Morphing between Polylines

Alon Efrat[*] Sariel Har-Peled[†] Leonidas J. Guibas[‡] T. M. Murali[§]

Abstract

Given two non-intersecting simple polylines in the plane, we study the problem of continuously transforming or morphing one polyline into the other. Our morphing strategies have the desirable property that every intermediate polyline is also simple. We also guarantee that no portion of the polylines to be morphed is stretched or compressed by more than a user-defined parameter during the entire morphing. Our algorithms are based on the morphing width, a new metric we have developed for measuring the similarity between two polylines. We develop an algorithm that computes the morphing width of the two polylines and constructs a corresponding morphing strategy in $O(n^2 \log^2 n)$ time using $O(n^2)$ space, where n is the total number of vertices in the polylines. We describe another algorithm that computes a factor-2 approximation of the morphing width and a corresponding morphing scheme in $O(n \log n)$ time.

1 Introduction

In the last few years, the problem of continuously morphing or deforming an object or image into another object or image has become increasingly popular in computer graphics and computer vision. This problem is also called shape transformation, shape metamorphosis, and shape interpolation. In this problem, we are given two shapes α and β and we are asked to produce a continuous sequence of shapes "between" α and β. Rendering this sequence continuously as an animation will show α transforming into β. This problem has applications in animation, special effects in movies and entertainment, contour interpolation in medical imaging, computer-aided geometric design, and handwriting recognition.

There are many qualities that are desirable in a good morphing scheme. Since α and β are usually connected and simple, all intermediate shapes should also be connected and simple. The morphing should transform a connected portion of α to a connected portion of β. It is also useful for the transformation process to treat α and β as near-rigid objects and to avoid superfluous deformations during the morphing.

There are two common ways used in the literature to specify morphing schemes. The first approach uses implicit functions to represent the morphing [6, 14, 16, 18]. Many of these techniques compute two implicit functions. The zero-set of the first implicit function assumes the shape of α and the zero-set of the second implicit functions assumes the shape of β. The sequence of morphed shapes is obtained by interpolating between these two implicit functions. Turk and O' Brien [24] combine the problems of creating and interpolating implicit functions by casting the problem in a dimension one more than the dimension that α and β are embedded in. This approach based on implicit functions is elegant and has proven to be quite successful. It can create reasonable transformations when α and β have different topologies. However, it has the drawback that unless the implicit functions are chosen with great care, intermediate shapes are not guaranteed to be simple. Another problem with this method is that the correspondences between various parts of α and β are not made explicit. Further, it is difficult for the user to specify or control important correspondences.

The second popular approach makes the correspondence induced by the morphing scheme explicit. These techniques first compute correspondences between points of α and β (for instance, the vertices of α and β, if α and β are polyhedral) and then creates intermediate shapes as defined by interpolated positions

[*]The author did part of this research when he was affiliated with Stanford University. Address: Room 747, Department of Computer Science, Gould-Simpson Building, The University of Arizona, PO Box 210077, Tucson AZ 85721-0077. Email: alon@cs.arizona.edu. WWW: http://www.cs.arizona.edu/people/alon/.

[†]The author did part of this research when he was affiliated with Duke University. Address: 2111 DCL, 1304 West Springfield Avenue, Urbana, IL 61801. Email: sariel@cs.uiuc.edu. WWW: http://www.cs.uiuc.edu/contacts/faculty/harpeled.html.

[‡]Leonidas Guibas was supported by NSF grant CCR–9910633 and U.S. Army Research Office MURI grant DAAH04–96–1–007, as well as by a grant from the Intel Corporation. Address: Computer Science Dept., Stanford University, 353 Serra Mall, Stanford CA 94305. Email: guibas@cs.stanford.edu. WWW: http://graphics.stanford.edu/~guibas.

[§]Address: Compaq Computer Corporation, Cambridge Research Lab, One Cambridge Center, Cambridge MA 02142. Email: murali@crl.dec.com. WWW: http://www.crl.research.digital.com/who/people/murali/bio.htm

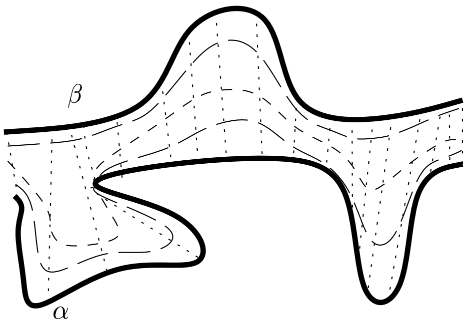

Figure 1: The two polylines α and β and an example of a morphing between them. Dashed lines represent intermediate polylines. Dotted lines show the paths traveled by the points on α and β during the morphing.

between corresponding points [10, 19, 22, 23]. Since the interpolated positions are usually chosen to lie on the segments or splines joining corresponding points, these techniques usually find it very difficult to ensure that intermediate shapes do not self-intersect.

Surprisingly, this interesting problem has received very little attention in the computational geometry community. Guibas and Hershberger [11] considered the problem when α and β are n-vertex *parallel* polygons, i.e., polygons with the same sequence of angles. They showed that any two parallel polygons can be morphed into one another such that every interpolating polygon is also parallel to the initial and final polygons. They described an algorithm to perform the morphing that requires $O(n^{4/3+\varepsilon})$ "parallel moves," for any $\varepsilon > 0$; roughly speaking, a parallel move is a translation of a side of a polygon parallel to itself. Hershberger and Suri improved this bound to $O(n \log n)$ [15]. When α and β are simple polygons, an approach that has been used is to find "compatible" decompositions of α and β and to use these decompositions to generate the correspondences between α and β. Aronov et al. [3] show that a compatible triangulation (also called a piecewise-linear homeomorphism) of size $\Theta(n^2)$ exists between two n-vertex simple polygons. Gupta and Wenger [13] construct compatible triangulations whose size is within a constant factor of optimal. Shapira and Rappaport [8] decompose the polygons into star-shaped pieces. Their technique is not able to avoid all self-intersections during the morphing and can take $O(n^4)$ time.

