High Quality Compatible Triangulations for 2D Shape Morphing

Zhiguang Liu*[1], Howard Leung[1], Liuyang Zhou[2], and Hubert P. H. Shum[3]

[1]City University of Hong Kong, Hong Kong
[2]Wisers Information Limited, Hong Kong
[3]Northumbria University, Newcastle upon Tyne, UK

Abstract

We propose a new method to compute compatible triangulations of two polygons in order to create a smooth geometric transformation between them. Compared with existing methods, our approach creates triangulations of better quality, that is, triangulations with fewer long thin triangles and Steiner points. This results in visually appealing morphing when transforming the shape from one to another. Our method consists of three stages. First, we compatibly decompose the target and source polygons into a set of sub-polygons, in which each source sub-polygon is triangulated. Second, we map the triangulation of a source sub-polygon onto the corresponding sub-polygon of the target polygon using linear transformation, thereby generating the compatible meshes between the source and the target. Third, we refine the compatible meshes, which can create better quality planar shape morphing with detailed textures. Experimental results show that our method can create compatible meshes of higher quality compared with existing methods, which facilitates smoother morphing process. The proposed algorithm is robust and computationally efficient. It can be applied to produce convincing transformations such as interactive 2D animation creation and special effects in movies.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Geometric algorithms, languages, and systems

Keywords: compatible triangulation, 2D morphing, shape interpolation

1 Introduction

Planar shape morphing, also known as shape blending, aims at smoothly transforming a source polygon into a target polygon [Wolberg 1998; Chiang et al. 1998]. 2D morphing techniques have been used widely in animation and special effects packages, such as Adobe After Effects and Adobe Flash. With the work on digital heritage, not only an ancient painting can be preserved and appreciated in a virtual reality environment, but also objects in paintings can be animated by shape morphing techniques to provide a more vivid viewing experience to the audience. The key research focus is to create high quality transformations that can avoid collapsing or overlapping of polygons during the morphing process.

2D image deformation algorithms such as the rigid shape deformation in [Igarashi et al. 2005; Schaefer et al. 2006] have been extensively explored in the research community. Users can manipulate constrained handlers to deform a given image. However, such kind of image warping techniques offer a limited range of transformations. Transforming a shape to a significantly different one is difficult due to the lack of feature correspondence.

Planar shape morphing methods offer solutions to blend two shapes with different silhouettes. A simple method to solve the shape morphing problem is to linearly interpolate the coordinates of each corresponding vertex pair between the source and the target polygons. However, simple linear interpolation sometimes creates intermediate polygons that intersect with themselves, resulting in geometrically incorrect transformations. While image space techniques such as [Schaefer et al. 2006; Fang and Hart 2007] achieve pleasant blending results, they suffer from the overlapping problems.

Previous work [Alexa et al. 2000; Gotsman and Surazhsky 2001; Surazhsky and Gotsman 2004; Baxter et al. 2009] has shown that computing compatible triangulations can successfully create smooth transformations for both the boundary and interior of a shape. However, in many situations compatible meshes can be generated only if Steiner points are added, which are points that are not part of the vertices of a polygon. [Aronov et al. 1993] first started the study of compatible triangulations by introducing at most $O(N^2)$ Steiner points, where N is the number of vertices of the polygon. Although the algorithm is conceptually simple, it introduces a large number of Steiner points and generates many long thin triangles, which can result in inconsistent rotations for shape interpolation algorithms such as [Alexa et al. 2000]. [Surazhsky and Gotsman 2004] constructed compatible meshes based on link paths, which requires a small number of Steiner points, but a high computational cost that is prohibitive.

We observed that the majority of existing compatible triangulation approaches may either create a large number of skinny triangles or are too complex for real-time shape morphing. In this paper, we propose an efficient framework for computing compatible triangulations of two simple polygons, which are defined as planar shapes with non-intersecting edges that form a closed path. Our method produce compatible meshes with few long thin triangles and a small number of Steiner points, which enables smooth transformations from one shape to another.

The major contributions of this paper are summarized as follows: First, our method decreases the number of Steiner points while maintaining the quality of the compatible meshes. Second, we maximize the minimum angle of the triangles within the source sub-polygons. The increase in regular triangles benefits the smooth transition during shape morphing. Third, our framework of compatible triangulations is simple to implement and computationally efficient. Lastly, we reduce the inconsistent rotation problem that appears from shape morphing algorithm such as rigid shape interpolation.

*zhigualiu2-c@my.cityu.edu.hk

2 Related Work

Planar shape morphing involves two sub-problems: vertex correspondence and vertex path computation [Sederberg et al. 1993]. Vertex correspondence determines how the vertex u of source polygon P matches the vertex v of target polygon Q. The vertex path determines the trajectory along which vertex u will travel to vertex v. In this paper, we concentrate on the vertex correspondence problem, i.e. computing compatible meshes.

Previous methods for computing compatible triangulations usually fall into two categories: (1) Transforming source and target polygons into another common space [Aronov et al. 1993; Alexa et al. 2000; Kranakis and Urrutia 1999]. (2) Iteratively partitioning the source and the target polygons until both inputs become a set of triangles [Suri 1986; Gupta and Wenger 1997; Surazhsky and Gotsman 2004; Baxter et al. 2009].

[Aronov et al. 1993] constructed the compatible triangulations by overlaying the triangulations of the source and target polygons in a convex polygon. The intersections of the two triangulations built a piecewise-linear homeomorphism, which introduced a large number of Steiner points. To solve this problem, [Alexa et al. 2000] employed Delaunay triangulations to reduce the Steiner points. [Kranakis and Urrutia 1999] proposed another method that the number of Steiner points can be determined by the number of inflection vertices. While their method can reduce the number of Steiner points, the algorithm sometimes results in Steiner points on the edge of polygon. Furthermore, although these methods are conceptually simple, they require high computational cost and are not suitable for real-time applications.

[Gupta and Wenger 1997] used the divide-and-conquer method to iteratively partition the source and target polygons. Their algorithm introduced a small number of Steiner points by using link paths. However, their method is not suitable for polygons with a small number of vertices. [Surazhsky and Gotsman 2004] simplified the algorithm of [Gupta and Wenger 1997] and they proposed a new remeshing method to greatly improve the mesh quality by adding a few Steiner points. Their algorithm requires implementation of many data structures and algorithms in [Suri 1986] that makes their method algorithmically complex. [Baxter et al. 2009] proposed a new way of finding compatible link paths. Based on this new link path generation algorithm, they used similar scheme as in [Surazhsky and Gotsman 2004] to compatibly partition two polygons. Although their algorithm of computing link paths is faster than that of [Surazhsky and Gotsman 2004], the proportion of regular-shaped triangles (as opposed to long thin triangles) still needs to be improved.

