

University of West Bohemia
Faculty of Applied Sciences
Department of Computer Science and Engineering

Master thesis

Application of data dependent discrete Laplacian

Místo této strany bude
zadání práce.

Declaration

I hereby declare that this master thesis is completely my own work and that I used only the cited sources.

Plzeň, 15th May 2018

Jan Dvořák

Acknowledgement

Author of this thesis would like to thank his supervisor, Doc. Ing. Libor Váša Ph.D., for his guidance, valuable comments on the topic and provision of source codes of Curvature and Compression benchmarking software.

This work was supported by the Ministry of Education, Youth and Sports of the Czech Republic, project SGS 2016-013 Advanced Graphical and Computing Systems and institutional research support (1311).

Abstract

Discrete Laplace operator has wide spectrum of applications in the mesh processing, for example in smoothing, parameterization, editing and compression. In the latter, Váša et al. [44] have shown, that using geometric discrete Laplace operator results in residual entropy reduction, when compressing dynamic meshes. To generalize the ideas of their work, a new type of discrete Laplace operator, which should reduce the entropy even further, is proposed in this thesis. Properties of such Laplacian are studied. It is also applied in various mesh processing techniques and results are discussed.

Abstrakt

Diskrétní Laplaceův operátor má široké spektrum využití při zpracování trojúhelníkových sítí, například při vyhlazování, parametrizaci, editaci a kompresi. V posledním zmiňovaném, Váša et al. [44] ukázali, že s využitím geometrického diskrétního Laplaceova operátoru lze dosáhnout snížení reziduální entropie v případě dynamických trojúhelníkových sítí. V této práci je navržen, jako zobecnění jejich práce, nový diskrétní Laplaceův operátor, který by měl danou reziduální entropii snížit ještě více. Vlastnosti takového Laplaciánu jsou studovány. Je také aplikován v různých technikách zpracování trojúhelníkových sítí, a výsledky jsou diskutovány.

Contents

1	Introduction	1
2	Laplace operator	3
2.1	Discrete Laplace operator	3
2.2	Properties	5
2.2.1	Sparsity	5
2.2.2	Singular matrix	7
2.2.3	Symmetry	8
2.2.4	Positive weights	9
2.2.5	Linear precision	9
2.2.6	Locality	10
2.2.7	Unit sum of weights	10
2.2.8	Definiteness	11
2.2.9	Eigenvalues and eigenvectors	11
2.3	Types of discrete Laplace operator	12
2.3.1	Kirchhoff Laplacian	12
2.3.2	Tutte Laplacian	14
2.3.3	Cotangent Laplacian	14
2.3.4	Mean Value Laplacian	16
3	Laplacian mesh processing	18
3.1	Mean curvature estimation	18
3.2	Smoothing	19
3.2.1	Filtering using manifold harmonics	19
3.2.2	Iterative smoothing	21
3.2.3	$\lambda \mu$ smoothing	23
3.2.4	Implicit fairing using diffusion flow	24
3.2.5	Dynamic mesh smoothing	25
3.2.6	Suitable Laplacians for Smoothing	26
3.3	Parameterization	27
3.4	Mesh editing	29
3.5	Mesh morphing	32
3.6	Least-squares meshes	34

4	Mesh compression	36
4.1	Connectivity coding	36
4.2	Geometry coding	37
4.2.1	Static mesh compression	38
4.2.2	Dynamic mesh compression	40
4.3	Error metrics	42
5	Data Dependent discrete Laplace operator	44
5.1	Theoretical background	44
5.2	Construction	44
5.3	Properties	47
5.4	Weight coding	49
5.5	Possible advantages	50
6	Experimental results	51
6.1	Verification of properties	51
6.1.1	Linear precision	51
6.1.2	Definiteness	53
6.2	Application in Laplacian mesh processing	54
6.2.1	Mean curvature estimation	54
6.2.2	Smoothing	57
6.2.3	Parameterization	66
6.2.4	Mesh editing	70
6.2.5	Mesh morphing	74
6.2.6	Least-squares meshes	78
6.3	Dynamic mesh compression	82
6.3.1	Entropy	82
6.3.2	Rate-Distortion curve	83
6.3.3	Normal matrix conditioning	84
6.3.4	Asymmetric Data Dependent Laplacian	85
7	Conclusion	87
	Bibliography	88

1 Introduction

Laplace operator is extensively used in geometry processing, for example in mesh filtering ([43], [39] or [11]) or parameterization [14]. In the continuous setting, it is very well understood. It has also some quite interesting properties. Its generalization to the discrete case is however ambiguous. Various discretizations exist, differing mainly in the weights used in the discretized formula. Each of the discretizations preserves different subset of the properties (or their discrete equivalents) of the smooth Laplace operator. It can be even proven, that no discretization can preserve a certain set of those properties at once[50]. This makes each of the discretizations suitable for different purposes.

As part of this thesis, a new discretization of Laplace operator will be proposed in Chapter 5, minimizing the lengths of differential coordinates used in compression of dynamic triangle meshes. It is based on the assumption, that such minimization of lengths should lead to a decrease of the entropy of the encoded data. The weights of such discretization are calculated using a linear system constructed from mesh connectivity and geometry. It has one other big advantage over other discretizations that require the geometry information - it can be constructed from the geometry of more than one mesh at once, without requiring any complex analysis of the shapes of those meshes.

However, there is no direct geometric relation between vertex positions and weights of such operator, which means that some of its properties are difficult to prove. As a consequence, it is unclear, what Laplacian mesh processing techniques is this discrete Laplace operator suitable for apart from dynamic mesh compression, for which it is constructed. The goal of this thesis is to study those properties and explore possible improvements the Data Dependent Laplace operator can provide, in various mesh processing tasks.

In the first part, a theoretical background of Laplacian mesh processing will be provided. Laplace operator will be defined, and its properties described in Sections 2.1 and 2.2. The most popular of its discretizations will be listed, with the properties they preserve in Section 2.3. Then, some of the mesh processing techniques, that require application of a discrete Laplace operator will be described in Chapter 3. Each of the methods requires different properties of the Laplacian used. These properties will be shown. Then, a more detailed description of mesh compression problem will be provided

in Chapter 4.

In the second part of the thesis, a new discretization of Laplace operator (so-called *Data Dependent Laplacian*) will be proposed. A process, how to calculate its weights, will be described in Section 5.2. From the construction process, some of the properties will be shown to be preserved or broken. Those properties, that are not easy to be proven, will be stated.

Third part of the thesis will show the experimental results of applying such Laplace operator in various mesh processing techniques. The results will be evaluated, pointing out some of its issues and possible applications.

2 Laplace operator

Laplace operator is a second order differential operator denoted Δ . It is defined as divergence of gradient [9]. That means:

$$\Delta = \nabla^2 = \nabla \cdot \nabla.$$

The Laplace operator can be understood as a generalization of second order derivative in a multi-dimensional space. For a function f on n -dimensional euclidean space, it is equal to [43]:

$$\Delta f = \text{div} \nabla f = \sum_{i=1}^n \frac{\partial^2 f}{\partial x_i^2}.$$

In mesh processing, however, the term Laplace operator usually refers to the *Laplace-Beltrami* operator $\Delta_{\mathcal{M}}$, which extends its functionality to manifolds. Interesting property of $\Delta_{\mathcal{M}}$ is that if it is applied on vertex coordinate function \mathbf{x} , it evaluates to a surface normal of length dependent on mean curvature [9], also called *mean curvature normal*:

$$\Delta_{\mathcal{M}} \mathbf{x} = -2H \mathbf{n}. \quad (2.1)$$

2.1 Discrete Laplace operator

Although triangle meshes are usually designed to approximate smooth surfaces, they are piecewise linear, thus no analytic second order differential operator can be applied to obtain mean curvature normal. That means, for an estimation, the Laplacian operator must be somehow discretized. When measuring values on a regular grid, the discretization of Laplace operator can be obtained using finite differences:

$$\Delta X_{i,j} = \frac{X_{i-1,j} + X_{i,j-1} - 4X_{i,j} + X_{i+1,j} + X_{i,j+1}}{h^2},$$

where $X_{i,j}$ is value on position (i, j) and h is horizontal and vertical distance between neighbouring cells. This can be viewed as a weighted sum of differences between neighbouring values and the value in the cell (i, j) :

$$\Delta X_{i,j} = \sum_{(k,l) \in N(i,j)} \frac{1}{h^2} (X_{i-k,j-l} - X_{i,j}),$$

where $N(i, j)$ is the set of positions of neighbouring cells. This can be extended on triangle meshes, where instead of value differences, displacement vectors will be used:

$$\delta_i = \sum_{v_j \in N(v_i)} w_{ij}(\mathbf{x}_j - \mathbf{x}_i), \quad (2.2)$$

where δ_i is so-called *differential coordinate* of vertex v_i , \mathbf{x}_i is its coordinate, $N(v_i)$ is a set of its neighbouring vertices and w_{ij} is a weight assigned to an edge between vertices v_i and v_j . Rearrangement of the formula reveals a more convenient interpretation - displacement vector between weighted sum of positions of neighbours and weighted position of the vertex itself.

$$\delta_i = \sum_{v_j \in N(v_i)} w_{ij}\mathbf{x}_j - \sum_{v_j \in N(v_i)} w_{ij}\mathbf{x}_i,$$

The weight that multiplies the position of v_i will be denoted as w_i and the formula shows, that it is equal to sum of all weights assigned to edges incident to vertex v_i . This leads to a formula 2.3, which will be used in the majority of this thesis:

$$\delta_i = \sum_{v_j \in N(v_i)} w_{ij}\mathbf{x}_j - w_i\mathbf{x}_i. \quad (2.3)$$

Given formula can be also rewritten in matrix form:

$$\begin{aligned} \mathbf{LX} &= \mathbf{D}, \\ \text{or} & \\ \mathbf{Lx} &= \mathbf{d}_x, \quad \mathbf{Ly} = \mathbf{d}_y \quad \text{and} \quad \mathbf{Lz} = \mathbf{d}_z, \end{aligned} \quad (2.4)$$

where \mathbf{X} is a matrix containing vertex coordinates, \mathbf{D} contains δ -coordinates and \mathbf{L} is so-called *Laplacian matrix*. The structure of \mathbf{L} is as follows:

$$l_{ij} = \begin{cases} -w_i & i = j \\ w_{ij} & v_j \in N(v_i) . \\ 0 & \text{otherwise} \end{cases} \quad (2.5)$$

The size of Laplacian matrix is $|V| \times |V|$, where $|V|$ is the number of vertices of given triangle mesh. From the way the diagonal weight was calculated it is clear that for each row of \mathbf{L} , sum of non-diagonal elements is equal to negative value of the diagonal element. In other words, sum of all coefficients in each row is zero, independently on the choice of the weights w_{ij} .

2.2 Properties

Discrete Laplace operators and Laplacian matrices can have some interesting properties which could be extensively utilized to improve the desired results or speed up the computation. In general, however, not all properties are present for every variant of Laplacian. In fact, Wardetzky et al. [50] have presented four properties no discrete Laplace operator can satisfy simultaneously on non-regular surface triangulation. This makes every variant suitable for different usages. In this section, the most important properties will be described with their effect on Laplacian mesh processing techniques.

First of all, however, a few assumptions will be made. First assumption is that all the weights are real, as some of the properties will be true only for real matrices. Other than that, no duplicate edges or vertices are present in the processed triangle mesh. The mesh should also have a large number of vertices, about 1000 and more.

2.2.1 Sparsity

It can be shown, that for most triangle meshes the Laplacian matrix is *sparse*. Structure of such matrix implies that when increasing its size, the ratio between non-zero and zero elements gets lower, reaching zero in limit. From definition 2.5, it can be seen that a non-diagonal element l_{ij} can be non-zero if and only if there exists an edge between vertex i and vertex j . This means that the number of non-diagonal non-zero elements can be estimated as sum of all vertex degrees:

$$n \leq \sum_{v_i \in V} \text{deg}(v_i) = d \cdot |V|,$$

where V is set of vertices and d is the average vertex degree. Inequality is used, because there is no rule forcing any weight to be non-zero. By adding the number of diagonal elements, which is equal to $|V|$, we get estimate of the total number of non-zero elements:

$$m \leq |V| + d \cdot |V|$$

$$m \leq (d + 1) \cdot |V|.$$

As the size of the Laplacian matrix is $|V| \times |V|$, it remains to show that:

$$(d + 1) \cdot |V| \ll |V| \cdot |V|$$

$$(d + 1) \ll |V|.$$

Euler formula implies that average vertex degree is less or equal to 6 [9]. With increasing number of vertices, the average vertex degree is still the same. This implies that the ratio between the non-zero and zero elements is in fact getting lower. Thus, the Laplacian matrix is indeed sparse. This enables the usage of special data structures and algorithms, which are more memory and performance efficient compared to the standard ones working with dense matrix structures.

The sparsity of \mathbf{L} is one of the properties present for all of its variants described in this thesis. However, it can be also shown that even $\mathbf{L}^T\mathbf{L}$ is sparse. Element of $\mathbf{L}^T\mathbf{L}$ on position i, j can be calculated as dot product of i -th and j -th column of \mathbf{L} . Dot product can be non-zero only if there exist an index k on which both row vectors are non-zero. If $i = j$, there can be always at least one non-zero value at $k = i = j$, as even for every vertex there is at least diagonal element l_{ii} with value $-w_i$. For $i \neq j$, there are two cases that can occur. First case is when

$$v_i \in N(v_j), \quad (2.6)$$

in other words, vertex v_i shares an edge with vertex v_j . Then there are at least two elements, with $k = i$ or $k = j$ respectively, that can be non-zero. The second case occurs when vertices v_i and v_j do not share a common edge, but there exists a vertex v_k for which the statement 2.7 is true:

$$v_k \in N(v_i) \wedge v_k \in N(v_j) \quad (2.7)$$

In other words, v_i and v_j must share at least one common neighbour v_k . Then for such index k , $l_{ki} \cdot l_{kj}$ might be a non-zero value. For any other pairs of columns there is no way their dot product can have any other value than zero. The statements 2.6 and 2.7 can be combined to get general requirement for non-diagonal non-zero element:

$$v_j \in N_2(v_i),$$

where $N_2(v_i)$ is 2-ring neighbourhood of vertex v_i (see Figure 2.1). As number of vertices in given set is bigger than or equal to the size of the $N(v_i)$, which determined the number of possible non-diagonal non-zero elements corresponding row of \mathbf{L} , it is obvious that $\mathbf{L}^T\mathbf{L}$ is less sparse. However, it is still not dense, because average size of the 2-ring neighbourhood has upper bound of square of average vertex degree, which is still relatively small compared to expected number of vertices of a triangle mesh. The next section will point out why this property is so important in Laplacian mesh processing techniques.

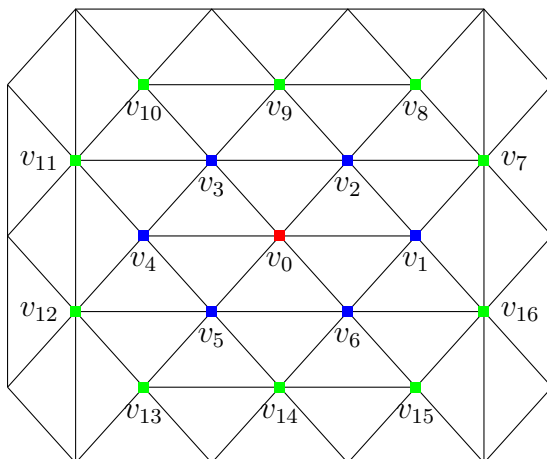


Figure 2.1: 2-ring neighbourhood of a vertex v_0 . Vertices v_1, \dots, v_6 form a 1-ring neighbourhood. The 2-ring neighbourhood is created by adding vertices v_7, \dots, v_{16} .

2.2.2 Singular matrix

Instead of calculating δ -coordinates from vertex positions, it is often required to do the opposite. Then, for n -dimensional space, n linear systems of the following form must be solved:

$$\mathbf{L}\mathbf{x}_i = \mathbf{d}_i,$$

where \mathbf{x}_i is an unknown vector of i -th coordinates of vertices and \mathbf{d}_i is a vector of i -th δ coordinates. Unfortunately, this cannot be solved, because the Laplacian matrix is *singular*. This is a consequence of a property of a discrete Laplace operator itself - *translation invariance* [34]. If the mesh was translated by a vector \mathbf{u} , using the formula 2.2, the δ -coordinates would be calculated as:

$$\begin{aligned} \delta_i' &= \sum_{v_j \in N(v_i)} w_{ij}((\mathbf{x}_j + \mathbf{u}) - (\mathbf{x}_i + \mathbf{u})), \\ \delta_i' &= \sum_{v_j \in N(v_i)} w_{ij}(\mathbf{x}_j - \mathbf{x}_i + \mathbf{u} - \mathbf{u}), \\ \delta_i' &= \sum_{v_j \in N(v_i)} w_{ij}(\mathbf{x}_j - \mathbf{x}_i) = \delta_i. \end{aligned}$$

There are infinitely many triangle meshes with equal differential coordinates and thus it is impossible to reconstruct the vertex positions without at least one vertex position, so-called *anchor*, provided for each mesh component. This statement directly implies the rank of the \mathbf{L} :

$$\text{rank}(\mathbf{L}) = |V| - k,$$

where k is the number of components of the given mesh. Providing position \mathbf{x}'_a of a vertex v_a means including equation of form

$$\mathbf{x}_a = \mathbf{x}'_a$$

to the system that is being solved. In matrix representation, this is achieved by extending \mathbf{L} by a row that has all elements equal to zero except the element at index a , which is equal to one. Vector \mathbf{d}_i must be also extended with a value x'_{a_i} . This creates a new linear system:

$$\tilde{\mathbf{L}}\mathbf{x}_i = \tilde{\mathbf{d}}_i, \quad (2.8)$$

where $\tilde{\mathbf{L}}$ is the extended rectangular Laplacian matrix and $\tilde{\mathbf{d}}_i$ is extended vector of differential coordinates. As stated Sorkine [34], with only one anchor vertex, the $\tilde{\mathbf{L}}$ is ill-conditioned. Thus it is usually required to provide more anchors. However, after that, the linear system is overdetermined and a solution must be estimated in the least-squares sense:

$$\tilde{\mathbf{L}}^T \tilde{\mathbf{L}}\mathbf{x}_i = \tilde{\mathbf{L}}^T \tilde{\mathbf{d}}_i.$$

It is important to point out that extending \mathbf{L} by only rows with single non-zero element cannot make the $\tilde{\mathbf{L}}^T \tilde{\mathbf{L}}$ dense, for such extension only affects diagonal elements, which were already non-zero.

Not only is the $\tilde{\mathbf{L}}^T \tilde{\mathbf{L}}$ matrix sparse, it is also symmetric and positive semi-definite, which are properties consequent to the transpose matrix multiplication with itself. This means a special direct sparse linear system solving algorithm can be used. For example, the Eigen library for *C++* [16] provides direct methods based on $\mathbf{L}^* \mathbf{L}^{*T}$ or $\mathbf{L}^* \mathbf{D} \mathbf{L}^{*T}$ Cholesky factorization [10] (there is no connection between Laplacian matrix \mathbf{L} and the factor \mathbf{L}^*), which are specially designed to work with sparse symmetric and positive semi-definite matrices.

2.2.3 Symmetry

Although it is not required, for some types of discrete Laplace operator, the Laplacian matrix is *symmetric*, in other words:

$$w_{ij} = w_{ji}. \quad (2.9)$$

Symmetry is one of the properties that as standalone have not as important influence, but are much stronger in combination with other properties. However, if assumption that all elements are real is true, even eigenvalues

are real. Additionally, each two eigenvectors \mathbf{v}_i and \mathbf{v}_j that correspond to different eigenvalues λ_i and λ_j are orthogonal, in other words:

$$\mathbf{v}_i^T \mathbf{v}_j = 0.$$

This will be further discussed in section 2.2.9.

Another advantage of symmetric matrices is lower storage requirements. Mathematical frameworks can save storage space by storing only upper triangle of the matrix, as the lower triangle contains the same values.

2.2.4 Positive weights

A discrete Laplace operator is said to have *positive weights* if all the weights assigned to edges have the same sign or are zero. Even when all the non-zero weights are negative, a similar discrete Laplace operator with the same properties can be obtained by simply multiplying the Laplacian matrix by -1 . Such operator has clearly all the weights positive or zero.

Positive weights are very important in mesh parameterization, where breaking this property may result in self intersections in the parameter space [15]. This will be further discussed in Section 3.3.

2.2.5 Linear precision

As was shown in formula 2.1, Laplace operator is related to the surface mean curvature. If the Laplace operator was applied to a point on plane, the result should be a zero vector [50]. In the discrete case, the requirements are the same, but they are not guaranteed by every discrete Laplacian. This can be shown by adjusting formula 2.3:

$$\delta_i = w_i \cdot \left(\sum_{v_j \in N(v_i)} \frac{w_{ij}}{w_i} \mathbf{x}_j - \mathbf{x}_i \right). \quad (2.10)$$

Note that the adjustment assumes the weight w_i to be non-zero. This can be interpreted as if δ coordinates were calculated by selecting a single point representing the neighbourhood of the vertex v_i , then calculating the displacement vector between this representing point and the position of v_i , and multiplying it by w_i . This means that δ coordinate is zero vector if the representing point has the same position as v_i . The representing point in the planar case, however, can be placed generally everywhere on the plane incident with the neighbouring vertices.

It is obvious that when the discrete Laplace operator is used to approximate surface mean curvature and surface normal, it is important to use a

variant that does not break linear precision, otherwise the result would be inaccurate for planar surfaces.

2.2.6 Locality

This property means that if the position of a single vertex is slightly adjusted, the only differential coordinates that change are those in given vertex neighbourhood. From equation 2.3, one would assume that for such defined Laplacian, the locality is always present. However, it depends on how the **weights** are calculated. Locality holds only when the weight w_{ij} can be proven to be calculated using at most the positions of the neighbours of vertices v_i and v_j .

Locality is required, when the processed mesh is expected to change frequently, for example in real-time applications for 3D object modelling. Instead of frequent recomputing of all weights and δ -coordinates, only those affected by the mesh modification must be actually recomputed.

2.2.7 Unit sum of weights

It is often desired that the sum of all non-diagonal elements of Laplacian matrix are equal to one. As shown in the Formula 2.10, the length of the displacement vector depends on the weight assigned to the vertex, which is equal to the sum of edge weights. Some of the mesh processing methods require, or have better results, if the discrete Laplace operator used is independent of such value, or in other words, all the diagonal elements of Laplacian matrix are equal to the same value k . Such Laplace operator then meets the requirements of unit sum of weights, because it is possible to obtain a similar Laplace operator with same properties by multiplying its Laplacian matrix by $\frac{1}{k}$.

In Laplacian mesh smoothing, the length of δ coordinate vector determines how much the vertex moves in the process. If this property is broken, there can be two vertices with the same surface curvature, which are after one iteration moved by different distance, even though they should not.

The length of the δ coordinate vector is also important in mesh compression, where it influences quantization. The more the diagonal values vary, the more problematic is to efficiently find quantization parameters that work for the whole mesh [45]. It also influences the entropy of the data, as the differential coordinates also contain additional information.

2.2.8 Definiteness

This property is defined only for symmetric Laplacian matrices. A symmetric matrix \mathbf{A} of size $n \times n$ is said to be *positive definite*, if for any $\mathbf{x} \in \mathbb{R}^n$:

$$\mathbf{x}^T \mathbf{A} \mathbf{x} > 0.$$

If the given scalar is always negative, the matrix is said to be *negative definite*. If the scalar also results in zero value, the matrix is *positive* (or *negative*) *semi-definite*. Otherwise, the matrix is said to be *indefinite*.

In Section 2.2.2 it was stated that matrix $\tilde{\mathbf{L}}^T \tilde{\mathbf{L}}$ is positive semi-definite for every discrete Laplace operator. However, for some of the variants, even the Laplacian matrix is positive semi-definite. Positive semi-definiteness is implied by positive edge weights. On the other hand, positive semi-definiteness does not imply positive weights [50].

In the Laplacian mesh processing, positive semi-definiteness of Laplacian matrix results in performance and accuracy improvements. Every linear system that is described by Laplacian matrix or any matrix created by removing any column and its corresponding row, can be directly solved using algorithms based on $\mathbf{L}^* \mathbf{L}^{*T}$ or $\mathbf{L}^* \mathbf{D} \mathbf{L}^{*T}$ Cholesky factorization, which also allow to quickly solve large amount of linear systems differing only by right-hand side vectors.

2.2.9 Eigenvalues and eigenvectors

Interesting properties of a Laplacian matrix are hidden in its eigenvalues and eigenvectors. Since \mathbf{L} is singular, its smallest eigenvalue in the sense of magnitude is $\lambda_0 = 0$, no matter its definiteness and symmetry. The corresponding eigenvector contains the same value in all elements. This means that for any vector, that has all the values same, multiplication with a Laplacian matrix results in zero vector. This can be proven using Formula 2.2. Suppose that all vertices have the same x -coordinate equal to some constant c . Then, for any vertex, the x -coordinate of δ is zero:

$$\delta_{i,x} = \sum_{v_j \in N(v_i)} w_{ij} (c - c) = 0.$$

One particularly interesting eigenvalue is the smallest non-zero eigenvalue that is often referred to as *Algebraic connectivity* or *Fiedler value*. Its corresponding eigenvector, so-called *Fiedler vector* [13], is used in the field of graph theory, for example in spectral graph clustering, or partitioning [6]. In computer graphics, this vector can be used for example in 1D embedding,

where it is desired to embed a mesh on a line, so that all the vertices lie as close as possible to their neighbours [54].

The most important property is, however, shared across all the eigenvectors. In the smooth one-dimensional case, it can be shown, that *sine* and *cosine* functions are eigenfunctions of Laplace operator:

$$\Delta(\sin(x)) = -\sin(x).$$

In the discrete case, the equivalents can be eigenvectors of Laplacian matrix[9]. If the Laplacian matrix is symmetric and positive semi-definite, the eigenvectors form an orthogonal basis. Finding a representation of a vector in such basis is equivalent to performing the Fourier transform on a signal [39]. This is performed accordingly:

$$\mathbf{f} = \sum_{i=0}^n \langle \mathbf{e}_i, \mathbf{f} \rangle \mathbf{e}_i, \quad (2.11)$$

where $\langle \mathbf{a}, \mathbf{b} \rangle = \mathbf{a}^T \mathbf{b}$, \mathbf{f} is a function assigning a value to each vertex, \mathbf{e}_i is i -th eigenvector of the Laplacian matrix and n is number of eigenvectors.

2.3 Types of discrete Laplace operator

In this section, various types of discrete Laplace operator will be discussed. For each type, its advantages and flaws will be described with their consequences pointed out. Generally, discrete Laplace operators can be divided into two separate groups according to the data needed for calculating the weights.

The first group is called *Combinatorial Laplacians*. These Laplacians are calculated using the mesh connectivity only. This means that two triangle meshes with identical connectivity, but completely different shape, share the same Laplacian matrix.

The second group, so-called *Geometric Laplacians*, are as the name implies calculated using also the mesh geometry. With more information required, geometric discrete Laplace operators usually approximate more precisely the smooth case. However, this also introduces some disadvantages.

