Shape Modeling and Geometry Processing

Optional Exercise 3 - Discrete Differential Quantities

Alexandre Binninger

alexandre.binninger@inf.ethz.ch

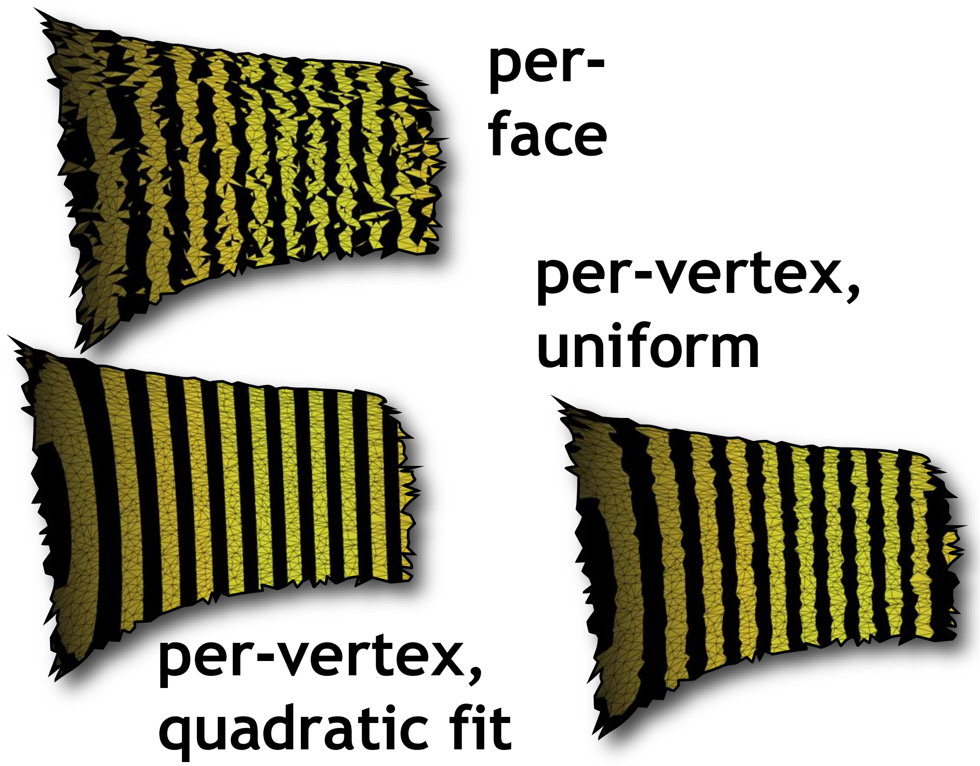
INTERACTIVE GEOMETRY LAB

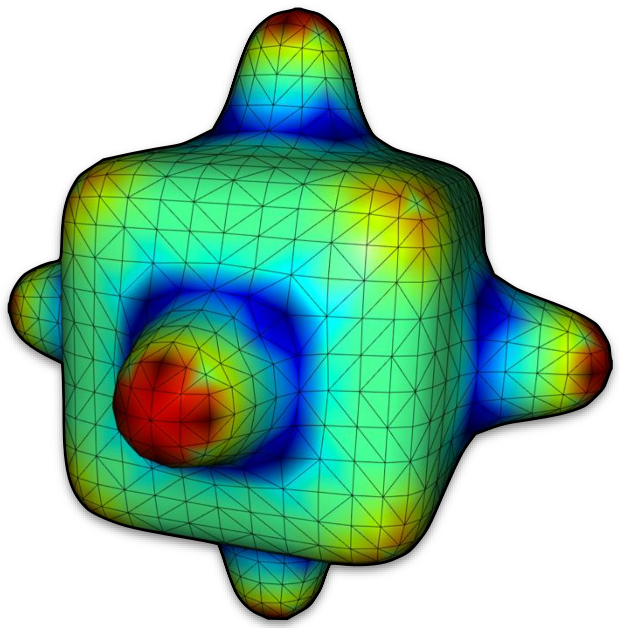
March 21, 2025

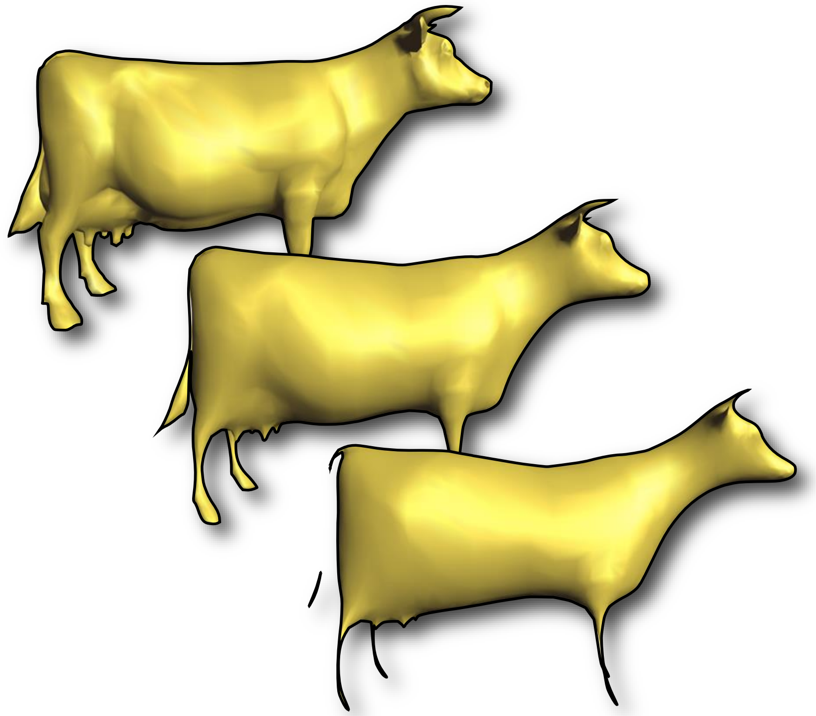
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

This exercise

- Topic: Discrete differential quantities with libigl
  - Normals, Curvature, Smoothing



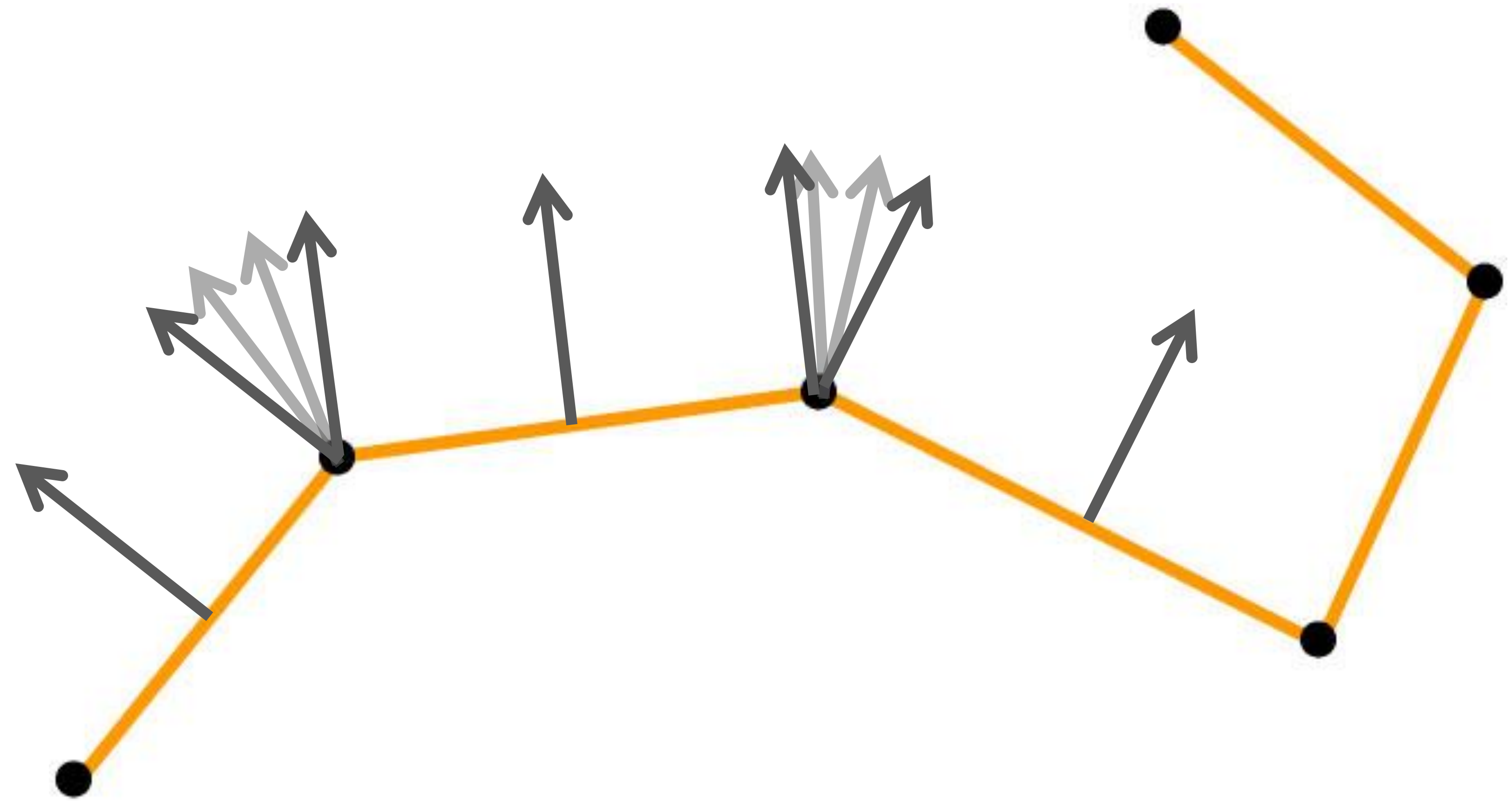




- Optional assignment: No grade!
- But it will help with next assignments & mini-exam.

Vertex Normals

- Normals on line segments and triangles are well defined
- Many options for vertices!



