Geometry Processing

3 Smoothing

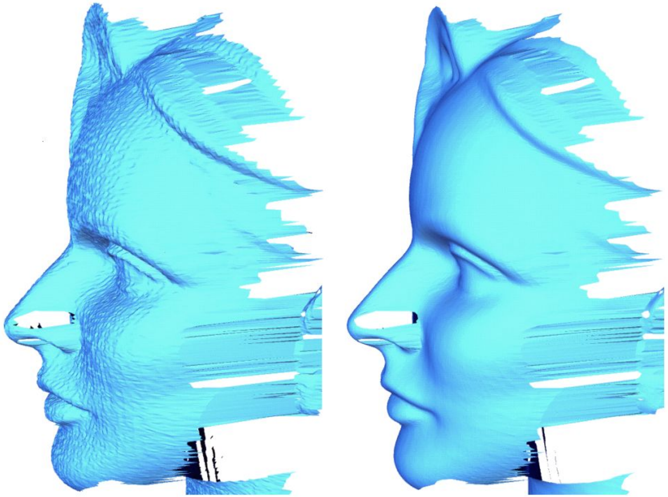
Ludwig-Maximilians-Universität München

Session 3: Smoothing

- Mesh Smoothing
  - Heat Equation and Laplacian Smoothing
  - Laplace and Mass Matrix
  - Linear Solvers
- "*No-free Lunch*"
- Summary
- Discussion

Mesh Smoothing

Motivation: Remove noise (high frequencies) while preserving the shape (the low frequencies)



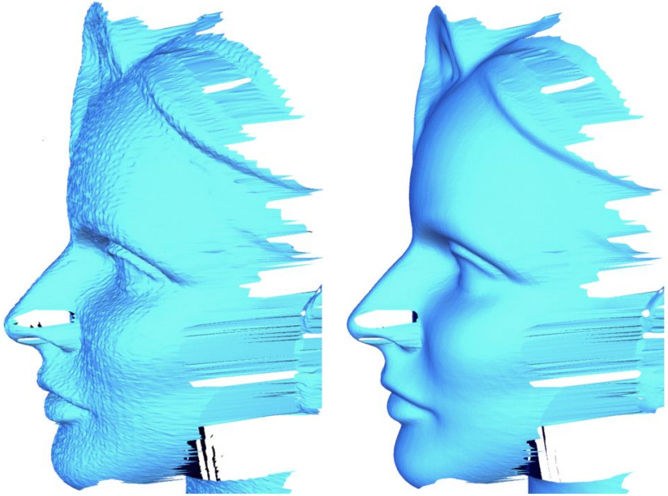
[Desbrun et al. 1999]

Mesh Smoothing

Motivation: Remove noise (high frequencies) while preserving the shape (the low frequencies)

Key questions:

- How to capture ***important patterns***? or what is a feature we want to preserve (very subjective)?
- How to distinguish feature and noise?



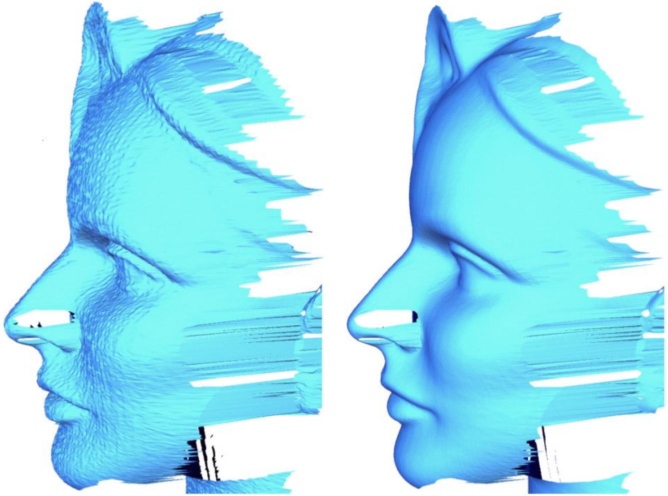
[Desbrun et al. 1999]

Mesh Smoothing

Motivation: Remove noise (high frequencies) while preserving the shape (the low frequencies)

Key questions:

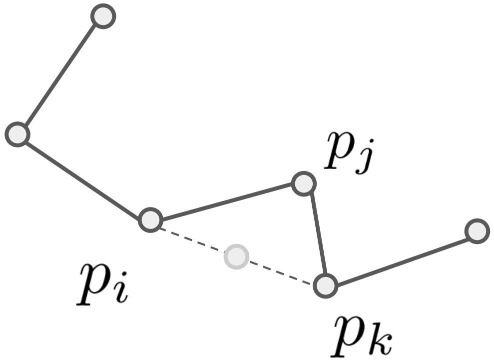
- How to capture ***important patterns***? or what is a feature we want to preserve (very subjective)?
- How to distinguish feature and noise?



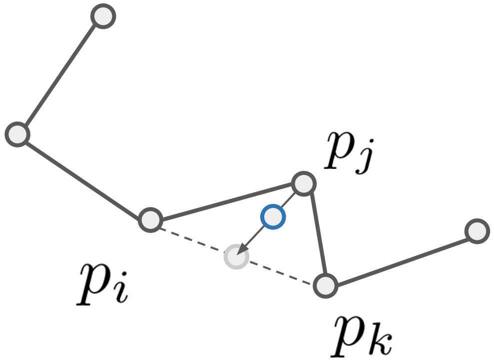
[Desbrun et al. 1999]

Equivalent terminologies: Denoising, filtering, *fairing*

Moving Vertex Position



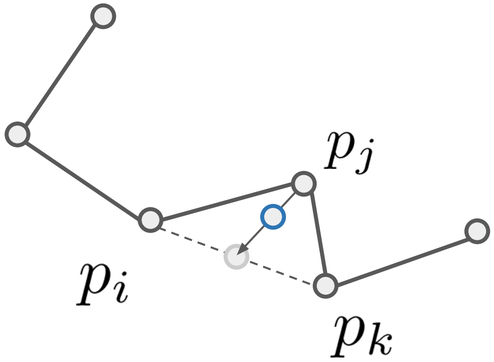
Moving Vertex Position

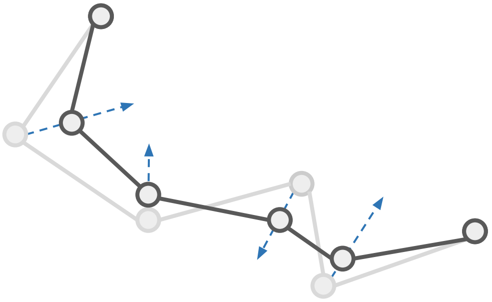


"Move vertex position to the to the midpoint of its neighbors"

$$p'_j = \frac{\frac{p_i + p_k}{2} + p_j}{2}$$

Moving Vertex Position



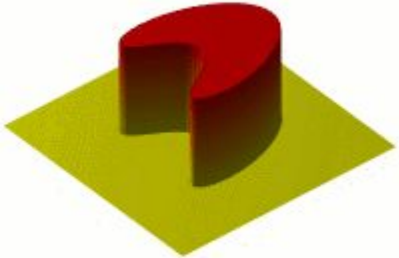


"Move vertex position to the to the midpoint of its neighbors"

$$p_j' = \frac{\frac{p_i + p_k}{2} + p_j}{2}$$

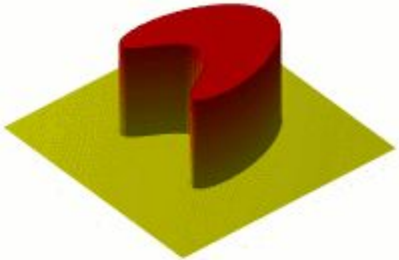
What could be a good direction to move the vertex position?