2.3.1 Kirchhoff Laplacian

This variant of combinatoric Laplacian, often referred to as Graph Laplacian, is primarily used in the field of graph theory. It is named after the German physicist Gustav Kirchhoff, who is most known for his contribution

in the field of electrical engineering. The matrix of the Kirchhoff Laplacian, usually called Kirchhoff's matrix, is very popular for its use in the Matrix-Tree-Theorem which is usually attributed to Kirchhoff [24], even though he never explicitly stated it [19]. Matrix-Tree-Theorem says that the number of all unique spanning trees of a graph can be calculated as determinant of a matrix generated by deleting any single row and any single column of the Kirchhoff's matrix [24].

Matrix of Kirchhoff Laplacian can be calculated as:

$$\mathbf{L}_K = \mathbf{D} - \mathbf{A},$$

where \mathbf{D} is matrix containing vertex degrees on diagonal and \mathbf{A} is so-called adjacency matrix, which contains elements equal to one on position i, j only if there exists an edge between vertices v_i and v_j . The structure of the \mathbf{L}_K is as follows [34]:

$$l_{ij}^K = \begin{cases} \text{deg}(v_i) & i = j \\ -1 & v_j \in N(v_i) \\ 0 & \text{otherwise} \end{cases} \quad (2.12)$$

Applied on the vertex positions, the differential coordinates of vertex v_i can be calculated from following formula:

$$\delta_{K_i} = \sum_{v_j \in N(v_i)} (\mathbf{x}_i - \mathbf{x}_j).$$

As any non-diagonal value is equal to minus one, it is obvious that \mathbf{L}_K is *symmetric*. Even though all the weights are negative, they have the same sign. Thus, the *positive weight* property is not broken. Those two properties imply, that Kirchhoff's matrix is positive semi-definite. In the section 2.2.6, it was pointed out, that the only way how the discrete Laplace operator, as defined in this thesis, can break the *locality*, is through the weight calculation. With Kirchhoff Laplacian, this can never occur, because the weights are constant.

However, the *unit sum of weights* property is obviously broken, and more importantly, the *linear precision* does not hold [50]. The reason is as follows: Applying Kirchhoff Laplacian on vertex positions can be interpreted as calculating displacement vector between average position of neighbours and the vertex itself, which is then multiplied by the vertex degree. This implies that the vertex must lay exactly in the average neighbour position to achieve zero length differential coordinate. On non-uniformly triangulated planar surfaces, vertices generally do not have to lay exactly in such position.

2.3.2 Tutte Laplacian

This combinatorial Laplacian is named after William Thomas Tutte, British mathematician, who in the field of mesh processing is mostly known for his barycentric mapping theorem [41], which will be discussed in the Section 3.3.

Formula for calculating the Tutte Laplacian matrix is the following:

$$\mathbf{L}_T = -\mathbf{D}^{-1}\mathbf{L}_K = \mathbf{D}^{-1}\mathbf{A} - \mathbf{I}.$$

It is obvious from the formula, that Tutte Laplacian is closely related to the Kirchhoff Laplacian. In fact, it can be interpreted as its normalized variant, because each row is multiplied by the negative value of the inverse of the corresponding vertex degree. This is even more clear when the structure of \mathbf{L}_T is examined:

$$l_{ij}^T = \begin{cases} -1 & i = j \\ \frac{1}{deg(v_i)} & v_j \in N(v_i) \\ 0 & \text{otherwise} \end{cases} \quad (2.13)$$

Such discrete Laplace operator results in following differential coordinates:

$$\delta_{Ti} = \frac{1}{deg(v_i)} \sum_{v_j \in N(v_i)} (\mathbf{x}_j - \mathbf{x}_i).$$

It can be seen that the representing point from which the displacement vector is calculated is again the barycenter of neighbours.

By normalizing, the *unit sum of weights* is achieved. However, the *symmetry* of Laplacian matrix is sacrificed. As weight calculation still involves only connectivity information, the *locality* is preserved. Even though it has all the weights positive, the matrix is not positive semi-definite as this terminology works only for symmetric matrices.

Although the differential coordinates are very similar to those generated by Kirchhoff Laplacian, because both variants use the barycenter as representing point of neighbours, the Tutte Laplacian has main advantage in that the lengths do not depend on the vertex degree. This is mostly important in mean curvature estimation. In spite of both variants breaking linear precision, the Kirchhoff Laplacian is even worse, because even a vertex on a plane could have the highest curvature from the whole mesh if it had the highest valence.

2.3.3 Cotangent Laplacian

Previously mentioned variants of discrete Laplace operator were not really suitable for differential operator approximation because of the broken linear

precision. This problem is targeted by Cotangent Laplacian. It is often attributed to Pinkall – Polthier, who used a similar formula based on finite element method to calculate discrete minimal surfaces[26]. Their research was further extended by Meyer et al., who then proposed a set of discrete differential operators on triangle meshes and used the cotangent Laplacian for estimating the *mean curvature normal vector* [23].

The formula to calculate *Cotangent* differential coordinates is as follows:

$$\delta_{C_i} = \frac{1}{2A_i} \sum_{v_j \in N(v_i)} (\cot \alpha_{ij} + \cot \beta_{ij})(\mathbf{x}_j - \mathbf{x}_i), \quad (2.14)$$

where α_{ij} and β_{ij} are angles opposite to the edge between vertices v_i and v_j , as can be seen in Figure 2.2.

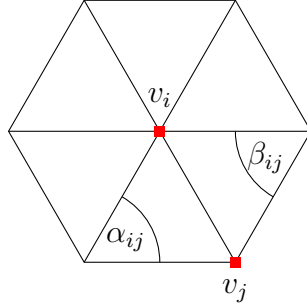


Figure 2.2: Angles α_{ij} and β_{ij} used to calculate weight w_{ij} of Cotangent Laplacian.

A_i is an area on the surface belonging to the vertex v_i . There are multiple ways to obtain A_i . The simplest way is to divide the area of each triangle to thirds:

$$A_{simple_i} = \sum_{v_j \in T(v_i)} \frac{1}{3} A_j,$$

where $T(v_i)$ is a set of triangles incident to vertex v_i [11]. Another approach is approximating the area of a Voronoi cell. If all the incident triangles are acute, the vertices forming the boundary of the cell are their circumcenters. However, for obtuse triangles, a problem arises. The circumcenter lays outside the triangle, making it possible for an incident area to be bigger than the actual area of the triangle. Meyer et al. suggested a triangle division, so that for the vertex incident to obtuse angle, the corresponding area is half of the triangles area and for other two vertices, it is one fourth (see Figure 2.3) for each. One other possibility is to assign each vertex unit area, which results in less accurate curvature estimation. However, the direction of such differential coordinates is still the same, it saves much of computing time, and most importantly, the corresponding Laplacian matrix is *symmetric*.

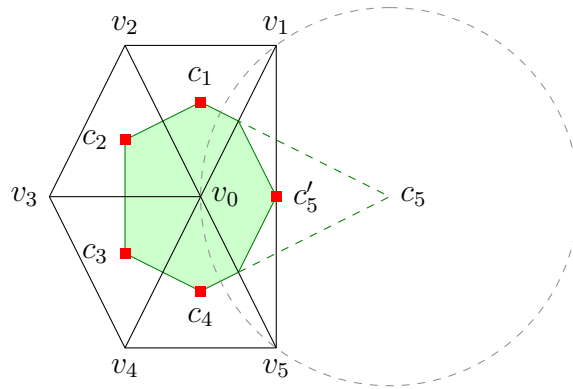


Figure 2.3: Area incident to vertex. For obtuse angle, the half point of the opposite edge is used to divide the triangle area instead of the circumcenter.

Calculating the weights and areas still requires only 1-ring neighborhood, which means that *locality* is preserved. As was already mentioned, the *symmetry* depends on how the incident area is estimated. In fact, as Wardetzky et al. [50] stated, the symmetric cotangent Laplacian is also positive semi-definite. If it is desired, the formula can be also adjusted so that the sum of weights is unit, but then again, the symmetry is sacrificed. The most important property of Cotangent Laplacian is *linear precision*. Given discrete Laplace operator was specially designed to approximate mean curvature normal vector which has obviously zero length on planar surface. However, if the sum of angles incident to an edge is greater than π , the corresponding weight is *negative*. Another problem arises, when one of the incident angles gets close to π , as

$$\lim_{x \rightarrow \pi^-} \cot(x) = -\infty.$$

In spite of all the issues related to large angles, the cotangent Laplacian is probably the most widely used discrete Laplace operator[9]. It is also important to note, that by constantly making sampling denser in a particular way, the cotangent Laplacian converges to the smooth Laplace operator[50].

2.3.4 Mean Value Laplacian

Mean Value Laplacian (often abbreviated to MV Laplacian) was designed by Floater [15] to address the negative weights of Cotangent Laplacian, which result in undesired triangle flipping in mesh parameterization.

Mean value coordinates are generalization of barycentric coordinates for star-shaped polygons in the sense that for a point v_0 lying inside of the planar star-shaped polygon consisting of points v_1, v_2, \dots, v_k a set of weights λ can

be found, so that

$$v_0 = \sum_{i=1}^k \lambda_i v_i,$$

$$\sum_{i=1}^k \lambda_i = 1.$$

One way to calculate the λ weights is to use the cotangent formula (2.14). However, as it was mentioned before, given λ weights can be negative, even though the point lies inside the polygon, which is in contradiction with the concept of barycentric coordinates. Floater proposed weights that are positive for star-shaped polygons, based on the mean value theorem for harmonic functions (hence the name Mean value coordinates):

$$\lambda_i = \frac{w_i}{\sum_{j=1}^k w_j}, \quad w_i = \frac{\tan(\frac{\alpha_{i-1}}{2}) + \tan(\frac{\alpha_i}{2})}{\|v_i - v_0\|},$$

where angles α_{i-1} and α_i are shown in Figure 2.4.

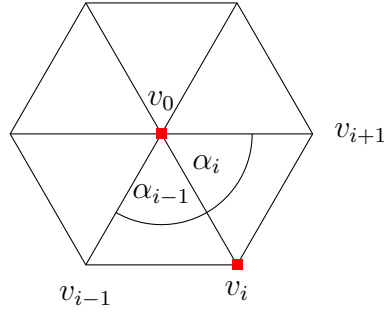


Figure 2.4: Angles α_{i-1} and α_i used to calculate weights of Mean Value Coordinates.

MV Laplacian is then constructed with weights calculated as λ weights. For vertex on planar surface, the representing point from which the displacement vector is equal to the weighted sum of neighbours with λ weights. Resulting weighted sum is exactly equal to the position of the processed vertex, which is obvious from the definition of mean value coordinates. As a consequence, the *linear precision* cannot be broken. Another property forced directly by the definition is the unit sum of weights. It can be also shown that all the weights are *positive* and the operator is *local*. However, the *symmetry* is broken and given operator does not converge to the smooth case with denser sampling[50].

3 Laplacian mesh processing

This chapter targets the most popular methods in mesh processing, that involve application of the discrete Laplace operator, except for the mesh compression, which has whole chapter reserved for itself, as the Data Dependent discrete Laplace operator was initially designed for compression.

3.1 Mean curvature estimation

Mean curvature, as well as other differential properties defined on triangle mesh, is extensively used in mesh processing, for example in shape analysis. It also serves for detection of significant features of triangle meshes, as usually those features are located in places with higher magnitude of curvature.

The mean curvature estimation can be derived from the Formula 2.1:

$$H(v_i) = \frac{1}{2} \|\delta_i\|,$$

where $H(v_i)$ is an absolute value of mean curvature at vertex v_i . To obtain a sign, one must compare the direction of normal vector and a differential coordinate. If the vectors point to opposite directions, the mean curvature is positive, otherwise it is negative. Although there exists a lot of algorithms that are more precise (for example [30] or [27]), the mean curvature estimation using discrete Laplace operator is still being used, as it is fast, straightforward and easy to understand.

The most suitable discrete Laplace operator should have following properties: *linear precision* and *convergence*. As was already explained, broken linear precision may result in non-zero curvature estimate on planar surfaces, particularly with non-uniform sampling. Convergence assures, that when the density of sampling approaches the smooth case, the estimate gets more accurate. The only discrete Laplace operator that fulfils both properties is the *Cotangent* Laplacian. In fact, the mean curvature estimation was one of the motivations to its construction [23]. From all its variants, the one normalized with areas incident to vertices is most accurate.

However, instead of requiring the most accurate results, an approximation that attempts to capture the distribution of curvature on the measured surface is often desired. The result still provides enough information to allow detection of significant points as the relations between values are preserved. This means, that other variants of discrete Laplace operator can still be used to obtain useful results.

3.2 Smoothing

Processed triangle meshes are often result of some 3D object capturing method. Such method usually introduces high frequency noise into data, which is undesired, as it distorts the information. To eliminate the noise, smoothing or some low-pass filtering technique is usually employed. Smoothing can be also used to smooth out rough edges of a model.

In the past, complex techniques based on energy minimization were usually used (e.g. [51]). Those techniques were computationally expensive [39]. By employing discrete Laplace operator, simpler methods with comparable results were created. Those techniques are easy to implement and result in much less expensive computations.

3.2.1 Filtering using manifold harmonics

In the Section 2.2.9, it was shown, that eigenvectors of a Laplacian matrix form an orthogonal harmonic basis. Just like in signal processing, where the signals are often filtered in the frequency domain, one can interpret vertex positions as 3D signals, transform them by finding their representation in the harmonic basis using Formula 2.11 and apply the filter on such representation.

The simplest way to perform smoothing in harmonic basis is to reconstruct the positions only from a subset of eigenvectors corresponding to lower frequencies:

$$\hat{\mathbf{f}} = \sum_{i=0}^m \langle \mathbf{e}_i, \mathbf{f} \rangle \mathbf{e}_i,$$

where $m < n$ is number of the eigenvectors corresponding to m smallest eigenvalues [43]. This also means, that it is not necessary to compute all the eigenvectors of Laplacian matrix, but only the ones that will be used in the reconstruction. The result of such smoothing technique can be seen in Figure 3.1.

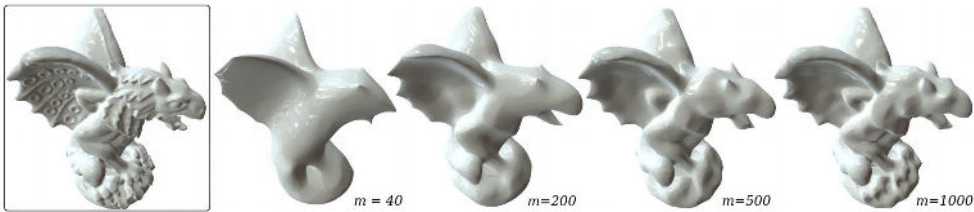


Figure 3.1: Reconstruction of the positions using subset of eigenvectors forming the harmonic basis. Source: [43]

Sometimes, it is not desired to remove all the high frequency information, only to attenuate it. In such case, another technique is required. In signal processing, this would be achieved by following process: User specifies the desired frequency response of the filter, which represents how much each frequency will be amplified or attenuated. Then, the signal is transformed into the frequency domain, and each of its coordinates in the frequency basis is multiplied according to the frequency response. Finally, the signal is transformed back to the time domain. In the mesh processing, one can apply almost identical technique. Only difference is the basis used to represent the frequencies on the mesh and the transformation used. This technique is not only suitable for low-pass filtering, it can be used for more advanced filtering tasks as band-attenuation etc. (see 3.2).

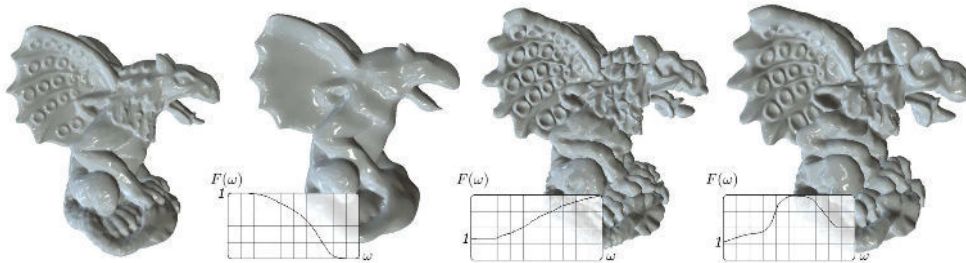


Figure 3.2: Mesh filtering using frequency response. From left to right: Original mesh, low-pass, enhancement and band-exaggeration filters. Source: [43]

The Laplacian matrix that can be used in the manifold harmonic basis approach, must be *symmetric* and *positive semi-definite*, because its eigenvectors must form orthogonal basis. Only two discrete Laplace operators have such Laplacian matrix - Kirchhoff's Laplacian and symmetric variant of Cotangent Laplacian. The issue of Kirchhoff's Laplacian is, that its eigenvectors are not geometry aware. This can be seen in Figure 3.3, where the distortion of iso-contours on the function represented by one of the eigenvectors (so-called eigenfunction) is present. The reason of that behaviour are the different sampling rates across the surface of the mesh.

The main issue with smoothing spectral approaches is that it is computationally expensive to calculate eigenvectors for Laplacian matrices representing meshes with thousands of vertices, although the sparsity of the matrix can be utilized, for example using iterative Arnoldi method [12], that is part of the ARPACK library. It is also almost impossible to store the whole basis in the system memory [43]. The following smoothing approaches will be applied directly on data, which is more efficient in this case.

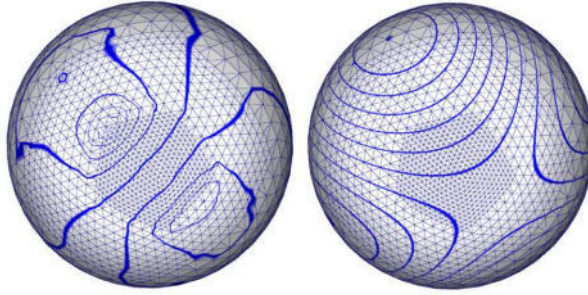


Figure 3.3: Iso-contours of fourth eigenfunction. Left: Kirchhoff's Laplacian, Right: Cotangent Laplacian. Source: [43]

3.2.2 Iterative smoothing

This is the most basic approach for mesh smoothing. It is in some sense inspired by Gaussian filtering method from signal-processing. Main idea of this method is to iteratively move vertices on surface, so that the mean curvature decreases. The movement can be described by following formula:

$$\mathbf{x}' = \mathbf{x} + \lambda\delta, \quad (3.1)$$

where \mathbf{x} is position of vertex at current step, λ is smoothing coefficient and δ is differential coordinate. This explains the connection between discrete Laplace operator and this smoothing technique. In fact, this vertex movement can be interpreted as if the vertices were moved in normal direction (or its approximation) by an amount derived from their mean curvature. This means, that vertices with high estimated mean curvature move significantly more, than vertices on nearly planar surface. The Figure 3.4 visualizes one step of this technique on discrete signal in two dimensions with $\lambda = 0.5$. Red arrows represent the differential coordinates, dashed line represents the result of the smoothing step.

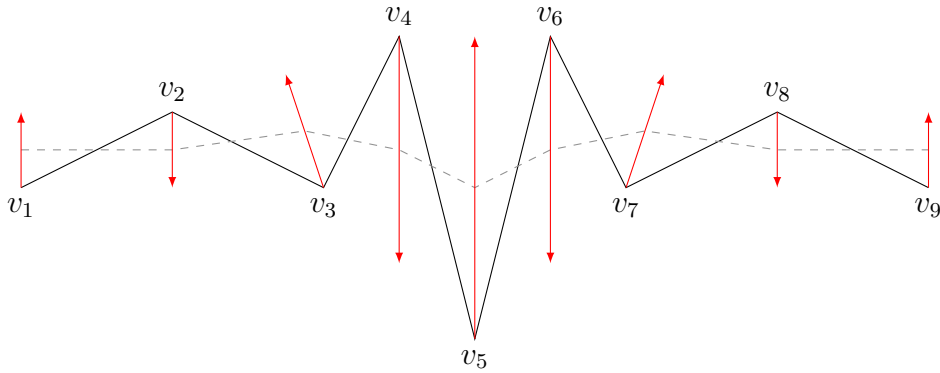


Figure 3.4: Visualization of single step of basic iterative smoothing in two dimensions ($\lambda = 0.5$).

Crucial for this technique is the selection of the λ coefficient. Positive value is required, as for negative values, instead of smoothing, the detail gets even sharper. The bigger the value is, the farther vertices move. However, $\lambda > 1$ results in curvature flipping - vertices that had positive mean curvature in previous step, are moved, so that they have negative mean curvature and vice versa. This can be seen in the Figure 3.5, where it is demonstrated on $\lambda = 1.5$.

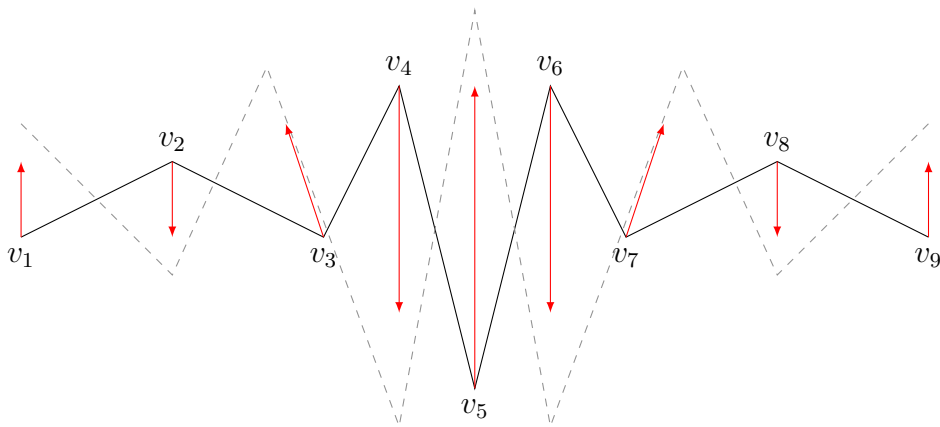


Figure 3.5: For $\lambda > 1$ ($\lambda = 1.5$ in this case), smoothing introduces curvature flipping.

The core problem of this technique is that it introduces shrinkage (see Figure 3.6). Geometrically this problem can be interpreted as follows: For a surface isomorphic with a sphere, this technique tries to obtain the lowest curvature for each vertex. If it was applied with infinitely many steps, it would result in all the vertices laying in the same position. In such case, the differential coordinates are zero vectors and thus the curvature at any

vertex is zero. The shrinkage is a problem of Gaussian filtering itself. The reason is that Gaussian filter is in fact not a low-pass filter [39].

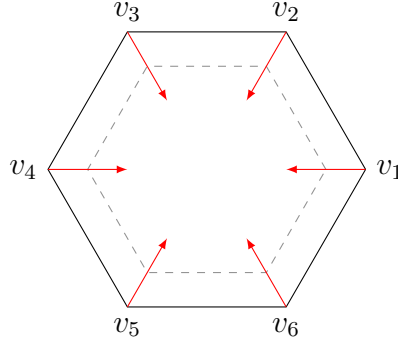


Figure 3.6: Basic iterative smoothing technique introduces shrinking

3.2.3 $\lambda|\mu$ smoothing

This technique, proposed by Taubin, [39] addresses the shrinkage problem of previous method. The principle of $\lambda|\mu$ smoothing is to divide the smoothing iteration step into two sub-steps:

$$\begin{aligned} \mathbf{x}' &= \mathbf{x} + \lambda\delta, \\ \mathbf{x}'' &= \mathbf{x}' + \mu\delta', \end{aligned} \tag{3.2}$$

where λ and μ are smoothing coefficients for which $\lambda > 0$, $\mu < 0$ and $\mu < -\lambda$. The first sub-step is exactly identical to the step of the basic smoothing method and it introduces shrinkage as well. For this reason it is usually referred to as *shrinking* sub-step. Then, the vertices are moved in the opposite direction, increasing the volume of the represented model. This is usually referred to as *growing* sub-step. Both steps are shown in Figure 3.7 with $\lambda = 0.35$ and $\mu = -0.5$.

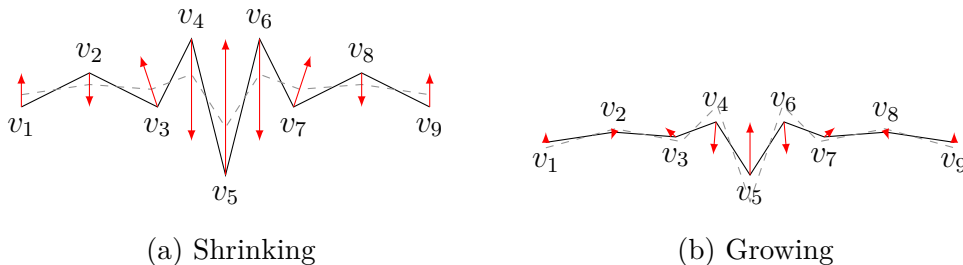


Figure 3.7: Substeps of $\lambda|\mu$ smoothing algorithm ($\lambda = 0.35$, $\mu = -0.5$)

Taubin has shown that not only this technique does not shrink the smoothed triangle mesh, but for proper configuration of smoothing coefficients, it is also a proper low-pass filter with transfer function

$$f(k) = (1 - \lambda k)(1 - \mu k)$$

in the region of interest $k \in [0, 2]$. The coefficient configuration can be calculated by choosing pass-band frequency k_{PB} for which

$$k_{PB} = \frac{1}{\lambda} + \frac{1}{\mu} > 0,$$

and coefficient λ , then calculating the μ from previous formula. Taubin stated that good smoothing results are produced by k_{PB} in interval between 0.01 and 0.1. Selecting λ is more difficult. It must be chosen, so that it is as large as possible, while keeping $|f(k)| < 1$ for any frequency higher than k_{PB} . Otherwise, these frequencies would not be attenuated [39].

The complexity of choosing suitable smoothing coefficient configuration is one of the two main disadvantages of this technique. Another one is that increased number of iterations is required to obtain similarly smooth surface compared to the basic technique [11].

3.2.4 Implicit fairing using diffusion flow

Previous methods both required many iterations to obtain fairly smooth surface. Desbrun et al. [11] proposed a new approach based on noise attenuation through the diffusion flow. That is usually modelled by diffusion equation:

$$\frac{\partial \mathbf{X}}{\partial t} = \lambda L(\mathbf{X}),$$

where \mathbf{X} is matrix of vertex coordinates and $L(\mathbf{X})$ is analytic Laplacian of \mathbf{X} . Integrating the equation by explicit *Euler* scheme, formula similar to the iterative smoothing is obtained:

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

It is no surprise that $\lambda dt < 1$ otherwise the curvature flipping is obtained. The approach derived from explicit *Euler* scheme still requires more steps to achieve satisfying results. If implicit integration was used instead, the results can be obtained in considerably fewer steps:

$$\mathbf{X}^{n+1} = \mathbf{X}^n + \lambda dt \mathbf{L}(\mathbf{X}^{n+1}).$$

The unknown \mathbf{X}^{n+1} is on both sides of the equation. This means that it must be further adjusted:

$$(\mathbf{I} - \lambda dt \mathbf{L}) \mathbf{X}^{n+1} = \mathbf{X}^n. \quad (3.3)$$

To obtain \mathbf{X}^{n+1} , a linear system must be solved. However, this system is sparse, as \mathbf{I} is diagonal and \mathbf{L} was already shown to be sparse in Section 2.2.1. Another advantage over explicit integration is that λdt can be chosen much larger, resulting in much smoother surface in single step. In comparison with previously mentioned smoothing techniques, Desbrun et al. have shown that in fact, implicit fairing obtains better results in less or the same computing time [11] (see Figure 3.8).

