High Quality Compatible Triangulations for Planar Shape Animation

Zhiguang Liu
INRIA MimeTIC, Rennes, France

Liuyang Zhou*
Shenzhen Zhiyan Technology Limited, China

Howard Leung
City University of Hong Kong, Hong Kong

Franck Multon
M2S/INRIA MimeTIC, Rennes, France
University Rennes 2, Rennes, France

Hubert P. H. Shum
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 use the common valid vertex pair to uniquely decompose the source and target polygons into pairs of sub-polygons, in which each concave sub-polygon is triangulated. Second, within each sub-polygon pair, we map the triangulation of a concave sub-polygon onto the corresponding sub-polygon using linear transformation, thereby generating 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 to existing methods with fewer long thin triangles and smaller triangle deformation values during shape morphing. These advantages enable us to create more consistent rotations for rigid shape interpolation algorithm and facilitate a 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 texture mapping.

KEYWORDS

character animation, shape morphing, compatible triangulation

1 INTRODUCTION

Planar shape morphing, also known as shape blending, aims to smoothly transform a source polygon into a target polygon [6, 7, 29]. 2D morphing techniques are used widely in animation and special effects packages, such as Adobe After Effects and HTML5. Given a sparse set of source data (polygon shapes), it's difficult to apply machine learning methods to synthesize in-between animation. In such a case, shape morphing algorithms based on the input data is necessary. With such algorithms, We can create character animations similar to data-driven methods with a small number of target shapes. This is done by interpolating some key shapes extracted from the source data to synthesize animation. The key research focus here is to synthesize high-quality character animation that avoids collapsing or overlapping of polygons during the morphing process.

2D image deformation algorithms such as the rigid shape deformation in [15, 22] 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. Previous attempts to tackle the shape morphing problem 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 each other, resulting in geometrically incorrect transformations. While other image space techniques such as [8, 22] achieve pleasant blending results, they usually suffer from overlapping problems due to the lack of topology information.

Previous work [1, 4, 12, 26] has shown that computing compatible triangulation can successfully create smooth transformations for both the boundary and interior of a shape. However, in many situations, compatible meshes can only be generated if Steiner points are added, which do not belong to the vertices of the polygon. [2] first started the study of compatible triangulation 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 that increases the morphing complexity. On the other hand, this method generates many long thin triangles, which can result in inconsistent rotations for shape interpolation algorithms such as [1]. [26] constructed compatible meshes based on link paths, which requires a small number of Steiner points, but at a high computational cost that is prohibitive.

We observed that the majority of existing compatible triangulation approaches 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 triangulation of two simple polygons, which are defined as planar shapes with non-intersecting edges that form a closed path. Our method produces compatible meshes with fewer long thin triangles and a smaller number of Steiner points, which enables smooth transformations from one shape to another.

*Corresponding author: liuyang.zhou@webot.ai

The major contributions of this paper are summarized as follows: First, we propose a new algorithm to calculate compatible polygon decomposition based on common valid vertex pairs, which results in flexible decompositions of the source and target polygons. Second, for each iteration, we choose one common valid vertex pair that can maximize the minimum interior angle to compatibly partition the source and target polygons, which increases mesh quality. The increase in the ratio of regular triangles leads to a smoother transition during shape morphing and texture mapping.

A preliminary research was presented in [19], in which a basic system to construct the compatible triangulations for two simple polygons was presented. Compared with such a work, our new compatible polygon decomposition algorithm is more flexible and leads to a better mesh quality with a lower number of Steiner points, as illustrated in Fig. 4 and Table 2. The method of [19] generates different triangulation results if we start the convex decomposition from the source or target polygon. However, our method always produces the same triangulation results even if start from different directions. This is because we consider the source and target polygon at the same time using common valid vertex pairs. Generally, our algorithm is faster than that of [19], please refer to Section 4 for more details. We have conducted extensive experiments to analyze the influence of the mesh quality on shape morphing.

2 RELATED WORK

Planar shape morphing involves two sub-problems: vertex correspondence and vertex path computation [23]. 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 [1, 2, 17]. (2) Iteratively partitioning the source and the target polygons until both inputs become a set of triangles [4, 14, 26, 27].

[2] 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, [1] employed Delaunay triangulations to reduce the Steiner points. [17] proposed another method by which 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 a polygon. Furthermore, although these methods are conceptually simple, they require high computational cost and are not suitable for real-time applications.