Insights from Physics: Heat Equation



Laplacian describes the deviation from local average, this matches the physical nature of describing heat diffusion.

Insights from Physics: Heat Equation



Laplacian describes the deviation from local average, this matches

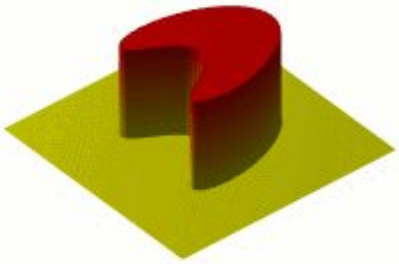
the physical nature of describing heat diffusion.

At each point x in time t , temperature moves towards average of nearby values:

$$\frac{\partial T(x,t)}{\partial t} = \lambda \Delta T(x,t)$$

Equivalent terminology: Diffusion equation

Insights from Physics: Heat Equation



Laplacian describes the deviation from local average, this matches

the physical nature of describing heat diffusion.

At each point *x* in time *t* , temperature moves towards average of nearby values:

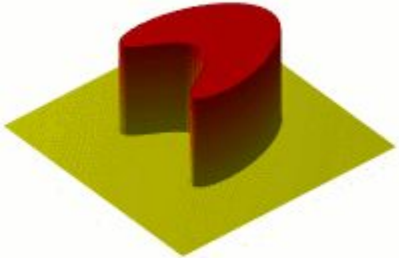
$$\frac{\partial T(x,t)}{\partial t} = \lambda \Delta T(x,t)$$

Equivalent terminology: Diffusion equation

Mesh smoothing can be seen as a time-dependent process along a diffusion flow, such as heat diffusion:

$$\frac{\partial f(x,t)}{\partial t} = \lambda \Delta f(x,t)$$

Insights from Physics: Heat Equation



Laplacian describes the deviation from local average, this matches

the physical nature of describing heat diffusion.

At each point *x* in time *t* , temperature moves towards average of nearby values:

$$\frac{\partial T(x,t)}{\partial t} = \lambda \Delta T(x,t)$$

Equivalent terminology: Diffusion equation

Mesh smoothing can be seen as a time-dependent process along a diffusion flow, such as heat diffusion:

$$\frac{\partial f(x,t)}{\partial t} = \lambda \Delta f(x,t)$$

Remaining question: **How to *discretize* the heat equation *both in space and time* for computation?**

Recall: Laplace-Beltrami Operator

The discrete version of the Laplace operator, of a function at a vertex *i* is given as

$$(\Delta f)_i = w_i \sum_{ij} w_{ij}(f_j - f_i)$$

Recall: Laplace-Beltrami Operator

The discrete version of the Laplace operator, of a function at a vertex *i* is given as

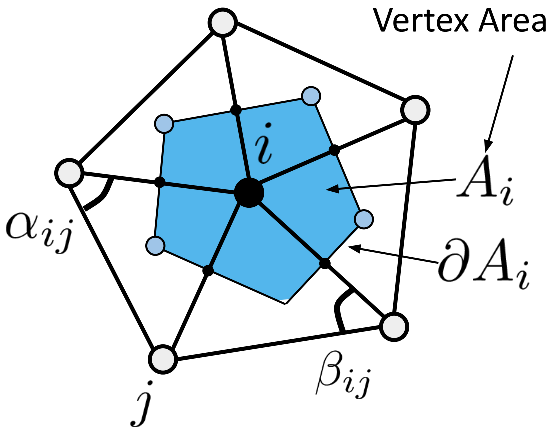
$$(\Delta f)_i = w_i \sum_{ij} w_{ij}(f_j - f_i)$$

The cotan-version is the most widely used discretization of

the Laplace-Beltrami operator for geometry processing:

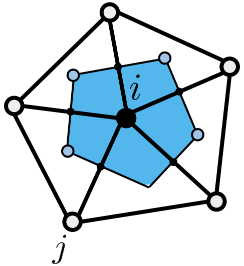
$$(\Delta f)_i = \frac{1}{2A_i} \sum_{ij} (\cot \alpha_{ij} + \cot \beta_{ij})(f_j - f_i)$$

Weights: $w_i = \frac{1}{2A_i}, w_{ij} = \cot \alpha_{ij} + \cot \beta_{ij}$



Laplace Matrix

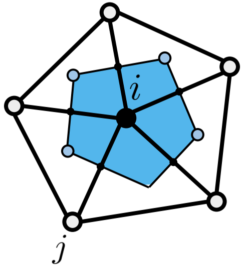
$$(\Delta f)_i = w_i \sum_{ij} w_{ij}(f_j - f_i) \quad \Rightarrow \quad \begin{pmatrix} (\Delta f)_1 \\ (\Delta f)_2 \\ \dots \\ (\Delta f)_n \end{pmatrix} = \mathbf{L} \begin{pmatrix} f_1 \\ f_2 \\ \dots \\ f_n \end{pmatrix}$$



Laplace Matrix

$$(\Delta f)_i = w_i \sum_{ij} w_{ij}(f_j - f_i) \quad \Rightarrow \quad \begin{pmatrix} (\Delta f)_1 \\ (\Delta f)_2 \\ \dots \\ (\Delta f)_n \end{pmatrix} = \mathbf{L} \begin{pmatrix} f_1 \\ f_2 \\ \dots \\ f_n \end{pmatrix}$$

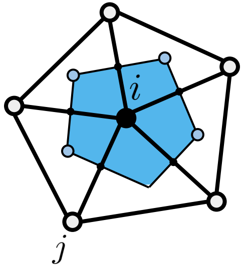
$\mathbf{L}$ is the so called Laplace matrix. What are the elements in the Laplace?



