Skeleton-Based Shape Deformation Using Simplex Transformations

Han-Bing Yan[1], Shi-Min Hu[1], and Ralph Martin[2]

[1] Dept. of Computer Science and Technology, Tsinghua University, P.R. China
yanhb02@mails.tsinghua.edu.cn,
shimin@tsinghua.edu.cn
http://cg.cs.tsinghua.edu.cn
[2] School of Computer Science, Cardiff University, U.K.
Ralph.Martin@cs.cardiff.ac.uk
http://ralph.cs.cf.ac.uk

Abstract. This paper presents a novel skeleton-based method for deforming meshes, based on an approximate skeleton. The major difference from previous skeleton-based methods is that they used the skeleton to control movement of vertices, whereas we use it to control the simplices defining the model. This allows errors, that occur near joints in other methods, to be spread over the whole mesh, giving smooth transitions near joints. Our method also needs no vertex weights defined on the bones, which can be tedious to choose in previous methods.

1 Introduction

Mesh deformation is widely used in computer animation and computer modeling. Many techniques have been developed to help artists deform body shapes for 2D and 3D characters, such as free-form deformation (FFD), differential methods, simplex transformation methods, and skeleton-based methods.

The latter use a 'skeleton', in which two or more 'bones' meet at each joint, to control shape deformation. This allows intuitive control, naturally describing the way in which many objects, e.g. animals, deform: the muscles and other tissues follow motions of the underlying bones. Such methods are usually controlled by an user-specified skeleton, rather than the exact medial axis. However, traditional skeleton-based methods are widely criticised for requiring a tedious process of weight selection to obtain satisfactory results. Seemingly, there is no criterion for weight selection which is universally applicable to *all* cases.

This paper presents a novel mesh deformation method which combines the skeleton-based method and the simplex transformation method, with two main differences from traditional skeleton-based methods. Firstly, we use the skeleton to drive the transformation of *simplices*, rather than vertices as in previous methods. Secondly, we avoid the use of *any* weights, yet our approach still gives high quality results.

Our approach can be applied to 2D and 3D meshes. Our inputs are the initial mesh, the initial skeleton—a set of straight line segments connected together at joints, and the deformed skeleton. The output is the deformed mesh.

The main steps of our method are as follows:

- Decide which bone of the skeleton controls each simplex.
- Find the transformation relating the initial and final position of each bone, and apply it to the simplices under its control.
- Use optimisation to stitch the simplices together while keeping each simplex transformation as close as possible to the value calculated above.

Our main contribution is to provide a skeleton-based deformation method which does not require a tedious weight adjustment process, yet which gives very good results. Our key new idea is to use the bones to control mesh simplices instead of mesh vertices, and hence to take advantage of the connectivity information between vertices. To achieve this, we also present a segmentation approach to determine the correspondence between mesh simplices and the bones.

2 Related Work

One of the best known methods for carrying out deformation, widely used in commercial software, is *FFD*. The classic FFD method [1] encloses a shape in an elastic control lattice, such as a Bézier volume, or a more general lattice [2], then deforms the volume by moving the control vertices: as a result, the shape inside is deformed.

Differential deformation methods have recently become popular [3, 4, 5, 6]. Laplacian coordinates [3] are used to represent surface detail as differences from the local mean. Poisson mesh methods [6] manipulate gradients of the mesh's coordinate functions and then reconstruct the surface using the Poisson equation.

Simplex transformation is another approach to deformation and morphing. The use of global transformations combined with matrix decomposition was proposed in [7] as a means to carry out morphing. This method was extended to local transformations by [8], in which meshes are stitched using an optimization method. Simplex transformation has also been used with surface triangle meshes to perform deformation learnt from existing examples [9, 10].

None of the above methods take into account the way in which shapes' features are naturally controlled. However, the shape and movement of an animal is determined by its skeleton, so the latter provides an intuitive approach to control the deformation of animal-like shapes. Such concepts are also referred to as *skinning*, *envelopes* or *skeletal subspace deformation* [11].

