Compatible Embedding for 2D Shape Animation

William V. Baxter, III, Pascal Barla, and Ken-ichi Anjyo, *Member*, *IEEE*

Abstract—We present new algorithms for the compatible embedding of 2D shapes. Such embeddings offer a convenient way to interpolate shapes having complex, detailed features. Compared to existing techniques, our approach requires less user input, and is faster, more robust, and simpler to implement, making it ideal for interactive use in practical applications. Our new approach consists of three parts. First, our boundary matching algorithm locates salient features using the perceptually motivated principles of scale-space and uses these as automatic correspondences to guide an elastic curve matching algorithm. Second, we simplify boundaries while maintaining their parametric correspondence and the embedding of the original shapes. Finally, we extend the mapping to shapes' interiors via a new compatible triangulation algorithm. The combination of our algorithms allows us to demonstrate 2D shape interpolation with instant feedback. The proposed algorithms exhibit a combination of simplicity, speed, and accuracy that has not been achieved in previous work.

Index Terms—Matching, interpolation, morphing, in-betweening, cross-parameterization, multiscale analysis, scale-space, compatible triangulation.

1 Introduction

While techniques for 3D have great appeal, 2D tools are still used extensively in the creation of various kinds of animation. Some tasks are simply easier in 2D: the tools are simpler, the algorithms are simpler, and the content itself is simpler, and therefore, easier and faster to work with. Examples of ongoing use include digital compositing in CG productions, cel animation and Web graphics, and animated billboards as impostors in complex scenes (e.g., [1]). Nevertheless, there remain many ways in which current 2D techniques can be improved upon.

Recently, the intuitive 2D deformation algorithm from Igarashi et al. [2] was added to Adobe's After Effects to the delight of users. However, despite the many virtues of the technique, it ultimately offers a fairly limited range of deformations. Deforming a shape to match a particular silhouette, for instance, is nearly impossible. A natural evolution would be to allow the user to modify the silhouette directly, or even supply a completely different silhouette, automatically updating the shape's interior accordingly. This can be accomplished using a 2D shape morphing algorithm.

While *image morphing* techniques [3], [4], [5], [6], [7], which apply a warping function to a simple rectangular domain, have been widely deployed in commercial tools, *shape morphing* techniques, where the domain is a complex shape, have been less widely adopted. This is despite the numerous shape morphing techniques which have been proposed in research papers (e.g., [8], [9], [10], [11], [12], [13]).

We believe the lack of adoption can be attributed to lack of a sufficiently simple, efficient, and robust solution to the problem of establishing the correspondence, or mapping, between two irregular shapes. Previous algorithms for matching have required the user to manually specify a large number of correspondences, or made strong assumptions about the nature of the inputs (e.g., similarity of shape, or orientation, etc.) that cause the algorithms to perform poorly when not met ([14], [9], [10]). Other algorithms are simply too slow for interactive use [15], [16], [17], [18]. Many algorithms have also dealt only with inputs which are simple polygons containing a small number of well-defined edges [14], [8], [9], [11], [12], but real-world inputs arising from practical demands are far more complex than this, often containing highly detailed and/or indistinct boundaries, such as clouds, trees, or long hair blowing in the wind.

The main observation motivating our approach is that exact geometric algorithms are infeasible when shapes have such complex boundaries. These details are best handled in image space. Yet, drastic global geometric changes are often desired, and since these are difficult for pure image-space methods, a hybrid approach is needed.

In this paper, we propose a solution to the correspondence problem that is suitable for complex 2D shapes, is fast, and requires less user interaction than previous techniques. The key philosophy behind our approach is to *embed* underlying shapes within loosely approximating boundaries rather than try to exactly match the geometry feature-for-feature. Our contributions lie in three new algorithms that exhibit a combination of simplicity, speed, and accuracy that has not been achieved in previous work (see Section 2):

- A *boundary matching algorithm* (Section 3) that finds correspondences by quickly locating and optimally matching salient features on shape boundaries: It allows an artist to establish boundary correspondences with significantly less manual input than existing techniques.
- A *compatible simplification algorithm* (Section 4) that efficiently simplifies boundaries while maintaining

- *W.V. Baxter, III and K.-i. Anjyo are with OLM Digital, Inc., Room 302, 1-8-8 Wakabayashi, Setagaya-ku, Tokyo 154-0023, Japan. E-mail: wbaxter@gmail.com, anjyo@olm.co.jp.*
- *P. Barla is with the INRIA Bordeaux University, 351 cours de la Libération, 33405 Talence Cedex, France. E-mail: pascal.barla@labri.fr.*

Manuscript received 23 June 2008; revised 6 Nov. 2008; accepted 4 Mar. 2009; published online 24 Mar. 2009.
Recommended for acceptance by G. Taubin.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number TVCG-2008-06-0079.
Digital Object Identifier no. 10.1109/TVCG.2009.38.

their parametric correspondence and embedding the full-resolution input shapes: It decouples the complexity of the final embedding from the complexity of the inputs.
- A *compatible triangulation algorithm* (Section 5) that extends the boundary correspondence to cover shapes' interiors: It is not only more efficient than all previous methods on practical inputs, but also significantly simpler than the previous best algorithms.

We believe that the combination of these three algorithms is of significant importance for the widespread adoption of recent interpolation techniques [11], [19], [20].

2 Previous Work

Much of the work in shape morphing falls into two categories: 2D morphing of images [5] and 3D morphing of meshes [21]. 2D shape morphing lies in between these two, being both two-dimensional, like image morphing (e.g., [3], [4]), while having geometry which must be taken into account, as in mesh morphing (e.g., [22], [23], [24]). Our approach takes full advantage of the 2D nature of the problem, in a way similar to previous work such as [9], [10], [11], [12].

Shape matching. Elastic distance methods have been a popular way to match shape boundaries (e.g., [14], [10], [25]). These techniques are fast and simple, but require parameters to balance curvature error against stretching error. It is difficult or impossible to find parameters that work well for a wide variety of inputs. Moreover, they work poorly in the presence of noise (see Fig. 2c). In the literature on content-based image retrieval [26], a number of methods based on the medial axis have been proposed (e.g, [15], [18]); however, the medial axis is sensitive both to noise and to slight variations in shape, and methods for robust matching based on the medial axis are slow and difficult to implement.

Curvature scale-space is an effective way to deal with noise, and sophisticated algorithms have been proposed using both a full scale-space [16] and just feature points [17]. However, these methods are also too computationally demanding for interactive applications. Our approach combines the simplicity and efficiency of elastic distance with the robustness of scale-space feature-based methods.

Shape simplification. Many techniques have been developed for geometric simplification of 2D lines and polygons (e.g., [27], [28], [29]). These methods find an approximation that minimizes deviation in terms of euclidean or perceptual metrics. However, we seek not only to minimize deviation, but also to completely enclose, or embed, the full-resolution input shapes while additionally preserving their parametric correspondence.

For 2D boundaries, Tal and Elber [10] perform compatible simplification by subsampling, with the predictable result that embedding is lost. In [11], [22], meshes are simplified compatibly, but the former does not simplify the boundary and the latter does not embed. The *progressive hull* of [30] provides an embedding simplification for 3D meshes that is similar to our method, though it only applies to a single input and is based on edge collapses rather than vertex removals. Our vertex removals lead more naturally to a compatible, parameterization-preserving simplification. We are not aware of any previous simplification algorithms, which both generate an embedding and work compatibly on multiple inputs. We believe our algorithm to be the first such algorithm.

Shape triangulation. Previous methods for compatible triangulation generally either map polygons onto a common domain [31], [10], [11], or use divide-and-conquer compatible partitioning [32], [12]. The first category is conceptually simple but not optimal in terms of runtime ($O(n^3)$) or in number of Steiner vertices introduced ($O(n^2)$). The latter methods are better ($O(n^2 \log n)$ time and often no Steiner vertices), but are algorithmically quite complex.

Our algorithm uses recursive partitioning like the latter. Efficiency is similar ($O(n^2)$ on typical inputs, $O(n^3)$ generally), and it also generates outputs with near-minimal Steiner vertices, often zero. Yet it is significantly simpler. Though the Surazhsky-Gotsman algorithm [12] is already a simplification of a previous algorithm [32], it still requires implementation of many nontrivial data structures and algorithms described in [33], [34], [35], [36] in addition to the $O(n)$ triangulation of either [37] or [38]. In contrast, our algorithm requires no subroutine more complex than vertex visibility [39], [40], for which code can readily be found on the Web (e.g., [41]). Furthermore, based on our experience, the reliance on only very simple geometric visibility calculations also appears to have the merit of making our algorithm more robust in the face of complex inputs.

