Research Article

Pose Space Surface Manipulation

Yusuke Yoshiyasu¹ and Nobutoshi Yamazaki²

¹ School of Science and Technology, Keio University, Kanagawa 223-8522, Japan

² Department of Mechanical Engineering, Keio University, Kanagawa 223-8522, Japan

Correspondence should be addressed to Yusuke Yoshiyasu, yusuke_2_ax_es@z6.keio.jp

Received 17 December 2011; Revised 10 February 2012; Accepted 24 February 2012

Academic Editor: Alexander Pasko

Copyright © 2012 Y. Yoshiyasu and N. Yamazaki. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Example-based mesh deformation techniques produce natural and realistic shapes by learning the space of deformations from examples. However, skeleton-based methods cannot manipulate a global mesh structure naturally, whereas the mesh-based approaches based on a translational control do not allow the user to edit a local mesh structure intuitively. This paper presents an example-driven mesh editing framework that achieves both global and local pose manipulations. The proposed system is built with a surface deformation method based on a two-step linear optimization technique and achieves direct manipulations of a model surface using translational and rotational controls. With the translational control, the user can create a model in natural poses easily. The rotational control can adjust the local pose intuitively by bending and twisting. We encode example deformations with a rotation-invariant mesh representation which handles large rotations in examples. To incorporate example deformations, we infer a pose from the handle translations/rotations and perform pose space interpolation, thereby avoiding involved nonlinear optimization. With the two-step linear approach combined with the proposed multiresolution deformation method, we can edit models at interactive rates without losing important deformation effects such as muscle bulging.

1. Introduction

Editing and animation of character models is an important task in computer graphics. The user, in general, demands the editing procedure to be interactive and intuitive. However, the ability to pose a model naturally and realistically is equally important for a character modeling system. Example-based techniques are a promising approach to improve the realism of interactive editing techniques. Pose space deformation (PSD) [1] is a skeleton-based approach which animates the model using skeletons and then corrects the resulting surface to conform to example shapes. In PSD, corrective displacements, the differences between a skinning surface and examples, are interpolated in pose space formed by joint angles. These displacements are then added to the skinning surface. While this method can model deformation effects such as muscle bulging, editing the pose of the model naturally is difficult and tedious as the number of joints increases.

In contrast, the mesh-based inverse kinematics (MeshIK) technique [2, 3] can pose a model meaningfully by just translating a few vertices of the mesh. Instead of interpolating

corrective displacements, MeshIK interpolates local deformations called deformation gradients. It automatically determines interpolation weights from the user-specified position constraints through nonlinear optimization. Because of this iterative process, it is time consuming for this method to edit a large model. Moreover, while MeshIK can edit the global pose naturally, it is counterintuitive to manipulate the local pose via translation; bending and twisting are more intuitive in this case.

In this paper, we present an example-driven mesh deformation technique that allows us not only to edit a global pose naturally but also to adjust a local pose precisely. Our system provides both translational and rotational controls to deform a model. By translating handle vertices, the user can produce a model in natural poses easily. By applying rotations to handle triangles, the user can bend and twist the model as if manipulating a joint. To this end, we formulate the problem by inferring a pose from handle translations/rotations and using a two-step linear surface optimization technique that is based on a linear rotation-invariant mesh representation. The benefits of this formulation are as follows: (1) it is

straightforward to incorporate translational and rotational constraints; (2) it achieves rotation invariance and thus is applicable to examples that involves large deformations (rotations and bending); (3) it avoids time-consuming non-linear optimization. In combination with the proposed multiresolution approach, we can edit high-resolution meshes at interactive rates while retaining important deformation effects such as muscle bulging.

The rest of the paper is organized as follows. In Section 2, we briefly review-related work. We provide an overview of our system in Section 3. We then describe our basic deformation framework in Section 4. In Section 5, we explain our PSD technique. Section 6 shows our multiresolution method. We demonstrate our method and show our results in Section 7. Finally, we conclude our work in Section 8.

2. Related Work

Pose Space Deformation. Pose space deformation (PSD) [1] is an example-based deformation method that corrects the artifacts of skeletal subspace deformation (SSD) and models subtle surface deformation, such as muscle bulging. This method is thought of as a combination of SSD and shape interpolation (morphing). In PSD, an initial mesh is first computed by SSD. This is then corrected by adding corrective displacements. These displacements, regarded as differences between SSD and examples, are interpolated such that they can properly correct the SSD surface in an arbitrary pose. Kry et al. [4] compressed corrective displacements using PCA to make them more compact. Kurihara and Miyata [5] introduced weighted pose space deformation to achieve PSD from a limited number of examples. Weber et al. [6] showed that the use of differential surface deformation in the PSD framework improves the quality of the result. They represented corrections with rotations and integrated them by solving a Poisson system. Wang et al. [7] introduced a similar method to [6] using a regression model.

Incorporating Real Dataset. To create realistic animation, some methods [8–11] use range scanned data as examples. Allen et al. [8] captured 3D shapes of body parts in different poses and animated them using a k-nearest neighbor approach. Anguelov et al. [9] and Hasler et al. [10] developed statistical human models that can create new human models with different body shapes and poses.

Linear Surface Deformation. Linear differential surface deformation methods (see, e.g., [12–14] and a well-organized survey [15]) deform a mesh by manipulating handle points or triangles, which provide position or transformation constraints, while preserving original details. The problem with these methods is the deformation of the model must be known beforehand to obtain the deformed vertex positions. Therefore, explicit methods [14, 16, 17] propagate user-specified transformations to other regions of the mesh. The problem of this approach is the translation-insensitivity: this approach cannot avoid shearing distortions when pure translations are applied, because there is no change in orientation. On the other hand, implicit methods [13] linearly

approximate transformations. This approach fails when handles are moved largely, because, in 3D, the approximation is only valid for small rotation angles. In 2D, however, we can completely determine transformations linearly. Igarashi et al. [18] exploits this fact and proposed a 2D deformation method that works for large translations, based on a two-step as-rigid-as possible deformation framework. Two-step optimization methods in 3D, based on a linear rotation-invariant mesh representation [10, 12, 19], optimize transformations first and then optimize positions. Because these methods optimize rotations and positions separately, they are also translation insensitive.

Nonlinear Surface Deformation. In contrast, nonlinear methods solve these problems by iteratively optimizing both positions and transformations and thus work for large deformations at the cost of additional computations [20–22]. The main criteria for optimizing vertex positions are rigidity, smoothness, and user-specified position constraints. The optimization problem is solved either using the Gauss-Newton method [23] or through a local/global approach [22, 24]. Kilian et al. [25], in contrast, introduced an interesting approach that formulates the optimization as a path problem instead of deforming the reference shape. To improve performance, recent methods [23, 24, 26–29] incorporate skeleton/space deformation to form a subspace. These methods are, therefore, scalable in mesh size, that is, the computational cost does not depend on the number of vertices. Consequently, they can edit a large model in real time. Our method also utilizes a subspace method, but we not only increase the speed but also compress example deformations.