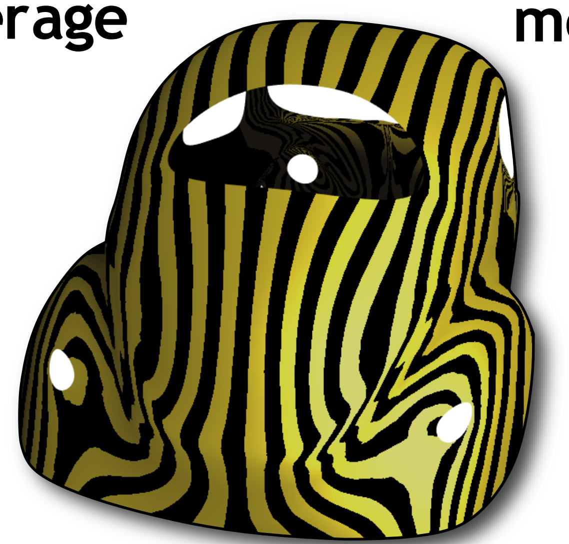
Vertex Normals

uniform average

area-weighted average

mean-curvature based

PCA on k-nearest neighbors



quadratic fitting on k-nearest neighbors

libigl tutorial #201
and
igl::principal_curvature()

Curvature

- How to compute curvature on triangle meshes?
- We know from the theory of smooth surfaces:

$$\Delta_{\mathcal{M}} \mathbf{p} = -2H\mathbf{n}$$

$\Delta_{\mathcal{M}}$: Laplace-Beltrami Operator aka. second derivative on the manifold $\mathcal{M}$.

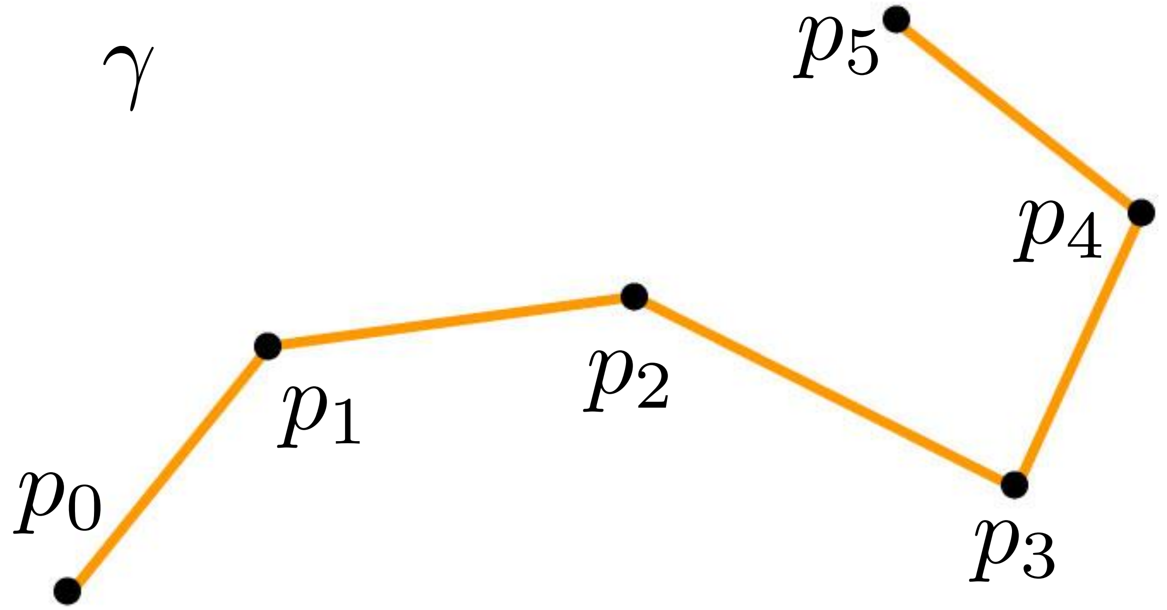
$\mathbf{p}$: Coordinate function of the surface.

H: Mean curvature.

$\mathbf{n}$: Surface normal.

The Laplace operator

- How to discretize the second derivative of a piecewise linear curve γ.



$\gamma(i) = p_i$ for $i = 0, 1, \ldots, 5$

- Use finite differences.

The Laplace operator

- Finite differences (backward) for first derivative:

$$\gamma'(i) = \gamma(i) - \gamma(i-1)$$

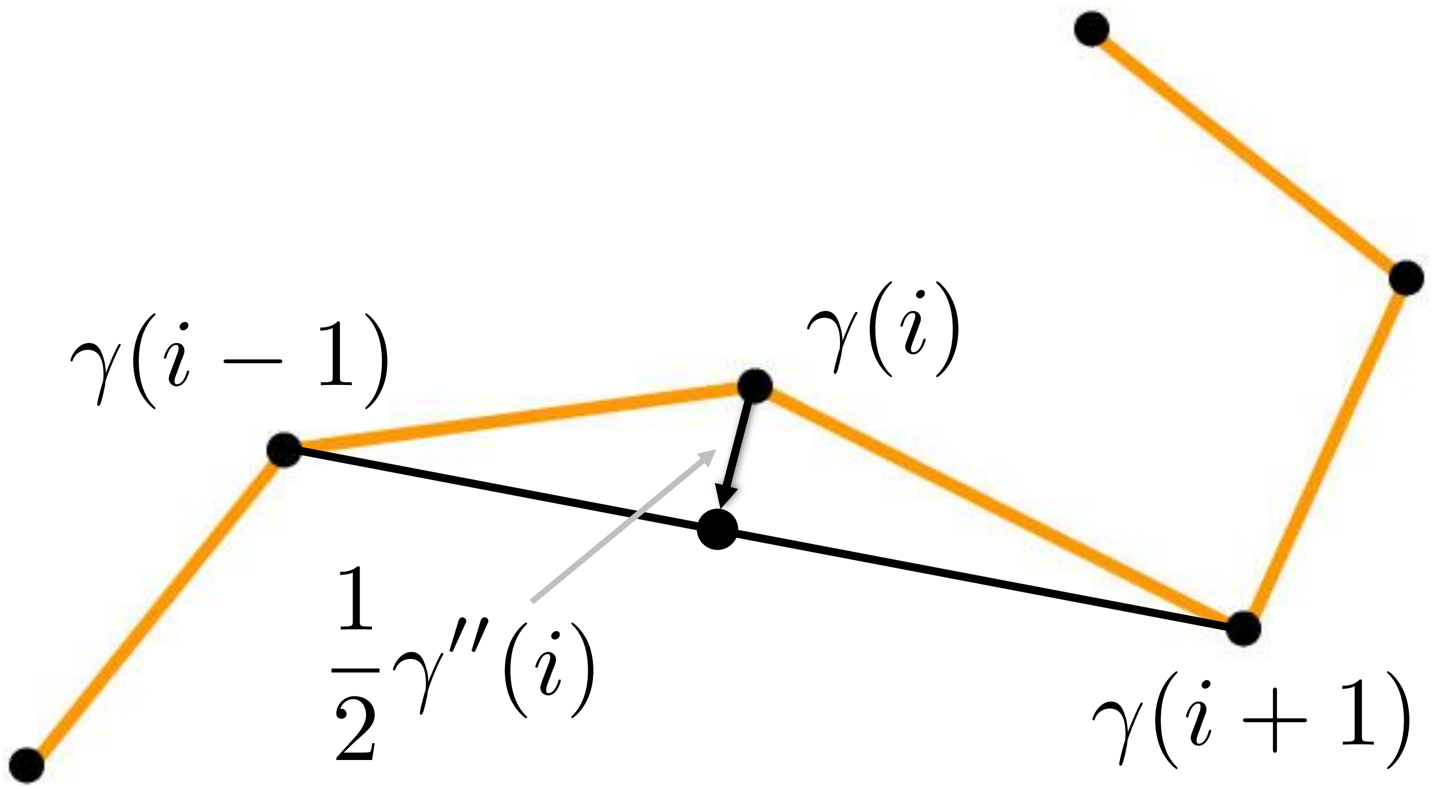
- Finite differences (forward) for second derivative:

$$\begin{aligned}\gamma''(i) &= \gamma'(i+1) - \gamma'(i) \\ &= \gamma(i+1) - \gamma(i) - \gamma(i) + \gamma(i-1) \\ &= \gamma(i+1) + \gamma(i-1) - 2\gamma(i)\end{aligned}$$

The Laplace operator

$$\frac{1}{2}\gamma''(i) = \frac{1}{2}(\gamma(i+1) + \gamma(i-1)) - \gamma(i)$$

$$\frac{1}{2}\gamma''(i) = H_i \mathbf{n}_i$$

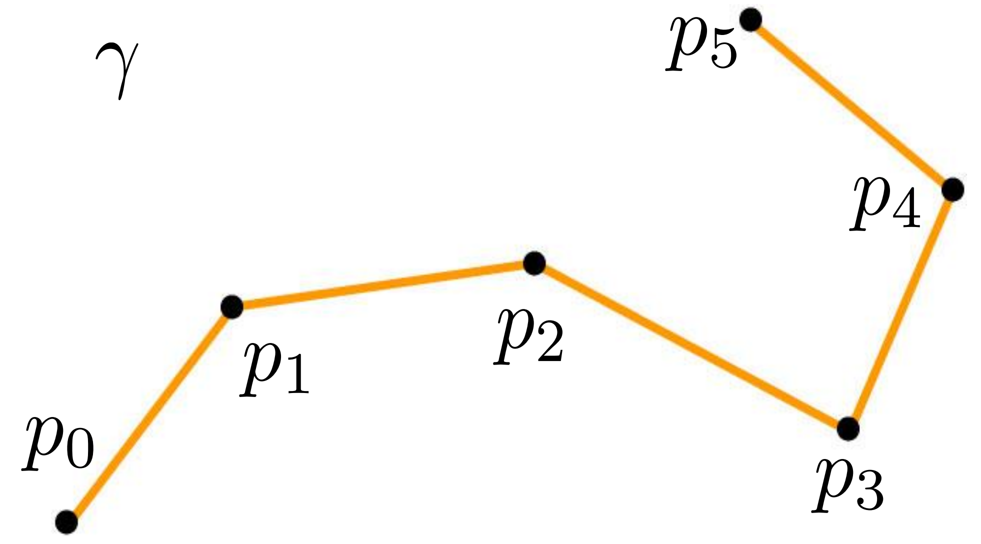


"Difference of the mean of neighbours and current position."

