Mesh Editing based on Discrete Laplace and Poisson Models

Marc Alexa
Faculty of EE & CS
TU Berlin

Abstract

Surface editing operations commonly require geometric details of the surface to be preserved as much as possible. We argue that geometric detail is an intrinsic property of a surface and that, consequently, surface editing is best performed by operating over an intrinsic surface representation. This intrinsic representation could be derived from differential properties of the mesh, i.e. its Laplacian. The modeling process poses nonzero boundary constraints so that this idea results in a Poisson model. Different ways of representing the intrinsic geometry and the boundary constraints result in alternatives for the properties of the modeling system. In particular, the Laplacian is not invariant to scaling and rotations. Either the intrinsic representation is enhanced to be invariant to (linearized) transformations, or scaling and rotation are computed in a preprocess and are modeled as boundary constraints. Based on this representation, useful editing operations can be developed: Interactive free-form deformation in a region of interest based on the transformation of a handle, transfer and mixing of geometric detail between two surfaces, and transplanting of a partial surface mesh into another surface. The main computation involved in all operations is the solution of a sparse linear system, which can be done at interactive rates. We demonstrate the effectiveness of this approach in several examples, showing that the editing operations change the shape while respecting the structural geometric detail.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—curve, surface, solid and object representations

1 Introduction

Surfaces in computer graphics are mostly represented in global coordinate systems: explicit representations are based on points, vertices, or nodes that are typically described using absolute Euclidean coordinates. Implicit representations describe the shape as the level set of a function defined in Euclidean space. A global coordinate system is the natural choice for all operations involving other objects such as rendering, intersection testing and computation, transformations, or CSG modeling. On the other hand, for local surface modeling, it would be desirable that the representation captures the local shape (i.e. the intrinsic geometry of the surface) rather than the absolute position or orientation in Euclidean space.

Manipulating and modifying a surface while preserving the geometric details is important for various surface editing operations, including free-form deformations [Sederberg and Parry 1986; Coquillart 1990], cut and paste [Ranta et al. 1993; Biermann et al. 2002], fusion [Kanai et al. 1999], morphing [Alexa 2003a], and others. Note that the absolute position of the vertices in a mesh is not important for these operations, which calls for an intrinsic surface representation.

A partially intrinsic surface mesh representation are multi-resolution decompositions [Forsey and Bartels 1988; Zorin et al. 1997; Kobbelt et al. 1998; Kobbelt et al. 1999; Guskov et al. 1999]. In a multi-resolution mesh, the geometry is encoded as a base mesh and several levels of refinement. The refinement is typically described locally, so that geometric details are mostly captured in a discrete set of intrinsic coordinates. Using this representation, several modeling operations can be performed on an appropriate user-specified level-of-detail. Note, however, that the locality of multi-resolution representations is potentially limited: The support (or extent) of the representation of a single vertex increases from fine to coarse levels of the hierarchy. Thus, modeling operations are restricted to a discrete set of regions and levels-of-detail. For example, when cutting out a partial mesh for transplanting operations, the original multi-resolution representation is invalidated because parts of the base domain and (potentially) other levels of the hierarchy are missing.

This approach to encode geometric details is to use differentials as coordinates for the vertices [Alexa 2003b; Botsch and Kobbelt 2004; Sorkine et al. 2004; Yu et al. 2004]. This provides a fully intrinsic representation of the surface mesh, where the reconstruction of global coordinates from the intrinsic representation always preserves the intrinsic geometry as much as possible given the modeling constraints. Using a differential representation for editing operations has been shown to be quite effective in image domain [Fattal et al. 2002; Pérez et al. 2003]. The image domain has a natural regular parameterization and resulting inherent definition of a gradient, which allows modeling many editing tasks as a discrete Poisson equation. However, this approach cannot be directly applied or adapted to discrete (as well as continuous) surfaces.

2 The Laplacian representation

Let the mesh $\mathscr{M}$ be described by a pair (K,V), where K is a simplicial complex representing the connectivity of vertices, edges, and faces, and $V = \{\mathbf{v}_1,\ldots,\mathbf{v}_n\}$ describes the geometric positions of the vertices in $\mathbb{R}^3$. We use the following terminology: the *neighborhood ring* of a vertex i is the set of adjacent vertices $\mathscr{N}_i = \{j|(i,j) \in K\}$ and the *degree* d_i of this vertex is the number adjacent edges (or vertices), i.e. the number of elements in $\mathscr{N}_i$. We assume that $d_i > 0$, i.e. that the mesh is connected.

Instead of using absolute coordinates V, we would like to describe the mesh geometry using a set of differentials $\Delta = \{\delta_i\}$. Specifically, coordinate i will be represented by its Laplace vector. There are different ways to define a discretized version of the Laplace operator for meshes, and each of them has certain advantages. Most of them are based on the one-ring of a vertex

$$\delta_i = \mathbf{v}_i - c_{ij} \sum_{j\in\mathscr{N}_i} \mathbf{v}_j, \quad \sum_j c_{ij} = 1, \tag{1}$$

and differ in the definition of the coefficients c_{ij}. The topological Laplacian ([Taubin 1995]) simply uses the similar weights for all neighboring vertices, i.e. $c_{ij} = 1/d_i$. In our and others' experience the the cotangent weights (e.g. [Meyer et al. 2003]) perform best in most applications.

The transformation between V and Δ can be described in matrix algebra. Let $C = \{c_{ij}\}$, then

$$\Delta = (I - C)V. \tag{2}$$

The Laplacian $L = I - C$ is invariant under translation, however, sensitive to linear transformations. Thus, L is expected to have rank

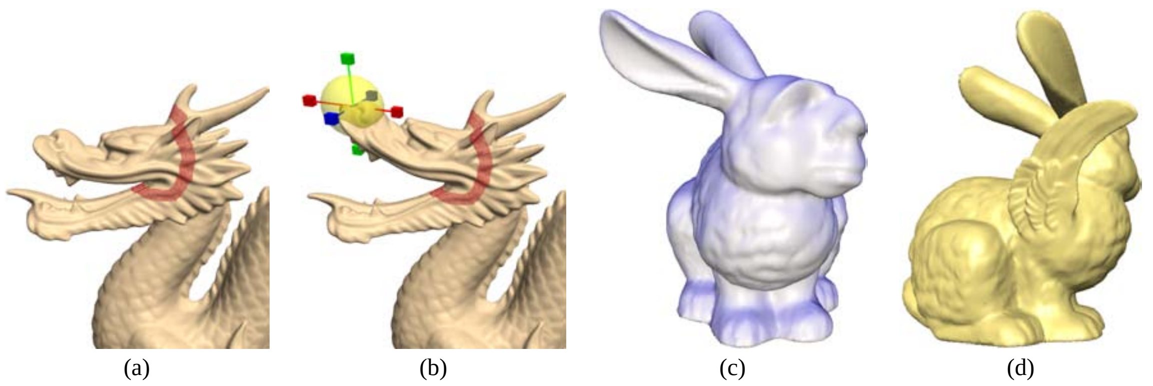

Figure 1: Advanced mesh editing operations using Laplacian coordinates: free-form deformations (a-b), detail transfer (c) and mesh transplanting (d). Representing the geometry using the Laplacian coordinates enables preservation of detail.

