Shape Blending of 2–D Piecewise Curves

Thomas W. Sederberg and Eugene Greenwood

Abstract. This paper presents an algorithm for blending (that is, smoothly transforming between) two 2–D shapes bounded by piecewise curves. The algorithm searches for the point correspondence between the two shapes which will minimize the energy required to bend and stretch one shape into the other. The new algorithm runs several times faster than does splitting each curve into five line segments and applying a polygon based shape blend algorithm.

§1. Introduction

Those bend with grace who resist the bending — G. K. Chesterton (from [8])

This paper addresses the problem of how to smoothly transform between two 2–D shapes — the *key* shapes — each defined using piecewise curves. For example, Fig. 1 shows a butterfly transforming into a moth using our algorithm (on the left) and using Adobe Illustrator (on the right). The butterfly is defined using 84 cubic Bézier curves, and the moth involves 45 cubic Bézier curves. The algorithm described in this paper automatically computed the three intermediate shapes shown.

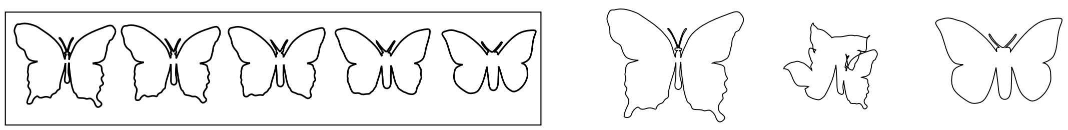

Fig. 1. Butterfly–to–moth blend.

We refer to this task as *shape blending.* Elsewhere it is referred to as shape averaging, shape interpolation, metamorphosis, and morphing, though morphing usually refers to the process of warping digital images, whereas "shape blending" changes the actual curve outline of the shape.

The problem of 2–D shape blending is of widespread interest, and its applications are well chronicled [10,9,7,3,4,2]. Several commercial drawing packages support the capability of blending between two shapes defined by

ISBN xxx.

Bézier curves. No existing algorithm of which we are aware will generally produce pleasing shape blends unless the user manually specifies several "anchor points" which guide the shape blend algorithm.

This paper builds directly on the algorithm in [10], which addresses the problem of 2–D *polygon* shape blending using a work minimization model. If we imagine that the two shapes are formed from pieces of wire, empirical tests suggest that the nicest looking shape blends are generally those that can be performed using the least amount of work in bending and stretching one shape into the other. For example, the blend in Fig. 1 (left) requires 5 units of work and the blend in Fig. 1 (right) requires 37 units of work.

This paper extends the algorithm in [10] to handle shapes bounded by B-spline curves. The algorithm has three main steps:

0. **Preprocess.** Since users of this algorithm are more likely to have data defined in terms of Bézier curves rather than B-spline curves, the first step is to convert those Bézier curves into B-splines. This operation is outlined in Section 4. After conversion, each shape is defined as a single B-spline curve.

1. **Correspondence problem.** The heart of the algorithm is to *insert knots* into the two B-spline curves so that they have the same number of knots in their respective knot vectors. The resulting knot vectors define the pointwise correspondence between the two curves. The knots are inserted so as to minimize the work required to bend and stretch one shape into the other.

2. **Path problem.** Intermediate B-splines in the blend can then be defined by linearly interpolating the control points and knot vectors of the two terminal B-splines. A different approach based on the intrinsic definition of the B-spline control polygon can also be used, a simple adaptation of [11].

Section 2 introduces B-spline shape blending by showing how the shape blending algorithm for polygons in [10] can be re-phrased in terms of degree one B-splines. Section 3 explains why the algorithm uses B-splines instead of Bézier curves, and Section 4 tells how to convert from piecewise Bézier curves to B-spline curves. Section 5 discusses the difference between *plastic* and *elastic* deformations of steel wire, which forms the basis of our work model. Section 6 discusses the graph theory solution.

§2. Linear B-Spline Curves and Polygon Shape Blending

We now show how the polygon shape blending algorithm in [10] can be expressed in terms of linear B-splines.

