Volume 25 (2006), number 4 pp. 789–807

COMPUTER GRAPHICS *forum*

Differential Representations for Mesh Processing

Olga Sorkine

School of Computer Science, Tel Aviv University, Israel
sorkine@gmail.com

Abstract

Surface representation and processing is one of the key topics in computer graphics and geometric modeling, since it greatly affects the range of possible applications. In this paper we will present recent advances in geometry processing that are related to the Laplacian processing framework and differential representations. This framework is based on linear operators defined on polygonal meshes, and furnishes a variety of processing applications, such as shape approximation and compact representation, mesh editing, watermarking and morphing. The core of the framework is the definition of differential coordinates and new bases for efficient mesh geometry representation, based on the mesh Laplacian operator.

Keywords: discrete Laplacian, surface representation, detail preservation, geometry compression, mesh editing

ACM CCS: I.3.5 Computer Graphics: *Computational Geometry and Object Modeling — Boundary representations*

1. Introduction

Surface representation and processing is one of the key topics in computer graphics, geometric modeling and computer-aided design. To put it simply, the way a 3D object is defined greatly affects the range of things one can do with it. Different object representations may reveal different geometric, combinatorial or even perceptive properties of the shape at hand. For instance, the very popular piecewise-linear surface representation (triangular meshes) provides immediate means to display the surface, learn about its topological properties (through Euler's characteristic) and, with some effort, differential properties of the smooth surface it approximates. On the other hand, many modeling tasks are difficult to perform on meshes. Moreover, nowadays the common scanned surfaces usually come as very detailed, complex and at times noisy meshes that require geometry processing, such as filtering, resampling and compression for efficient storage and streaming.

In this paper, we describe recent work on mesh processing and modeling that is based on the Laplacian framework and differential representations. In this framework, surface meshes are studied through their differential properties, derived from certain linear operators defined on the mesh. These operators are usually different variants of the mesh Laplacian, and they provide means to represent a surface by new bases that benefit various processing applications. In contrast to the traditional global Cartesian coordinates, which can only tell the spatial location of each point, a differential surface representation carries information about the local shape of the surface, the size and orientation of local details. Therefore, defining operations on surfaces that strive to preserve such a differential representation, results in detail-preserving operations. The linearity of the processing framework makes it very efficient, and it has become an attractive and promising research direction, as is evident from the amount of recent publications on the subject.

In the following section we will closely look into the Laplacian operator, the differential representations and the associated surface reconstruction problems. We will then review some results and applications that benefit from this framework, such as shape approximation, geometry compression and watermarking (Section 3) and interactive mesh editing and interpolation(Section 4).

Submitted September 2005
Revised April 2006
Accepted May 2006

2. Laplacian Operator and Differential Surface Representation

In the following, we review the definition of differential coordinates (δ-coordinates), the associated mesh Laplacian operator, describe the surface reconstruction framework and discuss various properties of the Laplacian operator.

2.1. Basic definitions

Let $\mathcal{M} = (V, E, F)$ be a given triangular mesh with n vertices. V denotes the set of vertices, E denotes the set of edges and F denotes the set of faces. Each vertex $i \in \mathcal{M}$ is conventionally represented using absolute Cartesian coordinates, denoted by $\mathbf{v}_i = (x_i, y_i, z_i)$. We first define the *differential* or *δ-coordinates* of $\mathbf{v}_i$ to be the difference between the absolute coordinates of $\mathbf{v}_i$ and the center of mass of its immediate neighbors in the mesh,

$$\delta_i = \left(\delta_i^{(x)}, \delta_i^{(y)}, \delta_i^{(z)}\right) = \mathbf{v}_i - \frac{1}{d_i} \sum_{j \in N(i)} \mathbf{v}_j$$

where $N(i) = \{j \,|(i, j) \in E\}$ and $d_i = |N(i)|$ is the number of immediate neighbors of i (the degree or valence of i).

The transformation of the vector of absolute Cartesian coordinates to the vector of δ-coordinates can be represented in a matrix form. Let A be the adjacency (connectivity) matrix of the mesh

$$A_{ij} = \begin{cases} 1 & (i, j) \in E \\ 0 & \text{otherwise} \end{cases}$$

and let D be the diagonal matrix such that $D_{ii} = d_i$. Then the matrix transforming the absolute coordinates to relative coordinates is

$$L = I - D^{-1}A.$$

It is often more convenient to consider the symmetric version of the L matrix, defined by $L_s = DL = D - A$,

$$(L_s)_{ij} = \begin{cases} d_i & i = j \\ -1 & (i, j) \in E \\ 0 & \text{otherwise} \end{cases}$$

That is, $L_s\mathbf{x} = D\delta^{(x)}$, $L_s\,\mathbf{y} = D\delta^{(y)}$ and $L_s\mathbf{z} = D\delta^{(z)}$, where $\mathbf{x}$ is an n-vector containing the x absolute coordinates of all the vertices and so on. See Figure 3 for an example of a mesh and its associated matrices.

The matrix L_s (or L) is called the topological (or graph) Laplacian of the mesh [1]. Graph Laplacians have been extensively studied in algebra and graph theory [2], primarily because their algebraic properties are related to the combinatorial properties of the graphs they represent. We will look into some of these properties later on, in Sections 2.2 and 3.

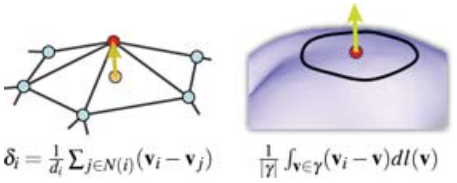

Figure 1: *The vector of the differential coordinates at a vertex approximates the local shape characteristics of the surface: the normal direction and the mean curvature.*

From a differential geometry perspective, the δ-coordinates can be viewed as a discretization of the continuous Laplace–Beltrami operator [3], if we assume that our mesh $\mathcal{M}$ is a piecewise-linear approximation of a smooth surface. We can write the differential coordinate vector at vertex $\mathbf{v}_i$ as

$$\delta_i = \frac{1}{d_i} \sum_{j \in N(i)} (\mathbf{v}_i - \mathbf{v}_j).$$

The sum above is a discretization of the following curvilinear integral: $\frac{1}{|\gamma|}\int_{\mathbf{v}\in\gamma}(\mathbf{v}_i - \mathbf{v})\,dl(\mathbf{v})$, where γ is a closed simple surface curve around $\mathbf{v}_i$ and $|\gamma|$ is the length of γ. It is known from differential geometry that

$$\lim_{|\gamma|\to 0} \frac{1}{|\gamma|} \int_{\mathbf{v}\in\gamma} (\mathbf{v}_i - \mathbf{v})\, dl(\mathbf{v}) = -H(\mathbf{v}_i)\, \mathbf{n}_i$$

where $H(\mathbf{v}_i)$ is the mean curvature at $\mathbf{v}_i$ and $\mathbf{n}_i$ is the surface normal. Therefore, the direction of the differential coordinate vector approximates the local normal direction and the magnitude approximates a quantity proportional to the local mean curvature [4] (the normal scaled by the mean curvature is termed mean-curvature normal). Intuitively, this means that the δ-coordinates encapsulate the local surface shape (see Figure 1).

It should be noted that geometric discretizations of the Laplacian have better approximation qualities. Meyer *et al.* [5] propose employ the *cotangent weights*, first proposed by Pinkall and Polthier [6], instead of uniform weights:

$$\delta_i^c = \frac{1}{|\Omega_i|} \sum_{j \in N(i)} \frac{1}{2}(\cot\alpha_{ij} + \cot\beta_{ij})(\mathbf{v}_i - \mathbf{v}_j)$$

where $|\Omega_i|$ is the size of the Voronoi cell of i and α_{ij}, β_{ij} denote the two angles opposite of edge (i, j) (see Figure 2), to approximate mean-curvature normals. These geometry-dependent weights lead to vectors $\boldsymbol{\delta}_i^c$ with normal components only, unlike the previously defined $\boldsymbol{\delta}_i$ which also have tangential components and may be nonzero on planar 1-rings. The cotangent weights may be negative and problematic to define for very large angles due to the properties of cotangent near π; convex weights that mimic the cotangent weights are

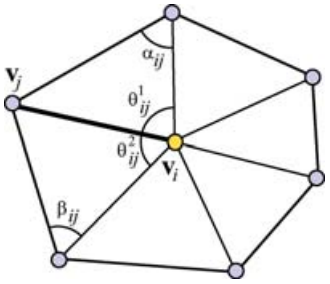

Figure 2: *The angles used in the cotangent weights and the mean-value coordinates formulae for edge (i, j).*

called mean-value coordinates [7]

$$w_{ij} = \frac{\tan(\theta^1_{ij}/2) + \tan(\theta^2_{ij}/2)}{\|\mathbf{v}_i - \mathbf{v}_j\|},$$

where the θ angles are depicted in Figure 2. These geometric discretizations are beneficial for applications where the mesh geometry is always known, such as mesh filtering and editing, since they approximate the differential properties more accurately. However, in compression applications we will define effective bases for geometry representation based on the topological Laplacian, so that the weights in the operator definition, and thus the bases themselves, are implicitly encoded in the mesh connectivity and no extra information is required to store them.

2.2. Surface reconstruction from δ-coordinates

In the following, we will be working with the above defined differential surface representation. We will perform different operations on the δ-coordinates, depending on the task at hand. The final step of any such manipulation must be surface reconstruction, or, in other words, we need to recover the Cartesian coordinates of $\mathcal{M}$'s vertices.

Given a set of δ-coordinates, can we uniquely restore the global coordinates? The immediate answer is no, because the matrix L (or L_s) that we use to transform from global to differential coordinates is singular, and therefore the expression $\mathbf{x} = L^{-1}\boldsymbol{\delta}^{(x)}$ is undefined. The sum of every row of L is zero, which implies that L has a nontrivial zero eigenvector $(1, 1, \ldots, 1)^T$. The matrix is singular since the δ-coordinates are translation invariant: if we use weights w_{ij} that sum up to 1 to define the Laplacian operator and we translate the mesh by vector $\mathbf{u}$ to obtain the new vertices $\mathbf{v}'_i$ then

$$\begin{aligned} L(\mathbf{v}'_i) &= \sum_{j\in N(i)} w_{ij}(\mathbf{v}'_i - \mathbf{v}'_j) \\ &= \sum_{j\in N(i)} w_{ij}((\mathbf{v}_i + \mathbf{u}) - (\mathbf{v}_j + \mathbf{u})) \\ &= \textstyle\sum_{j\in N(i)} w_{ij}(\mathbf{v}_i - \mathbf{v}_j) \;=\; L(\mathbf{v}_i). \end{aligned}$$

It follows that

$$\operatorname{rank}(L) = n - k$$

where k is the number of connected components of $\mathcal{M}$, because each connected component has one (translational) degree of freedom.

