2–D Shape Blending: An Intrinsic Solution to the Vertex Path Problem

Thomas W. Sederberg[1], Peisheng Gao[1], Guojin Wang[2], and Hong Mu[1]

Abstract

This paper presents an algorithm for determining the paths along which corresponding vertices travel in a 2–D shape blending. Rather than considering the vertex paths explicitly, the algorithm defines the intermediate *shapes* by interpolating the intrinsic definitions of the initial and final shapes. The algorithm produces shape blends which generally are more satisfactory than those produced using linear or cubic curve paths. Particularly, the algorithm can avoid the shrinkage that normally occurs when rotating rigid bodies are linearly blended, and avoids kinks in the blend when there were none in the key polygons.

Categories and Subject Descriptors: I.3.3 [**Computer Graphics**]: Picture/Image Generation; I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling.

General Terms: Algorithms

Additional Key Words and Phrases: Shape blending, character animation, numerical algorithms.

1 Introduction

This paper deals with shape blending of 2–D polygons. As illustrated in Figures 1 and 2, a shape blend algorithm determines the in-between polygons which provide a smooth transformation between two given 2–D polygons, referred to as the *key polygons.*

Shape blending requires the solution of two main subproblems: the vertex correspondence problem (that is, determining which vertex on one key polygon will travel to which vertex on the other key polygon), and the vertex path problem (that is, determining along what path each vertex will travel).

For 2–D polygonal shapes, a solution to the vertex correspondence problem is presented in [12]. Shape blending of 2–D Bézier curve shapes is addressed in [11]. Various solutions to the shape interpolation of 3–D polyhedra are presented in [3, 5, 7, 8].

This paper addresses the vertex path problem and is motivated by two figures from [12]. Figure 1.a provides an example of a shape blend in which the middle shapes is derived from its neighboring key polygons. This is basically a good shape blend, except that the dancer's arm in the middle frame is only half as long as it is in the key frames.

The shape blend in Figure 2.a looks fine except that the chicken's neck gets shorter. These shortenings occur because of the linear path followed by vertices during the shape blend, as shown by the path travelled by the chicken's beak.

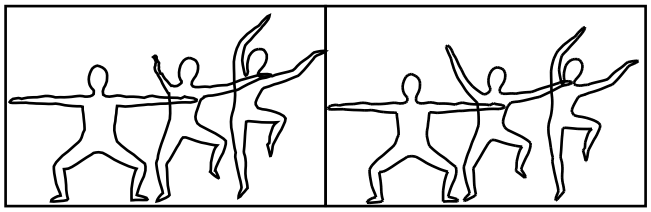

Figure 1: Dancer

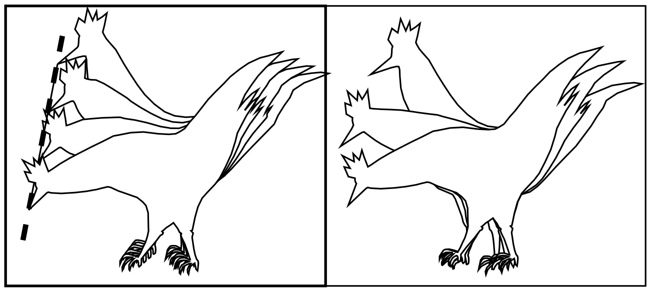

Figure 2: Chicken

These problems seriously weaken the practical use of 2–D shape blending using linear vertex motion, for applications such as character animation. The contribution of this paper is an improved method for computing in-between frames, once the correspondence between key polygons has been determined as in [12]. Sample results of the new vertex path algorithm are shown in Figures 1.b and 2.b.

