Visual Comput (2006) 22: 653–660
DOI 10.1007/s00371-006-0054-y

ORIGINAL ARTICLE

Yanlin Weng
Weiwei Xu
Yanchen Wu
Kun Zhou
Baining Guo

2D shape deformation using nonlinear least squares optimization*

Published online: 24 August 2006

W. Xu · K. Zhou (✉) · B. Guo
Microsoft Research Asia
{wwxu,kunzhou,bainguo}@microsoft.com

Y. Weng
University of Wisconsin – Milwaukee, USA
weng@uwm.edu

Y. Wu*
Zhejiang University
raincoat.zju@hotmail.com

Abstract This paper presents a novel 2D shape deformation algorithm based on nonlinear least squares optimization. The algorithm aims to preserve two local shape properties: the Laplacian coordinates of the boundary curve and the local area of the shape interior, which are together represented in a non-quadratic energy function. An iterative Gauss–Newton method is used to minimize this nonlinear energy function. The result is an interactive shape deformation system that can achieve physically plausible results that are difficult to achieve with previous linear least squares methods. In addition to this algorithm that preserves local shape properties, we also introduce a scheme to preserve the global area of the shape, which is useful for deforming incompressible objects.

Keywords Object manipulation · Image editing · Character animation · Area preservation

1 Introduction

2D shape deformation is a useful tool in various applications, e.g. real-time live performance and enriching graphical user interfaces. A good shape deformation system aims to produce visually pleasing results with simple operations and to provide interactive feedback to users. Many techniques to achieve a balance between these two objectives have been proposed.

Free-form deformation (FFD) [16] and skeleton-based techniques [9] are widely used methods in commercial software today. They run fast; however, setting FFD domains and skeleton configurations is tedious. Furthermore, it is laborious to manipulate many control points in FFD. Physically-based simulations [5] can achieve pleasing results, but with very low convergence.

*This work was done while Yanchen Wu was an intern at Microsoft Research Asia.

Recently, Igarashi et al. [7] presented an interactive system that allows the user to deform a 2D shape by manipulating a few points. The shape is represented by a triangle mesh and the user moves several vertices of the mesh as constrained handles. The system then computes the positions of the remaining free vertices by minimizing the distortion of each triangle. To make the problem linear, Igarashi et al. present a two-step closed-form algorithm: the first step to compute the rotation and the second step to compute the scale. This divides the problem into two least-squares minimization problems that can be solved quickly and stably. As the authors admitted, the two-step algorithm is merely a practical approximation to achieve interactive performance and may cause implausible results in some cases due to its linear nature.

In this paper, we present a novel 2D shape deformation algorithm based on nonlinear least squares optimization. The algorithm aims to preserve two geometric properties of 2D shapes: the Laplacian coordinates of the boundary

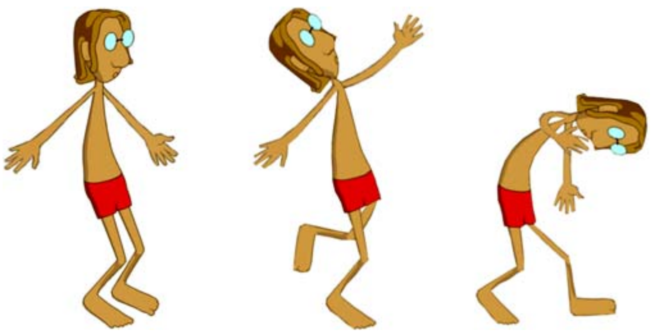


Fig. 1. 2D deformation of a cartoon character. Left: the original shape. Middle and right: the deformation results generated by our algorithm

curve of the shape and local areas inside the shape, which are together represented in a non-quadratic energy function. Instead of linearizing these nonlinear properties, we cast the problem as an nonlinear least squares minimization and solve it using an iterative method. The resulting system is able to achieve physically plausible deformation results and runs interactively. Besides preserving local shape properties, we also introduce a scheme to preserve the global area of the shape, which is useful for deforming incompressible objects.

1.1 Related work

Much previous work has been done on shape deformation, and we discuss here only those works most related to ours.

The best known method for shape deformation may be free form deformation (FFD) [11, 14, 16]. In FFD, a shape is embedded in a lattice and then deformed by moving the control points of the lattice. While FFD is simple and easy to use, it does not take into account the natural way in which shapes' features are controlled. For example, many animals have a skeleton. Skeleton-based deformation [9] provides an intuitive approach to control deformation of animal-like shapes. Skeleton-based algorithms define the position of a point as a weighted linear combination of the initial state of the point projected into several moving coordinate frames, corresponding to the bones, which is usually specified manually. Appropriate weight selection is a painful process.