$n-1$, which means V can be recovered from Δ by fixing one vertex and solving a linear system of equations.

3 Mesh modeling framework

The basic idea of the modeling framework is to satisfy linear modeling constraints (exactly, or in the least squares sense), while preserving differential properties of the original geometry in the least squares sense [Alexa 2003b; Lipman et al. 2004]. Without additional linear constraints the deformed geometry V' is then defined by

$$\min_{V'} \sum_{i=1}^{n} \left\| \delta_i - \left(\mathbf{v}'_i - \frac{1}{d_i} \sum_{j \in \mathcal{N}_i} \mathbf{v}'_j \right) \right\|^2. \tag{3}$$

If the original surface was a membrane, the necessary constraints for the minimizer lead to $L^2 V = 0$, which has been advocated by Botsch and Kobbelt [Botsch and Kobbelt 2004] in the context of modeling smooth surfaces. If, in contrast, the original surface contained some detail, the right-hand side is non-zero and we arrive at a variant of the discrete Poisson modeling approach of Yu et al. [Yu et al. 2004].

The basic type of linear modeling constraints is to prescribe the absolute position of some vertices, i.e. $\mathbf{v}'_i = \hat{\mathbf{v}}_i$. These constraints are best incorporated by also satisfying them in the least squares sense, possibly weighted to trade-off between modeling constraints and the reproduction of original surface geometry.

We found that the easiest way of implementing the approach is to write the conditions to be satisfied in the least squares sense as a large rectangular system $\mathbf{AV}' = \mathbf{b}$ and then solve $\mathbf{A}^T\mathbf{AV}' = \mathbf{A}^T\mathbf{b}$. Prescribing positions for some vertices then simply yields additional rows of the form

$$w_i \| \mathbf{v}'_i = \hat{\mathbf{v}}_i. \tag{4}$$

Note that in fact these are three rows for each constraint, as $\mathbf{v}$ are column vectors with three elements.

This framework can be extended towards constraints on arbitrary points on the mesh. Note that each point on the surface is the linear combination of two or three vertices. A point on an edge between vertices i and j is defined by one parameter as $(1-\lambda)\mathbf{v}_i + \lambda \mathbf{v}_j$, $0 \le \lambda \le 1$. Similarly, a point on a triangle is defined by two parameters. We can put positional constraints $\hat{\mathbf{v}}_{ij}$ on such a point by adding rows of the form

$$(1-\lambda)v'_i + \lambda v'_j = \hat{\mathbf{v}}_{ij} \tag{5}$$

to the system matrix A.

Furthermore, also differentials could be prescribed. Note that δ_i points roughly in normal direction at vertex i and that its length is proportional to the mean curvature. This allows us to prescribe a certain normal direction and/or curvature for a vertex, simply by adding a row of the form

$$\mathbf{v}'_i - \sum_{j \in \mathcal{N}_i} c_{ij} \mathbf{v}'_j = \hat{\delta}_i. \tag{6}$$

The modeling operation is typically localized on a part of the mesh. This part of the mesh is selected by the user as the region of interest (ROI) during the interactive modeling session. The operations are restricted to this ROI, padded by several layers of anchor vertices. The anchor vertices yield positional constraints $\mathbf{v}'_i = \hat{\mathbf{v}}_i$ in the system matrix A, which ensure a gentle transition between the altered ROI and the fixed part of the mesh.

Based on the constraints formulated so far, local surface detail is preserved if parts of the surface are translated, but changes with rotations and scales. There are several ways of dealing with linear transformations:

- They could be defined and prescribed based on the modeling operation [Yu et al. 2004].
- They could be deduced from the membrane solution (i.e. $LV' = 0$) [Lipman et al. 2004].
- They could be implicitly defined by the solution, if the rotational part is linearized [Sorkine et al. 2004].

In any case, we first need to extend the definition of the local intrinsic representation to incorporate linear transformations.

4 Incorporating linear transformations

The main idea to account for local linear transformations is to assign each vertex i an individual transformation T_i. These transformation are then applied to the original geometry by a transforming each local Laplacian δ_i with T_i. This results in a slightly modified functional defining the resulting geometry V':

$$\min_{V'} \sum_{i=1}^{n} \left\| T_i \delta_i - \left(\mathbf{v}'_i - \frac{1}{d_i} \sum_{j \in \mathcal{N}_i} \mathbf{v}'_j \right) \right\|^2 \tag{7}$$

Note that in the formulation of this minimization as solving a system $AV' = b$ the part $T_i\delta_i$ is contained in the right-hand side column vector b. This is important because it implies the system A can be

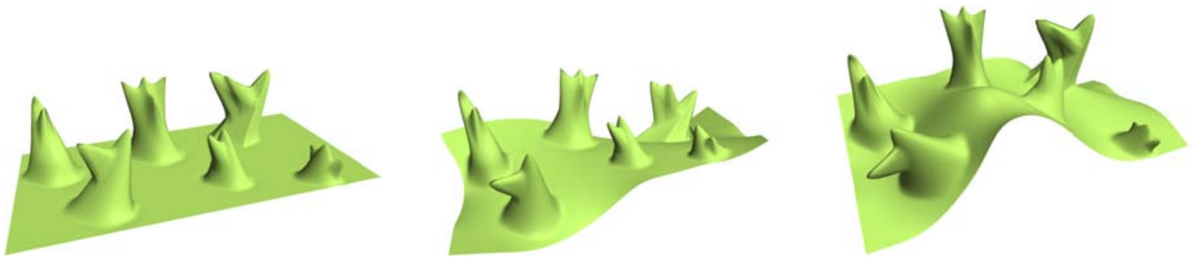
(a) (b) (c)

Figure 2: Deformations of a model (a) with detail that cannot be expressed by height field. The deformation changes the global shape while respecting the structural detail as much as possible.

solved independent of the transformations T_i to be applied to vertex i, allowing the T_i to be changed during interactive modeling.

The following approaches vary in how the local transformations T_i are computed.

4.1 Prescribing the transformations

Yu et al. [Yu et al. 2004] let the user specify a few constraint transformations and then interpolate them over the surface. In particular, the rotational and scaling parts are treated independently, i.e. the transformation is factored as $T_i = R_i S_i$, where R_i is the local rotation and S_i is a symmetric matrix containing scale and shear. Initially all vertices are assumed to be not rotated, scaled or sheared. Modeling operations might induce local linear transformations.

One could view this (slightly more general as in [Yu et al. 2004]) as a scattered data interpolation problem: In few vertices a (non-zero) rotation or non-unity scale are given. All vertices should then be assigned a scale and rotation so that the given constraints are satisfied and the field of rotations and scales is smooth. In order to apply well-known techniques only a distance measure for the vertices is necessary. Yu et al. [Yu et al. 2004] use the topological distance of vertices in the mesh.