A referee of paper [12] remarked: "I am unhappy with the phrase, 'physically based,' in this context. The 'physics' here has nothing to do with the physics of chickens, . . . , or any of the other nominal subjects of interpolation." That observation formulates precisely the problem we confront in trying to infer the correct motion between two changing shapes. While [12] demonstrates that an algorithm which knows nothing about the "physics of a chicken" is able to correlate the prominent features of two chicken outlines, the accurate computation of *motion* as a chicken lowers his head really calls for a model of the chicken's skeleton, musculature, etc.. What we seek is a tool that might assist a traditional animator to create convincing computer-assisted in-betweens, when the only information available is contained in the two key frames. The solution presented here is a heuristic whose justification lies

[1]Engineering Computer Graphics Laboratory
368 Clyde Building
Brigham Young University
Provo, UT 84602
tom@byu.edu
[2]Zhejiang University, China

in the fact that it generally seems to work rather well.

1.1 Proposed solution

A polygon definition which lists the Cartesian coordinates of its vertices might be called an *explicit* description. An alternate means of defining a polygon is in terms of the lengths of its edges and the angles at its vertices. Such a polygon description forms the basis of an approach to geometry, popular in elementary education, known as *Turtle Graphics* [2] wherein a polygon is defined by instructions such as: walk 10 paces to the east, turn 45° to the left and proceed 6 paces, turn 30° to the right and go 5 more paces, ...

This paper postulates that the heuristic of blending intrinsic definitions (edge lengths and vertex angles) of two key polygons will generally produce a more satisfactory in-between motion than will linear vertex paths. Evidence that this is so is provided in the figures.

1.2 Related work

One alternative to linear vertex paths is to define vertex paths of higher degree. For 3–D polyhedral shape transformations, [8] proposes using an Hermite cubic path with end tangents set equal to the vertex normals. While this idea evidently is effective for the transformations between highly dissimilar shapes addressed in [8], it would not generally work too well for character animation since motion does not uniformly occur normal to a curve outline.

[14] develops an approach to character animation using quadratic Bézier vertex paths. By default, vertices travel along a parabolic arc such that the distance from each vertex to the center of mass of all vertices changes monotonically. Also, it allows the user to signify a pivot point for appendages. The current algorithm works with less user interaction.

In other approaches to shape blending, such as Minkowski sums [7], the vertex path and vertex correspondence problems are coupled and solved simultaneously. Minkowski sums, however, blur even gross details such as arms and legs when blending non-convex objects, and hence are not suitable for character animation. Shape blends that operate on an *implicit* definition of the curve or surface, $f(x, y) = 0$ or $f(x, y, z) = 0$, [6] likewise don't currently support the detail required for character animation.

Of course, the substantial literature on physically based modelling and synthetic actors is also highly relevant, though such methods rely on more information than is available to us.

Ideas for modeling with intrinsically defined curves are proposed in [1], and [13] looks at curve and surface kinematics based on differential equations.

2 Intrinsic shape interpolation

Denote the vertices of the two key polygons by $\mathbf{P}_{A_i}, \mathbf{P}_{B_i}, (i = 0, 1, \ldots, n-1)$. We assume that both key polygons have the same number of edges, as will be the case after vertex correspondence is established [12]. In this discussion, we use the convention that counter-clockwise angles are positive. For convenience, we adopt the notation $m = n - 1$ where n is the number of polygon edges.

Our goal is to compute the vertices $\mathbf{P}_i (i = 1, 2, \ldots, m)$ for the polygon which is "t" of the way between $\mathbf{P}_A$ and $\mathbf{P}_B$, $0 \le t \le 1$. $\mathbf{P}_0$ will be taken as the *anchor point*, and its position determines the rigid body translation of the shape. This, along with the directed angles α_{A_0} and α_{B_0} formed by the x-axis and the vectors $\mathbf{P}_{A_0}\mathbf{P}_{A_1}$ and $\mathbf{P}_{B_0}\mathbf{P}_{B_1}$, is discussed further in section 3.