Much of work has been proposed for interpolating two shapes. [Alexa et al. 2000] proposed a method that attempted to preserve rigidity. They separately interpolated the rotation and scale/shear components of an affine transformation matrix, which generated pleasing results with small rotations for most of cases. Inspired by [Alexa et al. 2000], [Xu et al. 2006] presented a 3D morphing method based on Poisson equation that generated visual pleasing morphing sequences. However, their method suffered from the inherited problem of rigid interpolation methods that the rotations may be incorrectly interpolated. In order to fix this problem, [Baxter et al. 2008] proposed a method to consistently assign rotations. [Sumner and Popović 2004] proposed a method that transferred the 3D deformation of a source triangle mesh onto a different target triangle mesh. However, their algorithm is designed for the case where there is a clear semantic correspondence between the source and target. [Li et al. 2013] introduced a new type of coordinates for Hermite interpolation that can be applied to shape deformation. Other methods such as [Chen et al. 2013] trying to preserve certain properties like smoothness and distortion for 2D shape interpolation.

Interpolating boundary curve is another topic of shape morphing. [Jiang et al. 2002] represented curves by sequences of symbols. The curve morphing problem is formulated as computing a weighted mean of two strings. [Srivastava et al. 2011] introduced a square-root velocity representation for analyzing shapes of curves, which can generate natural deformation along the geodesic path. [Yang et al. 2014] proposed a new structure called *part figure* to represent the shape. Their method can create smooth transition between the source and target shapes by interpolating the part figure. However, these curve interpolation methods only solve the vertex path problem and cannot deal with detailed texture.

In this paper, we propose a new framework to construct the compatible meshes of two simple polygons. Our method draws inspiration from [Floater 1997], which uses barycentric coordinates to map a spatial surface triangulation to planar triangulation. However, [Floater 1997] demands that every Steiner point of the target polygon Q must be a strict convex combination of its neighbors, which cannot always be satisfied in practice. As a solution, we propose an efficient convex decomposition algorithm that partitions the target polygon Q into a set of convex polygons such that we can solve the compatible mapping from a sparse linear system. In addition, our framework decreases the number of Steiner points by calculating the link path of a vertex pair in the source polygon.

3 Compatible Triangulations

The input data of our system are two simple polygons P and Q with corresponding vertices ordered in counter-clockwise. We denote $P = \{U, E^P\}$ and $Q = \{V, E^Q\}$ as the source and target polygons with point set $u \in U$ and $v \in V$, together with the edge set E^P, E^Q respectively. P and Q are assumed to be simple polygons without holes, in which the edges do not cross each other and form a closed contour enclosing each polygon. We define $\mathcal{T}_P$ and $\mathcal{T}_Q$ as the triangulations of polygon P and Q. $\mathcal{T}_P$ and $\mathcal{T}_Q$ are compatible if they have equivalent topology that is defined as:

1. There is an one-to-one correspondence between the vertices of $\mathcal{T}_P$ and that of $\mathcal{T}_Q$.

2. There is an one-to-one correspondence between the edges of $\mathcal{T}_P$ and $\mathcal{T}_Q$, meaning that if there is an edge connecting two vertices of $\mathcal{T}_P$, then there is an edge connecting the corresponding vertices of $\mathcal{T}_Q$ and vice versa.

3. The boundary vertices of both $\mathcal{T}_P$ and $\mathcal{T}_Q$ are traversed in the same clockwise or counter-clockwise order.

Given two simple polygons P and Q with a boundary vertex correspondence, our algorithm works in three stages. First, we compatibly decompose the source polygon P and the target polygon Q into a set of sub-polygons, $p = \bigcup p_i$ and $q = \bigcup q_i$, where the target sub-polygon q_i is convex. Consider each source sub-polygon p_i of P, we triangulate p_i using Delaunay triangulation. Second, we map the triangulation $\mathcal{T}_{p_i}$ of source sub-polygon p_i onto corresponding target sub-polygon q_i using a sparse linear system. Third, we refine the compatible mesh to improve the mesh quality, which is important for high quality morphing in 2D animation and special effects for movies. Figure 1 summarizes our algorithm for compatible triangulations of two simple polygons.

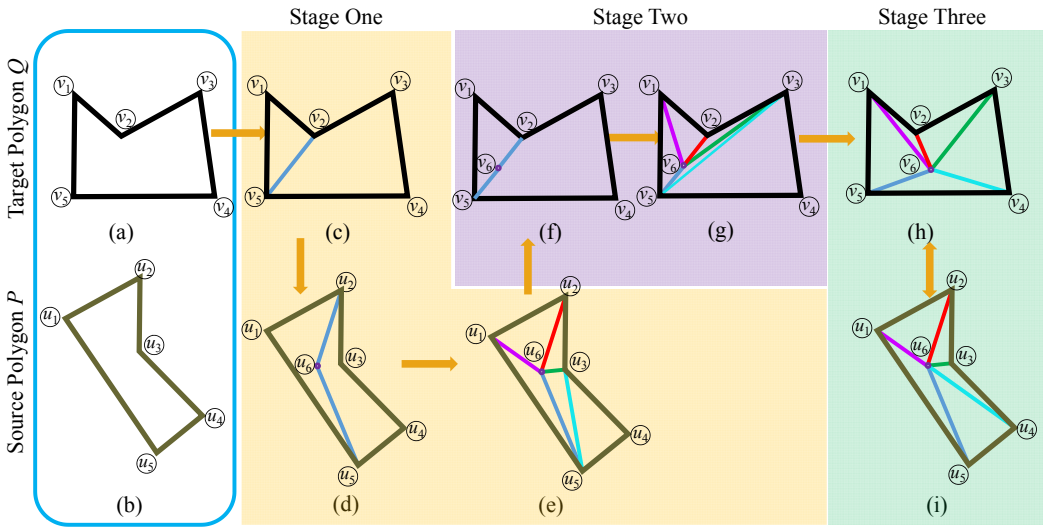


Figure 1: *Overview of the proposed framework to compatibly triangulate two simple polygons. (a) The target polygon Q. (b) The source polygon P. (c) We decompose the target polygon Q into a set of convex polygons $\bigcup q_i$ using the line segments, e.g. the line segment connecting vertex v_2 and v_5 shown in blue color. (d) We compatibly decompose source polygon P using the line segment found in (c). However, if polygon P does not contain such a line segment that lies completely inside P, we look for link path to partition polygon P, e.g. the 2-link path between vertex u_2 and u_5 shown in blue. (e) We triangulate each sub-polygon p_i of source polygon P using Delaunay triangulation. (f) We may need to add some Steiner points on the edge of sub-polygon q_i to keep equivalent topology. (g) We solve a linear system to map the triangulation of sub-polygon p_i onto the corresponding sub-polygon q_i of target polygon Q. (h-i) We finally refine the compatible meshes by operations such as splitting long edges and flipping interior edges so as to improve the interior angles of the mesh.*

3.1 Compatible Decomposition of the Target and Source Polygons

In the first phase, we decompose the target polygon Q into a set of convex sub-polygons. In a simple polygon, a vertex $v \in V$ is *convex* if the angle α formed by two edges at v is less than π radians; otherwise v is considered to be *concave*. Our goal is to break down all the concave vertices of target polygon Q such that all the vertices of sub-polygons are convex.

