Refolding Planar Polygons

Hayley N. Iben* James F. O'Brien* Erik D. Demaine**
iben@eecs.berkeley.edu *job@eecs.berkeley.edu* *edemaine@mit.edu*

*University of California, Berkeley **Massachusetts Institute of Technology

Abstract

This paper describes an algorithm for generating a guaranteed-intersection-free interpolation sequence between any pair of compatible polygons. Our algorithm builds on prior results from linkage unfolding, and if desired it can ensure that every edge length changes monotonically over the course of the interpolation sequence. The computational machinery that ensures against self-intersection is independent from a distance metric that determines the overall character of the interpolation sequence. This decoupled approach provides a powerful control mechanism for determining how the interpolation should appear, while still assuring against intersection and guaranteeing termination of the algorithm. Our algorithm also allows additional control by accommodating set of algebraic constraints that can be weakly enforced throughout the interpolation sequence.

Categories and Subject Descriptors: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—*Shape Interpolation*

General Terms: Algorithms

Keywords: Polygon interpolation, morphing, shape transformation, refolding.

1 Introduction

In this paper we describe an algorithm for interpolating, or "morphing," between two planar, non-self-intersecting polygons. We assume only that the polygons are simple (no initial self-intersections) and that they form a compatible pair (the same, finite, number of vertices in both polygons). With these assumptions, our algorithm is guaranteed to always find a continuous interpolation path between the two input polygons, and every intermediate polygon along the computed interpolation path is guaranteed to be intersection free.

Our algorithm is flexible in that it can accommodate substantial control over the character of the resulting interpolation sequence through two distinct methods. The first is the specification of a desired distance metric between a polygon pair. The algorithm will greedily move the polygons towards each other by following the gradient of this metric and detouring to avoid intersection. Second, additional algebraic constraints may be specified on the vertices in the intermediate polygons. The algorithm will attempt to stay within the tangent space of the constraint set, breaking constraints only when the constraints become incompatible to the conditions preventing intersection. In the special case, where the constraints require that the edge-lengths change monotonically, we can guarantee that the constraints never conflict with intersection avoidance.

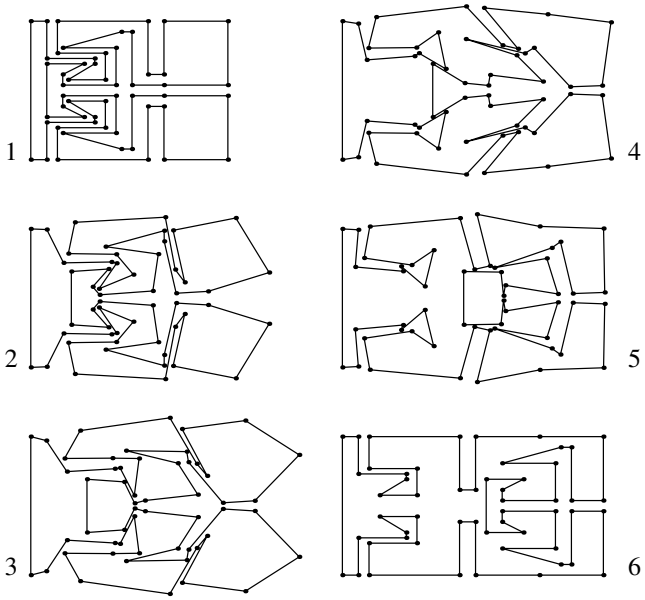

Figure 1. An intersection-free interpolation sequence generated using our algorithm. The first and last frames are the two polygons being interpolated. For this example, all edge lengths were held constant, and the distance metric was the l^2-norm on the vector of vertex positions. The total computation time was 1.6 minutes.

Our technique builds on recent theoretical results from discrete and computational geometry, specifically [8] and [21], which show that any planar collection of polygons and polylines can be "unfolded" to an "outer-convex" configuration. In the case of a single polygon, these results imply that any arbitrary polygon can be continuously deformed into a convex polygon without changing any of its edge lengths and without self-intersection along the way. The motions implied by [8] and [21] are difficult to compute directly, but based on the existence of these motions, other researchers have shown in [5] that a much simpler class of motions can also unfold any collection of polygons and polylines to an outer-convex config-

SCG'06, June 5–7, 2006, Sedona, Arizona, USA.

uration. The simpler motions are easy to compute, corresponding to the downward gradient of a "repulsive" energy function based on the vertex-to-edge distances within the polygon.

Because one can easily interpolate between any two compatible convex polygons (see e.g. [1]), these unfolding results provide an obvious way to build a path from one polygon to another. However, interpolating between two similar polygons by ballooning the first polygon into a convex shape and then folding it back down to the shape of the second polygon is probably not useful in most contexts. This paper builds the theory of polygon unfolding into an approach to polygon refolding that can be used to generate non-intersecting interpolation sequences between any two compatible polygons.

Our algorithm makes use of any valid metric for measuring distances between pairs of polygons. This metric should have the properties of a symmetric norm in the space used to describe polygon configurations, and one simple example is the l^2-norm on the vector of concatenated vertex positions. This metric provides a measure of how "direct" an interpolation is: the most direct interpolation simply follows the metric's gradient exactly. Of course, directness is not the only desire, because gradient descent may cause self-intersection. Our algorithm attempts to greedily find the most direct interpolation path subject to the constraint of no self-intersection. As the gradient descent attempts to build a path interpolating between the two polygons, our algorithm uses the repulsive energy function from polygon unfolding to steer around self-intersections. As demonstrated by Figure 1 and several more figures in Section 6, the appearance of the resulting motion is predominantly governed by the distance metric yet still avoids self-intersection. Although to guarantee convergence of our algorithm we require that "direct" paths follow the gradient of a distance metric, this condition is not strictly necessary for convergence. Any reasonable "direction heuristic" that locally determines how to make only polygon more similar to another would likely also cause our algorithm to converge.

The user can also specify a set of algebraic, or even semi-algebraic, constraints to be satisfied by polygons throughout the interpolation sequence. Our technique satisfies the specified constraints if they are consistent with the requirement of non-intersection. If the constraints cannot be satisfied, we still guarantee non-intersection and satisfy the constraints as much as possible in a locally-greedy least-squares sense. In particular, using the theory of polygon unfolding, we show that the algorithm can always satisfy the constraints of fixing the edge lengths throughout the motion, assuming that corresponding edges have matching lengths in the two polygons being interpolated. More generally, when the edge lengths do not match, the algorithm can force every edge length to change monotonically throughout the interpolation. These constraints in particular often lead to pleasing motions, but the user has the freedom to specify which if any edges should change length monotonically.

