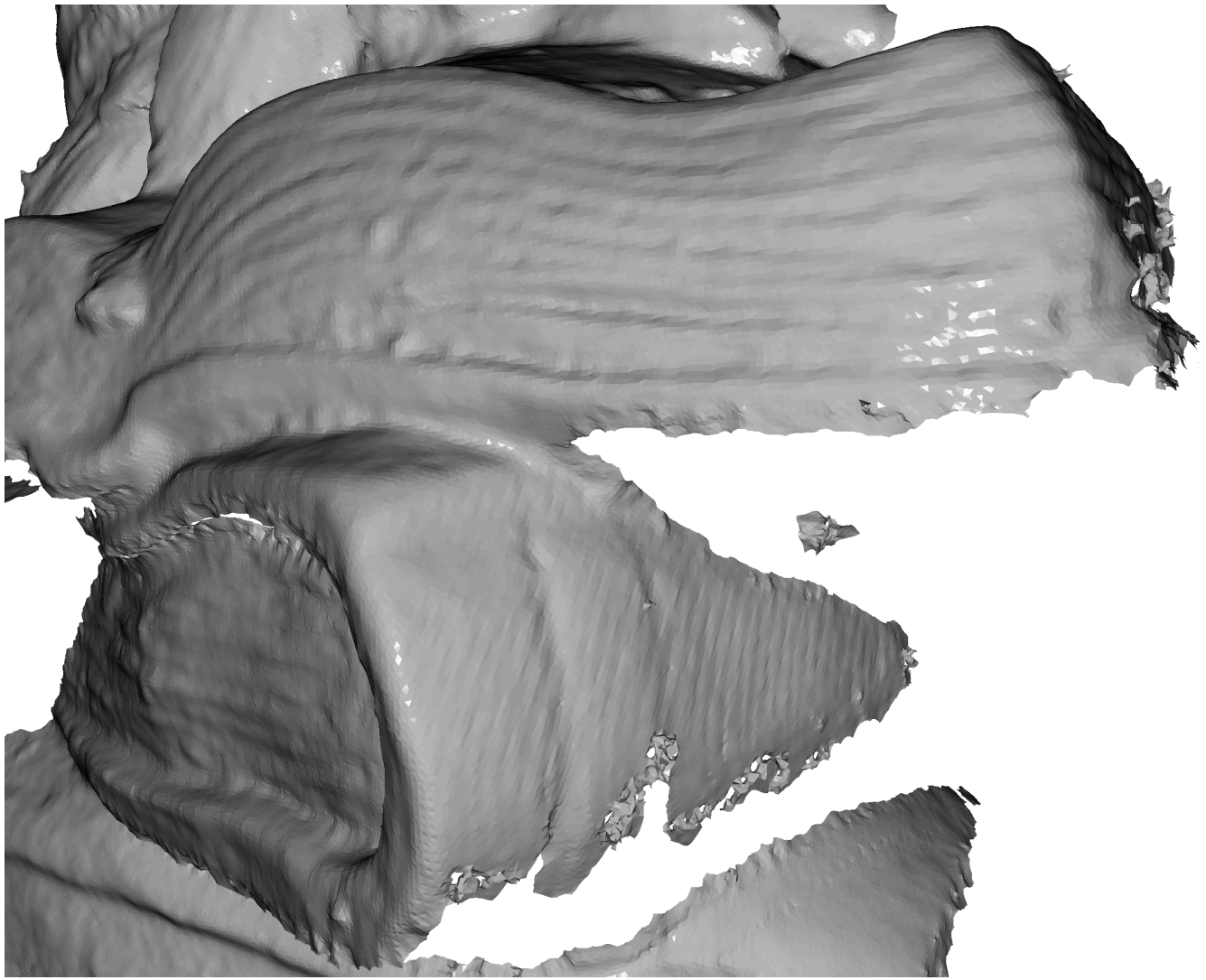
- We could use a kD-tree to find the points closest to each voxel, but it is too slow
- Instead, loop over a region of the volume close to each point.
  - For each voxel in that region add the weighted distance to one volume and the weight to another volume
- In a final pass, divide each voxel in the volume containing the distance sums by the corresponding voxel in the weight sum volume.

