Volume xx (200y), Number z, pp. 1–12

Effective Derivation of Similarity Transformations for Implicit Laplacian Mesh Editing

Hongbo Fu† Oscar Kin-Chung Au‡ Chiew-Lan Tai§

Department of Computer Science and Engineering
The Hong Kong University of Science and Technology

Abstract

Laplacian coordinates as a local shape descriptor have been employed in mesh editing. As they are encoded in the global coordinate system, they need to be transformed locally to reflect the changed local features of the deformed surface. We present a novel implicit Laplacian editing framework which is linear and effectively captures local rotation information during editing. Directly representing rotation with respect to vertex positions in 3D space leads to a nonlinear system. Instead, we first compute the affine transformations implicitly defined for all the Laplacian coordinates by solving a large sparse linear system, and then extract the rotation and uniform scaling information from each solved affine transformation. Unlike existing differential-based mesh editing techniques, our method produces visually pleasing deformation results under large angle rotations or big-scale translations of handles. Additionally, to demonstrate the advantage of our editing framework, we introduce a new intuitive editing technique, called configuration-independent merging, which produces the same merging result independent of the relative position, orientation, scale of input meshes.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling Boundary representations

1. Introduction

Recently, Laplacian coordinates have been proposed for mesh editing due to their ability to capture surface details intrinsically. The resulting editing tools [SLCO*04, ZRKS05] allow the user to simply manipulate parts of a mesh, called the handles, and the rest of the surface is reconstructed by minimizing the Laplacian coordinates before and after editing. As the Laplacian coordinates are defined in the global coordinate system, they are neither scale invariant nor rotation invariant [LSCO*04]. Therefore the main challenge of Laplacian editing frameworks is to find appropriate local transformations such that the transformed Laplacian coordinates fit the orientations of the details in the deformed surface.

Existing solutions either explicitly define the transformations at all the unconstrained vertices by interpolating the user-specified transformations at the constrained vertices [YZX*04, ZRKS05, ZHS*05] or implicitly define the transformations with respect to the (unknown) deformed surface [LSCO*04, SLCO*04, NSACO05]. However, these methods only partially solved the problem: explicit methods cannot infer the rotation information if the handles are only translated, while implicit methods cannot tackle large angle deformation.

In this paper, we present a novel implicit Laplacian editing framework, which is linear and able to effectively capture the local rotation transformations for the local features during editing. Like [SLCO*04], the ultimate goal of our framework is to find a similarity transformation consisting of a rigid transformation and a uniform scaling. To achieve interactive editing, like most previous Laplacian editing frameworks, we aim to formulate a linear reconstruction problem. It is well known that rigid transformations are nonlinearly dependent on vertex positions in 3D space. Therefore, our solution is to use affine transformations, which linearly de-

† e-mail: fuhb@cs.ust.hk
‡ e-mail: oscarau@cs.ust.hk
§ e-mail: taicl@cs.ust.hk

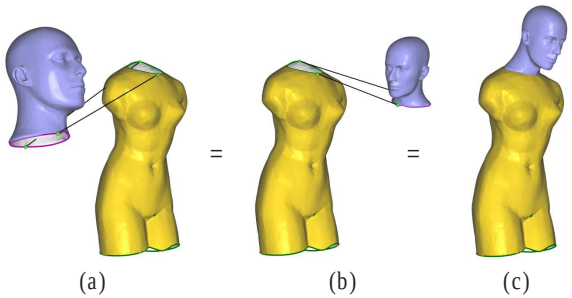

Figure 1: *Configuration-independent merging. The goal is to merge the Mannequin head model (source) to the Venus model (target). The user only specifies the correspondence between the merging boundaries. In (a) and (b), the Mannequin head model has different positions, orientations and scales. Our configuration-independent merging method produces the same result (c), given the same boundary correspondence. The lines indicate two user-specified key correspondences.*

pend on vertex positions. By enforcing neighborhood coherence and user-specified boundary constraints, we solve for the affine transformations in the least-squares sense. The downside of using affine transformations is the introduction of shearing distortion. To remove shearing distortion, we perform polar decomposition to extract from each of the solved affine transformations a similarity transformation consisting of only rotation and uniform scaling.

By binding each Laplacian coordinate with the corresponding similarity transformation, our method appropriately orients and scales the local features during editing. In other words, the changes of the local features are implicitly captured by the similarity transformations. Our framework produces more visually pleasing editing results than previous related work for deformation with large angle rotations and/or big-scale translations of handles.

Additionally, we present a new merging paradigm, called *configuration-independent merging*, based on our Laplacian framework. By configuration we refer to the relative position, orientation and scale of meshes. Most related previous merging techniques require the user to adjust the configuration of input meshes prior to merging [SLCO*04, BKZ01, MBWB02]. Thanks to the implicitly defined transformations, our merging approach can eliminate this user interaction requirement. Our method computes the transformations (corresponding to configuration adjustment in existing methods) and solves for the merged mesh simultaneously (Figure 1). The user only specifies the merging boundaries and indicates several key vertex correspondences between the boundaries.

2. Previous Work

2.1. Differential-Based Editing

Recently, intrinsic differential representations (the Laplacian coordinates [LSCO*04, SLCO*04, ZRKS05], the gradient field [YZX*04] and the first/second fundamental surface forms [LSLCO05]) have been adopted to mesh editing because the resulting editing tools are intuitive and detail-preserving. Using these tools, the user can interactively edit a region of interest (ROI) by manipulating a small set of handles. As our framework is based on the Laplacian coordinates, this discussion focuses on previous Laplacian-based work.

The Laplacian coordinate (LC) δ_i at vertex $\mathbf{v}_i$ ($1 \le i \le n$) is defined as follows [LSCO*04, SLCO*04]:

$$\delta_i = D(\mathbf{v}_i) = \sum_{j \in N(i)} w_{ij}(\mathbf{v}_j - \mathbf{v}_i),$$

where $N(i)$ is the index set of the 1-ring neighboring vertices of $\mathbf{v}_i$ and w_{ij} is the weight of the edge (i, j). The resulting LC is essentially a 3D vector in the global coordinate system. Two common weighting schemes are uniform weighting ($w_{ij} = 1/|N(i)|$) [LSCO*04, SLCO*04] and cotangent weighting ($w_{ij} = \frac{1}{|\Omega_i|}(\cot\alpha_i + \cot\beta_i)$) [MDSB02], where $|\Omega_i|$ is the area of the Voronoi cell associated with $\mathbf{v}_i$, and α_i, β_i are the two angles opposite to the edge (i, j).

The underlying theory of Laplacian editing can be formulated as the following linear system (subject to the user-specified boundary condition)

$$D(\tilde{\mathbf{v}}_i) = \mathbf{M}_i\delta_i, \quad 1 \le i \le n, \tag{1}$$

where $\tilde{\mathbf{v}}_i$ is an unconstrained vertex (to be solved) and $\mathbf{M}_i$ is a 3×3 transformation matrix. To avoid having the LCs deviate from the normal directions, $\mathbf{M}_i$ is mostly required to be a combination of rotation and uniform scaling [SLCO*04]. The method of Poisson mesh editing [YZX*04] also results in a linear system similar to Equation 1, since in theory the gradient operator followed by the divergence operator is equivalent to the Laplace operator.

