Shape Deformation Using a Skeleton to Drive Simplex Transformations

Han-Bing Yan, Shi-Min Hu, *Member*, *IEEE*, Ralph R. Martin, and Yong-Liang Yang

Abstract—This paper presents a skeleton-based method for deforming meshes (the skeleton need not be the medial axis). The significant difference from previous skeleton-based methods is that the latter use the skeleton to control movement of *vertices*, whereas we use it to control the *simplices* defining the model. By doing so, errors that occur near joints in other methods can be spread over the whole mesh, via an optimization process, resulting in smooth transitions near joints of the skeleton. By controlling simplices, our method has the additional advantage that no vertex weights need be defined on the bones, which is a tedious requirement in previous skeleton-based methods. Furthermore, by incorporating the translation vector in our optimization, unlike other methods, we do not need to fix an arbitrary vertex, and the deformed mesh moves with the deformed skeleton. Our method can also easily be used to control deformation by moving a few chosen line segments, rather than a skeleton.

Index Terms—Shape deformation, skeleton, simplex transformation, animation.

1 Introduction

Mesh deformation is widely used in computer animation and computer modeling to represent moving objects of changing shape. Many techniques have been developed to help artists sculpt stylized body shapes and corresponding deformations for 2D and 3D characters, for example. These techniques include free-form deformation (FFD), multiresolution approaches, differential methods, and skeleton-based methods.

The skeleton-based approach uses a *skeleton*, in which two or more *bones* meet at articulating joints to control shape deformation. This allows intuitive control as it naturally describes the way in which many objects such as animals deform the muscles and other tissues that follow motions of underlying bones in the skeleton. Such methods are usually controlled by a user-chosen skeleton, rather than a precisely determined mathematical medial axis. A serious problem, however, with traditional skeleton-based methods is that they require a tedious process of weight selection to obtain satisfactory results, as will be explained later. Worse, it seems that there is no criterion for weight selection that is universally applicable to *all* cases.

In this paper, we present a mesh deformation method that combines the skeleton-based method and the simplex transformation method. Although we still control deformation by a skeleton, our approach has two main differences from traditional skeleton-based methods. First, we use the skeleton motion to drive the transformation of *simplices*, rather than *vertices* as done in previous skeleton-based methods. Second, weights are *not used* in our method, avoiding the weight adjustment issue completely; nevertheless, our approach gives high-quality results.

Our approach can be applied to both 2D and 3D triangle meshes. The inputs to our method are the initial mesh, the initial skeleton, and the deformed skeleton; the skeleton is here considered to be a set of straight-line segments connected together at the joints. The output is the deformed mesh. In 2D, the simplices that are transformed are the triangles covering the 2D shape. In 3D, we produce a suitable set of tetrahedra for use as simplices based on the input 3D surface mesh.

The main steps of our method are listed as follows:

- We segment the mesh. We allocate each simplex to a nearby bone, which is the *controlling bone* for this simplex.
- We find a *transformation matrix* relating the initial and final position of each bone.
- We *apply* this transformation matrix to the simplices under that bone's control.
- We use *optimization* to ensure *connectivity* of the final simplices, keeping each simplex transformation as close as possible to the value determined by its bone.

This idea works not only for control based on adjustment of the skeleton, but can be extended to use *any* suitable small collection of internal lines to control mesh deformation. It can also be extended to *expand* or *shrink* bones if desired and to *twist* part of the mesh by defining twist axes.

This paper is an extended version of work reported at a conference [1]. Compared to our previous paper, the results are improved in several important places. First, we incorporate a translation term in the error energy function used to determine the deformed mesh: previously, like other simplex transformation methods, we needed to fix the location of an arbitrary vertex to locate the deformed mesh relative to the skeleton. Addition of the translation term

- *H.-B. Yan, S.-M. Hu, and Y.-L. Yang are with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, P.R. China. E-mail: {yanhb02, yangyongliang00}@mails.tsinghua.edu.cn, shimin@tsinghua.edu.cn.*
- *R.R. Martin is with the School of Computer Science, Cardiff University, CF24 3AA Wales, U.K. E-mail: Ralph.Martin@cs.cardiff.ac.uk.*

Manuscript received 21 July 2006; revised 28 Jan. 2007; accepted 2 Jan. 2008; published online 18 Jan. 2008.
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-0108-0706.
Digital Object Identifier no. 10.1109/TVCG.2008.28.

renders this unnecessary. This makes it easier for the user to generate a long animation sequence, giving an automatic way of ensuring smooth translation results if the skeleton is moved smoothly. The second improvement is that we have greatly improved the solving efficiency. The deformation speed is improved by incorporating Cholesky decomposition and back substitution, which makes our algorithm more competitive compared to other skeleton-based methods. Segmentation efficiency is increased. The third improvement is to incorporate graph cut in the segmentation method, and inaccurate segmentation results caused by incorrect joint positions at sharp feature can now be easily avoided. Finally, we show how to incorporate twists and also extend our skeleton-based method to allow control using a collection of internal lines, not just a skeleton; we give details and discuss limitations of this method.

2 Related Work

One of the best known methods for carrying out deformation is *FFD*. The classic FFD method [2] encloses a shape in an elastic lattice control volume such as a Bézier volume, then deforms the volume by moving its control vertices: as a result, the shape inside is deformed. This technique has been extended to use more general lattice structures [3]. Various curve-based deformation methods [4], [5] can also be classified as similar space-warping deformation methods. Such space-warping methods are very efficient, but they do not provide easy control over the details of complex models.

Multiresolution methods [6], [7], [8] have also been developed to deform shapes with intricate geometry. A detailed shape is first decomposed into a base shape and geometric details. The base shape is deformed using some suitable technique, and then, the geometric details are added back. The base mesh can be deformed using FFD or other approaches. This class of approach has the advantage of being efficient but care must be taken in how the details are added back if convincing results are to be obtained.

Differential deformation methods have become popular recently [9], [10], [11], [12]. Laplacian coordinates [9] are based on representing surface detail as differences from the local mean. Poisson mesh methods [12] are based on manipulating gradients of the mesh's coordinate functions and, then, reconstructing the surface from the Poisson equation. In [13], surface Laplacian coordinates are extended to volumetric Laplacian coordinates, allowing volume-preserving deformation. Shi et al. [14] developed a fast multigrid technique tailored for gradient field mesh deformation. Huang [15] use cotangent forms to represent Laplacian coordinates, which are invariant under rotation, scaling, and translation. This method needs to solve a nonlinear system; so, a subspace technique is used to accelerate the solving process.

Simplex transformation is another approach to deformation and morphing. The use of a matrix decomposition of a *global* transformation was proposed in [16] as a means of carrying out morphing. This method was extended to *local* transformations in [17], in which each triangle or tetrahedron is transformed independently, and the results are then made to connect consistently using an optimization method. Simplex transformation has also been used with surface triangle meshes to perform deformation learned from existing examples [18], [19]. This method was developed to control shape deformation by partition the mesh and control each partition by proxy vertices [20]. Botsch [21] give a mathematical proof that shows an equivalence between the simplex transformation method and those methods based on differential representations.

All of the above classes of methods have a similar disadvantage, in that they do not take into account the natural way in which many shapes' features are controlled. For example, vertebrate animals have a skeleton, and many other articulating objects such as robots can be modeled as if they also did. The shapes and movement of such objects can be understood in terms of the motion of a skeleton. Therefore, this provides a more intuitive approach to controlling the deformation of such shapes. Such ideas are also referred to as *skinning* [22], *envelopes* [23], or *skeletal subspace deformation* [24].

Existing skeleton-based algorithms define the final position of a point in the mesh as a weighted linear combination of the initial state of the point projected into several moving coordinate frames, one frame for each bone. The position of a point $\mathbf{p}'$ after deformation can be written as

$$\mathbf{p}' = \sum_{k=1}^{n} w_k \mathbf{p} M_k, \qquad (1)$$

where $\mathbf{p}$ is the point's initial position, M_k is a transformation matrix that transforms bone k from its initial position to its new position, w_k is the weight of this point relative to bone k, and n is the number of bones. Because in (1), each point is controlled by multiple bones, careful choice of weights w_k is needed, both to avoid self-intersections, especially near the joints, and also to keep the resulting shape smooth. Appropriate *weight selection* is an extremely tedious process if done manually. Thalmann [25] proposed the use of virtual bones to control the point positions, which avoids the use of weights. However, this method only allows a point to be influenced by at most two segments. Mohr [26] gave reasons why the traditional skeleton-based method is incapable of expressing complex deformations, and he suggested an interactive method to control weights of points.

In recent years, work has focused on how to learn weights from examples [24], [27], and [28]. Such learning methods mainly differ in the detail of how they represent the point displacements and in the particular interpolation method used. However, it seems that *no* single method works well in *all* cases [29]. To overcome this problem, the latter paper proposes a multiweight enveloping method: each *component* of the matrix M_k is given a *separate* weight to provide maximum flexibility, instead of a single weight for the whole matrix. Clearly, this means even more weights must be adjusted. A detailed introduction to the skinning method of learning deformation from examples is given in [30]. James and Twigg [28] describe a skinning method without the need for explicitly specifying a skeleton, where a virtual skeleton is generated by the mean shift clustering method.

There has also been much work [28], [30], [31] on accelerating skeleton-based methods using hardware.

In short, the basic problem with skeleton-based methods is that each point in the mesh is updated independently using (1), which requires the w_i to be carefully chosen to avoid gaps and artifacts. However, the points are embedded in a shape and are related. The mesh provides *connectivity constraints*; previous skeleton-based methods have not directly used this information. We use this information explicitly to our advantage. By retaining a skeleton, we keep its merits of providing a natural and easily understood control mechanism. By using the connectivity information, we avoid the weight adjustment issue arising in traditional skeleton-based methods and instead solve a linear equation to perform a similar task. This approach is easier and gives high-quality results.

There has been much work on skeletonization [32], [33], [34] and segmentation [35], [36], [37], [38] as independent problems, or on creating a skeleton and corresponding segmentation together [39], [40], [41], [42], [43]. In this paper, we are interested in the segmentation method, assuming there is a given skeleton. Therefore, we only focus on the methods that can create skeleton and segmentation together or those that can create segmentation by a skeleton. Katz and Tal [39] compute a fuzzy decomposition by an iterative clustering scheme, then refine the decomposition by Graph-Cut method using geodesic and angular distances. Further more, their segmentation can be used to compute a skeleton. Lien et al. [40] suggested how to create a skeleton and perform segmentation using an iterative approach, but this is not ideal as the skeleton changes during iteration. Cornea et al. [41] also proposed a segmentation method that follows skeleton creation. This method segments the mesh by tracing the field lines, which was defined during skeleton creation. Therefore, it also is not appropriate to be used in segmentation with a given skeleton.