In this paper, we consider the morphing problem in the plane. We are given two non-intersecting simple polygonal chains (or polylines) α and β in the plane. A *morphing scheme* $\Gamma(\alpha, \beta) = \{\gamma(t) \mid 0 \leq t \leq 1\}$ from α to β is a family of polylines such that $\alpha = \gamma(0), \beta = \gamma(1)$, and for every $0 \leq t \leq 1$, $\gamma(t)$ is connected and simple. See Figure 1 for an example of a morphing. We compute a morphing scheme from α to β that consists of two parts. The first part is an explicit mapping between α and β. Given two functions $f : [0,1] \to \alpha$ and $g : [0,1] \to \beta$ that parameterize α and β, respectively, we define the mapping as the set of pairs $\{(f(u), g(u)), 0 \leq u \leq 1\}$. We enforce the natural requirement that f and g are monotone along α and β. The second part of the morphing scheme specifies a route connecting every point $f(u)$ to the corresponding point $g(u)$, for $0 \leq u \leq 1$. In this paper, we adopt the policy of moving $f(u)$ to $g(u)$ along the Euclidean shortest path from $f(u)$ to $g(u)$ that avoids α and β. This policy guarantees that all intermediate polylines are simple, since f and g are monotone along α and β and shortest paths do not cross each other (although, two such shortest paths might have a common sub-path). The resulting morphing scheme is straightforward: we move each point $f(u)$ along its designated route at a constant speed proportional to the length of the route. This morphing is guaranteed to be connected. Note that once we specify the mapping functions f and g, the morphing scheme is completely determined.

Given these requirements on the morphing scheme, it is possible to generate an uncountable number of different morphing schemes. Clearly, some schemes are better than others. If α and β are similar to each other, it is natural to expect that a good morphing scheme will ensure that every point on α will travel roughly the same distance to the point corresponding to it on β. We now formalize this intuition. We define the *width* $W(f, g)$ of a morphing scheme specified by the functions f and g to be the maximum over all values of u of the length of the route connecting the point $f(u)$ on α to the point $g(u)$ on β. We seek morphing schemes that minimize the width over all possible morphing schemes. More formally, we define the *morphing width* of α and β as

$$\mathcal{W}(\alpha, \beta) = \min_{f,g} W(f, g).$$

The morphing width of two polylines is a metric. As far as we are aware, the morphing width is a new method of computing the similarity between two polylines. If we had adopted the policy of restricting ourselves to morphing schemes where the route connecting $f(u)$ and $g(u)$ is a line segment, we would have obtained the well-known Fréchet metric [2, 9]. Unfortunately, it is easy to come up with examples of α and β where such routes cause $\gamma(t)$ to self-intersect, for at least one value of t.

In this paper, we establish three main results. We first show that if n is the total number of vertices of α and β, we can compute $\mathcal{W}(\alpha, \beta)$ and a morphing scheme with that width in $O(n^2 \log^2 n)$ time using $O(n^2)$ space. To develop this algorithm, we study the *shortest path diagram*, a combinatorial structure that encodes all the shortest paths connecting points on α to points on β. We show that the shortest path diagram has $O(n^2)$ size and that we can construct it in $O(n^2 \log n)$ time using $O(n \log^2 n)$ working space.[1] We also show that, for any parameter $r > 0$, we can represent using $O(n^2)$ space all shortest paths whose length is at most r. Our algorithm uses this fact in combination with parametric search [20]. In this respect, our algorithm is similar to the technique used by Alt and Godau [2] to compute the Fréchet distance between two polylines. However, we need to establish several properties of the shortest path diagram in order to obtain our results.

We also consider the problem of computing morphing schemes that treat α and β as being "rigid." Consider the case when α and β correspond to physical objects. It may be desirable to control the extent by which a portion of α or β is stretched or compressed by the morphing scheme. Given a constant $\kappa > 1$, we restrict our attention to morphing schemes that stretch or compress any portion of α or β by at most a factor of κ; we show that we can compute in $O(n^2 \log^2 n)$ time using $O(n^2)$ space a morphing of minimum width among all such morphings.

Finally, we also describe an algorithm that computes in $O(n \log n)$ time a morphing scheme whose width is at most $2\mathcal{W}(\alpha, \beta)$. We obtain this result by proving that if σ is a polyline that separates α and β such that σ cannot be shortened, then we can morph both α and β to σ with a morphing scheme of width at most $\mathcal{W}(\alpha, \beta)$. As a technical tool in this algorithm, we develop a method to efficiently solve the following problem: given a polygon part of whose boundary is a concave chain, construct the portion of the Voronoi diagram of the chain that lies inside the polygon.

Due to lack of space, we omit many proofs and details from this version of the paper.

2 Geometric Preliminaries

We assume that α and β have common endpoints a and b. It is easy to modify our arguments and algorithms when this assumption is not true. Let P be the polygon formed by the union of α and β. For two points $p, q \in P$, let $\pi_P(p, q)$ denote the shortest path in the plane between p and q that lies inside P and let $d_P(p, q)$ denote the length of this path. We will drop the subscript when P is clear from the context. For two points $p, q \in P$, it is well-known that $\pi(p, q)$ is a polygonal chain whose vertices (other than p and q) are reflex vertices of P. We say that the *combinatorial structure* of $\pi(p, q)$ is the following sequence: the edge of P that p lies on, the sequence of reflex vertices of P that $\pi(p, q)$ passes through, and the edge of P that q lies on.

We assume that both α and β have unit length; our algorithms can easily be extended when this assumption is not true. Let $f : [0, 1] \to \alpha$, where $f(0) = a$ and $f(1) = b$ be a continuous, increasing[2] function. Let $g : [0, 1] \to \beta$, where $g(0) = a$ and $g(1) = b$ be a continuous, increasing function. We say that the pair of functions (f, g) induces a *mapping* from α to β that associates the point $f(u) \in \alpha$ with the point $g(u) \in \beta$, where $0 \le u \le 1$. We define the *width* $W(f, g)$ of a mapping to be $\max\{d(f(u), g(u)), 0 \le u \le 1\}$. The following lemma states a useful property of the total number of combinatorially-distinct paths $\pi(f(u), g(u)), 0 \le u \le 1$.

LEMMA 2.1. *Let (f, g) be a mapping from α to β. Let $0 \le u_1, u_2, \ldots, u_k \le 1$ be numbers such that for each $1 \le i \le k$, there exists $\varepsilon > 0$ such that $\pi(f(u - \varepsilon), g(u - \varepsilon))$ and $\pi(f(u + \varepsilon), g(u + \varepsilon))$ are combinatorially different. Then $k = O(n)$ and we can store the set of shortest paths $\{\pi(f(u_i), g(u_i)), 1 \le i \le k\}$ in $O(n)$ space.*

Given a mapping (f, g) from α to β, for $0 \le u, t \le 1$, let $\delta(u, t)$ be the point on $\pi(f(u), g(u))$ such that $d(f(u), \delta(u, t)) = td(f(u), g(u))$. The following lemma shows how to convert the mapping (f, g) into a morphing.