2 Background

The task of interpolating between polygons, also called "polygon morphing," is often divided into two subproblems: establishing vertex correspondences and computing vertex paths. In some cases, for example [19] and [6], researchers have focused primarily on establishing vertex correspondences while using a simple method, such as linear interpolation of the vertex positions, to create the intermediate polygons. In this paper, we do not discuss algorithms for finding vertex correspondences. We assume that some other algorithm, or the user, supplies suitable correspondences. So long as the correspondences order the vertices consistently, our interpolation algorithm is guaranteed to succeed.

Other approaches have focused on more sophisticated interpolating schemes for computing vertex paths. In [18], intermediate frames between two shapes are computed by linearly interpolating the vertex angles and the edge lengths, giving better results for rigid transformations than previous work using vertex positions. The authors of [11] create a multiresolution representation for each input polygon. Their algorithm interpolates between these representations to create the intermediate polygons. The method described in [20] decomposes each input polygon into a planar tree of star-shaped pieces, called a star skeleton. The points of the star skeleton, represented in polar coordinates, are linearly interpolated to create the intermediate shapes. In [2], they decompose the input objects into compatible triangulations. They then compute transformations between the triangulations that minimize local distortion. None of these methods guarantee that the intermediate polygons they generate will be intersection-free.

Both [13] and [10] generate non-intersecting sequences for limited types of input. The method in [13] operates on pairs of polygons that have corresponding parallel edges. The method in [10] operates on simple polylines.

A more general method appearing in [12] embeds the polygons inside a convex region, generates a pair of compatible triangulations, and then builds a sequence between them by interpolating the stochastic matrices whose unit eigenvectors encode the triangulations' geometries. In a related approach, [22] use the matrix representations to generate a morphing sequence where the trajectories of the interior vertices can be linear with constant velocities, or as close to linear as possible. This approach enables additional control over the morph, such as forcing the sequence through an intermediate triangulation. In [23], they present a method to generate a more natural-looking morph between compatible triangulations by interpolating the angles and edge lengths when computing the intermediate mean value barycentric coordinates. This enables morphing between two stick figures. Like the method we present here, these methods guarantee that all intermediate polygons will not self intersect, however the types of user control afforded by these systems differs substantially. The character of the motions created by these methods also differs dramatically from that of those generated by our method. Furthermore the methods derived from [12] cannot implement edge-length or other constraints.

Our algorithm ensures that the computed interpolation sequences are intersection-free, and it also decouples vertex correspondence and path computation from intersection avoidance. Intersection avoidance does, of course, affect the vertex paths, but users are free to supply a suitable distance metric to generate whatever type of path they like. The intersection avoidance machinery interferes as needed to prevent intersection. Thus, one could see our method either as an independent interpolation method, or as a wrapper to be used with any of the above methods that generate interesting, but possibly intersecting, vertex paths. For example, [2] produce paths that avoid needless distortion, but that might intersect. If combined with our method, we expect that the resulting algorithm would produce predominantly "rigid-as-possible" motions that distort only as needed to avoid intersection.

In addition to methods that operate directly on explicit polygonal representations, several other methods for interpolating shapes have been described in the literature. For example, both [24] and [7] interpolate between shapes by interpolating scalar fields that implicitly define the shapes. The authors of [14] and [15] discuss methods for interpolating volumetric data. A method based on Minkowski sums appears in [16].

3 Unfolding Groundwork

Our method stems from recent results showing that any planar collection of polygons and polylines can be *unfolded* to an *outer-convex* configuration. In an outer-convex configuration, all polygons or polylines that are not contained inside another polygon are separated from each other, and made either convex (polygons) or straight (polylines). An unfolding motion preserves edge lengths and avoids self-intersection. The existence of these unfolding motions has been demonstrated in both [8] and [21] using two distinct approaches.

While both imply the existence of unfolding motions, actually computing the motions directly implied by these proofs can be difficult. However, the motion implied by [8] has the additional property that it is *strictly expansive*, meaning that the motion strictly increases the distances between *all* vertices not sharing an edge. In [5] it is shown that given the existence of expansive motions, they can reformulate the unfolding problem as one where one simply seeks to minimize a suitable energy function. A suitable energy function is one with the following properties:

Charge — the value of the function is finite for any intersection-free configuration and approaches $+\infty$ as the system approaches self-intersection.

Repulsive — the energy function decreases to first order under any expansive motion.

Separable — as distinct connected components recede from each other, any energy terms relating them should vanish.

$C^{1,1}$ — the function should be C^1 continuous with bounded curvature.

It can then be shown that a simple optimization strategy, such as gradient descent, can be used to generate an intersection-free interpolation path from any polygon to a convex polygon, and that the space of valid configurations contains no local minima to get stuck in. The results also imply that a valid energy function contains no critical points of any kind at non-outer-convex points in the space of valid configurations and that the valid configuration space is simply connected. A detailed convergence proof with step bounds appears in [5], but in summary, for a single polygon:

1. By charge, the energy function is finite for any valid initial polygon and the energy function approaches $+\infty$ as the system approaches self-intersection, so any path that starts with a non-intersecting polygon and strictly decreases energy cannot lead to a self-intersection.

2. By repulsiveness, an expansive direction in configuration space is a direction that decreases the energy, and from [8] we know that such a direction always exists unless the polygon is already convex. Therefore, the gradient can never vanish except for convex polygons, and there can be no local minima that do not correspond to a convex configuration.

Together these two observations guarantee that any continuous gradient descent path starting from any valid polygon will converge to a convexified polygon, and that at no point along the path will the polygon intersect itself.

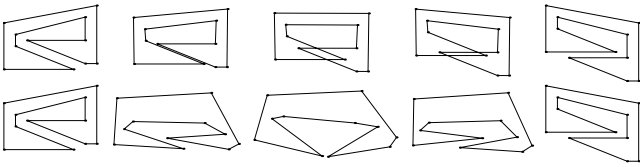

Figure 2. The *top row* demonstrates how using the vertex-position metric alone will, as expected, generate a sequence with self intersections. The *bottom row* illustrates how the collision avoidance machinery alters the vertex motions to avoid self intersection. Computation times were less than one second.