Existing skeleton-based algorithms define the final position of a point as a weighted sum over its initial position projected into n moving coordinate frames, corresponding to the n bones. Its position $\mathbf{p}'$ after deformation is:

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

where $\mathbf{p}$ is its initial position, M_k is a transformation matrix that transforms bone k from its initial position to its new position, and w_k is the weight of this point relative to bone k. Because each point is controlled by several bones, careful choice of weights w_k is needed to avoid self-intersections, especially near the joints, and to keep the surface smooth. Appropriate weight selection is an extremely tedious problem if done manually. Much research has focused on how to calculate appropriate weights [12, 13], or how to learn weights from examples [11, 14], but no single method works well in all cases [15]. As a result, [15] proposes that 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, To do so, this method calculates weights from a class of basic deformation shapes.

The underlying problem here is that each point is updated independently using Eqn. 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 information, but it is not directly used. We do so, to our advantage. By retaining skeleton-based control, we still have a natural and easily-understood approach. By using the connectivity information, we avoid the above weight adjustment problem and instead solve a linear equation to perform a similar task. This simpler approach still gives high quality results.

There has been much work on skeleton construction and segmentation, either independently, or doing both at once [16, 17, 18, 19]. We focus on how to segment the model from a given skeleton, as often artists wish to create the skeleton themselves. [19] proposed creating the skeleton and segmentation iteratively, but this changes the skeleton during iteration. [17] showed how to derive a segmentation from a given skeleton using space-sweeping method, but this does not work well if the skeleton is coarse. We give a new effective method to segment the model which considers both spatial distance, and the shortest path distance in the mesh, between each simplex and the bones.

We provide basic concepts concerning simplex transformations and skeletons in Section 3. We first apply our method to 2D triangle meshes in Section 4, then 3D triangle meshes in Section 5. We give conclusions in Section 6.

3 Simplex Transformations and Skeletons

Simplices are triangles in 2D and tetrahedra in 3D. Given two simplices S_1 and S_2 in some space, there exists a unique transformation that changes S_1 into S_2. In 2D, this can be written as: $v_i = Ru_i + T$, where the matrix R represents rotation and shape change information, T is a translation vector, u_i are the vertices of S_1, and v_i are the corresponding vertices of S_2. R and T can be calculated from the vertex coordinates of S_1 and S_2, by first finding R using $R = VU^{-1}$, where in 2D, $V = \begin{bmatrix} v_1 - v_3 \; v_2 - v_3 \end{bmatrix}$, $U = \begin{bmatrix} u_1 - u_3 \; u_2 - u_3 \end{bmatrix}$, and in 3D, $V = \begin{bmatrix} v_1 - v_4 \; v_2 - v_4 \; v_3 - v_4 \end{bmatrix}$, $U = \begin{bmatrix} u_1 - u_4 \; u_2 - u_4 \; u_3 - u_4 \end{bmatrix}$. Having found R, T can now be calculated.

The mathematical skeleton, or *medial axis*, is generally quite complex even for simple 3D shapes, and is sensitive to small perturbations of the shape boundary. It can also contain sheets rather than lines. For simplicity, most skeleton-based deformation methods use an approximate skeleton to control deformation, consisting of articulated straight lines or *bones*. As noted, often, artists prefer to create the skeleton by hand since this is easy to do interactively, and allows appropriate control. The skeleton can also be generated automatically [16, 17].

4 2D Triangle Mesh Deformation

We first consider the case of deforming a 2D triangle mesh in the plane.

4.1 Correspondence Between 2D Triangles and Bones

In our approach, each simplex is controlled by *one* bone, so we need to segment the model according to the given skeleton: deciding which bone controls each 2D triangle is the first step of our algorithm. Only after doing this can we decide how each triangle should deform. We determine the controlling bone as follows:

1. Calculate the minimum *effective distance with penalty* from the simplex to those bones for which it is *within range*.
2. Decide if the minimum *effective distance with penalty* is less than a threshold.
   (a) If so: the bone with the minimum *effective distance with penalty* controls this simplex.
   (b) Otherwise: calculate the *shortest path distance* from the simplex to those bones for which it is within range. The bone with the *shortest path distance* is the control bone.

We now explain the details. Normally, a skeleton lies within the volume defined by the mesh. However, we require that *free* ends of bones (i.e. ends not connected to other bones) must lie just *outside* the mesh, to ensure that each bone properly controls all the triangles in its control domain, as explained later.