To achieve physically plausible deformation, physically-based simulations can be employed [3, 5, 8]. Among these methods, the most popular is mass-spring models [5]. However, it is too slow to converge and needs careful tuning of various parameters. Finite-element methods [3] provide a more physically accurate simulation at the expense of lengthy computation. Therefore, they are inappropriate for interactive deformation applications. The ArtDefo system [8] can run interactively, but is limited to small deformations.

Gradient domain techniques [1, 7, 10, 18–20] cast deformation as an energy minimization problem. The energy function contains both a term for a detail-preserving constraint and a term for a position constraint. The detail-preserving constraint is nonlinear because it involves both the differentials for local details and the local transformations, which are position-dependent. For computational efficiency, existing techniques convert this nonlinear constraint into a linear one by using various approximations including local linearization of transformation [18], transformation interpolation from handles [10, 19, 20] and the decomposition of rotation and scaling computation [7]. The price for employing these least squares minimization schemes is suboptimal deformation results.

Our algorithm can be viewed as a variant of recent nonlinear mesh deformation methods [2, 6, 17]. All these methods try to minimize a nonlinear energy function representing local properties of the surface. Instead of a 3D mesh, our algorithm deals only with 2D shapes. Therefore, the local properties that we are trying to preserve are quite different from those of a 3D surface.

2 Overview

The input of our algorithm is a 2D shape (see Fig. 2(a)), with the boundary represented as a simple closed polygon. The shape can be represented either by vector graphics or a bitmap image. For bitmap images, we manually remove backgrounds and apply automatic silhouette tracing using the marching squares algorithm to get the boundary polygons. Our algorithm automatically inserts a set of points into the interior region of the shape and generates a graph by connecting the vertices of the boundary polygon and the inside points (see Fig. 2(b)). Then the user can drag the points to deform the shape.

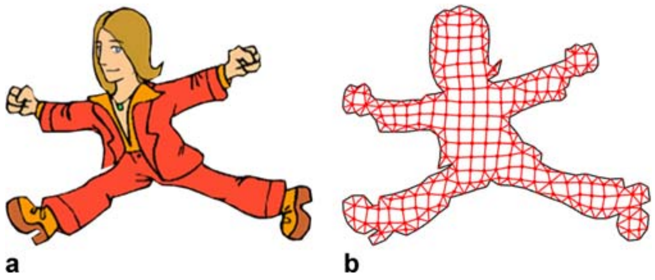


Fig. 2. 2D shape and its interior graph

The algorithm aims to preserve two local properties: the Laplacian coordinates of the boundary curve and the local area inside the shape. The Laplacian coordinates represent the local details of the shape boundary and are widely used in 3D mesh deformation methods [10, 18, 20]. While preserving Laplacian coordinates often produces

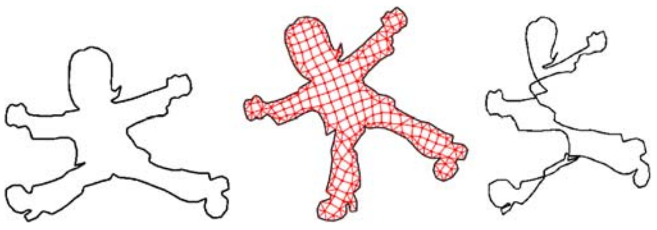


Fig. 3. Deformation results with and without local area preservation. Left: original shape. Middle: deformation result preserving both Laplacian coordinates and local area. Right: deformation result preserving Laplacian coordinates only

good deformation results for 3D meshes, it is not enough to produce visually pleasing deformation results for 2D shapes (see Fig. 3). Therefore, we also try to preserve the local areas inside the shape. To achieve this goal, we build a graph and introduce two new local properties for the graph: the relative position (mean value coordinates) of each interior point with respect to its neighbors and the length of each edge. To control a deformation, the user inputs the deformed positions for a subset of the graph points. The deformed positions of all graph points are then obtained by minimizing an energy function that consists of four parts: Laplacian coordinate preserving, mean value coordinates preserving, edge length preserving and position constraints (see details in Sect. 3).

To build the interior graph, one can generate a triangulated mesh inside the boundary polygon as in [7]. We instead adopt an easier approach similar to the volumetric graph construction in [20]. It consists of four steps (see Figure 6 in [20]). Firstly, we construct an inner polygon for the boundary polygon by offsetting each vertex a distance in the direction opposite its normal. Secondly, we embed the two polygons in a lattice, removing lattice nodes outside the inner polygon. Thirdly, we build edge connections among the two polygons and lattice nodes. Finally, we simplify the graph using edge collapse and smooth the graph.

Now we have a 2D graph (V, E), where V is the set of n vertices in the graph, and E is the set of edges. V includes two subsets: V_p, which contains m vertices of a polygon, and V_g, which contains $(n-m)$ interior points. Similarly, the edge set E can be divided into two sets: E_p, which contains polygon edges, and E_g, which represents the remaining edges in the graph.