Then, each local rotation and scale are a distance-weighted average of given transformations. The easiest way to derive the distance weights would be Shephard's approach. This defines T_i for each vertex and, thus, V'. Note that transformations can be changed interactively.

4.2 Transformations from the membrane solution

Lipman et al. [Lipman et al. 2004] compute the rotations from the membrane solution. They first solve $\Delta V' = 0$ and then compute each transformation T_i based on comparing the one-rings in $\mathbf{V}$ and $\mathbf{V}'$ of vertex i.

The basic idea for a definition of T_i is to derive it from the transformation of $\mathbf{v}_i$ and its neighbors to $\mathbf{v}'_i$ and its neighbors:

$$\min_{T_i} \left(\|T_i \mathbf{v}_i - \mathbf{v}'_i\|^2 + \sum_{j \in \mathcal{N}_i} \|T_i \mathbf{v}_j - \mathbf{v}'_j\|^2 \right). \tag{8}$$

This is a quadratic expression, so the minimizer is a linear function of V'.

Note that this is not significantly slower than computing the solution for the initial local identity transformations: The system matrix A has to be factored once, from the first solution all T_i are computed, b is modified accordingly, and the final positions V' are computed using back-substitution.

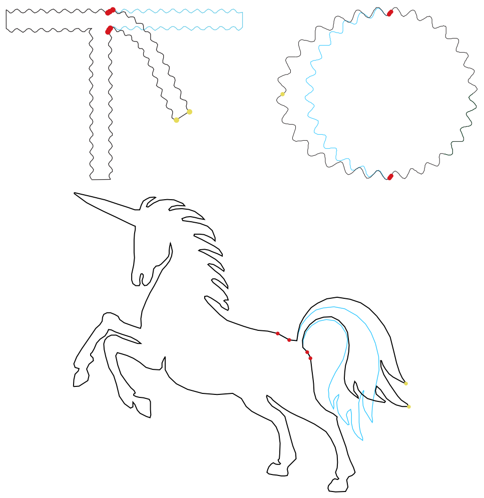

Figure 3: Editing 2D meshes using Laplacian-coordinates fitting. The red dots denote fixed anchor points and the yellow are the pulled handle vertices. The original meshes are colored blue.

4.3 Linearized implicit transformations

The main idea of [Sorkine et al. 2004] is to compute an appropriate transformation T_i for each vertex i based on the eventual new configuration of vertices V'. Thus, $T_i(V')$ is a function of V'.

Note that in Eq. 7 both the T_i and the V' are unknown. However, if the coefficients of T_i are a linear function in V', then solving for V' implies finding T_i (though not explicitly) since Eq. 7 is still a quadratic function in V'. If we define T_i as in Eq. 8, it is a linear function in V', as required.

However, if T_i is unconstrained, the natural minimizer is a membrane solution, and all geometric detail is lost. Thus, T_i needs to be constrained in a reasonable way. We have found that T_i should include rotations, isotropic scales, and translations. In particular, we want to disallow anisotropic scales (or shears), as they would allow removing the normal component from Laplacian representation.

The transformation should be a linear function in the target configuration but constrained to isotropic scales and rotations. The class of matrices representing isotropic scales and rotation can be written as $T = s\exp(H)$, where H is a skew-symmetric matrix. In 3D, skew symmetric matrices emulate a cross product with a vector, i.e. $H\mathbf{x} = \mathbf{h} \times \mathbf{x}$. Drawing upon several other properties of 3×3 skew matrices (see Appendix A), one can derive the following representation of the exponential above:

$$s\exp H = s(\alpha I + \beta H + \gamma \mathbf{h}^T \mathbf{h}) \tag{9}$$

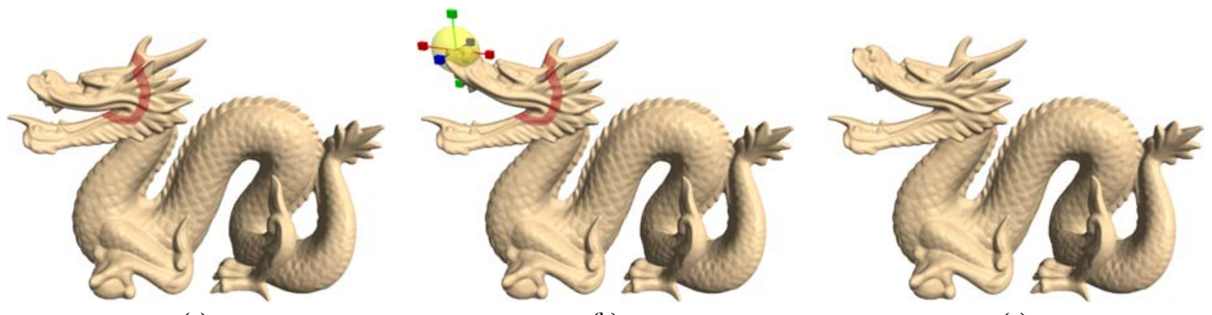
(a) (b) (c)

Figure 4: The editing process. (a) The user selects the region of interest – the upper lip of the dragon, bounded by the belt of stationary anchors (in red). (b) The chosen handle (enclosed by the yellow sphere) is manipulated by the user: translated and rotated. (c) The editing result.

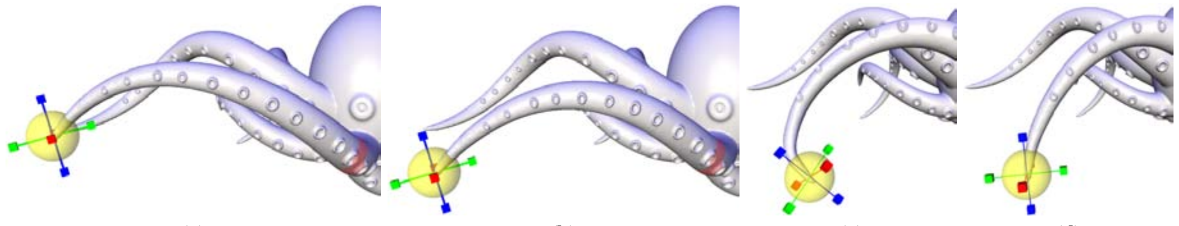
(a) (b) (c) (d)

Figure 5: Different handle manipulations. (a) The region of interest (arm), bounded by the belt of stationary anchors, and the handle. (b) Translation of the handle. (c), (d) Rotation of the handle. Note that the detail is preserved in all the manipulations.

Inspecting the terms we find that only s, I, and H are linear in the unknowns s and $\mathbf{h}$, while $\mathbf{h}^T\mathbf{h}$ is quadratic[1]. As a linear approximation of the class of constrained transformations we, therefore, use

$$T_i = \begin{pmatrix} s & h_1 & -h_2 & t_x \\ -h_1 & s & h_3 & t_y \\ h_2 & -h_3 & s & t_y \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{10}$$

This matrix is a good linear approximation for rotations with small angles.