Consider Fig. 1. AB, AC, and AD are three bones connected at the articulating joint A. We use *range lines* to define the border of each bones' control domain. The control domain for each bone determines which simplices that bone *may* control. If the centroid of a simplex lies within a given bone's control domain, the bone is a *candidate* for the control bone for that simplex; note that the

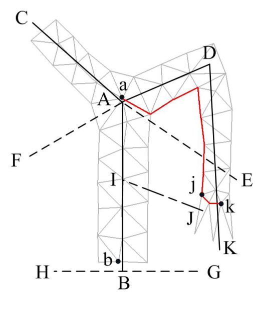

Fig. 1. Range lines

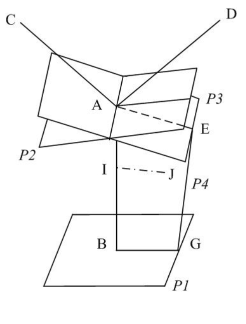

Fig. 2. Range planes

centroid of a given simplex may lie within the control domain of *several* bones. First, we construct *range lines* for each bone. For free ends of bones, like B, the range lines are *perpendicular rays*, like BG and BH. Where several bones meet, we determine the adjacent bones in both clockwise and anticlockwise directions, and draw *bisecting rays* between the current bone and the adjacent bones to give the range lines, like AE and AF. Each such range line divides the plane into 2 parts, one on the same side as the bone, and one on the opposite side. Given any point (or simplex), and a bone, if the point (or centroid of the simplex) lies on the same side as the bone for *all* of the bone's range lines, we say the point (or simplex) is within the *range* of this bone. We can now determine for which bones each simplex is within range.

If a given point J is within the range of some bone, we define its *effective distance* to the bone as follows (see Fig. 1). We find the corresponding point I on the bone AB as explained next; we call line IJ an *effective line* for bone AB. The direction of ray IJ is found by interpolating the normals of range lines AE and BG using Eqn. 2 where N denotes the normal to a line:

$$[(1 - t_I)N_{AE} + t_I N_{BG}] \times [(1 - t_I)v_A + t_I v_B - v_J] = 0. \qquad (2)$$

Here, t parameterises the bone from 0 at A to 1 at B; t_I is its value at I. Solving Eqn. 2 for t_I gives the position of point I. We call the distance from J to I the *effective distance* from point J to bone AB, or from the corresponding simplex if J is its centroid.

Deciding which bone has minimum effective distance from a simplex is not by itself sufficient to decide which bone should control a simplex. For example, if a man stands with hands by his sides, a point on his waist may have a smaller effective distance to a hand bone than to any spine bone, but clearly waist points should be controlled by spine bones. We overcome this problem by using a penalty δ, equal to twice the overall mesh size. We determine how many outer edges of the triangulation the effective line IJ intersects, by constructing a binary tree for the whole mesh using ideas from [20]. We now define the *effective distance with penalty* d_{effpen} as $d_{\text{effpen}} = d_{\text{eff}} + n\delta$, where n is the number of intersections, and d_{eff} is the *effective distance*.

Suppose point J is within the range of several bones, each giving a value for d_{effpen}. If d_{effpen} from J to *some* bone is smaller than δ, i.e. IJ does not intersect the boundary of the mesh, we say point J can be *seen* from the bone. If J can be *seen* by at least one bone, we select the bone that has the minimum d_{effpen} as the controller of point J, and hence the corresponding simplex.

If the minimum d_{effpen} of J is larger than δ, this means IJ intersects the mesh boundary. To determine the controller of point J, we calculate the shortest path from J to each join, using Dijkstra's algorithm across the mesh, after first finding the nearest mesh vertex to J and to the end of each bone. (This distance is not sensitive to the exact connectivity of the mesh). We select the bone with the minimum shortest path distance, amongst the bones that the point is within the range of, as the controller of the point. The red lines in Fig. 1 show the shortest paths from J to bone AB and to bone DK. The controller of point J is bone DK, since the shortest path distance from J to bone DK is smaller.