Without loss of generality, we assume the target polygon Q to be a simple polygon with N vertices arranged in counter-clockwise order. Here, we label the concave vertices of Q as $v_1, ..., v_C$ and the convex vertices v_{C+1}, v_N. The minimum number of convex decomposition without Steiner points of Q can be computed in $O(N + C^2 min\{C^2, N\})$ [Keil and Snoeyink 2002]. However, in order to reduce the overall algorithm complexity, we propose an approximate optimal algorithm to decrease the number of Steiner points for both the source and target polygons, which yields no more than four times of the optimal convex sized partitions.

The number of the introduced Steiner points would greatly influence the algorithm computational cost. Therefore, we aim at decreasing the number of Steiner points during the compatible decomposition of source and target polygons. We apply link path to determine the number of Steiner points. A link path between the vertex u_a and u_b is a polyline within the polygon that joins the vertex pair (u_a, u_b) such as vertex pairs (u_2, u_6) and (u_6, u_5) in Figure 1(d) that define a 2-link path between vertex u_2 and u_5. A minimum link distance for vertex pair (u_a, u_b), $linkDist(u_a, u_b)$, is the minimum number of line segments in a polyline, for example, the minimum link distance for vertex pair (u_2, u_5) in Figure 1(d) is 2. We follow [Baxter et al. 2009] to compute the link path with minimum link distance for all vertex pairs in $O(H \cdot N_i^3)$, where H is the number of sub-polygons and N_i is the number of vertices for sub-polygon p_i.

As shown in Figure 1(c), we find all the concave vertices $v_c, c \in C$, such as v_2, of the target polygon Q in $O(N)$. For each v_c, we find any non-adjacent vertices of the target sub-polygon q_i that have direct line-of-sight to v_c, and denote them as v_n. With the same vertex index pair (v_c, v_n) in the source polygon P, we check if the minimum link distance $linkDist(u_c, u_n)$ between vertex pair (u_c, u_n) is smaller. Our algorithm works in a greedy sense and finds a partition with a smaller link distance from the source polygon in each iteration. In practice, the minimum link distance may not be unique. We choose the vertex pair (v_c, v_n) that can partition the angle α_c at concave vertex v_c as balanced as possible, e.g. vertex pair (v_2, v_5) partitions the concave vertex v_2 in a more balanced way comparing with the vertex pair (v_2, v_4). The time cost of our convex decomposition algorithm is at most $O(N^2 + H \cdot N_i^3)$.

Algorithm 1 summarizes our polygon decomposition algorithm in an iterative sense. Our algorithm tries to decrease the number of Steiner points as much as possible using the minimum link distance, as shown in line 7 of Algorithm 1. However, we may create some long thin triangles with such a small number of Steiner points. Thus, there is a tradeoff between the number of Steiner points and the minimum angle of the mesh. In our work, we aim at generating fewer number of Steiner points and the resulting long thin triangles introduced can be further improved during the compatible mesh refining process described in Section 3.3.

By this stage, we have compatibly decomposed the source polygon P and target polygon Q into sub-polygons $\{p_i = (U^{p_i}, E^{p_i})\}$ and $\{q_i = (V^{q_i}, E^{q_i})\}$, where q_i is one of convex partitions of Q, p_i is the sub-polygon corresponding to q_i. We apply Delaunay triangulations as the initial triangulation of the source sub-polygon p_i, which can maximize the minimum angle with no extra Steiner points in $O(N_i log N_i)$ [Fortune 1987]. We denote $\mathcal{T}_{p_i}$ as the triangulation of sub-polygon p_i and aim at constructing compatible triangulation $\mathcal{T}_{q_i}$ of q_i based on $\mathcal{T}_{p_i}$.

Algorithm 1: Compatible decomposition of the target and the source polygons

Input: The source and target polygons, P and Q
v_c: concave vertex of Q
v_n: non-adjacent vertex of v_c that is visible to v_c
Output: A convex decomposition of Q, $q = \bigcup q_i$, and compatible decomposition of P, $p = \bigcup p_i$
convexDecompostion(P, Q)
    for *each* v_c **do**
        $u_n = \arg \min_{u_c, u_n \in U} linkDist(u_c, u_n)$
        Decompose Q using line segment connecting v_c and v_n that creates two convex sub-polygons:
            $\{q_i, q_{i+1}\}$
        Decompose P using link path between u_c and u_n that creates two sub-polygons:
            $\{p_i, p_{i+1}\}$
        convexDecompostion(p_i , q_i)
        convexDecompostion(p_{i+1}, q_{i+1})
    end

3.2 Compatible Triangulations Mapping

The compatible decomposition process may introduce Steiner points on the link path of source polygon P such as the vertex u_6 in Figure 1(d). In addition, in order to improve the mesh quality, the mesh refinement process detailed in Section 3.3 will create Steiner points within each sub-polygon. Therefore, we have two types of Steiner points: (1) Steiner points that lie on the link path of source sub-polygon p_i, and (2) Steiner points that lie within p_i. For (1), we map the Steiner points onto the corresponding edges of target sub-polygon q_i based on the simple line-segment-length proportion principle. For (2), we solve the mapping by a sparse linear system.

3.2.1 Mapping Steiner Points on the Link Path of Source Polygon

We denote u_s as a Steiner point lies on the link path between vertex u_a and u_b in the source sub-polygon p_i such as the vertex u_6 for vertex pair (u_2, u_5) in Figure 1(d). We add a Steiner point v_s for target sub-polygon q_i on the corresponding line segment $v_a v_b$ based on the linear ratio with the following equation:

$$v_s = \frac{polylineLength(u_a, u_s)}{polylineLength(u_a, u_b)} v_a + \frac{polylineLength(u_s, u_b)}{polylineLength(u_a, u_b)} v_b \tag{1}$$

where $polylineLength(u_a, u_b)$ is the summation of the length of all line segments on the link path between u_a and u_b.

As shown in Figure 1(d), the length of the polyline for vertex pair (u_2, u_5) is $polylineLength(u_2, u_5) = polylineLength(u_2, u_6) + polylineLength(u_6, u_5)$. We would place the vertex v_6 on the line segment $v_2 v_5$ based on the Equation (1).

3.2.2 Mapping Steiner Points Within the Source Polygon

In this section, we will explain how to map the Steiner points inside the source polygon onto the corresponding locations inside the target polygon. As shown in Figure 2, we have to decide how to map the Steiner point u_1 and u_2 onto v_1 and v_2 inside the target polygon. Here, we calculate the barycentric coordinates of u_1 and u_2. We then compute the proper locations for Steiner point v_1 and v_2 using the barycentric coordinates found in the source polygon.

