Polygon Morphing Using a Multiresolution Representation

Eli Goldstein and Craig Gotsman

Department of Computer Science
Technion - Israel Institute of Technology
Haifa 32000, Israel

e-mail: {goldy,gotsman}@cs.technion.ac.il
Tel: +972-4-294336

Abstract

We present an algorithm for morphing between two simple polygons. The two polygons are converted to multiresolution representations. Intermediate representations are generated from these two, from which intermediate polygons are reconstructed. Our algorithm is simpler than the few existing polygon morphing schemes, and its results compare favorably. The key to the success of our algorithm is the multiresolution shape representation, based on *curve evolution* schemes. This representation captures the geometric properties of a shape at different levels of detail, a crucial requirement for aesthetic shape deformations.

Keywords: Polygon Morphing, Multiresolution, Curve Evolution.

1 Introduction

Morphing (*metamorphosis*) is a term reserved for processes which, when given two objects, continuously deform one to the other. Morphing of objects is popular in animations seen in the entertainment and broadcasting industry. However, many of the spectacular effects that have been achieved, have, by large, been done manually, albeit with the support of a computer. This is notoriously time consuming, so a major research challenge is the design of algorithms which morph objects automatically with a minimum of manual intervention. The only manual intervention we would like to require is the input of some *correspondence points* between the objects, which guide the morphing process. The generation of the actual intermediate objects should be completely automatic. Algorithms have been designed for the morphing of images [1], polygons [19, 18, 20], polyhedra [2] and volume data [10, 9]. This work concentrates on 2D polygons.

More formally, the morphing problem may be formulated as follows: Given two objects O_0, O_1 and $t \in [0,1]$, construct an intermediate object O_t which is similar to O_0 as $t \to 0$ and similar to O_1 as $t \to 1$. In a way, the morphing problem is a multidimensional interpolation problem, and, as such, is ill-posed, in the sense that there are many possible interpolants O_t satisfying these very vague conditions. An accepted way of reducing the number of possible solutions to an interpolation problem is by *regularization*, namely, constraining the solution to satisfy a variety of other "natural" conditions not explicit in the input. In the case of 2D polygons, the following are some natural conditions that an intermediate morphed curve should satisfy: If P and Q are closed and simple, then the interpolant R should also be closed and simple. The area of R should vary smoothly between that of P and Q. If Q is a translated (rotated) version of P, then R should also be an appropriate translation (rotation) of P. These properties should hold both globally, and locally, in some sense.

The most naive algorithm for morphing between two polygons P and Q, namely, interpolation along a line segment between corresponding vertices of P and Q ("vertex interpolation"), fails to satisfy any of the conditions mentioned above. The interpolant easily intersects itself, and simple geometric properties, such as lengths, angles and areas, do not change in a consistent manner, as Fig. 1(b) demonstrates. The main reason why the linear vertex interpolation yields bad results is that each vertex is treated independently of all other vertices. Sederberg et. al [18] propose a method in which interpolation is performed on edge lengths and angles. Since this type of interpolation might transform closed polygons into open ones,

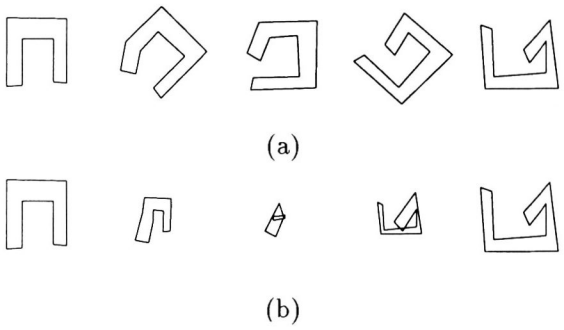

Figure 1: Morphing two polygons: (a) Natural morph. (b) Linear vertex interpolation morph.

