Shape Blending of 2–D Piecewise Curves

Thomas W. Sederberg and Eugene Greenwood
Engineering Computer Graphics Laboratory
Brigham Young University

April 20, 1999

Abstract

This paper presents an algorithm for blending (that is, smoothly transforming between) two 2–D shapes bounded by piecewise curves. The approach builds on that of an earlier paper on 2–D shape blending [SG92], which deals with shape blending of polygons. 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 physics of plastic and elastic bending and stretching of wire is used as a model.

The new algorithm runs typically ten times faster than does splitting each curve into five line segments and applying the earlier polygon based shape blend algorithm.

1 Introduction

Those bend with grace who resist the bending — G. K. Chesterton (quoted in [Meh74]).

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, Figure 1 shows a butterfly transforming into a moth. The butterfly on the left is defined using 84 cubic Bézier curves, and the moth on the right involves 45 cubic Bézier curves. The algorithm described in this paper automatically computed the three intermediate shapes shown.

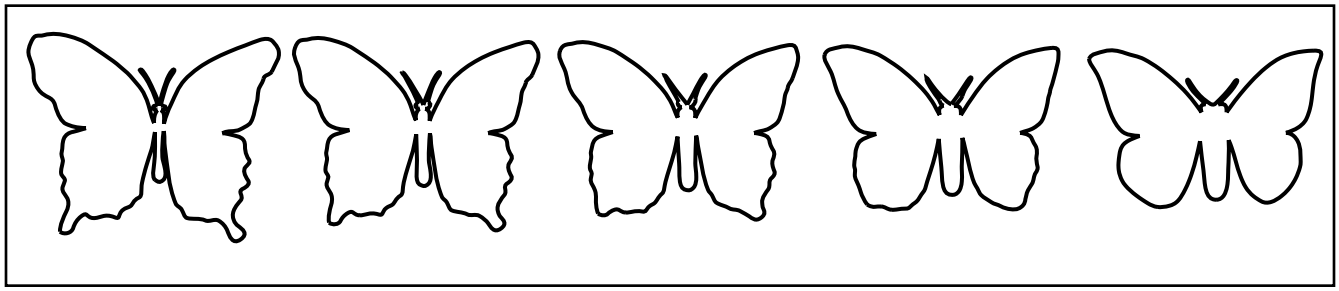

Figure 1: Moth–to–butterfly blend

We refer to this task as *shape blending*. Shape blending as discussed in this paper is sometimes referred to as shape averaging, shape interpolation, metamorphosis, and morphing, though morphing often refers to the process of warping digital images, whereas we use "shape blending" to mean changing the actual curve outline of the shape.

The problem of 2–D shape blending is of widespread interest, and its applications are well chronicled [SG92, Ree81, KR91, US90, Cat78, Cor90, Ado91]. Several commercial drawing packages support the capability of blending between two shapes defined by Bézier curves. To our knowledge, all such algorithms generally will not produce pleasing shape blends unless the user manually specifies several "anchor points" which guide the shape blend algorithm. The shape blend in Figure 2 shows a typical result from one such

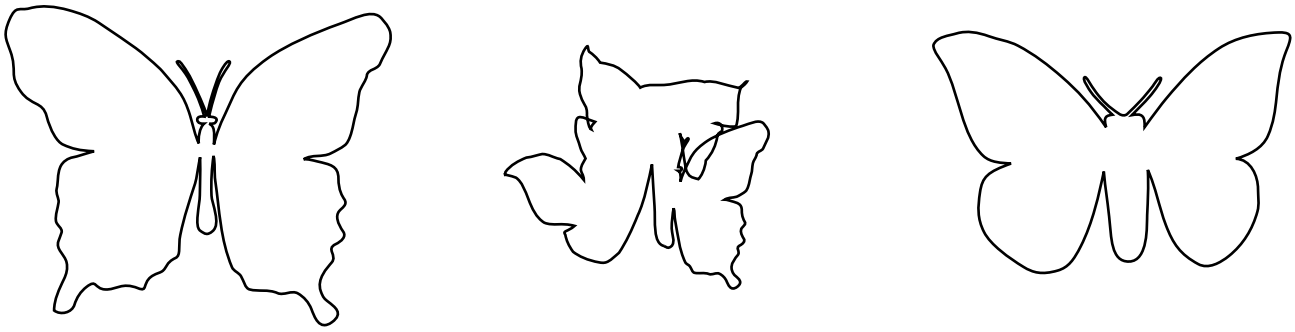

Figure 2: Shape blend using commercial software

commercial package, in which a single pair of anchor points was specified. If a dozen pairs of anchor points are specified by the user, this commercial algorithm can be manually guided to produce a shape blend approaching that in Figure 1.

This paper builds directly on the algorithm in [SG92], which contains a solution to the problem of 2–D *polygon* shape blending based on 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, while the blend in Figure 1 requires 5 units of work, the blend in Figure 2 requires 37 units of work.

Shape blending consist of two distinct subproblems: the correspondence problem, and the path problem. This paper is primarily concerned with the correspondence problem, or determining where each point on one key shape will travel to on the other key shape. The path problem amounts to determining the trajectory of each point as it travels between key shapes. An intermediate shape is simply the collection of all such points, at a given instant of time, as they travel their respective paths.