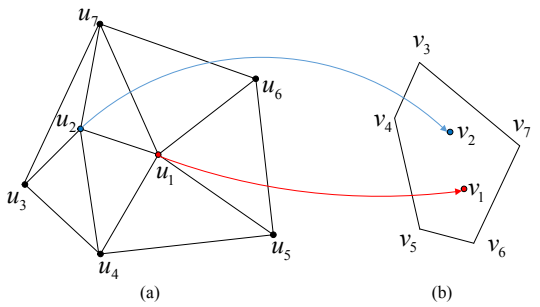


Figure 2: *Mapping Steiner points within the source sub-polygon onto the target sub-polygon. (a) The source sub-polygon with Steiner points u_1 and u_2. (b) The corresponding target sub-polygon with unknown Steiner points v_1 and v_2.*

Denoting $u_j, j \in \{1, ..., n_i\}$ as a Steiner point that lies within the source sub-polygon p_i, where n_i is the number of Steiner points within p_i. We use barycentric coordinates λ to map the Steiner point u_j of source sub-polygon p_i onto the Steiner point v_j of target sub-polygon q_i. Here, we employ Floater's mean value coordinates [Floater 2003] to calculate the barycentric coordinates λ. The barycentric coordinates λ of vertex u_j can be seen as a weight of its neighboring vertices, which allows us to generate continues data from these adjacent vertices. We represent the Steiner point u_j as a weighted average of its neighboring vertices:

$$u_j = \sum_{k=1}^{M} \lambda_{j,k} u_k, \quad \sum_{k=1}^{M} \lambda_{j,k} = 1 \tag{2}$$

where M is the total number of points including boundary vertices and Steiner points for source sub-polygon p_i, i.e. $M = N_i + n_i$.

We now explain how to map the Steiner point $u_j \in U^{p_i}$, $j \in \{1, ..., n_i\}$ of source sub-polygon p_i onto the corresponding Steiner point $v_j \in V^{q_i}$ of target sub-polygon q_i, where n_i is the number of Steiner points within p_i. We define $v_1, ..., v_{n_i}$ to be the solutions of linear equations with n_i variables.

$$v_j = \sum_{k=1}^{M} \lambda_{j,k} v_k, \quad \sum_{k=1}^{M} \lambda_{j,k} = 1 \tag{3}$$

where

$$\lambda_{j,k} = 0, \ \ (j,k) \notin E^{q_i}$$
$$\lambda_{j,k} > 0, \ \ (j,k) \in E^{q_i}$$

Note that the barycentric coordinates $\lambda_{j,k}$ can be uniquely determined by Equation (2).

We rewrite Equation (3) by breaking the summation term into two sub-terms:

$$v_j = \sum_{k=1}^{n_i} \lambda_{j,k} v_k + \sum_{k=n_i+1}^{n_i+N_i} \lambda_{j,k} v_k, j \in \{1, ..., n_i\}$$
$$v_j - \sum_{k=1}^{n_i} \lambda_{j,k} v_k = \sum_{k=n_i+1}^{n_i+N_i} \lambda_{j,k} v_k \tag{4}$$

where n_i is the number of Steiner points within the target sub-polygon q_i and N_i is the number of boundary vertices of q_i .

Denoting $v_j = (x_j, y_j)$ to be a Steiner point within target sub-polygon q_i that we want to solve, Equation (4) is equivalent to the following form:

$$Ax = b_1, \quad Ay = b_2 \tag{5}$$

where $x = (x_1, ..., x_{n_i})^T$, $y = (y_1, ..., y_{n_i})^T$, and matrix $A_{n_i \times n_i}$ is in the form:

$$a_{j,j} = 1, j \in \{1, ..., n_i\}$$
$$a_{j_1,j_2} = -\lambda_{j_1,j_2} \ (j_1, j_2 \in \{1, ..., n_i\}, j_1 \neq j_2).$$

This linear system in Equation 5 has n_i unknown variables and n_i equations. The solution to Equation (5) is unique as the matrix A is non-singular. We apply LU decomposition to solve Equation (5) in $O(n_i^3)$ [Murota 1983], where n_i is the number of Steiner points within target sub-polygon q_i.

3.3 Compatible Mesh Refining

While the compatible meshes generated by our method introduce a very small number of Steiner points, there may still be some long thin triangles such as the second row of Figure 7(a). In practice, we found that these long thin triangles can cause numerical problems such as inconsistent rotations for shape morphing as shown in Figure 3. Therefore, we have to refine the compatible meshes to avoid numerical problems.

To refine the compatible meshes, we apply a variation of the remeshing method in [Surazhsky and Gotsman 2004]. We only smooth those triangles with small interior angles and long edges. Specifically, we smooth the mesh using area and angle based remeshing, splitting long edges, and flipping interior edges to improve the interior angles. The smoothed results could be found in Figure 7(b).

4 As-Rigid-As-Possible Shape Morphing

Previous work [Alexa et al. 2000; Sumner and Popović 2004; Sorkine and Alexa 2007] has shown that rigid shape morphing methods could maximize the rigidity of a blended shape. In this section, we first review the rigid shape morphing process. We then discuss its problem in extracting rotation angle from rotation matrix and explain our solution.

4.1 Rigid Shape Morphing

We follow [Alexa et al. 2000] to solve the rigid shape morphing by minimizing a quadratic function. Given the vertices of two triangles $\mathcal{T}_1 = \{u_1, u_2, u_3\}$ and $\mathcal{T}_2 = \{v_1, v_2, v_3\}$, they transform $\mathcal{T}_1$ into $\mathcal{T}_2$ with an affine transformation $\mathcal{A}\mathcal{T}_1 = \mathcal{T}_2$. The matrix $\mathcal{A}$ can be factorized into a rotation matrix $\mathcal{R}$ and a scale-shear component $\mathcal{S}$ using polar decomposition, i.e. $\mathcal{A} = \mathcal{R}\mathcal{S}$. We then independently interpolate the matrix $\mathcal{S}$ and rotation angle β of $\mathcal{R}$ to compute intermediate transformation matrix $\mathcal{A}(t) = \mathcal{R}(t\beta)((1-t)\mathcal{I} + t\mathcal{S})$, where $t \in [0, 1]$ is time. Finally, finding the vertex path of all triangles can be solved by minimizing a quadratic error between the desired matrix $\mathcal{A}$ and actual matrix $\mathcal{B}$:

$$\mathcal{E} = \sum_{f \in \mathcal{T}_P} \left\| \mathcal{B}_f(t) - \mathcal{A}_f(t) \right\| \tag{6}$$

where $\mathcal{A}_f(t)$ is an affine transformation for the f^{th} triangle, $\mathcal{B}$ is the actual matrix corresponds to $\mathcal{A}_f(t)$, and $\|\cdot\|$ is the Frobenius norm.

This rigid shape interpolation is capable of integrating texture for each intermediate shape. For each pixel PIX within an intermediate triangle $\mathcal{T}_f(t)$, we calculate its three barycentric coordinates. The barycentric coordinates are applied back to the three vertices of the corresponding source and target triangles, $\mathcal{T}_{f1}$ and $\mathcal{T}_{f2}$, to calculate the pixel on the original source and target images. We denote the color of pixel PIX within f^{th} triangle for $\mathcal{T}_{f1}$ and $\mathcal{T}_{f2}$ as $color(PIX_1)$ and $color(PIX_2)$. The color of pixel PIX on the intermediate f^{th} triangle is assigned by:

$$color(PIX) = color(PIX_1)t + color(PIX_2)(1-t) \tag{7}$$

4.2 Assigning Consistent Rotation

The rigid shape morphing algorithm may suffer from inconsistent rotations whenever the rotation is more than π. As shown in Figure 3(top), some triangles rotate in opposite direction from their neighbors. This problem stems from the ambiguity of extracting the angle from rotation matrix $\mathcal{R}$. The rotation angle extracting techniques usually returns an angle β between $-\pi$ and π, which is the smallest magnitude rotation for each triangle. However, the desired rotation should be $\beta + 2l\pi$, where $l \in \mathbb{Z}$ is an integer.

