Multiresolution Mean Shift Clustering Algorithm for Shape Interpolation

Hung-Kuo Chu and Tong-Yee Lee, *Member*, *IEEE*

Abstract—In this paper, we solve the problem of 3D shape interpolation with significant pose variation. For an ideal 3D shape interpolation, especially the articulated model, the shape should follow the movement of the underlying articulated structure and be transformed in a way that is as rigid as possible. Given input shapes with compatible connectivity, we propose a novel multiresolution mean shift (MMS) clustering algorithm to automatically extract their near-rigid components. Then, by building the hierarchical relationship among extracted components, we compute a common articulated structure for these input shapes. With the aid of this articulated structure, we solve the shape interpolation by combining 1) a global pose interpolation of near-rigid components from the source shape to the target shape with 2) a local gradient field interpolation for each pair of components, followed by solving a Poisson equation in order to reconstruct an interpolated shape. As a result, an aesthetically pleasing shape interpolation can be generated, with even the poses of shapes varying significantly. In contrast to a recent state-of-the-art work [19], the proposed approach can achieve comparable or even better results and have better computational efficiency as well.

Index Terms—Shape interpolation, pose configuration, multiresolution mean shift (MMS) clustering.

1 Introduction

Shape interpolation has been an active research area in computer graphics and is a powerful technique in computer animation and entertainment. It aims at producing a gradually and naturally changing of transformation between two or more existing shapes. Generally, there are two major steps for polyhedral-surface shape interpolation: 1) establishing one-to-one correspondence among the input models and 2) interpolating the positions of corresponding vertices, known as the *trajectory problem*, to compute intermediate shapes. While the correspondence establishment has been thoroughly investigated by many previous works [2], [18], [20], [28], [30], the trajectory problem is simply realized by linearly interpolating corresponding vertices in their studies. It is well known that a naïve linear vertex interpolation [2], [18], [20], [30], [40], [41] potentially suffers the shrinkage problem (Fig. 1a), because the large-scale rotations cannot be correctly expressed by linear interpolation. Some researchers perform a global rigid [10], [13] or affine transform [2] prior to linear vertex interpolation and obtain better results. However, these approaches still cannot completely solve the problem. To solve the trajectory problem, Alexa et al. [4] propose a technique that is as rigid as possible in order to interpolate the local transformations of the interior triangles of the 2D shape by minimizing the distortion of the rigid transformation during the shape interpolation. Xu et al. [38] follow a similar theoretical basis and propose nonlinear gradient field interpolation to control orientations of surface triangles implicitly in 3D shape interpolation (Fig. 1b). Because the surface triangles are considered independently and locally, this technique potentially fails when the poses of input shapes vary significantly (Fig. 2a). Even though a global rigid transformation is applied to source shape before gradient field interpolation, it still generates artifacts (Fig. 2b).

In the proposed approach, the key for achieving a natural interpolation of articulated shapes (Fig. 2c) is decomposing the vertex trajectories into a *rigid* pose transformation and a *nonrigid* residual transformation. The rigid pose transformation is described in terms of hierarchical rigid transformations from the underlying articulated structure. The nonrigid residual transformation captures the local detail deformation. The two major contributions of this paper are listed below:

- We propose a novel multiresolution mean shift (MMS) clustering algorithm to automatically and efficiently extract a hierarchical, articulated structure which can be used to describe various poses of input shapes (Section 4).
- Using this articulated structure, we solve the trajectory problem by combining a global pose transformation with a local detail transformation, followed by solving a Poisson equation (Section 5).

In Section 6, we demonstrate the success of our method using several aesthetically pleasing shape interpolations with significant pose variation. We also show that our method is applicable to two useful applications: 1) multi-target shape blending and 2) keyframe animation.

2 Background

Several previous works have been proposed to study the trajectory problem in shape interpolation. A naïve linear vertex interpolation is the most popular and simplest proposal, but it suffers from artifacts such as shape shrinkage. To avoid shrinkage, Sederberg et al. [31] interpolate both edge length and dihedral angle information of the boundary of 2D polygons rather than vertex positions. Alexa et al. [4] formulate the vertex trajectory as an optimization of the rigid transformation of interior simplicial complexes between two

- *The authors are with Computer Graphics Group/Visual System Laboratory, Department of Computer Science and Information Engineering, National Cheng-Kung University, No. 1, Ta-Hsueh Road, Tainan 701, Taiwan, ROC. E-mail: hkchu@csie.ncku.edu.tw, tonylee@mail.ncku.edu.tw.*

Manuscript received 26 June 2008; revised 9 Oct. 2008; accepted 18 Mar. 2009; published online 24 Mar. 2009.
Recommended for acceptance by H. Qin.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number TVCG-2008-06-0082.
Digital Object Identifier no. 10.1109/TVCG.2008-06-0082.

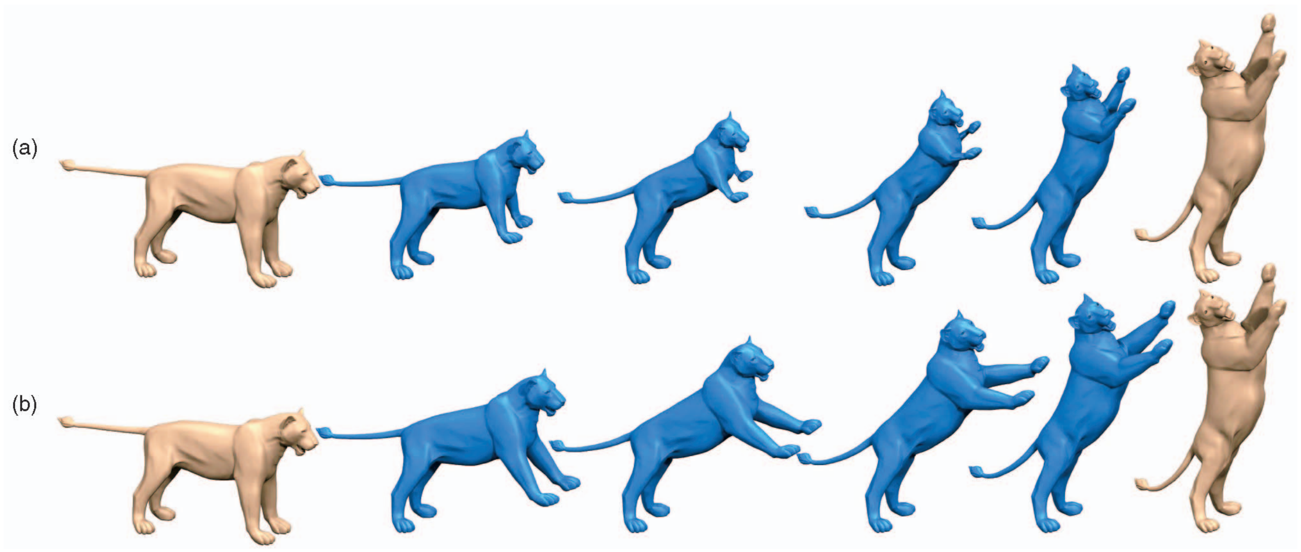

Fig. 1. (a) Linear interpolation of vertex position. Notice the shrinkage defects that are observed in the forelimbs of the lion shape. (b) Shape interpolation using gradient field interpolation. Input shapes are shown in the color tan.

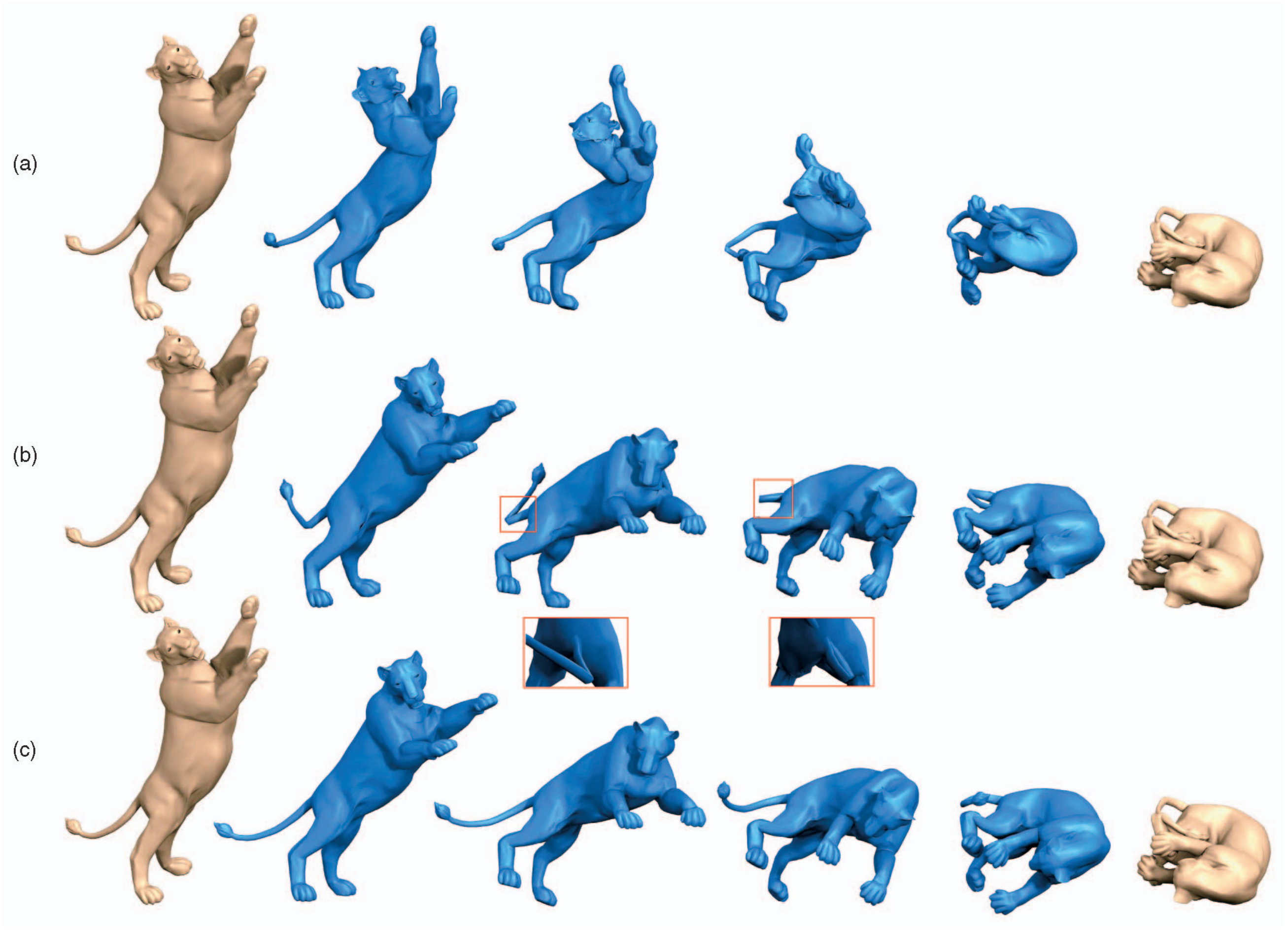

Fig. 2. Shape interpolation between two lion shapes (shown in the color tan) with significant pose variation. Intermediate shapes are shown in blue. (a) Gradient field interpolation generates serious self-intersection. (b) Apply a global rigid transformation before gradient field interpolation. Note the unnatural bending at the tail. (c) The proposed method.

compatible 2D triangulations. However, it is not easy to extend this technique from 2D to 3D because it requires nontrivial compatible tetrahedralizations.