LEMMA 2.2. *Suppose (f, g) is a mapping from α to β. For $0 \le t \le 1$, let $\gamma(t)$ be the sequence of points $\{\delta(u, t), 0 \le u \le 1\}$. The sequence $\gamma(t)$ is connected and simple and the set $\Gamma = \{\gamma(t) \mid 0 \le t \le 1\}$ is a morphing scheme from α to β.*

Given these two lemmas, we concentrate in the rest of the paper on computing a mapping (f, g) from α to β and ignore the process of converting such a mapping into a morphing scheme.

3 The Shortest Path Diagram

In this section, we define the shortest path diagram of a simple polygon and establish some of its properties. The shortest path diagram represents all the shortest paths inside P in terms of their combinatorial structures.

Let a be a point on ∂P. We parameterize every point $p \in \partial P$ by the counter-clockwise distance

[1] We use $O(n \log^2 n)$ space if we are allowed to output the elements of the shortest path diagram as we compute them. If we are required to store the entire shortest path diagram, we need $O(n^2)$ space.

[2] The function f is increasing if for every $0 \le u_1 < u_2 \le 1$, $f(u_1)$ is closer to a along α than $f(u_2)$.

from a to p along ∂P, divided by the perimeter of P. Let $\phi : [0,1) \to \partial P$ denote the bijective function corresponding to this parameterization; thus, $\phi()$ maps every point in ∂P to a point in the interval $[0,1)$. The *shortest-path diagram* $\mathcal{S}_P$ of P is a decomposition of the unit square into maximally connected regions such that for any points (u_1, v_1) and (u_2, v_2) in the same region, the combinatorial structures of the shortest paths $\pi(\phi(u_1), \phi(v_1))$ and $\pi(\phi(u_2), \phi(v_2))$ are identical. We will use the terms *faces*, *arcs*, and *nodes*, respectively, to denote the regions of $\mathcal{S}_P$, the one-dimensional boundaries between the regions of $\mathcal{S}_P$, and the points where two or more arcs of $\mathcal{S}_P$ meet. Two adjacent faces of $\mathcal{S}_P$ correspond to shortest paths whose combinatorial structures differ by one element. Therefore, every node in $\mathcal{S}_P$ has even degree. We now prove that the size of $\mathcal{S}_P$ is $O(n^2)$. We first characterize the arcs of $\mathcal{S}_P$.

LEMMA 3.1. *Let (u, v) be a point on an arc of $\mathcal{S}_P$. Then the shortest path $\pi = \pi(\phi(u), \phi(v))$ satisfies one of the following conditions:*

1. *If π consists of only one edge, then π passes through a vertex of P.*

2. *If π contains two or more edges, then either the first or last edge of π touches two vertices of P.*

Proof. Suppose the conditions of the lemma are not true. Then there exists a square s of suitably small size centered at (u, v) such that for any point $(u', v') \in s$, $\pi(\phi(u), \phi(v))$ and $\pi(\phi(u'), \phi(v'))$ are combinatorially equivalent, which contradicts the fact that (u, v) is a point on an arc of $\mathcal{S}_P$.

COROLLARY 3.1. *If (u, v) is a node of $\mathcal{S}_P$, then both the first and last edges of $\pi(\phi(u), \phi(v))$ are incident on two vertices of P.*

A simple consequence of this corollary is that no two nodes of $\mathcal{S}_P$ have identical combinatorial structures. This characterization of the nodes of $\mathcal{S}_P$ let us bound the number of nodes in $\mathcal{S}_P$.

LEMMA 3.2. *There are $O(n^2)$ nodes in $\mathcal{S}_P$.*

Proof. For two points $p, q \in \partial P$, we say that the shortest path $\pi(p, q)$ is *special* if both the first and last edges of $\pi(p, q)$ are incident on two vertices of P. By Corollary 3.1, every node in $\mathcal{S}_P$ corresponds to a unique special path. We prove that there are $O(n^2)$ special shortest paths in P, which proves the lemma. Let $\pi(p, q)$ be a special path and let e and f be the edges of $\pi(p, q)$ that are incident on p and q respectively. There are four types of special paths:

1. Both p and q are vertices of P. Clearly, there are $O(n^2)$ such special paths.

2. p is not a vertex of P but q is. In this case, let r be the vertex of P that lies in e and is closer to p. The special path $\pi(r, q)$ is of the previous type, so we can charge $\pi(p, q)$ to $\pi(r, q)$.

3. p is a vertex of P but q is not. We handle this case in a manner similar to the previous case.

4. Neither p not q is a vertex of P. We can handle this case by combining the arguments for the previous two cases.

Each special path is charged at most four times by this argument, thus proving the lemma.

A simple perturbation argument shows that any shortest path can be continuously transformed into a special path of the same combinatorial type, which implies that the number of faces of $\mathcal{S}_P$ is $O(n^2)$. We have now proved the following theorem:

THEOREM 3.1. *If P is a polygon with n vertices, then its shortest-path diagram has $O(n^2)$ size.*

In order to describe the algorithm to construct $\mathcal{S}_P$, we prove a property of the nodes of $\mathcal{S}_P$.

LEMMA 3.3. *There are four arcs incident on every node of $\mathcal{S}_P$.*

Proof. Let (u, v) be a node of $\mathcal{S}_P$. Let s be a square of suitably small size centered at (u, v). We can find a constant $\varepsilon > 0$ such that the four points $(u \pm \varepsilon, v \pm \varepsilon)$ are contained in s and the following two properties hold:

1. The combinatorial types of $\pi(\phi(u+\varepsilon), \phi(v+\varepsilon))$, $\pi(\phi(u+\varepsilon), \phi(v-\varepsilon))$, $\pi(\phi(u-\varepsilon), \phi(v+\varepsilon))$, and $\pi(\phi(u-\varepsilon), \phi(v-\varepsilon))$ are different and

2. for every point (u', v') in s, the combinatorial type of $\pi(\phi(u'), \phi(v'))$ is identical to that of one of these four paths.

Therefore, there are four faces of $\mathcal{S}_P$ adjacent to (u, v), which proves the lemma.