In order to uniquely restore the global coordinates, we need to solve a full-rank linear system. Assuming $\mathcal{M}$ is connected, we need to specify the Cartesian coordinates of one vertex to resolve the translational degree of freedom. Substituting the coordinates of vertex i is equivalent to dropping the ith row and column from L, which makes the matrix invertible. However, as we will see shortly, usually we place more than one constraint on spatial locations of $\mathcal{M}$s vertices. Denote by C the set of indices of those vertices whose spatial location is known. We have therefore $|C|$ additional constraints of the form

$$\mathbf{v}_j = \mathbf{c}_j, \quad j \in C. \tag{1}$$

If we assume w.l.o.g. that the vertices are indexed such that $C = \{1, 2, \ldots, m\}$ then our linear system now looks like this

$$\begin{pmatrix} L \\ \hline \omega\, I_{m\times m} \mid 0 \end{pmatrix} \mathbf{x} = \begin{pmatrix} \delta^{(x)} \\ \omega\, c_{1:m} \end{pmatrix}. \tag{2}$$

The same goes for y and z coordinate vectors, of course. We denote the system matrix in (2) by $\tilde{L}$. Note that we treat the positional constraints (1) in the least-squares sense: instead of substituting them into the linear system, we *add* the above equations as additional rows of the linear system. This way, for each $j \in C$ we are also able to retain the "smoothness" term, that is, the equation $L\mathbf{v}_j = \delta_j$. Note that in our discussion we will keep addressing (1) as "positional constraints" or "modeling constraints," although they may not be fully satisfied by the least-squares solution. The weight $\omega > 0$ can be used to tweak the importance of the positional constraints (generally, every constraint may have its own weight, and we can furthermore apply varying weights on the rows of the basic Laplacian matrix as well). The additional constraints make our linear system over-determined (more equations than unknowns) and in general no exact solution may exist. However, the system is full-rank and thus has a unique solution in the least-squares sense

$$\tilde{\mathbf{x}} = \operatorname*{argmin}_{\mathbf{x}} \left(\left\| L\mathbf{x} - \delta^{(x)} \right\|^2 + \sum_{j\in C} \omega^2 \, |x_j - c_j|^2 \right). \tag{3}$$

The analytical least-squares solution is expressed in terms of the rectangular $(n + m) \times n$ system matrix $\tilde{L}$ from Equation (2):

$$\tilde{\mathbf{x}} = (\tilde{L}^T \tilde{L})^{-1} \tilde{L}^T \mathbf{b}$$

where $\mathbf{b} = (\boldsymbol{\delta}, \omega c_1, \ldots, \omega c_m)^T$ is the right-hand side vector of the linear system. See Figure 3 for an example of a small mesh and its associated matrix $\tilde{L}$.

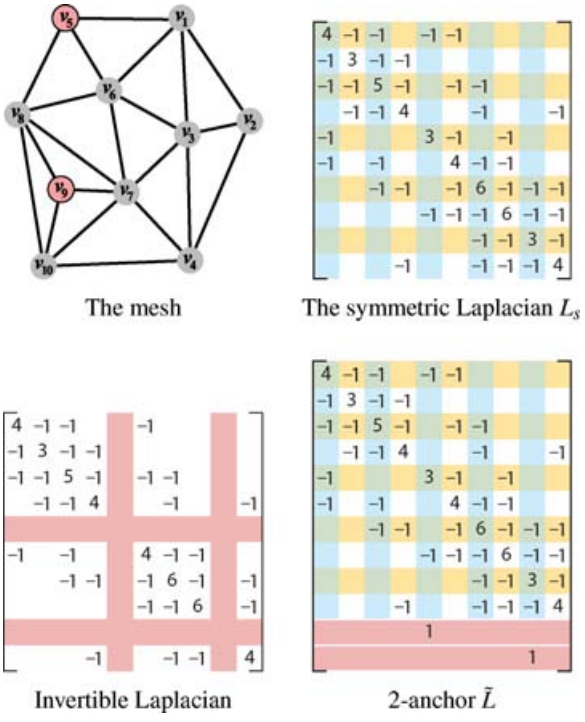

Figure 3: *A small example of a triangular mesh and its associated L_s matrix (top right). Second row: a 2-anchor invertible Laplacian and a 2-anchor $\tilde{L}$ matrix. The anchors are denoted in the mesh by red.*

An important practical aspect of surface reconstruction from δ-coordinates is the availability of efficient and accurate numerical methods for solving the least-squares problem (3). A recent comparative study [8] shows that direct methods prove to be superior for these types of problems on moderately sized meshes (up to a few hundreds of thousands of vertices). The system (3) can be solved by applying Cholesky factorization to the associated normal equations

$$(\tilde{L}^T \tilde{L})\,\mathbf{x} = \tilde{L}^T \delta.$$

The matrix $\tilde{L}$ is sparse, and $M = \tilde{L}^T \tilde{L}$ is also sparse (although not as sparse as $\tilde{L}$) and positive definite. By using fill-reducing reordering, it is possible to compute a *sparse* Cholesky factorization of M

$$M = R^T R$$

where R is an upper-triangular sparse matrix. The factorization is computed once, and we can solve for several mesh functions (x, y and z) by back substitution

$$R^T \xi = \tilde{L}^T \delta^{(x)}$$
$$R\,\mathbf{x} = \xi.$$

Table 1: *Time measurements of manipulations of the (extended) Laplacian matrices for several typical meshes, on a 3.4 GHz Pentium 4 computer.* Factorization *stands for Cholesky factorization of $\tilde{L}^T \tilde{L}$ and* Solve *denotes solving by back-substitution for one mesh function.*

Model	No. of vertices	Factor (seconds)	Solve (seconds)
Eight	2,718	0.085	0.004
Horse	19,851	0.900	0.032
Camel	39,074	2.096	0.073
Feline	49,864	2.750	0.110
Max Planck	100,086	7.713	0.240

The factorization takes the bulk of the computation time, and the back-substitution step is usually very fast. Using advanced linear solvers, such as TAUCS library [9], makes the computations very efficient. See Table 1 for timing data on several typical meshes. Another important advantage of direct solvers is the separation between the matrix processing (factorization) and the right-hand side, which is only involved in the fast back-substitution step. This permits frequently changing the right-hand sides at the price of the factorization pre-process, which is done only once per fixed matrix. This property is extremely useful for interactive mesh editing, as described in Section 4. When the system matrix is very large (million variables and more), the size of the factor becomes too large to fit into memory. In this case, it is possible to either use the out-of-core version of Cholesky factorization (available, e.g., with [9]) or iterative solvers whose complexity scales linearly with the size of the system (for instance, see the recent work on multigrid solvers [10,11]).

Note that the accuracy of the solution is highly dependent on the conditioning of the linear system. The Laplacian matrix is notoriously ill-conditioned: when we add only one anchor, the largest singular value of $\tilde{L}$ is proportional to the maximal degree in the mesh, while a good estimate for the smallest singular value is one over the product of the maximum topological distance of a vertex from the single anchor vertex and the number of vertices in the mesh [12,13]. For a typical n-vertex 3D mesh, the condition number is therefore likely to be $\Theta(n^{-1.5})$, and it is squared when we consider the normal equations. However, adding anchors to the system improves its conditioning and makes the factorization of the normal equations stable. A rigorous way to bound the smallest singular value of $\tilde{L}$ from below is shown in [14]; the bound depends on the number of anchors and their spacing across the mesh.

In summary, the transition from the global Cartesian representation to differential representation is performed by a *linear* operator with *local* support: the mesh Laplacian. And vice versa: in order to recover the Cartesian coordinates from the differential representation, one needs to solve a sparse linear least-squares problem. This provides a powerful

framework for surface manipulation: we will apply different modifications to the differential coordinates (such as quantization for compression purposes) and/or pose additional modeling constraints, and then reconstruct the surface by solving the least-squares problem. The advantages of this framework can be summarized as follows:

- It strives to preserve local surface detail as much as possible under the constraints.
- The least-squares solution smoothly distributes the error across the entire domain, providing a graceful reconstruction.
- Sparse linear systems can be solved very efficiently.

2.3. Spectral properties

Let us consider the L_s matrix, since it is symmetric and thus simpler to analyze. The matrix L_s is symmetric positive semi-definite and thus has an orthonormal eigenbasis

$$E = \{\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_n\}.$$

Denote the eigenvalues by λ_i,

$$0 = \lambda_1 < \lambda_2 \leq \lambda_3 \leq \cdots \leq \lambda_n.$$

It is known that the first nonzero λ_i are very small, and the largest eigenvalue λ_n is bounded by the largest vertex valence in $\mathcal{M}$ times 2. The first eigenvectors (corresponding to small eigenvalues) are smooth, slowly varying functions on the mesh, and the last eigenvectors have high frequency (rapid oscillations). For example, the first eigenvector $\mathbf{e}_1$ is the constant vector, that is, the "smoothest" mesh function that does not vary at all. In fact, the Laplacian eigenbasis is an extension of the discrete Fourier basis to irregular domains [4]. The eigenvalues are considered as mesh-frequencies. We will see in the following how this fact is exploited for signal processing on meshes, compact geometry representation, watermarking and more. See also [16] for an earlier state-of-the-art report on this subject.

3. Efficient Shape Representation

In this section we review several methods for efficient and compact geometry representation that benefit from the mesh Laplacian operator and the framework presented in Section 2. For an extensive recent survey on general compression methods, we refer the reader to [17].

3.1. Efficient bases for geometry representation

Finding a good basis for compact shape representation means that one can use only a fraction of the basis functions to approximate a given geometry well. This idea is extensively used in signal processing and in the image domain, by defining Fourier or wavelet bases. The regular 1D and 2D settings are well studied in both continuous and discrete settings, and the sampling theorem tells us the number of basis functions we need to use for perfect signal reconstruction. Things are more involved for the irregular setting of arbitrary triangular meshes. One possible approach of direct application of the signal processing theory on meshes is to first perform semi-regular remeshing by parameterizing the mesh over a simple base complex, and then apply variations of 2D signal processing methods [18,19,20,21,22]. Another possibility is, instead of altering the mesh, to work directly on the irregular mesh and develop generalizations of the methods from regular settings.

Karni and Gotsman [15] introduce a spectral method where the mesh is approximated by reconstructing its geometry using a linear combination of a number of basis vectors. The basis is the spectral basis E of the L_s matrix, which can be regarded as a generalization of the discrete Fourier basis (see Section 2.3). Similarly to JPEG compression of images, the mesh geometry functions $\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}$ are decomposed in the basis E:

$$\mathbf{x} = \alpha_1 \mathbf{e}_1 + \alpha_2 \mathbf{e}_2 + \cdots + \alpha_n \mathbf{e}_n \tag{4}$$

and the high-frequency components (coefficients of the last eigenvectors) are truncated. The coefficients α_i are quantized and subsequently entropy coded. It is assumed that the mesh connectivity is known to the decoder prior to receiving the geometry coefficients, so that the decoder first computes the spectral basis and then reconstructs the geometry by combining the coefficients. Note that this method is readily made for progressive streaming, since if one sends the α_i coefficients in ascending order of i, the decoder can first reconstruct a coarse (very smooth) approximation of the geometry, and then gradually add high-frequency detail, as more coefficients become available.

An example of the progressive encoding of [15] is shown in Figure 4. The geometry of the *Horse* model is first approximated using only a few of the first eigenvectors $\mathbf{e}_i$, and the resulting approximation is very smooth. As more and more eigenvectors participate in the approximation, the obtained surface receives more high-frequency detail.