The differential representation of mesh has become very popular, in particular, for mesh deformation application. For shape interpolation, 3D morphing, and editing, many works have been proposed to interpolate meshes represented in differential representation, including the interpolations of gradient fields [38], Laplacian coordinates [1], pyramid coordinates [32], and local frame representation

[23]. Among these representations, the gradient field approach is closely related to our work. Sumner and Popović [34] use a per-triangle deformation gradient to transfer source deformation to a target model. Later, based on the deformation gradient, a mesh-based inverse kinematics system, which learns a space of natural deformations from example meshes is proposed by Sumner et al. [35] and improved by Der et al. [15] using a compact set of proxy vertices inferred from example deformations. While both approaches focus on direct manipulation of mesh deformation through intuitive control of mesh vertices, their application to shape interpolation is not trivial because they need to carefully pose intermediate keyframes to obtain correct interpolation between shapes. Xu et al. [38] combine the work of [4], [34], and achieve shape interpolation by nonlinearly interpolating per-triangle gradient [9]. However, their approach may fail in the case where the pose between source and target shapes varies significantly, as described previously. Our approach also adopts the per-triangle gradient to serve as local detail transformation which is further controlled by the transformation of underlying articulated structure. Recently, the state-of-the-art shape interpolation method proposed by Kilian et al. [19] can solve the pose variation problem quite well by using Riemannian geometry. Their approach maps the problem of shape interpolation by finding the geodesic curve in the shape space. The vertex trajectory is formulated as a global optimization problem and solved using a multiresolution approach. However, this approach cannot guarantee to avoid trapping at local minimum and the optimization process is inherently slow [25]. In contrast, our method computes the vertex trajectory directly in the mesh domain, and therefore, achieves comparable or even better results and has better computational efficiency.

Skeleton-driven mesh deformation explicitly takes a skeleton as an articulated structure and controls the deformation via the skeleton. There are two recent approaches related to our work. Yan et al. [39] achieve shape deformation by using a skeleton to decompose the shape into near-rigid parts and drive the transformation of simplicial complexes. Weber et al. [37] factorize the mesh deformation problem into a global skeleton-driven deformation and a local gradient deformation. Both approaches require manually sculpting a skeleton which is a tedious task for complicated mesh.

On the other hand, the articulated structure can be realized by a set of (hierarchically organized) near-rigid components extracted from input shapes. Lengyel [22] uses an iterative algorithm to cluster triangles with similar affine transformation and applies the result to compress a time-varying geometry. Lee et al. [21] suggest extracting near-rigid components by analyzing the difference of deformation gradients among animating meshes. However, the quality of their results highly depends on the parameters tuning. Similarly, Schaefer and Yuksel [29] cluster the faces with similar rigid transformations among examples through a face-based mesh simplification. In their approach, the number of clusters is determined by either a user-specified number or a specific error tolerance. Authors also point out that an inappropriate cluster number leads to great approximation error in their skinning animation application. A symmetrization method [24], [26] based on statistical analysis of sampling point pairs can be used to extract near-rigid components from two input models with different poses. Chang and Zwicker [11] further extend the symmetrization to automatically align a pair of shapes with articulated motion and missing data. However, both approaches are limited to two input models and are computationally expensive. Anguelov et al. [5] use a set of registered scan meshes to automatically decompose mesh into approximately rigid parts by optimizing a maximum likelihood function. To discriminate rigid parts of difference sizes, a parameter tuning is required during the optimization process. Using the component-based articulated structure, the SCAPE system [6] uses a pose deformation model to parameterize the space of human shape from a set of dense range scans and is designed for the application of shape completion. Their results are further improved by Park and Hodgins [27] to capture subtle but visually significant surface deformation during the human motion.

Our work is inspired by James and Twigg [17], who use mean shift clustering to cluster surface triangles with similar rotation sequences. The number of clusters is automatically determined by the mean shift clustering algorithm. However, their approach will produce clusters with disconnected triangles and there are unclustered triangles after the clustering. Both factors are problematic for the construction of articulated structure. A naïve extension of their approach by iteratively performing mean shift clustering on unclustered triangles is still infeasible as explained in Section 4.1. In addition, to perform mean shift clustering directly on the original mesh domain is time-consuming. In this paper, we propose a novel MMS clustering algorithm to automatically extract the near-rigid components. In contrast to previous works, our method can handle more than two input shapes, generate an appropriate set of near-rigid components without tuning any parameter, and improve the performance by a multiresolution approach.

3 Overview

Our method comprises two major tasks: 1) computing an articulated structure to describe all *pose configurations* of input shapes and 2) executing shape interpolation. The articulated structure consists of hierarchically organized *near-rigid* components which are automatically extracted from input meshes using the proposed MMS clustering algorithm. For the shape interpolation, we require that the near-rigid components have the same connectivity across all articulated structures. Therefore, we assume that the input meshes to the MMS clustering algorithm have compatible connectivity [2], [20], [28], [30], [34]. To interpolate the intermediate shapes between the source mesh M_S and the target M_T, the former computes their pose configurations P_S and P_T. Each P_i is used to describe the pose configuration of an input shape and will be formally defined in Section 4. The shape interpolation task calculates a global pose transformation from P_S to P_T through a sequence of hierarchical and component-wise rigid transformations, followed by interpolating triangle gradients to account for the change in local details between two corresponding components, and finally, it solves a Poisson equation to reconstruct each interpolated shape. In general, given arbitrary number of input meshes with various poses, the proposed MMS clustering algorithm learns the pose space from input shapes and the extracted articulated

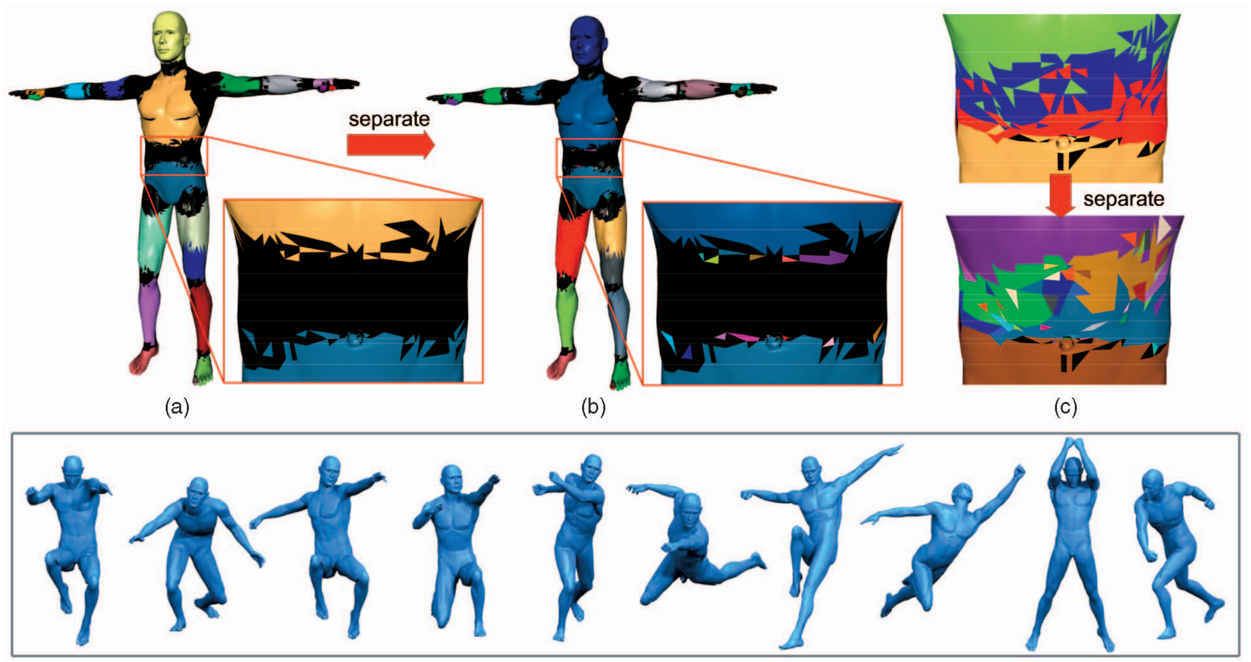

Fig. 3. A naïve application of the MS clustering potentially produces an excessive number of small clusters. The bottom images show 10 input meshes used in the MS clustering. (a) Reference mesh and the result of MS clustering. Each colored cluster represents a rigid cluster while the black regions represent flexible triangles. (b) Separate each rigid cluster in (a) into smaller, disjointed subclusters. (c) More small clusters are generated when using only the second and the ninth mesh as input.

structure can be used to interpolate or blend among input shapes. In Section 6, we will demonstrate this extension with some applications.

4 Pose Configuration and Articulated Structure Extraction

In this section, our goal is to automatically compute pose configurations $\{P_i | i = 1 \dots n\}$ from a set of input meshes $\{M_i | i = 1 \dots n\}$. One of the input meshes is selected as the reference mesh, called M_R. We define each pose configuration P as follows. Let $P = \{N, E\}$ be a hierarchical graph to describe the pose configuration. $N = \{c_1, \dots, c_k\}$ and $E = \{e_1, \dots, e_l\}$ represent nodes and edges of the graph, respectively. Among each node, $c_j = \{\mathbf{K}_j, \mathbf{X}_j\}$ denotes a near-rigid component where $\mathbf{K}_j$ encodes the connectivity of the simplicial complex (triangles, vertices, and edges) and $\mathbf{X}_j$ represents the vertex coordinates. In other words, each mesh M is decomposed into several disjointed components and each component contains connected triangles. All pose configurations have identical edges E and similar nodes N (i.e., all connectivities $\mathbf{K}_j$ are identical but vertex coordinates $\mathbf{X}_j$ can be different).

4.1 Mean Shift (MS) Clustering

To extract the near-rigid components, we use a nonparametric MS clustering algorithm [12], [14] to cluster triangles with similar rotation sequences, as was presented by James and Twigg [17]. In their work, the input data of the MS clustering algorithm are a set of points which represent rotation sequences (i.e., a row vector collecting and concatenating rotation matrices) of the mesh triangles. The output are the shifted points of triangles and a set of statistically significant modes. A triangle is assigned to the closest mode if the L^1-norm distance between its shifted point and the closest mode is within a threshold. As a result, several *rigid* clusters are obtained and each of them contains triangles with similar rotation sequences. The remaining unclustered triangles (i.e., not assigned to any significant mode) are called *flexible* triangles. However, each cluster is not always a connected component, i.e., the triangles in a rigid cluster may be scattered over several disconnected regions (Fig. 3a). The scattering result will hinder us from constructing an articulated structure for pose configuration. To obtain disjointed components, we separate each rigid cluster into several subclusters, each of which contains connected triangles. However, this naïve approach may potentially lead to an excessive number of small clusters (Fig. 3b) which represent the local surface variation rather than a global pose variation. Likewise, a repeat process of clustering flexible triangles until no flexible triangle is left will result in generating an excessive number of small clusters, too. In addition, the MS clustering algorithm is computationally expensive. The computational complexity of MS is $O(nF)$, where n and F represent the number of input meshes and triangles, respectively. As illustrated in Fig. 3, it requires 6.8 minutes to handle 11 input meshes with 35K triangles.