[14] 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. [26] simplified the algorithm of [14] and they proposed a new remeshing method to greatly improve the mesh quality by adding a few Steiner points. Their algorithm requires the implementation of many data structures and algorithms in [27] that makes their method algorithmically complex. [4] proposed a new way of finding compatible link paths. Based on this new link path generation algorithm, they used a similar scheme as in [26] to compatibly partition two polygons. Although their algorithm for computing link paths is faster than that of [26], the proportion of regular-shaped triangles (as opposed to long thin triangles) still needs to be improved.

A lot of work has been proposed for interpolating two shapes. [1] 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 [1], [30] presented a 3D morphing method based on Poisson's equation that generated visually 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, [3] proposed a method to consistently assign rotations. [24] 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. [18] introduced a new type of coordinates for Hermite interpolation that can be applied to shape deformation. Other methods such as [5] try to preserve certain properties like smoothness and distortion for 2D shape interpolation.

In this paper, we propose a new method to construct the compatible meshes of two simple polygons. Our method draws inspiration from [9], which uses barycentric coordinates to map a spatial surface triangulation to planar triangulation. However, [9] 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 compatible polygon decomposition algorithm that simultaneously partitions the source and target polygons into a set of sub-polygon pairs such that we can solve the compatible mapping from a sparse linear system for each sub-polygon pair. On the other hand, the resulted initial triangulation may still contain some long thin triangles that need to be improved. We propose some efficient schemes to further improve the mesh quality.

3 COMPATIBLE TRIANGULATIONS

As illustrated in Fig. 1 (a-b), 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 $\mathcal{T}_\mathcal{P}$ and $\mathcal{T}_Q$ as the triangulations of polygon P and Q. $\mathcal{T}_\mathcal{P}$ 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 $\mathcal{T}_Q$.

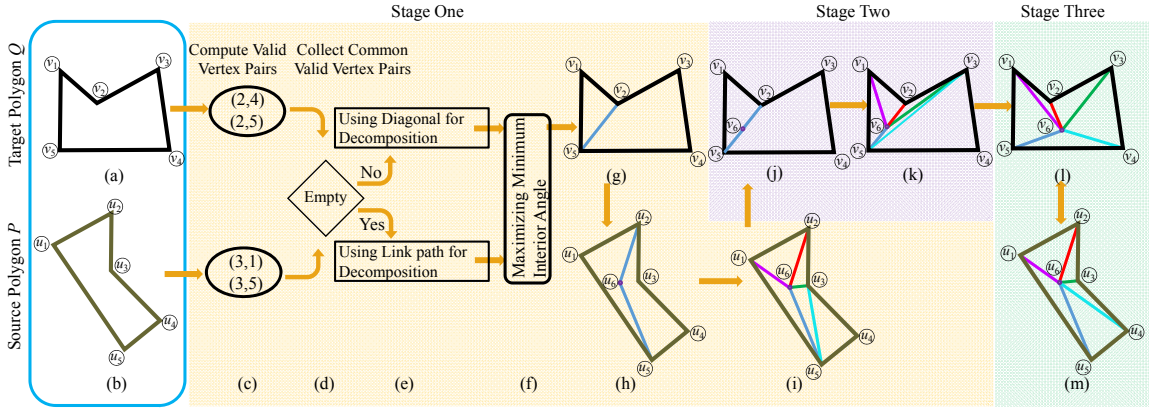


Figure 1: Overview of the proposed framework to compatibly triangulate two simple polygons. (a) The target polygon Q. (b) The source polygon P. (c) We compute the valid vertex pairs for both the source and target polygons. (d) We collect the common valid vertex pairs. (e) We use the common valid vertex pair for compatible decomposition if the common vertex pair exists; otherwise we calculate the link path, e.g. the 2-link path between vertex u_2 and u_5 with blue color shown in (h). (f-h) We use the polyline found in (e) that maximizes the minimum angle to decompose the source and target polygons. (i) We triangulate each sub-polygon p_i of source polygon P using Delaunay triangulation. (j) We may need to add some Steiner points on the edge of sub-polygon q_i to keep equivalent topology. (k) 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. (l-m) 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.

(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.

The core of our framework is that we propose a new algorithm for partitioning the source and target polygon pairs, which is more flexible to increase the mesh quality. Given two simple polygons P and Q with a boundary vertex correspondence as illustrated in Fig. 1 (a-b), our algorithm works in three stages. First, we compatibly decompose the source polygon P and the target polygon Q into sub-polygon pairs $(p, q) = \bigcup(p_i, q_i)$ as shown in Fig. 1 (c-g), where either the target sub-polygon q_i or the corresponding source sub-polygon p_i is convex. Considering a sub-polygon, e.g. p_i of P, we triangulate p_i using Delaunay triangulation as illustrated in Fig. 1 (h-i). 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 as shown in Fig. 1 (j-k). Third, we refine the compatible mesh to improve the mesh quality shown in Fig. 1 (l-m), which is important for high quality morphing in 2D animation, special effects for movies and texture mapping.