While [SG92] deals with polygonal shapes, this paper addresses shape blending for objects 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 fairly standard operation is outlined in section 3.4. After this 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-splines so that they have the same number of knots in their respective knot vectors. The resulting knot vectors define the correspondence between the two curves. The knots are inserted in such a way that the work to bend and stretch one shape into the other is minimized.

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. An approach based on the intrinsic definition of the B-spline control polygon can also be used, as discussed in Section 6.

The reader is assumed to be familiar with cubic Bézier curves. Section 2 introduces B-spline shape blending by showing how the shape blending algorithm for polygons in [SG92] can be re-phrased in terms of degree one B-splines. Cubic non-uniform B-splines are reviewed in Section 3.

This shape blend algorithm for curves requires us to look more closely at the physics of wires for a work model. In particular, we benefit from paying attention to the difference between *plastic* and *elastic* deformations. The work model is discussed in section 4.

The search for the least work solution in [SG92] resorted to a well-known technique from graph theory. This paper presents a more general graph search strategy which can diminish the number of cases where an entire curve segment on one key shape collapses to a point on the other key shape. Section 5 discusses this

graph theory solution.

2 Linear B-Splines and Polygon Shape Blending

Our algorithm is best introduced by showing how the polygon shape blending algorithm in [SG92] can be expressed in terms of degree one B-splines. A degree one B-spline is defined by specifying n control points

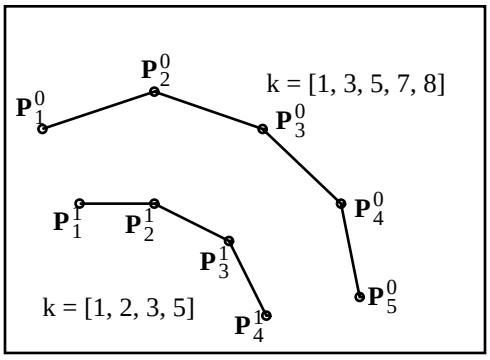

Figure 3: Polygons as linear B-splines

$\mathbf{P}_1 \dots \mathbf{P}_n$ and n *knot values*. The knot values $[k_1 \dots k_n]$ are a non-decreasing sequence of real numbers, and collectively they are called a *knot vector*. (Mathematical papers dealing with B-splines usually define the knot vector for linear B-splines as consisting of $n+2$ knots, $[k_0 \dots k_{n+1}]$. However, the knots k_0 and k_{n+1} have absolutely no influence on the linear B-spline and would only add "noise" to our discussion.)

A linear B-spline can be thought of simply as 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 $\mathbf{P}_i$. Figure 3 shows two polygons, which we express as linear B-splines with control points $\mathbf{P}_1^0 \dots \mathbf{P}_5^0$ and $\mathbf{P}_1^1 \dots \mathbf{P}_4^1$ as shown. Define the knot vectors of these two B-splines to be simply $k^0 = [k_1^0, \dots, k_{n_0}^0] = [1, 3, 5, 7, 8]$ and $k^1 = [k_1^1, \dots, k_{n_1}^1] = [1, 2, 3, 5]$. Then, each B-spline is a continuous piecewise-linear curve

$$\mathbf{P}^j(t) = \frac{(k_{i+1}^j - t)\mathbf{P}_i^j + (t - k_i^j)\mathbf{P}_{i+1}^j}{k_{i+1}^j - k_i^j}$$

where the index i is chosen to satisfy $k_i^j \le t \le k_{i+1}^j$. 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 [SG92] can be viewed as a *knot insertion* problem. For a linear B-spline, a single knot insertion simply amounts to placing a new control point anywhere along the polygon, then adding a new knot value at the appropriate spot in the knot vector so as to maintain numerical order. The new knot value is the parameter value corresponding to the location of the new control point. The resulting B-spline, consisting of $n+1$ control points and knots, is clearly identical to the B-spline before knot insertion.

For reasons of algorithmic complexity, the shape blend algorithm in [SG92] 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, Figure 4 shows the polygons from Figure 3 after one knot is inserted in $\mathbf{P}^1$ at vertex $\mathbf{P}_2^1$. The goal is to insert enough knots in either or both linear B-splines 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-splines "looks good". Any two linear B-splines 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, Figure 4 shows the result of blending half way between the given linear B-splines.

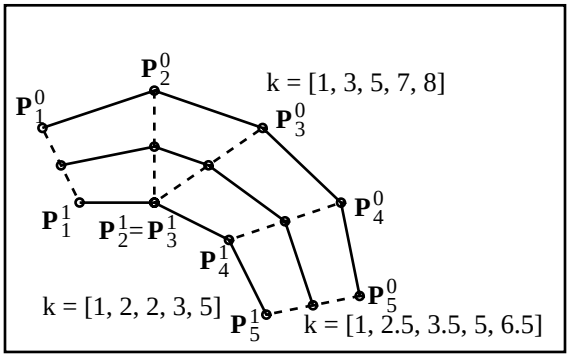

Figure 4: Polygon shape blend using linear B-splines

[SG92] 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 B-splines

This section reviews the pertinent properties of degree three B-splines. We assume that the typical reader of this paper will be familiar with Bézier curves, but not so comfortable with B-splines. Even though B-splines are a standard entity in computer graphics, they do not enjoy the widespread use among graphics practitioners that Bézier curves do.