The Laplace operator

- Discrete Laplace operator for a curve.

$$L = \begin{pmatrix} -1 & 1 & 0 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 & 0 \\ 0 & 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 0 & 1 & -1 \end{pmatrix} \quad \mathbf{P} = \begin{pmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \vdots \\ \mathbf{p}_5 \end{pmatrix} \quad \mathbf{H} = \begin{pmatrix} H_0\mathbf{n}_0 \\ H_1\mathbf{n}_1 \\ \vdots \\ H_5\mathbf{n}_5 \end{pmatrix}$$



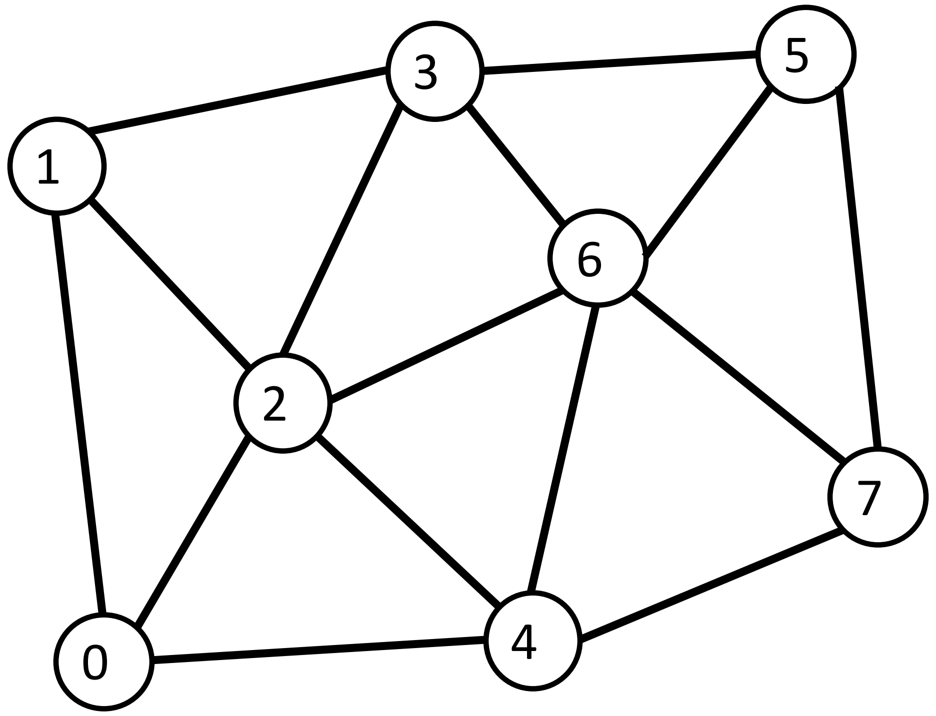
$$\frac{1}{2}L\mathbf{P} = \mathbf{H}$$

- But: This is only one possible discretization! (and not a particularly ”good” one).

The Laplace operator

- Discrete Laplace operator for a triangle mesh.

$$L = \begin{pmatrix} -3 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & -3 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & -5 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & -4 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & -4 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & -3 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & -5 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & -3 \end{pmatrix}$$

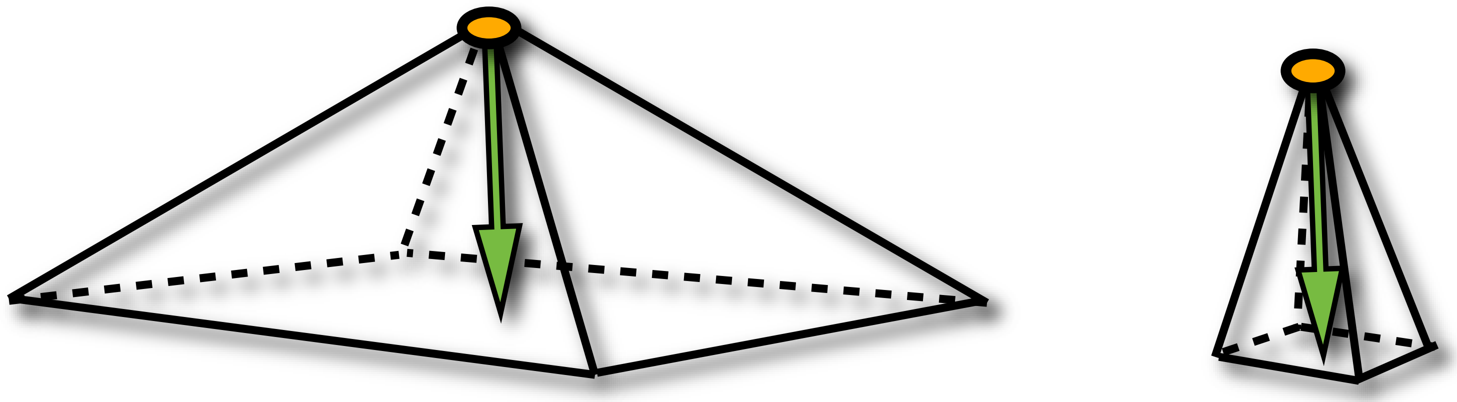


$$(L\mathbf{X})_i = \sum_{j \in \mathcal{N}_i} w_{ij}(x_j - x_i) \text{ with } w_{ij} = 1$$

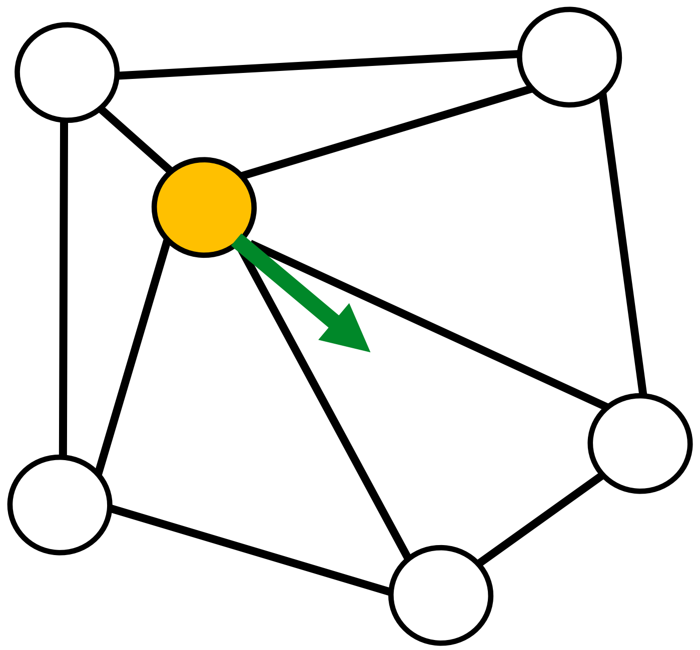
The Laplace operator

- Problems with the basic discretization:

<u>Frequency confusion:</u>
Scale does matter for curvature.



<u>Tangential component:</u>
A planar triangulation does not have curvature. The vector should vanish!

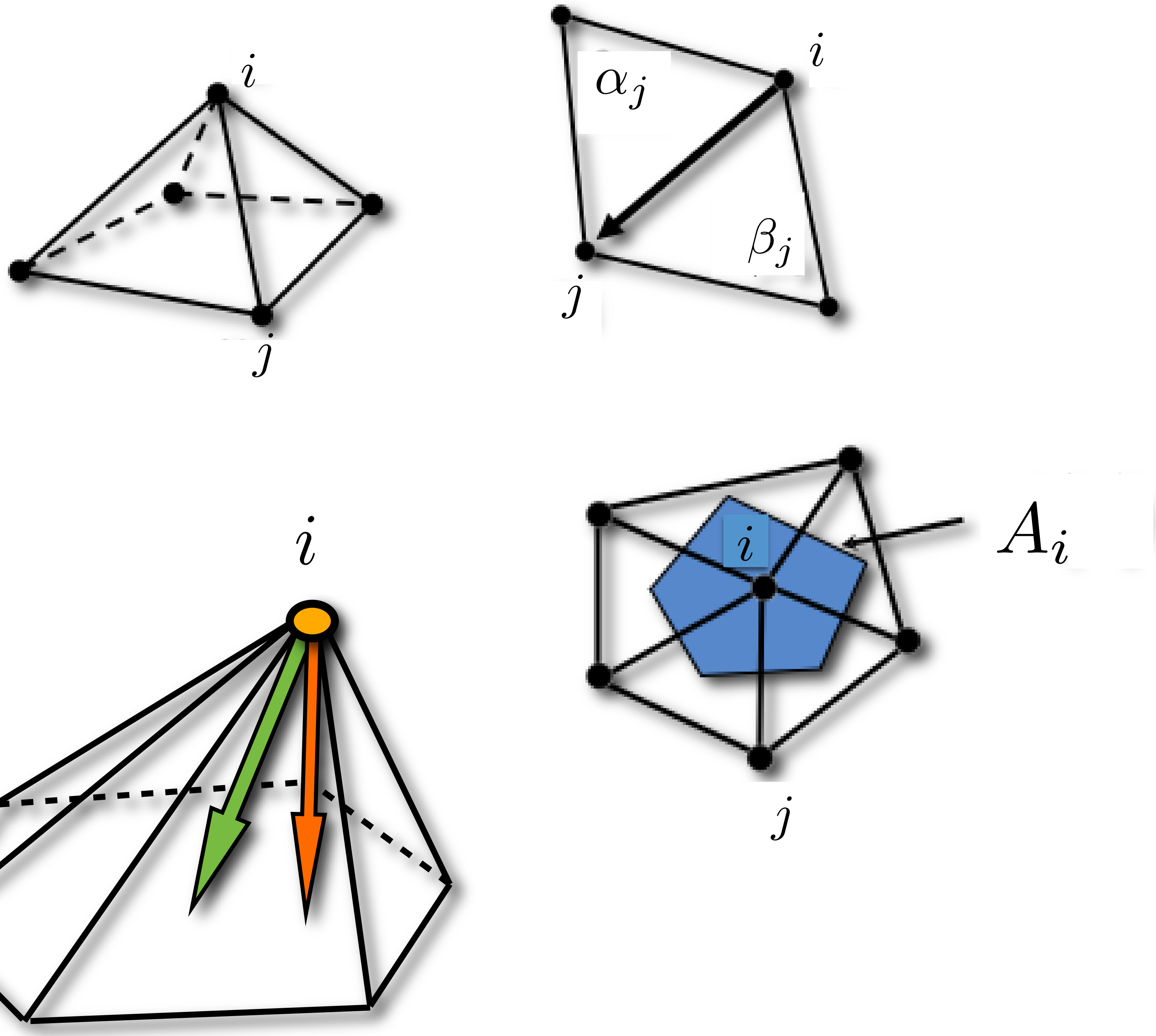


The Laplace operator

- Better: cotangent Laplacian

$$(L\mathbf{X})_i = \sum_{j \in \mathcal{N}_i} w_{ij}(x_j - x_i)$$

$$w_{ij} = \frac{1}{2A_i}(\cot \alpha_j + \cot \beta_j)$$



The Laplace operator

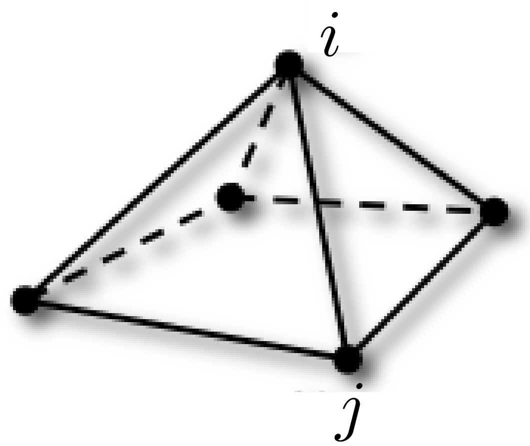
- Better: cotangent Laplacian

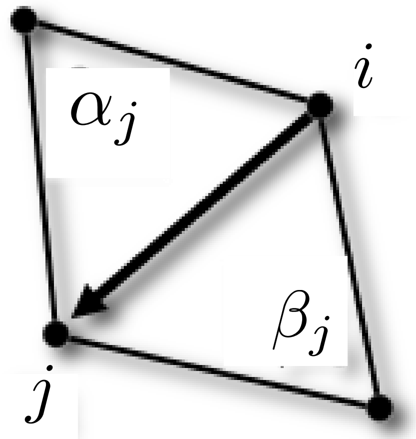
$$(L\mathbf{X})_i = \sum_{j \in \mathcal{N}_i} w_{ij}(x_j - x_i)$$