Mesh-Based Inverse Kinematics. The mesh-based inverse kinematics (MeshIK) system edits a mesh by manipulating handle points while constraining deformation in example space [2]. Although this method can pose a mesh meaningfully only by moving a few handles, it is expensive to edit a large mesh due to the nonlinear optimization it solves. The computational performance of MeshIK can be improved with the use of the reduced deformable model [3], but it loses deformation effects, for example, muscle bulging. In addition, because the feature vector used in MeshIK is not rotation invariant, there is the risk of producing erroneous results if the orientations and poses of examples differ significantly, as shown in Figure 4.

Deformation Transfer. Deformation transfer [30] is another related class of techniques that literally transfers deformation of one model to another to change the pose. In fact, as pointed out by Botsch et al. [31], this method solves exactly the same Poisson system as the deformation method of [14]. Semantic deformation transfer [32] allows the transfer to drastically different characters that move in unique ways. Like our method, they interpolate a rotation-invariant mesh representation, but the interpolation weights come from the projection of a new model to the example space. In contrast, our method manipulates a surface with handle translations/rotations and computes interpolation weights from them.

Our method is most similar to the variational PSD proposed by Weber et al. [6] and MeshIK proposed by Sumner et al. [2] in the sense that it formulates an example-based deformation technique using differential surface representation. However, we use the rotation-invariant mesh representation. Thus, our method can handle large rotations robustly. In addition, we allow the user to specify both translational and rotational constraints to deform a model. This enables us to edit both global and local poses of the model.

3. Overview

Our goal is to develop a mesh editing system that can manipulate a pose of the model both globally and locally. An overview of our method is depicted in Figure 1. We provide a translational control to achieve global pose editing (Figure 1(a)) and a rotational control for local pose editing (Figure 1(b)).

Our method is based on pose space interpolation and a two-step surface optimization method. The workflow is depicted in Figure 1(c). Prior to deformation, examples are encoded and stored as the rotation-invariant mesh representation. To achieve deformation, the user selects a set of vertices or triangles called handles. If a certain region on the mesh should be kept fixed, the user specifies the fixed region (blue), which is not affected by the examples. From handle translations or rotations, we infer a pose of a model (Section 5.1). When editing with the translational control, the user specifies a coordinate frame and handle vertices on the mesh. The pose is evaluated as relative positions of handles in the local coordinate frame. When editing with the rotational control, the user places a pair of handle triangles called *virtual joint*. The pose is evaluated as relative rotations between them. Once we have the pose, we then interpolate example deformations in pose space (Section 5.2). The deformed vertex coordinates are obtained using a two-step optimization approach. First, we optimize transformations (Section 4.2.1). Next, we optimize vertex positions from the resulting transformations. Finally, we perform multiresolution deformation to edit high resolution meshes (Section 6).

3.1. Notations. Inputs to our system are the base mesh and example meshes. The base mesh $\mathcal{M}^{(0)} = \{\mathcal{V}^{(0)}, \mathcal{T}\}$ is in the rest pose, and the example meshes are provided in the form of N additional geometries $\mathcal{V}^{(1)} \dots \mathcal{V}^{(N)}$ (the connectivity $\mathcal{T}$ is shared with all the examples). Each mesh consists of n vertices and m triangles. We denote the indices of a vertex and a triangle as $p \in \{1 \dots n\}$ and $i \in \{1 \dots m\}$, respectively. We denote the index of an edge between triangle i and j by $e \in \{1 \dots l\}$, where l is the number of triangle pairs. The vertex position is defined by $\mathbf{v}_p \in \mathbb{R}^3$, and it is concatenated into an $n \times 3$ matrix, $\mathbf{v} = [\mathbf{v}_1 \dots \mathbf{v}_n]^T$. Also, we denote vertices by $\tilde{\mathbf{v}}$ after deformation.

4. Basic Deformation Framework

Our system is based on the pose space deformation framework that is thought of as a combination of surface

deformation and shape interpolation. To integrate these two effectively, we use a two-step optimization method based on a rotation-invariant mesh representation similar to those proposed in [12, 19]. This formulation makes it straightforward to incorporate both translational and rotational constraints. We refer to our rotation-invariant representation as rotation-invariant encoding that is computed from a local deformation called the deformation gradient. In this section, we first describe derivations of these representations. Next, we explain the reconstruction process of vertex positions from these representations. Finally, we introduce surface manipulation tools: handle-based surface deformation and shape interpolation.

4.1. Basic Shape Representation

4.1.1. Deformation Gradient. Consider triangle i with its three vertices before and after deformation: $[\mathbf{v}_{i1}, \mathbf{v}_{i2}, \mathbf{v}_{i3}]$ and $[\tilde{\mathbf{v}}_{i1}, \tilde{\mathbf{v}}_{i2}, \tilde{\mathbf{v}}_{i3}]$. Given edge vectors before and after deformation, $\mathbf{V}_i = [\mathbf{v}_{i2} - \mathbf{v}_{i1}, \mathbf{v}_{i3} - \mathbf{v}_{i1}]$ and $\tilde{\mathbf{V}}_i = [\tilde{\mathbf{v}}_{i2} - \tilde{\mathbf{v}}_{i1}, \tilde{\mathbf{v}}_{i3} - \tilde{\mathbf{v}}_{i1}]$, we can approximate the deformation gradient by

$$\mathbf{T}_i = \tilde{\mathbf{V}}_i \mathbf{V}_i^+, \quad (1)$$

where $\mathbf{V}_i^+$ is the pseudoinverse of edge vectors before deformation. Thus, the computation of deformation gradients $\mathbf{T} = [\mathbf{T}_1 \dots \mathbf{T}_m]^T$ can be achieved linearly from vertex positions after deformation $\tilde{\mathbf{v}} = [\tilde{\mathbf{v}}_1 \dots \tilde{\mathbf{v}}_n]^T$. Using a $3m \times n$ deformation gradient operator $\mathbf{G}$, which contains the pseudoinverses of the edge vectors before deformation, we can compute $\mathbf{T}$ by

$$\mathbf{T} = \mathbf{G} \tilde{\mathbf{v}}. \quad (2)$$

4.1.2. Rotation-Invariant Encoding. The deformation gradient is translation invariant but not rotation invariant, which means that it is affected by global rotations. To address this issue, we first decompose the deformation gradient into rotation and scale/shear components by polar decomposition, that is, $\mathbf{T}_i = \mathbf{R}_i \mathbf{S}_i$ (Figure 2). Note that the scale/shear component $\mathbf{S}_i$ is already rotation invariant, but $\mathbf{R}_i$ is not. To achieve rotation invariance, we compute the relative rotation from the absolute rotations of two adjacent triangles, $\mathbf{R}_i$ and $\mathbf{R}_j$, as follows:

$$\mathbf{Q}_{ij} = (\mathbf{R}_j)^{-1} \mathbf{R}_i. \quad (3)$$

We call $\mathbf{Q}_{ij}$ and $\mathbf{S}_i$ rotation-invariant encoding because they encode changes in surface details and are invariant under the global rotation.