Begin by obtaining the intrinsic definitions of $\mathbf{P}_A$ and $\mathbf{P}_B$ by computing the polygon angles and edge lengths shown in Figure 3:

$$\theta_{A_i}, \theta_{B_i}, (i = 1, 2, \ldots, m). \tag{1}$$

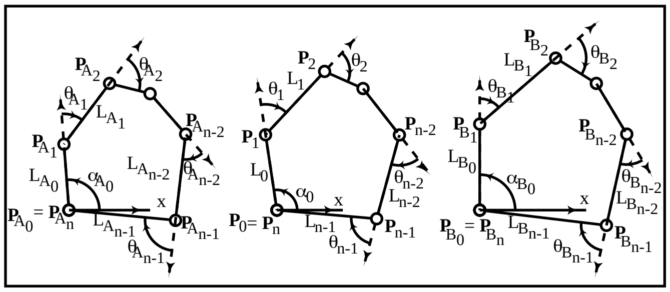

Figure 3: Intrinsic variables

$$L_{A_i} = |\mathbf{P}_{A_{i+1}} - \mathbf{P}_{A_i}| \text{ and } L_{B_i} = |\mathbf{P}_{B_{i+1}} - \mathbf{P}_{B_i}|. \tag{2}$$

The intermediate polygons in the shape blend are then computed by interpolating the respective vertex angles and edge lengths:

$$\alpha_0 = (1-t)\alpha_{A_0} + t\alpha_{B_0}, \tag{3}$$

$$\theta_i = (1-t)\theta_{A_i} + t\theta_{B_i}, \quad (i = 1, 2, \ldots, m). \tag{4}$$

$$L_i = (1-t)L_{A_i} + tL_{B_i}, \quad (i = 0, 1, 2, \ldots, m). \tag{5}$$

Unfortunately, the problem is not completely solved at this point, since the resulting polygon will not generally close. Figure 4 shows what the chicken and dancer polygons look like at this stage of the algorithm. It is somewhat surprising that

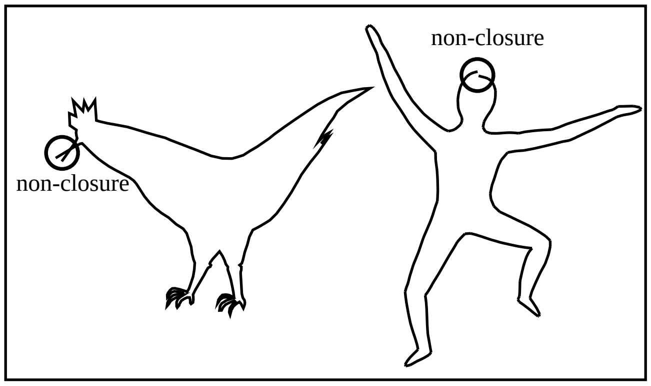

Figure 4: Unclosed polygons

these polygons, each with over 200 vertices, come so close to ending where they started (and this happens typically, in our experience). But the problem remains, how do we best adjust the lengths and angles so that the polygon *does* close.

There are two solutions to this problem. The first is to leave the angles unchanged and tweak the lengths (section 2.1). This turns out to have a straightforward, closed-form solution. The other approach is to treat the open polygon as a piece of wire for which we define the physical rules for stretching and vertex bending. Adjustments to angles and/or edges can then be computed iteratively by determining the equilibrium shape when the two open polygon vertices are forced to coincide.

2.1 Edge Tweaking

To close the polygons by adjusting the edge lengths only, rewrite equation 5 as

$$L_i = (1-t)L_{A_i} + tL_{B_i} + S_i, \quad (i = 0, 1, 2, \ldots, m). \tag{6}$$

It seems smart that the magnitudes of S_i should roughly be proportional to $|L_{A_i} - L_{B_i}|$, since if an edge has the same length on both key polygons, it ought to have about that same length throughout the shape blend. One can dream up simple examples for which this is not desirable, but for most reasonable cases, experience has verified this to be wise. Therefore, define

$$L_{AB_i} = \max\{|L_{A_i} - L_{B_i}|, L_{tol}\}, \quad (i = 0, 1, 2, \ldots, m). \quad (7)$$

where $L_{tol} = 0.0001 \times \left(\max_{i \in [0,m]} |L_{A_i} - L_{B_i}|\right)$ is needed to avoid division by zero.