In many cases, artists prefer to specify a skeleton, in order to be able to achieve the desired animation control, rather than having an automatically determined skeleton. Therefore, it is useful to perform segmentation from a given skeleton directly, without using the metainformation in the skeleton creation. Li et al. [42] showed how to obtain a segmentation corresponding to a given skeleton using a space sweeping method, but this does not seem to work well if the skeleton is rather coarse. A similar method was proposed in [43], while it takes the shortest geodesics between feature points as borders instead of a cross-section sweeping line. We thus give a new method to segment the model, which takes into account both euclidean distance and shortest path distance between simplices and skeleton bones. Our results show that this method, while simple, is effective.

For further discussions on skeletons and segmentation, the reader is referred to the excellent survey by Cornea et al. [44].

Bloomenthal [45] used a segment-based skeleton to deform the shape's mathematical medial axis, which was then used to drive the shape deformation. Yoshizawa et al. [46] extended this work by combining it with the Laplacian coordinates approach.

While revising this paper, the work by Anguelov et al. [47] came to our attention. Some of its techniques are similar to those in our earlier paper [1], but it differs in the following ways. First, we proposed a segmentation method by using a given skeleton. Second, we try to keep the transformation matrix of each mesh simplex as similar as possible to the corresponding bone's transformation while they try to keep the mesh edge vectors as close as possible to the transformed original mesh edge vectors. Third, like in our conference paper [1], their error energy function has no term to keep the mesh moving together with the skeleton. As pointed out earlier, this means that at least one vertex must be fixed before solving the optimization function, which our current work overcomes. This will be discussed in detail in Section 6.3. It seems that more and more people show their interest in combining the skeleton-based deformation methods and the deformation methods using mesh local attribute such as [48] and [49]. The difference between our paper and theirs is similar to our paper and [47].

In the rest of this paper, we outline basic concepts concerning simplex transformations and skeletons in Section 3. We first show our mesh segmentation method based on the skeleton in Section 4 and then show how to calculate the bones' transformations in Section 5. Sections 6 and 7 give our skeleton-based mesh deformation methods, illustrating them with practical results. Conclusions and discussions are given in Section 8. Both 2D and 3D triangle meshes are considered.

3 Simplex Transformations and Skeletons

3.1 Simplex Transformations

A simplex is the simplest possible polytope in a given space: triangles and tetrahedra are the highest dimension simplices in 2D and 3D. Given two such simplices S_1 and S_2 in some space, there exists a unique transformation that changes S_1 into S_2. This can be written as

$$v_i = Mu_i + T, \tag{2}$$

where M is an affine transformation matrix representing rotation and shape change information, T is the translation vector, the u_i are the vertices of S_1, and the v_i are the corresponding vertices of S_2. M and T can be calculated from the vertex coordinates of S_1 and S_2, as follows:

$$M = VU^{-1}, \tag{3}$$

where in 2D

$$\begin{aligned} V &= [\, v_1 - v_3 \quad v_2 - v_3 \,], \\ U &= [\, u_1 - u_3 \quad u_2 - u_3 \,], \end{aligned} \tag{4}$$

and in 3D,

$$\begin{aligned} V &= [\, v_1 - v_4 \quad v_2 - v_4 \quad v_3 - v_4 \,], \\ U &= [\, u_1 - u_4 \quad u_2 - u_4 \quad u_3 - u_4 \,]. \end{aligned} \tag{5}$$

Having found M, it can be substituted into (2) to find T.

3.2 Skeletons

The strict mathematical skeleton, or medial axis, is the locus of the centers of all maximal spheres contained within the object. Generally, it is quite complex even for simple 3D shapes and may contain sheets, as well as curvilinear elements. It is also sensitive to small perturbations of the shape boundary. For simplicity, most skeleton-based deformation methods use an approximate skeleton to control deformation, consisting of straight lines of zero thickness—*bones*—connected at articulated joints.

In many cases, artists prefer to create the skeleton by hand to give the desired degree of control over the shape. It is not very difficult to create such an approximate skeleton interactively. Automatic methods of skeleton generation also exist, such as [34] and [42].

4 Triangle Mesh Segmentation

In this section, we consider how to segment the triangle mesh using the skeleton, which decides which triangles are controlled by each bone. The results of our segmentation method intuitively correspond to near-rigid components such as a foot, a lower leg, an upper leg, and so on. We still follow the basic segmentation approach presented in our previous paper [1], but some important improvements have been made to accelerate it and to overcome previous limitations. We first summarize the segmentation method used in our previous work and then explain our improvements.

4.1 Mesh Segmentation Using the Skeleton

We now briefly review the segmentation method from our previous work; further details are given in [1]. In this approach, each triangle is controlled by just *one* bone. We thus need to *segment* the model according to the given skeleton or, in other words, decide which bone should be used to control each triangle. Having done this, we can then decide how each triangle should deform. It should be noted that although we create a tetrahedron for each triangle for the purposes of 3D deformation, as discussed in Section 6.2, the tetrahedra need not be created during the segmentation phase. We segment the mesh according to euclidean distance and the shortest path distances on the mesh as follows:

1. Decide the control domain of each bone using range planes placed at the end of each bone. Range planes decide which triangles a bone can possibly control. At bones with free ends, range planes are orthogonal to the bones; where bones meet, range planes bisect the angle between bones.
2. Decide which of those bones actually controls each triangle by choosing the bone with minimum *effective distance with penalty* to the triangle [1]. The *effective distance with penalty* can be written as

$$d_{\mathrm{effpen}} = d_{\mathrm{eff}} + n\delta, \tag{6}$$

   where d_{eff} is the *effective distance*, and n is the number of intersections of the *effective line* and the mesh boundary. *Effective distance* is the distance from the triangle center to the bone along the *effective line*, a line that takes into account the orientations of the range planes at the ends of the bones. The idea here is to that if we have to pass outside the mesh on the shortest line from the bone to the simplex, such a bone is (probably) not a good choice for the controlling bone for this simplex. A binary tree is constructed to accelerate the calculation of the number of intersections.
3. Check if the minimum *effective distance with penalty* is less than a threshold—if so, it means the simplex can be *seen* by one or more bones (that is, the straight line referred to above does not cross the mesh boundary):

   - a) If it is less than the threshold, the bone with the minimum *effective distance with penalty* controls this simplex.
   - b) If it is more than the threshold, calculate the *shortest path distance* in the mesh from the simplex to the bones for which it is within range. The bone with the shortest path distance is the control bone.

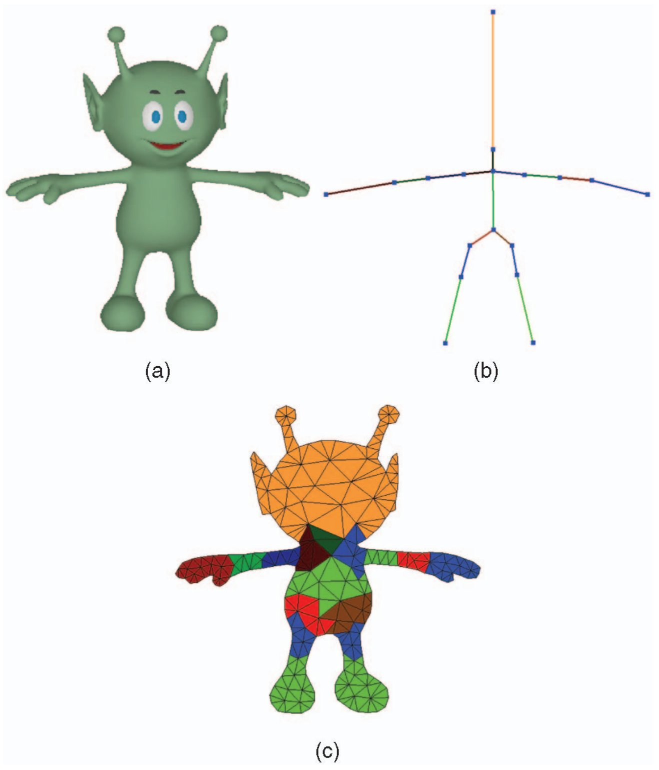


Fig. 1. 2D skeleton control domain: (a) cartoon character, (b) skeleton, (c) skeleton control domain.

Normally, a skeleton is thought of as lying within the volume defined by the mesh. However, in our method, we require any *free* ends of bones of the skeleton (that is, ends not connected to other bones) to lie just *outside* the mesh to ensure that each bone properly controls all the triangles in its control domain. If a given skeleton has free ends within the mesh, it is straightforward to extend them outside the mesh automatically.

Figs. 1, 2, and 3 show segmentation results using this method. Fig. 1a shows a 2D cartoon character, Fig. 1b shows

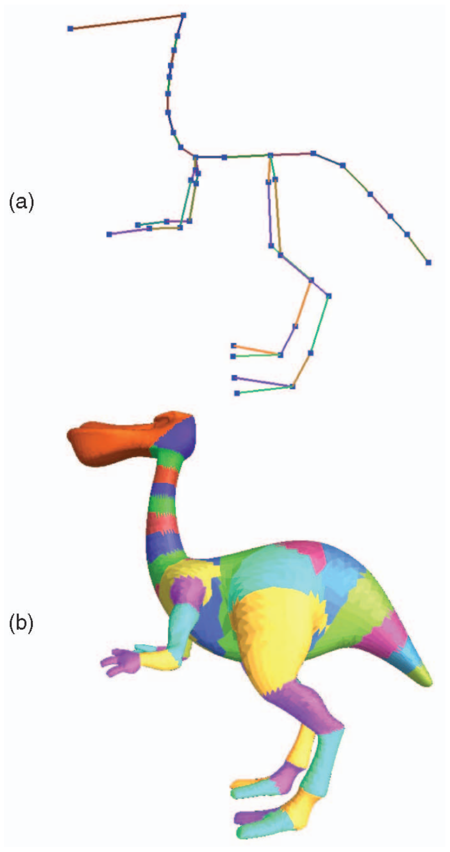


Fig. 2. (a) Skeleton. (b) Control domain of Dinopet.

a corresponding 2D skeleton, and Fig. 1c shows which bone controls each triangle, as determined by the method above. Triangles of a given color are controlled by the bone of the same color. Fig. 2a shows a 3D skeleton for the Dinopet, and Fig. 2b shows the control domain of each bone.