Laplace Matrix

$$(\Delta f)_i = w_i \sum_{ij} w_{ij}(f_j - f_i) \quad \Rightarrow \quad \begin{pmatrix} (\Delta f)_1 \\ (\Delta f)_2 \\ \dots \\ (\Delta f)_n \end{pmatrix} = \mathbf{L} \begin{pmatrix} f_1 \\ f_2 \\ \dots \\ f_n \end{pmatrix}$$

$\mathbf{L}$ is the so called Laplace matrix. What are the elements in the Laplace?



$$\mathbf{L} = \mathbf{DW}$$

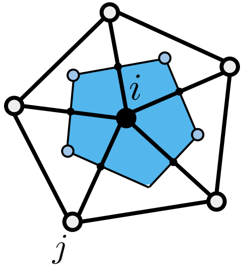
$$\Rightarrow \quad \mathbf{D} = \mathrm{diag}(w_1, \dots, w_n)$$

$$\mathbf{W} = (W_{ij})$$

Laplace Matrix

$$(\Delta f)_i = w_i \sum_{ij} w_{ij}(f_j - f_i) \qquad \Rightarrow \qquad \begin{pmatrix} (\Delta f)_1 \\ (\Delta f)_2 \\ \dots \\ (\Delta f)_n \end{pmatrix} = \mathbf{L} \begin{pmatrix} f_1 \\ f_2 \\ \dots \\ f_n \end{pmatrix}$$

$\mathbf{L}$ is the so called Laplace matrix. What are the elements in the Laplace?



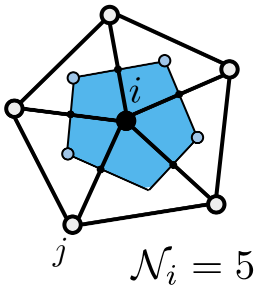
$$\mathbf{L} = \mathbf{DW}$$

$$\Rightarrow \quad \mathbf{D} = \mathrm{diag}(w_1, ..., w_n)$$

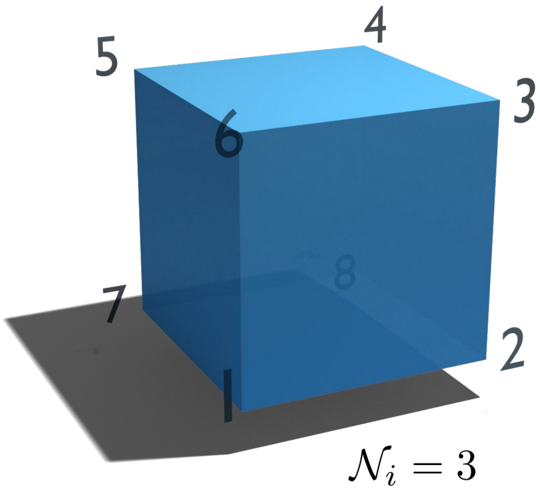
$$\mathbf{W} = (W_{ij}) \qquad \Rightarrow W_{ij} = \begin{cases} -\sum_{ik} w_{ik}, & \text{if } i = j \\ w_{ij}, & \text{if } j \text{ is a neighbor of } i \\ 0, & \text{otherwise} \end{cases}$$

Example: *Uniform Laplacian*

Let $w_i = \dfrac{1}{\mathcal{N}_i}, w_{ij} = 1 \Rightarrow (\Delta f)_i = \dfrac{1}{\mathcal{N}_i}\sum_{ij}(f_j - f_i)$

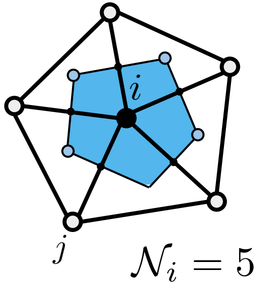


For the given cube, and (randomly) assign indices to each vertex, then the uniform Laplacian of vertex **1** is:

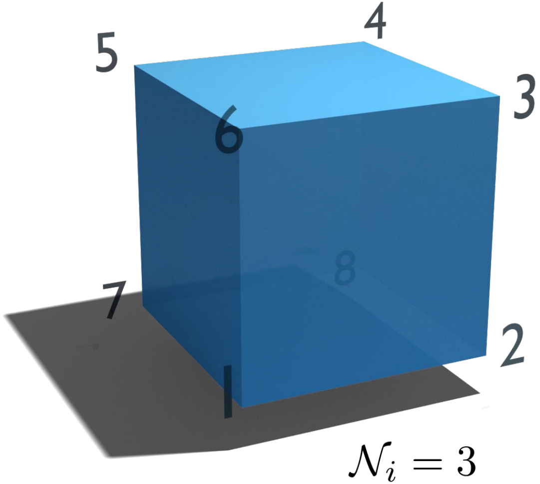


Example: *Uniform Laplacian*

Let $w_i = \frac{1}{\mathcal{N}_i}, w_{ij} = 1 \Rightarrow (\Delta f)_i = \frac{1}{\mathcal{N}_i} \sum_{ij} (f_j - f_i)$

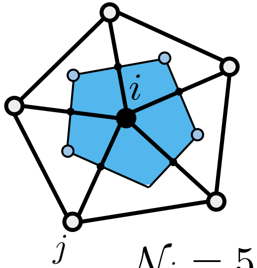


For the given cube, and (randomly) assign indices to each vertex, then the uniform Laplacian of vertex **1** is:



$$
\begin{aligned}
(\Delta f)_1 &= \frac{1}{3}[(f_2 - f_1) + (f_6 - f_1) + (f_7 - f_1)] \\
&= \frac{1}{3}(f_2 + f_6 + f_7 - 3f_1) \\
&= \frac{1}{3}\begin{pmatrix} -3 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \\ f_7 \\ f_8 \end{pmatrix}
\end{aligned}
$$

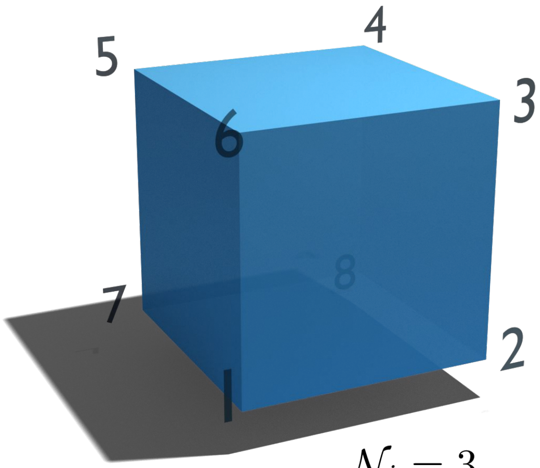
Example: *Uniform Laplacian*



$\mathcal{N}_i = 5$

Let $w_i = \dfrac{1}{\mathcal{N}_i}, w_{ij} = 1 \Rightarrow (\Delta f)_i = \dfrac{1}{\mathcal{N}_i}\sum_{ij}(f_j - f_i)$

For the given cube, and (randomly) assign indices to each vertex, then the uniform Laplacian is:



$\mathcal{N}_i = 3$

$$
\mathbf{L} = \mathbf{DW} = \begin{pmatrix}
\frac{1}{3} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & \frac{1}{3} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \frac{1}{3} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & \frac{1}{3} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{1}{3} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \frac{1}{3} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{3} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{3}
\end{pmatrix}
\begin{pmatrix}
-3 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\
1 & -3 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 1 & -3 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & -3 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & -3 & 1 & 1 & 0 \\
1 & 0 & 1 & 0 & 1 & -3 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & -3 & 1 \\
0 & 1 & 0 & 1 & 0 & 0 & 1 & -3
\end{pmatrix}
$$

$$
= \begin{pmatrix}
3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 3 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 3 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 3 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 3
\end{pmatrix}^{-1}
\begin{pmatrix}
-3 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\
1 & -3 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 1 & -3 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & -3 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & -3 & 1 & 1 & 0 \\
1 & 0 & 1 & 0 & 1 & -3 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & -3 & 1 \\
0 & 1 & 0 & 1 & 0 & 0 & 1 & -3
\end{pmatrix}
$$

$$
= \mathbf{M}^{-1}\mathbf{W}
$$

Mass Matrix

Laplace matrix contains two parts: The inverse of **mass matrix** multiply a weight matrix:

$$\mathbf{L} = \mathbf{M}^{-1}\mathbf{W}$$

Mass Matrix

Laplace matrix contains two parts: The inverse of **mass matrix** multiply a weight matrix:

$$\mathbf{L} = \mathbf{M}^{-1}\mathbf{W}$$

For the cotan version of Laplace matrix, the mass matrix is a diagonal matrix consists of vertex areas:

$$(\Delta f)_i = \frac{1}{2A_i}\sum_{ij}(\cot\alpha_{ij} + \cot\beta_{ij})(f_j - f_i)$$

Mass Matrix

Laplace matrix contains two parts: The inverse of **mass matrix** multiply a weight matrix:

$$\mathbf{L} = \mathbf{M}^{-1}\mathbf{W}$$

For the cotan version of Laplace matrix, the mass matrix is a diagonal matrix consists of vertex areas:

$$(\Delta f)_i = \frac{1}{2A_i}\sum_{ij}(\cot\alpha_{ij} + \cot\beta_{ij})(f_j - f_i)$$

$$w_i = \frac{1}{A_i}$$
$$w_{ij} = \frac{1}{2}(\cot\alpha_{ij} + \cot\beta_{ij})$$

Mass Matrix

Laplace matrix contains two parts: The inverse of **mass matrix** multiply a weight matrix:

$$\mathbf{L} = \mathbf{M}^{-1}\mathbf{W}$$

For the cotan version of Laplace matrix, the mass matrix is a diagonal matrix consists of vertex areas:

$$(\Delta f)_i = \frac{1}{2A_i}\sum_{ij}(\cot\alpha_{ij} + \cot\beta_{ij})(f_j - f_i)$$

$$\begin{aligned} w_i &= \frac{1}{A_i} \\ w_{ij} &= \frac{1}{2}(\cot\alpha_{ij} + \cot\beta_{ij}) \end{aligned} \quad \Rightarrow \quad \mathbf{L} = \begin{pmatrix} A_1 & & \\ & \ddots & \\ & & A_n \end{pmatrix}^{-1} \mathbf{W}$$

Spatial Discretization: Laplace-Beltrami Operator

Basic idea: Replace the Laplacian operator using the discretized version, i.e. the Laplace-Beltrami Operator

$$\frac{\partial f(x,t)}{\partial t} = \lambda \Delta f(x,t) \quad \Rightarrow \quad \frac{\partial f(v_i,t)}{\partial t} = \lambda \Delta f(v_i,t)$$

Spatial Discretization: Laplace-Beltrami Operator

Basic idea: Replace the Laplacian operator using the discretized version, i.e. the Laplace-Beltrami Operator

$$\frac{\partial f(x,t)}{\partial t} = \lambda \Delta f(x,t) \quad \Rightarrow \quad \frac{\partial f(v_i,t)}{\partial t} = \lambda \Delta f(v_i,t)$$

$$\Rightarrow \quad \frac{\partial \mathbf{f}(t)}{\partial t} = \lambda \mathbf{L} \mathbf{f}(t)$$

Laplace Matrix

Spatial Discretization: Laplace-Beltrami Operator

Basic idea: Replace the Laplacian operator using the discretized version, i.e. the Laplace-Beltrami Operator

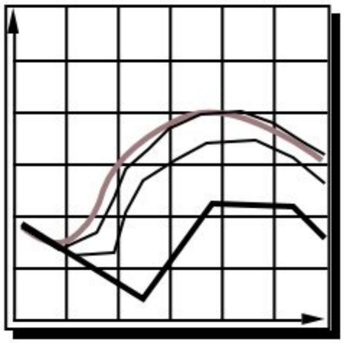
$$\frac{\partial f(x,t)}{\partial t} = \lambda \Delta f(x,t) \quad \Rightarrow \quad \frac{\partial f(v_i,t)}{\partial t} = \lambda \Delta f(v_i,t)$$

$$\Rightarrow \quad \frac{\partial \mathbf{f}(t)}{\partial t} = \lambda \mathbf{L} \mathbf{f}(t)$$

Laplace Matrix

Remaining question: ***How to deal with temporal discretization?***

Euler's Method

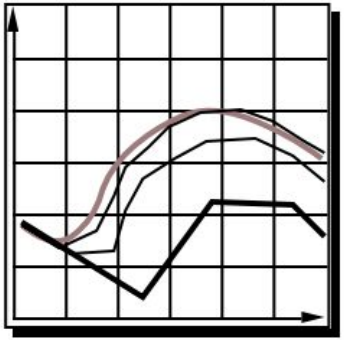


Euler's Method(a.k.a. Forward Euler, Explicit Euler)

$$\mathbf{f}(t+h) = \mathbf{f}(t) + h\frac{\partial \mathbf{f}(t)}{\partial t}$$

Very simple iterative method

Euler's Method

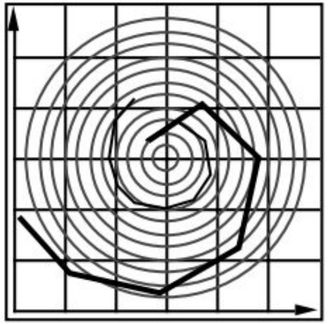


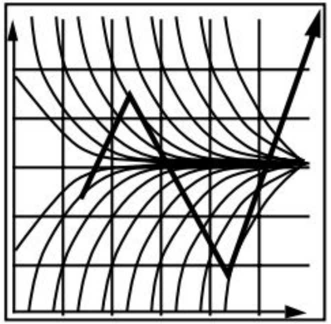
Euler's Method(a.k.a. Forward Euler, Explicit Euler)

$$\mathbf{f}(t+h) = \mathbf{f}(t) + h\frac{\partial \mathbf{f}(t)}{\partial t}$$

Very simple iterative method, but with two key issues:

- Inaccurate as time step increases
- Unstable and leads the simulation to diverge