Therefore, this section presents in simple form how to break a B-spline up into its constituent Bézier curves, how to combine a set of piecewise Bézier curves into a single B-spline, and how to insert a knot. For readers who would like some deeper background, we recommend Ramshaw's explanation of B-splines using an approach called polarization [Ram87, Ram88]. An excellent, thorough treatment of B-splines using traditional approaches is [RBB87].

The remaining discussion is "hard wired" for cubic non-uniform polynomial B-splines — the special case of **NURBS** (**N**on-**U**niform **R**ational **B**-**S**plines) for which the degree is three (or, in traditional B-spline jargon, the *order* is four) 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 unless stated otherwise.* Interested readers will find no problem extending this discussion to B-splines of any degree, or to the rational case (NURBS).

The authors have implemented a B-spline shape blending function in the PostScript language. The user provides two sets of B-spline control points and knot vectors with the same number of knots, and it will draw the blended B-spline. We can email this function to interested readers.

3.1 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 [Ado91] and CorrelDraw! [Cor90]. They are also a drawing primitive in the PostScript [Ado89] graphics description language, so an explanation of why we have chosen to base our algorithm on B-splines rather than on the more familiar piecewise Bézier curves is in order.

One motivation for using B-splines rather than Bézier curves is that continuity is maintained throughout the shape blend, regardless of the paths travelled by the control points. Figure 5 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 Figure 6. Here, the initial

and final two Bézier curves are expressed as B-splines. The conversion from piecewise Bézier to B-spline is discussed in Section 3.4. The knot vectors k are reviewed in Section 3.2.

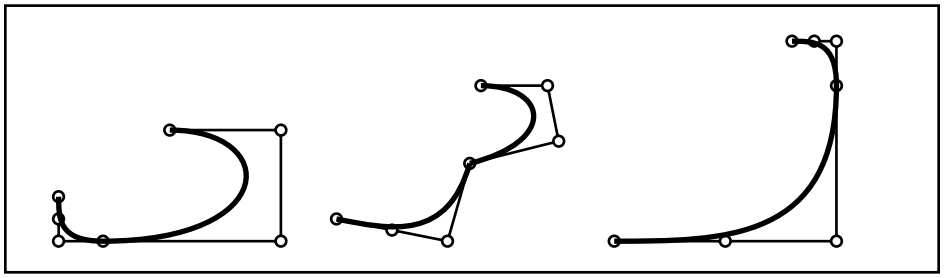

Figure 5: Blend-induced kink

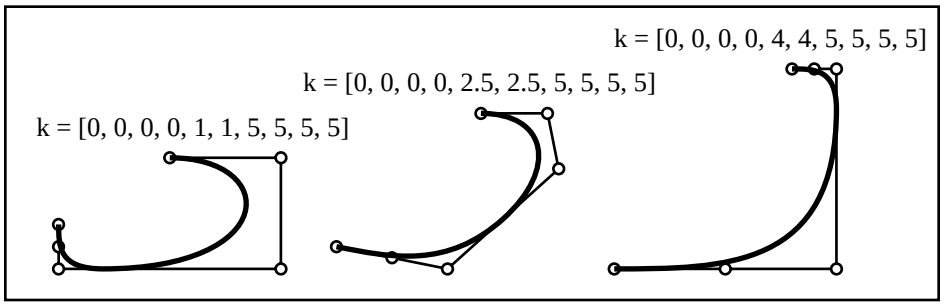

Figure 6: No kink with B-splines

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

As reviewed in Sections 3.2 and 3.4, direct conversion is possible both ways between piecewise polynomial cubic Bézier curves and cubic non-uniform polynomial B-spline curves.

3.2 Splitting a B-spline into Bézier Curves

B-splines are defined by specifying $n > 3$ *control points* $\mathbf{P}_1, \ldots, \mathbf{P}_n$, and a *knot vector* $[k_{-2}, \ldots, k_{n+1}]$, which is a sequence of non-decreasing real numbers.

It is helpful to think of B-splines as primarily a technique for connecting together a string of cubic Bézier curves such that neighboring curves are automatically C^2 continuous. Every B-spline with n control points can be decomposed into $n-3$ cubic Bézier curves. Conventionally, Bézier curves use the parameter range $0 \le t \le 1$. For the i^{th} Bézier curve in a B-spline, the parameter range is $k_i \le t \le k_{i+1}$. As far as the *appearance* of a Bézier curve is concerned, for four given control points, a Bézier curve defined over the range $k_i \le t \le k_{i+1}$ looks identical to one defined over the parameter interval $0 \le t \le 1$. So if one knows how to plot a cubic Bézier curve, all one needs to know to plot a B-spline is how to extract the control points of each of the $n-3$ Bézier curves comprising it.