3.1 Compatible Decomposition of the Target and Source Polygons

In the first phase, we compatibly decompose the source and target polygons, P and Q, into pairs of sub-polygons. In a simple polygon, a vertex $u \in U$ is *convex* if the angle α formed by two edges at u is less than π radians; otherwise u is considered to be *concave*. Our goal is to turn some concave vertices into convex ones through the decomposition and construct pairs of sub-polygons from the source and target polygons such that each of a sub-polygon pair contains at least one convex sub-polygon.

Without loss of generality, we assume the source and target polygons P and Q each 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 $v_{C+1}, ..., v_N$. Similarly, we label $u_1, ..., u_{C'}$ as the concave vertices and $u_{C'+1}, ..., u_N$ as the convex vertices of P. We call a vertex pair (i, j) of P valid if u_i is visible from u_j and at least one of the two vertices is a concave vertex, e.g. (1, 4) is valid as shown in Fig. 2. If two vertices are visible to each other but they are not a valid pair, then it implies that both vertices are convex such as vertex pair (2, 4) as illustrated in Fig. 2. A diagonal $u_a u_b$ of P is a line segment that joins vertex u_a and u_b of P and remains strictly inside P. A diagonal such as $u_2 u_4$ shown in Fig. 2 that connects two convex vertices is redundant in our compatible decomposition algorithm because it can be removed and the two convex sub-polygons on its sides can be merged into a convex polygon. Therefore, for the construction of a compatible decomposition, we consider only the diagonals that connect two vertices that belong to valid vertex pairs.

In some cases, compatible triangulation can only be constructed if Steiner points are added. In order to introduce the minimum number of Steiner points, we need to search for all the potential decomposition combinations in the solution space. Thus, there can be an exponential number of ways of decomposing a simple polygon into convex sub-polygons using the valid vertex pair, which forbids the practical use of the algorithm. Previous work converted the compatible triangulation problem into a common base domain [1, 2] or used a divide-and-conquer methods [4, 19, 25] to iteratively partition the source and target polygons. However, these methods

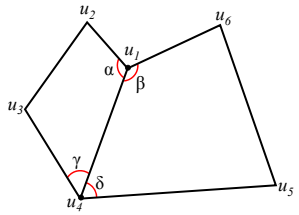


Figure 2: A valid vertex pair $(1, 4)$ used to partition the source polygon, which yields four interior angles between vertex u_1 and u_4.

may either be too complex for real-time application or produce a mesh with poor quality. Therefore, we want to find an efficient compatible triangulation algorithm with an improved mesh quality compared with the existing work.

We start from the source polygon P and find all the valid vertex pairs VP_P for P, similarly, we find the valid vertex pairs VP_Q for the target polygon Q. Among all the valid vertex pairs in VP_P and VP_Q, we collect the common valid vertex pairs $VP = VP_P \cap VP_Q$ that appear in both VP_P and VP_Q. The best partition for P and Q is the common valid vertex pair that generates the maximum minimum interior angle $IntAng$ by:

$$(a, b) = \arg \max_{\substack{v_a, v_b \in V \\ u_a, u_b \in U \\ a \neq b}} \min\{IntAng_P(a, b), IntAng_Q(a, b)\} \quad (1)$$

where the $IntAng_P(a, b)$ contains four angles formed by the intersection of the source polygon P and the diagonal $u_a u_b$ that connects a valid vertex pair (a, b). For example, $IntAng_P(1, 4)$ contains $\angle\alpha$, $\angle\beta$, $\angle\gamma$ and $\angle\delta$ in Fig. 2.

Decomposing polygons with Equation 1 generates a balanced angle partition for both the source and target polygons, which maximizes the interior angle for both the source and target sub-polygons in the current iteration. [19] only considered a balanced angle partition for the target polygon; however, the source polygon may still generate small interior angles. [4, 26] only considered balanced index partition of the source and target polygons, which is likely to decrease the mesh quality in terms of the proportion of small angles of compatible meshes discussed in Section 5.2.