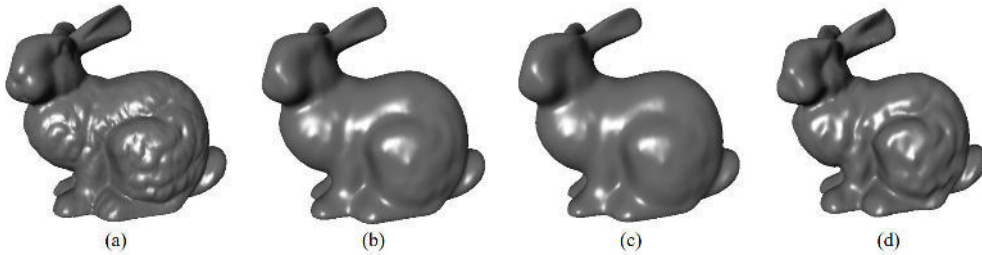


Figure 3.8: Comparison of smoothing techniques: (a) Original mesh, (b) 10 steps of basic iterative smoothing, (c) One step of implicit smoothing, (d) 20 iterations of $\lambda|\mu$ smoothing. Source: [11]

Although the whole method based on diffusion flow seems complicated, there exists a trivial geometric interpretation. In each step, the goal is to find a smoother surface, from which the surface from previous step can be derived by amplifying high frequencies.

3.2.5 Dynamic mesh smoothing

All of the smoothing techniques mentioned in this thesis can be also applied to a sequence of triangle meshes with identical connectivity. Such data structure is called *dynamic mesh* and it will be rigorously defined in section 4.2.2. If such mesh sequence was smoothed with each frame being smoothed independently using unique Laplacian matrices, some inconsistencies between the smoothed results of subsequent frames might occur, creating visual artifacts when rendering the animated sequence.

This can be addressed by using the same Laplacian matrix to smooth all the frames. In the case of a combinatorial Laplacian, this is not an issue, since all the frames already share the same Laplacian matrix. However, in the case of a geometric Laplacian, there is no simple way to calculate the

Laplacian matrix, as it requires some geometry information of the mesh. The problem is, how to choose a geometry, from which the matrix will be calculated, while still representing the geometry of all the frames. One particular approach of how to obtain such geometry will be described in Section 4.2.2.

3.2.6 Suitable Laplacians for Smoothing

It is difficult to select a single Laplace operator as the most suitable for mesh smoothing. It depends, on what results are desired. The combinatorial Laplacian is used, when the user requires smoother surface with more regular triangulation. This more regular triangulation means, however, that the vertices experience so-called *tangential drift*, which means that they were moved in the tangential plane of the surface. The tangential drift is undesired, for example, when the smoothed model has already calculated parameterization. When texture is mapped on the surface using such parameterization, it is visibly distorted, no matter how precise the parameterization was for the original mesh. A geometric Laplacian, on the other hand, is used, when the user requires angle preservation in triangles. It is also better to use geometric Laplacians on mesh with different sampling rates, as the combinatorial Laplacians introduce distortion, when applied on such mesh (see Figure 3.9).

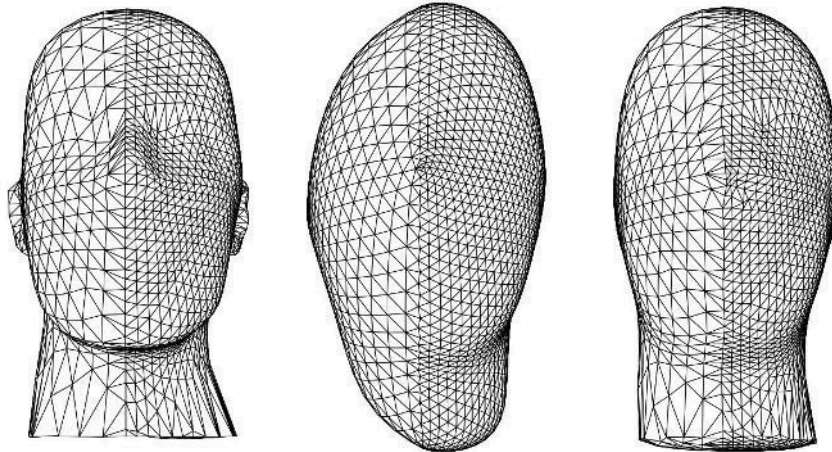


Figure 3.9: From left to right: original mesh with different sampling rates, result of smoothing with *Tutte Laplacian*, result of smoothing with *Cotangent Laplacian*. Source: [11]

3.3 Parameterization

Parameterization is a process of mapping points from the surface of a triangle mesh onto a parametric, usually two dimensional space. In other words, finding a parametric representation

$$\mathbf{x} = \mathbf{f}(u, v),$$

where $\mathbf{x} \in \mathbb{R}^3$ is coordinate of point on surface, and u and v are parameters of triangle vertices. In every triangle forming the surface of the mesh, the parameters u and v can be obtained for any point on its surface knowing only the parametric representations of the triangle vertices, by calculating the barycentric coordinates:

$$\mathbf{u} = \sum_{i=1}^3 \lambda_i \mathbf{u}_i,$$

where $\mathbf{u}, \mathbf{u}_i \in \mathbb{R}^2$ are vectors consisting of the u and v parameters, and λ_i are barycentric coordinates.

Parameterization is extensively used for texture and normal mapping, where values of an image are mapped to properties on surface (colour in case of texture mapping, surface normal in case of normal mapping). This allows adding additional detail to the modelled three dimensional object, which then looks more realistic even with a lower number of faces (see Figure 3.10). However, it can be used in many more fields, for example in remeshing [5].

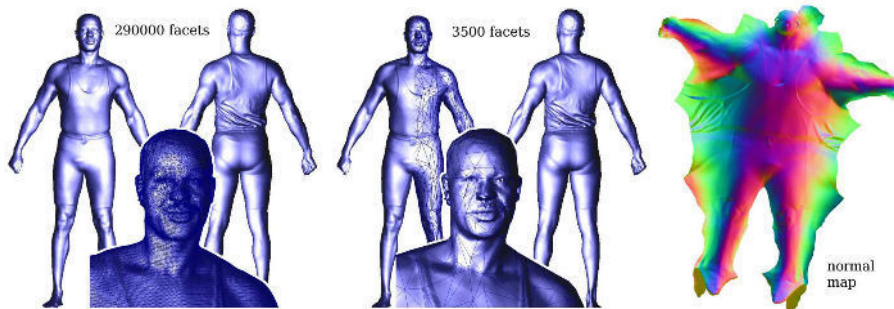


Figure 3.10: Appearance preserving simplification. Mesh is simplified and normal mapping is applied. Source: [17]

Although there are many algorithms for mesh parameterization with better results (for example [33] or [31]), only the basic algorithm involving usage of discrete Laplace operator - barycentric mapping, will be described in this thesis. The algorithm is based on Barycentric mapping theorem formulated by Tutte [41], which states, that for any triangulated surface isomorphic to a disk, a valid parameterization (in the sense that no edges intersect) can be obtained, if boundary points lie on a convex polygon in the parametric space, and any internal vertex is a convex combination of its neighbours [9]. Such parameterization can be obtained by fixing boundary vertices to a convex polygon and iteratively moving internal vertices to the barycenter of their neighbours, until there is no movement. This can be interpreted as if the edges of such mesh were substituted by springs with fixed boundary vertices, and the parameterization is obtained as positions of vertices when the system of springs reaches the equilibrium.

Instead of iteratively moving the vertices, their positions in equilibrium state can be calculated by solving two linear systems based on following equations:

$$a_{ii}\mathbf{u}_i = - \sum_{v_j \in N(v_i)} a_{ij}\mathbf{u}_j,$$

where $\mathbf{u}_j, \mathbf{u}_i \in \mathbb{R}^2$ are parametric coordinates, a_{ij} is the stiffness of the spring between vertices v_i and v_j and $a_{ii} = -\sum_{v_j \in N(v_i)} a_{ij}$. Moving all the values that are unknown to the left-hand side results in:

$$a_{ii}\mathbf{u}_i + \sum_{v_k \in N(v_i) \wedge v_k \notin B} a_{ik}\mathbf{u}_k = - \sum_{v_j \in N(v_i) \wedge v_j \in B} a_{ij}\mathbf{u}_j,$$

where B is set of boundary vertices. This directly leads to the two linear systems [14]:

$$\mathbf{A}\mathbf{u} = \bar{\mathbf{u}}, \quad \mathbf{A}\mathbf{v} = \bar{\mathbf{v}}.$$

However, it is important to state that indexing inside of the matrix \mathbf{A} does not correspond to the indexing of the vertices, as the size of the matrix \mathbf{A} is $n \times n$, where $n = |V| - |B|$, and $|B|$ is number of boundary vertices.

The connection between barycentric mapping and discrete Laplace operator is in the stiffness of springs. In fact, the weights of Laplacian can be used as the values for the a_{ij} and a_{ii} coefficients. If *Tutte* or *Kirchhoff Laplacian* is used, the result corresponds to the parameterization suggested by Tutte. However, such parameterization usually does not capture the geometry correctly, because the Laplacian breaks *linear precision*. Another property that is crucial for parameterization is *positive weights*. Negative weight can be interpreted as a spring that instead of pulling, pushes the vertices away from each other. It also means, that the position found for

a vertex incident to an edge with such weight is not a convex combination of its neighbours. This may lead to triangle overlapping. However, if the weights are positive, it is guaranteed that the mapping to parametric space is one-to-one [15]. The only Laplacian, that satisfies both properties, is *MV Laplacian*, which makes it the most suitable discrete Laplace operator for this purpose.

The barycentric mapping can be, as well as all the smoothing techniques described in previous section, applied to mesh sequences. When using a geometric Laplacian, to prevent visual differences between frames in the texture during animation of such sequence, one must use the same Laplacian matrix for each of the frames, which is, as was already stated, not trivial to obtain.

3.4 Mesh editing

In 3D modelling, it is often desired to edit or deform a surface, while preserving visual appearance of the edited model. There are generally two approaches: *space deformation* and *surface deformation* [9]. The first approach is based on deforming the space in which the model is embedded, thus implicitly deforming the model as well. The most known algorithm using this approach is Free-Form deformation [32]. The latter approach works with the surface itself, allowing better control over the vertex movement. The Figure 3.11 visualizes the differences between both approaches.

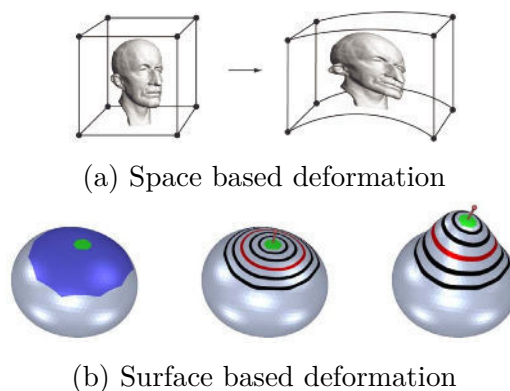


Figure 3.11: Difference between space and surface based deformation approaches. Images taken from [8]

The surface based mesh deformation works usually on this principle: First, the vertices of the mesh are divided into three subsets:

- \mathcal{F} (Fixed),
- \mathcal{H} (Handle),
- \mathcal{U} (Unconstrained).

Usually, it is enough to specify \mathcal{H} and \mathcal{U} and define \mathcal{F} as a set of vertices that are not in any of the two previously mentioned subsets. After the division, a transform (translation, rotation, scaling etc.) is applied to all vertices in \mathcal{H} . Then, positions of all the unconstrained vertices must be calculated with the knowledge of the positions of fixed points and the transformed positions of handles. The main difference between surface based deformation methods is in the process of calculating the unconstrained vertex positions.

It is desired to preserve the details of the surface before deformation. For example, if a model of a person with a wristband on the hand was deformed so that translation was applied to the hand, the wristband should still be visually recognizable. One approach that targets this, was described by Alexa [1]. It is based on an interesting property of differential coordinates: by representing relation between a vertex and its neighbourhood, the differential coordinates store information of the surface detail. To preserve the detail, the differential coordinates should be as close as possible to those of the original mesh.

To obtain positions of unconstrained vertices, Formula 2.3 for calculating δ -coordinates can be utilized. Moving all the known values (differential coordinates and positions of fixed vertices and handles) to right-hand side results in three overdetermined linear systems, one for each coordinate. The process will be described only for x -coordinate:

$$\mathbf{L}'\mathbf{x}' = \mathbf{d}'_x,$$

where \mathbf{L}' is matrix obtained by removing all the columns corresponding to fixed vertices and handles, \mathbf{x}' is vector of x -coordinates of all unconstrained vertices, and \mathbf{d}'_x is a vector which consists of rows d'_{x_i} :

$$d'_{x_i} = \delta_{x_i} - \sum_{v_j \in K} l_{ij}x_j,$$

where δ_{x_i} is δ -coordinate of vertex v_i , l_{ij} is element of Laplacian matrix at position (i, j) , and $K = (N(i) \cup i) \cap (\mathcal{H} \cup \mathcal{F})$ is set of all indices coresponding to non-zero elements in row i of L at removed columns. The matrix \mathbf{L}' is of

size $|V| \times |\mathcal{U}|$, however, it is important to state, that for any vertex $v_k \notin \mathcal{U}$ with no unconstrained neighbour, the corresponding equation in the linear system is $0 = 0$. It then remains to solve the system in the least-squares sense:

$$\mathbf{x}' = (\mathbf{L}'^T \mathbf{L}')^{-1} \mathbf{L}'^T \mathbf{d}'_x.$$

$\mathbf{L}'^T \mathbf{L}'$ is symmetric, sparse and positive semi-definite, allowing quick solution using for example Cholesky decomposition.

Sometimes, a new set of *anchor* vertices is introduced to the process [37]. This set is similar to \mathcal{U} in the sense that their positions need to be calculated. However, they are somewhat constrained by including equation of the form $x'_i = x_i$ to the overdetermined linear system. When the system is solved in the least-squares sense, their position approximate the original values, while smoothly transitioning between fixed and unconstrained vertices.

The technique described so far has one big issue. The differential coordinates are only translation invariant. This means that if the handles were for example rotated or scaled, one would expect the orientation and length of the displacement vectors to be altered as well. However, in such case, the technique would still try to force the orientation and length of the original δ (see Figure 3.12). The issue is addressed by Lipman et al. [20] and Sorkine et al. [37]. Both approaches alter the orientation of differential coordinates before inputting them to the overdetermined linear system.

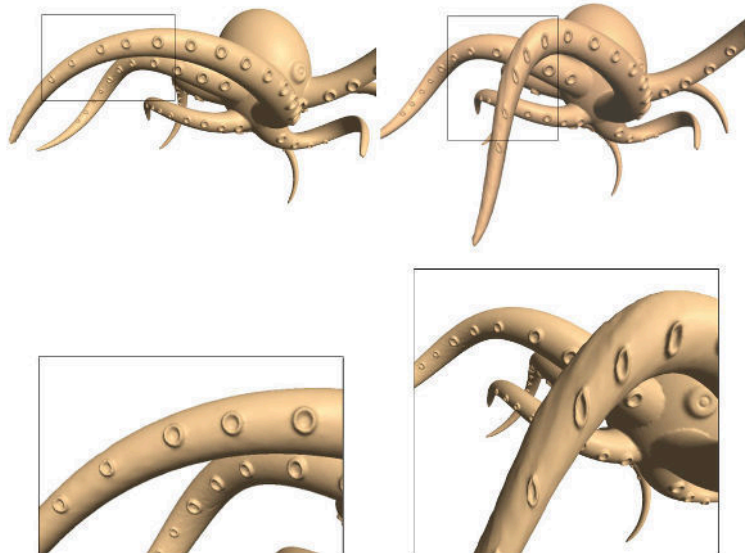


Figure 3.12: Distortion of detail caused by the techniques rotation variance. Left column: Original mesh, Right column: Result. Bottom row shows detail. Source: [20]

The choice of discrete Laplace operator again depends on what are the desired results. Botsch – Sorkine [7] have shown that geometric Laplacians preserve better the details. However, if interactivity is desired, the combinatorial Laplacians provide faster performance as their weights require little to no computation time.

3.5 Mesh morphing

Mesh morphing (or mesh interpolation) is a process of combining two different triangle meshes to obtain a new one that is similar to them. Sometimes, it is only desired to map detail of one mesh on part of the second meshes geometry. For the sake of simplicity, it will be assumed that both meshes share the same connectivity and each vertex of the first mesh corresponds exactly to the vertex of the second mesh with the same index. Otherwise, remeshing and feature matching would be performed. For more information, reader can refer for example to [25]. Mesh morphing techniques are quite often used in animation, when a user wants to insert interpolated frame between two consecutive animation frames.

The most basic and naïve mesh morphing technique is simple euclidean coordinate interpolation:

$$\mathbf{x}_i = (1 - t)\mathbf{x}_{1i} + t\mathbf{x}_{2i}, \quad (3.4)$$

where \mathbf{x}_i denotes result position of vertex v_i , \mathbf{x}_{1i} is its position in the first input mesh, \mathbf{x}_{2i} the position in the second mesh and $0 \leq t \leq 1$ is the interpolation parameter. In the case when there is no significant difference between the two meshes, the method performs quite well. The problems, however, occur, when only some parts of the model are being morphed. When the two meshes do not align well, undesired effects can be seen on the boundary between morphed vertices and the static parts of the mesh (see Figure 3.13).

This can be solved by using some intrinsic parameters [38], which store information relative to the vertex neighbourhood. Such intrinsic properties can be differential coordinates, which are employed in the technique proposed by Alexa [1]. The technique is very similar to the surface editing approach described in the previous section. In fact, both techniques were described in the same article. Again, the vertices are subdivided into three subsets. However, the positions of handles are not moved by the user, but are calculated using Formula 3.4. Another difference is in the differential coordinates used in the solved linear system which are calculated as an interpolation between differential coordinates of both meshes, to morph the



Figure 3.13: Undesired effects of naïve morphing. From left to right: Two input meshes, result of naïve morphing, morphing technique proposed by Alexa [1]. Source: [1]

details. To smooth the transition between handles and fixed vertices, one can specify interpolation parameter for each vertex independently [1].

It is important to note that when the morphing is applied on the whole mesh, the linear interpolation of differential coordinates does not introduce any improvement over the vertex position interpolation, because of the distributive property of matrix multiplication:

$$\mathbf{D} = (1 - t)\mathbf{L}\mathbf{X}_1 + t\mathbf{L}\mathbf{X}_2 = \mathbf{L}((1 - t)\mathbf{X}_1 + t\mathbf{X}_2),$$

where \mathbf{D} is matrix of differential coordinates, \mathbf{X}_1 is matrix of vertex positions of the first mesh and \mathbf{X}_2 is matrix of vertex positions of the second mesh. In other words, linear interpolation of differential coordinates is equivalent to calculating the differential coordinates from linearly interpolated positions.

Again, the problem of differential coordinates is that they are not rotation and scaling invariant. In fact, interpolating between two vectors \mathbf{d}_1 and \mathbf{d}_2 that are oriented in opposite direction may result in a zero vector, although both lengths are non-zero. It is desired, that scale and rotation are interpolated independently. In the case of scale, simple linear interpolation can be used. The orientation, however, requires a more sophisticated approach. The vector must be rotated around an axis which is perpendicular to both \mathbf{d}_1 and \mathbf{d}_2 . This can be achieved using Rodrigues' rotation formula [28]:

$$\begin{aligned} \mathbf{d}_{rot} &= \mathbf{d}_1 \cos(\alpha t) + (\mathbf{n} \times \mathbf{d}_1) \sin(\alpha t), \\ \mathbf{n} &= \frac{\mathbf{d}_1 \times \mathbf{d}_2}{|\mathbf{d}_1 \times \mathbf{d}_2|}, \\ \alpha &= \arccos \left(\frac{\mathbf{d}_1 \cdot \mathbf{d}_2}{|\mathbf{d}_1| |\mathbf{d}_2|} \right), \end{aligned}$$

where \mathbf{n} denotes a unit vector in the direction of axis of rotation and α is an angle between \mathbf{d}_1 and \mathbf{d}_2 . The final delta coordinate is then obtained as:

$$\mathbf{d} = d_{scale} \cdot \frac{\mathbf{d}_{rot}}{|\mathbf{d}_{rot}|}.$$

When interpolating differential coordinates using such interpolation technique, the result is no longer equivalent to calculating the differentials coordinate from interpolation of vertex positions. This means that the morphing of the whole mesh can be also improved by this interpolation technique.

This technique requires a discrete Laplacian that results in the same Laplacian matrix for both meshes, because otherwise it would cause inconsistency of values in the solved linear system. For this reason, geometric Laplacians cannot be used without any complex adjustment, even though they result in better results when applied in linear systems like the one in this morphing method. On the other hand, the combinatorial Laplacians determine the weights only from connectivity, which as was already mentioned, is shared between both meshes.

3.6 Least-squares meshes

This interesting technique proposed by Sorkine – Cohen-Or [35] is used to approximate a set of input control points by a mesh with arbitrary connectivity. Usually, the set of points is first approximated by some function, from which the surface must be then constructed. This technique skips this step entirely resulting directly in the desired surface.

The surface is required to be as smooth as possible, ideally with zero curvature. In such case

$$\mathbf{Lx} = \mathbf{0}, \quad \mathbf{Ly} = \mathbf{0} \quad \text{and} \quad \mathbf{Lz} = \mathbf{0},$$

where \mathbf{x} , \mathbf{y} , \mathbf{z} are vectors corresponding to individual coordinates of vertex positions. This yields three linear systems to be solved. To enforce its position, one must add an equation of following type for each control point:

$$\omega_c \mathbf{x}'_c = \omega_c \mathbf{x}_c,$$

where \mathbf{x}_c and \mathbf{x}'_c are original and calculated positions of control point c and ω_c is the weight forcing the position. It is important to specify at least two control points, because as was already mentioned, Laplacian matrix is singular and for one point, there exists a trivial solution of all vertices laying in the position of the single control point. For any other control point count, the linear system is overdetermined, which means only an approximate solution in the least-squares sense can be obtained. The control points thus generally do not lay on the resulting surface. However, the distance of these points from the surface can be manipulated by adjusting the weights ω_c . Higher the weight is, the more important is the position of given control

point. This can be seen in Figure 3.14. To find the least-squares solution to this system is equivalent to minimizing the value of the following expression:

$$\|\mathbf{Lx}\|^2 + \sum_{c \in C} \omega_c^2 |x_c - x'_c|^2,$$

where x_c is x -th coordinate of control point c , x'_c is its calculated x -th coordinate and C is set of all control points.

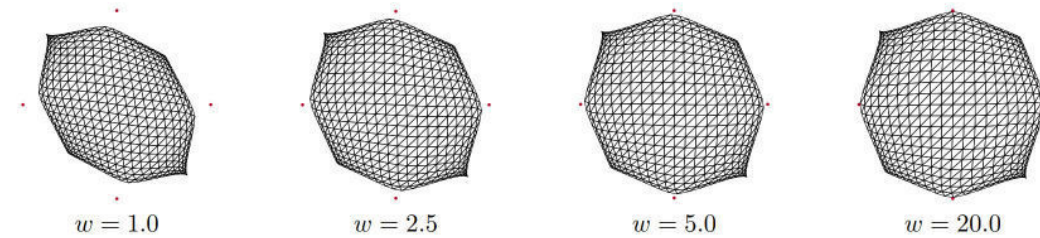


Figure 3.14: Weight influence in the Least-squares meshes algorithm. The red dots denote a control point. Source: [35]

Other than for shape approximation, the Least-squares meshes can be used in surface editing. A set of vertices of the mesh is selected as control points. Then the positions of such control points are altered. Finally, the mesh is reconstructed from the altered control points. This can be seen in Figure 3.15

The technique, in the form as was defined in this section, requires using combinatorial types of discrete Laplace operator, because the surface was constructed knowing only the connectivity and positions of few control points. To allow application of a geometric Laplacian, the method must be reformulated - geometry must be known to form the Laplacian matrix, which is not always the case. Another reason, why the combinatorial Laplacians are used more often, is that they produce better triangulation of the surface with triangles much closer to regular.

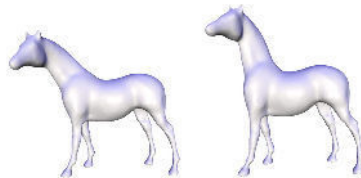


Figure 3.15: Mesh editing using Least-squares meshes algorithm. Source: [35]

4 Mesh compression

The main goal of mesh compression is to reduce the amount of data required to describe a triangle mesh without introducing too much distortion. This is important, especially when the mesh is transmitted over a network or archived. The usual format used to store uncompressed mesh data is 3 floating point numbers per vertex for storing the geometry, one for each coordinate, and 3 integers per triangle, one for each index of triangle vertex. However, such format contains significant amount of data redundancy.

4.1 Connectivity coding

It is undesired to distort information about connectivity, in the sense, that two neighbouring vertices do not share a common edge after compression. However, the data can be still reduced in a lossless manner. The naïve storage format of three integers per triangle requires $3 \cdot 32 = 96$ bits per triangle assuming 32 bit integers. From Euler formula, it can be derived that $F \approx 2V$, where F is the number of triangles of the mesh, and V is the number of vertices. This implies that the naïve format requires approximately $2 \cdot 96 = 192$ bits per vertex (bpv).

The most redundant information in the naïve storage format is ordering. Even though the vertices would be reordered, the connectivity would still remain the same. This is exploited by the algorithm called *Edgebreaker* [29]. The algorithm first selects a random triangle. Through its edges, the remaining triangles are then traversed. During the process, the algorithm builds two data structures - a list of all already found vertices and a list of edges forming the boundary of already processed area. There are five situations that can occur while processing single triangle, according to the edge e through which the triangle was found and a vertex v that is opposite to this edge:

- C - v was not yet found.
- L - v was already found and lies on the boundary on the left of e .
- R - v was already found and lies on the boundary on the right of e .
- E - v was already found and lies on the boundary on the left and right of e .

- $S - v$ was already found, but does not lay near e on the boundary.

All of the situations are shown in Figure 4.1.

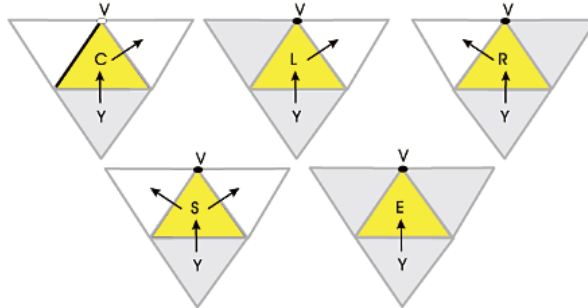


Figure 4.1: All the situations during the traversal of *Edgebreaker* algorithm. Yellow triangle denotes currently processed triangle. Gray color denotes already processed triangle. The arrow pointing inside the yellow triangle marks the direction, from which the triangle was found. The outgoing arrows denote next traversal directions. The vertex, that was not yet found is marked white, already found vertex is marked black. Source: <https://www.computer.org/cms/Computer.org/dl/trans/tg/2004/05/figures/v049916.gif>

Instead of coding the vertex-triangle incidences, it is enough to encode only the symbols representing the situations in the order of the triangle traversal. Huffman or arithmetic coding can be used to reduce the bit rate even further, as it can be shown, the symbol C occurs more likely. The average bit rate is then less than 4 bpv.

