Advanced Topics in Computer Graphics II
Geometry Processing

Mesh Smoothing

November 14, 2024

Smoothing of an object can be interpreted as heat diffusion flow of a scalar field over its boundary.

Given:

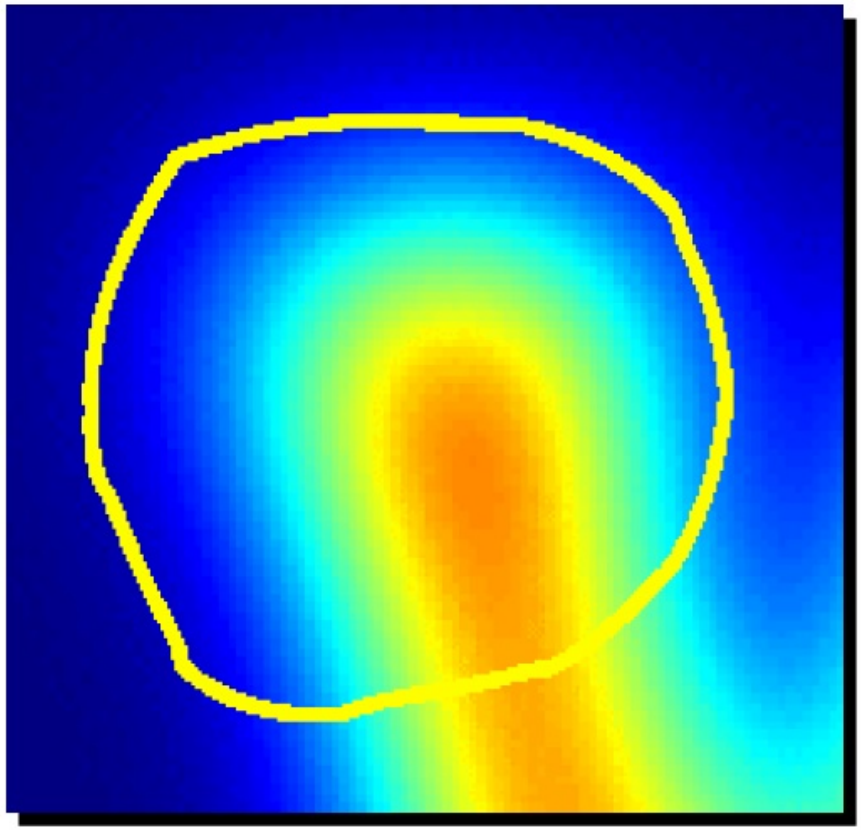
- A scalar field

$$\rho \colon \mathbb{R}^2 \times \mathbb{R}^+ \to \mathbb{R}$$
$$(\boldsymbol{x}, t) \mapsto \rho(\boldsymbol{x}, t)$$

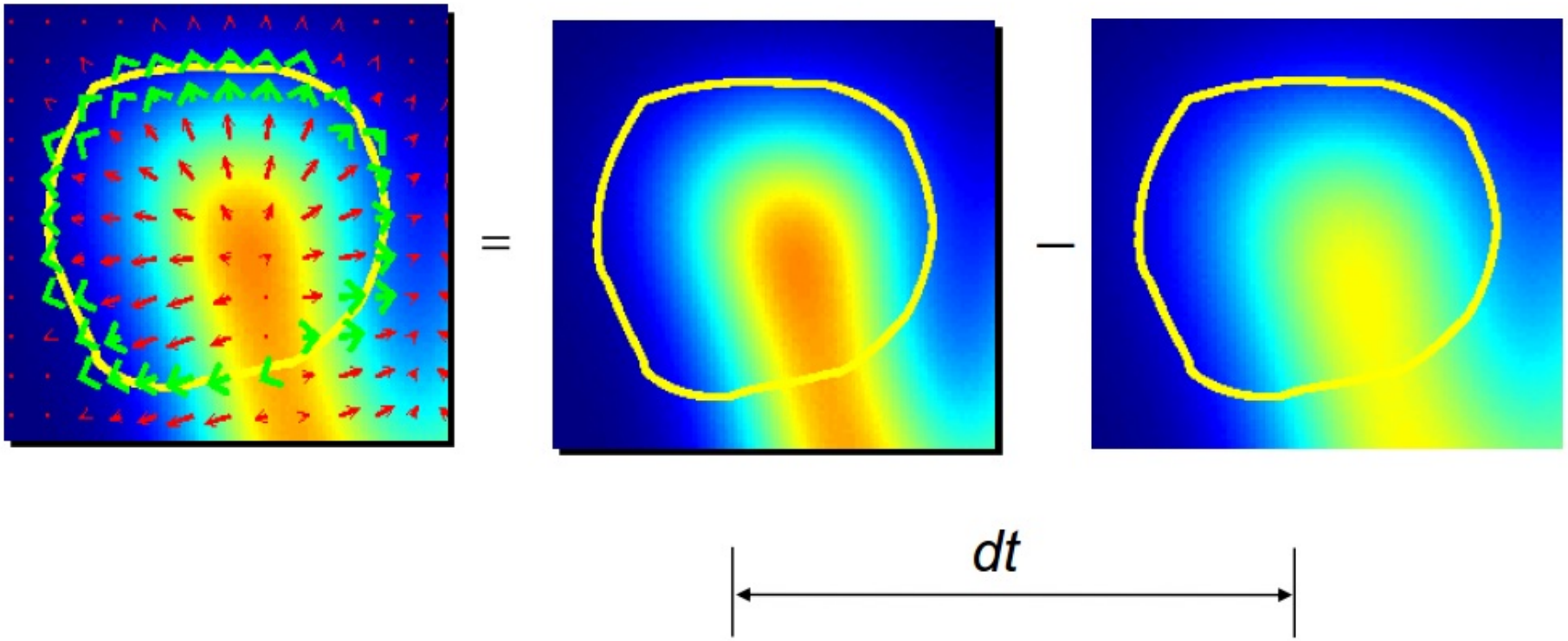
  with initial state

$$\rho(\boldsymbol{x}, 0) = \rho(\boldsymbol{x})$$

- The boundary of an object $\partial\Omega$

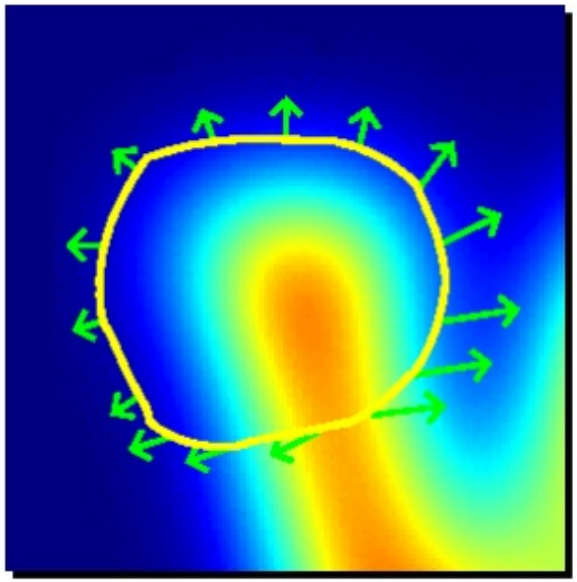
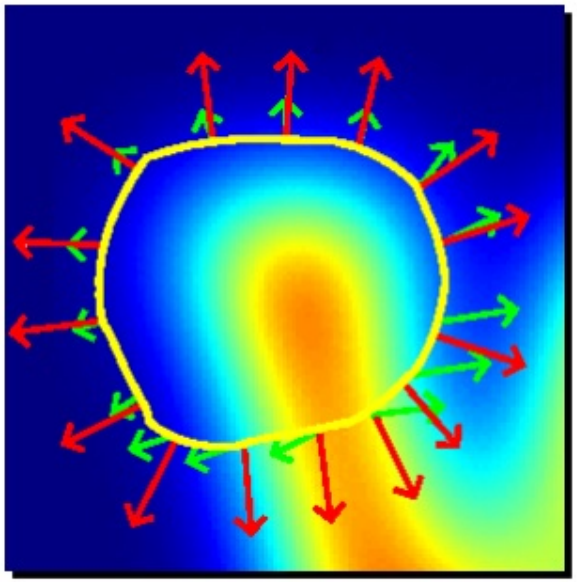
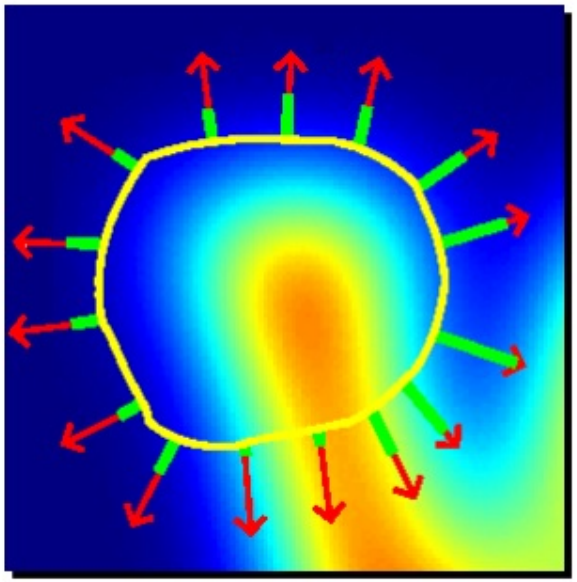


Diffusion Flow



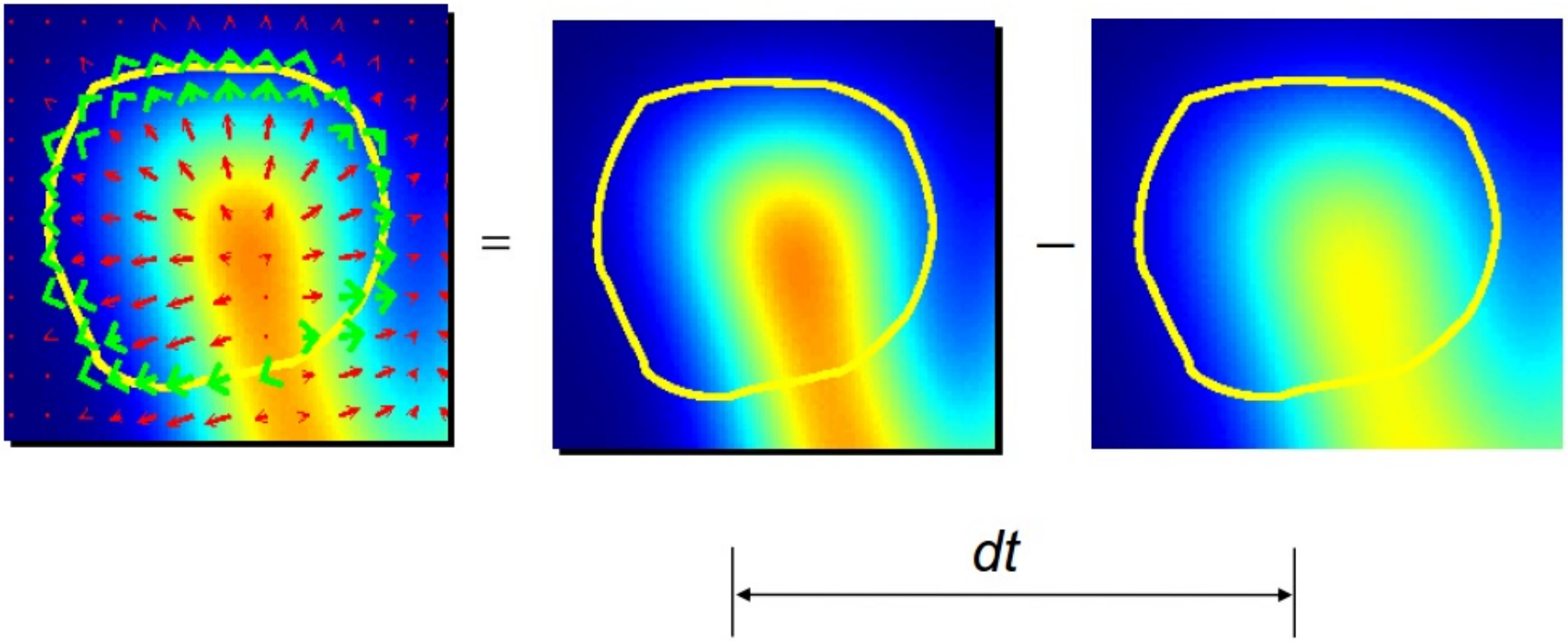
The amount of heat flux $\boldsymbol{f}$ flowing through the scalar field is proportional to its negative gradient (Fourier's law):

$$\boldsymbol{f} = -\lambda \cdot \nabla \rho(\boldsymbol{x}, t)$$

Since we are interested at the total amount of flux Φ passing through the boundary of the object, we have to project it:

$$\Phi = \int_{\partial\Omega} \langle \boldsymbol{f} \mid \boldsymbol{n} \rangle \, dA$$



One the other hand, the total amount of flux Φ is given by (Law of Energy Conservation):

$$\Phi = -\int_{interior(\Omega)} \frac{\partial}{\partial t}\rho(\boldsymbol{x}, t)\, dV$$

So we get the equation:

$$\int_{\partial\Omega} \langle \boldsymbol{f} \,|\, \boldsymbol{n} \rangle \, dA = - \int_{interior(\Omega)} \frac{\partial}{\partial t} \rho(\boldsymbol{x}, t) \, dV$$

Definition

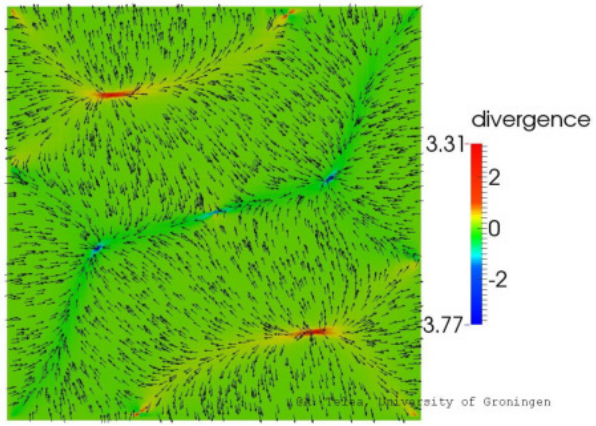
The divergence of an n-dimensional vector field

$$\mathbf{F} = \begin{bmatrix} F_1(\mathbf{x}) \\ F_2(\mathbf{x}) \\ \vdots \\ F_n(\mathbf{x}) \end{bmatrix}$$ in n-dimensional space is given by:

$$\nabla \cdot \mathbf{F} = \frac{\partial F_1}{\partial x_1} + \frac{\partial F_2}{\partial x_2} + \cdots + \frac{\partial F_n}{\partial x_n},$$

where

- $\nabla \cdot \mathbf{F}$ represents the divergence of the n-dimensional vector field $\mathbf{F}$.
- $\frac{\partial F_i}{\partial x_i}$ is the partial derivative of the i-th component of $\mathbf{F}$ with respect to the i-th coordinate x_i, where $i = 1, 2, \ldots, n$.



Theorem (Divergence Theomrem 2D)

Let $\mathbf{F} = \begin{bmatrix} M(x,y) \\ N(x,y) \end{bmatrix}$ *be a 2D vector field defined in a simply connected region containing a closed curve* C *that bounds a simply connected region* D *in the xy-plane. If* $\mathbf{F}$ *is continuously differentiable in* D*, then the Divergence Theorem for 2D states:*

$$\oint_C \mathbf{F} \cdot d\mathbf{r} = \iint_D (\nabla \cdot \mathbf{F})\, dA$$

where:

- $\oint_C$ *represents the line integral around the closed curve* C*.*
- $\iint_D$ *represents the double integral over the region* D *enclosed by* C*.*
- $\mathbf{F} \cdot d\mathbf{r}$ *is the dot product of the vector field* $\mathbf{F}$ *and the differential displacement vector* $d\mathbf{r}$ *along* C*.*
- $(\nabla \cdot \mathbf{F})$ *is the divergence of the vector field* $\mathbf{F}$*.*
- dA *is the differential area element in the xy-plane.*

Theorem (Divergence Theomrem 3D)

Let $\mathbf{F} = \begin{bmatrix} M(x,y,z) \\ N(x,y,z) \\ P(x,y,z) \end{bmatrix}$ *be a 3D vector field defined in a region of space containing a closed surface* S *that encloses a volume* V*. If* $\mathbf{F}$ *is continuously differentiable in* V*, then the Divergence Theorem for 3D states:*

$$\oint_S \mathbf{F} \cdot d\mathbf{S} = \iiint_V (\nabla \cdot \mathbf{F})\, dV$$

where:

- $\oint_S$ *represents the line integral around the closed curve* S*.*
- $\iint_V$ *represents the double integral over the region* V *enclosed by* S*.*
- $\mathbf{F} \cdot d\mathbf{r}$ *is the dot product of the vector field* $\mathbf{F}$ *and the differential displacement vector* $d\mathbf{S}$ *along* S*.*
- $(\nabla \cdot \mathbf{F})$ *is the divergence of the vector field* $\mathbf{F}$*.*
- dV *is the differential area element in 3D space.*

So we get the equation:

$$\int_{\partial\Omega} \langle\, \boldsymbol{f} \,|\, \boldsymbol{n} \,\rangle \, dA = -\int_{interior(\Omega)} \frac{\partial}{\partial t}\rho(\boldsymbol{x},t)\, dV$$

By substituting the flux $\boldsymbol{f} = -\lambda \cdot \nabla\rho$ and using the divergence theorem, we obtain

$$\int_{\partial\Omega} \langle\, \boldsymbol{f} \,|\, \boldsymbol{n} \,\rangle \, dA = -\int_{interior(\Omega)} \frac{\partial}{\partial t}\rho(\boldsymbol{x},t)\, dV$$

$$-\int_{\partial\Omega} \langle\, \lambda \cdot \nabla\rho(\boldsymbol{x},t) \,|\, \boldsymbol{n} \,\rangle \, dA = -\int_{interior(\Omega)} \frac{\partial}{\partial t}\rho(\boldsymbol{x},t)\, dV$$

$$-\int_{interior(\Omega)} \underbrace{\operatorname{div}\left(\lambda \cdot \nabla\rho(\boldsymbol{x},t)\right)}_{=\lambda\cdot\Delta\rho(\boldsymbol{x},t)=\lambda\left(\frac{\partial^2\rho(x,t)}{\partial x_1^2}+\frac{\partial^2\rho(x,t)}{\partial x_2^2}\right)} dV = -\int_{interior(\Omega)} \frac{\partial}{\partial t}\rho(\boldsymbol{x},t)\, dV$$

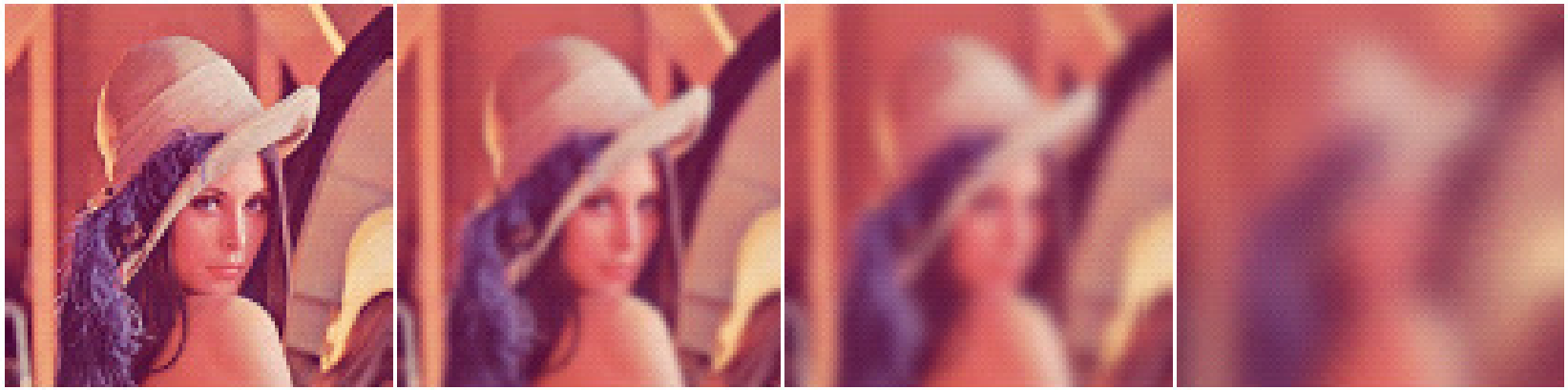
Definition (Diffusion Equation)