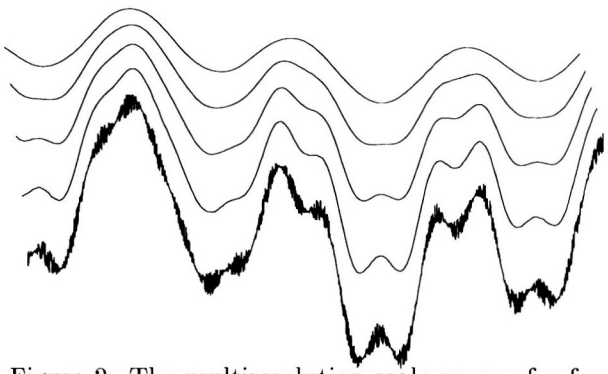

Figure 2: The multiresolution scale-space of a function, obtained by smoothing with a Gaussian filter of increasing width. The number of second derivate zero crossings decreases with scale.

an optimization algorithm must be used to guarantee closedness. The procedure they use, however, does not guarantee simpleness of the interpolant. Furthermore, the method tends to distort polygon area, as it does not consider the polygon interior. Shapira at. al [20] propose a complex algorithm which takes into account the polygon interior as well as its boundary. Their approach builds on the so-called *star-skeleton* representation of a polygon. Morphing is then performed on the *representations*, from which intermediate polygons are reconstructed. The advantage of this approach is that it yields intermediate polygons which are closed and have natural shapes. Simplicity, however, is not guaranteed. The disadvantage of this algorithm is the complexity of the star-skeleton representation extraction procedure.

The reason the algorithm of [20] achieves such good results is its use of a natural representation of a polygon, accounting for its important geometric features. It seems that this is the key to successful shape morphing algorithms. Such a representation should emphasize the intrinsic object geometry in a way consistent with human perception of that geometry.

We propose a different representation of polygons, with a multiresolution character. The human visual system is able to appreciate the fine details of an object, but also its overall geometric shape. This type of *multiresolution* perception has found its way into computing in the form of multiresolution data structures [5], analysis [13], and synthesis [7, 6]. The multiresolution representation we use is based on *polygon evolution* schemes. This representation contains information on the polygon at many levels of detail. Morphing is then performed in the more natural space of the representations, from which the intermediate polygons are reconstructed.

2 Curve Evolution

In his, by now classic, work on multiresolution representation of one-dimensional functions (signals), Witkin [21] describes a scheme in which the function is continuously smoothed by filtering with a Gaussian kernel of increasing width. This produces a sequence of functions whose number of second-derivative zero-crossings does not increase in time (see Fig. 2). Advancing along the time axis, also known as the *scale* axis, reduces the *resolution* of the function so that only its low-frequency, or very coarse, behavior is visible. At the limit of very low resolution - a straight line is obtained. This (highly redundant) decomposition of a signal has been useful in a variety of signal processing applications. In a series of papers, Mokhatarian et al. extended the methods of Witkin to 2D closed parametric curves by first performing 1D smoothing techniques independently on each of the coordinates [14], and then accounting for the curve geometry by normalizing the filter kernel coordinate system by the curve arc-length [15]. It can be shown that application of these so-called *curve evolution* techniques cause any smooth closed curve to converge smoothly to a point. These methods, however, do not take into account directly the intrinsic curve geometry. Others [16, 11] have used a curve evolution method which does achieve this. Rather than explicitly *smoothing* the curve by a filtering process, the evolution is described by the motion of the points on the curve in time. Loosely speaking, during evolution, a point on the curve advances in the direction of the vector normal at that point, by a distance proportional to a function of the curvature at that point (see Fig. 3). The precise mathematical

formulation of the process is: Given a simple closed parametric curve $C_0(s) \in G^2$, its geometric evolution at time $t > 0$ is

$$C_t(s) = C_0(s) + \int_0^t F(k_\tau(s))\vec{N}_\tau(s)d\tau \qquad (1)$$

where $k_\tau(s)$ is the curvature function, and $\vec{N}_\tau(s)$ the normal vector function at time τ. Expressing the quantities involved as as a function of two variables, s and t, the differential form of the evolution rule is similar to the *heat equation* [8]:

$$\frac{\partial C(t,s)}{\partial t} = F(k(t,s))\vec{N}(t,s) \qquad (2)$$

In this process, a point on a convex region of the curve advances "inwards", and a point on a concave region advances "outwards".

Many functions F have been proposed for use in (1) [16, 11, 17], each defining an evolution with different mathematical properties. Some caution must be excercised when choosing F. In the degenerate case $F(x) = const$, a point on the curve evolves along the direction of the normal a *constant* distance (independent of the curvature at that point). This corresponds to *offsetting* the curve [12]. Not only does this type of evolution rule not preserve the geometry of the curve, but singularities ("shocks") develop along the evolving curve, especially at areas of high curvature.