3 Boundary Matching

Our approach to boundary matching is a compromise between the robustness of curvature scale-space analysis and the efficiency of elastic curve distance methods. It works in two steps. **First**, we locate the set of *feature points* along each shape boundary by extracting stable curvature extrema. This is done by analyzing extrema "signatures" through scale-space. **Second**, we use a new feature-based elastic curve distance, which finds *automatic correspondences* between feature points on each shape boundary.

Compared to previous work, the first step is faster due to our simple signature measure, while being easier to implement and similarly robust to noise. The second step is as fast as previous methods, but requires no complex parameter tuning, less user input, and is more accurate since the matching is guaranteed to go through feature points.

3.1 Finding Feature Points

Our first step is to find feature points on each shape to serve as candidate correspondence points (Fig. 1). These points are *curvature extrema*, which represent boundary concavities (minima) or convexities (maxima). However, noise and detailed variations in line style tend to create many such extrema that do not correspond to an intuitive notion of "feature" (Fig. 1a). Smoothing can help with the noise problem, but the difficulty is in choosing a proper scale for the smoothing.

A robust alternative consists of analyzing a boundary curve smoothed at multiple scales to find stable feature

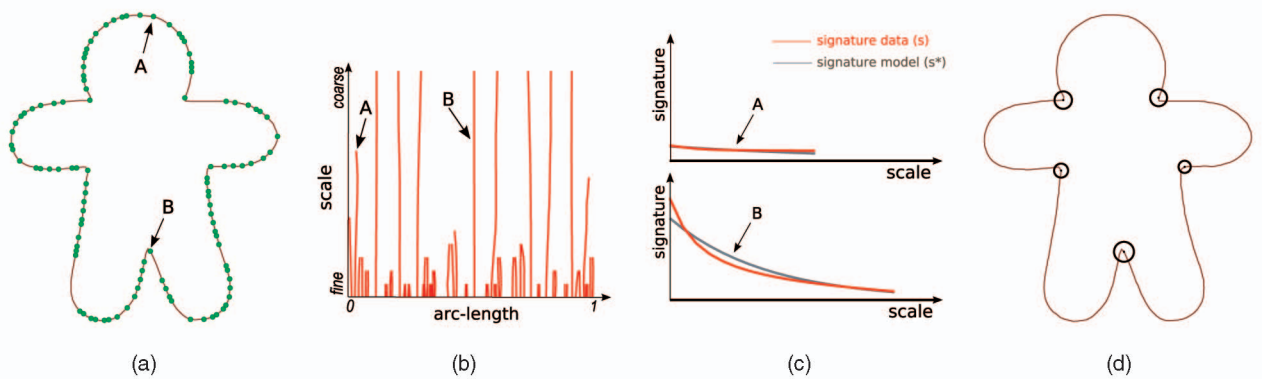


Fig. 1. **Boundary analysis:** (a) A boundary extracted from an image: discretization causes many fine-scale curvature extrema (green dots). (b) The fingerprint image of curvature extrema across scales. Stability of extrema A and B cannot be determined from this fingerprint data alone. (c) Comparing signatures: B (bottom) exhibits behavior characteristic of stable extrema; A does not. (d) Circles indicate feature points extracted by our signature analysis, with radii proportional to feature strength.

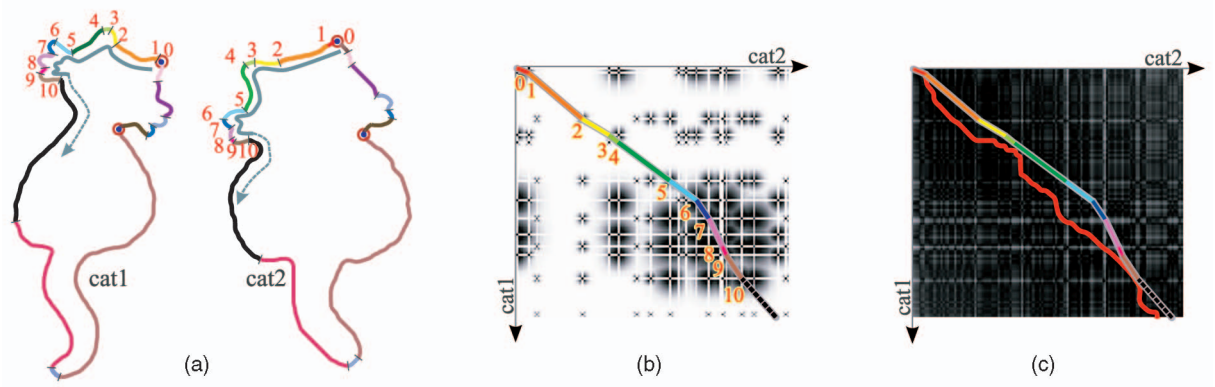


Fig. 2. **Boundary matching:** (a) Two input shapes and their boundary correspondence computed with our matching algorithm. User correspondences are circled in red, while automatic correspondences are identified by numbers and delimit boundary segments of similar color. (b) Cost matrix of our approach for a subset of the boundary: dark blobs represent attractors around potential correspondences and the matching path goes *exactly* through curvature extrema. (c) Cost matrix for curvature-based matching: the path (in red) does not go exactly through the desired correspondences and is sensitive to noise.

points. This is the approach taken in scale-space methods, where stable feature points are found by tracking boundary extrema at increasing scales. In the original approach of Mokhtarian and Mackworth [16], the tracked extrema are inflection points (zero-crossings of curvature). More recently, Zabulis et al. [17] have argued that curvature extrema lead to more intuitive feature points; they show how the evolution of these extrema through scale reveals the most important structures of a shape's boundary. Fig. 1b shows such an evolution with a so-called "fingerprint" image, where each curve corresponds to a single extremum.

While these methods deal with noise robustly, they are also quite slow, requiring up to a minute to find features in our tests. This is because they detect stable features by computing the scale-space out to very coarse scales. Our key observation is that stable extrema have a specific behavior along the scale dimension: their curvature starts with a peak, and is progressively smoothed out (Fig. 1c). We call this the extremum's *signature* function. We assert that the signatures of stable extrema resemble *decaying exponentials* (Fig. 1c, bottom). This can be understood by noting that sharp boundary features resemble delta functions in a curvature plot, and thus, decay exponentially with increasing scale. In contrast, unstable extrema typically have flat or nondecaying signatures (Fig. 1c, top).

Using our signature analysis, we can detect stable features reliably at much finer scales. Moreover, the analysis also produces a measure of *feature strength* that is valuable for the subsequent matching step of our algorithm. In all the examples found in the paper, we built the scale-space from scale 0 up to a maximum scale of 10, with scale increments of 0.5. Using these parameters, we were able to robustly extract features in under a second. Increasing the maximum scale or decreasing the increment did not significantly improve the quality of the results.

Computing signatures. We begin with a densely sampled, closed polyline with uniform arc-length parameterization. This can be extracted from an image using standard techniques. We then build a curvature scale-space and track curvature extrema at increasing scales as in [17]. As proposed in [16], we extract extrema only in the vicinity of extrema found at the previous finer scale, which significantly speeds up the process. Our contribution is in the final step where we analyze signatures to determine feature strengths and reject weak extrema, as explained below.

To decide whether an extremum is a good candidate feature point, we analyze its signature function. For feature extraction, we use curvature magnitudes, disregarding sign. To ease the analysis and improve numerical stability, we map curvature magnitudes from $[0..\infty)$ to $[0..1)$ using the

$\tanh$ function. The particulars of the remapping function are not critical since ultimately we only consider curvature magnitudes *relative* to one another (see Section 3.2). We thus take an extremum's signature data to be $s(\sigma_i) = |\tanh c(\sigma_i)|$, where $c(\sigma_i)$ is the curvature measured at the discrete scale σ_i (red curves in Fig. 1c).

In order to compare a curvature extremum to the ideal, we fit the measured data to an exponential signature model of the form $s^*(\sigma) = \exp(ax + b)$. We immediately reject any extrema with $a \geq 0$ (Fig. 1c, top). We fit using a weighted least-squares minimization of an error function of the form $e(a,b) = \sum_{i=1}^{N} w(\sigma_i)(ax + b - \log s(\sigma_i))^2$, with $w(x) = ((N - x)/N)^2$, so that values near the minimum scale (where the shape is less smoothed out) have more influence. As mentioned, we use a to reject unstable extrema, and use b to determine the feature strength. The feature strength f is defined by its peak value, the value at $x = 0$, i.e., $f = s^*(0) = \exp(b)$. Fig. 1d shows the feature points extracted with our approach. The radius of each circle locator is proportional to the strength of the corresponding extremum.