Figure 7 shows the i^{th} Bézier curve (whose control points are labelled $\mathbf{Q}_0$, $\mathbf{Q}_1$, $\mathbf{Q}_2$, $\mathbf{Q}_3$) of a sample B-spline (whose control points are labelled $\mathbf{P}_{i-1} \ldots \mathbf{P}_{i+4}$). The Bézier control points are obtained using the formulas:

$$\mathbf{Q}_1 = \frac{(k_{i+2} - k_i)\mathbf{P}_{i+1} + (k_i - k_{i-1})\mathbf{P}_{i+2}}{k_{i+2} - k_{i-1}} \tag{1}$$

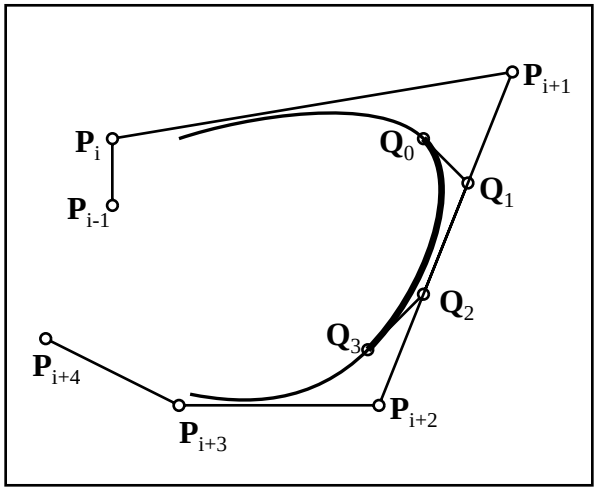

Figure 7: Extracting a Bézier curve from a B-spline

$$\mathbf{Q}_0 = \frac{k_{i+1}-k_i}{k_{i+1}-k_{i-1}} \frac{(k_{i+1}-k_i)\mathbf{P}_i + (k_i-k_{i-2})\mathbf{P}_{i+1}}{k_{i+1}-k_{i-2}} + \frac{k_i-k_{i-1}}{k_{i+1}-k_{i-1}}\mathbf{Q}_1 \tag{2}$$

$$\mathbf{Q}_2 = \frac{(k_{i+2}-k_{i+1})\mathbf{P}_{i+1} + (k_{i+1}-k_{i-1})\mathbf{P}_{i+2}}{k_{i+2}-k_{i-1}} \tag{3}$$

$$\mathbf{Q}_3 = \frac{k_{i+1}-k_i}{k_{i+2}-k_i} \frac{(k_{i+3}-k_{i+1})\mathbf{P}_{i+2} + (k_{i+1}-k_i)\mathbf{P}_{i+3}}{k_{i+3}-k_i} + \frac{k_{i+2}-k_{i+1}}{k_{i+2}-k_i}\mathbf{Q}_2 \tag{4}$$

Earlier in this section, we made two general statements that we must now modify slightly. Notice from equations 1–4 that if $k_i = k_{i+1}$, Bézier curve i collapses to a single point:

$$\mathbf{Q}_0 = \mathbf{Q}_1 = \mathbf{Q}_2 = \mathbf{Q}_3 = \frac{(k_{i+2}-k_i)\mathbf{P}_{i+1} + (k_i-k_{i-1})\mathbf{P}_{i+2}}{k_{i+2}-k_{i-1}} \tag{5}$$

Thus, a B-spline with n control points can always be thought of as being made up of $n-3$ Bézier curves, but some of those curves might be degenerate (zero length). Also, the continuity between any two adjacent Bézier curves will be C^2 only if neither of them is degenerate. If $k_i < k_{i+1} = k_{i+2} < k_{i+3}$, Bézier curves i and $i+2$ will generally have C^1 continuity, and if $k_i < k_{i+1} = k_{i+2} = k_{i+3} < k_{i+4}$, Bézier curves i and $i+3$ will generally have C^0 continuity.

3.3 Knot Insertion

Knot insertion is a fundamental operation on B-splines that consists of adding a knot to the knot vector, then replacing two control points with three in such a way that the appearance of the B-spline is not changed. This has the effect of splitting one of the component Bézier curves into two pieces. To insert a new knot k_j, first find where it fits in the knot vector by locating index i such that

$$k_i \le k_j \le k_{i+1}.$$

Then, replace control points $\mathbf{P}_{i+1}$ and $\mathbf{P}_{i+2}$ with the three control points

$$\mathbf{P}_A = \frac{(k_{i+1}-k_j)\mathbf{P}_i + (k_j-k_{i-2})\mathbf{P}_{i+1}}{k_{i+1}-k_{i-2}} \tag{6}$$

$$\mathbf{P}_B = \frac{(k_{i+2}-k_j)\mathbf{P}_{i+1} + (k_j-k_{i-1})\mathbf{P}_{i+2}}{k_{i+2}-k_{i-1}} \tag{7}$$

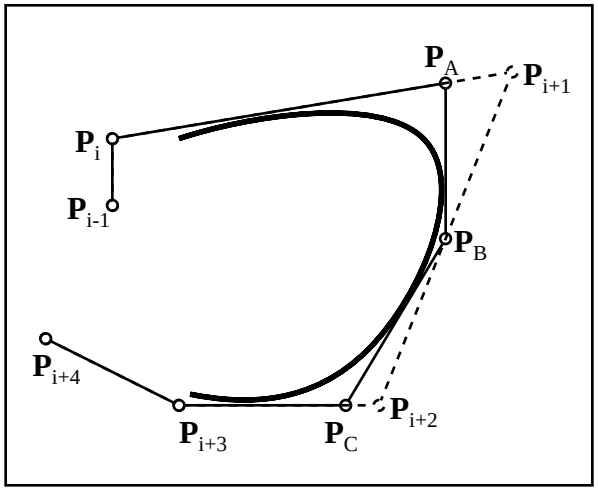

Figure 8: Knot insertion

$$\mathbf{P}_C = \frac{(k_{i+3} - k_j)\mathbf{P}_{i+2} + (k_j - k_i)\mathbf{P}_{i+3}}{k_{i+3} - k_i}. \tag{8}$$

Of course, this requires renumbering of the control points $\mathbf{P}_j$, $j > i+2$. The B-spline before knot insertion is identical to the one after knot insertion, except that the later has one additional knot and control point.