The next lemma establishes the relationship between two nodes that are connected by an arc. The intuition behind this lemma is that if two nodes are connected by an arc, then the shortest paths corresponding to these nodes are closely related. In particular, it is true that the first edge on one path is adjacent to the first edge in the other path in the visibility ordering around the first or second vertex of one of the two path or that an analogous statement holds for the last edges of the two paths.

LEMMA 3.4. *Let (u, v) and (u', v') be two nodes of $\mathcal{S}_P$ that are incident on the same arc of $\mathcal{S}_P$. Assume that $u \neq u'$. Let $r_1, r_2, \ldots$ be the sequence of reflex vertices on $\pi(\phi(u), \phi(v))$ and let $r_1', r_2', \ldots$ be the sequence of reflex vertices on $\pi(\phi(u'), \phi(v'))$. Then one of the following conditions holds:*

1. *The paths $\pi(\phi(u), \phi(v))$ and $\pi(\phi(u'), \phi(v'))$ pass through the same number of reflex vertices of P, $r_1 = r_1'$, and the triangle with vertices r, $\phi(u)$, and $\phi(u')$ does not contain any vertices of P in its interior,*

2. *The paths $\pi(\phi(u), \phi(v))$ and $\pi(\phi(u'), \phi(v'))$ pass through the same number of reflex vertices of P, $r_2 = r_2'$, and the triangle with vertices r_2, $\phi(u)$, and $\phi(u')$ does not contain any vertices of P in its interior, or*

3. *The path $\pi(\phi(u), \phi(v))$ passes through one more reflex vertex than $\pi(\phi(u'), \phi(v'))$, $r_1' = r_2$, and the triangle with vertices r_2, $\phi(u)$, and $\phi(u')$ does not contain any vertices of P in its interior.*

We are now ready to describe the algorithm for constructing $\mathcal{S}_P$. There are two main steps.

1. We construct the set S of all the nodes of $\mathcal{S}_P$. To do so, we compute the shortest paths between all pairs of vertices of P using the techniques of Guibas et al. [12]. We add the node corresponding to each of these paths to S. If we compute the shortest path for a pair of vertices (p, q), where p is a reflex vertex, then we extend the first edge of $\pi(p, q)$ backward till it intersects ∂P at a point p' by performing a ray-shooting query [5]. We then add the node corresponding to (p', q) to S. We perform a similar process if q is a reflex vertex. As a by-product of the process of computing all the shortest paths, we also store for each reflex vertex v of P a sorted list L_v of all the vertices of P that are visible to v.

2. We use Lemma 3.4 to connect all the nodes in S by computing the arcs of $\mathcal{S}_P$. Let (u, v) be a node computed in the previous step and let r and s be the first and last vertices on $\pi(\phi(u), \phi(v))$, respectively. We assume, without loss of generality, that $\pi(\phi(u), \phi(v))$ is directed from $\phi(u)$ to $\phi(v)$ and that r is to the left of $\pi(\phi(u), \phi(v))$.

   (a) We do a binary search to locate $\phi(u)$ in the list L_r. Let p be the vertex in L_r to the left of $\phi(u)$. Let p' be the point on ∂P obtained by extending the segment rp beyond p (we have already computed p' in the previous step). We connect (u, v) to the node $(\phi^{-1}(p'), v)$.

   (b) If r' is the second vertex on $\pi(\phi(u), \phi(v))$ and r' lies on the right of $\pi(\phi(u), \phi(v))$, we locate $\phi(u)$ in the list $L_{r'}$, find the point p' in $L_{r'}$ to the right of $\phi(u)$, and connect (u, v) to the node $(\phi^{-1}(p''), v)$, where p'' is the point on ∂P obtained by extending the segment $r'p'$ beyond p'.

   (c) We execute an analogous procedure with $\phi(v)$ and s.

The correctness of the algorithm follows from Corollary 3.1 and Lemma 3.4. We now analyze the running time of the algorithm. We can compute the shortest path tree for each vertex of P in $O(n)$ time per vertex [12]. We can answer each ray-shooting query made in Step 1 in $O(\log n)$ time [5]. Thus, we spend a total of $O(n^2 \log n)$ time in Step 1. In Step 2, for each arc we add, we spend $O(\log n)$ time. Therefore, the running time of the algorithm is $O(n^2 \log n)$. The algorithm uses $O(n^2)$ space. If it is enough to output the nodes and arcs of $\mathcal{S}_P$ as we find them, we can reduce the space requirement to $O(n \log^2 n)$ using the technique of Agarwal et al. [1] to store the visibility graph of a polygon compactly.

THEOREM 3.2. *We can construct the shortest path diagram of a polygon with n vertices in $O(n^2 \log n)$ time using $O(n \log^2 n)$ working space.*

4 Exact Algorithm

In this section, we present an algorithm for computing a mapping (f, g) between α and β whose width is $\mathcal{W}(\alpha, \beta)$, the morphing width of α and β. We will use the shortest path diagram $\mathcal{S}_P$ as the basis of our algorithm in combination with parametric search. To make the description of the algorithm simpler, we slightly modify the definition of $\mathcal{S}_P$. Recall that a and b are the common endpoints of α and β and that α and β have unit length. Let $\phi_1 : [0, 1] \to \partial\alpha$ denote the bijective function such that $\phi_1(u), 0 \leq u \leq 1$ is the point on α whose distance from a along α is u. Define $\phi_2 : [0, 1] \to \partial\beta$ similarly. The *shortest-path diagram* $\mathcal{S}_P$ of $P = \alpha \cup \beta$ is a decomposition of the unit square into maximally connected regions such that for any points (u_1, v_1) and (u_2, v_2) in the same region, the combinatorial structures of the shortest paths $\pi(\phi_1(u_1), \phi_2(v_1))$ and $\pi(\phi_1(u_2), \phi_2(v_2))$ are identical. Note that we can use the algorithm of the previous section (with minor modifications) to construct $\mathcal{S}_P$.

We define a *trajectory* to be an xy-monotone path in $\mathcal{S}_P$ connecting the points $(0, 0)$ and $(1, 1)$. Let T be a trajectory in $\mathcal{S}_P$. We observe that there is a unique mapping (f_T, g_T) corresponding to T; for every point

$(x,y) \in T$, this mapping associates the point $\phi_1(x) \in \alpha$ with the point $\phi_2(y) \in \beta$. We abuse notation and let $d(u,v)$ denote the shortest-path distance $d(\phi(u),\phi(v))$. We define $d_T = \max_{(u,v)\in T} d(u,v)$ as the maximum value that $d(u,v)$ assumes over all points $(u,v) \in T$. The following simple lemma is central to our algorithm:

LEMMA 4.1. *If T is a trajectory in $\mathcal{S}_P$ and (f_T, g_T) is the mapping corresponding to T, then the width $W(f_T, g_T) = d_T$.*

Thus, it is enough to compute the trajectory T^* that minimizes the value of d_{T^*} over all trajectories. We omit the details of constructing the mapping corresponding to T^*.