3.2 Extracting Automatic Correspondences

From the multiscale analysis, for each shape, we get a set of feature points along with their strength and sign of curvature (indicating convexity). We normalize feature strengths by dividing the maximum strength value over the shape pair, and we use a small threshold ($f_{\min} = 0.05$) to discard extrema that obviously come from noise, but we keep most of the extrema. The main purpose of our elastic matching algorithm is then to make use of this information to *automatically* extract meaningful correspondence points. The full boundary mapping is obtained by establishing a uniform parameterization in-between each successive pair of correspondences (Fig. 2). The only information a user need specify is the first pair of correspondence points.

The problem of finding a boundary mapping is then solved using a dynamic programming approach: a cost matrix is built where each cell stores the dissimilarity between every possible pair of points on each boundary, and the dynamic programming algorithm finds a *matching path* of least cost (Figs. 2b and 2c) that links the first pair of points to the last pair (i.e., user-given correspondences). Classic curve matching techniques (e.g., [14], [10], [25]) set up a cost matrix where each individual cost is based on an arc length, curvature, or orientation dissimilarity (Fig. 2c). Instead of using raw curvature, orientation, or arc length as features to be matched, we use the feature strength derived in Section 3.1.

We require that both features be strong in order to consider the pair a potential correspondence. We turn this constraint into a *confidence* measure defined by $c(p,q) = \min\{f(p), f(q)\}$, where p and q are two points on different boundaries, and f is the feature strength. The $\min$ function acts as a fuzzy AND operator. Moreover, we enforce $c(p,q) = 0$ whenever p and q are not feature points or have opposite signs of curvature. Our goal is then to make use of this confidence measure to build a cost matrix, with the additional constraint that the matching runs *exactly* through automatically selected pairs of feature points, and that each feature point is used *at most once*.

Building the cost matrix. The intent is that the higher the confidence value, the more likely the matching will go through it. This is translated to a cost measure by first filling the cost matrix with a maximum cost of 1 for each possible pair of points, and then positioning inverted Gaussian blobs everywhere $c(p,q) \neq 0$. These blobs act like "attractors" as they reduce costs toward 0 around the center of the Gaussian (Fig. 2b). The standard deviation of a Gaussian blob corresponds to the attraction strength of a candidate correspondence, so we make it proportional to the confidence measure: $\sigma(p,q) = \frac{1}{K} c(p,q)$. The constant K is used to normalize attractors' influence, and we make it equal to the minimum of average confidences in each shape $K = \min\{\frac{1}{n}\sum_p f(p), \frac{1}{m}\sum_q f(q)\}$. This heuristic proved to work well in all our examples. Moreover, to ensure that the matching runs exactly through selected correspondences, we penalize point pairs where only one point is a feature, setting the cost to 1. This corresponds to the white horizontal and vertical lines in Fig. 2b.

Comparison of matching paths in Figs. 2b and 2c demonstrate the advantage of our boundary matching procedure: it extracts significant correspondences more accurately while requiring no parameter tuning and being fast enough for interactive editing. Perhaps more importantly, in tests, we were always able to find a satisfactory matching with our algorithm, which was not the case with the other algorithms we evaluated. More results and comparisons can be found in Section 6.

4 Compatible Boundary Simplification

The output of the matching algorithm of the previous section is a densely sampled pair of corresponding boundary polygons. For generating the subsequent compatible tessellation, it is preferable to work with a simplified version of this matching, but the simplification must maintain parametric correspondence while still completely enclosing the original shapes. To satisfy these requirements, we present a new algorithm for compatible boundary simplification (Fig. 3).

The key idea is that when vertices are removed through simplification, we restore the local embedding by moving neighboring vertices along their normals. In particular, for maximal efficiency, we consider moving only the immediate neighbors of a removed vertex. Moving normal to the boundary is the key element that serves to preserve the parametric identity of each point.

We follow the general structure of progressive mesh simplification [42], assigning costs to each simplification operation—in our case vertex removals—and using a priority heap with greedy selection to generate a simplification sequence. The cost of each vertex removal operation is taken to be the change (increase) in polygon area that will result (see Fig. 4). For the overall compatible simplification cost, we take the maximum cost of the two (inputs are normalized to have approximately the same scale). Once the removal sequence is computed, the heap

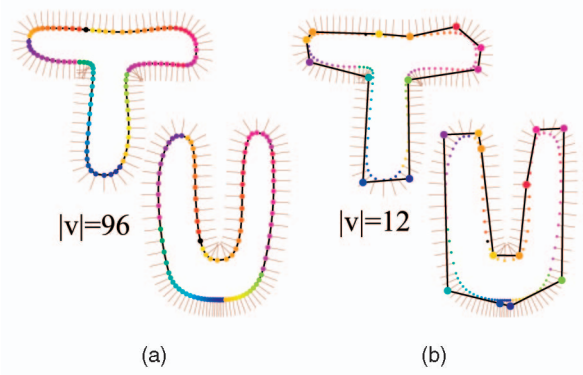
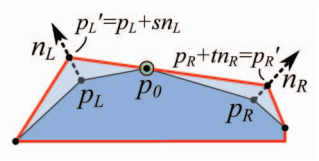


Fig. 3. **Compatible simplification:** A result from our compatible, embedding, parameterization-preserving boundary simplification. (a) Initial shapes and (b) simplified shapes.

may be discarded, and the recorded collapse sequence can be used to quickly switch between different simplification levels. In practice, the time spent for simplification is negligible in the overall pipeline, but it significantly speeds up the subsequent triangulation.

Restoring local embedding. After removing a vertex, p_0, we need to restore the embedding property for convex vertices. Concave vertices as in Fig. 4a can simply be removed. Figs. 4b, 4c, 4d, and 4e show several convex examples. As mentioned, we restore embedding by perturbing the immediate neighbors, p_L and p_R, along their unit normals, n_L and n_R, so that new vertex positions are $p'_L = p_L + sn_L$ and $p'_R = p_R + tn_R$. This gives us a two-parameter search space (s,t) for "good" vertex positions. We choose to place the vertices so as to equally distribute the amount of perturbation between p_L and p_R (i.e., make $s = t$), subject to the constraint that the segment $\overrightarrow{p'_L p'_R}$ intersects p_0. This leads to a quadratic equation in s



$$0 = s^2(n_R \cdot n_L^{\perp}) + (p_L - p_R)\cdot(p_0 - p_L)^{\perp} - s[(n_R - n_L)\cdot(p_0 - p_L)^{\perp} + (p_L - p_R)\cdot n_L^{\perp}],$$

where $(x,y)^{\perp} = (y,-x)$. The minimum positive root is the solution sought. If no positive root exists, it means the normals are separated by an angle of greater than 180 degrees. In these cases, we relax the normal direction constraint. We do this by turning both normals inward by a fixed amount (we use a step size of 45 degrees) and try the procedure again. Each time we turn the angles inward, we add a large penalty to this vertex's removal cost to ensure we use collapses that do not require going off-normal first.

Modifications. First, we can obtain an *inscribed simplification* (right) simply by reversing the direction of the boundary normals used. Second, the algorithm trivially generalizes to *N-way compatible simplification* by taking the maximum of N vertex costs rather than of two.

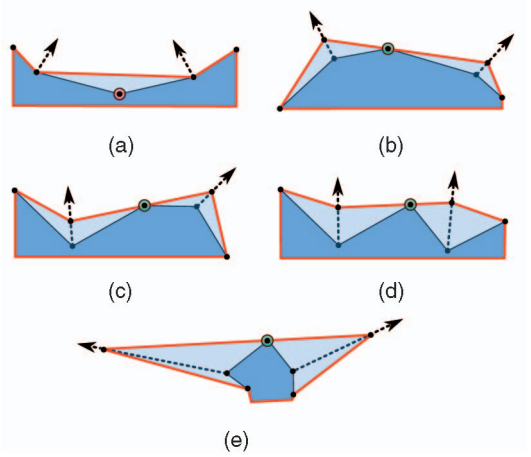
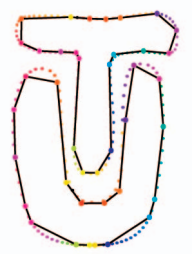


Fig. 4. **Simplification via vertex removal:** The original polygon (dark blue) and boundary after removal (red). The cost of a removal is taken to be the area added by the operation. Highly convex vertices like (e) are poor candidates for simplification, and the area cost metric reflects this.



5 Compatible Triangulation

We establish interior correspondence using compatible triangulation. Given two polygons with compatible boundaries as input, we recursively and compatibly partition the two polygons using *compatible link paths* as in [32], [12]. Recursion on a branch terminates when both inputs are triangles, which have a trivial linear mapping. We introduce a new algorithm for finding compatible link paths, which is significantly simpler than the previous algorithms and also more efficient on typical inputs.