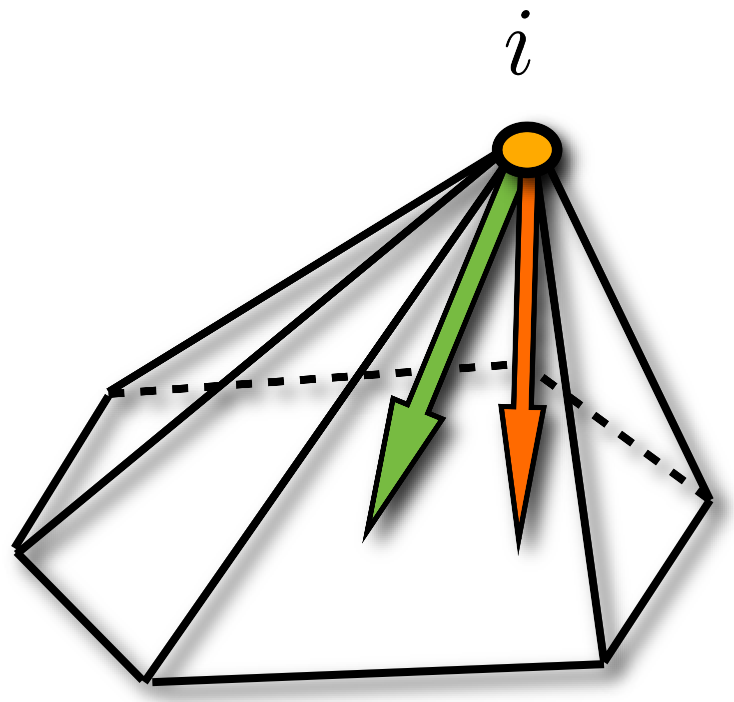
$$w_{ij} = \frac{1}{2A_i}(\cot \alpha_j + \cot \beta_j)$$

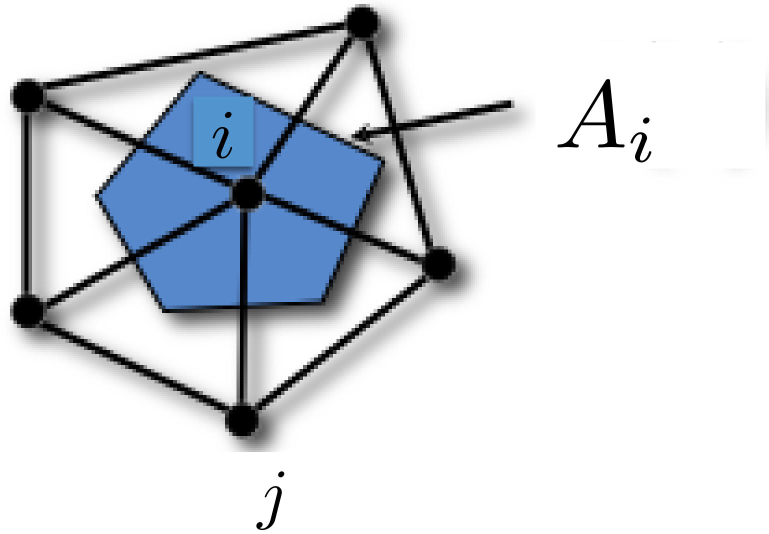
Fixes tangential component.

Fixes scale dependence.









Eigen Sparse Matrix

- Full-sized Laplacian can be huge for large meshes
  - but most elements are zero!
- Instead, only store non-zero elements: **Sparse Matrix**

```
#include <Eigen/Sparse>

Eigen::SparseMatrix<double> Laplacian;
```

Eigen Sparse Matrix

- How to initialize the matrix

```
//declare size of matrix
Eigen::SparseMatrix<double> L(V.rows(), V.rows());

//declare list of non-zero elements (row, column, value)
std::vector<Eigen::Triplet<double> > tripletList;

//insert element to the list
//if multiple triplets exist with the same row and column, values will be *added*
tripletList.push_back(Eigen::Triplet<double>(source,dest,value));

//construct matrix from the list
L.setFromTriplets(tripletList.begin(), tripletList.end());
```

see igl::cotmatrix (V,F,L)

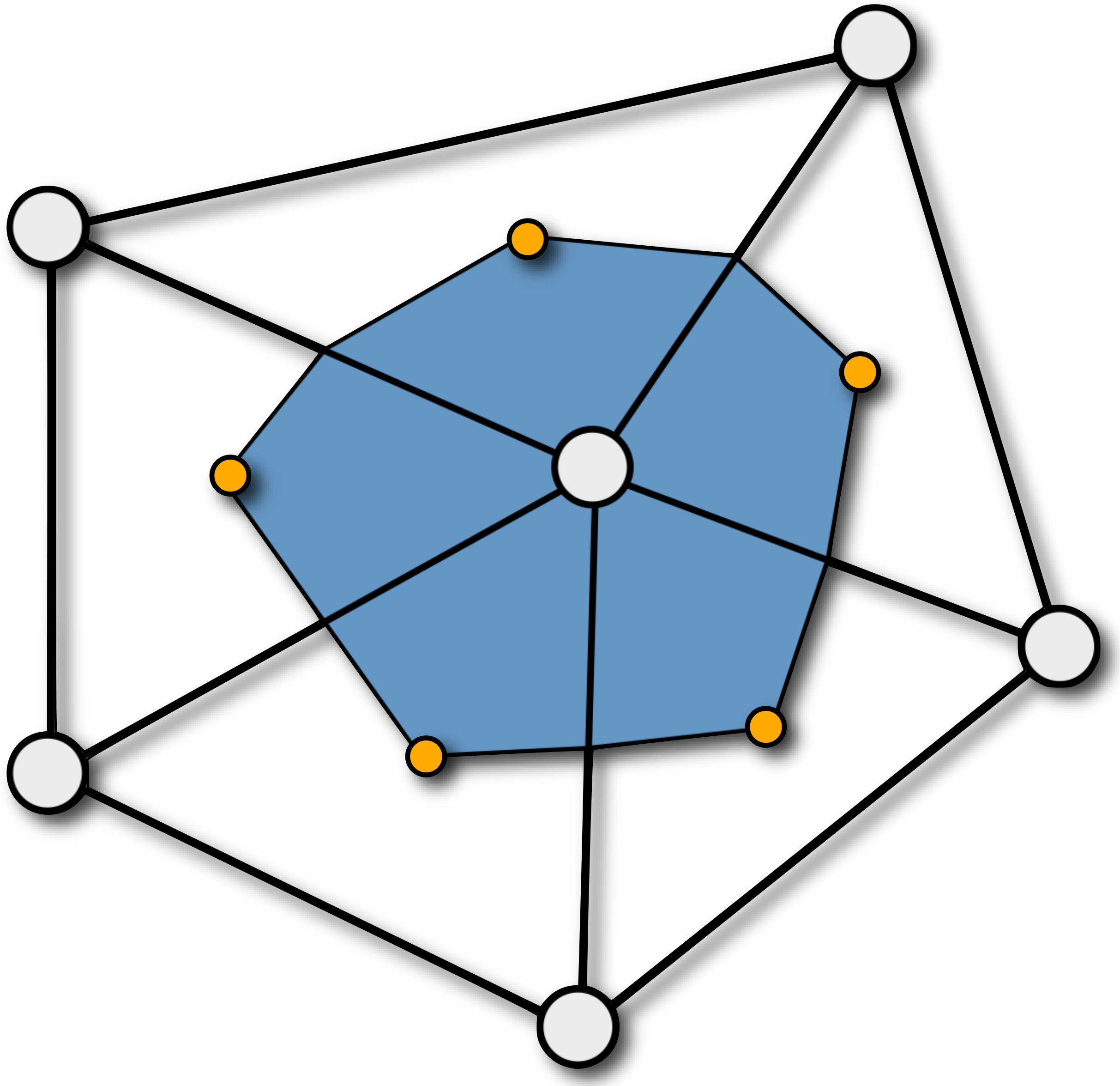
Do NOT use L.insert() or L.coeffRef() for creation → very slow !