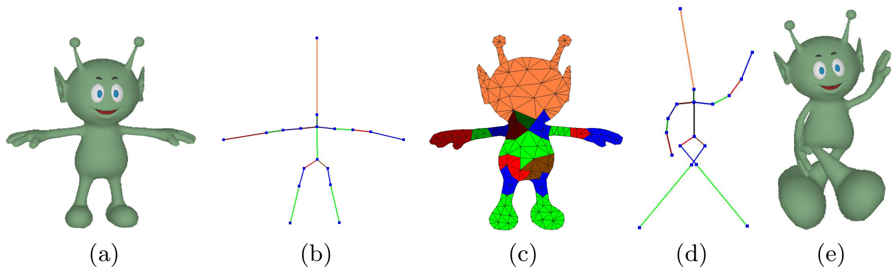

Fig. 3. (a) 2D Cartoon character (b) Skeleton (c) Skeleton control domain (d) Deformed 2D skeleton, (e) Deformed cartoon character

Fig. 3(a) shows a 2D cartoon character, Fig. 3(b) shows an appropriate 2D skeleton and Fig. 3(c) shows, using corresponding colors, which bone controls each triangle, as determined by the method above.

4.2 Transformation for 2D Bones

Given the initial skeleton, and user-determined deformed skeleton, the transformation matrix for each bone can be calculated. Fig. 4 shows a bone at A_1B_1 in the initial skeleton, and at A_2B_2 in the deformed skeleton. Normally the bone transformation matrix does not involve scaling, but later we show how to take it into account if required.

We translate A_1B_1 so that A_1 is at the origin, the translation vector being T_1. A_1B_1 is then rotated around the origin to lie in the same direction as A_2B_2, θ being the anticlockwise angle of rotation. We then translate A_1B_1 so that A_1 coincides with A_2, the translation vector being T_2. This transformation process can be expressed as $v^{'} = R'(u' + T_1) + T_2$, where u' is a point on the bone before deformation, and v' is the corresponding point afterwards. The transformation matrix of this equation is R', given as usual by

$$R' = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}. \tag{3}$$

If scaling is desired, we translate A_1 to the origin as before, then rotate A_1B_1 to lie along the x axis using a rotation R_1. we then scale A_1B_1 until it has the same length as A_2B_2, using a scaling matrix S:

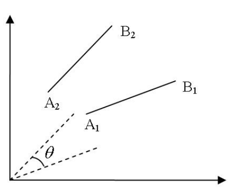

Fig. 4. 2D Bone transformation

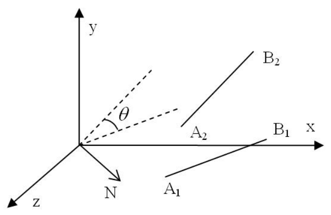

Fig. 5. 3D Bone transformation

$$S = \begin{bmatrix} \alpha & 0 \\ 0 & \beta \end{bmatrix}. \tag{4}$$

Here, α represents the scaling along the direction of the bone, determined by the relative lengths before and after deformation, whereas β represents scaling in the direction perpendicular to the bone. The animator will usually choose this to be 1.0, or the same as α, according to his needs. Finally we rotate A_1B_1 into the same orientation as A_2B_2, using a matrix R_2 and translate A_1B_1 until A_1 coincides with A_2 as before. Overall, we can write $v' = R_2SR_1(u' + T_1) + T_2$ where the transformation matrix in this step is given by $S' = R_2SR_1$.

We can write the overall transformation matrix between bone A_1B_1 and A_2B_2 as the combination of a rotation part and a scaling part:

$$M' = S'R'. \tag{5}$$

4.3 2D Triangle Mesh Deformation

If every triangle were to transform rigidly in the same way as its controlling bone, gaps would arise between the triangles controlled by adjacent bones, causing tears in the object. We must enforce vertex consistency requirements to prevent this. We do so using an optimization method, while trying to keep each simplex transformation as close as possible to that of its control bone. For simplicity, as in previous work on simplex transformations [8, 9], we only take into account the non-translation part of the transformation, and in practice, doing so provides good results for the deformed shape. An error function is used to represent the difference between the *actual* simplex deformation and the deformation determined by the control bone:

$$E = \sum_{i=1}^{n} A_i \|M_i - M_i'\|_F^2, \tag{6}$$

where F is the Frobenius norm, A_i is the area of the i^{th} triangle, n is the number of simplices in the mesh, M_i is the *actual* transformation matrix for the i^{th} triangle, given by Section 3 and M_i' is the ideal transformation matrix of this triangle, given in Section 4.2. 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, giving

$$K'X' = d'. \tag{7}$$

This set of equations can be separated into 2 independent groups corresponding to the x and y coordinates of the deformed mesh. Furthermore, the coefficient matrix for each group is the same, providing a more efficient solution than by treating them as a single system: $KX = d_x$, $KY = d_y$. Here X and Y are the x and y coordinate vectors of the deformed mesh, of size m, where m is the number of vertices in the mesh. K is a sparse $m \times m$ matrix, and d_x and d_y are vectors with dimension m. Generally, m is small enough that LU-decomposition provides an efficient solution method. In order to ensure a unique solution, at least one vertex position should be fixed in advance.