Depending on whether $\mathbf{M}_i$ is defined with respect to the (unknown) deformed surface, existing differential based mesh editing can be classified as implicit methods [LSCO*04, SLCO*04] or explicit methods [YZX*04, ZRKS05, ZHS*05]. Explicit methods define $\mathbf{M}_i$ without considering the deformed surface. Instead, $\mathbf{M}_i$ is defined by propagating the transformations at the handles to all the unconstrained vertices, weighted by geodesic distances [YZX*04, ZHS*05] or a set of harmonic fields [ZRKS05]. These methods can produce good deformation when the handles undergo large-angle rotations, including rotation angles greater than 2π. However, if the handles are only translated, there is no change of orientation to be propagated; thus these approaches cannot avoid shearing distortion caused by handle translation (see an example in Figure 7).

Implicit methods define $\mathbf{M}_i$ with respect to the deformed surface. This is essentially a chicken-and-egg problem. On the one hand, the deformed surface is to be reconstructed from the transformed LCs; on the other hand, the transformations are dependent on the resulting deformed surface. Lipman et al. [LSCO*04] proposed a heuristic method: they first reconstruct a rough surface using the original LCs and use the reconstructed surface to estimate a local rotation for each vertex. Sorkine et al. [SLCO*04] approximately represent $\mathbf{M}_i$ as a function of the unknown vertex positions. Existing implicit methods work well for small translations and/or small rotation angles of handles. However, they do not produce visually pleasing results when handles undergo large angle rotations (Figure 6) or big-scale translations (Figure 7). Our implicit method can handle these types of large deformation. Our framework has some similarity to Lipman et al. [LSLCO05], which appeared after the initial submission of this paper [FT05] (see more detailed comparison in Section 6).

Recently, Sheffer and Krayevoy [SK04] proposed a representation called pyramid coordinates to encode mesh details. Their method produces more natural editing results than all the existing differential based methods, including ours, at the cost of expensive computation. Sumner et al. [SZGP05] proposed an example-based deformation technique. Satisfying the specified vertex constraints, the deformed surface is reconstructed from interpolated feature vectors provided by examples meshes.

2.2. Mesh Merging and Surface Pasting

Mesh merging and surface pasting produce new models by composing existing models. Biermann et al. [BMBZ02] proposed a cut-and-paste editing technique that allows the user to decide the degree of details of the source mesh to be pasted onto the target mesh. Lévy [LÓ3] proposed a merging method by extrapolating parameterizations. Both methods involve parameterizing the regions to be merged or pasted, and thus are applicable only to regions homeomorphic to a disk.

The merging methods of Yu et al. [YZX*04] and Sorkine et al. [SLCO*04] connect two meshes at their open boundaries without 2D parameterization. Therefore, these methods only require the merging boundaries to have the same topology. In [SLCO*04], the merging operation first fills the gap between two boundaries and then mixes the details by surface reconstruction. In [YZX*04], the two boundaries are first deformed to an intermediate boundary, and the deformation is propagated from the deformed boundaries to the interior of the meshes. The smoothness along the merging boundary is improved by Poisson normal smoothing. Both methods require the user to adjust the configuration of the meshes to be merged.

3. Implicit Laplacian Editing Framework

This section introduces our implicit Laplacian editing framework. We implicitly define local transformations in terms of the (known) original vertex positions and (unknown) deformed vertex positions. To avoid having the LCs deviate from the normal directions, thus suppressing shearing distortion, these transformations are required to be rigid [SLCO*04]. However, rigid transformations in 3D space nonlinearly depend on the vertex positions. To attain linearity in the reconstruction in Equation 1, we therefore adopt affine transformations. Since simply representing affine transformations in terms of the unknown vertex positions makes the resulting system (Equation 1) underconstrained, we enforce neighborhood coherence to make the reconstruction problem well-posed. The shearing distortion accompanying affine transformations will be removed in the second step described in the next section.

3.1. Implicitly Defined Local Deformation Gradients

Like [SLCO*04, SP04], we define an affine transformation for each vertex using that vertex and its neighbors as follows:

$$\mathbf{M}_i\mathbf{v}_k + \mathbf{d}_i = \tilde{\mathbf{v}}_k, \qquad k \in \{i\} \cup N(i), \tag{2}$$

where $\mathbf{M}_i$ is a 3×3 matrix, $\mathbf{d}_i$ is the translation vector, and $\tilde{\mathbf{v}}_k$ is the unknown vertex. $\mathbf{M}_i$ and $\mathbf{d}_i$ together define an affine transformation at $\mathbf{v}_i$. Since the LCs to which the defined affine transformations will be applied are local difference of vertex positions, what we really care for is only $\mathbf{M}_i$, called the *deformation gradient* [SZGP05].

By eliminating the translation vector $\mathbf{d}_i$, we rewrite Equation 2 in matrix form as

$$\mathbf{M}_i\mathbf{V}_i = \tilde{\mathbf{V}}_i \tag{3}$$

where

$$\begin{matrix}\mathbf{V}_i = [\mathbf{v}_i - \mathbf{v}_{i_0} & \mathbf{v}_{j_1} - \mathbf{v}_{i_0} & \cdots & \mathbf{v}_{j_{|N(i)|}} - \mathbf{v}_{i_0}] \\ \tilde{\mathbf{V}}_i = [\tilde{\mathbf{v}}_i - \tilde{\mathbf{v}}_{i_0} & \tilde{\mathbf{v}}_{j_1} - \tilde{\mathbf{v}}_{i_0} & \cdots & \tilde{\mathbf{v}}_{j_{|N(i)|}} - \tilde{\mathbf{v}}_{i_0}]\end{matrix}, \quad j_* \in N(i)$$

with

$$\mathbf{v}_{i_0} = \frac{1}{|N(i)|}\sum_{j\in N(i)}\mathbf{v}_j \quad \text{and} \quad \tilde{\mathbf{v}}_{i_0} = \frac{1}{|N(i)|}\sum_{j\in N(i)}\tilde{\mathbf{v}}_j.$$

To get a least-squares solution for $\mathbf{M}_i$ in Equation 3, we consider the following equations

$$\mathbf{M}_i\mathbf{V}_i\mathbf{V}_i^{\mathbf{T}} = \tilde{\mathbf{V}}_i\mathbf{V}_i^{\mathbf{T}}.$$

If $\mathbf{V}_i\mathbf{V}_i^{\mathbf{T}}$ is invertible, we can directly derive $\mathbf{M}_i$ as

$$\mathbf{M}_i = \tilde{\mathbf{V}}_i\mathbf{V}_i^{\mathbf{T}}(\mathbf{V}_i\mathbf{V}_i^{\mathbf{T}})^{-1}. \tag{4}$$

The above expression is similar to the derived deformation gradient in [SP04]. The deformation gradient in their work always has a closed form expression because the vertices used to define the deformation gradient form a basis of 3D space, whereas we need to handle the special case when $\mathbf{V}_i\mathbf{V}_i^{\mathbf{T}}$ degenerates to a singular matrix (Section 3.3).

3.2. Neighborhood Coherence

We first assume that $\mathbf{M}_i$ is well defined for each unconstrained vertex; the degenerate case will be discussed in the next subsection. By applying $\mathbf{M}_i$ to the corresponding LC δ_i, we wish to reconstruct the editing vertex positions by minimizing the following error functional

$$E_L = \sum_{i=1}^{n} \|D(\tilde{\mathbf{v}}_i) - \mathbf{M}_i\delta_i\|^2, \tag{5}$$

However, this reconstruction problem by itself is underconstrained [SLCO*04].

