

# Geometric Modeling

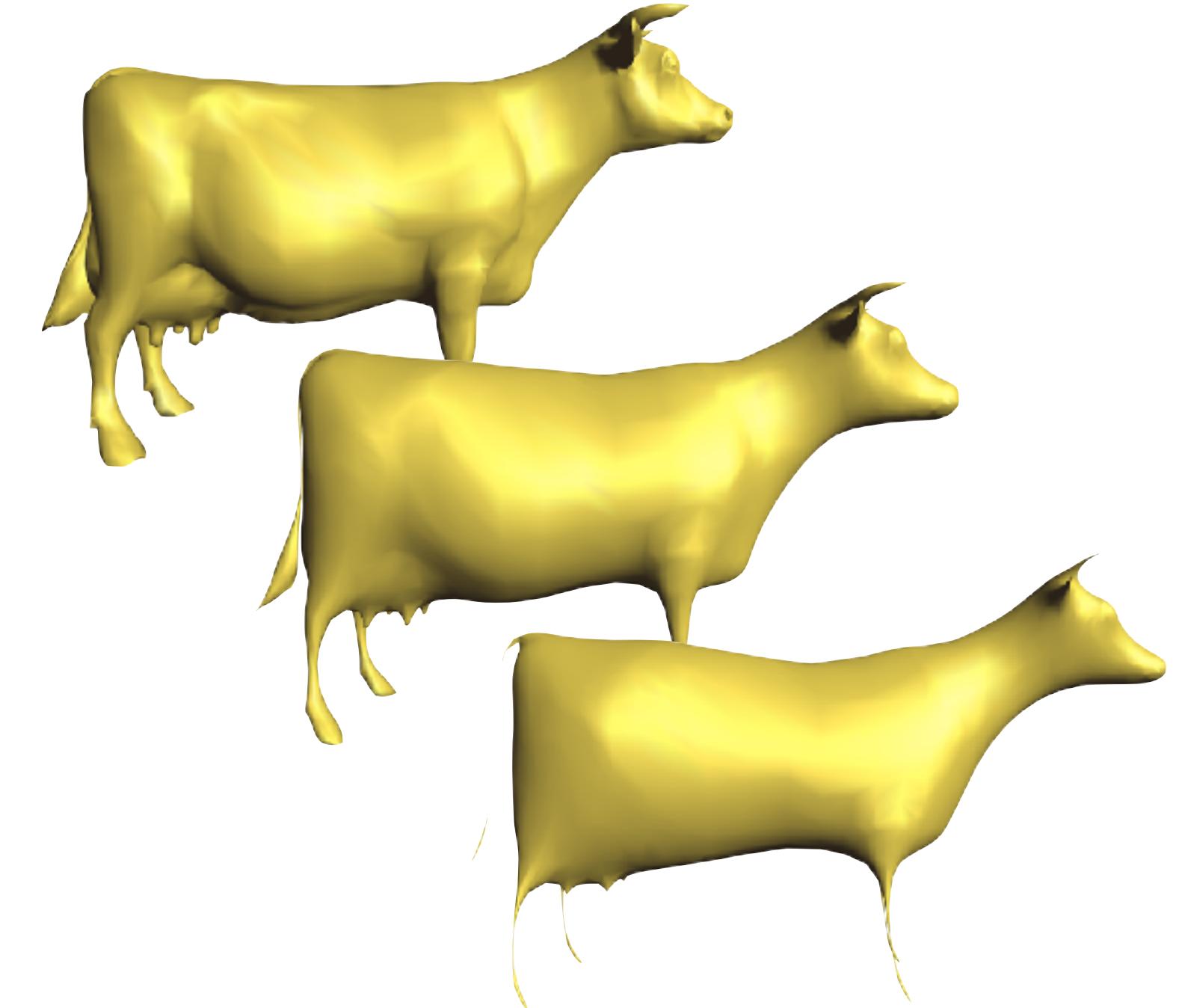
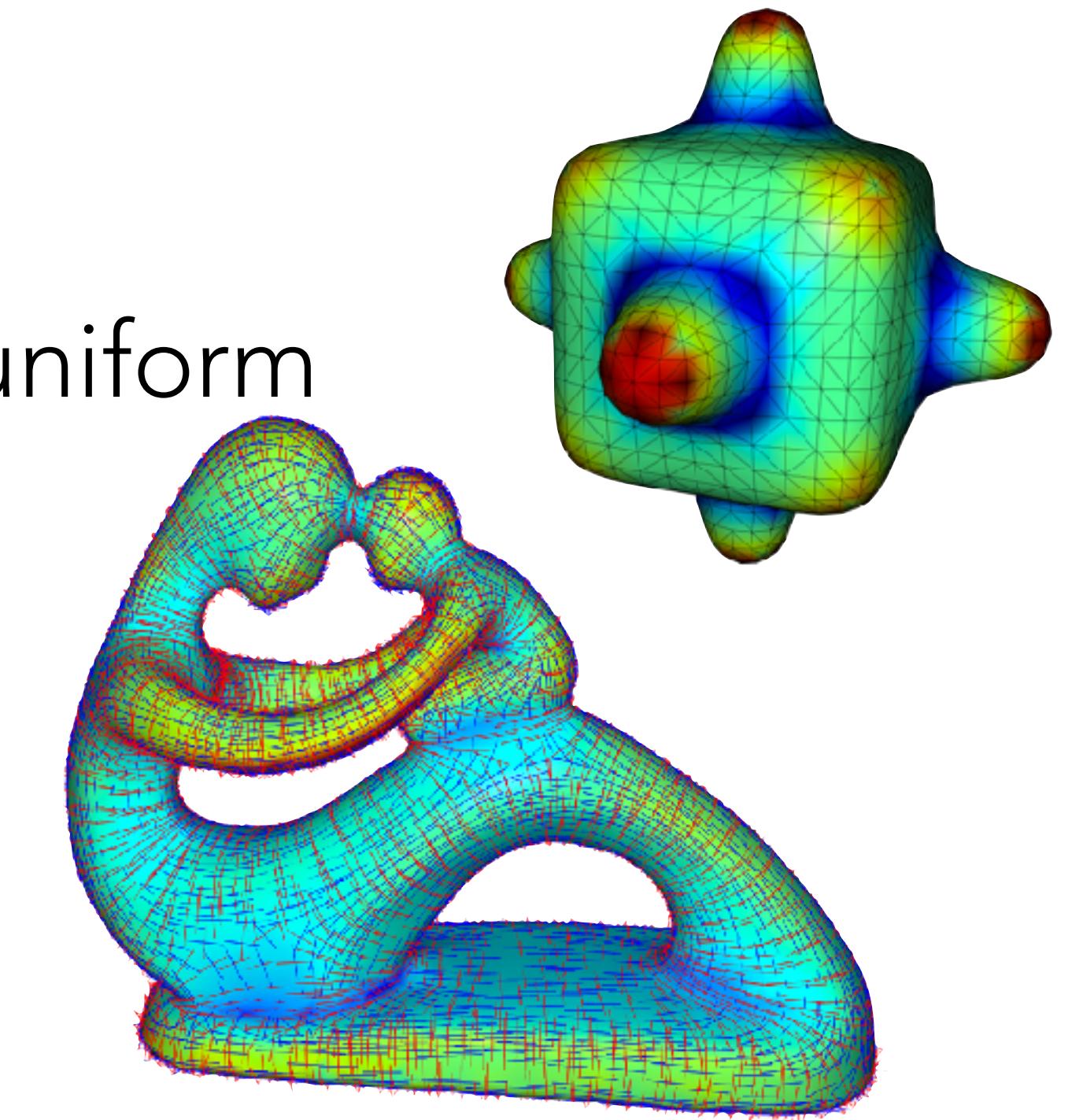
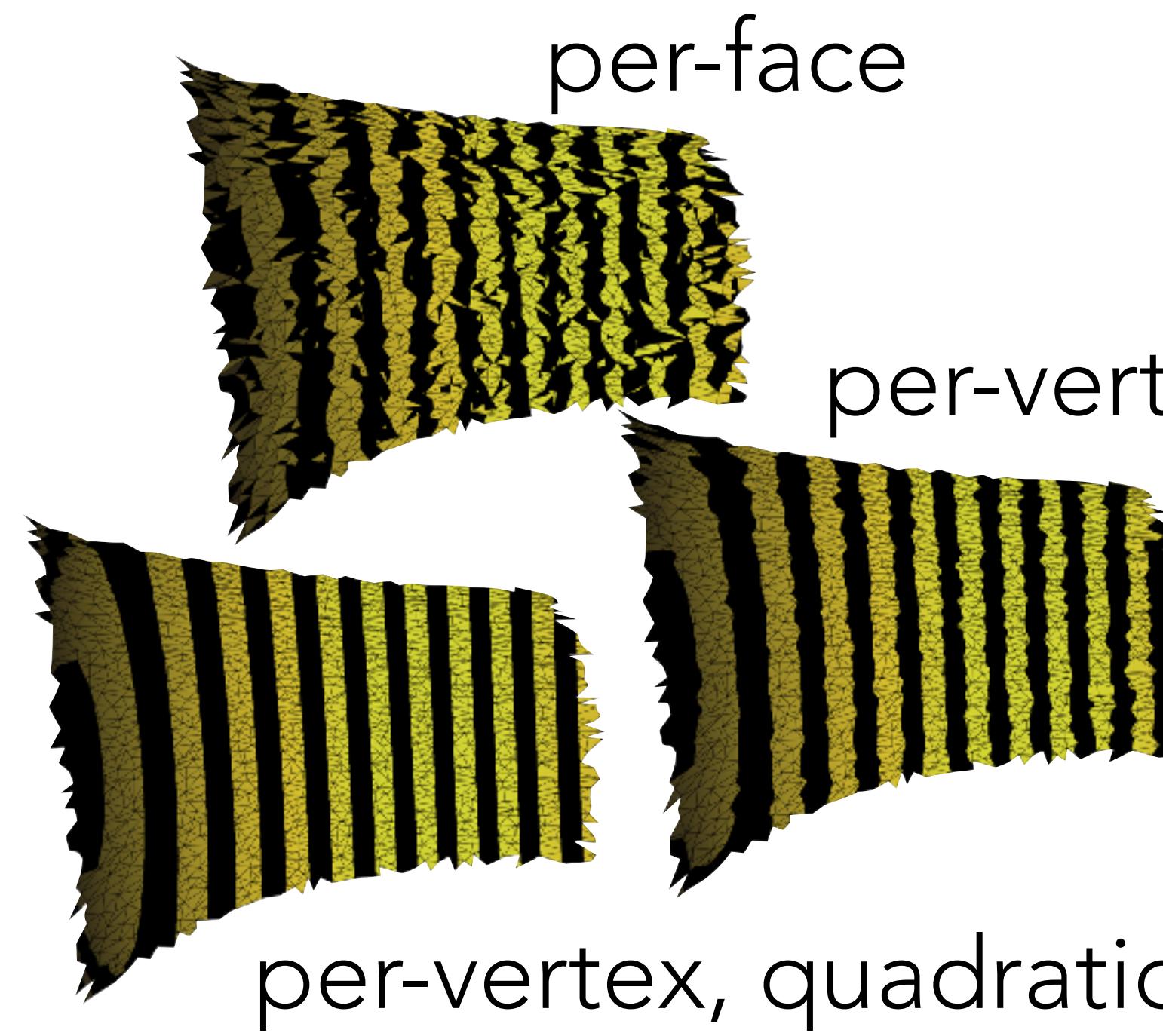
## Assignment 3: Discrete Differential Quantities

Acknowledgements: Julian Panetta, Olga Diamanti

CSC 486B/586B - Geometric Modeling - Teseo Schneider

# Assignment 3

- Topic: Discrete Differential Quantities with libigl
  - Vertex Normals, Curvature, Smoothing (Tutorials 201-205)