Fig. 3(d) shows a deformed skeleton and Fig. 3(e) the resulting deformed mesh for the cartoon character in Fig. 3(a). The character's legs are scaled using a factor of 2 both along and perpendicular to the the bone, while other parts are unscaled. The corresponding mesh has 251 vertices, and 0.12s were required to calculate the result on a 2.4Ghz Pentium 4 machine.

Local self-intersection seldom happens in our method, because the errors near the joints, which often arise in traditional skeleton-based methods, are spread from the joints to the neighboring domain by our optimization method.

5 3D Triangle Mesh Deformation

The method used for a 2D triangle mesh can also be extended to a 3D volume tetrahedron mesh, but in practice *surface* triangle mesh models are far more widely used. Furthermore, the latter have far fewer elements than tetrahedron models and thus require much lower processing times.

5.1 Correspondence Between 3D Triangles and Bones

As in 2D, we use the minimum *effective distance with penalty* and the minimum *shortest path distance* to decide the controlling bone for each triangle.

We now create range *planes* for bones in 3D, instead of range *lines* in 2D. In Fig. 2, AB, AC, AD are three bones connected at joint A. At free ends of bones, such as B, we create a range plane like $P1$, through B, and perpendicular to bone AB. At other ends of bones, such as A, we create a bisection range plane like planes $P2$ and $P3$ corresponding to each other bone meeting at this joint. Each bone may now have a varying number of range planes, unlike the 2D case where each bone has exactly 4 range lines.

Again we calculate the *effective line* from point J to bone AB. We create a plane $P4$ that passes both through bone AB and point J. This plane intersects the range planes in many rays. At each end of the bone, we select the nearest ray to point J as the *range line*: these are AE and BG in the example. Thus, at each joint of the bone we now have one range line. We now interpolate the *effective line* from the range lines as before.

Having found the effective line IJ, we can calculate the *effective distance with penalty* between the triangle S and the bone as in 2D. The intersection count n in 3D in computing the effective distance with penalty does not include the current triangle itself. If the minimum d_{effpen} of J is smaller than δ, we can decide the controlling bone for this triangle directly by selecting the bone with minimum d_{effpen}. If the minimum d_{effpen} of a triangle is larger than δ, we calculate the shortest path distance across the mesh from the triangle to those bones for which it is within range. The bone with minimum shortest path distance to the triangle is selected as its controller.

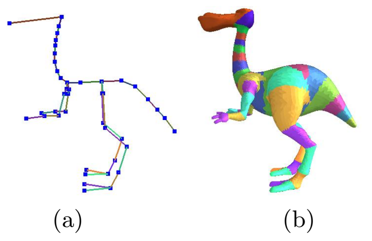

Fig. 6. Skeleton (a) and control domain (b) of Dinopet

Fig. 6(a) shows a skeleton for the Dinopet, and Fig. 6(b) shows the control domain of each bone.

5.2 Transformation for 3D Bones

We now consider how to calculate the transformation matrix for bones in 3D. In Fig. 5, 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 coincides with the origin. We then create a unit vector N based at the origin, perpendicular to 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 R' (ignoring the translation) can be calculated in a similar way to the 2D case and is found to be:

$$R = \begin{bmatrix} a^2+(b^2+c^2)cos\theta & ab(1-cos\theta)+csin\theta & ac(1-cos\theta)-bsin\theta \\ ab(1-cos\theta)-csin\theta & b^2+(a^2+c^2)cos\theta & bc(1-cos\theta)+absin\theta \\ ac(1-cos\theta)+bsin\theta & bc(1-cos\theta)-absin\theta & c^2+(a^2+b^2)cos\theta \end{bmatrix} \quad (8)$$

where $N=(a,b,c)$. If scaling is also required, we can determine the scale matrix S as in Section 4.2; the transformation matrix has the same form as Eqn. 5.