To make the reconstruction problem well-posed, we introduce a neighborhood coherence term to regularize the implicitly defined deformation gradients. Similar to [ACP03, SP04], we require the deformation gradients applied within a surface region to be as similar as possible. Specifically, besides E_L, we also minimize the following error functional

$$E_R = \sum_{i=1}^{n} \sum_{(i,j)\in SF} \|\mathbf{M}_i - \mathbf{M}_j\|_F^2, \tag{6}$$

where SF is the set of pairs of neighboring vertex indices and $\|\cdot\|_F$ is the Frobenius norm. Since the deformation gradient defined at each vertex is used to deform the features locally (by transforming the corresponding LC), the neighborhood coherence term essentially minimizes the difference in deformation at neighboring vertices.

We now prove that given the position and the deformation gradient at an arbitrary mesh vertex $\mathbf{v}_0$, the minimization problem in Equation 5 with the neighborhood coherence constraint $\|\mathbf{M}_i - \mathbf{M}_j\|_F = 0$ is well-posed. First, we prove that with the neighborhood coherence constraint and the given deformation gradient, all the deformation gradients at the unconstrained vertices can be uniquely determined. As $\|\mathbf{M}_i - \mathbf{M}_j\|_F = 0$ implies $\mathbf{M}_i = \mathbf{M}_j$, the transformations at vertices adjacent to $\mathbf{v}_0$ must be the same as the given deformation gradient at $\mathbf{v}_0$. With the same argument, all the deformation gradients at the unconstrained vertices can be determined, and are equal to the given deformation gradient at $\mathbf{v}_0$. Second, we prove that, with all the computed deformation gradients and the given position constraint at $\mathbf{v}_0$, the positions of all the unconstrained vertices can be uniquely determined. After $\mathbf{M}_i$ is computed, the optimization in Equation 5 is equivalent to solving a simple Poisson system, which has a unique and exact solution given one position constraint [SLCO*04].

When more than one position and more than one deformation gradient are constrained, exact solution is not guaranteed. The unique solution is solved by minimizing $E_L + E_R$ in the least squares sense. Formally, the final optimization is formulated as follows

$$\begin{aligned} &\arg\min E(\tilde{\mathbf{v}}_1, \ldots, \tilde{\mathbf{v}}_n) = E_L + E_R \\ \text{subject to} \quad &\tilde{\mathbf{v}}_{b_j} = \mathbf{u}_j, \qquad j \in \{1 \ldots s\} \\ &\mathbf{M}_{c_k} = \mathbf{W}_k, \qquad k \in \{1 \ldots t\} \end{aligned} \tag{7}$$

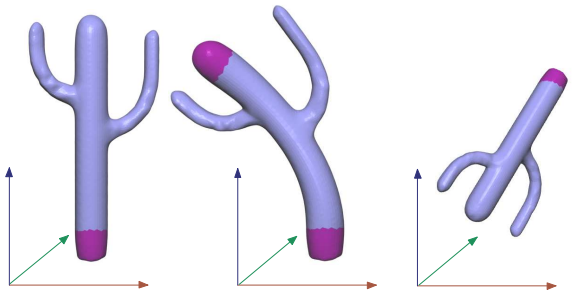

Figure 2: *Left: the original cactus model. Middle: the deformed cactus by rotating and translating the top handle. Right: The reconstructed model after a global transformation (including translation, rotation and uniform scaling) is applied to the handle of the original cactus model (left). We render the ROI in blue and the handle(s) in purple in all the examples.*

where $\mathbf{u}_j$ is the position of the vertex with index b_j on the boundary condition, $\mathbf{W}_k$ is the local deformation gradient at the vertex with index c_k on the boundary condition, and s and t are the number of position constraints and the number of transformation constraints, respectively. Usually, we specify both the positions and the deformation gradients for the same set of boundary vertices (i.e. $\{\tilde{\mathbf{v}}_{b_j}\} = \{\tilde{\mathbf{v}}_{c_k}\}$). Nevertheless, it is possible to specify only the deformation gradients for the boundary vertices, and let their positions be free, or vice versa.

As $\mathbf{M}_i$ is linearly dependent on the unknown vertex positions, $\mathbf{M}_i$ and $\tilde{\mathbf{v}}_i$ are simultaneously solved from the optimization in Equation 7. Therefore, the formulated optimization can be used for mesh deformation: like previous Laplacian surface editing work [SLCO*04, YZX*04, LSLCO05], we allow the user to interactively change the boundary condition through manipulating the handles and reconstruct the unconstrained region of interest from the optimization. During editing, the locally defined deformation gradients accommodate the local changes of details (see an example in Figure 2).

3.3. Handling Degenerate Vertices

This section explains how we handle the special case when $\mathbf{V}_i\mathbf{V}_i^{\mathbf{T}}$ is singular in Equation 4. We refer to the corresponding vertex $\mathbf{v}_i$ as a *degenerate vertex*. For a degenerate vertex $\mathbf{v}_i$, the rank of $\mathbf{V}_i\mathbf{V}_i^{\mathbf{T}}$ is equal to 2, which means that $\mathbf{v}_i$ and its neighbors are coplanar. In practice, meshes with high resolution are locally smooth. Therefore, degenerate vertices are common. As the uniqueness of the solution to the optimization in Equation 7 is under the condition that the neighborhood coherence term is well defined for every pair of neigh-

boring vertices, we have to implicitly define the transformations for all the vertices, including the degenerate vertices.

Our basic idea of handling degenerate vertices is to move $\mathbf{v}_i$ out of the plane defined by its neighboring vertices. Let $\mathbf{v}'_i$ denote the modified vertex of $\mathbf{v}_i$. Since the direction of a nondegenerate LC approximates the normal, we place $\mathbf{v}'_i$ in the normal direction of $\mathbf{v}_i$ (see Figure 3 (b)). Typically, we set

$$\mathbf{v}'_i = \mathbf{v}_i + \frac{s_{disp}}{|N(i)|} \sum_{j \in N(i)} \|\mathbf{v}_i - \mathbf{v}_j\| \cdot \mathbf{n}_i, \tag{8}$$

where $\mathbf{n}_i$ is the unit normal vector at $\mathbf{v}_i$ and s_{disp} is a scaling factor. We traverse the entire vertex list to handle all the degenerate vertices. We classify $\mathbf{v}_i$ as a degenerate vertex if and only if the condition number of $\mathbf{V}_i\mathbf{V}_i^{\mathbf{T}}$ is greater than a given threshold (e.g., 2.0×10^5). If a degenerate vertex is on an open boundary, we adopt a small variation: if its valence is less than 3, we insert a new vertex at the midpoint of its opposite edge; if its neighboring vertices are collinear, we displace one of its neighboring non-boundary vertices instead of $\mathbf{v}_i$ itself.