It appears difficult to compute T^* directly from $\mathcal{S}_P$. We adopt an approach that is similar to the technique used by Alt and Godau [2] for computing the Fréchet distance between two polylines. In a pre-processing step, we compute $\mathcal{S}_P$. We then construct an oracle that determines for a parameter $r > 0$, whether there exists a trajectory T_r with $d(T_r) \le r$. We combine this oracle with parametric search [20] to compute T^*. We develop only the details of the oracle since we use parametric search in the standard manner.

For $r > 0$, we define $\mathcal{S}_P(r)$ to be set of all points (u,v) such that $d(u,v) \le r$; clearly, $\mathcal{S}_P(r)$ is a refinement of $\mathcal{S}_P$. Our oracle is simple: we first construct $\mathcal{S}_P(r)$ and then check if there is a trajectory that passes only through the points of $\mathcal{S}_P(r)$. To construct $\mathcal{S}_P(r)$, we note that in each face of $\mathcal{S}_P$, the function $d(u,v)$ has constant description complexity. Therefore, the complexity of the portion of $\mathcal{S}_P(r)$ inside a face of $\mathcal{S}_P$ is proportional to the size of that face. Thus, the size of $\mathcal{S}_P(r)$ is $O(n^2)$ and we can compute it from $\mathcal{S}_P$ in $O(n^2)$ time. We use a standard graph search to check if there is a trajectory that passes only through points of $\mathcal{S}_P(r)$. Plugging this oracle into the parametric search machinery, we obtain the following result:

THEOREM 4.1. *Given two non-intersecting, simple polylines α and β, we can compute a mapping (f,g) between α and β whose width is the morphing width of α and β in $O(n^2 \log^2 n)$ time using $O(n^2)$ space.*

We now show how to extend this result when we want to control the amount of distortion α and β undergo during the morphing. We start by formalizing the notion of distortion. Let (f,g) be a mapping from α to β. For $0 \le u_1 < u_2 \le 1$, let $f(u_1,u_2)$ denote the portion of α between $f(u_1)$ and $f(u_2)$ and let $|f(u_1,u_2)|$ denote the length of $f(u_1,u_2)$; define similar notation for g. For $\kappa > 1$, we say that the mapping (f,g) is *κ-stretched* if for every $0 \le u_1 < u_2 \le 1$, we have that $(1/\kappa)|f(u_1,u_2)| \le |g(u_1,u_2)| \le \kappa|f(u_1,u_2)|$. If T is the trajectory corresponding to a κ-stretched mapping, we say that T is κ-stretched; it is easy to see that the slope at every point on T lies between $1/\kappa$ and κ.

In the rest of this section, we present an algorithm to find the κ-stretched trajectory T^* that minimizes the value of d_{T^*} over all κ-stretched trajectories. Once again, we construct an oracle that checks, given a parameter $r > 0$, whether there is a κ-stretched trajectory T such that $d_T \le r$. We omit the details of how we combine this oracle with parametric search.

Let S be the set of cells of the vertical decomposition of the complement of $\mathcal{S}_P(r)$. Each cell in S has constant description complexity; thus the size $|S|$ of S is $O(n^2)$. To describe the algorithm, we will find it convenient to think of these cells as obstacles. We say that a κ-stretched trajectory T is *legal* if it does not contain any point in the interior of a cell in S, i.e., if it lies entirely inside $\mathcal{S}_P(r)$. The following lemma is central to our argument:

LEMMA 4.2. *Given $r \ge 0, \kappa \ge 1$, we can find a legal κ-stretched trajectory in $\mathcal{S}_P(r)$ or determine that no such trajectory exists in $O(n^2 \log n)$ time.*

Proof. Let p be a point in $\mathcal{S}_P(r)$. We define a *flashlight* F_p centered at p to be the wedge with p as origin and whose two edges have slope κ and $1/\kappa$, respectively. A point q is *illuminated by* F_p if q is inside F_p and the segment pq does not intersect the interior of any cells in S. A point q is *illuminated* if it is illuminated either by the flashlight $F_{(0,0)}$ or by a flashlight F_p such that p is also illuminated. It is clear that the point $(1,1)$ is illuminated iff $\mathcal{S}_P(r)$ contains a legal κ-stretched trajectory. Therefore, we settle the question of whether $\mathcal{S}_P(r)$ contains a κ-stretched trajectory or not by computing the set of all illuminated points and checking if the point $(1,1)$ is illuminated.

We compute the set of illuminated points by performing a sweep starting at $(0,0)$. We use a sweep line ℓ making an angle of $3\pi/4$ with the x-axis and moving to the right. We initially place a flashlight at $(0,0)$ and add more flashlights as the sweep proceeds. At each instant, we maintain the invariant that we have computed the set of illuminated points to the left of ℓ. We also maintain the intervals of ℓ that are illuminated and the intervals in which ℓ intersects the cells of S. The event points of the sweep are the nodes of the cells of S, the intersections of the edges of the flashlights with the arcs of the cells of S, and intersections between the flashlights themselves. The last two types of events are interesting.

1. The edge of a flashlight F_p intersects the boundary of a cell $c \in S$. We trace the illuminated portion of

∂c until we reach a point r where this illuminated portion ends. If the line through p and r is tangent to c and does not intersect the interior of any other cell in S adjacent to r, we add a flashlight centered at r.

2. The intervals of ℓ illuminated by two flashlights intersect. It is easy to see that the union of these two flashlights is connected to the right of ℓ. Therefore, we merge these two flashlights into one.

There are at most $2|S|+1$ flashlights created during this sweep, since we place at most two flashlights on the boundary of each cell in S. Every time we merge two flashlights, we decrease the number of flashlights by one. Therefore, there are $O(|S|)$ events during the sweep, each of which we can process in $O(\log n)$ time using standard data structures. Since $|S| = O(n^2)$, we have proven the lemma.

It is easy to see that if there is a legal κ-stretched trajectory, then there is a polygonal trajectory whose vertices are the origins of the flashlights placed by the algorithm. Therefore, the algorithm will detect that $(1, 1)$ is illuminated. Since the converse is also true, we have established the correctness of the algorithm.