In practice, the common valid vertex pair may not always be available in some cases. For example, as shown in Fig. 1(c-d), the intersection of two valid vertex pair sets $\{(2, 4), (2, 5)\} \cap \{(3, 1), (3, 5)\}$ is empty. Here, we apply link path to determine the partition line between two vertices instead of using common valid vertex pair. A link path between vertex u_a and u_b is a polyline within the polygon that joins the vertex pair (a, b) such as vertex pairs $(2, 6)$ and $(6, 5)$ in Figure 1(h) that defines a 2-link path between vertex u_2 and u_5. A minimum link distance for vertex pair (a, 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 $(2, 5)$ in Figure 1(h) is 2. We follow [4] 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-polygon pairs and N_i is the number of vertices for the *i-th* sub-polygon. Algorithm 1 summarizes our polygon decomposition algorithm in an iterative sense.

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

Input: The source and target polygons, P and Q
Output: A decomposition of P, $p = \bigcup p_i$, and Q, $q = \bigcup q_i$, where either p_i or q_i is a convex sub-polygon
convexDecomposition(P, Q)
    if *P or Q is convex* **then**
        exit
    end
    Compute valid vertex pairs VP_P and VP_Q
    Find common valid vertex pairs
        $VP = VP_P \cap VP_Q$
    if *VP is not empty* **then**
        Calculate the best partition by:
            $(a, b) =$
            $\arg \max_{\substack{v_a, v_b \in V \\ u_a, u_b \in U \\ a \neq b}} \min\{IntAng_P(a, b), IntAng_Q(a, b)\}$
        Decompose P and Q using (a, b) that creates two sets of sub-polygons:
            $\{p_i, p_{i+1}\}, \{q_i, q_{i+1}\}$
    else
        Decompose P or Q using link path that creates two sets of sub-polygons:
            $\{p_i, p_{i+1}\}, \{q_i, q_{i+1}\}$
    end
    convexDecomposition(p_i, q_i)
    convexDecomposition(p_{i+1}, q_{i+1})

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 (p_i, q_i) is a pair of sub-polygons and either p_i or q_i is convex. We apply Delaunay triangulation as the initial triangulation of a sub-polygon, which can maximize the minimum interior angle with no extra Steiner points in $O(N_i log N_i)$ [11]. Here, we denote $\mathcal{T}_{p_i}$ as the triangulation of the sub-polygon p_i and aim to construct compatible triangulation $\mathcal{T}_{q_i}$ of q_i based on $\mathcal{T}_{p_i}$.

3.2 Compatible Triangulations Mapping

The compatible decomposition process may introduce Steiner points on the link path of either the source polygon P and target polygon Q. 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 that 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(h). We add a Steiner point v_s for the target sub-polygon q_i on the corresponding line segment $v_a v_b$ based on the linear ratio with the following equation:

$$v_s = \frac{polylineLength(u_b, u_s)}{polylineLength(u_a, u_b)} v_a + \frac{polylineLength(u_s, u_a)}{polylineLength(u_a, u_b)} v_b \quad (2)$$

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(h), 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 (2).

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 3, 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.

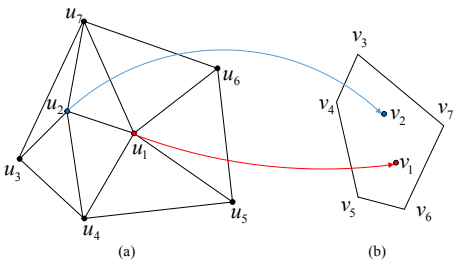


Figure 3: Mapping Steiner points within the source sub-polygon onto the target sub-polygon. (a) The source sub-polygon with Steiner points u_1 and u_2. (b) The corresponding target sub-polygon with unknown Steiner points v_1 and v_2.

Denoting $u_j, j \in \{1, ..., S_i\}$ as a Steiner point that lies within the source sub-polygon p_i, where S_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 [10] 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 continuous data from these adjacent vertices. We represent the Steiner point u_j as a weighted average of its neighboring vertices:

$$u_j = \sum_{k=1}^{M} \lambda_{j,k} u_k, \ \sum_{k=1}^{M} \lambda_{j,k} = 1 \quad (3)$$

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

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

$$v_j = \sum_{k=1}^{M} \lambda_{j,k} v_k, \ \sum_{k=1}^{M} \lambda_{j,k} = 1 \quad (4)$$

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 (3).