A linear B-spline is simply a polygon connecting the n control points, and the knot vector specifies a parametrization of the polygon: knot k_i indicates the parameter value at control point $\mathbf{P}_i$. Fig. 2 (left) shows two polygons, which we express in linear B-spline form with control points $\mathbf{P}_1^0 \ldots \mathbf{P}_5^0$ and $\mathbf{P}_1^1 \ldots \mathbf{P}_4^1$ as shown. Define the knot vectors of these two B-splines to be simply $k^0 = [k_1^0, \ldots, k_{n_0}^0] = [1, 3, 5, 7, 8]$ and $k^1 = [k_1^1, \ldots, k_{n_1}^1] = [1, 2, 3, 5]$.

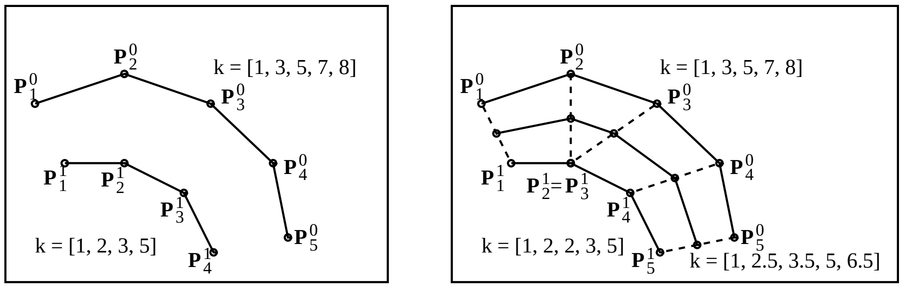

Fig. 2. Polygons as linear B-splines, before and after shape blend.

Then, each B-spline curve is piecewise-linear given by

$$\mathbf{P}^j(t) = \frac{(k^j_{i+1} - t)\mathbf{P}^j_i + (t - k^j_i)\mathbf{P}^j_{i+1}}{k^j_{i+1} - k^j_i}$$

where the index i is chosen to satisfy $k^j_i \le t \le k^j_{i+1}$. Note that two adjacent knots can be identical only if their corresponding control points are identical. For a given set of control points, *any* legal knot vector will produce identically shaped linear B-splines — only the parametrization changes.

The shape blend algorithm in [10] can be viewed as a *knot insertion* problem. For reasons of algorithmic complexity, the shape blend algorithm in [10] only permits new knots to be inserted at values where knots already exist. This amounts to inserting additional vertices into the polygons at points where vertices already exist, thereby creating multiple vertices. For example, Fig. 2 (right) shows the polygons from Fig. 2 (left) after one knot is inserted in $\mathbf{P}^1$ at vertex $\mathbf{P}^1_2$. The goal is to insert enough knots in either or both linear B-spline curves so that they each have the same number of knots (and therefore the same number of vertices), and so that the resulting shape blend between the two linear B-spline curves "looks good". Any two linear B-spline curves with the same number of knots in their knot vectors can be shape-blended by interpolating between the control points and knot vectors of the two initial B-splines. Thus, Fig. 2 (right) shows the result of blending half way between the given linear B-spline curves.

[10] explains where to insert the knots so as to minimize the work required to bend and stretch one shape into the other. This paper extends that algorithm to deal directly with shapes bounded by Bézier or B-spline curves.

§3. Why B-splines?

Cubic polynomial Bézier curves are now standard elements of most commercial drawing packages that support 2–D shape blending, such as Adobe Illustrator [2] and CorrelDraw! [4]. They are also a drawing primitive in the PostScript [1] graphics description language.

One motivation for using B-splines rather than Bézier curves in our shape blending algorithm is that continuity is maintained throughout the shape blend, regardless of the paths travelled by the control points. Fig. 3 (left) shows how two Bézier curves which begin and end with tangent continuity can develop a kink during a shape blending in which the control points follow linear paths. B-splines prevent this problem, as seen in Fig. 3 (right). Here, the two Bézier curves are expressed in B-spline form.