Screened Poisson vs BFS

Kazhdan, Michael, and Hugues Hoppe. "Screened poisson surface reconstruction." ACM Transactions on Graphics (ToG) 32.3 (2013)

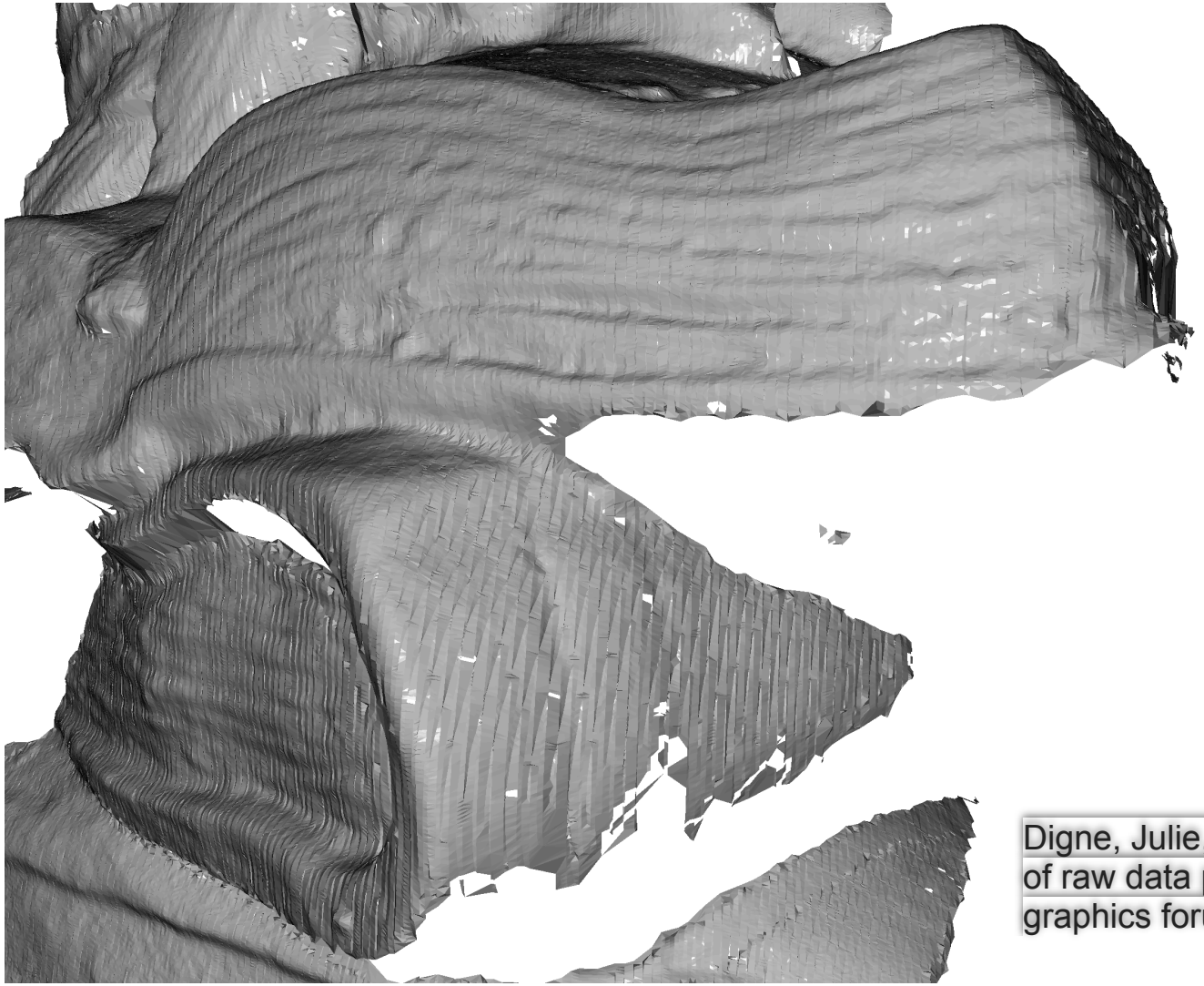