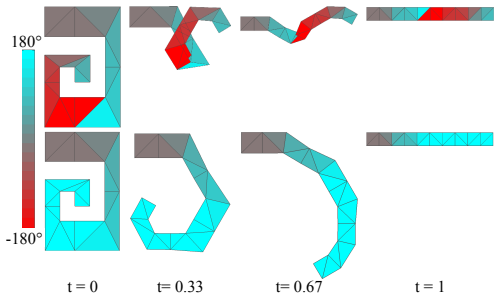


Figure 3: *Inconsistent rotations: before fixing inconsistent rotations (top) and after (bottom). The color represents the rotation magnitudes of clockwise (red) and counter-clockwise (cyan).*

Our input is a set of rotation angles β extracted from rotation matrix $\mathcal{R}$ that lie in the closed interval $[-\pi, \pi]$. We want to generate consistent rotation angles such that the jump between adjacent angles is less than π. Previous work [Alexa et al. 2000; Baxter et al. 2008] has shown that we may fail to find a solution for inconsistent rotation problem due to the numerical problem caused by the long thin triangles. While current solutions [Choi and Szymczak 2003; Baxter et al. 2008] usually assume it is possible to assign consistent rotations, their performance decreases significantly when there is no solution to assign consistent rotations such as Figure 4(top).

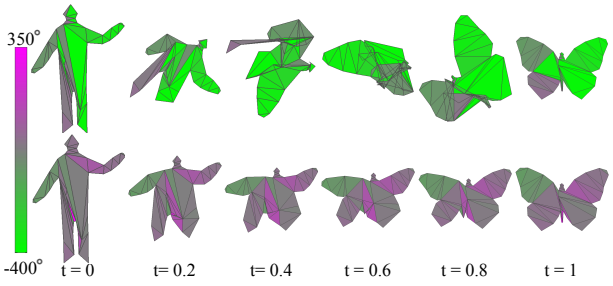


Figure 4: *Existing methods tend to create excess rotations when there is no solution for consistent rotations (top). The fixed results of our method (bottom). The color indicates the rotation magnitudes of clockwise (green) and counter-clockwise (purple).*

Here, we offer an efficient algorithm that gives a unique rotation assignment with minimum rotation angles, even when consistent

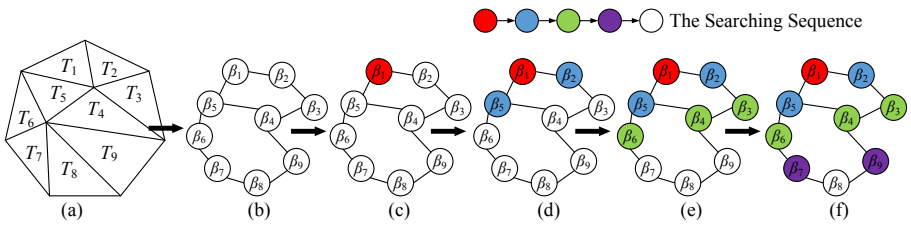


Figure 5: *Making rotation consistent. (a) The triangulation of a polygon. (b) The graph of rotation angles β that has equivalent topology to (a). (c) We start from one boundary rotation such as β_1. (d-f) We traverse the graph and fix the inconsistency.*

rotations do not exist. First, as illustrated in Figure 5(b), we treat each original rotation angle β as one vertex of a graph G. Second, we set the vertex connectivity of the graph G the same as the connectivity of the triangles in triangulation $\mathcal{T}_\mathcal{P}$ created in Section 3 such as the triangulation in Figure 5(a). Lastly, we start from one boundary rotation and examine all its neighbors, fixing any jump that is larger than π by adding $2l\pi$ such as Figure 5(c). We keep searching the graph and adjusting any adjacent rotation that is inconsistent with the current rotation until all the vertices have been traversed such as Figure 5(d-f).

During the searching and fixing process, we keep the rotation of the long thin triangle unchanged if there is a jump of more than π. This is because the inconsistent rotations often stem from these long thin triangles, which results in numerical problem for extracting rotation angles. In addition, one triangle lies on the interior of a source polygon usually need a small rotation, within $-\pi$ and π, to transform to the target one. After this process, our method would find a solution to fix these discontinuity such as the results in Figure 4(bottom). All the correct rotations in this paper are generated by this simple scheme. Although we cannot prove our method can always find a consistent rotation assignment, it works well for all the results in this paper and supplemental demo video.

5 Method Complexity

In this section, we will analyze the computational complexity of our method. The time cost of decomposing target polygon Q into a convex set of polygons is $O(N^2)$, where N is the number of vertices of Q. Finding a corresponding link path for each sub-polygon p_i in source polygon P is $O(N_i^3)$, where N_i is the number of vertices of source sub-polygon p_i. The Delaunay triangulation can be finished in $O(N_i log N_i)$. Compatible mapping for sub-polygon p_i requires solving a linear equation using LU decomposition that leads to $O(n_i^3)$ operations, where n_i is the number of Steiner points of sub-polygon p_i.

The main computation of our method is dominated by computing link paths and solving a linear system, i.e. $O(H \cdot max(N_i^3, n_i^3))$, where H is the number of source sub-polygons p_i. Table 1 compares the computational complexity between our method and existing methods, where N is the number of vertices for source polygon P, N_i is the number of vertices for source sub-polygon p_i, n_i is the number of Steiner points for p_i. Since in general, $N_i << N$ and $n_i << N$, our algorithm is more computationally efficient than existing methods.

The matrix A in Equation (5) is sparse and non-symmetric, thus, we further speed it up by using iterative methods such as Bi-CGSTAB [Van der Vorst 1992]. Here, we apply an open library Eigen [Guennebaud et al. 2015] to solve the sparse linear system. The compatible mapping process can be even faster before mesh refinement operations and it can be completed in $O(n)$. This is because the Delaunay triangulation can triangulate the sub-polygon p_i with no Steiner points such that we only need to map the Steiner points on the link path as discussed in Section 3.2.1.

Table 1: *Computational complexity: the main computational cost of our method is computing the link paths and solving the sparse linear system by LU decomposition, where N is the total number of boundary vertices of source polygon P, H is the number of source sub-polygons p_i, N_i is the number of boundary vertices of p_i, and n_i is the number of Steiner points of p_i.*

Aronov et al., 93		$O(N^3)$
Surazhsky-Gotsman, 04		$O(N^3 logN)$
Baxter et al., 09		$O(2N^3)$
Proposed Method	Convex decomposition	$O(N^2)$
	Link paths generation	$O(HN_i^3)$, $N_i << N$
	Linear system computation	$O(Hn_i^3)$, $n_i << N$

6 Experimental Results

In this section, we will show the experimental results and present the comparisons between alternative approaches including [Aronov et al. 1993], [Surazhsky and Gotsman 2004], and [Baxter et al. 2009]. Qualitative analysis is conducted to evaluate the mesh quality between the proposed method and other alternatives. The experiments are conducted on a Intel Core i3-2350M 2.3 GHZ PC with 4GB RAM.