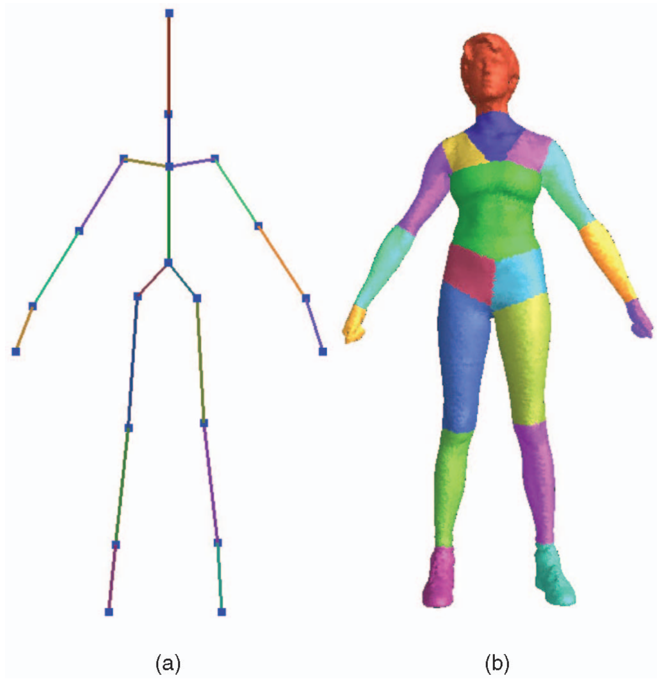


Fig. 3. (a) Skeleton. (b) Control domain of woman.

TABLE 1
Timing Information

	Arma	Dinopet	Horse	Female	Palm
Vertices	50852	13324	8433	30432	12782
BSP Tree	1.21	0.42	0.31	0.81	0.42
Segmentation	1.48	0.37	0.28	0.74	0.36
Cholesky	5.13	0.92	0.57	2.97	0.89
Back Substitution	0.69	0.10	0.06	0.33	0.10

4.2 Accelerating Segmentation

In 3D, due to the potentially very large size of the mesh, it would be very time consuming to test all bones against all triangles to decide the controlling bone for each triangle. It is easy to show that if a given triangle is controlled by a particular bone, then each of its neighboring triangles must be controlled by the same bone or some bone adjacent to that bone. This observation can be used to accelerate segmentation via a traversal process.

We start by selecting a triangle *intersected* by a bone with a free end as the initial triangle. Clearly, this triangle can be seen from that bone and has minimum *effective distance with penalty* to that bone so is controlled by it. We next find the control bones of those triangles adjacent to this triangle using the same criteria as before but only considering the same bone and bones connected to it. We then pass to the neighbors of these triangles, and so on. In principle, it is possible to construct extreme examples for which this accelerated method does not work well, but in practice, this method greatly speeds the segmentation process, giving greater savings the more bones there are in the skeleton. Of course, a similar method can also be used to accelerate 2D segmentation, but the generally smaller mesh sizes lead to reduced benefits.

The time spent in creating the segmentation and BSP tree creation for various 3D models is listed in Table 1. Experiments show that our segmentation algorithm is very effective and robust.

4.3 Overcoming Limitations

Our segmentation method is based on the skeleton, and thus, its results depend on the positions of joints. In places where features are not sharp, positions of joints and corresponding range planes are not crucial. However, they must be carefully located where the mesh has sharp features.

For example, Fig. 4a shows a mesh with a sharp feature and an inaccurately placed skeleton joint: Fig. 4b is the corresponding segmentation result created from the skeleton.

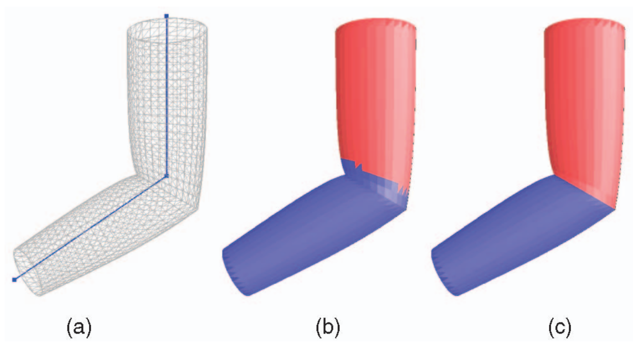


Fig. 4. Segmentation affected by the joint position.

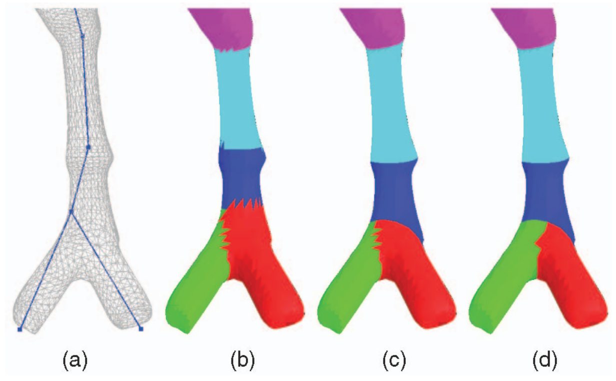


Fig. 5. Optimization process for inaccurately placed joint position.

Obviously, the segmentation result is not appropriate and will adversely influence the final deformation results.

In our previous work, we provided an interactive approach allowing the artist to modify both the joint positions and make small changes to range line and range plane orientations. Such an approach is labor intensive and requires careful work.

As an alternative, in this paper, we propose the use of a graph-cut method to optimize the segmentation boundaries to accommodate sharp features, following [39]. If a feature exists near the boundary where two segmentation components meet, we do the following:

1. Find the boundary vertices between these two components and generate an extended region on either side for boundary optimization.
2. Build the *dual graph* of the extended region. In this dual graph, compute the capacity of each edge based on dihedral angles.
3. Find the optimized boundary by applying a maximum flow algorithm.

Fig. 4c gives the final segmentation result optimized from the result in Fig. 4b.

If several components meet near a sharp feature, we use a hierarchical algorithm to refine the segmentation result from coarse to fine. First, we allocate these components to two large regions, where each large region contains one or several connected components. The boundary between the two large regions is optimized with respect to the feature. The optimization algorithm is then applied iteratively by dividing each large region into two smaller regions until each region has only one component.

Fig. 5 shows the optimization process for one leg of the Dinopet. Fig. 5a gives the mesh and the inappropriate placed skeleton, where its ankle joint and heel joint are inaccurately placed. Fig. 5b shows the original segmentation obtained by the method in Section 4.1. Fig. 5c shows the improved segmentation result after the first round optimization. The boundary at the knee is smoother, while the two toes are combined to give one region with a smooth boundary at the ankle. In Fig. 5d, the boundary between two toes is optimized. This boundary optimization process is fast enough to interactively display its effects on the segmentation: much less time is taken than for segmenting the whole model, as given in Table 1.

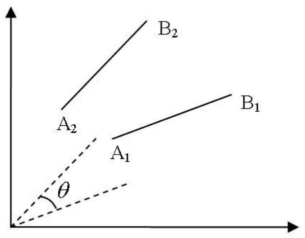


Fig. 6. 2D bone transformation.

We should note that this maximum flow algorithm is based on dihedral angle calculations, which can give good results at places where *sharp features* exist but which works less well in smooth regions. Thus, we only use this method to interactively optimize the component boundaries near sharp feature when we are not satisfied, after performing segmentation, as in Section 4.1. Small variations in joint positions in *smooth* regions have very little visual effect on deformation results, and such optimization is not necessary there.

5 Transformation of Bones

In our method, simplex transformations are derived from the transformations of bones. This section discusses how the bones' transformations are computed.

5.1 Transformation for 2D Bones

Given an initial skeleton and the corresponding deformed skeleton determined by the user, the transformation matrix for each bone can be calculated. Fig. 6 shows a bone at A_1B_1 in the initial skeleton and at A_2B_2 in the deformed skeleton. We initially calculate the bone transformation matrix without scaling, as bones normally have a fixed length. We later show how to take scaling into account if additionally required. When we have the transformation matrix of each bone, the translation vectors can be easily calculated. In the following, we use $\tilde{}$ to represent quantities related to bones.

Without scaling, we translate A_1B_1 so that A_1 coincides with the origin, the translation vector being $\widetilde{T}_{R1}$. A_1B_1 is then rotated anticlockwise through an angle θ around the origin until it lies in the same direction as A_2B_2. We then translate A_1B_1 so that A_1 coincides with A_2, the translation vector being $\widetilde{T}_{R2}$. This transformation process can be expressed as

$$\tilde{v} = \widetilde{R}(\tilde{u} + \widetilde{T}_{R1}) + \widetilde{T}_{R2}, \tag{7}$$

where $\tilde{u}$ is any point on the bone A_1B_1, and $\tilde{v}$ is the corresponding point after transformation. The transformation matrix is given by

$$\widetilde{R} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}, \tag{8}$$

and the translation vector is

$$\widetilde{T}_R = \widetilde{R}\widetilde{T}_{R1} + \widetilde{T}_{R2}. \tag{9}$$

Now, consider the case with scaling. Suppose the scale factor is α, so that after deformation, the bone has a length α

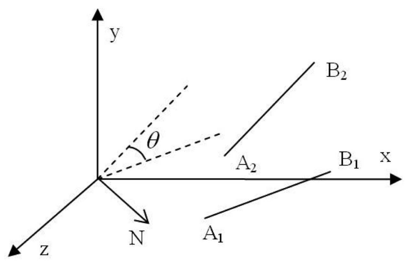


Fig. 7. 3D bone transformation.

times its original length. After translating A_1B_1 as before, using the translation vector $\widetilde{T}_{S1}$, we rotate it until B_1 is located on the x-axis using a rotation matrix $\widetilde{R}_{S1}$. We then scale A_1B_1 until it has the same length as A_2B_2 using a scaling matrix $\widetilde{S}$:

$$\widetilde{S} = \begin{bmatrix} \alpha & 0 \\ 0 & \beta \end{bmatrix}, \tag{10}$$

where β is the scale factor in the direction perpendicular to the bone. Usually, the animator will choose β to be 1.0, or the same as α, but may also use other values if desired. Finally, we rotate A_1B_1 into the same orientation as A_2B_2 and translate A_1B_1 until A_1 coincides with A_2; the rotation matrix involved is $\widetilde{R}_{S2}$, and the translation vector is $\widetilde{T}_{S2}$. Overall, we can write

$$\widetilde{w} = \widetilde{R}_{S2}\widetilde{S}_x\widetilde{R}_{S1}(\widetilde{v} + \widetilde{T}_{S1}) + \widetilde{T}_{S2}, \tag{11}$$

where $\widetilde{v}$ is a point on bone A_1B_1 after rotation, and $\widetilde{w}$ is the corresponding point after scaling. The overall transformation matrix in this step is given by $\widetilde{S} = \widetilde{R}_{S2}\widetilde{S}_x\widetilde{R}_{S1}$.

