Computers & Graphics (2018)

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

High-quality Compatible Triangulations and their Application in Interactive Animation

Zhiguang Liu[a], Liuyang Zhou[b], Howard Leung[c], Hubert P. H. Shum[d]

[a]*INRIA MimeTIC, Rennes, France*
[b]*Zhiyan Technology (Shenzhen) Limited, China*
[c]*City University of Hong Kong, Hong Kong*
[d]*Northumbria University, Newcastle upon Tyne, UK*

ARTICLE INFO

Article history:
Received August 1, 2018

Keywords: character animation, shape morphing, compatible triangulation

ABSTRACT

We propose a new method to compute compatible triangulations of two polygons in order to create smooth geometric transformations between them. Compared to 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 morphings when transforming the shape from one into another. Our method consists of three stages. First, we use a 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 a linear transformation, thereby generating compatible meshes between the source and the target. Third, we refine the compatible meshes, which creates better quality planar shape morphing with detailed textures. In order to evaluate the quality of the resulting mesh, we present a new metric that assesses the deformation of each triangle during the shape morphing process. Finally, we present an efficient scheme to handle compatible triangulations for a shape with self-occlusion, resulting in an interactive shape morphing system. Experimental results show that our method can create compatible meshes of higher quality as 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 algorithms and facilitate a smoother morphing process. The proposed algorithm is both robust and computationally efficient. It can be applied to produce convincing transformations such as interactive 2D animation and texture mapping. The proposed interactive shape morphing system enables normal users to generate morphing video easily without any professional knowledge.

1. Introduction

Planar shape morphing, also known as metamorphosis or shape blending, allows smoothly transforming a source shape into a target one [1, 2, 3]. Shape morphing techniques have been used widely in animation and special effects packages, such as Adobe After Effects and HTML5, generating visual effects for both the film and television. The key research focus here is to synthesize high-quality character animations that can handle shapes with self-occlusion and avoid collapsing of polygons during the morphing process.

2D image deformation algorithms such as rigid shape deformation in [4, 5] have been extensively explored in the research community. With these algorithms, users can manipulate constrained handlers to deform a given image.

However, such image warping techniques offer a limited range of transformations. Transforming a shape into a significantly different one is difficult due to the lack of feature correspondence.

Planar shape morphing methods offer solutions that determine the trajectory along which the source vertex will travel to the target one. Previous attempts to tackle the shape morphing problem by linearly interpolating the coordinates of each corresponding vertex pair between the source and the target polygons. However, simple linear interpolation sometimes creates intermediate polygons that overlap with each other, resulting in geometrically incorrect transformations. While other image space techniques such as [5, 6] achieve pleasant blending results, they usually suffer from overlapping problems due to the lack of topology information.

Previous work [7, 8, 9, 10] has shown that computing compatible triangulation can successfully create smooth transformations for both the boundary and the interior of a shape. Two triangulations are compatible if they have the same combinatorial structure, i.e., if their face lattices are isomorphic [11]. However, in many situations, compatible meshes can be generated only if additional points, known as Steiner points, are added. Thus, one challenge of building compatible triangulation is to use a small number of Steiner points such that we can reduce the shape morphing complexity. Another challenging problem of computing compatible triangulation is to avoid the generation of some long thin triangles using a computationally efficient algorithm. The long thin triangles can cause inconsistent rotation problems and create artifacts when applied to shape interpolation algorithms [12]. Therefore, a good compatible triangulation contains a small number of Steiner points and keeps a small percentage of long thin triangles. In this paper, we propose a heuristic polygon decomposition method that reduces the overall algorithm complexity.

We observed that most existing compatible triangulation approaches either create a large number of skinny triangles or are too complicated for real-time shape morphing. In this paper, we propose an efficient framework to compute compatible triangulation of two simple polygons defined as planar shapes with non-intersecting edges that form a closed path. Our method produces compatible meshes with fewer long thin triangles and fewer Steiner points, thereby enabling smooth transformations from one shape into another. The proposed method applies to any 2D shape without holes. Here, we use the human shape as an example to illustrate our interactive animation system. We demonstrate an interactive entertainment system that transforms a human into a bird or other objects that people may never experience in real life.

The major contributions of this paper are summarized as follows:

- First, we propose a new algorithm to calculate the compatible polygon decomposition based on the common valid vertex pairs that results in a flexible decomposition of the source and target polygons.
- Second, we present a new metric to measure the quality of the resulting mesh during the shape morphing process.
- Finally, we propose an enhanced scheme that can compute a compatible triangulation for a shape with self-occlusion by introducing a calibration image. To demonstrate the effectiveness of the proposed algorithm, we present an interactive morphing system that uses human silhouette as the source input shape.

Our preliminary research documented in [12] proposed a basic system to construct the compatible triangulation for two simple polygons. Compared with this work, our new compatible polygon decomposition algorithm is more flexible and leads to better mesh quality with fewer number of Steiner points, as illustrated in Fig. 8 and Table 2. The method of [12] generates different triangulation results depending on whether we start the convex decomposition from the source or target polygon. However, our method always produces the same triangulation results even when started from different directions. This is because we consider the source and target polygons at the same time using the common valid vertex pairs. Generally, our algorithm is faster than [12], as shown in Section 5. Compared to [13], we proposed a new metric to measure the quality of the resulting meshes during the rigid shape deformation process. We have also conducted extensive experiments to analyze the influence of the mesh quality on shape morphing such as texture mapping. Finally, to produce sensible transformations, we proposed an improved scheme to deal with compatible triangulations with self-occlusion, and we tested the proposed interactive animation system using a human silhouette as our source shape input.

2. Related Work

Planar shape morphing involves two sub-problems: vertex correspondence and vertex path computation [14]. 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 the vertex u will travel to the 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 the source and target polygons into another common space [11, 7, 15]. (2) Iteratively partitioning the source and the target polygons until both inputs become a set of triangles [16, 17, 9, 10].

Aronov et al. [11] 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, [7] employed Delaunay triangulation to reduce the number of Steiner points. Kranakis and Urrutia [15] 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, it sometimes results in Steiner points on the edge of a polygon.

Gupta and Wenger [17] used the divide-and-conquer method to partition the source and target polygons iteratively. Their algorithm introduced a small number of Steiner points by using the link paths. However, it is not suitable for polygons with a small number of vertices. Surazhsky and Gotsman [9] simplified the algorithm of [17] and proposed a new remeshing method that greatly improves the mesh quality by adding a few number of Steiner points. Their algorithm requires the implementation of many data structures and algorithms in [16], and therefore is algorithmically complex. Baxter et al. [10] proposed a new way to find compatible link paths. Based on this new link path generation algorithm, they used a similar scheme as in [9] to compatibly partition the two polygons. Although their algorithm for computing link paths is faster than [9], 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. Alexa et al. [7] proposed a method that attempted to preserve rigidity. They separately interpolated the rotation and the scale/shear components of an affine transformation matrix, which generated pleasing results with small rotations for most of the cases. Inspired by [7], [18] presented a 3D morphing method based on the 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. As a solution, [19] proposed a method to consistently assign rotations. Sumner and Popovic [20] 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. [21] introduced a new type of coordinates for Hermite interpolation that can be applied to shape deformation. Other methods such as [22] try to preserve certain properties such as the smoothness and the distortion for 2D shape interpolation.