Our goal is to find $S_0, S_1, \ldots, S_m$, so that the objective function

$$f(S_0, S_1, \ldots, S_m) = \sum_{i=0}^{m} \frac{S_i^2}{L_{AB_i}^2}$$

is minimized subject to the two equality constraints (which force closure of the polygon):

$$\varphi_1(S_0, S_1, \ldots, S_m) = \sum_{i=0}^{m} [(1-t)L_{A_i} + tL_{B_i} + S_i] \cos\alpha_i = 0,$$

$$\varphi_2(S_0, S_1, \ldots, S_m) = \sum_{i=0}^{m} [(1-t)L_{A_i} + tL_{B_i} + S_i] \sin\alpha_i = 0,$$

where α_i are the directed angles from the x-axis to the vectors $\mathbf{P}_i\mathbf{P}_{i+1}$,

$$\alpha_i = \alpha_{i-1} + \theta_i, \quad (i = 1, 2, \ldots, m). \quad (8)$$

The method of Lagrange multipliers [9] can now solve for the desired tweak values S_i as follows. Set

$$\Phi(\lambda_1, \lambda_2, S_0, S_1, \ldots, S_m) = f + \lambda_1\varphi_1 + \lambda_2\varphi_2,$$

where λ_1 and λ_2 are the multipliers.

From

$$\begin{cases} \frac{\partial\Phi}{\partial S_i} = \frac{2S_i}{L_{AB_i}^2} + \lambda_1 \cos\alpha_i + \lambda_2 \sin\alpha_i = 0 (i = 0, 1, \ldots, m) \\ \sum_{i=0}^{m} [(1-t)L_{A_i} + tL_{B_i} + S_i] \cos\alpha_i = 0 \\ \sum_{i=0}^{m} [(1-t)L_{A_i} + tL_{B_i} + S_i] \sin\alpha_i = 0, \end{cases}$$

we obtain

$$\begin{cases} E\lambda_1 + F\lambda_2 = U \\ F\lambda_1 + G\lambda_2 = V, \end{cases} \quad (9)$$

where

$$E = \sum_{i=0}^{m} L_{AB_i}^2 \cos^2\alpha_i, \quad (10)$$

$$F = \sum_{i=0}^{m} L_{AB_i}^2 \sin\alpha_i cos\alpha_i, \quad (11)$$

$$G = \sum_{i=0}^{m} L_{AB_i}^2 \sin^2\alpha_i, \quad (12)$$

$$U = 2\left\{\sum_{i=0}^{m} [(1-t)L_{A_i} + tL_{B_i}] \cos\alpha_i\right\}, \quad (13)$$

$$V = 2\left\{\sum_{i=0}^{m} [(1-t)L_{A_i} + tL_{B_i}] \sin\alpha_i\right\}. \quad (14)$$

Thus under the condition $EG - F^2 \neq 0$ we can get

$$\lambda_1 = \begin{vmatrix} U & F \\ V & G \end{vmatrix} / \begin{vmatrix} E & F \\ F & G \end{vmatrix}, \quad (15)$$

$$\lambda_2 = \begin{vmatrix} E & U \\ F & V \end{vmatrix} / \begin{vmatrix} E & F \\ F & G \end{vmatrix}, \quad (16)$$

and

$$S_i = -\frac{1}{2} L_{AB_i}^2 (\lambda_1 \cos\alpha_i + \lambda_2 \sin\alpha_i), \quad (i = 0, 1, \ldots, m). \quad (17)$$

Using equations 4, 6, and 8, we can now calculate the coordinates (x_i, y_i) of the vertices $\mathbf{P}_i (i = 1, 2, \ldots, m)$:

$$x_i = x_{i-1} + L_{i-1} \cos\alpha_{i-1}, \quad y_i = y_{i-1} + L_{i-1} \sin\alpha_{i-1}. \quad (18)$$

2.2 Tweaking Lengths and/or Angles

The edge-tweaking-only method generally gives good results and is relatively fast. Also, as suggested from Figure 4, often very little edge length adjustment is needed.