The function F we chose for our purposes is the scaled identity function $F(x) = ax$. The effect of the resulting process is to continuously smooth the curve, having the following attractive properties [8, 16]:

- A non-simple curve becomes simple.
- A simple curve remains simple.
- A smooth curve remains smooth.
- A non-convex curve becomes convex.
- The number of curvature zero crossings is non-increasing.
- The curve converges to a point in the asymptotic form of an circle.

Starting with any planar curve, at some stage the curve becomes simple, then convex, then circular, then shrinks until it reaches a point (see Fig. 4). The number of curvature zero-crossings does not increase and no singularities develop along the curve. The curves obtained at later times during the evolution correspond to coarse low-resolution versions of the original, which has been indirectly "smoothed" to eliminate its finer details. One of the crucial properties of this curve evolution process is the second

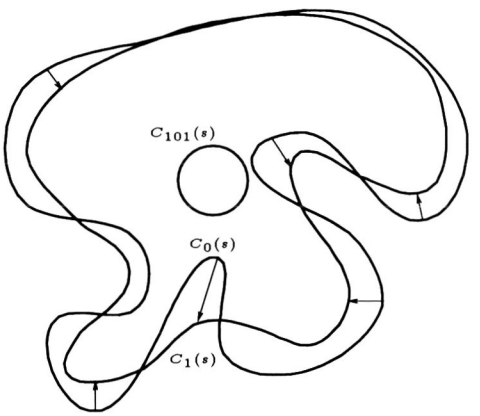

Figure 3: $C_1(s)$ and $C_{101}(s)$ obtained from geometric evolution of the closed simple curve $C_0(s)$.

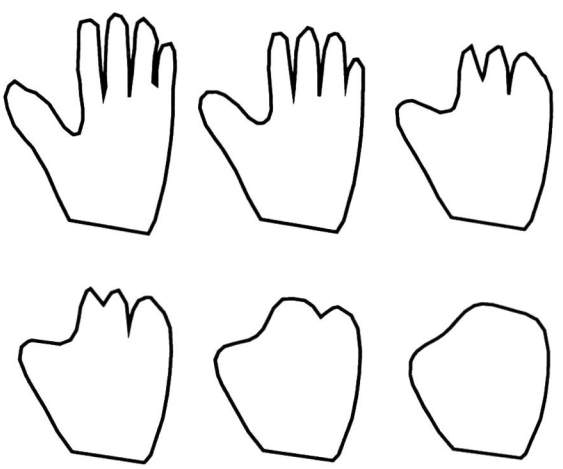

Figure 4: Geometric evolution of a closed simple curve.

in the list above: a simple curve remains simple, i.e. does not intersect itself during evolution. This will be important for the morphing process. Not all functions F used in conjunction with (1) yield evolution schemes with this property.

3 Multiresolution Representations

3.1 The Continuous Case

As described in Section 2, during a geometric curve evolution process, every point on the curve advances continuously in the plane on a path beginning at the point on the original curve, and terminating at a common point. The geometry of this path, and the paths of other curve points, describes the shape of the curve at different scales, first at a very fine scale, and later

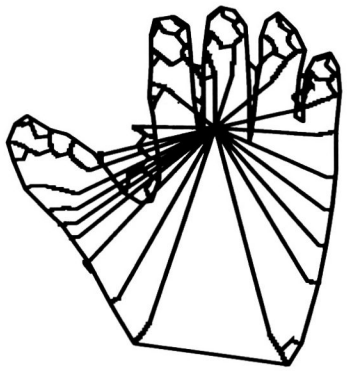

Figure 5: Geometric evolution paths of some points on the curve of Fig. 4.

at a very coarse one. Note that paths may extrude from the curve, especially in concave regions, as Fig. 5 demonstrates. A possible multiresolution representation of a curve is the set of paths of the curve points generated by the geometric evolution process. This representation has a high degree of redundancy (as the actual curve is just the set of path endpoints), but it is precisely this redundancy which makes it so useful for morphing purposes.