In this paper, we propose a new method to construct the compatible meshes of two simple polygons. Our approach draws inspiration from [23], which uses barycentric coordinates to map a spatial surface triangulation to a planar triangulation. However, [23] requires 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 with a sparse linear system for each sub-polygon pair. On the other hand, the resulted initial triangulation may still contain 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 triangulation of the polygon P and Q. $\mathcal{T}_\mathcal{P}$ and $\mathcal{T}_Q$ are compatible if they have an equivalent topology, which is defined as:

1. There is a one-to-one correspondence between the vertices of $\mathcal{T}_P$ and that of $\mathcal{T}_Q$.
2. There is a 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 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 decompose the source polygon P and the target polygon Q into compatible sub-polygons $(p, q) = \bigcup(p_i, q_i)$ 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, 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 the source sub-polygon p_i onto the corresponding target sub-polygon q_i using a sparse linear system as shown in Fig. 1 (j-k). Third, we refine the compatible meshes 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 the 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 to construct pairs of sub-polygons from the source and target polygons such that each of the 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 while not being a valid pair, then it implies that both vertices are convex such as vertex pair (2, 4) as illustrated in Fig. 2. A

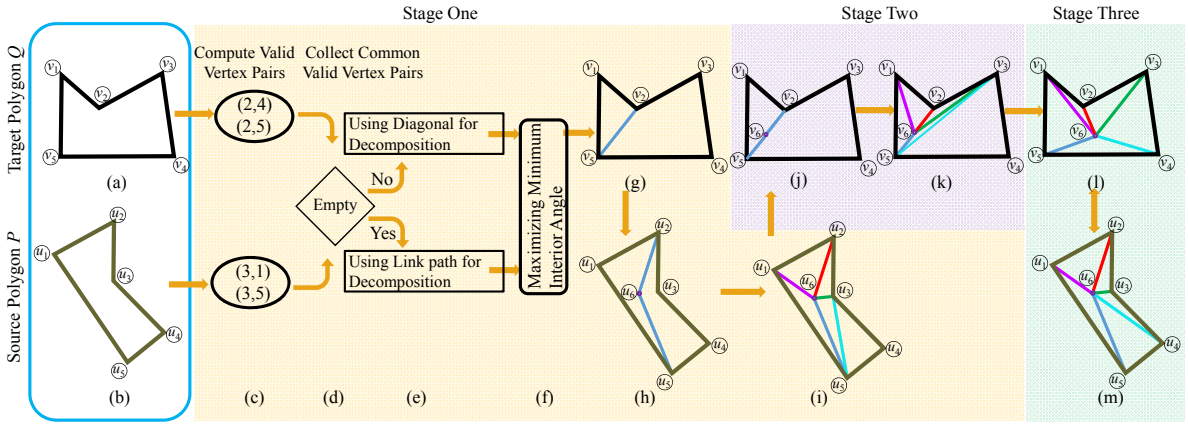

Fig. 1. The overview of the proposed framework to compatibly triangulate two simple polygons. (a) The target polygon Q. (b) The source polygon P. (c) We compute the valid vertex pairs for both the source and target polygons. (d) We collect the common valid vertex pairs. (e) We use the common valid vertex 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 the 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 to improve the interior angles of the mesh.

diagonal u_au_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_2u_4 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.

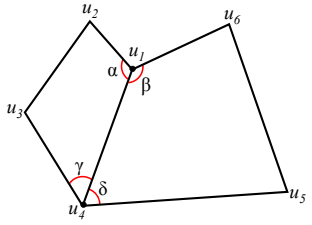

Fig. 2. A valid vertex pair $(1,4)$ used to partition the source polygon, which yields four interior angles between vertex u_1 and u_4.

In some cases, the compatible triangulation can be constructed only 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. As a result, there can be an exponential number of ways to decompose 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 [11, 7] or used a divide-and-conquer methods [24, 10, 12] to iteratively partition the source and target polygons. However, these methods may either be too complex for a real-time application or produce a mesh of poor quality. This motivates us to design an efficient compatible triangulation algorithm with improved mesh quality.

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_au_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 of both the source and target sub-polygons in the current iteration. Liu et al. [12] only considered a balanced angle partition for the target polygon; however, the source polygon may still generate small interior angles. Previous methods such as [9, 10] only considered balanced index partition of the source and target polygons, which is likely to decrease the mesh quality regarding the proportion of small angles in the compatible meshes, which will be discussed in Section 6.2.

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 a link path to determine the partition line between two vertices instead of using the 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), which 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

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})

example, the minimum link distance for vertex pair $(2, 5)$ in Figure 1(h) is 2. We follow [10] to compute the link path with the 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 recursive polygon decomposition algorithm.

By this stage, we have compatibly decomposed the source polygon P and the 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 $O(N_i log N_i)$ [25]. Here, we denote $\mathcal{T}_{p_i}$ as the triangulation of the sub-polygon p_i and aim to construct the 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 or the target polygon Q. Moreover, to improve the mesh quality, the mesh refinement process detailed in Section 3.3 creates 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 the target sub-polygon q_i based on the simple line-segment-length proportion principle. For (2), we solve the mapping with a sparse linear system.

3.2.1. Mapping the Steiner Points onto the Link Path of the 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 place the vertex v_6 on the line segment $v_2 v_5$ using Equation (2).

3.2.2. Mapping the 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.

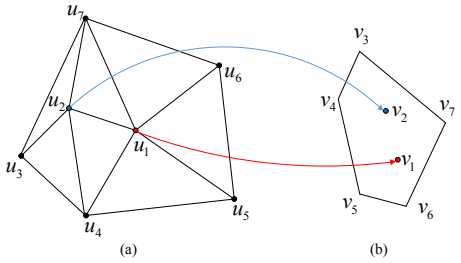

Fig. 3. Mapping the Steiner points within the source sub-polygon onto the target sub-polygon. (a) The source sub-polygon with the Steiner points u_1 and u_2. (b) The corresponding target sub-polygon with the unknown Steiner points v_1 and v_2.

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 the Steiner points within p_i. We use the barycentric coordinates λ to map the Steiner point u_j of the source sub-polygon p_i onto the Steiner point v_j of the target sub-polygon q_i. Here, we employ the Floater's mean value coordinates [26] 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 allow us to generate continuous data from these adjacent vertices. We represent the Steiner point u_j, including the Steiner points on the link path of source polygon and Steiner

points inside the source polygon, as a weighted average of its neighboring vertices:

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

where M is the total number of points including the boundary vertices and the 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 the source sub-polygon p_i onto the corresponding Steiner point $v_j \in V^{q_i}$ of the 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.