The spectral method of [15] provides high compression ratios, since most of the energy is concentrated in the low-frequency coefficients (the first α_i), and the high-frequency components add negligible weight. However, computing even the first few eigenvectors of typical meshes (with tens of thousands of vertices) is an extremely computationally expensive task of superlinear complexity, that has to be carried out both on the encoder and the decoder side. Karni and Gotsman [15] propose to divide the mesh into patches and compute the spectral decomposition separately for each patch. This speeds up the computation, but the quality of the low bit-rate reconstructions suffers from artifacts along the boundaries between patches. Later, Karni and Gotsman developed another technique, where *fixed* spectral bases are

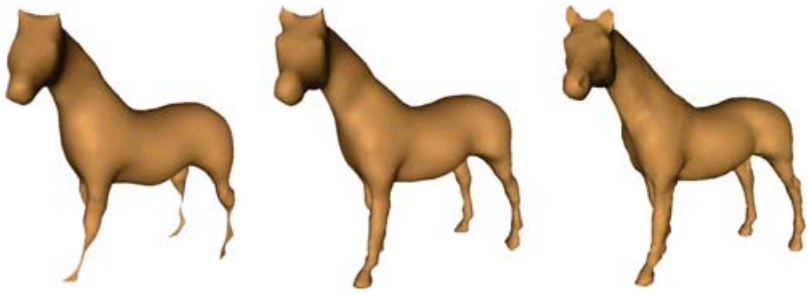

Figure 4: *Progressive approximations of the* Horse *geometry using the eigenvectors of the topological Laplacian [15]. The left figure shows an approximation using only a small number of the first eigenvectors (those that correspond to small eigenvalues). When more eigenvectors are added to the approximation, the surface attains more high-frequency detail (middle and right). Data courtesy of Zachi Karni.*

employed [23]. The spectral basis of a regular mesh with n vertices is used to approximate the Laplacian eigenbasis, and the computational cost of spectral analysis is avoided on the decoder side altogether. Of course, this technique works better for meshes that are close to being regular.

In [24,25], a different class of shape approximation techniques for irregular triangular meshes was introduced. The method approximates the geometry of the mesh using a linear combination of a small number of basis vectors that are functions of the mesh connectivity and of the mesh indices of a number of *anchor* vertices. The initial motivation was to improve the shortcomings of the spectral basis [15]. One must bear in mind that, together with their appealing properties, the spectral bases are geometry-oblivious, since the basis vectors are functions of the connectivity alone. In contrast, the new *geometry-aware* method derives the basis both from the mesh connectivity and geometry. The basis vectors in [25] are centered around selected "geometrically important" anchor vertices. This allows a terse capturing of important features of the surface and leads to compact and efficient representation of the mesh geometry. Figure 5 illustrates the reconstruction using such basis vectors on a 2D curve example. The bottom row shows the meshes reconstructed using geometry-aware bases. The locations of the anchors are marked by small dots. The reconstructed mesh passes close to the original locations of the anchor points, which enables good approximation of such features as the tips of the bird's wings and tail. For comparison, reconstruction of this mesh using an analogous number of spectral basis vectors misses out the features. This behavior is evident in large as well as in small scale.

It should be noted that explicit computation of the basis vectors is, generally speaking, too expensive for large meshes. The method in [25] avoids explicit computation of the underlying basis. Instead of directly representing the geometry by the coefficients of the linear combination of the basis vectors, the reconstruction problem is reduced to solving a sparse linear least-squares system of the type (2). Each basis function is selected so that it fulfills the following conditions in the least-squares sense: it attains the value 1 at one of the anchors and 0 at the other anchors, and it is the smoothest among all the functions that satisfy these requirements (a slightly different definition for strictly interpolatory anchors is also proposed, but the principle is the same). The smoothness of a function is

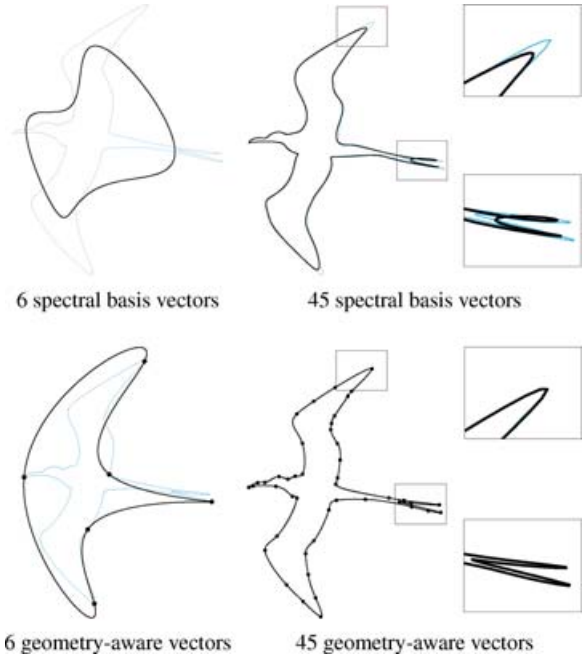

Figure 5: *Reconstruction of the swallow curve (simple closed path) using different bases. The top row shows reconstruction using the Laplacian eigenvectors, which are the discrete Fourier basis function in this case. The bottom row displays reconstruction with geometry-aware basis vectors. The reconstructed mesh is shown in black, while the original mesh is tinted in blue. The geometry-aware bases better approximate the features of the shape, on a large as well as on a small scale.*

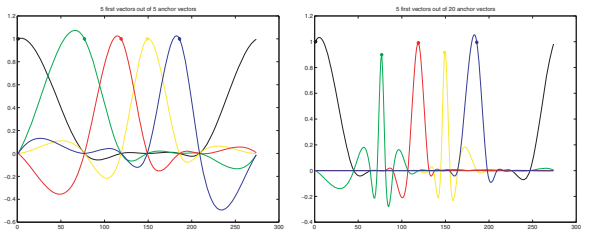

Figure 6: *Geometry-aware basis functions on a 1D domain. The mesh here is a simple closed path with 274 vertices. Plot (a) displays the five basis functions corresponding to a set of five anchors; (b) shows the first five basis functions out of a 20-anchor basis.*

defined in a discrete manner using the Laplacian of the mesh. Specifically, the basis functions are defined as follows: Given a set of k anchor vertex indices $1 \le a_1, a_2, \ldots, a_k \le n$, the ith function $\mathbf{f}_i$ in the basis minimizes

$$\|L\,\mathbf{f}_i\|^2 + \left(\omega^2 \left|(f_i)_{a_i} - 1\right|^2 + \sum_{j \neq i} \omega^2 \left|(f_i)_{a_j} - 0\right|^2\right)$$

where ω is a positive weight. See Figure 6 for a 2D example of these basis functions (note that they were computed here for illustration purposes only; the algorithm in [25] does not require the bases explicitly). The definition results in smooth basis functions that are easy to combine into an approximation that attains specific values at the anchors. The compact geometry representation thus consists of the indices of the anchor vertices and the coefficients associated with the basis functions. To recover the surface geometry, we simply need to solve the system (3), where instead of δ we put zero

$$\tilde{\mathbf{x}} = \sum_{i=1}^{k} c_i\,\mathbf{f}_i = \operatorname*{argmin}_{\mathbf{x}} \left\{ \|L\mathbf{x}\|^2 + \sum_{i=1}^{k} \omega^2 \left|x_{a_i} - c_i\right|^2 \right\}.$$

Taking a sufficiently large value for ω (in the order of n) makes the reconstruction effectively interpolatory at the anchors. Figure 7 demonstrates an example of shape approximation obtained with these basis functions.

3.2. High-pass quantization

One of the main ways to compress floating-point data, such as mesh geometry, is by quantization. Quantization necessarily introduces errors and causes a certain loss of data. Loosely speaking, quantizing the Cartesian coordinates of the mesh produces high-frequency errors across the surface. This especially damages the fine-sampled areas, since the relative error is greater when the polygons are smaller. Aggressive quantization significantly alters the surface normals, causing the irritating "jaggies" effect. Thus, only mild quantization of Cartesian coordinates is possible without causing visible artifacts (10–16 bits per coordinate).

In [26], a different approach to geometry quantization is proposed. Instead of directly quantizing the Cartesian coordinates, the quantization is applied to the δ-coordinates, and the geometry of the mesh can be restored on the decoder side by solving a linear least-squares system defined by the extended Laplacian matrix (discussed in Section 2.2). Introducing high-frequency errors by quantizing the δ-coordinates results in *low-frequency* errors in the reconstructed Cartesian coordinates. By considering a visual error metric between meshes, which takes into account not only the Euclidean distance between corresponding vertices (or the "Metro" distance [27]) but also the smoothness error, it can be argued that low-frequency displacements in the surface geometry are less noticeable than high-frequency displacements that modify the local characteristics of the surface such as normals and curvature. Consequently, strong quantization of the δ-coordinates yields a small visual error, in contrast to standard Cartesian coordinate quantization.

The strategy of [26] is called *high-pass quantization*, to emphasize the fact that it tends to concentrate the quantization error at the low-frequency end of the spectrum, in contrast to the spectral method of [15]. Mesh frequencies are defined as in Section 2.3. Let us look what happens to quantization errors. The δ-coordinates are obtained by applying the mesh Laplacian, $\delta = L_s\mathbf{x}$. When the δ-coordinates are quantized, an error ε (with small norm) is added. To reconstruct the mesh function $\mathbf{x}$ from $\delta + \varepsilon$, we write

$$\mathbf{x}' = L_s^{-1}(\delta + \varepsilon) = L_s^{-1}\delta + L_s^{-1}\varepsilon = \mathbf{x} + L_s^{-1}\varepsilon$$

(We assume a nonsingular L_s by adding one anchor.) Thus, the quantization error is $\mathbf{q}_\varepsilon = L_s^{-1}\,\varepsilon$, and its norm is no longer small. In fact, if we look at $\mathbf{q}_\varepsilon$ in the spectral basis, we will see that its low-frequency component is amplified

$$\begin{aligned} \varepsilon &= c_1\mathbf{e}_1 + c_2\mathbf{e}_2 + \cdots + c_n\mathbf{e}_n \\ \mathbf{q}_\varepsilon &= L_s^{-1}\varepsilon = \frac{1}{\lambda_1}c_1\mathbf{e}_1 + \frac{1}{\lambda_2}c_2\,\mathbf{e}_2 + \cdots + \frac{1}{\lambda_n}\,c_n\mathbf{e}_n. \end{aligned}$$

The last statement holds since a matrix and its inverse share the same set of eigenvectors, and the corresponding eigenvalues are inversed. The first λ_i are very small, thus the first $\frac{1}{\lambda_i}$ are large. This means that $\mathbf{q}_\varepsilon$ has very strong components in the direction of the first $\mathbf{e}_i$, that is, low frequencies. The high-frequency components are damped, because the last λ_i are relatively large, so for large i, $\frac{1}{\lambda_i} < 1$. Thus we can conclude that when quantizing δ-coordinates, we get mostly low-frequency errors. Note that the eigendecomposition is used solely for analysis purposes; the quantization method only requires linearly computing the δ-coordinates and linear surface reconstruction on the decoder side.