The equation

$$\frac{\partial}{\partial t}\rho(\boldsymbol{x},t) = \lambda \cdot \Delta\rho(\boldsymbol{x},t)$$

is called the *Diffusion or Heat Equation.*

Example: Laplace/Gauss Filter on images with $\lambda = 1$:



$$\frac{\partial}{\partial t}\rho(\boldsymbol{x},t) = \Delta\rho(\boldsymbol{x},t) = \frac{\partial^2\rho}{\partial x_1^2}(x,t) + \frac{\partial^2\rho}{\partial x_2^2}(x,t)$$

Filtering with larger standard deviation over time ($\sigma(t) = t$) corresponds to diffusion flow:

$$\rho(\cdot,t) = g(\cdot,t) \otimes \rho(\cdot,0)$$

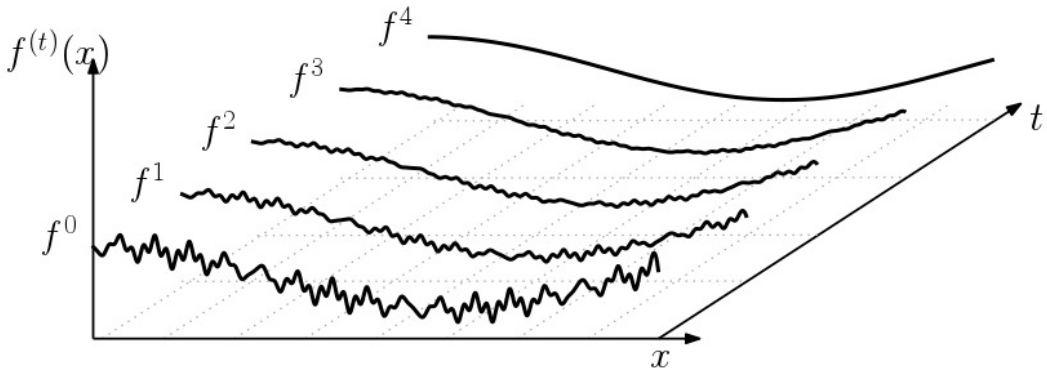
with

$$g(\boldsymbol{x},t) = \frac{1}{\sqrt{2\,\pi\,t^2}}\,\exp\left(-\frac{\|\boldsymbol{x}\|_2^2}{2\,t^2}\right)$$

Diffusion Flow in 1D

Consider a real-valued one-dimensional function $f^{(t)} : \mathbb{R} \to \mathbb{R}$. Then the heat equation with respect to this function at the value x_i leads to

$$\begin{aligned}\frac{\partial}{\partial t} f^{(t)}(x_i) &= \lambda \cdot \Delta f^{(t)}(x_i) \\ &= \lambda \cdot \frac{\partial^2 f^{(t)}}{\partial x^2}(x_i)\end{aligned}$$



Using finite differences, we can approximate the left-hand side by (forward difference)

$$\frac{\partial}{\partial t} f^{(t)}(x_i) \approx \frac{f^{(t+dt)}(x_i) - f^{(t)}(x_i)}{dt}$$

and get the following result (with $dt = 1$):

$$f^{(t+1)}(x_i) = f^{(t)}(x_i) + \lambda \cdot \frac{\partial^2 f^{(t)}}{\partial x^2}(x_i)$$

This is called the *Explicit Euler Step.*

What remains is an efficient way to compute the second derivate of our function $f^{(t)}$.

First, we apply a central difference at x_i from its neighbors:

$$\frac{\partial^2 f^{(t)}}{\partial x^2}(x_i) = \frac{1}{2}\left(\frac{\partial f^{(t)}}{\partial x}(x_{i+1}) - \frac{\partial f^{(t)}}{\partial x}(x_{i-1})\right)$$

For simplicity, we sample with a distance of one, so $x_{i-1} := x_i - 1$ and $x_{i+1} := x_i + 1$.

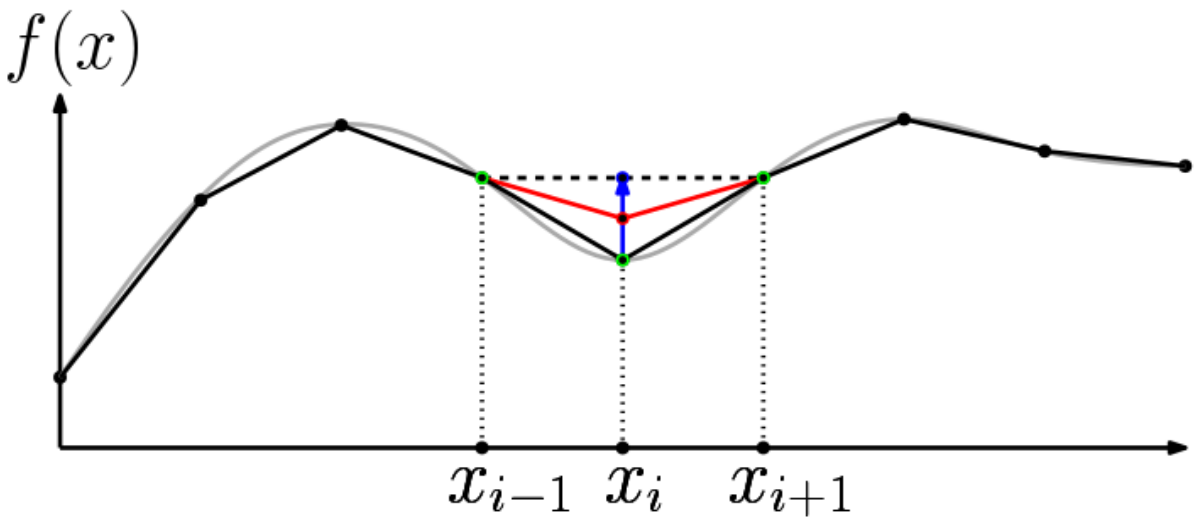
To avoid additional sample points, the first derivatives in this formula are approximated by a simple backward and forward difference:

$$\begin{aligned}\frac{\partial^2 f^{(t)}(x_i)}{\partial x^2} &= \frac{\left(f^{(t)}(x_{i+1}) - f^{(t)}(x_i)\right) - \left(f^{(t)}(x_i) - f^{(t)}(x_{i-1})\right)}{2} \\ &= \frac{f^{(t)}(x_{i+1}) - 2\,f^{(t)}(x_i) + f^{(t)}(x_{i-1})}{2}\end{aligned}$$

$$\frac{\partial^2 f^{(t)}}{\partial x^2}(x_i) = \frac{f^{(t)}(x_{i+1}) - 2\,f^{(t)}(x_i) + f^{(t)}(x_{i-1})}{2} = \frac{f^{(t)}(x_{i+1}) + f^{(t)}(x_{i-1})}{2} - f^{(t)}(x_i)$$

Plugging this into the Explicit Euler Step leads to

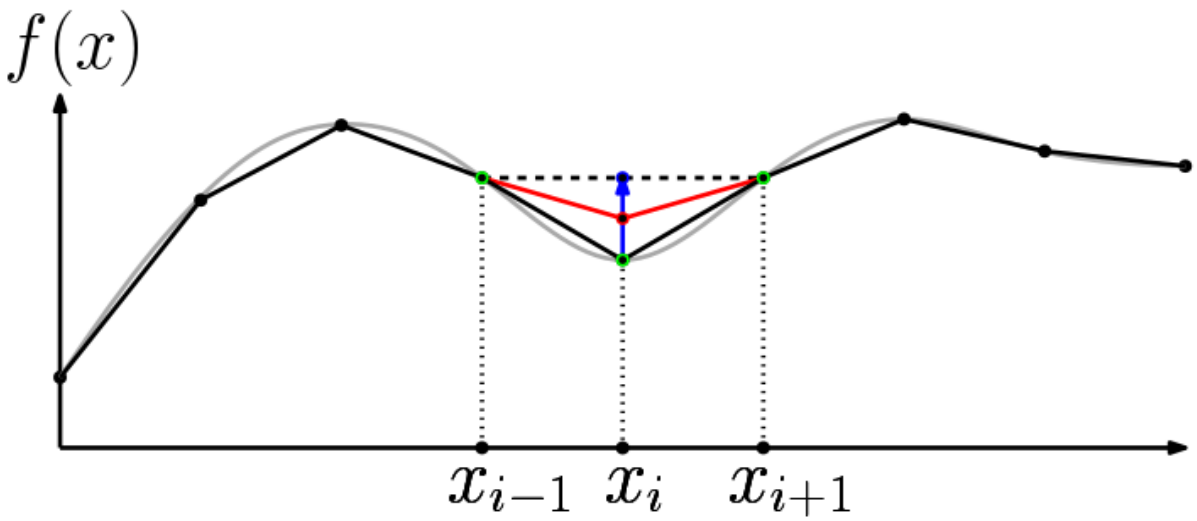
$$f^{(t+1)}(x_i) = f^{(t)}(x_i) + \lambda \cdot \frac{f^{(t)}(x_{i+1}) - 2\,f^{(t)}(x_i) + f^{(t)}(x_{i-1})}{2}$$



$$\frac{\partial^2 f^{(t)}}{\partial x^2}(x_i) = \frac{f^{(t)}(x_{i+1}) - 2\,f^{(t)}(x_i) + f^{(t)}(x_{i-1})}{2} = \frac{f^{(t)}(x_{i+1}) + f^{(t)}(x_{i-1})}{2} - f^{(t)}(x_i)$$

Plugging this into the Explicit Euler Step leads to

$$f^{(t+1)}(x_i) = f^{(t)}(x_i) + \lambda \cdot \frac{f^{(t)}(x_{i+1}) - 2\,f^{(t)}(x_i) + f^{(t)}(x_{i-1})}{2}$$



Observation: Laplacian is the deviation from the local average!

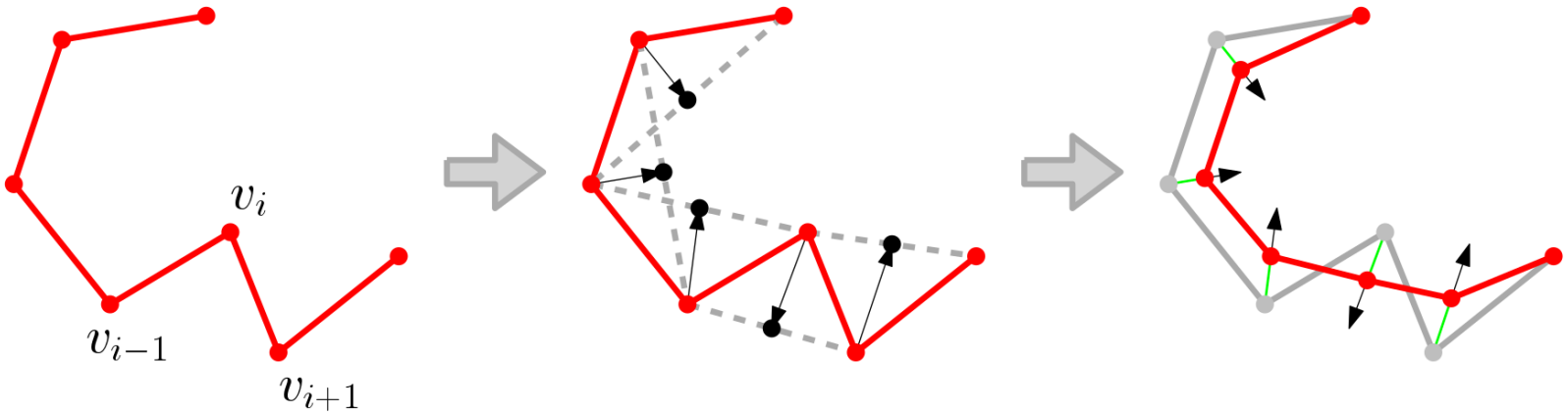
Idea: Apply the Explicit Euler Step and the Laplace operator to the x and y coordinates of a given polygon:

$$\boldsymbol{v}_i^{(t+1)} = \boldsymbol{v}_i^{(t)} + \lambda \cdot \Delta \boldsymbol{v}_i^{(t)} = \boldsymbol{v}_i^{(t)} + \lambda \cdot \frac{\boldsymbol{v}_{i+1}^{(t)} - 2\,\boldsymbol{v}_i^{(t)} + \boldsymbol{v}_{i-1}^{(t)}}{2}$$

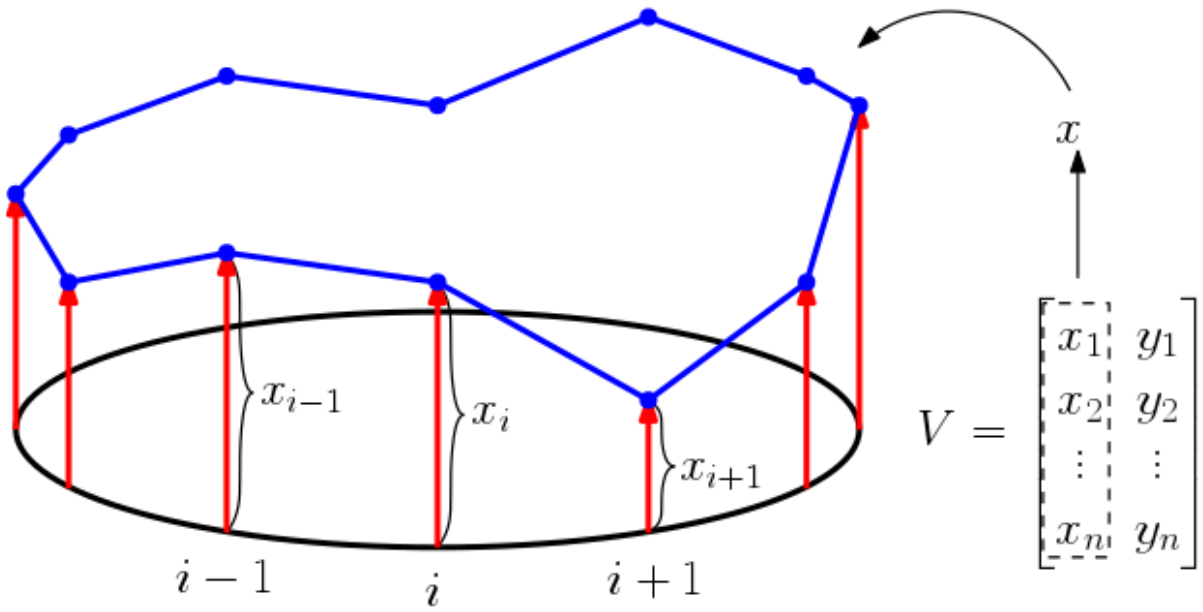
For $\lambda = 1$, there is an especially simple and intuitive interpretation of this scheme:

$$\boldsymbol{v}_i^{(t+1)} = \frac{\boldsymbol{v}_{i+1}^{(t)} + \boldsymbol{v}_{i-1}^{(t)}}{2}$$

So we move the vertex $\boldsymbol{v}_i^{(t)}$ to the "mean" of its neighborhood. The parameter λ controls how far we move along this direction.



Given a polygon (e.g. contour of a 2D shape) with a linear index $i \in \{1, \dots, n\}$. Let $\boldsymbol{V} = [\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}] \in \mathbb{R}^{n \times 3}$ be the vertex matrix of the polygon where each row corresponds to a single vertex $\boldsymbol{v}_i = (x_i, y_i, z_i)$.



Question: How does the linear operator $\boldsymbol{L} \in \mathbb{R}^{n \times n}$ that calculates the Laplacian of each vertex of a polygon look like?

$$\boldsymbol{V}^{(t+1)} = \boldsymbol{V}^{(t)} + \lambda \boldsymbol{L} \boldsymbol{V}^{(t)}$$

Using the previous results

$$\boldsymbol{v}_i^{(t+1)} = \boldsymbol{v}_i^{(t)} + \lambda \cdot \Delta \boldsymbol{v}_i^{(t)} = \boldsymbol{v}_i^{(t)} + \lambda \cdot \frac{1 \cdot \boldsymbol{v}_{i+1}^{(t)} - 2 \cdot \boldsymbol{v}_i^{(t)} + 1 \cdot \boldsymbol{v}_{i-1}^{(t)}}{2}$$

we can simply construct our linear operator as

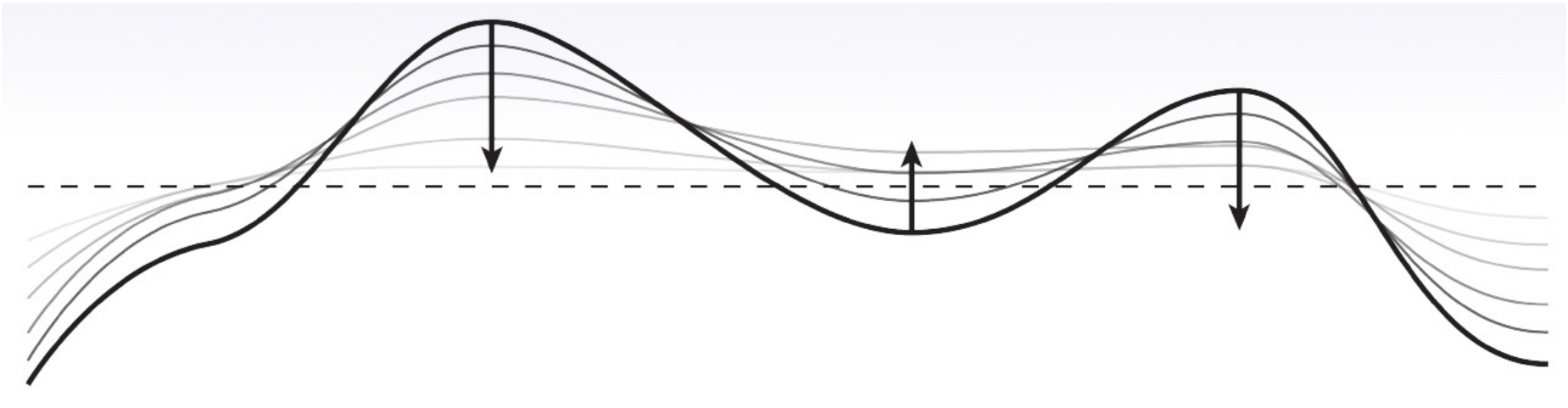
$$\boldsymbol{L} = \frac{1}{2} \begin{pmatrix} -2 & 1 & 0 & \dots & \dots & 0 & 1 \\ 1 & -2 & 1 & 0 & \dots & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & \dots & 0 & 1 & -2 & 1 \\ 1 & 0 & \dots & \dots & 0 & 1 & -2 \end{pmatrix}$$