# Vertex Normals



uniform average



area-weighted average



mean-curvature based

- In this exercise: experiment with different vertex normals



PCA on k-nearest neighbors

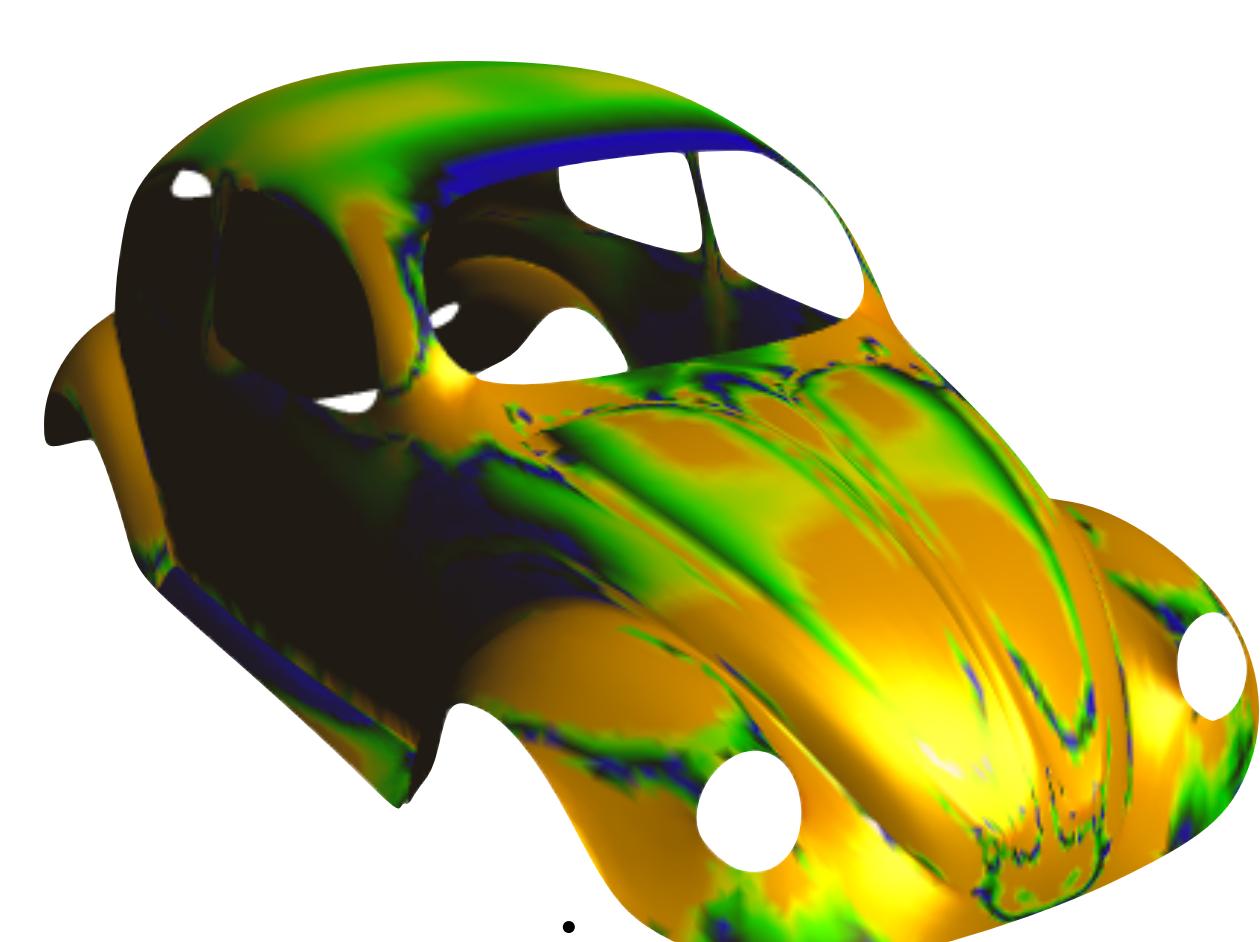


quadratic fitting on k-nearest neighbors

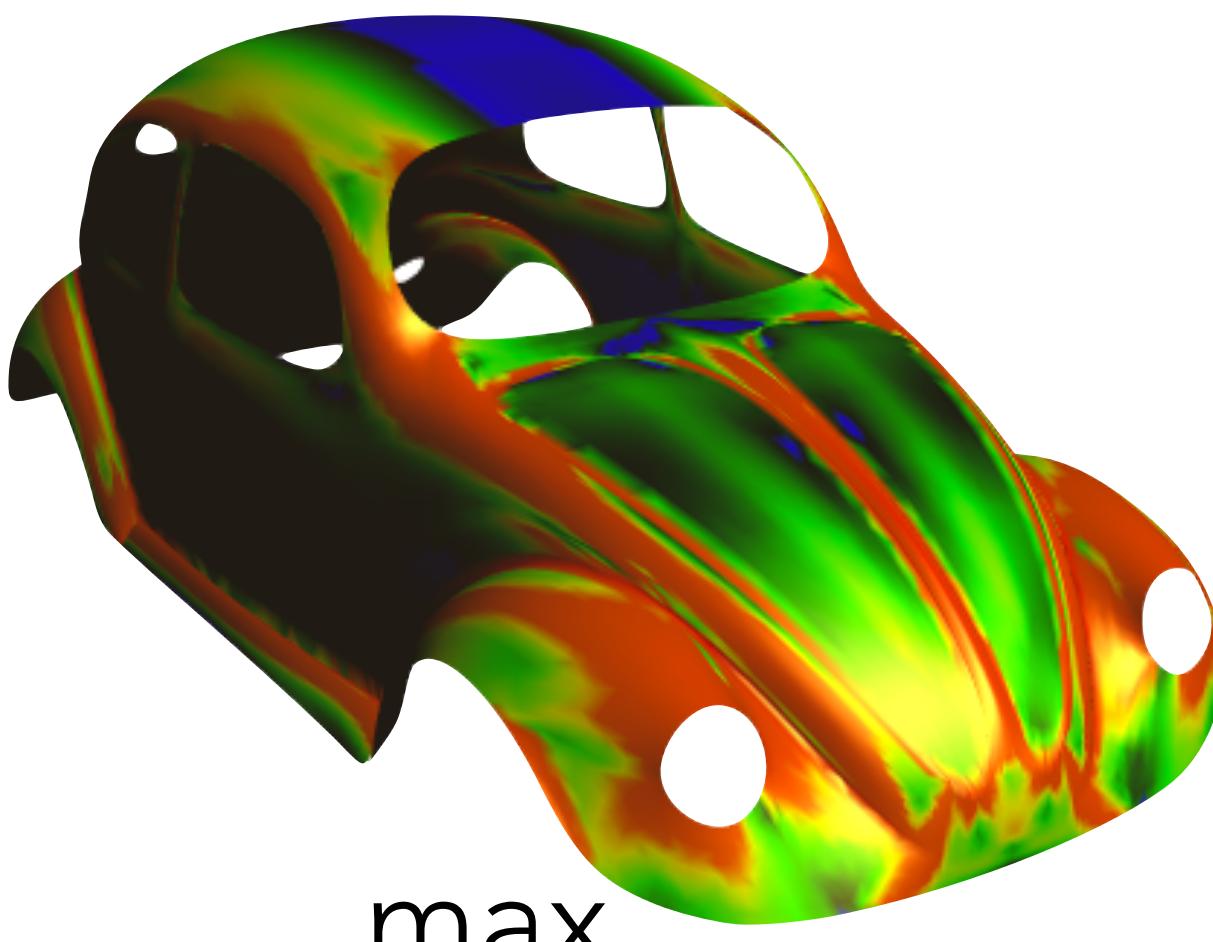
`igl.principal_curvature`

# Curvature

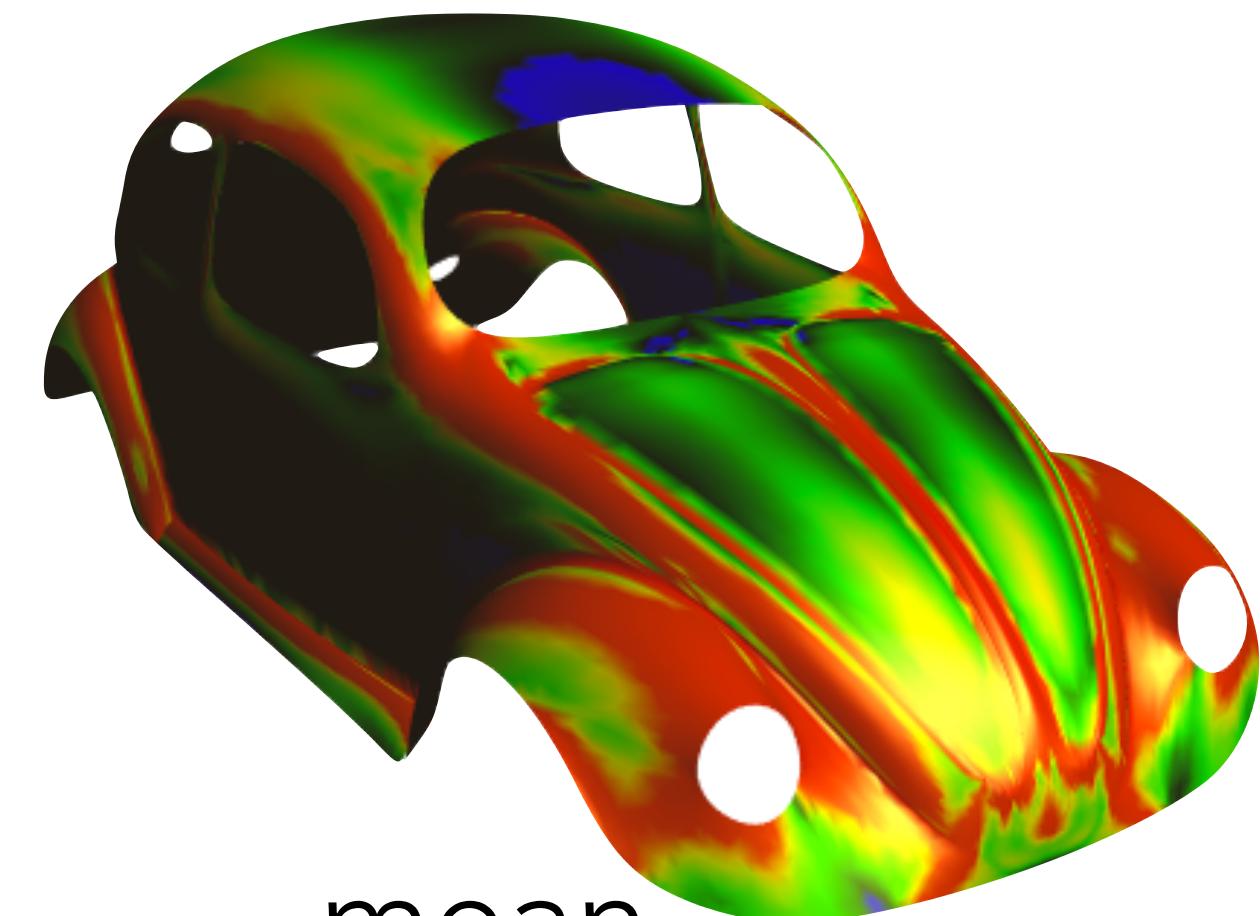
libigl tutorials  
Chapter 1



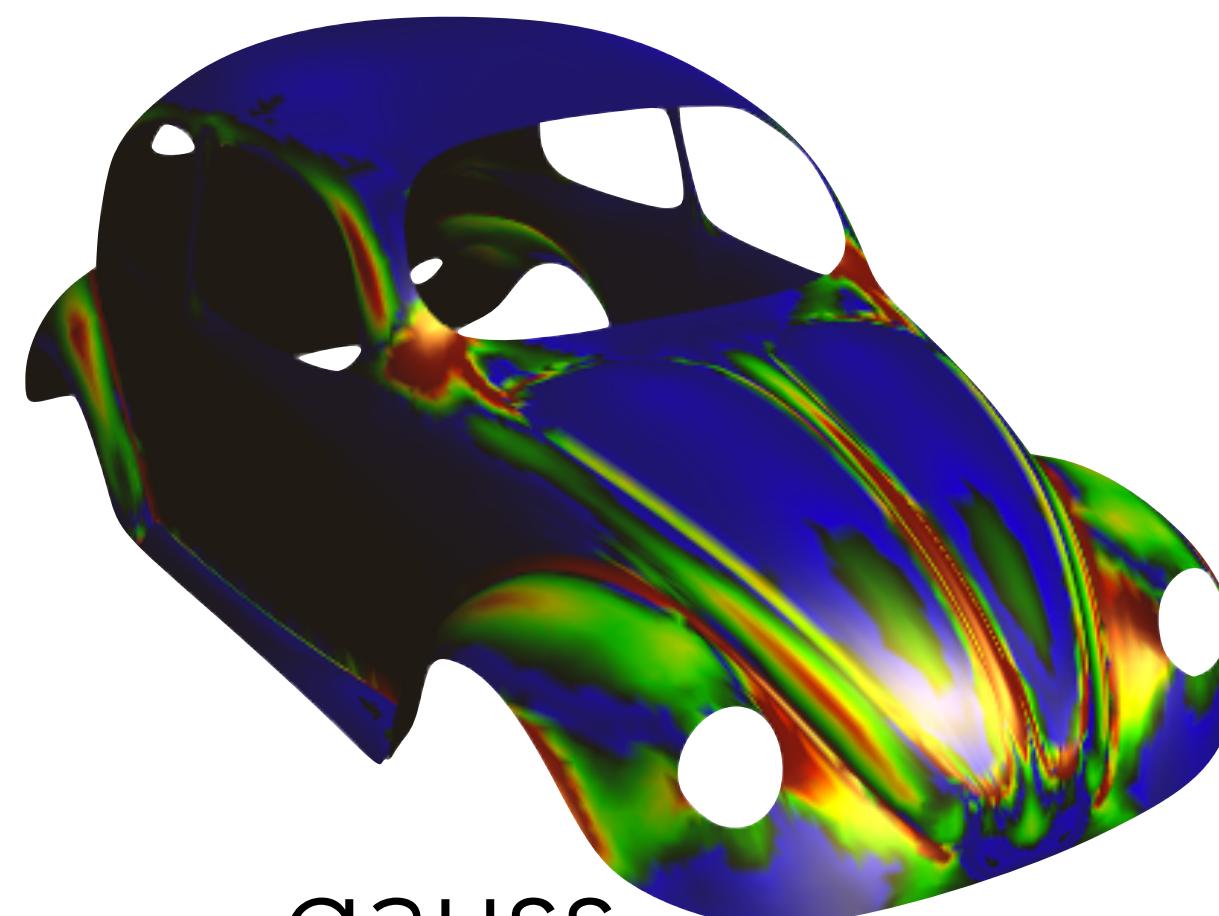
min



max



mean



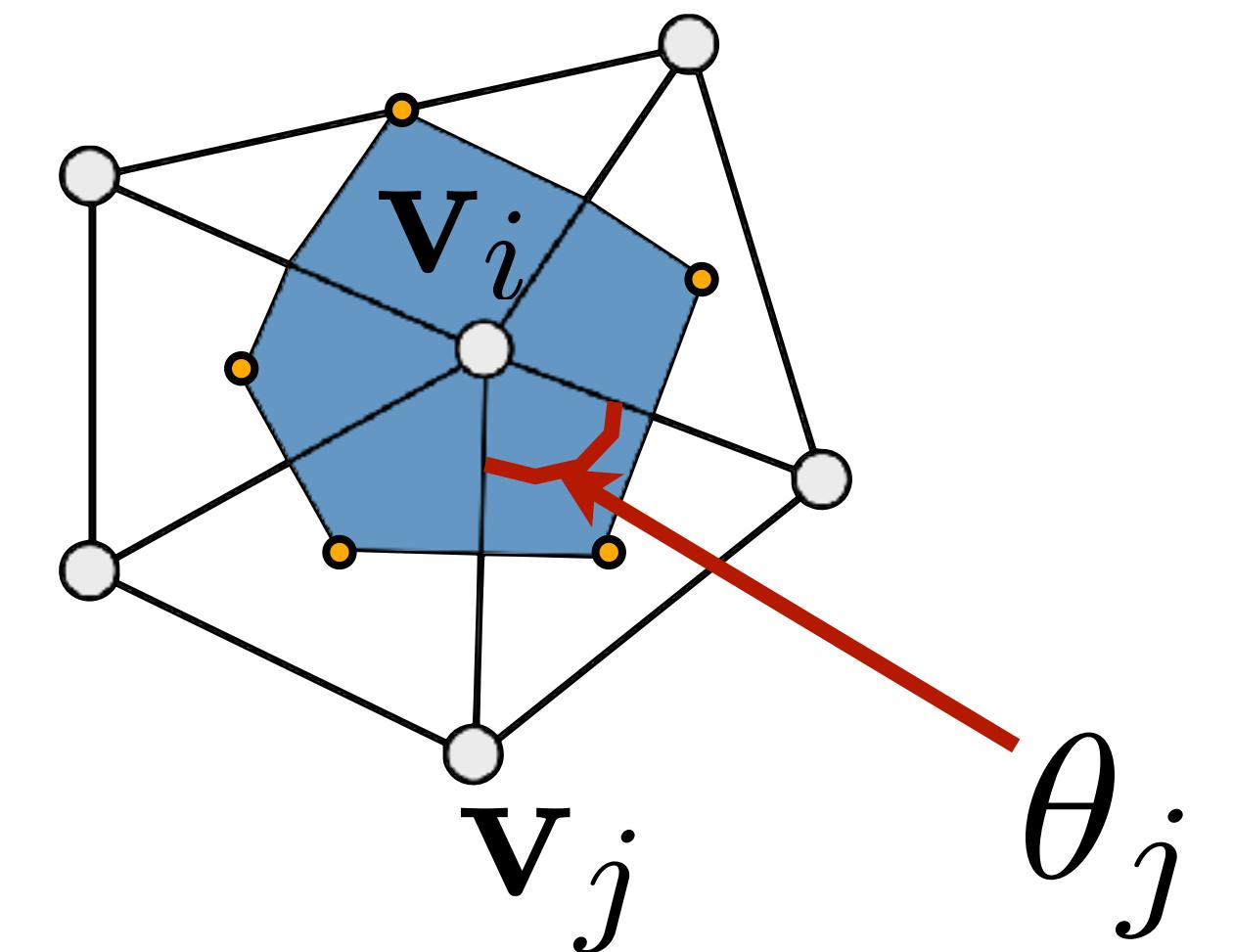
gauss

# Curvature

$$\Delta_S \mathbf{p} = -2H \mathbf{n}$$

- Mean Curvature  $H(\mathbf{v}_i) = 0.5 \|L_c(\mathbf{v}_i)\|$

- Gaussian Curvature  $G(\mathbf{v}_i) = \frac{2\pi - \sum_j \theta_j}{A(\mathbf{v}_i)}$



- Min Curvature  $\kappa_1 = H + \sqrt{H^2 - G}$

- Max Curvature  $\kappa_1 = H - \sqrt{H^2 - G}$



$$G = \kappa_1 \kappa_2$$

$$2H = \kappa_1 + \kappa_2$$