3.2 The Discrete Case

In a computerized morphing system, a curve will usually be given in discrete form, i.e. as a polygon $P = (p_0, p_1, ..., p_n)$, where $p_i = (x_i, y_i)$. In order to produce a multiresolution representation for polygons, the continuous curve evolution schemes described in Section 2 must be adapted to a discrete setup, where only polygon vertices undergo evolution. This is called *polygon evolution*. A number of different polygon evolution schemes have been studied [3, 4]. We choose to use a discrete version of (2) with $F(x) = ax$. This is because the continuous version preserves simplicity, so hopefully the discrete one will too. To use (2) in a discrete form, the notions of "normal" and "curvature" must be redefined, while keeping their meaning similar to that of the continuous case. Towards this end, a variety of definitions are possible [4]. We use the following discrete "derivatives" (all indices modulu n):

$$\dot{x}_i = \frac{x_{i+1} - x_{i-1}}{s_{i+1} - s_{i-1}} \qquad \dot{y}_i = \frac{y_{i+1} - y_{i-1}}{s_{i+1} - s_{i-1}}$$

$$\dot{x}_i^{for} = \frac{x_{i+1} - x_i}{s_{i+1} - s_i} \qquad \dot{x}_i^{back} = \frac{x_i - x_{i-1}}{s_i - s_{i-1}}$$

$$\ddot{x}_i = \frac{\dot{x}_i^{for} - \dot{x}_i^{back}}{s_{i+1} - s_{i-1}} \qquad \ddot{y}_i = \frac{\dot{y}_i^{for} - \dot{y}_i^{back}}{s_{i+1} - s_{i-1}}$$

where $s_i = \Sigma_{j=0}^{i-1} ||p_{j+1} - p_j||$. The curvature k_i at p_i is then

$$k_i = \frac{\dot{x}_i \ddot{y}_i - \ddot{x}_i \dot{y}_i}{(\dot{x}_i{}^2 + \dot{y}_i{}^2)^{3/2}}$$

The normal direction $\vec{n}_i$ is the direction of the angle bisector at vertex p_i if p_i is on a convex region, and the opposite direction if p_i is on a concave region.

Polygon evolution is performed in discrete steps (iterations), generating paths which are vector sequences. The "size" of one evolution step is controlled by the positive real parameter a in $F(x) = ax$. Alternatively, a is the *speed* of the process. When implementing the polygon evolution, care should be taken not to use too small or too large an a. Theoretically, if the step size is not too large, polygon evolution also converges to a point via a (discrete) circular shape. In practice, we terminate the evolution when a convex polygon is obtained, and then add vectors from these vertices to the polygon centroid. The number of iterations required to achieve convexity is called the *depth* of the evolution, or just the depth of the polygon. It obviously depends on the evolution step size.

The polygon *representation* is the set of vector paths generated for each vertex during the polygon evolution process. Each vector in the representation is accompanied by an indication of its in/out direction relative to the polygon. Most evolution path vectors point towards the polygon interior, but some of the high-resolution path vectors may point towards the exterior (in concave regions). The vectors close to the polygon centroid describe low-resolution geometric properties of the polygon, and the vectors close to the polygon vertices describe high-resolution properties. Note that the number of vertices of the polygons generated during evolution is fixed, in contrast to wavelet-style multiresolution decompositions of signals [7], where the number of samples decreases at lower detail levels. This redundancy makes the reconstruction process easier.

As we expected, the polygons evolving from a simple polygon, using our method, almost never intersected themselves, as opposed to other polygon evolution schemes we experimented with.

4 User-Defined Correspondence

In order that *any* morphing algorithm between polygons P and Q produce reasonable results, it must be guided by some minimal user input. An accepted way of doing this is to provide a small number of *correspondence* pairs indicating vertices of P which

transform to vertices of Q. This is a list of index pairs $\{(I_i, J_i) : i = 1, .., k\}$, such that P_{I_i} corresponds to P_{J_i} for $i = 1, .., k$. These correspondence pairs are usually feature points with special meaning for the user, and a morph not preserving this correspondence is not of interest. The number of user-supplied correspondence pairs is usually much less than the number of vertices of P or Q. We call this a *partial* correspondence. A correspondence involving *all* vertices of P and Q is called a *full* correspondence. Note that even if a full correspondence is supplied by the user, there are still many possible ways to interpolate the two polygons, and the morphing problem is far from solved.