Temporal Discretization: Implicit Euler Method

Key idea: use derivatives in the future, for the current step

Temporal Discretization: Implicit Euler Method

Key idea: use derivatives in the future, for the current step

The heat equation via a sufficiently small time step h:

$$\frac{\partial \mathbf{f}(t)}{\partial t} = \frac{\mathbf{f}(t+h) - \mathbf{f}(t)}{h} = \lambda \mathbf{L} \mathbf{f}(t) \Rightarrow \mathbf{f}(t+h) = \mathbf{f}(t) + \lambda h \mathbf{L} \mathbf{f}(t)$$

Temporal Discretization: Implicit Euler Method

Key idea: use derivatives in the future, for the current step

The heat equation via a sufficiently small time step h:

$$\frac{\partial \mathbf{f}(t)}{\partial t} = \frac{\mathbf{f}(t+h) - \mathbf{f}(t)}{h} = \lambda \mathbf{L}\mathbf{f}(t) \Rightarrow \mathbf{f}(t+h) = \mathbf{f}(t) + \lambda h \mathbf{L}\mathbf{f}(t)$$

$$\mathbf{f}(t+h) = (\mathbf{I} + \lambda h \mathbf{L})\mathbf{f}(t) \quad \Rightarrow \quad (\mathbf{I} - h\lambda\mathbf{L})\mathbf{f}(t+h) = \mathbf{f}(t)$$

(Forward Euler, fast) **(Backward Euler, stable)**

Temporal Discretization: Implicit Euler Method

Key idea: use derivatives in the future, for the current step

The heat equation via a sufficiently small time step h:

$$\frac{\partial \mathbf{f}(t)}{\partial t} = \frac{\mathbf{f}(t+h) - \mathbf{f}(t)}{h} = \lambda \mathbf{L}\mathbf{f}(t) \Rightarrow \mathbf{f}(t+h) = \mathbf{f}(t) + \lambda h \mathbf{L}\mathbf{f}(t)$$

$$\mathbf{f}(t+h) = (\mathbf{I} + \lambda h \mathbf{L})\mathbf{f}(t) \qquad \Rightarrow \qquad (\mathbf{I} - h\lambda\mathbf{L})\mathbf{f}(t+h) = \mathbf{f}(t)$$

(Forward Euler, fast) **(Backward Euler, stable)**

Therefore, Laplacian smoothing is to solve such a linear system:

$$(\mathbf{I} - h\lambda\mathbf{L})\mathbf{f}(t+h) = \mathbf{f}(t)$$

Laplacian Smoothing

Key idea: move vertex along the mean curvature flow ($\Delta f = 2HN$)

Laplacian Smoothing

Key idea: move vertex along the mean curvature flow ($\Delta f = 2HN$)

vertices move along the normal direction by an amount determined by the mean curvature

$$(\mathbf{I} - h\lambda\mathbf{L})\mathbf{f}(t+h) = \mathbf{f}(t)$$

Laplacian Smoothing

Key idea: move vertex along the mean curvature flow ($\Delta f = 2HN$)

vertices move along the normal direction by an amount determined by the mean curvature

$$(\mathbf{I} - h\lambda\mathbf{L})\mathbf{f}(t+h) = \mathbf{f}(t)$$

$$\Rightarrow \; (\mathbf{M} - h\lambda\mathbf{W})\mathbf{f}(t+h) = \mathbf{Mf}(t)$$

Mass Matrix **(Cotan) Weight Matrix**

Laplacian Smoothing

Key idea: move vertex along the mean curvature flow ($\Delta f = 2HN$)

vertices move along the normal direction by an amount determined by the mean curvature

$$(\mathbf{I} - h\lambda\mathbf{L})\mathbf{f}(t+h) = \mathbf{f}(t)$$

$$\Rightarrow \ (\mathbf{M} - h\lambda\mathbf{W})\mathbf{f}(t+h) = \mathbf{M}\mathbf{f}(t)$$

Mass Matrix **(Cotan) Weight Matrix**

Generally, Laplacian smoothing applies to an arbitrary function, one can manipulate not only positions but also other quantities, such as colors, normals (e.g. smooth normal, then recover the vertex)

Implementation Side: Cholesky Solver

How to solve the linear system efficiently? Many linear solvers exist, the *best choice depends on the matrix property.*

An **efficient** approach for the laplacian smoothing is to use ***the Cholesky solver***.

Implementation Side: Cholesky Solver

How to solve the linear system efficiently? Many linear solvers exist, the *best choice depends on the matrix property.*

An **efficient** approach for the laplacian smoothing is to use ***the Cholesky solver***.

Similar to what you learned about LU decomposition,

Find a decomposition such that $\mathbf{A} = \mathbf{L}\mathbf{L}^\top$, where $\mathbf{L}$ is a lower triangular matrix.

Implementation Side: Cholesky Solver

How to solve the linear system efficiently? Many linear solvers exist, the *best choice depends on the matrix property.*

An **efficient** approach for the laplacian smoothing is to use ***the Cholesky solver***.

Similar to what you learned about LU decomposition,

Find a decomposition such that $\mathbf{A} = \mathbf{L}\mathbf{L}^\top$, where $\mathbf{L}$ is a lower triangular matrix.

With the factorization, we can solve the following equation first:

$$\mathbf{L}\mathbf{y} = \mathbf{b}$$

(easy, why?)

Implementation Side: Cholesky Solver

How to solve the linear system efficiently? Many linear solvers exist, the *best choice depends on the matrix property.*

An **efficient** approach for the laplacian smoothing is to use ***the Cholesky solver***.

Similar to what you learned about LU decomposition,

Find a decomposition such that $\mathbf{A} = \mathbf{L}\mathbf{L}^\top$, where $\mathbf{L}$ is a lower triangular matrix.

With the factorization, we can solve the following equation first:

$$\mathbf{L}\mathbf{y} = \mathbf{b}$$

(easy, why?)

Then we have

$$\begin{aligned} \mathbf{A}\mathbf{x} = \mathbf{b} &\Rightarrow \mathbf{L}\mathbf{L}^\top = \mathbf{b} \\ &\Rightarrow \mathbf{L}\mathbf{L}^\top\mathbf{x} = \mathbf{L}\mathbf{y} \\ &\Rightarrow \mathbf{L}^\top\mathbf{x} = \mathbf{y} \end{aligned}$$

(easy, why?)

Comparison: Direct Solver v.s. Cholesky Solver

```python
import numpy as np
from scipy.linalg import solve_triangular

def prepare_problem(size):
    x = np.random.random((size, 1))
    H = np.random.random((size, size))
    A = H@H.T
    b = A@x
    return A, b

def direct_solver(A, b):
    x_hat = np.linalg.solve(A, b)

def cholesky_solver(A, b):
    L = np.linalg.cholesky(A)
    y = solve_triangular(L, b, lower=True)
    x_hat_cho = solve_triangular(L.T, y, lower=False)
```

Comparison: Direct Solver v.s. Cholesky Solver

```python
import numpy as np
from scipy.linalg import solve_triangular

def prepare_problem(size):
    x = np.random.random((size, 1))
    H = np.random.random((size, size))
    A = H@H.T
    b = A@x
    return A, b

def direct_solver(A, b):
    x_hat = np.linalg.solve(A, b)

def cholesky_solver(A, b):
    L = np.linalg.cholesky(A)
    y = solve_triangular(L, b, lower=True)
    x_hat_cho = solve_triangular(L.T, y, lower=False)
```

Comparison: Direct Solver v.s. Cholesky Solver

```python
import numpy as np
from scipy.linalg import solve_triangular

def prepare_problem(size):
   x = np.random.random((size, 1))
   H = np.random.random((size, size))
   A = H@H.T
   b = A@x
   return A, b

def direct_solver(A, b):
   x_hat = np.linalg.solve(A, b)

def cholesky_solver(A, b):
   L = np.linalg.cholesky(A)
   y = solve_triangular(L, b, lower=True)
   x_hat_cho = solve_triangular(L.T, y, lower=False)
```

Q: Why Cholesky solver?