4.2. Reconstruction. To reconstruct vertex positions from rotation-invariant encodings, we solve two sparse linear systems. We use the penalty method [33] to enforce rotation/position constraints. Although the use of the penalty method can only provide approximate constraint satisfaction, it can produce sufficiently close results to the exact method by setting the penalty weight sufficiently high [33]. On the other

FIGURE 1: Overview of our method. (a) Global pose editing is achieved with the translational control. (b) Local pose editing is done using the rotational control. (c) Flow of deformation.

hand, if we use small values for penalty weights, the deformed mesh does not necessarily conform to handle positions. This flexibility is useful in our pose space surface deformation framework such that the user can adjust deformations to satisfy position constraints exactly or to conform to example deformations. Using a small weight, we can avoid local distortions around handles, caused by the deviation of the pose from the interpolation range.

4.2.1. Rotation Optimization. To compute the new absolute rotation $\tilde{\mathbf{R}}_i$ from the relative encoding $\mathbf{Q}_{ij}$, we rewrite (3) as follows:

$$\mathbf{Q}_{ij}^T \tilde{\mathbf{R}}_j^T - \tilde{\mathbf{R}}_i^T = 0. \quad (4)$$

By gathering this equation for all the triangle pairs in the mesh, we can write this with a sparse linear system in the following form:

FIGURE 2: Rotation-invariant encoding. We first compute the deformation gradients. We then decompose these into absolute rotations and scale/shear components. While the scale/shear components are rotation invariant, the absolute rotations are not. Thus, we compute a relative rotation of two adjacent triangles to remove a global rotation.

$$\mathbf{H}\tilde{\mathbf{R}} = \begin{bmatrix} \cdots - \mathbf{I} \cdots \mathbf{Q}_{ij}^T \cdots \\ \vdots \\ \tilde{\mathbf{R}}_i^T \\ \vdots \\ \tilde{\mathbf{R}}_j^T \\ \vdots \end{bmatrix} = 0, \quad (5)$$

where $\mathbf{I}$ is a 3×3 identity matrix and the size of $\mathbf{H}$ is $3l \times 3m$. When achieving surface deformation, rotation constraints are applied to the set of triangles that include $\mathcal{R} \subseteq \{1 \cdots m\}$. Thus, the absolute rotations are optimized such that they satisfy both (5) and the rotation constraints. In general, there is no exact solution for this, and thus we need a least-squares approximation, that is, (5) becomes $\mathbf{H}^T \mathbf{H} \tilde{\mathbf{R}} = 0$. Using the penalty method [33], the problem is formulated as follows:

$$(\mathbf{H}^T \mathbf{H} + \omega^R \mathbf{C}^R) \tilde{\mathbf{R}} = \omega^R \mathbf{R}^C, \quad (6)$$

where ω^R is the weight of rotation constraints; $\mathbf{R}^C$ is a $3m \times 3$ matrix of rotation constraints containing transposed target rotations at $3i-2$ to $3i$ rows if $i \in \mathcal{R}$, and otherwise 0; $\mathbf{C}^R$ is a $3m \times 3m$ diagonal matrix containing 1 at $3i-2$ to $3i$ rows if $i \in \mathcal{R}$, and otherwise 0. With $\mathbf{C}^R$, we can choose actual rotations that can be constrained to target rotations as follows:

$$\mathbf{C}^R \tilde{\mathbf{R}} = \begin{bmatrix} \cdots \mathbf{I} \cdots \\ \vdots \\ \tilde{\mathbf{R}}_i^T \\ \vdots \end{bmatrix}, \quad (7)$$

where $\mathbf{I}$ is a 3×3 identity matrix. Note that $\mathbf{H}^T \mathbf{H} + \omega^R \mathbf{C}^R$ must be constructed and factorized once for every frame, which is somewhat costly for a large mesh.

Having obtained $\tilde{\mathbf{R}}$, we perform matrix orthonormalization using SVD to factor out undesired shears. We then

compute the new deformation gradient by $\tilde{\mathbf{T}}_i = \tilde{\mathbf{R}}_i \tilde{\mathbf{S}}_i$. Note that we do not allow the user to use scale constraints in our method. Thus, $\tilde{\mathbf{S}}_i$ comes only from the result of shape interpolation (Section 4.3.2). For a detail-preserving deformation, $\tilde{\mathbf{S}}_i$ is omitted (Section 4.3.1).

4.2.2. Vertex Optimization. Once the deformation gradients $\tilde{\mathbf{T}}$ are obtained, we can reconstruct vertex coordinates $\tilde{\mathbf{v}}$ by solving (2). Equation (2) is solved in a weighted least-square sense: $\mathbf{G}^T \mathbf{D} \mathbf{G} \tilde{\mathbf{v}} = \mathbf{G}^T \mathbf{D} \tilde{\mathbf{T}}$ where $\mathbf{D}$ is a $3m \times 3m$ diagonal weight matrix containing the areas of the triangles [15]. As indicated by Botsch et al. [31], $\mathbf{G}^T \mathbf{D} \mathbf{G}$ is equivalent to the Laplace Beltrami operator $\mathbf{L}$, which can be computed directly from the cotangent weights [34]. In contrast to $\mathbf{H}^T \mathbf{H}$, $\mathbf{L}$ is factorized once for all the frames, which requires only back substitution and is, therefore, very efficient. When a set of position constraints, $\mathcal{P} \subseteq \{1 \cdots n\}$, is provided, this is solved via the penalty method, which leads to the following system:

$$(\mathbf{G}^T \mathbf{D} \mathbf{G} + \omega^P \mathbf{C}^P) \tilde{\mathbf{v}} = \mathbf{G}^T \mathbf{D} \tilde{\mathbf{T}} + \omega^P \mathbf{v}^C, \quad (8)$$

where ω^P is the weight for position constraints; $\mathbf{v}^C$ is a $n \times 3$ matrix containing target positions at row p if $p \in \mathcal{P}$, and otherwise 0; $\mathbf{C}^P$ is a $n \times n$ diagonal matrix having 1 at row p if $p \in \mathcal{P}$, and otherwise 0. Like $\mathbf{C}^R$, we can choose vertices that should be constrained to the target positions using $\mathbf{C}^P$ as follows:

$$\mathbf{C}^P \tilde{\mathbf{v}} = \begin{bmatrix} \cdots \mathbf{I} \cdots \\ \vdots \\ \tilde{\mathbf{v}}_p \\ \vdots \end{bmatrix}. \quad (9)$$

4.3. Surface Manipulation Tools. In this section, we introduce two basic surface manipulation tools for formulating our pose space deformation.