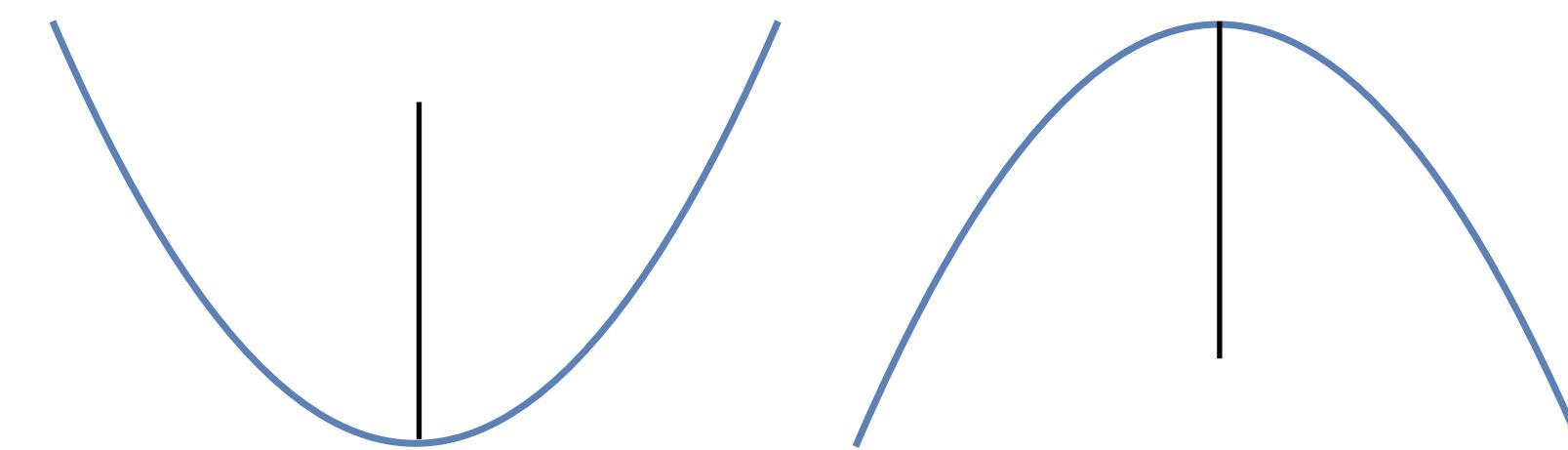


# Data Smoothing: Grid Case

- Image/2D data smoothing:  
Filter out noise/rapid oscillations
- Solve the heat equation over some time period:

$$\frac{\partial f}{\partial t} = \lambda \Delta f = \lambda \left( \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \right)$$

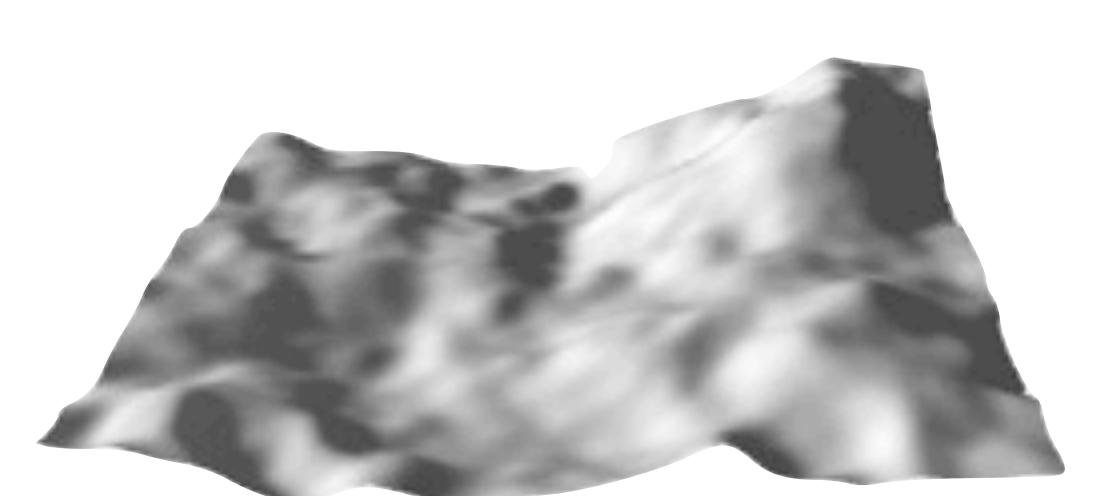
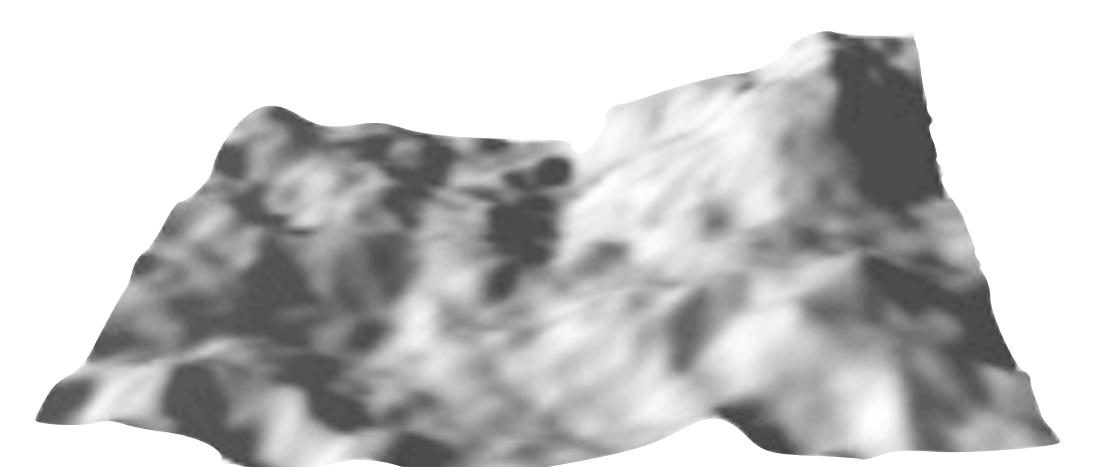
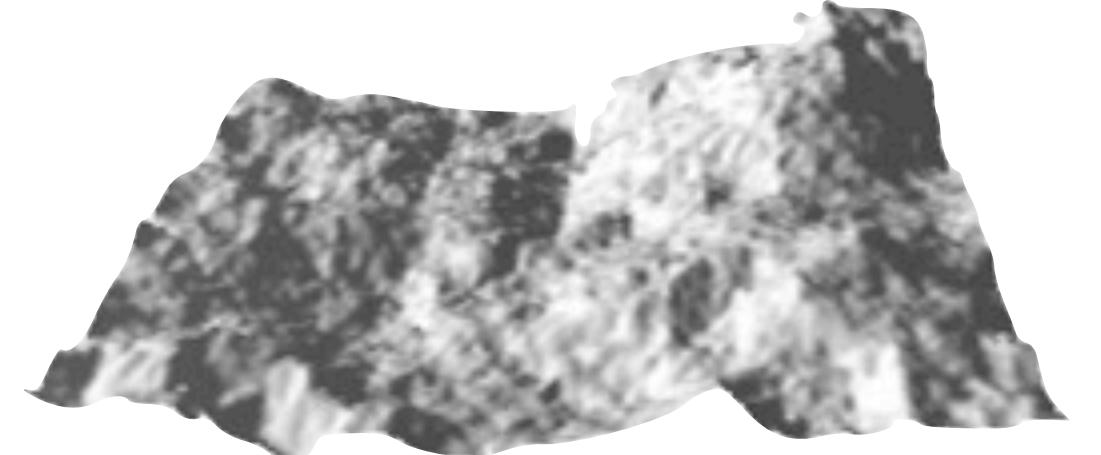
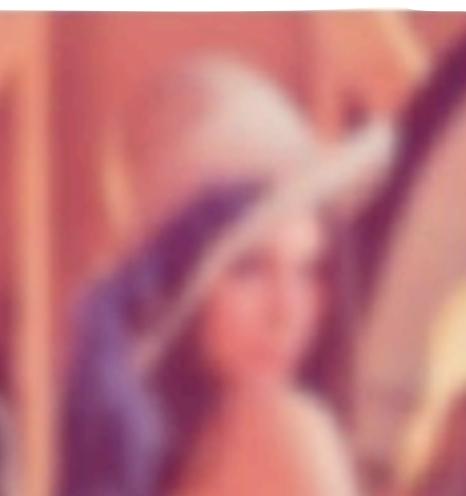
- Intuition: move toward the average of neighboring values



# Data Smoothing: Grid Case

$$\frac{\partial f}{\partial t} = \lambda \Delta f = \lambda \left( \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \right)$$

- Approach: discretize laplacian with **finite differences**
- Now we have an ordinary differential equation (ODE)
- Integrate (time-step) the ODE, e.g., with forward/backwards Euler