After defining the deformation gradients for all the unconstrained vertices, we solve for the positions of the unconstrained vertices using Equation 7. Since the optimized position of $\tilde{\mathbf{v}}'_i$, corresponding to the modified vertex $\mathbf{v}'_i$ of a degenerate vertex $\mathbf{v}_i$, is influenced by our deliberate displacement (Figure 3 (c)), we need to pull $\tilde{\mathbf{v}}'_i$ back to the correct position $\tilde{\mathbf{v}}_i$ (Figure 3 (d) and (e)). As $\mathbf{M}_i$ is already computed

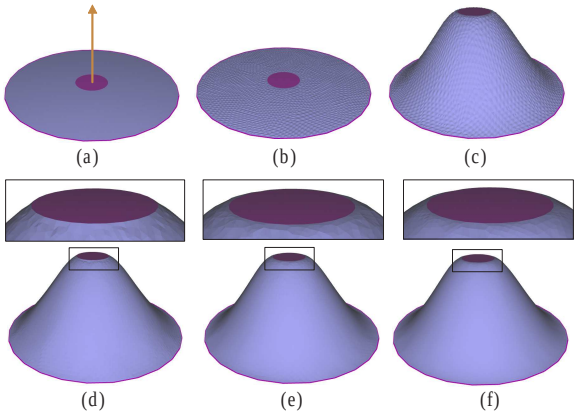

Figure 3: *Handling degenerate vertices. (a) A planar irregular triangular mesh with the boundary vertices and the vertices in the central region as the boundary constraints. (b) After displacing every degenerate vertex. (c) Editing without removing the displacement. (d) and (e): Editing with the displacement removed, using uniform weighting and cotangent weighting, respectively. (f) Reconstruction errors are distributed over the ROI. All the images in this paper are flat-shaded to better demonstrate that the errors introduced by the handling of degenerate vertices are unnoticeable.*

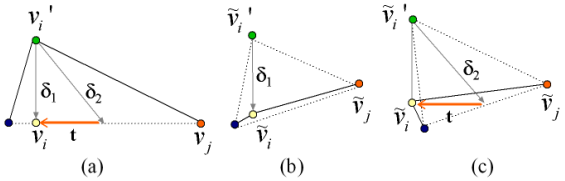

Figure 4: *Difference between using cotangent weights (b) and uniform weights (c).*

at this stage, we obtain the new positions of the degenerate vertices by applying the associated affine transformations to the original vertex positions (before the deliberate displacements): $\tilde{\mathbf{v}}_i = \mathbf{M}_i(\mathbf{v}_i - \mathbf{v}'_i) + \tilde{\mathbf{v}}'_i$.

There is no stability problem as long as all the deformation gradients for the degenerate vertices are defined prior to the optimization (cf. Section 3.2). Nevertheless, the handling of degenerate vertices does introduce two types of errors in the editing results, albeit unnoticeable. Computing the edited position $\tilde{\mathbf{v}}'_i$ instead of $\tilde{\mathbf{v}}_i$ through optimization introduces the first type of error, which is an increasing function of s_{disp}. We experimented with various scaling factors (e.g., $s_{disp} = 0.2$ and $s_{disp} = 10$) and found that the resulting shapes are of little visual difference (see an example in Figure 6). Thus we use a small fixed scaling factor (e.g., $s_{disp} = 0.2$) in all our experiments.

The second type of error arises because the edited positions of the degenerate vertices are computed through *linear* transformations ($\tilde{\mathbf{v}}_i = \mathbf{M}_i(\mathbf{v}_i - \mathbf{v}'_i) + \tilde{\mathbf{v}}'_i$) whereas the edited positions of the nondegenerate vertices are solved in the *least-squares* sense. We found that discretizing the Laplacian using cotangent weights gives much better results (Figure 3 (e)) than using uniform weights (Figure 3 (d)). We illustrate the reason using a 2D example in Figure 4. Vertex $\mathbf{v}_i$ is an unconstrained degenerate vertex, which is displaced to $\mathbf{v}'_i$ before editing. The LC at $\mathbf{v}'_i$ is δ_1 (close to the normal direction) if cotangent weights are used or δ_2 (with a tangential component $\mathbf{t}$) if uniform weights are used. After editing, the tangential component $\mathbf{t}$ in δ_2 would cause tangential drifting, bringing the vertex "away" from the surface (Figure 4 (c)), especially at vertices in curved regions after editing (Figure 3 (d)), producing visual artifact and a bigger reconstruction error.

It is noteworthy that in the shearing removal step (described in the next section), solving the system (Equation 9 or 10) in the least-squares sense makes the second type of errors uniformly distributed to the whole ROI, making the errors unnoticeable (Figure 3 (f)).

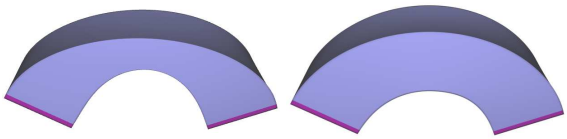

Figure 5: *An deformation example without shearing removal (Left) and with shearing removal (Right).*

4. Shearing Removal by Polar Decomposition

The deformation gradients may contain shearing transformations, making the LCs deviate from the normal directions and thus causing shearing distortion.

The neighborhood coherence term only partially solves the shearing problem. When different rigid transformations are applied to different handles, shearing problem may occur. The neighborhood coherence term only guarantees that features close to the handle boundaries deform almost rigidly[†]. Shearing distortion may occur at regions far away from the constrained regions since the rigid transformations applied to the handles has less influence on the transformations defined over the far-away regions (Figure 5 left).

To overcome the shearing problem, we introduce an additional step after solving the optimization in Equation 7. Since what we really care for in the information encoded in $\mathbf{M}_i$ is the rotation part and the uniform scaling part, similar to [MHTG05], our solution is to extract the rotation and uniform scaling information from $\mathbf{M}_i$ by polar decomposition.

We first perform the singular value decomposition (SVD) on $\mathbf{M}_i$: $\mathbf{M}_i = \mathbf{U}_i\mathbf{W}_i\mathbf{V}_i^{\mathbf{T}}$, where $\mathbf{U}_i$ and $\mathbf{V}_i$ are two 3×3 orthogonal matrices and $\mathbf{W}_i$ is a 3×3 diagonal matrix whose elements are the eigenvalues $(\lambda_j, 1 \le j \le 3)$ of $\mathbf{M}_i$. Next we define the modified the deformation gradient without shearing as $\mathbf{M}_i' = \mathbf{R}_i\mathbf{S}_i$, where $\mathbf{R}_i = \mathbf{U}_i\mathbf{V}_i^{\mathbf{T}}$ and $\mathbf{S}_i = \mathrm{diag}(s_i, s_i, s_i)$ with $s_i = \sqrt{\frac{\lambda_1^2+\lambda_2^2+\lambda_3^2}{3}}$.

We provide the user two options: to retain the sizes of the local features or automatically scale the geometric features during deformation. The respective underlying systems of the former and latter choices are

$$D(\tilde{\mathbf{v}}_i) = \mathbf{R}_i\delta_i \tag{9}$$

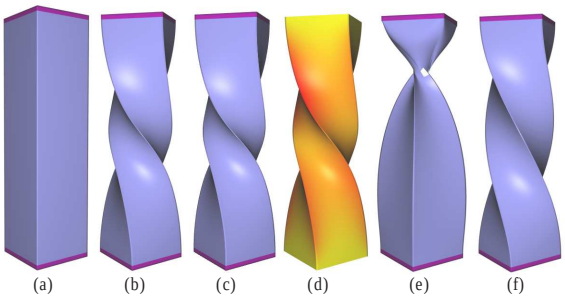

Figure 6: *(a) The original bar model with an ROI (blue) and two handles (purple). After the top handle undergoes a large rotation, (b) and (c) are our results by setting $s_{disp} = 0.2$ and $s_{disp} = 10$, respectively, which have little visual difference. The difference between corresponding vertices is shown in (d) with red color denoting relatively big difference and yellow color denoting no difference. (e) and (f) are the deformation results with methods in [LSCO*04] and [ZRKS05], respectively.*

or

$$D(\tilde{\mathbf{v}}_i) = \mathbf{R}_i\mathbf{S}_i\delta_i. \tag{10}$$

The first choice is mainly used when the user applies rigid transformations (including rotation and translation and excluding scaling) to the handles. The second choice is often adopted when the transformations applied to the handles include scaling. Note that the second choice is necessary for configuration-independent merging (Section 5.1), as we want the scales of the features to be automatically determined.