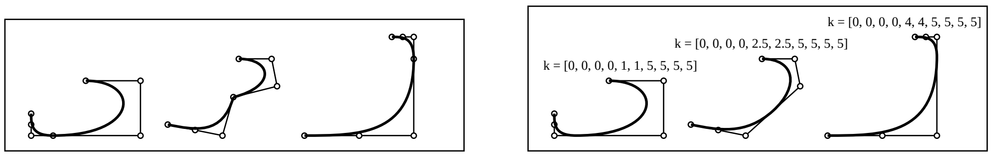
k = [0, 0, 0, 0, 4, 4, 5, 5, 5, 5]
k = [0, 0, 0, 0, 2.5, 2.5, 5, 5, 5, 5]
k = [0, 0, 0, 0, 1, 1, 5, 5, 5, 5]

Fig. 3. Blend-induced kink in Bézier curves, not in B-spline curves.

Another reason for preferring B-splines is that they make bookkeeping much easier. Each shape is defined using a single B-spline, and the correspondence between those two shapes is established with the knot vectors.

Direct conversion is possible both ways between piecewise polynomial cubic Bézier curves and cubic non-uniform polynomial B-spline curves. We assume the reader is familiar with B-spline–to–Bézier conversion; Section 4 outlines Bézier –to–B-spline conversion.

The remaining discussion is "hard wired" for cubic non-uniform polynomial B-splines — the special case of **NURBS** for which the degree is three and for which all control point weights are one. This is done for pedagogical reasons, since it leads to a more concrete presentation. *For the remainder of the paper, the word "B-spline" will mean specifically a cubic non-uniform polynomial B-spline curve unless stated otherwise.*

§4. Combining Bézier curves into a B-spline

Here we suggest how to convert a string of cubic Bézier curves into a single B-spline. The process initializes by assigning the first four B-spline control points to be the control points of the first Bézier curve, and the knot vector is initially $[0, 0, 0, 0, 1, 1, 1, 1]$.

Assume that at some step in this process, the B-spline has a knot vector $[\ldots, k_{i-3}, k_{i-2}, k_{i-1}, k_i, k_{i+1}, k_{i+1}, k_{i+1}, k_{i+1}]$ with $k_{i-2} \le k_{i-1} \le k_i < k_{i+1}$, and the B-spline control points are labelled

$$\mathbf{P}_1, \ldots, \mathbf{P}_{n-3}, \quad \mathbf{P}_{n-2}, \quad \mathbf{P}_{n-1}, \quad \mathbf{P}_n.$$

The control points of the Bézier curve to be appended are

$$\mathbf{Q}_0 = \mathbf{P}_n, \quad \mathbf{Q}_1, \quad \mathbf{Q}_2, \quad \mathbf{Q}_3.$$

We first determine the continuity between the B-spline and Bézier curve. C^0 continuity occurs if control points $\mathbf{P}_{n-1}$, $\mathbf{P}_n$, and $\mathbf{Q}_1$ are not collinear. If they are collinear, then the value of knot e is chosen so as to satisfy

$$|[\mathbf{P}_n - \mathbf{P}_{n-1}](k_{i+1} - k_i) - [\mathbf{Q}_1 - \mathbf{P}_n](e - k_{i+1})| < TOL.$$

This provides for C^1 (not merely G^1) continuity. TOL is a small number which is needed to account for floating point error. An appropriate value for TOL is the width of the reverse map of a pixel into world space.

C^2 continuity occurs if, in addition to C^1 continuity, the relationship

$$\begin{aligned}|(\mathbf{P}_{n-2} - \mathbf{Q}_2)(k_{i+1} - k_{i-1})(k_{i+1} - e) + (\mathbf{P}_{n-1} - \mathbf{P}_{n-2})(e - k_{i-1})(k_{i+1} - e)\\ + (\mathbf{Q}_2 - \mathbf{Q}_1)(k_i - e)(k_{i+1} - k_{i-1})| < TOL\end{aligned}$$

is satisfied. Defining

$$\mathbf{P}_\alpha = \frac{(k_{i+1} - e)\mathbf{P}_{n-2} + (e - k_{i-1})\mathbf{P}_{n-1}}{k_{i+1} - k_{i-1}} = \frac{(k_{i+1} - k_i)\mathbf{Q}_2 + (e - k_{i+1})\mathbf{Q}_1}{e - k_i},$$