The remainder of this paper is organized as follows. The following section explains the three local properties in detail. In Sect. 4, we combine all the local properties and present an iterative solver to compute the deformation results efficiently. Section 5 describes how to preserve the global area in our algorithm, which is useful for deforming incompressible objects. Experimental results are shown in Sect. 6, and the paper concludes with a discussion of future work in Sect. 7.

3 Preservation of local properties

This section describes the three local properties: Laplacian coordinates, mean-valued coordinates and edge length. Laplacian coordinates represents the local details of the boundary polygon. Mean-valued coordinates and edge length are used to achieve local area preservation.

3.1 Curve Laplacian coordinates

A curve Laplacian is defined for each point in V_p and it is analogous to the Laplacian on 3D meshes. Specifically, the curve Laplacian coordinate δ_i of point v_i is computed as the difference between v_i and the average of its neighbors on the curve:

$$\delta_i = \mathcal{L}_p(v_i) = v_i - (v_{i-1} + v_{i+1})/2,$$

where v_{i-1} and v_{i+1} are the points adjacent to v_i on the curve; and $\mathcal{L}_p$ is called the Laplace operator of the curve.

To preserve the Laplacian coordinates during deformation, we try to minimize the following energy function:

$$\sum_{v_i \in V_p} \|\mathcal{L}_p(v_i) - \delta_i\|^2,$$

which is equivalent to the matrix form:

$$\|\boldsymbol{L}_p \boldsymbol{V}_p - \delta(\boldsymbol{V}_p)\|^2, \tag{1}$$

where $\boldsymbol{V}_p$ is the point positions of the boundary polygon and $\boldsymbol{L}_p$ is an $m \times m$ matrix, called the Laplace matrix; δ is the vector of Laplacian coordinates. Note that we view δ as a general function of the point positions $\boldsymbol{V}_p$ instead of a linear function of $\boldsymbol{V}_p$ as in [18].

To make the description clear in the following, we expand $\boldsymbol{L}_p$ to an $m \times n$ matrix $\boldsymbol{L}$ by adding zero elements. Then Eq. 1 can be rewritten as:

$$\|\boldsymbol{L}\boldsymbol{V} - \delta(\boldsymbol{V})\|^2. \tag{2}$$

3.2 Mean value coordinates

We want to maintain the relative position of each point v_i in V_g, with respect to its neighboring points during deformation. To do this, we first compute its mean value coordinates [4] in the polygon formed by its neighboring points:

$$w_{i,j} = \frac{\tan(\alpha_j/2) + \tan(\alpha_{j+1}/2)}{|v_i - v_j|},$$

where α_j is the angle formed by the vector $v_j - v_i$ and $v_{j+1} - v_i$. Normalizing each weight function $w_{i,j}$ by the sum of all weight functions yields the mean value coordinates of v_i with respect to its neighboring points.

According to the property of mean value coordinates, we have:

$$v_i - \sum_{(i,j)\in E} w_{i,j} * v_j = 0, \quad \text{for } v_i \in V_g,$$

which can also be represented in matrix form:

$$\boldsymbol{M}_g \boldsymbol{V}_g = 0,$$

where $\boldsymbol{M}_g$ is a $(n-m)\times(n-m)$ matrix. Similar to $\boldsymbol{L}_p$, $\boldsymbol{M}_g$ can be expanded to an $(n-m)\times n$ matrix $\boldsymbol{M}$ by adding zero elements.

To preserve the mean value coordinates during deformation, we minimize the following energy function:

$$\|\boldsymbol{MV}\|^2. \tag{3}$$

3.3 Edge lengths

Note that mean value coordinates are invariant to scaling. Preserving mean value coordinates is not enough to preserve the local areas inside the shape. Therefore, we further try to preserve edge lengths during deformation.

We penalize the edge length changes for all edges in E_g using the following energy equation:

$$\sum_{(i,j)\in E_g} \|(v_i - v_j) - e(v_i, v_j)\|^2, \tag{4}$$

where $e(v_i, v_j) = \frac{\widetilde{l_{i,j}}}{l_{i,j}}(v_i - v_j)$; $l_{i,j}$ is the current length of edge (i, j) and $\widetilde{l_{i,j}}$ is the original length before deformation.

Note that the energy associated with each edge is computed in vector form such that the whole energy in Eq. 4 can be represented in the matrix form:

$$\|\boldsymbol{HV} - e(\boldsymbol{V})\|^2, \tag{5}$$

where $\boldsymbol{H}$ is a $|E_g|\times n$ matrix.