# Data Smoothing: Surface Case

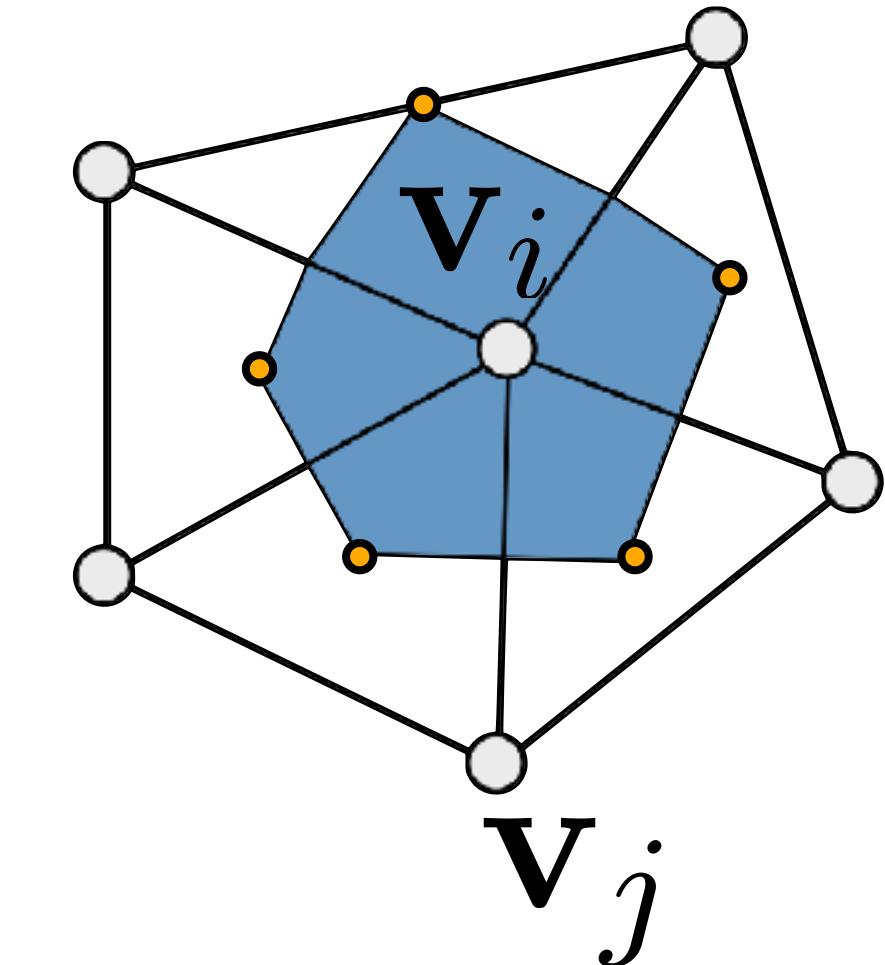
- Laplace-Beltrami operator lets us smooth functions on surfaces,  $S$

$$\frac{\partial f}{\partial t} = \lambda \Delta_S f$$

- E.g., with forward Euler (explicit smoothing) on a triangle mesh,  $M$ :

$$\begin{aligned} f^{t+\Delta t}(\mathbf{v}_i) &= f^t(\mathbf{v}_i) + \lambda \Delta t \Delta_M f^t(\mathbf{v}_i) \\ &= f^t(\mathbf{v}_i) + \lambda \Delta t \left( \sum_{\mathbf{v}_j \in N(\mathbf{v}_i) \cup \{\mathbf{v}_i\}} w_{ij} f^t(\mathbf{v}_j) \right) \end{aligned}$$

vertices adjacent vi      discrete Laplacian weights



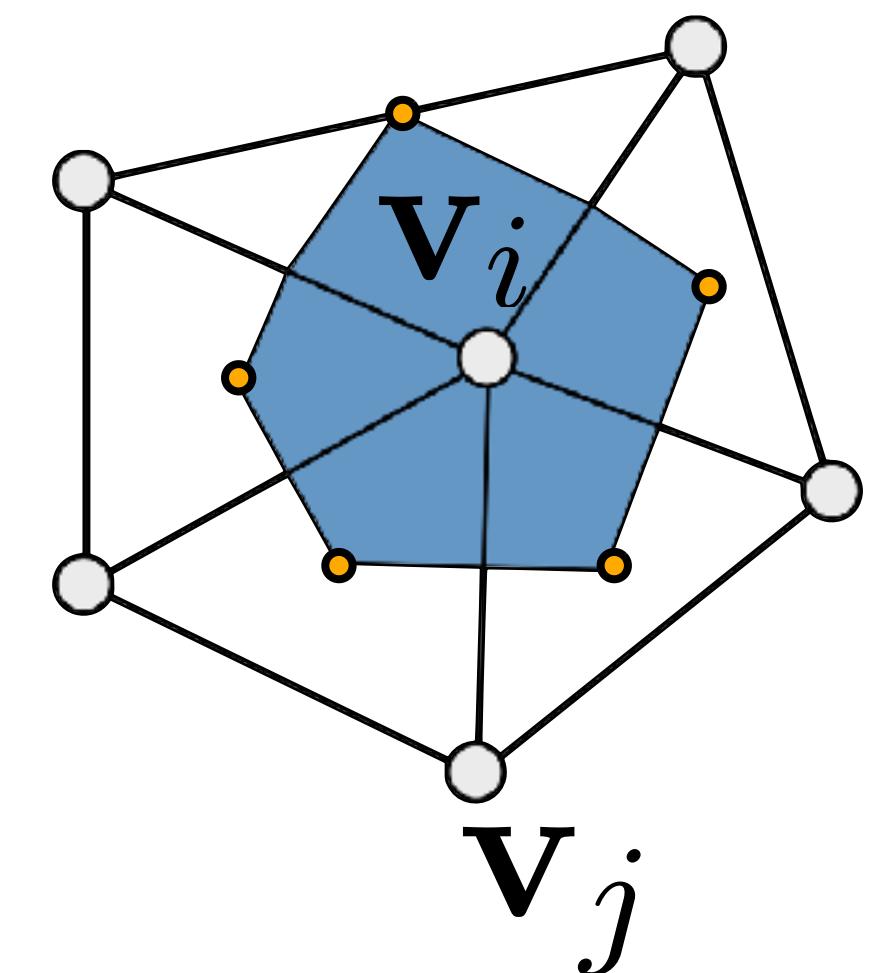
# Mesh Smoothing

- We can view the vertex positions as a (vector-valued) function to smooth!

$$f(\mathbf{v}_i) = \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix}$$

- Here, first-order explicit time stepping looks like:

$$\begin{aligned} f^{t+\Delta t}(\mathbf{v}_i) &= f^t(\mathbf{v}_i) + \lambda \Delta t \Delta_M f^t(\mathbf{v}_i) \implies \\ \mathbf{v}_i^{\text{new}} &= \mathbf{v}_i + \lambda \Delta t (\Delta_M \mathbf{v})_i \\ &= \mathbf{v}_i + \lambda \Delta t \left( \sum_{\mathbf{v}_j \in N(\mathbf{v}_i) \cup \{\mathbf{v}_i\}} w_{ij} \mathbf{v}_j \right) \end{aligned}$$



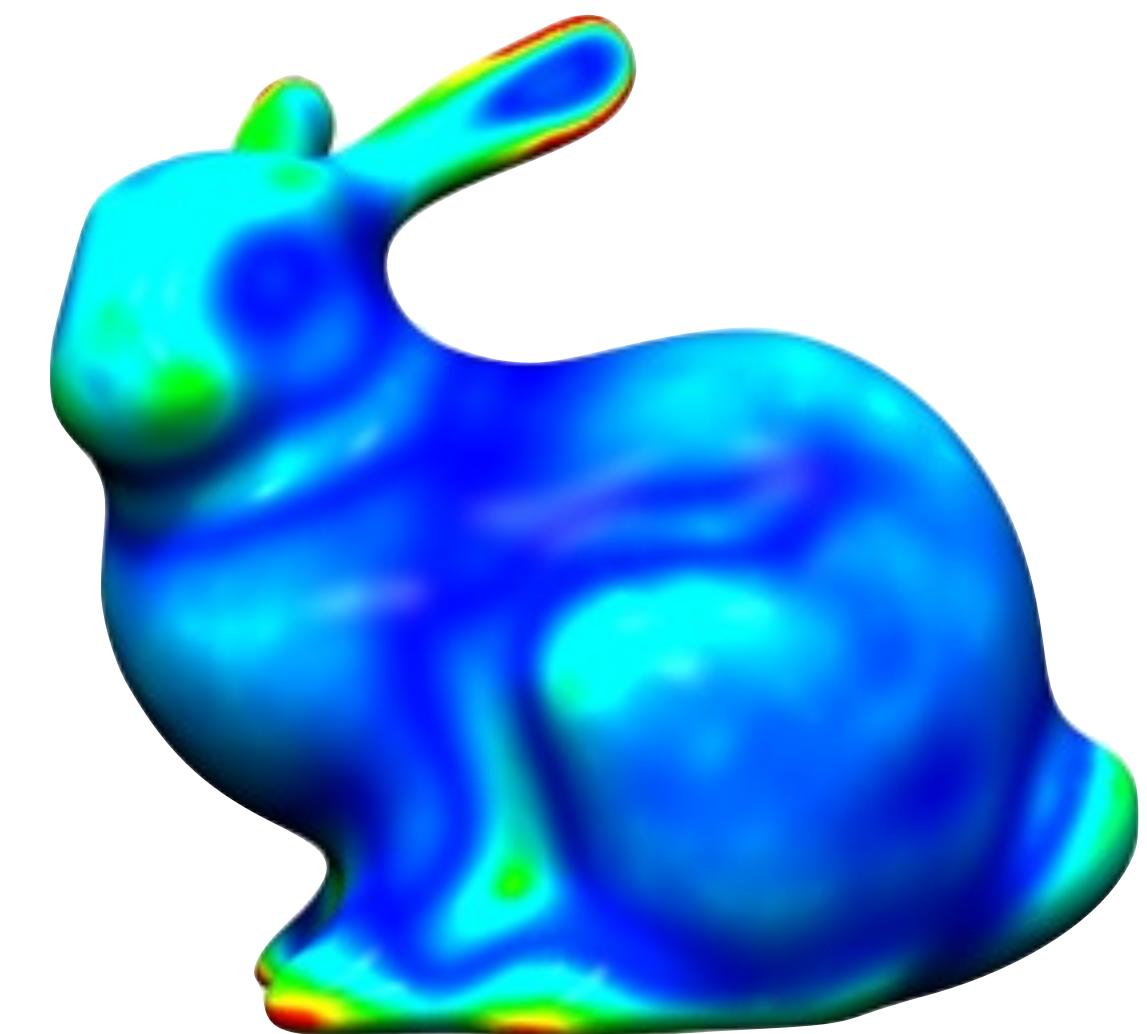
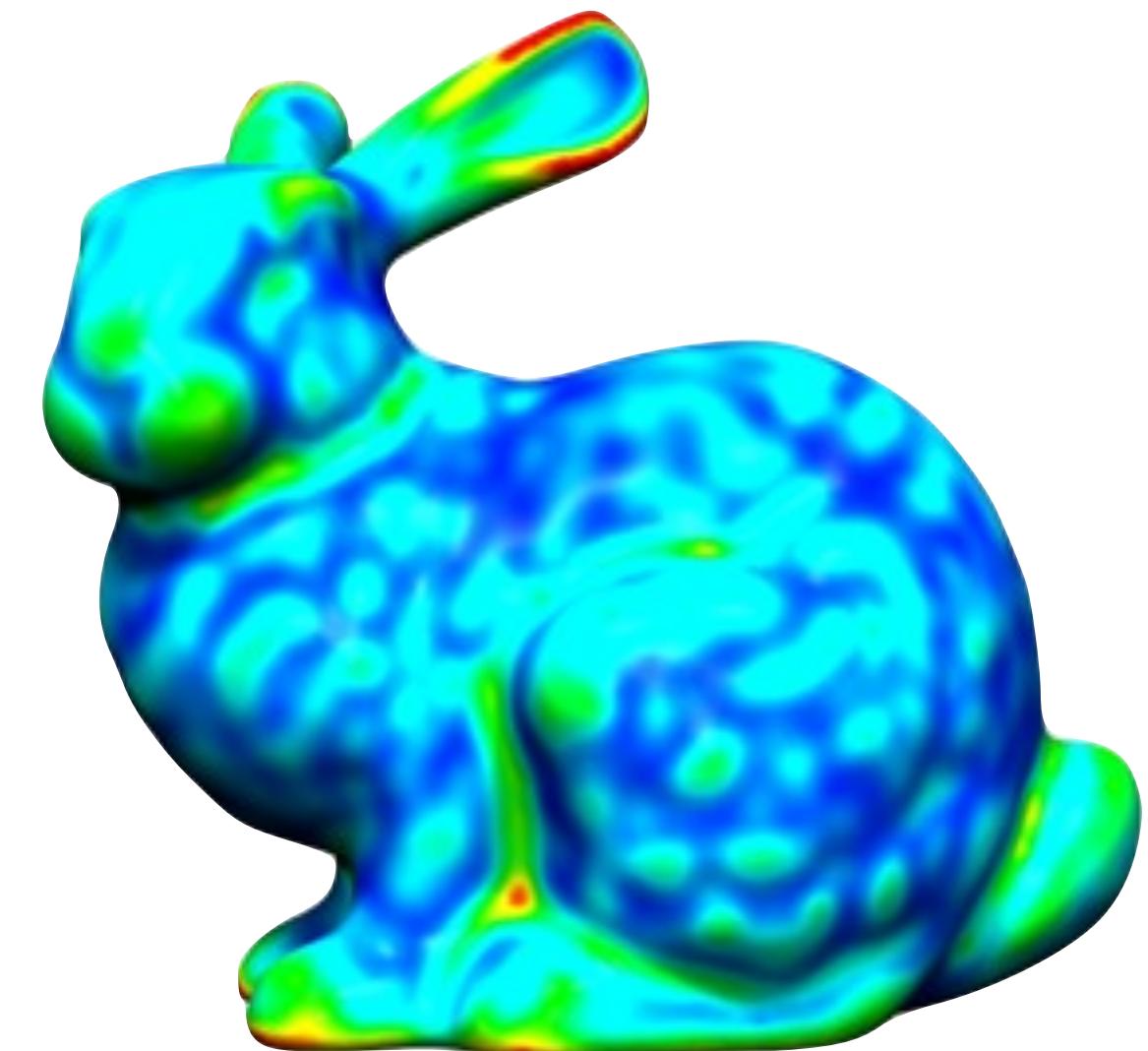
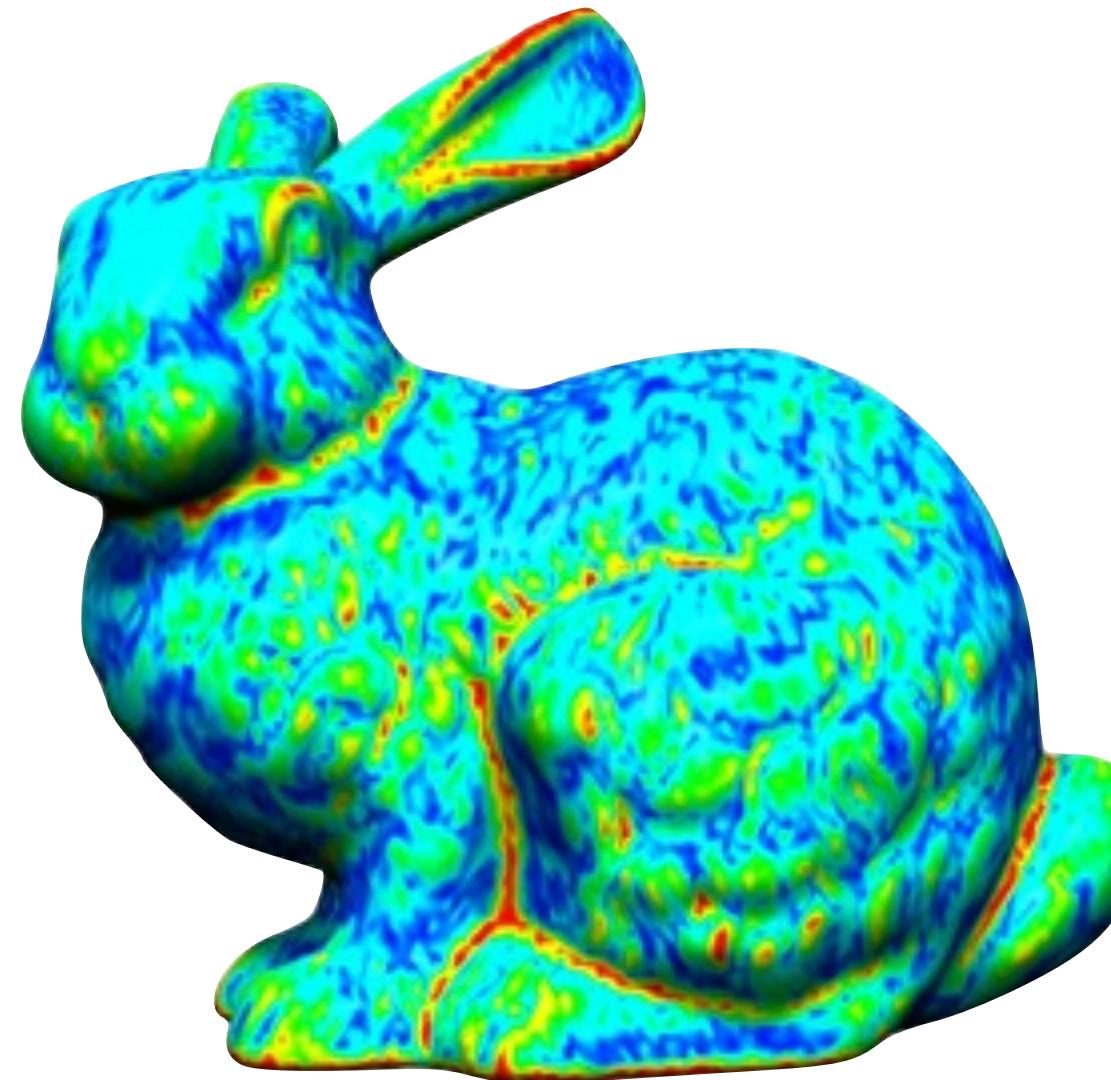
- Note: Laplacian weights generally change when mesh changes!