We rewrite Equation (4) by breaking the summation term into two sub-terms:

$$v_j = \sum_{k=1}^{S_i} \lambda_{j,k} v_k + \sum_{k=S_i+1}^{S_i+N_i} \lambda_{j,k} v_k, j \in \{1, ..., S_i\}$$
$$v_j - \sum_{k=1}^{S_i} \lambda_{j,k} v_k = \sum_{k=S_i+1}^{S_i+N_i} \lambda_{j,k} v_k \quad (5)$$

where S_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 (5) is equivalent to the following form:

$$Ax = b_1, \ Ay = b_2 \quad (6)$$

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

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

This linear system in Equation 6 has S_i unknown variables and S_i equations. The solution to Equation (6) is unique as the matrix A is non-singular. We apply LU decomposition to solve Equation (6) in $O(S_i^3)$ [20], where S_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 in Figure 5(a). In practice, we found that these long thin triangles can cause numerical problems such as inconsistent rotations for shape morphing. 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 [26]. 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 can be found in Figure 5(b).

4 METHOD COMPLEXITY

In this section, we will analyze the computational complexity of our method. It takes $O(N)$ time to determine the concave vertices and $O(N)$ time to find a valid vertex pair using visibility polygon algorithm [16], where N is the number of vertices of a polygon.

Finding common valid vertex pairs using methods like hash table usually requires $O(1)$ time. Thus, the time cost of decomposing the source and target polygons into pairs of sub-polygons is $O(N^2)$. Finding a corresponding link path for a sub-polygon, e.g. p_i in source polygon P, is $O(N_i^3)$, where N_i is the number of vertices of a source sub-polygon p_i. The Delaunay triangulation can be finished in $O(N_i log N_i)$. Compatible mapping between a pair of sub-polygons requires solving a linear equation using LU decomposition that leads to $O(S_i^3)$ operations, where S_i is the number of Steiner points of sub-polygon p_i.

Table 1 compares the computational complexity between our method and alternative approaches. The main computation of our algorithm is dominated by computing link paths and solving a linear system, i.e. $O(H \cdot max(N_i^3, S_i^3))$, where H is the number of sub-polygon pairs. In practice, the most time consuming part of our algorithm is building the link path as S_i is often smaller than N_i. Generally, our algorithm is faster than that of [19]. This is because our method simultaneously decomposes the source and target polygon and we will stop partitioning a polygon pair if one of them is convex. However, [19] keeps partitioning the target polygon until all the target sub-polygons are convex. Our method is much faster than that of [4, 26] as we solve a small linear sparse system within each sub-polygon pair.

Table 1: Computational complexity: the main computational cost of our method is computing the link paths, where N is the total number of boundary vertices of source polygon P, C_P is the number of concave vertices of P, L and H are the number of sub-polygon pairs created by Liu et al. and our method, N_i and S_i are the number of boundary vertices and the number of Steiner points of the i-th sub-polygon respectively.

Surazhsky-Gotsman, 04			$O(N^3 logN)$
Baxter et al., 09			$O(2N^3)$
Liu et al., 15			$O(L \cdot \max(N_i^3, S_i^3)),\ S_i, N_i \ll N$ $\left\lceil \frac{1}{2} C_Q \right\rceil + 1 \le L \le C_Q + 1$
Proposed Method	Convex decomposition	Common valid vertex pairs computation	$O(N^2)$
		Link paths generation	$O(HN_i^3),\ N_i \ll N$
	Linear system computation		$O(HS_i^3),\ S_i \ll N,\ H \le L$

The matrix A in Equation (6) is sparse and non-symmetric, thus, we further speed it up by using iterative methods such as Bi-CGSTAB [28]. Here, we apply an open library Eigen [13] to solve the sparse linear system. The compatible mapping process of a sub-polygon pair can be even faster before mesh refinement operations and it can be completed in $O(S_i)$. 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.

5 EXPERIMENTAL RESULTS

In this section, we will show the experimental results and present the comparisons between alternative approaches including [26], [4] and [19]. 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.

5.1 Compatible Triangulations

To demonstrate the effectiveness of our method, we implemented the as-rigid-as-possible shape interpolation method introduced in [1]. Figure 4 and 5 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 5.

Figure 5(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 5(b) shows that our compatible meshes can be further refined by methods such as splitting long edges and averaging the area of adjacent triangles.

Given the compatible triangulations of two input polygons, shape interpolation can be applied to create animations showing the transitions from one shape to another. Figure 5(b) shows some interpolation results using our compatible meshes. For more transformations, please see our supplemental demo video or go to the next url: https://youtu.be/DrkzPDqySNg.