4.3.1. Handle-Based Surface Deformation. To deform a surface, we specify handle vertices/triangles on the mesh and apply translations/rotations to them (Figure 3). In addition, if a certain region on the mesh should be kept fixed, the user specifies the fixed region. To obtain the deformed vertices $\tilde{\mathbf{v}}$, we solve sparse linear systems (6) and (8). Note that, because we optimize rotations and vertices separately, rotation constraints and translation constraints should be compatible with each other, otherwise, it produces distortions.

4.3.2. Shape Interpolation. The advantage of performing interpolation using the rotation-invariant encoding is that it is robust to large rotations. However, when simply interpolating the components of $\mathbf{Q}_{ij}$, it leads to the artifacts which can be seen in the linear blend skinning. One practical way to correctly interpolating rotations is to map them to $\text{so}(3)$ using the matrix logarithm, interpolate linearly in $\text{so}(3)$, and map back to $\text{SO}(3)$ using the matrix exponential [2, 7, 10, 17, 32].

FIGURE 3: Handle-based surface deformation. We manipulate handle vertices/triangles on the surface by applying translations/rotations to them. The mesh deforms according to handle movements while preserving the original details. Note that, because we optimize rotations and vertices separately, rotation constraints and translation constraints should be compatible with each other, otherwise, it produces distortions. Without incorporating examples, our method deforms the model smoothly regardless of anatomical structures.

To achieve this, we use Rodrigues' formula and convert $\mathbf{Q}_{ij} \in \mathbb{R}^{3 \times 3}$ into an axis angle representation (rotation vector) $\mathbf{q}_{ij} \in \mathbb{R}^3$ which is a compact form of $\log \mathbf{Q}_{ij} \cdot \mathbf{S}_i$, however, can be interpolated directly. Note that, when linearly interpolating matrix logarithms, the interpolation path is not the shortest path, which introduces error. This error is minimal when the rotation is small, that is, the rotation is close to the identity. Therefore, we encode deformation gradients and rotation-invariant encodings relative to the rest pose [6, 32]. As a result, when we encode the rest pose, $\mathbf{Q}_{ij}$ is the identity for all triangle pairs.

We concatenate $\mathbf{q}_{ij}$ into a $3 \times l$ matrix, $\mathbf{q}$. Likewise, a symmetric matrix $\mathbf{S}_i \in \mathbb{R}^{3 \times 3}$ is flattened to $\mathbf{s}_i \in \mathbb{R}^6$ and concatenated into a $6 \times m$ matrix, $\mathbf{s}$. Interpolation of the rotation-invariant encodings of N meshes is achieved with a simple weighted combination. The interpolated values $\tilde{\mathbf{s}}$ and $\tilde{\mathbf{q}}$ are obtained from the following:

$$\begin{aligned} \tilde{\mathbf{s}} &= \sum_{P=1}^N \alpha^{(P)} \mathbf{s}^{(P)}, \\ \tilde{\mathbf{q}} &= \sum_{P=1}^N \alpha^{(P)} \mathbf{q}^{(P)}, \end{aligned} \quad (10)$$

where $\alpha^{(P)}$ is the weight for example P , which satisfies $\sum_{P=1}^N \alpha^{(P)} = 1$. After converting $\tilde{\mathbf{s}}$ and $\tilde{\mathbf{q}}$ to matrix form, the deformed vertex positions $\tilde{\mathbf{v}}$ are obtained by solving (6) and (8). We constrain one of the rotations to an identity matrix when solving (6) and fix one of the positions when solving (8).

Interpolation Properties. The rotation-invariant encoding is robust-to-extreme orientation differences of input models [12, 19, 32]. To show this, we compared the interpolation results using rotation-invariant encodings and using deformation gradients (Figure 4). Given two meshes, $\mathcal{M}^{(1)}$ and $\mathcal{M}^{(2)}$, which are in extremely different orientations, we create the intermediate steps between them. Let $\mathbf{s}^{(1)}$, $\mathbf{q}^{(1)}$, and $\mathbf{s}^{(2)}$, $\mathbf{q}^{(2)}$ be rotation-invariant encodings of two meshes. Then, we can perform interpolation with $\tilde{\mathbf{s}} = \alpha^{(1)} \mathbf{s}^{(1)} + (1 - \alpha^{(1)}) \mathbf{s}^{(2)}$ and

FIGURE 4: Comparison of interpolation results. We perform interpolation on two meshes in different orientations: $\mathcal{M}^{(1)}$ and $\mathcal{M}^{(2)}$. The interpolation result using deformation gradients (Def Grad) is distorted because there is an ambiguity for interpolation if the orientations of the triangles differ by more than 180 degrees. The result obtained using the rotation-invariant encoding (Rot Inv) does not suffer from this ambiguity.

$\tilde{\mathbf{q}} = \alpha^{(1)} \mathbf{q}^{(1)} + (1 - \alpha^{(1)}) \mathbf{q}^{(2)}$. The interpolation of deformation gradients is performed in the same manner except that the rotations are represented in absolute world coordinates. If triangle orientations differ by more than 180 degrees, there is ambiguity in interpolating deformation gradients. Because of this ambiguity, the result using deformation gradients is distorted around the foot, as shown in Figure 4. Shape interpolation using rotation-invariant encodings does not suffer from this ambiguity and produces a natural result.

5. Pose Space Surface Manipulation

Now, we are ready to formulate our pose space surface manipulation technique by unifying the handle-based surface deformation and shape interpolation methods described in the previous section. To integrate these two, rotation-invariant encodings must be altered according to handle movements, that is, a technique to associate handle translations/rotations with interpolation weights must be developed.

5.1. Pose Estimation. To obtain interpolation weights, the original MeshIK [2] solves nonlinear optimization to determine the weights from handle positions. We, in turn, compute interpolation weights from the handle translations/rotations based on pose space interpolation. This allows us to avoid involved nonlinear optimization and to incorporate both translational/rotational controls.

To evaluate the pose of the model, we need the measure that is rotation invariant. To this end, we compute relative positions and rotations from handles (Figure 5).