$$= \frac{1}{2} \left[\begin{pmatrix} 0 & 1 & 0 & \dots & \dots & 0 & 1 \\ 1 & 0 & 1 & 0 & \dots & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & \dots & 0 & 1 & 0 & 1 \\ 1 & 0 & \dots & \dots & 0 & 1 & 0 \end{pmatrix} - \begin{pmatrix} 2 & 0 & \dots & \dots & \dots & \dots & 0 \\ 0 & 2 & 0 & \dots & \dots & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & \dots & \dots & 0 & 2 & 0 \\ 0 & \dots & \dots & \dots & \dots & 0 & 2 \end{pmatrix} \right]$$

$$= \frac{1}{2} \left(\boldsymbol{A} - \boldsymbol{D} \right)$$

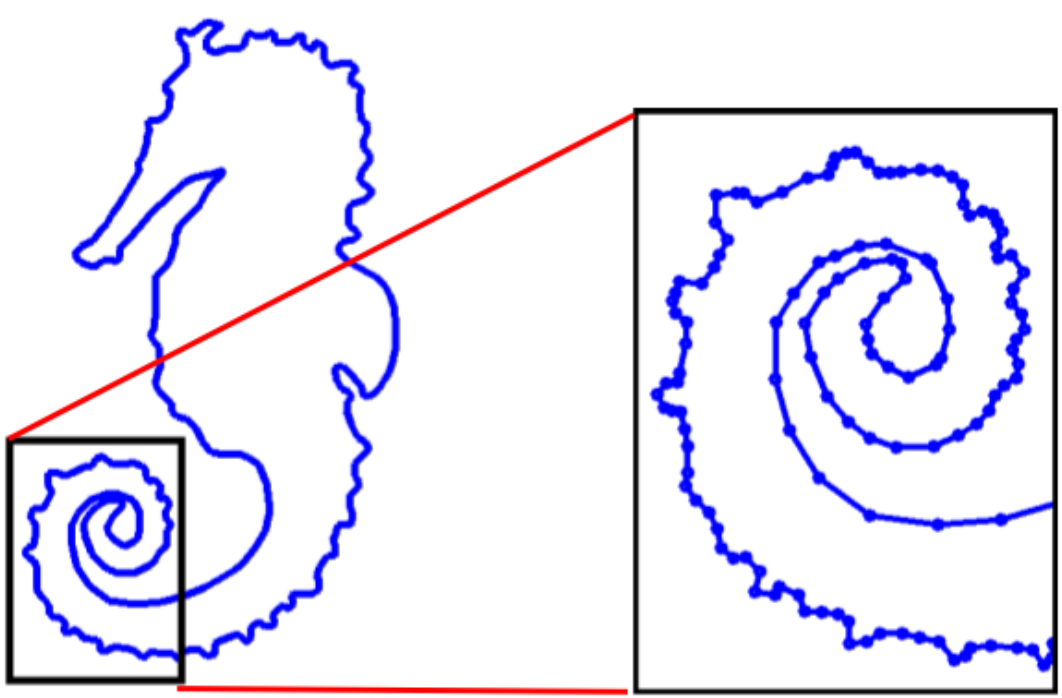
For a curve $\gamma : \mathbb{R} \to \mathbb{R}^2, \quad \gamma(u) = (x(u), y(u))^T$ we get

$$\Delta\gamma(u) = (\Delta x(u), \Delta y(u))^T$$



If the curve is arc length parametrized this is exactly the curvature normal $\kappa\, \boldsymbol{n}$.

Example: 10 iterations of the Explicit Euler Step



(a) Original seahorse

(b) Smoothed seahorse

Note: A closed curve converges to a single point!

This smoothing concept directly generalizes to meshes. Only the discrete operator $\boldsymbol{L}$ changes.

Theorem (Explicit Euler Step)

Given a mesh with vertices $\boldsymbol{V} = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n)^T$. A smoothed version of this mesh can be computed iteratively by applying the Explicit Euler Step:

$$\boldsymbol{v}_i^{(t+1)} = (\boldsymbol{I} + \lambda \cdot \boldsymbol{L})\, \boldsymbol{v}_i^{(t)}$$

The Explicit Euler Step is also called Laplacian Smoothing.

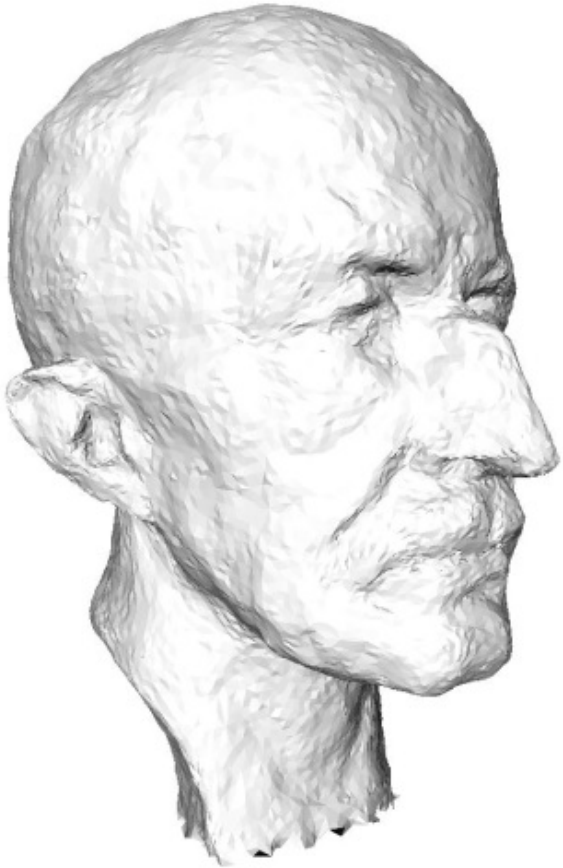
Formally, the above formula is interpreted as applying $\boldsymbol{L}$ to the vertex matrix and then selecting the i-th vertex:

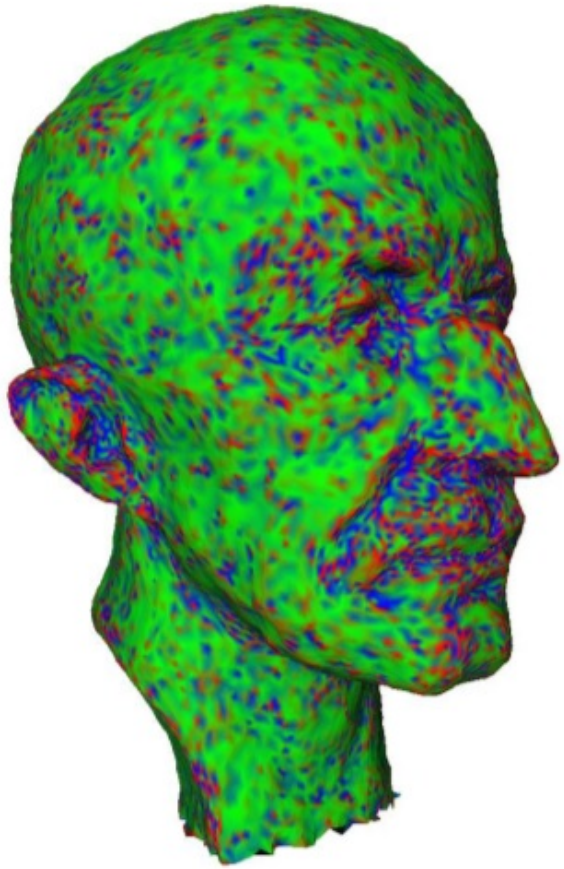
$$\left(\begin{pmatrix} \boldsymbol{v}_1^{(t+1)} \\ \vdots \\ \boldsymbol{v}_n^{(t+1)} \end{pmatrix} \right)_i = \left((\boldsymbol{I} + \lambda \cdot \boldsymbol{L}) \begin{pmatrix} \boldsymbol{v}_1^{(t)} \\ \vdots \\ \boldsymbol{v}_n^{(t)} \end{pmatrix} \right)_i$$

On surface meshes, the discrete Laplace-Beltrami operator is defined as

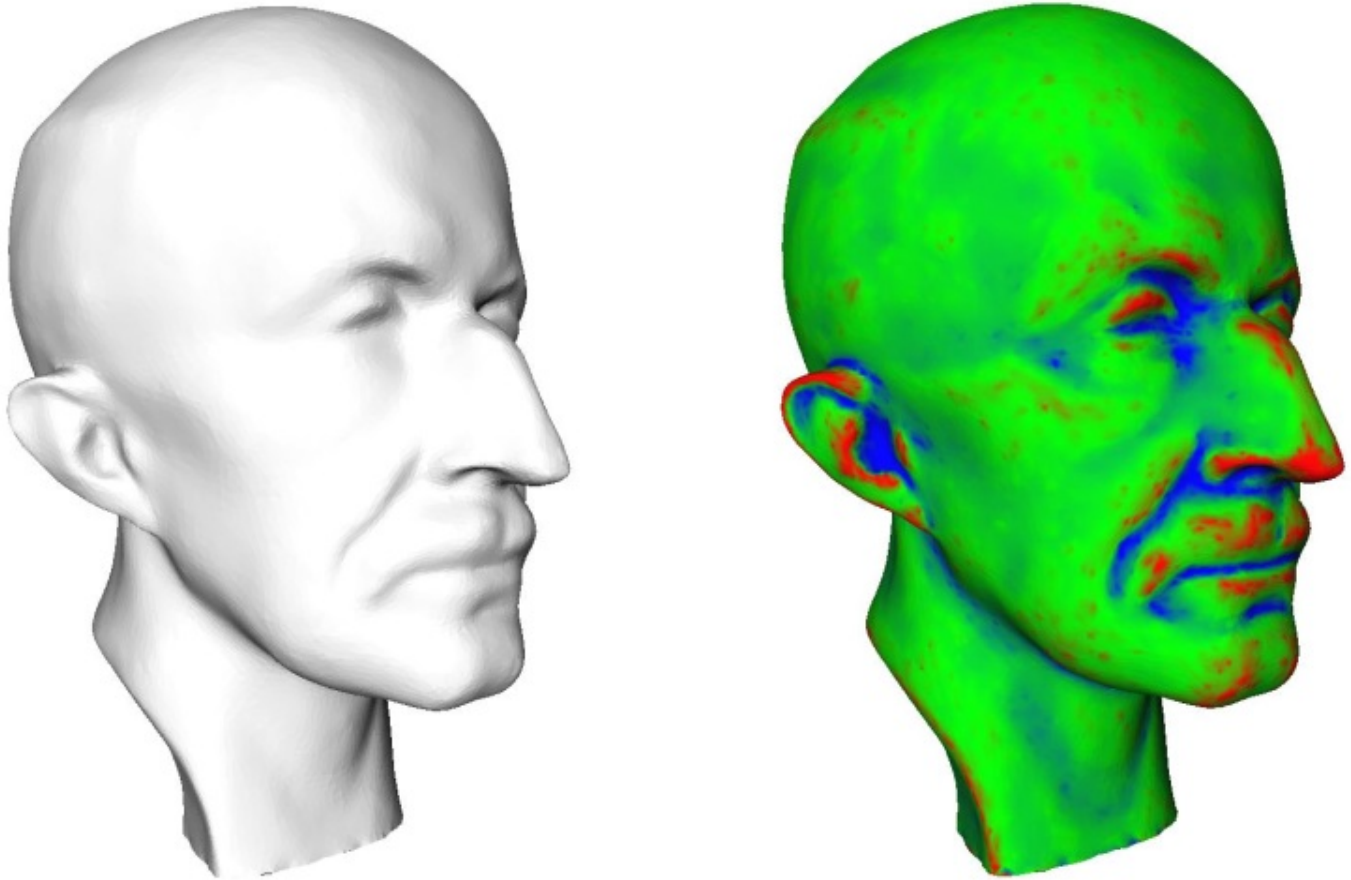
$$\boldsymbol{L}(\boldsymbol{v}_i) = \frac{\sum_{\boldsymbol{v}_j \in \mathcal{N}(\boldsymbol{v}_i)} w_{ij} (\boldsymbol{v}_j - \boldsymbol{v}_i)}{\sum_{\boldsymbol{v}_j \in \mathcal{N}(\boldsymbol{v}_i)} w_{ij}}, w_{ij} \in \mathbb{R}$$

Results: 4 iterations

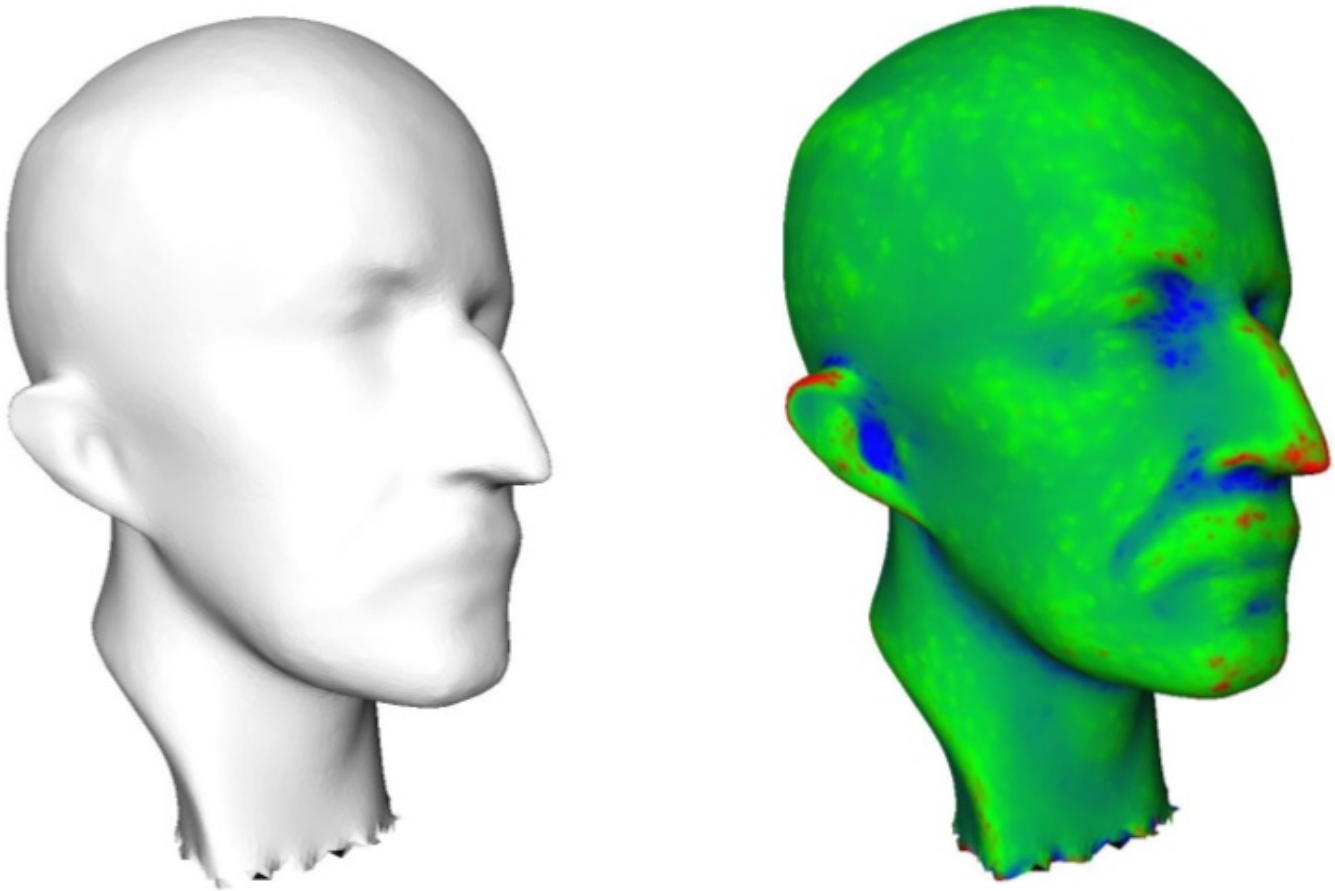




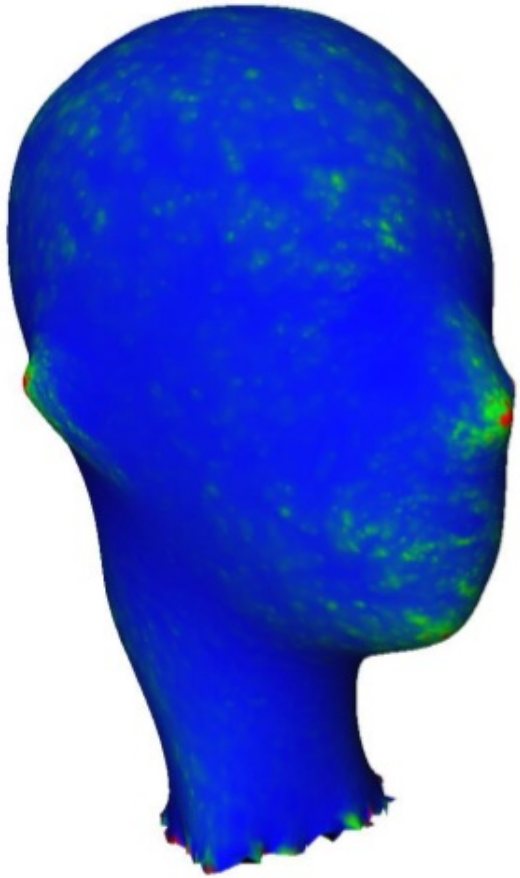
Results: 10 iterations



Results: 80 iterations



Results: 400 iterations



Problem: After 400 iterations in addition to the smoothing a severe contraction of the original mesh becomes visible.

Laplacian at vertices as average deviation on smooth surfaces

$$\Delta u(x_0) \propto \lim_{\varepsilon \to 0} \frac{1}{\varepsilon^2} \left(\frac{1}{|B_\varepsilon(x_0)|} \int_{B_\varepsilon(x_0)} u(x)\, dx - u(x_0) \right)$$

1. **Averaging of Differences:** The expression involves the integration of the difference between the function value $u(x)$ at a point x and a central point $u(x_0)$, over a small ball $B_\epsilon(x_0)$ around x_0.
2. **Volume Normalization:** The term $\frac{1}{|B_\epsilon(x_0)|}$ normalizes the integral by the volume of the region $B_\epsilon(x_0)$, effectively computing the average of the differences $u(x) - u(x_0)$ over this small neighborhood.
3. **Limit as $\epsilon \to 0$:** As $\epsilon \to 0$, the neighborhood $S_\epsilon(x_0)$ shrinks towards x_0, making the integral focus more on the infinitesimal variations of the function near the point x_0.
4. **Scaling by $\frac{1}{\epsilon^2}$:** The factor $\frac{1}{\epsilon^2}$ accounts for the scaling of the small region $S_\epsilon(x_0)$ as it shrinks. This ensures that the limit captures the second-order variation of the function rather than just vanishing as the region shrinks.