4 Shape deformation using nonlinear least squares optimization

4.1 Deformation energy

To control a deformation, the user inputs the deformed positions for a subset S of the graph points. This information is used to compute the deformed positions of all graph points by minimizing the following sum of all energy terms:

$$\|\boldsymbol{LV} - \delta(\boldsymbol{V})\|^2 + \|\boldsymbol{MV}\|^2 + \|\boldsymbol{HV} - e(\boldsymbol{V})\|^2 + \|\boldsymbol{CV} - \boldsymbol{U}\|^2, \tag{6}$$

where $\|\boldsymbol{CV} - \boldsymbol{U}\|^2$ represents the position constraints specified by the user; $\boldsymbol{C}$ is a $|S|\times n$ matrix and $\boldsymbol{U}$ is a vector of dimension $|S|$ representing the target positions specified by the user. To balance these objectives, we also allow the user to specify a weighting parameter for each energy term.

The above energy minimization problem can be reformulated as the following:

$$\min_{\boldsymbol{V}} \|\boldsymbol{AV} - \boldsymbol{b}(\boldsymbol{V})\|^2, \tag{7}$$

where:

$$\boldsymbol{A} = \begin{pmatrix} \boldsymbol{L} \\ \boldsymbol{M} \\ \boldsymbol{H} \\ \boldsymbol{C} \end{pmatrix}, \boldsymbol{b}(\boldsymbol{V}) = \begin{pmatrix} \delta(\boldsymbol{V}) \\ 0 \\ e(\boldsymbol{V}) \\ \boldsymbol{U} \end{pmatrix}.$$

Note that the matrix $\boldsymbol{A}$ is dependent only on the graph before deformation, while $\boldsymbol{b}$ is dependent on the current point positions $\boldsymbol{V}$. This is a nonlinear least squares problem. Previous methods try to make this a linear least squares problem solvable either by removing the dependency of $\boldsymbol{b}$ on $\boldsymbol{V}$ or by using a linear approximation for $\boldsymbol{b}$. In the following, we introduce an iterative Gauss–Newton method [12] to solve this nonlinear problem directly.

4.2 Nonlinear least squares optimization

The iterative Gauss–Newton method solves the problem in the following way:

$$\min_{\boldsymbol{V}^{k+1}} \|\boldsymbol{AV}^{k+1} - \boldsymbol{b}(\boldsymbol{V}^k)\|^2, \tag{8}$$

where $\boldsymbol{V}^k$ is the point positions solved from the k-th iteration and $\boldsymbol{V}^{k+1}$ is the point positions that we want to solve at iteration $k+1$. Since $\boldsymbol{b}(\boldsymbol{V}^k)$ is known at the current iteration, Eq. 8 can be solved through a linear least squares system:

$$\boldsymbol{V}^{k+1} = (\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T\boldsymbol{b}(\boldsymbol{V}^k) = \boldsymbol{Gb}(\boldsymbol{V}^k). \tag{9}$$

Let $\boldsymbol{G} = (\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T$. Since $\boldsymbol{A}$ is dependent only on the graph before deformation, $\boldsymbol{G}$ can be precomputed before deformation and is fixed during deformation. Therefore, only a back substitution is executed for each iteration. In this way, the deformation algorithm is able to run interactively.

During each iteration, $\boldsymbol{b}$ is computed according to the point positions $\boldsymbol{V}^k$ from the last iteration. In other words, we need to compute $\delta(\boldsymbol{V}^k)$ and $e(\boldsymbol{V}^k)$.

$e(\boldsymbol{V}^k)$ is computed as follows:

$$e(v_i^k, v_j^k) = \frac{\widetilde{l_{i,j}}}{|v_i^k - v_j^k|}(v_i^k - v_j^k), \quad \text{for } (i, j) \in E_g.$$

To compute the new Laplacian coordinates $\delta(\boldsymbol{V}^k)$ is somewhat complicated. Specifically, we compute a transformation matrix T_i^k for each point $v_i \in V_p$:

$$\delta(v_i^k) = T_i^k \delta(v_i^0),$$

where $\delta(v_i^0)$ is the curve Laplacian coordinate before deformation.

By taking v_i^0 and v_i^k as the rotation centers, the transform matrix T_i^k can be computed by minimizing the following energy [15]:

$$\sum_{(i,j)\in E_p} \|T_i^k(v_j^0 - v_i^0) - (v_j^k - v_i^k)\|^2.$$

Taking the derivatives to all coefficients of T_i^k to be zero, we have:

$$T_i^k = \sum_{(i,j)\in E_p} (v_j^k - v_i^k)(v_j^0 - v_i^0)^T D_i,$$

where $D_i = \left(\sum_{(i,j)\in E_p}(v_j^0 - v_i^0)(v_j^0 - v_i^0)^T\right)^{-1}$, which depends on the original shape only and can also be precomputed to accelerate the algorithm.

5 Preservation of the global area

In this section, we introduce a way to preserve the global area of the shape to simulate an incompressible 2D object. As can be seen in the following, global area preservation is handled as a hard constraint in the nonlinear least squares problem (Eq. 7), and the iterative solver described above can be adapted to solve this constrained problem efficiently.