4.2 Multiresolution Mean Shift Clustering

To resolve the above-mentioned problems in directly applying the MS clustering on the original resolution of mesh sequences, we propose a multiresolution version of the MS clustering algorithm. By means of a coarse-to-fine approach, we can extract most significant near-rigid clusters in the coarser level and progressively extract smaller ones. As shown in Fig. 4, the larger clusters such as head, trunk, thighs, and shanks are extracted in the coarse level while the smaller ones, such as fingers, are extracted in the finer

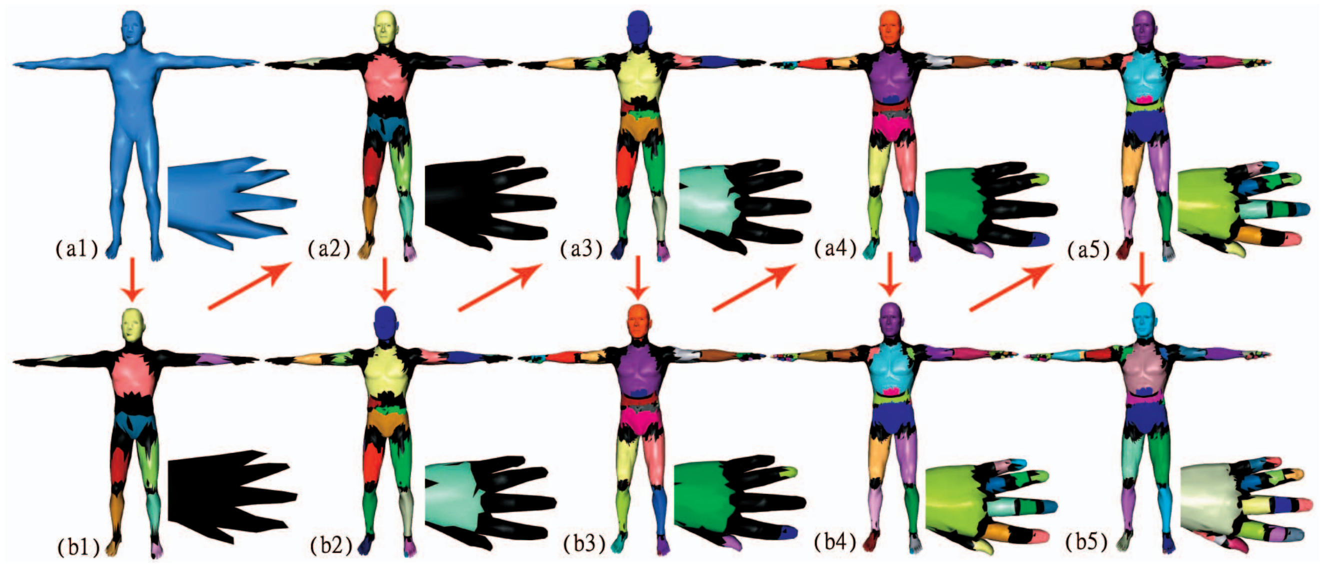

Fig. 4. Illustration of the MMS clustering algorithm. Input meshes are the same as those in Fig. 3. The MMS clustering algorithm proceeds along the arrow direction with the down arrow indicating the step of MS clustering and the oblique arrow representing the reversing (vertex splitting) operation. At each level of resolution (increases from left to right), (a1)-(a5) show the mesh, rigid clusters, and flexible triangles, and (b1)-(b5) show the result of MS clustering. A close view of male's left palm is shown at the right bottom corner of each image.

level. To reduce the number of small clusters, we consider the rotation sequences of rigid clusters as input to the MS clustering and propose three clustering conditions to postprocess resulting rigid clusters and flexible triangles. Our key insight is that the behavior of a cluster is more representative of global pose variation than a single triangle. Using the proposed clustering conditions, our scheme will assign flexible triangles to neighboring rigid clusters and merge two clusters with similar rotation sequences into a larger one, thereby suppressing the number of small clusters. Furthermore, the multiresolution approach also greatly reduces the complexity of the MS clustering algorithm and improves the performance of overall MMS clustering algorithm.

Fig. 5 shows the flowchart of the MMS clustering algorithm. In the initial stage, we simplify input meshes simultaneously using the QEM method [16], [42]. We essentially simplify all input meshes in parallel but constrain their topology and mesh correspondence to be the same under simplification. Therefore, we create an aggregate error by summing the individual errors from the meshes together where the minimized vertex position of the collapsed edge from mesh i is determined solely by the vertices of mesh i. We denote the mesh hierarchy of M_i as

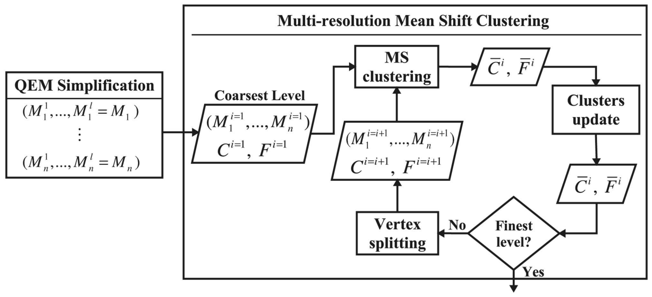

Fig. 5. The flowchart of MMS clustering algorithm.

$(M_i^1,\ldots,M_i^l = M_i)$ with l level of resolutions. The MMS clustering algorithm starts from the coarsest level $(M_1^1,\ldots,M_n^1)$ and then repeats the following two steps until the finest level $(M_1^l,\ldots,M_n^l)$.

Step 1: MS clustering at ith level. The input to this step includes: 1) the ith level meshes $(M_1^i,\ldots,M_n^i)$, 2) rigid clusters $C^i = \{\sigma_j^i | j = 1\ldots k^i\}$, and 3) flexible triangles F^i. Initially, i.e., $i = 1$, C^1 is empty and F^1 contains all triangles of the mesh in the coarsest level. After the first iteration of MS clustering, we obtain new $\bar{C}^1$ and $\bar{F}^1$, which are further postprocessed and serve as inputs to the next iteration (as described later). At ith level, $i > 1$, we first compute the rotation sequences of each rigid cluster in C^i as follows. For each rigid cluster with vertex positions $\mathbf{x}_R$ in the reference mesh M_R^i and corresponding vertex positions $\mathbf{x}_a$ in M_a^i, we compute an optimal rigid transformation composed of a rotation $\mathbf{q}_{R\rightarrow a}$ (unit quaternion) and a translation $\mathbf{t}_{R\rightarrow a}$ using the approach presented in [7], such that (1) is minimized:

$$\min_{\{\mathbf{q}_{R\rightarrow a},\mathbf{t}_{R\rightarrow a}\}} = \sum_j ||(\mathbf{q}_{R\rightarrow a}\cdot \mathbf{x}_R^j + \mathbf{t}_{R\rightarrow a}) - \mathbf{x}_a^j||^2. \qquad (1)$$

We collect the optimal rotation matrix (converted from $\mathbf{q}$) for each input mesh to represent the rotation sequences of each rigid cluster. The rotation sequences of rigid clusters are served as input to the MS clustering and used to merge flexible triangles with similar rotation sequences. Therefore, we call each rigid cluster as a *virtual flexible* triangle and denote the set of virtual flexible triangles as $\tilde{F}^i$. Regarding the rotation sequences of each flexible triangle, we first compute its deformation gradient as $\mathbf{G} = (\mathbf{x}_a^1 - \mathbf{x}_a^3, \mathbf{x}_a^2 - \mathbf{x}_a^3, \mathbf{n}_a)\cdot(\mathbf{x}_R^1 - \mathbf{x}_R^3, \mathbf{x}_R^2 - \mathbf{x}_R^3, \mathbf{n}_R)^{-1}$ with $(\mathbf{x}_i^1, \mathbf{x}_i^2, \mathbf{x}_i^3)$ and $\mathbf{n}_i$ as triangle's vertex positions and normal in M_i, respectively. Then, we extract the triangle rotation from $\mathbf{G}$ using polar decomposition [33] and collect the rotation matrix for each mesh. We perform MS clustering on rotation sequences of

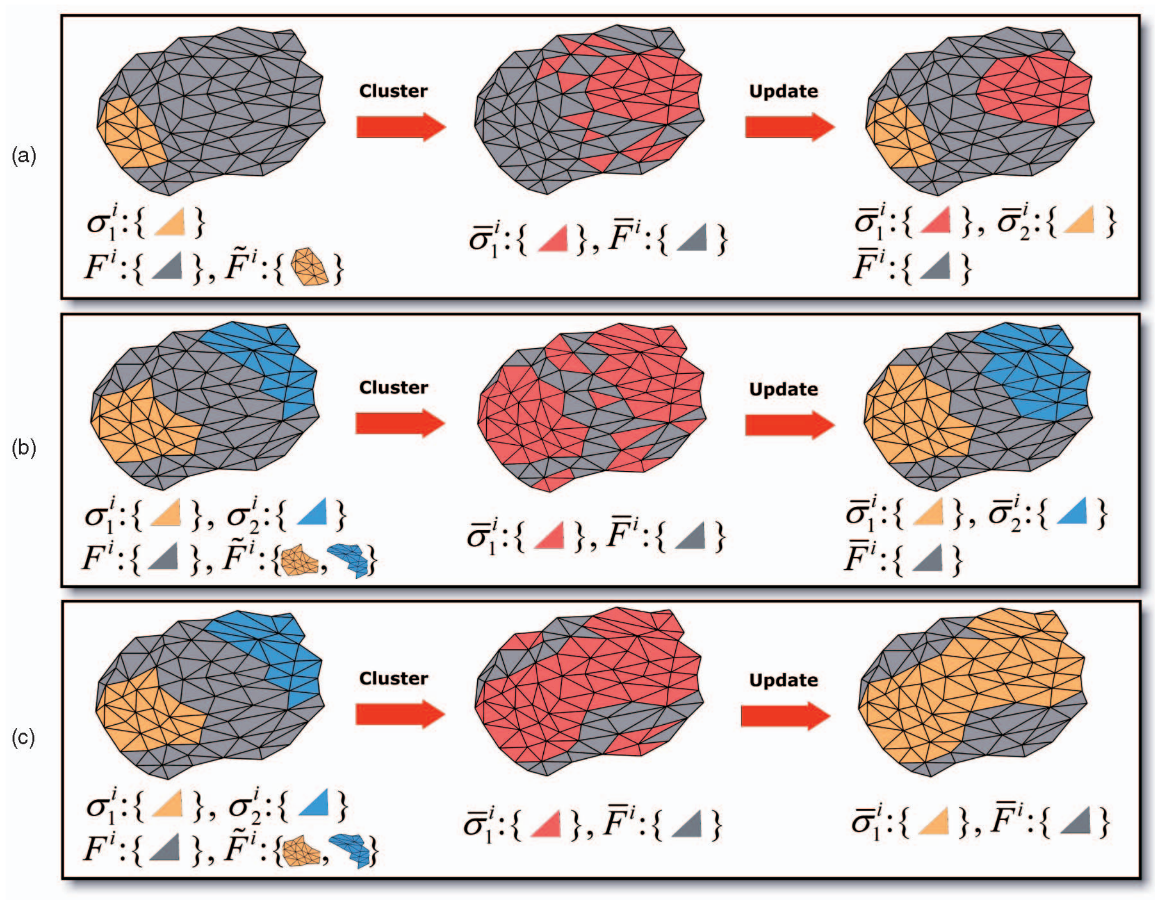

Fig. 6. Illustration of the clustering condition #1 and #2. Images in the first column show the inputs to the MMS clustering algorithm with rigid clusters colored in tan and blue and flexible triangles colored in gray. The second column shows the clustering result with a new pink cluster. After applying our clustering conditions, the updated clusters are shown in the third column.

all triangles in $F^i \cup \tilde{F}^i$. The output includes new rigid clusters $\bar{C}^i = \{\bar{\sigma}^i_j | j = 1 \dots \bar{k}^i\}$ and flexible triangles $\bar{F}^i$. Note that $\bar{C}^i$ and $\bar{F}^i$ may contain virtual flexible triangles which originally represent the rigid clusters in C^i. In the next step, we use three (and only) clustering conditions to update $\bar{C}^i$ and $\bar{F}^i$, such that the final rigid clusters are representative of global pose variation and no excessive number of small clusters are generated.