Given a smooth function $u(x)$, the Taylor expansion around x_0 up to second order is:

$$u(x) \approx u(x_0) + \sum_{i=1}^{n} \frac{\partial u}{\partial x_i}(x_0)(x_i - x_{0,i}) + \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \frac{\partial^2 u}{\partial x_i \partial x_j}(x_0)(x_i - x_{0,i})(x_j - x_{0,j}) + \mathcal{O}(\|x - x_0\|^3)$$

We integrate the deviation $u(x) - u(x_0)$ over a symmetric region (e.g. ball) $B_\epsilon(x_0)$ centered at x_0, the mixed second derivative terms $\frac{\partial^2 u}{\partial x_i \partial x_j}(x_0)(x_i - x_{0,i})(x_j - x_{0,j})$ cancel out due to the symmetry of the region. Specifically, the product $(x_i - x_{0,i})(x_j - x_{0,j})$ is odd with respect to reflection across the origin. Therefore, for every point x in the region, there exists a point $x' = 2x_0 - x$ where $(x'_i - x_{0,i})(x'_j - x_{0,j})$ has the opposite sign, causing the integral of the mixed terms to vanish:

$$\int_{B_\epsilon(x_0)} (x_i - x_{0,i})(x_j - x_{0,j})\, dx = 0 \quad \text{for} \quad i \neq j$$

After the cancellation of the mixed second derivative terms, only the diagonal terms remain, corresponding to the second partial derivatives:

$$\frac{\partial^2 u}{\partial x_i^2}(x_0)$$

These terms are even functions in $x_i - x_{0,i}$ and do not cancel during integration. In the limit as $\epsilon \to 0$, this expression converges to the classical Laplace operator:

$$\Delta u(x_0) = \sum_{i=1}^{n} \frac{\partial^2 u}{\partial x_i^2}(x_0)$$

The second-order term $(x_i - x_{0,i})^2$ in the Taylor expansion of $u(x)$ around x_0 scales quadratically with ϵ, as the neighborhood $B_\epsilon(x_0)$ is of size ϵ. Specifically:

$$(x_i - x_{0,i})^2 \sim \epsilon^2$$

Since $|x_i - x_{0,i}| \leq \epsilon$ for points x_i within the region $B_\epsilon(x_0)$, the quadratic term $(x_i - x_{0,i})^2$ shrinks with the square of ϵ which is compensated for by $\frac{1}{\epsilon^2}$. Thus, the factor $\frac{1}{\epsilon^2}$ scales the deviations appropriately, ensuring that the Laplacian reflects the sum of the second partial derivatives, without any additional weighting of the diagonal elements of the Hessian.

Observation: Applying explicit Euler steps for mesh smoothing leads to a severe contraction of the mesh!

Idea: Analyze the frequency behavior of the Laplace operator.

$$\boldsymbol{L} = \boldsymbol{D}^{-1}\,\boldsymbol{A} - \boldsymbol{I} \in \mathbb{R}^{n \times n}$$

This is motivated by Fourier theory.

For a continuous frequency variable ω consider the Fourier Basis functions

$$e_\omega : x \mapsto e^{2\pi i\omega x}$$

Applying the Laplace operator to these basis functions, we get

$$\Delta(e^{2\pi i\omega x}) = \frac{d^2}{dt^2} e^{2\pi i\omega x} = -(2\pi\omega)^2 \cdot e^{2\pi i\omega x}$$

Therefore, the Fourier Basis functions $e^{2\pi i\omega x}$ are Eigenfunctions of the Laplace operator with Eigenvalue $-(2\pi\omega)^2$.

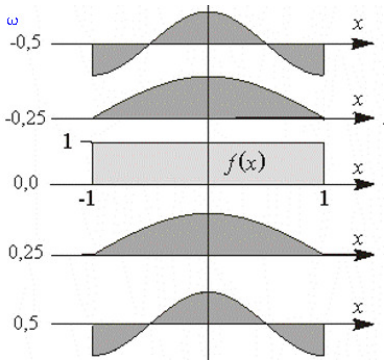


Figure: Examples of Eigen functions with different oscillation frequencies ω.

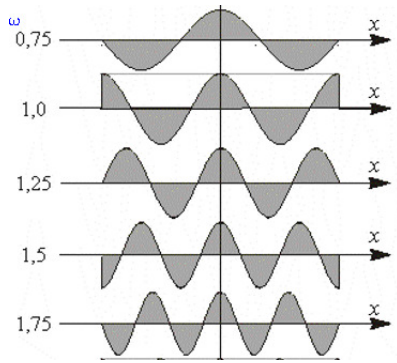


Figure: Smoothed seahorse

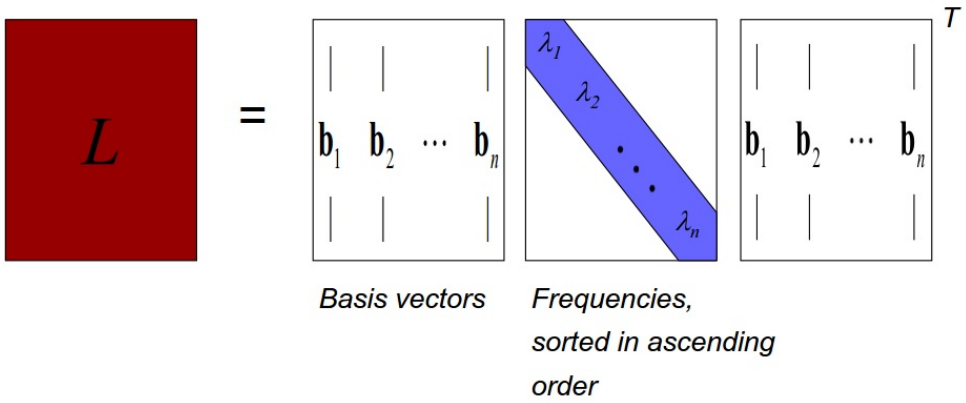
Therefore, in signal processing, the frequency behavior of a function is analyzed by projecting the function into the Fourier Basis (also called frequency space).

Definition (Eigenvector of the Laplace operator)

A vector $\boldsymbol{v}$ is called *eigenvector to the eigenvalue* λ *of the Laplace operator if it satisfies*

$$\Delta \boldsymbol{v} = \lambda \boldsymbol{v}$$

Using the eigenvectors we can compute an eigen decomposition of $\boldsymbol{L}$:



Note, that the Laplace operator of a polygon with n vertices as defined above is symmetric and therefore has n real eigenvalues and eigenvectors

$$0 = \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n \quad , \quad \boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_n$$

Example For a polygon the Laplace Operator

$$\boldsymbol{L} = \frac{1}{2}\begin{pmatrix} -2 & 1 & 0 & \dots & \dots & 0 & 1 \\ 1 & -2 & 1 & 0 & \dots & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & \dots & 0 & 1 & -2 & 1 \\ 1 & 0 & \dots & \dots & 0 & 1 & -2 \end{pmatrix}$$

is symmetric. $\boldsymbol{L}$ has the following n real eigenvalues k_i and eigenvectors u_i:

$$k_i = \cos\left(\frac{2}{n}\,\pi\,\left\lfloor\frac{i}{2}\right\rfloor\right) - 1 \quad , \quad i = 1, \dots, n$$

$$(\boldsymbol{u}_i)_j = \begin{cases} \sqrt{\frac{1}{n}} & \text{if } i = 1 \\ \sqrt{\frac{2}{n}}\sin\left(\frac{2}{n}\,\pi\,j\,\left\lfloor\frac{i}{2}\right\rfloor\right) & \text{if } i \text{ is even} \\ \sqrt{\frac{2}{n}}\cos\left(\frac{2}{n}\,\pi\,j\,\left\lfloor\frac{i}{2}\right\rfloor\right) & \text{if } i \text{ is odd} \end{cases}$$

Here we call the eigenvalues k_i to distinguish them from the smoothing parameter λ!

Now, the column vectors $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$ of the vertex matrix $\boldsymbol{V}$ of the polygon can be written as a linear combination of the n eigenvectors $\boldsymbol{u}_j$ of $\boldsymbol{L}$:

$$\boldsymbol{x} = \sum_{j=1}^{n} \hat{x}_j \, \boldsymbol{u}_j, \quad \boldsymbol{y} = \sum_{j=1}^{n} \hat{y}_j \, \boldsymbol{u}_j, \quad \boldsymbol{z} = \sum_{j=1}^{n} \hat{z}_j \, \boldsymbol{u}_j,$$

So the representation in frequency space of the polygon $\boldsymbol{v}$ is

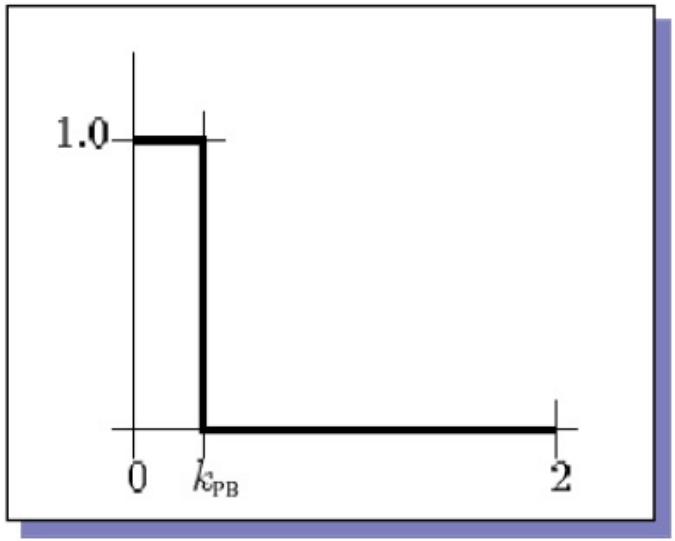
$$\hat{\boldsymbol{x}} = (\hat{x}_1, \ldots, \hat{x}_n), \quad \hat{\boldsymbol{y}} = (\hat{y}_1, \ldots, \hat{y}_n), \quad \hat{\boldsymbol{z}} = (\hat{z}_1, \ldots, \hat{z}_n).$$

An ideal low-pass filter has now a clear meaning for each column vector $\boldsymbol{v}$ of $\boldsymbol{V}$:

$$\boldsymbol{v} = \sum_{j=1}^{PB} \hat{v}_j \, \boldsymbol{u}_j + \sum_{j=PB+1}^{n} \hat{v}_j \, \boldsymbol{u}_j$$

Cut the frequencies behind some maximal frequency PB off:

$$\boldsymbol{v}' = \sum_{j=1}^{PB} 1 \cdot \hat{v}_j \, \boldsymbol{u}_j + \sum_{j=PB+1}^{n} 0 \cdot \hat{v}_j \, \boldsymbol{u}_j$$



Example: Low pass filter

(a) Original.

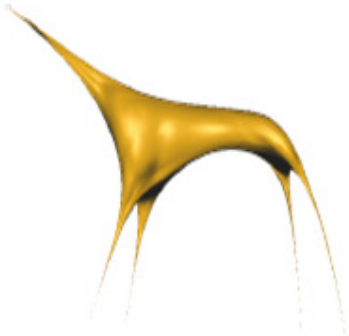
(b) $k = 300$.

(c) $k = 200$.

(d) $k = 100$.

(e) $k = 50$.

(f) $k = 10$.



(g) $k = 5$.

(h) $k = 3$.

Interpreting Laplacian Smoothing as a low pass filter: To interpret Laplacian Smoothing as a low-pass filter we need the following Lemma

Lemma

Let $\boldsymbol{L} : \mathbb{R}^n \to \mathbb{R}^n$ a diagonizable linear mapping, i.e there exists an ordered basis of $\mathbb{R}^n$ consisting of eigenvectors of $\mathbf{L}$, and $f(x) = \sum_{i=0}^{m} c_i x^i$ be a univariate polynomial of degree m. For $j = 1, \cdots, n$ let $\boldsymbol{u}_j$ be the eigenvector to the eigenvalue k_j. Then

- $f(\boldsymbol{L})$ has the same eigenvectors as $\boldsymbol{L}$, namely $\boldsymbol{u}_j$, $j = 1, \ldots, n$
- The eigenvalues of $f(\boldsymbol{L})$ to the eigenvectors $\boldsymbol{u}_j$ of $f(\mathbf{L})$ are given by

$$f(k_j) = f = \sum_{i=0}^{m} c_j k_j^i \quad , \quad j = 1, \ldots, n$$

Now, we define the Explicit Euler step as a function of the matrix $\boldsymbol{L}$:

$$\boldsymbol{v}^{(t+1)} = f(\boldsymbol{L})\, \boldsymbol{v}^{(t)} := (\boldsymbol{I} + \lambda \cdot \boldsymbol{L})\, \boldsymbol{v}^{(t)}$$

Then $f(\boldsymbol{L})$ has the same eigenvectors as $\boldsymbol{L}$, namely $\boldsymbol{u}_j, \quad j = 1, \ldots, n$ and its eigenvalues are given by $f(k_j) = 1 + \lambda \cdot k_j, \quad j = 1, \ldots, n$

Representing $\boldsymbol{v}^{(t)} = \sum_{j=1}^{n} \hat{v}_j \, \boldsymbol{u}_j$ in the eigenbasis, we obtain after filtering by one explicit Euler step,

$$\boldsymbol{v}^{(t+1)} = f(\boldsymbol{L}) \, \boldsymbol{v}^{(t)} = \sum_{j=1}^{n} f(k_j) \, \hat{v}_j \, \boldsymbol{u}_j$$

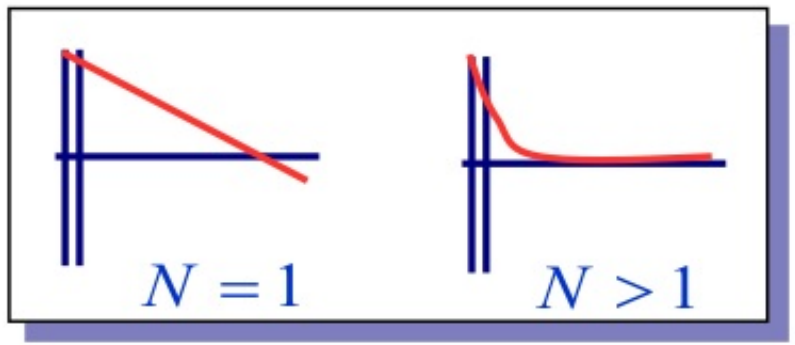
since $\boldsymbol{L} \, \boldsymbol{u}_j = k_j \, \boldsymbol{u}_j$. After N Euler steps, we get

$$\boldsymbol{v}^{(t+N)} = f(\boldsymbol{L})^N \, \boldsymbol{v}^{(t)} = \sum_{j=1}^{n} f(k_j)^N \, \hat{v}_j \, \boldsymbol{u}_j$$

The Explicit Euler step now yields the following Eigen values (frequencies):

$$f(k_j)^N = (1 + \lambda \cdot k_j)^N \to 0 \text{ for } N \to \infty, \; j > 0$$

This shows that if $|1 + \lambda \cdot k_j| < 1$, e.g. in the case of polygon smoothing where $k_j \in [-2, 0]$ and $\lambda \in [0, 1]$ the explicit Euler Step results in a contraction!



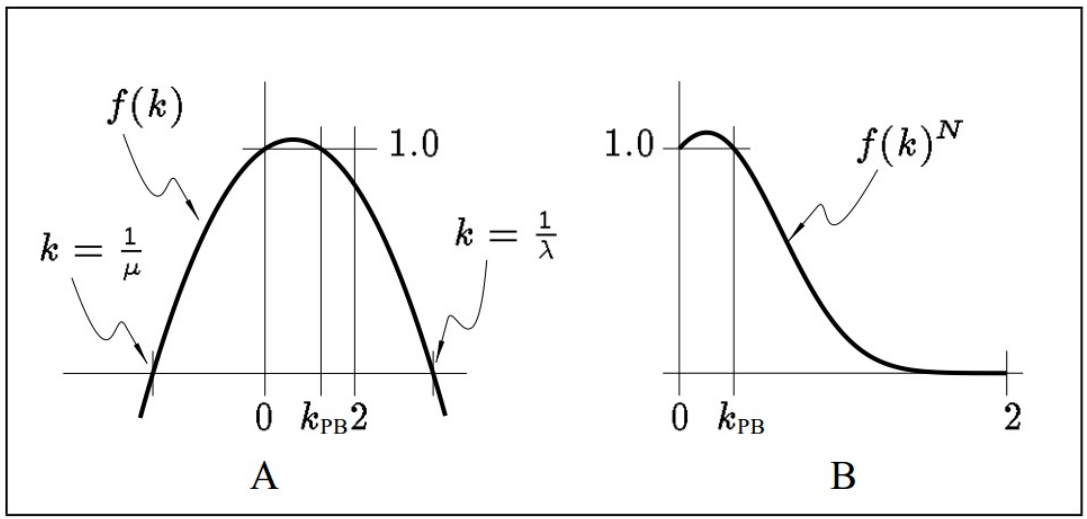
Note: By the above definition of L the k_j for $j > 0$ are negative!

Idea to avoid contraction[1]: Define $f(k)$ in a way that no contraction occurs

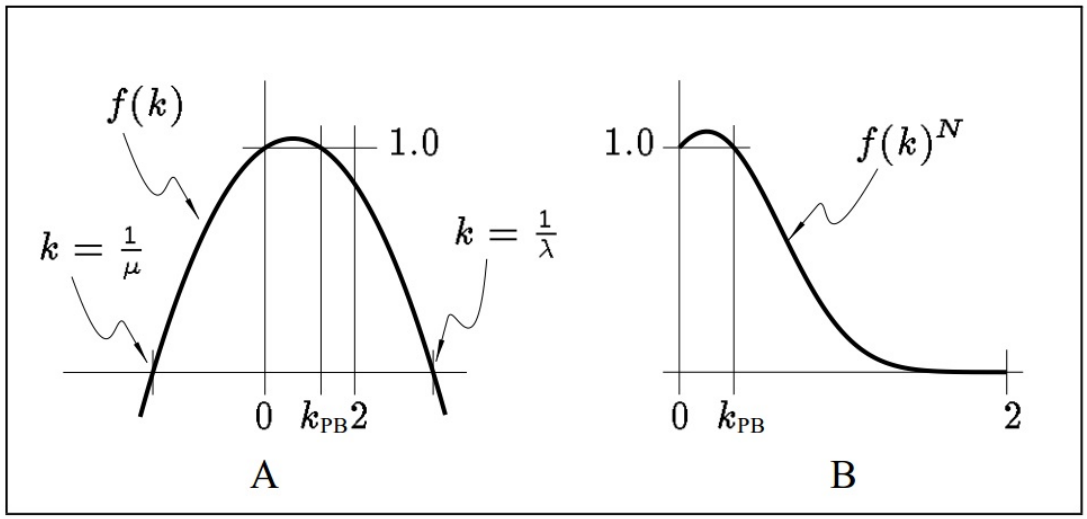
$$f(k) := (1 + \mu\, k)(1 + \lambda\, k) \quad , \quad \mu < 0 \ , \ \lambda > 0 \ , \ |\mu| > |\lambda|$$

The procedure is the following:

1. First, contract the surface $\rightarrow \lambda > 0$
2. Second, expand the smoothed surface again $\rightarrow \mu < 0 \ , \ |\mu| > |\lambda|$



[1] G. Taubin. "A signal processing approach to fair surface design". In: *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*. ACM. 1995, pp. 351–358.