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

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:

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

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:

$$Ax = b_1, \quad Ay = b_2 \tag{6}$$

where $x = (x_1, ..., x_{S_i})^T$, $y = (y_1, ..., y_{S_i})^T$, and the 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 $O(S_i^3)$ [27], where S_i is the number of Steiner points within target sub-polygon q_i.

In practice, the source sub-polygon p_i maybe concave and we cannot guarantee that any point inside p_i can be mapped onto its corresponding point inside the target sub-polygon q_i using the barycentric coordinates. Our decomposition algorithm generates a pair of sub-polygons in which at least one of the sub-polygons is convex. As shown in Fig. 1 (g) and (h), although the source sub-polygon P_{2345} is concave, its corresponding target sub-polygon Q_{2345} is convex. We can triangulate the target sub-polygon Q_{2345} and map it onto the source sub-polygon P_{2345}. Because the target sub-polygon Q_{2345} is convex, we can map the points inside Q_{2345} onto the countering-points inside P_{2345} using the barycentric coordinates.

3.3. Compatible Mesh Refining

While the compatible meshes generated by our method introduce a small number of Steiner points, there may still be some long thin triangles such as the second column in Figure 7(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 [9]. 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. We follow [28] to employ the minimum interior angle as a criterion to measure the mesh quality. We want to increase the minimum interior angle for both the source and target meshes. We apply the refinement operations to improve a pair of meshes only if these operations can further improve the mesh quality for both the source and target meshes. The smoothed results can be found in Figure 7(b).

4. Computing Compatible Triangulation with Self-occlusion

4.1. The Problem of Shape Morphing with Self-occlusion

As shown in Fig. 4(t=0), the user adopts a pose with self-occlusions, e.g., with the right hand placed in front of the torso and the left hand behind it. We apply the compatible triangulation method discussed in Section 3, which generates the compatible meshes. However, these meshes cannot distinguish the hands and the other body parts due to self-occlusion. We apply the rigid shape interpolation method introduced in [7] to blend the mesh, which results in the transformations shown in Fig. 4. We can see the transformations of the in-between images such as the time slice t=0.2, which dose not make sense because we want the hands of the user to be gradually transformed into the wings of the butterfly.

4.2. Enhancing Shape Morphing with Self-occlusion

To generate sensible transformation, we need to enable our compatible triangulation method to deal with the shape with self-occlusion. However, the shape extracted from an image with self-occlusion does not contain any overlapping information, which makes it hard to compute the compatible triangulation between two shapes with self-occlusion. Thus, we propose an improved scheme to tackle triangulation with self-occlusion by introducing a calibration image.

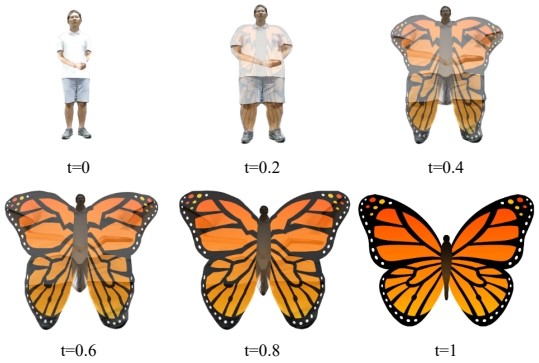

Fig. 4. An example of shape morphing with self-occlusion. The transformation does not make sense as the limbs of the user are not transformed into the corresponding wings of a butterfly due to self-occlusion.

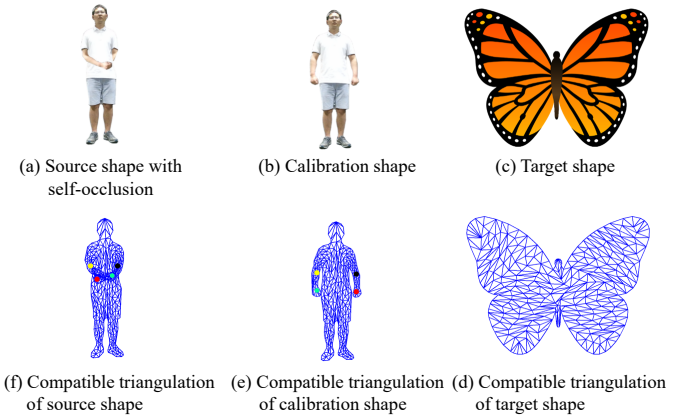

Fig. 5. The overview of compatible triangulation for shapes with self-occlusion using a human posture as an example. Our inputs are the source shape with self-occlusion (a) and the target shape (c). (b) We introduce a calibration shape without self-occlusion. (d-e) We build the compatible triangulation between the calibration shape (b) and target shape (c). We deform the calibration mesh (e) into the source shape with occlusion (a) using the four color-coded control points.

Our inputs are the RGB image of the source object, together with its deformation control points, and the target shape with texture. Here are the steps to build the compatible triangulation between the source shape with self-occlusion and the target shape: (1) We capture a calibration shape of the source object that gives us the full body texture of the source object. Here, the calibration shape is used to extract the topology information, e.g., the deformation control points, and it requires the shape to have no overlapping parts. Additionally, the calibration shape enables us to synthesize texture for transformation with overlapping as it contains the full texture of the source shape. In particular, the calibration shape for human in this case is the full body image as shown in Fig. 5(b). (2) We build the compatible triangulation between the calibration and target shapes, which bridges the shape with self-occlusion and the target shape. (3) We use the control points to deform the mesh of the calibration shape into the source shape with occlusion, which implicitly builds the compatible triangulation between the source shape with occlusion and the target shape. As illustrated in Fig. 5, we take the human posture as an example to explain the process of computing compatible triangulation with self-occlusion.

Fig. 6. Collision detection and depth adjustment. Without appropriate depth assignment, one can see interpenetration (b). We detect overlapping regions and adjust depth on the fly (c).

Using as-rigid-as-possible shape morphing, the points on the medial axis of a shape experience only rotations [7, 29]. Therefore, it is reasonable to use a sparse set of points on the media axis as the deformation control points. Methods such as [30, 31] have been proposed to extract the skeleton of a shape. Since we are using the human as an example, we follow the Kinect's posture estimation method [32] to identify the skeleton of human and use these skeleton joints as our deformation control points. In addition, Kinect also simplifies the work of capturing the silhouette and texture of the source object.

4.3. Collision Detection and Depth Adjustment

As the searching sequence of our polygon decomposition algorithm is similar to the breadth-first search method, the triangles are not stored in sequence. We must be careful when different parts of the shape overlap. If we assign depths inappropriately, the overlapping parts may interpenetrate as shown in Fig. 6(b). We continuously monitor the mesh for self-intersection and assign appropriate depth values to the overlapping parts. The depth value we assigned to each triangle is estimated from the posture reconstruction algorithm studied in [33]. As we have recovered the joint positions for each joint, we know if the hands are in front of or behind the spine as shown in Fig. 6(c).