How to build cotangent matrix

```
//declare size of matrix
Eigen::SparseMatrix<double> L(V.rows(), V.rows());

//declare list of non-zero elements (row, column, value)
std::vector<Eigen::Triplet<double> > tripletList;

for (int = 0; i < n_faces; ++i)
{
        for(int j = 0; j < 3; ++j)
        {
                double wij = cotanWeight(i, j); //laplacian weight 1 / (2 A_i) \cot(\alpha_j)
                int j1 = (j+1) % 3;
                int j2 = (j+2) % 3;

                tripletList.push_back(Eigen::Triplet<double>(F(i, j1)), F(i, j2, wij));
                tripletList.push_back(Eigen::Triplet<double>(F(i, j2)), F(i, j1, wij));
tripletList.push_back(Eigen::Triplet<double>(F(i, j1)), F(i, j1, -wij));
                tripletList.push_back(Eigen::Triplet<double>(F(i, j2)), F(i, j2, -wij));
        }
}

//construct matrix from the list
L.setFromTriplets(tripletList.begin(), tripletList.end());
```

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

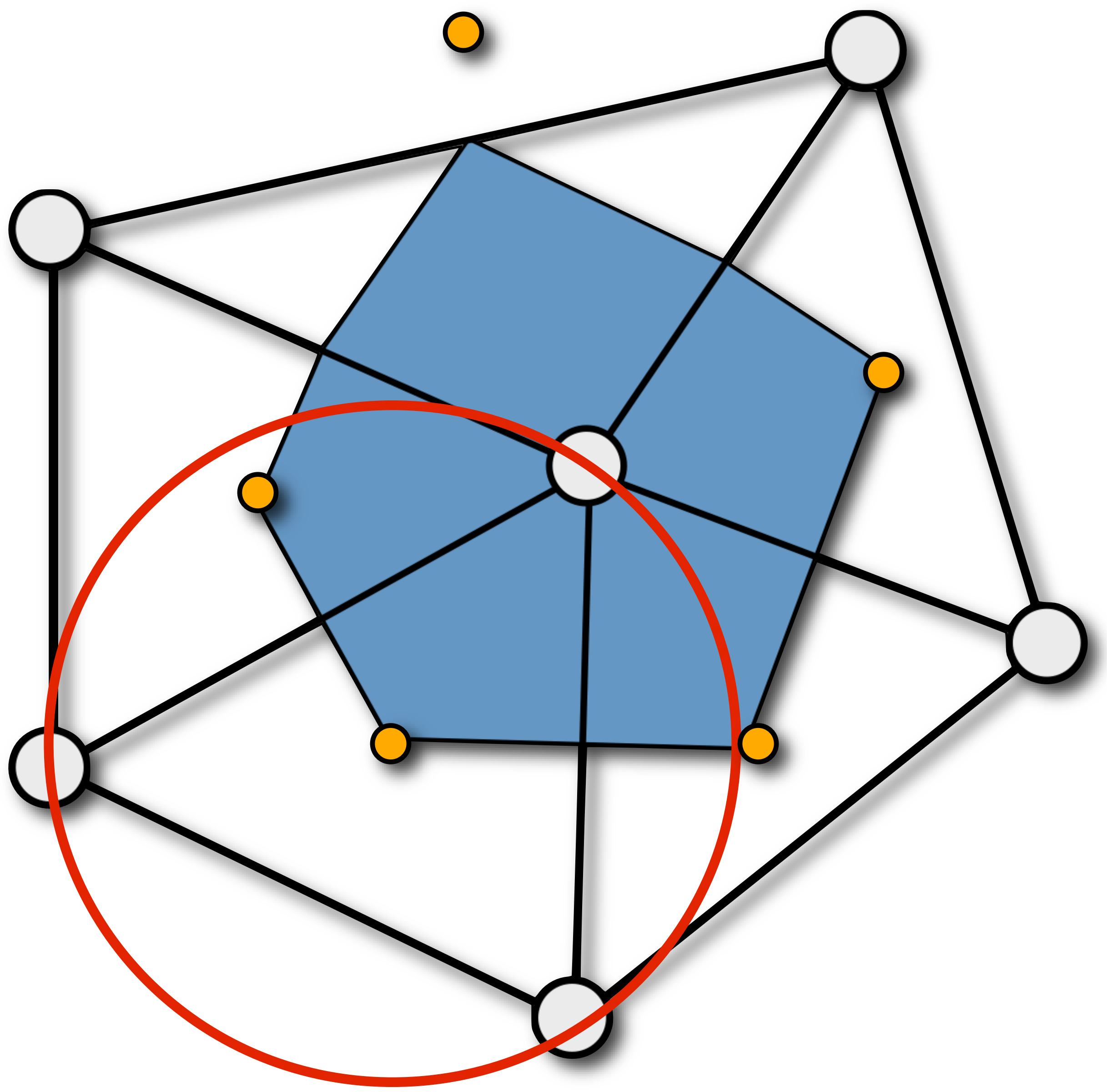
+ Always positive weights

- Heavily connectivity dependent

- Changes if edges are flipped

How to compute A(v)

- **Voronoi area**
  - Connect edge midpoints and triangle circumcenters
  - Sum contributions from incident triangles

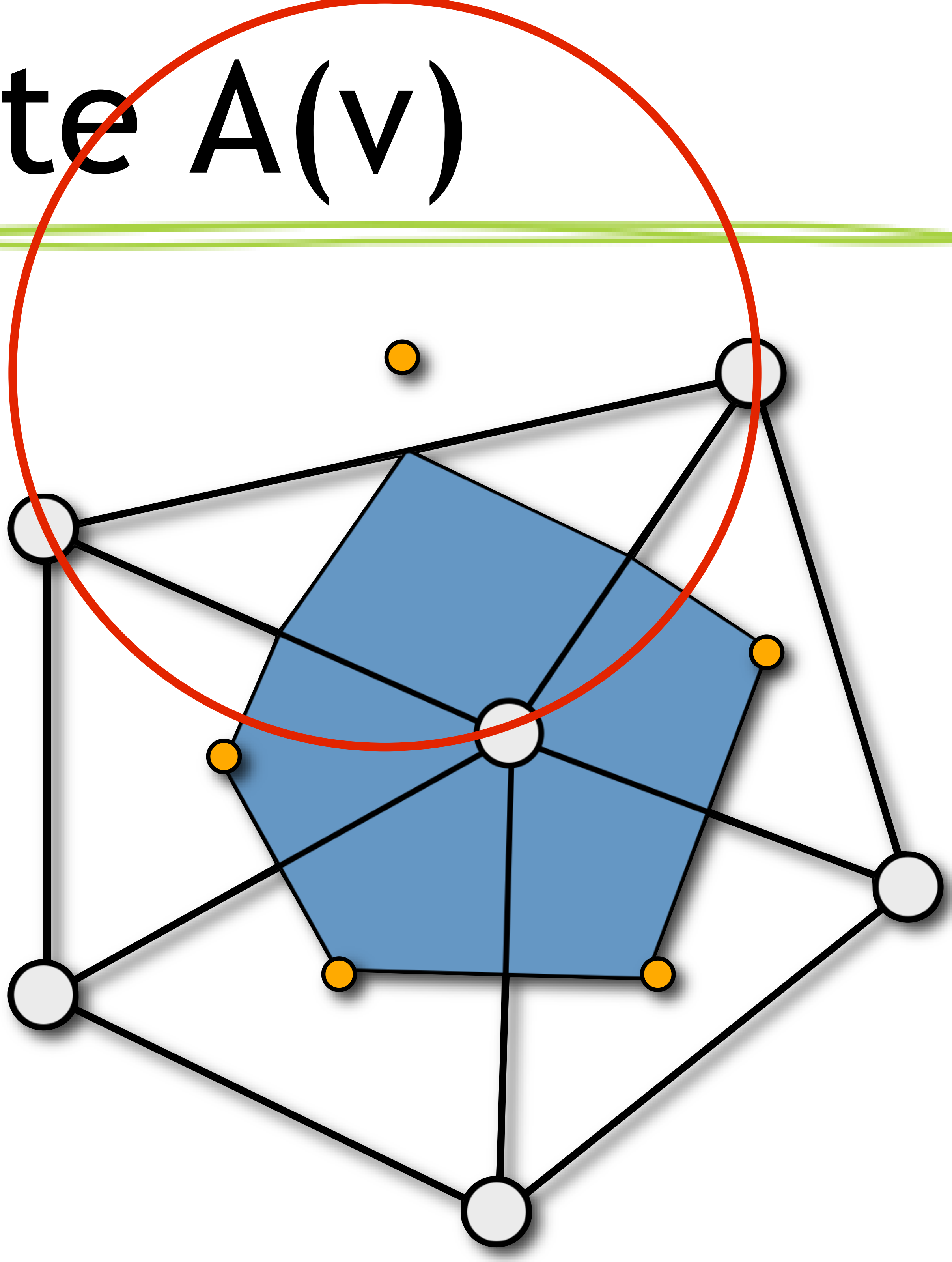


\+ **Only depends on vertex positioning**

\- **More complicated computations**

How to compute A(v)

- **Voronoi area**
  - Connect edge midpoints and triangle circumcenters
  - Sum contributions from incident triangles
  - For obtuse triangles, circumcenter is outside the triangle -> negative areas!



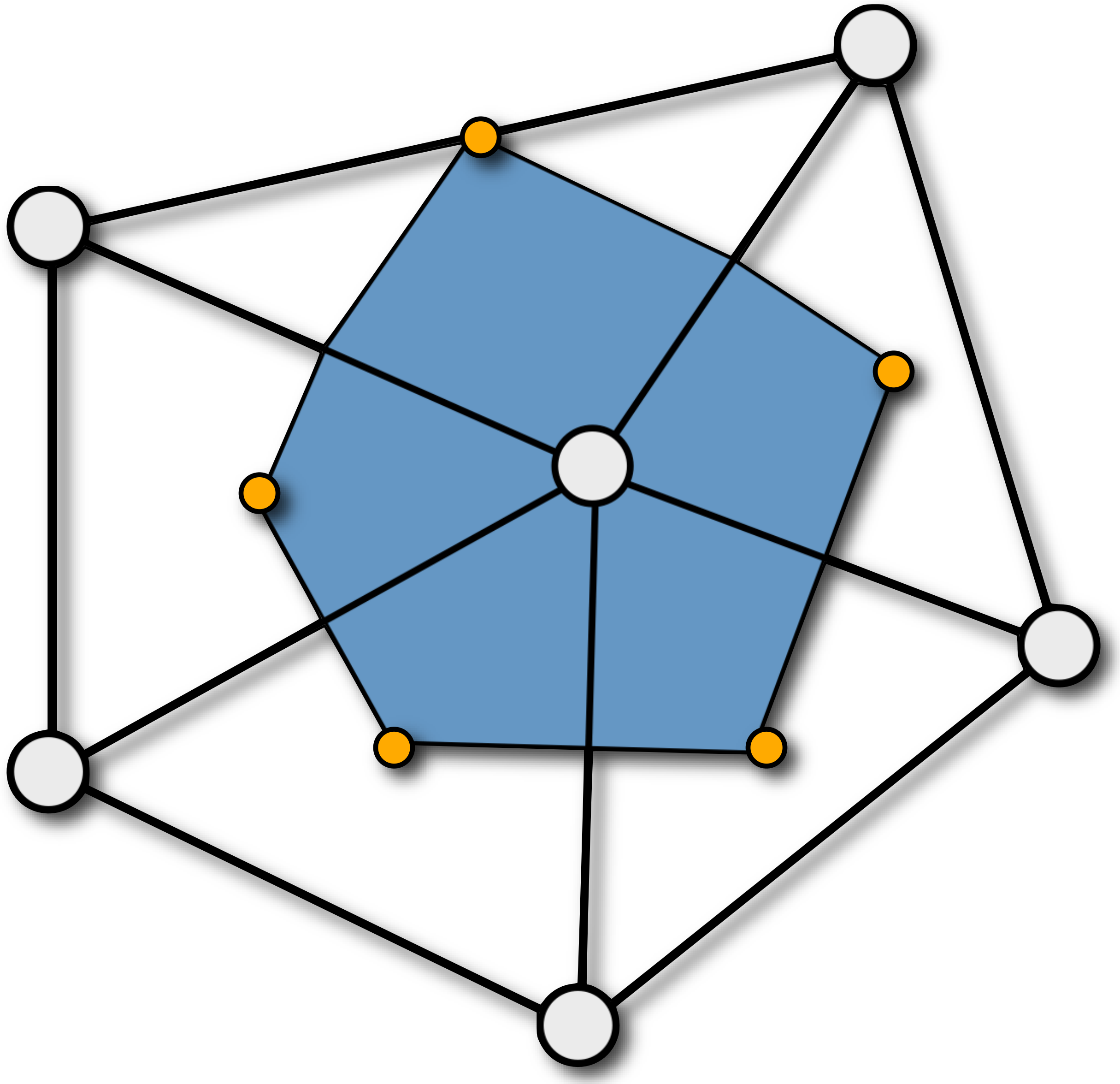
\+ **Only depends on vertex positioning**