Theorem (Taubin Smoothing[2])

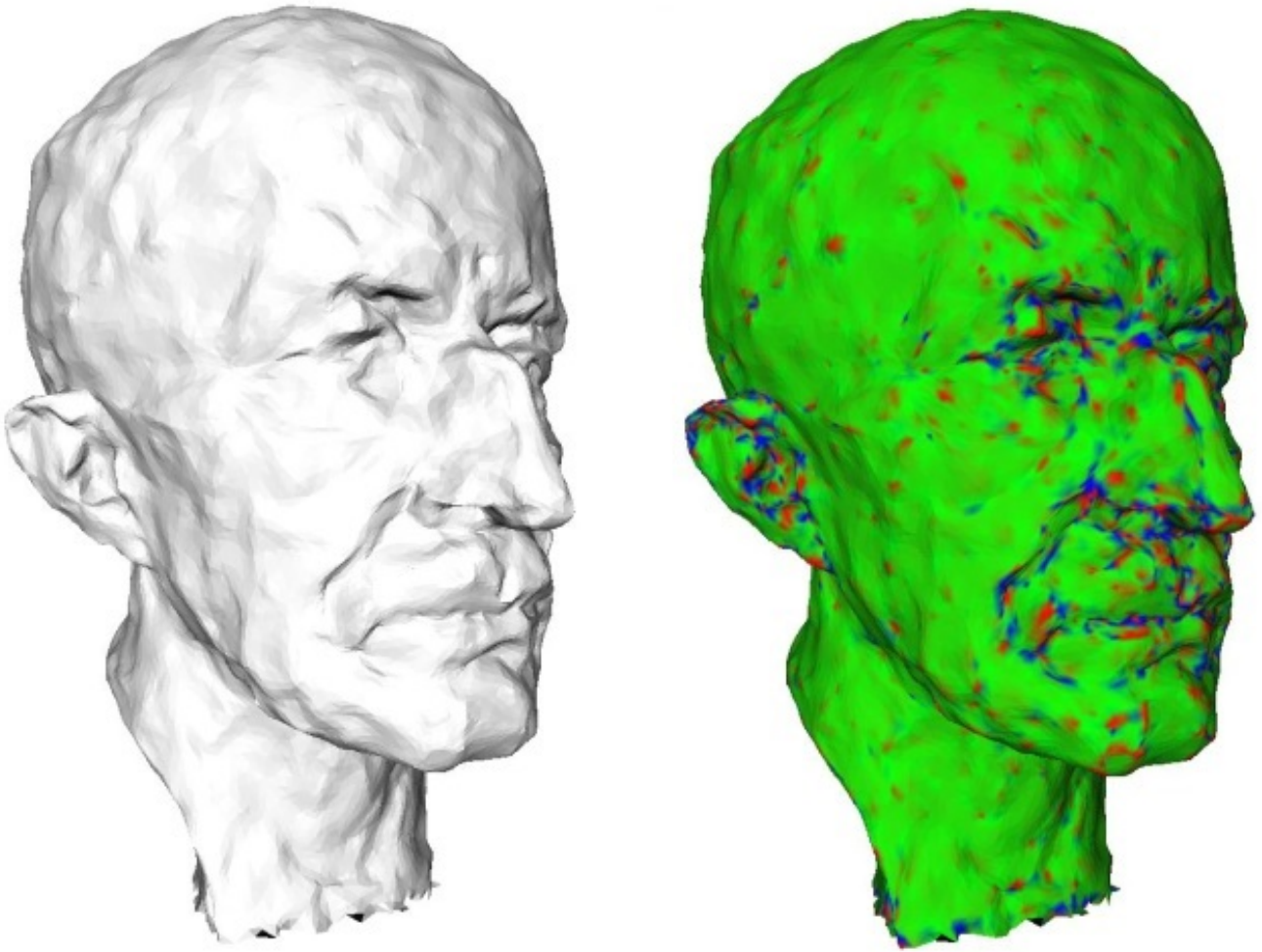
Given a mesh with vertices $\boldsymbol{V} = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n)^T$. *A smoothed version of this mesh can be computed iteratively by applying Taubin Smoothing:*

$$\boldsymbol{v}_i^{(t+1)} = ((\boldsymbol{I} + \mu \cdot \boldsymbol{L})(\boldsymbol{I} + \lambda \cdot \boldsymbol{L}))\, \boldsymbol{v}_i^{(t)}$$

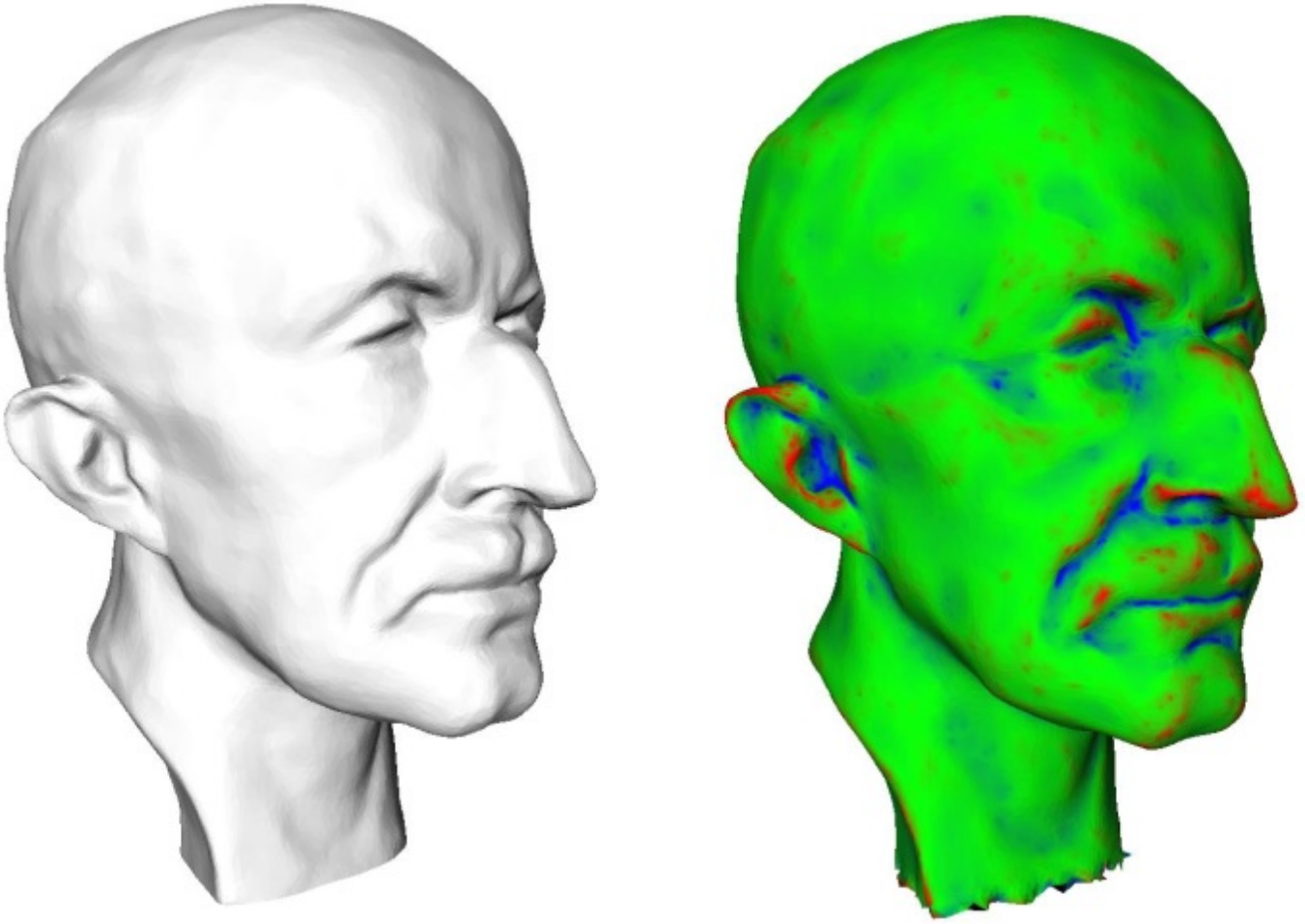
with $\mu < 0, \lambda > 0, |\mu| > |\lambda|$.

[2] G. Taubin. "A signal processing approach to fair surface design". In: *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*. ACM. 1995, pp. 351–358.

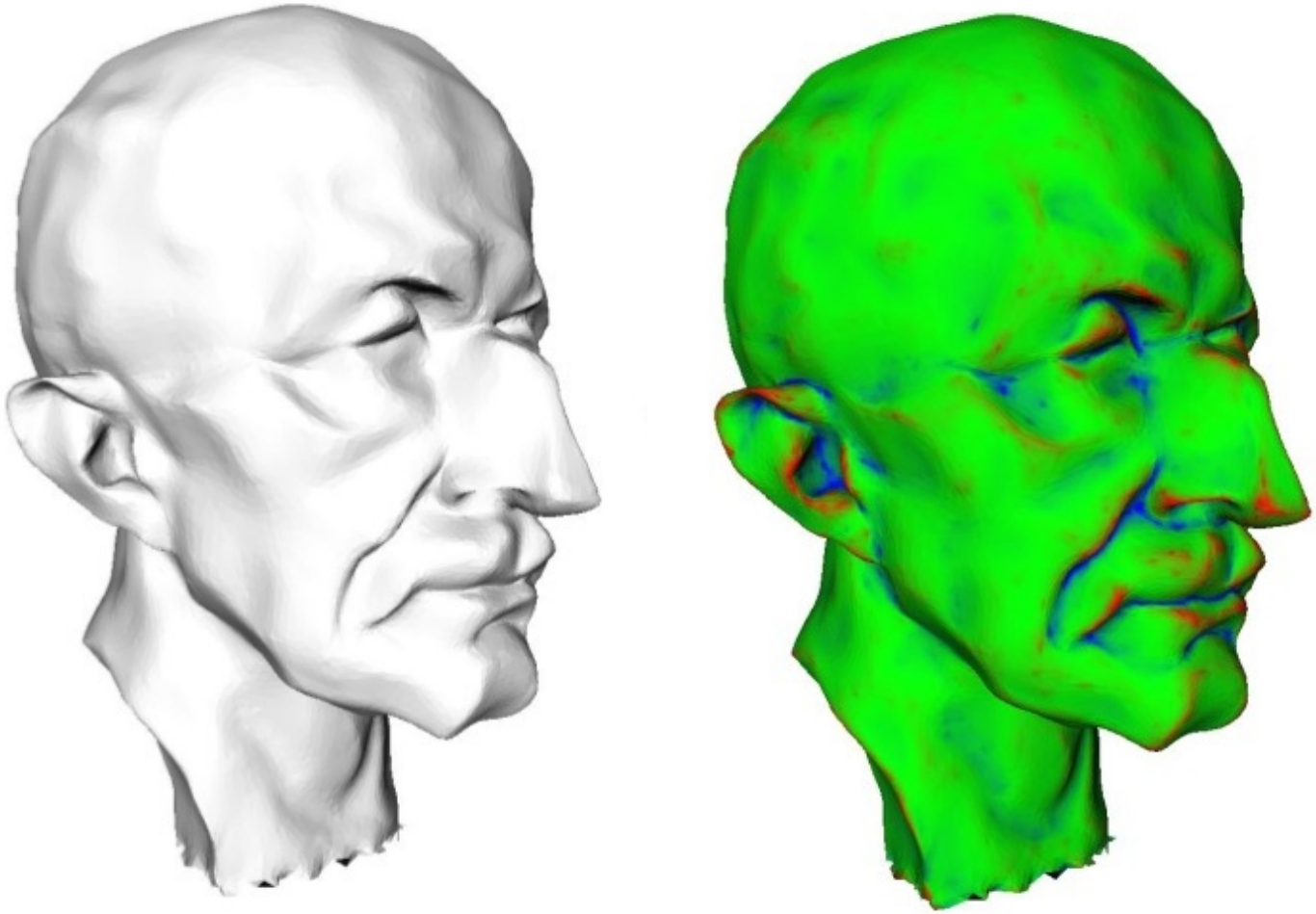
Results: 10 iterations



Results: 40 iterations



Results: 400 iterations



The Explicit Euler Step is derived by solving the diffusion equation for point $\boldsymbol{v}_i$

$$\frac{\partial}{\partial t}\rho(\boldsymbol{v}_i, t) = \lambda \cdot \Delta\rho(\boldsymbol{v}_i, t) \quad , \quad i = 1, \ldots, n$$

using finite differences for the vertex as a function of time $\boldsymbol{v}_i(t)$

$$\boldsymbol{v}_i(t + dt) = \boldsymbol{v}_i(t) + \lambda\, dt \cdot \boldsymbol{L}\, \boldsymbol{v}_i(t)$$

Idea: Approximate the derivative in the heat equation at the new mesh!

$$\frac{\partial}{\partial t}\rho(\boldsymbol{v}_i, t + dt) = \lambda \cdot \Delta\rho(\boldsymbol{v}_i, t + dt) \quad , \quad i = 1, \ldots, n$$

which can be written in matrix form

$$\frac{\partial}{\partial t}\boldsymbol{v}(t + dt) = \lambda \cdot \boldsymbol{L}\, \boldsymbol{v}(t + dt)$$

Using finite differences, we can approximate the left-hand side by (backward difference)

$$\frac{\partial}{\partial t}\boldsymbol{v}(t + dt) \approx \frac{\boldsymbol{v}(t + dt) - \boldsymbol{v}(t)}{dt}$$

to get the final equation:

$$\boldsymbol{v}(t + dt) = \boldsymbol{v}(t) + \lambda\, dt \cdot \boldsymbol{L}\, \boldsymbol{v}(t + dt)$$

$$\boldsymbol{v}(t+dt) = \boldsymbol{v}(t) + \lambda\, dt \cdot \boldsymbol{L}\, \boldsymbol{v}(t+dt)$$

This can be reformulated to an equation system in the form $\boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{b}$:

$$(\boldsymbol{I} - \lambda\, dt \cdot \boldsymbol{L})\, \boldsymbol{v}(t+dt) = \boldsymbol{v}(t)$$

Theorem (Implicit Euler Step[3])

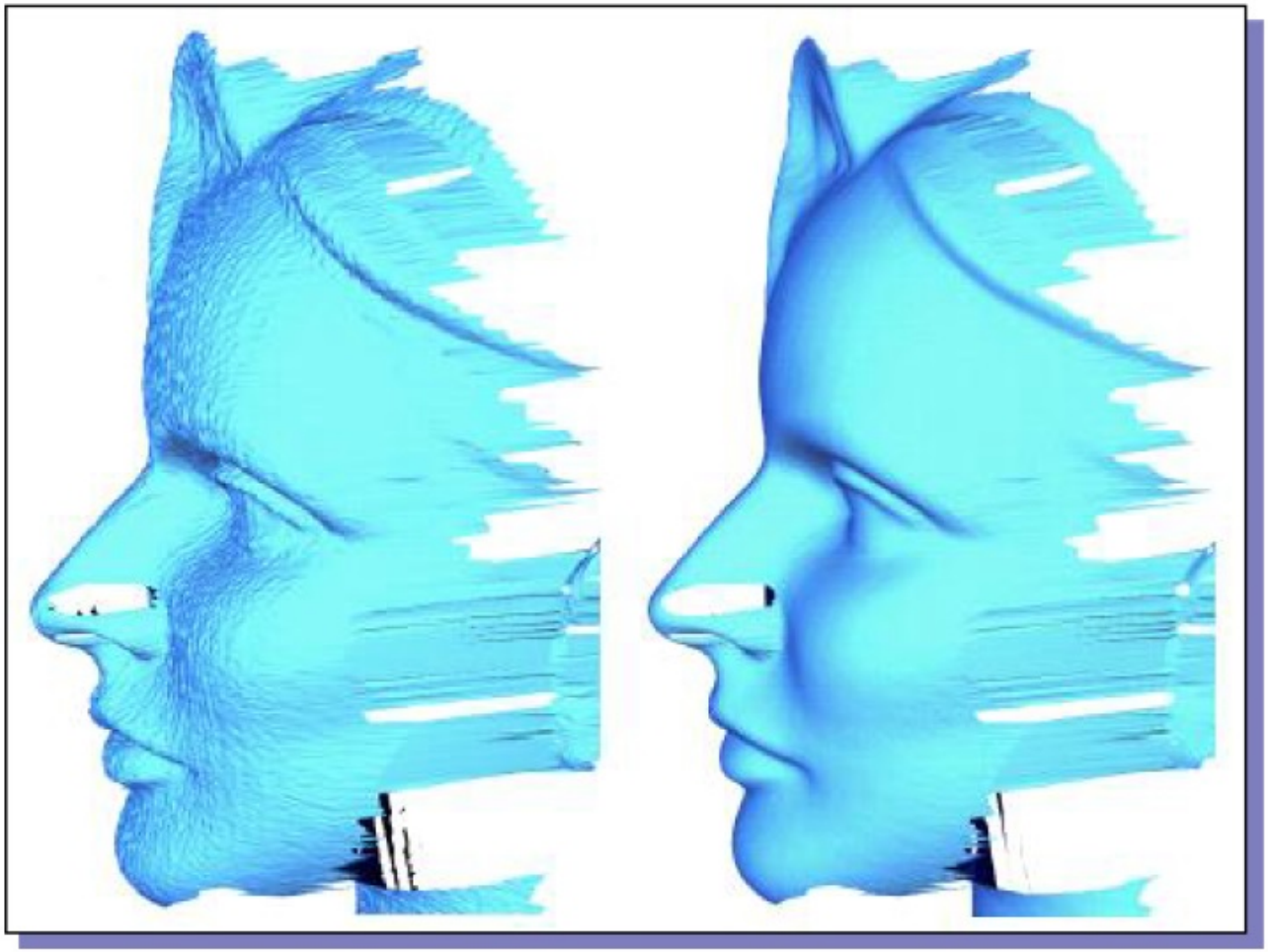
Given a mesh with vertices $\boldsymbol{V} = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n)^T$. A smoothed version of this mesh can be computed iteratively by applying the Implicit Euler Step:

$$(\boldsymbol{I} - \lambda \cdot \boldsymbol{L})\, \boldsymbol{v}_i^{(t+1)} = \boldsymbol{v}_i^{(t)}$$

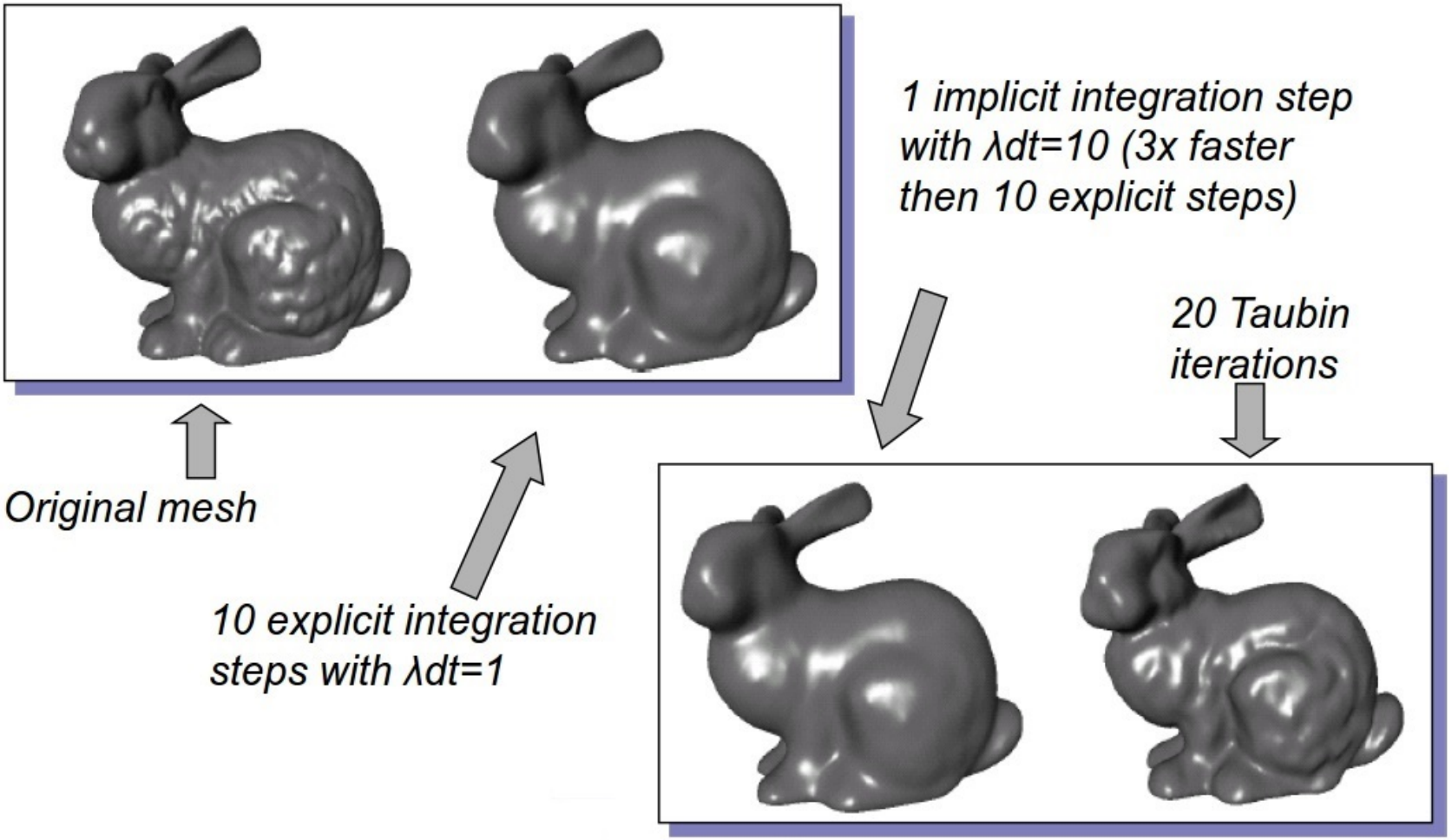
The Implicit Euler Step is much more stable than the Explicit Euler Step and can deal with large λ!