With this shearing removal step, our deformation tool always produces more visually pleasing results than previous methods for large angle rotation or big-scale translation. Figure 6 shows an example when the top handle of the vertical bar undergoes a large rotation. Our method outperforms existing implicit methods [LSCO*04, SLCO*04] [‡]. Note that, like ours, explicit methods [ZRKS05, YZX*04] also produce natural deformation results as transformations are explicitly propagated (see more discussions in Section 6).

Figure 7 demonstrates the effectiveness of our method for deformation under big-scale translation of the handle. Explicit methods [YZX*04, ZHS*05, ZRKS05] cannot infer any rotation information from translations of handles (Figure 7 left). Implicit methods [LSCO*04, SLCO*04] can capture rotation under small-scale translation but do not work for big-scale translation (Figure 7 middle). With our method, the global shapes in the deformed model (e.g., the whole

[†] According to the formulation of the neighborhood coherence term, the more neighboring vertices we use to define the coherence term, the wider will be the rigid region (close to the handle boundaries). This is a desirable property. However, there are two side effects of using more vertices. First, the *local* deformation of features will be suppressed. An extreme example is to define the coherence term using the whole region of interest. Second, the introduction of more vertices makes the system matrix denser. We choose to use 2-ring neighbors (cf. *SF* in Equation 6) as it is a good trade-off.

[‡] Here we only compare with [LSCO*04] and not with [SLCO*04] as the latter only includes a small refinement on the results of the former when the approximation error is large.

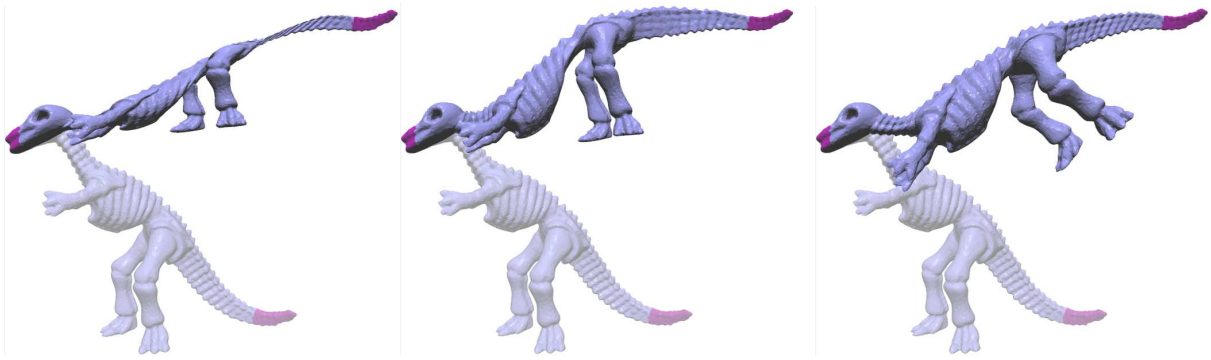

Figure 7: *An example of rotation of local features resulting from translation of handles. The deformation results by [ZRKS05] (Left), [LSCO*04] (Middle) and ours (Right) when the handle at the tail of the dinosaur undergoes a big-scale translation.*

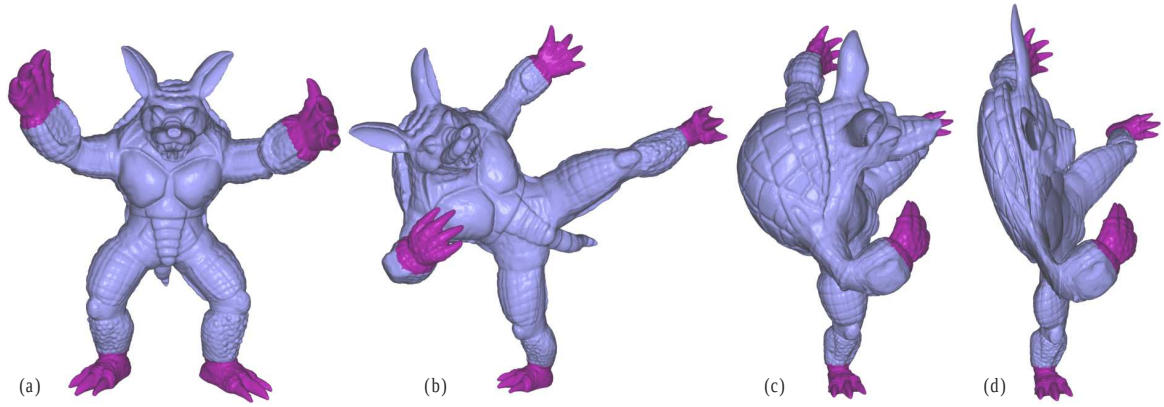

Figure 8: *(a) The original Armadillo model with four handles specified. (b) and (c): Two views of the deformed model by applying several rigid transformations to the handles. (d) The deformation result with the same view as (c) but without shearing removal.*

Figure 9: *An example of scaling local features edited by translating handles. Left: the input lion model. Middle: the deformed model without scaling the LCs. Right: the deformed model with scaling the LCs.*

arms and legs) are appropriately rotated and the local details are well preserved. This example also demonstrates that most rotation information is successfully captured by the affine transformations and effectively extracted in the shearing removal step. Figure 8 illustrates that it is easy to use our deformation tool to produce complicated but visually pleasing deformation results by simply applying several rigid transformations to the handles. The contrast be-

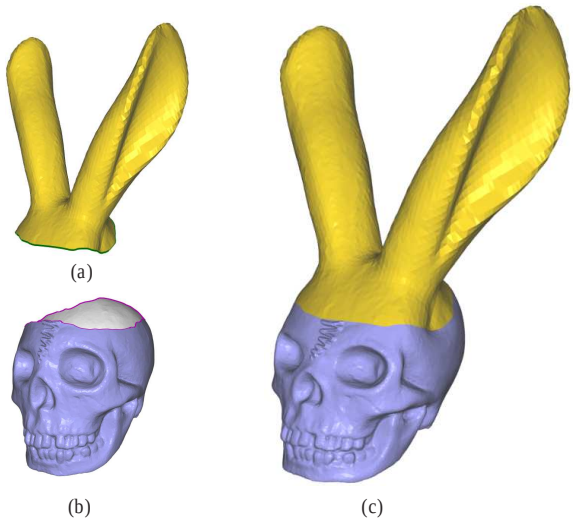

Figure 10: *Configuration-independent merging: the Headus skull model (b) is deformed and merged to the ears of the Stanford Bunny (a). The two merging boundaries have undulations and are of different shapes.*

tween (c) and (d) clearly demonstrates the effectiveness of our shearing removal procedure.