5.2 Mesh Quality Evaluation

The quality of the compatible meshes greatly influences the intermediate shapes generated by morphing techniques. In particular, meshes with those long and skinny triangles would suffer from the inconsistent rotation problem [1, 3].

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 [21]. We would like to increase the minimum interior angle of a mesh and decrease the percentage of small angles.

Table 2 shows a quantitative comparison between our algorithm and three other alternative methods. [26] tends to create more long thin triangles than the others. Compared with the results of [26], [4] improves the minimum interior angle. [19] enhances the proportion of regular triangles but sometimes introduces a few more Steiner points than [26]. While our results are similar to [26] in terms of the number of Steiner points, our algorithm creates a much smaller percentage of small angles than [4, 26]. Compared with [4, 19, 26], the minimum angle of our method has been improved greatly while we generally add a fewer number of Steiner points than the alternative methods.

6 CONCLUSIONS

We propose a new method for computing compatible triangulation of two simple polygons and apply them to 2D shape morphing . Our method compatibly decomposes the source and target polygons into sub-polygon pairs and maps the triangulation between a pair of sub-polygons with a sparse linear system.

As an improvement on previous methods, our compatible polygon decomposition algorithm offers a more flexible way of decomposing the source and target polygons such that the

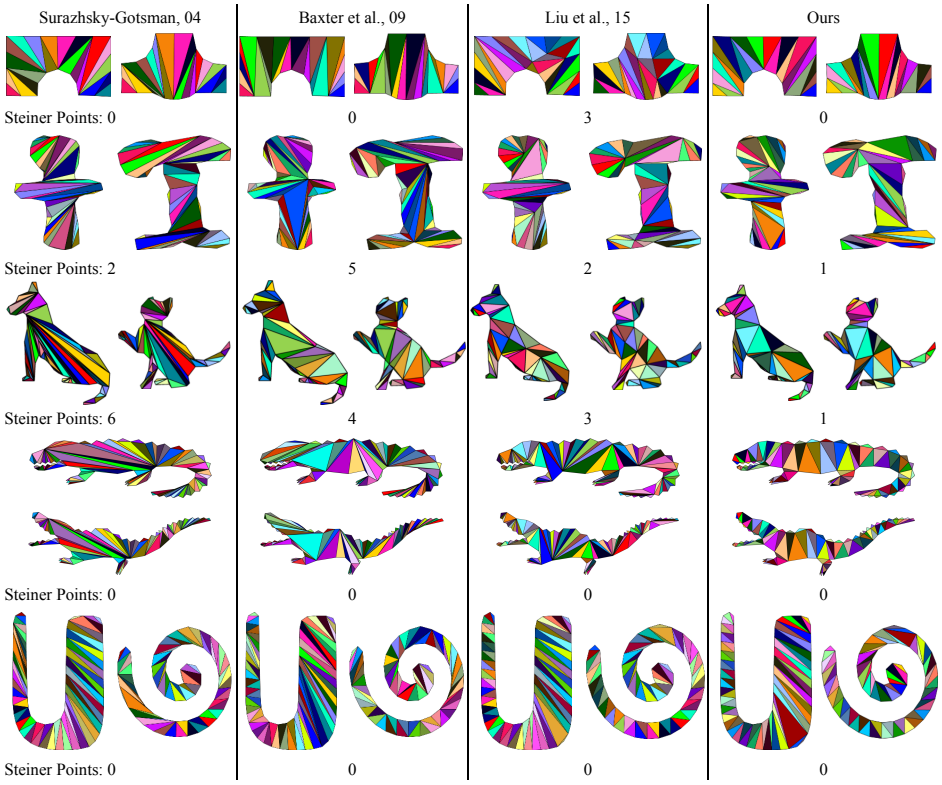


Figure 4: Compatible triangulations comparisons. We compare our results with those of [26], [4] and [19]. While we generally use less Steiner points than the others, our algorithm creates high quality compatible mesh in terms of the proportion of long thin triangles.