The area of a polygon is computed using the coordinates of the polygon points: $g(\boldsymbol{V}_p) = \frac{1}{2}\sum_{i=0}^{m}(x_i y_{i+1} - x_{i+1} y_i)$, where (x_i, y_i) is the coordinate of point v_i. Then the global area constraint can be formulated as follows:

$$g(\boldsymbol{V}) - \widetilde{g} = 0,$$

where $\widetilde{g}$ is the area of the original shape before deformation.

Since the global area constraint is a nonlinear function of the coordinates of the polygon points, it cannot be written into matrix form. Thus, we treat this constraint as a hard constraint and extend Eq. 7 to:

$$\min_{\boldsymbol{V}} \|\boldsymbol{AV} - \boldsymbol{b}(\boldsymbol{V})\|^2, \quad \text{subject to } g(\boldsymbol{V}) - \widetilde{g} = 0. \tag{10}$$

This constrained non-linear least squares problem can also be solved by extending the iterative solver (Eq. 8) to the following formula:

$$\min_{\boldsymbol{V}^{k+1}} \|\boldsymbol{AV}^{k+1} - \boldsymbol{b}(\boldsymbol{V}^k)\|^2, \quad \text{subject to } g(\boldsymbol{V}^{k+1}) - \widetilde{g} = 0. \tag{11}$$

Letting $\boldsymbol{h} = \boldsymbol{V}^{k+1} - \boldsymbol{V}^k$, $\boldsymbol{AV}^{k+1} - \boldsymbol{b}(\boldsymbol{V}^k)$ can be reformulated as a new function $l(\boldsymbol{h})$, which only depends on $\boldsymbol{h}$:

$$\begin{aligned} l(\boldsymbol{h}) &= \boldsymbol{AV}^{k+1} - \boldsymbol{b}(\boldsymbol{V}^k) \\ &= \boldsymbol{A}(\boldsymbol{V}^k + \boldsymbol{h}) - \boldsymbol{b}(\boldsymbol{V}^k) \\ &= \boldsymbol{Ah} + \boldsymbol{AV}^k - \boldsymbol{b}(\boldsymbol{V}^k). \end{aligned} \tag{12}$$

The problem 11 is converted to:

$$\min_{\boldsymbol{h}} \frac{1}{2}\|l(\boldsymbol{h})\|^2, \quad \text{subject to } g(\boldsymbol{V}^k + \boldsymbol{h}) - \widetilde{g} = 0. \tag{13}$$

By locally linearizing

$$g(\boldsymbol{V}^k + \boldsymbol{h}) \approx g(\boldsymbol{V}^k) + \boldsymbol{J}_g(\boldsymbol{V}^k)\boldsymbol{h},$$

and applying Lagrange multipliers [13] with Newton's method, the solution to Eq. 13 is:

$$\begin{aligned} &\boldsymbol{h} = (\boldsymbol{A}^T\boldsymbol{A})^{-1}(\boldsymbol{A}^T\boldsymbol{S} - \boldsymbol{J}_g^T\lambda) \\ &\lambda = -(\boldsymbol{J}_g(\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{J}_g^T)^{-1}(t - \boldsymbol{J}_g(\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T\boldsymbol{S}), \end{aligned}$$

where $\boldsymbol{J}_g$ is the Jacobian of g, $\boldsymbol{S} = \boldsymbol{b}(\boldsymbol{V}^k) - \boldsymbol{AV}^k$, and $t = \widetilde{g} - g(\boldsymbol{V}^k)$.

6 Experimental results

We implemented the described deformation algorithm on a 3.2 GHz Pentium 4 workstation with 1 GB memory. Table 1 shows the data statistics and timings for the models presented in this paper. The solution time refers to the per-iteration cost. The number of iterations needed for convergence of the solver varies significantly and depends on many factors, for example, the shape itself and the magnitude of the deformation. For the models used in this paper, the average number is 10. Therefore, the performance of our deformation system is comparable to previous linear methods [7]. As shown in the

Table 1. Statistics and timings

2D Shape	Flower	Horse	Character
# Boundary vertices	114	247	143
# Interior vertices	256	189	163
Precomputing time	22 ms	22.7 ms	18.3 ms
Solution time	0.589 ms	0.593 ms	0.470 ms

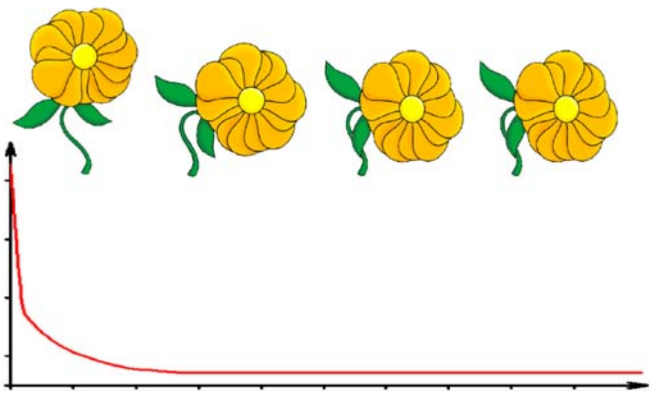


Fig. 4. Convergence of our iterative solver. The red curve indicates energy

accompanying video, our system is very easy to use and runs in real-time. The user only needs to drag a few points on the shape to the desired locations, and the whole shape will be deformed in a visually pleasing manner.