Substituting (7) into (11), we can write the overall transformation as the combination of a rotation and a scaling

$$\widetilde{M} = \widetilde{S}\widetilde{R}, \tag{12}$$

where if there is no scaling, $\widetilde{S}$ is a unit matrix. The whole translation vector is now

$$\widetilde{T} = \widetilde{S}(\widetilde{T}_R + \widetilde{T}_{S1}) + \widetilde{T}_{S2}. \tag{13}$$

For the later convenience, we write $\widetilde{T}_S = \widetilde{S}(\widetilde{T}_R + \widetilde{T}_{S1}) + \widetilde{T}_{S2}$.

5.2 Transformation for 3D Bones

We now consider how to calculate the transformation matrix for bones in 3D. In Fig. 7, suppose A_1B_1, A_2B_2 represent a bone in 3D before and after deformation. We translate A_1B_1 so that A_1 lies at the origin. We then create a unit vector N based at the origin, perpendicular to both A_1B_1 and A_2B_2, and rotate A_1B_1 around N until A_1B_1 is in the same direction as A_2B_2; let θ be the rotation angle. Finally, we translate A_1B_1 until A_1 coincides with A_2. The transformation matrix $\widetilde{R}$ can be calculated in a similar way to the 2D case and is found to be

$$\widetilde{R} = \begin{bmatrix} a^2 + \rho_{bc}\nu & ab\lambda + c\mu & ac\lambda - b\mu \\ ab\lambda - c\mu & b^2 + \rho_{ac}\nu & bc\lambda + ab\mu \\ ac\lambda + b\mu & bc\lambda - ab\mu & c^2 + \rho_{ab}\nu \end{bmatrix}, \tag{14}$$

where $N = (a, b, c)$, $\mu = \sin\theta$, $\nu = \cos\theta$, $\lambda = (1 - \cos\theta)$, $\rho_{ab} = a^2 + b^2$, $\rho_{bc} = b^2 + c^2$, and $\rho_{ac} = a^2 + c^2$. If scaling is also required, we can determine the scale matrix S, as in Section 5.1. The overall transformation matrix has the same form, as in (12), while the translation vector for each bone has the same form as given in (9) and (13).

6 Triangle Mesh Deformation

We now discuss how to drive the triangle mesh deformation using the skeleton transformations.

6.1 2D Triangle Mesh Deformation

If every triangle were to transform rigidly in the same way as its controlling bone, gaps or overlaps would occur between the triangles controlled by adjacent bones, causing tears or overlaps in the object. We need to enforce vertex consistency requirements to ensure the mesh retains its original *connectivity*.

We do this using an *optimization* method, which enforces connectivity while trying to keep each simplex transformation as close as possible to that of its control bone. An error function is used to represent the difference between the actual triangle deformation and the deformation indicated by the control bone, defined by

$$E = \sum_{i=1}^{n} A_i \Big(\|M_i - \widetilde{M}_i\|_F^2 + \alpha \|T_i - \widetilde{T}_i\|_2^2 \Big), \tag{15}$$

where n is the number of triangles in the mesh, M_i is the actual transformation matrix for the ith triangle, given by (3). T_i is the actual translation vector, which can be calculated by (2). $\widetilde{M}_i$ is the ideal transformation matrix of this triangle, which is the transformation matrix of the controlling bone of this simplex and is given by (12). $\widetilde{T}_i$ is the ideal translation vector and is given by (13). F is the Frobenius norm. α is the square of the reciprocal of the diagonal length of the original mesh bounding box, which is used to eliminate the influence of the mesh size. A_i is the area of the ith triangle, which is used to take account of the triangle area: large triangles should provide a greater contribution to the error energy function. We minimize E to get the best deformation results while ensuring mesh connectivity: the variables in the minimization problem are the vertex coordinates of the deformed mesh.

This classical quadratic optimization problem can be transformed into a linear equation by setting the gradient of E to zero, which can be written in the form:

$$K'X' = d', \tag{16}$$

where this linear system factors into two independent subsystems corresponding to the x- and y-coordinates of the deformed mesh; furthermore, the coefficient matrix for each subsystem is the same. We obtain

$$K^TKX = K^Td_x, \quad K^TKY = K^Td_y, \tag{17}$$

where X and Y are the x- and y-coordinate vectors of the deformed mesh, of dimension m, the number of vertices in the mesh. K is a sparse matrix of size $m \times m$, and d_x and d_y are vectors with dimension m. K^T is the transpose form of matrix K. We use the direct Cholesky decomposition and back substitution to solve these sparse linear systems.

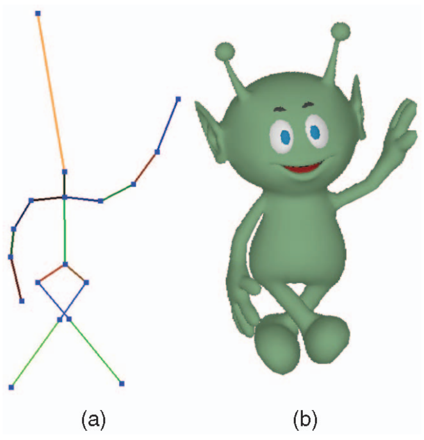


Fig. 8. (a) Deformed 2D skeleton. (b) Deformed cartoon character.

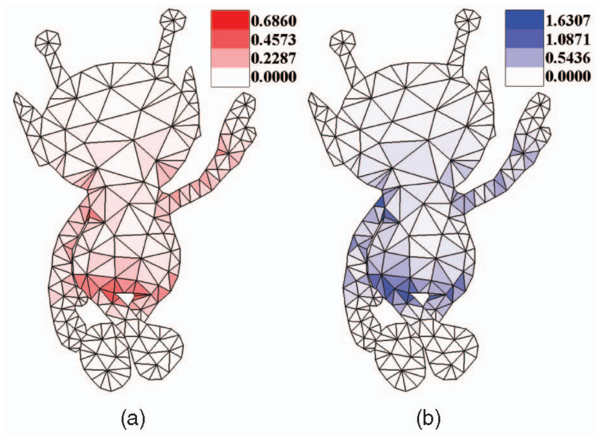


Fig. 9. Distortion spreading. (a) Area change. (b) Angle change.

Fig. 8 shows a deformed skeleton and the resulting deformed mesh for the cartoon character in Fig. 1. The corresponding mesh has 251 vertices, and 0.02 s were required to calculate the result on a 3.2-Ghz Pentium 4 machine.

The main difference between our method and traditional skeleton-based deformation methods is that we use bones to drive the *triangles* while they use bones to drive *vertices*. Although each triangle tries to follow the transformation determined by its control bone, it cannot follow it absolutely—otherwise, there would be gaps between adjacent triangles, especially for those located near joints. Note, on the other hand, that if only a few triangles located near joints changed their shape to preserve connectivity while others precisely followed the transformations given by their control bones, the error energy defined in (15) would be very large. By spreading such triangle distortion to surrounding triangles, the error is greatly reduced: our optimization method results in triangle shapes that are as close as possible to the original triangle shapes.

In Fig. 9, an example is given to show how the distortion varies according to distance from a joint. The distortion extent can be described by area change and internal angle change of triangles. Fig. 9a represents the triangle area change over the mesh. The triangle area change is calculated by $|\triangle A|/A$, where A is the original triangle area, and $|\triangle A|$ is the triangle area change after deformation. Fig. 9b represents the triangle angle change, which is calculated by $|\sum_{i=1}^{3}\triangle\alpha_i|$, where $|\triangle\alpha_i|$ is an internal angle change in the triangle. In these images, the lighter the triangle color, the less the triangle distortion. The distortion is the heaviest near joints but not limited at joints. The distortion spreads to the middle of the bone and to the free joints while it becomes lighter. The triangles near the middle of the bone, and the free joints always have the lightest distortion.

We can also use *strain* [50], a quantitative analysis tool of deformation to analyze the distortion extent of our deformation results. Using strain, we get very similar results, as in Fig. 9, that the triangle deformation is spread from joints to the middle of bone and free joint while becoming lighter.

6.2 3D Triangle Mesh Deformation

The above method can also be extended to a 3D *tetrahedron* mesh, but in practice, surface *triangle* mesh models are far more widely used in computer graphics. Furthermore, triangle mesh models have far fewer elements than tetrahedron models—the latter would require much higher processing times. Thus, for simplicity, here, we consider the 3D triangle mesh case, rather than the tetrahedron case.

With regards to deformation, the 3D triangle mesh case is very different from the 2D triangle mesh case, because a triangle is not a maximal dimension simplex in 3D, nor is there a unique transformation matrix for changing one triangle into another. Sumner and Popovic [18] gave an ingenious way of extending a simplex transformation method to a 3D triangle mesh by constructing a tetrahedron for each triangle. Here, we basically use the same method for constructing a tetrahedron, except that we put the new vertex above the centroid of the triangle rather than over one of its vertices. Doing so makes the following equations symmetric in x, y, and z coordinates, simplifying the coding of (18).

We add a fourth vertex to each triangle of both the initial and deformed mesh to give a tetrahedron. For the initial mesh, the fourth vertex is added in the normal direction over the triangle's centroid. Let v_1, v_2, and v_3 be the vertices of a triangle on the initial mesh. The fourth vertex is placed at

$$v_4 = \frac{(v_1+v_2+v_3)}{3} + \frac{(v_2-v_1)\times(v_3-v_2)}{\sqrt{(v_2-v_1)\times(v_3-v_2)}}.$$

The distance between v_4 and the centroid is determined in such a way as to ensure a well-shaped tetrahedron. The above equation is only used to calculate v_4 in the initial mesh; vertices in the deformed mesh, including v_4, are determined by the optimization process.