6.1 Compatible Triangulations

To demonstrate the effectiveness of our method, we implemented the as-rigid-as-possible shape interpolation method [Alexa et al. 2000] introduced in Section 4.1 as well as our proposed method for improving the rotation consistency discussed in Section 4.2. Figure 6, 7 and 9 show some compatible triangulation results and some challenging polygon pairs that are quite different such as the shark and sea horse in the third row of Figure 7 and Tai Chi motions in Figure 9.

Figure 7(a) shows that our initial compatible triangulation contains few long thin triangles and we may only need to flip some edges of triangles to enlarge the minimum interior angles. Figure 7(b) shows that our compatible meshes can be further refined by methods such as splitting long edges and average the area of adjacent triangles.

Given the compatible triangulations of two input polygons, shape

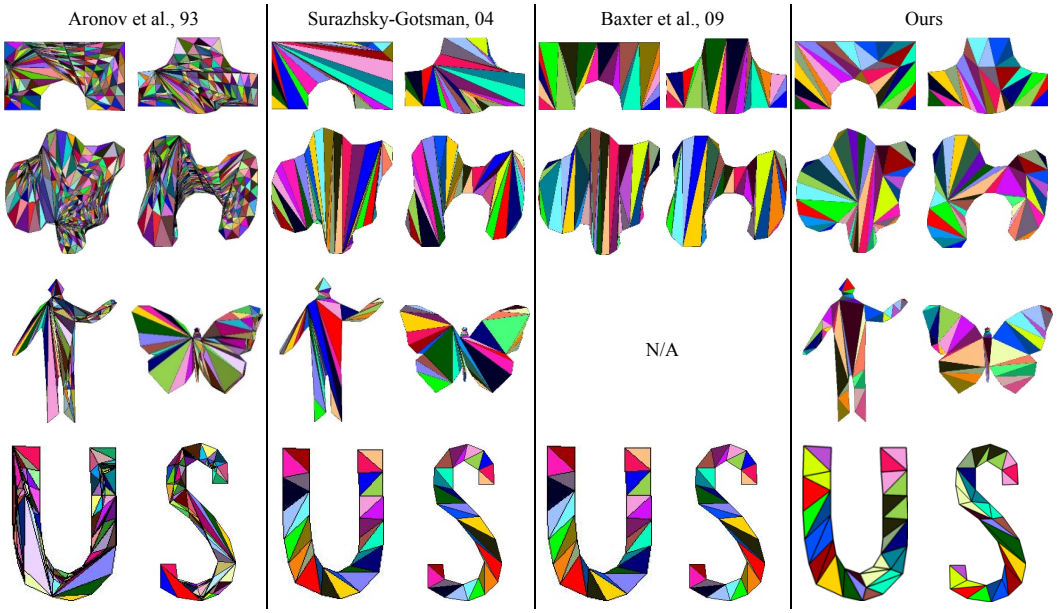


Figure 6: *Compatible triangulations comparisons. We compare our the results with those of [Aronov et al. 1993], [Surazhsky and Gotsman 2004], and [Baxter et al. 2009]. While we may add a few more Steiner points than [Aronov et al. 1993; Baxter et al. 2009], our algorithm creates high quality compatible mesh in terms of the number of long thin triangles. 'N/A' indicates that our implementation of [Baxter et al. 2009] failed due to the numerical robustness when calculating visibility polygons.*

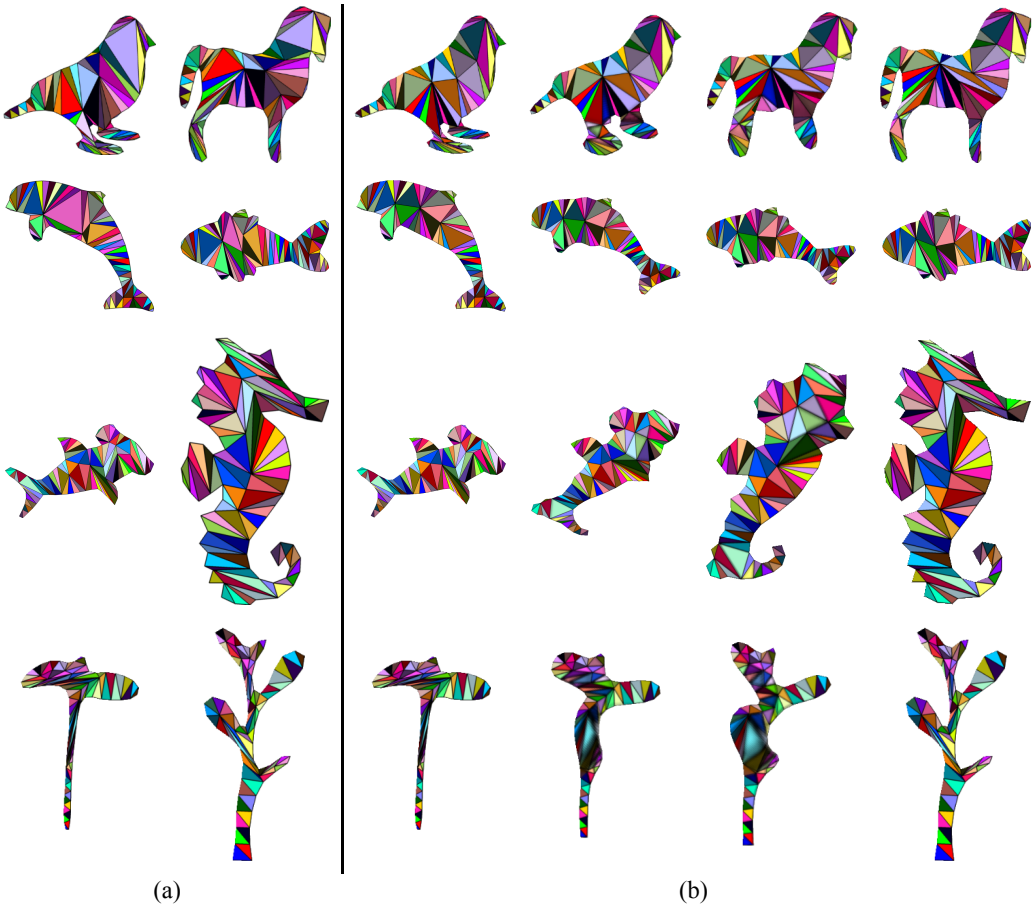


Figure 7: *Compatible triangulation results. (a) The initial tessellations of two polygons. (b) mesh refinement and morphing. Note that our compatible mesh can be used to blend shapes with large rotations, e.g. shapes in the third row.*

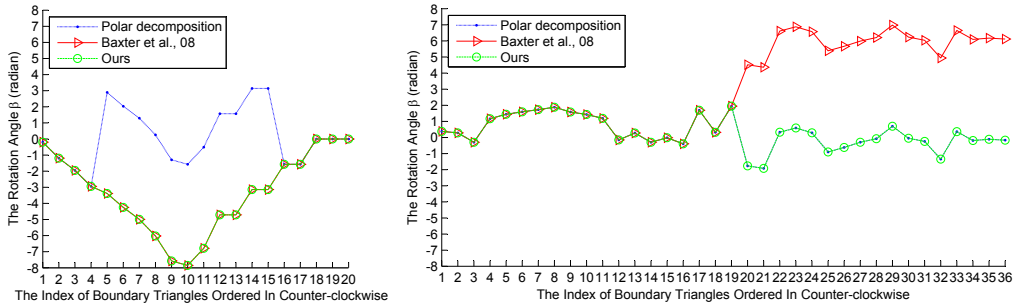


Figure 8: *Computing consistent rotations. (a) The boundary rotations of the swirl-like shape to stick, (b) human to butterfly.*

interpolation can be applied to create animations showing the transitions from one shape to another. Figure 7(b) and 9 show some interpolation results using our compatible meshes. For more transformations, please see our supplemental demo video.