# Explicit Smoothing

- In matrix form:  $\frac{\partial \mathbf{v}}{\partial t} = \lambda \mathbf{L}(\mathbf{v})$
- Intuition: vertices moving toward neighbor average.

$$\mathbf{v}^{n+1} - \mathbf{v}^n = \lambda dt \mathbf{L} \mathbf{v}^n$$

$$\mathbf{v}^{n+1} = (I + \lambda dt \mathbf{L}) \mathbf{v}^n$$



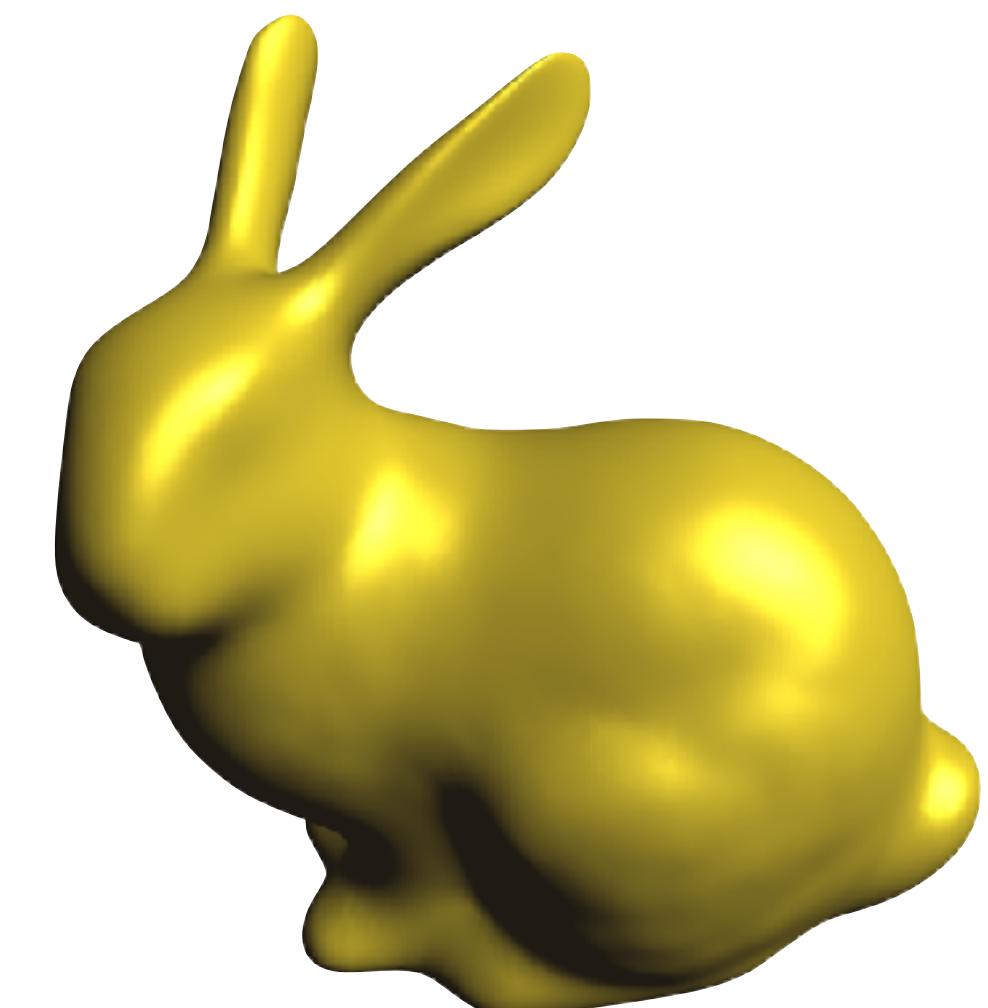
# Implicit Smoothing

- “Backward Euler” is more stable
- Take larger time steps without mesh “blowing up” (becoming jagged)  
Laplacian at next (unknown) step!

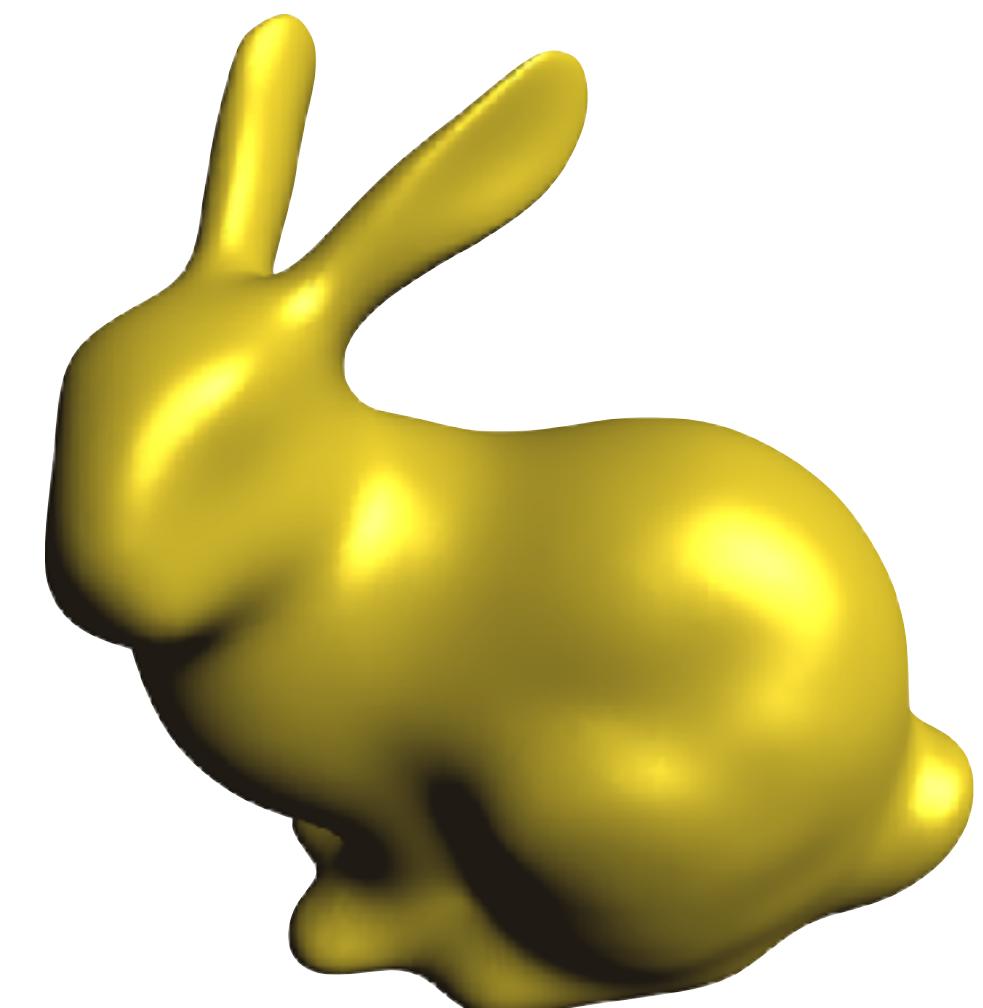
$$\mathbf{v}^{n+1} = \mathbf{v}^n + \lambda dt \mathbf{L} \mathbf{v}^{n+1}$$



original



explicit, 1k iterations,  
 $\lambda = 0.01$



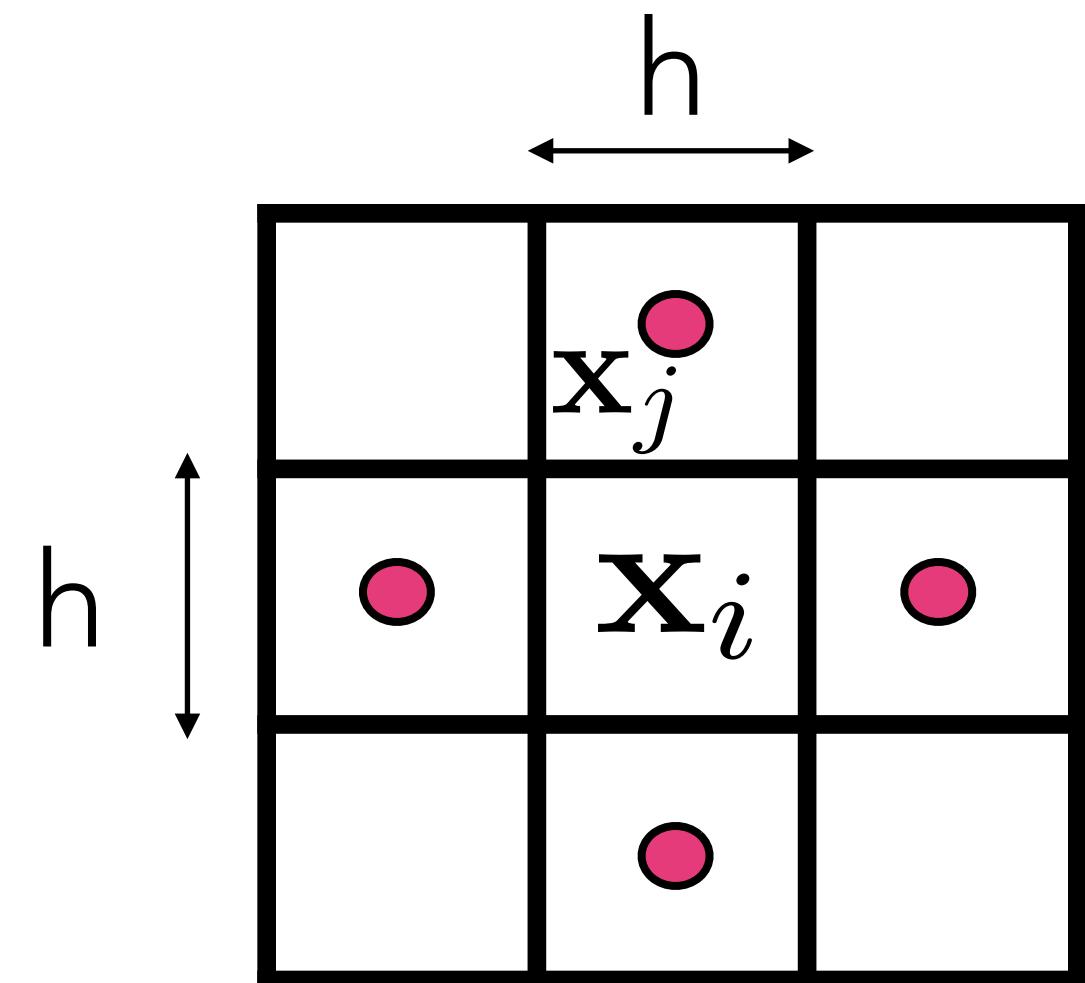
implicit, 1 iteration  
 $\lambda = 20$

Must solve a linear system:

$$(I - \lambda dt \mathbf{L}) \mathbf{v}^{n+1} = \mathbf{v}^n$$

libigl tutorial  
Chapter 1

# 2D Rectangular Grid Laplacian



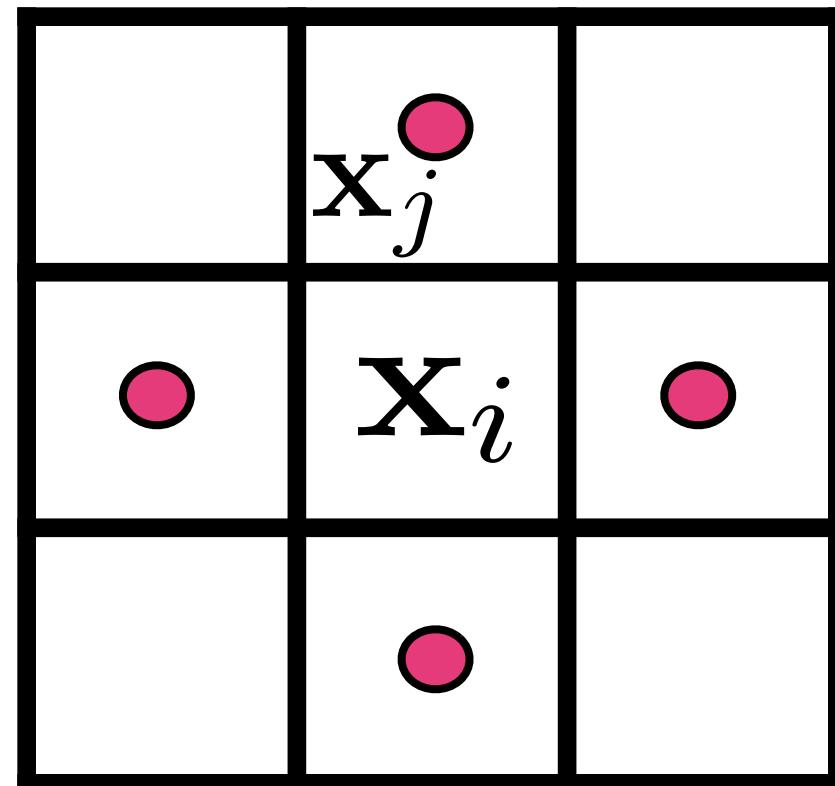
$$\Delta_h f(\mathbf{x}_i) := \frac{1}{h^2} \left[ \left( \sum_{\mathbf{x}_j \in N(\mathbf{x}_i)} f(\mathbf{x}_j) \right) - 4f(\mathbf{x}_i) \right]$$

- Measures difference from neighbor's average (if we divide by  $4/h^2$ ):

Weights can be represented by a “stencil:”

	1	
1	-4	1
	1	

# 2D Rectangular Grid Laplacian



$$\Delta_h f(\mathbf{x}_i) := \frac{1}{h^2} \left[ \left( \sum_{\mathbf{x}_j \in N(\mathbf{x}_i)} f(\mathbf{x}_j) \right) - 4f(\mathbf{x}_i) \right]$$

	1	
1	-4	1
	1	

## Derivation:

Compute second derivatives by finite-differencing the forward difference approximation to first derivative:

$$\begin{aligned} \frac{\partial^2 f}{\partial x^2}(x, y) &\approx \frac{1}{h} \left( \frac{f(x + h, y) - f(x, y)}{h} - \frac{f(x, y) - f(x - h, y)}{h} \right) \\ &= \frac{f(x + h, y) + f(x - h, y) - 2f(x, y)}{h^2} \quad \Rightarrow \end{aligned}$$

$$\Delta f(x, y) = \frac{\partial^2 f}{\partial x^2}(x, y) + \frac{\partial^2 f}{\partial y^2}(x, y)$$

$$\approx \boxed{\frac{f(x + h, y) + f(x - h, y) + f(x, y + h) + f(x, y - h) - 4f(x, y)}{h^2}}$$

Sum second partial derivative approximations for each dimension:

# Irregular Grid (Mesh) Laplacian

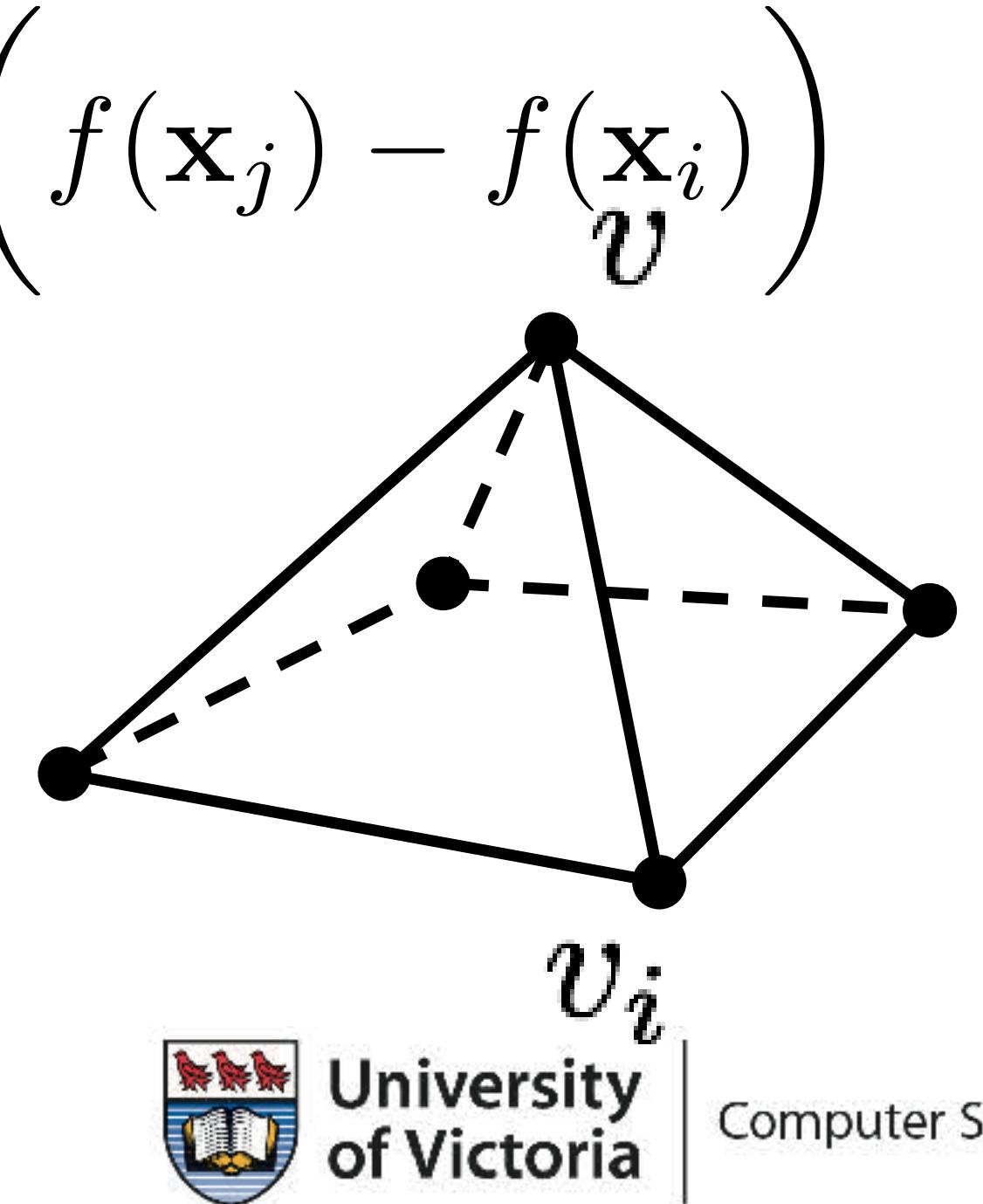
- First approach: “umbrella operator”
  - assume all edges are unit length
  - depends only on connectivity, not vertex locations
- Notice, when edge length  $h = 1$  the 2D grid laplacian is:

$$\Delta_h f(\mathbf{x}_i) = \frac{1}{h^2} \left[ \left( \sum_{\mathbf{x}_j \in N(\mathbf{x}_i)} f(\mathbf{x}_j) \right) - 4f(\mathbf{x}_i) \right] = \sum_{\mathbf{x}_j \in N(\mathbf{x}_i)} \left( f(\mathbf{x}_j) - f(\mathbf{x}_i) \right)$$

- This version easily generalizes to an arbitrary mesh,  $M$ :

$$\begin{aligned} \Delta_M^{\text{uniform}} f(\mathbf{v}) &= \sum_{\mathbf{v}_j \in N(\mathbf{v})} \left( f(\mathbf{v}_j) - f(\mathbf{v}) \right) \\ &= \left( \sum_{\mathbf{v}_j \in N(\mathbf{v})} f(\mathbf{v}_j) \right) - \underline{|N(\mathbf{v})|} f(\mathbf{v}) \end{aligned}$$

#neighbors



# The “Umbrella” (Uniform) Laplacian

$$\Delta_M^{\text{uniform}} f(\mathbf{v}) = \left( \sum_{\mathbf{v}_j \in N(\mathbf{v})} f(\mathbf{v}_j) \right) - |N(\mathbf{v})|f(\mathbf{v})$$

- In Matrix Form:

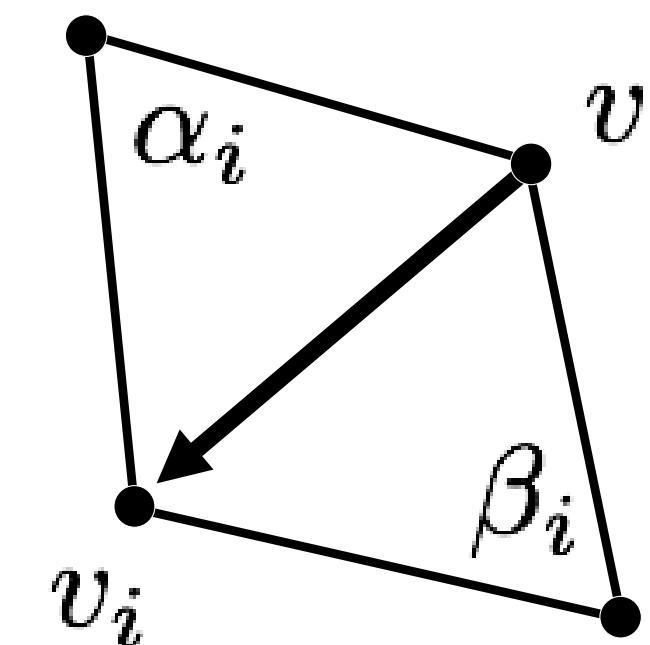
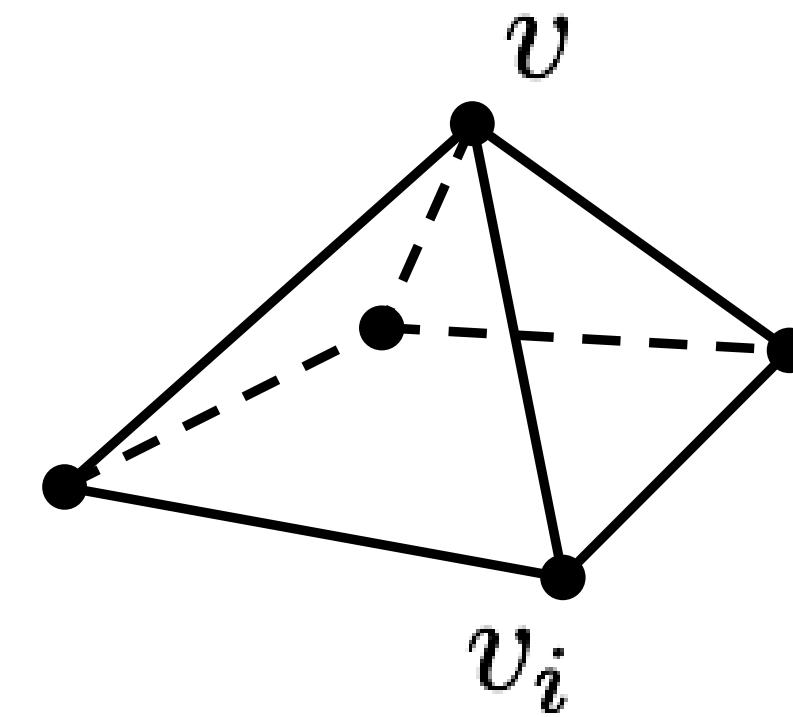
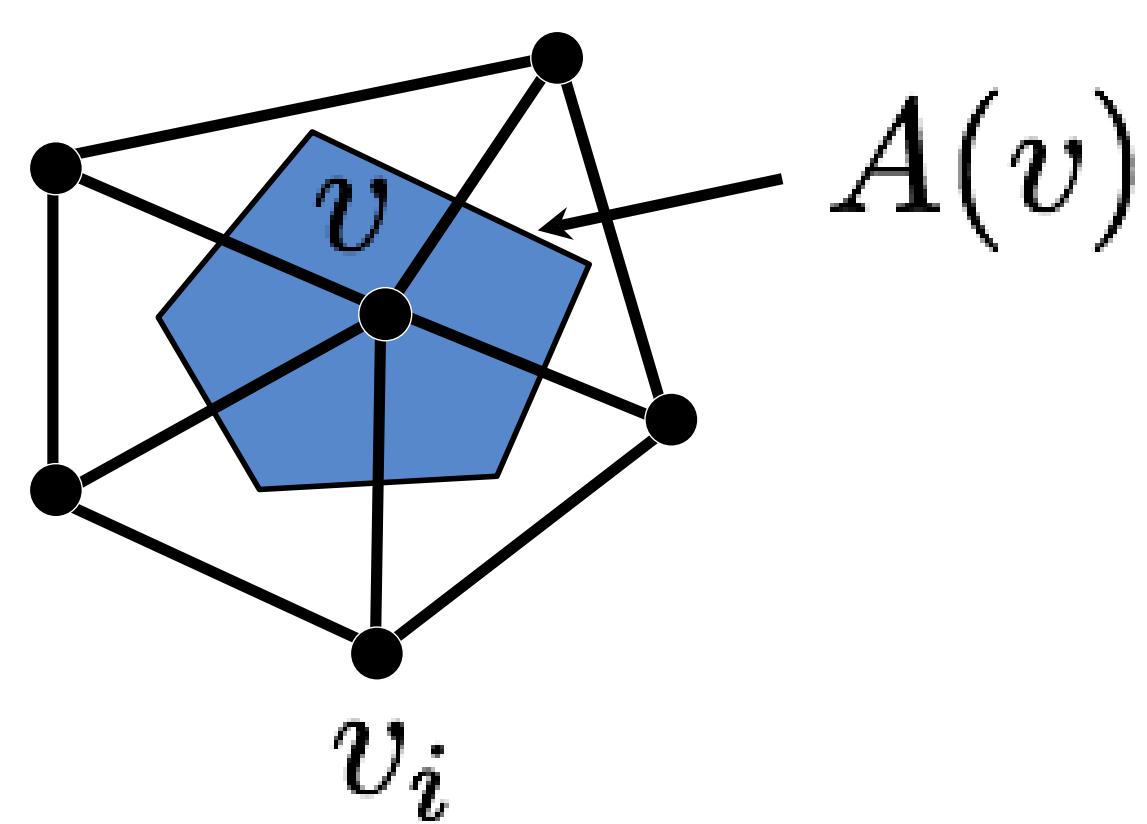
$$\mathbf{Y} = \mathbf{LF}$$