As shown in the second row of Fig. 12, we blend the human with self-occlusion and the butterfly. Compared with the transformations that do not consider body parts overlap as shown in Fig.4, the results in the second row of Fig. 12 make more sense as we now transform the human's limbs into the butterfly's wings.

5. 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 the visibility polygon algorithm [34], where N is the number of vertices of a polygon. Finding the 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 $O(N_i log N_i)$. The 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 in the 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 [12]. This is because our method simultaneously decomposes the source and target polygons, and we will stop partitioning a polygon pair if one of them is convex. However, [12] keeps partitioning the target polygon until all the target sub-polygons are convex. Our method is much faster than [9, 10] as we solve a small linear sparse system within each sub-polygon pair.

Table 1. The 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. Therefore, we further speed it up by using iterative methods such as Bi-CGSTAB [35]. Here, we apply an open library Eigen [36] to solve the sparse linear system. The compatible mapping process of a sub-polygon pair can be even faster before the 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.

6. Experimental Results

In this section, we will show the experimental results and present the comparisons with the alternative approaches including [9], [10] and [12]. Qualitative analysis is conducted to evaluate the mesh quality between the proposed method and other alternatives. The experiments are conducted on an 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 introduced in [7]. Figure 7 and 8 show some compatible triangulation results and some challenging polygon pairs that are quite different such as the shark and the seahorse in the third row of Figure 7. More examples of comparing morphing against previous triangulation strategies can be found in Figure 13. In practice, in order to create good correspondences between two polygons, shape matching algorithms such as [37] and [38] can be employed to automatically construct a few key correspondences between the source and target polygons, e.g., the vertices around the head, the hands and the feet of the human. Then, the remaining vertices between the user selected key points can be aligned based on linear interpolation. The user can specify a small number of matching points to ensure that the matching points are selected with similarity context. The mismatched correspondences can be detected by observing the generated transformations.

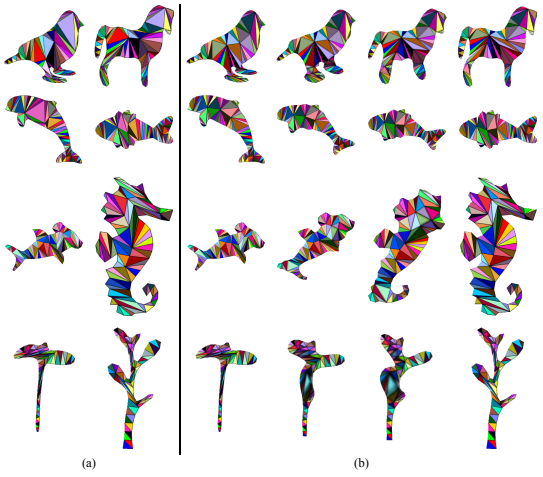

Fig. 7. Compatible triangulation results. (a) The initial tessellations of two polygons. (b) Mesh refinement and morphing. Note that our compatible meshes can be used to blend shapes with large rotations, e.g., shapes in the third row.

Figure 7(a) shows that our initial compatible triangulation contains few long thin triangles and we only need to flip some edges of such 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 averaging the area of adjacent triangles. However, it should also be noted that not every long thin triangle can be further enhanced with our refinement method. For example, in the third row of Figure 7, some thin triangles around the head of the seahorse cannot be improved.

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 7(b) shows some interpolation results using our compatible meshes. For more transformations, please see our supplemental demo video.

6.2. 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 [7, 19].

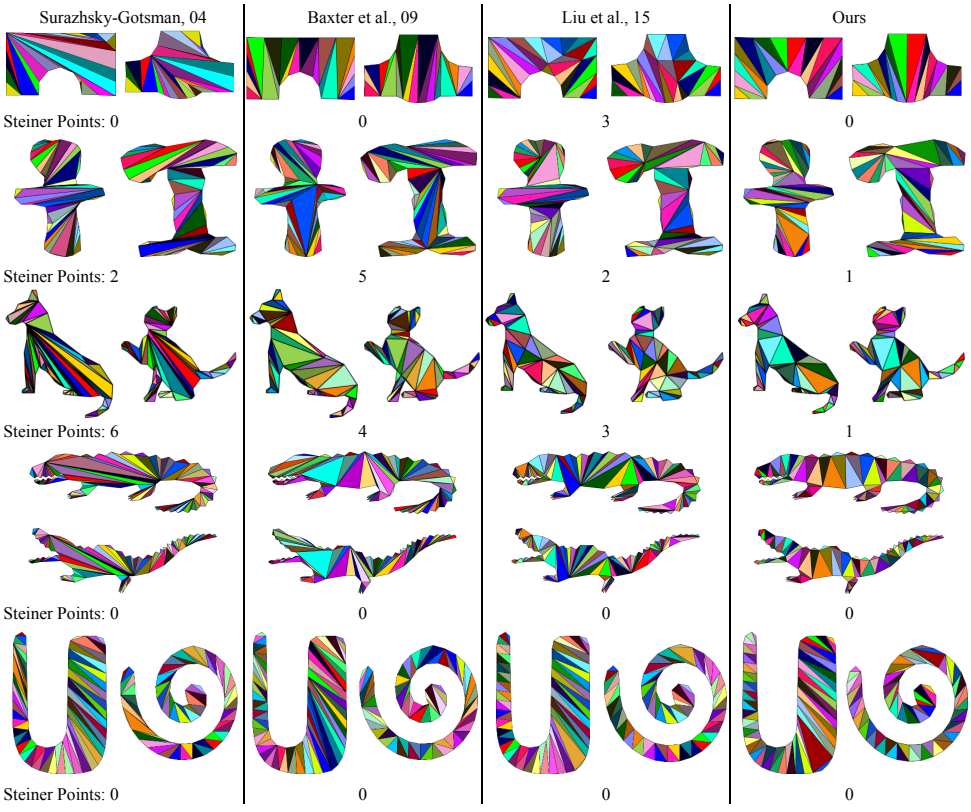

Fig. 8. Compatible triangulations comparisons. We compare our results with [9], [10] and [12]. While we generally use fewer number of Steiner points than the others, our algorithm creates high-quality compatible meshes concerning the proportion of long thin triangles.