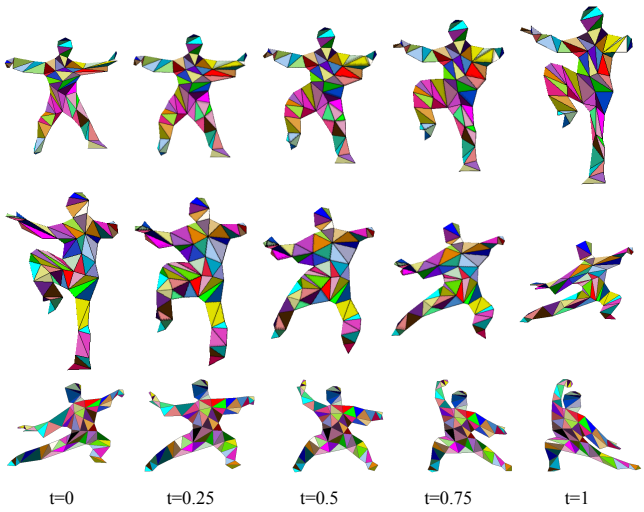


Figure 9: *Morphing of Tai-chi motions: we adopt the compatible meshes generated by our algorithm to blend Tai-chi motions.*

6.2 Regulating the Rotation

As shown in Figure 8(a), the rotations extracted from polar decompositions are the original rotations without any fix. More details about polar decomposition can be found in Section 4.1 and [Alexa et al. 2000]. We can see that the rotations extracted from polar decompositions have big jump at the 5^{th} and 16^{th} boundary rotations in Figure 8(a). [Baxter et al. 2008] fixes this inconstant rotation problem through adding $2l\pi$, where l is an integer. The results of our method are the same as [Baxter et al. 2008]. Figures 8(b) shows another example that there is no consistent rotations, in which the compatible meshes are created by the method of [Surazhsky and Gotsman 2004] in the third row of Figure 6. There is a big jump at the 20^{th} boundary triangle rotation. The problem stems from the rotation of the 19^{th} boundary triangle, which is a long thin triangle. [Baxter et al. 2008] starts from one boundary rotation and propagates the rotations inward either clockwise or counter-clockwise. In order to reduce the jump between adjacent rotation, their method tends to add additional rotations that lead to a big jump between the starting and ending rotation. In contrast, our method starts from one boundary rotation and searches both clockwise and counter-clockwise to fix inconsistent rotations, which can avoid the poor results by [Baxter et al. 2008]. Our method *ignores* these long thin triangles and keeps these rotations unchanged, which minimizes the jump between all adjacent rotations. This is because (1) We observed that triangles lie on the interior of a source polygon usually need a small rotation, within $-\pi$ and π, to transform to the target one. (2) Long thin triangles are more likely to cause inconsistent rotations.

6.3 Quantitative Analysis

The quality of the compatible meshes would greatly influence the intermediate shapes generated by morphing techniques. In particular, the mesh with those long and skinny triangles would suffer from the inconsistent rotation problem [Alexa et al. 2000; Baxter et al. 2008].

We employ the following criteria to measure the mesh quality: (1) minimum interior angle of a given mesh; and (2) the proportion of angles that are smaller than a certain constant value, which are known to be reasonable mesh quality criteria [Sarrate et al. 2003]. We would like to increase the minimum interior angle of a mesh and decrease the percent of small angles.

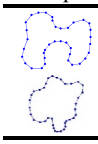
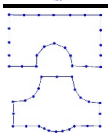
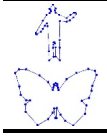
Table 2 shows a quantitative comparison between our algorithm and three other alternative methods. [Aronov et al. 1993] create a larger number of Steiner points than the others and these Steiner points do not help too much to improve the proportion of small angles. While our results are similar to [Surazhsky and Gotsman 2004; Baxter et al. 2009] in terms of the number of Steiner points, our algorithm creates a much smaller percent of small angles than [Surazhsky and Gotsman 2004; Baxter et al. 2009]. Compared with [Surazhsky and Gotsman 2004], the minimum angle of our method has been improved greatly while we add nearly the same number of Steiner points as [Surazhsky and Gotsman 2004].

7 Conclusions

We propose a new method for compatible triangulation of two simple polygons and apply them to 2D shape morphing. Our method compatibly decomposes the target and source polygons and map the triangulations of the source sub-polygons to the corresponding target sub-polygons with a sparse linear system. We also propose a solution to solve the inconsistent rotation problem of rigid shape interpolation method.

Our approach can create compatible triangulations with more regular-shaped triangles (as opposed to long thin triangles) as

Table 2: *Quantitative comparisons between triangulation quality*

Shape	Method	#Steiner Point	Minimum angle	Angles⩽10°	Angles⩽15°	Angles⩽20°
	Aronov et al., 93	68	0.0135°	21.15%	27.01%	31.07%
	Surazhsky-Gotsman, 04	**3**	**0.3833°**	24.82%	32.27%	38.30%
	Baxter et al., 09	**3**	0.3562°	23.74%	29.96%	33.29%
	Ours	**3**	0.3122°	**12.57%**	**17.83%**	**29.82%**
	Aronov et al., 93	182	0.1319°	18.84%	23.63%	28.25%
	Surazhsky-Gotsman, 04	2	0.4528°	21.15%	25.64%	32.69%
	Baxter et al., 09	**0**	3.3052°	10.61%	14.39%	30.30%
	ours	3	**3.7557°**	**5.35%**	**11.90%**	**22.02%**
	Aronov et al., 93	68	0.0311°	26.29%	32.10%	36.10%
	Surazhsky-Gotsman, 04	4	0.3833°	24.82%	32.27%	38.30%
	Baxter et al., 09	N/A	N/A	N/A	N/A	N/A
	Ours	**1**	**0.4706°**	**5.69%**	**10.16%**	**16.26%**
	Aronov et al., 93	56	0.0508°	21.55%	27.98%	34.40%
	Surazhsky-Gotsman, 04	**0**	0.1429°	6.54%	7.74%	10.71%
	Baxter et al., 09	**0**	2.2998°	9.44%	13.33%	17.22%
	Ours	2	**3.5657°**	**3.125%**	**3.64%**	**7.29%**

illustrated by the fact that there are fewer triangles whose minimum angles are small under our approach compared with other methods in [Aronov et al. 1993; Surazhsky and Gotsman 2004; Baxter et al. 2009]. In addition, the proportion of small angles is significantly smaller than existing methods, which relieves the inconsistent rotations problem for the shape interpolation methods such as [Alexa et al. 2000], and thus improves the quality of shape morphing. Another advantage is the simplicity of the three stages that all we need is to decompose a polygon, calculate link paths, and solve a sparse linear system, which enables real-time morphing.