The above examples are obtained by retaining the sizes of the geometric features (cf. Equation 9). Sometimes letting the system automatically scales the LCs is useful (cf. Equation 10). In Figure 9, the handles are moved closer to each other, thus the space between them becomes smaller and cannot accommodate the big global features of the original body. Automatically scaling the LCs (using Equation 10) gives a better visual result.

5. Mesh Merging

In this section, we present two methods of mesh merging: configuration-independent merging and configuration-dependent merging. Although some existing techniques [YZX*04, ZRKS05, LSLCO05] could be adopted to implement configuration-independent merging, we believe that this is the first time such an application is proposed.

5.1. Configuration-Independent Merging

The configuration of the objects to be merged refers to their relative position, orientation and scale. For simplicity, we assume that the *target mesh* is fixed during merging, and let the relative position, orientation and scale of the *source mesh* be free. The merging is accomplished by deforming the merging boundary of the source mesh (and the entire source mesh as the ROI) to the merging boundary of the target mesh[§].

Having the user adjust the configuration of input meshes is equivalent to applying a global similarity transformation to the source mesh. Recall that differential-based editing frameworks deform an ROI by modifying the boundary condition. Therefore, the basic requirement of performing configuration-independent merging using such frameworks is that when a global similarity transformation is applied to the boundary condition, the positions of the vertices in the ROI, obtained through solving the optimization with the modified boundary condition, must reflect the same transformation. Figure 2 (right) demonstrates that our method satisfies this requirement.

To perform configuration-independent merging, the user only needs to establish the correspondence between the source merging boundary and the target merging boundary. We implement a vertex correspondence tool similar to [YZX*04]. From a set of user-specified key vertex correspondences, our system finds the corresponding position on the target merging boundary for each remaining vertex on the source merging boundary by curve parameterization. These positions will be used as position constraints of the optimization.

Next we need to compute a rotation transformation $\mathbf{R}_i$ and a scaling factor s_i for each pair of corresponding vertices $(\mathbf{v}_i^s, \mathbf{v}_i^t)$, where $\mathbf{v}_i^s$ is a vertex on the source merging boundary and $\mathbf{v}_i^t$ is its corresponding position on the target merging boundary. By defining two local frames at $\mathbf{v}_i^s$ and $\mathbf{v}_i^t$ (composed of the associated tangent vector along the boundary curve, unit normal vector and the cross product of the previous two vectors), we compute $\mathbf{R}_i$ as the transformation from the local frame at $\mathbf{v}_i^s$ to the local frame at $\mathbf{v}_i^t$. The scaling factor s_i is computed as the ratio of the tangent magnitude at $\mathbf{v}_i^t$ to the tangent magnitude at $\mathbf{v}_i^s$. The deformation gradient $\mathbf{M}_i$ is then defined as $\mathbf{R}_i\mathbf{S}_i$ ($\mathbf{S}_i = \mathrm{diag}(s_i, s_i, s_i)$) and used as the transformation constraint of the optimization.

With the position and transformation constraints, we solve the optimization in Equation 7 to deform the source mesh and merge it to the target mesh. The shearing distortion is removed in the same way as in mesh deformation application. To get a watertight seam, we trivially zip the target merging boundary and the deformed source merging boundary [SLCO*04] by removing one band of triangles adjacent to the target merging boundary and re-triangulating the resulting gap. To improve smoothness at the seam, we apply several iterations of the umbrella operator [Tau95]. Detail-preserving merging is discussed in Section 5.3.

Figure 1 shows the result of merging a source mesh to a target mesh in two different configurations. This example

§ An alternative approach, which we did not implement, is to merge the meshes along a user-specified intermediate boundary [YZX*04].

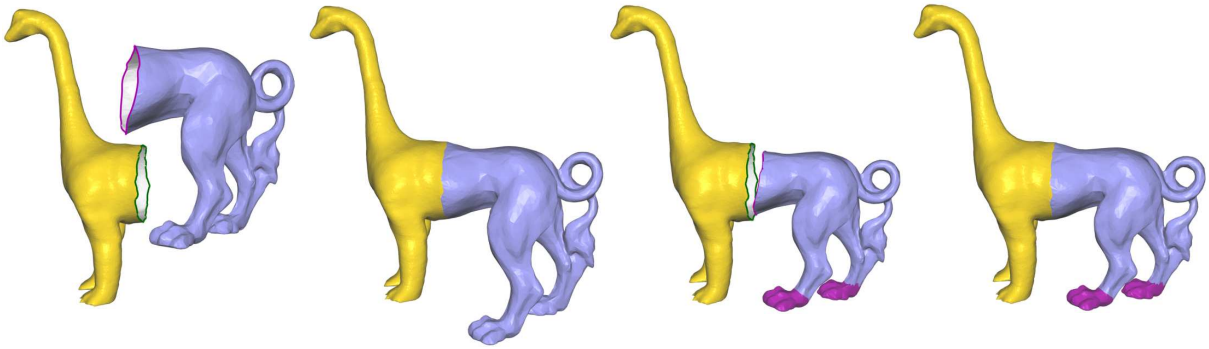

Figure 12: *The hind part of the Feline model is deformed and merged to the fore part of the Dinosaur model. (b) is the result of configuration-independent merging with the configuration in (a), and (d) is the result of configuration-dependent merging with the configuration in (c). In the configuration-dependent merging, the two feet are specified as handles (purple), thus remain fixed. The same boundary correspondence is used for both merging.*

demonstrates that, given the same boundary correspondence, our configuration-independent merging method produces the same result. Figure 10 demonstrates that our method also works well for merging boundaries of different shapes and with undulations. Figure 11 shows an example with multiple pairs of merging boundaries. The center CAD model (source mesh to be deformed) is to be merged to the four cylinders (fixed target meshes). The CAD model is symmetrical, but the four cylinders have different scales. With configuration-dependent merging [SLCO*04], the user would have great difficulty adjusting the configuration of the CAD model. Our configuration-independent merging method creates the same result independent of the configuration. This example also demonstrates that the merging method is applicable to meshes with non-zero genus. The close-up shows that the smoothness across the merging boundary is not satisfactory. Section 5.3 addresses this problem by using overlapping transition merging regions.

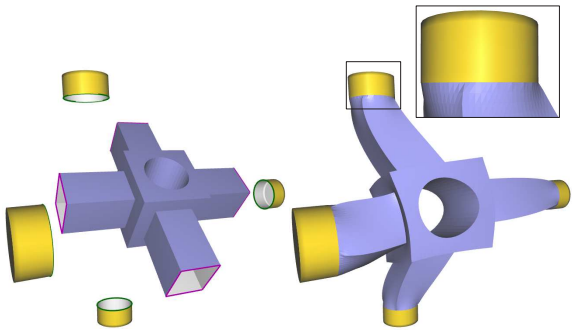

Figure 11: *A configuration-independent merging example with multiple pairs of merging boundaries.*

5.2. Configuration-Dependent Merging

In configuration-independent merging, the entire source mesh is deformed and merged to the target mesh. If the size of the source mesh is very large, computation would be expensive. Allowing only the region near the source merging boundary to be deformed can accelerate the merging progress. Moreover, the user may want to specify handles so as to fix certain features on the source mesh. Such tasks can be achieved using configuration-dependent merging.

For configuration-dependent merging, the user specifies the configuration of the source and the target mesh as well as a set of handles on the source mesh. The remaining algorithm is the same as the configuration-independent merging except the following. We modify only the positions of the source boundary vertices and their scaling factors, keeping their local frames unchanged. The deformed source merging boundary and the fixed handles together provide the boundary condition to the optimization problem. This is similar to the transplanting method in [SLCO*04].