4 Energy and Parameterization

In [5], the authors used an energy function based on the elliptic distance between edges and vertices because a C^2 energy function facilitates placing an actual bound on the worst-case number of Euler steps that might be required to convexify a given collection of polygons and polylines. They also used an angle-based parameterization because it allows them to guarantee that all edge lengths are preserved exactly.

Here, however, we prefer to use an energy based on Euclidean distances because we have found that it converges faster in practice. Additionally, we choose to parameterize using the vertex positions directly and enforce any desired edge-length preservation using algebraic constraints. This decision simplifies interpolation between polygons with different edge lengths, and it also preserves any symmetries by treating all edges equivalently.

For a polygon with N vertices, let $\boldsymbol{v}_i$ with $i \in [1 \dots N]$ denote the positions of the vertices, let e_i be the edge between $\boldsymbol{v}_i$ and $\boldsymbol{v}_{i+1}$, and let l_i be the edge's length[1]. The energy corresponding to the polygon's configuration is given by

$$E = \sum_{i=1}^{N} \sum_{\substack{j=1 \\ j \neq i, j \neq i-1}}^{N} \frac{1}{\operatorname{dist}(\boldsymbol{v}_i, e_j)^2} \tag{1}$$

where $\operatorname{dist}(\boldsymbol{v}_i, e_j)$ is the Euclidean distance between edge j and vertex i. It is easy to verify that this energy function is charge, separable, $C^{1,1}$, and, except for the trivial cases of $N \leq 4$, repulsive.

5 Refolding

Our interpolation algorithm relies on the energy-based unfolding framework to guarantee that it can always construct an intersection-free sequence between any two polygons. In the worst case, the algorithm will convexify both polygons, trivially interpolate between the two convex polygons, and produce the sequence begin-polygon → convexified-begin-polygon → convexified-end-polygon → end-polygon.

In most contexts, this worst-case result is not particularly useful, so the algorithm uses an additional distance metric to generate a more desirable path. Because the energy function provides a guiding framework, this metric can be quite simplistic and still produce good results. In fact, many of the examples shown in this paper were produced using the trivial metric based on the norm of differences in vertex positions. That metric would simply move the vertices on a straight line to their target location. As shown in Figure 2, this metric alone produces intersecting sequences, but it can be guided around intersections by an appropriate energy function.

[1] Index arithmetic is modulo N, so $\boldsymbol{v}_{N+1}$ is equivalent to $\boldsymbol{v}_1$.

We can also include algebraic constraints that should be enforced throughout the interpolation. These constraints could be simply bundled into the distance metric, but then the intersection-avoidance machinery would tend to violate them needlessly. Instead, we combine the projection step that prevents self-intersection with the projections that preserve the user constraints. In the special case where the user constraints seek to make edge lengths constant (or change them monotonically) we can guarantee, based on the previously described unfolding results, that they will not conflict with intersection avoidance. However, arbitrary constraints may conflict with intersection avoidance, so they will only be enforced to the extent that they do not cause the algorithm to fail.

5.1 The Algorithm

The following pseudo-code describes our algorithm for generating an interpolation sequence between two polygons, A and B:

1. Establish compatibility and correspondence:

   The user, or some heuristic, indicates the desired correspondence between A and B and renumbers vertices accordingly. If one of the polygons contains fewer vertices than the other, then additional vertices are inserted by splitting edges.

2. While A and B are different:

   a. Compute the energy for A and B.

   b. Use the gradient of the distance metric to determine a direction, D, that would move the higher energy polygon, H, toward the lower energy one, L.

   c. Optional: Project D to enforce edge-length or other constraints.

   d. If D would move H to a higher energy configuration:

      i. Project D so that it is perpendicular to the energy gradient. (Attempt to honor any constraints if they are in use.)

   e. If D is null:

      i. Set D to the direction of the downward energy gradient at H. (Again, attempt to honor any constraints if they are in use.)

   f. Move H in the direction D.

3. Output the path taken by A to the common configuration and the reverse of the path taken by B.

At each iteration of the while loop, the higher-energy polygon, H, attempts to move closer to the other, lower-energy one, L. The projection step in (2.d.i) ensures that H does not move up in energy and therefore protects against self-intersection. If the direction from H toward L is the same as the upward energy gradient at H, the projection would take D to the null vector[2]. In that case the algorithm simply moves H downward in energy, which we know is always possible from [5].

If we assume that the direction used in step (2.b) to compute D is the gradient of a suitable distance metric, we can guarantee that the above algorithm will always converge. Informally, we note that each iteration of the while loop makes either an "approach" move (bringing A and B closer to one another) or a "descent" move (decreasing the energy of H). The descent moves may undo some of the progress made by approach moves, but the approach moves cannot undo progress made by the descent moves. The algorithm cannot fail to converge by taking an infinite number of descent moves because each decreases the energy toward a minimal value and no moves ever increase the energy. Similarly, the algorithm should not be able to take an infinite number of approach moves because each move decreases the distance between A and B as measured by the distance metric. A sequence of an infinite number of interleaved approach and descent moves continually undoing each other cannot occur because the approach moves cannot undo descent progress.

A more rigorous proof that the algorithm converges is too bulky to include in this short paper. In addition to guaranteeing that the step directions exist and lead to convergence, one must also deal with issues such as avoiding step sizes that converge toward zero. We refer the reader to [5] where they present a rigorous proof that the descent steps do converge in bounded time. They also show that for any two polygons one can establish a minimal step size that does not "pass over" an intersecting configuration, and allows us to assert that our algorithm can always take some minimal size step thus guaranteeing progress and eventual termination. We also suggest [9] for a discussion of the conditions under which descent methods generally converge, and [3] or [17] for a general introduction to relevant numerical methods. For our current *implementation* we have found it sufficient to use a fixed step size that has been selected conservatively by the user.

5.2 A Distance Metric

As described above, the interpolation algorithm is designed to work with a user-supplied distance metric. Given an initial configuration, S, and a target configuration T, the gradient of the metric indicates a direction, D, that moves S closer to T in the space of polygon configurations.

In our implementation, each polygon configuration is represented as a vector of length $2N$ that contains the interleaved x and y coordinates of each vertex. The most obvious distance metric is simply $||T - S||$ so that D is the unit vector in the direction $T - S$. If we were to use this naïve direction alone, the resulting motion would most often include self-intersections. However, when embedded in our energy guided algorithm it generates an interpolation sequence free of self-intersection.