We employ the following criteria to measure the mesh quality: (1) the minimum interior angle of a given mesh; and (2) the proportion of angles that are smaller than a certain constant value, which is known to be a reasonable mesh quality criteria [28]. We want 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 alternative methods. [9] tends to create more long thin triangles than the others. Compared with the results of [9], [10] improves the minimum interior angle. [12] enhances the proportion of regular triangles but sometimes introduces a few more Steiner points than [9]. While our results are similar to [9] regarding the number of Steiner points, our algorithm creates a much smaller percentage of small angles than [9, 10]. Compared with [9, 10, 12], the minimum angle of our method has been improved greatly while we generally add a fewer number of Steiner points than the alternative methods. Regarding our computational time, most of the examples in this paper take less than 5 seconds. Additionally, as the compatible decomposition and mapping are highly independent, our method can potentially benefit from the parallel computing of GPU, and hence the entire computing process may be done in real-time.

Table 2. Quantitative comparisons between triangulation quality

Shape	Method	#Steiner Point	Minimum angle	Angles ≤10°	Angles ≤15°	Angles ≤20°	Computation time(second)
	Surazhsky-Gotsman, 04	**0**	1.6730°	11.57%	16.12%	31.27%	21
	Baxter et al., 09	**0**	3.3052°	10.61%	14.39%	30.30%	12
	Liu et al., 15	3	3.7557°	**5.35%**	**11.90%**	**22.02%**	7
	Ours	**0**	**6.4161°**	8.75%	12.87%	26.93%	**3**
	Surazhsky-Gotsman, 04	2	0.0441°	27.43%	36.81%	42.36%	24
	Baxter et al., 09	5	0.9779°	21.91%	29.32%	37.96%	14
	Liu et al., 15	2	0.9913°	15.27%	22.91%	32.29%	6
	Ours	**1**	**1.3653°**	**12.49%**	**21.08%**	**26.37%**	**5**
	Surazhsky-Gotsman, 04	6	0.4837°	8.60%	13.03%	20.59%	27
	Baxter et al., 09	4	0.5849°	6.49%	12.42%	18.64%	15
	Liu et al., 15	3	0.6120°	5.29%	11.64%	17.46%	8
	Ours	**1**	**1.6855°**	**5.18%**	**9.11%**	**15.72%**	**7**
	Surazhsky-Gotsman, 04	**0**	0.0347°	28.96%	35.47%	44.88%	35
	Baxter et al., 09	**0**	0.0229°	21.45%	29.21%	35.48%	18
	Liu et al., 15	**0**	0.3294°	16.01%	23.77%	29.54%	6
	Ours	**0**	**5.6835°**	**3.99%**	**7.63%**	**12.11%**	**4**
	Surazhsky-Gotsman, 04	**0**	0.8893°	12.43%	19.16%	24.13%	29
	Baxter et al., 09	**0**	2.1933°	10.95%	14.68%	21.89%	16
	Liu et al., 15	**0**	2.6746°	9.95%	14.18%	20.15%	9
	Ours	**0**	**2.9338°**	**6.21%**	**10.94%**	**15.92%**	**5**

6.3. *Triangle Deformation Evaluation*

We apply the rigid shape interpolation algorithm [7] to transform a source mesh $\mathcal{T}_{\mathcal{P}}$ into the target one $\mathcal{T}_{\mathcal{Q}}$. Here, we define an edge deformation function to measure the deformation of each triangle face during the transformation. Given the vertices of a source triangle $\mathcal{T}_{\mathcal{P}_1} = \{u_1, u_2, u_3\}$ and the target triangle $\mathcal{T}_{\mathcal{Q}_1} = \{v_1, v_2, v_3\}$, the edge deformation function is defined as:

$$E_{u_a u_b} = \frac{|\,\|u_a u_b\| - \|v_a v_b\|\,|}{\|u_a u_b\|},\ a, b \in \{1, 2, 3\},\ a \neq b \qquad (7)$$

where $\|u_a u_b\|$ is the length of the edge that connects vertex u_a and u_b.

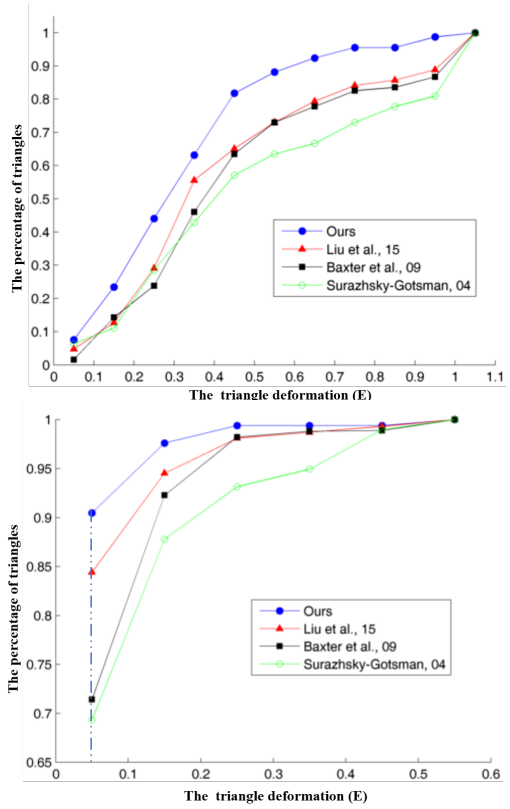

Fig. 9. Mesh deformation evaluation. (top) Dog and cat. (bottom) Alligator.

The deformation of the triangle $\mathcal{T}_{P_f}$ is defined as:

$$E_f = \frac{1}{3} \sum_{u_a, u_b \in \mathcal{T}_{P_f}} E_{u_a u_b} \tag{8}$$

where u_a and u_b are vertices of the f-th source triangle in $\mathcal{T}_P$.

The deformation function E_f measures the deformation degree of each triangle. A source triangle $\mathcal{T}_{P_f}$ will experience very small deformation to transform into the target triangle $\mathcal{T}_{Q_f}$ as E_f approaches 0; Otherwise, a source triangle will experience a big distortion as E_f becomes larger. For a good compatible triangulation, a larger percentage of small deformation E is preferred, which benefits applications such as shape morphing and texture mapping. The horizontal axis of Fig. 9(bottom) shows the deformation values that range from the smallest to the largest deformation values in a mesh. For some specific deformation amount of the horizontal axis, the value of vertical axis demonstrates the percentage of triangles that need a deformation smaller than such a particular deformation value. For example, one triangle in the mesh needs a deformation value of 0.5, and more than 90% of triangles generated by our method experience the deformation values less than 0.5. Fig. 9 shows that our method generally creates the compatible meshes with a higher percentage of triangles that experience small deformation E. On the other hand, our method generates fewer triangles that need large deformation during the shape morphing.