[3] M. Desbrun et al. "Implicit fairing of irregular meshes using diffusion and curvature flow". In: *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co. 1999, pp. 317–324.

Results: 2 iterations



Comparison:


1 implicit integration step with λdt=10 (3x faster then 10 explicit steps)
20 Taubin iterations
Original mesh
10 explicit integration steps with λdt=1

In general the Laplace operator is not symmetric but it can be factored into a diagonal and a symmetric matrix using the matrices $\boldsymbol{S}$ and $\boldsymbol{D}$ defined above:

$$\boldsymbol{L} = \boldsymbol{D}^{-1}\,\boldsymbol{S}$$

$\boldsymbol{L}$ is similar to the symmetric matrix

$$\boldsymbol{O} = \boldsymbol{D}^{-\frac{1}{2}}\,\boldsymbol{S}\,\boldsymbol{D}^{-\frac{1}{2}}$$

The matrix $\boldsymbol{O}$ is related to $\boldsymbol{L}$ by the following transformation

$$\boldsymbol{L} = \boldsymbol{D}^{-1}\,\boldsymbol{S} = \boldsymbol{D}^{-\frac{1}{2}}\,\boldsymbol{D}^{-\frac{1}{2}}\,\boldsymbol{S}\,\boldsymbol{D}^{-\frac{1}{2}}\,\boldsymbol{D}^{\frac{1}{2}} = \boldsymbol{D}^{-\frac{1}{2}}\,\boldsymbol{O}\,\boldsymbol{D}^{\frac{1}{2}}$$

Because of the similarity $\boldsymbol{L}$ and $\boldsymbol{O}$ have the same real eigenvalues: If $\boldsymbol{v}$ is an eigenvector of $\boldsymbol{O}$ with eigenvalue λ then $\boldsymbol{u} = \boldsymbol{D}^{-\frac{1}{2}}\,\boldsymbol{v}$ is an eigenvector of $\boldsymbol{L}$ with the same eigenvalue.

Note, that instead of computing eigenvectors of $\boldsymbol{O}$, one can solve the **generalized eigenvalue problem**:

$$\boldsymbol{S}\,\boldsymbol{v} = \lambda\,\boldsymbol{D}\,\boldsymbol{v}$$

- The eigenvalues and eigenvectors of the generalized problem are the same as for $\boldsymbol{O}$
- This observation is exploited to facilitate the computations of the eigenvectors of $\boldsymbol{L}$
- The eigenvectors of $\boldsymbol{O}$ are mutually orthogonal

This does not hold for the eigenvectors of $\boldsymbol{L}$ but if we define a scalar product by

$$\langle\, \boldsymbol{f} \,|\, \boldsymbol{g} \,\rangle_{\boldsymbol{D}} := \boldsymbol{f}^T\,\boldsymbol{D}\,\boldsymbol{g}$$

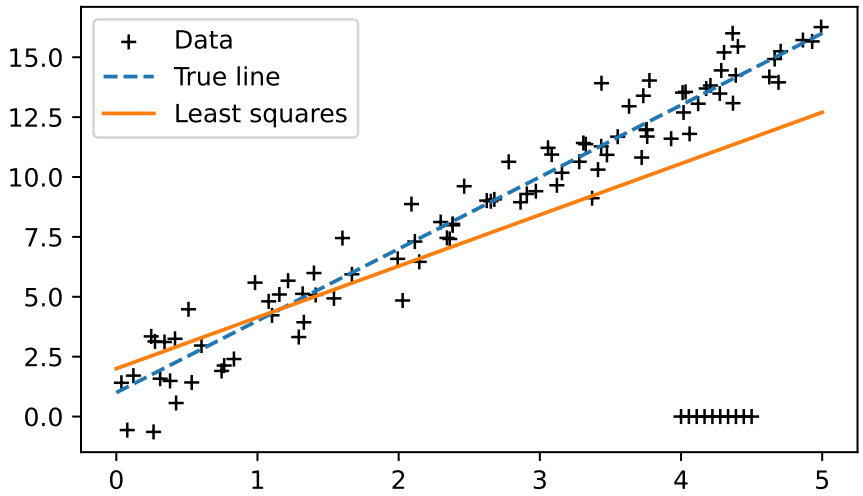
The eigenvectors of $\boldsymbol{L}$ are orthogonal with respect to this product:

$$\langle\, \boldsymbol{u}_i \,|\, \boldsymbol{u}_j \,\rangle_{\boldsymbol{D}} = \boldsymbol{u}_i^T\,\boldsymbol{D}\,\boldsymbol{u}_j = \boldsymbol{v}_i^T\,\boldsymbol{v}_j = \delta_{ij}$$

$$\bar{\boldsymbol{y}} = \arg\min_{\boldsymbol{y}} \sum_{i=1}^{N} |d(\boldsymbol{y}, \boldsymbol{x}_i)|^2$$

- Points with large distances contribute a lot to the error function.
- Standard least squares fit is not robust
- **Idea**: Use additional function to weight the contribution of each point, depending on their distance

  -> **M-Estimators**



- For a family of probability density functions p parameterized by θ:
  - A **maximum likelihood estimator, M-estimator** of θ maximizes the likelihood function over the parameter space.
  - For independent and identically distributed observations,the ML-estimate $\hat{\theta}$ satisfies:

$$\hat{\theta} = \arg\max_{\theta} \left(\prod_{i=1}^{n} p(x_i, \theta) \right)$$

  - Or equivalently:

$$\hat{\theta} = \arg\min_{\theta} \left(\sum_{i=1}^{n} -\log(p(x_i, \theta)) \right)$$

- ML-estimators are optimal in the limit of infinitely many observations, but may be biased for finite samples.

- For N independent observations $x_1, x_2, \ldots, x_N$ with probability density $p(x_i|\theta)$, the likelihood function and the log-likelihood are:

$$L(\theta) = \prod_{i=1}^{N} p(x_i|\theta) \qquad \log L(\theta) = \sum_{i=1}^{N} \log p(x_i|\theta)$$

- Assume that the data x_i follow a Gaussian (normal) distribution with mean $f(x_i;\theta)$ and fixed variance σ^2:

$$p(x_i|\theta) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - f(x_i;\theta))^2}{2\sigma^2}\right)$$

  The log-likelihood for N observations becomes:

$$\log L(\theta) = -\frac{N}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{N}(y_i - f(x_i;\theta))^2$$

- Maximizing the log-likelihood is equivalent to minimizing the negative log-likelihood. Ignoring constant terms, this reduces to minimizing:

$$\sum_{i=1}^{N}(y_i - f(x_i;\theta))^2$$

- Huber suggested to generalize the maximum likelihood estimation to the minimization of:

$$\sum_{i=1}^{n} \rho(x_i, \theta),$$

  where ρ is a function with certain properties.
- The solutions $\hat{\theta}$ are called M-estimators ("M" for "maximum likelihood-type"):

$$\hat{\theta} = \arg\min_{\theta} \left(\sum_{i=1}^{n} \rho(x_i, \theta) \right)$$

- This minimization can be done directly or by differentiating and solving for the root of the derivative.
- M-estimators of ψ-type if differentiation is possible, otherwise of ρ-type.
- The method of least squares is a prototypical M-estimator with

$$E_{L^2} = \sum_{i=1}^{n} ||(y_i - f(x_i)||^2$$

- Total energy to be minimized using Huber loss:

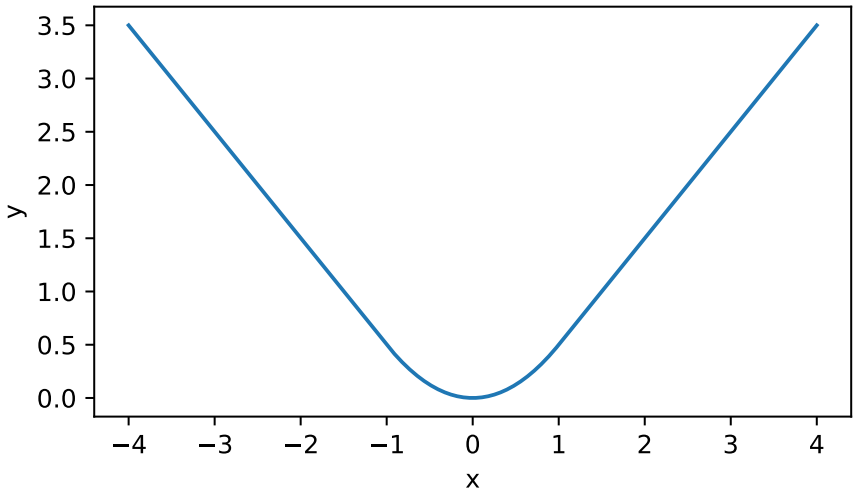
$$E_{\text{Huber}} = \sum_{i=1}^{n} L_{\text{Huber}}(y_i - f(x_i, \theta))$$

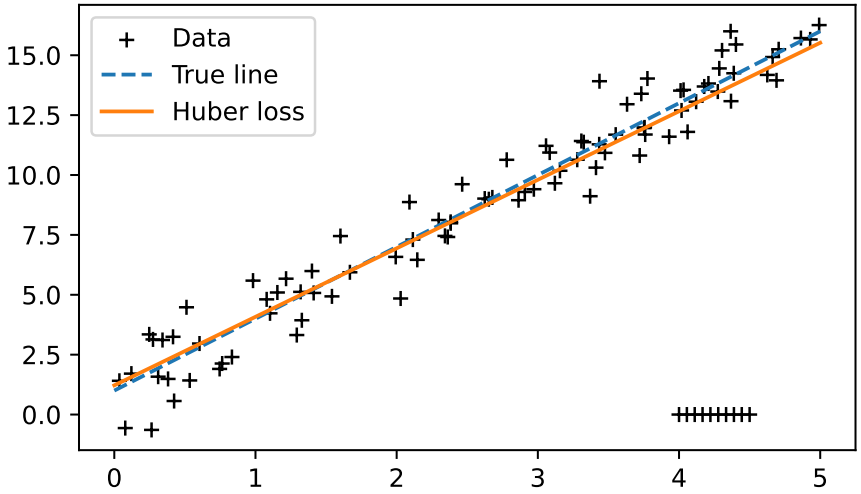
- Where L_{Huber} is the Huber loss function, y_i are observed values, $f(x_i, \theta)$ is the model prediction, and n is the number of data points.
- The Huber loss function combines L1 and L2 losses:

$$L_{\text{Huber}}(a) = \begin{cases} \frac{1}{2}a^2 & \text{for } |a| \leq \delta \\ \delta(|a| - \frac{1}{2}\delta) & \text{otherwise} \end{cases}$$

- Here, a is the residual, and δ is a threshold parameter.

$$\bar{\boldsymbol{y}} = \arg\min_{\boldsymbol{y}} \sum_{i=1}^{N} \rho(d(\boldsymbol{y}, \boldsymbol{x}_i), \sigma)$$



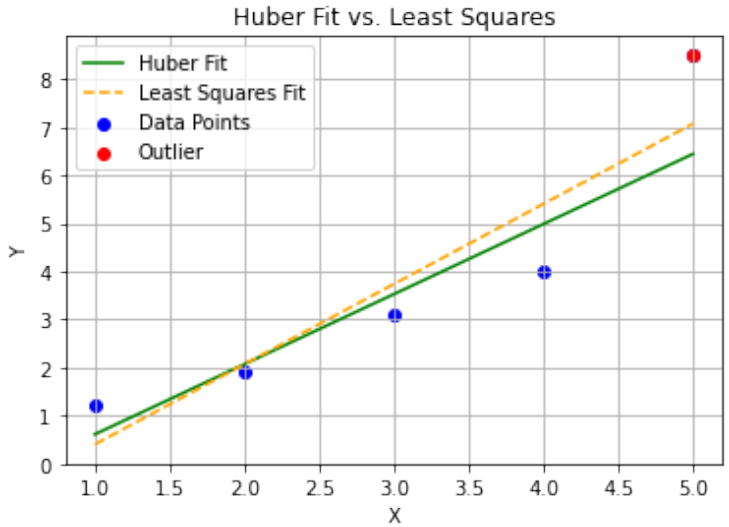


$$L_{\text{Huber}}(a) = \begin{cases} \frac{1}{2}a^2 & \text{for } |a| \leq \delta \\ \delta(|a| - \frac{1}{2}\delta) & \text{otherwise} \end{cases}$$

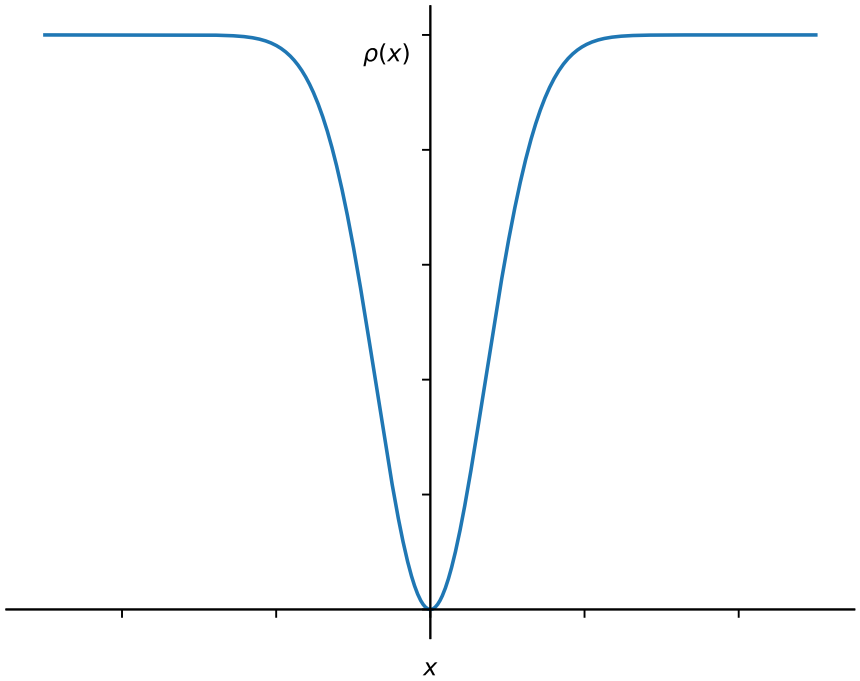
- Assume you have the following data points:
- X: [1, 2, 3, 4, 5]
- Y: [1.2, 1.9, 3.1, 4.0, 8.5] # Note the last point is an outlier
- Using a Huber loss function, fit a linear model to these points.
- The process involves minimizing the Huber loss, defined as:

$$\text{Huber loss} = \begin{cases} \frac{1}{2}(y - (ax + b))^2 & \text{for } |y - (ax + b)| \leq \delta \\ \delta \cdot (|y - (ax + b)| - \frac{1}{2}\delta) & \text{for } |y - (ax + b)| > \delta \end{cases}$$

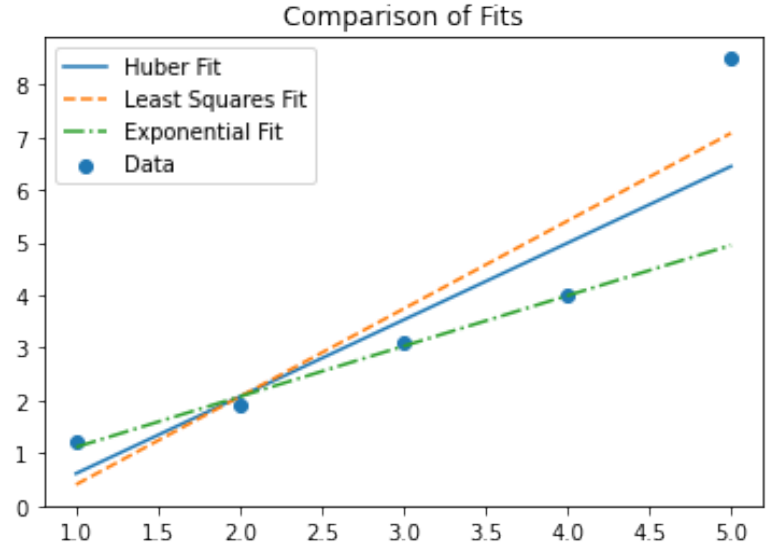
- Here, δ is a threshold parameter that determines when to switch from quadratic to linear loss.