In Fig. 4, we show an example to demonstrate the convergence of our iterative solver. The curve is generated by setting the constraint points to the target positions and letting the solver iterate until convergence. In this example, the solver converges after about 10 iterations. The solution time of our solver (see Table 1) is very fast.

Figure 3 compares the deformation results with and without local area preservation. If we only preserve Laplacian coordinates, the deformation result looks unnatural with obvious self-intersection. By adding graph mean-

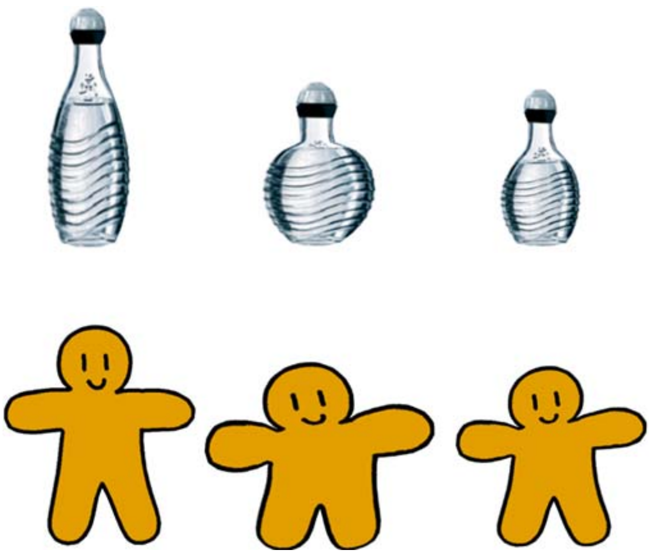


Fig. 5. Deformation with (middle) and without (right) global area preservation. The original 2D shapes are shown on the left. Note that we can exactly preserve the global area by taking it as a hard constraint

value coordinates and edge length constraints to control the local area inside the 2D shape, the result looks much more pleasing.

Figure 5 demonstrates the effect of the global area constraint. With global area preservation, an object is squashed horizontally when it is stretched vertically. Therefore, the deformation results with global area preservation look fatter than those without global area con-

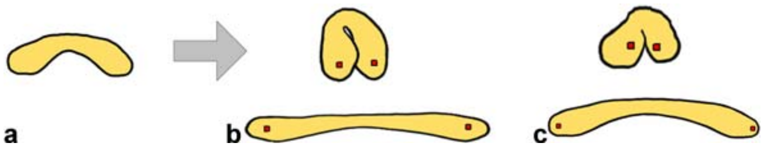


Fig. 6a–c. Comparison between our algorithm and [7]. **a** Rest shape; **b** our algorithm; **c** Igarashi et al. [7]

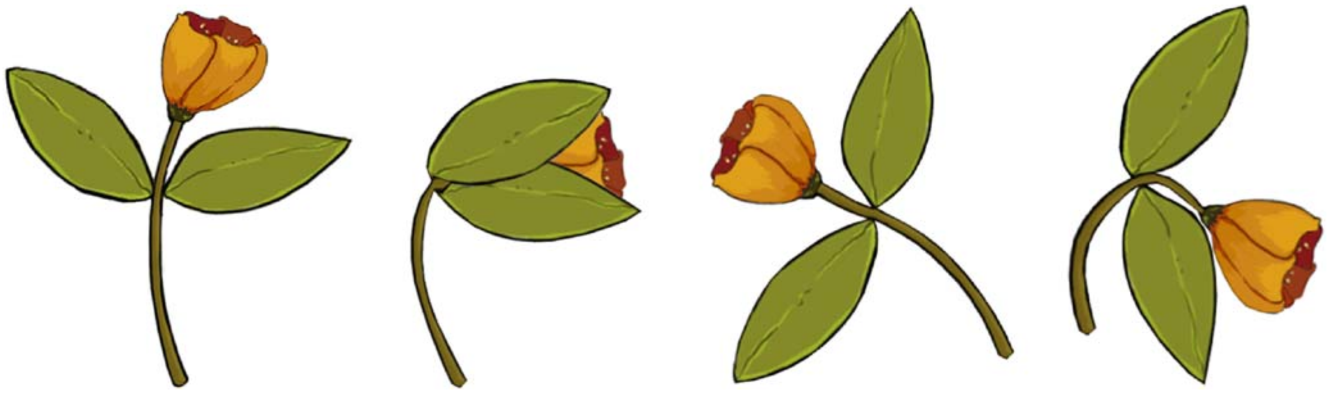


Fig. 7. Deformation of a flower. From left to right are the original shape and the deformation results, respectively