If the mesh is more regular, the data can be reduced even more by assuming the vertex valence does not vary too much. This is the principle of the technique called *Valence encoding* [4]. A lot of other techniques for connectivity compression exists with comparable results. However, connectivity compression is not a main subject of this thesis, thus it is not necessary to describe them all.

4.2 Geometry coding

This section will be divided into two parts. The first part will discuss possible compression techniques for reducing data rate of a static triangle mesh geometry. The second part will be reserved for geometry of triangle mesh sequences. It will be shown, that even though both problems look similar, each of them often requires a different approach.

4.2.1 Static mesh compression

The term static mesh refers to the triangle mesh as was described until now. It is thus defined by the connectivity and positions of each vertex in three dimensional space. The positions are mostly represented using single or double precision floating point numbers. In 3D, this requires $3 \cdot 32 = 96$ bpv or $3 \cdot 64 = 192$ bpv. To allow application of standard coding techniques, these values must be quantized. This means that geometry compression is usually lossy [36]. If the coordinates are quantised directly, too many quantization levels are required in order not to introduce too much distortion. For this reason, the compression techniques usually encode some other information, that has lower entropy. Most techniques are based on a prediction scheme, where instead of encoding the original value, only a difference between this value and some prediction is encoded. The prediction must be known to both the encoder and the decoder, otherwise the original value cannot be reconstructed. In this thesis, two such prediction techniques will be described - *parallelogram prediction*[40] and *high-pass quantization* [36].

Parallelogram prediction is a simple, yet powerful compression technique that is extensively used to this date. The mesh is traversed in a way similar to the Edgebreaker algorithm. In fact, the method can be used alongside this algorithm and the geometry can be encoded when the case with symbol C occurs. Then, a prediction is performed using already encoded positions of vertices of the triangle opposite to currently processed vertex. The prediction assumes that both currently processed triangle and the opposite triangle lay on a single plane, and that they form a parallelogram (see Figure 4.2). The position of any single vertex of a parallelogram can be calculated using the positions of the other three vertices:

$$\mathbf{x} = \mathbf{x}_o + (\mathbf{x}_l - \mathbf{x}_o) + (\mathbf{x}_r - \mathbf{x}_o) = \mathbf{x}_l + \mathbf{x}_r - \mathbf{x}_o,$$

where \mathbf{x}_r and \mathbf{x}_l denote positions of directly connected vertices and \mathbf{x}_o is position of the opposite vertex. Instead of the vertex position, only a displacement vector between the correct and predicted value is encoded. It is important to note, that for stopping the error propagation, the encoder must itself reconstruct the encoded values and use them for the prediction instead of the original ones.

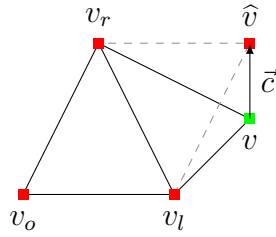


Figure 4.2: Parallelogram prediction. v_o, v_r, v_l are already encoded vertices, v is currently coded vertex, \hat{v} is the prediction of its position and \vec{c} is correction vector.

This technique and many others that work directly in euclidean coordinates tend to introduce distortion in high frequencies, where it is more visible, especially when the mesh represents a smooth surface (see Figure 4.3). To address this issue, Sorkine et al. [36] have proposed a technique called *high-pass quantization*. The process is simple: on the encoder side, δ -coordinates are calculated using the Formula 2.4. After that, they are quantized and encoded. The decoder then must reconstruct the positions by solving the linear system derived from the calculation formula. However, it was already stated in Section 2.2.2 that the Laplacian matrix has not a full rank, thus at least one vertex position for each connected component must be also encoded. The decoder then must add equation of form $\mathbf{x}'_i = \mathbf{x}_{received}$ for each anchor vertex, where i is index of such vertex. This results in an overdetermined linear system described by Formula 2.8, which is then solved in the least-squares sense.

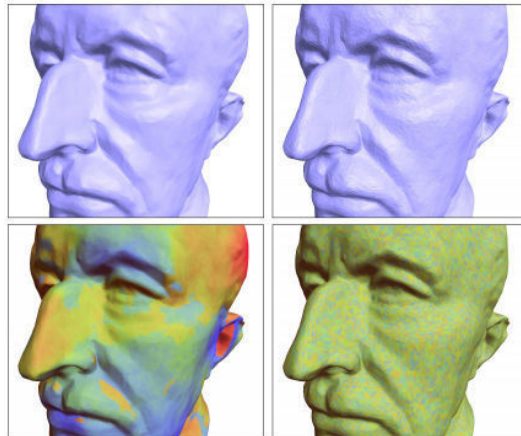


Figure 4.3: Comparison of distortion introduced by High-pass quantization (Left) and Compression working with euclidean coordinates (Right). Visible high frequency distortion in the second technique. Source: [36]

In the case of static meshes, only combinatorial Laplacians can be used, because they require only the knowledge of connectivity, which in the geometry compression phase is already known to both encoder and decoder. Transmitting the amount of information required to construct a geometric Laplacian, turns out to be as expensive as transmitting the mesh itself [44]. The high-pass quantization can be classified as a prediction technique, because of the interpretation stated in Section 2.2.6 - the encoded differential coordinate can be interpreted as displacement vector between the position of a vertex and a point representing the vertex neighbourhood.

Sorkine et al. have shown, that high-pass quantization results in low-frequency errors, thus it produces less visible distortion [36]. There is still one big issue with this technique - there is no direct control over error propagation. By increasing the number of anchor points, the error propagation can be only attenuated, at the cost of increasing the data redundancy. Two modifications were already proposed to reduce the effects even more: *Error diffusion* [2], and *Hierarchical Laplacian-based compression* [21].

4.2.2 Dynamic mesh compression

The term *dynamic mesh* will in this thesis refer to a sequence of triangle meshes representing an object continuously moving in time (see Figure 4.4). Each single triangle mesh of this sequence will be referred to as a *frame*. It will be also assumed that the connectivity is shared between all the frames, in other words, if the first frame of the sequence contains an edge between vertices v_i and v_j , such edge is present in all the other frames.

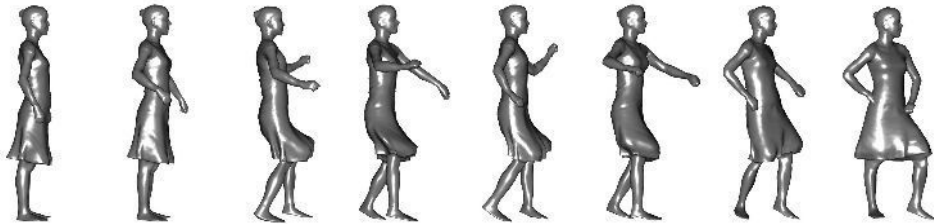


Figure 4.4: Example of dynamic triangle mesh: dataset *Samba* [49]

In the uncompressed form, the individual frames are usually stored as static meshes in separate files. This means, that each frame contains information about the connectivity, even though this information can be stored only once. The geometry now requires $3f \cdot 32$ bpv or $3f \cdot 64$ bpv for double precision format, where f is the number of frames in the sequence. This means, that the positions of a vertex in a dynamic mesh are now represented by an

array of $3f$ floating point numbers:

$$\mathbf{x} = [x_1, y_1, z_1, x_2, y_2, z_2, \dots, x_f, y_f, z_f].$$

Assuming the continuous movement of the represented object, the position of a vertex must change only slightly between any two subsequent frames. To reduce data redundancy, the compressing tool can work with so-called *trajectories* instead. Trajectory is an array of the form:

$$\mathbf{t} = [x_1, y_1, z_1, t_{2x}, t_{2y}, t_{2z}, \dots, t_{fx}, t_{fy}, t_{fz}],$$

where for example $t_{2x} = x_2 - x_1$, is difference between the x-coordinate of the vertex in the first and second frame.

To reduce the dimensionality of the problem, *Principal Component Analysis* (PCA) is usually performed. Alexa – Müller [3] applied this technique directly on the mesh geometry, to find the set of so-called *key-frames* - basis of the shapes, which represents the encoded sequence. Such approach is, however, computationally expensive. Váša – Skala [47] proposed a more efficient algorithm called *CODDYAC*. This algorithm performs PCA on the trajectories to find the significant trajectories characterizing the motion of the shape over the sequence [44] by calculating eigenvectors of autocorrelation matrix of the input data. These significant trajectories then form a reduced basis of the space of vertex trajectories. The basis is then encoded alongside the compressed geometry data.

The original *CODDYAC* algorithm uses the *parallelogram prediction* to encode the PCA coefficients. This technique, however, as was noted in the previous section, introduces a distortion in the high frequencies. Váša – Petřík [45] thus proposed using the *high-pass quantization* [36] instead. Compared to the static mesh compression, the dynamic mesh compression has one big advantage. With the large number of the frames, it is less expensive to transmit some information about the overall mesh geometry that will serve to construct a geometric Laplacian on the decoder side. Váša et al. [44] have shown, that **applying the geometric Laplacian, results in a more accurate prediction**, which means, that fewer quantization bits are required to achieve comparable results. Their algorithm first encodes a representing frame, the so-called average mesh. From the *average mesh*, a geometric Laplacian is computed. This Laplacian is then used to calculate the δ -trajectories, which are encoded. It is important to adjust the vertex positions of this mesh according to the compression error, so that both the encoder and the decoder work with the same data.

The way, how the average mesh is calculated, is crucial for this technique. The easiest way would be to spatially align all the frames, and then to

calculate the average position over time for each of the vertices. Such mesh may, however, suffer from shrinkage and self-intersections [44]. For this reason, they proposed to represent each of the frames in the *edge shape space* [52] where each edge is represented by edge length and dihedral angle. Calculating the average mesh in this space results in a more visually plausible result (see Figure 4.5).

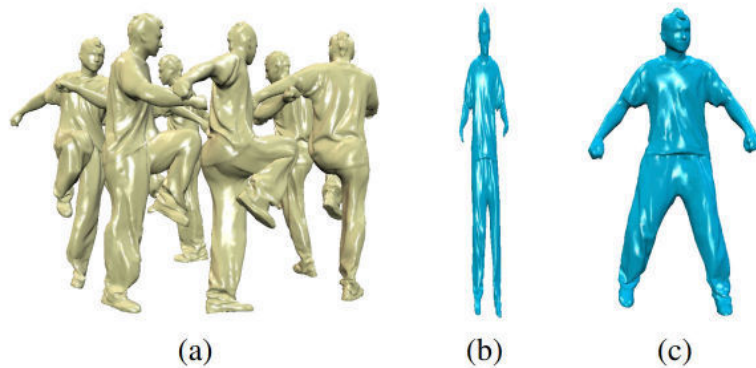


Figure 4.5: Average mesh used as representing frame in dynamic mesh compression. (a) Input mesh sequence, (b) Average mesh calculated the naïve way, (c) Average mesh calculated in the *edge shape space*. Source: [44]

The accuracy of the prediction in the *high-pass quantization* is directly related to the variance in the lengths of the δ vectors. For this reason, it is better to use a discrete Laplace operator with *unit sum of weights*, for example MV Laplacian. Also, the *linear precision* is important, because if it is broken, the δ coordinates become a combination of the mean curvature normal and tangential shift. The second of the two negatively affects the entropy of the encoded data [44].

4.3 Error metrics

To compare the results of different geometry compression techniques, a *Rate-Distortion* curve is usually used. This curve visualizes how changing the parameters of the compression affects the data bit rate and the distortion. To measure the distortion, some error metric must be applied.

In the past, the *mechanistic* error metrics were usually used. Such metrics usually try to measure distance between original and reconstructed vertex positions. The most basic mechanistic error metric is called *Mean Squared*

Error (MSE), which is computed:

$$MSE = \frac{1}{|V|} \sum_{i=1}^{|V|} (\|\mathbf{x}_i - \mathbf{x}'_i\|)^2,$$

where $|V|$ is the number of vertices, \mathbf{x}_i is the position of the vertex with the index i and \mathbf{x}'_i is its reconstructed position. In the case of dynamic meshes, for example the *Karni-Gotsman* (KG) [18] error metric can be used:

$$KG_{error} = 100 \cdot \frac{\|\mathbf{A} - \mathbf{A}'\|}{\|\mathbf{A} - \mathbf{E}(\mathbf{A})\|},$$

where $\|\cdot\|$ is the Frobenius norm, \mathbf{A} is matrix of size $3|V| \times f$, where each row represents the development of the corresponding coordinate over time, \mathbf{A}' is matrix of the same size containing data from reconstructed positions and $\mathbf{E}(\mathbf{A})$ is a matrix created from \mathbf{A} by substituting values in each column with the mean value of that column [48].

The mechanistic error metrics are most suitable, when the user defines a tolerance to which the reconstructed positions must be close to the original positions. However, these techniques do not capture how much is the distortion visible. For such criteria, *perceptual* error metrics must be applied. These metrics are specially designed to detect visible compression artifacts. For static meshes, for example, the *Dihedral Angle Mesh Error* (DAME) [46] metric can be used. This metric calculates the difference between the original and the reconstructed dihedral angles. For dynamic meshes, to the best of the authors knowledge, there are only few error metrics, for example *Spatio-Temporal Edge Difference* (STED) proposed by Váša – Skala [48], which is based on measuring the relative change in edge lengths, or a metric proposed by Yildiz – Capin [53].

The compression techniques that work directly with euclidean coordinates usually provide better results when mechanistic error metrics are used. However, as was already mentioned, they introduce distortion in high frequencies, where it is more visible, thus they usually do not perform well from the perspective of perceptual error metrics. On the other hand, the techniques working with intrinsic representations do not usually provide competitive results in mechanistic error metrics, while performing excellent in the perceptual metrics.

5 Data Dependent discrete Laplace operator

In this chapter, a new type of discrete Laplace operator will be proposed. First, the motivation of creating such type will be noted. Next, the process of calculating its weights will be described. All the properties, that do not require experimental verification, will be also stated. It will be also shown, how to transmit the data required to construct such operator on both encoder and decoder side during dynamic mesh compression.

5.1 Theoretical background

In Section 4.2.2, it was stated, that Váša et al. have shown significantly better compression results using geometric Laplacians. One reason is more accurate prediction, as one would expect the point representing the vertex neighbourhood to be closer to the vertex itself in comparison with a combinatorial Laplacian. The better prediction allows fewer quantization levels with comparable results. To generalize the idea, lets assume a Laplacian matrix \mathbf{L}_0 , for which

$$\mathbf{L}_0 \mathbf{T} = \mathbf{O}, \quad (5.1)$$

where \mathbf{T} is a matrix of size $|V| \times r$, where r is reduced dimension of data, whose rows represent vertex trajectories in the reduced basis and \mathbf{O} is zero matrix. Such matrix results in all delta trajectories equal to a zero length vector. If non-trivial \mathbf{L}_0 existed, there would be no need to transmit the δ trajectories. Instead, only the positions of anchors would be required. However, as will be shown, such matrix cannot be constructed.

5.2 Construction

To construct a discrete Laplace operator, for which the \mathbf{L}_0 is its Laplacian matrix, the weights w_{ij} and w_i must be calculated. For this purpose, Formula 5.1 can be used to construct a linear system of equations of the following form:

$$\sum_{v_j \in N(v_i)} w_{ij} t_{jk} - w_i t_{ik} = 0,$$

where t_{ik} is k-coordinate of the trajectory vector of vertex i . For the sake of simplicity, it will be assumed, that $w_i = 1$ and that \mathbf{L}_0 is symmetric. Moving all the constant values to the right-hand side results in:

$$\sum_{v_j \in N(v_i)} w_{ij} t_{jk} = t_{ik}.$$

The resulting linear system consists of $|V| \cdot r$ equations of $|E|$ unknowns, where $|E|$ is number of edges. The Euler characteristic implies, that $|E| \simeq 3|V|$. Also, usually $|V| \cdot r > 3|V|$, which means, that the system is overdetermined and generally no exact solution exists. The least-squares solution results in a matrix L_0^* , which generates δ -trajectories that are as close to zero vectors as possible. Such δ -trajectories still must be encoded, but one would expect their entropy to get lower in comparison with a geometric Laplacian.

There is still one issue with the way, how the matrix is constructed. There is no mechanism forcing the sum of all weights assigned to the edges incident to a vertex to be equal to the weight assigned to that vertex. This means that L_0^* is not a proper Laplacian matrix, as:

$$\sum_{v_j \in N(v_i)} w_{ij} t_{jk} - w_i t_{ik} \neq \sum_{v_j \in N(v_i)} w_{ij} (t_{jk} - t_{ik}).$$

The δ -trajectories calculated using such matrix are, in fact, translation variant, thus L_0^* is not singular, which is a property of all discrete Laplace operators.

As far as the author of this thesis is concerned, there exist two approaches to solve this issue. The first approach is to use the method of *Lagrange multipliers*. This method is usually used to solve the problem of constrained minimization. In this case, the function to be minimized is the least squares residual of the linear system in the Formula 5.3 and the constraints are the equalities of the sum of weights (see Formula 5.2). The method will not be described in this thesis, however, it would lead to increasing the number of unknowns.

The second approach leads to the exact opposite - decrease of the number of unknowns. For each vertex, it is desired to force

$$\sum_{v_j \in N(v_i)} w_{ij} = 1. \tag{5.2}$$

One of the weights in this formula is selected and marked as *dependent*. Value of such weight then depends on the values of other weights and is therefore no longer unknown, and must be substituted in all the equations, where it

occurs. For a vertex, that has no incident edge with already substituted weight, the substitution formula is as follows:

$$w_{il} = 1 - \sum_{v_m \in N(v_i) \wedge m \neq l} w_{im}.$$

Generally, the substitution formula has, however, a different structure:

$$w_{il} = c_i - \sum_{w_{im} \in I \wedge m \neq l} c_{im} w_{im},$$

where c_i and c_{im} are some constants resulting from previous substitutions and I is the set of independent weights assigned to edges incident to vertex v_i , or the weights, that got substituted to the equation. Applying this process, the number of unknowns reduces to $2|V|$, and solving such linear system results in weights that form a discrete Laplace operator, which will be referred to as *Data Dependent Laplacian*.

The Data Dependent Laplacian can be calculated even on a static mesh, as for this mesh, the weights can be obtained by solving a system of $3|V|$ equations of $2|V|$ unknowns. It is important to note, that for dynamic meshes, one could even calculate Data Dependent Laplacian with a non-symmetric Laplacian matrix. However, this Laplacian requires solving a linear system of $2|E| - |V| = 5|V|$ unknowns, which means that for static meshes, this linear system would be underdetermined.

The linear system, that was mentioned before can be written in matrix form:

$$\mathbf{Q}\mathbf{w} = \mathbf{b}, \tag{5.3}$$

where \mathbf{Q} is a matrix of the size $2|V| \times (r \cdot |V|)$ representing the linear system, \mathbf{w} is column vector of the size $2|V|$ of unknown weights and \mathbf{b} is a right-hand side vector of the size $r \cdot |V|$. It is obvious that storing such matrix and vertices directly is memory inefficient.

In the case of matrix \mathbf{Q} , there are usually only few non-zero values in each row. One would then assume, that \mathbf{Q} is sparse. Additionally, the system will be solved in the least-squares sense. Thus, one can directly store the matrix $\mathbf{Q}^T\mathbf{Q}$, which is only of size $2|V| \times 2|V|$. This matrix can be constructed as sum of outer products of its rows:

$$\mathbf{Q}^T\mathbf{Q} = \sum_{i=1}^{r \cdot |V|} \mathbf{q}_i^T \mathbf{q}_i,$$

where \mathbf{q}_i denotes the i -th row of \mathbf{Q} . Because of the sparsity of \mathbf{Q} , the outer product can be computed as follows: for each pair of non-zero elements with

indices k and l on the given row, one would add the product of their values to the element of $\mathbf{Q}^T\mathbf{Q}$ on the position (k, l) . It is also important to note, that for each vertex, there are r corresponding equations, one for each coordinate. The rows, that correspond to such equations, share the same structure, in the sense, that all have the non-zero values at the same positions. In fact, instead of adding the products coordinate after coordinate, it is possible to add a scalar product of trajectories \mathbf{t}_l and \mathbf{t}_k .

It is also possible to construct $\mathbf{Q}^T\mathbf{b}$ directly instead of the original right-hand side. The elements of this column vector can be calculated as scalar product of columns of \mathbf{Q} and the vector \mathbf{b} :

$$\hat{b}_i = \sum_{j=1}^{r \cdot |V|} q_{ji} b_j,$$

where \hat{b}_i is element of $\mathbf{Q}^T\mathbf{b}$ on the position i , q_{ji} is value of \mathbf{Q} on the position (i, j) and b_j is element of vector \mathbf{b} on the position j . Values of a single column of \mathbf{Q} corresponding to a weight w_{kl} are non-zero only in rows corresponding either to vertex v_k or v_l . Thus, instead of calculating using all the values, only the non-zero values can be used.

5.3 Properties

In this section, all the properties that are not shared across all discrete Laplace operators will be considered, if they are present for the discrete Data Dependent Laplace operator. It is not expected, that this Laplacian does not break any of those properties, as this would be in contradiction of the research of Wardetzky et al. [50].

The Laplacian matrix of Data Dependent Laplace operator is obviously *symmetric*, because the linear system used to calculate the weights was explicitly formulated to result in symmetric weights. It was also already mentioned, that even a non-symmetric variant can be constructed, however, such Laplacian cannot be used with static triangle meshes, unless some other criteria were applied to raise the number of equations.

Another property, that is obviously not broken, is the unit sum of weights. In both symmetric and non-symmetric variant, this property is directly enforced by the weight substitutions.

There is no guarantee, that the weights are *positive*. This property is thus generally broken. To force positive weights, some constrained minimization technique would be used, with inequality constraints. Such techniques are, however, much more complex than linear constrained minimization.

In the case of *linear precision*, it is difficult to decide, whether this property is broken or not. There is no proof, that for any vector on a planar surface, the δ coordinate is zero. However, the matrix was designed, so that all of the δ coordinates are as close to zero as possible. For a vertex that lies on a planar surface, there certainly exists a combination of weights, for which the δ coordinate has zero length. One would expect, that for a vertex far enough from the non-planar part of the surface or boundary, the construction algorithm will assign such combination of weights. To show, whether this is actually true, experiment will be done, and the results will be discussed in Section 6.1.1.

Another property, that is difficult to decide, is *definiteness*. As was already mentioned in the Section 2.2.8, the definiteness in the simple form, is defined only for symmetric matrices. This means, that the non-symmetric variant will not be further concerned. As the property of positive weights can be generally broken, there is no proof, that the matrix is positive definite or positive semi-definite. However, it can be verified experimentally. If one attempts to apply Cholesky decomposition to a matrix derived from a Laplacian matrix by removing one column and its corresponding row, and the decomposition fails, the matrix clearly breaks the positive definite or positive semi-definite property.

The *locality* is unfortunately broken. This is caused by the mechanism that forces the equality of the sum of weights. When for vertex v_i , the weight w_{ij} is marked as dependent and this weight is substituted in all of its occurrences, it means, that the equation of the sum of weights for vertex v_j now contains also the weights assigned to the edges incident to v_i . In the worst case, this dependency chain affects the whole mesh. This means, that by changing the position of a single vertex, not only the δ coordinate of this vertex changes, but generally any of the δ coordinates can change. In the case of forming the weight dependencies, the order, in which the vertices are processed is crucial. One particular order that causes long dependency chains is the one, in which the vertices are reordered during connectivity compression, when encoded by the *Edgebreaker* algorithm. As the Data Dependent Laplacian was initially designed for dynamic mesh compression, it is important to be aware of such issue. To reduce the effect, the vertices can be processed in randomized order. However, the order must be the same on both encoder and decoder side to construct the same Discrete Laplace operator. It is also important to note, that the dependency propagation does not occur in the case of the non-symmetric variant of discrete Data Dependent Laplace operator, as the weight w_{ij} occurs always only in one weight sum equation.

5.4 Weight coding

The compression technique proposed by Váša et al. [44] used a representing frame to construct the geometric Laplacian on both sides of compression. This means, that connectivity and $3|V|$ values had to be transmitted. The discrete Data Dependent Laplace operator, however, can be defined using connectivity and $2|V|$ values, as this is the number of unknowns of the linear system 5.3 in the symmetric case. Thus, instead of a representing frame, one can transmit only the weights that are not dependent.

A naïve approach is to simply quantize all the weights and encode them using a lossless compression technique, for example arithmetic coding. However, it is much more efficient to use a prediction scheme, as the correction value is expected to be much lower than the original value, resulting in lower entropy. However, it is crucial to design an efficient prediction function. One can utilize the formula 5.2. If there is no information about any weight, the best prediction of weight w_{ij} is:

$$p_{ij} = \frac{1}{|N(v_i)|}.$$

The more weights are encoded, the more information the encoder has about their values. This information can be of course used in the prediction. The formula 5.2 can be adjusted by moving all the already known weights to the right-hand side. The sum of all the weights unknown to the encoder decreases, which also means a decrease of their uncertainty and increase of the accuracy of the prediction. The new prediction formula is the following:

$$p_{ij} = \frac{1 - \sum_{v_l \in N_k(v_i)} \hat{w}_{il}}{|N_u(v_i)|},$$

where $N_k(v_i)$ is the set of neighbours of the vertex i that are connected by an edge with already decoded weight and $N_u(v_i)$ is set of neighbours that are connected by an edge with weight unknown to the decoder. The weights \hat{w}_{il} are not the original values, but those reconstructed by the decoder.

The weights are decoded during some vertex traversal. The order of the vertices is important, because it influences the accuracy of the prediction. For example, if they were processed, so that vertices with the highest number of the known weights have higher priority, the rest of the unknown weights would be encoded more accurately, while increasing the number of known weights of neighbouring vertices.

5.5 Possible advantages

At this point, it would be useful to recapitulate all the expected advantages of the proposed Laplacian over the other types. The biggest advantage is obviously the minimization of the lengths of differential coordinates. It is expected, that this would cause a significant decrease of the encoded differential coordinate entropy. Such minimization, however might be also a disadvantage, for example, in mesh smoothing, where slower convergence is expected.

The Data Dependent Laplacian also requires only $2|V|$ values to be encoded instead of the $3|V|$ required by the average mesh. This leads to an assumption, that the encoded weights can be encoded more precisely, while still decreasing the bitrate.

There is still another advantage over other geometric Laplacians that should be mentioned. In order to construct a Laplacian representing the geometry of a whole mesh sequence, a representing frame is usually required to be calculated using some complex, sophisticated method, for example the average in the edge-shape space [44]. With Data Dependent Laplace operator, this can be directly bypassed, because of the way how the weights are calculated. As a consequence, it can be used in mesh morphing without significant adjustments.

6 Experimental results

In this chapter, the experimental results will be shown. The experiments were divided into three groups. The first group of experiments was designed to verify some of the properties of discrete Data Dependent Laplace operator. The second group of experiments serves to compare the results of application of the proposed Laplacian and other types in selected Laplacian mesh processing techniques. The last group of experiments compares the results of application of the proposed Laplacian in dynamic mesh compression.