Condition #1: $\exists \bar{\sigma}^i \in \bar{C}^i : \bar{\sigma}^i \cap \tilde{F}^i = \emptyset$.[1] There is no virtual flexible triangle in $\bar{\sigma}^i$, i.e., no rigid cluster in C^i has similar rotation sequences with triangles in $\bar{\sigma}^i$. We separate $\bar{\sigma}^i$ into several disjointed subclusters and retain the one with the maximum area of triangles. Triangles in the remaining subclusters are removed from $\bar{\sigma}^i$ and inserted into $\bar{F}^i$ (Fig. 6a). In this manner, we ensure that the most representative cluster $\bar{\sigma}^i$ is obtained.

Condition #2: $\exists \bar{\sigma}^i \in \bar{C}^i : \bar{\sigma}^i \cap \tilde{F}^i \neq \emptyset$. Assume that there are $n \geq 1$ virtual flexible triangles in $\bar{\sigma}^i$ and denote the corresponding rigid clusters in C^i as $\{\sigma^i_1, \dots, \sigma^i_n\}$. Then, we iteratively update each cluster $\sigma^i \in \{\sigma^i_1, \dots, \sigma^i_n\}$ by assigning a triangle $f \in \bar{\sigma}^i$ to σ^i if f is connected (i.e., sharing any edge) to σ^i. If two updated clusters become connected, they are merged into a single cluster. For those triangles that are not merged into any rigid cluster, we remove them from $\bar{\sigma}^i$ and insert them into $\bar{F}^i$. As a result, $\bar{\sigma}^i$ is separated into several rigid clusters and we insert these rigid clusters back into $\bar{C}^i$. Therefore, triangles are merged into existing rigid clusters with similar rotation sequences and rigid clusters with similar rotation sequences are merged, too. Fig. 6b illustrates the case: there are two virtual flexible triangles in $\bar{\sigma}^i$ and the corresponding rigid clusters are disconnected from each other after merging connected triangles in $\bar{\sigma}^i$. Fig. 6c illustrates another case: two updated rigid clusters are connected and merged into a single cluster.

Condition #3: $\exists f \in \bar{F}^i : f \in \tilde{F}^i$. We remove the triangle f from $\bar{F}^i$ and insert the corresponding rigid cluster in C^i into $\bar{C}^i$. In this condition, this cluster will not be updated at the current iteration (Fig. 6a).

Step 2: Reversing process using vertex-split operation. Up to now, we have rigid clusters $\bar{C}^i$ and flexible triangles $\bar{F}^i$ in the ith level of resolution. To proceed to the next level, we reverse the mesh sequences from $(M^i_1, \dots, M^i_n)$ to $(M^{i+1}_1, \dots, M^{i+1}_n)$ and update $\bar{C}^i$ and $\bar{F}^i$ using vertex-split operations. Each vertex-split operation involves one vertex and its two neighboring edges. After splitting, each edge is split into two edges and two new triangles are created. Therefore, for each new triangle, the only task is to determine to which rigid cluster the triangle should be assigned or to insert the triangle to $\bar{F}^i$. We accomplish this

1. The logic expression $\exists x \in X : P(x)$ means that there exists one x in X such that the expression $P(x)$ is true.

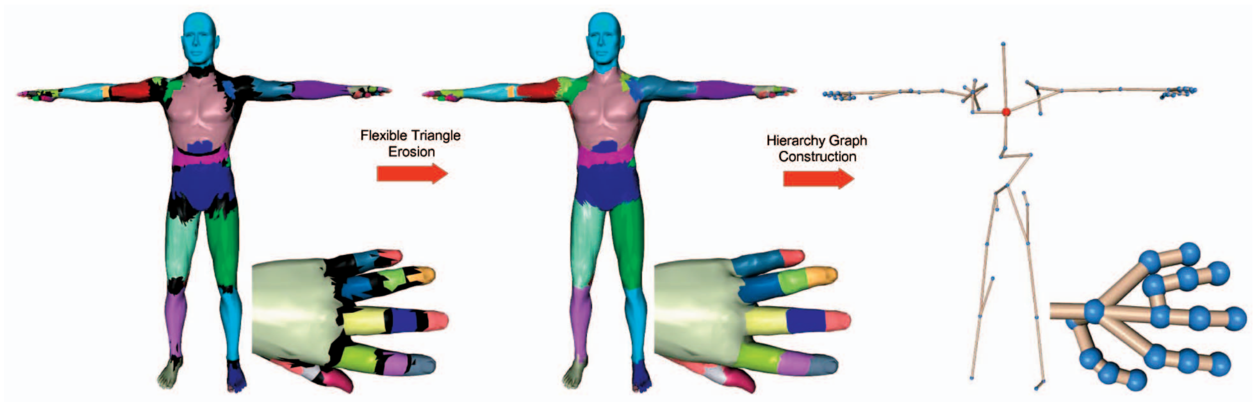

Fig. 7. Flexible triangle (black regions) erosion and final near-rigid components.

task using the following rules. For each edge of the vertex-split operation,

- if one of its two neighboring triangles belongs to $\bar{F}^i$, we insert the new triangle incident to the edge into $\bar{F}^i$ (Fig. 8 (left)).
- if both neighboring triangles belong to a rigid cluster in $\bar{C}^i$, we assign the new triangle incident to the edge to that cluster (Fig. 8 (middle)).
- if two neighboring triangles belong to two different rigid clusters in $\bar{C}^i$, we assign the new triangle to the rigid cluster with smaller similarity distance between their rotation sequences. The similarity distance is computed by the following equation:

$$\max\left(||\mathbf{r}^f_{R\to a} - \mathbf{r}^\sigma_{R\to a}||^2_F\right), a = 1 \dots n, \qquad (2)$$

where $\mathbf{r}^f_{R\to a}$ and $\mathbf{r}^\sigma_{R\to a}$ are rotation matrices from rotation sequences of the new triangle f and the rigid cluster σ, respectively.

The results of this step are $(M^{i+1}_1, \dots, M^{i+1}_n)$, rigid clusters C^{i+1}, and flexible triangles F^{i+1}, and these results are used as input to the next iteration of MS clustering step described in previous paragraphs.

4.3 Flexible Triangle Erosion and Hierarchy Graph Construction

The output of the MMS clustering algorithm consists of several rigid clusters and flexible triangles (Fig. 4(b5)). To find the final near-rigid components, we first merge these flexible triangles into several disjointed flexible clusters based on shared edges and then iteratively remove each triangle from flexible clusters and insert it into a nearby rigid cluster by an erosion approach. The metric of erosion priority for each flexible triangle is determined by the rigidity of its edge [21]:

$$\max\left(||\mathbf{G}^i_{R\to a} - \mathbf{G}^j_{R\to a}||^2_F\right), a = 1 \dots n, \qquad (3)$$

where i and j represent the neighboring triangles of the edge and $\mathbf{G}^i_{R\to a}$ represents the matrix of deformation gradient of the triangle of M_a relative to M_R. Our motivation is to assign each flexible triangle to a neighboring rigid cluster such that the boundary of the final near-rigid cluster passes through the nonrigid region in the flexible cluster. For each flexible cluster σ_f, we start with the erosion process from its boundary edges which are adjacent to the neighboring rigid clusters by finding an adjacent edge e with the minimum rigidity, and then removing a triangle sharing e from σ_f, assigning it to the rigid cluster sharing e and updating the boundary of σ_f. This process is repeated until σ_f is empty. So, in this iterative manner, each σ_f is eroded and the neighboring rigid clusters are growing. Fig. 7 shows the result of the erosion process.

Then, we treat each near-rigid component as a graph node of $P = \{N, E\}$. If any pair of nodes is sharing the boundary, these two nodes are considered to be connected by an edge. We determine the hierarchical relationship among near-rigid components by finding a minimum spanning tree to connect them. The weight of each edge is equal to the inverse length of the shared boundary between two components. The default root of this hierarchical graph is the node whose centroid position is the closest to that of M_R. Finally, the construction of pose configuration $P = \{N, E\}$ is accomplished.

Fig. 9 shows several near-rigid components extracted from two input shapes. Although the shared boundaries of near-rigid components are not smooth, they are good enough for shape interpolation described in the next section.

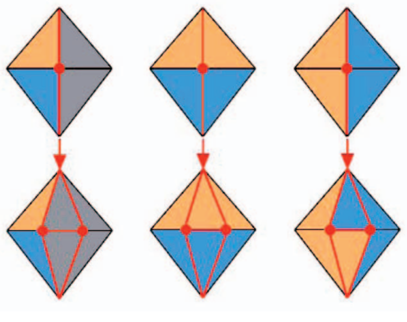

Fig. 8. Assignment of two new triangles after a vertex-split operation. The gray triangle represents the flexible triangle, while the tan and blue triangles represent two different rigid clusters.

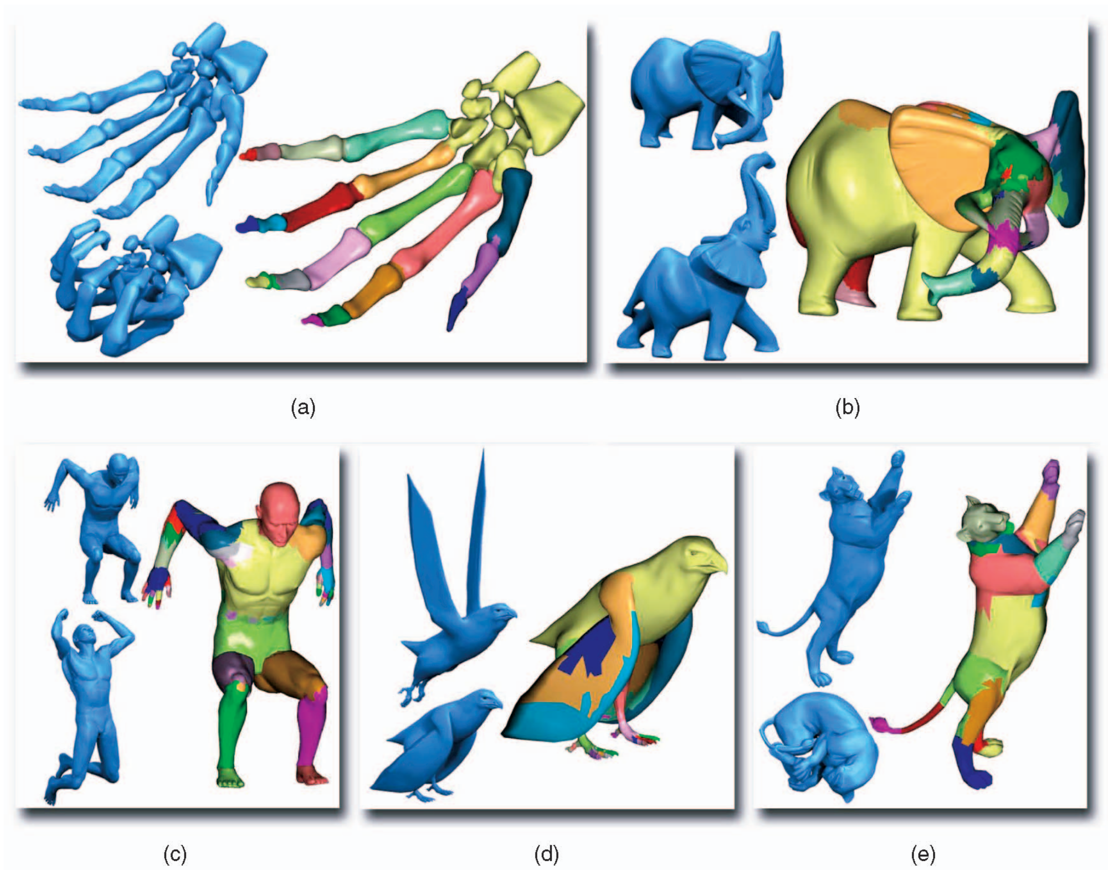

Fig. 9. Extracted near-rigid components and their input meshes.