Screened Poisson vs BFS

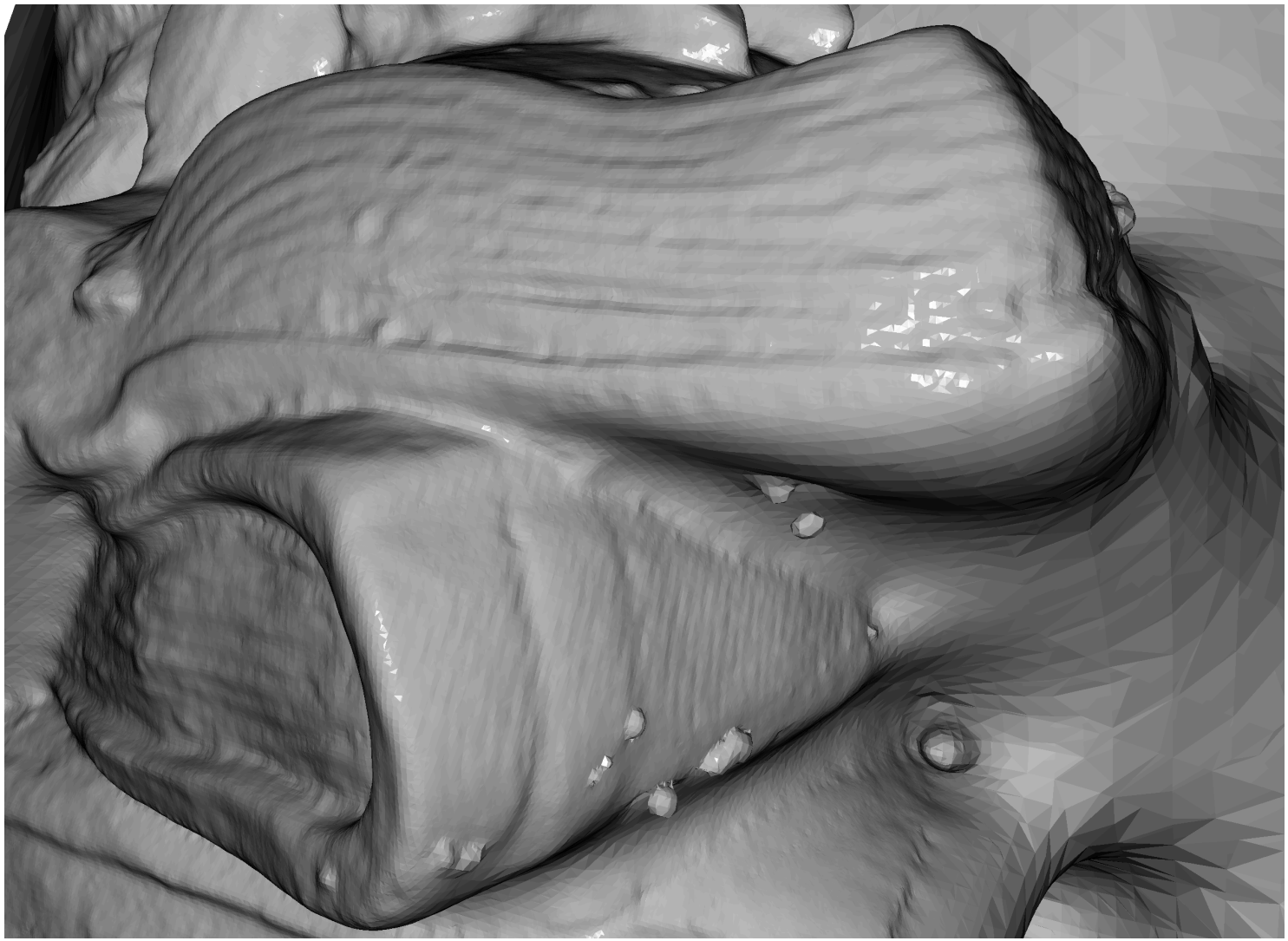


BFS

Combinatorial



Digne, Julie, et al. "Scale space meshing of raw data point sets." Computer graphics forum. Vol. 30. No. 6.