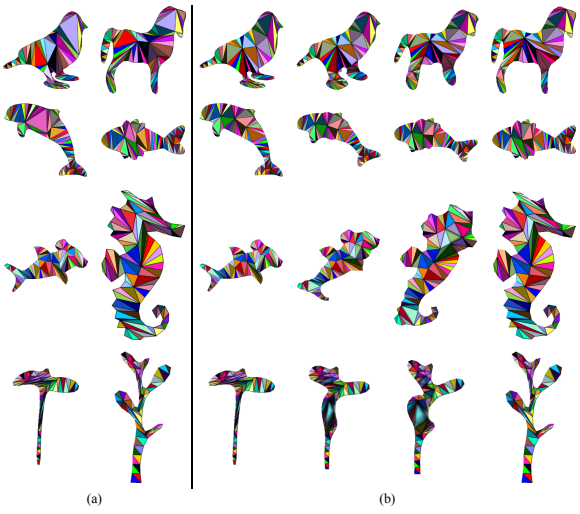


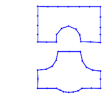
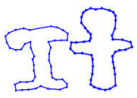
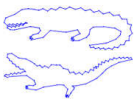
Figure 5: Compatible triangulation results. (a) The initial tessellations of two polygons. (b) mesh refinement and morphing. Note that our compatible mesh can be used to blend shapes with large rotations, e.g. shapes in the third row.

minimum interior angle can be maximized in each iteration. This leads to compatible triangulations with more regular-shaped triangles (as opposed to long thin triangles) as illustrated by the fact that there are fewer triangles whose minimum angles are small under our approach compared with other methods in [4, 19, 26]. Second, compared with our preliminarily work [19], our method generates the same compatible mesh no matter if we start the decomposition from the source or target polygon. 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 a compatible mesh, we simply crossfade between textures in image space. More sophisticated texture blending or image warping algorithms such as [22] could be incorporated into our method. Currently, the intermediate images interpolated are uniquely determined by rigid interpolation method [1], which offers no means of control. 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 [3] to increase user creativity.

Another drawback of our method is that we cannot deal with polygons with holes. One possible solution is that we add a *bridge*

Table 2: Quantitative comparisons between triangulation quality

Shape	Method	#Steiner Point	Minimum angle	Angles ≤10°	Angles ≤15°	Angles ≤20°
	Surazhsky-Gotsman, 04	**0**	1.6730°	11.57%	16.12%	31.27%
	Baxter et al., 09	**0**	3.3052°	10.61%	14.39%	30.30%
	Liu et al., 15	3	3.7557°	**5.35%**	**11.90%**	**22.02%**
	Ours	**0**	**6.4161°**	8.75%	12.87%	26.93%
	Surazhsky-Gotsman, 04	2	0.0441°	27.43%	36.81%	42.36%
	Baxter et al., 09	5	0.9779°	21.91%	29.32%	37.96%
	Liu et al., 15	2	0.9913°	15.27%	22.91%	32.29%
	Ours	**1**	**1.3653°**	**12.49%**	**21.08%**	**26.37%**
	Surazhsky-Gotsman, 04	6	0.4837°	8.60%	13.03%	20.59%
	Baxter et al., 09	4	0.5849°	6.49%	12.42%	18.64%
	Liu et al., 15	3	0.6120°	5.29%	11.64%	17.46%
	Ours	**1**	**1.6855°**	**5.18%**	**9.11%**	**15.72%**
	Surazhsky-Gotsman, 04	**0**	0.0347°	28.96%	35.47%	44.88%
	Baxter et al., 09	**0**	0.0229°	21.45%	29.21%	35.48%	
	Liu et al., 15	**0**	0.3294°	16.01%	23.77%	29.54%
	Ours	**0**	**5.6835°**	**3.99%**	**7.63%**	**12.11%**
	Surazhsky-Gotsman, 04	**0**	0.8893°	12.43%	19.16%	24.13%
	Baxter et al., 09	**0**	2.1933°	10.95%	14.68%	21.89%
	Liu et al., 15	**0**	2.6746°	9.95%	14.18%	20.15%
	Ours	**0**	**2.9338°**	**6.21%**	**10.94%**	**15.92%**

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.

7 ACKNOWLEDGEMENTS

This work was partially supported by the INRIA PRE "Smart sensors and novel motion representation breakthrough for human performance analysis" project, the Engineering and Physical Sciences Research Council (EPSRC) (Ref: EP/M002632/1) and the Royal Society (Ref: IE160609).

REFERENCES

[1] Marc Alexa, Daniel Cohen-Or, and David Levin. 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.

[2] Boris Aronov, Raimund Seidel, and Diane Souvaine. 1993. On compatible triangulations of simple polygons. *Computational Geometry* 3, 1 (1993), 27–35.

[3] William Baxter, Pascal Barla, and Ken-ichi Anjyo. 2008. Rigid shape interpolation using normal equations. In *Proceedings of the 6th international symposium on Non-photorealistic animation and rendering*. ACM, 59–64.