All the experiments were executed on a 64-bit computer with Intel Core i7-6700HQ (2.6 GHz) CPU, 16GB of RAM, NVidia GeForce GTX 960 M GPU. It is important to note, that time performance was not measured in any of the experiments. All the experiments were written in *C#* programming language, using Bluebit Matrix Library for matrix algebra.

6.1 Verification of properties

In this section, all the properties of the Data Dependent Laplacian that could not be proven to be broken or not, will be verified experimentally.

6.1.1 Linear precision

For this experiment, a mesh representing a cube was created in the Blender 3D modelling software. The connectivity of the cube was altered to obtain non-uniform sampling and different sampling rates. Positions of some points were also altered to create more rough surface. The model can be seen in Figure 6.1a. Following discrete Laplace operators were compared:

- Mean Value Laplacian (*MV*)
- Cotangent Laplacian with unit sum of weights (*Cotan*)
- Cotangent Laplacian weighted by incident areas (*CotanA*)
- Tutte Laplacian (*Tutte*)
- Kirchhoff Laplacian (*Kirchhoff*)
- Data Dependent Laplacian (*DD*)

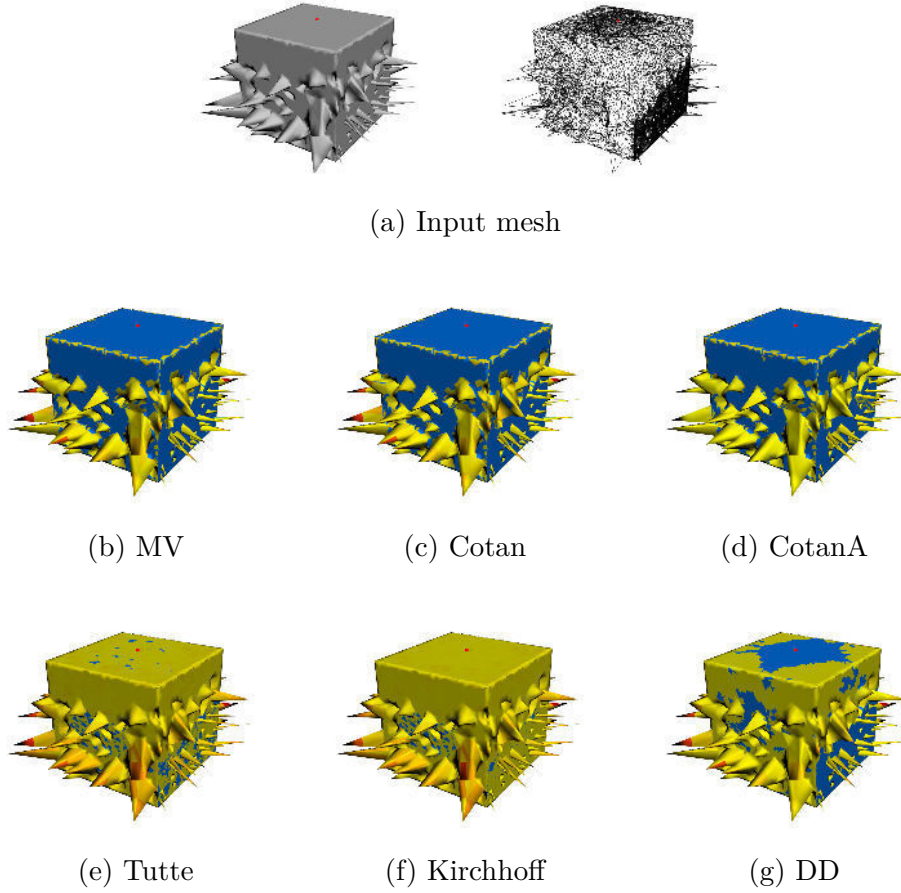


Figure 6.1: Visualization of length of differential coordinates. Azure colour denotes a surface that is considered planar.

During the experiment, for each of the compared Laplacians, the lengths of differential coordinates were measured. Figure 6.1 shows the lengths visualized on the mesh surface. It is important to note, that flat shading was used and that colour scales are not the same between the subfigures. However, the surface, that is considered planar for the used type of discrete Laplace operator, is visualized in azure colour. A point is considered to be on a plane, if the length of its δ -coordinate is less than a threshold value $\varepsilon = 10^{-3}$. The figure shows, that MV Laplacian and both variants of Cotangent Laplacian detected planar surfaces correctly. The Tutte Laplacian and Kirchhoff Laplacian failed to detect most of the planar surface. In the case of the Data Dependent Laplacian, the effects of broken locality are clearly visible around the corners of the cube. However, vertices far enough from a non-planar surface were detected planar correctly by the Data Dependent Laplacian.

Table 6.1 compares the length of differential coordinate of the vertex shown as a red dot in the Figure 6.1. The vertex was specially selected, so that it does not lay in the barycenter of its neighbours. The table shows, that the Data Dependent Laplace operator still results in considerably long differential coordinate in comparison with other geometric Laplacians. However, when compared to Combinatorial Laplacians, the results are better in the order of magnitudes. It is also expected, that with increasing sampling rate, the difference between Geometric Laplacians and Data Dependent Laplacian would decrease.

Table 6.1: Comparison of lengths of differential coordinates in selected point.

	MV	Cotan	CotanA	Tutte	Kirchhoff	DD
$ \delta $	5,55E-17	5,00E-17	4,66E-13	4,39E-02	3,07E-01	1,50E-04

From the results, it can be concluded, that the Data Dependent Laplacian, unfortunately, breaks the linear precision. However, the issues, that are a consequence of this property being broken, should have less impact, than for the Kirchhoff and Tutte Laplacian, because this Laplacian is considerably close to preserving it.

6.1.2 Definiteness

To show, whether a symmetric Laplacian matrix is positive semi-definite or not, two experimental approaches can be used. The first approach would be to calculate all of its eigenvalues and check their signs. The Laplacian matrix is then positive semi-definite if all of the eigenvalues are higher than or equal to zero. This is, however, computationally inefficient, due to the size of the matrix and distance between the eigenvalues.

A simpler approach is to try to apply the $\mathbf{L}^*\mathbf{L}^{*T}$ Cholesky factorization on a matrix created by removing single a row and corresponding column for each directly connected component of the mesh. Removing such rows and columns results in the non-singular square matrix. Thus, if the Laplacian matrix was positive semi-definite, the resulting matrix should be positive definite. Most of the implementations of the Cholesky factorization raise an exception or set some flag, when the matrix is not positive definite.

The experiment was done using Eigen C++ library [16]. All the symmetric Laplacians were tested by applying the Cholesky factorization implemented in the `SimplicialLLT` class. To check whether the factorization was successful, `info()` function was then called. The returned value was either 0 - Success, or 1 - Failure. The compared Laplacians were:

- Kirchhoff Laplacian (*Kirchhoff*)
- Symmetric Cotangent Laplacian (*SymCotan*)
- Data Dependent Laplacian (*DD*)

Both Kirchhoff and Symmetric Cotangent Laplacian are positive semi-definite, thus it was expected that the factorization of the reduced matrix would be successful. The Table 6.2 shows the factorization results for the average frame [44] of the Samba dataset [49]. It is important to note, that by finding any mesh, for which the Data Dependent Laplacian is not positive semi-definite, it is actually proven that generally the positive semi-definiteness property is broken. However, when the Cholesky factorization is successful for a single mesh, it is not guaranteed that it will be successful for any arbitrary mesh.

Table 6.2: The $\mathbf{L}^*\mathbf{L}^{*T}$ Cholesky factorization results for the average frame[44] of the Samba dataset[49]

	Kirchhoff	SymCotan	DD
Result	Success	Success	Failure

Both Kirchhoff and symmetric Cotangent Laplacian are positive semi-definite, so it was no surprise, that the Cholesky factorization did not fail. For the Data Dependent Laplacian, the Cholesky factorization failed. Thus, it can be concluded that the Laplacian matrix of such Laplacian is not positive semi-definite.

6.2 Application in Laplacian mesh processing

In this section, experimental application of the proposed Laplacian in mesh processing techniques described in Chapter 3 will be shown. The results were visualized using *SlimDXRenderer* framework [22], which was also used in the project *Mesh statistics for robust curvature estimation* [42], and Blender modelling software. The program used for testing is included on the attached DVD.

6.2.1 Mean curvature estimation

The Data Dependent Laplacian was designed to minimize the lengths of δ -coordinates. This means, that it is not expected to estimate the values

of mean curvature exactly in all the vertices. However, the distribution of the values should still correspond to the exact values. The correspondence can be measured by correlation, which tells, how close two sets of randomly distributed values are to being linearly dependent.

For this experiment, Pearson correlation coefficient was used as a metric:

$$r = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^n (y_i - \bar{y})^2}},$$

where \bar{x} is mean value of the first set of values, \bar{y} is mean value of the second set, and $-1 \leq r \leq 1$ is the Pearson correlation coefficient. If $|r| = 1$, there exists a value s , for which every value $x_i = s \cdot y_i$. The closer to one $|r|$ is, the more correlated the two sets of values are.

It is important to calculate the correlation with exact curvature values. For this reason, some analytic function, on which the exact values can be calculated, must be used. In this experiment, the correlation was measured on the surface of so-called *peaks* function:

$$f(x, y) = 3(1 - x)^2 e^{-(x^2 - (y+1)^2)} - 10\left(\frac{x}{5} - x^3 - y^5\right) e^{-(x^2 - y^2)} - \frac{1}{3} e^{-(x+1)^2 - y^2}.$$

This function can be seen in Figure 6.2. The function was randomly sampled in the interval from -4 to 4 in both x and y axe. The number of samples was 10000 and the surface of the triangle mesh was then created using Delaunay triangulation. For generating mesh and calculating the exact mean curvature values, the *CurvatureBenchmark* [42] was used.

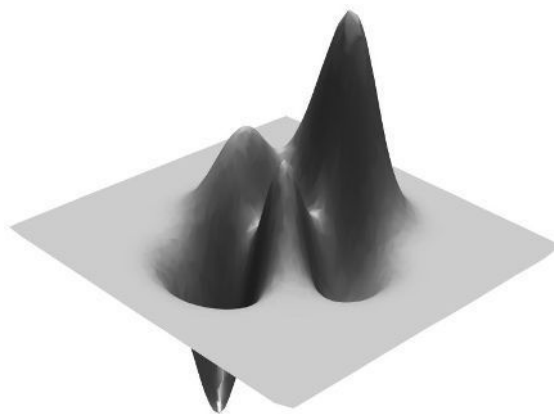


Figure 6.2: *Peaks* function

Following discrete Laplace operators were compared:

- Mean Value Laplacian (*MV*)
- Cotangent Laplacian with unit sum of weights (*Cotan*)
- Cotangent Laplacian weighted by incident areas (*CotanA*)
- Tutte Laplacian (*Tutte*)
- Kirchhoff Laplacian (*Kirchhoff*)
- Data Dependent Laplacian (*DD*)

Mean curvature estimation using discrete Laplace operator usually results in inaccurate values near the boundary of the mesh. For this reason, both exact and estimated curvature were considered zero on boundary vertices and their neighbours. Figure 6.3 shows the measured mean curvature visualized on the surface.

All of the Laplacians performed considerably well in the sense that they all detected positive mean curvature on the peaks and negative curvature in valleys. The performance of combinatorial Laplacians was clearly worse, as a red colour, that marks the places with highest curvature, is frequently present in places where no local extreme should be. The Data Dependent Laplacian seems to be less vulnerable to this issue, however, for its broken locality, there are places, where it detects negative curvature instead of positive. However, the places with highest estimated curvature correspond to the exact local maxima. In the case of correlation with exact values, the results are shown in Table 6.3.

Table 6.3: Pearson correlation coefficient of measured mean curvatures and exact values

	MV	Cotan	CotanA	Tutte	Kirchhoff	DD
r	0,781	0,764	0,866	0,389	0,377	0,636

The highest correlation was achieved using Cotangent Laplacian weighted by areas incident to vertices. The lowest correlation was achieved by combinatorial Laplacians. The Data Dependent Laplacian is unfortunately the worst of all geometric Laplacians, although any result with correlation higher than 0,5 can be considered a decent mean curvature estimation.

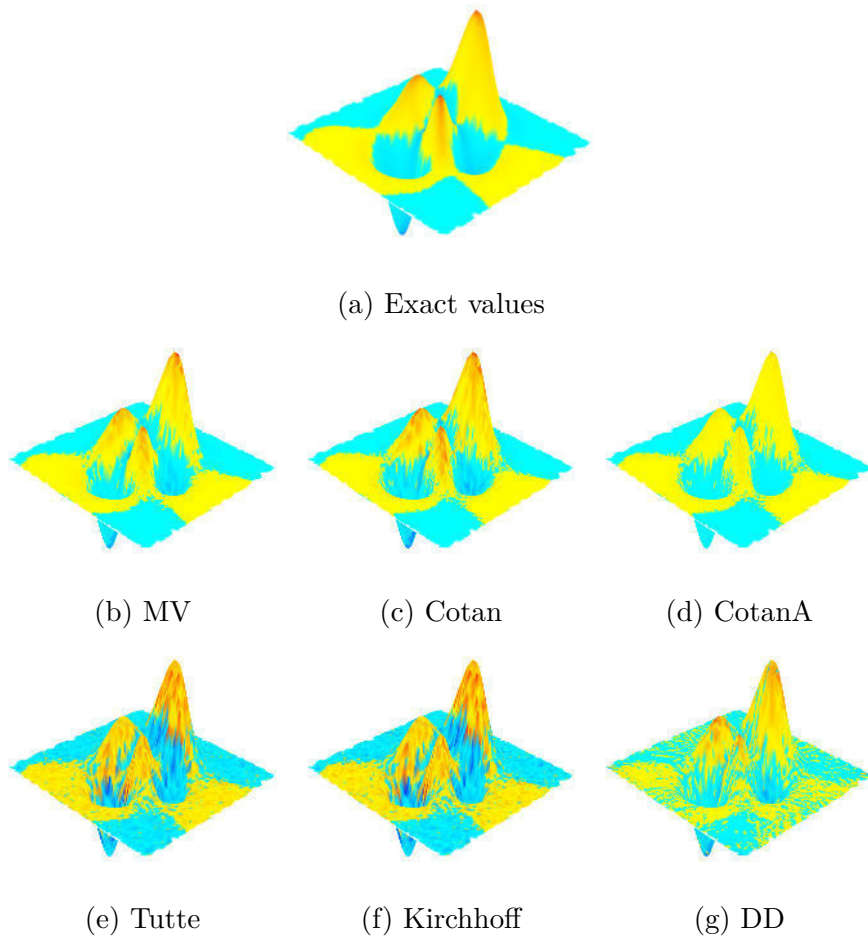


Figure 6.3: Estimation of mean curvature visualized on the surface of *peaks function*. Yellow and red colour denotes a surface with positive mean curvature, blue colour denotes a surface with negative mean curvature.

6.2.2 Smoothing

Three smoothing techniques were tested: Basic iterative algorithm, $\lambda|\mu$ algorithm, and Implicit mesh fairing. No metric was used during this experiment, only a visual evaluation of results was done. Unfortunately, no tests were made on dynamic meshes, due to the large scope of this thesis.

In the case of basic iterative algorithm, two tests were done. Only the Laplacians that do not break unit sum of weights were considered:

- Tutte Laplacian (*Tutte*)
- Mean Value Laplacian (*MV*)
- Cotangent Laplacian with unit sum of weights (*Cotan*)

- Data Dependent Laplacian (*DD*)

The first test was performed on the *Lion* model consisting of 2119 vertices (see Figure 6.4a), with $\lambda = 0,6$ and 20 iterations. The results can be seen in Figure 6.4.

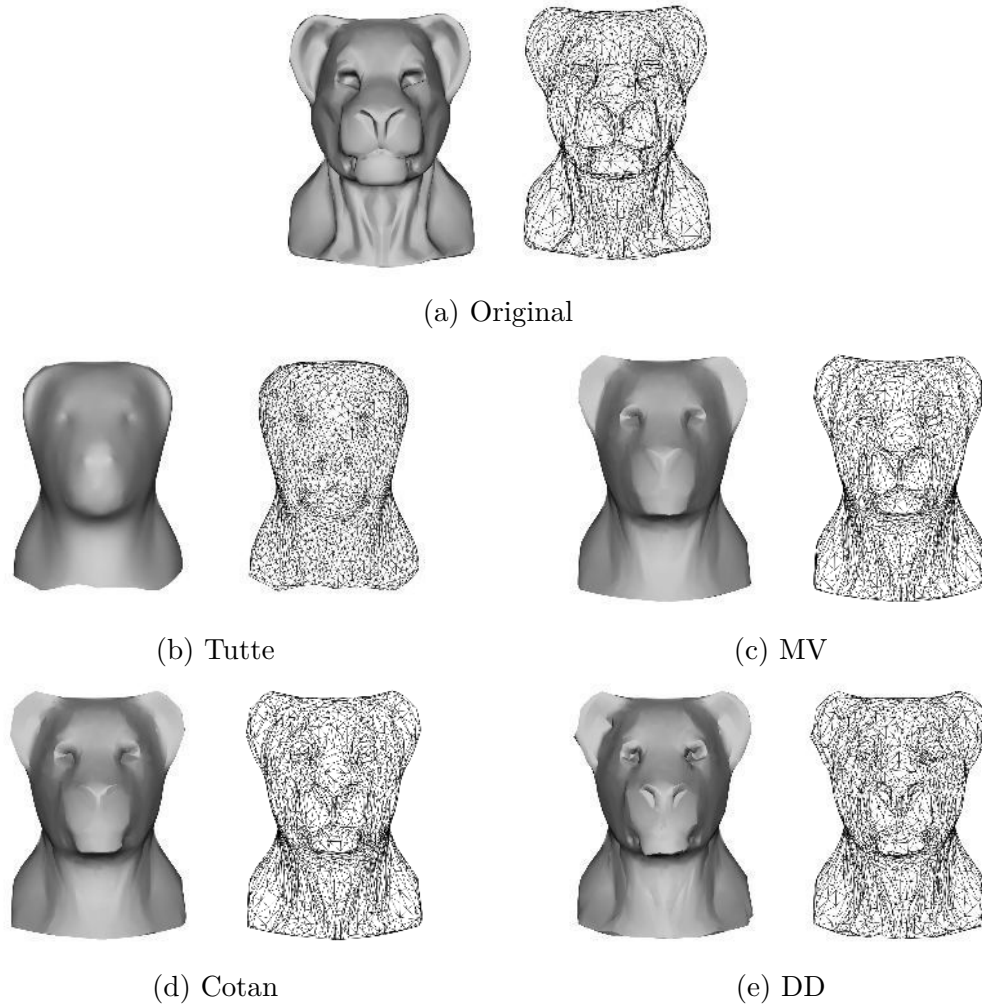


Figure 6.4: Basic iterative smoothing of Lion model ($\lambda = 0,6$, 20 iterations)

Application of Tutte Laplacian resulted in the most smoothed model. However, the result is also shrunk and details were lost. It also experienced the tangential drift. The Mean Value and Cotangent Laplacians have comparable results - visible detail, but decently smoothed surface. The Data Dependent Laplacian unfortunately results in the least smoothed model. This might be caused by the fact, that it tries to minimize the length of differential coordinates, which means less movement in each iteration. Another issue with Data Dependent Laplacian seems to be that some of the details

that should be smoothed are not smoothed at all. Notice the eyes, nose and chin.

Second test was done on a model of a cube with randomly displaced vertices in the direction of surface normal. The displacement was performed to simulate the noise in 3D model data. The results of smoothing such mesh with 20 iterations and $\lambda = 0,6$ can be seen in Figure 6.5.

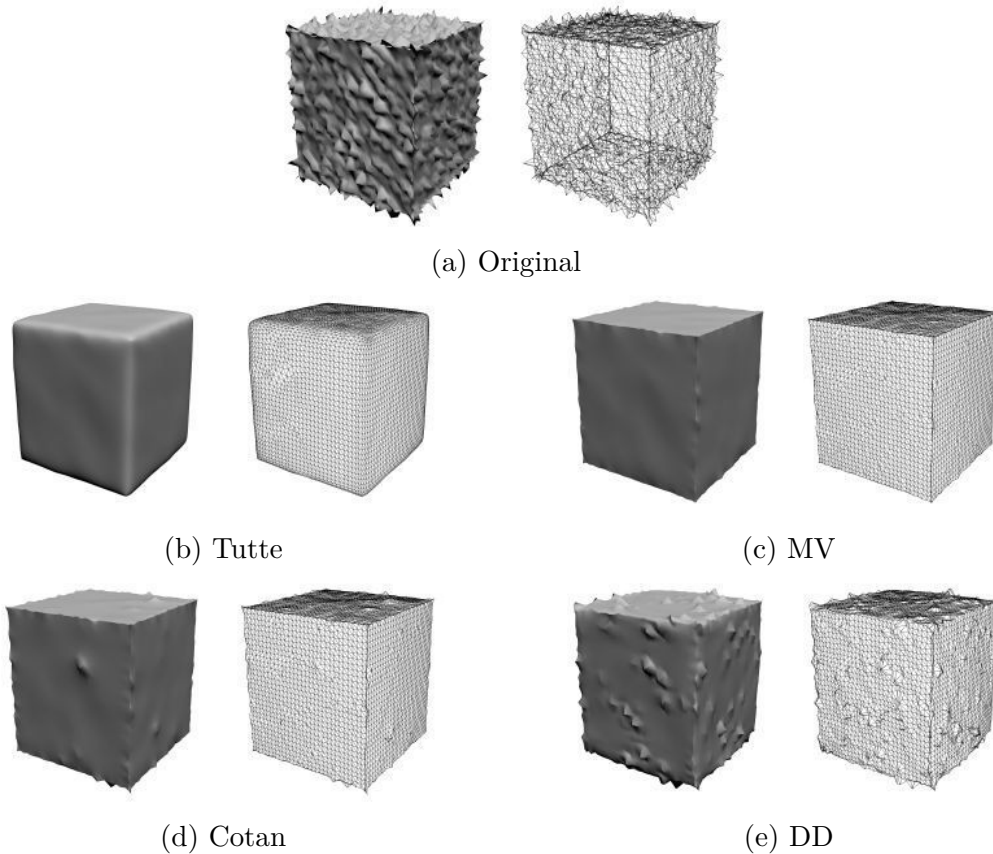


Figure 6.5: Basic iterative smoothing of cube with noise ($\lambda = 0,6, 20$ iterations)

Again, the Tutte Laplacian resulted in the most smoothed model. It is also the only Laplacian that smoothed the edges of the cube. If the edges of the cube should be preserved, the Mean Value Laplacian achieves better results. After applying the Cotangent Laplacian, there is still some visible noise. However, it still performed better than the Data Dependent Laplacian, which, again, resulted in least smoothed model. From these two tests, it seems that the Data Dependent Laplacian is not very suitable for basic iterative smoothing algorithm.

Same models from previous two tests were also processed using the $\lambda|\mu$ algorithm. The compared discrete Laplace operators were identical to the

ones in the previous test. Instead of tweaking the two parameters to obtain the best results, the values suggested by Desbrun et al. [11], when they compared this algorithm with their method, were used: $\lambda = 0,6307, \mu = -0,6732$. The Lion model was processed in 50 iterations and the cube with noise was processed in 35 iterations.

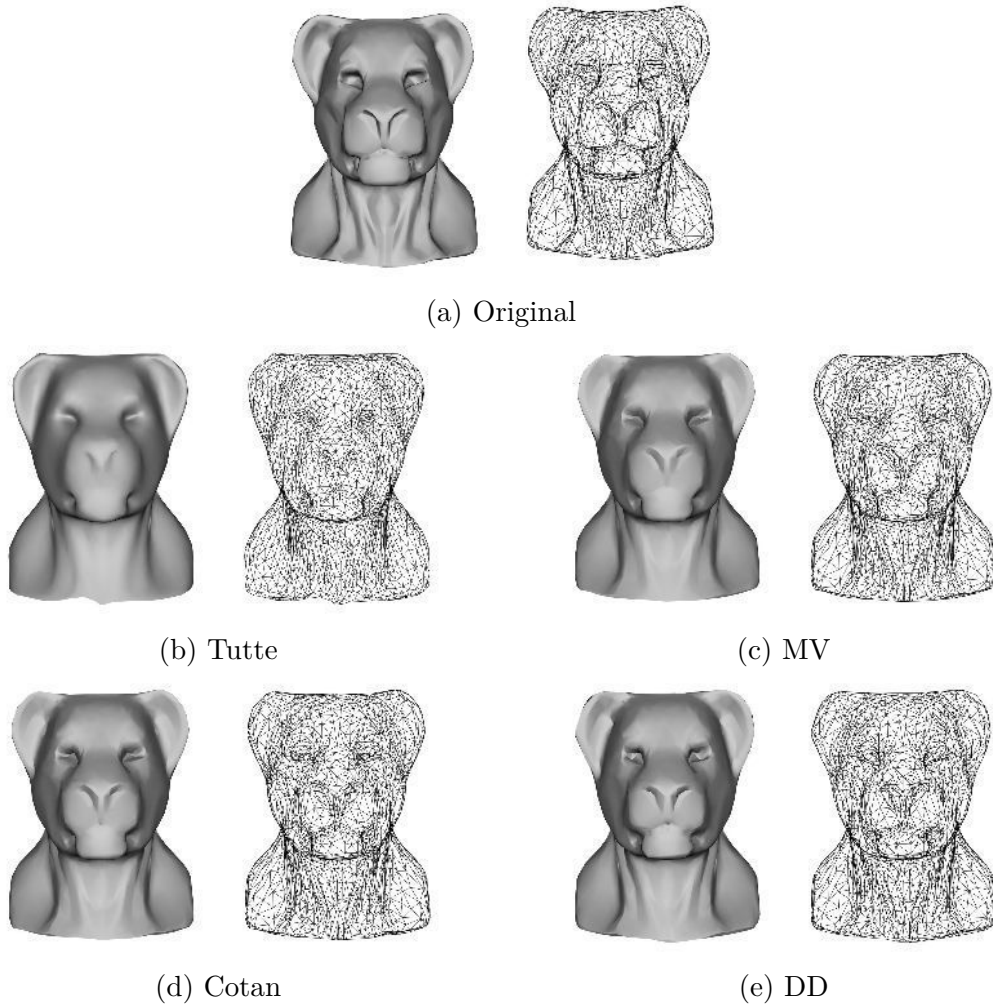


Figure 6.6: $\lambda|\mu$ smoothing of Lion model ($\lambda = 0,6307, \mu = -0,6732, 50$ iterations)

The results in Figure 6.6 have verified, that this technique converges slower to smooth surface than the basic iterative technique. On the other hand, all the resulting meshes are approximately of the same size as the input mesh, thus the shrinking was prevented. Again, the Tutte Laplacian resulted in the smoothest surface. In this case, there is still visible detail of the mesh. The results of Mean Value Laplacian and Cotangent Laplacian are indistinguishable. The Data Dependent Laplacian, again, converged

the slowest. However, there are no artifacts, that were visible, when basic iterative technique was used.