3.4 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 $[0,0,0,0,1,1,1,1]$.

Thereafter, each subsequent Bézier curve is analyzed to determine what order of continuity exists between it and the current B-spline, and it is appended to the B-spline as follows. Assume that at some step in this process, the B-spline has a knot vector $[k_{i-3}, k_{i-2}, k_{i-1}, k_i, k_{i+1}, k_{i+1}, k_{i+1}, k_{i+1}]$ 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

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

Then, depending on the continuity order between the B-spline and the Bézier , the B-spline after appending the Bézier becomes

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

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

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

is satisfied. We can then compute

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

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

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

4 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. These formulas are based on the work equations for small displacements of a piece of steel wire.

4.1 Work Due to Stretching

Any material behaves like a very stiff spring — the harder you pull on it, the more it stretches. In a physics class, the deflection of a spring is often plotted as a function of the force exerted on the spring. Such a force-displacement diagram is linear for a typical spring.

A force-displacement diagram can also be plotted to show how a piece of, say, steel wire elongates when it is stretched by a force. In this case, 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 initial length of the wire). Stress and strain are illustrated in Figure 9. The same basic stress–strain diagram will

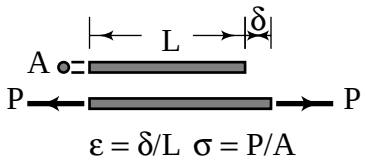

Figure 9: Stress and Strain

be obtained experimentally for any piece of steel wire, regardless of its length or cross section. Stress is measured in units of force per area. Strain is dimensionless.

As seen in Figure 10 (taken from [HOS+76]), steel exhibits a linear stress–strain behavior until it experiences *yield stress*. The slope of the stress–strain diagram is referred to as the *modulus of elasticity* and is denoted by an upper case E. 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 Figure 10). 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 is said to undergo *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

$$W_s = \left|\frac{1}{2}\sigma\epsilon AL\right| = \left|c_{es}\frac{\delta^2}{L}\right|. \tag{9}$$

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

$$W_s \approx |\sigma_y \epsilon AL| = c_{ps}|\delta| \tag{10}$$

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

Equation 9 has proven most effective for shape blending. If we are trying to minimize the total work, equation 10 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 Figure 10, 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 equation 9.

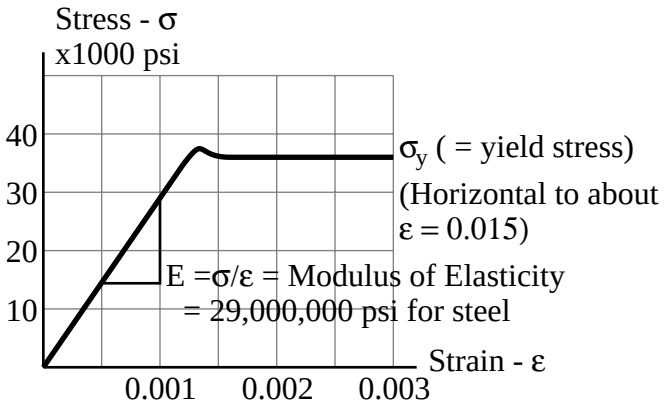

Figure 10: Stress–Strain Diagram for Steel

4.2 Stress in Bending

Bending effort is measured in terms of *bending moment* M, (or, *torque*) with units of force times distance. Imagine you are holding a piece of wire and you apply a bending moment at each end of the wire so as to bend it into a smiling shape. This causes the top portion of the wire to experience compression stress and

the bottom portion to experience tension stress, as shown in Figure 11. This same basic stress distribution occurs when a yardstick, a concrete highway bridge, or any other relatively thin object experiences bending. As long as the stress 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, $I = \frac{\pi r^4}{64}$.

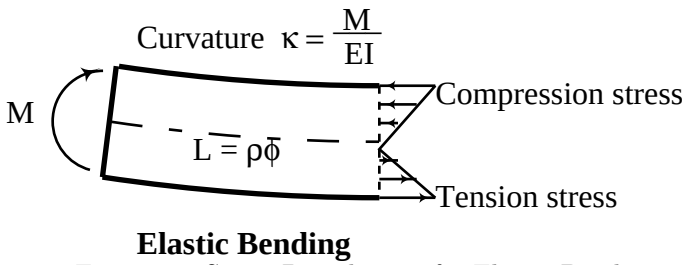

Figure 11: Stress Distribution for Elastic Bending

The work required to impart an elastic bend, by bending a straight wire of length L into a circular arc of curvature κ (see Figure 12) is

$$W_e = \int M \, d\phi = M\phi. \tag{11}$$

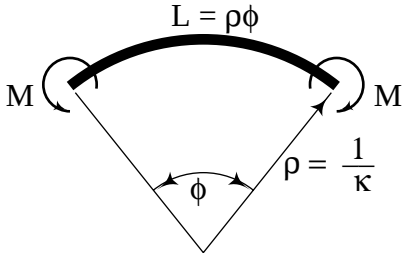

Figure 12: Curvature

To kink the wire (that is, create a sharp C^0 bend in it), one normally slides one's hands along the wire so that thumbs touch, and bend vigorously. In this case, the bending stress increases until yield stress is reached, as shown in Figure 13. When this happens, the angle of bending can increase without much increase in the applied moment M. This type of bending is known as *plastic bending*.