Terminology. A link path is an interior polyline that joins two vertices, and compatible link paths are two such paths which join corresponding pairs of vertices in two polygons. The *link distance* is the number of line segments in a link polyline. As a shorthand, we refer to a link path with a link distance of k as a "k-link." A k-link requires the introduction of $k-1$ Steiner vertices. The goal is to triangulate by finding compatible partitions with minimal k. It is beneficial to minimize the number of Steiner vertices because it reduces not only the runtime and memory usage of the algorithm itself, but also the runtime of any subsequent mesh processing, such as nonlinear optimization of parametric stretch (e.g., [43], [22]), which can be a significant bottleneck in the overall processing pipeline.

Our algorithm works in three stages. **First**, we compute per-vertex visibility polygons and use these to find compatible 1-links. **Second**, if no compatible 1-link is found, we look for compatible 2-links. These can be found by looking for *intersections* of the vertex visibility polygons. **Third**, in the rare case that neither a compatible 1- or 2-link exists, we resort to a

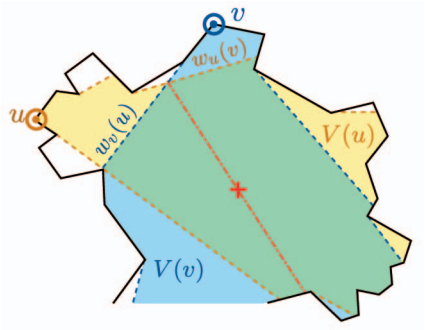


Fig. 5. **Visibility polygons and windows:** The visibility polygons $V(u)$ and $V(v)$ for vertices u and v and the *windows* $w_u(v)$ and $w_v(u)$ are indicated. The red "+" indicates where our algorithm places the Steiner vertex for this case.

general k-link finding mechanism which builds pseudooptimal k-links out of combinations of the existing 1- and 2-links by solving the classic all-pairs-shortest-paths graph problem.

Stage 1. As a first step, we compute the visibility polygon of each vertex in each polygon [39], [40]. A visibility polygon is the set of points, all of which have direct line-of-site from some given feature (see Fig. 5). A vertex visibility polygon can be computed in $O(n)$ for each of n vertices, so this step requires $O(n^2)$ work. We also modify the basic algorithm to simultaneously record the complete $n \times n$ visibility matrix for each polygon as it proceeds, making later determination of direct visibility between any two vertices in $O(1)$ operation. In fact, we record not just visibility in these matrices, but best-estimate minimal link distances between pairs of vertices as we find them, for use in stage 3.

Specifically, let the two link distance matrices be D^0 and D^1. At this stage, we record a 1 in any element D^0_{ij} or D^1_{ij} for which vertices i and j have direct visibility in the corresponding shape, and $\min(|i-j|, n-|i-j|)$ for those which do not. The latter is simply the distance from i to j walking along the boundary in the shorter direction. We refer to these as *trivial* k-links. Partitioning using these is not generally useful, and so it is to be avoided. But they serve as an upper bound on the link distance between any two vertices.

The approach is greedy. If we can find a corresponding pair of vertices in each polygon that have direct visibility (i.e., the pair is a 1-link in both polygons), then we use it. Given the visibility matrix from the preprocess, all pairs can be easily checked in $O(n^2)$ time. Often, there are many compatible 1-links. In this case, like [12], we choose a pair that results in the most balanced split of the polygons in terms of number of vertices on each side.

For practical inputs, a compatible 1-link can almost always be found; however, Steiner vertices must sometimes be added to one or both polygons to create a compatible partitioning (see Fig. 8, or Fig. 6 for a particularly challenging case). If no compatible 1-links are found, then we proceed to stage 2.

Stage 2. The second stage of our algorithm looks for vertex pairs (u, v) with minimal link distance of two, or 2-links. 2-Link visibility means that there is some point q in polygon P which is visible from both vertex u and vertex v. The locus of such points is the intersection of the visibility polygons for u and v. Thus, all 2-links could be found by calculating the

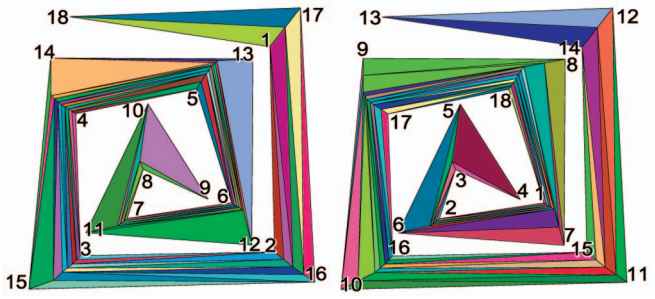


Fig. 6. **Compatible triangulation torture test:** Results of our algorithm on shapes that are difficult to triangulate compatibly due to necessity of k-links, $k > 2$. Numbers indicate vertex correspondences; colors indicate triangle correspondences. Thirty two Steiner vertices were added.

intersections of the vertex visibility polygons. However, polygon intersection, in general, requires $O(n^2)$ time, so this would lead to $O(n^4)$ complexity to test all pairs. Fortunately, for this case, we can test for intersection in $O(1)$ by simply intersecting two particular edges.

These two particular edges are the so-called *windows* that separate u from v, inside P (Fig. 5). A window $w_u(v)$ is an edge of the visibility polygon $V(u)$ that separates vertex u from vertex v. This edge exists whenever u and v do not have direct line-of-sight visibility in P, and is always unique [34], [33]. The same holds for $w_v(u)$. Using these facts and some other results from Suri, we prove the following lemma in [44].

Lemma 1. *The two windows $w_u(v)$ and $w_v(u)$ intersect iff there exists a minimal 2-link path between u and v.*

Lemma 1 is the key insight that gives us an $O(1)$ algorithm for determining whether two visibility polygons intersect: They intersect if and only if $w_u(v)$ and $w_v(u)$ do. This leads to an overall $O(n^2)$ runtime for stage 2. Note that the windows separating each pair of vertices are easily found as a by-product of the vertex visibility algorithm run in stage 1. If a compatible 2-link is found, we place the Steiner vertex along the ray that bisects the angle between the two window edges (Fig. 5).

While performing this search, we also update the visibility matrices from the first stage with the newly acquired 2-link visibility information. If no compatible 2-links are found, then we proceed to stage 3.

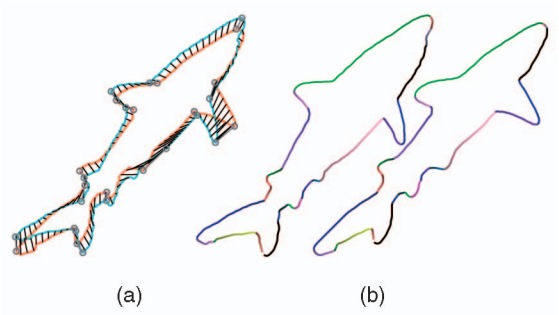


Fig. 7. **Matching comparison:** A comparison with a matching result taken from Zabulis et al. [17] (a) versus our algorithm (b). No manual correspondences were specified. Our algorithm obtains very similar results but in a fraction of the time.