The 3D triangle mesh is now deformed using the same optimization approach as for the 2D triangle mesh in Section 6.1. In this case, the 3D version of (16) separates into three independent linear subsystems:

$$K^TKX = K^Td_x, K^TKY = K^Td_y, K^TKZ = K^Td_z. \tag{18}$$

The dimension of the vectors in (16) is now $m+k$, and K is an $(m+k)\times(m+k)$ matrix for a mesh with m vertices

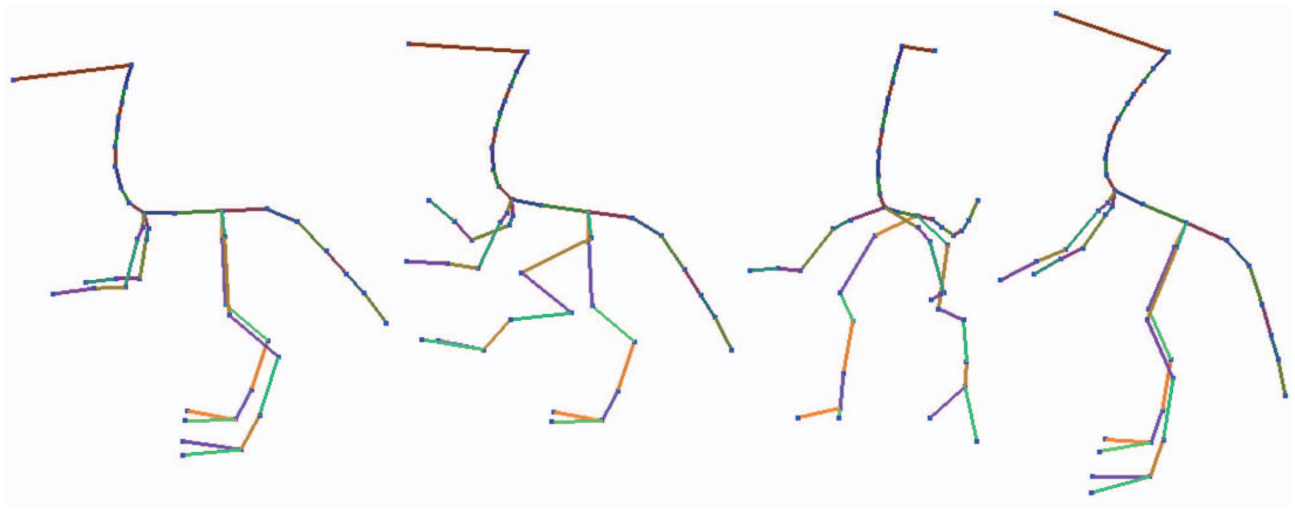


Fig. 10. Dinopet skeleton.

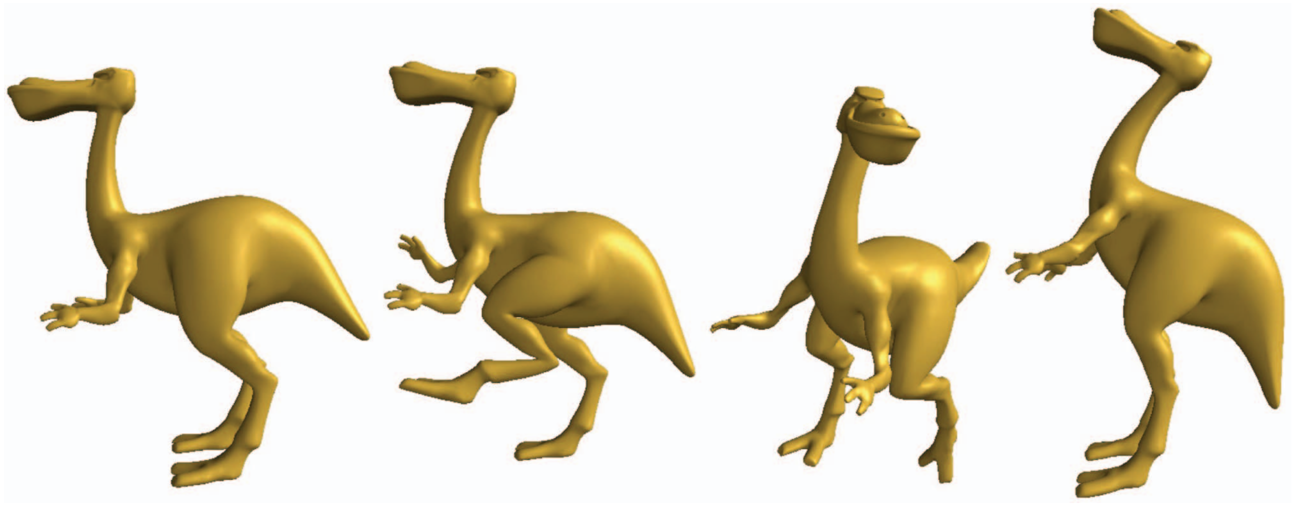


Fig. 11. Dinopet model.

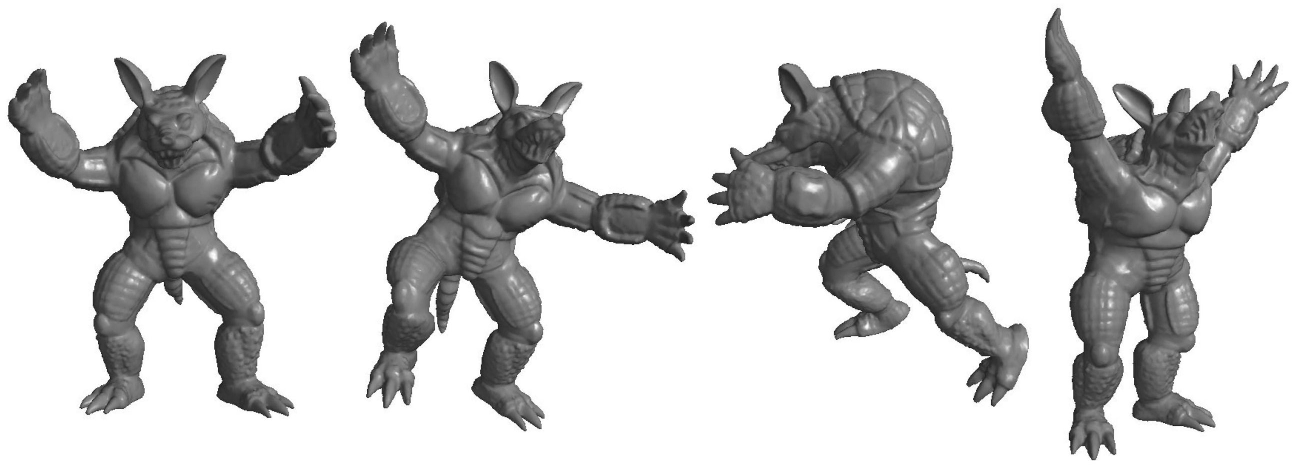


Fig. 12. Armadillo model.

and k faces. We use direct Cholesky decomposition and back substitution to efficiently solve these large sparse linear equations.

Figs. 10 and 11 give the Dinopet skeletons and results using our technique. Figs. 12, 13, 14, and 15 illustrate other 3D deformation results. The first model in each Figure is the original model; others are deformed results produced by our method. All results were calculated on a 3.2-Ghz Pentium 4 machine. Table 1 shows the times taken to deform the 3D models illustrated in this paper for one time, listing

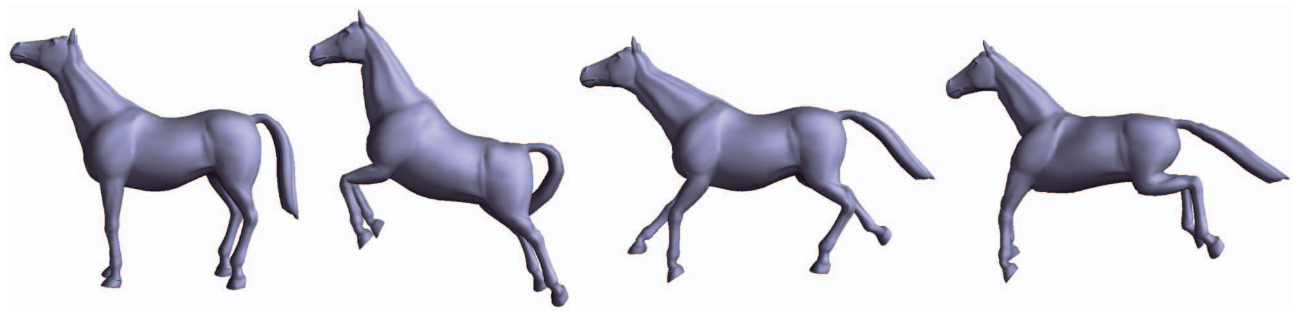


Fig. 13. Horse model.

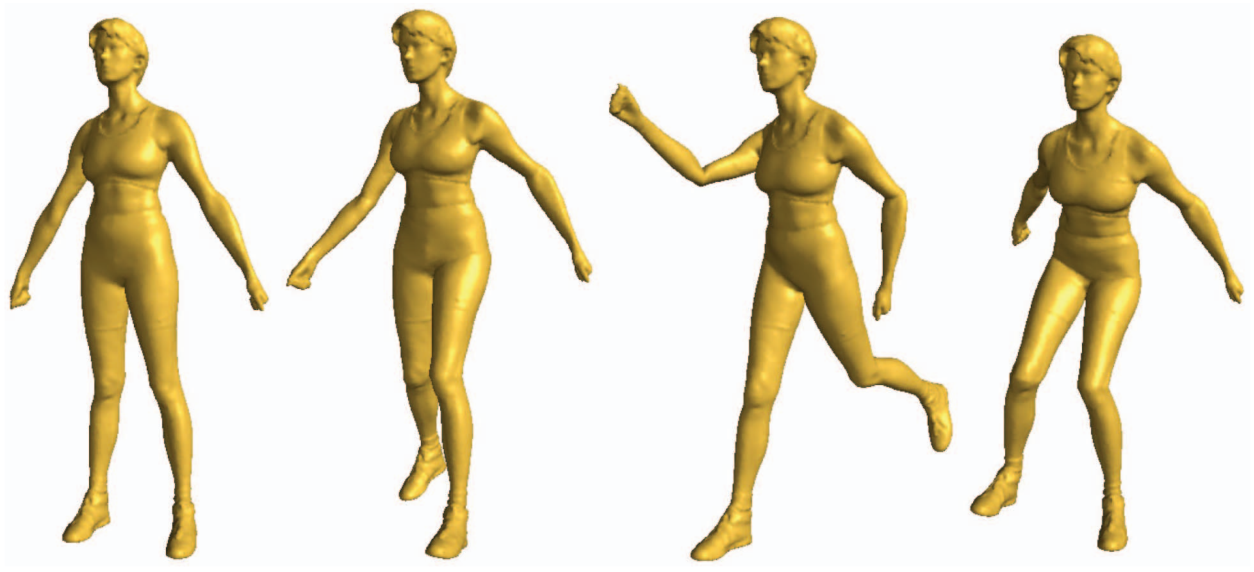


Fig. 14. Female model.