When editing with the translational control, the user places a coordinate frame and handle vertices on the mesh. The frame is placed on a triangle that exists around the trunk of the model and is used to form a local coordinate to compute relative positions of handles. This way, we can eliminate the effect of global rotations and can evaluate the pose successfully. In addition, our system allows the user to manipulate global orientation and position of the model by rotating and translating the frame.

FIGURE 5: We infer a pose of a model from relative positions and rotations computed from handle translations/rotations.

When editing with the rotational control, the user places a pair of handle triangles called *virtual joint*. The virtual joint consists of *parent*, which is placed on the proximal segment, and *child*, which is placed on the distal segment. The user applies rotations to the child triangle whereas the parent triangle is not manipulated. To apply rotations, the user places a frame on a triangle that exists near the actual joint center in order to use its centroid as a rotation center. By rotating the frame, we can achieve twisting easily. By dragging the child triangle around the frame, bending is achieved. The pose is evaluated as a relative rotation between the virtual joint, which is represented as Euler angles.

5.2. Pose Space Deformation. Next, we compute the interpolation weights from the pose using a scattered data interpolation method. The interpolation weights are obtained by evaluating the pose computed from handle translations/rotations, in the pose space formed by example poses. We would like to compute the interpolation weights such that they satisfy the following criteria.

- (1) At an example point, the weight for that example must be one, and all other weights must be zero.
- (2) The weights must add up to 1.
- (3) The weight must change continuously to produce smooth animation.
- (4) The absolute value of the weight must be small to avoid excessive exaggeration of deformation.

For this task, we use k -nearest-neighbors interpolation [8] because it satisfies all of the above criteria and returns the weights within the range [0–1]. This method chooses the k closest example points and assigns weights based on the proximity (distance). All other example points are assigned a weight of zero. Before normalization, the weights for each examples are computed as

$$\alpha^{(P)} = \left(1 - \frac{d^{(P)}}{d_{\max}}\right)^2. \quad (11)$$

This is then normalized to sum to one. $d^{(P)}$ is the distance to the P th example and the $d_{\max}$ is the distance to the $k + 1$ nearest point. We use $k = 6$ in this paper.

Let $K \in \{1 \dots M\}$ be the index of handle vertex or virtual joint. Then, the pose is defined by a $3M \times 1$ vector, $\mathbf{f}$, that contains relative positions or relative rotations. The distance between an arbitrary pose and example P is defined as follows:

$$d^{(P)} = \left(\|\mathbf{f}^{(P)} - \mathbf{f}\|^2\right)^{1/2}, \quad (12)$$

where $\mathbf{f}^{(P)}$ is a pose of example P .

After computing interpolation weights, we use (10) to obtain new rotation-invariant encodings. The new vertices are calculated by solving (6) and (8). When editing with translational controls, we constrain the rotation of triangle where the frame is placed, to the frame orientation relative to the rest pose. When editing with rotational controls, at least one vertex must be fixed to provide position constraints.

6. Multiresolution Deformation

By incorporating examples, our method can edit the model naturally and capture deformation effects such as muscle bulging. However, solving two large linear systems is still relatively time consuming, mainly due to the rotation optimization, which requires performing factorization for every frame. In addition, it requires a relatively large memory space ($3l$ for $\mathbf{q}$ and $6m$ for $\mathbf{s}$). To make our representation more compact and to speed up the computation, we use a multiresolution approach.

Our observation is that, if we layer a surface with a coarse mesh and fine details, then deformation induced by pose changes mainly affects the underlying coarse mesh and has little effect on the fine details. In fact, Weber et al. [6] reported that deformation behaviors of a character tend to have a low-frequency nature. They then used this fact to make example deformations more compact using a method similar to the geometry compression proposed in [35]. We, in turn, take a multiresolution approach using coarse meshes to compress example deformations. We compute rotation-invariant encodings from coarse meshes and perform PSD with them. We then add original fine details to the deformed coarse mesh to obtain a dense output model.

Inspired by Sumner et al. [23], we employ linear blend skinning to decouple a dense mesh into a coarse mesh and fine-details. Here, we represent the vertex position of the coarse mesh by $\mathbf{v}_k$, an affine transformation by $\mathbf{T}_k \in \mathbb{R}^{3 \times 3}$, and a translation vector by $\mathbf{b}_k \in \mathbb{R}^3$. The deformed vertex $\tilde{\mathbf{v}}_p$ is obtained as follows:

$$\tilde{\mathbf{v}}_p = \sum_{k=1}^c w_p^{(k)} \left[\mathbf{T}_k (\mathbf{v}_p - \mathbf{v}_k) + \mathbf{v}_k + \mathbf{b}_k \right], \quad (13)$$

where c is the number of vertices in the coarse mesh, and $w_p^{(k)}$ is a weight for vertex p . $w_p^{(k)}$ represents the extent to which $\mathbf{v}_p$ is influenced by $\mathbf{v}_k$.

To compute the weight $w_p^{(k)}$, we use harmonic interpolation [16] as opposed the Euclid distance-based weighting method [23]. This eliminates distortions caused by assigning large weights to the vertices that are geodesically distant. As

FIGURE 6: Editing a horse (1k triangles) with translations and rotations. By translating the handles on the feet, we can pose the horse naturally (a). Once the overall pose is determined, the user edit the pose locally with the rotational control, if required (b). The blue region is frozen so that deformations remain local.

FIGURE 7: With 6 coarse examples (1k triangles) (a), we can create walking elephant (30k triangles) by translating handles on the feet (b).

for the boundary condition, vertex k is assigned the value $\mathbf{w}_p^{(k)} = 1$, and all other vertices in the coarse mesh are assigned the value $\mathbf{w}_p^{(k)} = 0$. If the harmonic value is $\mathbf{w}_p^{(k)} < 0.01$, then we replace it with 0 so that we can reduce memory space. The values are then normalized such that their sum is 1.

To construct a coarse mesh, we use a variant of the quadratic mesh simplification method (Qslim) [36]. Because Qslim only considers a static shape, the resulting coarse mesh is not suitable for deformation. For example, if we apply Qslim to a character model with its knee extended, then we would lose vertices around the knee. This loss would be a problem when we want to bend the knee. To solve this problem, we modify Qslim slightly by computing the error metric using all the examples. This method is similar to that proposed in [37], but we do not move the vertices during edge collapse; thus, we can easily specify boundary conditions of harmonic interpolation.