Poisson

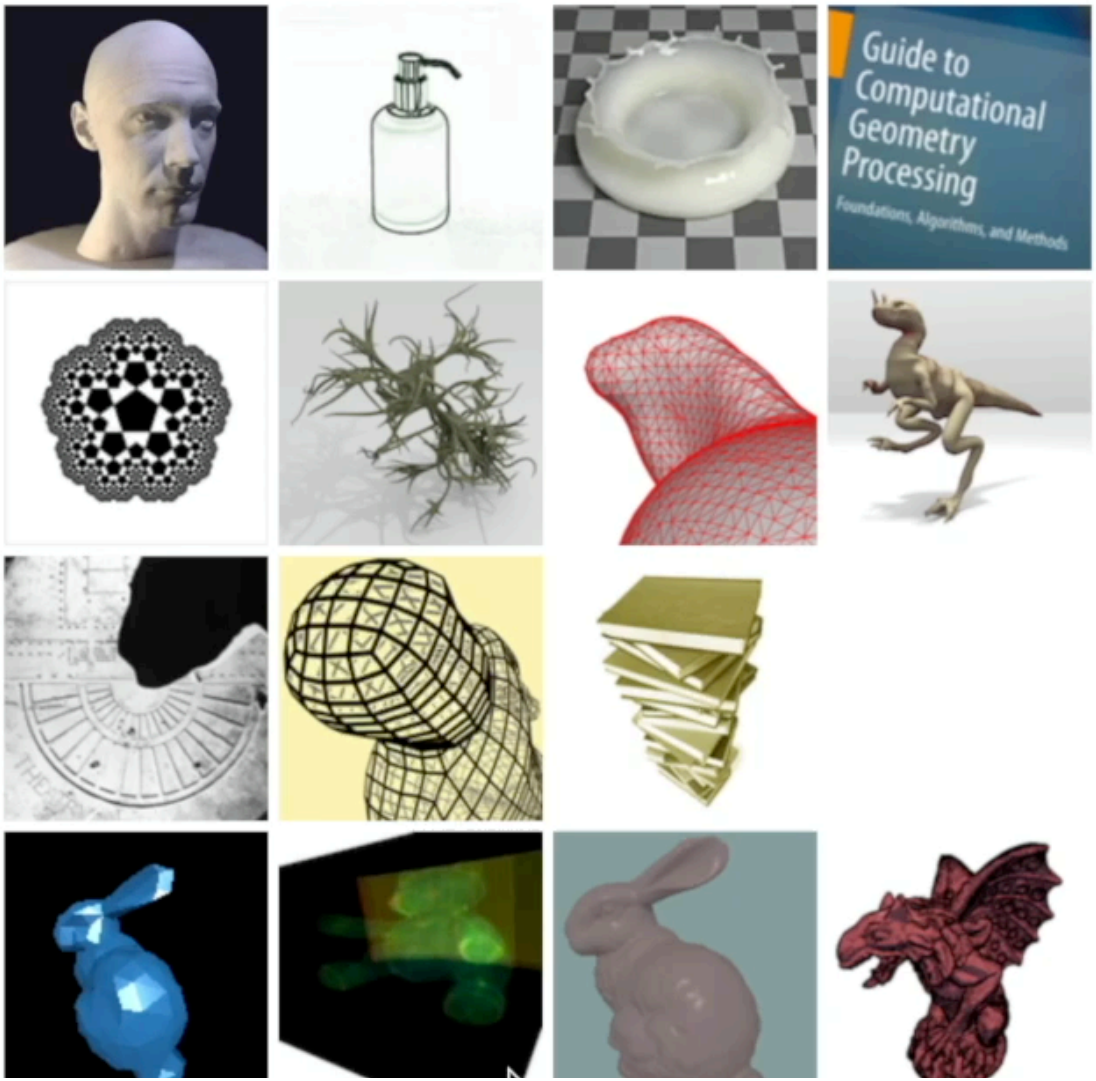
GEL and PyGEL

https://github.com/janba/GEL
https://pypi.org/project/PyGEL3D/
http://www2.compute.dtu.dk/projects/GEL/PyGEL

- GEL is a C++ library of **geometry processing tools** including (but not limited to)
  - a half-edge based polygonal mesh,
  - a graph data structure, and
  - various spatial data structures
- PyGEL
  - a set of Python bindings for core features in GEL
  - has its own viewer based on OpenGL
  - PyGEL can be used from Jupyter notebooks (Also Google Colab)

Not Secure — www2.compute.dtu.dk

VoxelRay Volume Rendering Program



Thanks

Do you have any questions?

Acknowledgements

Thanks to collaborators!