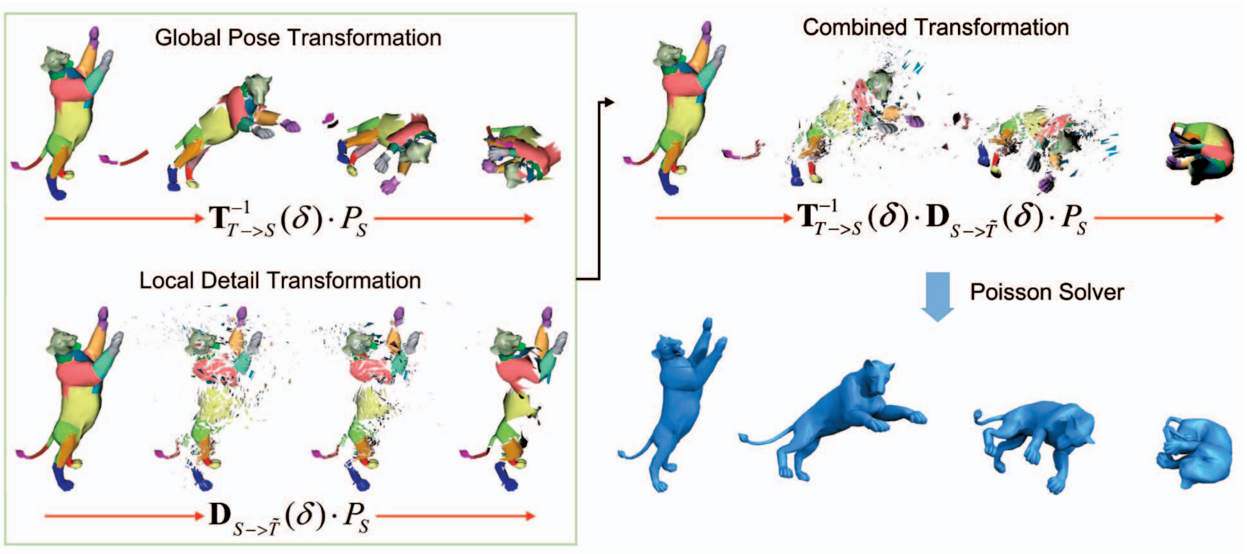
Global Pose Transformation
$\mathbf{T}^{-1}_{T->S}(\delta)\cdot P_S$
Local Detail Transformation
$\mathbf{D}_{S->\tilde{T}}(\delta)\cdot P_S$
Combined Transformation
$\mathbf{T}^{-1}_{T->S}(\delta)\cdot \mathbf{D}_{S->\tilde{T}}(\delta)\cdot P_S$
Poisson Solver

Fig. 10. Illustration of shape interpolation.

5 Shape Interpolation

With the extracted pose configurations, we are now ready to interpolate between shapes. Fig. 10 illustrates the progress of shape interpolation and each step will be elaborated in the following sections.

5.1 Pose Transformation

The pose transformation consists of a set of rigid transformations and each of them transforms a component of P_T to the corresponding component of P_S. Since both P_S and P_T are organized hierarchically, we compute the rigid transformations hierarchically by applying the parent's rigid transformation to its children components before optimizing (1) to find the local rigid transformations of children components. We denote the pose transformation from P_T to P_S as $\mathbf{T}_{T\to S} = \{(\mathbf{t}^i_{T\to S}, \mathbf{q}^i_{T\to S})|i = 1\dots k\}$, where $(\mathbf{t}^i_{T\to S}, \mathbf{q}^i_{T\to S})$ indicates a local rigid transformation (translation and rotation, respectively) of the ith component and k is the number of components.

5.2 Local Detail Transformation

Applying $\mathbf{T}_{T\to S}$ to P_T will obtain a pose configuration $\tilde{P}_T$ whose components are aligned with the corresponding components of P_S. To complete the transformation from M_T to M_S, we compute the local detail deformations from P_S to $\tilde{P}_T$ and denote it as $\mathbf{D}_{S\to\tilde{T}} = \{\mathbf{D}^i_{S\to\tilde{T}} | i = 1\dots k\}$. Each $\mathbf{D}^i_{S\to\tilde{T}}$ encodes deformation gradients of triangles of the ith component in P_S relative to triangles of the corresponding component in $\tilde{P}_T$. In order to facilitate the nonlinear interpolation described in the next section, we further factorize each triangle deformation gradient into rotation and stretch part through polar decomposition. We denote each $\mathbf{D}^i_{S\to\tilde{T}}$ as $\{(\mathbf{q}^j_{S\to\tilde{T}}, \mathbf{s}^j_{S\to\tilde{T}}) | j = 1\dots m^i\}$, where m^i is the number of triangles of the ith component, $\mathbf{q}^j_{S\to\tilde{T}}$ and $\mathbf{s}^j_{S\to\tilde{T}}$ represent unit quaternion and symmetric stretch matrix, respectively.

5.3 Nonlinear Global Pose and Local Detail Interpolation

Given pose transformation $\mathbf{T}_{T\to S}$ and local detail transformation $\mathbf{D}_{S\to\tilde{T}}$, it is easy to derive the following equation:

$$P_T = \mathbf{T}^{-1}_{T\to S} \cdot (\mathbf{D}_{S\to\tilde{T}} \cdot P_S), \tag{4}$$

where $\mathbf{T}^{-1}_{T\to S}$ indicates the inverse pose transformation. Therefore, the interpolation from P_S to P_T corresponds to the interpolation of the combined transformation, $\mathbf{T}^{-1}_{T\to S} \cdot \mathbf{D}_{S\to\tilde{T}}$. We interpolate $\mathbf{T}^{-1}_{T\to S}$ and $\mathbf{D}_{S\to\tilde{T}}$ separately and write the equation as

$$P(\delta) = \mathbf{T}^{-1}_{T\to S}(\delta) \cdot (\mathbf{D}_{S\to\tilde{T}}(\delta) \cdot P_S), \delta \in [0,1], \tag{5}$$

with the boundary conditions $P(0) = P_S$ and $P(1) = P_T$. Instead of linearly interpolating the transformation matrix, we nonlinearly interpolate $\mathbf{T}^{-1}_{T\to S}(\delta)$ and $\mathbf{D}_{S\to\tilde{T}}(\delta)$ by the following equations (we denote $\mathbf{T}^{-1}_{T\to S}$ as $\{(\bar{\mathbf{t}}^i_{T\to S}, \bar{\mathbf{q}}^i_{T\to S}) | i = 1\dots k\}$):

$$\begin{aligned}
\mathbf{T}^{-1}_{T\to S}(\delta) &= \big\{\big(\delta \cdot \bar{\mathbf{t}}^i_{T\to S}, slp\big(\mathbf{q}_{\mathrm{I}}, \bar{\mathbf{q}}^i_{T\to S}, (1-\delta)\big)\big) | i = 1\dots k\big\},\\
\mathbf{D}_{S\to\tilde{T}}(\delta) &= \big\{\mathbf{D}^i_{S\to\tilde{T}}(\delta) | i = 1\dots k\big\}, \; and\\
\mathbf{D}^i_{S\to\tilde{T}}(\delta) &= \big\{\big(slp\big(\mathbf{q}_{\mathrm{I}}, \mathbf{q}^j_{S\to\tilde{T}}, (1-\delta)\big), \big((1-\delta)\cdot \mathbf{I} + \delta \cdot \mathbf{s}^j_{S\to\tilde{T}}\big)\big)\\
&\qquad | j = 1\dots m^i\big\},
\end{aligned} \tag{6}$$

where slp denotes the quaternion slerp interpolation, and $\mathbf{q}_{\mathrm{I}}$ and $\mathbf{I}$ represent the quaternion identity and the identity matrix, respectively.

5.4 Poisson Solver

Although the nonlinear interpolation of (6) satisfies the boundary conditions at $\delta = 0, 1$, it generates shapes with disconnected triangles at $0 < \delta < 1$ because the interpolated transformations are not consistent and applied to each triangle independently (Fig. 10 (right top)). To solve this issue, we adopt the Poisson solver [9] to stitch disconnected triangles of interpolated shapes. At each time step, the vertex positions of an intermediate shape are obtained by solving a linear system of equation $\mathbf{AX} = \mathbf{b}$, where sparse matrix $\mathbf{A}$ encodes edges' connectivity of M_S and a vector $\mathbf{b}$ contains the interpolated gradients, $\mathbf{T}^{-1}_{T\to S}(\delta) \cdot \mathbf{D}_{S\to\tilde{T}}(\delta)$. Since the interpolated gradients are translation invariant, we need to specify at least one positional constraint to solve the Poisson equation [34], [38]. Therefore, we choose the interpolated trajectory of one vertex of the root component to serve as the positional constraint at each time step. Then, we prefactorize the matrix $\mathbf{A}$ using sparse Cholesky factorization [8], [36]. Given the interpolated gradients at certain time step, we can obtain the vertex positions efficiently by back substitution.

6 Experiments and Applications

In this section, we demonstrate experimental results of shape interpolation and two applications in keyframe animation and multitarget shape blending. All the results shown in the paper were generated on a PC with Intel Core 2 Duo E6420 2.13 GHz CPU and 1 GB RAM. To experiment MMS clustering algorithm, we use $l = 5$ and set 3,000 triangles in the coarsest level. The number of triangles in the subsequent levels is $3{,}000 + (\frac{n_f - 3{,}000}{l-1}) * (i-1), i = 2\dots l$, where n_f is the number of triangles of original mesh. The L^1-norm distance threshold used in MS clustering is $h = 9n\varepsilon/4$, where n is the number of input meshes and $\varepsilon = 0.05$. This setting is exactly the same as [17]. In addition to this parameter, there is no parameter required for MMS. The accompanying video and data sets can be found in our project Web, http://graphics.csie.ncku.edu.tw/Shape_Interpolation/.

Shape interpolation. In Fig. 11, we interpolate the male shape from its sneaky crouched pose to an extremely stretched pose. As a result, the natural pose interpolation of the limbs and fingers is demonstrated, and the local detail of shape (lines of the muscle) is well preserved and changes smoothly during the interpolation. In this example, the input shapes are chosen from Fig. 3, and their extracted near-rigid components are shown in Fig. 7. Using these near-rigid components to construct pose configurations, we can smoothly interpolate any two or more arbitrary shapes in Fig. 3. Fig. 12 shows other four interesting interpolation results and their near-rigid components are shown in Fig. 9. In addition to interpolation between articulated shapes, we also experiment with nonarticulated shapes, as shown in Fig. 14. Our results on these two nonarticulated cases are pleasing too.