Given the matrix T_i as in Eq. 10, we can write down the linear dependency (cf. Eq. 8) of T_i on V' explicitly. Let $(s_i, \mathbf{h}_i, \mathbf{t}_i)^T$ be the vector of the unknowns in T_i, then we wish to minimize

$$\|A_i(s_i, \mathbf{h}_i, \mathbf{t}_i)^T - \mathbf{b}_i\|^2, \tag{11}$$

where A_i contains the positions of $\mathbf{v}_i$ and its neighbors and $\mathbf{b}_i$ contains the position of $\mathbf{v}_i'$ and its neighbors. The structure of $(s_i, \mathbf{h}_i, \mathbf{t}_i)^T$ yields

$$A_i = \begin{pmatrix} v_{k_x} & v_{k_y} & -v_{k_z} & 0 & 1 & 0 & 0 \\ v_{k_y} & -v_{k_x} & 0 & v_{k_z} & 0 & 1 & 0 \\ v_{k_z} & 0 & v_{k_x} & -v_{k_y} & 0 & 0 & 1 \\ \vdots & & & & & & \end{pmatrix}, k \in \{i\} \cup \mathcal{N}_i, \tag{12}$$

[1]Figure 3 illustrates editing of a 2D mesh. Note that in 2D the matrices of class $s\exp(H)$ can be completely characterized with the linear expression

$$T_i = \begin{pmatrix} a & w & t_x \\ -w & a & t_y \\ 0 & 0 & 1 \end{pmatrix}.$$

and

$$\mathbf{b_i} = \begin{pmatrix} v'_{k_x} \\ v'_{k_y} \\ v'_{k_z} \\ \vdots \end{pmatrix}, k \in \{i\} \cup \mathcal{N}_i. \tag{13}$$

The linear least squares problem above is solved by

$$(s_i, \mathbf{h}_i, \mathbf{t}_i)^T = \left(A_i^T A_i\right)^{-1} A_i^T \mathbf{b}_i, \tag{14}$$

which shows that the coefficients of T_i are linear functions of $\mathbf{b}_i$, since A_i is known from the initial mesh V. The entries of $\mathbf{b}_i$ are simply entries of V' so that $(s_i, \mathbf{h}_i, \mathbf{t}_i)$ and, thus, T_i is a linear function in V', as required.

4.4 Adjusting T_i

In many modeling situations solving for absolute coordinates in the way explained above is sufficient. However, there are exceptions that might require adjusting the transformations.

A good way of updating transformations for all three mentioned approaches is this: The current set of transformations $\{T_i\}$ is computed from V and V'. Then each T_i is inspected, the corresponding Laplacian coordinate δ_i is updated appropriately depending on the effect to be achieved, and the system is solved again. For example, if anisotropic scaling has been suppressed but is wanted, the $\{\delta_i\}$ are scaled by the inverse of the anisotropic scale implied by the constraints.

5 Mesh Editing

There are many different tools to manipulate an existing mesh. Perhaps the simplest form consists of manipulating a *handle*, which is a

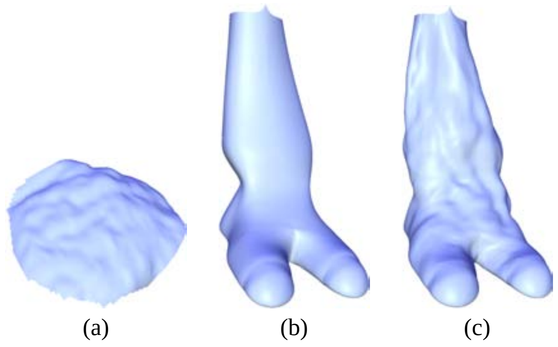

Figure 6: Detail transfer; The details of the Bunny (a) are transferred onto the mammal's leg (b) to yield (c).

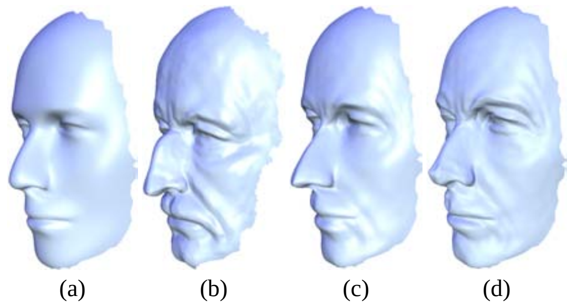

Figure 7: The details of the *Max Planck* are transferred onto the *Mannequin*. Different levels of smoothing were applied to the *Max Planck* model to peel the details, yielding the results in (c) and (d).

set of vertices that can be moved, rotated and scaled by the user. The manipulation of the handle is propagated to the shape such that the modification is intuitive and resembles the outcome of manipulating an object made of some physical soft material. This can be generalized to a free-form deformation tool which transforms a small set of control points defining a complex of possibly weighted handles, enabling mimicking other modeling metaphors (see e.g., [Bendels and Klein 2003] and the references therein).

The editing interaction is comprised of the following stages: First, the user defines the region of interest (ROI) for editing. Next, the handle is defined. In addition, the user can optionally define the amount of "padding" of the ROI by *stationary anchors*. These stationary anchors form a *belt* that supports the transition between the ROI and the untouched part of the mesh. Then, the user manipulates the handle, and the surface is reconstructed with respect to the relocation of the handle and displayed.

The submesh of the ROI is the only part considered during the editing process. The positions of the handle vertices and the stationary anchors constrain the reconstruction and hence the shape of the resulting surface. The handle is the means of user control, therefore its constraints are constantly updated. The unconstrained vertices of the submesh are repeatedly reconstructed to follow the user interaction. The stationary anchors are responsible for the transition from the ROI to the fixed untouched part of the mesh, resulting in a soft transition between the submesh and stationary part of the mesh. Selecting the amount of these padding anchor vertices depends on the user's requirements, as mentioned above. We have observed in all our experiments that setting the radius of the "padding ring" to be up to 10% of the ROI radius gives satisfying results.

The reconstruction of the submesh requires solving linear least-squares system as described in Section 2. The method of building the system matrix (Eq. 14), including the computation of a sparse factorization, is relatively slow, but constructed only once when the ROI is selected. The user interaction with the handle requires solely updating the positions of the handle vertices in the right-hand-side vector, and solve.

Figures 4 and 5 illustrate the editing process. Note that the details on the surface are preserved, as one would intuitively expect. Figure 2 demonstrates deformation of a model with large extruding features which cannot be represented by a height field.

6 Detail Transfer

Detail transfer is the process of peeling the coating of a *source* surface and transferring it onto a *target* surface. See Figure 6 for an example of such operation.

Let S be the source surface from which we would like to extract the details, and let $\tilde{S}$ be a smooth version of S. The surface $\tilde{S}$ is a low-frequency surface associated with S, which can be generated by filtering [Desbrun et al. 1999; Fleishman et al. 2003]. The amount of smoothing is a user-defined parameter, and it depends on the range of detail that the user wishes to transfer.