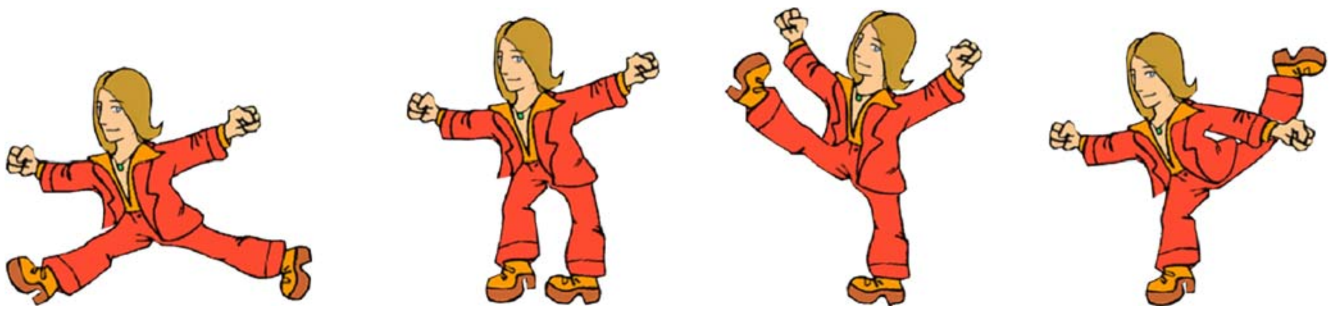


Fig. 8. Deformation of a cartoon character. From left to right are the original shape and the deformation results, respectively

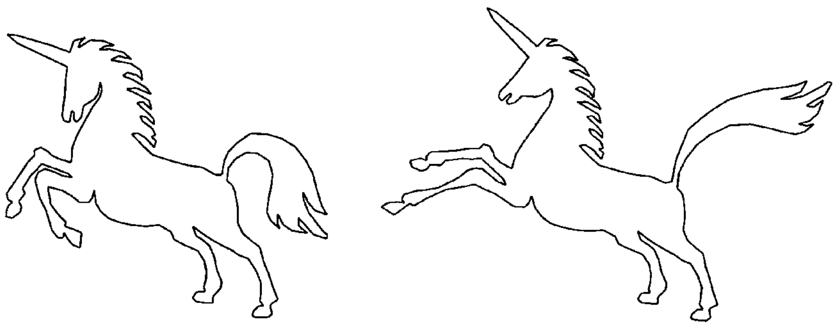


Fig. 9. Deformation of a horse. Left: the original shape. Right: the deformation result

straint, as would be expected for incompressible objects.

For most examples presented in this paper, our results are as good as those generated by the linear method [7]. In some cases, our nonlinear least squares optimization leads to more physically plausible results than those presented in [7]. Figure 6 shows the deformation results for the shape in Fig. 19 of [7] (see the accompanying video for the deformation process).

We have tested our deformation algorithm on various kinds of 2D shapes. Figure 7 shows the deformation of a flower. The stem of the flower is deformed naturally, and the shape of the flower is preserved well. Our system can also be used to deform cartoon characters (Figs. 1 and 8). Figure 8 shows a large scale deformation of the legs of the cartoon man. Figure 9 illustrates the deformation result of a horse. The details at the tail and back of the horse are well preserved even with large deformations.

7 Conclusion

We have described a real-time 2D shape deformation algorithm based on nonlinear least squares optimization. Our algorithm is able to preserve both local and global properties of the input shape. The nonlinear nature of our algorithm allows it to outperform previous linear methods.

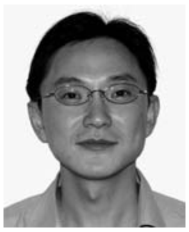
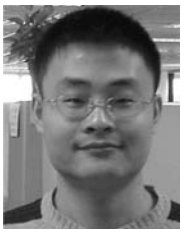
In future work it may be interesting to experiment with some methods that dynamically adjust the depth when different parts of the shape overlap. Currently, we are using a statically predefined depth order, which does not work well in some cases. Our algorithm can also be applied to 2D cartoon animation retargeting by defining a set of corresponding points between 2D shapes.

Acknowledgement We would like to thank Yuan Tian for providing the drawings. Thanks to Christine Cheng and Steve Lin for their help in proof-reading this paper.