5.3 3D Triangle Mesh Deformation

The 3D triangle mesh case is very different from the 2D triangle mesh case, because a triangle is not a simplex in 3D, nor is there a unique transformation matrix for changing one triangle into another. [9] gave a clever way of extending simplex transformation methods to a 3D triangle mesh by constructing a tetrahedron for each triangle. We follow these ideas, except that we put the new vertex above the centroid of the triangle rather than over one of its vertices.

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, v_3 be the vertices of a triangle on the initial mesh. The fourth vertex is placed at

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

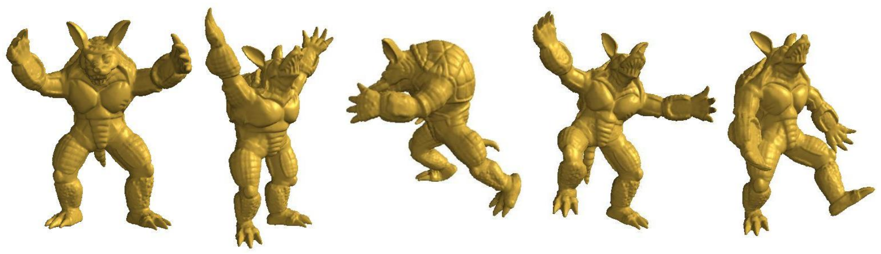

Fig. 7. Armadillo model

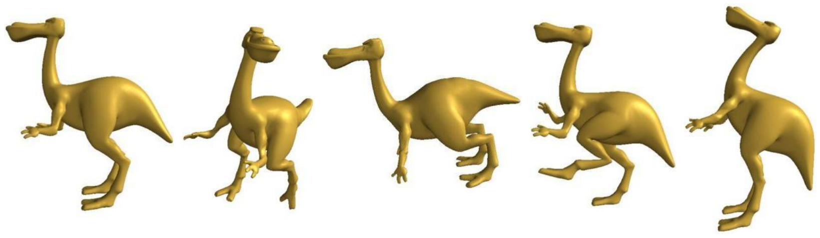

Fig. 8. Dinopet model

Table 1. Statics and timing

	Armadillo	Dinopet
Mesh Vertices	50852	13324
Segmentation time (seconds)	22.26	3.12
Deformation time (seconds)	68.63	7.83

Note that the above equation is only used to calculate v_4 in the initial mesh. v_4 in the deformed mesh are determined by the optimisation process.

The 3D triangle mesh is now deformed using the same process as for the 2D triangle mesh in Section 4.3; Eqn. 7 in 3D separates into 3 independent groups: $KX = d_x$, $KY = d_y$, $KZ = d_z$. The dimension of the vectors in Eqn. 7 is now $m + k$ for a mesh with m vertices and k faces. We use the conjugate gradient method to efficiently solve these large sparse linear equations.

Figures 7–8 illustrate 3D deformation results obtained using our technique. The first model in each Figure is the original model; others are deformed results by our method. All results were calculated on a 2.4Ghz Pentium 4 machine. Table 1 shows the timings of 3D models presented in this paper.

6 Conclusions

We have presented a novel 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 matrix to those simplices it controls. The correspondence between simplices and bones is determined automatically. We use an optimization method to eliminate gaps between triangles controlled by different bones, while keeping the mesh deformation as close as possible to the deformation of the skeleton.

The main advantage over earlier skeleton-based methods is that we directly use the connectivity information in the mesh while they do not. As a result, our method is much simpler since no weight selection nor any arbitrary parameters are needed, yet we can achieve high quality results.

We currently only take into account the non-translation part of the transformation. Although this provides good shape results, we need to arbitrarily fix one vertex to decide the final position of the deformed model. It would be more useful to make the deformed mesh automatically follow the deformed skeleton, and we are investigating including the translation in the Equation system as a way of doing this.

Our method can be easily adapted to control deformation by moving a few chosen line segments or vertices embedded in the object, rather than a skeleton. It can also be extended to twist part of the mesh if required, by defining twist axes. Space precludes demonstration of these capabilities.