Cholesky solver utilizes the property of symmetric (semi-)positive definiteness.

Session 3: Smoothing

- Mesh Smoothing
  - Heat Equation and Laplacian Smoothing
  - Laplace and Mass Matrix
  - Linear Solvers
- "*No-free Lunch*"
- Summary
- Discussion

Desired Properties for Discrete Laplacians [Wardetzky et al. 2007]

Property (in smooth settings)	Condition (in discrete settings)	Reasons (will see more in future sessions)
Symmetry (SYM)	$w_{ij} = w_{ji}$	Real symmetric matrices exhibit real eigenvalues and orthogonal eigenvectors
Locality (LOC)	$w_{ij} = 0$ if *i* and *j* do not share an edge	Smooth Laplacians govern diffusion process
Linear precision (LIN)	$(\mathbf{Lf})_i = 0$ when vertices are in a plane	Expect to remove noise only but not to introduce vertex drift
Positive weights (POS)	$w_{ij} \geq 0$, whenever $i \neq j$	Assures diffusion process travel from higher potential region to lower ones

Desired Properties for Discrete Laplacians [Wardetzky et al. 2007]

Property	Condition	Reasons (will see more in future sessions)
Symmetry (SYM)	$w_{ij} = w_{ji}$	Real symmetric matrices exhibit real eigenvalues and orthogonal eigenvectors
Locality (LOC)	$w_{ij} = 0$ if *i* and *j* do not share an edge	Smooth Laplacians govern diffusion process
Linear precision (LIN)	$(\mathbf{Lf})_i = 0$ when vertices are in a plane	Expect to remove noise only but not to introduce vertex drift
Positive weights (POS)	$w_{ij} \geq 0$, whenever $i \neq j$	Assures diffusion process travel from higher potential region to lower ones

The perfect/ideal case: Positive Semi-definite (PSD)

Sufficient condition: SYM+POS $\longrightarrow$ PSD

Uniform Laplacian: Revisit

$$(\Delta f)_i = w_i \sum_{ij} w_{ij}(f_j - f_i) \qquad w_i = \frac{1}{\mathcal{N}_i}, w_{ij} = 1$$

SYM: ✔

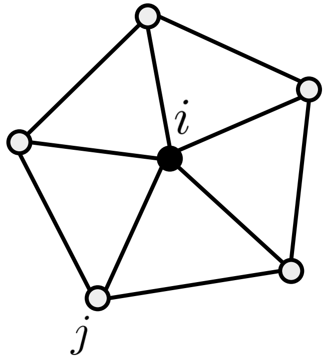
LOC: ✔

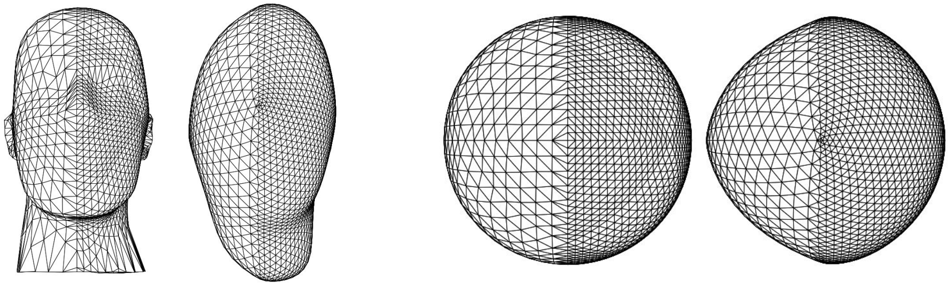
LIN: ❌

POS: ✔

Uniform Laplacian **does not encode the spatial quantity**

but only connectivity in the weights (think about Graph NN)





[Desbrun et al. 1999]

Cotan Laplacian: Revisit

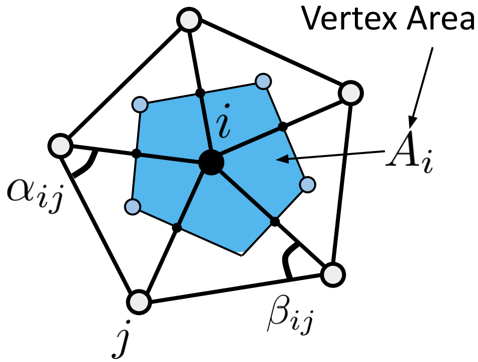
$$(\Delta f)_i = w_i \sum_{ij} w_{ij}(f_j - f_i) \qquad w_i = \frac{1}{2A_i}, w_{ij} = \cot \alpha_{ij} + \cot \beta_{ij}$$

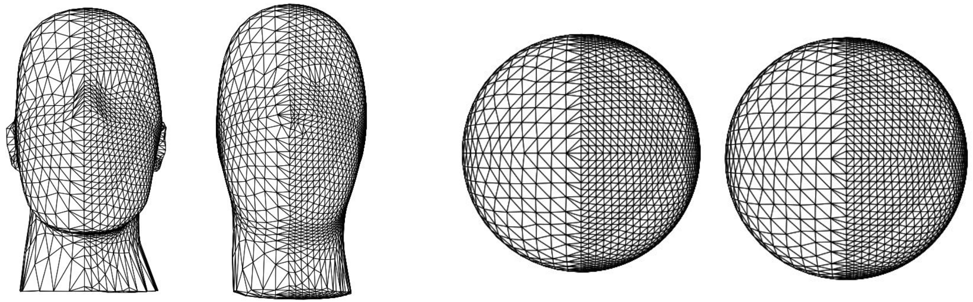
SYM: ✔

LOC: ✔

LIN: ✔

POS: ❌ $\alpha_{ij} + \beta_{ij} > \pi \Rightarrow \cot \alpha_{ij} + \cot \beta_{ij} < 0$





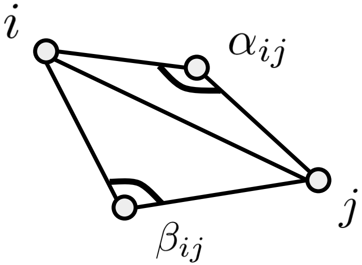
[Desbrun et al. 1999]

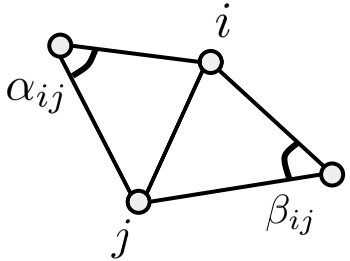
No Free Lunch (The Laplacian Version) [Wardetzky et al. 2007]

Not all meshes admit Laplacians satisfying properties SYM, LOC, LIN and POS simultaneously.