However, some simple examples can be found where the edge lengths may change more than is desirable using the edge-tweaking-only method. This can be detected by checking the values of S_i. In such a case, the required edge length adjustments can be diminished by also allowing the angles to change.

A good solution to this problem is to treat the unclosed polygon as a piece of wire which can possibly stretch, but which can only bend at polygon vertices. The stretching stiffness for each polygon edge is inversely proportional to the change in length experienced by that edge between the two key frames. Likewise, the bending stiffness of each angle is inversely proportional to the change between key frames of the respective angle. These stiffness values tend to enforce rigidity for identical portions of the two key polygons.

The shape of the closed intermediate polygon is then computed by forcing the two unclosed joints to coincide, and determining the unique equilibrium shape of the wire. Further details can be found in [4].

3 Anchor points and angle lines

Since an intrinsic definition of a polygon is invariant to rigid body motion, a shape blend must specify translations and rotations for the intermediate shapes. Translation is specified using an anchor point path, and rotation is constrained by designating the rotation function of an angle line. The anchor point can be a polygon vertex, or any other point that is well defined for each step in the shape blend. For example, center of area is a good anchor point for objects in free fall. For bodies in free fall, such as a diver, a parabolic anchor path simulates the effects of gravity.

The angle line can be any line whose association with each shape in the blend can be determined, such as a non-degenerate polygon edge, the line between any two points on the polygon, or a principle axis of a shape (if the major and minor axes are well defined, ie., the product of inertia is non-zero). In Figure 3, the anchor point is $\mathbf{P}_0$ and the angle line is L_0.

4 Discussion

Figure 1 shows that the main advantage of using turtle graphics in shape blending is that it helps solve the withering arm problem. Another benefit is that it can provide more nearly monotonic angle changes than does linear-vertex-path shape blending. Figure 5 shows a shape blend, taken from [12], in which a shape which should undergo a simple rigid body motion experiences shrinking *and* kinking. The kinking occurs in this case because of a poor choice of vertices, a common occurrence in linear-vertex-path shape blending. Clearly, turtle graphics shape blending would have no problem in this case.

A more impelling example is the dancer's arm which withers under linear vertex path motion. The magnification in Figure 6 illustrates that the intrinsic method produces inherently smoother blends than the linear vertex paths. [12] goes

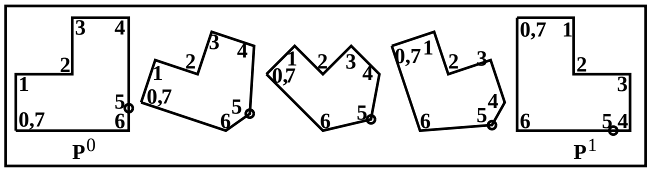

Figure 5: Shrinking plus kinking

to great lengths investigating how to minimize this angle non-monotonicity which can occur with linear vertex paths. As an added benefit of the intrinsic algorithm, this detailed search for non-monotonic angle changes is rendered unnecessary.

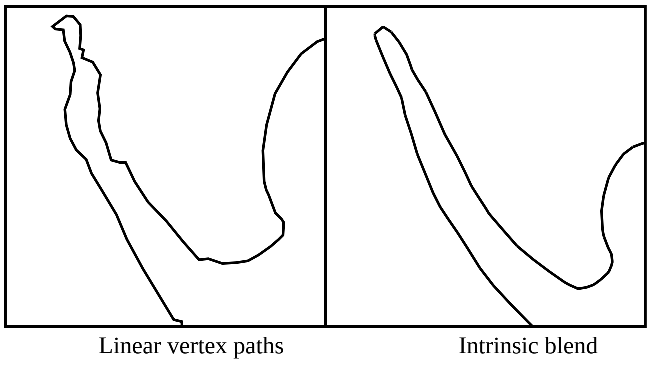

Figure 6: Closeup of dancer's arm

Although slower than the linear patch method, the intrinsic algorithm can compute a shape blend for the chicken in Figure 2.b (which has 230 vertices) in 0.02 seconds using the method in Section 2.1 and in 0.05 seconds using the method in Section 2.2, on an HP 730 workstation.