Keyframe animation. Fig. 13 shows an example of interpolating three male shapes in different poses using the same pose configuration as in Fig. 11. These three shapes are very dissimilar from the input shapes used in the construction of pose configurations (Fig. 3). Therefore, as long as the pose space is well spanned by the input shapes, the extracted pose configuration is reusable and applicable to shape interpolation between shapes with various poses. Besides, this kind of shape interpolation is very useful in traditional keyframe animation. An intuitive extension to interpolate more than two keyframe shapes by our method is to interpolate consecutive pair of keyframe shapes. However, our shape interpolation requires the factorization of a coefficient matrix (i.e., $\mathbf{A}$ matrix) in the Poisson equation (i.e., $\mathbf{AX} = \mathbf{b}$), if each keyframe shape is treated as a new source shape, it will require $(m-1)$ times factorization cost, where m is the number of keyframe shapes. To avoid this, we only treat the first key shape as

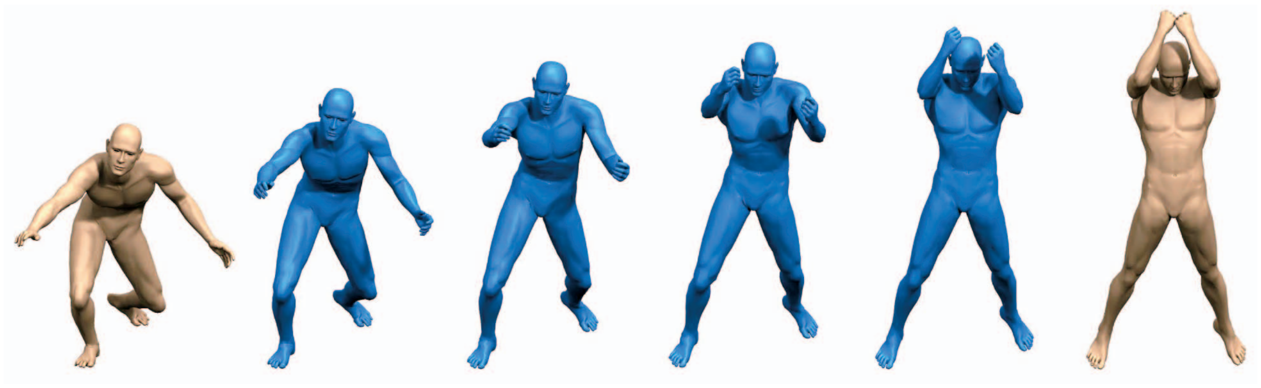

Fig. 11. Shape interpolation of the male shape from a crouched pose to a stretched pose (both shown in tan).

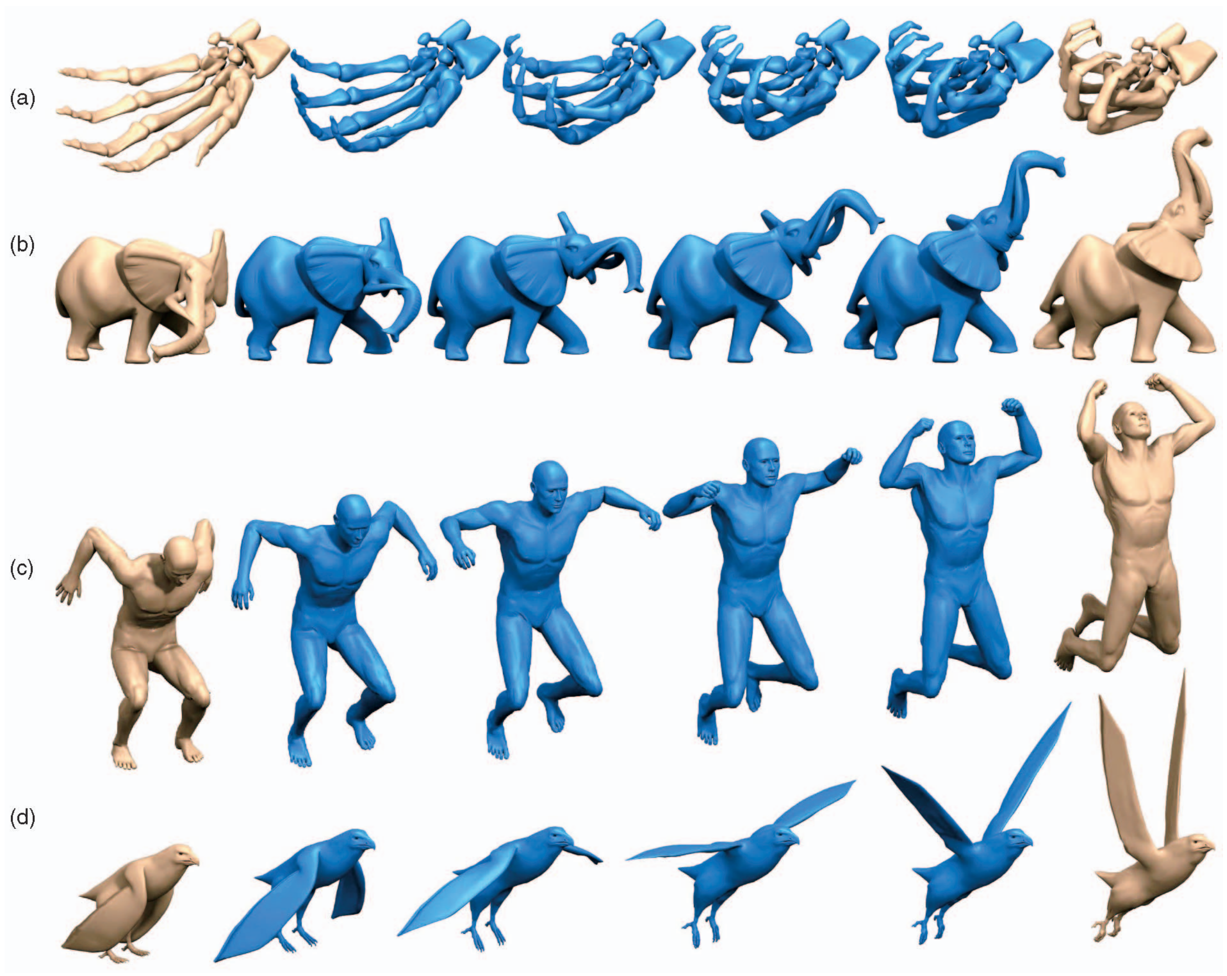

Fig. 12. More shape interpolations with significant poses variation.

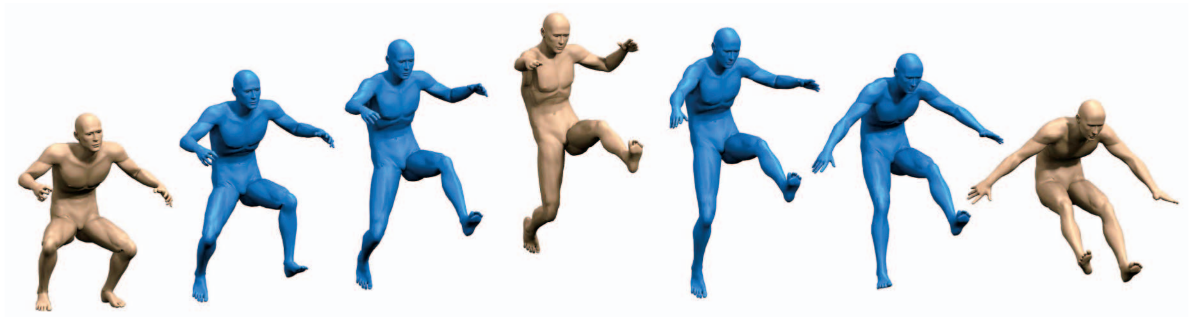

Fig. 13. Given three keyframe shapes (shown in tan), our method smoothly interpolates the intermediate shapes (shown in blue), providing a useful tool for animators to rapidly create keyframe animation sequences.

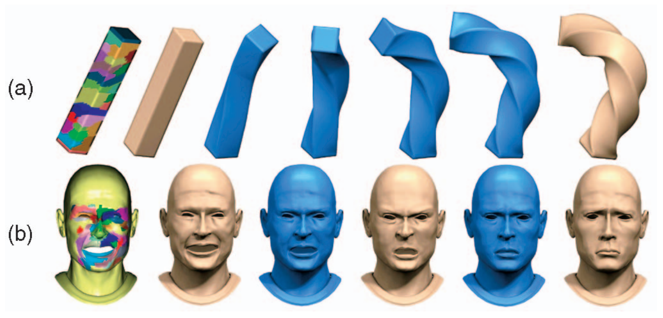

Fig. 14. Two examples of interpolation between shapes with no apparent articulated structure. (a) Shape interpolation from a normal pillar to a highly twisted one. (b) Shape interpolation between three facial expressions (smile, anger, and sad).

the source shape, and compute the pose and local detail transformations of other keyframe shapes relative to this source shape. Then, we interpolate both pose and local detail transformations between each pair of keyframe shapes and generate intermediate shapes by solving the Poisson equation. For example, assume there are three keyframe shapes denoted as A (i.e., source shape), B, and C, and A interpolates to B at $0.0 \leq \delta_1 \leq 0.5$, while B to C at $0.5 < \delta_2 \leq 1.0$. The pairwise pose and local detail transformation are denoted as $\mathbf{T}^{-1}_{B\to A}$ and $\mathbf{D}_{A\to\tilde{B}}$, respectively (similarly for $B \to C$). We can compute the interpolated triangle gradient relative to A as follows:

$$\mathbf{T}^{-1}(\delta)\cdot\mathbf{D}(\delta) = \begin{cases} \mathbf{T}^{-1}_{B\to A}(\delta+0.5)\cdot\mathbf{D}_{A\to\tilde{B}}(\delta+0.5), & \textit{if } \delta\in\delta_1, \\ \mathbf{T}^{-1}_{C\to B}(\delta-0.5)\cdot\mathbf{D}_{B\to\tilde{C}}(\delta-0.5)\cdot\mathbf{T}^{-1}(0.5)\cdot\mathbf{D}(0.5), & \\ & \textit{if } \delta\in\delta_2. \end{cases}$$

Since all the computations are done in terms of the same source shape, we only need to factorize the coefficient matrix once and solve the Poisson equation efficiently by back substitution.

Multitarget shape blending. Another useful application of our method is n-way shape blending, also called multitarget shape blending. Given a set of shapes, multitarget shape blending generates shapes which are weighted combinations of the input shapes. This technique provides the user a practical tool to explore the space of possible shapes among input shapes. The multitarget shape blending is achieved using the following steps:

1. Given a set of input shapes $\{M_i | i = 1 \ldots l\}$, we choose any one of them as a source (reference) shape M_S, and then compute the pose configurations from these shapes.
2. Next, we construct the pose transformations $\mathbf{T}_{i\to S}$ and local detail transformation $\mathbf{D}_{S\to\tilde{i}}$ for each input shape.
3. Given the specified blending weight for each shape, we blend the rotational part of $\mathbf{T}_{i\to S}$ and $\mathbf{D}_{S\to\tilde{i}}$ using the exponential map [3] and linearly blend the translational and scale/shear part of $\mathbf{T}_{i\to S}$ and $\mathbf{D}_{S\to\tilde{i}}$, respectively. Then, we combine the blended pose and local detail transformations.
4. Finally, we reconstruct the blended shape by solving the Poisson equation.