References

1. Alexa, M., Cohen-Or, D., Levin, D.: As-rigid-as-possible shape interpolation. In: SIGGRAPH 2000 Conference Proceedings, pp. 157–164 (2000)
2. Au, O.K.C., Tai, C.L., Liu, L., Fu, H.: Mesh editing with curvature flow Laplacian operator. Computer Science Technical Report, HKUST-CS05-10 (2005)
3. Celniker, G., Gossard, D.: Deformable curve and surface finite-elements for free-form shape design. In: SIGGRAPH 91 Conference Proceedings, pp. 257–266 (1991)
4. Floater, M.S.: Mean value coordinates. Comput. Aided Geom. Des. **20**(1), 19–27 (2003)
5. Gibson, S.F.F., Mirtich, B.: A survey of deformable modeling in computer graphics. Technical report TR-97-19, Mitsubishi Electric Research Laboratories (1997)
6. Huang, J., Shi, X., Liu, X., Zhou, K., Wei, L., Teng, S., Bao, H., Guo, B., Shum, H.Y.: Subspace gradient domain mesh deformation. ACM Trans. Graph. **25**(3), 1126–1134 (2006)
7. Igarashi, T., Moscovich, T., Hughes, J.F.: As-rigid-as-possible shape manipulation. ACM Trans. Graph. **24**(3), 1134–1141 (2005)
8. James, D.L., Pai, D.K.: Artdefo: Accurate real time deformable objects. In: SIGGRAPH 99 Conference Proceedings, pp. 65–72 (1999)

9. Lewis, J.P., Cordner, M., Fong, N.: Pose space deformation: a unified approach to shape interpolation and skeleton-driven deformation. In: SIGGRAPH 2000 Conference Proceedings, pp. 165–172 (2000)
10. Lipman, Y., Sorkine, O., Levin, D., Cohen-Or, D.: Linear rotation-invariant coordinates for meshes. ACM Trans. Graph. **24**(3), 479–487 (2005)
11. MacCracken, R., Joy, K.: Free-form deformations with lattices of arbitrary topology. In: SIGGRAPH 96 Conference Proceedings, pp. 181–188 (1996)
12. Madsen, K., Nielsen, H., Tingleff, O.: Methods for nonlinear least squares problems. Tech. rep., Informatics and Mathematical Modelling, Technical University of Denmark (2004)
13. Madsen, K., Nielsen, H., Tingleff, O.: Optimization with constraints. Tech. rep. Informatics and Mathematical Modelling, Technical University of Denmark (2004)
14. Milliron, T., Jensen, R., Barzel, R., Finkelstein, A.: A framework for geometric warps and deformations. ACM Trans. Graph. **21**(1), 20–51 (2002)
15. Müller, M., Heidelberger, B., Teschner, M., Gross, M.: Meshless deformations based on shape matching. ACM Trans. Graph. **24**(3), 471–478 (2005)
16. Sederberg, T., Parry, S.: Free-form deformation of solid geometric models. SIGGRAPH '86 Conference Proceedings **20**(4), 151–160 (1986)
17. Sheffer, A., Kraevoy, V.: Pyramid coordinates for morphing and deformation. In: Proceedings of 3DPVT, pp. 68–75. IEEE, Thessaloniki, Greece (2004)
18. Sorkine, O., Lipman, Y., Cohen-Or, D., Alexa, M., Rössl, C., Seidel, H.P.: Laplacian surface editing. In: Symposium on Geometry Processing, pp. 179–188. ACM SIGGRAPH / Eurographics (2004)
19. Yu, Y., Zhou, K., Xu, D., Shi, X., Bao, H., Guo, B., Shum, H.Y.: Mesh editing with poisson-based gradient field manipulation. ACM Trans. Graph. **23**(3), 644–651 (2004)
20. Zhou, K., Huang, J., Snyder, J., Liu, X., Bao, H., Guo, B., Shum, H.Y.: Large mesh deformation using the volumetric graph Laplacian. ACM Trans. Graph. **24**(3), 496–503 (2005)

YANLIN WENG is a PhD student at the University of Wisconsin – Milwaukee. She received her BE and MS degrees in control science and engineering from Zhejiang University in 1999 and 2002, respectively.

WEIWEI XU is an associate researcher of the graphics group at Microsoft Research Asia. He received his BS and MS degrees in computer science from Hohai University and his PhD from Zhejiang University. His research interests include character animation and geometric modeling.

YANCHEN WU is an undergraduate student in Zhejiang University. His research interests include geometry processing and real-time rendering.

KUN ZHOU is a researcher/project leader of the graphics group at Microsoft Research Asia. He received his BS and PhD degrees in computer science from Zhejiang University in 1997 and 2002, respectively. His current research focus is geometry processing, texture processing and real-time rendering. He holds over 10 granted and pending US patents. Many of these techniques have been integrated in Windows Vista, DirectX and XBOX SDK.

BAINING GUO is the research manager of the graphics group at Microsoft Research Asia. Before joining Microsoft, Baining was a senior staff researcher in Microcomputer Research Labs at Intel Corporation in Santa Clara, California, where he worked on graphics architectures. Baining received his PhD and MS degrees from Cornell University and his BS degree from Beijing University. Baining is an associate editor of IEEE Transactions on Visualization and Computer Graphics. He holds over 30 granted and pending US patents.