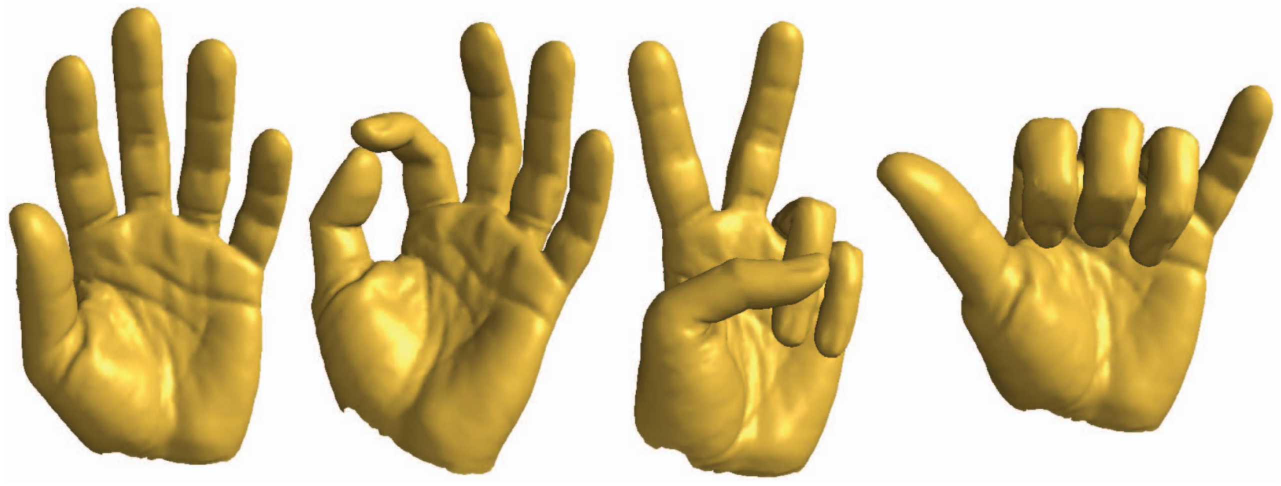


Fig. 15. Palm model.

separately times for BSP Tree Creation, Segmentation, Cholesky decomposition, and Back Substitution for each deformation.

6.3 Deformation without Translation

In Section 6.1 and 6.2, the error energy function contains both a transformation matrix and a translation vector. As done in our previous work [1], it is possible to create an error energy function that ignores the translation vector. The simplified energy function is

$$E = \sum_{i=1}^{n} A_i \| M_i - \widetilde{M}_i \|_F^2, \tag{19}$$

which can essentially be solved as before with one significant difference—this basically affects the position of the deformed model and has an insignificant effect on its shape.

In this case, if the deformed mesh is translated by some vector, the translated mesh will have the same error energy as the untranslated mesh: translation does not change the

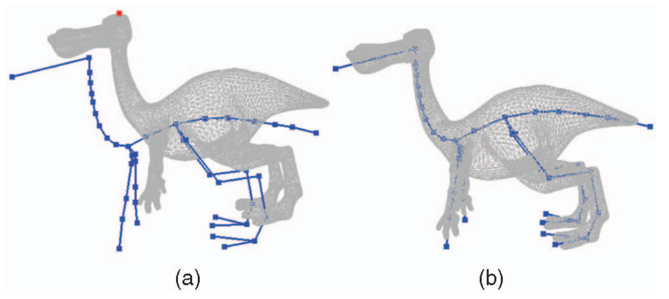


Fig. 16. (a) Deformation without translation. (b) Deformation with translation.

error energy in (19). Thus, the problem of minimizing the error energy in (19) has an infinite number of solutions, and the coefficient matrix K in (17) or (18) is singular. To obtain a unique solution, the simplest approach is to fix the position of one vertex of the mesh.

Fig. 16a shows the deformed skeleton and the deformed Dinopet model if translation vectors are not taken into account; the red point is the fixed vertex. Fig. 16b shows the results taking into account the translation vector.

Different choices of the fixed vertex may lead to different final mesh positions, even though the deformed meshes have the same shape. However, when making an animation using a skeleton, it is important that, as well as the mesh *deformation* following the skeleton deformation, any mesh *movement* should also follow the skeleton movement. Mesh deformation techniques should ensure that the deformed shape moves with the skeleton to give smooth results in animation making.

This provides a sound reason for including the translation term in the error energy function in (15). By doing so, we avoid singularity in the matrix K in (17) or (18), and the linear system has a unique solution; the mesh moves naturally with the skeleton.

6.4 Discussion

The main difference between our method and earlier skeleton-based deformation methods is that we use the skeleton motion to drive the transformation of *simplices*, rather than *vertices*. Thus, we make use of the *connectivity information* in the mesh directly, while they do not.

Another advantage is that our method is much simpler since *no* weight selection is needed, nor are any other arbitrary parameters.

Examples demonstrate that while our method is simple, it can nevertheless achieve high-quality results. Fig. 17 compares deformation results produced by our method and the SSD method; in the latter case, we used weights calculated by the inverse-square approach detailed in [45]. Artifacts are present where the leg meets the body in the SSD case, see Fig. 17b, but are absent using our method.

A further improvement of this paper over our earlier work [1] is that by including the translation vector in the optimization process, there is no need to fix an arbitrary vertex of the deformed mesh. This is also a key difference between this work and the other recent techniques that combine the skeleton and differential-based (or edge-based) methods [47], [48], [49]. By incorporating the translation vector, we keep the skin and the skeleton synchronized, which is very important when generating a long animation sequence.

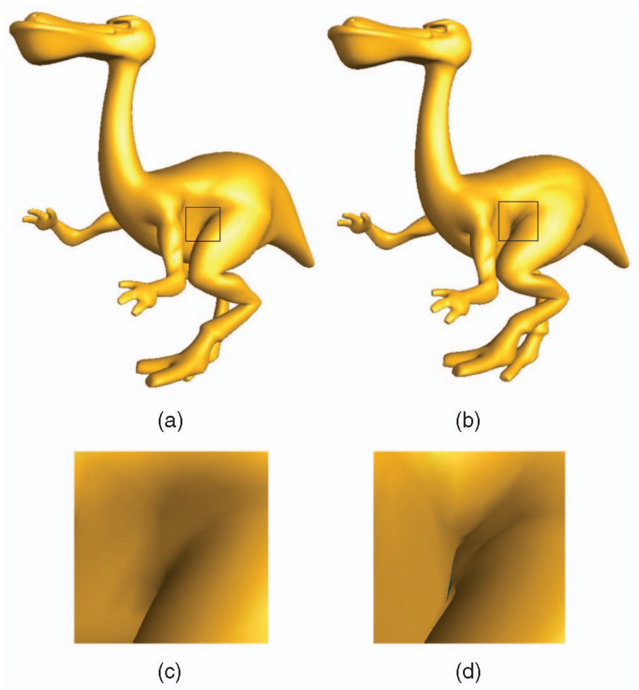


Fig. 17. Deformation using our method and SSD. (a) Our method. (b) SSD. (c) Details of our method. (d) Details of SSD.

7 Control by Lines

Sometimes, we only need to deform part of a model, while other parts remain more or less unchanged. In such cases, it is convenient to control the deformation just by moving a few lines, rather than having to define and manipulate the whole skeleton. Our method can easily be extended to do this. We can also extend our method to twist part of the mesh.

7.1 Deformation by Lines

To base the deformation on a few lines, we place lines that work in a similar way to bones into the object: certain triangles lie inside the control range of each line. We next determine which triangles are controlled by each line segment using the methods in Section 4. Clearly, some triangles may not have any corresponding control lines, since they may not be in any line's control domain. A simple approach to this problem is given as follows: For a triangle with an associated control line, $\widetilde{M}$ and $\widetilde{T}$ in (15) is set to the transformation matrix and translation vector of its control line, calculated using the method in Section 5.2. For any triangle without a control line, $\widetilde{M}$ is set to an identity matrix, and $\widetilde{T}$ is set to zero, which means that it tries to keep its original shape. We now solve (18).

However, using the above procedure directly may mean a line segment controls triangles over too large a part of the mesh. Two approaches can be used to avoid this problem, according to the animator's requirements. First, we can artificially decide that any triangle whose minimum effective distance with penalty is larger than δ has no

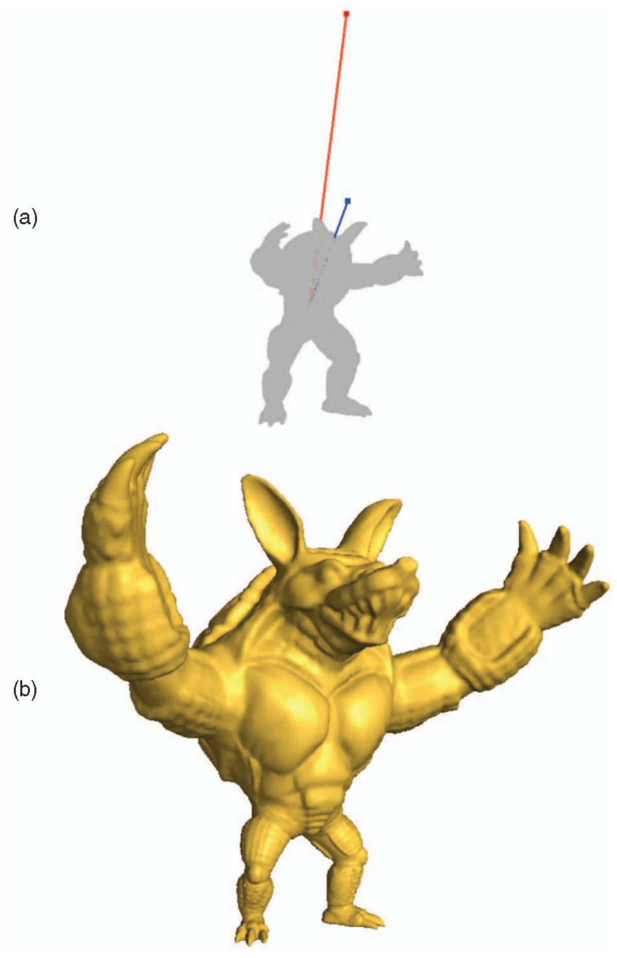


Fig. 18. (a) Armadillo model and control line. (b) Deformed Armadillo.

control line. Second, we may let the user directly select an effective domain for each line segment to select those triangles it should influence such as a tube centered on the control line, a bounding cuboid, or some other user-defined shape.