5 The Morphing Algorithm

Given polygons P, Q, a partial correspondence between them, and $t \in [0, 1]$, our morphing algorithm proceeds in five stages:

1. Complete the partial correspondence between P and Q to a full correspondence.
2. Generate multiresolution representations R_0 and R_1 for P and Q respectively.
3. Correlate the representation depths of R_0 and R_1.
4. "Interpolate" R_0 and R_1 to obtain an intermediate representation R_t.
5. Reconstruct an intermediate polygon from R_t.

Of course, in order to create a complete morph sequence for a set of discrete times $t_1, .., t_m \in [0, 1]$, it suffices to perform Steps 1-3 once in a preprocessing stage, and Steps 4-5 for each of the different t_i. In the following sections we elaborate on the various stages of the algorithm.

5.1 Correspondence Completion

To prepare the two polygons for morphing, we first complete the user-specified partial correspondence to a full correspondence by a closest vertex method. In this method a vertex of P corresponds to the closest vertex (by arc length) of Q within the partial correspondence segment, and vice versa. A vertex of P (Q) which corresponds to more than one vertex of Q (P) is duplicated. Given P and Q with m and n vertices respectively and partial correspondence containing $k \leq \min(m, n)$ vertex pairs, the full correspondence will contain $\max(m, n)$ pairs (see Fig. 6).

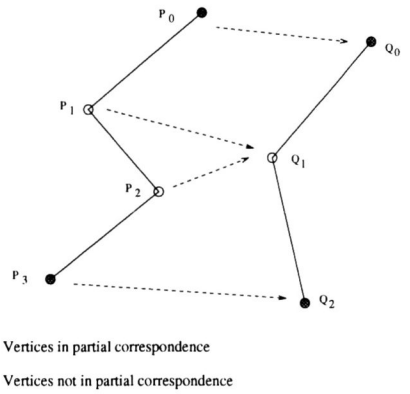

Figure 6: Completing a partial correspondence to a full correspondence: given segments between correspondence points in polygons P and Q with 4 and 3 vertices respectively, the partial correspondence is completed with 2 more pairs by duplicating a vertex of Q.

5.2 Depth Correlation

Since the representation depths of polygons P and Q will probably be different, it is neccesary to correlate them before attempting morphing. If d_1 and d_2 are the representation depths of P and Q, respectively, the correlated depth will be $d_1 + d_2 - \gcd(d_1, d_2)$. This is obtained by merging together the paths by proportional depth. For example, if the depth of P is 50, and the depth of Q is 100, each vector in every path in the representation of P is broken into 2 smaller (collinear) vectors. P will then also have depth 100.

5.3 Interpolating Representations

After correspondence completion of P and Q, polygon evolution, and depth correlation, we have multiresolution representations R_0 and R_1 with an identical depth and number of paths. To obtain an intermediate representation R_t from R_0 and R_1, we must specify how an individual path of R_0, and the corresponding path of R_1, are interpolated for any $t \in [0, 1]$. This is done independently for all paths in the representations, and the intermediate polygon is then reconstructed from R_t. It is important to emphasize that, although the polygon evolution process *generates* the multiresolution representation, the temporal morph sequence *between* the two input polygons will in no way resemble the temporal evolution of any of these input polygons. These are two separate and very different time axes.

Between two paths, $\mathcal{P}_0$ and $\mathcal{P}_1$, we interpolate starting from the path vectors corresponding to the

low resolution version of the polygon, and work our way out towards the path vectors corresponding to the high resolution details. The former capture global transformations that must be performed in order to align the shapes, and the latter capture the local transformations (beyond the global ones) needed to align the fine details of the two polygons. The first global rotation is determined using the full correspondence. This is calculated as the average angle between the *low resolution* vectors in the corresponding path pairs. These low resolution vectors are used first because they contain *global* information on the polygon. The additional "local" transformations required beyond this global one are determined from the higher resolution path vector pairs, working from low resolution to high resolution.