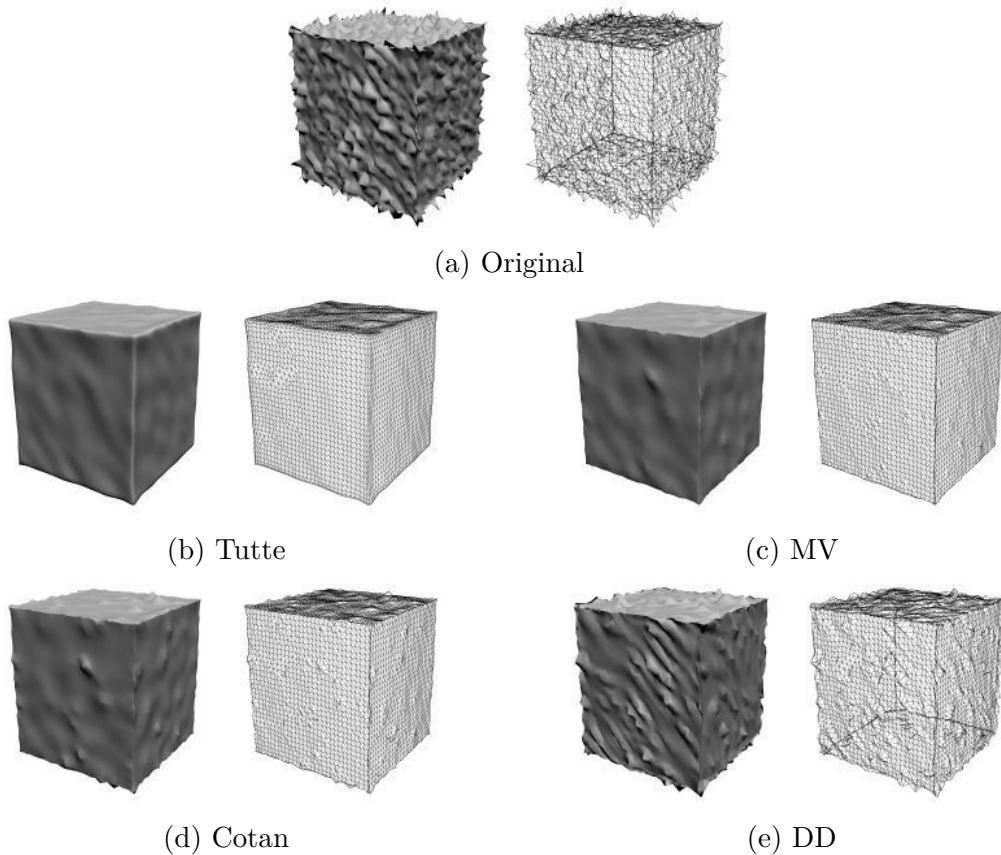


Figure 6.7: $\lambda|\mu$ smoothing of of cube with noise ($\lambda = 0,6307, \mu = -0,6732, 35$ iterations)

In the case of a cube with noise, the best smoothing was achieved using Tutte and MV Laplacian. The MV Laplacian achieved sharper edges of the cube, however, the Tutte Laplacian removed more of the noise. The noise is still significantly visible for Data Dependent Laplacian and Cotangent Laplacian.

The last set of tests in this experiment were done to compare the results of implicit mesh fairing technique. As the implicit mesh fairing does not require unit sum of weights, it was possible to compare more variants of discrete Laplace operators:

- Tutte Laplacian (*Tutte*)
- Kirchhoff Laplacian (*Kirchhoff*)
- Mean Value Laplacian (*MV*)

- Cotangent Laplacian with unit sum of weights (*Cotan*)
- Cotangent Laplacian weighted by incident areas (*CotanA*)
- Data Dependent Laplacian (*DD*)

However, there is one issue with the Laplacians that break the unit sum of weights: It is impossible to use the same smoothing coefficient for all the types, if it is desired to obtain comparable results. For this reason, when using Kirchhoff or area weighted Cotangent Laplacian, the smoothing coefficient had to be adjusted:

$$\lambda' = \frac{\lambda}{s_L},$$

where s_L is average absolute value on the diagonal of \mathbf{L} .

First, the implicit mesh fairing was applied on the Lion model. Only two iterations were performed, with $\lambda = 10$. The results can be seen in Figure 6.8.

The best result was obtained using Cotangent Laplacian weighted by areas incident to vertices, because it smoothed the surface the best of all geometric Laplacians, while preserving the most detail. It is only Laplacian of all, that preserved the shape of the Lion's shoulders. All the other Laplacians still performed well. Even the Data Dependent Laplacian seems to smooth the surface significantly.

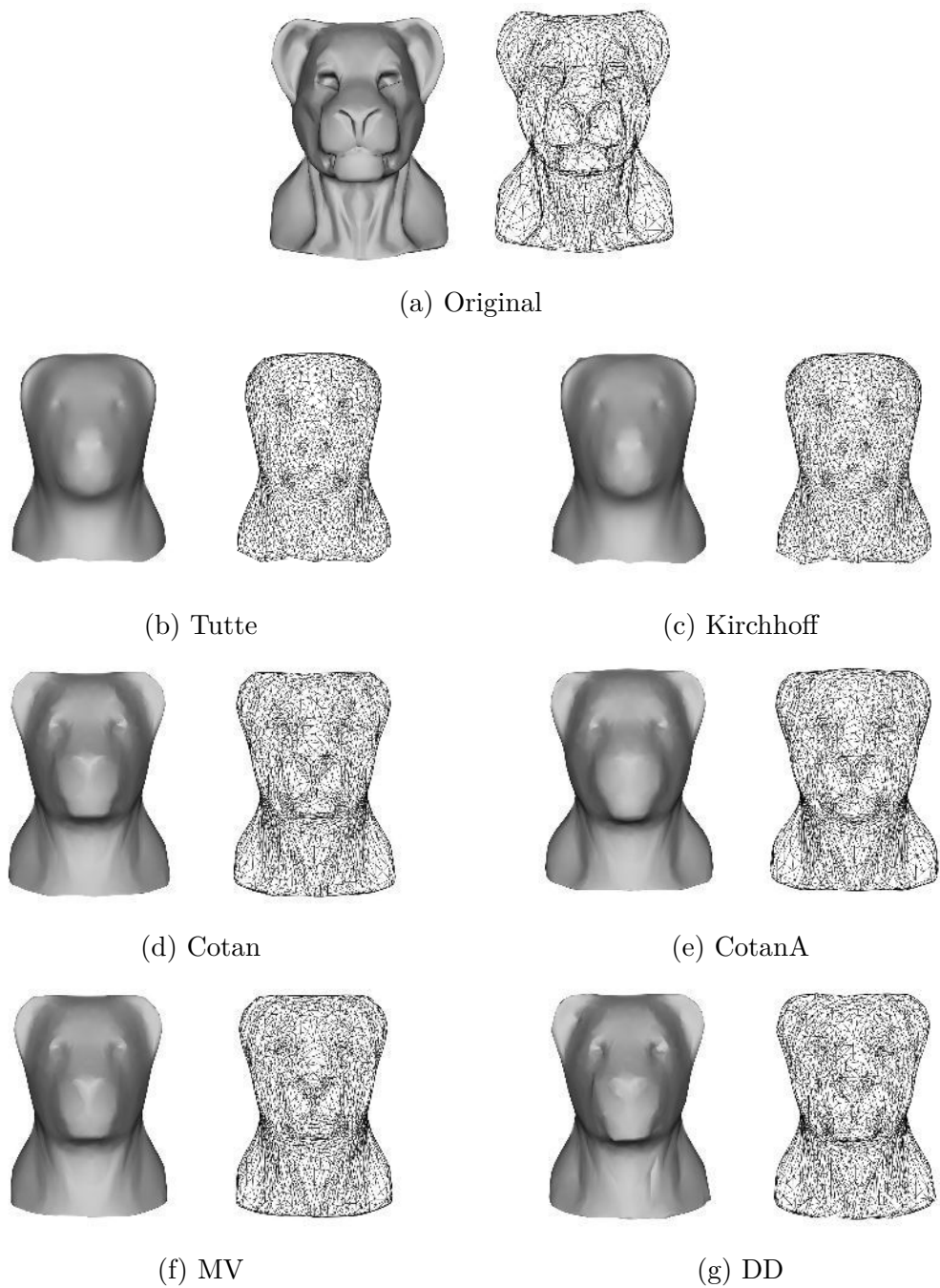


Figure 6.8: Implicit mesh fairing applied on the Lion model ($\lambda = 10$, 2 iterations)

In the second test, again, the cube with noise was smoothed in two iterations, with $\lambda = 0,5$. The reason, why the λ is so small, will be explained in following paragraph, where the results will be evaluated. The results can be seen in Figure 6.9.

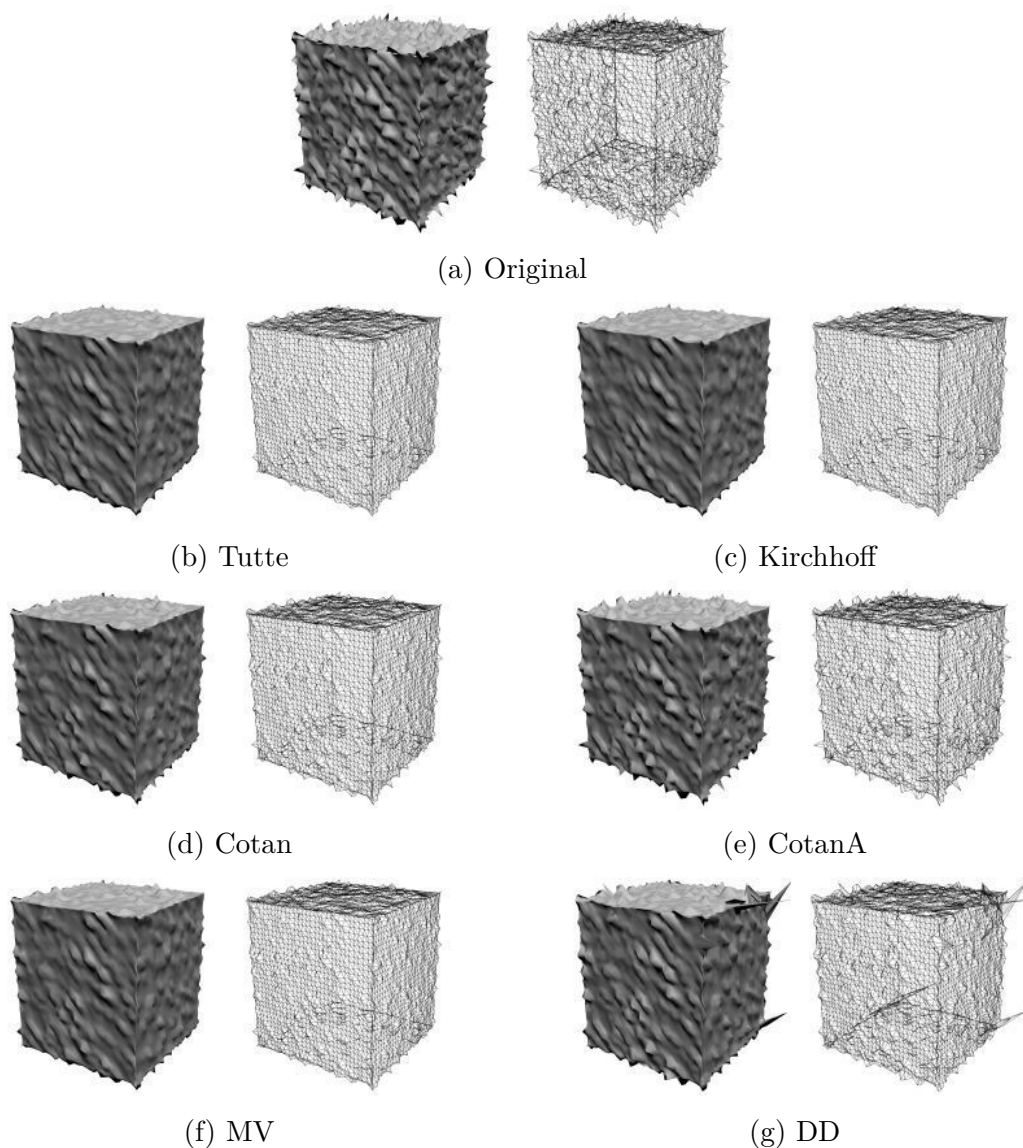


Figure 6.9: Implicit mesh fairing applied on a cube with noise ($\lambda = 0.5$, 2 iterations)

The smoothing coefficient is so small, that none of the Laplacians managed to erase the noise. However, significant artifacts are visible in the result of smoothing by Data Dependent Laplacian.

To further investigate the issue, additional test was done. In this test, the undistorted model of the cube was used (see Figure 6.10a). Instead of comparing the results with other types of Laplacians, the comparison was done between different smoothing coefficients:

- $\lambda = 8$
- $\lambda = 8,5$
- $\lambda = 9$
- $\lambda = 9,5$
- $\lambda = 10$

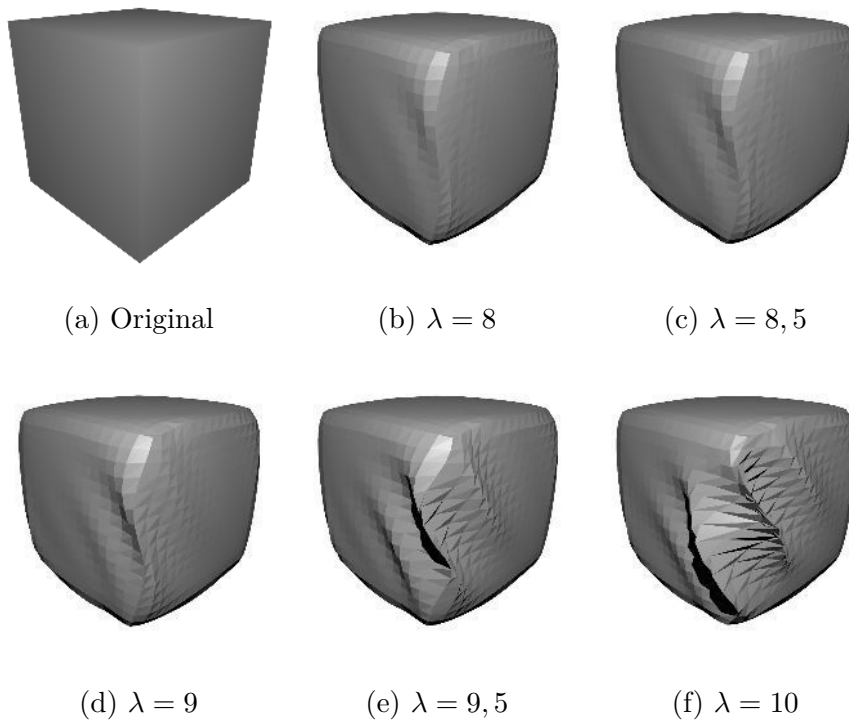


Figure 6.10: Influence of the smoothing parameter λ on artifact visibility in implicit mesh fairing technique.

The results have shown, that with increasing the smoothing parameter, the artifacts are more visible and unpleasant. This issue occurs only in this technique, which differs from other tested techniques by requiring to solve a linear system. This leads to a conclusion, that the matrix of the system is ill-conditioned, which also could mean that the matrix of Data Dependent Laplacian has conditioning issues.

6.2.3 Parameterization

Previous experiment has pointed out one big issue of Data Dependent Laplacian. During barycentric mapping parameterization technique, it is required to solve a linear system that is formed by the weights of Laplacian. Even though the matrix of the system is not the same as the one in implicit mesh fairing, some conditioning issues are expected, when Data Dependent Laplacian used.

During this experiment, the barycentric mapping was applied to three models: already mentioned Lion model, and two models created from the first frame of the Samba dataset [49] (see Figures 6.12a and 6.13a). All of the processed models have boundary, as it is required by the tested technique.

To measure the quality, difference in angles was measured. The difference around a vertex was visualized on the surface of the mesh, to show the distribution of the distortion over the surface. Comparison was also done in maximum and average angle difference. The most importantly, the resulting parameterization was applied in texture mapping, where checkerboard pattern was mapped on the surface to inspect texture distortion. Compared Laplacians were:

- Tutte Laplacian (*Tutte*)
- Mean Value Laplacian (*MV*)
- Cotangent Laplacian with unit sum of weights (*Cotan*)
- Data Dependent Laplacian (*DD*)

First, the method was applied on Lion model. The results can be seen in the Figure 6.11 and in the Table 6.4. First row of results shows textures mapped on the surface using the resulting parameterizations. The second row visualizes angle difference around vertices. All the results have the same color scale. The red colour denotes a part of the surface with high angle difference. In such places, the texture is expected to be the most distorted. The lowest row shows the obtained parameterizations.

Table 6.4: Measured angle difference in parameterizations of Lion model

	Tutte	MV	Cotan	DD
Sum	5308,699	1463,492	885,760	2753,574
Max	2,025	1,075	1,028	2,710
Avg	0,422	0,116	0,070	0,219

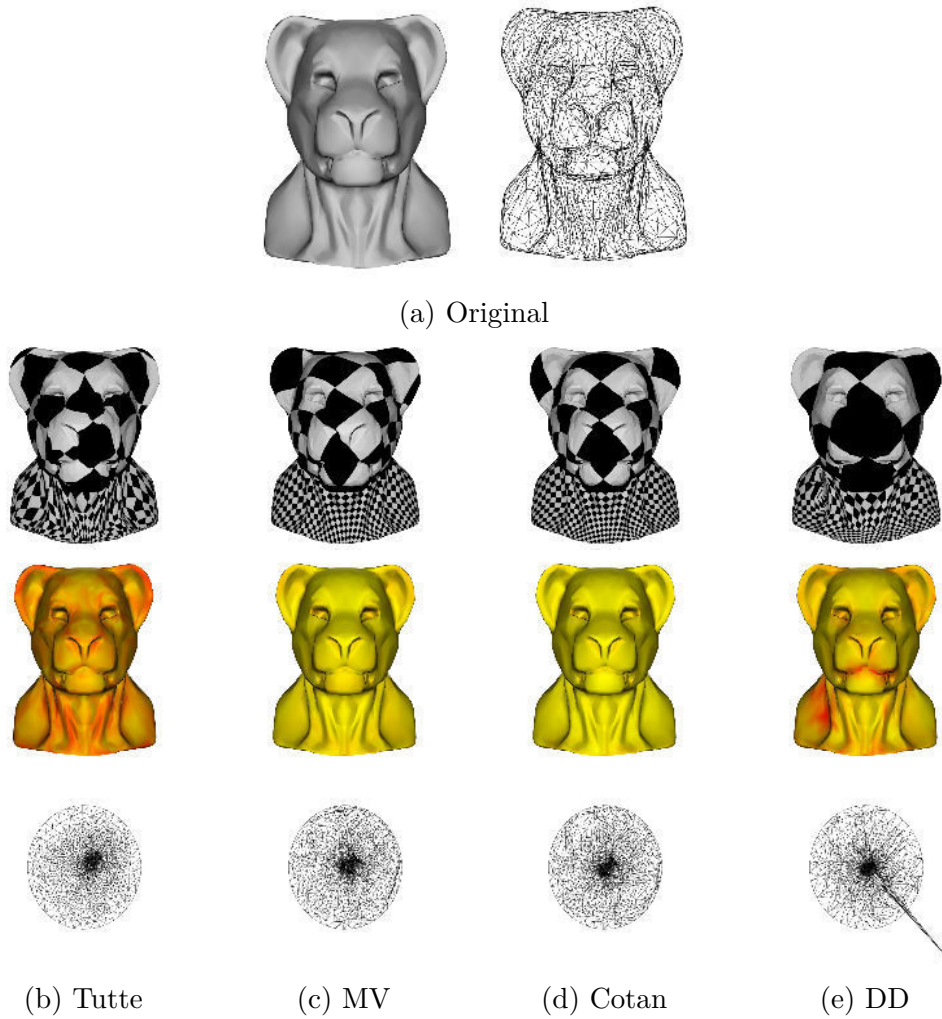


Figure 6.11: Parameterization of the Lion model

Visually, the worst result was obtained using Tutte Laplacian. The distortion of the checker pattern is distributed all over the surface. This is also visible in angle differences. However, the obtained parameterizations have shown, that there were issues with Data Dependent Laplacian. Some of the points were mapped outside of the convex area formed by the boundary, and also there is visible triangle overlap, which means that such mapping is not one-to-one. This effect is also visible on the right shoulder of the model in the visualized angle differences and on the texture. It could be caused by the two factors: the negative weights of Data Dependent Laplacian and conditioning issues. Even though there is no visible difference in Figure 6.11 between results of barycentric mapping using Mean value and Cotangent Laplacian, the second one seems to be a bit better, as shows the Table 6.4.

The second test was done on a modified frame of the Samba dataset

[49], where only an upper front part of the modelled body was preserved. The model consisted of the 2648 vertices. The biggest angle difference was expected to be on the arms of the model. The results are shown in Figure 6.12 and Table 6.5.

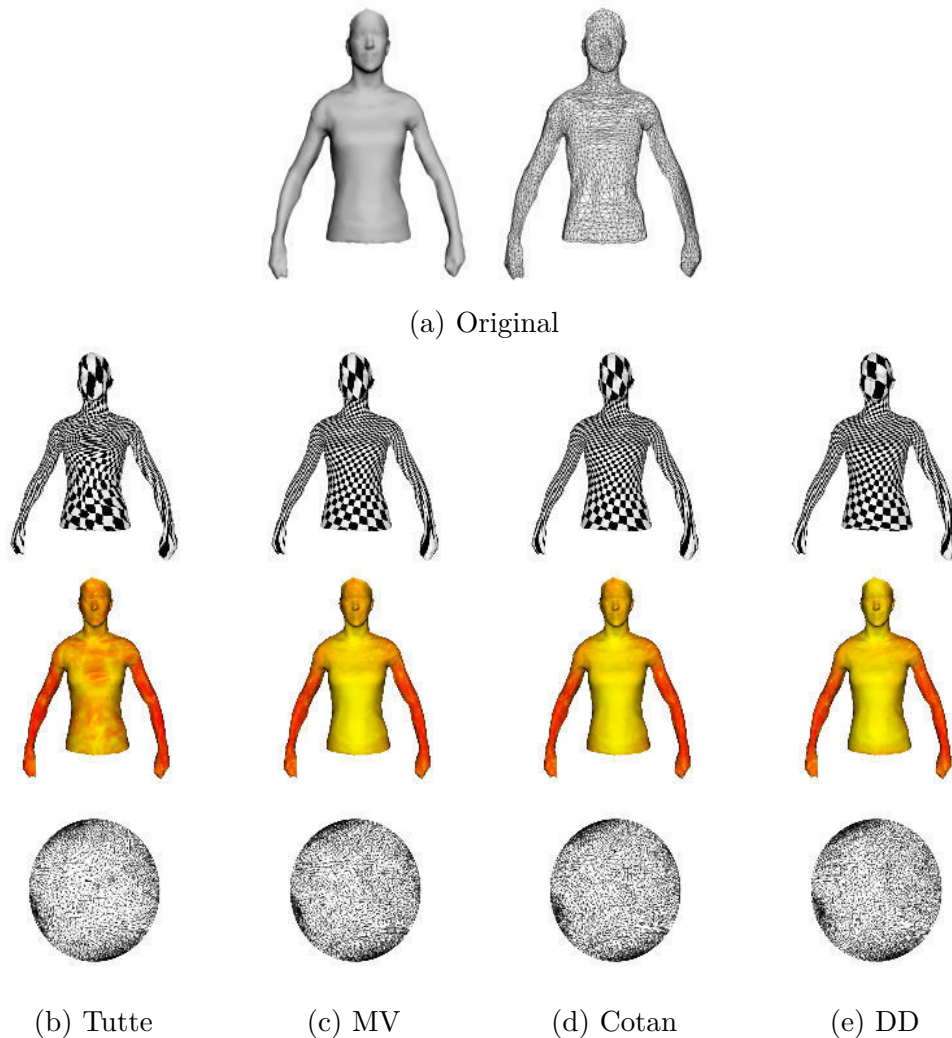


Figure 6.12: Parameterization of the first modified model from the Samba dataset [49]

As was expected, the biggest angle difference was on arms of the model. Again, visually, the worst results were obtained using Tutte Laplacian. This time, however, there was no issue with Data Dependent Laplacian. In fact, the Table 6.5 shows, that the Data Dependent Laplacian resulted in the lowest sum of angle differences and average angle difference. Also, when inspecting the texture mapped on the surface, this Laplacian obtained the best parameterization in the lower part of the model.

Table 6.5: Measured angle difference in parameterizations of the first modified model from the Samba dataset [49]

	Tutte	MV	Cotan	DD
Sum	10770,094	8846,555	8811,342	8312,775
Max	2,590	2,392	2,383	2,420
Avg	0,724	0,595	0,592	0,559

The second tested modification of the model from Samba dataset [49] was directly derived from the previous one. This time, only the face of the model was preserved. Such triangle mesh had only 317 vertices. As the boundary was quite close to the convex polygon, it was expected, that there would not be as much of the distortion in such places.

Table 6.6: Measured angle difference in parameterizations of the second modified model from the Samba dataset [49]

	Tutte	MV	Cotan	DD
Sum	612,056	322,303	325,706	412,636
Max	1,746	1,681	1,681	1,681
Avg	0,354	0,187	0,188	0,239

The results have shown, that for such mesh, all of the Laplacians except of the Tutte Laplacian resulted in decent parameterization. The Data Dependent Laplacian seems to obtain slightly worse results. There is slightly higher distortion of the texture on the nose. Interestingly, the maximum angle difference was the same for all the geometric Laplacians. The reason could be, that such difference occurred on the boundary, where the vertices are mapped on the convex polygon. Such polygon is the same for all the applied Laplacians.

Unfortunately, for the extended scope of all the other experiments, the parameterization was tested only on static meshes.

The experiments have shown, that Data Dependent Laplacian can be quite useful in the barycentric mapping technique. There might still occur some issues with matrix conditioning, however, for some triangle meshes, this Laplacian can obtain better results than the Mean Value Laplacian that is usually used.