We encode the details of a surface based on the Laplacian representation. Let δ_i and $\tilde{\delta}_i$ be the Laplacian coordinates of the vertex i in S and $\tilde{S}$, respectively. We define ξ_i to be the encoding of the detail at vertex i defined by

$$\xi_i = \delta_i - \tilde{\delta}_i \,. \tag{15}$$

The values of ξ_j encode the details of S, since given the bare surface $\tilde{S}$ we can recover the original details simply by adding ξ_j to $\tilde{\delta}_i$ and reconstructing S with the inverse Laplacian transform L^{-1}. That is,

$$S = L^{-1}(\tilde{\delta} + \xi) \,. \tag{16}$$

In this case of a detail transfer of S onto itself, S is faithfully reconstructed. However, in general, instead of coating $\tilde{S}$ with ξ, we would like to add the details ξ onto an arbitrary surface U. If the target surface U is not smooth, it can be smoothed first, and then the detail transfer is applied. In the following we assume that the target surface U is smooth. Before we move on, we should note that the detail transfer from S onto $\tilde{S}$ is simple, since the neighborhoods of the corresponding vertices i have the same *orientation*. We define the orientation of a vertex i in a surface S by the normal direction of i over $\tilde{S}$. Loosely speaking, the orientation of a point reflects the general orientation of its neighborhood, without respecting the high-frequencies of the surface.

When applying a detail transfer between two surfaces, the detail ξ should be first aligned, or rotated with respect to the target. This compensates for the different local surface orientations of corresponding points in the source and target surfaces.

The following is an important property of the Laplacian coordinates:

$$R \cdot L^{-1}(\delta_j) = L^{-1}(R \cdot \delta_j) \,, \tag{17}$$

where L^{-1} is the transformation from Laplacian coordinates to absolute coordinates, and R a *global* rotation applied to the entire mesh. The mapping between corresponding points in S and U defines different local orientations across the surfaces. Thus, our key idea is to use the above property of the Laplacian coordinates locally, assuming that locally the rotations are similar.

Assume that the source surface S and the target surface U share the same connectivity, but different geometry, and that the correspondence between their vertices is given. In the following we generalize this to arbitrary surfaces.

The local rotation R_i at each vertex i in S and U is taken to be the local rotation between their corresponding orientations. Let $\mathbf{n}_s$ and $\mathbf{n}_u$ be the normals associated with the orientations of i in S and

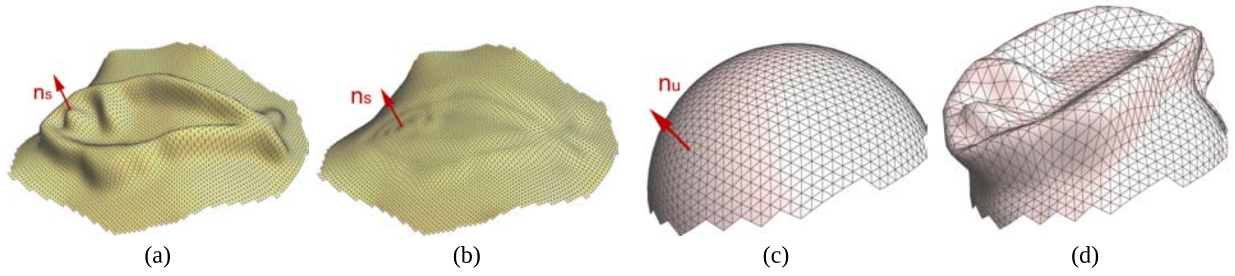

Figure 8: Detail transfer. The orientation of details (a) are defined by the normal at the corresponding vertex in the low frequency surface in (b). The transferred detail vector needs to be rotated to match the orientation of the corresponding point in (c) to reconstruct (d).

U, respectively. We define the rotation operator R_i by setting the axis of rotation as $\mathbf{n}_s \times \mathbf{n}_u$ and requiring $\mathbf{n}_u = R_i(n_s)$. Denote the rotated detail encoding of vertex i by $\xi_i' = R_i(\xi_i)$. Having all the R_i associated with the ξ_i, the detail transfer from S onto U is expressed as follows:

$$U' = L^{-1}(\Delta + \xi') \tag{18}$$

where Δ denotes the Laplacian coordinates of the vertices of U. Now the new surface U' has the details of U.

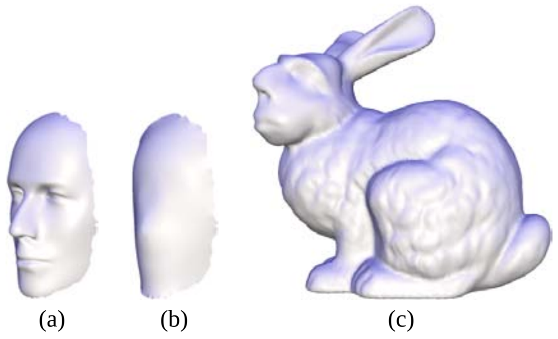

Figure 9: Transferring the details of the *Mannequin* onto the face of the *Bunny*. (a) The source surface S. It is significantly smoothed to peel the details. (b) The smoothed surface $\tilde{S}$. (c) The result of detail transfer onto the *Bunny*.

6.1 Mapping and Resampling

So far we assumed that the source and target meshes (S and U) share the same connectivity, and hence the correspondence is readily given. However, detail transfer between arbitrary surfaces is more involved. To sample the Laplacian coordinates, we need to define a mapping between the two surfaces.

This mapping is established by parameterizing the meshes over a common domain. Both patches are assumed to be homeomorphic to a disk, so we may chose either the unit circle or the unit square as common domain. We apply the mean-value coordinates parameterization [Floater 2003], as it efficiently produces a quasi-conformal mapping, which is guaranteed to be valid for convex domains. We fix the boundary conditions for the parameterization such that a correspondence between the source and target surfaces is achieved, i.e. we identify corresponding boundary vertices and fix them at the same domain points. In practice, this is a single vertex in S and in U that constrains rotation for the unit circle domain, or four boundary vertices for the unit square domain.

Some applications require a more careful correspondence than what can be achieved from choosing boundary conditions. For example, the mapping between two faces (see Figure 7) should link

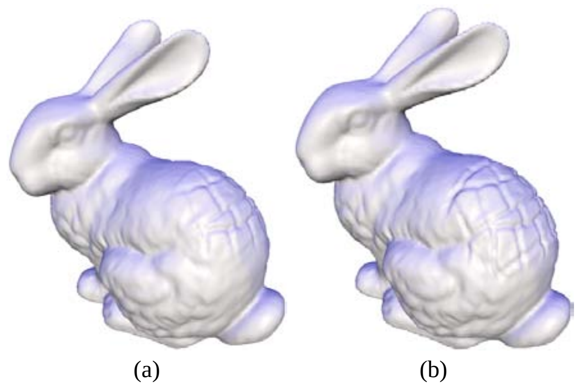

Figure 11: Transplanting of *Armadillo*'s details onto the *Bunny* back with a soft transition (a) and a sharp transition (b) between the two types of details. The size of the transition area in which the Laplacians are blended is large in (a) and small in (b).

relevant details like facial features such as the brow wrinkles of the *Max Planck*. In this case the user provides a few additional (inner) point-to-point constraints which define a warp of the mean-value parameterization. In our implementation we use a radial basis function elastic warp, but any reasonable warping function can do.