While our method well handles the mapping between shapes, the morphing results need to be further improved. As we focus on generating compatible mesh, we simply crossfade between textures in image space. More sophisticated texture blending or image warping algorithms such as [Schaefer et al. 2006] could be incorporated into our method. Currently, the intermediate images interpolated are uniquely determined by rigid interpolation method [Alexa et al. 2000], which offers no means of controls. It would be desirable to modify some parts of the intermediate shapes if the users are not satisfied with them. We could explore possible solutions such as the linear constraints proposed in [Baxter et al. 2008] to increase user creativity.

Another drawback of our method is that we cannot deal with the polygon with holes. One possible solution is that we add a *bridge* between the outer polygon and inner polygons (i.e. the holes). We connect the outer polygon with all the holes such that we can treat a polygon with holes as a single polygon. We can then apply our previous method to compatibly decompose the source and target polygons. While we have shown many examples of compatible triangulation both in the paper and supplemental video, we also want to test our algorithm on shapes with complex structure or completely different topology in the future.

Acknowledgements

This project was partly supported by the Engineering and Physical Sciences Research Council (EPSRC) (Ref: EP/M002632/1) and a grant from City University of Hong Kong (project No. 9220077).

References

ALEXA, M., COHEN-OR, D., AND LEVIN, D. 2000. As-rigid-as-possible shape interpolation. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., 157–164.

ARONOV, B., SEIDEL, R., AND SOUVAINE, D. 1993. On compatible triangulations of simple polygons. *Computational Geometry 3*, 1, 27–35.

BAXTER, W., BARLA, P., AND ANJYO, K.-I. 2008. Rigid shape interpolation using normal equations. In *Proceedings of the 6th international symposium on Non-photorealistic animation and rendering*, ACM, 59–64.

BAXTER, W., BARLA, P., AND ANJYO, K.-I. 2009. Compatible embedding for 2d shape animation. *Visualization and Computer Graphics, IEEE Transactions on 15*, 5, 867–879.

CHEN, R., WEBER, O., KEREN, D., AND BEN-CHEN, M. 2013. Planar shape interpolation with bounded distortion. *ACM Transactions on Graphics (TOG) 32*, 4, 108.

CHIANG, C.-C., WAY, D.-L., SHIEH, J.-W., AND SHEN, L.-S. 1998. A new image morphing technique for smooth vista transitions in panoramic image-based virtual environment. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, ACM, New York, NY, USA, VRST '98, 81–90.

CHOI, J., AND SZYMCZAK, A. 2003. On coherent rotation angles for as-rigid-as-possible shape interpolation. In *CCCG*, 111–114.

FANG, H., AND HART, J. C. 2007. Detail preserving shape deformation in image editing. In *ACM Transactions on Graphics (TOG)*, vol. 26, ACM, 12.

FLOATER, M. S. 1997. Parametrization and smooth approximation of surface triangulations. *Computer aided geometric design 14*, 3, 231–250.

FLOATER, M. S. 2003. Mean value coordinates. *Computer aided geometric design 20*, 1, 19–27.

FORTUNE, S. 1987. A sweepline algorithm for voronoi diagrams. *Algorithmica 2*, 1-4, 153–174.

GOTSMAN, C., AND SURAZHSKY, V. 2001. Guaranteed intersection-free polygon morphing. *Computers & Graphics 25*, 1, 67–75.

GUENNEBAUD, G., JACOB, B., ET AL., 2015. Eigen v3. http://eigen.tuxfamily.org.

GUPTA, H., AND WENGER, R. 1997. Constructing piecewise linear homeomorphisms of simple polygons. *Journal of Algorithms 22*, 1, 142–157.

IGARASHI, T., MOSCOVICH, T., AND HUGHES, J. F. 2005. As-rigid-as-possible shape manipulation. *ACM transactions on Graphics (TOG) 24*, 3, 1134–1141.

JIANG, X., BUNKE, H., ABEGGLEN, K., AND KANDEL, A. 2002. Curve morphing by weighted mean of strings. In *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, vol. 4, IEEE, 192–195.

KEIL, M., AND SNOEYINK, J. 2002. On the time bound for convex decomposition of simple polygons. *International Journal of Computational Geometry & Applications 12*, 03, 181–192.

KRANAKIS, E., AND URRUTIA, J. 1999. Isomorphic triangulations with small number of steiner points. *International Journal of Computational Geometry & Applications 9*, 02, 171–180.

LI, X.-Y., JU, T., AND HU, S.-M. 2013. Cubic mean value coordinates. *ACM Trans. Graph. 32*, 4 (July), 126:1–126:10.

MUROTA, K. 1983. Lu-decomposition of a matrix with entries of different kinds. *Linear Algebra and its Applications 49*, 275–283.

SARRATE, J., PALAU, J., AND HUERTA, A. 2003. Numerical representation of the quality measures of triangles and triangular meshes. *Communications in numerical methods in engineering 19*, 7, 551–561.

SCHAEFER, S., MCPHAIL, T., AND WARREN, J. 2006. Image deformation using moving least squares. In *ACM Transactions on Graphics (TOG)*, vol. 25, ACM, 533–540.

SEDERBERG, T. W., GAO, P., WANG, G., AND MU, H. 1993. 2-d shape blending: an intrinsic solution to the vertex path problem. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, ACM, 15–18.

SORKINE, O., AND ALEXA, M. 2007. As-rigid-as-possible surface modeling. In *Symposium on Geometry processing*, vol. 4.

SRIVASTAVA, A., KLASSEN, E., JOSHI, S. H., AND JERMYN, I. H. 2011. Shape analysis of elastic curves in euclidean spaces. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 33*, 7, 1415–1428.

SUMNER, R. W., AND POPOVIĆ, J. 2004. Deformation transfer for triangle meshes. *ACM Transactions on Graphics (TOG) 23*, 3, 399–405.

SURAZHSKY, V., AND GOTSMAN, C. 2004. High quality compatible triangulations. *Engineering with Computers 20*, 2, 147–156.

SURI, S. 1986. A linear time algorithm for minimum link paths inside a simple polygon. *Computer Vision, Graphics, and Image Processing 35*, 1, 99–110.

VAN DER VORST, H. A. 1992. Bi-cgstab: A fast and smoothly converging variant of bi-cg for the solution of nonsymmetric linear systems. *SIAM Journal on scientific and Statistical Computing 13*, 2, 631–644.

WOLBERG, G. 1998. Image morphing: a survey. *The visual computer 14*, 8, 360–372.

XU, D., ZHANG, H., WANG, Q., AND BAO, H. 2006. Poisson shape interpolation. *Graphical Models 68*, 3, 268–281.

YANG, W., WANG, X., AND WANG, G. 2014. Part-to-part morphing for planar curves. *The Visual Computer 30*, 6-8, 919–928.