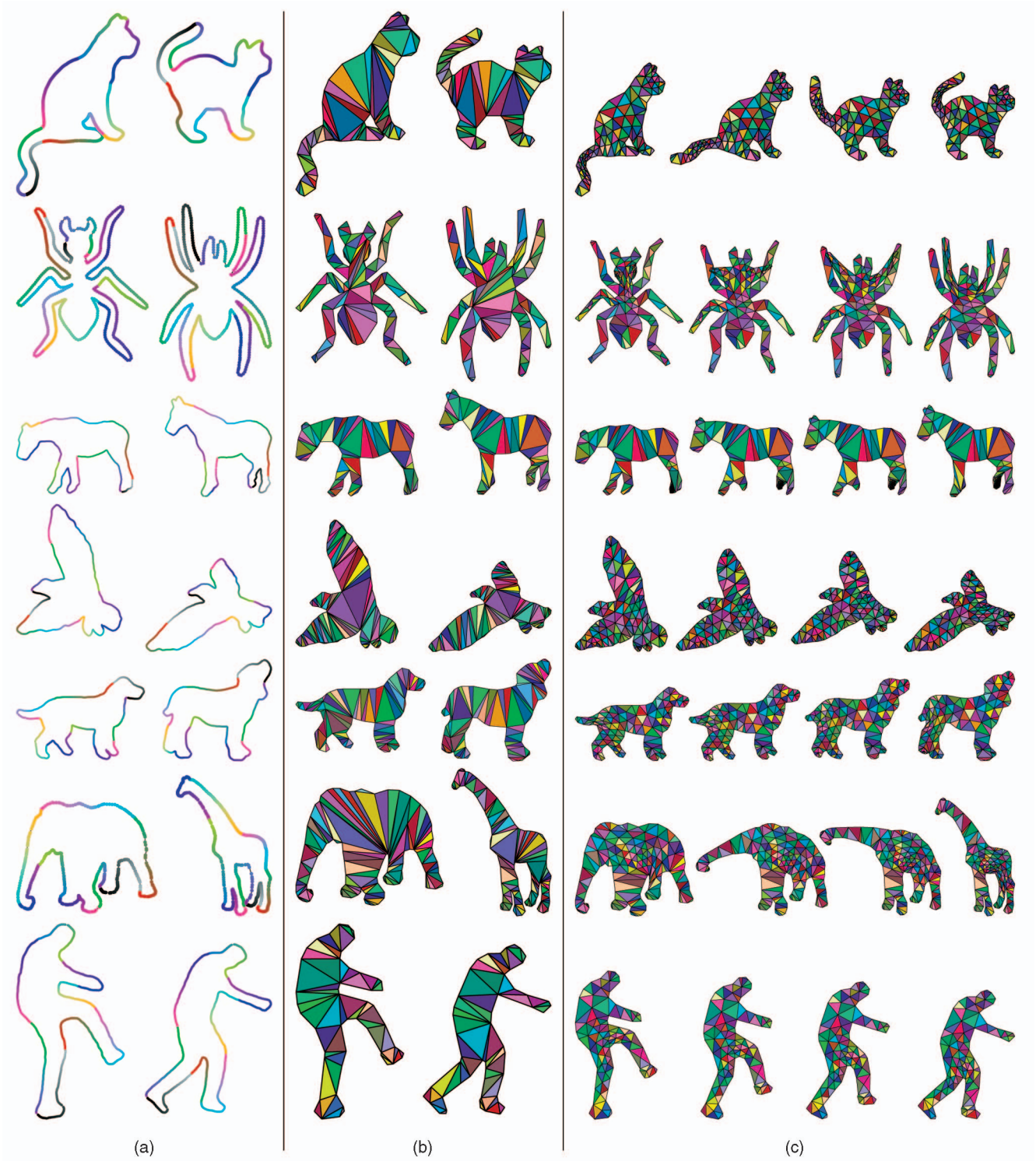


Fig. 8. **Embedding results:** (a) Boundary matching, (b) compatible triangulation of compatibly simplified boundaries, and (c) mesh refinement and morphing. Note that how our compatible triangulation handles extreme stretching (e.g., spider shape, second row). From top to bottom, these examples required two, five, two, two, four, four, and one manual boundary correspondences, and no interior correspondences. The initial tessellations required zero, three, four, four, four, two, and zero Steiner vertices.

Stage 3. The main idea is to find compatible k-links, where $k > 2$, by simply combining the existing 1- and 2-link paths. If we can get from u to w via a 2-link and w to v using a 1-link, then we know we can get from u to v in three steps. This is the description of a classic graph distance problem. Our link distance matrix encodes the initial distances in the graph, and we wish to solve the all-pairs-shortest-paths problem on that graph. The Floyd-Warshall algorithm finds these in $O(n^3)$.

Once a nontrivial pair of compatible k-link paths is found, we place the Steiner vertices. The path consists of a combination of 1-link and 2-link portions. We first place the central vertex of the 2-link portions using the technique

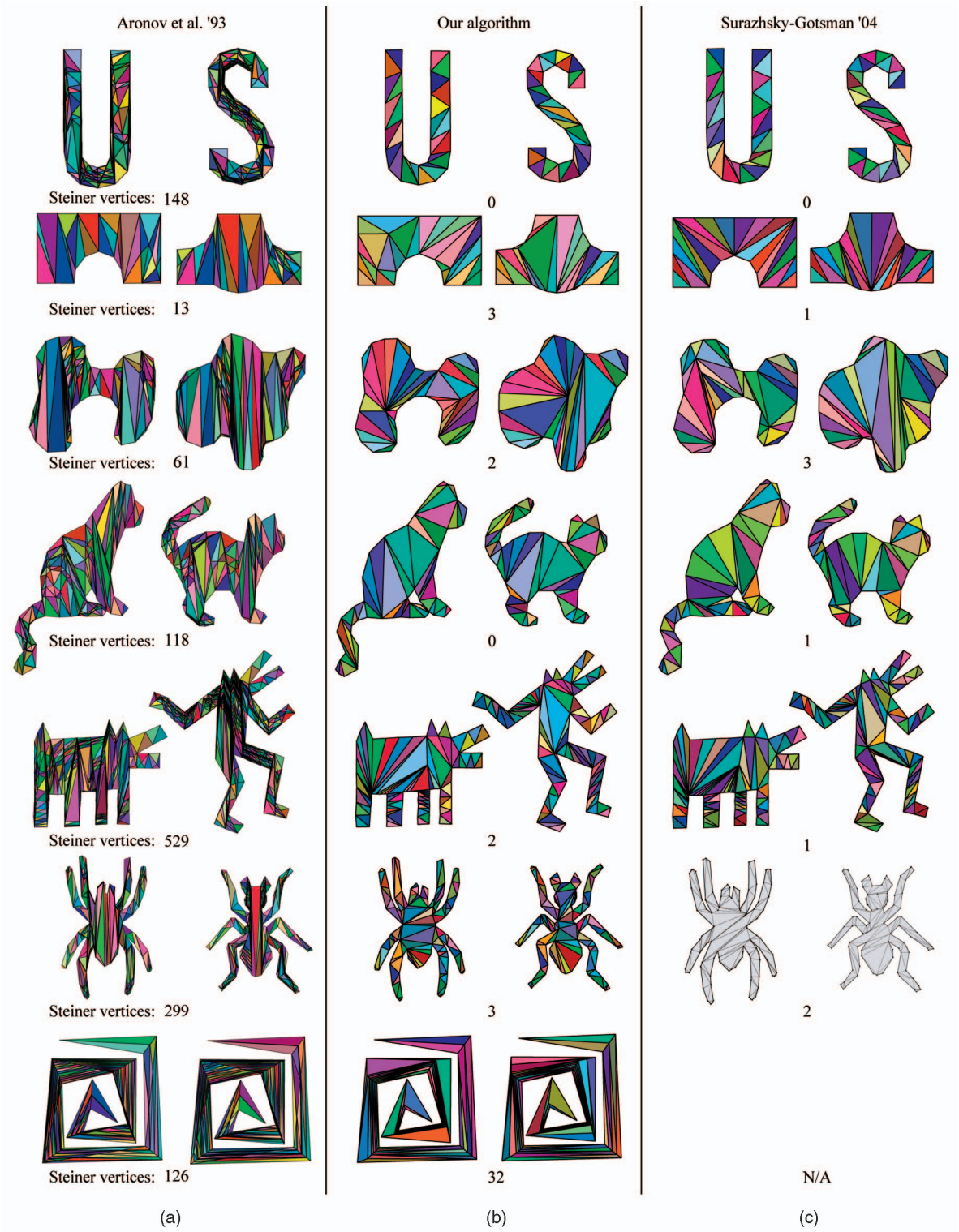


Fig. 9. **Compatible tessellation comparisons:** Comparing our results with those of Aronov et al. [31] and Surazhsky and Gotsman [12]. Our algorithm generates tessellations similar to [12], but with a significantly simpler algorithm. The last image in column 3 was created using Surazhsky and Gotsman's implementation of [12]. "N/A" indicates that both our implementation of [12] and theirs failed due to numerical robustness issues.

mentioned above. All remaining vertices lie on the polygon boundary initially and these can be adjusted toward the interior of the polygon by various means.

Extensions: The triangulations generated by our algorithm are nearly minimal; however, they usually include many long thin triangles which are undesirable for applications. To improve the mesh quality, we use a variation of the compatible refinement scheme in [12]. Quality meshes can be generated much more efficiently working coarse-to-fine in this way than in the approaches which start with a fine mesh of low quality [31], [10], [11]. Fig. 8 shows some compatible triangulations generated by our algorithm both before and after refinement. Additionally, we have implemented a

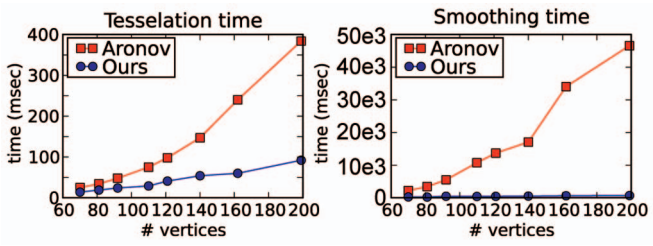


Fig. 10. **Tessellation timing comparison:** Plots of the left two columns of Table 2.

simple scheme for incorporating interior correspondences, similar to that of [10]. Essentially, when there are interior correspondences, we perform a preprocess that eliminates the holes and interior points by cutting bridges to them.