In general, a vertex $i \in U$ is mapped to some arbitrary point inside a triangle $\tau \in S$. We experimented with several methods for sampling the Laplacian for a vertex. The best results are obtained by first mapping the 1-ring of i onto S using the parameterization, and then computing the Laplacian from this mapped 1-ring. Note that this approach assumes a locally similar distortion in the mapping. This is usually the case for the detail transfer; we used the 1-ring sampling in all the respective examples. We obtain similar results by linear interpolation of the three Laplacian coordinates sampled at the vertices of the triangle τ. While this approach leads to some more "blurring" compared to the first one, it is even simpler and does not suffer from extremely different parametric distortion. In addition, no special treatment is required at the boundary of the domain in case the patch was initially cut to be homeomorphic to a disk.

After the mapping between U and S has been established and the Laplacians have been sampled, the detail transfer proceeds as explained before. Note that now the corresponding ξ_i is the difference between the *sampled* Laplacian coordinates in S and $\tilde{S}$. See the examples in Figures 6, 7 and 9.

6.2 Mixing Details

Given two meshes with the same connectivity and different details, the above transfer mechanism can be applied on a third target mesh

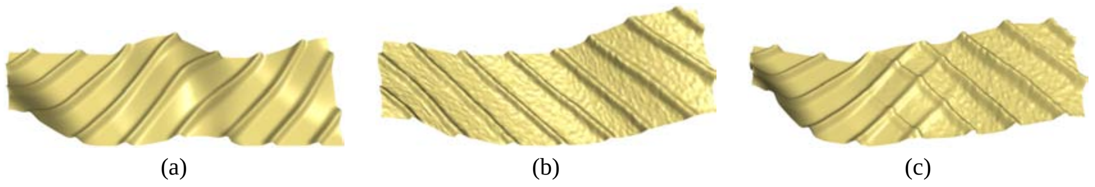

Figure 10: Mixing details using Laplacian coordinates. The Laplacian coordinates of surfaces in (a) and (b) are linearly blended in the middle to yield the shape in (c).

from the two sources. Figure 10 illustrates the effect of blending the details. This example emphasizes the mixing of details, as the details of the two source meshes differ in the smoothness, form and orientation. Note that the details are gradually mixed and the global shape of the target mesh is deformed respectively. By adding anchor points over the target, its shape can be further deformed. Figure 11 shows the application of this mechanism to transplant *Armadillo*'s details onto the *Bunny*'s back with a soft transition. In the next section we further discuss this transplanting operation.

7 Transplanting surface patches

In the previous sections we showed how the Laplacian coordinates allow to transfer the details of surface onto another and how to gradually mix details of two surfaces. These techniques are refined to allow a seamless transplanting of one shape onto another. The transplanting operation consists of two apparently independent classes of operations: topological and geometrical. The topological operation creates one consistent triangulation from the connectivities of the two submeshes. The geometrical operation creates a gradual change of the geometrical structure of one shape into the other. The latter operation is based on the Laplacian coordinates and the reconstruction mechanism.

Let S denote the mesh that is transplanted onto a surface U. See Figure 12, where the right wing (S) of the *Feline* is transplanted onto the *Bunny* (U). The transplanting requires the user to first register the two parts in world coordinates. This defines the desired location and orientation of the transplanted shape, as well as its scale.

The user selects a region $U_\circ \subset U$ onto which S will be transplanted. In the rest of the process we only work with $U_\circ$ and do not alter the rest of U. $U_\circ$ is cut such that the remaining boundary is homeomorphic to the boundary of S. We simply project the boundary of S onto $U_\circ$. The two boundary loops are zipped, thus creating the connectivity of the resulting mesh D (Figure 12(a)).

The remaining transplanting algorithm is similar to detail transfer and mixing. The user specifies a region of interest on D, vertices outside the ROI remain fixed.

Next, the respective *transitional regions* $S' \subset S$ and $U' \subset U_\circ$ are selected starting from the cut boundaries on S and $U_\circ$. Since $S' \subset D$, this implicitly defines the transitional region $D' \subset D$ along with a trivial mapping between vertices of S' and D'.

For sampling, we require an additional correspondence between S' and U', hence we parameterize both meshes over the unit square. The user guides this construction by cutting S' and U' such that both meshes are homeomorphic to a disk. The cuts enable the mapping to the common domain, and in addition they serve as intuitive means to align the mappings such that there is a correspondence between the patches. In our experiments no further warping was necessary to improve the correspondence (cf. Section 6.1).

Once the transitional regions and the mappings are defined, the transplanting procedure is ready to sample the Laplacian coordinates of S' and U' over D'. The corresponding Laplacian coordinates are linearly blended with weights defined by their relative position in the unit square parameter domain. More precisely, if $v \in [0,1]$ defines the coordinate along the "height" axis (the blue and red lines in Figure 12(b), then the weights are v and $(1-v)$, respectively. Since the length distortion of the maps may significantly differ, we linearly interpolate the Laplacian coordinates for sampling (cf. Section 6.1). The remainder of the ROI is sampled over D, and the reconstruction respects the belt of anchors which is placed to pad the boundaries of the ROI. Figures 12(c),(d) show the result.

8 Implementation details

All the techniques presented in this paper are implemented and tested on a Pentium 4 2.0 GHz computer. The main computational core of the surface reconstruction algorithm is solving a sparse linear least-squares problem. We use a direct solver which first computes a sparse triangular factorization of the normal equations and then finds the minimizer by back-substitution. As mentioned in Section 5, constructing the matrix of the least-squares system and factorizing it takes the bulk of the computation time. This might seem as a heavy operation for such an application as interactive mesh editing; however, it is done only once per ROI selection. The solve by back-substitution is quite fast and enables to reconstruct the surface interactively, following the user's manipulations of the handle. It should be noted that the system is comprised only of the vertices that fall into the ROI; thus the complexity is not directly dependent on the size of the entire mesh, but rather on the size of the ROI. We experimented with various ROIs of sizes in the order of tens of thousands of vertices. The "intermediate preprocess" times observed were a few seconds, while the actual editing process runs at interactive framerates. Some short editing sessions are demonstrated in the accompanying video.

9 Conclusions

Intrinsic geometry representation for meshes fosters several local surface editing operations. Geometry is essentially encoded using differential properties of the surface, so that the local shape (or, surface detail) is preserved as much as possible given the constraints posed by the user. We show how to use this representation for interactive free-form deformations, detail transfer or mixing, and transplanting partial surface meshes.