\- **More complicated computations**

\- **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 angles
  - Sum contributions from incident triangles



\+ Only depends on vertex positioning

\- More complicated computations

Curvature

- Mean Curvature

$$\Delta_{\mathcal{M}} \mathbf{p} = -2H\mathbf{n}$$

$$H_i = 0.5||(L\mathbf{P})_i|| = \frac{1}{2}(\kappa_1 + \kappa_2)$$

Curvature

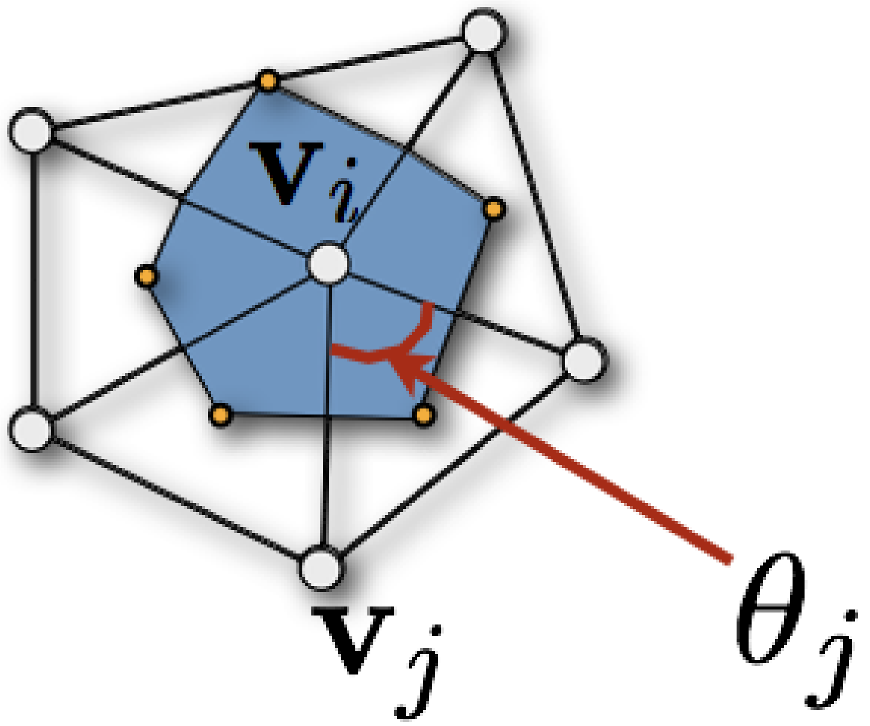
- Mean Curvature

$$\Delta_{\mathcal{M}} \mathbf{p} = -2H\mathbf{n}$$

$$H_i = 0.5||(L\mathbf{P})_i|| = \frac{1}{2}(\kappa_1 + \kappa_2)$$

- Gaussian Curvature

$$G_i = \frac{2\pi - \sum_j \theta_j}{A_i} = \kappa_1 \kappa_2$$



Curvature

- Mean Curvature

$$\Delta_{\mathcal{M}}\mathbf{p} = -2H\mathbf{n}$$

$$H_i = 0.5||(L\mathbf{P})_i|| = \frac{1}{2}(\kappa_1 + \kappa_2)$$

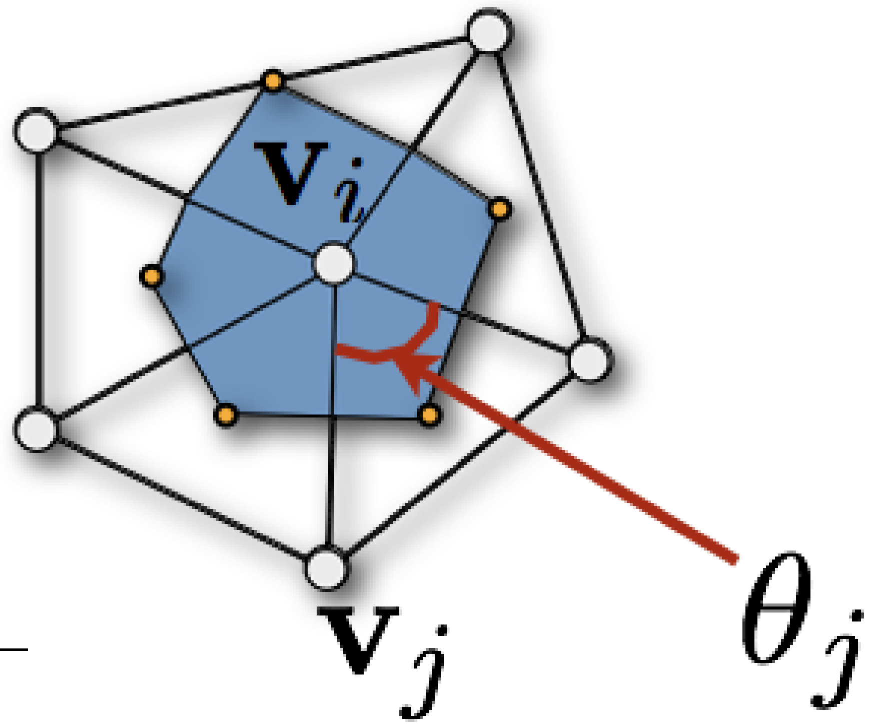
- Gaussian Curvature

$$G_i = \frac{2\pi - \sum_j \theta_j}{A_i} = \kappa_1 \kappa_2$$

- Minimum/Maximum Curvature

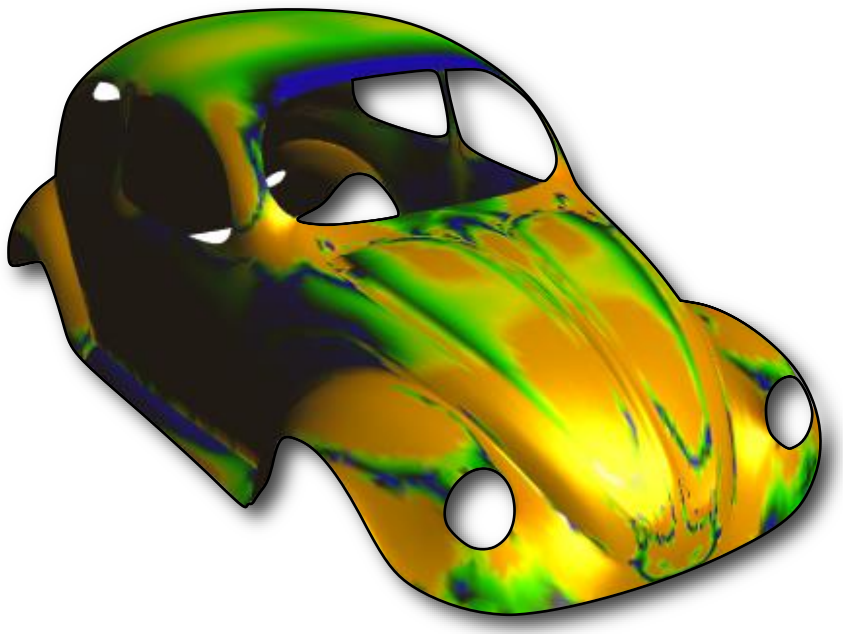
$$\kappa_1 = H_i + \sqrt{H_i^2 - G_i}$$

$$\kappa_2 = H_i - \sqrt{H_i^2 - G_i}$$

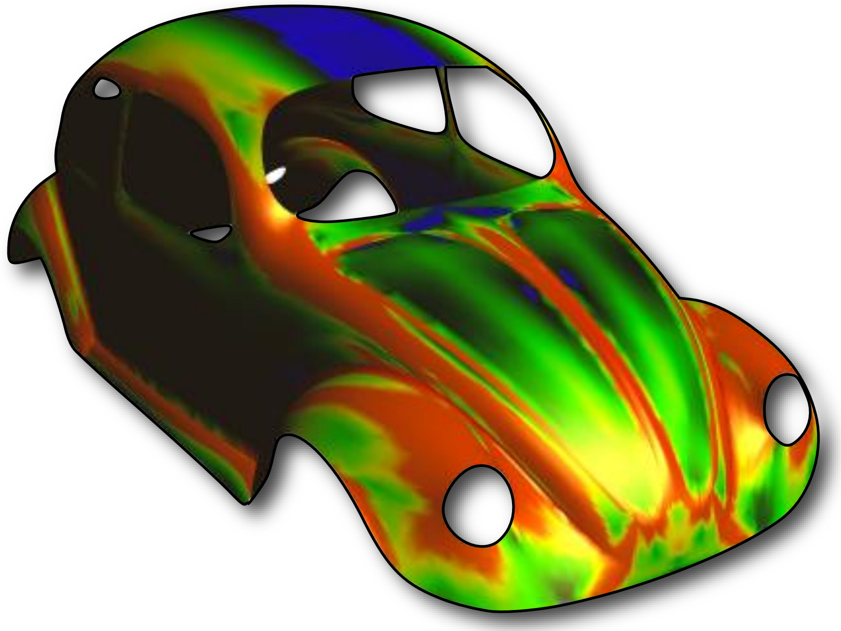


Curvature

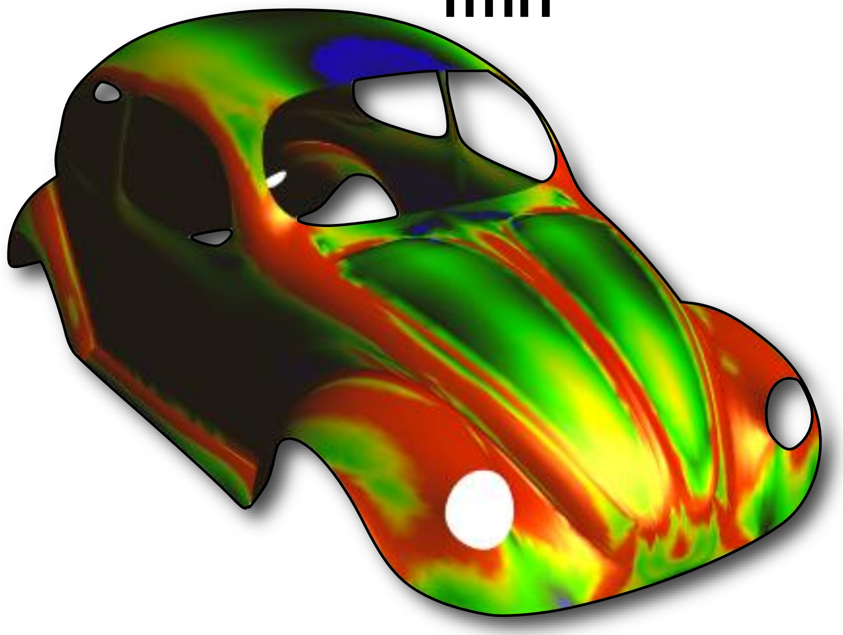
libigl tutorials
#202, #203



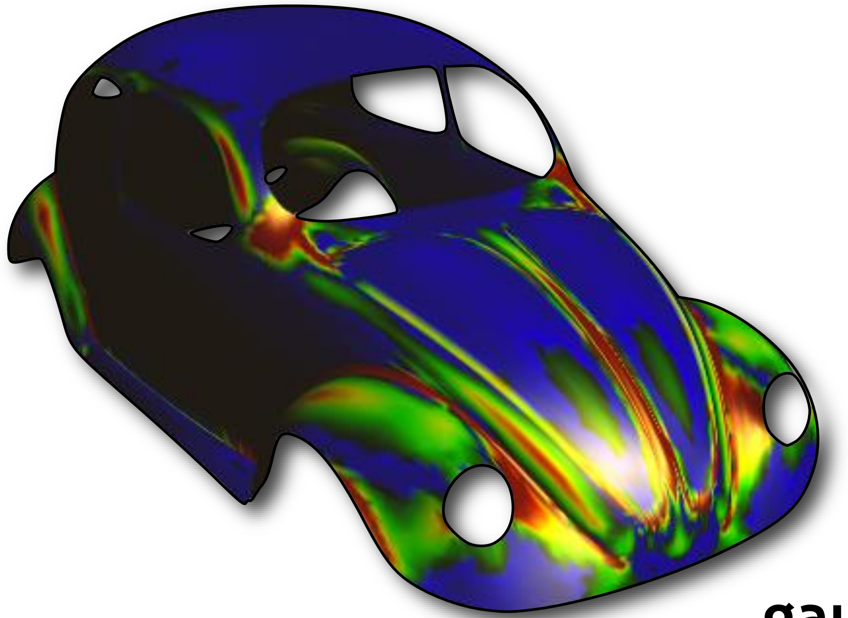
min



max