[37, 38]

As illustrated in Fig. 10, we demonstrate the texture mapping using compatible triangulations generated by methods of [9], [10], [12] and ours. The system inputs are a source polygon with texture and a target polygon without texture. We first build the compatible triangulations of two shapes with alternative approaches, as shown in the third and fourth rows in Fig. 8. Based on the compatible meshes, we map the texture of a source shape onto the target one. In general, mapping the texture of a shape onto another very different one always suffers from the texture stretching. As shown in the dog-cat example in Fig. 10 (b-e), nearly all the squares experience some distortion due to the creation of some long thin triangles as shown in the third example in Fig. 8. These long thin triangles need large distortion to be transformed into the target triangles. We can still observe that both [12] and our method preserved some regular squares around the upper body of the cat while our method only generates 1 Steiner point. We then try to map the texture of an alligator between two postures. For the method of [9] in Fig. 8(b), we can see some distortions appear around the abdomen of the alligator. The stretched pattern can also be observed at the back of the alligator for both methods of [10] and [12] as shown in Fig. 8(c) and Fig. 8(d) respectively. Compared with the other methods, ours generates a smoother pattern around the back of the alligator, and the deformation of the abdomen makes more sense. This is because our method generates much more regular triangles that only involve small deformation during the shape morphing process, as shown in Fig. 9(b).

6.4. Interactive Shape Morphing System

To test the effectiveness of our approach, we have implemented a prototype of the proposed interactive shape morphing system using a human posture as the input of source shape. Fig. 11(a) shows the setup of our interactive shape morphing system. We use Kinect as the input device of the source shape. The user stands in front of the Kinect, and the system can be controlled by gesture command. For example, the system starts to capture and extract the shape of the user when the user in the scene raises his/her left hand over the head. More commands such as raising two hands to go back to the default capture view have been implemented.

Fig. 11(b) and 11(c) show the interface of the interactive shape morphing system. As shown in the bottom left of Fig. 11(b), the user adopts a pose as the source input shape. The user can select the target shape in the shape database with a certain gesture such as waving the hand, and then the target shape is rendered in the right of *Edit* window in Fig. 11(b). We then compute the compatible triangulation between the source and target shapes. Finally, we transform the source shape into the target shape based on the compatible meshes. The intermediate results are shown in the *Transformation* window as shown in Fig. 11(c). More animations generated by our system can be found in Fig. 12.

On the other hand, our interactive shape morphing system can be applied to create animation, movie and even special effects. The typical users may not have the professional resources to create some interesting morphing video, our method and system can simplify the work to produce the interactive morphing video.

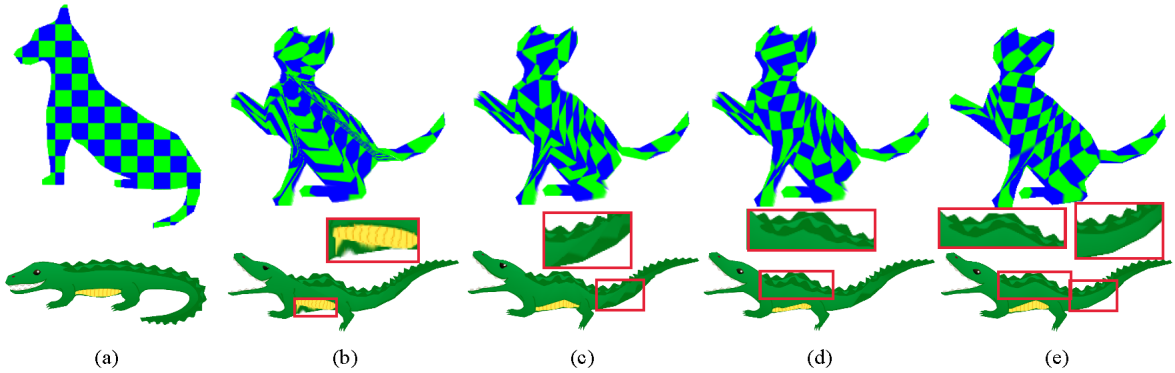

Fig. 10. Texture mapping comparisons. (a) The source shape with texture. Adding texture to the target shape using the compatible meshes generated by methods of (b) Surazhsky-Gotsman, 04. (c) Baxter et al., 09. (d) Liu et al., 15. (e) Ours.

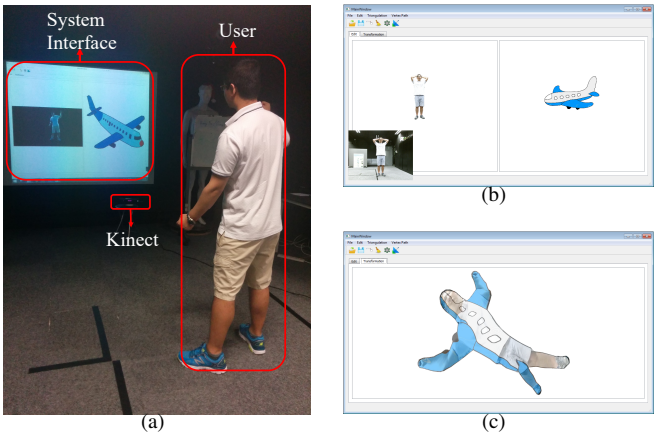

Fig. 11. The interactive shape morphing system. (a) The system setup. The system interface for capturing the source object image (b) and generating transformations (c).

7. Conclusions

We propose a new method to compute the compatible triangulations 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 using a sparse linear system. We present a new metric to measure the quality of the resulting mesh during the transformation. In addition, we propose an enhanced scheme to fix compatible triangulations with self-occlusion that benefits sensible transformations. Finally, to demonstrate the proposed algorithm, we build an interactive shape morphing system using the human silhouette as the source shape input.

Comparing with previous methods, our compatible polygon decomposition algorithm offers a more flexible way to decompose the source and target polygons such that the minimum interior angle can be maximized at each iteration. This leads to compatible triangulations with more regular-shaped triangles as opposed to long thin triangles. This is supported by the analysis that we generate fewer triangles whose minimum angles are small under our approach when compared to methods in [9, 10, 12]. Second, compared to our preliminary work [12], the proposed method generates the same compatible meshes whether we start the decomposition from the source or target polygon. Another advantage is the simplicity of our system that involves only three stages. All we need is to decompose a polygon, to calculate link paths, and to solve a sparse linear system, enabling real-time morphing.

While our method handles well the mapping between shapes, the morphing results need to be further improved. As we focus on generating the compatible meshes, we simply crossfade between textures in the image space. More sophisticated texture blending or image warping algorithms such as [5] can be incorporated into our technique. Currently, the intermediate images interpolated are uniquely determined by a rigid interpolation method [7], which offers no means of control. It would thus be desirable to modify some parts of the intermediate shapes if the users were not satisfied with them. We can explore possible solutions such as the linear constraints proposed in [19] to increase the user controllability.

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

Finally, we want to make better use of the features afforded by commodity depth cameras. It is possible to detect self-occlusions in a video sequence using the depth image captured by commodity depth cameras automatically. However, it is hard to recover the occluded textures for human figures with self-occlusion, which makes it difficult to compute the cross-fade textures for each in-between transformation. That is why we need a calibration image that offers the full body texture for shape morphing with occlusions. An interesting direction of future work would be to skip the calibration image and compute the compatible triangulation directly from shapes with self-occlusions using data from a commodity depth camera.