Figure 12 compares some results of the two merging methods. With configuration-independent merging, the legs of the Feline model are not in harmony with the legs of the Dinosaur model. With configuration-dependent merging, we adjust the position, orientation and scale of the Feline model and fix the two feet, and the merging result is better.

5.3. Merging with Overlapped Transition Region

We have assumed so far that no transition region is specified for merging. The target mesh is never deformed. In order to coincide with the target merging boundary, the source merging boundary is deformed (consequently deforming the whole source ROI too). Since the geometry near the target merging boundary and the geometry near the deformed

source boundary are usually different, smoothing needs to be performed [YZX*04]. This smoothing step however also filters out geometry details.

To produce merging region with smoothly transited details, the user could specify a transition region on both the source mesh and the target mesh [SLCO*04]. The ROIs of both meshes are deformed and the details are mixed by interpolating the LCs in the transition regions. For configuration-independent merging, we first apply the implicitly defined local deformation gradients onto the LCs before interpolating them.

We let the user specify a width for each mesh and define the transition region as the region encompassing all vertices whose shortest path to the merging boundary has a length less than a specified width. Correspondence between the merging boundaries is established by specifying several key vertex correspondences. (Figure 13 (a)). For correspondence between the transition regions, we cut each transition region along the shortest path between two key vertices and parameterize the region over the unit square domain. A vertex $\mathbf{v}_i$ in one transition region is mapped to a position $\overline{\mathbf{v}}_i$ within a triangle τ in the other transition region, $\overline{\mathbf{v}}_i = b_{i_1}\mathbf{v}_{i_1} + b_{i_2}\mathbf{v}_{i_2} + b_{i_3}\mathbf{v}_{i_3}$, where $\mathbf{v}_{i_*}$ are the vertices of triangle τ and b_{i_*} are the corresponding barycentric coordinates.

We add an additional error term E_T to the optimization problem.

$$E_T = \sum_{i \in TR} \|M_i\delta_i - \sum_{j=1}^{3} b_{i_j} M_{i_j}\delta_{i_j}\|^2,$$

where TR is the index set of vertices in the transition regions, and M_{i_*} and δ_{i_*} are the corresponding affine transformations and Laplacian coordinates of $\mathbf{v}_{i_*}$, respectively. Adding the error term to the objective function of the optimization problem, we get

$$\arg\min E(\tilde{\mathbf{v}}_1, \ldots, \tilde{\mathbf{v}}_n) = E_L + E_R + w_T E_T,$$

where w_T is a weight (e.g. $w_T = 0.1$ in our experiments). By minimizing the objective function with the same boundary condition as before, we obtain a merged mesh. The optimal solution, however, generally does not overlap two transition regions exactly. Therefore, like the method in [SLCO*04], we reconstruct a smooth transition by using the connectivity information of one transition region and the linearly interpolated LCs (Figure 13 (b)).

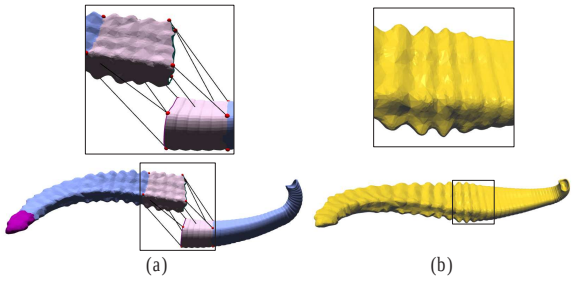

Figure 13: *Configuration-independent merging with overlapped transition regions. (a) Two transition regions with key correspondences specified. (b) The merging result.*

6. Implementation Details and Discussions

The solution of the optimization in Equation 7 and the solution of the linear system in Equation 9/10 are obtained by solving the following form of normal equations

$$\mathbf{A}^\mathbf{T}\mathbf{A}[\mathbf{V}_x\ \mathbf{V}_y\ \mathbf{V}_z] = \mathbf{A}^\mathbf{T}[\mathbf{b}_x\ \mathbf{b}_y\ \mathbf{b}_z],$$

where $\mathbf{V}_x$, $\mathbf{V}_y$ and $\mathbf{V}_z$ are corresponding components (x, y, and z) of the unconstrained vertex positions, $\mathbf{b}_x$, $\mathbf{b}_y$ and $\mathbf{b}_z$ are three known vectors constructed from the position and transformation constraints, and $\mathbf{A}$ is a large sparse matrix. As matrix $\mathbf{A}$ is only dependent on the original mesh and each row of the deformation gradients (Equation 4) is only dependent on one dimension (e.g. x coordinates), we can prefactorize $\mathbf{A}^\mathbf{T}\mathbf{A}$ using Choleskey factorization and solve $\mathbf{V}_x$, $\mathbf{V}_y$ and $\mathbf{V}_z$ separately by back substitution. In practice, we use an efficient sparse linear solver [Tol03]. Table 1 lists the number of unconstrained vertices, the number of degenerate vertices, the factorization time and back-substitution time in Equations 7 and 9/10 as well as the time for computing SVD for all deformation gradients.

Given the boundary condition, the unknown surface is reconstructed through optimization, guaranteeing that any errors are distributed over the entire unknown surface. However, the resulting deformation is still sensitive to the resolution of the boundary condition. For example, if a complex ROI is subject to a boundary condition consisting of only 3 vertices, then the reconstructed surface would be extremely sensitive to the changes in the positions or in the local deformation gradients of the boundary vertices. Therefore, for boundaries with considerably fewer vertices relative to the ROI, we first perform a local refinement. This is particularly useful in configuration-independent merging (without overlapped transition regions), since the boundary condition is composed of only one boundary curve.

We note that, due to the fact that transformations are explicitly propagated rather than determined via optimization, explicit methods [YZX*04, ZRKS05] can handle rotation angles of handles larger than π. Being an implicit method, our system does not support editing with rotation angles larger than π. The transformations are implicitly defined and solved simultaneously with the vertex positions by minimizing distortion. Since a larger angle would give greater distortion, our system always chooses the transformation with the smallest rotation angle ($< \pi$). To perform deformation with large rotation angles ($> \pi$), one possible way is to use

Mesh	Free Vertices	Degenerate Vertices	Factor (Eqn 7)	Solve (Eqn 7)	Factor (Eqn 9/10)	Solve (Eqn 9/10)	SVD
Feline (Fig. 12 (c))	7,733	186	0.593s	0.032s	0.203s	0.015s	0.125s
Lion	7,903	77	0.313s	0.015s	0.063s	0.016s	0.156s
Cactus	8,978	41	0.359s	0.031s	0.078s	0.015s	0.172s
CAD (Fig. 11)	9,050	1,839	0.563s	0.031s	0.110s	0.015s	0.156s
Feline (Fig. 12 (a))	9,434	206	0.718s	0.032s	0.125s	0.016s	0.187s
Dinosaur	13,252	3	0.625s	0.031s	0.110s	0.015s	0.266s
Mannequin head	16,219	534	1.437s	0.062s	0.234s	0.032s	0.328s
Headus skull	16,982	77	1.390s	0.062s	0.203s	0.031s	0.329s
Bar	24,480	5,852	2.516s	0.109s	0.453s	0.032s	0.469s
Armadillo	48,027	61	4.813s	0.188s	0.875s	0.078s	0.985s

Table 1: *The timing results for our examples on a 3.2GHz Pentium IV machine with 1G RAM.*

more handles such that rotation angles between successive handles is smaller than π.