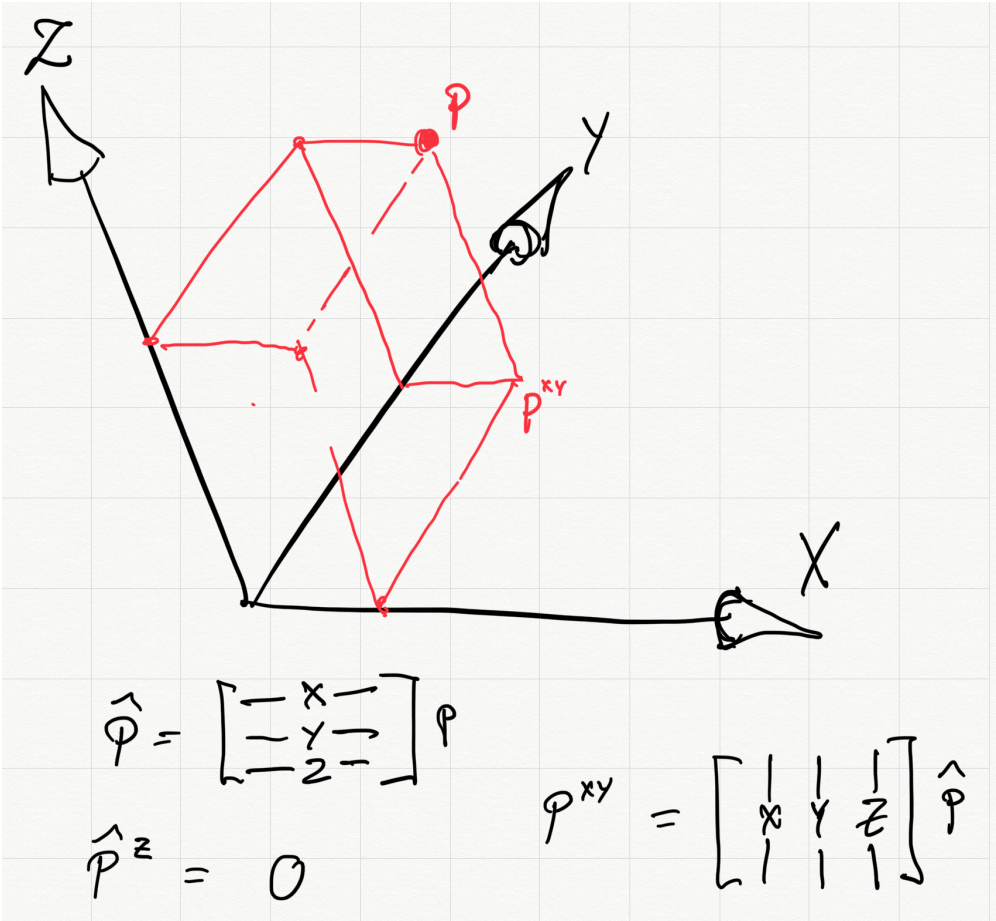
Katarzyna Welnicka, Henrik Aanæs, Rasmus Larsen, Eva Rotenberg, Jens Gravesen,

BONUS

Material or junk

3D Example

- Given a vector $\mathbf{p}$ in 3D, we can multiply $\mathbf{p}$ onto basis $[\mathbf{XYZ}]^T$
- From the vector in this basis, we can get back $\mathbf{p}$ by multiplying onto $[\mathbf{XYZ}]$
- If we set some coefficients to zero, we project $\mathbf{p}$ onto a space of lower dimension