In Section 6 we show results generated using this simple distance metric and with others. The ability to specify an arbitrary distance metric, or even a direction heuristic not explicitly tied to some metric, affords the user with some aesthetic control over the resulting interpolation sequence. The use of a direction heuristic not explicitly tied to some metric could also cause the algorithm to fail. If given the opportunity, the heuristic must cause the two polygons to converge in a finite number of steps. Further, the directions generated by the heuristic should not include any extraneous components or else the energy projection could potentially cancel the useful portion leaving a non-zero vector that might then fail to converge. Alternatively, the direction heuristic could be allowed to include additional spurious components that do not correspond to the gradient of any distance metric, but the condition in step (2.e) should then test to see if the projected vector lacks a component in the direction of the distance metric's downward gradient, rather than just testing whether it is null.

5.3 Energy Projection

To avoid self-intersection, each step must move H to an equal- or lower-energy configuration. This requires that $D \cdot G \leq 0$ where G

[2]Because the gradient of the nonlinear energy function varies over configuration space this situation will occur occasionally.

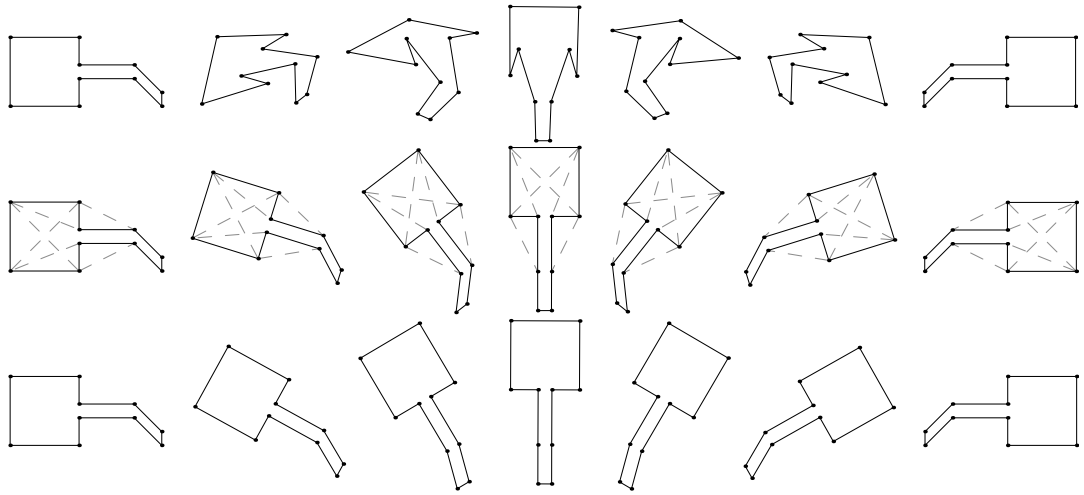

Figure 3. These images show interpolation between a box with an arm-like protrusion and a rotated version of the box with the arm bent. These simple examples demonstrate how the direction metric and constraints can affect the computed sequence. The *first row* shows the result computed using a vertex position metric. The *second row* shows the result for the vertex position metric after several distance constraints have been added. The *bottom row* uses a metric based on joint angles with no constraints. For each row, the edge lengths were held fixed and less than two seconds of computation was required.

is the normalized gradient of the repulsive energy function evaluated at H. The algorithm accomplishes this by testing a candidate direction against the gradient direction. If the dot product is less than or equal to zero, then the direction is left alone. Otherwise, the direction is replaced with

$$D := \left(I - G\,G^{\mathsf{T}}\right) D \tag{2}$$

where I is the identity matrix.

Because the gradient is not constant, a finite sized step following D may still yield an increase in energy even if $D \cdot G \leq 0$. When this condition occurs, we bias D downward by subtracting γG from the direction where γ is a small positive number determined numerically to ensure that the step leads to an equal or lower energy level. This standard technique, commonly used in numerical minimization codes, does not adversely affect our convergence guarantee.

5.4 Constraints

In addition to specifying vertex correspondences and a distance metric, the user can also control the interpolation by specifying constraints that should be satisfied by each polygon in the sequence. One could choose to incorporate user constraints into the direction given by the distance metric, but the energy gradient projection done by Equation (2) would tend to violate the constraints needlessly. Instead, when the user desires constraints we can attempt to satisfy both them and the energy constraint simultaneously. If they cannot all be satisfied simultaneously, then the energy constraint will be satisfied and the user constraints only as much as possible. We treat the energy constraint with higher priority because it is what assures convergence and non-intersection.

We assume that each constraint applies to an individual polygon P, is differentiable, and can be expressed in the form

$$\Omega\,(P) = 0 \quad . \tag{3}$$

For example, we could constrain the edge lengths of a polygon to be constant with

$$\|\boldsymbol{v}_i - \boldsymbol{v}_{i+1}\|^2 - l_i^2 = 0 \quad \forall i \in [1 \ldots N] \tag{4}$$

where the $\boldsymbol{v}_i$ and l_i are the vertex positions and edge lengths of P.

If there are M constraints, let $\boldsymbol{J}$ be the $M \times N$ matrix whose rows are the gradient vectors for each of the constraints. If the initial polygons honor the constraints, then in step (2.c) we can project D to a direction that will not violate them with

$$D := D - \boldsymbol{J}^{\mathsf{T}}\,\boldsymbol{l} \tag{5}$$

where $\boldsymbol{l}$ is solved for using

$$\boldsymbol{J}\,\boldsymbol{J}^{\mathsf{T}}\,\boldsymbol{l} = \boldsymbol{J}\,D \quad . \tag{6}$$

In general, a finite step in this direction would still allow any nonlinear constraints to be violated by a small amount, and this error could accumulate to unacceptable levels if not dealt with. If $\boldsymbol{e}$ is the length M vector whose entries are each of the Ω evaluated at H, then we can prevent error accumulation by instead solving for $\boldsymbol{l}$ using

$$\boldsymbol{J}\,\boldsymbol{J}^{\mathsf{T}}\,\boldsymbol{l} = \boldsymbol{J}\,D + \alpha\,\boldsymbol{e} \tag{7}$$

where α is a small constant. (See, for example, [4] for a discussion of constraint stabilization and how α should be selected.)