To reduce the low-frequency error, more anchor vertices, whose spatial location is known, can be added to the

Figure 7: *Reconstruction of the* Feline *model using an increasing number of geometry-aware basis vectors. The sizes of the encoded geometry files are displayed below the models. The letter e denotes the L^2 error value, given in units of 10^{-4}.*

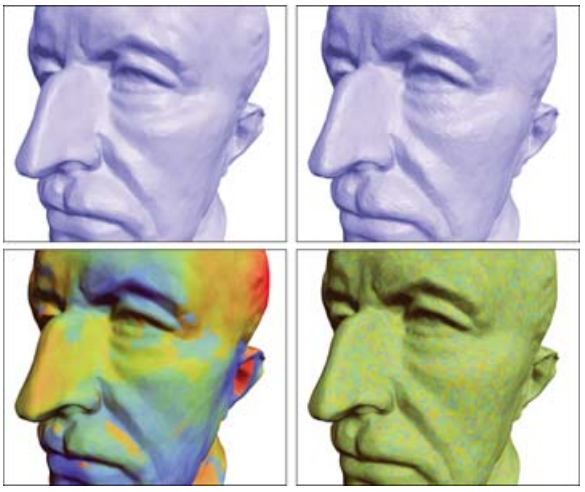

Figure 8: *The δ-coordinates quantization to 5 bits/coordinate (left) introduces low-frequency errors, whereas Cartesian quantization to 11 bits/coordinate (right) introduces noticeable errors. The upper row shows the quantized model, and the bottom figures use color to visualize the corresponding quantization errors. It is recommended to zoom in into the electronic version to see the quantization artifacts.*

representation. These vertices "nail" the geometry in place and prevent large shifts or rotations. Algebraically, anchors improve the conditioning of the Laplacian matrix by making the smallest singular value larger. Adding anchors requires more storage space, and therefore their placement needs to be optimized so as to minimize the error. A greedy placement technique is given in [26], where the anchors are chosen one-by-one, each new anchor is placed at the vertex that achieved highest reconstruction error in the previous iteration. A different placement strategy that conforms with provable upper bounds on the geometry error is introduced in [14].

Figures 8 and 9 show some quantization results from [26].

3.3. Mesh watermarking

The visual properties of low-frequency errors in geometry prove to be useful not only for geometry compression. The insensitivity of the human visual system to low-frequency errors (such as smooth shifts or rotations of parts of the surface) is exploited in mesh watermarking techniques [28,29,30]. Mesh watermarking, similar to image watermarking, is essential for copyright protection and ownership authentication of geometric data. The goal is to embed an *invisible* bit string into the geometric representation of the object, so that the actual owner of the authentic digital object would be able to robustly compare the marked copy with the real (unmodified) version of the 3D object and to verify the presence of the correct watermark.

The idea of Ohbuchi *et al.* [29,30] was to embed a watermark bit string into the spectral representation of the 3D mesh at hand, by slightly modifying the low-frequency coefficients of the spectral decomposition (4). As previously discussed, small changes in the low-frequency component of the surface geometry, or low-frequency errors, do not alter the local surface appearance, and thus they are less apparent to the human eye. Therefore, in applications where the 3D meshes are used primarily for display, such an approach to watermarking is definitely plausible. In brief, the watermarking algorithm works as follows: the input mesh is partitioned into balanced patches to reduce the cost of spectral decomposition, and then the eigendecomposition of the symmetric topological Laplacian matrix is computed for each patch (see Equation (4)). Next, the low-frequency coefficients (the first α_i) are modulated by the selected watermark bit string, and the mesh geometry is reassembled by summing up the right-hand side of (4) with the modified coefficients. This completes the watermarking process. To extract the watermark from a "suspected" mesh, it is first aligned with the original mesh and remeshed so that it has the same connectivity as the original mesh. Then, spectral analysis is performed, and the spectral coefficients of the suspected mesh are subtracted from the original (unmodified) coefficients, so that the watermark bit string can be retrieved.

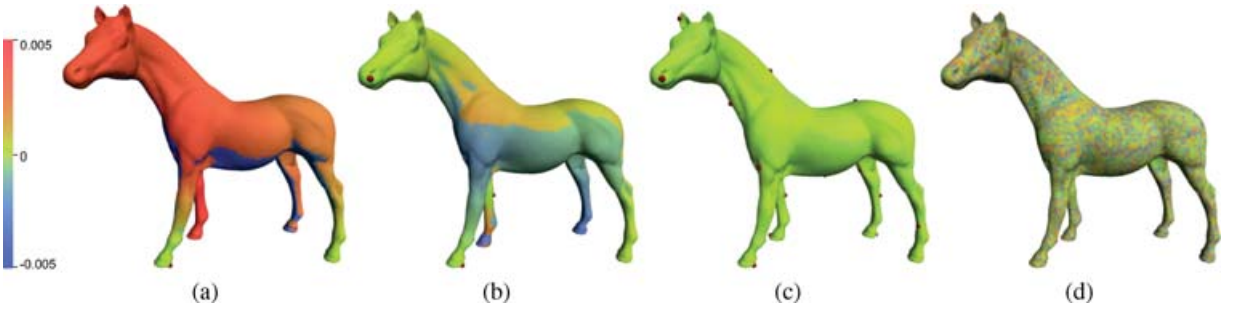

Figure 9: *Visualization of the visual error across the mesh surface. The surface was reconstructed from δ-coordinates quantized to 7 bits/coordinate using 2 anchors (a), 4 anchors (b) and 20 anchors (c). The anchor points are shown as small red spheres. Each vertex v is colored according to its visual error value $E_{vis}(v)$. We have also added a sign to these values, so that vertices that move outside of the surface have positive error values (colored by red), and vertices that move inwards have negative error values (colored by blue). In (d), the visual error of direct quantization of the Cartesian coordinates is shown.*

Recently, the above watermarking scheme was extended to point-set data where explicit connectivity is not present (see [31]).

4. Mesh Editing and Shape Interpolation

Manipulating and modifying a surface while preserving the geometric details is important for various surface editing operations, including free-form deformations [32,33], cut and paste [34,35,36], fusion [37], morphing [38] and others. Note that the absolute position of the vertices in a mesh is not important for these operations; what is needed is a local description of the shape that is not dependent on the particular placement of the shape in the Euclidean space. We call such shape representation "shape-intrinsic": it is invariant under rigid transformations of the space in which the shape is embedded, and it allows to reconstruct the shape uniquely, up to a rigid transformation. In the following we will describe several types of local surface representation that employ the Laplacian framework and other differential representations, and show how they are used for mesh editing. In all these representation methods the main question is how to make them behave in a shape-intrinsic manner.

4.1. The modeling metaphor

Let us first briefly describe the traditional way of direct surface manipulation. The modeling metaphor is very simple from the user's point of view [39]. First, the user marks the desired region of interest (ROI) on the mesh that is to be modified. The rest of the mesh will remain unchanged (and thus does not participate in the calculations). Next, the user selects a group of vertices that will serve as the manipulation handle. The user can then drag the handle around and apply arbitrary transformations to it (such as rotation, scaling, etc.), and the surface of the ROI is modified on the fly, following the handle manipulation.

From the algorithmic standpoint, the handle vertices function as boundary conditions for the calculation of the edited surface. These conditions change every time the user manipulates the handle. An additional set of boundary constraints expresses the static parts of the surface that remain unchanged during the editing process. Usually, this set of constraints consists of some padding belt of vertices around the ROI boundary, and it ensures a gradual transition between the ROI and the rest of the mesh (which, as mentioned above, is completely static and is usually not included the editing process at all for efficiency reasons). Thus, if we denote the group of constrained vertices (those that belong to the handle and the static ones) by U, the boundary constraints are written simply as

$$\mathbf{v}_i = \mathbf{u}_i, \qquad \forall i \in U \tag{5}$$

where $\mathbf{u}_i$ are the desired positions for the constrained vertices.

Any editing algorithm in fact changes the shape of the ROI so as to satisfy the modeling constraints (5) while striving to give the modified surface a *natural* shape. Here, we will concentrate on one particular definition of natural deformation, namely: a deformation that preserves the local details of the original surface. In the next sections we will see how the above modeling metaphor is implemented using various surface representations.

4.2. Multiresolution mesh editing

Partially shape-intrinsic surface mesh representations are given by multiresolution decompositions [19,21,39,40,41,42,43]. In a multi-resolution mesh, the geometry is encoded as a base mesh and several levels of refinement. The base mesh is the low-frequency component of the shape, and it is typically represented in Cartesian coordinates. Thus it is not shape-intrinsic, but the shape details are represented in an intrinsic manner. The

refinements are described locally, so that the geometric details are mostly captured in a discrete set of translation- and rotation-invariant coordinates. Using this representation, modeling operations can be performed on an appropriate user-specified level-of-detail. The multiresolution hierarchy can be constructed for both the connectivity and the geometry (as in, e.g., [19,21]), or the only geometry (as in, e.g., [39,42,43]), where the connectivity of the mesh remains full and unchanged in all the levels-of-detail in the hierarchy. Different levels of refinement are usually obtained by gradual smoothing of the mesh.

We will outline the editing mechanism of the multiresolution framework of Kobbelt and colleagues [39,41,43] as an exemplary multiresolution technique that also uses the Laplacian framework. For simplicity, let us assume that the mesh hierarchy consists of two levels only, sharing the same connectivity: the smooth base mesh and the detailed mesh. The detailed level is encoded with respect to the base level by computing the offset of each vertex and representing this offset in the *local* frame of the corresponding vertex in the base mesh. The choice of the local frames is rotation- and translation-invariant, and therefore the representation of the offsets (which are, in fact, the surface details in this case) is independent of the location of the mesh in space. Mesh editing is performed as follows: first, the smooth base mesh is deformed according to the constraints (5); then the local frames are recomputed on the deformed base mesh, and finally, the original detail offsets (in their local frame representation) are added to the base mesh. This way, the local details are reproduced correctly in the deformed mesh: for example, if the surface was bent, the local frames of the smooth base mesh rotate, and thus so do the offsets. This key concept of invariance in local representation is illustrated in Figure 10.

How is the base mesh deformed? It is assumed that the base mesh is smooth enough and does not contain local details that need to be preserved. Therefore, the base mesh is deformed by solving an optimization problem with two objectives: (i) satisfying the user constraints (5) and (ii) keeping the surface of the base mesh smooth. The latter goal is achieved by describing smoothness in terms of the mesh Laplacian operator, as we already saw in Section 3.1. Botsch and Kobbelt [43] propose different orders of smoothness, obtained via different orders of the Laplacian operator. The optimization is formulated as

$$L^k \mathbf{x} = 0$$

leading to C^{k-1} smoothness of the resulting surface. Probably the most common value is $k = 2$, producing the so-called "thin-plate" surface. The Equations (5) are incorporated as hard constraints into the above system by substitution, securing its nontrivial solution. When $k = 2$, this optimization is equivalent to solving the least-squares problem (3) where all the δ-coordinates are set to zero and sufficiently large weight is put on the positional constraints, to effectively make them hard constraints. Note that the factorization of the system matrix can be computed once per choice of ROI and handle vertices; each time the user transforms the handle, only the right-hand side of the system changes and thus only a back-substitution step is required. This allows interactive response of the editing mechanism for fairly large ROIs (up to several tens of thousands of vertices, refer to Table 1).