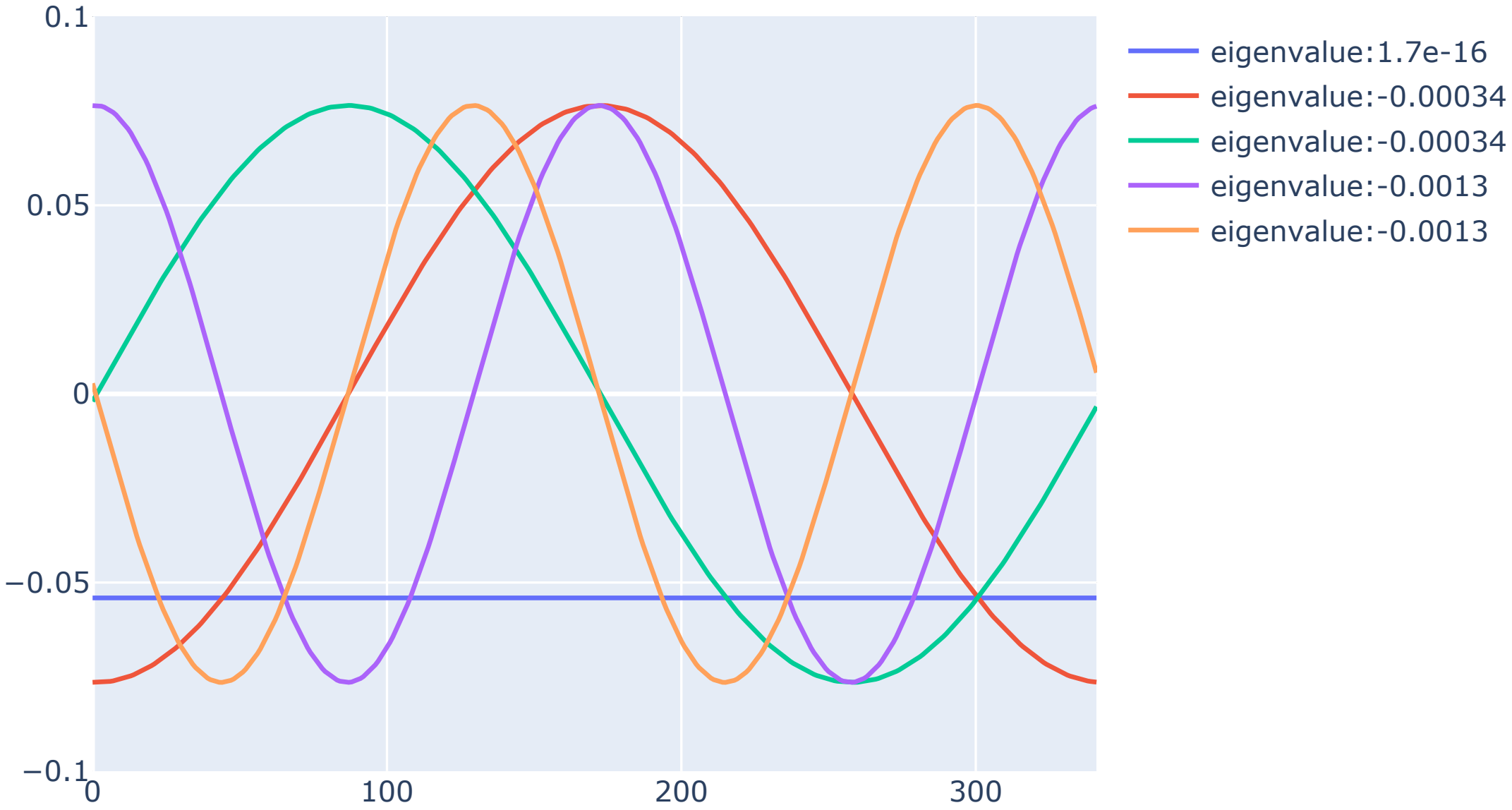
Eigensolutions 1D Laplacian

- We are looking for solutions to $\mathbf{L}\mathbf{e}_i = \lambda_i \mathbf{e}_i$

Hint, use:
`scipy.linalg.eigh`

- where

$$L_{ij} = \begin{cases} -2 & j = i \\ 1 & j = i \pm 1 \\ 0 & \text{otherwise} \end{cases}$$


0.1
0.05
0
−0.05
−0.1
0
100
200
300
eigenvalue:1.7e-16
eigenvalue:-0.00034
eigenvalue:-0.00034
eigenvalue:-0.0013
eigenvalue:-0.0013

Spectral Smoothing

- We can now project our signal onto the basis of eigenvectors (*analysis*)