As before, if the adjusted direction would move upward in energy, it must be adjusted. However, using Equation (2) could break the projection done by Equation (7) because, in general, G will not be orthogonal to all of the constraints (rows of $\boldsymbol{J}$). To avoid violating the constraints needlessly, let $\boldsymbol{K}$ be the matrix formed by appending G as an extra row to $\boldsymbol{J}$ and let $\boldsymbol{f}$ be the vector formed by appending $-\gamma/\alpha$ to $\boldsymbol{e}$. Step (2.d) sets

$$D := D - \boldsymbol{K}^{\mathsf{T}}\,\boldsymbol{l} \tag{8}$$

where $\boldsymbol{l}$ was solved for using

$$\boldsymbol{K}\,\boldsymbol{K}^{\mathsf{T}}\,\boldsymbol{l} = \boldsymbol{K}\,D + \alpha\,\boldsymbol{f} \tag{9}$$

with some small value used for γ. This value is iteratively increased until a downward energy step results.

Both Equations (7) and (9) can be solved efficiently using the conjugate-gradient method. The matrices $\boldsymbol{J}\,\boldsymbol{J}^{\mathsf{T}}$ and $\boldsymbol{K}\,\boldsymbol{K}^{\mathsf{T}}$ may be under-constrained, over-constrained, or both. When the matrix is over-constrained, not all of the constraints can be satisfied and the conjugate-gradient method will produce a solution that satisfies them all equally in a least-squares sense. Increasing γ causes

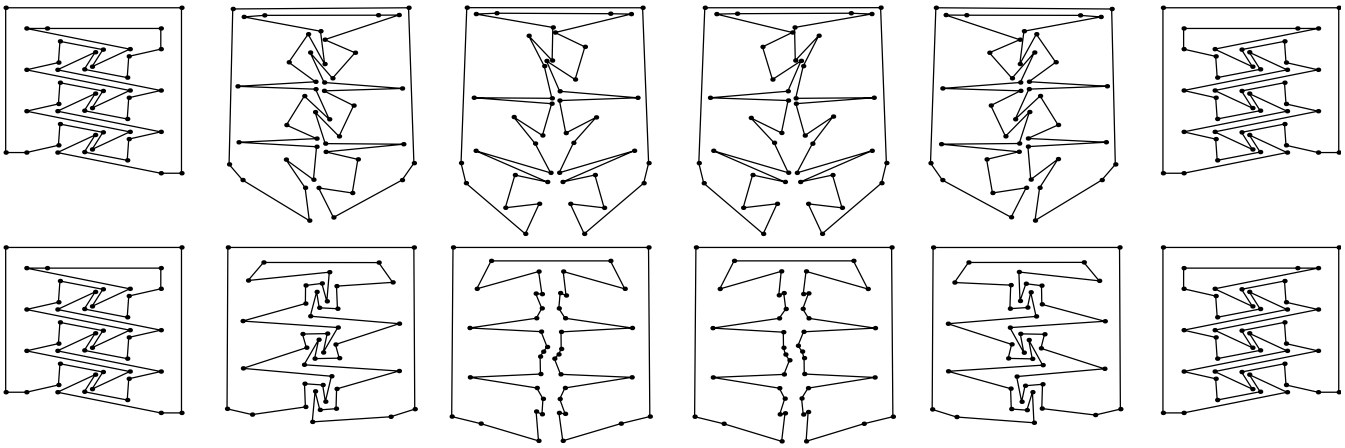

Figure 4. This example interpolates between two configurations of interlocked teeth. The *top row* shows the result computed with the edge lengths constrained to change monotonically and required 5.0 minutes of computation. The *bottom row* shows the result computed with unconstrained edge lengths and required 1.8 minutes of computation.

the energy constraint to have greater importance until it is satisfied. Figure 3 shows a simple example computed with and without additional constraints.

For the special case where all of the user constraints correspond to edge-length preservation, we know from [5] that Equation (9) is never over-constrained because an energy-decreasing motion exists even when the edge lengths are fixed. Thus, for two polygons with the same edge lengths we can always interpolate between them while holding the edge lengths constant. When the polygons have different edge lengths, we can force them to change monotonically by only including the appropriate row of $\boldsymbol{J}$ or $\boldsymbol{K}$ if omitting that row would result in the an edge getting further in length from its target rather than closer. This type of linear-programming approach could also be used to include other semi-algebraic constraints.

6 Results and Discussion

We have implemented our algorithm and used it to create the examples shown in this paper. The accompanying video contains animations corresponding to these examples[3]. Information about the running times and the methods used to create the examples can be found in their respective figure captions. The running times for our C^{++} implementation were measured in CPU seconds on a 3.06 GHz Pentium IV computer with 1 GB of memory.

The rows of images in Figure 3 illustrate the use of different distance metrics. As can be seen in the top and bottom rows, metrics based on the Cartesian coordinates of the vertices and on joint angle coordinates produce very different results. The middle row shows how the motion can be modified by adding additional constraints. In Figure 7, distance constraints were added to maintain the shape of the six arms and outside box and also to keep the inside box rigid throughout the motion. Figure 10 illustrates adding distance constraints to control an animation sequence. Using the directions based on vertex position differences alone, as illustrated in the top row, produces an animation that expands unnecessarily. Adding distance constraints creates a more rigid motion, as shown in the middle and bottom rows.

A feature of the method is that it preserves spatial and temporal symmetries. In Figures 1 and 7, the input polygons are symmetric about a central horizontal axis. It is evident that the animation preserves this symmetry throughout the interpolation. Similarly, the input for Figure 9 is symmetric about a central vertical axis. Figures 3 and 4, both demonstrate animations where the input polygons mirror each other and the method creates temporally symmetric sequences.

Our method also enables the user to choose the behavior of the edge lengths during the animation. The sequence in Figure 1 shows our method with the constraint that edge lengths are held constant. The examples in Figures 4 and 5 illustrate the difference between constraining the edge lengths to change monotonically (top row) or allowing them to change freely (bottom row). For some examples, constraining the edge lengths generated pleasing results. However, in the leaf-plane example the constraint causes an ugly pinch to form in the leaf-stem/plane-tail. Because a "good" sequence depends on the subjective criteria applied by the user, we feel the flexibility afforded by our approach is highly desirable. Other examples using unconstrained edge lengths are pictured in Figures 8 and 9. In Figure 11, we decided based on aesthetic considerations to morph from T to E with constrained edge lengths while the other letters' animations are unconstrained.