6 Results and Discussion

To demonstrate the benefits of the newly introduced algorithms, we have created a system which implements both the rigid morphing of [11] with improvements detailed in [45], as well as the rigid deformation technique of [2]. Deformation techniques can be used to create new inputs for morphing which are trivially compatible, allowing for the simple creation of animations that are impossible with either technique alone. Figs. 7, 8, 9, 10, 11, and 12 show various results of our boundary matching, simplification, and triangulation algorithms. Supplemental videos demonstrating our system can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TVCG.2009.38.

Boundary matching. Though it is impossible for any automatic algorithm to guess the user's intention correctly 100 percent of the time, with our technique, a single user correspondence is often sufficient. Fortunately, applying the dynamic programming algorithm is also very fast (in milliseconds), so users can obtain instant feedback while adding and removing correspondences. The scale-space matching algorithm of [17] can achieve similar results (see Fig. 7), but the method is not fast enough to be used

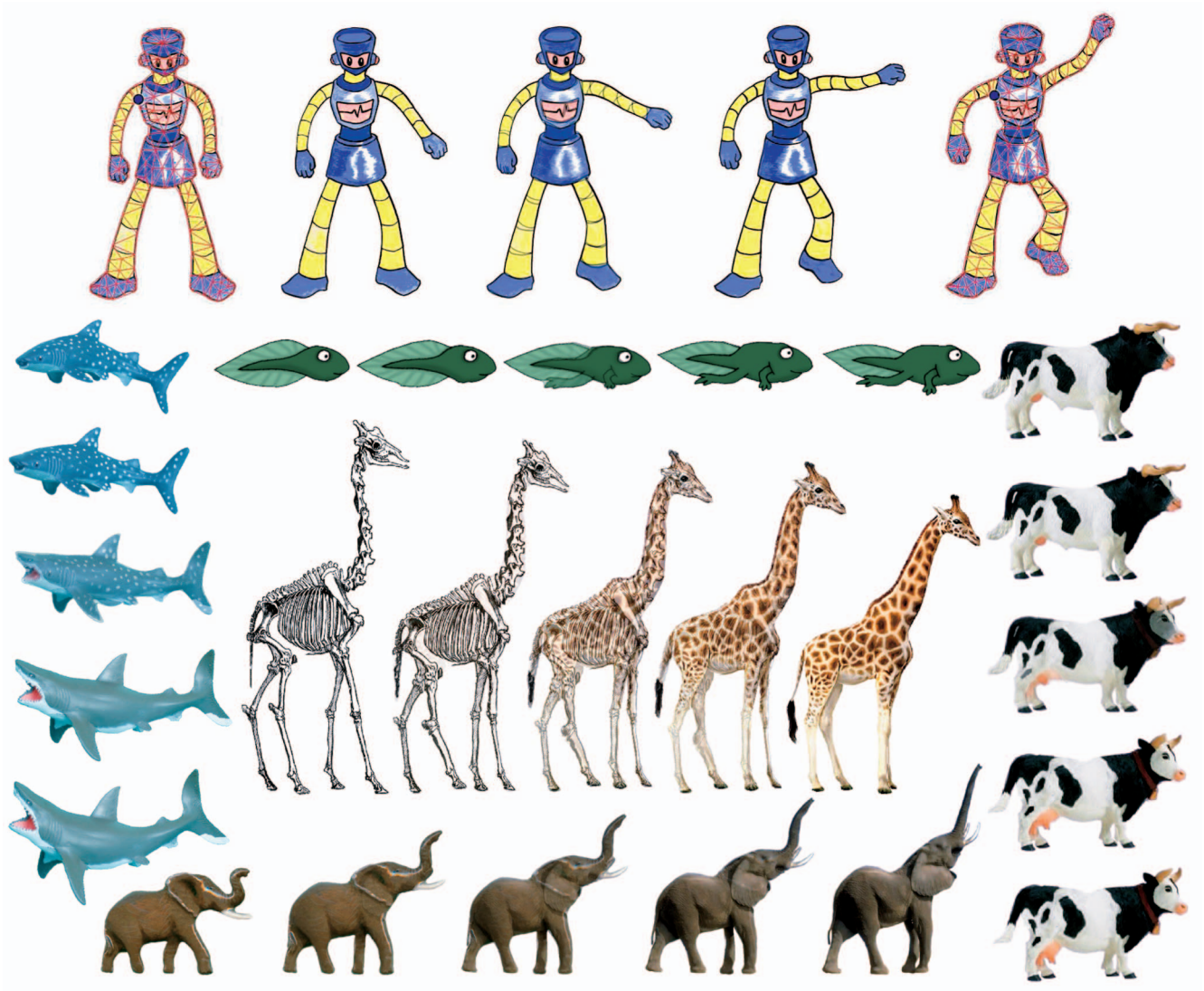


Fig. 11. **Morphing results:** Morphing results using compatible meshes generated by our techniques, with linear texture blending. In the top row, the first and last frames show triangulations, and the blue dot indicates the single manual correspondence specified.

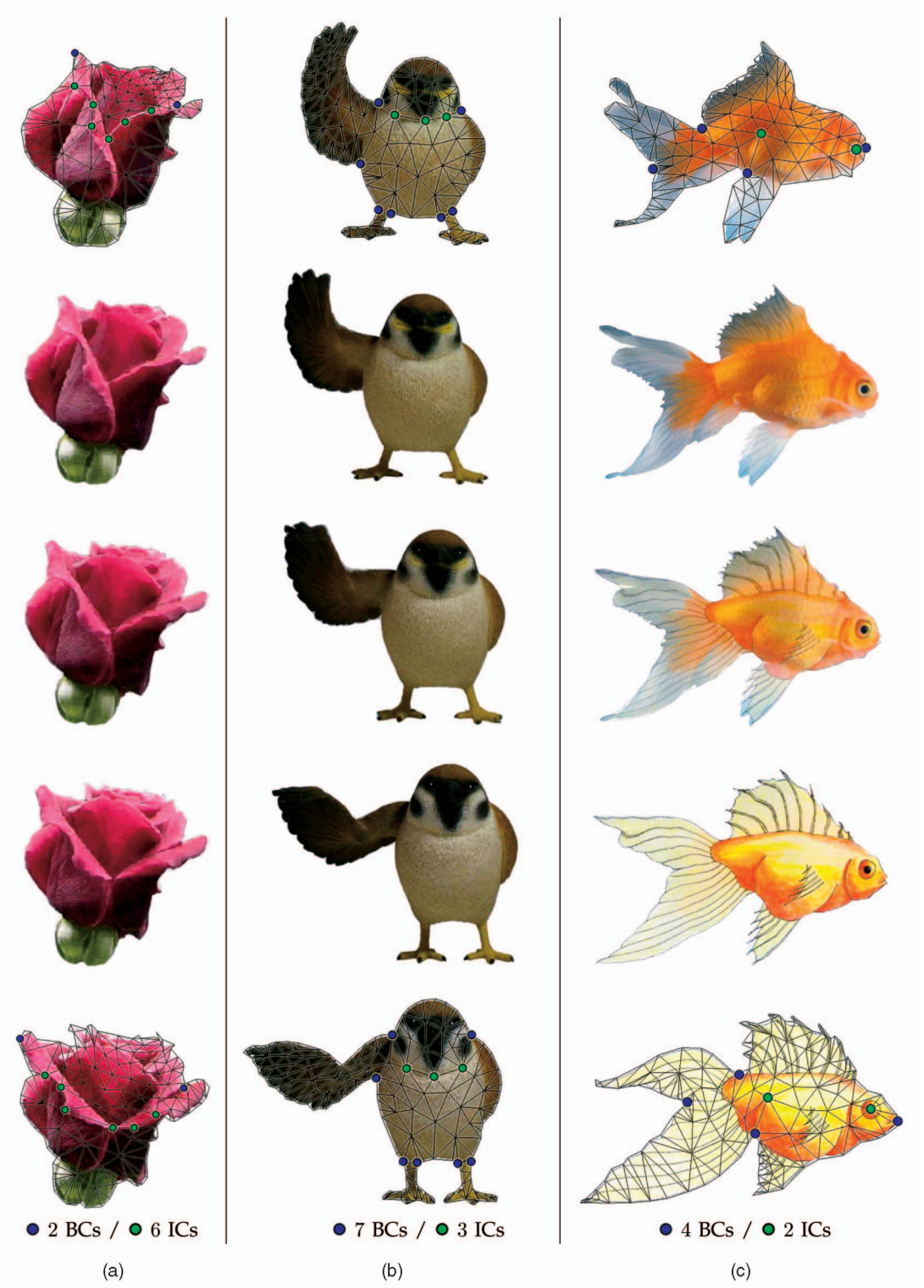


Fig. 12. **Additional morphing results:** Morphing results on more complex inputs, showing compatible meshes and the user-specified correspondences. Correspondences are boundary (BC) or interior (IC) as indicated.

interactively. On the other hand, we find the previous techniques that are interactive require many more manual correspondences than our method, and often fail to produce a satisfying matching, even after manual specification of many correspondences. Table 1 summarizes these findings. It compares the number of user-given correspondences necessary to obtain a satisfying matching for three different methods: ours (feature-based), curvature-based, and tangent-based elastic curve matching.