The local transformations that we use are scaling and rotation. The basic issue at hand is how to interpolate two corresponding path vectors. Consider two vectors $v_0 \in \mathcal{P}_0$ and $v_1 \in \mathcal{P}_1$ such that $\text{ang}(v_0, v_1) = \theta$. Two possible interpolants v_t for $t \in [0,1]$ would either be the v_t satisfying $||v_t|| = t||v_1|| + (1-t)||v_0||$ and $\text{ang}(v_0, v_t) = t\theta$, or that satisfying $||v_t|| = -t||v_1|| + (1-t)||v_0||$ and $\text{ang}(v_0, v_t) = t(\theta - \pi)$. Heuristically, we choose between these two options depending on the in/out directions of v_0 and v_1 relative to their polygons. The first option is used if and only if v_0 and v_1 have the same relative direction. Vector pair interpolation is performed for later path vector pairs after accumulating the effects of all interpolating transformations on earlier path vector pairs (see Fig. 7). Note that if *only* the linear interpolant $v_t = tv_1 + (1-t)v_0$ was used on all path vector pairs, our interpolation method would be equivalent to linear vertex interpolation, as the accumulative effect of many such local interpolations would still be a linear interpolation.

5.4 Polygon Reconstruction

Reconstructing intermediate polygons from intermediate representations is very easy. Simply integrate (sum) the path vectors for each vertex from the centroid. Each path yields a vertex on the intermediate polygon.

6 Complexity

The space requirements of the algorithm are dominated by the need to store the representations of the two input polygons. This space is proportional to the number of vectors in those representations - $O(nd)$, where n is the number of polygon vertices, and d is the polygon depth. The representations of intermediate polygons are not stored explicitly, rather calculated on the fly.

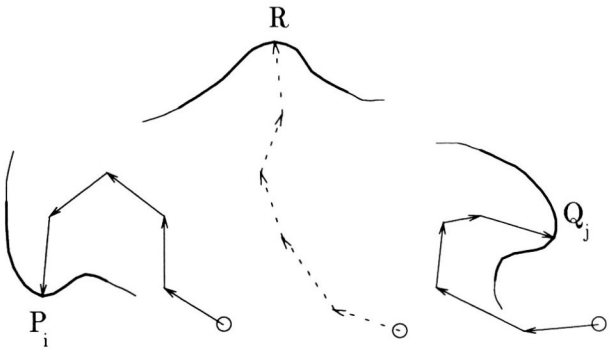

Figure 7: Path interpolation: R is the vertex on the intermediate polygon corresonding to the vertices P_i and Q_j on each of the input polygons.

The run time of the algorithm per intermediate polygon generated is also $O(nd)$, as every representation vector undergoes interpolation. The polygon depth d depends on the evolution step size and on the polygon complexity, but is insensitive to small perturbations in the input, as these are smoothed out and eliminated very quickly during evolution.

7 Experimental Results

It is virtually impossible to quantify the quality of the results of a morphing algorithm, as the true test is visual.

We have implemented our algorithm in an interactive morphing system with a X/Motif graphical user interface, run it on numerous test cases, and visually compared the results with those of Sederberg et. al [18] and Shapira et. al [20]. Fig. 8 shows some of these comparisons. The algorithm of Sederberg et. al (a) produces self intersections and shape and area distortions. The Shapira et. al algorithm (b) eliminates them. Our algorithm (c) yields similar intermediate polygons, sometimes preserving some of the more subtle features present in the shapes. In all our tests, we almost never encountered self-intersections in the morph sequences. This is probably due to the fact that our polygon evolution scheme almost always preserves polygon simplicity. We cannot, however, prove that one is a direct consequence of the other, as these are two different processes.

Fig. 9 shows just our results on a variety of polygons. In our opinion, the morph sequences produced

are quite natural, even for input polygons with significantly different shapes (see Fig. 9(d) in particular).

8 Conclusions

The use of natural geometric shape representations is a key element in any morphing scheme. In our opinion, it is hopeless to even try without this.

We have proposed a representation which accounts for shape geometry at various scales, based on a natural smoothing scheme. Using this representation in *reverse*, we are able to morph polygons, accounting first for global shape, and then for local deformations. Our results are at least as good as the only other existing algorithm using a similar approach of interpolation between representations, except our algorithm is simpler.

The morphing results depend on the exact polygon evolution scheme used to generate the polygon representations. We have experimented with a few schemes, and, while the results are similar, the scheme described here seemed to be the most successful. The similarity of the results is probably due to the fact that all the polygon evolution schemes perform the same basic multiresolution analysis. Some, however, preserve the polygon geometry better than others. Extensions to 3D object morphing are being investigated.

Acknowledgements

Thanks to the Technion polygon evolution gang: Alfred Bruckstein, Ronny Kimmel, Guillermo Sapiro and Doron Shaked, for discussions and online help. The input polygons of Figs. 1 and 8 were kindly provided by Michal Shapira and Ari Rappoport.