We obtain the following theorem once we use this lemma in combination with parametric search:

THEOREM 4.2. *Given two non-intersecting, simple polylines α and β and a parameter $\kappa > 1$, we can compute a κ-stretched mapping (f, g) between α and β with minimum width in $O(n^2 \log^2 n)$ time using $O(n^2)$ space.*

5 Approximating the Morphing Width

In this section, we describe an approximation algorithm for computing the morphing width $\mathcal{W}(\alpha, \beta)$ of α and β. As before, a and b are the the two common endpoints of α and β and P is the polygon formed by the union on α and β. Let $\sigma = \pi_P(a, b)$ denote the shortest path between a and b inside P. Let P_1 and P_2 be the two polygons created from P by adding σ; see Figure 2(i). The boundary ∂P_1 may be degenerate in the sense that non-consecutive vertices of P_1 have identical coordinates. This degeneracy occurs whenever σ has two consecutive vertices such that one vertex lies in ∂P_1 and the other vertex lies in ∂P_2. A similar remark holds for ∂P_2. We first prove a structural lemma that states that we can approximate $\mathcal{W}(\alpha, \beta)$ by a factor of two. Then we show how to compute such an approximation and a morphing scheme with that width.

5.1 A Structural Lemma

The intuition behind our factor-2 approximation for $\mathcal{W}(\alpha, \beta)$ is that width of the morphing scheme obtained by first transforming α into σ and then transforming σ into β is at most $2\mathcal{W}(\alpha, \beta)$. We now provide a formal basis for this intuition. For a point $x \in P$, let $d_P(\sigma, x)$ denote the minimum length of a shortest path between x and a point of σ.

Clearly, any morphing scheme for P has width at least w, since for all $y \in \beta, d(q, y) \geq d(\sigma, q)$ and for all $x \in \alpha, d(x, r) \geq d(\sigma, r)$. We claim that the morphing width of both P_1 and P_2 is at most w, which implies the result. We prove the claim for P_1. A similar argument holds for P_2.

We parameterize α and σ monotonically such that $\alpha = \cup_{s=0}^{1} \alpha(s)$ and $\sigma = \cup_{t=0}^{1} \sigma(t)$. Let $u(s, t)$ denote the length of the shortest path connecting the points $\alpha(s)$ and $\sigma(t)$ inside P_1. By construction, for any $0 \leq a_0 \leq 1$, there exits a b_0 such that $u(a_0, b_0) \leq w$. Furthermore, we claim that if there exists $b_1 \geq b_0$ such that $u(a_0, b_1) \leq w$ then $u(a_0, v) \leq w$, for all $b_0 \leq v \leq b_1$.

Indeed, assume that this claim is false, and let a_0, b_0, and b_1 be such that $b_0 \leq b_1$, $b_1 - b_0$ is minimized, $u(a_0, b_0) = u(a_0, b_1) = w$, and $u(a_0, v) > w$ for all $b_0 < v < b_1$. Let $p = \alpha(a_0), q = \sigma(b_0), r = \sigma(b_1)$, $\gamma_{pq} = \pi_{P_1}(p, q)$, $\gamma_{qr} = \sigma(q, r)$, and $\gamma_{rp} = \pi_{P_1}(r, p)$. Let P' be the polygon bounded by $\gamma = \gamma_{pq} \cup \gamma_{qr} \cup \gamma_{rp}$. We assume without loss of generality that γ is oriented such that the interior of P' lies to the left of γ. See Figure 2(ii). We assume that γ_{pq} and γ_{rp} are disjoint in their interior. Otherwise, the following argument can be applied to their portions that are disjoint.

Note that γ_{pq}, γ_{qr} and γ_{rp} are all portions of shortest paths, and can not be shortened inside P', since $P' \subseteq P$. In particular, except for p, q, r all other vertices of P' have an angle at least π. Following the same reasoning, the angles of P' at q and r must also be at least $\pi/2$ (otherwise, γ_{pq} or γ_{rp} can be shortcut). The angle μ of P' at p is nonzero. Let k be the number of vertices of P'. The arguments above imply that the sum of the angles of P' is at least $(k-3)\pi + 2(\pi/2) + \mu > (k-2)\pi$, which is impossible, since the sum of the angles of a polygon with k vertices is equal to $(k-2)\pi$.

It is now straightforward to prove, that the set

$$U = \left\{ (s, t) \;\middle|\; 0 \leq s, t \leq 1, u(s, t) \leq w \right\}$$

is connected and contains the points $(0, 0)$ and $(1, 1)$. In particular, U contains an xy-monotone trajectory that connects $(0, 0)$ to $(1, 1)$. The xy-monotonicity of the trajectory follows from the structure of the Voronoi diagram induced by σ inside P. (See the next section for details.) This trajectory corresponds to a morphing scheme for α and σ with width at most w. This argument proves the following theorem:

THEOREM 5.1. *Let $q \in \alpha$ and $r \in \beta$ be the two points that maximize $d_P(\sigma, q)$ and $d_P(\sigma, r)$, respectively. Let $w = \max(d_P(\sigma, q), d_P(\sigma, r))$; then $w \leq \mathcal{W}(\alpha, \beta) \leq 2w$.*

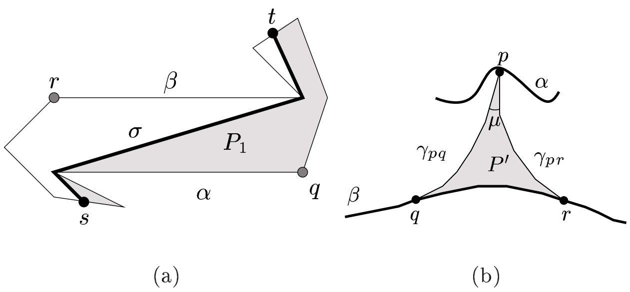

Figure 2: Illustration of the proof of Theorem 5.1: (a) The shortest path σ. (b) The polygon P'. If the angles at q and r are smaller than $\pi/2$ (as in the figure), we can shorten γ_{pq}, γ_{pr} to realize a shorter path between p and a point inside $\sigma(q, r)$.