Figure 6 shows interpolation between different levels of the two-dimensional Hilbert curve. The large bottom edge connecting the two sides of the curve is changing monotonically throughout the animation, while the rest of the edges are constrained to be constant length. To maximize visibility, configurations are uniformly scaled to give a constant image size.

We can also relax the requirement that steps never increase the energy. As an experiment, we allowed the leaf-plane examples in Figure 5 to take steps that increase the energy up to a threshold. This modified algorithm still avoids self-intersection, but it could potentially fail to converge.

One possible problem with our method is that it uses information, the energy function gradient, that is local to the current polygon. As a result, we cannot guarantee that the path generated is globally optimal in any sense: we can guarantee only that we find a path. In practice, however the algorithm appears to do a good job finding paths that do not detour needlessly. We have experimented with applying relatively expensive optimization procedures to, for example, shorten a computed path as much as possible. So far, we have not observed that this effort produces any significant improvements. These experiments suggest that the computed paths

[3]The video, in QuickTime format, may be accessed on-line at the following URL: http://www.cs.berkeley.edu/b-cam/Papers/Iben-2006-RPP .

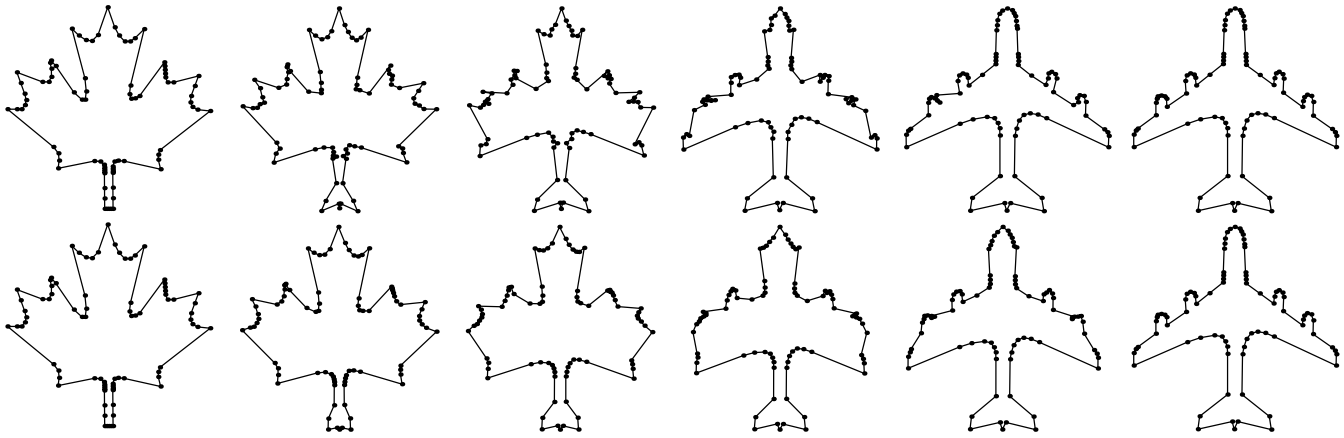

Figure 5. Examples with relaxed energy constraint; see text. In the *top row* constrained edge lengths, 4.1 minutes computation. In the *bottom row* unconstrained edge lengths, 2.0 minutes computation. The leaf and plane outlines were provided by Marc Alexa. Note that these input objects are not symmetric.

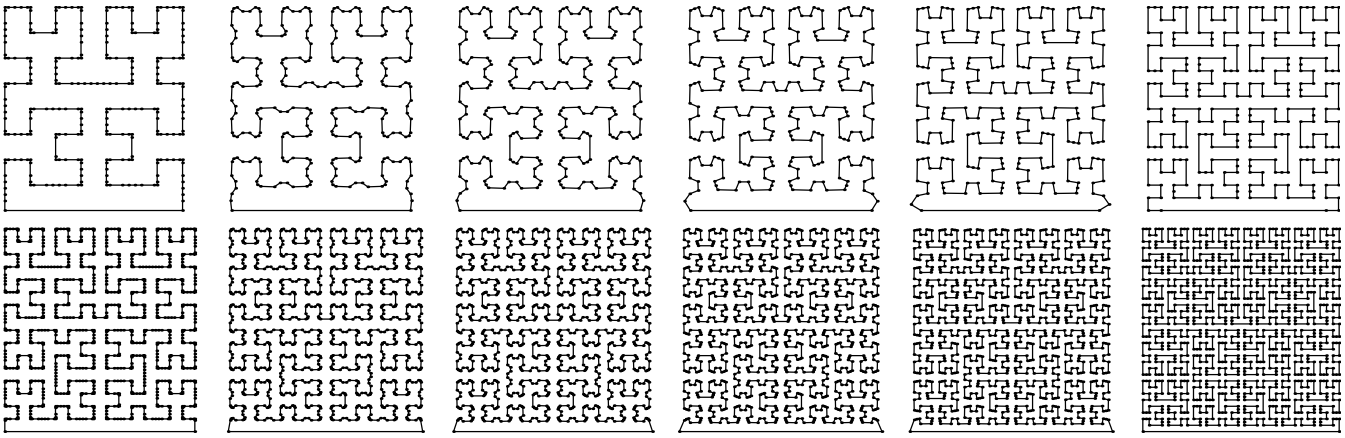

Figure 6. Examples of interpolating with constrained edge lengths between levels of the Hilbert curve. The frames in the *top row* have 260 vertices and computation time was 2.0 minutes. The *bottom row* frames have 1026 vertices and required 1.3 hours of computation.

might be at least locally optimal, at all times greedily minimizing the deviation from the greedy direction given by the distance metric's gradient. It is tempting to wonder whether properties of the energy landscape might mean that locally optimality implies some global property.

The collision avoidance technique presented here provides a method for generating intersection-free interpolation sequences between arbitrary, non-intersecting, planar polygons. We can guarantee that such a path can be found when used with any suitable distance metric or direction heuristic. The examples illustrate that our method can handle a variety of polygons and produce pleasing results. In addition to shape morphing applications in computer graphics, the facility to include length and other constraints may allow our work to be useful for other problems, such as finding efficient, direct motion paths for planar robotic arm manipulators.

There are several directions for improving our results further. Although our C^{++} implementation is robust and fast, using an adaptive time step would likely improve running times. Other areas for future work include exploring interesting direction heuristics and adding other types of constraints to the system. It would also be interesting to explore the extent to which our techniques can be applied to 3D polygons and tree skeletons. In these contexts, interpolation sequences would, in general, be forced to intersect. However, non-intersecting solutions between similar objects might be useful in contexts such as character animation.