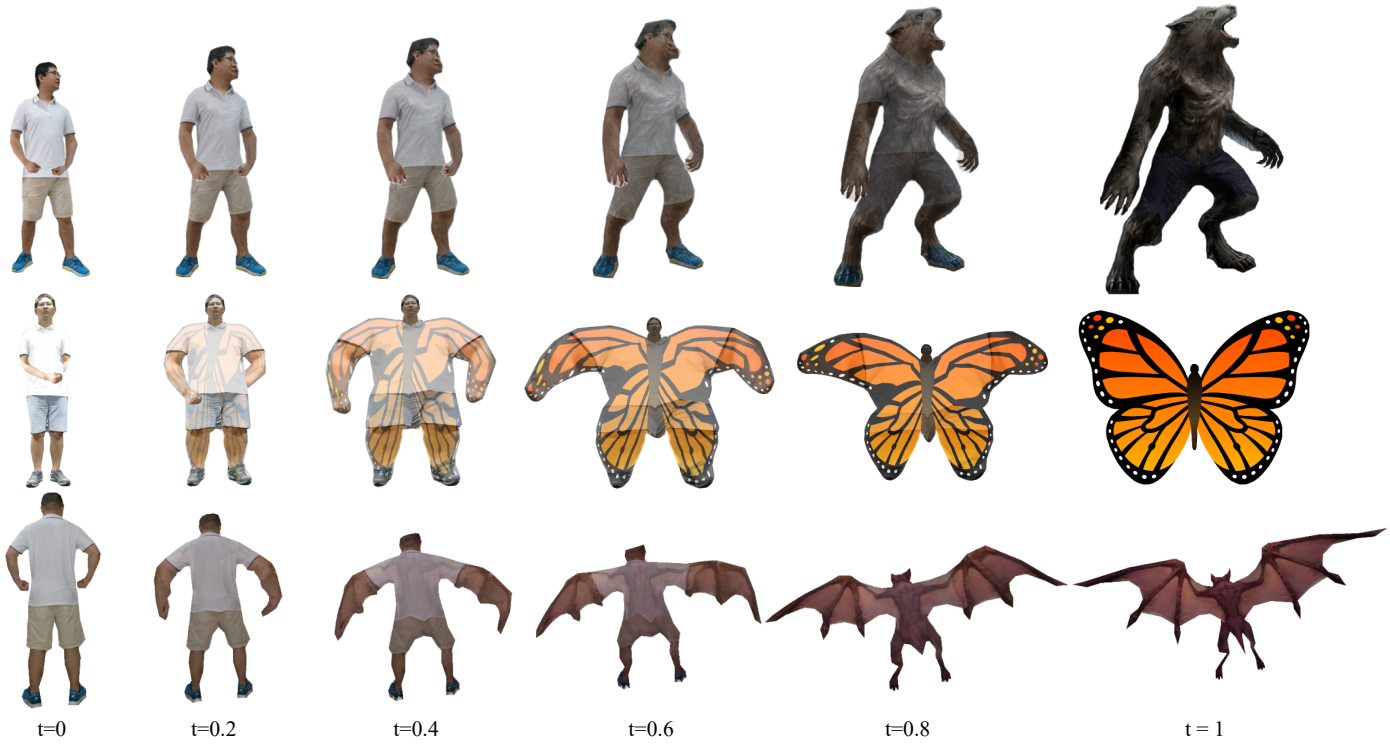

Fig. 12. Producing interactive animation using our interactive animation system: transforming a man into one wolf beast (first row), butterfly (middle row), and bat monster (bottom row).

8. 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] Wolberg, G. Image morphing: a survey. The visual computer 1998;14(8):360–372.
[2] Chiang, CC, Way, DL, Shieh, JW, Shen, LS. 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; New York, NY, USA: ACM. ISBN 1-58113-019-8; 1998, p. 81–90.
[3] Dym, N, Shtengel, A, Lipman, Y. Homotopic morphing of planar curves. Computer Graphics Forum 2015;34(5):239–251.
[4] Igarashi, T, Moscovich, T, Hughes, JF. As-rigid-as-possible shape manipulation. ACM transactions on Graphics 2005;24(3):1134–1141.
[5] Schaefer, S, McPhail, T, Warren, J. Image deformation using moving least squares. ACM Transactions on Graphics 2006;25(3):533–540.
[6] Fang, H, Hart, JC. Detail preserving shape deformation in image editing. ACM Transactions on Graphics 2007;26(3).
[7] Alexa, M, Cohen-Or, D, Levin, D. 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.; 2000, p. 157–164.
[8] Gotsman, C, Surazhsky, V. Guaranteed intersection-free polygon morphing. Computers & Graphics 2001;25(1):67–75.
[9] Surazhsky, V, Gotsman, C. High quality compatible triangulations. Engineering with Computers 2004;20(2):147–156.
[10] Baxter, W, Barla, P, Anjyo, Ki. Compatible embedding for 2d shape animation. IEEE Transactions on Visualization and Computer Graphics 2009;15(5):867–879.
[11] Aronov, B, Seidel, R, Souvaine, D. On compatible triangulations of simple polygons. Computational Geometry 1993;3(1):27–35.
[12] Liu, Z, Leung, H, Zhou, L, Shum, HPH. High quality compatible triangulations for 2d shape morphing. In: Proceedings of the 21st ACM Symposium on Virtual Reality Software and Technology. VRST '15; New York, NY, USA: ACM; 2015, p. 85–94.
[13] Liu, Z, Zhou, L, Leung, H, Multon, F, Shum, HPH. High quality compatible triangulations for planar shape animation. In: Proceedings of the ACM SIGGRAPH ASIA Workshop. 2017, p. 1–8.
[14] Sederberg, TW, Gao, P, Wang, G, Mu, H. 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; 1993, p. 15–18.
[15] Kranakis, E, Urrutia, J. Isomorphic triangulations with small number of steiner points. International Journal of Computational Geometry & Applications 1999;9(02):171–180.
[16] Suri, S. A linear time algorithm for minimum link paths inside a simple polygon. Computer Vision, Graphics, and Image Processing 1986;35(1):99–110.
[17] Gupta, H, Wenger, R. Constructing piecewise linear homeomorphisms of simple polygons. Journal of Algorithms 1997;22(1):142–157.
[18] Xu, D, Zhang, H, Wang, Q, Bao, H. Poisson shape interpolation. Graphical Models 2006;68(3):268–281.
[19] Baxter, W, Barla, P, Anjyo, Ki. Rigid shape interpolation using normal equations. In: Proceedings of the 6th international symposium on Non-photorealistic animation and rendering. ACM; 2008, p. 59–64.
[20] Sumner, RW, Popović, J. Deformation transfer for triangle meshes. ACM Transactions on Graphics 2004;23(3):399–405.
[21] Li, XY, Ju, T, Hu, SM. Cubic mean value coordinates. ACM Transactions on Graphics 2013;32(4):126:1–126:10.
[22] Chen, R, Weber, O, Keren, D, Ben-Chen, M. Planar shape interpolation with bounded distortion. ACM Transactions on Graphics 2013;32(4):108.
[23] Floater, MS. Parametrization and smooth approximation of surface triangulations. Computer aided geometric design 1997;14(3):231–250.
[24] Surazhsky, V, Gotsman, C. Explicit surface remeshing. In: Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing. Eurographics Association; 2003, p. 20–30.