Fig. 18 shows the deformation of the Armadillo model controlled by a line segment. Blue and red lines identify the original and deformed control lines. The upper part of the Armadillo's body is rotated and enlarged 2.5 times.

7.2 Deformation Using a Twist Axis

Sometimes, we may wish to twist part of the model, for example an animal's neck. We can control such twists using a twist axis. Kho and Garland [51] uses line segments for similar purposes, although their method does not make use of the simplex connectivity information in the mesh.

The difference between twisting and the simpler rotation and scaling done earlier is that the transformation matrix for each bone includes not only a (constant) rotation and scaling but also a twist that varies linearly from zero at one end of the bone to a maximum value at the other. Thus, different triangles along the bone require different transformation equations.

Suppose AB is a twist axis, with twist angles specified to be 0 at A and γ at B. Parameterizing the t at I, the twist angle at J is $t\gamma$. We can compute the twist matrix at J using a process similar to the scaling process in Section 5.2. First, we translate

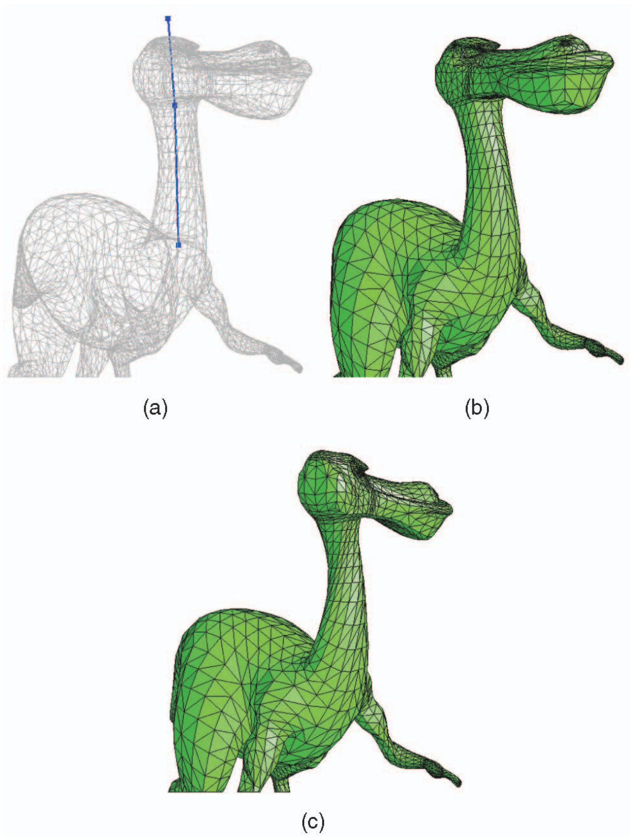


Fig. 19. Twisted Dinopet: (a) control lines, (b) original model, (c) twisted model.

AB until A coincides with the original point: the translation vector is $\widetilde{T}_{W1}$. Then, we use a twist transformation around AB, with twist angle $t\gamma$: the transformation matrix in this twist step is $\widetilde{W}$. Then, we translate AB back to its original place: the translation vector is $\widetilde{T}_{W2}$. The twist transformation matrix $\widetilde{W}$ can be calculated as in (14), replacing $\widetilde{R}$ by $\widetilde{W}$.The twist process can be written as

$$\widetilde{z} = \widetilde{W}(\widetilde{w} + \widetilde{T}_{W1}) + \widetilde{T}_{W2}. \tag{20}$$

Substituting (7) and (11) into (20), we get an overall transformation matrix and translation vector for J, which take into account the rotation, scaling, and twist. The transformation matrix can be written as

$$\widetilde{M} = \widetilde{W}\widetilde{S}\widetilde{R}, \tag{21}$$

while the translation vector can be written as

$$\widetilde{T} = \widetilde{W}(\widetilde{T}_S + \widetilde{T}_{W1}) + \widetilde{T}_{W2}. \tag{22}$$

As in the previous section, the ideal transformation matrix $\widetilde{M}$ for any simplex not controlled by a twist line is set to the identity, with ideal translation vector is zero.

Fig. 19 gives an example of twisting the neck of the Dinopet model by 90 degrees. A twist axis is used along the neck. However, the whole head needs to turn through the *same* constant angle. This is achieved by placing a second control line, which extends the first into the head, with a constant twist along its length equal to the twist at the top of the neck.

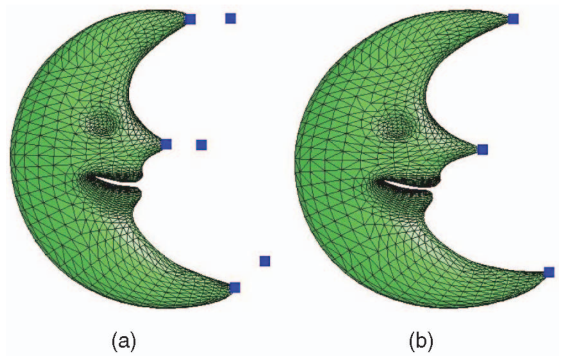


Fig. 20. (a) Original moon and vertex positions. (b) Deformed moon.

8 Conclusion and Future Work

We have presented an improved mesh deformation method, which combines the *skeleton-based* and *simplex transformation* approaches. We first determine the transformation for bones of the skeleton and then transfer each bone's transformation to those triangles it controls. The correspondence between simplices and bones is determined automatically. We use an optimization method to ensure connectivity between triangles controlled by different bones while keeping the mesh deformation as close as possible to the deformation of the skeleton. Our method can be used to deform a mesh using control lines and twist axes.

We may also control the deformation of a mesh by only moving a few vertices, rather than a skeleton or line segments. In this case, we simply set the transformation matrix $\widetilde{M}$ to the identity matrix and translation vector $\widetilde{T}$ to zero for *all* triangles. Fig. 20 shows deformation of a moon shape by choosing new positions for a few constrained vertices: the blue points identify the original and deformed positions of these constrained points. If large rotations or scaling exist, this simple approach does not work well since the identity matrix is far from the real transformation matrix. However, many other previous methods have given ways to modify local intrinsic attributes—see [10], [11], and [12]. These methods could be extended to modify the transformation matrix and translation vector to be used in conjunction with our vertex constraint deformation method. However, investigating such possibilities is outside the scope of this paper, and we intend to consider them in future.

Acknowledgments

The authors would like to thank Stanford University, Cyberware, and AIM@SHAPE for providing the 3D models and Peng Jiang, Gwenael Allard, Yi-Fei Zhang, and Yu Zang for their kind help. They also thank Craig Gotsman, Tao Ju, Jovan Popovic, and Robert Sumner for helpful discussions. They appreciate the suggestions of the anonymous reviewers. This work was supported by the National Basic Research Project of China (Project 2006CB303106), the National High Technology Research and Development Program of China (Project 2007AA01Z336), and the Natural Science Foundation of China (Project 60673004 and 60333010).

References

[1] H.-B. Yan, S.-M. Hu, and R. Martin, "Skeleton-Based Shape Deformation Using Simplex Transformations," *Proc. 24th Computer Graphics Int'l Conf. (CGI '06),* Advances in Computer Graphics, pp. 66-77, 2006.

[2] T.W. Sederberg and S.R. Parry, "Free-Form Deformation of Solid Geometric Models," *Computer Graphics (Proc. ACM SIGGRAPH '86),* vol. 20, no. 4, pp. 151-160, 1986.

[3] S. Coquillart, "Extended Free-Form Deformation: A Sculpturing Tool for 3D Geometric Modeling," *Computer Graphics (Proc. ACM SIGGRAPH '90,)* vol. 24, no. 4, pp. 187-196, 1990.

[4] F. Lazarus, S. Coquillart, and P. Jancène, "Axial Deformations: An Intuitive Deformation Technique," *Computer Aided Design,* vol. 26, no. 8, pp. 607-613, 1994.

[5] K. Singh and E. Fiume, "Wires: A Geometric Deformation Technique," *Proc. ACM SIGGRAPH '98,* pp. 405-414, 1998.

[6] D. Zorin, P. Schöder, and W. Sweldens, "Interactive Multiresolution Mesh Editing," *Proc. ACM SIGGRAPH '97,* pp. 249-268, 1997.

[7] L. Kobbelt, S. Campagna, J. Vorsatz, and H.-P. Seidel, "Interactive Multi-Resolution Modeling on Arbitrary Meshes," *Proc. ACM SIGGRAPH '98,* pp. 105-114, 1998.

[8] I. Guskov, W. Sweldens, and P. Schroder, "Multiresolution Signal Processing for Meshes," *Proc. SIGGRAPH '99,* pp. 325-334, 1999.

[9] M. Alexa, "Differential Coordinates for Local Mesh Morphing and Deformation," *The Visual Computer,* vol. 19, no. 2, pp. 105-114, 2003.

[10] Y. Lipman, O. Sorkine, D. Cohen-Or, D. Levin, C. Rössl, and H.-P. Seidel, "Differential Coordinates for Interactive Mesh Editing," *Proc. Shape Modeling Int'l,* pp. 181-190, 2004.

[11] O. Sorkine, Y. Lipman, D. Cohen-Or, M. Alexa, C. Rössl, and H.-P. Seidel, "Laplacian Surface Editing," *Proc. Eurographics/ACM SIGGRAPH Symp. Geometry Processing,* pp. 179-188, 2004.

[12] Y. Yu, K. Zhou, D. Xu, X. Shi, H. Bao, B. Guo, and H.-Y. Shum, "Mesh Editing with Poisson-Based Gradient Field Manipulation," *ACM Trans. Graphics (Proc. ACM SIGGRAPH '04,)* vol. 23, no. 3, pp. 644-651, 2004.