It is interesting to compare the Laplacian-based approach to multi-resolution approaches: Because each vertex is represented individually as a Laplacian coordinate, the user can freely choose the editing region and model arbitrary boundary constraints, however, computing absolute coordinates requires the solution of a linear system. On the other hand, the non-local bases in multi-resolution representations limit the choice of the editing region and boundary constraints, but absolute coordinates are computed much simpler and faster by summing displacements through the hierarchy. Additionally, we would like to mention that we have found the Laplacian approach to be easier to implement and less brittle in practice.

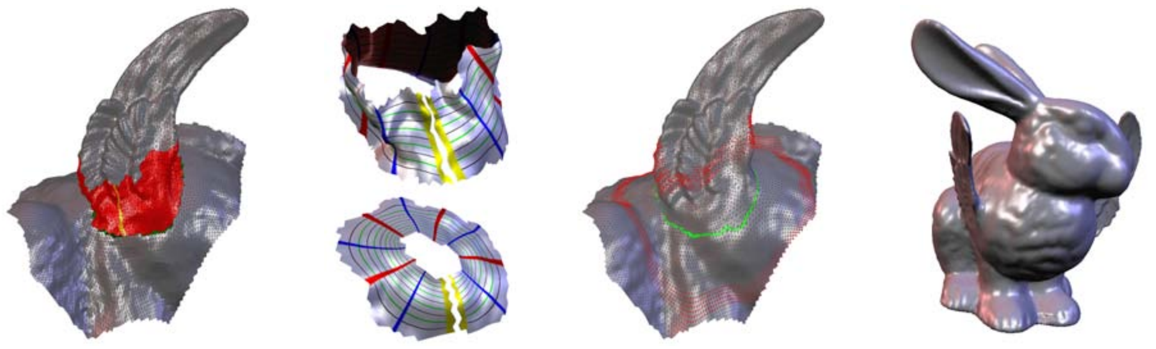

Figure 12: Transplanting of *Feline*'s wings onto the *Bunny*. (a) After cutting the parts and fixing the desired pose, the zipping (in green) defines the target connectivity D. The transitional region D' is marked red. Additional cut in D' (in yellow) enables mapping onto a square. (b) D' is sampled over the respective regions $U' \subset U_\circ$ ($U_\circ$ is the cut part of the *Bunny*'s back) and S' (the bottom of the wing). The texture with *uv*-isolines visualizes the mapping over the unit square. The cut (in yellow) aligns the two maps. (c) The result of reconstruction. The ROI is padded by a belt of anchors (in red). Note the change of the zipping seam triangles (green) and the details within the transition region. (d) The flying *Bunny* (see also Figure 1(d)).

In general, modeling geometry should be coupled to modeling other surface properties, such as textures. The machinery of discrete Poisson equations has already shown to be effective for image editing, so that editing textured surface should probably be performed on a combined differential geometry/texture representation.

Acknowledgment

Models are courtesy of Stanford University and Max-Planck-Institut für Informatik.

References

ALEXA, M. 2003. Differential coordinates for local mesh morphing and deformation. *The Visual Computer 19*, 2, 105–114.

ALEXA, M. 2003. Differential coordinates for local mesh morphing and deformation. *The Visual Computer 19*, 2, 105–114.

BENDELS, G. H., AND KLEIN, R. 2003. Mesh forging: editing of 3d-meshes using implicitly defined occluders. In *Proceedings of the Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, 207–217.

BIERMANN, H., MARTIN, I., BERNARDINI, F., AND ZORIN, D. 2002. Cut-and-paste editing of multiresolution surfaces. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, 312–321.

BOTSCH, M., AND KOBBELT, L. 2004. An intuitive framework for real-time freeform modeling. *ACM Trans. Graph. 23*, 3, 630–634.

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

DESBRUN, M., MEYER, M., SCHRÖDER, P., AND BARR, A. H. 1999. Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proceedings of ACM SIGGRAPH 99*, 317–324.

FATTAL, R., LISCHINSKI, D., AND WERMAN, M. 2002. Gradient domain high dynamic range compression. In *Proceedings of ACM SIGGRAPH 2002*, 249–256.

FLEISHMAN, S., DRORI, I., AND COHEN-OR, D. 2003. Bilateral mesh denoising. In *Proceedings of ACM SIGGRAPH 2003*, 950–953.

FLOATER, M. S. 2003. Mean-value coordinates. *Computer Aided Geometric Design 20*, 19–27.

FORSEY, D., AND BARTELS, R. 1988. Hierarchical b-spline refinement. In *Proceedings of ACM SIGGRAPH 88*, 205–212.

GUSKOV, I., SWELDENS, W., AND SCHRÖDER, P. 1999. Multiresolution signal processing for meshes. In *Proceedings of ACM SIGGRAPH 99*, 325–334.

KANAI, T., SUZUKI, H., MITANI, J., AND KIMURA, F. 1999. Interactive mesh fusion based on local 3D metamorphosis. In *Graphics Interface '99*, 148–156.

KOBBELT, L., CAMPAGNA, S., VORSATZ, J., AND SEIDEL, H.-P. 1998. Interactive multi-resolution modeling on arbitrary meshes. In *Proceedings of ACM SIGGRAPH 98*, 105–114.

KOBBELT, L., VORSATZ, J., AND SEIDEL, H.-P. 1999. Multiresolution hierarchies on unstructured triangle meshes. *Computational Geometry: Theory and Applications 14*, 5–24.

LIPMAN, Y., SORKINE, O., COHEN-OR, D., AND LEVIN, D. 2004. Differential coordinates for interactive mesh editing. In *International Conference on Shape Modeling and Applications 2004 (SMI'04)*, 181–190.

MEYER, M., DESBRUN, M., SCHRDER, P., AND BARR, A. H. 2003. Discrete differential-geometry operators for triangulated 2-manifolds. *Visualization and Mathematics III*, pages 35–57.

PÉREZ, P., GANGNET, M., AND BLAKE, A. 2003. Poisson image editing. In *Proceedings of ACM SIGGRAPH 2003*, 313–318.

RANTA, M., INUI, M., KIMURA, F., AND MÄNTYLÄ, M. 1993. Cut and paste based modeling with boundary features. In *SMA '93: Proceedings of the Second Symposium on Solid Modeling and Applications*, 303–312.

SEDERBERG, T. W., AND PARRY, S. R. 1986. Free-form deformation of solid geometric models. In *Proceedings of SIGGRAPH 86*, 151–160.

SORKINE, O., LIPMAN, Y., COHEN-OR, D., ALEXA, M., RÖSSL, C., AND SEIDEL, H.-P. 2004. Laplacian surface editing. In *Proceedings of the Eurographics/ACM SIGGRAPH Symposium on Geometry processing*, Eurographics Association, 179–188.

TAUBIN, G. 1995. A signal processing approach to fair surface design. In *Proceedings of ACM SIGGRAPH 95*, 351–358.

YU, Y., ZHOU, K., XU, D., SHI, X., BAO, H., GUO, B., AND SHUM, H.-Y. 2004. Mesh editing with poisson-based gradient field manipulation. *ACM Trans. Graph. 23*, 3, 644–651.