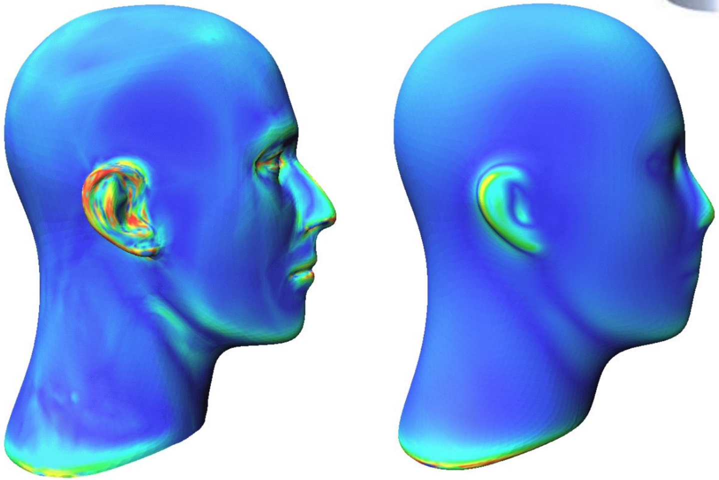
A triangulation of the plane allows for discrete Laplacians which satisfy SYM+LOC+LIN+POS if and only if triangulation is regular.

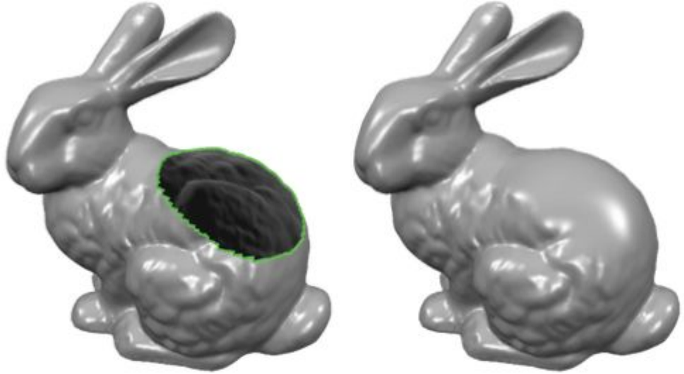
Many approaches for obtaining good triangulation. e.g. edge flip ⇒ Delaunay





More Applications!



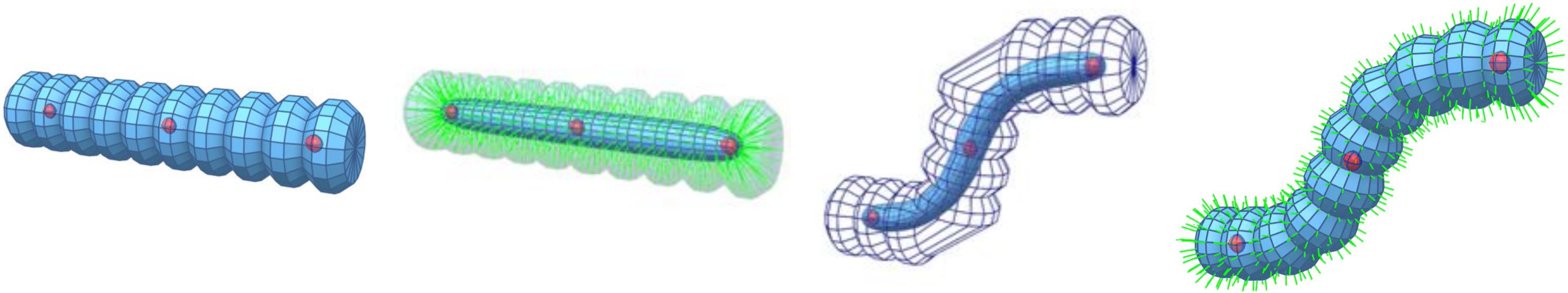


A Recent Example: Delta Mush [Mancewicz et al. 2014]

Motivation: Rigid Binding

Mush = Laplacian Smoothing (Lose surface details)

Delta = Displacement encoding



Major limitation: laplacian smoothing on every frame (~24fps x 1 model on 99% CPU+GPU)

A recent advance [Le et al. 2019] 100 models on 5% GPU in < 16ms from EA

Session 3: Smoothing

- Mesh Smoothing
  - Heat Equation and Laplacian Smoothing
  - Laplace and Mass Matrix
  - Linear Solvers
- "*No-free Lunch*"
- Summary
- Discussion

Summary

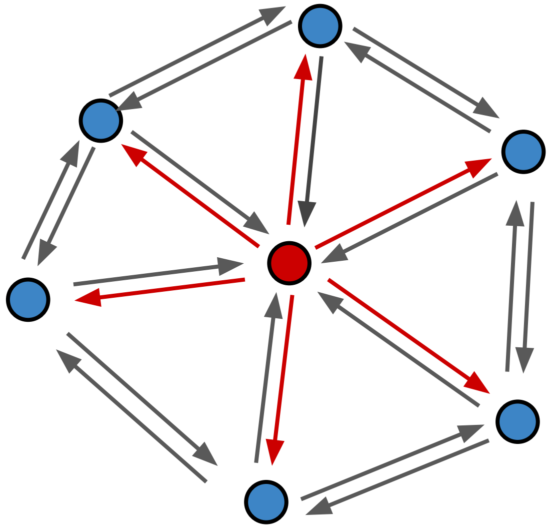
- Geometry processing tasks are often turned into a linear system, and Laplacian is the key
- No free lunch: A perfect Laplacian does not exist, one must adapts the weights depending on the task
- Smoothing via Laplacian as an entry level example to more geometry processing tasks

Session 3: Smoothing

- Mesh Smoothing
    - Heat Equation and Laplacian Smoothing
    - Laplace and Mass Matrix
    - Linear Solvers
- "*No-free Lunch*"
- Summary
- Discussion

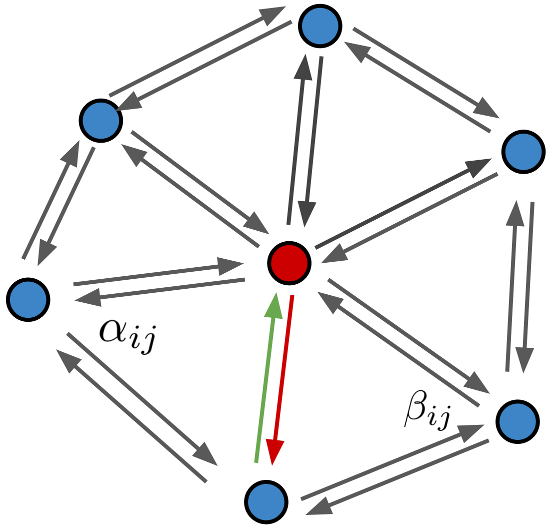
Halfedge Traversal

```
halfedges(fn) { // given vertex
  let start = true
  let i = 0
  for (let h = this.halfedge; start || h != this.halfedge; h = h.twin.next) {
    fn(h, i)
    start = false
    i++
  }
}
```