[13] K. Zhou, J. Huang, J. Snyder, X.-G. Liu, H.-J. Bao, B.-N. Guo, and H.-Y. Shum, "Large Mesh Deformation Using the Volumetric Graph Laplacian," *ACM Trans. Graphics (Proc. ACM SIGGRAPH '05),* vol. 24, no. 3, pp. 496-503, 2005.

[14] L. Shi, Y. Yu, N. Bell, and W.-W. Feng, "A Fast Multigrid Algorithm for Mesh Deformation," *ACM Trans. Graphics (Proc. ACM SIGGRAPH '06),* vol. 25, no. 3, pp. 1108-1117, 2006.

[15] J. Huang, X. Shi, X. Liu, K. Zhou, L.-Y. Wei, S.-H. Teng, H. Bao, B. Guo, and H.-Y. Shum, "Subspace Gradient Domain Mesh Deformation," *ACM Trans. Graphics (Proc. ACM SIGGRAPH '06),* vol. 25, no. 3, pp. 1126-1134, 2006.

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

[17] M. Alexa, D. Cohen-Or, and D. Levin, "As-Rigid-as-Possible Shape Interpolation," *Proc. ACM SIGGRAPH '00,* pp. 157-165, 2000.

[18] R.-W. Sumner and J. Popovic, "Deformation Transfer for Triangle Meshes," *ACM Trans. Graphics (Proc. ACM SIGGRAPH '04),* vol. 23, no. 3, pp. 399-405, 2004.

[19] R.-W. Sumner, M. Zwicker, C. Gotsman, and J. Popovic, "Mesh-Based Inverse Kinematics," *ACM Trans. Graphics (Proc. ACM SIGGRAPH '05),* vol. 24, no. 3, pp. 488-495, 2005.

[20] K.G. Der, R.W. Sumner, and J. Popović, "Inverse Kinematics for Reduced Deformable Models," *ACM Trans. Graphics (Proc. ACM SIGGRAPH '06),* vol. 25, no. 3, pp. 1174-1179, 2006.

[21] M. Botsch, R. Sumner, M. Pauly, and M. Gross, "Deformation Transfer for Detail-Preserving Surface Editing," *Proc. Vision, Modeling and Visualization '06,* pp. 357-364, 2006.

[22] Alias|Wavefront, *Learning Maya, Version 3.0.* Alias, 2000.

[23] Softimage, *Softimage 3D, Animating, User's Guide.* Softimage, 2002.

[24] J.-P. Lewis, M. Cordner, and N. Fong, "Pose Space Deformation: A Unified Approach to Shape Interpolation and Skeleton-Driven Deformation," *Proc. ACM SIGGRAPH '00,* pp. 165-172, 2000.

[25] N. Magnenat-Thalmann, R. Laperriere, and D. Thalmann, "Joint-Dependent Local Deformations for Hand Animation and Object Grasping," *Proc. Graphics Interface '88,* pp. 26-33, 1988.

[26] A. Mohr, L. Tokheim, and M. Gleicher, "Direct Manipulation of Interactive Character Skins," *Proc. Symp. Interactive 3D Graphics,* pp. 27-30, 2003.

[27] B. Allen, B. Curless, and Z. Popovic, "Articulated Body Deformation from Range Scan Data," *ACM Trans. Graphics (Proc. ACM SIGGRAPH '02),* vol. 21, no. 3, pp. 612-619, 2002.

[28] D.L. James and C.D. Twigg, "Skinning Mesh Animations," *ACM Trans. Graphics, (Proc. ACM SIGGRAPH '05),* vol. 24, no. 3, pp. 399-407, 2005.

[29] X.-H.C. Wang and C. Phillips, "Multi-Weight Enveloping: Least-Squares Approximation Techniques for Skin Animation," *Proc. ACM SIGGRAPH/Eurographics Symp. Computer Animation,* pp. 129-138, 2002.

[30] T. Rhee, J. Lewis, and U. Neumann, "Real-Time Weighted Pose-Space Deformation on the GPU," *Computer Graphics Forum,* vol. 25, no. 3, pp. 439-448, 2006.

[31] P.G. Kry, D.L. James, and D.K. Pai, "Eigenskin: Real Time Large Deformation Character Skinning in Hardware," *Proc. ACM SIGGRAPH '02,* pp. 253-260, 2002.

[32] L. Wade and R.E. Parent, "Automated Generation of Control Skeletons for Use in Animation," *The Visual Computer,* vol. 18, no. 2, pp. 97-110, 2002.

[33] P.-C. Liu, F.-C. Wu, W.-C. Ma, R.-H. Liang, and M. Ouhyoung, "Automatic Animation Skeleton Construction Using Repulsive Force Field," *Proc. Pacific Graphics,* pp. 409-413, 2003.

[34] A. Verroust and F. Lazarus, "Extracting Skeletal Curves from 3D Scattered Data," *The Visual Computer,* vol. 16, no. 1, pp. 15-25, 2000.

[35] S. Katz, G. Leifman, and A. Tal, "Mesh Segmentation Using Feature Point and Core Extraction," *The Visual Computer,* vol. 21, no. 8-10, pp. 649-658, 2005.

[36] T.-Y. Lee, Y.-S. Wang, and T.-G. Chen, "Segmenting a Deforming Mesh into Near-Rigid Components," *The Visual Computer,* vol. 22, no. 9-10, pp. 729-739, 2006.

[37] Y.-K. Lai, Q.-Y. Zhou, S.-M. Hu, and R.R. Martin, "Feature Sensitive Mesh Segmentation," *Proc. ACM Symp. Solid and Physical Modeling,* pp. 7-16, 2006.

[38] S. Berretti, A.D. Bimbo, and P. Pala, "Partitioning of 3D Meshes Using Reeb Graphs," *Proc. 18th Int'l Conf. Pattern Recognition,* pp. 19-22, 2006.

[39] S. Katz and A. Tal, "Hierarchical Mesh Decomposition Using Fuzzy Clustering and Cuts," *ACM Trans. Graphics (Proc. ACM SIGGRAPH '03),* vol. 22, no. 3, pp. 954-961, 2003.

[40] J.-M. Lien, J. Keyser, and N.M. Amato, "Simultaneous Shape Decomposition and Skeletonization," *Proc. ACM Solid and Physical Modeling Symp.,* pp. 219-228, 2005.

[41] N.D. Cornea, D. Silver, X. Yuan, and R. Balasubramanian, "Computing Hierarchical Curve-Skeletons of 3D Objects," *The Visual Computer,* vol. 21, no. 11, pp. 945-955, 2005.

[42] X.-T. Li, T.-W. Woon, T.-S. Tan, and Z.-Y. Huang, "Decomposing Polygon Meshes for Interactive Applications," *Proc. ACM Symp. Interactive 3D Graphics '01,* pp. 35-42, 2001.

[43] D. Reniers and A. Telea, "Skeleton-Based Hierarchical Shape Segmentation," *Proc. IEEE Int'l Conf. Shape Modeling and Applications,* pp. 179-188, 2007.

[44] N.D. Cornea, D. Silver, and P. Min, "Curve-Skeleton Properties, Applications, and Algorithms," *IEEE Trans. Visualization and Computer Graphics,* vol. 13, no. 3, pp. 530-548, May/June 2007.

[45] J. Bloomenthal, "Medial-Based Vertex Deformation," *Proc. ACM SIGGRAPH '02/Eurographics Symp. Computer Animation,* pp. 147-151, 2002.

[46] S. Yoshizawa, A. Belyaev, and H.-P. Seidel, "Skeleton-Based Variational Mesh Deformations," *Computer Graphics Forum,* vol. 26, no. 3, pp. 255-264, 2007.

[47] D. Anguelov, P. Srinivasan, D. Koller, S. Thrun, and J. Rodgers, "Scape: Shape Completion and Animation of People," *ACM Trans. Graphics (Proc. ACM SIGGRAPH '05),* vol. 24, no. 3, pp. 408-416, 2005.

[48] R.Y. Wang, K. Pulli, and J. Popović, "Real-Time Enveloping with Rotational Regression," *ACM Trans. Graphics, (Proc. ACM SIGGRAPH '07),* vol. 26, no. 3, pp. 73-1-73-9, 2007.

[49] O. Weber, O. Sorkine, Y. Lipman, and C. Gotsman, "Context-Aware Skeletal Shape Deformation," *Computer Graphics Forum (Proc. Eurographics '07),* vol. 26, no. 3, pp. 265-273, 2007.

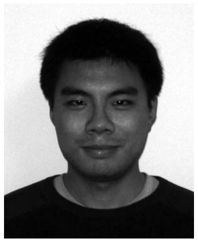
[50] H.-B. Yan, S.-M. Hu, and R.R. Martin, "Morphing Based on Strain Field Interpolation," *Computer Animation and Virtual Worlds,* vol. 15, nos. 3-4, pp. 443-452, 2004.

[51] Y. Kho and M. Garland, "Sketching Mesh Deformations," *Proc. Symp. Interactive 3D Graphics and Games,* pp. 147-154, 2005.

Han-Bing Yan received the PhD degree from the Department of computer science and technology, Tsinghua University, China, in 2006. He is currently with the National Computer Network Emergency Response Technical Team/Coordination Center of China and the Department of Computer Science and Technology, Tsinghua University, Beijing. His research interests include computer graphics, computer animation, computer network security, and information security.

Shi-Min Hu received the PhD degree from Zhejiang University in 1996. He is currently a chair professor of computer science at Tsinghua University. His research interests include digital geometry processing, video processing, rendering, computer animation, and computer-aided geometric design. He is on the editorial boards of *Computer Aided Design*. He is a member of the IEEE and the IEEE Computer Society.

Ralph R. Martin received the PhD degree from Cambridge University in 1983. He is currently a professor at Cardiff University. He has published more than 170 papers and 10 books, covering such topics as solid and surface modeling, intelligent sketch input, geometric reasoning, reverse engineering, and various aspects of computer graphics. He is on the editorial boards of *Computer Aided Design* and the *International Journal of Shape Modelling*.



Yong-Liang Yang received the bachelor's degree in computer science from Tsinghua University in 2004. He is currently working toward the PhD degree in the Department of Computer Science and Technology, Tsinghua University. His research interests include computer graphics and geometric modeling and processing.