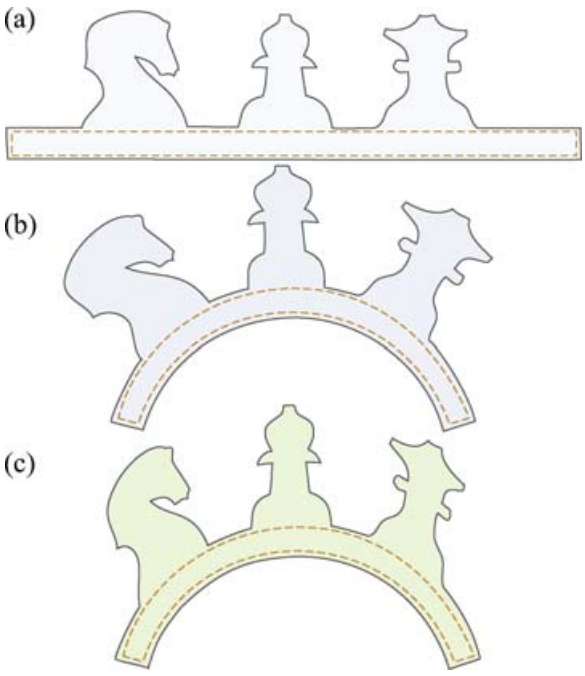

Figure 10: *The importance of rotation-invariant representation. (a) the original shape, where the base shape is drawn in brown. In (b), the base shape was bent, while the details were encoded in a rotation-invariant manner, and thus their orientation with respect to the base was preserved. In (c), the same deformation leads to distorted orientation of the details (stretch) because the encoding was not rotation-invariant.*

The multiresolution approach is probably the most intuitive method for mesh editing, and its advantage is the straightforward representation of the details of the shape in a manner that is invariant to the global coordinate system. On the other hand, this approach defines details *explicitly*, or in other words, one may need to explicitly set the required amount of smoothing to create a satisfactory base mesh. In meshes with complex details, many levels of multi-resolution hierarchy may be required in order to handle the details correctly.

4.3. Single-resolution local representations for mesh editing

Local details can be defined in a more implicit way with a single-resolution mesh, by using differential coordinates.

Mesh editing using this representation was pioneered by Alexa [44]. The basic idea is simply to add the positional constraints (5) to the system that reconstructs the Cartesian coordinates of the mesh from the differential coordinates (3). Once again, note that the system matrix needs to be constructed and factored only once per choice of ROI and handle; during user manipulation one only needs to solve the linear system by back-substitution, which is very efficient.

The above idea for surface representation and reconstruction was further elaborated in [45,46,47]. Yu *et al.* [47] introduce a technique called Poisson Editing, formulated by manipulation of the gradients of the coordinate functions (x, y, z) of the mesh. The surface is reconstructed by solving the least-squares system resulting from discretizing the Poisson equation

$$\nabla^2 f = \Delta f = \nabla \cdot \mathbf{w} \tag{6}$$

with Dirichlet boundary conditions. Here, f is the unknown scalar function on the mesh, that is, one of the coordinate functions x, y, z of the deformed mesh. The vector field $\mathbf{w}$ is the gradient field of the corresponding *original* coordinate function. The set of boundary constraints consists of both positional constraints of the form (5) and constraints on the gradients of the handle vertices, directly expressed by the desired linear transformation on the handle. The spatial Laplacian operator Δ, restricted to the mesh, is discretized using the cotangent weights. By inspecting Equation (6), we can see that it has a similar form to the Laplacian surface reconstruction system (3); the difference lies on the right-hand side, that is expressed in terms of a vector field rather than a scalar field, and the constraints are substituted into the system (so that this is not a least-squares optimization).

In [45], the surface is reconstructed from the Laplacian δ-coordinates of the mesh and spatial boundary conditions by solving (3). Both Lipman *et al.* [45] and Yu *et al.* [47] point out the main problem of the above approach: the need to rotate the local frames that define the Laplacians, or the gradients, to correctly handle the orientation of the local details. If this issue is not properly addressed, the system in (3) (or in (6)), tries to preserve the Laplacian vectors (or the gradients) in their original orientation with respect to the *global* coordinate system. Therefore a result in the spirit of the sketch in Figure 10c would be obtained.

Yu *et al.* and Lipman *et al.* propose remedy to this problem by explicit assignment of the local rotations. Lipman *et al.* [45] estimate the local rotations of the frames on the underlying smooth surface by smoothing the "naive" solution of (3); then the original Laplacians are rotated by these estimated rotations and the reconstruction system is solved again. This heuristic achieves good results on meshes with relatively small details; applying it to complex meshes where the details are far from being height-fields above a smooth base surface would require a lot of smoothing iterations. An important feature of this approach is that it deduces the local rotations based on *translational*, or positional constraints alone; the user is not expected to provide rotational transformations of the handle. So even in the simplest case of grabbing one handle vertex and "pulling" it out of the surface, the technique will produce rotation estimates for the surface details and they will be transformed accordingly.

Yu *et al.* [47] propagate the rotation (or any other linear transformation) of the editing handle, defined by the user, to all the gradient vectors of the ROI. This is the major difference from [45], because here the user is expected to specify *compatible* translation and rotation constraints on the handle; translational constraints alone will result in no local rotations (and thus the surface details may be significantly distorted). The propagated transformations are interpolated with the identity, where the interpolation weight for each triangle is proportional to the geodesic distance of the triangle from the handle. As pointed out in [48], this solution may also produce unexpected results for meshes with complex features protruding from the smooth base surface. This is because the tip of a protruding feature is geodesically further from the handle than the vertices around the feature's base, and thus the tip receives a smaller share of the deformation than the base (whereas for a natural result, it should be roughly the same amount of deformation).

A different interpolation scheme for deformation propagation was recently proposed by Zayer *et al.* [49]: it avoids geodesics computation, creating a harmonic scalar field on the mesh instead. The harmonic field attains the value of 1 at the handle vertices and 0 at the boundary of the ROI, and varies monotonically on the rest of the ROI, thus providing a valid and smoother alternative to the geodesic weights. The harmonic field is obtained by solving (2), only the 1 and 0 constraints are substituted into the system rather than added in least-squares sense. An additional advantage of this approach is that it uses the same system matrix both for the computation of the weights and for the actual editing (only with different right-hand sides), thus the factorization of the matrix serves both tasks at once. As a side note, we mention that the mesh Laplacian operator is tightly coupled with Hodge theory and computation of globally conformal (harmonic) mesh parameterizations for arbitrary topology. This topic is outside of the scope of this paper; the reader is kindly referred to recent publications [6,50,51,52] for further information.

It is important to note that all the above-mentioned methods [45,47,49] are fast: the size of the system matrices involved in the reconstruction is $n \times n$, where n is the number of vertices in the ROI. This size will grow in the follow-up works, as we will see shortly.

Subsequent research continued to look for truly shape-intrinsic representation suitable for mesh editing. Sheffer and Kraevoy [53] introduced the so-called *pyramid coordinates*: at each vertex i of the mesh, the normal and the tangential components of the local frame are stored, independently of

the global coordinate frame. To construct this representation, a local projection plane is fitted to the vertices of the 1-ring by averaging the face normals of the 1-ring. The normal component of the center vertex i is its distance to the projection plane, whereas the tangential component is encoded as the edge lengths and the angles between the edges of the 1-ring, projected onto the plane. Since this representation is evidently rotation-invariant, it allows performing natural and large deformations on meshes based on translational (or other) constraints on the handle, as well as linear blend between different shapes (given the correspondence between them). However, the reconstruction of the Cartesian coordinates from the pyramid coordinates is clearly a nonlinear optimization (involving cross-product and norm computations to construct the projection planes), which makes it time consuming.

In [46], the Laplacian editing method that *implicitly* transforms the differential δ-coordinates is proposed, based on finding an *optimal* transform for each vertex. The editing paradigm implies positional constraints on some vertices, and it deduces the local rotations based on translations of the handle, similarly to [45]. The local transforms are defined by comparing 1-rings in the original mesh and the unknown deformed mesh, and are formulated as linear expressions in the vertices of the unknown mesh. The reconstruction Equation (3) is thus altered to make the differential coordinates similarity-invariant: we solve for the reconstructed geometry $V' = \{\mathbf{v}'_1, \ldots, \mathbf{v}'_n\}$ by minimizing

$$E(V') = \sum_{i=1}^{n} \left\| T_i(V')\delta_i - L(\mathbf{v}'_i) \right\|^2 + \sum_{i \in U} \|\mathbf{v}'_i - \mathbf{u}_i\|^2 \quad (7)$$

where U is the set of constrained vertices. The transformations T_i are themselves unknowns that linearly depend on the unknown vertices V'. Each T_i is a transformation that takes the original 1-ring of vertex i to the newly reconstructed one

$$T_i = \operatorname*{argmin}_{T_i} \sum_{j \in \{i\} \bigcup N(i)} \|T_i\, \mathbf{v}_j - \mathbf{v}'_j\|^2$$

In addition, the T_i's are constrained to be translations, rotations and scales only, in order to enforce local detail preservation and to disallow shears. In 2D, this is easy since similarity transformations have linear form

$$T_i = \begin{pmatrix} s & -h & t_x \\ h & s & t_y \\ 0 & 0 & 1 \end{pmatrix}$$

In 3D, however, a linearized version of rotation constraint is needed, since true 3D rotation matrices cannot be expressed linearly in 3D. If $\mathbf{h}$ represents the rotation axis and H is the skew-symmetric matrix such that $H\mathbf{x} = \mathbf{h} \times \mathbf{x}$, $\forall \mathbf{x} \in \mathbb{R}^3$, then the class of translations, rotations and scales has the form

$$s \exp H = s(\alpha I + \beta H + \gamma\, \mathbf{h}^T \mathbf{h}).$$

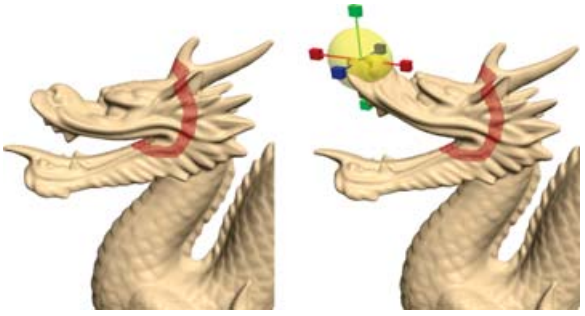

Figure 11: *Laplacian mesh editing [46]. The red belt bounds the region of interest; editing is performed by manipulating the handle object represented by the yellow sphere.*

The last term is quadratic, and is dropped, which results in the following linearized form

$$T_i = \begin{pmatrix} s & -h_3 & h_2 & t_x \\ h_3 & s & -h_1 & t_y \\ -h_2 & h_1 & s & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

For further details on the derivation of the expressions for T_i's see [54]. The linearization causes anisotropic scaling by $\cos\,\theta$ along the rotation axis $\mathbf{h}$, where θ is the rotation angle. Therefore, when large rotations are involved in the editing constraints, this would lead to scaling artifacts. To rectify the anisotropy, [46] propose to remove this anisotropic scaling in postprocessing (by scaling the transformed differential coordinates back to their original size) and reconstruct again using the basic Equation (3). Note that this repair step also handles the isotropic scaling that may have been caused due to the scale-invariance of (7). Also note that the size of the linear system that corresponds to (7) is $3n \times 3n$ because in the expressions for T_i's there is a dependence between the x, y and z coordinates of the mesh, and thus we can no longer solve the system separately for each coordinate. Figure 11 shows an example of editing with this method.