$$\mathbf{Y} = \begin{bmatrix} \Delta f(\mathbf{v}_1) \\ \Delta f(\mathbf{v}_2) \\ \Delta f(\mathbf{v}_3) \\ \dots \\ \Delta f(\mathbf{v}_N) \end{bmatrix} \quad \mathbf{L} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1N} \\ w_{21} & w_{22} & \cdots & w_{2N} \\ \vdots & \vdots & \cdots & \vdots \\ w_{N1} & w_{N2} & \cdots & w_{NN} \end{bmatrix} = \{w_{ij}\} \quad \mathbf{F} = \begin{bmatrix} f(\mathbf{v}_1) \\ f(\mathbf{v}_2) \\ f(\mathbf{v}_3) \\ \dots \\ f(\mathbf{v}_N) \end{bmatrix}$$

$$w_{ij} = \begin{cases} 0 & i \neq j, \nexists \text{ edge } (i, j) \\ 1 & i \neq j, \exists \text{ edge } (i, j) \\ -|N(v_i)| & i = j \end{cases}$$

# The Cotangent Laplacian

$$\Delta_M^{\text{cotan}} f(\mathbf{v}) = \frac{1}{2A(\mathbf{v})} \sum_{\mathbf{v}_i \in N(\mathbf{v})} (\cot(\alpha_i) + \cot(\beta_i)) (f(\mathbf{v}_i) - f(\mathbf{v}))$$



- Accounts for edges' differing lengths.
- Converges to the continuous Laplace-Beltrami operator  $\Delta_S$  with mesh refinement
- Derivation in next recitation lecture.

# The Cotangent Laplacian

$$\Delta_M^{\text{cotan}} f(\mathbf{v}) = \frac{1}{2A(\mathbf{v})} \sum_{\mathbf{v}_i \in N(\mathbf{v})} (\cot(\alpha_i) + \cot(\beta_i)) \left( f(\mathbf{v}_i) - f(\mathbf{v}) \right)$$

- In Matrix Form:

$$\mathbf{Y} = \mathbf{LF}$$

$$\mathbf{Y} = \begin{bmatrix} \Delta f(\mathbf{v}_1) \\ \Delta f(\mathbf{v}_2) \\ \Delta f(\mathbf{v}_3) \\ \dots \\ \Delta f(\mathbf{v}_N) \end{bmatrix} \quad \mathbf{L} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1N} \\ w_{21} & w_{22} & \cdots & w_{2N} \\ \vdots & \vdots & \cdots & \vdots \\ w_{N1} & w_{N2} & \cdots & w_{NN} \end{bmatrix} = \{w_{ij}\} \quad \mathbf{F} = \begin{bmatrix} f(\mathbf{v}_1) \\ f(\mathbf{v}_2) \\ f(\mathbf{v}_3) \\ \dots \\ f(\mathbf{v}_N) \end{bmatrix}$$

$$w_{ij} = \begin{cases} 0 & i \neq j, \not\exists \text{ edge } (i, j) \\ \frac{\cot \alpha_j + \cot \beta_j}{2A(v_i)} & i \neq j, \exists \text{ edge } (i, j) \\ - \sum_{v_k \in N(v_i)} w_{ik} & i = j \end{cases}$$

# SciPy Sparse Matrix

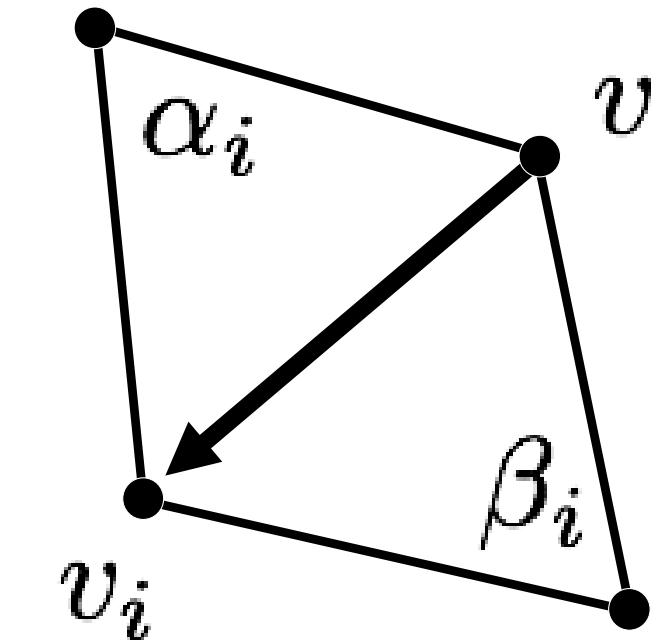
- Full-sized Laplacian can be huge for large meshes
  - but most elements are zero!
- Instead, only store non-zero elements: **Sparse Matrix**
- <https://docs.scipy.org/doc/scipy/reference/sparse.html>

# How to build cotangent matrix

$$\Delta_M^{\text{cotan}} f(\mathbf{v}) = \frac{1}{2A(\mathbf{v})} \sum_{\mathbf{v}_i \in N(\mathbf{v})} (\cot(\alpha_i) + \cot(\beta_i)) \left( f(\mathbf{v}_i) - f(\mathbf{v}) \right)$$

- Iterating over vertices is slow
- Instead, iterate over faces and add cot terms to the vertices incident each face edge

```
for i = 1:number_of_faces
    for j = 1:face_valence
        source_vertex = faces(i,j);
        destination_vertex = faces(i,(j+1) % face_valence);
        weight = ....; //laplacian weight for edge (source_vertex, destination_vertex)(cotan or uniform)
        Laplacian(source_vertex, destination_vertex) += weight;
        Laplacian(destination_vertex, source_vertex) += weight;
        Laplacian(destination_vertex, destination_vertex) -= weight;
        Laplacian(source_vertex, source_vertex) -= weight;
    end
end
```



# Averaged or Summed?

- We'll see next time that the cotan laplacian we introduced:

$$\Delta_M^{\text{cotan}} f(\mathbf{v}) = \frac{1}{2A(\mathbf{v})} \sum_{\mathbf{v}_i \in N(\mathbf{v})} (\cot(\alpha_i) + \cot(\beta_i)) (f(\mathbf{v}_i) - f(\mathbf{v}))$$

computes the **average** Laplacian over some area "A" around v (hence the division by A)

- This gives an **asymmetric matrix**. But if we instead compute the "summed" laplacian (i.e., don't divide by A), we end up with a symmetric matrix.
- This "summed," symmetric Laplacian is more standard in the finite element community, and is related to our Laplacian by a "lumped (diagonal) mass matrix" holding the areas:

lumped mass matrix  
(averaging region areas)

$$\mathbf{M} = \begin{bmatrix} A(0) & 0 & \cdots & 0 \\ 0 & A(1) & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & A(N) \end{bmatrix}$$

(Laveraged corresponds to formula above)

$$L^{\text{averaged}} = M^{-1} L^{\text{summed}}$$

$$[L^{\text{summed}}]_{ij} = \begin{cases} \frac{1}{2} \left( \cot(\alpha_{ij}) + \cot(\beta_{ij}) \right) & \text{if } j \in N(i) \\ -\sum_{j \in N(i)} [L^{\text{summed}}]_{ij} & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

# Averaged or Summed?

- Suppose we have a linear system:

$$L^{\text{averaged}} \mathbf{x} = \mathbf{b}$$

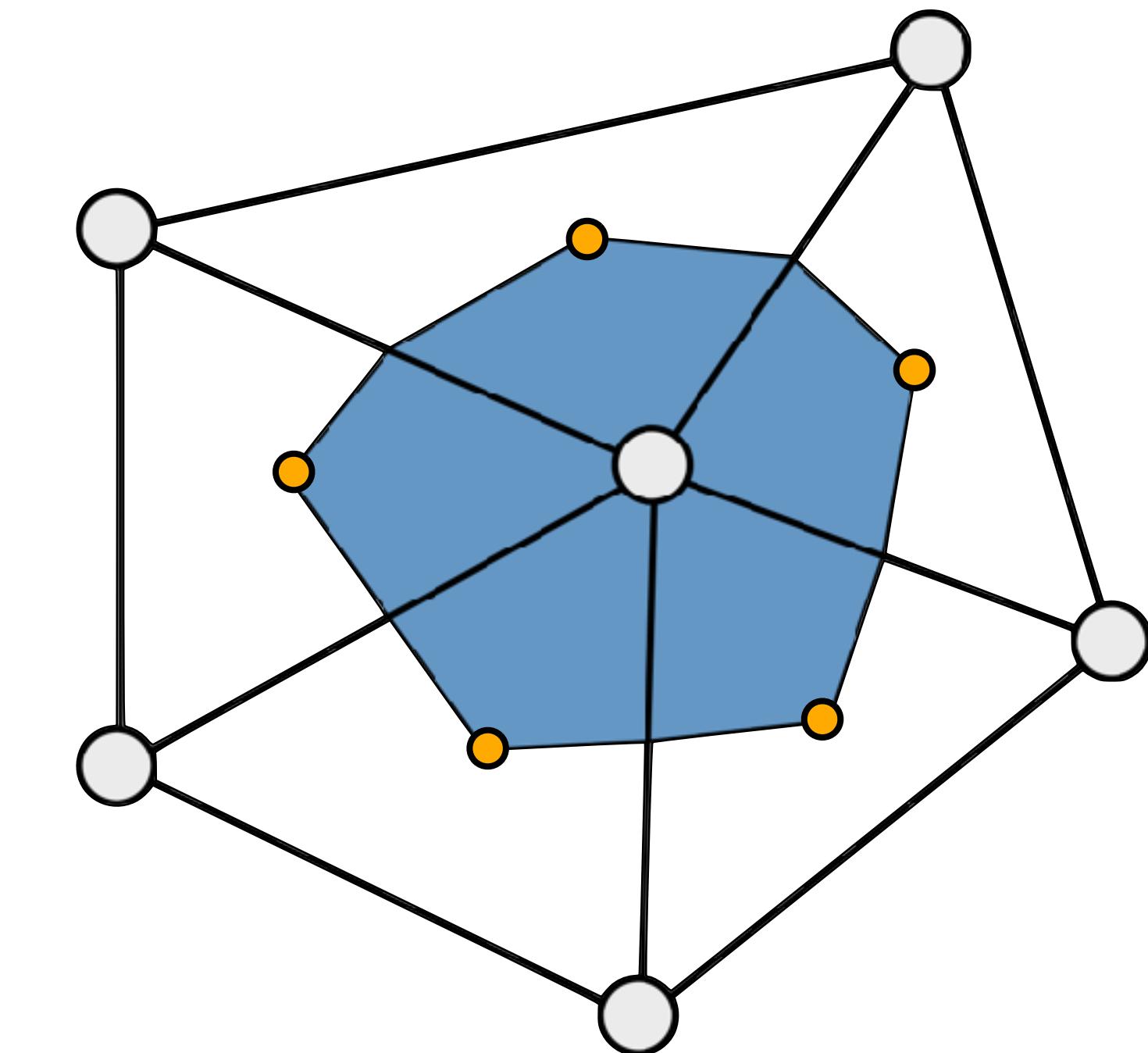
- i.e. using the asymmetric cotangent Laplacian we introduced earlier.
- Right hand side vector “b” represents a scalar field, storing the desired Laplacian value (**not the summed value!**) at each vertex.
- Solving symmetric sparse linear systems matrices is much more efficient, so we prefer to solve with the summed quantities:

$$ML^{\text{averaged}} \mathbf{x} = \boxed{L^{\text{summed}} \mathbf{x} = M\mathbf{b}}$$