Acknowledgments

We thank Jonathan Shewchuk, Jason Cantarella and Carlo Séquin for helpful discussions. Demaine was partially supported by a NSF CAREER award CCF-0347776 and DOE grant DE-FG02-04ER25647. Iben was supported by NSF and GAANN Fellowships. Iben and O'Brien were supported in part by NSF CCR-0204377, State of California MICRO 04-066 and 05-044, and by generous support from Pixar Animation Studios, Intel Corporation, Sony Computer Entertainment America, Apple Computer Inc., Autodesk, the Okawa Foundation, and the Alfred P. Sloan Foundation.

References

[1] Oswin Aichholzer, Erik D. Demaine, Jeff Erickson, Ferran Hurtado, Mark Overmars, Michael A. Soss, and Godfried T. Toussaint. Reconfiguring convex polygons. *Computational Geometry: Theory and Applications*, 20(1–2):85–95, October 2001.

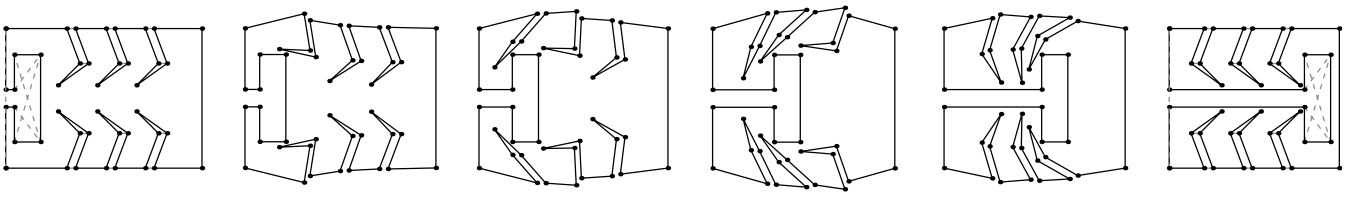

Figure 7. In addition to constrained edge lengths, this example has eighteen distance constraints (drawn light gray in the first and last frames). The total computation time was 1.3 minutes.

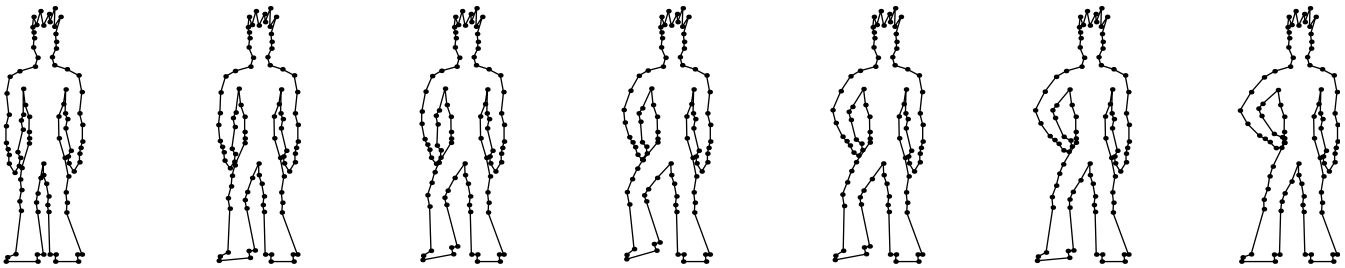

Figure 8. This example was created by generating two successive sequences using three key frames. The keys are shown in the first, center, and last positions. Total computation time was three seconds.

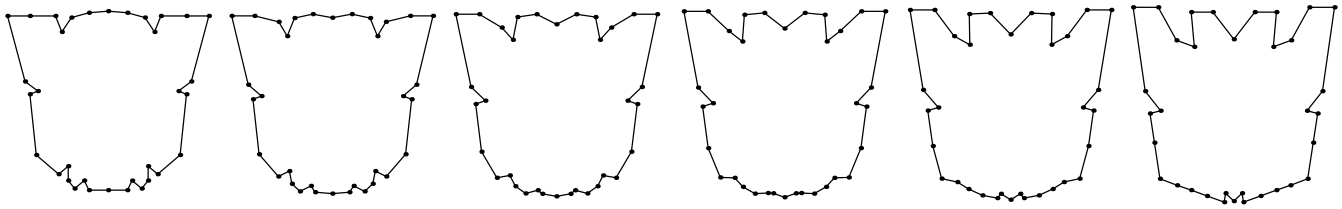

Figure 9. Transforming between two polygons. Unrestricted edge lengths, less than three second computation.

[2] Marc Alexa, Daniel Cohen-Or, and David Levin. As-rigid-as-possible shape interpolation. In *Proceedings of ACM SIGGRAPH 2000*, pages 157–164, July 2000.

[3] Kendall E. Atkinson. *An introduction to numerical analysis*. John Wiley & Sons Inc., New York, second edition, 1989.

[4] J. Baumgarte. Stabilization of constraints and integrals of motion in dynamical systems. *Computer Methods in Applied Mechanics and Engineering*, 1:1–16, 1972.

[5] Jason H. Cantarella, Erik D. Demaine, Hayley N. Iben, and James F. O'Brien. An energy-driven approach to linkage unfolding. In *Proceedings of the 20th annual Symposium on Computational Geometry*, pages 134–143, June 2004.

[6] Eyal Carmel and Daniel Cohen-Or. Warp-guided object-space morphing. *The Visual Computer*, 13:465–478, 1997.

[7] Daniel Cohen-Or, Amira Solomovic, and David Levin. Three-dimensional distance field metamorphosis. *ACM Transactions on Graphics*, 17(2):116–141, 1998.

[8] Robert Connelly, Erik D. Demaine, and Günter Rote. Straightening polygonal arcs and convexifying polygonal cycles. *Discrete & Computational Geometry*, 30(2):205–239, September 2003.

[9] J. E. Dennis and Robert B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. SIAM, Englewood Cliffs, NJ, 1996.

[10] Alon Efrat, Sariel Har-Peled, Leonidas J. Guibas, and T. M. Murali. Morphing between polylines. In *Proceedings of the twelfth ACM-SIAM Symposium on Discrete Algorithms*, pages 680–689, 2001.

[11] Eli Goldstein and Craig Gotsman. Polygon morphing using a multiresolution representation. In *Proceedings of Graphics Interface*, pages 247–254, 1995.