A 2D variant of the editing algorithm of [46] was successfully applied to textured 2D shapes with meshed interior in the work of Igarashi *et al.* [55]. As mentioned, the 2D version of the linear reconstruction problem (7) contains *exact* linear expressions for transformations T_i that consist of rotations, isotropic scale and translations only. Igarashi *et al.* propose a different solution to the elimination of isotropic scale in the deformed shape: after the initial solution to (7) given the user constraints, each triangle of the original shape is fitted into its corresponding triangle in the deformed shape, mimicking the new orientation and translation but retaining the original size of the triangle. Then, another linear system is solved to consolidate all these fitted triangles together, similar to the approaches in [56,57]. Since the 2D editing method works on

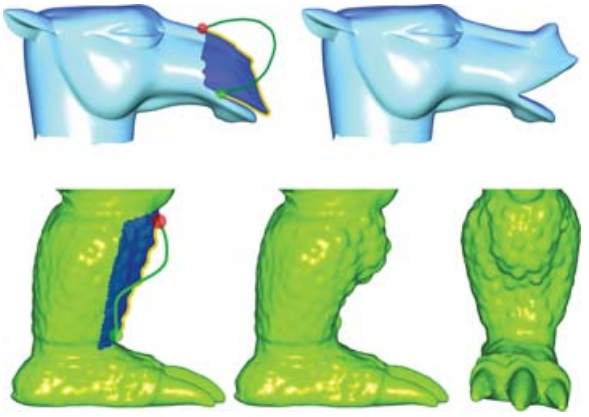

Figure 12: *Examples of sketch-based editing of silhouettes [58]. The sketched curve hints at the new desired shape of the silhouette. By weakly weighting the sketch constraints in the reconstruction process (Equation (7)), the details of the shape are preserved, and the deformation follows the hint of the user.*

a "volumetric" representation (meshed area of the interior of the shape in the 2D case), the deformations exhibit realistic rigidity and shape preservation. Additional important contribution of this work is the integration of new input devices for direct interaction with the shapes, such as two-handed mouse devices and touch-pads.

The 3D Laplacian editing [46] has been recently extended and adapted for advanced modeling metaphors. Nealen *et al.* [58] employ the Laplacian framework for sketch-based editing that provides an intuitive yet powerful interface for the user. The main building block of the interface is manipulation of sketched curves: the user can edit silhouette curves or any other paths on the mesh via direct sketching of the new shape for the curves. The underlying machinery is an adaptation of Equation (7), where the positional constraints are weighted differently according to the intention of the user: strong weighting if the intention is to closely follow the sketch, or weak weighting for approximate sketching. See Figure 12 for an example of such approximate silhouette sketching. In addition, the new editing system allows to create sharp or smooth ridges and ravines over the sketched curves, by constraining the Laplacians to specific values (attained by user-controlled scaling). Finally, suggestive contours (see [59]) can be created by assigning specific rotations to the Laplacians in the region of interest around the sketched curve.

The pursuit after a local differential representation that is invariant to rigid transformations and enables linear reconstruction mechanism achieved a significant progress in the work of Lipman *et al.* [48]. In this paper, a new differential representation is proposed, which bears similarity to the classical differential geometry notions of the first and the second fundamental forms. Assuming a normal to the tangent plane is given at each vertex of the mesh, Lipman *et al.* define two discrete forms: the first represents a scalar product in the tangent plane of each vertex, and is represented by edge lengths and the angles between the edges of the 1-ring, projected onto the tangent plane. The second form is linear and expresses the height function of the mesh with respect to the tangent plane. It is represented by the signed distance of the 1-ring vertices to the tangent plane. Clearly, such a representation is invariant to the global coordinate frame, because it consists solely of distances and angles (similarly to the pyramid coordinates [53]).

Lipman *et al.* [48] prove that, given the values of the discrete forms that were taken from an existing mesh, the mesh Cartesian geometry can be uniquely restored up to global translation and rotation. The *linear* reconstruction process is the key issue: to achieve it, two decoupled steps are needed. First, the local frames of each vertex are reconstructed. This can be done by constructing and solving a linear system of equations that encodes the *changes* between the local frames of adjacent vertices. For each mesh edge (i, j) we need an equation that expresses the difference between $\{\mathbf{b}_1^i, \mathbf{b}_2^i, \mathbf{N}^i\}$ (the local frame at i) and $\{\mathbf{b}_1^j, \mathbf{b}_2^j, \mathbf{N}^j\}$ (the local frame at j) in the coordinates of the local frame at i

$$\begin{aligned}
\mathbf{b}_1^j - \mathbf{b}_1^i &= \alpha_{11}^{ij}\,\mathbf{b}_1^i + \alpha_{12}^{ij}\,\mathbf{b}_2^i + \alpha_{13}^{ij}\,\mathbf{N}^i \\
\mathbf{b}_2^j - \mathbf{b}_2^i &= \alpha_{21}^{ij}\,\mathbf{b}_1^i + \alpha_{22}^{ij}\,\mathbf{b}_2^i + \alpha_{23}^{i}\,\mathbf{N}^i \\
\mathbf{N}^j - \mathbf{N}^i &= \alpha_{31}^{ij}\,\mathbf{b}_1^i + \alpha_{32}^{ij}\,\mathbf{b}_2^i + \alpha_{33}^{ij}\,\mathbf{N}^i
\end{aligned}$$

The coefficients α_{km}^{ij} of this equation can be completely defined by the discrete forms representation. Note that since at each vertex we have three local frames vectors, and the system of equations is separable in the three coordinates, the width of the matrix is $3n$. By adding modeling constraints on the local frames we arrive at a least-squares system whose normal equation matrix is of size $3n \times 3n$. The next step is to reconstruct the actual vertex positions from the local frames. This is done by solving a second linear system of equations, which simply encodes the edge vectors in the coordinates of the local frames

$$\mathbf{v}_j - \mathbf{v}_i = \beta_1^{ij}\,\mathbf{b}_1^i + \beta_2^{ij}\,\mathbf{b}_2^i + \beta_3^{ij}\,\mathbf{N}^i, \qquad \forall (i, j) \in E$$

Here, the right-hand side is known: the coefficients β_k^{ij} are defined by the discrete forms and the local frame vectors were already computed from the previous system. It is possible to add positional modeling constraints to this system and to solve it in the least-squares sense (it is an $n \times n$ system).

The power of the method of Lipman *et al.* [48] lies in the fact that the proposed geometry representation is completely rotation-invariant, and the reconstruction process is linear. Figure 13 shows strong deformations obtained with this method, while preserving the local details of the surface. The price is solving *two* linear systems, and it remains

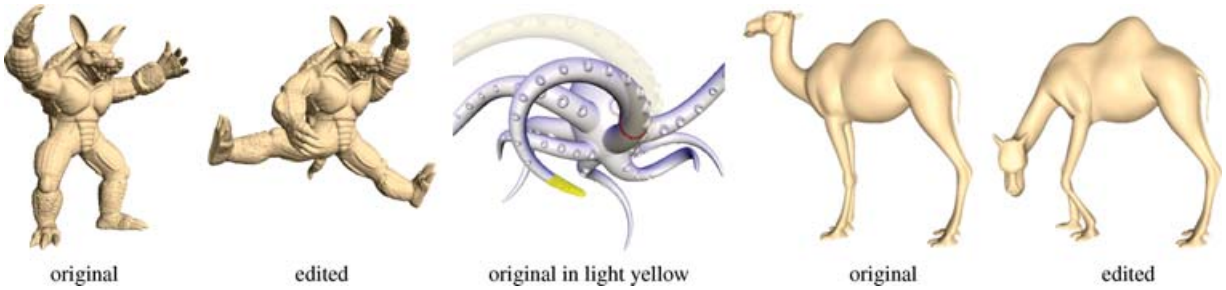

Figure 13: *Some editing results using the rotation-invariant representation of [48]. Note the large rotational deformations achieved and the preservation of local surface details.*

unclear whether it is possible to avoid this. Another drawback is that the manipulation of the frames and the positions is decoupled (as in the Poisson editing method of [47]) and the user must adjust the transformation of the handle frame to the transformation of the handle position. Sole translation of the handle will not affect the local frames of the mesh here, unlike in Laplacian editing, for instance, where the needed local rotations are automatically deduced from the handle translation.

It is important to note that all the above-described surface-based deformation methods do not take the volume of the object into account. As a result, self-intersections might happen during the editing process, and in general the volume of the shape cannot be preserved. The problem of local self-intersections was treated, for example, in [60], where a multiresolution framework was used, with details encoded as volumetric elements rather than displacement vectors. However, rigorously treating volume conservation in this manner leads to time-consuming computations that hinder interactive response. Recently, Zhou *et al.* [61] proposed to augment the Laplacian surface representation with a volumetric graph. More precisely, they place a 3D grid of vertices inside the shape, as well as a layer of additional vertices wrapping the shape on the exterior. These additional vertices are connected between themselves as a standard grid, and also linked to the surface mesh. Zhou *et al.* then apply the Laplacian editing technique to this *volumetric* mesh graph. It is observed that such deformation tends to preserve the shape volume better while retaining local surface detail, at the expense of enhancing the complexity of the representation. Yet, the challenge remains to find a theoretically sound deformation approach that would tie the surface properties with the volumetric ones.

4.4. Shape interpolation using differential representations

For a recent survey on morphing techniques the reader is referred to [62]; below we summarize the newest approaches related to the Laplacian framework and differential representations.

Local differential representations described above allow basic mesh manipulation, as well as advanced editing operations. By mixing and interpolating the differential representations of several meshes, it is possible to blend between different shapes, perform detail transfer from one shape onto another, transplant parts of meshes onto other meshes, etc. Local shape morphing with differential δ-coordinates was first proposed by Alexa [44] (see also the extended paper [38]). When simple linear interpolation of Laplacians is performed, the result is identical to linear interpolation of Cartesian coordinates, which is known for its artifacts [56,63]. However, Alexa proposed to use *varying* interpolation weights, which lead to better results. For example, when attaching one mesh part to another, the weighting can vary proportionally to the distance from the merge region, which leads to a more natural, gradual blend. This idea was further explored in [46], where the "coating transfer" tool was introduced: by "peeling" mesh detail via smoothing, one can transfer the details onto another mesh. The peeled coating is represented by differential coordinates; their orientation is adapted to the orientation of the corresponding local frames on the target mesh, and then they are added to the differential coordinates of the target mesh. Finally, the target mesh with the new coating is reconstructed via Equation (3). Figure 14 shows some coating transfer results and mesh transplanting results from [46].

When interpolating between different shapes (given they have the same connectivity in full correspondence), correct handling of rotations is extremely important. In the simplest case, when the target shape is a rigid transformation of the source shape (say, rotation by 90 degrees about some axis), we expect the blending between the two shapes to be gradual rotation of the source shape towards the target. This does not happen if we linearly blend the Cartesian (or the differential) coordinates of the two shapes. We need a correct interpolation of the *orientation* of the shape. In general, naturally-looking shape interpolation exhibits rigidity and minimizes elastic distortion of the in-between shapes [56].