C^3 continuity occurs if, further, the relationship

$$\left|\mathbf{P}_\alpha - \frac{(e - k_{i+1})\mathbf{P}_\beta + (k_{i+1} - k_{i-1})\mathbf{P}_\gamma}{e - k_{i-1}}\right| < TOL$$

is satisfied, where

$$\mathbf{P}_\beta = \frac{(e - k_{i-2})\mathbf{P}_{n-2} + (k_{i-1} - e)\mathbf{P}_{n-3}}{k_{i-1} - k_{i-2}}$$

and

$$\mathbf{P}_\gamma = \frac{(k_{i+1} - k_i)\mathbf{Q}_3 + (k_i - e)\mathbf{Q}_2}{k_{i+1} - e}$$

Then, the B-spline after appending the Bézier becomes

C	**Knot Vector**	**Control Points**
C^0	$[\ldots, k_{i-2}, k_{i-1}, k_i, k_{i+1}, k_{i+1}, k_{i+1}, e, e, e, e]$	$\ldots, \mathbf{P}_{n-1}, \mathbf{P}_n, \mathbf{Q}_1, \mathbf{Q}_2, \mathbf{Q}_3$
C^1	$[\ldots, k_{i-2}, k_{i-1}, k_i, k_{i+1}, k_{i+1}, e, e, e, e]$	$\ldots, \mathbf{P}_{n-2}, \mathbf{P}_{n-1}, \mathbf{Q}_1, \mathbf{Q}_2, \mathbf{Q}_3$
C^2	$[\ldots, k_{i-2}, k_{i-1}, k_i, k_{i+1}, e, e, e, e]$	$\ldots, \mathbf{P}_{n-3}, \mathbf{P}_{n-2}, \mathbf{P}_\alpha, \mathbf{Q}_2, \mathbf{Q}_3$
C^3	$[\ldots, k_{i-2}, k_{i-1}, k_i, e, e, e, e]$	$\ldots, \mathbf{P}_{n-3}, \mathbf{P}_\beta, \mathbf{P}_\gamma, \mathbf{Q}_3$

§5. Stress, Strain, and Work

This section explains the model used in assessing how much work is required to bend and stretch one B-spline shaped wire into another, based on the work equations for small displacements of a piece of steel wire.

Any material behaves like a very stiff spring — the harder you pull on it, the more it stretches. A force-displacement diagram can be plotted to show how a piece of wire elongates when it is stretched by a force. The data values are typically expressed in terms of *stress* $\sigma = \frac{F}{A}$ (force divided by the cross sectional area of the wire) and *strain* $\epsilon = \frac{\delta}{L}$ (elongation divided by the

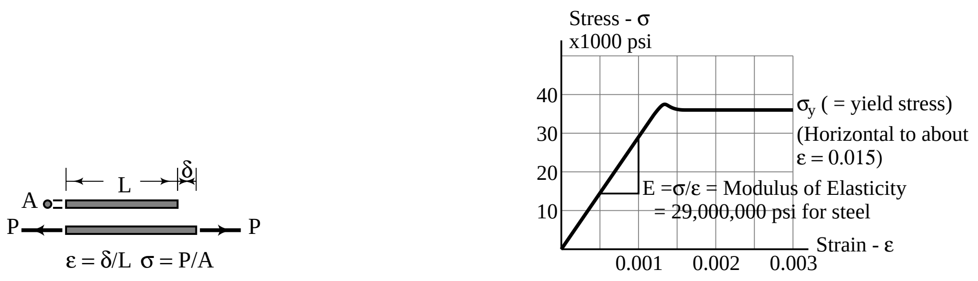

Fig. 4. Stress–Strain Diagram for Steel.

initial length of the wire). For a given material, the stress–strain diagram is independent of the length or cross section of the specimen wire.

As seen in Fig. 4 (right), steel exhibits a linear stress–strain relationship until it experiences *yield stress* [6]. If a piece of steel wire supports a weight W such that $W/A = 29,000$ psi (pounds per square inch), the wire will elongate about 0.1% of its total length (as deduced from Fig. 4). If that weight is increased so that $W/A \approx 38,000$ psi, (so that the yield stress is reached), the wire will suddenly stretch about 1.5% of its initial length. *Elastic* stretching refers to stresses below the yield stress. If the yield stress is exceeded, the wire undergoes *plastic* stretching.