REFERENCES

[1] T. Beier and S. Neely. Feature-based image metamorphosis. *Computer Graphics*, 26:35–42, 1992.

[2] E.W. Bethel and S.P. Uselton. Shape distortion in computer-assisted keyframe animation. In *Proceedings of Computer Animation '89*. Springer, 1989.

[3] A.M. Bruckstein, G. Sapiro, and D. Shaked. Evolution of planar polygons. *International Journal of Pattern Recognition and Artificial Intelligence*, To appear, 1995.

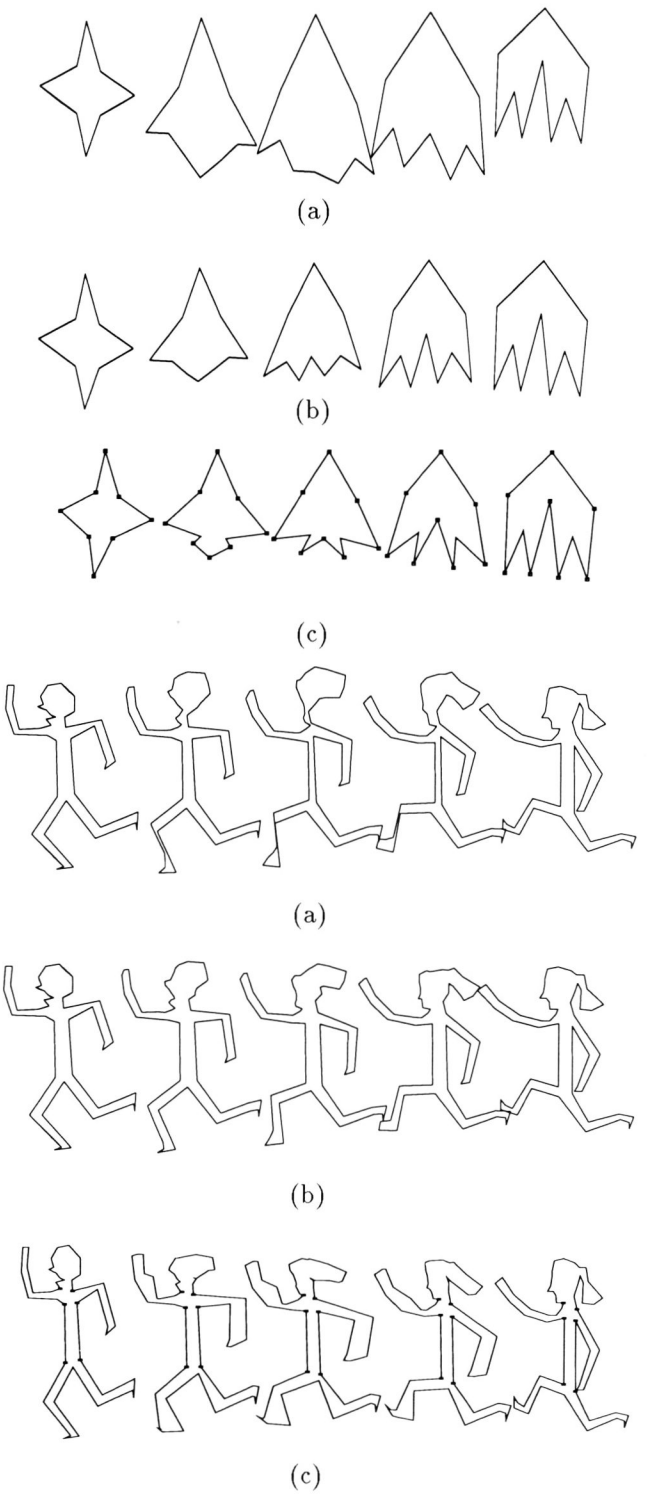

Figure 8: Comparative results of morphing algorithms. The user-supplied correspondence pairs are similar for all algorithms and are marked on the morph sequences produced by our algorithm. (a) Sederberg et. al (b) Shapira et. al (c) Ours.