As mentioned, the Laplacians (or the gradients [47]) are linear functions of the Cartesian coordinates and thus their

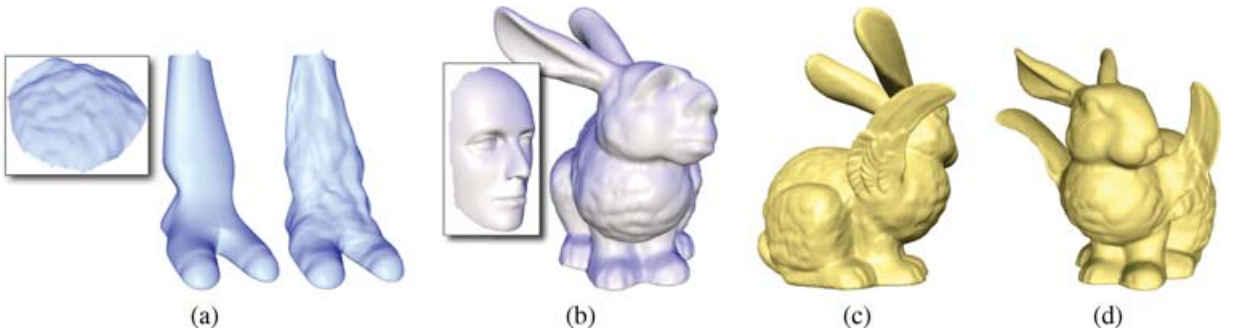

Figure 14: *Examples of shape blending using the Laplacian framework [46]. (a-b) show coating transfer (the sources for coating are framed). In (c-d), the wings of the* Feline *model were transplanted onto the* Bunny.

linear interpolation is equivalent to simple Cartesian interpolation. When a volumetric shape representation is available, Alexa *et al.* [56] proposed to consider the local linear transformation between the elements of the simplicial complexes that describe the two shapes, and interpolate separately between the rotational and the shearing parts of this transformation. This was also employed by Sumner and Popović [57] to transfer local deformations from one mesh animation sequence onto another, and by Xu *et al.* [64] for surface mesh morphing. For each pair of source and target mesh triangles, the affine transformation between their local frames is computed by considering the triangles' edges and normals (this transformation is in fact the so-called deformation gradient, see [57]). This affine transformation H_i is factored using the polar decomposition: $H_i = R_i S_i$, where R_i is a rotation and S_i is symmetric and represents the elastic (scaling) part of H_i. To interpolate between the identity and H_i, R_i is interpolated in the Lie algebra (using e.g., quaternions) and S_i is linearly interpolated. Therefore, for a given interpolation parameter t, for each triangle i we have a local transformation

$$H_i(t) = \text{LIE\_INTERP}(R_i, I, t) \cdot ((1-t)S_i + t\, I). \quad (8)$$

This transformation $H_i(t)$ is applied to the gradients of the source triangle i, and the intermediate mesh is reconstructed from the transformed gradients as in [47]. This approach results in a more natural interpolation of rotations of surface meshes.

A notable limitation of the above approach is its failure to correctly interpolate very large rotations: it is impossible to determine how many laps of rotations an element has gone through by considering solely the source and target state of the element (we will not be able to distinguish between rotation of α and $\alpha + 2\pi k$ for any integer k). The above approach always chooses the "shortest path" interpolation, assuming the minimal angle between each pair of elements. Thus, for example, it is impossible to achieve a natural morphing between a straight line and a spiral in 2D, or a straight bar and its multiply twisted form (as in Figure 15).

Observe that in order to correctly interpolate representations that are *linear* functions of Cartesian geometry, we have to perform *nonlinear* interpolation. This is due to the fact that those representations are not rotation-invariant. The importance of having a linear interpolation method with the possibility of linear reconstruction is evident in the work of Sumner *et al.* [65]. Sumner *et al.* propose a new algorithm for mesh editing, which relies on a space of example edits. This idea enables easy creation of naturally behaving shapes when we have an input data base (e.g., key frames of a manually designed animation or key-frame output of a simulator). Roughly speaking, given a rest mesh M_0 in some representation $F(M_0)$, the example space is a *span* of the deformed states $F(M_1), \ldots, F(M_k)$ of the mesh M_0, again, using the chosen representation. The representation $F(\cdot)$ should be local (at least translation-invariant), such as the Laplacian coordinates or the deformation gradients (which was the choice of Sumner *et al.*). When the user interactively edits the mesh M_0, he/she poses positional modeling constraints of the form (5). The algorithm searches for a shape that (i) satisfies the modeling constraints and (ii) whose representation is as close as possible to the example space. In other words, the algorithm computes a mesh M and a set of scalar weights $w_1, \ldots, w_k$ that minimize the following energy:

$$\left\| F(M) - \text{INTERP}_{w_1,\ldots,w_k} \cdot (F(M_1), \ldots, F(M_k)) \right\|^2 \quad (9)$$

under positional constraints of the form (5). When the example space is very rich (meaning, when a dense set of example edits is available), it is enough to assume that it is a linear space, that is,

$$\text{INTERP}_{w_1,\ldots,w_k}(F(M_1), \ldots, F(M_k)) = \sum_{i=1}^{k} w_i\ F(M_i).$$

Since $F(\cdot)$ is a linear function of the coordinates of M, the global optimization in (9) can be solved linearly. However, when we do not have a dense set of examples, a nonlinear interpolation is required, so that the generated shapes include correct rotations and extrapolate well. Sumner *et al.* [65] employ the interpolation method (8), which turns the

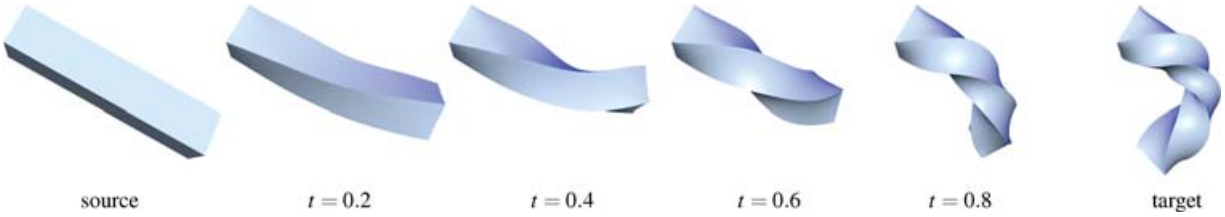

Figure 15: *Linear shape interpolation sequence of the* Bar *mesh using the discrete forms representation [48]. Note the natural rotation of the in-between shapes.*

above problem (9) into a global nonlinear optimization, and thus more expensive.

As mentioned by the authors of [65], a valid alternative for representation of the example space could be the pyramid coordinates [53]. Since they are rotation-invariant, they can be simply linearly interpolated in (9). However, the optimization problem would be still nonlinear because the reconstruction of Cartesian coordinates from pyramid coordinates is not linear ($F(\cdot)$ is not a linear representation in this case).

The discrete forms [48] provide an especially powerful representation for shape interpolation. They are rotation-invariant and can thus be linearly interpolated, and the reconstruction process requires solving two linear systems. In the case of the discrete forms, the representation $F(\cdot)$ is not linear in the Cartesian coordinates, but, figuratively speaking, it can be "factored" so that the reconstruction is done in two *linear* stages. In the context of morphing, this representation can be viewed as an extension of the 2D representation of Sederberg *et al.* [63]: they represented polygonal curves by their edge lengths and angles between successive edges, whereas Lipman *et al.* [48] represent 3D meshes by edge lengths and angles, projected onto the tangent plane, plus the additional height component. Both methods produce natural shape interpolations. Figure 15 demonstrates a morphing sequence from [48]. Since interpolation of discrete forms does not require explicit computation of global rotations between pairs of mesh elements, but rather only considers the *difference* between adjacent local orientations (which contains rotations of up to π radians), it permits correct interpolation of large ("multiple-lap") rotations.

5. Conclusions

In this paper, we have described the Laplacian mesh processing framework and reviewed the recent applications of differential surface representations. The advantages of such surface representations are in their capability for local differential shape description and definition of effective bases for geometry representation. The framework is useful for applications in geometric modeling where surface details are important, especially when processing complex scanned surfaces with abundant detail. The power of the framework stems from its linearity, coupled with the availability of advanced linear solvers. We hope that our paper will help familiarizing researches with this area of study and believe that similar approaches will lead to new useful tools in geometric modeling yet to be discovered.

Acknowledgments

I would like to express my deepest gratitude to my colleagues and co-authors, whose work constitutes the core of this paper: Marc Alexa, Daniel Cohen-Or, Doron Chen, Dror Irony, David Levin, Yaron Lipman, Andrew Nealen, Christian Rössl, Hans-Peter Seidel and Sivan Toledo. I am also grateful to the anonymous reviewers for their extensive help in improving this paper. This work was supported in part by grants from the Israel Science Foundation (founded by the Israel Academy of Sciences and Humanities) and the Israeli Ministry of Science. The author's PhD studies are supported by the Colton Foundation.

References

1. M. Fiedler: Algebraic connectivity of graphs. *Czech. Math. Journal*, 23, 298–305, 1973.

2. F. R. K. Chung: *Spectral Graph Theory*. American Mathematical Society, 1997.

3. M. P. do Carmo: *Differential Geometry of Curves and Surfaces*. Prentice-Hall, 1976.

4. G. Taubin: A signal processing approach to fair surface design. In *Proceedings of ACM SIGGRAPH*, ACM Press, pp. 351–358, 1995.

5. M. Meyer, M. Desbrun, P. Schröder, A. H. Barr: Discrete differential-geometry operators for triangulated 2-manifolds. In *Visualization and Mathematics III*, Hege, H.-C., Polthier, K., (Eds.). Springer-Verlag, Heidelberg, pp. 35–57, 2003.

6. U. Pinkall, K. Polthier: Computing discrete minimal surfaces and their conjugates. *Experiment. Math.*, 2(1): 15–36, 1993.

7. M. S. Floater: Mean value coordinates. *CAGD*, 20(1): 19–27, 2003.

8. M. Botsch, D. Bommes, L. Kobbelt: Efficient linear system solvers for mesh processing. *IMA Mathematics of Surfaces XI, Lecture Notes in Computer Science*, 3604: 62–83, 2005.

9. S. Toledo: *Taucs: A Library of Sparse Linear Solvers, version 2.2*. Tel-Aviv University, Available online at http://www.tau.ac.il/∼stoledo/taucs/, Sept. 2003.

10. B. Aksoylu, A. Khodakovsky, P. Schröder: Multilevel solvers for unstructured surface meshes. *SISC*, 26(4): 1146–1165, 2005.

11. L. Shi, Y. Yu, N. Bell, W.-W. Feng: A fast multigrid algorithm for mesh deformation. In *Proceedings of ACM SIGGRAPH*, To appear, 2006.

12. E. G. Boman, B. Hendrickson: Support theory for preconditioning. *SIAM Journal on Matrix Analysis and Applications*, 25(3): 694–717, 2003.

13. S. Guattery, G. L. Miller: Graph embeddings and Laplacian eigenvalues. *SIAM Journal on Matrix Analysis and Applications*, 21(3): 703–723, 2000.