The work per unit volume $\frac{W_s}{AL}$ consumed in stretching a piece of wire is simply the area under the stress-strain curve. For elastic stretching, we have the stretching work W_s is

$$W_s = \left|\frac{1}{2}\sigma\epsilon AL\right| = \left|c_{es}\frac{\delta^2}{L}\right|. \tag{1}$$

with $c_{es} = \frac{1}{2}EA$. For plastic stretching, we have

$$W_s \approx |\sigma_y \epsilon AL| = c_{ps}|\delta| \tag{2}$$

with $c_{ps} = \sigma_y A$.

Equation (1) has proven most effective than (2) for shape blending. If we are trying to minimize the total work, (2) is much less discriminating since if the initial shape is shorter than the final shape, *any* correspondence between them will result in the same amount of stretching work, so long as no differential segment shrinks during the shape blend.

Of course, these equations for work are physically accurate only for small displacements (say $\epsilon < 0.015$). Beyond that, the stress–strain relationship is non-linear. For low-carbon steel, fracture occurs at about $\epsilon \approx 0.3$. Also, the stress-strain diagram in Fig. 4, with the horizontal segment, is unique to steel. Furthermore, we use these equations for wires under compression as well as tension, which clearly is invalid since a wire will buckle under very little force. Nonetheless, our goal is not to accurately model wire shapes, but to look for clues on how to measure shape blends. On that basis, experience justifies (1).

Bending resistance is measured in terms of *bending moment* M, (or, *torque*) with units of force times distance. If you apply a bending moment at each end of a piece of wire so as to bend it into a smiling shape, the top portion of the wire will experience compression stress and the bottom portion experiences tension stress. These stresses increase linearly, proportionate to the distance from the center of the wire as shown in Fig. 5(left). As long as the maximum stress in the wire is less than the yield stress, the wire is said to undergo *elastic bending.* For small displacements, the shape of the wire will be a circular arc with curvature $\kappa = \frac{M}{EI}$ where I is the area moment of inertia. For a wire with circular cross section of radius r, $I = \frac{\pi r^4}{64}$.

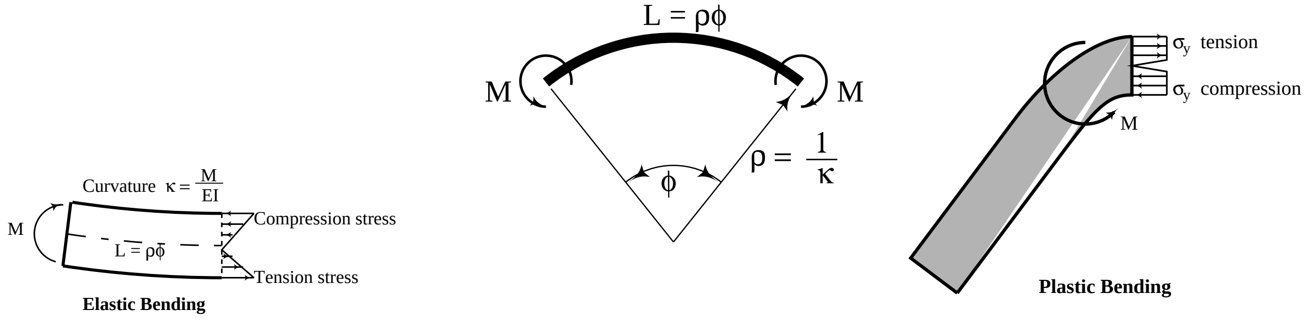

Fig. 5. Stress Distribution for Elastic and Plastic Bending.

The work required to impart an elastic bend, by bending a straight wire of length L into a circular arc of curvature κ (see Fig. 5— middle) is

$$W_e = \int M\, d\phi = \frac{1}{2} M \phi^2.$$

The work required to impart a plastic bend (kink) in a piece of initially straight wire is approximated by the equation

$$W_p = M_p \phi$$

where ϕ is the angle of the bend and M_p is the *plastic moment.* When a wire bends sharply, the material in the vicinity of the bend reaches its yield stress (Fig. 5— right). Then, its capacity to resist further bending is limited to the plastic moment, which remains fairly constant as bending continues.