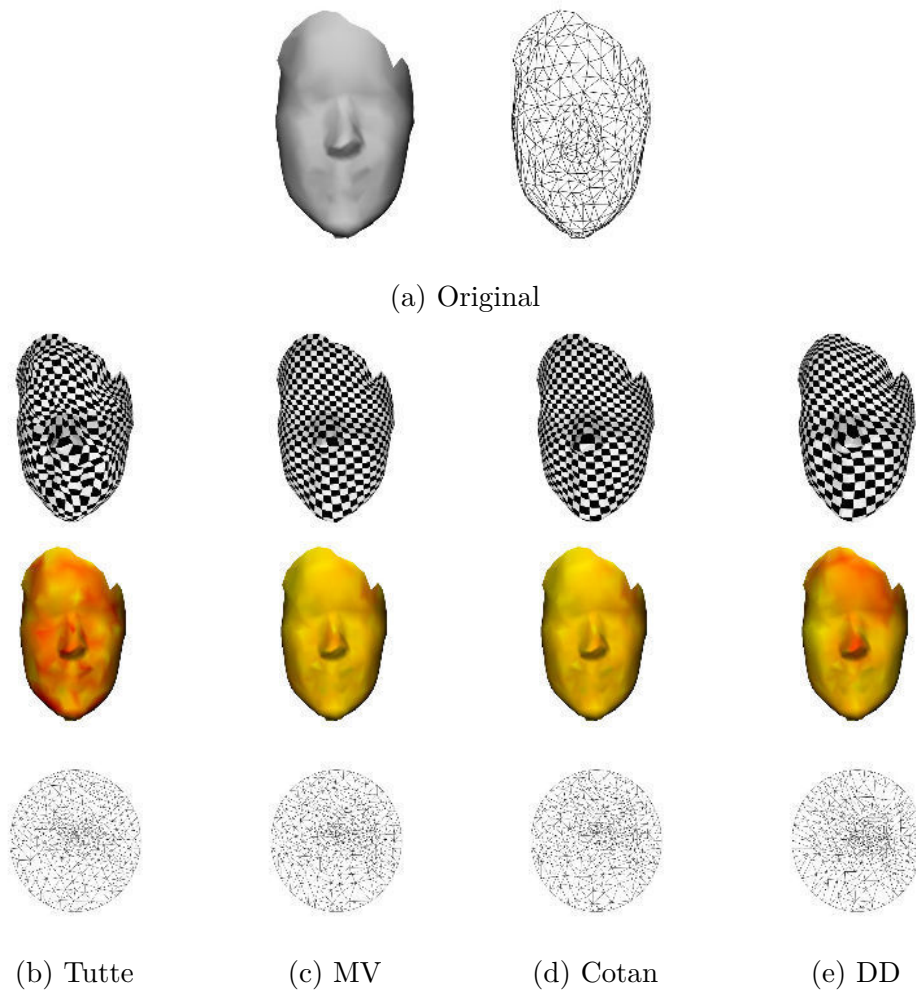


Figure 6.13: Parameterization of the second modified model from the Samba dataset [49]

6.2.4 Mesh editing

In this experiment, the Laplacian surface editing technique[37] was applied on the Lion model and the first frame of the Samba dataset [49]. The selection of handles, unconstrained vertices and soft constraints is seen in Figure 6.14. Angle and area differences were measured, although, in this case, their value is only informative and cannot serve as a measurement of the quality of the method. Compared discrete Laplace operators were:

- Tutte Laplacian (*Tutte*)
- Kirchhoff Laplacian (*Kirchhoff*)
- Mean Value Laplacian (*MV*)

- Cotangent Laplacian with unit sum of weights (*Cotan*)
- Cotangent Laplacian weighted by incident areas (*CotanA*)
- Data Dependent Laplacian (*DD*)

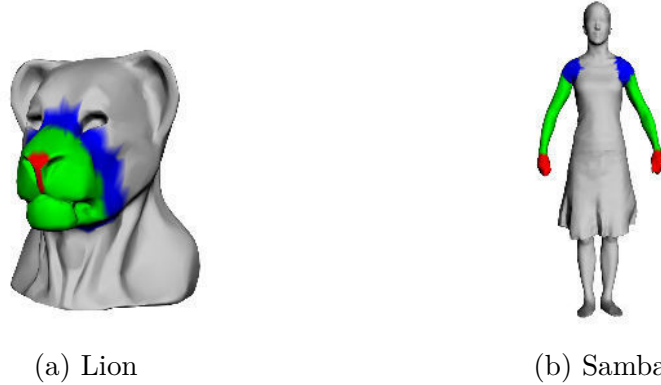


Figure 6.14: Testing models for Laplacian surface editing. Red area denotes handles, green area denotes unconstrained vertices and blue area denotes soft constraints.

In the case of the Lion model, the tip of the nose was selected as a handle, while a circle around the nose was selected as soft constraints. Rest of the vertices between handles and soft constraints was marked as unconstrained. The handle was moved forward and slightly rotated. The results can be seen in Figure 6.15 and Tables 6.7 and 6.8.

Table 6.7: Measured angle difference in Laplacian surface editing of the Lion model

	MV	Cotan	CotanA	Tutte	Kirchhoff	DD
Sum	511,630	523,355	662,862	628,086	703,965	536,575
Max	1,441	1,473	1,095	1,943	1,810	1,799

Table 6.8: Measured area difference in Laplacian surface editing of the Lion model

	MV	Cotan	CotanA	Tutte	Kirchhoff	DD
Sum	1,656	1,724	2,460	1,889	2,055	1,617
Max	0,025	0,027	0,037	0,023	0,020	0,026

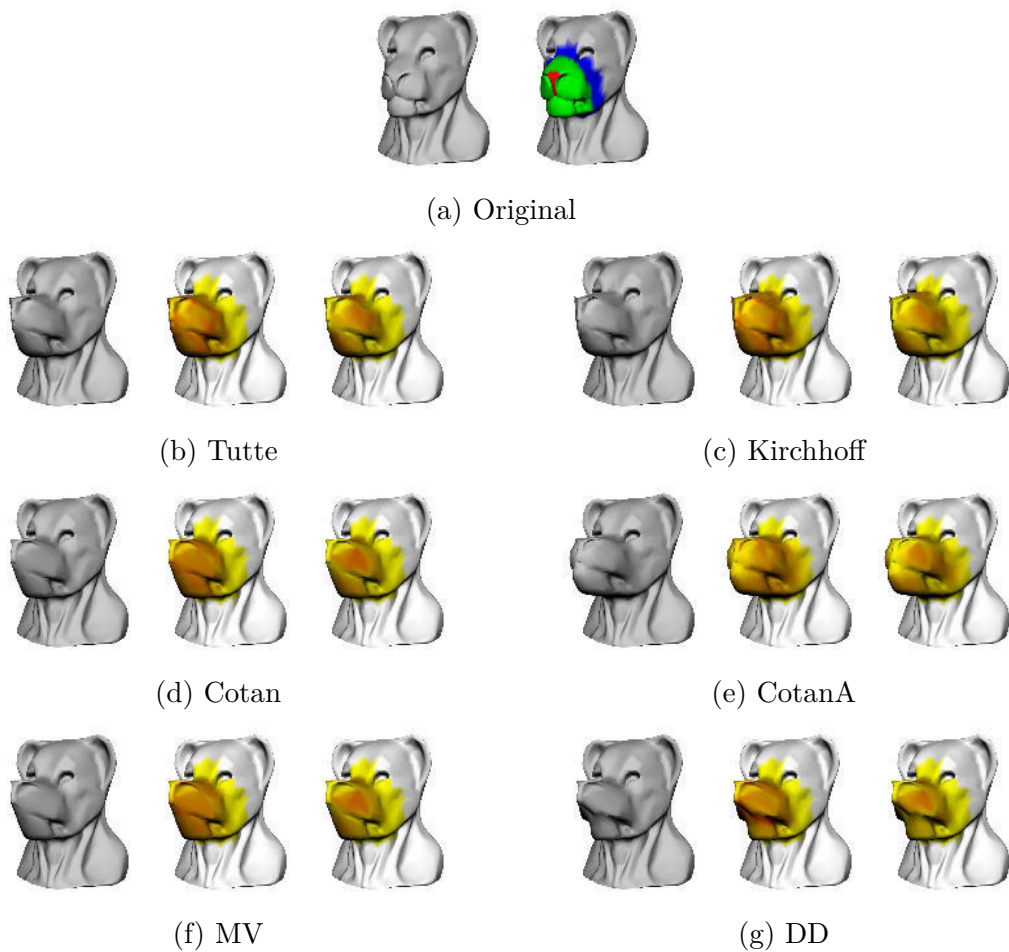


Figure 6.15: Laplacian surface editing applied on the Lion model. From left to right for each discrete Laplace operator: result of the deformation, angle difference, area difference.

In this case, all the discrete Laplace operators obtained satisfying results. Only issue was a triangle intersection on the boundary of the handles in the case of combinatorial Laplacians. It is impossible to determine which of the Laplacians obtained best results. For example, even though the Cotangent Laplacian weighted by incident areas resulted in the most visually pleasing deformation, it was worst in the area preservation. On the other hand, the result obtained using Data Dependent Laplacian seems unrealistic, however, it has the lowest sum of area differences and it is the only discrete Data Dependent Laplacian, that preserved the shape of the Lion's chin.

In the second test, the first frame of the Samba dataset was deformed. The hands of the modelled person were selected as handles, the rest of the arms as unconstrained vertices and shoulders as soft constraints. The hands

were then moved farther from the body and slightly rotated. The results of such deformation can be seen in Figure 6.16 and Tables 6.9 and 6.10.

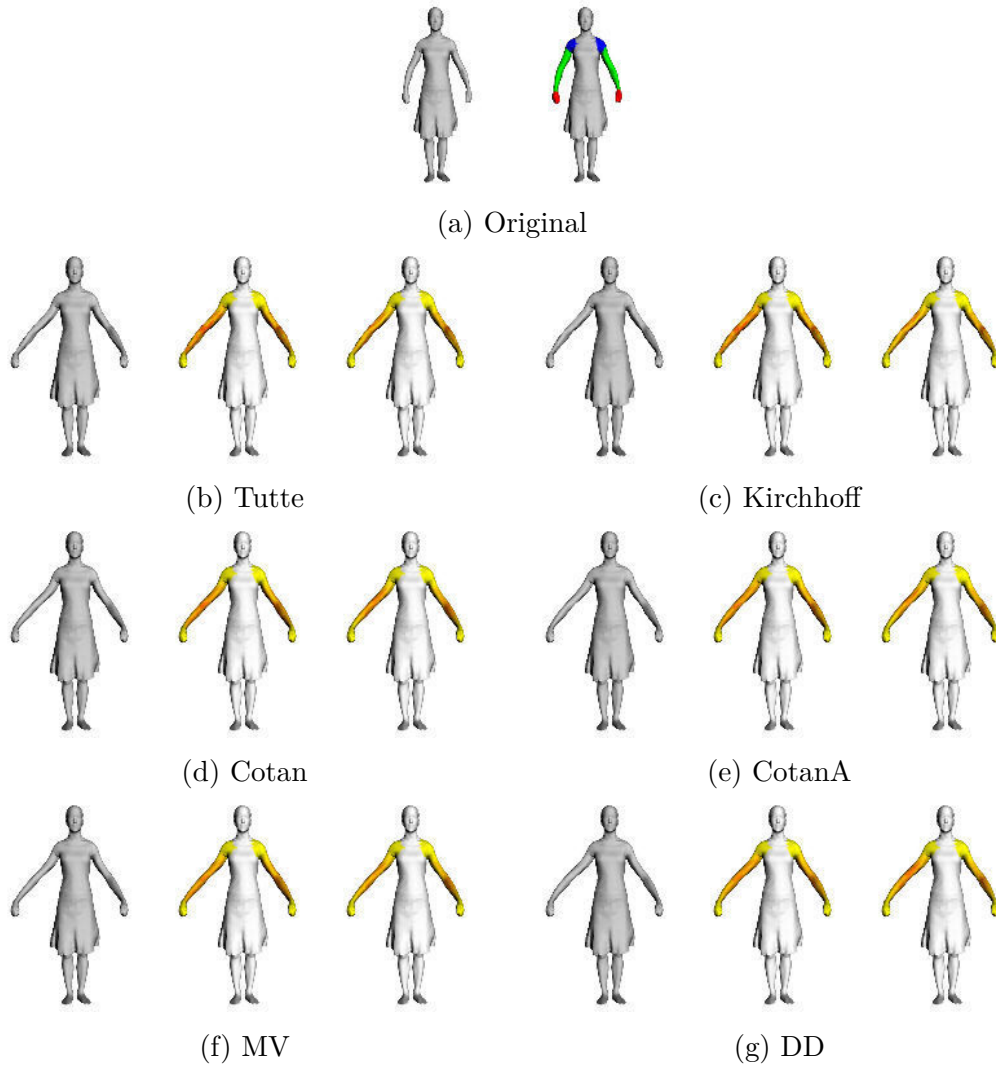


Figure 6.16: Laplacian surface editing applied on the first frame of the Samba dataset [49]. From left to right for each discrete Laplace operator: result of the deformation, angle difference, area difference.

Table 6.9: Measured angle difference in Laplacian surface editing of the Lion model

	MV	Cotan	CotanA	Tutte	Kirchhoff	DD
Sum	1323,380	1348,490	1456,877	1561,659	1626,670	1302,569
Max	0,644	0,705	0,677	1,515	1,491	0,718

Table 6.10: Measured area difference in Laplacian surface editing of the Lion model

	MV	Cotan	CotanA	Tutte	Kirchhoff	DD
Sum	6,63E-02	6,86E-02	6,70E-02	8,06E-02	8,05E-02	6,82E-02
Max	1,24E-04	1,62E-04	1,29E-04	2,49E-04	2,44E-04	1,48E-04

In this case, the performance of combinatorial Laplacians was significantly worse. The results of such Laplacians contained slight visual artifacts. Even the angle and area differences are worse. Geometric Laplacians resulted in visually smoother deformation. The differences in each of the geometric discrete Laplace operator are almost imperceptible. The Data Dependent Laplacian obtained satisfying result, and even had the lowest sum of angle differences.

This section has shown, that the Data Dependent Laplacian is surely suitable for Laplacian surface editing technique. However, the conditioning issue may still occur, even though it did not occur in the experiments.

6.2.5 Mesh morphing

Even though the Data Dependent Laplacian can be classified as geometric, it can be used in the mesh morphing. Only adjustment to be made is to calculate the weights using positions of both meshes, just like it would be calculated for a dynamic mesh with only two frames. The resulting discrete Laplace operator should contain information of both input meshes.

Two tests were done in this experiment. In the first one, a mesh morphing technique proposed by Alexa [1] was tested against two frames of the Samba dataset (See figure 6.17a). Compared interpolations were:

- Linear interpolation of Tutte Laplacian δ coordinates (Naïve Tutte)
- Linear interpolation of Kirchhoff Laplacian δ coordinates (Naïve Kirchhoff)
- Linear interpolation of Data Dependent Laplacian δ coordinates (Naïve DD)
- Proposed interpolation of Tutte Laplacian δ coordinates (Tutte)
- Proposed interpolation of Kirchhoff Laplacian δ coordinates (Kirchhoff)

- Proposed interpolation of Data Dependent Laplacian δ coordinates (DD)

In this case, no metric was applied, because there was no frame that would be similar to both input meshes that could be used for angle and area difference measurement. The results can be seen in Figure 6.17.

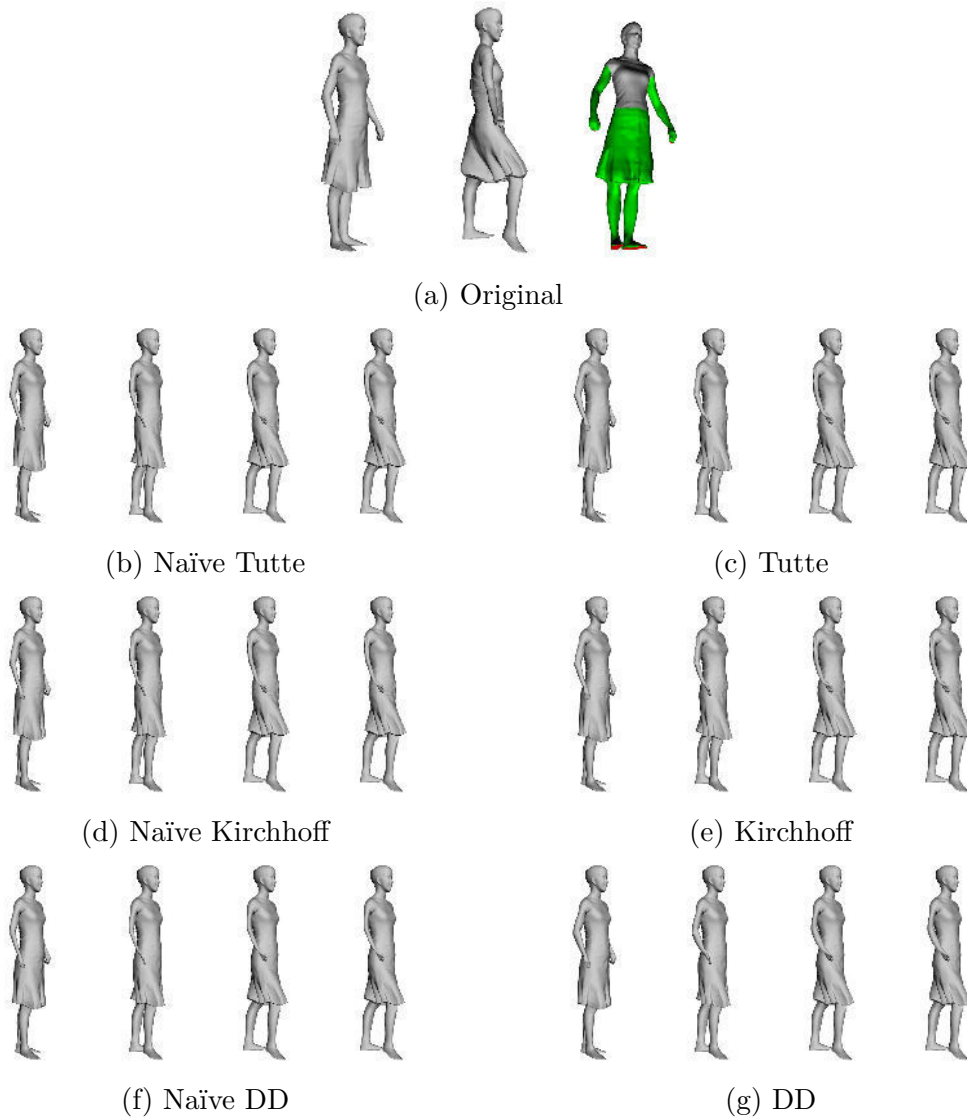


Figure 6.17: Morphing of parts of two frames from the Samba dataset [49]

All of the approaches using linear interpolation of the differential coordinates result in the distortion of the mesh. This effect is most visible on the hands that clearly shrank. The improved interpolations using combinatorial Laplacians did not result in such shrinkage. The result, however,

contains visible artifacts. The best result was obtained using the Data Dependent Laplacian, which shrank the result the least, and obtained the most smooth surface formed by unconstrained vertices. This, however, does not mean, that the artifacts might never appear. Again, this approach requires a linear system to be solved, thus some conditioning issues might occur.

In the second test, two frames of the Squat dataset [49] were processed by morphing technique working with the whole meshes. To measure the quality of the morphing approach, the results were compared with actual frames of the sequence that corresponded to to each used morphing parameter. In this case, four different interpolations were tested:

- Linear interpolation of Tutte Laplacian δ coordinates (Naïve)
- Proposed interpolation of Tutte Laplacian δ coordinates (Tutte)
- Proposed interpolation of Kirchhoff Laplacian δ coordinates (Kirchhoff)
- Proposed interpolation of Data Dependent Laplacian δ coordinates (DD)

The Linear interpolation was tested only for Tutte, as it was expected that the results would be the same for any other tested Laplacian. All of the interpolations used 10 vertices as anchors.

Table 6.11: Sum of angle differences between expected frame of Squat dataset and the whole mesh morphing result in each morphing step.

t	Naïve	Tutte	Kirchhoff	DD
0,25	1945,339	2140,677	2137,548	2126,406
0,5	2828,021	3079,763	3079,182	3038,582
0,75	2238,963	2495,174	2489,948	2450,931

Table 6.12: Sum of area differences between expected frame of Squat dataset and the whole mesh morphing result in each morphing step.

t	Naïve	Tutte	Kirchhoff	DD
0,25	2,37E-01	1,49E-01	1,49E-01	1,55E-01
0,5	3,26E-01	2,09E-01	2,08E-01	2,18E-01
0,75	2,46E-01	1,62E-01	1,61E-01	1,64E-01

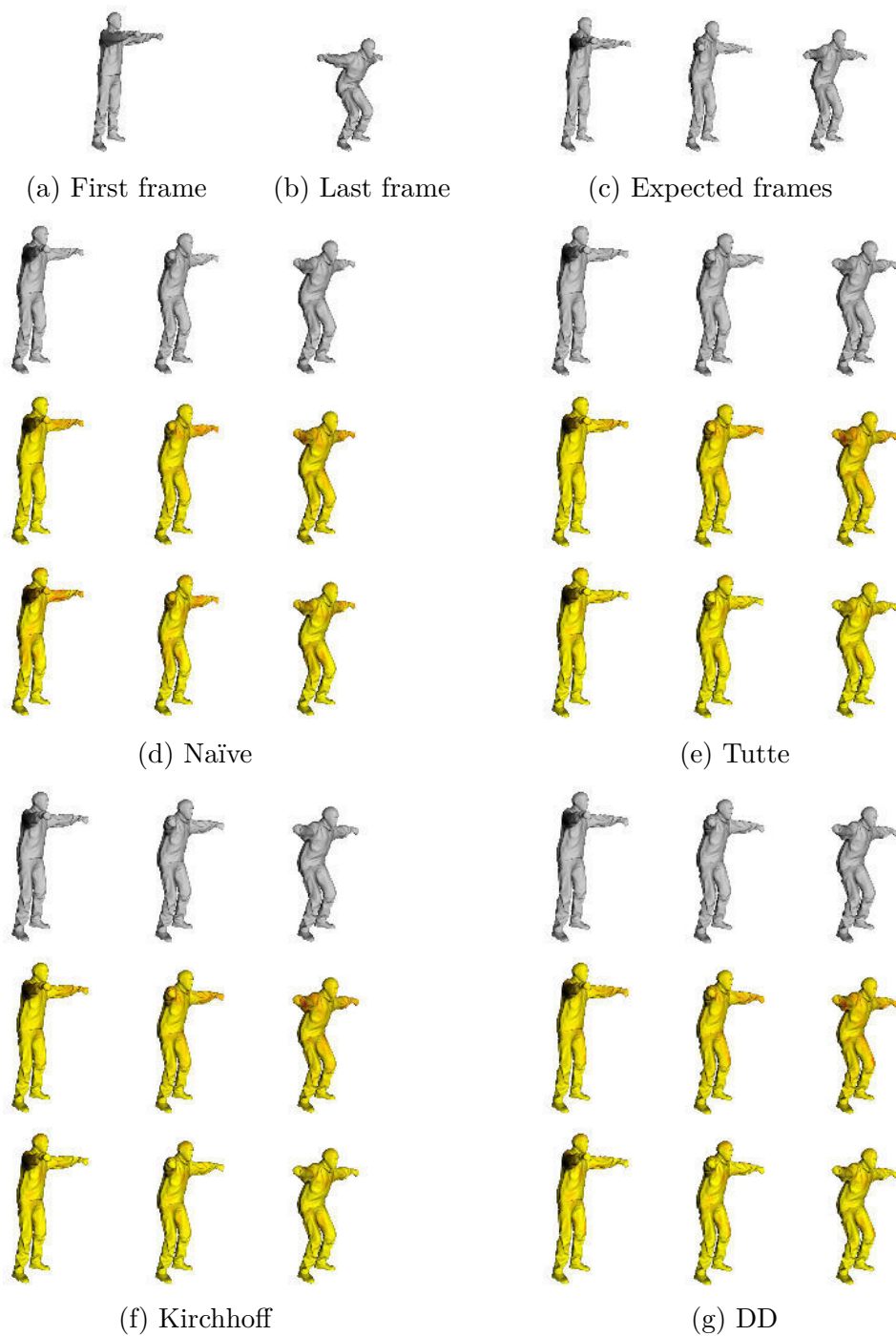


Figure 6.18: Parameterization of the first modified model from the Samba dataset [49]

Although the naïve approach resulted in the lowest angle difference, some issues are visible. For example, the hands of the model were shrunk. The shrinkage can be seen even in the Table 6.12, where this approach obtained

the highest area difference. The Kirchhoff and Tutte Laplacians, on the other hand, resulted in higher angle difference. The compromise between the two approaches would be the Data Dependent Laplacian, which is close to the best result in both metrics. Even visually, its result is more appealing. However, there were visual artifacts on the back of the right leg (See Figure 6.19). Again, in this technique, linear system is solved in the least-squares sense, which means, that this should be a matrix conditioning issue.



Figure 6.19: Artifacts on the right leg occurring during the whole mesh morphing with Data Dependent Laplacian. Colour visualizes the angle difference from the expected result.

From all the results of this experiment, it can be concluded, that the discrete Data Dependent Laplace operator is indeed very suitable for mesh morphing, as it combines abilities of both geometric and combinatorial Laplacians. It contains the geometric information, while allowing two meshes to have the same Laplacian matrix. However, the matrix conditioning is still a big issue.

6.2.6 Least-squares meshes

The results of this technique were evaluated using two tests. In both tests, the angle and area difference between the result and the input mesh. In the first test, reconstruction of the geometry of a hole was simulated on the Lion model. First, the Laplacian was calculated, Then, all of the vertices of the mesh except the ones forming a nose (denoted by blue color in Figure 6.20a) were selected as control points, the geometry of the rest was thrown away. Finally, using the Least-squares meshes technique, the geometry was again reconstructed. The results can be seen in Figure 6.20, the angle difference in Table 6.13 and area difference in Table 6.14.

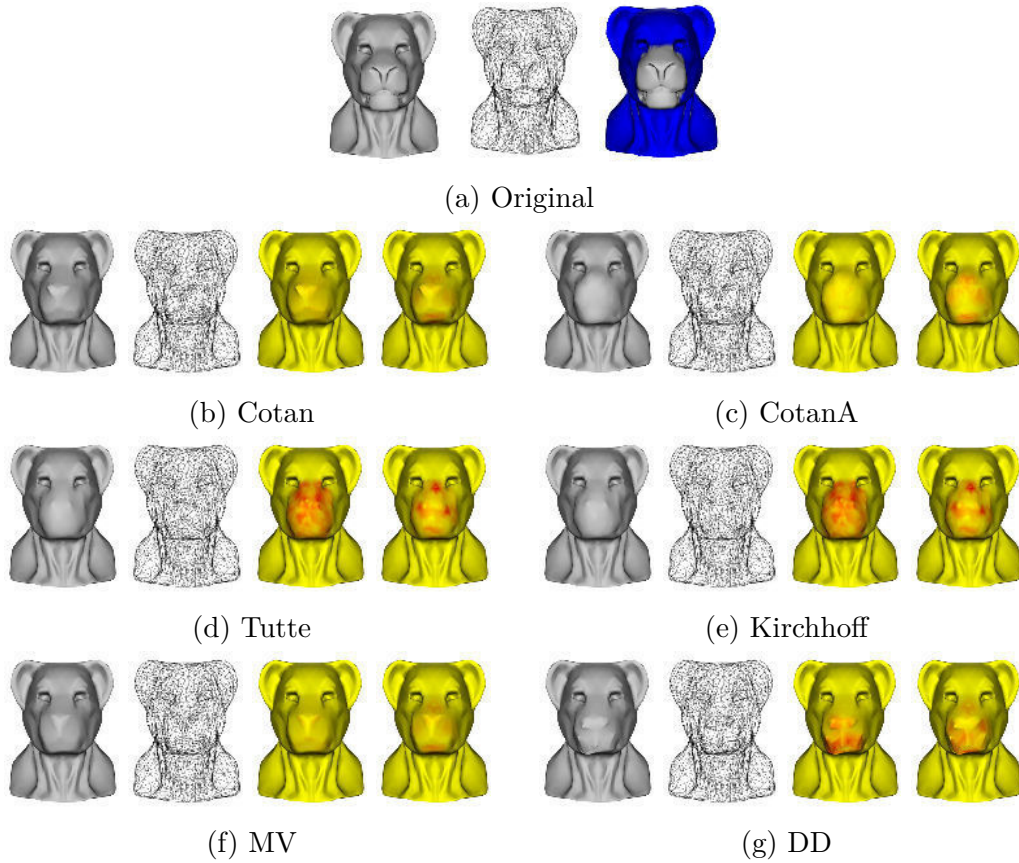


Figure 6.20: Hole geometry reconstruction of the Lion model using Least-squares meshes technique [35]

Table 6.13: Measured angle difference of Least-Square mesh of the Lion model

	MV	Cotan	CotanA	Tutte	Kirchhoff	DD
Sum	151,657	181,009	177,698	781,022	794,418	488,597
Max	0,499	0,543	0,550	1,918	1,929	1,573
Avg	0,012	0,014	0,014	0,062	0,063	0,039

Table 6.14: Measured area difference in Laplacian surface editing of the Lion model

	MV	Cotan	CotanA	Tutte	Kirchhoff	DD
Sum	5,83E-01	5,72E-01	7,08E-01	1,16E+00	1,17E+00	9,26E-01
Max	6,53E-03	7,20E-03	6,76E-03	1,79E-02	1,78E-02	9,96E-03
Avg	1,39E-04	1,36E-04	1,69E-04	2,78E-04	2,80E-04	2,21E-04

The highest angle and area difference was obtained using combinatorial Laplacians. This does not mean, that these Laplacians are not suitable for least-square meshes. It only pointed out that tangential drift occurred. Also, when no geometry is provided for the hole in the object before calculating the Laplacian weights, they are the only option, as they require only the connectivity information. When inspecting the visual results, the worst result is clearly achieved by Data Dependent Laplacian. There are quite visible artifacts. Again, this seems to be caused by the matrix conditioning issue.