Acknowledgements

We would like to thank Gwenael Allard, Shuai Xin and Yu Zang for kind help, and C. Gotsman, T. Ju, R. Sumner and J. Popovic for helpful discussions. This work was partially supported by the Natural Science Foundation of China (Projects 60225016, 60333010, 60321002) and the National Basic Research Project of China (Project 2002CB312101).

References

1. Sederberg, T., Parry, S.: Free-from deformation of solid geometric models. Computer Graphics (Proc. SIGGRAPH1986) **20**(4) (1986) 151–160
2. Coquillart, S.: Extended free-form deformation: A sculpturing tool for 3d geometric modeling. Computer Graphics (Proc. SIGGRAPH1990) **24**(4) (1990) 187–196
3. Alexa, M.: Differential coordinates for local mesh morphing and deformation. The Visual Computer **19**(2) (2003) 105–114
4. Lipman, Y., Sorkine, O., Cohen-Or, D., Levin, D., Rössl, C., Seidel, H.P.: Differential coordinates for interactive mesh editing. In: Proceedings of Shape Modeling International, IEEE Computer Society Press (2004) 181–190
5. Sorkine, O., Lipman, Y., Cohen-Or, D., Alexa, M., Rössl, C., Seidel, H.P.: Laplacian surface editing. In: Proceedings of the Eurographics/ACM SIGGRAPH symposium on Geometry processing, Eurographics Association (2004) 179–188
6. Yu, Y., Zhou, K., Xu, D., Shi, X., Bao, H., GUO, B., Shum, H.Y.: Mesh editing with poisson-based gradient field manipulation. ACM Trans. Graphics (Proc. SIGGRAPH2004) **23**(3) (2004) 644–651

7. Shoemake, K., Duff, T.: Matrix animation and polar decomposition. In: Proc. Conference on Graphics Interface '92. (1992) 258–264
8. Alexa, M., Cohen-Or, D., Levin, D.: As-rigid-as-possible shape interpolation. In: Proc. SIGGRAPH2000. (2000) 157–165
9. Sumner, R.W., Popovic, J.: Deformation transfer for triangle meshes. ACM Trans. Graphics (Proc. SIGGRAPH2004) **23**(3) (2004) 399–405
10. Sumner, R.W., Zwicker, M., Gotsman, C., Popovic, J.: Mesh-based inverse kinematics. ACM Trans. Graphics (Proc. SIGGRAPH2005) **24**(3) (2005) 488–495
11. Lewis, J., Cordner, M., Fong, N.: Pose space deformation: a unified approach to shape interpolation and skeleton-driven deformation. In: Proc. SIGGRAPH2000. (2000) 165–172
12. Bloomenthal, J.: Medial-based vertex deformation. In: Proc. 2002 ACM SIGGRAPH/Eurographics Symp. Computer Animation. (2002) 147–151
13. Mohr, A., Tokheim, L., Gleicher, M.: Direct manipulation of interactive character skins. In: Proc. 2003 Symp. Interactive 3D Graphics. (2003) 27–30
14. Allen, B., Curless, B., Popovic, Z.: Articulated body deformation from range scan data. ACM Trans. Graphics (Proc. SIGGRAPH2002) **23**(3) (2002) 612–619
15. Wang, X.C., Phillips, C.: Multi-weight enveloping: least-squares approximation techniques for skin animation. In: Proc. 2002 ACM SIGGRAPH/Eurographics Symp. Computer Animation, New York, NY, USA, ACM Press (2002) 129–138
16. Verroust, A., Lazarus, F.: Extracting skeletal curves from 3D scattered data. The Visual Computer **16**(1) (2000) 15–25
17. Li, X., Woon, T.W., Tan, T.S., Huang, Z.: Decomposing polygon meshes for interactive applications. In: ACM Symposium on Interactive 3D Graphics 2001. (2005) 35–42
18. Katz, S., Tal, A.: Hierarchical mesh decomposition using fuzzy clustering and cuts. ACM Transactions on Graphics **22**(3) (2003) 954–961
19. Lien, J.M., Amato, N.M.: Simultaneous Shape Decomposition and Skeletonization. Technical Report, TR05-015, Parasol Laboratory, Department of Computer Science, Texas A&M University (2005)
20. de Berg, M., van Kreveld, M., Overmars, M., Schwarzkopf, O.: Computational geometry : algorithms and applications. Springer (1997)