Given two circular arc sections, C_1 and C_2 with arc lengths L_1 and L_2 and curvatures κ_1 and κ_2, the work required to stretch one wire arc so that it has the same length as the other is

$$W_s = k_s \frac{(L_1 - L_2)^2}{L_1 + L_2}. \tag{12}$$

where k_s is a user defineable constant with units of force. For an actual wire, $k_s = AE$ where A is the cross sectional area of the wire and E is the modulus of elasticity of the material the wire is made of. For steel, $E = 29{,}000{,}000$ lbs./in^2.

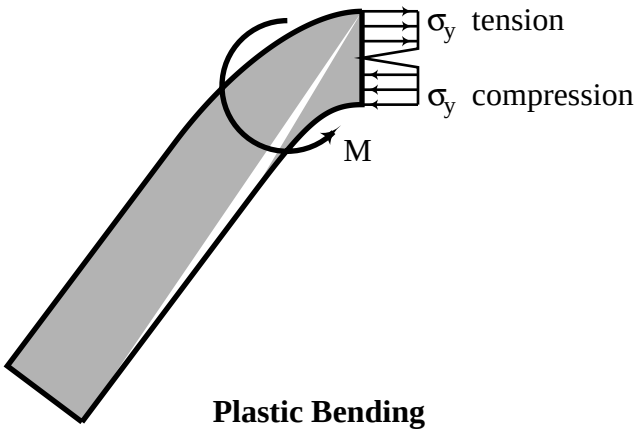

Figure 13: Stress Distribution for Plastic Bending

To change the angle of bend in a circular arc, the work required to perform this elastic bending is

$$W_e = \frac{k_e \phi^2}{L_0 + L_1} = k_e(\kappa_1 - \kappa_2)^2(L_1 + L_2) \tag{13}$$

where $k_e = \frac{1}{2}EI$ where I is the area moment of inertia. For a wire with a circular cross section, $I = \frac{\pi r^4}{64}$.

The work required to impart a sharp bend in a piece of wire is approximated by the equation

$$W_p = M_p \phi \tag{14}$$

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. Then, its capacity to resist further bending is limited to the plastic moment, which remains fairly constant as bending continues. Hence, the work required to impart a sharp bend is equal to the plastic moment times the angle through which the wire is bent.

For a circular cross section, $M_p = \frac{4}{3}\sigma_y r^3$ where σ_y is the *yield stress*, or the maximum stress that the material can bear before stretching plastically. Thus,

$$M_p = \frac{512}{3\pi}\frac{\sigma_y}{E}\frac{1}{r} \tag{15}$$

For steel, $38,000\ psi \le \sigma_y \le 150,000\ psi$, depending on its makeup and how it is formed. Taking $\sigma_y \approx 53,000\ psi$, we have

$$M_p \approx \frac{k_b}{10r} \tag{16}$$

with units in pounds and inches. Thus, for a steel wire of radius $.1''$, $M_p \approx k_b$.

If the wire has a sharp bend (ie., a *plastic bend*) of angle θ_1 at a point, the work required to bend it into an angle θ_2 is

$$W_p = k_p|\theta_1 - \theta_2|^{e_p} \tag{17}$$

where normally $e_p = 1$.

In summary, three distinct work quantities must be addressed: **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). Figure 14 shows a segment of wire before and after bending and stretching. The segment initially contains a kink of angle θ_1 separating two arcs of constant curvature. In its final state, the wire has a kink of angle θ_2, again separating two arcs of constant curvature (though these curvatures may differ from the initial ones). The work equations involved are:

$$W_s = c_s \frac{(L_2 - L_1)^2}{L_1 + L_2} \tag{18}$$

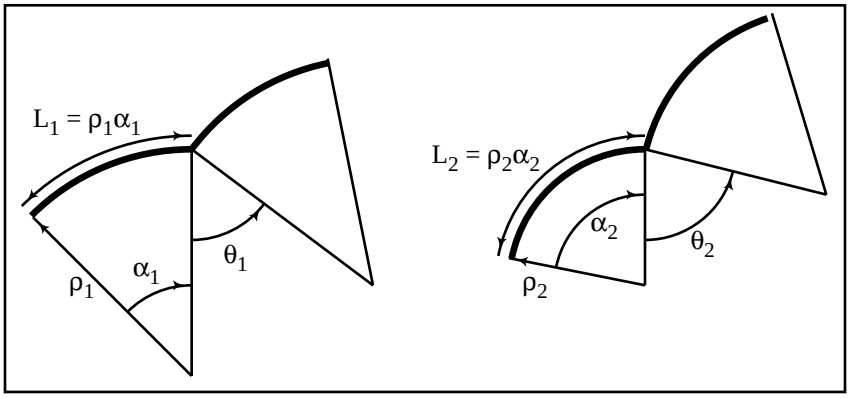

Figure 14: Stretching, Bending, and Kinking

$$W_b = c_b \frac{(\alpha_2 - \alpha_1)^2}{L_1 + L_2} \tag{19}$$

$$W_k = c_k \left|\theta_2 - \theta_1\right|^{e_k} \tag{20}$$

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.