The second test was done to simulate shape approximation. A set of 1000 random vertices on the surface of the first frame of the Samba dataset was selected as control points (selected vertices can be seen in Figure 6.21a), and position of the rest of the vertices were calculated using this technique. The results are shown in Figure 6.21 and Tables 6.15 and 6.16.

Table 6.15: Measured angle difference of Least-Square mesh of the first frame of the Samba dataset [49]

	MV	Cotan	CotanA	Tutte	Kirchhoff	DD
Sum	1890,504	2352,444	2630,000	18551,416	18791,218	2306,554
Max	1,197	1,487	1,697	2,221	2,168	1,886
Avg	0,032	0,039	0,044	0,310	0,314	0,039

Table 6.16: Measured area difference of Least-Square mesh of the first frame of the Samba dataset [49]

	MV	Cotan	CotanA	Tutte	Kirchhoff	DD
Sum	7,95E-02	9,83E-02	1,24E-01	6,57E-01	6,62E-01	1,02E-01
Max	1,17E-04	3,46E-04	1,01E-03	7,96E-04	8,00E-04	3,90E-04
Avg	3,99E-06	4,93E-06	6,20E-06	3,30E-05	3,32E-05	5,14E-06

Again, the combinatorial Laplacians resulted in the tangential drift. Inspecting the area and angle differences, the best result seems to be obtained using Mean Value Laplacian. In the case of the Data Dependent Laplacian, there are visible artifacts on the feet of the model, again, probably caused by conditioning issues.

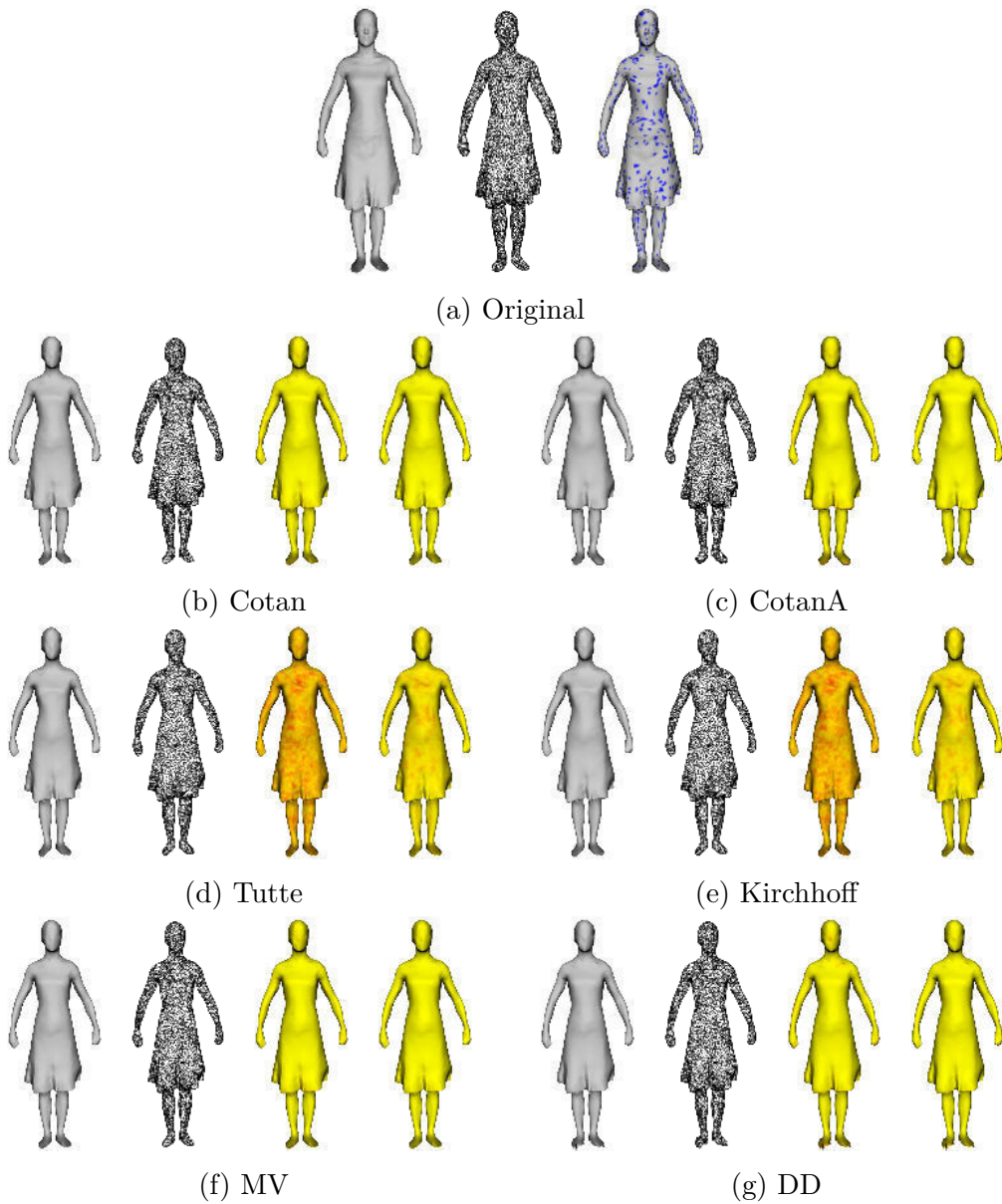


Figure 6.21: Shape approximation of the first frame of the Samba dataset[49] using Least-squares meshes technique [35] (1000 control points)

The both tests have shown, that the Data Dependent Laplacian is not very suitable for least-squares meshes technique. It seems like it performs quite badly in all the techniques, where it is desired to obtain an as smooth as possible result, as it had many issues even during experiments with smoothing.

6.3 Dynamic mesh compression

The method proposed by Váša et al. [44] was modified to use the discrete Data Dependent Laplace operator. In this section, the compression results will be discussed. The experiments were done directly in the compression framework used in [44], which was written in C# programming language. The compression framework is not part of the attached DVD as it is not publicly available. The results were primarily compared to the results obtained using original method with Mean value Laplacian, that Váša et al. pointed out to work the best.

6.3.1 Entropy

First experiment was made to ensure that the assumption about decrease of entropy with more accurate prediction is true. Each coordinate of the trajectory space has its own distribution, which means that the entropy must be based on probability of the corresponding coordinate distributions, not on probability across all values. The measured value can be calculated from following:

$$H = \sum_{i=1}^B H_i = \sum_{i=1}^B \sum_{j=1}^{N_i} p_{ij} \log_2(p_{ij}),$$

where N_i is number of quantized values with non-zero probability in i -th coordinate and p_{ij} is probability of j -th value in the set of all quantized values with non-zero probability in i -th coordinate.

Both methods rely on input parameters: *Reduced dimension of trajectories*, *number of bits per quantization level of trajectory basis* and *number of bits per quantization level of residuals*. The configurations of these parameters will be referred to in this experiment as a triplet of values ($dim, basisQ, residualQ$). Experiment was done on dataset *Samba* [49]. Results can be seen in Table 6.17.

Table 6.17: Measured residual entropy of compared compression methods for *Samba* dataset

Configuration	MV	DD
(55, 18, 4)	78,70201	72,03983
(70, 18, 5)	135,53843	125,93480
(125, 20, 7)	361,55749	342,36423

The experiment has shown that the proposed discrete Laplace operator truly lowers the entropy of encoded residuals.

6.3.2 Rate-Distortion curve

As a part of the compression framework, there is optimization algorithm, that estimates input parameters of the method for specified target data rate. For each step, the optimization algorithm outputs current bitrate (in bits per frame vertex - *bpfv*) and data distortion measured in the *STED* error metric [48]. Visualizing the given data results in so-called Rate-Distortion curve. This curve is very important for comparison of compression methods, as it shows trends that cannot be simply seen from measured data.

For this experiment, the optimization algorithm was simply executed for 5 *bpfv* target bitrate. The measured R-D curves are shown only for bitrates under 3 *bpfv* as that way the results are more convenient. Tests were made again for *Samba* dataset. Results are shown in Figure 6.22.

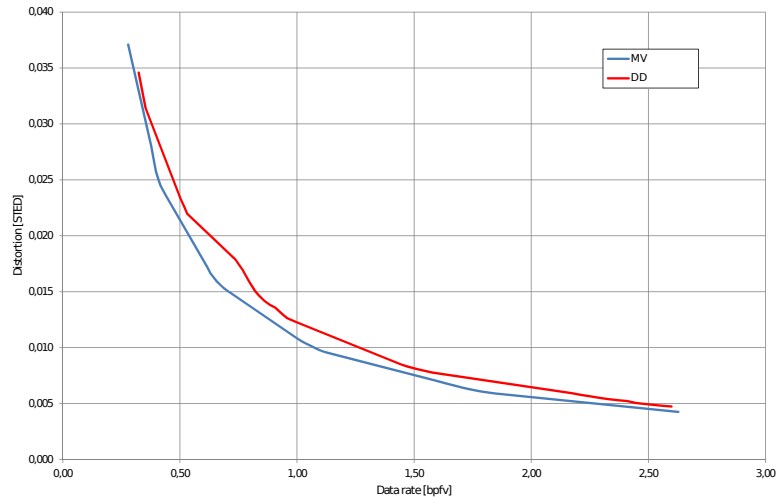


Figure 6.22: Comparison of *R-D* curves of tested compression methods

The results are surprising, as for the same parameter configuration, the *Data Dependent* Laplacian outputs lower bitrate, as can be seen in Table 6.18. However, with lower bitrate, the *Data Dependent* Laplacian outputs slightly higher distortion. If instead of comparing same configurations we compare same bitrate, the *Mean value* Laplacian achieves slightly better results.

Table 6.18: Measured bitrate in bits-per-frame-vertex of compared compression methods for *Samba* dataset

Configuration	MV	DD
(55, 18, 4)	0,63132	0,57736
(70, 18, 5)	0,95384	0,88490
(125, 20, 7)	2,32923	2,21431

6.3.3 Normal matrix conditioning

The distortion of the data highly depends on the condition number of so-called *normal matrix* \mathbf{N} which is used for geometry reconstruction. Such matrix can be calculated as:

$$\mathbf{N} = (\bar{\mathbf{L}}^*)^T \bar{\mathbf{L}}^*,$$

where $\bar{\mathbf{L}}^*$ is Laplacian matrix extended by one row with single value 1 in it for each anchor point. The higher the condition number of \mathbf{N} is, the more the data is distorted. The condition number is

$$\kappa(\mathbf{N}) = \frac{\sigma_{max}(\mathbf{N})}{\sigma_{min}(\mathbf{N})},$$

where $\sigma_{max}(\mathbf{N})$ and $\sigma_{min}(\mathbf{N})$ are the maximal and minimal singular values of \mathbf{N} . Instead of performing Singular value decomposition of \mathbf{N} , simple experiment can be performed to measure how much error is introduced by solving the least squares linear system represented by such matrix.

Let \mathbf{x} be a vector of random values $x_i \in \langle -1, 1 \rangle$ of size V . By multiplying given vector with \mathbf{N} vector \mathbf{b} is obtained:

$$\mathbf{b} = \mathbf{N}\mathbf{x}.$$

If a vector \mathbf{c} of small random values $c_i \in \langle -k, k \rangle, k > 0 \wedge k \ll 1$ is added to \mathbf{b} , the result is a vector \mathbf{b}' . Then, a linear system is solved:

$$\mathbf{N}\mathbf{x}' = \mathbf{b}'.$$

Finally, a metric for matrix conditioning can be calculated:

$$\Delta = \|\mathbf{x} - \mathbf{x}'\|.$$

Given metric actually simulates the process of geometry encoding and decoding, where the vector \mathbf{b}' represents the residuals with introduced quantization error.

Table 6.19: Measured Δ of compared compression methods for *Samba* dataset

Configuration	MV	Tutte	DD
(55, 18, 4)	847,16491	727,49836	1592,98270
(70, 18, 5)	847,16491	727,49836	1665,44730
(125, 20, 7)	847,16491	727,49836	1537,78865

For this experiment, Δ was measured on Mean value, Tutte and Data Dependent Laplacian. The results are shown in Table 6.19.

The results have proven the hypothesis about worse conditioning of the normal matrix for Data Dependent Laplacian. Although it seems conditionality of MV and Tutte Laplacian is static, it is not true. However, the values differ so little, that it cannot be seen, when the numbers are rounded to 5 decimal digits.

6.3.4 Asymmetric Data Dependent Laplacian

Only $2V$ degrees of freedom to define a Laplacian matrix might be too constraining. By using asymmetric Laplacian matrix, a further decrease of the entropy of δ -trajectories can be expected, as the weights generated from such least squares solution can produce a better prediction. Another advantage of asymmetric Laplacian is simpler weight dependency handling. With symmetric Laplacian, each weight w_{ij} occurs in all equations corresponding to vertices v_i and v_j . If one weight is selected as dependent, it must be substituted in all occurrences, even in substitutions of already substituted dependent weights, which can lead to significant increase of complexity of the substitution process. In the case of asymmetric Laplacian, each weight w_{ij} occurs only in the equations corresponding to vertex v_i .

Problem is, that by forcing weight dependences, the number of unknowns is reduced only by $1V$. This means that the number of weights necessary to be encoded is $5V$. To encode such amount of data can be advantageous, only if the entropy of δ -trajectories decreases enough. Simple comparison of R-D curves between symmetric and asymmetric Laplacian was made on the Samba dataset [49]. The results are shown in Figure 6.23.

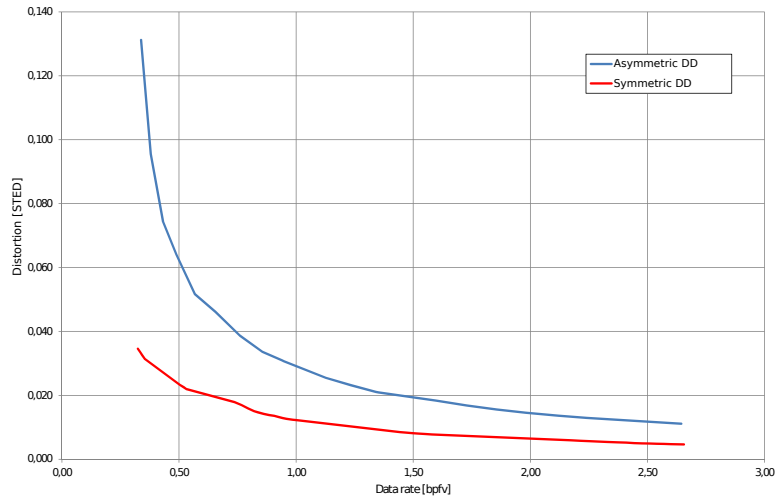


Figure 6.23: Comparison of R - D curves of symmetric and asymmetric Data Dependent Laplacian

As R - D curves have shown, the asymmetric Data Dependent Laplacian did not introduce any improvement over the symmetric variant. In fact, it performs far worse. The reason can be seen in Table 6.20 - for its normal matrix, the Δ is significantly bigger than for the normal matrix of the symmetric Data Dependent Laplacian. In fact, it can be considered ill-conditioned.

Table 6.20: Measured Δ of symmetric and asymmetric Data Dependent Laplacian

Configuration	Symmetric DD	Asymmetric DD
(55, 18, 4)	1592,98270	606640,65607
(70, 18, 5)	1665,44730	579801,88392
(125, 20, 7)	1537,78865	721965,44510

7 Conclusion

In this thesis, a new discretization of Laplace operator, called Data Dependent Laplacian, was proposed. The results have shown, that the assumption that minimizing the lengths of differential coordinates leads to residual entropy reduction, was correct. It was, however, also found, that entropy reduction is not the only concern in the case of mesh compression. It is also important to reduce, or at least preserve, a condition number of the Laplacian matrix, as the conditioning directly affects the amount of distortion of data. It was also shown, that an asymmetric variant of such Laplace operator has even worse conditioning, while requiring even more data to be transmitted.

In the case of static mesh processing, the results have shown, that there are some possible applications of such operator. The conditioning issue is also present, however, it does not occur all the time. In the case of surface editing, the Data Dependent Laplacian introduces quite interesting detail preservation. However, the best improvement this Laplacian introduces is in the mesh morphing, where it behaves like a geometric Laplacian, while allowing to construct a single Laplacian matrix from geometry data of two triangle meshes without any complex shape analysis.

In the future, it would be interesting to investigate application of such Laplacian in dynamic mesh smoothing and parameterization, two mesh processing problems which were unfortunately not investigated in this thesis due to the large scope of all the other experiments. Another thing yet to be done is to address further the conditioning issue of the Laplacian matrix.

Bibliography

- [1] ALEXA, M. Differential coordinates for local mesh morphing and deformation. *The Visual Computer*. 2003, 19, 2, pages 105–114.
- [2] ALEXA, M. – KYPRIANIDIS, J. E. Error diffusion on meshes. *Computers & Graphics*. 2015, 46, pages 336–344.
- [3] ALEXA, M. – MÜLLER, W. Representing animations by principal components. In *Computer Graphics Forum*, 19, pages 411–418. Wiley Online Library, 2000.
- [4] ALLIEZ, P. – DESBRUN, M. Valence-Driven Connectivity Encoding for 3D Meshes. In *Computer graphics forum*, 20, pages 480–489. Wiley Online Library, 2001.
- [5] ALLIEZ, P. et al. Isotropic surface remeshing. In *Shape Modeling International, 2003*, pages 49–58. IEEE, 2003.
- [6] BIYIKOGLU, T. – LEYDOLD, J. – STADLER, P. F. Laplacian eigenvectors of graphs. *Lecture notes in mathematics*. 2007, 1915.
- [7] BOTSCH, M. – SORKINE, O. On linear variational surface deformation methods. *IEEE transactions on visualization and computer graphics*. 2008, 14, 1, pages 213–230.
- [8] BOTSCH, M. et al. Geometric modeling based on triangle meshes. In *ACM SIGGRAPH 2006 Courses*, pages 1. ACM, 2006.
- [9] BOTSCH, M. et al. *Polygon mesh processing*. CRC press, 2010.
- [10] DAVIS, T. A. – RAJAMANICKAM, S. – SID-LAKHDAR, W. M. A survey of direct methods for sparse linear systems. *Acta Numerica*. 2016, 25, pages 383–566.
- [11] DESBRUN, M. 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*, pages 317–324. ACM Press/Addison-Wesley Publishing Co., 1999.
- [12] DONG, S. et al. Spectral surface quadrangulation. In *ACM Transactions on Graphics (TOG)*, 25, pages 1057–1066. ACM, 2006.
- [13] FIEDLER, M. Algebraic connectivity of graphs. *Czechoslovak mathematical journal*. 1973, 23, 2, pages 298–305.

- [14] FLOATER, M. S. Parametrization and smooth approximation of surface triangulations. *Computer aided geometric design*. 1997, 14, 3, pages 231–250.
- [15] FLOATER, M. S. Mean value coordinates. *Computer aided geometric design*. 2003, 20, 1, pages 19–27.
- [16] GUENNEBAUD, G. – JACOB, B. – OTHERS. Eigen v3. <http://eigen.tuxfamily.org>, 2010.
- [17] HORMANN, K. – LÉVY, B. – SHEFFER, A. Mesh parameterization: Theory and practice. 2007.
- [18] KARNI, Z. – GOTSMAN, C. Compression of soft-body animation sequences. *Computers & Graphics*. 2004, 28, 1, pages 25–34.
- [19] KIRBY, E. C. et al. What Kirchhoff actually did concerning spanning trees in electrical networks and its relationship to modern graph-theoretical work. *Croatica Chemica Acta*. 2016, 89, 4, pages 403–417.
- [20] LIPMAN, Y. et al. Differential coordinates for interactive mesh editing. In *Shape Modeling Applications, 2004. Proceedings*, pages 181–190. IEEE, 2004.
- [21] LOBAZ, P. – VÁŠA, L. Hierarchical Laplacian-based compression of triangle meshes. *Graphical Models*. 2014, 76, 6, pages 682–690.
- [22] MALÍK, Z. Image-driven simplifikace trojúhelníkových sítí s použitím percepční metriky. 2012.
- [23] MEYER, M. et al. Discrete differential-geometry operators for triangulated 2-manifolds. *Visualization and mathematics III*. 2003, pages 35–57.
- [24] MOHAR, B. et al. The Laplacian spectrum of graphs. *Graph theory, combinatorics, and applications*. 1991, 2, 871-898, pages 12.
- [25] PARUS, J. Morphing of meshes: technical report no. DCSE/TR-2005-02. 2005.
- [26] PINKALL, U. – POLTHIER, K. Computing discrete minimal surfaces and their conjugates. *Experimental mathematics*. 1993, 2, 1, pages 15–36.
- [27] RAZDAN, A. – BAE, M. Curvature estimation scheme for triangle meshes using biquadratic Bézier patches. *Computer-Aided Design*. 2005, 37, 14, pages 1481–1491.

- [28] RODRIGUES, O. Des lois geometriques qui regissent les déplacements d'un systeme solide dans l'espace et de la variation des coordonnees provenant de déplacements consideres independamment des causes qui peuvent les produire. *J Mathematiques Pures Appliquees*. 1840, 5, pages 380–440.
- [29] ROSSIGNAC, J. Edgebreaker: Connectivity compression for triangle meshes. *IEEE transactions on visualization and computer graphics*. 1999, 5, 1, pages 47–61.
- [30] RUSINKIEWICZ, S. Estimating curvatures and their derivatives on triangle meshes. In *3D Data Processing, Visualization and Transmission, 2004. 3DPVT 2004. Proceedings. 2nd International Symposium on*, pages 486–493. IEEE, 2004.
- [31] SAWHNEY, R. – CRANE, K. Boundary First Flattening. *ACM Transactions on Graphics (TOG)*. 2017, 37, 1, pages 5.
- [32] SEDERBERG, T. W. – PARRY, S. R. Free-form deformation of solid geometric models. *ACM SIGGRAPH computer graphics*. 1986, 20, 4, pages 151–160.
- [33] SHEFFER, A. – STURLER, E. Parameterization of faceted surfaces for meshing using angle-based flattening. *Engineering with computers*. 2001, 17, 3, pages 326–337.
- [34] SORKINE, O. Laplacian mesh processing. In *Eurographics (STARs)*, pages 53–70, 2005.
- [35] SORKINE, O. – COHEN-OR, D. Least-squares meshes. In *Shape Modeling Applications, 2004. Proceedings*, pages 191–199. IEEE, 2004.
- [36] SORKINE, O. – COHEN-OR, D. – TOLEDO, S. High-Pass Quantization for Mesh Encoding. In *Symposium on Geometry Processing*, 42, 2003.
- [37] SORKINE, O. et al. Laplacian surface editing. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 175–184. ACM, 2004.
- [38] SUN, Y. M. – WANG, W. – CHIN, F. Y. Interpolating polyhedral models using intrinsic shape parameters. *The Journal of Visualization and Computer Animation*. 1997, 8, 2, pages 81–96.
- [39] TAUBIN, G. A signal processing approach to fair surface design. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 351–358. ACM, 1995.

- [40] TOUMA, C. – GOTSMAN, C. Triangle mesh compression. *Proceedings - Graphics Interface*. 1998, pages 26–34.
- [41] TUTTE, W. T. How to draw a graph. *Proceedings of the London Mathematical Society*. 1963, 3, 1, pages 743–767.
- [42] VÁŠA, L. et al. Mesh Statistics for Robust Curvature Estimation. *Computer Graphics Forum*. 2016. ISSN 1467-8659. doi: 10.1111/cgf.12982.
- [43] VALLET, B. – LÉVY, B. Spectral geometry processing with manifold harmonics. In *Computer Graphics Forum*, 27, pages 251–260. Wiley Online Library, 2008.
- [44] VÁŠA, L. et al. Compressing dynamic meshes with geometric laplacians. In *Computer Graphics Forum*, 33, pages 145–154. Wiley Online Library, 2014.
- [45] VÁŠA, L. – PETŘÍK, O. Optimising perceived distortion in lossy encoding of dynamic meshes. In *Computer Graphics Forum*, 30, pages 1439–1449. Wiley Online Library, 2011.
- [46] VÁŠA, L. – RUS, J. Dihedral angle mesh error: a fast perception correlated distortion measure for fixed connectivity triangle meshes. In *Computer Graphics Forum*, 31, pages 1715–1724. Wiley Online Library, 2012.
- [47] VÁŠA, L. – SKALA, V. Coddycac: Connectivity driven dynamic mesh compression. In *3DTV Conference, 2007*, pages 1–4. IEEE, 2007.
- [48] VÁŠA, L. – SKALA, V. A perception correlated comparison method for dynamic meshes. *IEEE transactions on visualization and computer graphics*. 2011, 17, 2, pages 220–230.
- [49] VLASIC, D. et al. Articulated mesh animation from multi-view silhouettes. In *ACM Transactions on Graphics (TOG)*, 27, pages 97. ACM, 2008.
- [50] WARDETZKY, M. et al. Discrete Laplace operators: no free lunch. In *Symposium on Geometry processing*, pages 33–37, 2007.
- [51] WELCH, W. – WITKIN, A. Variational surface modeling. In *ACM SIGGRAPH computer graphics*, 26, pages 157–166. ACM, 1992.
- [52] WINKLER, T. et al. Multi-Scale Geometry Interpolation. In *Computer graphics forum*, 29, pages 309–318. Wiley Online Library, 2010.
- [53] YILDIZ, Z. C. – CAPIN, T. A perceptual quality metric for dynamic triangle meshes. *EURASIP Journal on Image and Video Processing*. 2017, 2017, 1, pages 12.

- [54] ZHANG, H. – VAN KAICK, O. – DYER, R. Spectral mesh processing. In *Computer graphics forum*, 29, pages 1865–1894. Wiley Online Library, 2010.