5.2 Computing a Morphing Scheme with Width $2\mathcal{W}(\alpha, \beta)$

In this section, we describe an algorithm that uses Theorem 5.1 to compute a morphing scheme with width at most $2\mathcal{W}(\alpha, \beta)$. We sketch the algorithm since the details are straightforward but somewhat tedious. We parameterize α and β as before. For each point $x \in \alpha$, we find the shortest path $\pi(x, \sigma)$ and the point $x' \in \sigma$ that realizes this shortest path. Using this information, we construct a morphing scheme for α and σ. We perform a similar procedure for each point on β to compute a morphing scheme for β and σ. Finally, we merge these two morphing schemes to obtain a morphing scheme for α and β. The key step in this algorithm is finding for each point in α and β the closest point on σ. We spend the rest of the section describing our procedure for doing so in $O(n \log n)$ time. Since we can compute the curve σ in linear time using the algorithm of Guibas et al. [12], we obtain the following theorem:

THEOREM 5.2. *We can compute in $O(n \log n)$ time a morphing scheme (f, g) for α and β such that the width $W(f, g)$ satisfies $\mathcal{W}(\alpha, \beta) \leq W(f, g) \leq 2\mathcal{W}(\alpha, \beta)$.*

We now turn our attention to the problem of computing for each point on α the point on σ that is closest to it. We adopt an identical procedure for β. For a point $x \in P_1$, let $v(x)$ be the point of σ such that the length of the Euclidean shortest path $\pi_{P_1}(x, v(x))$ is the smallest over all the paths from x to points on σ; $v(x)$ is either a vertex of σ or a point that lies in the interior of an edge of σ. We now define the *geodesic Voronoi diagram of σ inside* P_1, denoted by $\mathcal{V}_{P_1}(\sigma)$, as the decomposition of P_1 into maximal cells such that for any two points x and y in the same cell of $\mathcal{V}_{P_1}(\sigma)$, the shortest paths $\pi_{P_1}(x, v(x))$ and $\pi_{P_1}(y, v(y))$ are combinatorially identical. It is clear that once we compute $\mathcal{V}_{P_1}(\sigma)$, we can compute for each point on α the point on σ that is closest to it. We are not aware of any near-linear time algorithm for computing such a Voronoi diagram in the general case (when σ is an arbitrary polyline and not a shortest-path). In the rest of the section, we drop the subscript from $\mathcal{V}_{P_1}(\sigma)$.

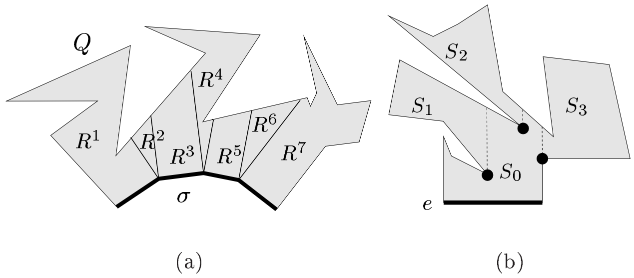

Figure 3: Illustration of the proof of Lemma 5.1: (a) Decomposing a sub-polygon Q into cells of the Voronoi diagram. (b) Decomposing the Voronoi cell corresponding to the edge e by performing vertical sweeping. The resulting sub-polygons are S_0, S_1, S_2, and S_3. Each polygon except S_0 has an associated gateway; all shortest paths fom inside such a polygon to e pass through the gateway.

Let $\mathcal{Q}$ be the set of polygons that are the closures of the maximal connected components of the interior of P_1. Clearly, the overall combinatorial complexity of the polygons in $\mathcal{Q}$ is bounded by $O(n)$. We compute the portion of $\mathcal{V}(\sigma)$ inside each polygon $Q \in \mathcal{Q}$ separately. Let σ' denote the portion of σ inside Q. The portion of $\mathcal{V}(\sigma)$ inside Q is identical to the Voronoi diagram $\mathcal{V}_Q(\sigma')$ of σ' inside Q. However, since σ' cannot be shortened inside Q and since ∂Q is not degenerate, we conclude that σ' is concave.

The Voronoi diagram of σ' inside Q is induced either by the interior of an edge of σ' or by a vertex of σ'. The bisector of the interior of an edge of σ' and its endpoint is simply a ray perpendicular to the edge emanating from the vertex directed into the interior of Q. Each vertex of σ' induces two such rays and the region between them is the cell of $\mathcal{V}_Q(\sigma')$ corresponding to the vertex. Since σ' is concave, these rays do not intersect. In particular, these rays decompose Q into a set of polygons $R^1, R^2, \ldots, R^u$ such that each such polygon is the cell in $\mathcal{V}_Q(\sigma')$ corresponding to either a vertex of σ' or an edge of σ'.

We can compute the decomposition of Q into the polygons $R^1, R^2, \ldots, R^u$ by preprocessing Q for ray shooting and performing $u-1$ ray shooting queries in Q. If we use the algorithm of Guibas et al. [12] to perform these queries, this procedure takes a total of $O(\sum_{Q \in \mathcal{Q}} |Q| \log |Q|) = O(n \log n)$ time, where $|Q|$ denotes the number of vertices of Q.

We are now left with the task of computing the portion of $\mathcal{V}_Q(\sigma')$ inside each R^j. If R^j is the Voronoi cell of a vertex of σ', we can use the algorithm of Papadopoulou and Lee [21] to compute the Voronoi diagram inside R^j in $O(|R^j| \log |R^j|)$ time. If R^j is the Voronoi cell of an edge e of σ', we observe that e is on the boundary of R^j and we would like to compute the Voronoi diagram $\mathcal{V}_{R^j}(e)$ of e inside R^j. Rotate R^j so that e is parallel to the x-axis (note that the two edges adjacent to e are vertical). For a point $x \in R^j$, the shortest-path to e ends in a vertical segment connected to e. From each vertex of R^j that is vertically visible from e (we assume that all the vertices that see e are above it), if we shoot a vertical ray upwards until the ray intersects ∂R^j, we decompose R^j into a set $\mathcal{S}$ of polygons. One of the polygons $S \in \mathcal{S}$ has e on its boundary; for every point $x \in S$, $\pi_S(x, e)$ is a vertical segment. Every other polygon $S' \in \mathcal{S}$ has a vertex $p(S')$ that is vertically visible from e. For every point $x \in S'$, $\pi_{S'}(x, e)$ passes through $p(S')$. Thus, we can now compute the Voronoi diagram of e inside R^j by computing the Voronoi diagram of $p(S')$ inside each polygon $S' \in \mathcal{S}$.