Incorporating the above technique into the pose space surface manipulation framework is straightforward. We prepare coarse examples and store their rotation-invariant encodings. We also store harmonic weights and fine details that are the vectors emanating from the vertex of a coarse mesh to that of a dense mesh, $\mathbf{v}_p - \mathbf{v}_k$. We then perform PSD using those values. Once the result using coarse meshes is obtained, fine details are added using (13). Unlike [23], who place them on the dense mesh, we place handles directly on the coarse mesh. However, we found that the manipulation

of handles on a coarse mesh of around 1k triangles retains sufficient freedom for interactive mesh editing.

7. Results

We implemented the pose space surface manipulation technique on an Intel Core2 Quad Q9400 2.66 GHz machine. We used the sparse Cholesky solver provided with the CHOLMOD [38] library. We found that the range $\omega^R = [10^4 - 10^8]$ worked well for all the examples presented in this paper. In contrast, we provide a wider range of $\omega^P = [10^{-2} - 10^8]$.

Interactive Editing. In the accompanying video, we show use cases of the pose space manipulation framework for interactive editing of mesh models. Figure 6 demonstrates how our method edits a model using both translational and rotational control. By translating the handles on the feet, we can pose the horse (1k triangles) naturally (Figure 6(a)). Once the overall pose is determined, the user manipulates the part of the model with the rotational control (Figure 6(b)). The blue region is frozen so that deformations remain local.

Using our translational editing method, we can create animation sequences. As shown in Figure 7, we can create walking elephant (30k triangles) from 6 coarse examples (1k triangles) by translating handles on the feet. We can also pose the performer (dense: 20k triangles, coarse: 1k triangles)

FIGURE 8: Posing of the performer model (dense: 20k triangles, coarse: 1k triangles). We place a frame on the back and handles on the feet (a). We first adjust the global orientation of the performer (b). We can pose the performer in a running sequence by translating the handles (c).

FIGURE 9: Editing of highly nonrigid objects. (a) We provide 14 examples (1k triangles) to edit a cloth. (b) By just dragging the lower right corner of the model to apply translations, we can animate the cloth realistically.

in a running sequence by moving the handles on the feet (Figure 8).

Figure 9 demonstrates the applicability of our method to highly nonrigid objects such as a cloth. We used 14 examples (1k triangles) created by a simulation. We fixed the top edge of a model and placed a frame on the top-left corner. By just dragging the handle on the bottom right corner, we can animate the cloth realistically.

Figure 10 demonstrates the ability of our method to accomplish twisting. We placed handle triangles on the upper arm and shoulder blade. We manipulate the frame placed on the triangle which is near the joint center. The rotational control is more intuitive for controlling shoulder movements than the translational control. Even for these challenging cases involving twisting and bending, our method can produce natural results.

Handle Locations and Sizes. In Figure 11, we evaluated the sensitivity of our method to the locations of triangles. Our method works when the handle triangle is distally or proximally placed in the segment (Figure 11(b) and 11(c)). Even when the handle is placed on the region where it is deformed by muscle bulging, pose space interpolation was successfully achieved (Figure 11(d)). Although we recommend placing the triangles in the middle of the segments, our method works robustly as long as child and parent triangles are placed on different segments. Our method works even when the triangle is placed far from the virtual joint. However, it is somewhat difficult for the user to manipulate the triangle in this case; because joints exist between the triangle and the virtual joint, the distance between the triangle and the

virtual joint changes during dragging and the model will deform unexpectedly. In this case, the translational control is more suitable and easy to use. In addition, the size of the triangle is also important. If the size of the triangle is so small and its orientation changes drastically with high-frequency deformations, then the pose space interpolation will probably introduce large approximation errors. For the examples presented in this paper, however, pose space interpolation is successfully achieved because we manipulate relatively large triangles that is on a coarse mesh with around 1k triangles.

Comparisons. In Figure 12, we compare our method using rotational controls with the transformation propagation approach of Zayer et al. [16]. As expected, our method is more realistic where the elbow bends sharply, and the biceps bulges naturally, whereas the transformation propagation method just deforms the model smoothly. In Figure 13, we compared our method with the material modulation approach [17]. Although material modulation exploits the examples, it is difficult to control details precisely because it only computes a weight per vertex from all examples and does not account for pose changes. The result of our method looks nearly identical to the ground truth. We also quantitatively compared our method with [17]. We measured the error defined by the mean distance between the deformed model and the ground truth, which was normalized by the bounding box diagonal. Our method is more accurate than [17] where the error of our method is approximately 1.5% of the bounding box diagonal whereas that of [17] is 5.8%.

FIGURE 10: Twisting of the shoulder. We provide 9 examples (a) and are able to obtain natural results using the rotational control (b). The rotational control is more intuitive for controlling shoulder movements than the translational control.

FIGURE 11: Locations of triangles. Our method works even when the handle triangle is placed distally (b), proximally (c), or on the region where a muscle bulges (d).

FIGURE 12: Comparison with the transformation propagation approach [16]. Our method produces a more realistic result because the elbow bends sharply and the biceps bulges naturally. In contrast, the transformation propagation method deforms the model smoothly.

Next, we compared our method using translational controls with MeshIK [2]. When the pose is in the example space, our method is on par with MeshIK (compare Figure 14 of this paper and Figure 5 of [2]). As with MeshIK, our method fails when the pose is outside of the example space (Figure 14(a)). The advantage of our method is the applicability to the examples with large bending (Figure 15); this is important when editing models that have tails, tentacles, and

FIGURE 13: Comparison with the material modulation method [17]. Although material modulation produces a plausible result, it is still difficult to control details precisely. Our method is nearly identical to the ground truth.

FIGURE 14: Edits of a bar model with the translational control. (a) We first provide two examples (gray). The shape in-between can be created meaningfully. However, it causes shearing distortions when we deform the bar in the vertical direction. (b) By providing another example in this direction, we can produce a natural result. (c) Because the distance-based interpolation method lacks the extrapolation capability, local distortions around handles occur when the pose is outside of the interpolation range.

so forth. Since MeshIK uses deformation gradient (Def Grad) as its basic representation, which is defined in the absolute world coordinate, it results in the discontinuity artifact (Figure 15 top right). Our method based on a rotation-invariant mesh representation naturally deforms the model using the examples that involve large rotations and bending (Figure 15 bottom). The downside of our method is the lack of extrapolation capability due to the use of the distance-based interpolation method, which violates the smoothness around handles (Figure 14(c)). This can be avoided by incorporating an adequate number of example poses with extreme poses, since our method accepts examples with large

FIGURE 15: If we use deformation gradient (Def Grad) that was used in MeshIK [2], our method results in the discontinuity artifact because Def Grad is defined in the absolute world coordinate (top right). Our method based on a rotation-invariant mesh representation (Rot Inv) naturally deforms the model using the examples that involve large rotations and bending (bottom).