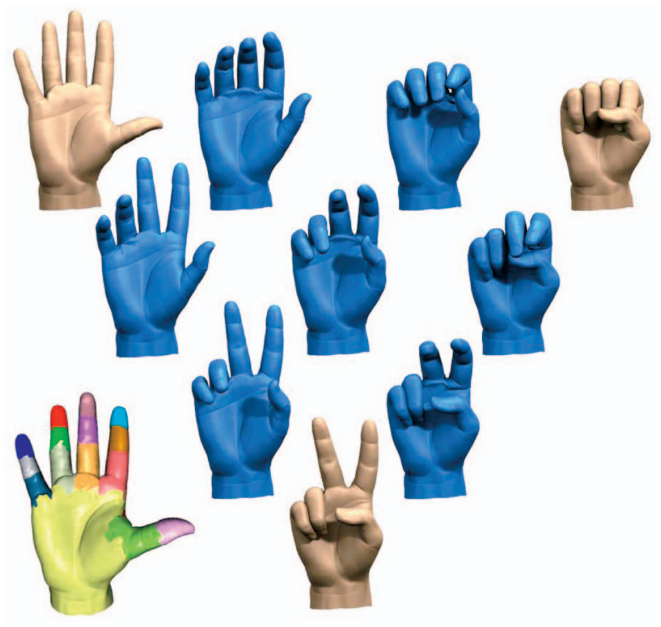

Fig. 15. Multitarget shape blending of three hand shapes (shown in tan). The near-rigid components are extracted from three shapes and shown in the lower-left corner.

Fig. 15 shows an example of blending three hand shapes using the above four steps.

Timing and comparison. We show the timing statistics of all experiments in this paper in Table 1. The second to fifth columns indicate the number of triangles (#Tri), the number of key shapes used in shape interpolation or multitarget blending (#Key), near-rigid components (NRC), and the number of input shapes used in the MMS clustering algorithm (#Input). The sixth to tenth columns list the timing of the pose configuration extraction (Extract), calculation and interpolation of global pose and local detail

TABLE 1
Timing Statistics of Pose Configuration Construction and Shape Interpolation

Fig.	#Tri	#Key	NRC	#Input	Extract	Interpolate	Factor	Solve	Total
Fig. 2 (bottom)	10K	2	Fig. 9(e)	2	8.86	0.64	0.62	0.01	10.12
Fig. 11	35K	2	Fig. 7	11	160.57	2.11	2.84	0.06	165.22
Fig. 13	35K	3	Fig. 7	11	160.57	2.78	2.84	0.06	166.19
Fig. 12 (a)	26K	2	Fig. 9 (a)	2	8.28	1.45	2.59	0.04	12.32
Fig. 12 (b)	80K	2	Fig. 9 (b)	2	77.54	4.9	11.82	0.15	94.26
Fig. 12 (c)	35K	2	Fig. 9 (c)	2	33.48	2.0	2.84	0.06	38.32
Fig. 12 (d)	25K	2	Fig. 9 (d)	2	19.62	1.5	2.28	0.03	23.4
Fig. 15	16K	3	Fig. 15	3	14.47	0.73	0.91	0.07	16.11

TABLE 2
Timing Comparison with Those of Martin et al. [19]

Fig.	Our Method	Martin *et al.* [19]
Fig. 2 (bottom)	10.12	39
Fig. 11	35.3	41
Fig. 12 (a)	12.32	42
Fig. 12 (b)	94.26	164

transformations (Interpolate), prefactorization (Factor) of the coefficient matrix of the Poisson equation, the average timing to generate each intermediate shape by solving the Poisson equation (Solve), and total timing of the process $(\mathrm{Total} = \mathrm{Extract} + \mathrm{Interpolate} + \mathrm{Factor})$. Since each intermediate shape is generated on the fly, we do not include the timing of generating 50 frames and just list the average timing of solving the Poisson equation for each frame. For the multitarget shape blending, the "Solve" time also includes the blending of transformations among each input shape. As shown in the table, the complexity of the proposed method is dominated by the extraction of pose configuration which scales according to the complexity of the shape and the number of input shapes. In Table 2, we also compare five experimental results with [19] in terms of performance. The timing of [19] is measured on a PC with Intel Core 2 Duo T7700 2.4 GHz CPU and 4 GB RAM. Note that for the purpose of comparison, we regenerate the result of Fig. 11 by using only source and target shapes as input to the MMS clustering algorithm. The unit of timing of both tables is measured in seconds.

We also compare our MMS clustering algorithm with MS clustering algorithm using examples of Figs. 3 and 9. We perform five iterations of MS clustering on flexible triangles to get rigid and flexible clusters. The final near-rigid components are obtained by the erosion process (Section 4.3). Fig. 16 shows the side-by-side comparison of the MMS and MS clustering algorithm. Table 3 indicates that our MMS clustering algorithm can not only generate smaller number of components than MS clustering algorithm, but also 3-11 times faster than direct application of MS clustering algorithm.

TABLE 3
Comparison with MS Clustering Algorithm

	Timing (sec)		**#Rigid cluster**		**#Flexible cluster**	
	MS	MMS	MS	MMS	MS	MMS
Fig. 3	579.32	160.57	247	69	188	44
Fig. 9 (a)	53.7	8.28	408	21	129	23
Fig. 9 (b)	853.25	77.54	5848	66	22	44
Fig. 9 (c)	120.68	33.48	1134	62	349	34
Fig. 9 (d)	95.42	19.62	1264	37	34	18
Fig. 9 (e)	25.46	8.86	406	21	136	12

In Fig. 17, we compare our results with those of Xu et al. [38] and Kilian et al. [19]. For each example, only a certain frame with noticeable difference is shown. We refer readers to the accompanying video, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TVCG.2009.40, for a complete comparison of the interpolation sequences. As shown in the video, our results are superior to those of Xu et al. [38] and comparable or even better than those of Kilian et al. [19].

7 Conclusion and Future Work

In this paper, we propose a practical solution to interpolate shapes with a wide range of pose variation. A novel MMS clustering algorithm is proposed to automatically and efficiently compute the pose configuration. Then, we solve the vertex trajectory problem of shape interpolation as a combination of global pose transformation and local detail transformation of surface triangles, followed by solving a Poisson equation to reconstruct an interpolated shape. We demonstrate the success and usefulness of our method through several examples and two applications. There are some future works to be further investigated. Our MMS clustering algorithm may potentially fail in the case that shapes not only vary in pose but also have large deformation on the surface. Under this circumstance, the MMS clustering algorithm fails to extract near-rigid components for representing pose variation because the rotation sequences of triangles are quite different, i.e., generating

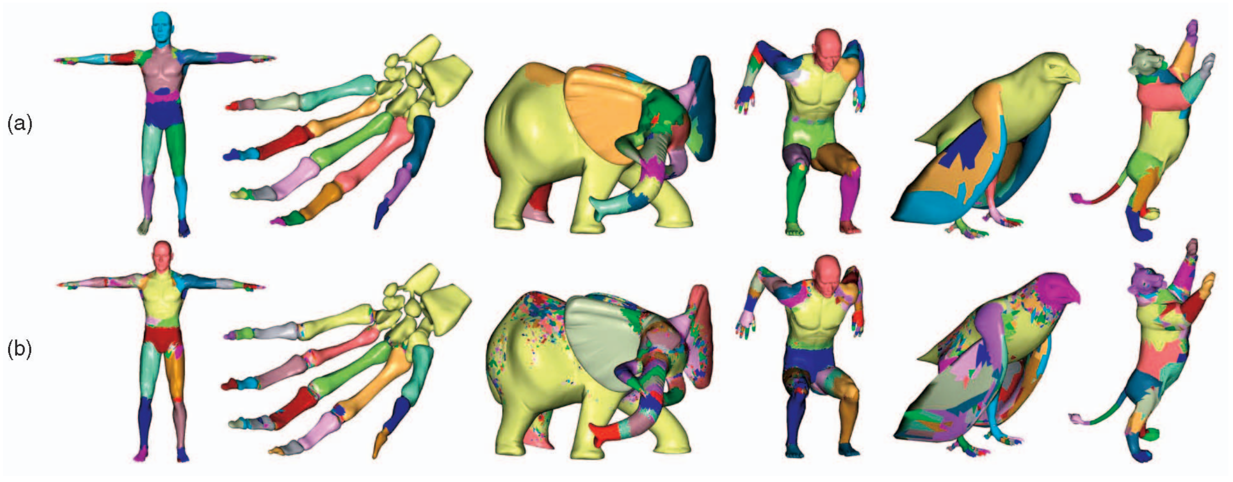

Fig. 16. Comparison of extracted near-rigid components. (a) The proposed MMS clustering algorithm. (b) The iterative application of MS clustering algorithm.

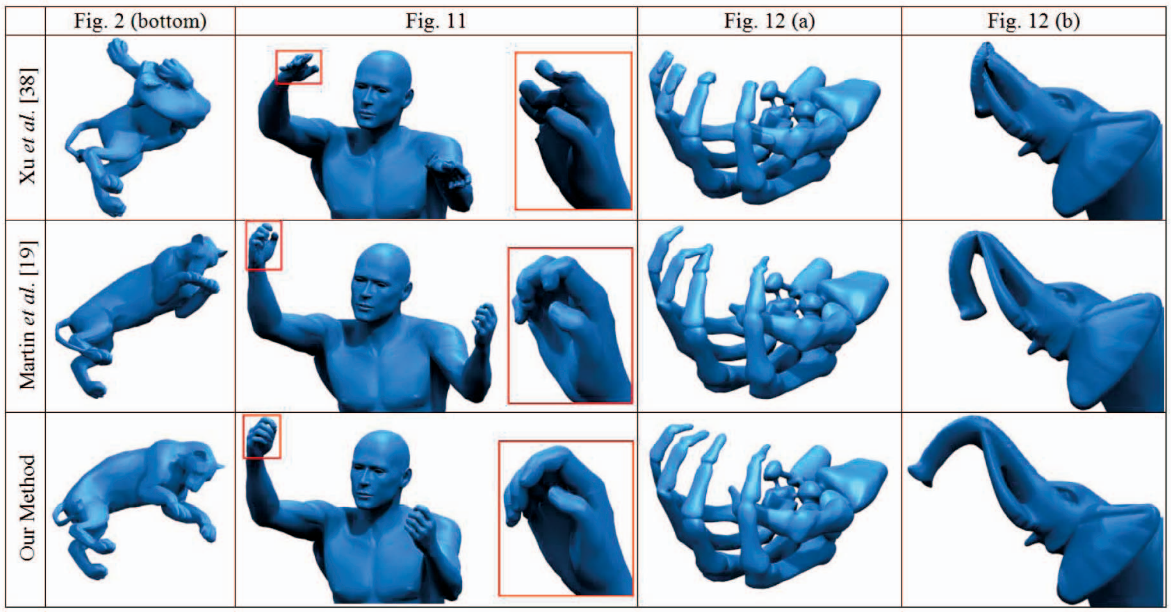

Fig. 17. Comparison of shape interpolation results with those of Xu et al. [38] and Martin et al. [19].

an excessive number of components. This limitation can be reduced by deforming one of the shapes using the technique in [34] to make two shapes vary only in pose and then extract the pose configuration from these two shapes. A more robust feature used in the MMS clustering, which is invariant to surface deformation is worth further exploration in the future. We do not apply any collision detection during the shape interpolation, and therefore, it may potentially generate self-collision among components which are far from each other in geodesic. In the future, we plan to include collision detection to resolve the self-collision problem in the shape interpolation.