ZORIN, D., SCHRDER, P., AND SWELDENS, W. 1997. Interactive multiresolution mesh editing. In *Proceedings of ACM SIGGRAPH 97*, 259–268.

A Exponential of a 3×3 skew symmetric matrix

Let $\mathbf{h} \in \mathbb{R}^3$ be a vector and $H \in \mathbb{R}^{3\times3}$ be a skew symmetric matrix so that $H\mathbf{x} = \mathbf{h} \times \mathbf{x}, \forall \mathbf{x} \in \mathbb{R}^3$. We are interested in expressing the exponential of H in terms of the coefficients of H, i.e. the elements of $\mathbf{h}$. The matrix exponential is computed using the series expansion

$$\exp H = I + \frac{1}{1!}H + \frac{1}{2!}H^2 + \frac{1}{3!}H^3 + \dots$$

The powers of skew symmetric matrices in three dimensions have particularly simple forms. For the square we find

$$H^2 = \begin{pmatrix} -h_2^2 - h_3^2 & h_1h_2 & h_1h_3 \\ h_1h_2 & -h_1^2 - h_3^2 & h_2h_3 \\ h_1h_3 & h_2h_3 & -h_1^2 - h_2^2 \end{pmatrix} = \mathbf{h}\mathbf{h}^T - \mathbf{h}^T\mathbf{h}I$$

and using this expression (together with the simple fact that $H\mathbf{h} = 0$) it follows by induction that

$$H^{2n} = (-\mathbf{h}^T\mathbf{h})^{n-1}\mathbf{h}\mathbf{h}^T + (-\mathbf{h}^T\mathbf{h})^n I$$

and

$$H^{2n-1} = (-\mathbf{h}^T\mathbf{h})^{n-1}H$$

for $n \in \mathbb{N}$. Thus, all powers of H can be expressed as linear combinations of I, H, and $\mathbf{h}\mathbf{h}^T$, and, therefore,

$$\exp H = \alpha I + \beta H + \gamma \mathbf{h}\mathbf{h}^T$$

for appropriate factors α, β, γ.

B Implementation Details

For ease of re-implementation, we explicitly give the rows of the basic system matrix $\mathbf{A}$. The main complication results from the rotations, which are linearized and computed from the displacements of one-rings.

We focus on one vertex $\mathbf{v}_0 = (v_{0_x}, v_{0_y}, v_{0_z})$ and its Laplacian $\delta_0 = (\delta_{0_x}, \delta_{0_y}, \delta_{0_z})$, yielding three rows in the system matrix. The transformation T adjusting δ_0 minimizes the squared distances between corresponding vertices $\mathbf{V} = (\mathbf{v}_1, \mathbf{v}_2, \dots)$ in the one-ring of $\mathbf{v}$:

$$T = \begin{pmatrix} s & -h_3 & h_2 \\ h_3 & s & -h_1 \\ -h_2 & h_1 & s \end{pmatrix} \tag{19}$$

The coefficients are linear expression in the displaced coordinates $\mathbf{V}'$ (see [Sorkine et al. 2004] for details on how to derive the coefficients)

$$\begin{aligned} s &= \sum_i s_{i_x} v'_{i_x} + s_{i_y} v'_{i_y} + s_{i_z} v'_{i_z} \\ &= \mathbf{s}_x \mathbf{v}'_x + \mathbf{s}_y \mathbf{v}'_y + \mathbf{s}_z \mathbf{v}'_z, \end{aligned} \tag{20}$$

where the abbreviations $\mathbf{v}'_{\{x,y,z\}}$ are the rows of $\mathbf{V}'$. Similar computations lead to the linear expressions for h_1, h_2, h_3 and coefficient vectors $\mathbf{h}$.

Now we can plug these expressions into the matrix T and multiply with δ_0 to find

$$T\delta_0 = \begin{pmatrix} \sum_{k\in\{x,y,z\}} \left(\delta_{0_x}\mathbf{s}_k - \delta_{0_y}\mathbf{h}_{3_k} + \delta_{0_z}\mathbf{h}_{2_k})\mathbf{v}'_k\right) \\ \sum_{k\in\{x,y,z\}} \left(\delta_{0_x}\mathbf{h}_{3_k} + \delta_{0_y}\mathbf{s}_k - \delta_{0_z}\mathbf{h}_{1_k})\mathbf{v}'_k\right) \\ \sum_{k\in\{x,y,z\}} \left(-\delta_{0_x}\mathbf{h}_{2_k} + \delta_{0_y}\mathbf{h}_{1_k} + \delta_{0_z}\mathbf{s}_k)\mathbf{v}'_k\right) \end{pmatrix} \tag{21}$$

The constraint $T\delta_0 = \delta'_0 = \mathbf{v}'_0 - \sum_i w_i \mathbf{v}'_i$ results in three rows of the system $AV = b$ of the form

$$\begin{aligned} &\sum_{k\in\{x,y,z\}} \left(\delta_{0_x}\mathbf{s}_k - \delta_{0_y}\mathbf{h}_{3_k} + \delta_{0_z}\mathbf{h}_{2_k})\mathbf{v}'_k\right) - \mathbf{v}'_{0_x}(1, w_1, w_2, \dots) = 0 \\ &\sum_{k\in\{x,y,z\}} \left(\delta_{0_x}\mathbf{s}_k - \delta_{0_y}\mathbf{h}_{3_k} + \delta_{0_z}\mathbf{h}_{2_k})\mathbf{v}'_k\right) - \mathbf{v}'_{0_y}(1, w_1, w_2, \dots) = 0 \\ &\sum_{k\in\{x,y,z\}} \left(\delta_{0_x}\mathbf{s}_k - \delta_{0_y}\mathbf{h}_{3_k} + \delta_{0_z}\mathbf{h}_{2_k})\mathbf{v}'_k\right) - \mathbf{v}'_{0_z}(1, w_1, w_2, \dots) = 0 \end{aligned} \tag{22}$$

or explicitly for, e.g., x:

$$\begin{aligned} &\mathbf{v}'_x \left((1, w_1, w_2, \dots) - \delta_{0_x}\mathbf{s}_x - \delta_{0_y}\mathbf{h}_{3_x} + \delta_{0_z}\mathbf{h}_{2_x}\right) + \\ &\mathbf{v}'_y \left(\delta_{0_x}\mathbf{s}_y - \delta_{0_y}\mathbf{h}_{3_y} + \delta_{0_z}\mathbf{h}_{2_y}\right) + \\ &\mathbf{v}'_z \left(\delta_{0_x}\mathbf{s}_z - \delta_{0_y}\mathbf{h}_{3_z} + \delta_{0_z}\mathbf{h}_{2_z}\right) = 0. \end{aligned} \tag{23}$$

We see that the basic system matrix essentially contains three block copies of the Laplace matrix on the main diagonal, one for each coordinate direction. The additional coefficients in the off-diagonal blocks link the coordinate directions to accommodate rotations.