For a piece of wire made of a material with known stress–strain relationships and with a circular cross-section of radius r, we can determine that

$$c_b = \frac{\pi r^4}{4} E \tag{21}$$

where E is the modulus of elasticity of the material. This value of c_b is accurate if $L_1 = L_2$ and if α_1 and α_2 are small, where "small" is a function of r and of the material's yield stress.

Kinking indicates that the material has experienced plastic deformation, meaning that the bending has induced stresses in the wire beyond the magnitude where a linear stress-strain relationship holds. In the case of a material such as steel, with a well defined yield stress σ_y, the kinking work coefficients become

$$c_k = \frac{4}{3} r^3 \sigma_y; \qquad e_k = 1. \tag{22}$$

This is accurate as long as the yield stress is maintained, which, for small r, can be valid for large angles $|\theta_2 - \theta_1| < \pi$.

The stretching work defined in equation 18 is accurate only for quite small elongations. The constant

$$c_s = \pi r^2 E \tag{23}$$

is valid until the strain reaches the elastic limit, which for typical steel occurs at about $\frac{L_2 - L_1}{L_1 + L_2} \approx 0.001$.

For various grades of steel, $38,000\ psi \le \sigma_y \le 150,000\ psi$ and $E = 29,000,000\ psi$, so c_s for an actual piece of steel wire would be orders of magnitude larger than c_b and c_k. Thus, while the physical properties of steel wire formed the basis for our work equations, it is preferable to imagine that our shapes are formed of some idealized material that stretches as readily as it bends. Otherwise, the stretching work would competely dominate the search for the least work solution.

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

On the other hand, there 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 [SG92]. In such cases, the best option is probably to invite the user to specify a few anchor points to guide the shape blend.

5 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 [SG92]. A more thorough discussion of the pertinent principles of graph theory is found in [FKU77]. Here we give just enough background on the graph to convey the modifications needed to support B-splines.

The algorithm works best if the Bézier segments of a B-spline each have fairly uniform curvature, and are not too long. We have not found it necessary to analyze the change in curvature for each Bézier segment, but experience suggests that it suffices to simply limit the arc length of all Bézier segments to a value of L_{max} — somewhere between 3% and 10% of the width of a box bounding the B-spline. This limit is imposed by measuring the arc length L_i between each pair of knot values $k_i \neq k_{i+1}$ and, if $L_i > L_{max}$, inserting $\lfloor L_i/L_{max} \rfloor$ "soft" knots uniformly between k_i an k_{i+1}.

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. Each row and column stores the multiplicity and value of the the knot it represents. The points at which rows and columns on the graph intersect are called graph vertices.

The shape blending problem amounts to determining a correspondence between points on the two key B-splines, and inbetween shapes occur as points on one key B-spline travel towards their respective corresponding points on the other key B-spline. This correspondence can be represented by a piecewise curve (a *path*) running from the lower left corner of the graph to the upper right corner. In fact, *all* possible correspondences between the two key B-splines can be thus represented. Our task becomes that of identifying which of all possible such paths requires the least amount of blending work, where work is defined by equations 18–20.

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 [SG92], for which the guaranteed least work solution can be determined in $O(V)$ time, where V is the number of vertices in the graph.

The graph theory solution we use proceeds virtually identically to that in [SG92] with two differences. First, the work for mapping one curve segment to another involves the work component W_b equation 19 which does not occur in the polygon case. 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.

6 Vertex Paths

7 Examples and Discussion

Figure 15 shows the Bézier font outline of an upper case G blending from Adobe 64 point Bold Courier font (left) to 64 point Bold Times font (right). 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.

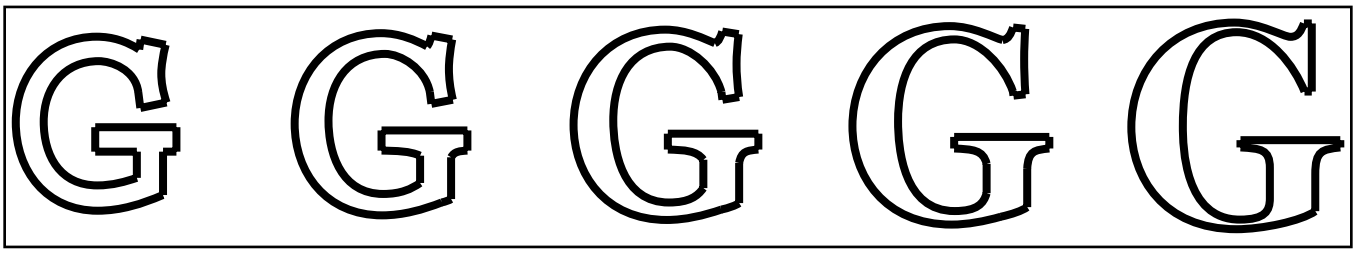

Figure 15: Shape blend example

The shape blend in Figure 16 is typical of those produced by commercial packages, in which a single pair of anchor points was specified. If six or eight pairs of anchor points are specified by the user, this commercial algorithm can be manually guided to produce a shape blend similar to that in Figure 1.