We can compute the set of vertices $\{p(S'), S' \in \mathcal{S}\}$ in $O(|R^j| \log |R^j|)$ time (in fact, we can do so in linear time by computing the vertical decomposition of R^j [4]). Inside each sub-polygon $S' \in \mathcal{S}$, we can compute the Voronoi diagram of $p(S')$ in $O(|S'| \log |S'|)$ time using the algorithm of Papadoupoulou and Lee [21]. It is now straightforward to show that the overall running time of the algorithm is $O(n \log n)$.

LEMMA 5.1. *If P is a polygon with n vertices and σ is a concave chain forming a portion of ∂P, we can compute the Voronoi diagram induced by σ inside P in $O(n \log n)$ time.*

6 Conclusions

In this paper, we have examined the problem of finding a continuous transformation or "morph" between two non-intersecting simple polylines in the plane. The strength of our algorithms lies in the fact that we guarantee that any polyline formed during the transformation is also simple. While it would be interesting to improve the efficiency of the algorithms we propose, it might be more useful to study generalizations of this problem to sets of (possibly intersecting) polygons in the plane and to higher dimensions. In practice, our policy of routing points along Euclidean shortest paths can produce transformations that are not visually pleasing. Therefore, developing a model for morphing that leads to visually appealing results and also allows the development of efficient algorithms is very important.

In this paper, we have also introduced the morphing width, a new metric for measuring the similarity between two polylines. One interesting by-product of our algorithms to compute the morphing width of two polylines is a solution to the following problem, variants of which have been considered in the literature [7, 17]: Sweep a polygon with a chain of guards that start and end at two pre-specified points on the boundary of the polygon such that a target moving continuously inside the polygon is discovered. Our techniques ensure that we can minimize the length of the chain of the guards (or approximate the minimum length) while ensuring that each point in the polygon is swept over only once. The morphing width may be useful in other applications.

References

[1] P. K. Agarwal, N. Alon, B. Aronov, and S. Suri. Can visibility graphs be represented compactly? *Discrete Comput. Geom.*, 12:347–365, 1994.

[2] H. Alt and M. Godau. Computing the Fréchet distance between two polygonal curves. *Internat. J. Comput. Geom. Appl.*, 5:75–91, 1995.

[3] B. Aronov, R. Seidel, and D. Souvaine. On compatible triangulations of simple polygons. *Comput. Geom. Theory Appl.*, 3(1):27–35, 1993.

[4] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete Comput. Geom.*, 6(5):485–524, 1991.

[5] B. Chazelle, H. Edelsbrunner, M. Grigni, L. J. Guibas, J. Hershberger, M. Sharir, and J. Snoeyink. Ray shooting in polygons using geodesic triangulations. *Algorithmica*, 12:54–68, 1994.

[6] D. Cohen-Or, A. Solomovic, and D. Levin. Three-dimensional distance field metamorphosis. *ACM Transactions on Graphics*, 17(2):116–141, Apr. 1998.

[7] A. Efrat, L. J. Guibas, S. Har-Peled, D. C. Lin, J. S. B. Mitchell, and T. M. Murali. Sweeping simple polygons with a chain of guards. In *Proc. 11th ACM-SIAM Symposium on Discrete Algorithms*, pages 927–936, 2000.

[8] M. S. (Etzion) and A. Rappoport. Shape blending using the star-skeleton representation. *IEEE Comput. Graph. Appl.*, 15(2):44–50, Mar. 1995.

[9] M. Fréchet. Sur quelques points du calcul fonctionnel. *Rendiconti del Circolo Mathematico di Palermo*, 22:1–74, 1906.

[10] A. Gregory, A. State, M. Lin, D. Manocha, and M. Livingston. Interactive surface decomposition for polyhedral morphing. *The Visual Computer*, 15:453–470, 1999.

[11] L. J. Guibas and J. Hershberger. Morphing simple polygons. In *Proc. 10th Annu. ACM Sympos. Comput. Geom.*, pages 267–276, 1994.

[12] L. J. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. E. Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2:209–233, 1987.

[13] H. Gupta and R. Wenger. Constructing piecewise linear homeomorphisms of simple polygons. *Journal of Algorithms*, 22:142–157, 1997.

[14] T. He, S. Wang, and A. Kaufman. Wavelet-based volume morphing. In R. D. Bergeron and A. E. Kaufman, editors, *Proceedings of IEEE Visualization*, pages 85–92, Los Alamitos, CA, USA, Oct. 1994. IEEE Computer Society Press.

[15] J. Hershberger and S. Suri. Morphing binary trees. In *Proc. 6th ACM-SIAM Sympos. Discrete Algorithms*, pages 396–404, 1995.

[16] J. F. Hughes. Scheduled fourier volume morphing. In *Proc. SIGGRAPH 92*, In *Comput. Graph.* Proceedings, Annual Conference Series, pages 43–46. ACM SIGGRAPH, 1992.

[17] C. Icking and R. Klein. The two guards problem. *Internat. J. Comput. Geom. Appl.*, 2(3):257–285, 1992.

[18] A. Kaul and J. Rossignac. Solid-interpolating deformations: construction and animation of PIPs. In *Proc. Eurographics '91*, pages 493–505, 1991.

[19] J. R. Kent, W. E. Carlson, and R. E. Parent. Shape transformation for polyhedral objects. In *Proc. SIGGRAPH 92*, In *Comput. Graph.* Proceedings, Annual Conference Series, pages 47–54. ACM SIGGRAPH, 1992.

[20] N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *J. ACM*, 30(4):852–865, 1983.

[21] E. Papadopoulou and D. T. Lee. Efficient computation of the geodesic Voronoi diagram of points in a simple polygon. In P. G. Spirakis, editor, *Algorithms—ESA '95, Third Annual European Symposium*, volume 979 of *Lecture Notes Comput. Sci.*, pages 238–251, Corfu, Greece, Sept. 1995. Springer-Verlag.

[22] T. Sederberg, P. Gao, G. Wang, and H. Mu. 2-d shape blending: an intrinsic solution to the vertex path problem. *Comput Graph.*, 27:15–18, 1993. Proc. SIGGRAPH '93.

[23] T. Sederberg and E. Greenwood. A physically based approach to 2-d shape blending. *Comput. Graph.*, 26:25–34, 1992. Proc. SIGGRAPH '92.

[24] G. Turk and J. F. O'Brien. Shape transformation using variational implicit functions. In *Proc. SIGGRAPH 99*, In *Comput. Graph.* Proceedings, Annual Conference Series, pages 335–342. ACM SIGGRAPH, 1999.