The first observation is that our method requires fewer user interactions than the other two (between two and three times fewer manual correspondences). However, even more important is that with our method, it was always possible to find a satisfying match using a small number of manual

TABLE 1
Boundary Matching Comparisons

Input	Our method	Curvature	Tangent
Cat standing	$= 2$	> 6	> 6
Ant-spider	$= 5$	> 13	> 13
Pony	$= 2$	$= 5$	$= 8$
Bird	$= 2$	$= 5$	$= 2$
Barking dog	$= 4$	> 7	> 6
Elephant-giraffe	$= 4$	> 7	> 7
Kicking man	$= 1$	> 3	> 10

The number of manual correspondences required to obtain a satisfactory boundary matching. The indication "$= n$" means it took exactly n correspondences; "$> n$" means that after n manual correspondences, the result still contained regions with significant artifacts that could not be eliminated without manually specifying the correspondence of nearly every point in the region. The inputs used for the comparison are those shown in Fig. 8, from top to bottom.

TABLE 2
Tessellation and Smoothing Timing Comparison

Verts	Steiner verts	Smoothed verts	t tess (msec)	t smooth (msec)
70	0/232	110/342	14/25	279/2212
81	0/294	121/415	19/34	231/3406
92	4/412	136/544	24/48	473/5508
110	0/580	150/730	29/75	408/10723
121	2/704	163/865	41/98	448/13689
140	3/975	183/1155	54/147	521/17066
162	1/1394	203/1596	60/240	627/34036
199	4/2043	243/2282	92/384	686/46587

We compare our algorithm with that of Aronov et al. [31]. Results are in the form (ours/Aronov). For both tessellations, smoothing was done using a variation on the method of Surazhsky and Gotsman [12]. The test was performed on the pair of cats at the top of Fig. 8, simplified to different numbers of initial vertices by the compatible simplification algorithm of Section 4.

correspondences. The reason is that we only do direct matching on significant features or user-specified correspondences. Our technique effectively falls back to uniform parametrizations where there are no significant features. In contrast, with matching algorithms based directly on matching curvatures, arc-lengths, or tangents globally, it is easy for the matching to get hung up on noisy features, or to discount good matches simply because of slight differences in feature magnitude.

We believe that there is still room for improvement in terms of the number of required manual correspondences. One possibility would be to make use of a richer boundary descriptor, and an interesting avenue of future work would be to fit a priori primitives on pixel neighborhoods in order to gain more information from input shapes. Another approach would be to couple the boundary analysis with a higher level analysis, for instance, taking local symmetries into account.

Compatible triangulation. In [11], it is noted that "the most time-consuming step is optimizing triangle shape." With the combination of our boundary simplification and subsequent near-optimal triangulation, even this step (i.e., our mesh refinement based on [12]) takes just a few tenths of a second (see Table 2 and Fig. 10). In comparison, with a method that generates a large number of Steiner vertices like that of [31] used in [11], the time for smoothing easily reaches tens of seconds, which is unacceptable for an interactive application. With a more complex smoothing metric to reduce the cross-distortion, as in [30], the difference would likely be even more dramatic.

Regarding simplicity of implementation, it took one coauthor of this paper two days to implement [31], one week for our new algorithm, and three months for Surazhsky and Gotsman [12]. In fact, we have subsequently learned that even the authors of [12] themselves did not implement the Suri minimal-link algorithm described in the paper, because it was deemed too complex [46]. Though [31] is clearly the simplest of the algorithms, the time required to smooth the resulting meshes can be prohibitive. On the other hand, the relatively complicated data structures involved in [12] proved challenging to implement and debug, and appear to be more prone to numerical degeneracies than our algorithm, as can be seen in Fig. 9.

7 Conclusions and Future Work

We presented a set of algorithms for the compatible embedding of 2D shapes, a necessary intermediate step for the morphing of practical inputs having complex silhouettes. Fig. 12 demonstrates the effectiveness of our coarse embedding approach on several such shapes with relatively complex silhouettes.

The major advantage of our approach is that it allows a user to create shape morphings easily and efficiently with just a few clicks. Each step in our processing pipeline is fast enough to provide a user with a truly interactive system for viewing and modifying results, which is important for the intended use as part of an animation system. A second significant advantage is the simplicity of each of the algorithms.

However, our current system has some limitations. Our full vision of a compatible embedding animation system is yet to be realized. In this paper, we focused on the geometric aspects of the problem, neglecting image space. Thus, for example, some "ghosting" artifacts can still be seen in our results that use texture cross-fading. Our geometric techniques handle the large-scale mapping between shapes, but ultimately the finer details, the "residual," should be handled in image space, perhaps by warping the underlying textures or by more sophisticated texture blending schemes. We are also interested in improving our approach to better handle shapes of varying genus, though the compatible embedding already gives us a way to handle small differences by simply treating holes as texture.

Additionally, the approach presented here only works for a pair of inputs. For more complex animations, it is advantageous to be able to construct compatible embeddings of several shapes at a time and morph between them, as in [24], [47]. Since the initial submission of this work, we have completed just such an extension [48].

From a user-interface standpoint, it would be beneficial to have an advanced layering system that simplifies assembly of multiple compatible embeddings into composite animations. Also useful would be a boundary-based deformation tool similar to [13]. Such features would greatly expand the creative potential for users, leading to a qualitatively new approach to the creation of 2D animations.

Also, we have assumed a texture-based representation of the details. However, the parameterization provided by the underlying compatible triangulation can also be used as a coordinate system for vector drawing elements. Thus, compatible embeddings can also be used for interpolation-based animation of vector drawings.

ACKNOWLEDGMENTS

This work was supported in part by the Japan Science and Technology Agency, CREST project.

REFERENCES

[1] L. Kavan, S. Dobbyn, S. Collins, J. Zara, and C. O'Sullivan, "Polypostors: 2D Polygonal Impostors for 3D Crowds," *Proc. ACM SIGGRAPH Symp. Interactive 3D Graphics and Games,* Feb. 2008.

[2] T. Igarashi, T. Moscovich, and J.F. Hughes, "As-Rigid-As-Possible Shape Manipulation," *Proc. ACM SIGGRAPH '05,* pp. 1134-1141, 2005.

[3] T. Beier and S. Neely, "Feature-Based Image Metamorphosis," *Proc. ACM SIGGRAPH '92,* pp. 35-42, 1992.

[4] S.-Y. Lee, K.-Y. Chwa, and S.Y. Shin, "Image Metamorphosis Using Snakes and Free-Form Deformations," *Proc. ACM SIGGRAPH '95,* pp. 439-448, 1995.

[5] G. Wolberg, "Image Morphing: A Survey," *The Visual Computer,* vol. 14, no. 8, pp. 360-372, 1998.

[6] S. Schaefer, T. McPhail, and J. Warren, "Image Deformation Using Moving Least Squares," *Proc. ACM SIGGRAPH '06,* pp. 533-540, 2006.

[7] H. Fang and J.C. Hart, "Detail Preserving Shape Deformation in Image Editing," *Proc. ACM SIGGRAPH '07,* vol. 26, no. 3, p. 12, 2007.

[8] T.W. Sederberg, P. Gao, G. Wang, and H. Mu, "2D Shape Blending: An Intrinsic Solution to the Vertex Path Problem," *Proc. ACM SIGGRAPH '93,* pp. 15-18, 1993.

[9] E. Goldstein and C. Gotsman, "Polygon Morphing Using a Multiresolution Representation," *Proc. Graphics Interface Conf.,* pp. 247-254, 1995.