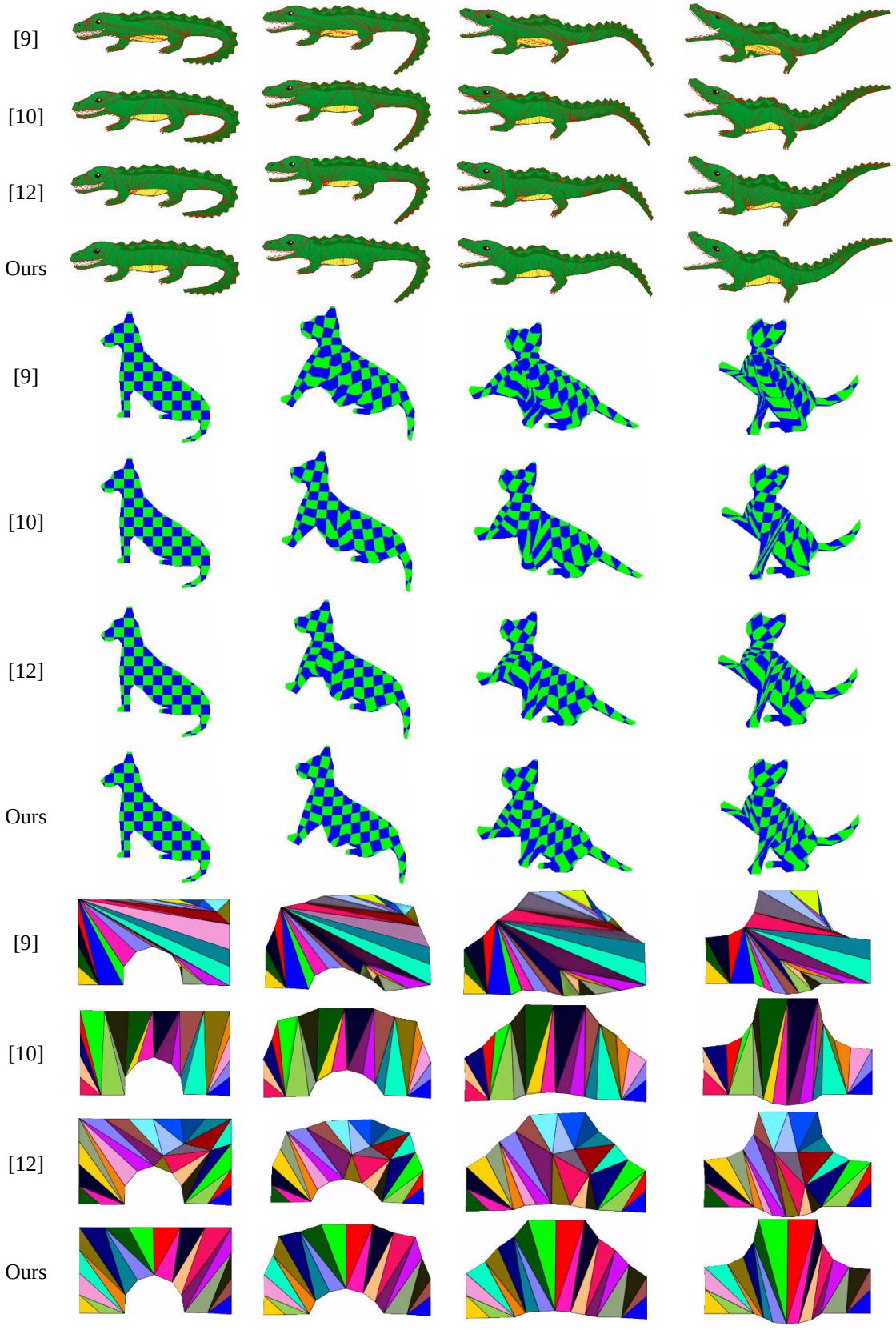

Fig. 13. Shape morphing comparisons by using different triangulation algorithms

[25] Fortune, S. A sweepline algorithm for voronoi diagrams. Algorithmica 1987;2(1-4):153–174.

[26] Floater, MS. Mean value coordinates. Computer aided geometric design 2003;20(1):19–27.

[27] Murota, K. Lu-decomposition of a matrix with entries of different kinds. Linear Algebra and its Applications 1983;49:275–283.
[28] Sarrate, J, Palau, J, Huerta, A. Numerical representation of the quality measures of triangles and triangular meshes. Communications in numerical methods in engineering 2003;19(7):551–561.
[29] Ben-Chen, M, Weber, O, Gotsman, C. Variational harmonic maps for space deformation. ACM Transactions on Graphics 2009;28(3):34:1–34:11.
[30] Au, OKC, Tai, CL, Chu, HK, Cohen-Or, D, Lee, TY. Skeleton extraction by mesh contraction. ACM Transactions on Graphics 2008;27(3):44:1–44:10.
[31] Tagliasacchi, A, Alhashim, I, Olson, M, Zhang, H. Mean curvature skeletons. Computer Graphics Forum 2012;31(5):1735–1744.
[32] Shotton, J, Fitzgibbon, A, Cook, M, Sharp, T, Finocchio, M, Moore, R, et al. Real-time human pose recognition in parts from single depth images. In: CVPR 2011. 2011, p. 1297–1304.
[33] Liu, Z, Zhou, L, Leung, H, Shum, HPH. Kinect posture reconstruction based on a local mixture of gaussian process models. IEEE Transactions on Visualization and Computer Graphics 2016;22(11):2437–2450.
[34] Joe, B, Simpson, RB. Corrections to lee's visibility polygon algorithm. BIT Numerical Mathematics 1987;27(4):458–473.
[35] Van der Vorst, HA. 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 1992;13(2):631–644.
[36] Guennebaud, G, Jacob, B, et al. Eigen v3. http://eigen.tuxfamily.org; 2015.
[37] Belongie, S, Malik, J, Puzicha, J. Shape matching and object recognition using shape contexts. IEEE transactions on pattern analysis and machine intelligence 2002;24(4):509–522.
[38] Mai, F, Chang, CQ, Hung, YS. Affine-invariant shape matching and recognition under partial occlusion. In: 2010 IEEE International Conference on Image Processing. 2010, p. 4605–4608.