14. D. Chen, D. Cohen-Or, O. Sorkine, S. Toledo: Algebraic analysis of high-pass quantization. *ACM Transactions on Graphics*, 24(4): 1259–1282, 2005.

15. Z. Karni, C. Gotsman: Spectral compression of mesh geometry. In *Proceedings of ACM SIGGRAPH*, pp. 279–286, 2000.

16. G. Taubin: Geometric signal processing on polygonal meshes. In *Proceedings of Eurographics (STAR volume)*, State of the Art Report, 2000.

17. P. Alliez, C. Gotsman: Recent advances in compression of 3D meshes. In *Advances in Multiresolution for Geometric Modelling*, Dodgson, N., Floater, M., Sabin, M., (Eds.), Springer-Verlag, pp. 3–26, 2005.

18. M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, W. Stuetzle: Multiresolution analysis of arbitrary meshes. In *Proceedings of ACM SIGGRAPH*, pp. 173–182, 1995.

19. D. Zorin, P. Schröder, W. Sweldens: Interactive multiresolution mesh editing. In *Proceedings of ACM SIGGRAPH*, pp. 259–268, 1997.

20. A. W. F. Lee, W. Sweldens, P. Schröder, L. Cowsar, D. Dobkin: MAPS: Multiresolution adaptive parameterization of surfaces. In *Proceedings of ACM SIGGRAPH*, ACM Press, pp. 95–104, 1998.

21. I. Guskov, W. Sweldens, P. Schröder: Multiresolution signal processing for meshes. In *Proceedings of ACM SIGGRAPH*, ACM Press, pp. 325–334, 1999.

22. A. Khodakovsky, P. Schröder, W. Sweldens: Progressive geometry compression. In *Proceedings of ACM SIGGRAPH*, pp. 271–278, 2000.

23. Z. Karni, C. Gotsman: 3D mesh compression using fixed spectral bases. In *Proceedings of Graphics Interface*, Canadian Information Processing Society, pp. 1–8, 2001.

24. O. Sorkine, D. Cohen-Or: Least-squares meshes. In *Proceedings of Shape Modeling International*, IEEE Computer Society Press, pp. 191–199, 2004.

25. O. Sorkine, D. Cohen-Or, D. Irony, S. Toledo: Geometry-aware bases for shape approximation. *IEEE Transactions on Visualization and Computer Graphics*, 11(2): 171–180, 2005.

26. O. Sorkine, D. Cohen-Or, S. Toledo: High-pass quantization for mesh encoding. In *Proceedings of the Eurographics/ACM SIGGRAPH Symposium on Geometry processing*, Eurographics Association, pp. 42–51, 2003.

27. P. Cignoni, C. Rocchini, R. Scopigno: Metro: Measuring error on simplified surfaces. *Computer Graphics Forum*, 17(2): 167–174, 1998.

28. E. Praun, H. Hoppe, A. Finkelstein: Robust mesh watermarking. In *Proceedings of ACM SIGGRAPH*, pp. 49–56, 1999.

29. R. Ohbuchi, S. Takahashi, T. Miyazawa, A. Mukaiyama: Watermarking 3D polygonal meshes in the mesh spectral domain. In *Proceedings of Graphics Interface*, Morgan Kaufmann Publishers, pp. 9–17, (June 2001).

30. R. Ohbuchi, A. Mukaiyama, S. Takahashi: A frequency-domain approach to watermarking 3D shapes. *Computer Graphics Forum (Proceedings of Eurographics)*, 21(3): 373–382, 2002.

31. D. Cotting, T. Weyrich, M. Pauly, M. Gross: Robust watermarking of point-sampled geometry. In *Proceedings of Shape Modeling International*, pp. 233–242, 2004.

32. T. W. Sederberg, S. R. Parry: Free-form deformation of solid geometric models. In *Proceedings of ACM SIGGRAPH*, pp. 151–160, 1986.

33. S. Coquillart: Extended free-form deformation: A sculpturing tool for 3D geometric modeling. In *Proceedings of ACM SIGGRAPH*, 90, pp. 187–196, 1990.

34. M. Ranta, M. Inui, F. Kimura, M. Mäntylä: Cut and paste based modeling with boundary features. In *SMA '93:*

Proceedings of the Second Symposium on Solid Modeling and Applications (May 1993), pp. 303–312.

35. S. Kuriyama, T. Kaneko: Discrete parametrization for deforming arbitrary meshes. In *Proceedings of Graphics Interface*, pp. 132–139, 1999.

36. H. Biermann, I. Martin, F. Bernardini, D. Zorin: Cut-and-paste editing of multiresolution surfaces. In *Proceedings of ACM SIGGRAPH*, pp. 312–321, 2002.

37. T. Kanai, H. Suzuki, J. Mitani, F. Kimura: Interactive mesh fusion based on local 3D metamorphosis. In *Proceedings of Graphics Interface*, pp. 148–156, (June 1999.

38. M. Alexa: Differential coordinates for local mesh morphing and deformation. *The Visual Computer*, 19(2): 105–114, 2003.

39. L. Kobbelt, S. Campagna, J. Vorsatz, H.-P. Seidel: Interactive multi-resolution modeling on arbitrary meshes. In *Proceedings of ACM SIGGRAPH*, ACM Press, pp. 105–114, 1998.

40. D. Forsey, R. Bartels: Hierarchical B-spline refinement. In *Proceedings of ACM SIGGRAPH*, ACM Press, pp. 205–212, 1988.

41. L. Kobbelt, J. Vorsatz, H.-P. Seidel: Multiresolution hierarchies on unstructured triangle meshes. *Computational Geometry: Theory and Applications*, 14: 5–24, 1999.

42. S. Lee: Interactive multiresolution editing of arbitrary meshes. *Computer Graphics Forum (Eurographics 99)*, 18(3): 73–82, 1999.

43. M. Botsch, L. Kobbelt: An intuitive framework for real-time freeform modeling. In *Proceedings of ACM SIGGRAPH*, pp. 630–634, 2004.

44. M. Alexa: Local control for mesh morphing. In *Proceedings of the International Conference on Shape Modeling and Applications (SMI-01)* (Los Alamitos, CA, May 7–11 2001), Werner, B. (Ed.), pp. 209–215.

45. Y. Lipman, O. Sorkine, D. Cohen-Or, D. Levin, C. Rössl, H.-P. Seidel: Differential coordinates for interactive mesh editing. In *Proceedings of Shape Modeling International*, IEEE Computer Society Press, pp. 181–190, 2004.

46. O. Sorkine, Y. Lipman, D. Cohen-Or, M. Alexa, C. Rössl, H.-P. Seidel: Laplacian surface editing. In *Proceedings of the Eurographics/ACM SIGGRAPH Symposium on Geometry processing*, ACM Press, pp. 179–188, 2004.

47. Y. Yu, K. Zhou, D. Xu, X. Shi, H. Bao, B. Guo, H.-Y. Shum: Mesh editing with Poisson-based gradient field manipulation. *ACM Transactions on Graphics* (*Proceedings of ACM SIGGRAPH)*, 23(3): 644–651, 2004.

48. Y. Lipman, O. Sorkine, D. Levin, D. Cohen-Or: Linear rotation-invariant coordinates for meshes. *ACM Transactions on Graphics* (*Proceedings of ACM SIGGRAPH)*, 24(3): 479–487, 2005.

49. R. Zayer, C. Rössl, Z. Karni, H.-P. Seidel: Harmonic guidance for surface deformation. In *Computer Graphics Forum (Proceedings of Eurographics)*, Eurographics, Blackwell, pp. 601–609, 2005.

50. X. Gu, S.-T. Yau: Global conformal surface parameterization. In *Proceedings of the Eurographics/ACM SIGGRAPH Symposium on Geometry processing*, Eurographics Association, pp. 127–137, 2003.

51. X. Ni, M. Garland, J. C. Hart: Fair morse functions for extracting the topological structure of a surface mesh. *ACM Transactions on Graphics* (*Proceedings of ACM SIGGRAPH)*, 23(3): 613–622, 2004.

52. S. Dong, S. Kircher, M. Garland: Harmonic functions for quadrilateral remeshing of arbitrary manifolds. *CAGD*, 22(5): 392–423, 2005.

53. A. Sheffer, V. Kraevoy: Pyramid coordinates for morphing and deformation. In *Proceedings of the Second International Symposium on 3DPVT (3D Data Processing, Visualization, and Transmission*), IEEE Computer Society Press, pp. 68–75, 2004.

54. Y. Lipman, O. Sorkine, M. Alexa, D. Cohen-Or, D. Levin, C. Rössl, H.-P. Seidel: Laplacian framework for interactive mesh editing. *International Journal of Shape Modeling*, 11(1): 43–62, 2005.

55. T. Igarashi, T. Moscovich, J. F. Hughes: As-rigid-as-possible shape manipulation. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH)*, 24(3): 1134–1141, 2005.

56. M. Alexa, D. Cohen-Or, D. Levin: As-rigid-as-possible shape interpolation. In *Proceedings of ACM SIGGRAPH*, pp. 157–164, 2000.

57. R. W. Sumner, J. Popović: Deformation transfer for triangle meshes. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH)*, 23(3): 399–405, 2004.

58. A. Nealen, O. Sorkine, M. Alexa, D. Cohen-Or: A sketch-based interface for detail-preserving mesh editing. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH)*, 24(3): 1142–1147, 2005.

59. D. DeCarlo, A. Finkelstein, S. Rusinkiewicz, A. Santella: Suggestive contours for conveying shape. *ACM Transactions on Graphics (Proccedings of ACM SIGGRAPH)*, 22(3): 848–855, 2003.

60. M. Botsch, L. Kobbelt: Multiresolution surface representation based on displacement volumes. *Computer Graphics Forum*, 22(3): 483–492, 2003.

61. K. Zhou, J. Huang, J. Snyder, X. Liu, H. Bao, B. Guo, H.-Y. Shum: Large mesh deformation using the volumetric graph Laplacian. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH)*, 24(3): 496–503, 2005.

62. M. Alexa: Recent advances in mesh morphing. *Computer Graphics Forum*, 21(2): 173–196, 2002.

63. T. W. Sederberg, P. Gao, G. Wang, H. Mu: 2-D shape blending: an intrinsic solution to the vertex path problem. In *Proceedings of ACM SIGGRAPH*, ACM Press, pp. 15–18, 1993.

64. D. Xu, H. Zhang, Q. Wang, H. Bao: Poisson shape interpolation. In *Proceedings of the ACM Symposium on Solid and Physical Modeling*, ACM Press, pp. 267–274, 2005.

65. R. W. Sumner, M. Zwicker, C. Gotsman, J. Popović: Mesh-based inverse kinematics. *ACM Transactions on Graphics* (*Proceedings of ACM SIGGRAPH)*, 24(3): 488–495, 2005.

About the author

Olga Sorkine received the BSc degree in mathematics and computer science from Tel Aviv University in 2000. Currently, she is a PhD student at the School of Computer Science at Tel Aviv University. Her research interests are in computer graphics and include shape modeling, mesh processing and approximation. She has worked in the area of using Laplacian transforms for mesh processing and will submit a PhD thesis on this topic.