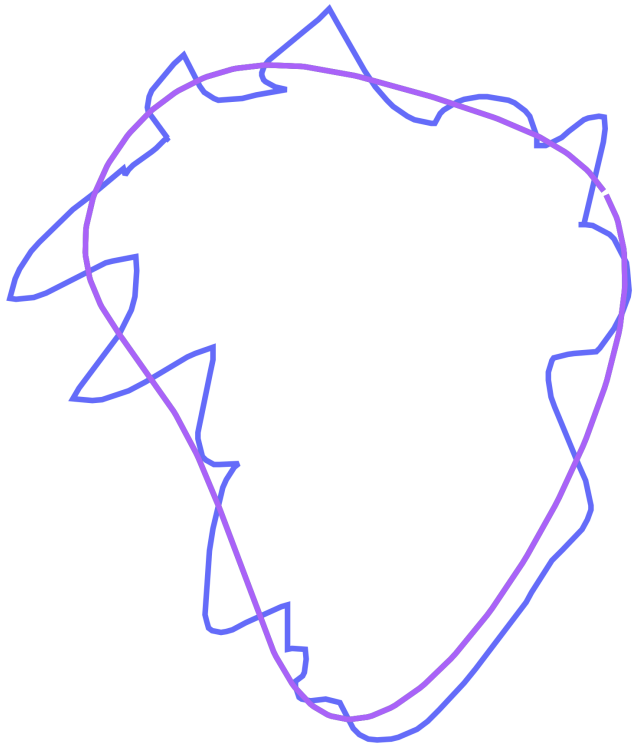
$$\hat{\mathbf{p}} = \mathbf{e}^T\mathbf{p}$$

- and *reconstruct* simply by

$$\mathbf{p} = \mathbf{e}\hat{\mathbf{p}}$$

- Note that $\mathbf{p}$ is actually a matrix of dimension Nx2, so we treat the vertices in parallel

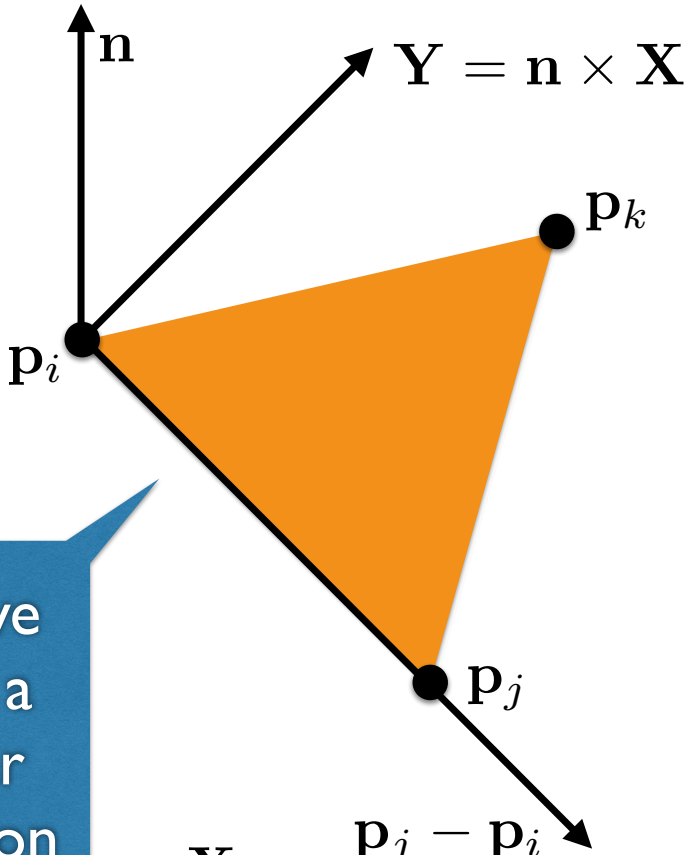
- Entries of $\hat{\mathbf{p}}$ can be set to zero.
  This corresponds to removing frequencies.



Head (blue) and reconstruction using seven eigenvectors (frequencies) (purple)

LSCM

Closely following
Polygonal Mesh Processing by
Botsch et al. CRC Press, 2010



Say we have a linear function

$$\mathbf{X} = \frac{\mathbf{p}_j - \mathbf{p}_i}{\|\mathbf{p}_j - \mathbf{p}_i\|}$$

$$\nabla u = \frac{1}{2A} \begin{bmatrix} y_j - y_k & y_k - y_i & y_i - y_j \\ x_k - x_j & x_i - x_k & x_j - x_i \end{bmatrix} \begin{bmatrix} u_i \\ u_j \\ u_k \end{bmatrix}$$

$$= \mathbf{M} \begin{bmatrix} u_i \\ u_j \\ u_k \end{bmatrix}$$

Now, we need to solve

$$\nabla u - \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \nabla v = 0$$

$$\mathbf{M} \begin{bmatrix} u_i \\ u_j \\ u_k \end{bmatrix} - \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \mathbf{M} \begin{bmatrix} v_i \\ v_j \\ v_k \end{bmatrix} = 0$$