Acknowledgments

The authors would like to thank anonymous reviewers' helpful comments to improve this paper. They are also grateful to Niloy J. Mitra and Martin Kilian for their help in performing experimental study with their work [19]. In addition, they thank AIM@SHAPE Shape Repository and Stanford 3D Scanning Repository for the 3D polyhedral models used in this paper. The eagle and human poses are taken from Poser 7. This work is supported in part by the Landmark Program of the NCKU Top University Project (Contract B0008), and the National Science Council (Contracts NSC-97-2628-E-006-125-MY3 and NSC-96-2628-E-006-200-MY3), Taiwan, Republic of China.

References

[1] M. Alexa, "Differential Coordinates for Local Mesh Morphing and Deformation," *The Visual Computer*, vol. 19, nos. 2/3, pp. 105-114, 2003.

[2] M. Alexa, "Merging Polyhedral Shapes with Scattered Features," *The Visual Computer*, vol. 16, no. 1, pp. 26-37, 2000.

[3] M. Alexa, "Linear Combination of Transformations," *ACM Trans. Graphics*, vol. 21, no. 3, pp. 380-387, 2002.

[4] M. Alexa, D. Cohen-Or, and D. Levin, "As-Rigid-As-Possible Shape Interpolation," *Proc. ACM SIGGRAPH '00*, pp. 157-164, 2000.

[5] D. Anguelov, D. Koller, H.-C. Pang, P. Srinivasan, and S. Thrun, "Recovering Articulated Object Models from 3D Range Data," *Proc. 20th Conf. Uncertainty in Artificial Intelligence*, pp. 18-26, 2004.

[6] D. Anguelov, P. Srinivasan, D. Koller, S. Thrun, J. Rodgers, and J. Davis, "Scape: Shape Completion and Animation of People," *ACM Trans. Graphics*, vol. 24, no. 3, pp. 408-416, 2005.

[7] P.J. Besl and N.D. McKay, "A Method for Registration of 3D Shapes," *IEEE Trans. Pattern Analysis Machine Intelligence*, vol. 14, no. 2, pp. 239-256, Feb. 1992.

[8] M. Botsch, D. Bommes, and L. Kobbelt, "Efficient Linear System Solvers for Mesh Processing," *Proc. IMA Conf. Math. Surfaces*, pp. 62-83, 2005.

[9] M. Botsch, R. Sumner, M. Pauly, and M. Gross, "Deformation Transfer for Detail-Preserving Surface Editing," *Proc. Vision, Modeling, and Visualization Conf. (VMV)*, pp. 357-364, 2006.

[10] E. Carmel and D. Cohen-Or, "Warp-Guided Object-Space Morphing," *The Visual Computer*, vol. 13, nos. 9/10, pp. 465-478, 1997.

[11] W. Chang and M. Zwicker, "Automatic Registration for Articulated Shapes," *Proc. Computer Graphics Forum*, vol. 27, no. 5, pp. 1459-1468, 2008.

[12] Y. Cheng, "Mean Shift, Mode Seeking, and Clustering," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 17, no. 8, pp. 790-799, Aug. 1995.

[13] D. Cohen-Or, A. Solomovic, and D. Levin, "Three-Dimensional Distance Field Metamorphosis," *ACM Trans. Graphics*, vol. 17, no. 2, pp. 116-141, 1998.

[14] D. Comaniciu and P. Meer, "Mean Shift: A Robust Approach Toward Feature Space Analysis," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 24, no. 5, pp. 603-619, May 2002.

[15] K.G. Der, R.W. Sumner, and J. Popović, "Inverse Kinematics for Reduced Deformable Models," *ACM Trans. Graphics*, vol. 25, no. 3, pp. 1174-1179, 2006.

[16] M. Garland and P.S. Heckbert, "Surface Simplification Using Quadric Error Metrics," *Proc. ACM SIGGRAPH '97*, pp. 209-216, 1997.

[17] D.L. James and C.D. Twigg, "Skinning Mesh Animations," *Proc. ACM SIGGRAPH '05*, pp. 399-407, 2005.

[18] T. Kanai, H. Suzuki, and F. Kimura, "Metamorphosis of Arbitrary Triangular Meshes," *IEEE Computer Graphics and Applications*, vol. 20, no. 2, pp. 62-75, Mar./Apr. 2000.

[19] M. Kilian, N.J. Mitra, and H. Pottmann, "Geometric Modeling in Shape Space," *ACM Trans. Graphics*, vol. 26, no. 3, p. 64, 2007.

[20] V. Kraevoy and A. Sheffer, "Cross-Parameterization and Compatible Remeshing of 3D Models," *Proc. ACM SIGGRAPH '04,* pp. 861-869, 2004.

[21] T.-Y. Lee, Y.-S. Wang, and T.-G. Chen, "Segmenting a Deforming Mesh into Near-Rigid Components," *The Visual Computer,* vol. 22, no. 9, pp. 729-739, 2006.

[22] J.E. Lengyel, "Compression of Time-Dependent Geometry," *Proc. Symp. Interactive 3D Graphics,* pp. 89-95, 1999.

[23] Y. Lipman, O. Sorkine, D. Levin, and D. Cohen-Or, "Linear Rotation-Invariant Coordinates for Meshes," *ACM Trans. Graphics,* vol. 24, no. 3, pp. 479-487, 2005.

[24] N.J. Mitra, L. Guibas, and M. Pauly, "Partial and Approximate Symmetry Detection for 3D Geometry," *ACM Trans. Graphics,* vol. 25, no. 3, pp. 560-568, 2006.

[25] N.J. Mitra, Personal Comm., 2009.

[26] N.J. Mitra, L.J. Guibas, and M. Pauly, "Symmetrization," *Proc. ACM SIGGRAPH '07,* p. 63, 2007.

[27] S.I. Park and J.K. Hodgins, "Capturing and Animating Skin Deformation in Human Motion," *ACM Trans. Graphics,* vol. 25, no. 3, pp. 881-889, 2006.

[28] E. Praun, W. Sweldens, and P. Schröder, "Consistent Mesh Parameterizations," *Proc. ACM SIGGRAPH '01,* pp. 179-184, 2001.

[29] S. Schaefer and C. Yuksel, "Example-Based Skeleton Extraction," *Proc. Symp. Geometry Processing,* pp. 153-162, 2007.

[30] J. Schreiner, A. Asirvatham, E. Praun, and H. Hoppe, "Inter-Surface Mapping," *ACM Trans. Graphics,* vol. 23, no. 3, pp. 870-877, 2004.

[31] T.W. Sederberg, P. Gao, G. Wang, and H. Mu, "2D Shape Blending: An Intrinsic Solution to the Vertex Path Problem," *Proc. ACM SIGGRAPH '93,* pp. 15-18, 1993.

[32] A. Sheffer and V. Kraevoy, "Pyramid Coordinates for Morphing and Deformation," *Proc. 3D Data Processing, Visualization, and Transmission Conf. (3DPVT),* pp. 68-75, 2004.

[33] K. Shoemake and T. Duff, "Matrix Animation and Polar Decomposition," *Proc. Conf. Graphics Interface,* pp. 258-264, 1992.

[34] R.W. Sumner and J. Popović, "Deformation Transfer for Triangle Meshes," *Proc. ACM SIGGRAPH '04,* pp. 399-405, 2004.

[35] R.W. Sumner, M. Zwicker, C. Gotsman, and J. Popović, "Mesh-Based Inverse Kinematics," *ACM Trans. Graphics,* vol. 24, no. 3, pp. 488-495, 2005.

[36] S. Toledo, "Taucs: A Library of Sparse Linear Solvers, Version 2.2," http://www.tau.ac.il/stoledo/taucs, 2003.

[37] O. Weber, O. Sorkine, Y. Lipman, and C. Gotsman, "Context-Aware Skeletal Shape Deformation," *Proc. Computer Graphics Forum,* vol. 26, no. 3, pp. 265-273, 2007.

[38] D. Xu, H. Zhang, Q. Wang, and H. Bao, "Poisson Shape Interpolation," *Proc. ACM Symp. Solid and Physical Modeling,* pp. 267-274, 2005.

[39] H.-B. Yan, S.-M. Hu, and R. Martin, "Skeleton-Based Shape Deformation Using Simplex Transformations," *Proc. Int'l Conf. Computer Graphics,* pp. 66-77, 2006.

[40] T.-Y. Lee and P.-H. Huang, "Fast and Intuitive Polyhedra Morphing Using SMCC Mesh Merging Scheme," *IEEE Trans. Visualization and Computer Graphics,* vol. 9, no. 1, pp. 85-98, 2003.

[41] C.-H. Lin and T.-Y. Lee, "Metamorphosis of 3D Polyhedral Models Using Progressive Connectivity Transformations," *IEEE Trans. Visualization and Computer Graphics,* vol. 11, no. 1, pp. 2-12, Jan./Feb. 2005.

[42] T.-Y. Lee, C.-H. Lin, Y.-S. Wang, and T.-G. Chen, "Animation Key-frame Extraction and Simplification Using Deformation Analysis," *IEEE Trans. Circuits and Systems for Video Technology,* vol. 18, no. 4, pp. 478-486, Apr. 2008.

Hung-Kuo Chu received the BS degree in computer science/engineering from the National Cheng-Kung University, Tainan, Taiwan, in 2003. Now, he is pursuing the PhD degree at the Department of Computer Science and Information Engineering, National Cheng-Kung University. His research interests include geometry modeling, mesh deformation, and perception analysis.

Tong-Yee Lee received the PhD degree in computer engineering from Washington State University, Pullman, in May 1995. He is currently a distinguished professor in the Department of Computer Science and Information Engineering, National Cheng-Kung University, Tainan, Taiwan, ROC. He leads the Computer Graphics Group, Visual System Laboratory, the National Cheng-Kung University (http://graphics.csie.ncku.edu.tw/). He is the recipient of the 2008 Distinguished Research Award, the 2005 and the 2006 First-Class Principal Investigator Award from the National Science Council of Taiwan, ROC, and the 2003 Youth Engineer Award, the Chinese Institute of Engineers, ROC. His current research interests include computer graphics, nonphotorealistic rendering, image-based rendering, visualization, virtual reality, surgical simulation, medical visualization and medical system, and distributed and collaborative virtual environments. He is an associate editor for the *IEEE Transactions on Information Technology in Biomedicine* with a tenure from 2000 to 2010. He is also on the editorial advisory board of the *Journal Recent Patents on Engineering*, an editor of the *Journal on Information Science and Engineering*, and a region editor of the *Journal of Software Engineering*. He served as a member of the international program committees of several conferences including IEEE Visualization, Pacific Graphics, the IEEE Pacific Visualization Symposium, the IEEE Virtual Reality, the IEEE-EMBS International Conference on Information Technology and Applications in Biomedicine, the Joint Virtual Reality Conference of EGVE-ICAT-EuroVR, the International Conference on Artificial Reality and Telexistence, and the International Conference in Central Europe on Computer Graphics, Visualization, and Computer Vision. He is a member of the IEEE and the ACM.