FIGURE 16: The effect of using a small weight for position constraints. Using a small value for w^p , we can avoid violating the smoothness around the handle.

bending. Another way to solve this problem is to use a small value for w^p such that the deformed model conforms to example deformations by not obeying handle positions exactly (Figure 16).

Memory Consumptions. Storing rotation-invariant encodings requires a relatively large amount of memory. To solve this problem, we propose a multiresolution approach. In Figure 17, we show the morphing results of Horse in several mesh resolutions. With only 1k triangles, we can generate a result that is nearly identical to that of the original resolution (17k triangles). Using our multiresolution technique, the memory consumptions in this case can be reduced to at least 6% of the original without degrading the quality of the output.

Performance. In Table 1, we show the performance results. The most time consuming processes in our method are the

FIGURE 17: Interpolation results using coarse meshes. We perform interpolation on several mesh resolutions. Top: interpolation of the original model (17k triangles). Middle: the interpolation results of coarse meshes (4k, 2k and 1k triangles). Bottom: comparison of the dense results. The results are nearly identical for all the resolutions.

rotation optimization and the subsequent matrix orthonormalization. Because the matrix $\mathbf{H}$ is constructed from relative rotations computed after interpolation, the matrix $\mathbf{H}^T \mathbf{H}$ must be factorized for every frame. For the rotation optimization of 17k mesh, construction and factorization of $\mathbf{H}^T \mathbf{H}$, and back substitution take 0.21 sec, 0.25 sec, and 0.027 sec, respectively. The matrix orthonormalization of absolute rotations using SVD also takes a relatively long time, which is about 0.5 sec for a 17k mesh. However, the vertex optimization is done very efficiently because the construction and factorization of $\mathbf{L}$ is required once for all the frames, and we execute only back substitution during runtime. To summarize, without the multiresolution method, our method requires approximately 1 second to edit a 17k mesh.

Using the multiresolution method, we can edit a mesh in an interactive rate. Our method requires approximately 0.1 sec to edit a 17k mesh and 0.2 sec to edit a 140k mesh. This is a significant speedup over the original MeshIK that requires approximately 1 sec to solve a *single* Gauss-Newton iteration to edit an 80k mesh [2].

Limitations. Our method has limitations that need to be overcome. Although the use of the multiresolution method helps achieving interactive rates, fine details remain static. It is, therefore, difficult for our method to model complex deformation, for example, facial expressions generated by muscle contractions. Our method is probably suitable for large- and medium-scale deformation, and it is rather unfit for modeling small scale surface-detail deformations. Also, because we assign one interpolation weight for each example, our method do not accept the user to place many handles, for example, more than 10 or 20 handles. We believe that our global-to-local editing strategy based on

TABLE 1: Timing comparisons of each step with and without the multiresolution method (in seconds).

	Mesh size	No. Ex.	Interp. Rot Inv	Rotation optimization			Vertex optimization			Matrix ortho.	Coarse-to-Dense
				Const.	Factor	Solve	Const.	Factor	Solve		
w/o multires.	17k	10	0.12	0.21	0.25	0.027	[0.78]	[0.023]	0.003	0.50	—
With multires.	1k	10	0.01	0.01	0.01	0.001	[0.06]	[0.002]	0.001	0.03	0.008 (17k), 0.075 (140k)

From left to right: Mesh size, number of examples, interpolation of rotation-invariant encodings, rotation optimization (construction and factorization of $\mathbf{H}^T \mathbf{H}$, and back substitution), vertex optimization (construction and factorization of $\mathbf{L}$, and back substitution), matrix ortho-normalization using SVD, conversion of a coarse mesh to a dense mesh. Timings in brackets are done during the preprocessing stage.

translational/rotational controls works well for interactive posing of the model without using many handles, which can create a new shape in a wide range of poses. However, in the case where simultaneous manipulations of a large number of handles are required, for example, deforming the model using motion-capture data points, our method would fail to approximate deformations. This problem could be alleviated by using the weighted pose space deformation framework [5] to compute weights for each vertex instead of each example. Finally, our method lacks the extrapolation capability, which causes shear distortions when the pose is outside of the interpolation range. The ways to solve this problem may be to use a small weight for position constraints or to use examples with extreme poses.

8. Conclusion

Pose space surface manipulation is a novel example-based mesh editing technique that can deform the model naturally and that is able to adjust a pose locally. We provide both translational and rotational constraints to achieve direct manipulation of a surface to pose the model. Our method is reasonably efficient because, by inferring a pose from handles and by performing pose space interpolation, we are able to solve the problem with two linear systems, which avoids involved nonlinear optimization. With this two-step linear approach combined with the multiresolution deformation method, we achieve interactive rates without losing important deformation effects such as muscle bulging. The performance of our method would be further improved by a GPU implementation [11, 39]. It would be interesting to extend our method to edit other surface representations, such as multicomponent objects.

Acknowledgment

The authors would like to thank Brett Allen for the Arm and Shoulder datasets, Robert W. Sumner for the Elephant, Lion, Horse, and Cloth models, and Daniel Vlasic for the Performer model.