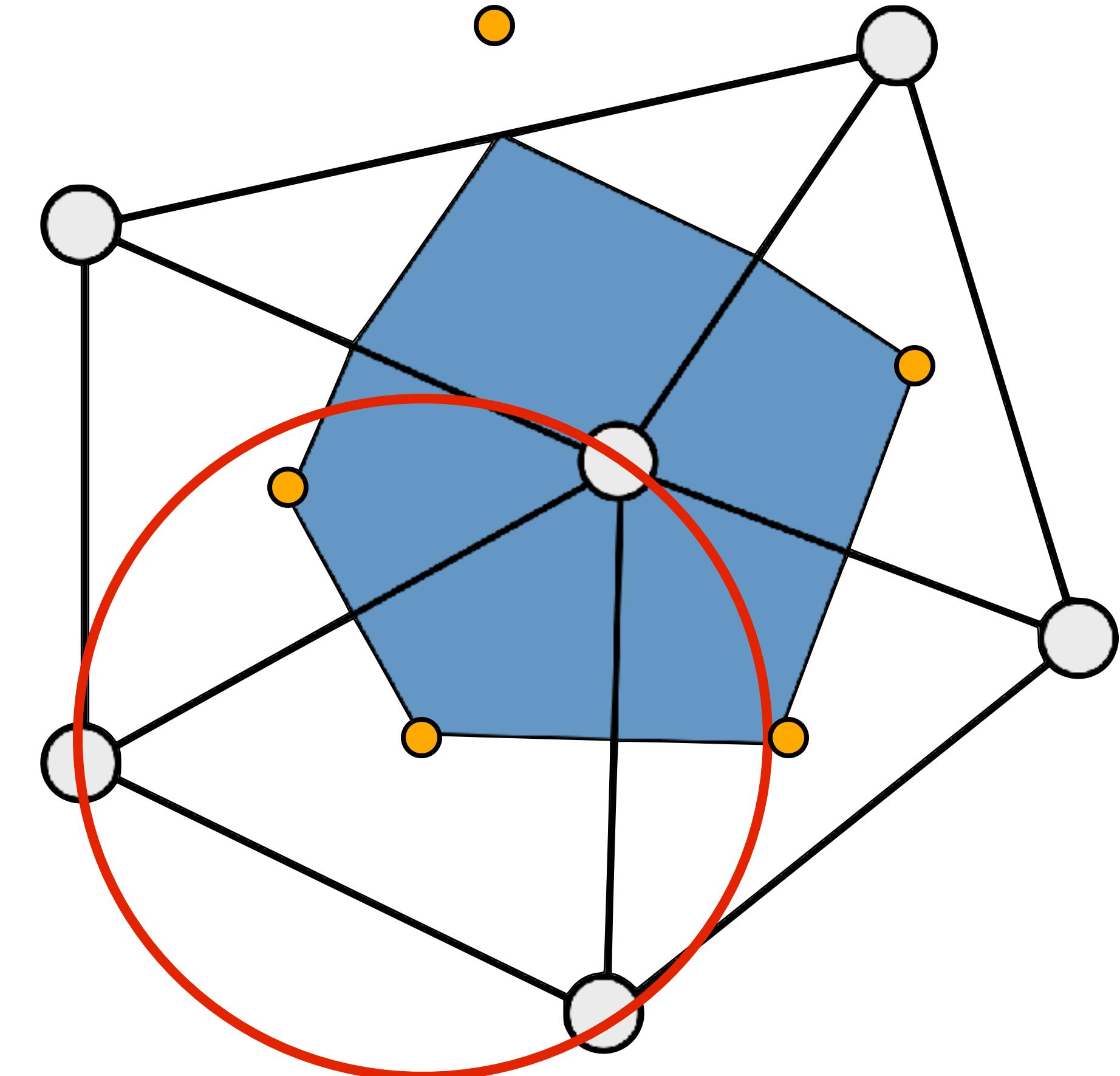
# How to compute $A(v)$ - Barycentric area

- Connect edge midpoints and triangle barycenters
- Each of the incident triangles contributes  $1/3$  of its area to all its vertices, regardless of the placement
  - + Simple to compute
  - + Always positive weights
  - Heavily connectivity-dependent
  - Changes if edges are flipped



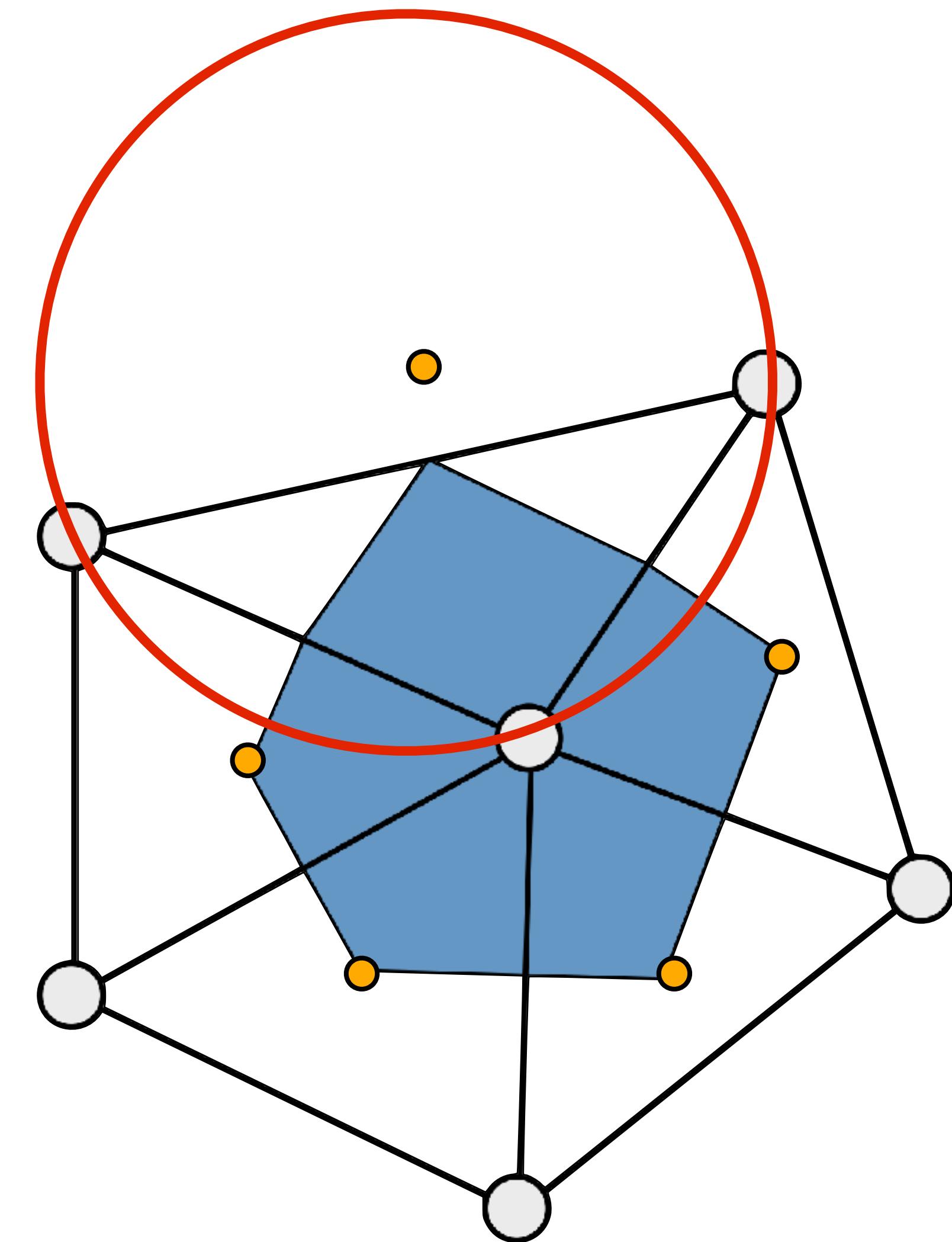
# How to compute $A(v)$ - Voronoi area

- Connect edge midpoints and triangle circumcenters
- The 3 resulting quadrilaterals are not equareal
- Sum contributions from incident triangles
  - + Only depends on vertex positioning
  - More complicated computations



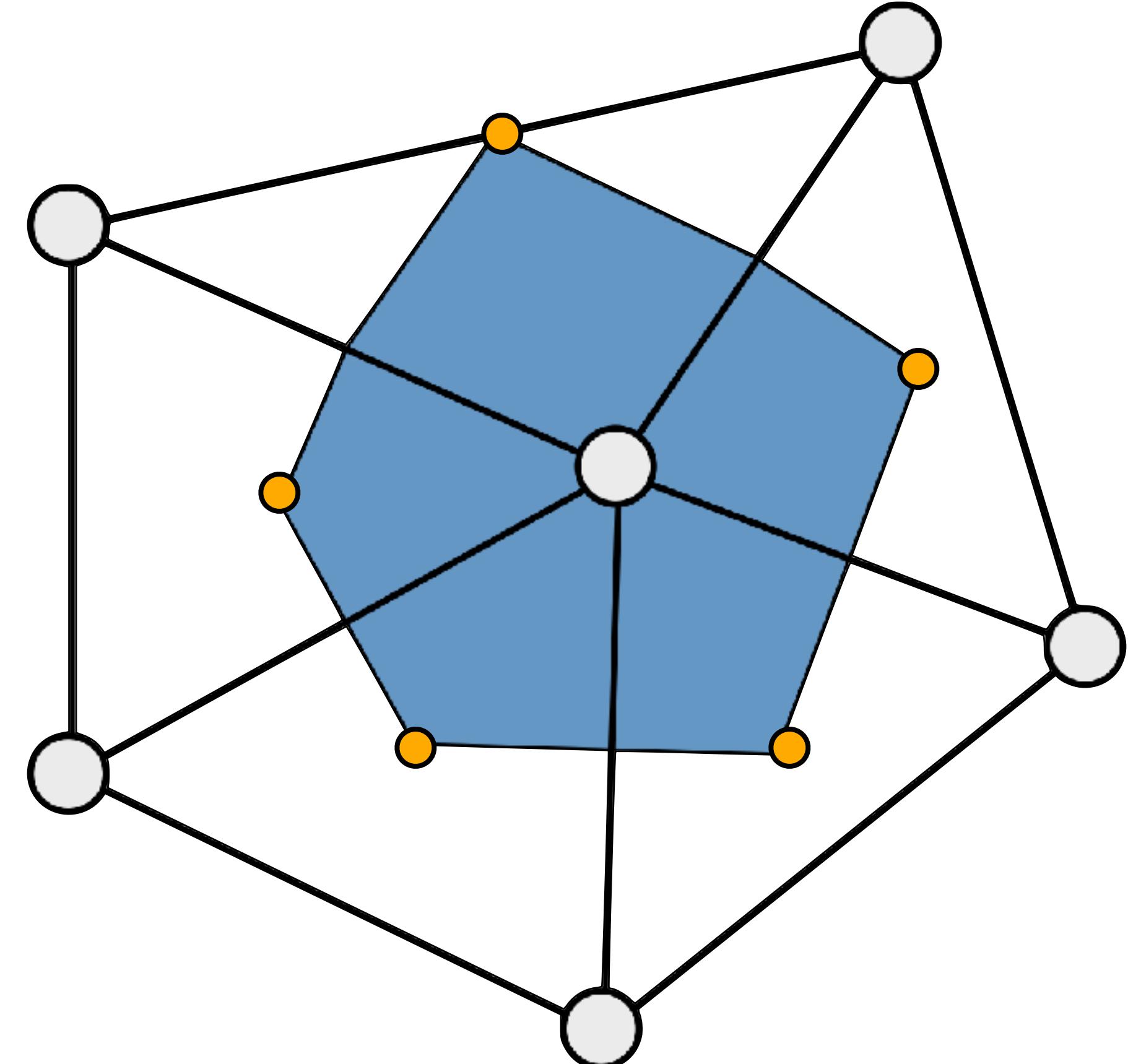
# How to compute $A(v)$ - Voronoi area

- Connect edge midpoints and triangle circumcenters
- The 3 resulting quadrilaterals are not equiareal
- Sum contributions from incident triangles
- For obtuse triangles, circumcenter is outside the triangle -> negative areas!
  - + Only depends on vertex positioning
  - More complicated computation
  - May introduce negative weights (obtuse triangles)



# How to compute $A(v)$ - Voronoi area - compromise

- Connect edge midpoints with:
  - triangle circumcenters , for non-obtuse triangles
  - midpoint of opposite edge, for obtuse ones
- Sum contributions from incident triangles
  - + Only depends on vertex positioning
  - More complicated computations



# Linear Systems with SciPy

```
from scipy.sparse.linalg import spsolve
```

```
A = ... # a SciPy sparse matrix
```

```
b = ... # a NumPy array
```

```
x = spsolve(A, b)
```

# Questions?