Although performing SVD on a 3×3 matrix is fast, performing SVD on a large set of deformation gradients still becomes the bottleneck of our system. Since solving the optimization in Equation 7 is fast (only involving the back-substitution), the screen is updated using the solved vertex positions when the user is manipulating a handle. SVD and the back-substitution of the second linear system are performed only when the user stops moving the handle. This allows our system to remain at interactive rate for editing large scale meshes.

The parallel work of Lipman et al. [LSLCO05] is similar to ours. Their method binds triple vectors defining a discrete frame to each vertex. The discrete frames before and after editing essentially determine a deformation gradient (like $\mathbf{M}_i$ in our framework). Essentially, in their method, the local details (rigid-invariant coordinates) at each vertex are encoded in the corresponding discrete frame, while in ours the local details (the LCs) are defined with respect to the corresponding deformation gradients. Since the discrete frames in [Lipman et al. 2005] are nonlinearly dependent on vertex positions, the discrete frames and vertex positions cannot be formulated and solved as a single linear system. To make the reconstruction problem linear, their method uses two sparse linear systems: one for solving the discrete frames (using only the transformation constraints) and the other for solving the vertex positions given the discrete frames obtained from the first system (using the position constraints only). In our method, the deformation gradients are solved using only one linear system (considering both the position and transformation constraints). Our solution avoids the incompatibility between the position and transformation constraints encountered in [Lipman et al. 2005] at the cost of introducing more shearing distortion. Consequently, we introduce an additional step to remove the shearing distortion.

7. Conclusion

This paper presents an implicit Laplacian mesh editing framework that is resistant to the non-rigid-invariant property of the Laplacian coordinates. We demonstrate the advantage of our framework in a new application which we call configuration-independent merging. We also integrate configuration-dependent merging and mesh deformation into the same framework. Due to the implicitly defined local deformation gradients (with shearing removed) introduced at each vertex, all these editing techniques are automatically detail-preserving and produce visually pleasing deformed results. The deformed or merged results are obtained by solving two sparse linear systems at interactive rate.

We have shown that, in configuration-dependent merging, the user can choose to fix specific features during merging. This capability can be extended to configuration-independent merging. For each feature to be fixed, we implicitly define a common deformation gradient for all the Laplacian coordinates associated with the feature. In other words, the feature is subject to a transformation rather than being completely fixed like in configuration-dependent merging.

Acknowledgments

We thank the anonymous reviewers for the helpful comments. The lion model is courtesy of Robert W. Sumner. Other models are courtesy of Stanford University and 3D CAFE. This work was supported by a grant from the Research Grant Council of the Hong Kong Special Administrative Region, China (Project No. HKUST6295/04E).

References

[ACP03] ALLEN B., CURLESS B., POPOVIĆ Z.: The space of human body shapes: reconstruction and parameterization from range scans. *ACM Trans. Graph. 22*, 3 (2003), 587–594. 4

[BKZ01] BIERMANN H., KRISTJANSSON D., ZORIN D.: Approximate boolean operations on free-form solids. In *Proceedings of ACM SIGGRAPH 2001* (2001), pp. 185–194. 2

[BMBZ02] BIERMANN H., MARTIN I., BERNARDINI F., ZORIN D.: Cut-and-paste editing of multiresolution surfaces. *ACM Trans. Graph. 21*, 3 (2002), 312–321. 3

[FT05] FU H., TAI C.-L.: *Mesh editing with affine-invariant Laplacian coordinates*. Tech. rep., The Hong Kong University of Science and Technology, January 2005. Technical Report HKUST-CS05-01. 3

[LÓ3] LÉVY B.: Dual domain extrapolation. *ACM Trans. Graph. 22*, 3 (2003), 364–369. 3

[LSCO*04] LIPMAN Y., SORKINE O., COHEN-OR D., LEVIN D., RÖSSL C., SEIDEL H.-P.: Differential coordinates for interactive mesh editing. In *Proceedings of Shape Modeling International* (2004), pp. 181–190. 1, 2, 3, 6, 7

[LSLCO05] LIPMAN Y., SORKINE O., LEVIN D., COHEN-OR D.: Linear rotation-invariant coordinates for meshes. *ACM Trans. Graph. 24* (2005). 2, 3, 4, 8, 11

[MBWB02] MUSETH K., BREEN D. E., WHITAKER R. T., BARR A. H.: Level set surface editing operators. *ACM Trans. Graph. 21*, 3 (2002), 330–338. 2

[MDSB02] MEYER M., DESBRUN M., SCHRÖDER P., BARR A. H.: Discrete differential-geometry operators for triangulated 2-manifolds. In *Proceedings of VisMath* (2002). 2

[MHTG05] MÜLLER M., HEIDELBERGER B., TESCHNER M., GROSS M.: Meshless deformations based on shape matching. *ACM Trans. Graph. 24*, 3 (2005), 471–478. 6

[NSACO05] NEALEN A., SORKINE O., ALEXA M., COHEN-OR D.: A sketch-based interface for detail-preserving mesh editing. *ACM Trans. Graph. 24*, 3 (2005). 1

[SK04] SHEFFER A., KRAYEVOY V.: Pyramid coordinates for morphing and deformation. In *3D Data Processing, Visualization, and Transmission* (2004), pp. 68–75. 3

[SLCO*04] SORKINE O., LIPMAN Y., COHEN-OR D., ALEXA M., RÖSSL C., SEIDEL H.-P.: Laplacian surface editing. In *Symposium on Geometry Processing* (2004), pp. 179–188. 1, 2, 3, 4, 6, 8, 9, 10

[SP04] SUMNER R. W., POPOVIĆ J.: Deformation transfer for triangle meshes. *ACM Trans. Graph. 23*, 3 (2004), 399–405. 3, 4

[SZGP05] SUMNER R. W., ZWICKER M., GOTSMAN C., POPOVIĆ J.: Mesh-based inverse kinematics. *ACM Trans. Graph. 24*, 3 (2005), 488–495. 3

[Tau95] TAUBIN G.: A signal processing approach to fair surface design. In *Proceedings of ACM SIGGRAPH 95* (1995), pp. 351–358. 8

[Tol03] TOLEDO S.: Taucs: A library of sparse linear solvers, version 2.2, 2003. Tel-Aviv University, Available online at http://www.tau.ac.il/ stoledo/taucs/. 10

[YZX*04] YU Y., ZHOU K., XU D., SHI X., BAO H., GUO B., SHUM H.-Y.: Mesh editing with poisson-based gradient field manipulation. *ACM Trans. Graph. 23*, 3 (2004), 644–651. 1, 2, 3, 4, 6, 8, 10

[ZHS*05] ZHOU K., HUANG J., SNYDER J., LIU X., BAO H., GUO B., SHUM H.-Y.: Large mesh deformation using the volumetric graph laplacian. *ACM Trans. Graph. 24*, 3 (2005). 1, 2, 6

[ZRKS05] ZAYER R., RÖSSL C., KARNI Z., SEIDEL H.-P.: Harmonic guidance for surface deformation. In *Computer Graphics Forum, Proceedings of Eurographics 2005* (2005). 1, 2, 6, 7, 8, 10