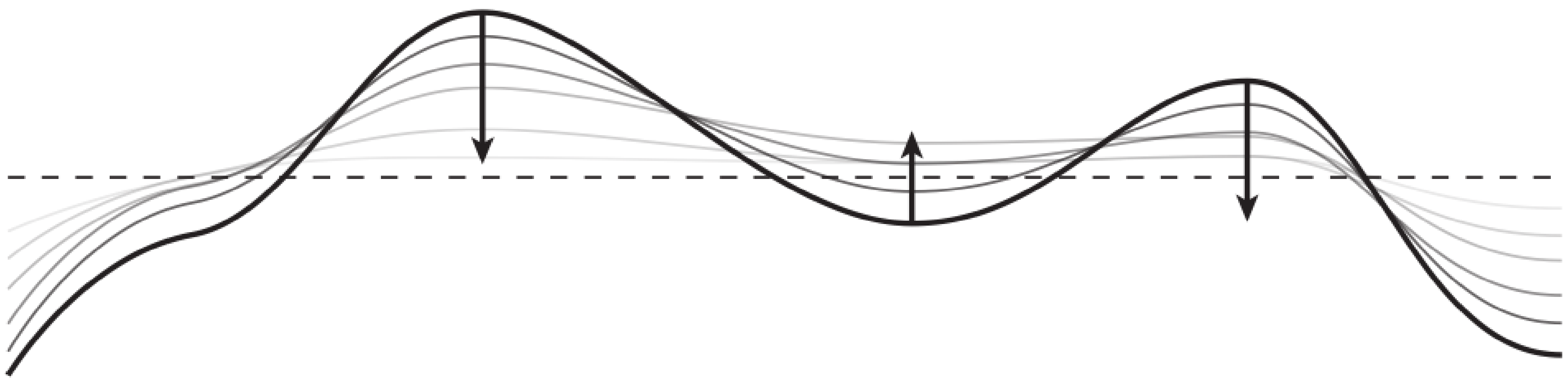
mean



gauss

Smoothing – Mean curvature flow

$$\frac{\partial f}{\partial t} = \Delta f$$



The change of function values is given by the (scaled) Laplacian applied to the function.

Explicit Smoothing

$$\frac{\partial f}{\partial t} = \Delta f$$

Discretize:

$$\frac{\mathbf{X}_{i+1} - \mathbf{X}_i}{t} = L\mathbf{X}_i$$

This scheme is called “explicit Euler integration”.

Explicit Smoothing

$$\frac{\partial f}{\partial t} = \Delta f$$

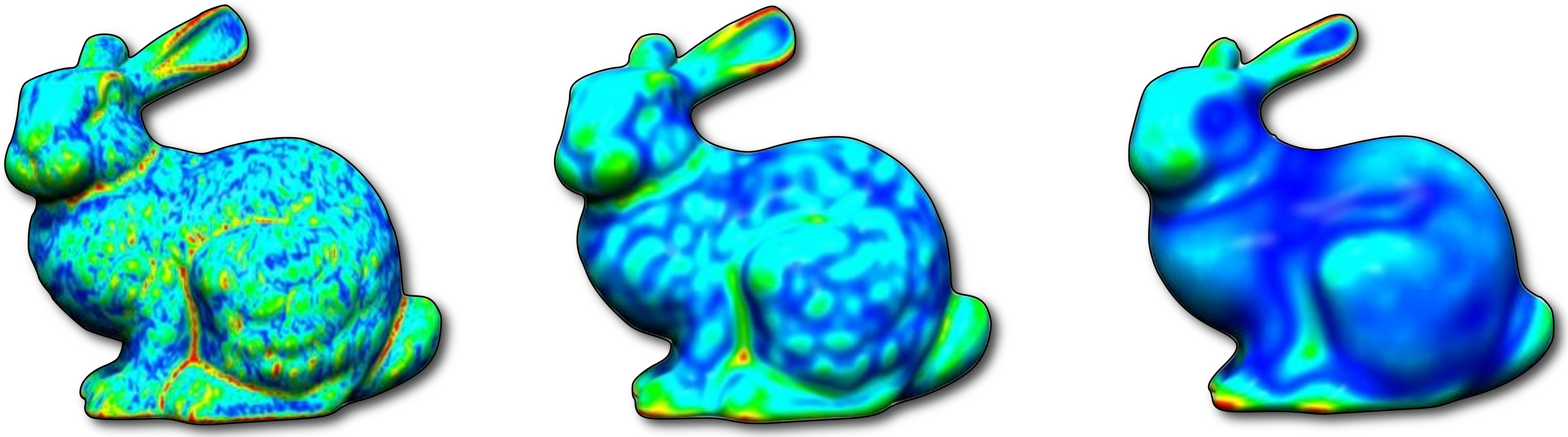
Discretize:

$$\frac{\mathbf{X}_{i+1} - \mathbf{X}_i}{t} = L\mathbf{X}_i$$

$$\Rightarrow \mathbf{X}_{i+1} = (\mathrm{I} + tL)\mathbf{X}_i$$

This scheme is called “explicit Euler integration”.

Explicit Smoothing



Implicit Smoothing

$$\frac{\partial f}{\partial t} = \Delta f$$

Discretize:

$$\frac{\mathbf{X}_{i+1} - \mathbf{X}_i}{t} = L\mathbf{X}_{i+1}$$

Alternative: (semi-)implicit Euler integration.

Implicit Smoothing

$$\frac{\partial f}{\partial t} = \Delta f$$

Discretize:

$$\frac{\mathbf{X}_{i+1} - \mathbf{X}_i}{t} = L\mathbf{X}_{i+1}$$

$$\Rightarrow \quad \mathbf{X}_i = (\mathrm{I} - tL)\mathbf{X}_{i+1}$$

Alternative: (semi-)implicit Euler integration.

Implicit Smoothing

$$\frac{\partial f}{\partial t} = \Delta f$$

Discretize:

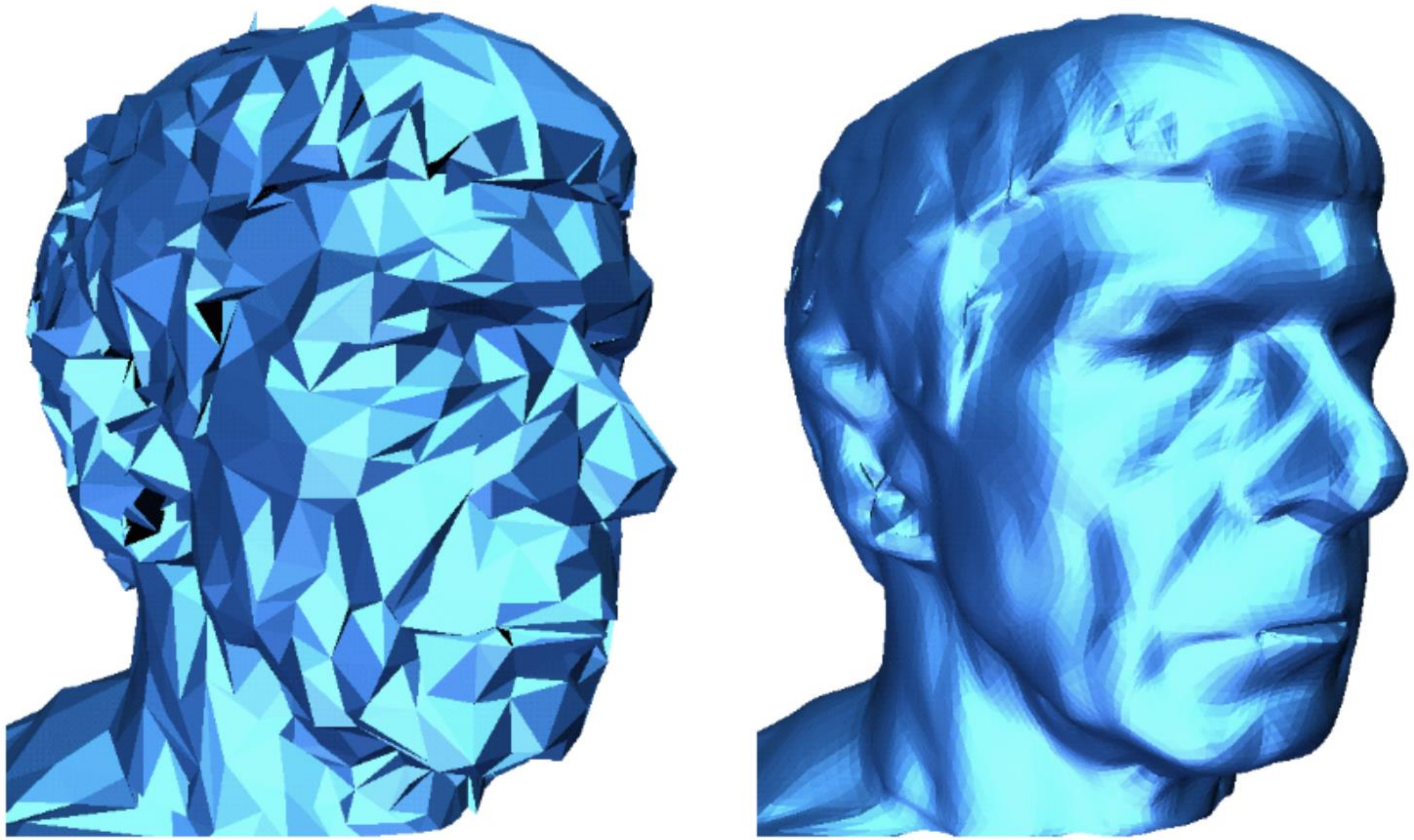
$$\frac{\mathbf{X}_{i+1} - \mathbf{X}_i}{t} = L\mathbf{X}_{i+1}$$

$$\Rightarrow \quad \mathbf{X}_i = (\mathrm{I} - tL)\mathbf{X}_{i+1}$$

$$\Rightarrow \quad \mathbf{X}_{i+1} = (\mathrm{I} - tL)^{-1}\mathbf{X}_i$$

Alternative: (semi-)implicit Euler integration.

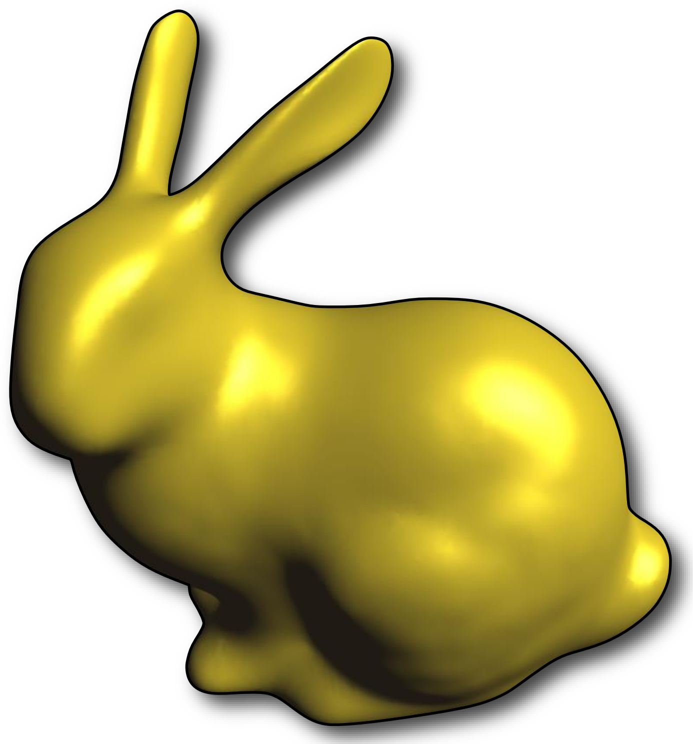
Implicit Smoothing



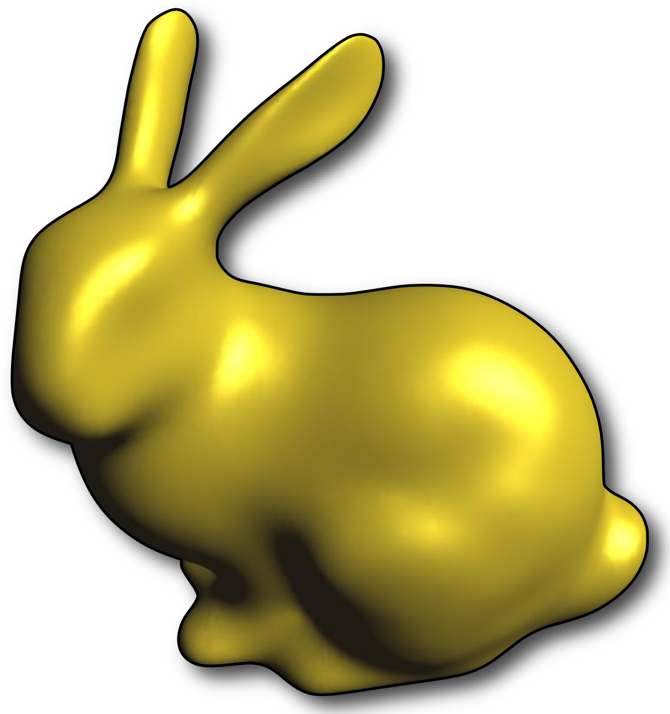
Implicit fairing of irregular meshes using diffusion and curvature flow
[Desbrun et al. 1999]

Implicit Smoothing

original



explicit, 1000 iterations, t = 0.01



implicit, 1 iteration, t = 20

libigl tutorial #205

Implementation

- We need to solve the linear system

$$(\mathrm{I} - tL)\mathbf{X}_{i+1} = \mathbf{X}_i$$

Implementation

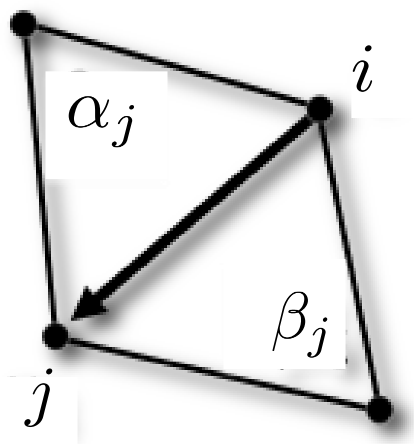
- We need to solve the linear system

$$(\mathrm{I} - tL)\mathbf{X}_{i+1} = \mathbf{X}_i$$

- The system is not symmetric!

$$(L\mathbf{X})_i = \sum_{j \in \mathcal{N}_i} w_{ij}(x_j - x_i)$$

$$w_{ij} = \frac{1}{2A_i}(\cot \alpha_j + \cot \beta_j)$$



Implementation

- We need to solve the linear system

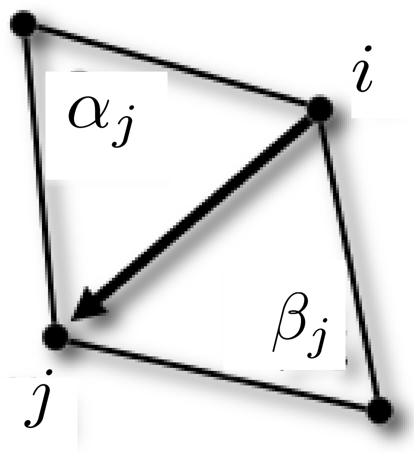
$$(\mathrm{I} - tL)\mathbf{X}_{i+1} = \mathbf{X}_i$$

- The system is not symmetric!

$$M_{ii} = A_i$$

$$(L_c\mathbf{X})_i = \sum_{j \in \mathcal{N}_i} w_{ij}(x_j - x_i)$$

$$w_{ij} = \frac{1}{2}(\cot\alpha_j + \cot\beta_j)$$



Implementation

- We need to solve the linear system

$$(\mathrm{I} - tL)\mathbf{X}_{i+1} = \mathbf{X}_i$$

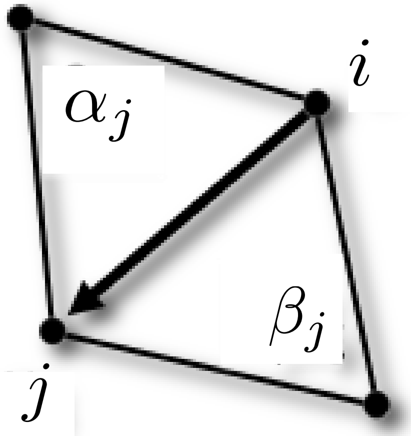
- The system is not symmetric!

$$M_{ii} = A_i$$

$$(L_c\mathbf{X})_i = \sum_{j \in \mathcal{N}_i} w_{ij}(x_j - x_i)$$

$$\boxed{L = M^{-1}L_c}$$

$$w_{ij} = \frac{1}{2}(\cot\alpha_j + \cot\beta_j)$$



Implementation

- We need to solve the linear system

$$(\mathrm{I} - tL)\mathbf{X}_{i+1} = \mathbf{X}_i$$

Implementation

- We need to solve the linear system

$$(\mathrm{I} - tL)\mathbf{X}_{i+1} = \mathbf{X}_i$$
$$(\mathrm{I} - tM^{-1}L_c)\mathbf{X}_{i+1} = \mathbf{X}_i$$

Implementation

- We need to solve the linear system

$$(\mathrm{I} - tL)\mathbf{X}_{i+1} = \mathbf{X}_i$$
$$(\mathrm{I} - tM^{-1}L_c)\mathbf{X}_{i+1} = \mathbf{X}_i$$
$$(M - tL_c)\mathbf{X}_{i+1} = M\mathbf{X}_i$$

Implementation

- We need to solve the linear system

$$
\begin{aligned}
(\mathrm{I} - tL)\mathbf{X}_{i+1} &= \mathbf{X}_i \\
(\mathrm{I} - tM^{-1}L_c)\mathbf{X}_{i+1} &= \mathbf{X}_i \\
(M - tL_c)\mathbf{X}_{i+1} &= M\mathbf{X}_i
\end{aligned}
$$

- The system is ~~not~~ symetric!
- We can use Cholesky Factorization

```
Eigen::SimplicialLDLt<Eigen::SparseMatrix<double>>
```

Implementation

L_c : igl::cotmatrix

M : igl::massmatrix

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Questions?

Thank you!