In summary, three distinct work quantities must be addressed in our model: **stretching work** W_s, **arc bending work** W_b, and **"kinking" work**, W_k (or, the work involved in imparting or changing a C^0 bend).

Fig. 6 shows a generic segment of wire before and after bending and stretching. The work equations involved are:

$$W_s = c_s \frac{(L_2 - L_1)^2}{L_1 + L_2} \tag{3}$$

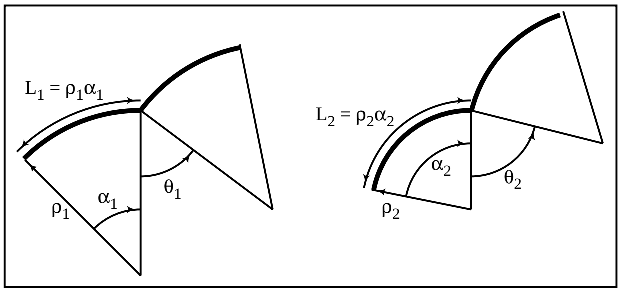

Fig. 6. Stretching, Bending, and Kinking.

$$W_b = c_b \frac{(\alpha_2 - \alpha_1)^2}{L_1 + L_2} \tag{4}$$

$$W_k = c_k \left|\theta_2 - \theta_1\right|^{e_k} \tag{5}$$

Note that stretching and bending are uncoupled: if an arc undergoes stretching work but not bending work, L changes but α and θ stay the same. However, a change in L *does* change the radius of curvature ρ. This is appropriate, since if one shape is a simple scale of another, the transformation is arrived at by a pure stretching. Thus, bending alters the local shape, while stretching changes the local size.

In practice, shape blend problems tend to be somewhat indifferent to what values are chosen for c_b, c_k and c_s. For example, similar results for the shape blend in Fig. 1 can be obtained with widely varying values for the work coefficients. For example, with $c_b = 1$, $c_k = 4$, and $e_k = 1$, $0.01 \leq c_s \leq 16$ provides the same good result as $c_s = 1$, $c_k = 4$, and $c_b = 1$, $0.00001 \leq e_k \leq 10$.

There do exist shape blend problems for which no single set of work coefficients can provide the desired shape blend. A simple, provable example of such a case is presented in Figure 26 of [10]. In such cases, the best option is probably to invite the user to specify a few anchor points to guide the shape blend.

§6. Graphs and Knots

The heart of the shape blend algorithm is the search for the minimum work solution, which is solved using graph theory in a manner very similar to that in [10]. A more thorough discussion of the pertinent principles of graph theory is found in [5]. Due to space limitations, here we give just enough background on the graph to convey the modifications to [10] needed to support B-splines.

The algorithm works best if the Bézier segments of a B-spline each have fairly uniform curvature. Experience suggests that it suffices to simply limit the arc length of all Bézier segments to about 5% of the width of a box bounding the B-spline. This limit is enforced by inserting knots as needed.

A graph is defined with rows corresponding to distinct knot values in B-spline 1 and columns corresponding to distinct knot values in B-spline 2. By distinct we mean that a multiple knot corresponds to a single row or column.

The points at which rows and columns on the graph intersect are called graph vertices.

The correspondence between the two B-splines is represented by a piecewise curve (a *path*) running from the lower left corner of the graph to the upper right corner. *All* possible correspondences between the two key B-splines can be thus represented. Our task is to identifying which of all possible such paths requires the least amount of blending work, where work is defined by equations (3)–(5).

It is unreasonable to seek the *exact* least-work path, since it is a complicated curve whose description is rooted in the calculus of variations. It can only be approximated using non-linear numerical optimization algorithms which are unacceptably slow for problems of this size, and cannot assure a global optimum anyway. Instead, we use the piecewise linear approximation to the least-work path presented in [10], for which the guaranteed least work solution can be determined in $O(r + c)$ time, where $r + c$ is the number of rows and columns in the graph.