[10] A. Tal and G. Elber, "Image Morphing with Feature Preserving Texture," *Computer Graphics Forum (Proc. Eurographics '99),* vol. 18, no. 3, pp. 339-348, 1999.

[11] M. Alexa, D. Cohen-Or, and D. Levin, "As-Rigid-As-Possible Shape Interpolation," *Proc. ACM SIGGRAPH '00,* pp. 157-164, 2000.

[12] V. Surazhsky and C. Gotsman, "High Quality Compatible Triangulations," *Eng. with Computers,* vol. 20, no. 2, pp. 147-156, July 2004.

[13] M. Eitz, O. Sorkine, and M. Alexa, "Sketch Based Image Deformation," *Proc. Workshop Vision, Modeling, and Visualization,* pp. 135-142, 2007.

[14] T.W. Sederberg and E. Greenwood, "A Physically Based Approach to 2D Shape Blending," *Proc. ACM SIGGRAPH '92,* pp. 25-34, 1992.

[15] T.B. Sebastian, P.N. Klein, and B.B. Kimia, "Recognition of Shapes by Editing Their Shock Graphs," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 26, no. 5, pp. 550-571, May 2004.

[16] F. Mokhtarian and A.K. Mackworth, "A Theory of Multiscale, Curvature-Based Shape Representation for Planar Curves," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 14, no. 8, pp. 789-805, Aug. 1992.

[17] X. Zabulis, J. Sporring, and S.C. Orphanoudakis, "Perceptually Relevant and Piecewise Linear Matching of Silhouettes," *Pattern Recognition,* vol. 38, no. 1, pp. 75-93, 2005.

[18] M. van Eede, D. Macrini, A. Telea, C. Sminchisescu, and S.S. Dickinson, "Canonical Skeletons for Shape Matching," *Proc. 18th Int'l Conf. Pattern Recognition (ICPR '06),* pp. 64-69, 2006.

[19] C. Gotsman and V. Surazhsky, "Guaranteed Intersection-Free Polygon Morphing," *Computers and Graphics,* vol. 25, no. 1, pp. 67-75, 2001.

[20] T. Surazhsky, V. Surazhsky, G. Barequet, and A. Tal, "Blending Polygonal Shapes with Different Topologies," *Computers and Graphics,* vol. 25, no. 1, pp. 29-39, 2001.

[21] M. Alexa, "Recent Advances in Mesh Morphing," *Computer Graphics Forum,* vol. 21, no. 2, pp. 173-197, 2002.

[22] J. Schreiner, A. Asirvatham, E. Praun, and H. Hoppe, "Inter-Surface Mapping," *Proc. ACM SIGGRAPH '04,* pp. 870-877, 2004.

[23] D. Xu, H. Zhang, Q. Wang, and H. Bao, "Poisson Shape Interpolation," *Proc. 2005 ACM Symp. Solid and Physical Modeling (SPM '05),* pp. 267-274, 2005.

[24] R.W. Sumner, M. Zwicker, C. Gotsman, and J. Popović, "Mesh-Based Inverse Kinematics," *Proc. ACM SIGGRAPH '05,* pp. 488-495, 2005.

[25] T.B. Sebastian, P.N. Klein, and B.B. Kimia, "Alignment-Based Recognition of Shape Outlines," *Proc. Fourth Int'l Workshop Visual Form (IWVF-4),* pp. 606-618, 2001.

[26] R.C. Veltkamp and M. Hagedoorn, *State of the Art in Shape Matching,* pp. 87-119. Springer-Verlag, 2001.

[27] D. Douglas and T. Peucker, "Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or Its Caricature," *Canadian Cartographer,* vol. 10, no. 2, pp. 112-122, 1973.

[28] A. Finkelstein and D.H. Salesin, "Multiresolution Curves," *Proc. ACM SIGGRAPH '94,* pp. 261-268, 1994.

[29] P. Barla, J. Thollot, and F. Sillion, "Geometric Clustering for Line Drawing Simplification," *Proc. Eurographics Symp. Rendering,* 2005.

[30] P.V. Sander, X. Gu, S.J. Gortler, H. Hoppe, and J. Snyder, "Silhouette Clipping," *Proc. ACM SIGGRAPH '00,* pp. 327-334, 2000.

[31] B. Aronov, R. Seidel, and D. Souvaine, "On Compatible Triangulations of Simple Polygons," *Computational Geometry: Theory and Applications,* vol. 3, pp. 27-35, 1993.

[32] H. Gupta and R. Wenger, "Constructing Piecewise Linear Homeomorphisms of Simple Polygons," *J. Algorithms,* vol. 22, no. 1, pp. 142-157, 1997.

[33] S. Suri, "On Some Link Distance Problems in a Simple Polygon," *IEEE Trans. Robotics and Automation,* vol. 6, no. 1, pp. 108-113, Feb. 1990.

[34] S. Suri, "A Linear Time Algorithm with Minimum Link Paths Inside a Simple Polygon," *Computer Vision, Graphics, and Image Processing,* vol. 35, no. 1, pp. 99-110, 1986.

[35] L. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. Tarjan, "Linear Time Algorithms for Visibility and Shortest Path Problems Inside Simple Polygons," *Proc. Symp. Computational Geometry (SCG '86),* pp. 1-13, 1986.

[36] L. Guibas, E. McCreight, M. Plass, and J. Roberts, "A New Representation for Linear Lists," *Proc. Ninth ACM Symp. Theory of Computing,* pp. 49-60, 1977.

[37] B. Chazelle, "Triangulating a Simple Polygon in Linear Time," *Discrete and Computational Geometry,* vol. 6, no. 5, pp. 485-524, 1991.

[38] N.M. Amato, M.T. Goodrich, and E.A. Ramos, "A Randomized Algorithm for Triangulating a Simple Polygon in Linear Time," *Discrete and Computational Geometry,* vol. 26, no. 2, pp. 245-265, 2001.

[39] B. Joe and R.B. Simpson, "Corrections to Lee's Visibility Polygon Algorithm," *Behaviour and Information Technology,* vol. 27, no. 4, pp. 458-473, 1987.

[40] C.A. Hipke, "Computing Visibility Polygons with LEDA," http://ad.informatik.uni-freiburg.de/mitarbeiter/hipke/_hipke.work.vis_polygon.ps.gz, 1996.

[41] "Applets for Visibility in Simple Polygons," Univ. of Bonn, http://web.informatik.uni-bonn.de/I/GeomLab/VisPolygon/, 2008.

[42] H. Hoppe, "Progressive Meshes," *Proc. ACM SIGGRAPH '96,* pp. 98-108, 1996.

[43] P.V. Sander, J. Snyder, S.J. Gortler, and H. Hoppe, "Texture Mapping Progressive Meshes," *Proc. ACM SIGGRAPH '01,* pp. 409-416, 2001.

[44] W. Baxter, P. Barla, and K. Anjyo, "Notes on Compatible Mapping of 2D Shapes," Technical Report OLMTRE-2008-002, OLM Digital, Inc., http://www.olm.co.jp/en/rd/2008/06/compatible_embedding_notes.html, 2008.

[45] W. Baxter, P. Barla, and K.-i. Anjyo, "Rigid Shape Interpolation Using Normal Equations," *Proc. Symp. Nonphotorealistic Animation and Rendering,* pp. 59-64, 2008.

[46] V. Surazhsky, Personal Comm., June 2008.

[47] W. Baxter and K. Anjyo, "Latent Doodle Space," *Computer Graphics Forum,* vol. 25, no. 3, pp. 477-485, 2006.

[48] W. Baxter, P. Barla, and K. Anjyo, "N-Way Morphing for 2D Animation," *Computer Animation and Virtual Worlds,* 2009.

William V. Baxter, III received the PhD degree in computer science from the University of North Carolina at Chapel Hill in 2004. He has been a senior researcher in the R&D Group at OLM Digital, Inc., Tokyo, Japan, since January 2005. His main research interests are NPR, HCI, and physically based models, with a particular interest in how these can be used to enhance and support creative activities.

Pascal Barla received the PhD degree in computer science from the INRIA Grenoble University in 2006, and has recently started as a researcher at the INRIA Bordeaux Sud-Ouest in the IPARLA team. His main research interest is expressive rendering, with a strong interest in human vision.

Ken-ichi Anjyo received the PhD degree in information engineering from Nagoya University in 1994, and has been with OLM Digital, Inc., since 2000. At OLM Digital, he is the lead of the Research and Development Division, responsible for making visual effects and in-house software tools. His research focus is the construction of mathematical and computationally tractable models for computer graphics. He is a member of the IEEE and the IEEE Computer Society.