References

- [1] J. P. Lewis, M. Cordner, and N. Fong, “Pose space deformation: a unified approach to shape interpolation and skeleton-driven deformation,” in *Proceedings of the SIGGRAPH*, pp. 165–172, July 2000.
- [2] R. W. Sumner, M. Zwicker, C. Gotsman, and J. Popovic, “Mesh-based inverse kinematics,” in *Proceedings of the ACM SIGGRAPH*, pp. 488–495, August 2005.
- [3] K. G. Der, R. W. Sumner, and J. Popovic, “Inverse kinematics for reduced deformable models,” in *Proceedings of the ACM SIGGRAPH*, pp. 1174–1179, August 2006.
- [4] P. G. Kry, D. L. James, and D. K. Pai, “Eigenskin: real time large deformation character skinning in hardware,” in *Proceedings of the ACM SIGGRAPH Symposium on Computer Animation (SCA '02)*, pp. 153–159, July 2002.
- [5] T. Kurihara and N. Miyata, “Modeling deformable human hands from medical images,” in *Proceedings of the ACM SIGGRAPH Eurographics Symposium on Computer Animation (SCA '04)*, pp. 355–363, 2004.
- [6] O. Weber, O. Sorkine, Y. Lipman, and C. Gotsman, “Context-aware skeletal shape deformation,” *Computer Graphics Forum*, vol. 26, no. 3, pp. 265–274, 2007.
- [7] R. Y. Wang, K. Pulli, and J. Popovic, “Real-time enveloping with rotational regression,” *ACM Transactions on Graphics*, vol. 26, no. 3, Article ID 1276468, 2007.
- [8] B. Allen, B. Curless, and Z. Popovic, “Articulated body deformation from range scan Data,” *ACM Transactions on Graphics*, vol. 21, no. 3, pp. 612–619, 2002.
- [9] D. Anguelov, P. Srinivasan, D. Koller, S. Thrun, J. Rodgers, and J. Davis, “Scape: shape completion and animation of people,” *ACM Transactions on Graphics*, vol. 24, no. 3, 2005.
- [10] N. Hasler, C. Stoll, M. Sunkel, B. Rosenhahn, and H. P. Seidel, “A statistical model of human pose and body shape,” *Computer Graphics Forum*, vol. 28, no. 2, pp. 337–346, 2009.
- [11] B. Bickel, M. Lang, M. Botsch, M. Otaduy, and M. Gross, “Pose-space animation and transfer of facial details,” in *Proceedings of the ACM SIGGRAPH Eurographics Symposium on Computer Animation*, pp. 57–66, July 2008.
- [12] Y. Lipman, O. Sorkine, D. Levin, and D. Cohen-Or, “Linear rotation-invariant coordinates for meshes,” in *Proceedings of the ACM SIGGRAPH*, pp. 479–487, August 2005.
- [13] O. Sorkine, D. Cohen-Or, Y. Lipman, M. Alexa, C. Ross, and H. P. Seidel, “Laplacian surface editing,” in *Proceedings of the 2nd Symposium on Geometry Processing (SGP '04)*, pp. 175–184, July 2004.
- [14] Y. Yu, K. Zhou, D. Xu et al., “Mesh editing with poisson-based gradient field manipulation,” *ACM Transactions on Graphics*, vol. 23, no. 3, pp. 644–651, 2004.
- [15] M. Botsch and O. Sorkine, “On linear variational surface deformation methods,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, no. 1, pp. 213–230, 2008.
- [16] R. Zayer, C. Ross, Z. Karni, and H. P. Seidel, “Harmonic guidance for surface deformation,” in *Proceedings of the Eurographics*, 2005.

- [17] T. Popa, D. Julius, and A. Sheffer, "Material-aware mesh deformations," in *Proceedings of the IEEE International Conference on Shape Modeling and Applications (SMI '06)*, p. 22, June 2006.
- [18] T. Igarashi, T. Moscovich, and J. F. Hughes, "As-rigid-as-possible shape manipulation," *ACM Transactions on Computer Graphics*, vol. 24, no. 3, 2005.
- [19] S. Kircher and M. Garland, "Free-form motion processing," *ACM Transactions on Graphics*, vol. 27, no. 2, pp. 1–13, 2008.
- [20] O. K. C. Au, C. L. Tai, L. Liu, and H. Fu, "Dual laplacian editing for meshes," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 3, pp. 386–395, 2006.
- [21] V. Kraevoy and A. Sheffer, "Mean-value geometry encoding," *International Journal of Shape Modeling*, vol. 12, no. 1, pp. 29–46, 2006.
- [22] O. Sorkine and M. Alexa, "As-rigid-as-possible surface modeling," in *Proceedings of the EUROGRAPHICS/ACM SIGGRAPH Symposium on Geometry Processing (SGP '07)*, 2007.
- [23] R. W. Sumner, J. Schmid, and M. Pauly, "Embedded deformation for shape manipulation," *ACM Transactions on Graphics*, vol. 26, no. 3, Article ID 1276478, 2007.
- [24] M. Ben-Chen, O. Weber, and C. Gotsman, "Variational harmonic maps for space deformation," *ACM Transactions on Graphics*, vol. 28, no. 3, 2009.
- [25] M. Kilian, N. J. Mitra, and H. Pottmann, "Geometric modeling in shape space," *Acm Transactions on Graphics*, vol. 26, no. 3, Article ID 1276457, 2007.
- [26] O. K. C. Au, H. Fu, C. L. Tai, and D. Cohen-Or, "Handle-aware isolines for scalable shape editing," *Acm Transactions on Graphics*, vol. 26, no. 3, Article ID 1276481, 2007.
- [27] M. Botsch, M. Pauly, M. Gross, and L. Kobbelt, "PriMo: coupled prisms for intuitive surface modeling," in *Proceedings of the Eurographics Symposium on Geometry Processing (SGP '06)*, pp. 11–20, 2006.
- [28] J. Huang, X. Shi, X. Liu et al., "Subspace gradient domain mesh deformation," in *Proceedings of the ACM SIGGRAPH*, pp. 1126–1134, August 2006.
- [29] X. Shi, K. Zhou, Y. Tong, M. Desbrun, H. Bao, and B. Guo, "Mesh puppetry: cascading optimization of mesh deformation with inverse kinematics," *ACM Transactions on Graphics*, vol. 26, no. 3, Article ID 1276479, 2007.
- [30] R. W. Sumner and J. Popovic, "Deformation transfer for triangle meshes," in *Proceedings of ACM SIGGRAPH*, pp. 399–405, August 2004.
- [31] M. Botsch, R. Sumner, M. Pauly, and M. Gross, "Deformation transfer for detail-preserving surface editing," in *Proceedings of the Vision, Modeling and Visualization*, pp. 357–364, 2006.
- [32] I. Baran, D. Vlastic, E. Grinspun, and J. Popovic, "Semantic deformation transfer," *ACM Transactions on Graphics*, vol. 28, no. 3, 2009.
- [33] K. Xu, H. Zhang, D. Cohen-Or, and Y. Xiong, "Dynamic harmonic fields for surface processing," *Computers and Graphics*, vol. 33, no. 3, pp. 391–398, 2009.
- [34] U. Pinkall and K. Polthier, "Computing discrete minimal surfaces and their conjugates," *Experimental Mathematics*, vol. 2, no. 1, pp. 15–36, 1993.
- [35] O. Sorkine, D. Cohen-Or, D. Irony, and S. Toledo, "Geometry-aware bases for shape approximation," *IEEE Transactions on Visualization and Computer Graphics*, vol. 11, no. 2, pp. 171–180, 2005.
- [36] M. Garland and P. S. Heckbert, "Surface simplification using quadric error metrics," in *Proceedings of the SIGGRAPH Conference on Computer Graphics*, pp. 209–216, August 1997.
- [37] A. Mohr and M. Gleicher, "Deformation sensitive decimation," Tech. Rep., 2003.
- [38] Y. Chen, T. Davis, W. Hager, and S. Rajamanickam, "Algorithm 8xx: CHOLMOD, supernodal sparse cholesky factorization and update/downdate," Tech. Rep. TR-2006-005, 2006.
- [39] T. Rhee, J. P. Lewis, and U. Neumann, "Real-time weighted pose-space deformation on the gpu," in *Proceedings of the Eurographics*, 2006.