[12] Craig Gotsman and Vitaly Surazhsky. Guaranteed intersection-free polygon morphing. *Computers and Graphics*, 25(1):67–75, 2001.

[13] Leonidas Guibas and John Hershberger. Morphing simple polygons. In *Proceedings of the tenth annual Symposium on Computational Geometry*, pages 267–276, 1994.

[14] Taosong He, Sidney Wang, and Arie Kaufman. Wavelet-based volume morphing. In Daniel Bergeron and Arie Kaufman, editors, *Proceedings of Visualization '94*, pages 85–92, 1994.

[15] John F. Hughes. Scheduled fourier volume morphing. In *Proceedings of ACM SIGGRAPH 1992*, pages 43–46, 1992.

[16] Anil Kaul and Jarek Rossignac. Solid-interpolating deformations: Construction and animation of PIPs. In *Proceedings of Eurographics '91*, pages 493–505, 1991.

[17] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C*. Cambridge University Press, second edition, 1994.

[18] Thomas W. Sederberg, Peisheng Gao, Guojin Wang, and Hong Mu. 2-d shape blending: an intrinsic solution to the vertex path problem. In *Proceedings of ACM SIGGRAPH 1993*, pages 15–18, August 1993.

[19] Thomas W. Sederberg and Eugene Greenwood. A physically based approach to 2-d shape blending. In *Proceedings of ACM SIGGRAPH 1992*, pages 25–34, July 1992.

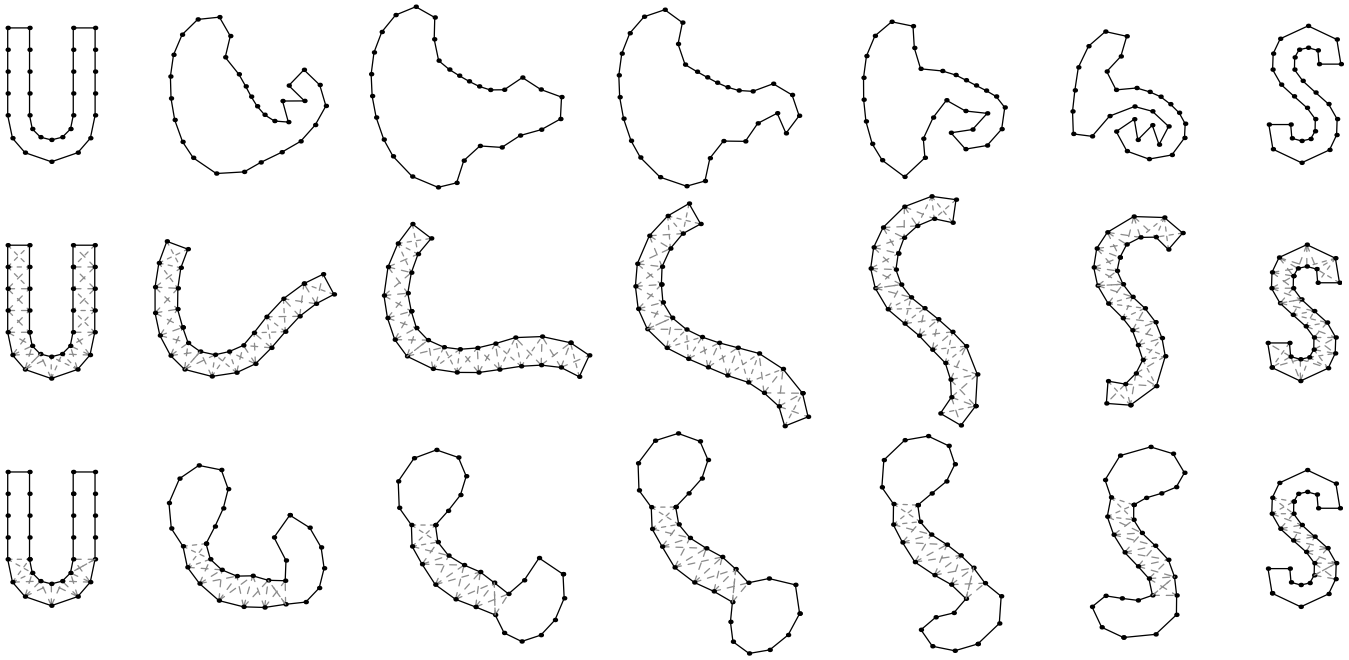

Figure 10. This example interpolates between a letter U and S, demonstrating that adding distance constraints can control the animation sequence. The *first row* shows the result computed using the vertex position metric alone, requiring eighteen seconds of computation time. The *middle row* shows the animation after adding 41 distance constraints to create a more rigid motion, requiring 5.5 minutes. The *bottom row* displays a different motion created using less distance constraints (19), requiring 3.0 minutes. For each row, the edge lengths were constrained to change monotonically. The middle row may be compared to results shown in [12] and [22].

[20] Michal Shapira and Ari Rappoport. Shape blending using the star-skeleton representation. *IEEE Computer Graphics and Applications*, 15:44–50, March 1995.

[21] Ileana Streinu. A combinatorial approach to planar non-colliding robot arm motion planning. In *Proceedings of the 41st annual Symposium on Foundations of Computer Science*, pages 443–453, Redondo Beach, California, November 2000.

[22] Vitaly Surazhsky and Craig Gotsman. Controllable morphing of compatible planar triangulations. *ACM Transactions on Graphics*, 20(4):203–231, 2001.

[23] Vitaly Surazhsky and Craig Gotsman. Intrinsic morphing of compatible triangulations. *International Journal of Shape Modeling*, 9(2):191–201, 2003.

[24] Greg Turk and James F. O'Brien. Shape transformation using variational implicit functions. In *Proceedings of ACM SIGGRAPH 1999*, pages 335–342, August 1999.

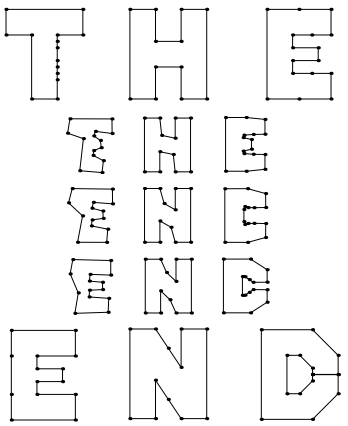

Figure 11. Our final example. Total computation time less than a second.