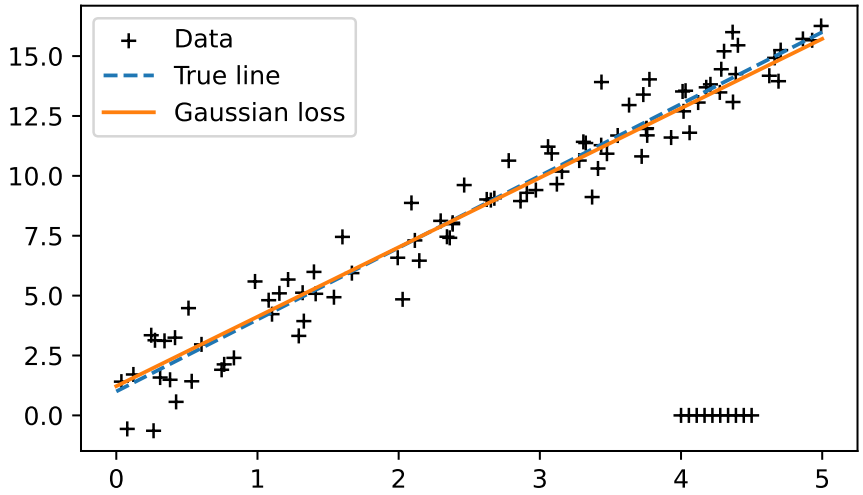


$$\bar{\boldsymbol{y}} = \arg\min_{\boldsymbol{y}} \sum_{i=1}^{N} \rho(d(\boldsymbol{y}, \boldsymbol{x}_i), \sigma)$$



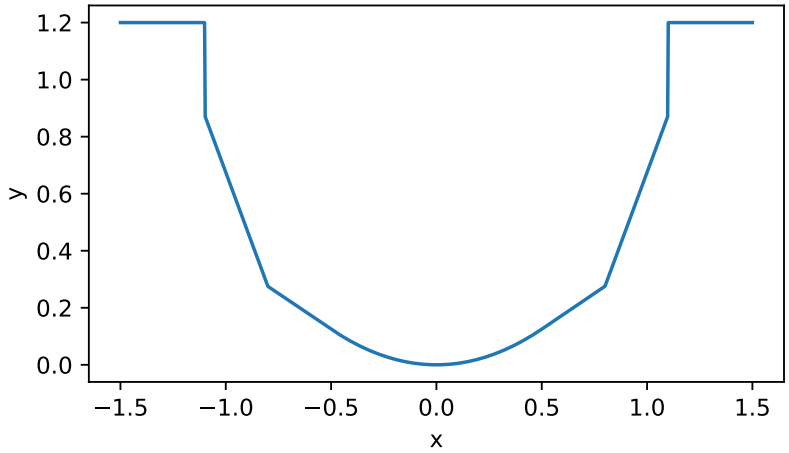
$$\rho(d, \sigma) = 1 - \exp\left(-\frac{d^2}{\sigma^2}\right)$$





- Total energy to be minimized using Hampel loss:

$$E_{\text{Hampel}} = \sum_{i=1}^{n} L_{\text{Hampel}}(y_i - f(x_i, \theta))$$

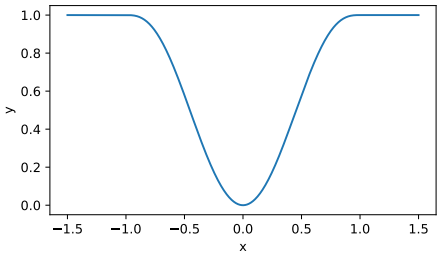


- With L_{Hampel} as the Hampel loss function.
- The Hampel loss function has three zones for different error sizes:

$$L_{\text{Hampel}}(a) = \begin{cases} \frac{1}{2}a^2 & \text{for } |a| \leq a_1 \\ a_1\left(|a| - \frac{1}{2}a_1\right) & \text{for } a_1 < |a| \leq a_2 \\ a_1(a_2 - \frac{1}{2}a_1) + c(|a| - a_2) & \text{for } a_2 < |a| \leq a_3 \\ \text{constant} & \text{for } |a| > a_3 \end{cases}$$

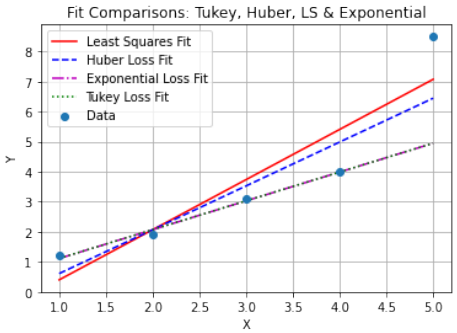
- With a_1, a_2, a_3 as transition parameters and c as a scaling factor.

- Total energy to be minimized using Tukey's Biweight loss:



$$E_{\text{Tukey}} = \sum_{i=1}^{n} L_{\text{Tukey}}(y_i - f(x_i, \theta))$$

- Where L_{Tukey} is Tukey's Biweight loss function.
- Tukey's biweight loss is bounded and reduces outliers' influence:



$$L_{\text{Tukey}}(a) = \begin{cases} \frac{c^2}{6}\left[1 - \left(1 - \left(\frac{a}{c}\right)^2\right)^3\right] & \text{for } |a| \leq c \\ \frac{c^2}{6} & \text{otherwise} \end{cases}$$

- Here, c is a tuning parameter, and a is the residual.

Observe diffusion in 2D images:

$$\frac{\partial}{\partial t}T = -\Delta T$$

Represent T using pixel intensity and discretize

$$I_s^{(t+1)} = I_s^{(t)} + \lambda \underbrace{\sum_{p \in \mathcal{N}(s)} I_p^{(t)} - I_s^{(t)}}_{\text{Laplacian}}$$

Edge conservation: Large intensity gradients get smaller weight

$$I_s^{(t+1)} = I_s^{(t)} + \lambda \sum_{p \in \mathcal{N}(s)} g\left(\left\| I_p^{(t)} - I_s^{(t)} \right\|\right)\left(I_p^{(t)} - I_s^{(t)}\right)$$

with

$$g(x) = \frac{1}{1 + \frac{x^2}{\sigma^2}} \text{ or } g(x) = \exp\left(-\frac{x^2}{\sigma^2}\right)$$

Construction of the bilateral filter

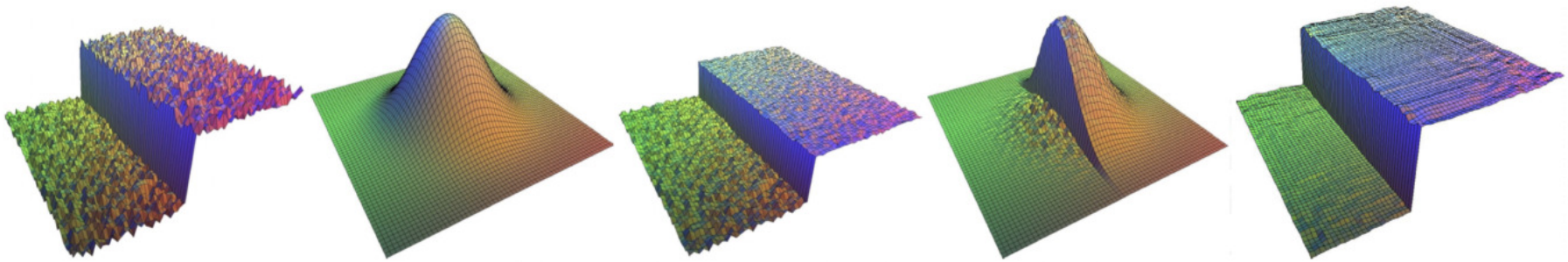
$$I_s^{(t+1)} = I_s^{(t)} + \lambda \sum_{p \in \Omega} \underbrace{f(|p-s|)}_{\text{Generalized distance function}} g\left(\left\|I_p^{(t)} - I_s^{(t)}\right\|\right)\left(I_p^{(t)} - I_s^{(t)}\right)$$

which results into the fix point

$$I'_s = \frac{1}{k(s)} \sum_{p \in \Omega} f(|p-s|) g(\|I_p - I_s\|) I_p$$

with

$$k(s) = \frac{1}{k(s)} \sum_{p \in \Omega} f(|p-s|) g(\|I_p - I_s\|)$$

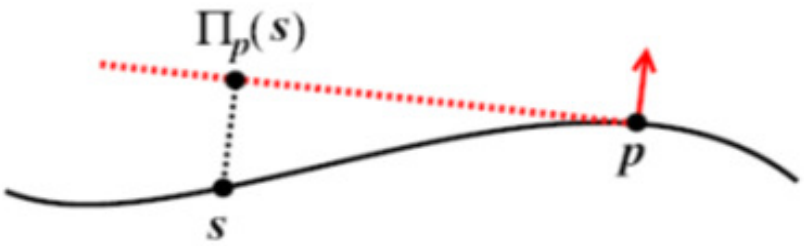


input | spatial kernel f | influence g in the intensity domain for the central pixel | weight $f \times g$ for the central pixel | output

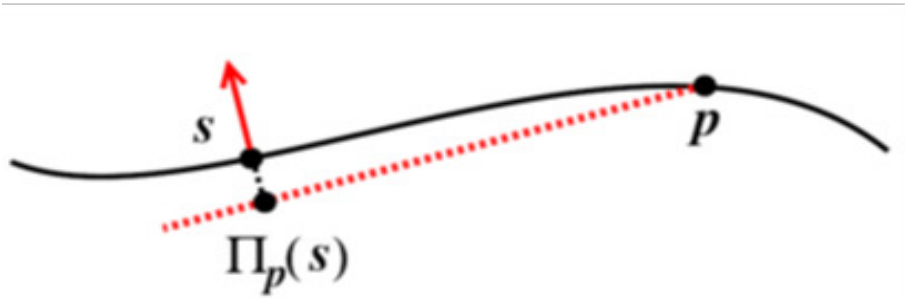
Transfer on 3D

$$\bar{\boldsymbol{s}} = \frac{1}{k(\boldsymbol{s})} \sum_{p \in X} f(\|\boldsymbol{p} - \boldsymbol{s}\|) g(\|\boldsymbol{\Pi_p}(\boldsymbol{s}) - \boldsymbol{s}\|) \boldsymbol{\Pi_p}(\boldsymbol{s})$$

with predictions $\boldsymbol{\Pi_p}(\boldsymbol{s})$ for the filtered position $\boldsymbol{s}$

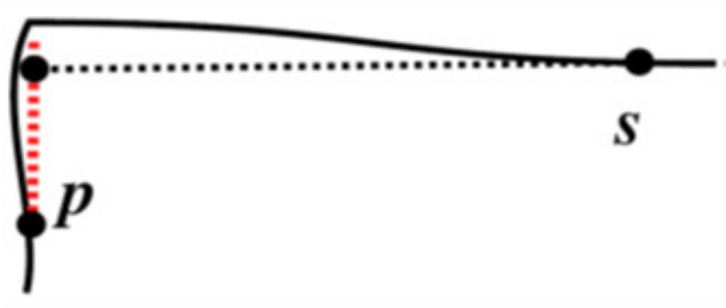


(a) Jones et al., 2003



(b) Fleishman et al., 2003

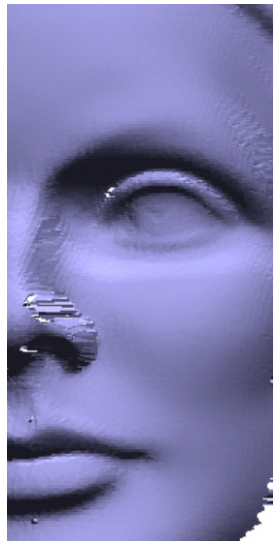
Effect of the estimator a corners



(a) Jones et al., 2003

(b) Fleishman et al., 2003

(a) Original



(b) Jones et al., 2003

(a) Original mesh

(b) Desbrun et al., 1999

(c) Fleishman et al., 2003

Two-stage methods consisting of normal filtering and vertex position updates are able to address challenges of surface denoising.

Key Aspects of Normal Filtering

- Normal filtering leads to effective noise removal and feature preservation.
- Linear and non-linear filter methods are effective in different aspects.
- The **Robust Statistics Framework** offers an overview and comparability of filters within the robust statistics framework.

Let T be a triangle with unit normal $\boldsymbol{n}(T)$, area $A(T)$ and a neighborhood $\mathcal{N}(T)$ (sharing an edge or vertex with T).

The two step method

1. Smooth normals, e.g. by

$$\hat{\boldsymbol{m}}(T) = \sum_{S \in \mathcal{N}} A(S)\boldsymbol{n}(S) \qquad \boldsymbol{m}(T) = \frac{\hat{\boldsymbol{m}}(T)}{\|\hat{\boldsymbol{m}}(T)\|}$$

2. Adjust surface to normals

$$E_{\text{fit}}(P) = \sum A(T)\|\boldsymbol{n}(T) - \boldsymbol{m}(T)\|^2, \qquad P_{\text{new}} = \arg\min_P E_{\text{fit}}(P)$$

   This step can efficiently be solved using conjugate gradients.

3. Iterate Steps 1 and 2[4].

Challenge: how to deal with outliers and how to preserve sharp features?

[4] Y. Belyaev and H. Seidel. "Mesh smoothing by adaptive and anisotropic Gaussian filter applied to mesh normals". In: *Vision, modeling, and visualization*. IOS press Amsterdam, The Netherlands. 2002, pp. 203–210.

Vertex Normal Filtering versus Face Normal Filtering

In terms of sharp feature preservation, vertex normal filtering will not be as effective as face normal filtering because of the following reasons:

- The vertex normals of a mesh are usually derived from face normals. Therefore, processing face normals will avoid the ill-posedness and increase the robustness of the algorithm.
- At a sharp feature, the angle between vertex normals is smaller than the angle between the face normals. Therefore, face normals are more robust in feature-preservation compared to vertex normals.

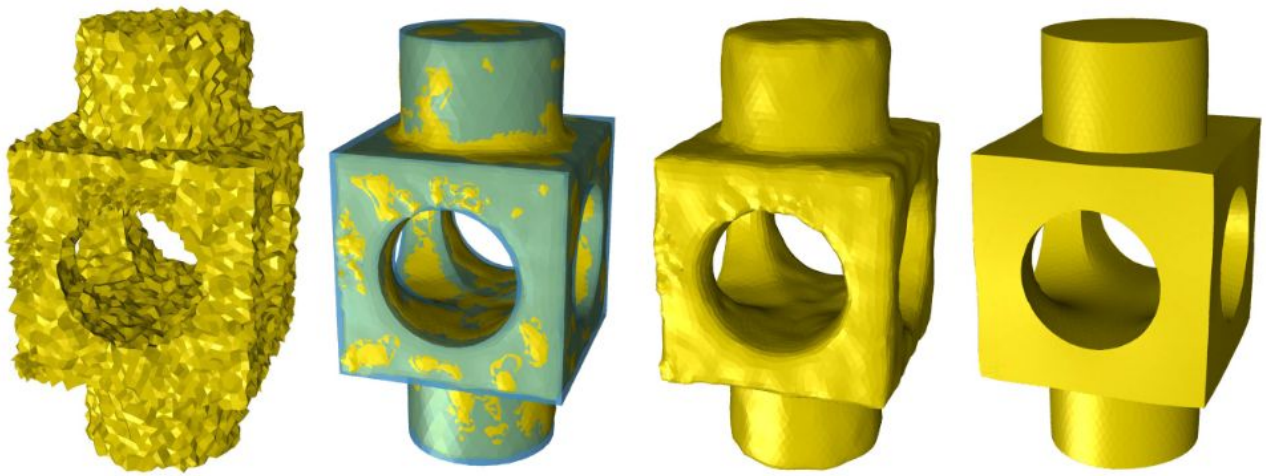


Figure: Visual comparison between vertex position, vertex normal, and face normal filtering methods (from Yadav 2020).

Robust Statistics applied to surface denoising identifies noise and mesh features as outliers in a statistical data modelling process Yadav et al., 2021.

Use robust error norms $\rho_\sigma(x)$ in the form of M-Estimators to handle outliers, obtained by minimizing

$$E_\sigma(\hat{n}) = \sum_{n \in \Omega} \rho_\sigma(\hat{n} - n)$$

through noise-free normal estimate $\hat{n}$ from noisy normals n.

This equation can be extended to take into account spatial weights in local neighborhoods using the following formulation:

$$E_{\sigma,\sigma_d}(\hat{n}) = \sum_{n \in \Omega} \rho_\sigma(\hat{n} - n) f_{\sigma_d}(d)$$

Isotropic weighting factor $f_{\sigma_d}(d)$ to take spatial weights in local neighbourhood into account.

Generalization of maximum likelihood estimators, consisting of several robust error norms $\rho_\sigma(x)$ with corresponding influence functions $\psi_\sigma(x)$ and weighting functions $g_\sigma(x)$.

Properties for Surface Denoising

- The error norm $\rho_\sigma(x)$ should not increase rapidly to minimize the effect of outliers.
- The influence function $\psi_\sigma(x) = \rho'_\sigma(x)$ should be bounded and redescending to preserve sharp features.
- The weighting function $g_\sigma(x) = \frac{\psi_\sigma(x)}{x}$ captures anisotropic behaviour.

Filtered face normals $\hat{n}_i$ are computed by a weighted average over neighbouring normals n_j:

$$\hat{n}_i = \frac{1}{w} \sum_{j \in \Omega_i} g_\sigma(x) f_{\sigma_d}(d) n_j$$

where:

- $g_\sigma(x)$ is an anisotropic weighting function.
- $f_{\sigma_d}(d)$ is an isotropic weighting function.
- Ω_i is the mesh neighbourhood.
- $w = \sum_{j \in \Omega_i} g_\sigma(x) f_{\sigma_d}(d) n_j$ ensures unit length normals.

Vertex positions $\mathbf{p}_i'$ are adjusted based on filtered normals $\hat{n}$ by computing an area-weighted average of projections of the vertex onto the modified surface [Ohtake 2001]:

$$\mathbf{p}_i' = \frac{1}{\sum_{j \in \Omega_i} A_j} \sum_{j \in \Omega_i} A_j (\mathbf{p}_i + \hat{n}_j (\hat{n}_j \cdot (\mathbf{c}_j - \mathbf{p}_i)))$$

where:

- $\mathbf{p}_i + \hat{n}_j (\hat{n}_j \cdot (\mathbf{c}_j - \mathbf{p}_i))$ represents the vertex projection.
- $\hat{n}_j (\mathbf{p} - \mathbf{c}_j) = 0$ describes the modified plane based on filtered normals.

Unilateral Normal Filters:

- Only use anisotropic weighting function $g_\sigma(x)$ and do not make use of spatial filter $f_{\sigma_d}(d)$.
- Remove low levels of noise and preserve sharp features.
- Not robust against moderate to high levels of noise.

Bilateral Normal Filters:

- Use both weighting functions $g_\sigma(x)$ and $f_{\sigma_d}(d)$ in the form of a Gaussian range and spatial kernel.
- Effective for enhancing features and robust against significant noise.

Different feature values and spatial distances x, d for weighting functions $g_\sigma(x)$ and $f_{\sigma_d}(d)$:

- Euclidean distance $\|n_i - n_j\|^2, \|c_i - c_j\|^2$.
- Angle $\angle n_i, n_j, \angle c_i, c_j$.
- Arccosine $\arccos(n_i \cdot n_j), \arccos(c_i \cdot c_j)$.

User input σ and σ_d adapt the filters to the given geometry and are chosen depending on resolution, amount of noise, and curvature.

- Zheng et al. suggest $\sigma \in [0.1, 0.5]$ and $\sigma_d \in [0.01, 0.5]$ [Zheng 2017].
- σ_d set as the average distance between neighbouring face centres.

Remember definitions: $\psi_\sigma(x) = \rho'_\sigma(x)$ and $g_\sigma(x) = \frac{\psi_\sigma(x)}{x}$

- L_2-**norm:**

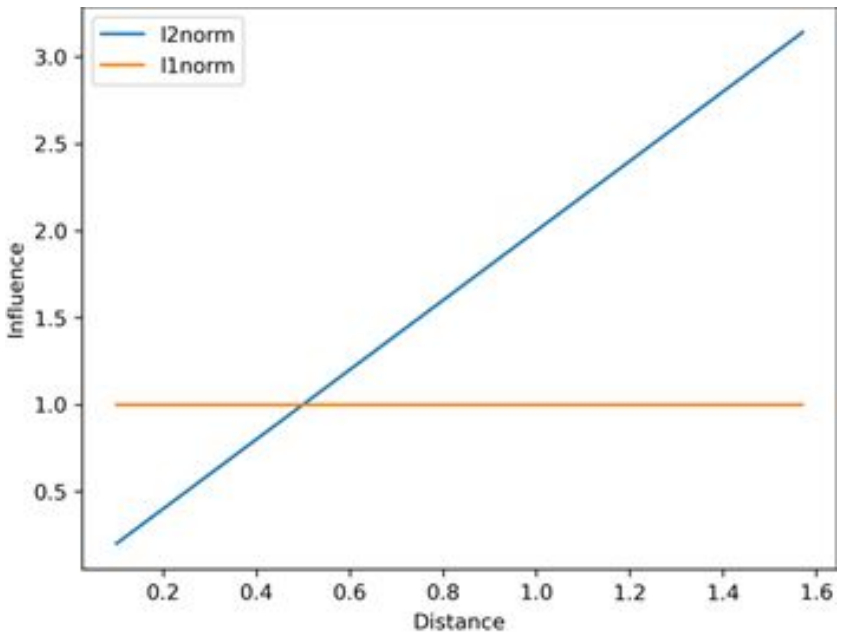
$$\rho_\sigma(x) = x^2$$

  The resulting weighting function $g_\sigma(x) = 2$.