Figure 16: Shape blend using commercial software

Figure 17 shows a chicken made of 60 cubic Bézier curves.

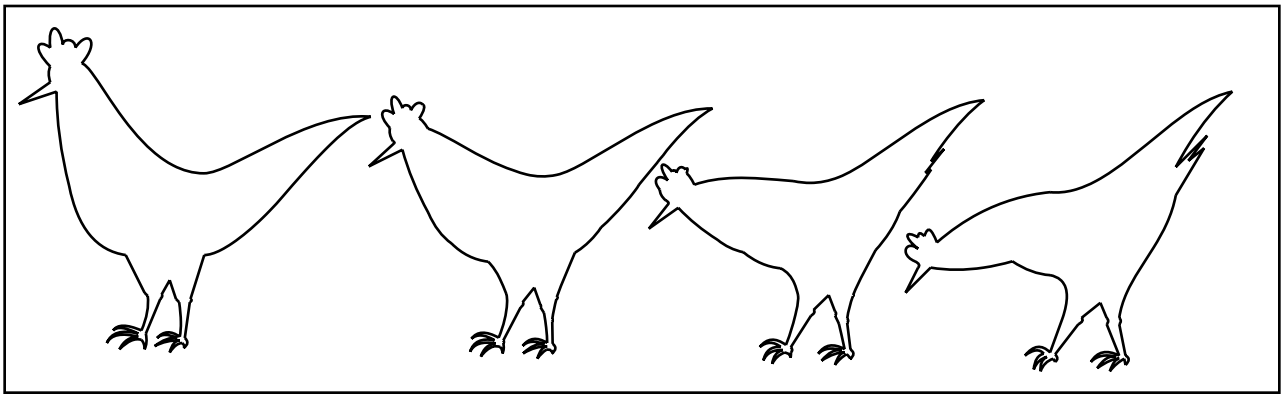

Figure 17: Chicken

7.1 Execution speed

The butterfly-moth blend in Figure 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 [SG92], the polygon data took 29 seconds to compute. The observation made in [SG92] about only performing the graph search within a percentage distance from the graph is valid for the current algorithm, and can speed up execution by a factor of five or more.

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. The help of Alan Zundel and Kris Klimaszewski was also appreciated.

References

[Ado89] Adobe Systems, Inc. *PostScript Language Reference Manual*, 1989.

[Ado91] Adobe Systems, Inc., Silicon Valley. *Adobe Illustrator*, next version 3 edition, 1991.

[Cat78] Edwin Catmull. The problems of computer-assisted animation. *Computer Graphics*, 12(3):348–353, 1978.

[Cor90] Corel Systems Corporation, Ottowa, Canada. *CorelDraw! 2.0*, 1990.

[FKU77] Henry Fuchs, Z. M. Kedem, and S. P. Uselton. Optimal surface reconstruction from planar contours. *Comm. ACM*, 20(10):693–702, 1977.

[HOS+76] Archie Higdon, Edward H. Ohlsen, William B. Stiles, John A. Weese, and William F. Riley. *Mechanics of Materials*. John Wiley & Sons, Inc., New York, 1976.

[KR91] Anil Kaul and Jarek Rossignac. Solid-interpolating deformations: Construction and animation of PIPs. In F.H. Post and W. Barth, editors, *Proc. Eurographics '91*, pages 493—505. Elsevier Science Publishers B.V, 1991.

[Meh74] Even Mehlum. Nonlinear splines. In Robert E. Barnhill and Richard F. Riesenfeld, editors, *Computer Aided Geometric Design*, pages 173–207, San Diego, 1974. Academic Press.

[Ram87] Lyle Ramshaw. Blossoming: A connect-the-dots approach to splines. Technical Report 19, Digital Systems Research Center, Palo Alto, June 1987.

[Ram88] Lyle Ramshaw. Béziers and b-splines as multiaffine maps. In R. A. Earnshaw, editor, *Theoretical Foundations of Computer Graphics and CAD*, pages 757–776, New York, 1988. Springer-Verlag.

[RBB87] John Beatty Richard Bartles and Brian Barsky. *An Introduction to Splines for use in Computer Graphics and Geometric Modeling*. Morgan Kaufmann, 1987.

[Ree81] William T. Reeves. Inbetweening for computer animation utilizing moving point constraints. *Computer Grapics (Proc. SIGGRAPH)*, 15(3):263–269, 1981.

[SG92] Thomas W. Sederberg and Eugene Greenwood. A physically based approach to 2–d shape blending. *Computer Graphics (Proc. SIGGRAPH)*, 26(2):25–34, 1992.

[US90] Naonori Ueda and Satoshi Suzuki. Automatic shape model acquisition using multiscale segment matching. In *Proc. 10th ICPR*, pages 897—902. IEEE, 1990.

Contact Author:
Thomas W. Sederberg
3374 TMCB
Brigham Young University
Provo, Utah 84602
tom@byu.edu