It is easy to contrive examples for which this algorithm performs poorly, although most of the realistic cases we have tried produced good results. In cases where some adjustment is called for, additional constraints can be imposed, such as specifying that the distance between specified pairs of non-adjacent polygon vertices should change monotonically from one key frame to the next. See [4] for more details.

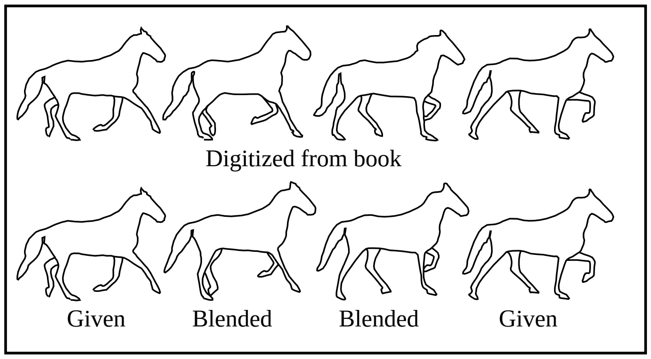

Figure 7: Cantering Horse

Experience suggests that this algorithm may work well enough for many applications to character animation. The sequence of a cantering horse in Figure 7 was taken from the classic photographic study, *Animals in Motion* [10], first published in 1887. The top four figures are digitizations of actual photographs from the book. In the bottom row, the middle two figures are shape blends interpolating the first and last figures. The vertex correspondence was determined using the algorithm in [12], and the vertex paths were computed using the algorithm in this paper. The horse's two left legs were treated as independent shape blends.

Acknowledgements

Geoffrey Slinker, Hank Christiansen, Alan Zundel, and Kris Klimaszewski provided much helpful discussion. This work was supported under NSF grant DMC-8657057.

References

[1] J. Alan Adams. The intrinsic method for curve definition. *Computer-Aided Design*, 7(4):243–249, 1975.

[2] Harold J. Bailey, Kathleen M. Brautigam, and Trudy H. Doran. *Apple Logo.* Brady Communications Company, Inc., Bowie, MD, 1984.

[3] Shenchang Eric Chen and Richard Parent. Shape averaging and its applications to industrial design. *IEEE CG&A*, 9(1):47–54, 1989.

[4] Peisheng Gao. 2–d shape blending:
an intrinsic solution to the vertex path problem. Master's thesis, Brigham Young University, Department of Civil Engineering, 1993.

[5] Andrew Glassner. Metamorphosis. *preprint*, 1991.

[6] John F. Hughes. Scheduled fourier volume morphing. *Computer Graphics (Proc. SIGGRAPH)*, 26(2):43–46, 1992.

[7] 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.

[8] James R. Kent, Wayne E. Carlson, and Richard E. Parent. Shape transformation for polyhedral objects. *Computer Graphics (Proc. SIGGRAPH)*, 26(2):47–54, 1992.

[9] S. C. Malik. *Mathematical Analysis.* John Wiley & Sons, Inc., New York, 1984.

[10] Eadweard Muybridge. *Animals in Motion.* Dover Publications, Inc., New York, 1957.

[11] Thomas W. Sederberg and Eugene Greenwood. Shape blending of 2–d piecewise curves. *Submitted.*

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

[13] Yoshihisa Shinagawa and Tosiyasu L. Kunii. The differential model: A model for animating transformation of objects using differntial information. In Tosiyasu L. Kunii, editor, *Modeling in Computer Graphics*, pages 5–15, Tokyo, 1991. Springer-Verlag.

[14] Geoffrey Slinker. Inbetweening using a physically based model and nonlinear path interpolation. Master's thesis, Brigham Young University, Department of Computer Science, 1992.