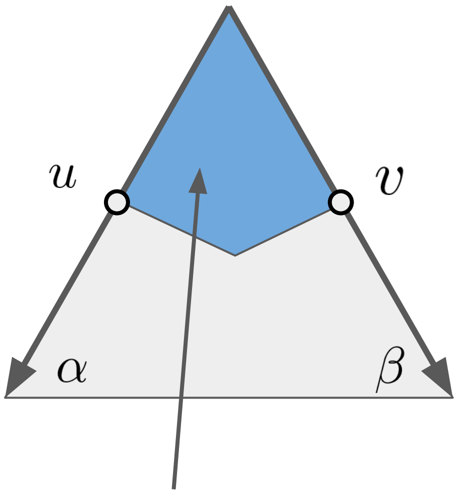
Calculating Cotan Laplacian

```
cotanLaplaceBeltrami() {
  const a = this.voronoiCell()
  let sum = new Vector()
  this.halfedges(h => { sum = sum.add(h.vector().scale(h.cotan() + h.twin.cotan())) })
  return sum.norm()*0.5/a
}
```



Voronoi Vertex Area

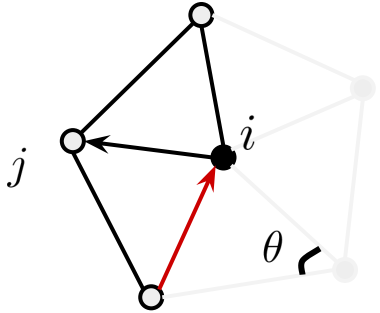
```
voronoiCell() {
  let a = 0
  this.halfedges(h => {
    const u = h.prev.vector().norm()
    const v = h.vector().norm()
    a += (u*u*h.prev.cotan() + v*v*h.cotan())/8
  })
  return a
}
```

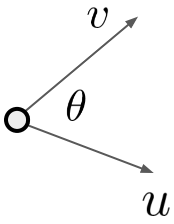


$$\frac{1}{8}(u^2 \cot \alpha + v^2 \cot \beta)$$

Dealing with Mesh Boundaries

```
cotan() {
  if (this.onBoundary) {
    return 0
  }
  const u = this.prev.vector()
  const v = this.next.vector().scale(-1)
  return u.dot(v) / u.cross(v).norm()
}
```





$$\cot\theta = \frac{u \cdot v}{||u \times v||}$$

Computing Normal/Curvature

Normal:

```
case 'angle-weighted':
  this.halfedges(h => { n = n.add(h.face.normal().scale(h.next.angle())) })
  return n.unit()
...
```

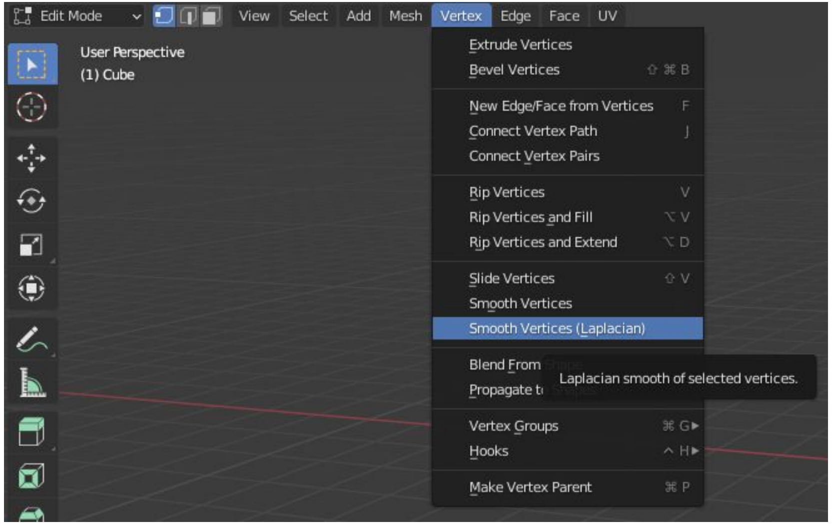
Curvature:

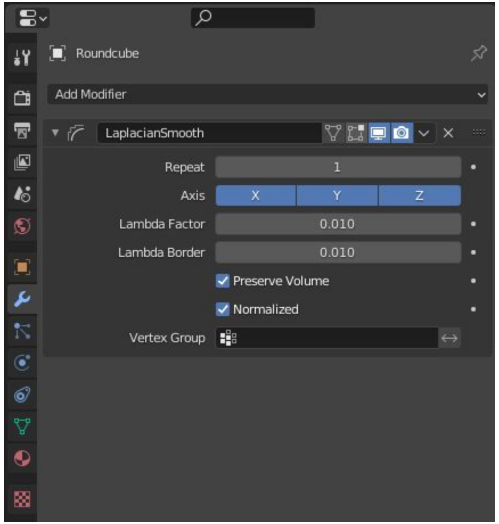
```
const [k1, k2] = this.principalCurvature()
switch (method) {
case 'Mean':
  return (k1+k2)*0.5
case 'Gaussian':
  return k1*k2
...
```

Smooth Modifiers in Blender

https://docs.blender.org/manual/en/latest/modeling/modifiers/deform/laplacian_smooth.html

See Blender's implementation: In source/blender/modifiers/intern/MOD_laplaciansmooth.c (e4facbbea540)





Further Readings

[Desbrun et al. 1999] Desbrun M, et al. Implicit fairing of irregular meshes using diffusion and curvature flow. InProceedings of the 26th annual conference on Computer graphics and interactive techniques 1999 Jul 1.

[Shewchuk. 2002] Shewchuk, Jonathan Richard. What is a good linear finite element? interpolation, conditioning, anisotropy, and quality measures. University of California at Berkeley 2002.

[Wardetzky et al. 2007] Wardetzky, Max, et al. Discrete Laplace operators: no free lunch. Symposium on Geometry processing. 2007.

[Mancewicz et al. 2014] Mancewicz, Joe, et al. Delta Mush: smoothing deformations while preserving detail. Proceedings of the Fourth Symposium on Digital Production. 2014.

[Zhang et al. 2015] Zhang H, et al. Variational mesh denoising using total variation and piecewise constant function space. IEEE transactions on visualization and computer graphics. 2015 Feb 2.

[Le et al. 2019] Le BH, Lewis JP. Direct delta mush skinning and variants. ACM Trans. Graph.. 2019 Jul 12.