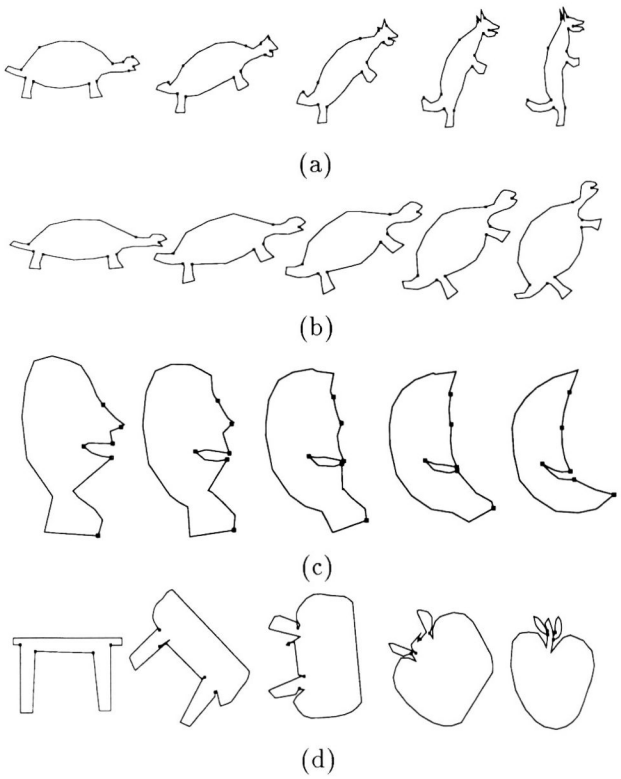

Figure 9: Results of our morphing algorithm. The user-supplied correspondence pairs are marked.

[4] A.M. Bruckstein and D. Shaked. On projective invariant smoothing and evolutions of planar curves and polygons. In *Aspects of Visual Form Processing: Proceedings of the workshop on Visual Form, Capri*, pages 109–118, May, 1994.

[5] P.J. Burt and E.H. Adelson. The Laplacian pyramid as a compact image code. *IEEE Transactions on Communications*, 31:532–540, 1983.

[6] G. Elber and C. Gotsman. Multiresolution control for non-uniform B-spline curve editing. *Preprint*, 1994.

[7] A. Finkelstein and D.H. Salesin. Multiresolution curves. *Computer Graphics*, 28:261–268, 1994.

[8] M. Gage and R.S. Hamilton. The heat equation shrinking convex plane curves. *Journal of Differential Geometry*, 23:69–96, 1986.

[9] T. He, S. Wang, and A. Kaufman. Wavelet-based volume morphing. In *Proceedings of Visualization '94*. IEEE Computer Society, 1994.

[10] J Hughes. Scheduled Fourier volume morphing. *Computer Graphics*, 26:43–46, 1992.

[11] B.B. Kimia, A. Tannenbaum, and S.W. Zucker. Shapes, shocks and deformations I. *International Journal of Computer Vision*, To appear, 1994.

[12] R. Kimmel and A.M. Bruckstein. Shape offsets via level sets. *Computer Aided Design*, 23(3):154–162, 1993.

[13] S.G. Mallat. A theory for multiresolution signal decomposition: The wavelet representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(7):674–693, 1989.

[14] F. Mokhatarian and A. Mackworth. Scale-based description of planar curves and two-dimensional shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8:34–43, 1986.

[15] F. Mokhatarian and A. Mackworth. A theory of multiscale, curvature-based shape representation for planar curves. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14:789–805, 1992.

[16] S.J. Osher and J.A. Sethian. Fronts propogation with curvature dependent speed: Algorithms based on Hamilton-Jacobi formulations. *Journal of Computational Physics*, 79:12–49, 1988.

[17] G. Sapiro and A. Tannenbaum. Affine invariant scale-space. *International Journal of Computer Vision*, 11(1):25–44, 1993.

[18] T.W. Sederberg, P. Gao, G. Wang, and H. Mu. 2D shape blending: An intrinsic solution to the vertex path problem. *Computer Graphics*, 27:15–18, 1993.

[19] T.W. Sederberg and E. Greenwood. A physically based approach to 2D shape blending. *Computer Graphics*, 26:25–34, 1992.

[20] M. Shapira and A. Rappoport. Shape blending using the star-skeleton representation. *IEEE Computer Graphics and Applications*, March 1995.

[21] A.P. Witkin. Scale-space filtering. In *Proceedings of Intl. Joint Conf. on AI*, pages 1019–1021, 1983.