The graph theory solution we use is virtually identical to that in [10] with two differences. First, the work for mapping one curve segment to another involves the work component W_b (4). Also, once the least-work path is determined, it must be translated into knot insertions to actually realize the shape blend. This is done by examining the graph vertices through which the least-work path passes. If at any such graph vertex the row (or column) knot multiplicity is less than the column (or row) multiplicity, enough knots are inserted in the row (or column) so that the multiplicities are the same. At the conclusion of this knot insertion procedure, both B-splines have the same number of knots and control points.

§7. Examples and Discussion

Fig. 7 shows the Bézier font outline of an upper case G blending from Adobe 64 point Bold Courier font to 64 point Bold Times font. The Courier outline consists of 12 line segments and 10 cubic Bézier curves, and the Times outline comprises 8 line segments and 10 cubic Bézier curves. Our algorithm computed the three intermediate shapes. The shape blend algorithm converted both outlines to B-splines with 62 knots each (16 triple knots and 7 double knots), each consisting of 22 non-degenerate Bézier curves.

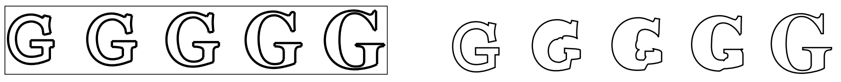

Fig. 7. Shape blend example.

The shape blend in Fig. 7 (right) is typical of those produced by commercial packages, in which a single pair of anchor points was specified.

The butterfly-moth blend in Fig. 1 required 3 seconds to compute on an HP 730 workstation. We converted the Bézier outlines for those shapes into polygons of 225 vertices for the moth and 420 for the butterfly. Using the algorithm in [10], the polygon data took 29 seconds to compute. Enhancements in [10] can speed up both algorithms by a factor of five or more.

References

1. Adobe Systems, Inc. *PostScript Language Reference Manual*, 1989.
2. Adobe Systems, Inc., Silicon Valley. *Adobe Illustrator*, Next Version, Edition 3, 1991.
3. Catmull E., The problems of computer-assisted animation, Computer Graphics, **12** (1978) 348–353.
4. Corel Systems Corporation, Ottowa, Canada. *CorelDraw! 2.0*, 1990.
5. Fuchs, H., Z. M. Kedem, and S. P. Uselton, Optimal surface reconstruction from planar contours, Comm. ACM **20** (1977), 693–702.
6. Higdon, A., E. H. Ohlsen, W. B. Stiles, J. A. Weese, and W. F. Riley, *Mechanics of Materials*, John Wiley & Sons, Inc., New York, 1976.
7. Kaul, A. and J. Rossignac, Solid-interpolating deformations: Construction and animation of PIPs, in *Proc. Eurographics '91*, F.H. Post and W. Barth, (eds.), Elsevier Science Publishers B.V., 1991, 493–505.
8. Mehlum, E., Nonlinear splines, in *Computer Aided Geometric Design*, R. E. Barnhill and R. F. Riesenfeld, (eds.), Acedemic Press, San Diego, 1974, 173–207.
9. Reeves, W. T., Inbetweening for computer animation utilizing moving point constraints. *Computer Grapics (Proc. SIGGRAPH)* **15** (1981) 263–269.
10. Sederberg, T. W. and E. Greenwood. A physically based approach to 2–d shape blending. *Computer Graphics (Proc. SIGGRAPH)* **26** (1992) 25–34.
11. Sederberg, T. W., P. Gao, G. Wang, and H. Mu, 2–D shape blending: An intrinsic solution to the vertex path problem, *Computer Graphics (Proc. SIGGRAPH)* **27** (1993) 15–18.

Acknowledgements. This work was supported under NSF grant DMC-8657057 and by ONR under grant number N000-14-92-J-4064. Guojin Wang, Hong Mu, and Peisheng Gao served as an enthusiastic support team.

Thomas W. Sederberg
Engineering Computer Graphics Laboratory
Brigham Young University
Provo, UT 84602
tom@byu.edu

Eugene Greenwood
WordPerfect, The Novell Applications Group
1555 N. Technology Way
Orem, UT 84057
EUGENEG@WordPerfect.Com