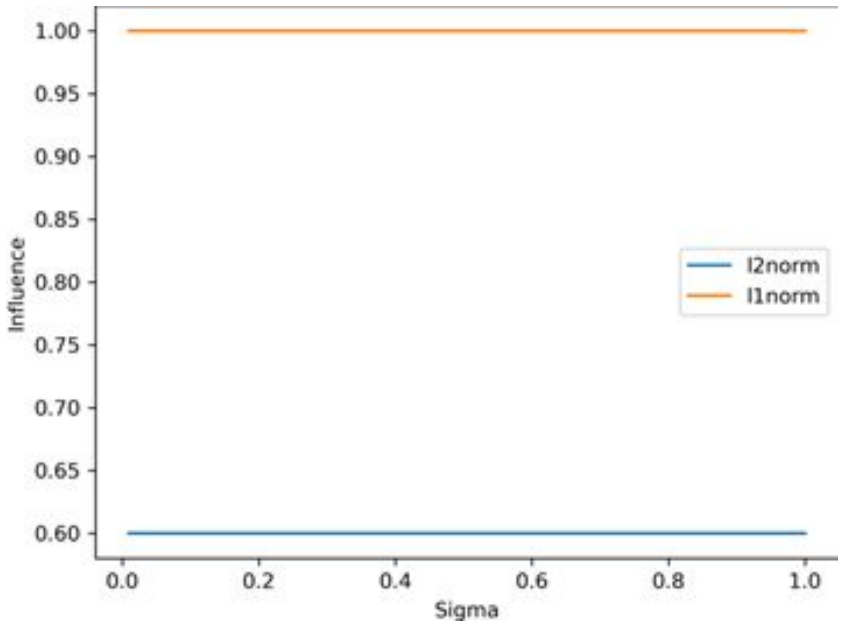
- L_1-**norm:**

$$\rho_\sigma(x) = |x|$$

  The resulting weighting function $g_\sigma(x)$ is set to $1/x$.



(a) $\Psi_\sigma(x)$ with respect to x



(b) $\Psi_\sigma(x)$ with respect to σ

- **Truncated L_2-norm:**

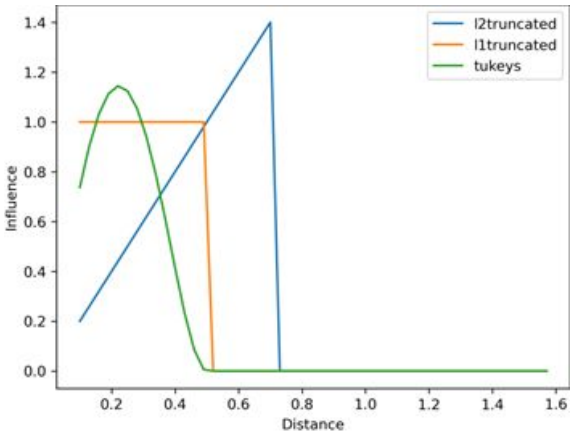
$$g_\sigma(x) = \begin{cases} 2 & \text{if } x < \sigma \\ 0 & \text{otherwise} \end{cases}$$

- **Truncated L_1-norm:**

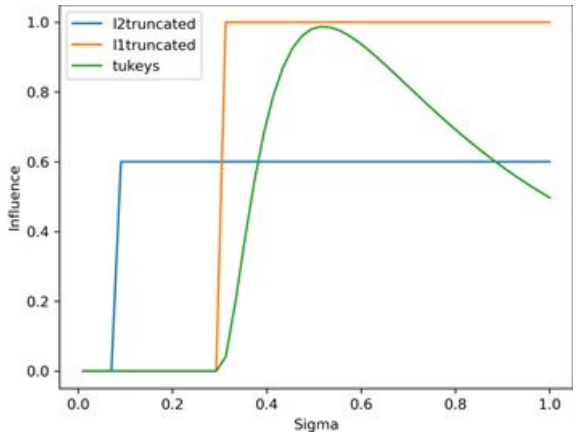
$$g_\sigma(x) = \begin{cases} 1/x & \text{if } x < \sigma \\ 0 & \text{otherwise} \end{cases}$$

- **Tukey's norm:**

$$g_\sigma(x) = \begin{cases} \frac{1}{2}\left(1 - \left(\frac{x}{\sigma}\right)^2\right)^2 & \text{if } x < \sigma \\ 0 & \text{otherwise} \end{cases}$$



(a) $\Psi_\sigma(x)$ with respect to x



(b) $\Psi_\sigma(x)$ with respect to σ

- **Huber's Minimax:**

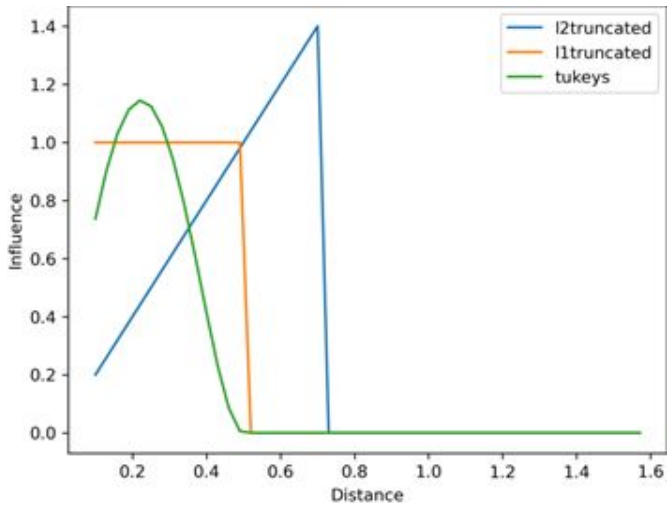
$$g_\sigma(x) = \begin{cases} \frac{1}{\sigma} & \text{if } x < \sigma \\ \frac{1}{x} & \text{otherwise} \end{cases}$$

- **Lorentzian Norm:**

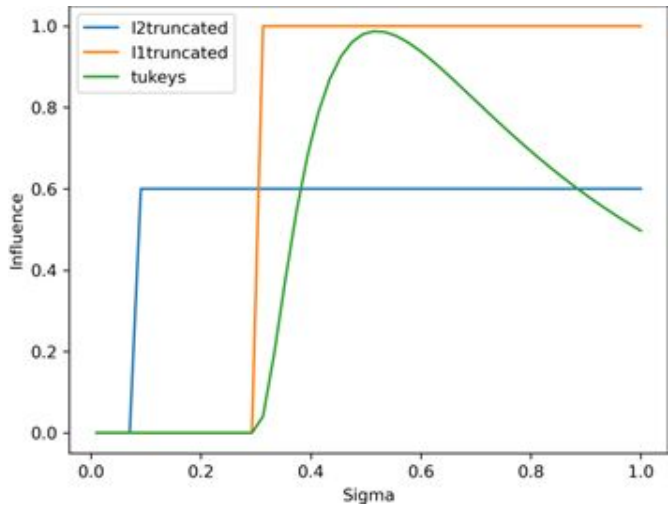
$$g_\sigma(x) = \frac{2x}{x^2 + 2\sigma^2}$$

- **Gaussian Norm:**

$$g_\sigma(x) = e^{-\frac{x^2}{\sigma^2}}$$

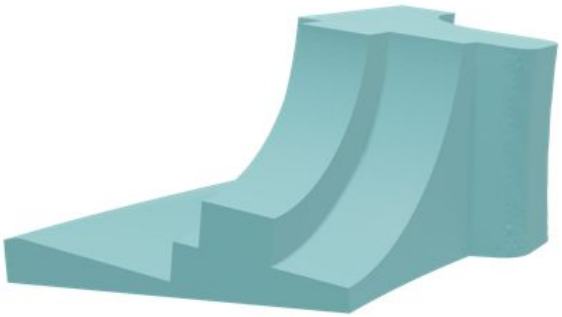


(a) $\Psi_\sigma(x)$ with respect to x



(b) $\Psi_\sigma(x)$ with respect to σ

Experiments: Unilateral filtes

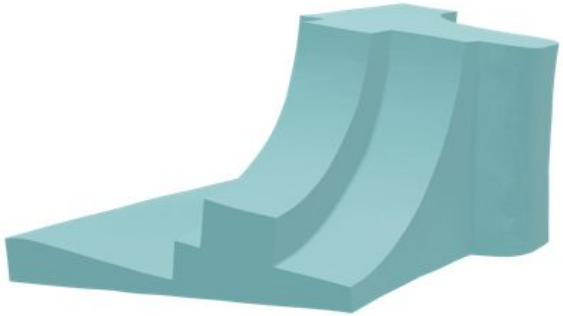


a) Original

b) Moderate noise

c) L_2-norm

d) L_1-norm



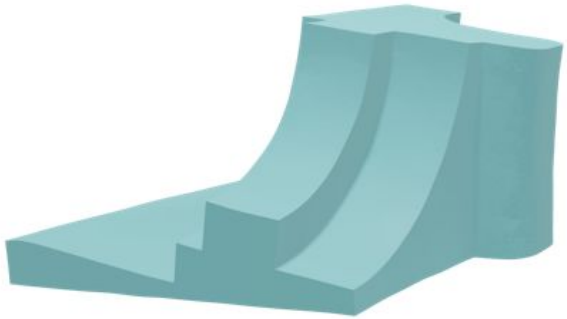
e) Truncated L_2-norm

f) Gaussian norm

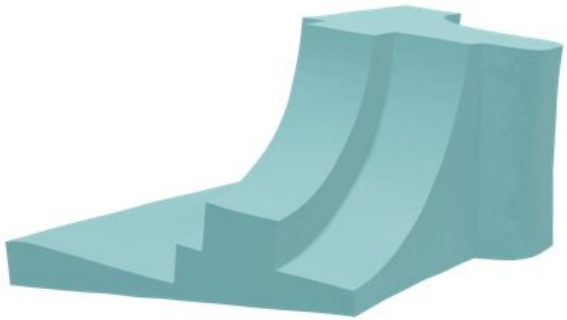
Experiments: Unilateral filtes

a) Truncated L_2-norm for $\sigma = 0.5$

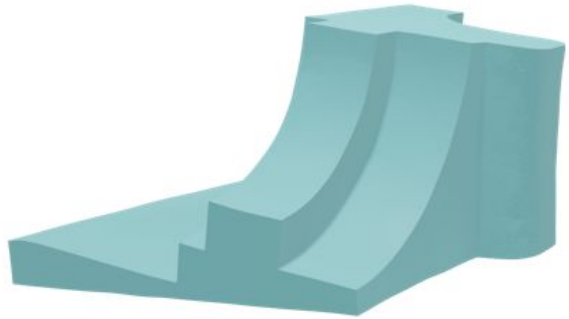
b) Truncated L_2-norm for $\sigma = 2$



c) Truncated L_2-norm for $x = \|n_i - n_j\|^2$



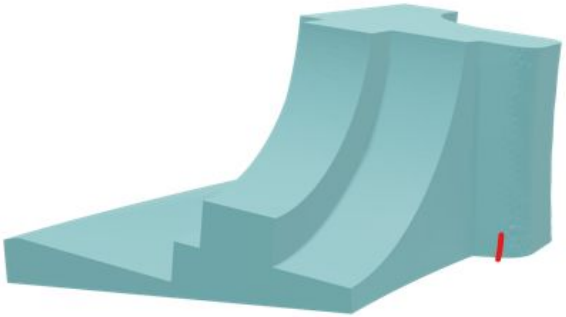
d) Truncated L_2-norm for $x = \arccos(n_i \cdot n_j)$



e) Truncated L_2-norm for noise $\sigma_n = 0.3$

f) Truncated L_2-norm for noise $\sigma_n = 1$

Experiments: Unilateral filtes



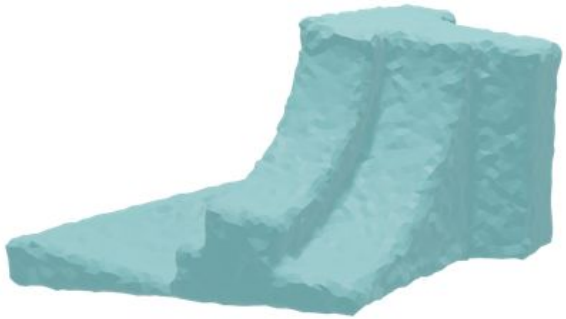
a) Original

b) Moderate noise

c) L_1-norm & L_2-norm

d) Truncated L_1-norm & L_2-norm

e) Gaussian norm & Gaussian norm



f) Huber's minimax & Tukey's norm

Experiments: Unilateral filtes

a) Original

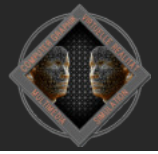
b) Moderate noise

c) L_1-norm & L_2-norm

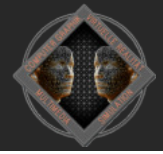
d) Truncated L_1-norm & L_2-norm

e) Gaussian norm & Gaussian norm

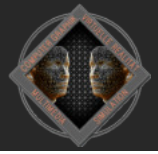
f) Huber's minimax & Tukey's norm



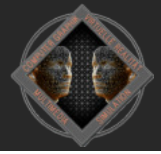
Results: L_2L_1-norm



- average noise removal
- L_2-norm not useful for preserving features due to unbounded and non re-descending influence function
- L_1-norm better at enhancing features due to bounded influence function
- independent of user input $\sigma \rightarrow$ more useful as spatial filters for additional noise removal



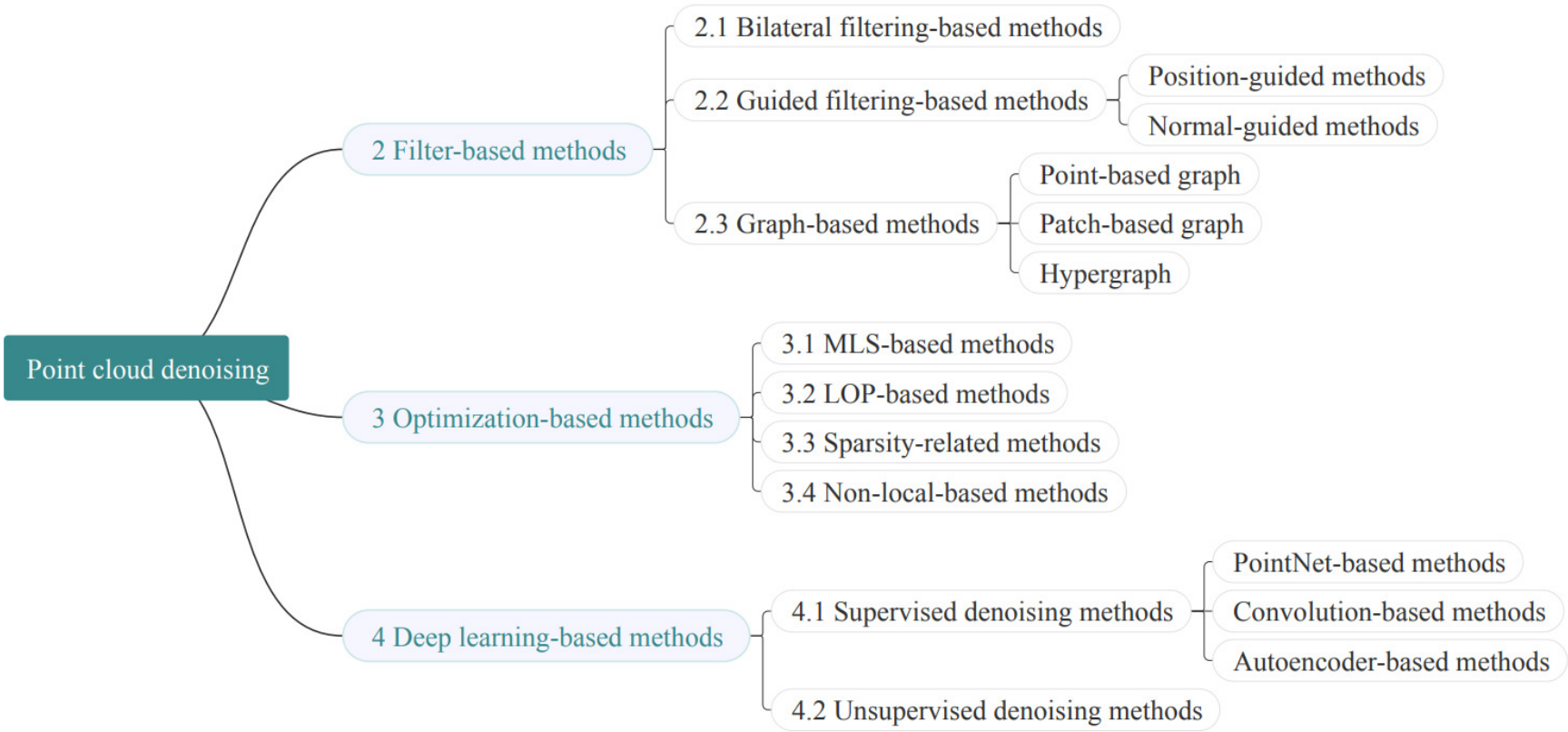
Results: Truncated L_2L_1-norm & Tukey's norm



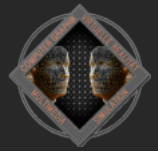
- very good at preserving features due to re-descending influence functions' sharper cut off
- robust against low to moderate levels of noise for angular and less detailed meshes
- sensitive to noise on detailed meshes with curvature
- not suitable as spatial filters

- **Huber's Minimax, Lorentzian, & Gaussian Norms:**
  - Good at noise removal due to redescending influence functions but average in feature preservation.
  - Tend to smooth over sharp features of angular meshes.
  - Robust against low to moderate levels of noise, especially effective for meshes with more curvature and details.
  - Ideal as spatial filters and demonstrate solid performance on non-angular meshes.

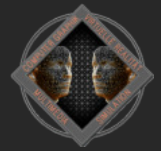
Current state of mesh/point cloud smoothing[5]



[5] L. Zhou et al. "Point cloud denoising review: from classical to deep learning-based approaches". In: *Graphical Models* 121 (2022), p. 101140.



References I



[1] Y. Belyaev and H. Seidel. "Mesh smoothing by adaptive and anisotropic Gaussian filter applied to mesh normals". In: *Vision, modeling, and visualization*. IOS press Amsterdam, The Netherlands. 2002, pp. 203–210.

[2] M. Desbrun, M. Meyer, P. Schröder, and A. H. Barr. "Implicit fairing of irregular meshes using diffusion and curvature flow". In: *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co. 1999, pp. 317–324.

[3] S. Fleishman, I. Drori, and D. Cohen-Or. "Bilateral mesh denoising". In: *ACM SIGGRAPH 2003 Papers*. 2003, pp. 950–953.

[4] T. R. Jones, F. Durand, and M. Desbrun. "Non-iterative, feature-preserving mesh smoothing". In: *ACM SIGGRAPH 2003 Papers*. 2003, pp. 943–949.

[5] G. Taubin. "A signal processing approach to fair surface design". In: *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*. ACM. 1995, pp. 351–358.

[6] S. K. Yadav, M. Skrodzki, E. Zimmermann, and K. Polthier. “Surface denoising based on normal filtering in a robust statistics framework”. In: *Proceedings of the Forum" Math-for-Industry" 2018: Big Data Analysis, AI, Fintech, Math in Finances and Economics*. Springer. 2021, pp. 103–132.

[7] L. Zhou, G. Sun, Y. Li, W. Li, and Z. Su. “Point cloud denoising review: from classical to deep learning-based approaches”. In: *Graphical Models* 121 (2022), p. 101140.