[4] WV Baxter, Pascal Barla, and K-i Anjyo. 2009. Compatible embedding for 2d shape animation. *Visualization and Computer Graphics, IEEE Transactions on* 15, 5 (2009), 867–879.

[5] Renjie Chen, Ofir Weber, Daniel Keren, and Mirela Ben-Chen. 2013. Planar shape interpolation with bounded distortion. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 108.

[6] Cheng-Chin Chiang, Der-Lor Way, Jun-Wei Shieh, and Li-Sheng Shen. 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 (VRST '98)*. ACM, New York, NY, USA, 81–90.

[7] Nadav Dym, Anna Shtengel, and Yaron Lipman. 2015. Homotopic morphing of planar curves. In *Computer Graphics Forum*, Vol. 34. Wiley Online Library, 239–251.

[8] Hui Fang and John C Hart. 2007. Detail preserving shape deformation in image editing. In *ACM Transactions on Graphics (TOG)*, Vol. 26. ACM, 12.

[9] Michael S Floater. 1997. Parametrization and smooth approximation of surface triangulations. *Computer aided geometric design* 14, 3 (1997), 231–250.

[10] Michael S Floater. 2003. Mean value coordinates. *Computer aided geometric design* 20, 1 (2003), 19–27.

[11] Steven Fortune. 1987. A sweepline algorithm for Voronoi diagrams. *Algorithmica* 2, 1-4 (1987), 153–174.

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

[13] Gaël Guennebaud, Benoît Jacob, et al. 2015. Eigen v3. http://eigen.tuxfamily.org. (2015).

[14] Himanshu Gupta and Rephael Wenger. 1997. Constructing piecewise linear homeomorphisms of simple polygons. *Journal of Algorithms* 22, 1 (1997), 142–157.

[15] Takeo Igarashi, Tomer Moscovich, and John F Hughes. 2005. As-rigid-as-possible shape manipulation. *ACM transactions on Graphics (TOG)* 24, 3 (2005), 1134–1141.

[16] Barry Joe and Richard B Simpson. 1987. Corrections to Lee's visibility polygon algorithm. *BIT Numerical Mathematics* 27, 4 (1987), 458–473.

[17] Evangelos Kranakis and Jorge Urrutia. 1999. Isomorphic triangulations with small number of Steiner points. *International Journal of Computational Geometry & Applications* 9, 02 (1999), 171–180.

[18] Xian-Ying Li, Tao Ju, and Shi-Min Hu. 2013. Cubic Mean Value Coordinates. *ACM Trans. Graph.* 32, 4, Article 126 (July 2013), 10 pages.

[19] Zhiguang Liu, Howard Leung, Liuyang Zhou, and Hubert P. H. Shum. 2015. High Quality Compatible Triangulations for 2D Shape Morphing. In *Proceedings of the 21st ACM Symposium on Virtual Reality Software and Technology (VRST '15)*. ACM, New York, NY, USA, 85–94. https://doi.org/10.1145/2821592.2821594

[20] Kazuo Murota. 1983. LU-decomposition of a matrix with entries of different kinds. *Linear Algebra Appl.* 49 (1983), 275–283.

[21] J Sarrate, J Palau, and Antonio Huerta. 2003. Numerical representation of the quality measures of triangles and triangular meshes. *Communications in numerical methods in engineering* 19, 7 (2003), 551–561.

[22] Scott Schaefer, Travis McPhail, and Joe Warren. 2006. Image deformation using moving least squares. In *ACM Transactions on Graphics (TOG)*, Vol. 25. ACM, 533–540.

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

[24] Robert W Sumner and Jovan Popović. 2004. Deformation transfer for triangle meshes. *ACM Transactions on Graphics (TOG)* 23, 3 (2004), 399–405.

[25] Vitaly Surazhsky and Craig Gotsman. 2003. Explicit surface remeshing. In *Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*. Eurographics Association, 20–30.

[26] Vitaly Surazhsky and Craig Gotsman. 2004. High quality compatible triangulations. *Engineering with Computers* 20, 2 (2004), 147–156.

[27] Subhash Suri. 1986. A linear time algorithm for minimum link paths inside a simple polygon. *Computer Vision, Graphics, and Image Processing* 35, 1 (1986), 99–110.

[28] Henk A Van der Vorst. 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 (1992), 631–644.

[29] George Wolberg. 1998. Image morphing: a survey. *The visual computer* 14, 8 (1998), 360–372.

[30] Dong Xu, Hongxin Zhang, Qing Wang, and Hujun Bao. 2006. Poisson shape interpolation. *Graphical Models* 68, 3 (2006), 268–281.