Fast and Intuitive Metamorphosis of 3D Polyhedral Models

Tong-Yee Lee*
tonylee@mail.ncku.edu.tw

Po-Hua Huang
bohaw@vision.csie.ncku.edu.tw

Han-Ying Lin
talor@vision.csie.ncku.edu.tw

Chao-Hung Lin
jendon@vision.csie.ncku.edu.tw

Department of Computer Science and Information Engineering
National Cheng-Kung University, ROC.

* Corresponding author

Abstract

In this paper, we present a very fast and intuitive approach to generate the metamorphosis of two genus 0 3D polyhedral models. There are two levels of correspondence specified by animators to control morphs. The higher level requires the animators to specify scatter features to decompose the input models into several corresponding patches. The lower level optionally allows the animators to specify extra features on each corresponding patch for the finer control of correspondence. Once these two levels of correspondence are established, the proposed schemes automatically and efficiently establish a complete one-to-one correspondence between the two models. The performance of the proposed methods is comparable to or even much better than the state-of-the-art in 3D polyhedral metamorphosis. We demonstrate several examples of aesthetically pleasing morphs, which can be very fast and intuitively created with little user interaction times.

Keywords: Polyhedral metamorphosis, embedding, relaxation, warping, merging, SMCC

1. Introduction

Three-dimensional metamorphosis (or morphing) is a widespread technique in entertainment and animation to generate a smooth transition from a source object to a target object. In this paper, we focus on morphs between two genus 0 3D polyhedra. In the literature, most polyhedral morphing techniques could consist of two main steps: one is to find one-to-one correspondence between two polyhedral meshes and the other is to define interpolation paths for each pair of corresponding vertices on the meshes to calculate the in-between shapes. A lot of previous work has been published in the area of 3D polyhedral morphing. There are two excellent surveys of 3D morphing to be referred to in [1,2].

In this paper, our design goal is to provide animators the easy control of morphs and the fast creation of morphs. There are two-level controls for animators to intuitively establish correspondence. First, animators specify scatter features to decompose the models into the corresponding patches, i.e., called morphing patches in this paper. This higher-level control provides the approximate correspondence between two models. If the finer correspondence is required on each patch, the lower-level control allows the animators to specify extra feature points on each patch. With the help of the lower-level control, animators only need to identify few corresponding morphing patches. Therefore, in contrast to other methods such as [3], manual interaction times can be reduced. Afterwards, there are three techniques designed to automatically and efficiently establish a complete one-to-one correspondence between models. In contrast to most other approaches, the proposed methods are very fast and can be computed within a few seconds. These proposed schemes are the main contributions of this paper and are listed below:

- An efficient relaxation method is proposed to embed (i.e., map) 3D morphing patches to 2D regular polygons. In contrast to other relaxation methods such as in [4], our method is computed within a few seconds (usually less than 1 second for our examples).
- A foldover-free warping method is used to align corresponding feature vertices between two embeddings of morphing patches. This is a very essential part in the lower-level control of correspondence.
- An efficient but simple merging method is presented to create a merged embedding that contains the faces, edges and vertices of two

given embeddings. The **SMCC** (structures of minimal contour coverage) is designed to speed up the merging process and the merging is always computed in less a second. In addition, there are several lookup tables are used for easy and efficient implementation.

The rest of our paper is organized as follows. Section 2 reviews the related work of 3D polyhedral metamorphosis. The proposed techniques are presented in Section 3. We experimentally evaluate the proposed schemes and experimentally compare with other wok in Section 4. Finally, the conclusion and future work are given in Section 5.

2. Related Work

Lazarus and Verroust [1] give an excellent survey of previous work on the 3D morphing problem. There are two major classes of 3D morphing technique: volume based approach and surface based approach. In this paper, we focus on surface based approach. For more related work of volume-based approach, please refer to [1]. The surface based approach works on boundary representations such as polyhedral meshes or patch complexes. Methods for this approach usually create an interpolation mesh, which is a common embedding of both source and target meshes. This interpolation mesh is then geometrically deformed to create morphed shapes. This common embedding solves the correspondence problem, associating the vertices or triangles between the source mesh and target mesh. This is a key issue of surface based approach and will be the focus of our paper.

A lot of work has been published in the correspondence issue. Kent et al [6] introduce parameterizations for solving the correspondence problem. Their approach projected star-shaped objects onto spheres to accomplish parameterization. Similarly, Alexa [4] employs a relaxation method to embed polyhedral shapes on the spheres. Lazarus and Verroust [7] introduce skeletons for cylinder-like objects. This approach is an extension of [6] for objects that are star-shaped around an axis. Parent [8] presents a recursive algorithm that automatically finds a correspondence between the surfaces of two objects of equivalent topologies. Decarlo et al. [9] present a method to transform objects with different topologies. The user must identify a sparse control mesh on each surface of objects and this control mesh specifies how to transform one surface to another. Therefore, it will become very complicated and difficult to transform complex shapes. Kanai et al. [10] and Zockler et al. [5] utilize mesh parameterization technique such as harmonic mapping to embed a region of a mesh to a 2D convex polygon. Gregory et al. [3] apply a user-specified control mesh to decompose the surface into a large number of disk-like patches. However, this task is very slow and tedious. Lee et al. [11] employ the MAPS algorithm [12] to parameterize input meshes over simple base domains and an additional harmonic map bringing the latter into the correspondence. Their approach can have fold-over problem and user interaction is required to manually fix this problem.

3. Methodology

3.1 Overview

In this paper, the inputs are genus 0 3D polyhedral models that consist of 1-ring structure triangular meshes. Our work is closest in spirit to Gregory et al.'s [3] work and to Alexa's work [4]. Our overall system structure is similar to their theme. However, we present novel techniques in our design. The main procedures of our design are listed below:

- **Selection of Vertex Pairs and Decomposition into Morphing Patches:** For given two 3D polyhedral models, animators select corresponding vertices on each polyhedron to define correspondences of regions and points in both models. The algorithm automatically partitions the surface of each polyhedron into the same number of morphing patches by computing a shortest path between the selected vertices. The above is corresponding to the high-level control of morphs in our design.
- **3D-to-2D Embedding:** Each 3D morphing patch is mapped onto a 2D regular polygon by the proposed relaxation method.
- **Aligning Feature Vertices:** The interior vertices in the regular 2D polygons are matched by using a foldover-free warping technique. Users can specify extra feature vertices to have a better control over correspondence. This design corresponds to the lower-level control of morphs.
- **Merging, Re-meshing and Interpolation:** The algorithm merges the topological connectivity of morphing patches in the regular 2D polygon. Inserting additional edges retriangulates the regions in the merged regular 2D polygon. This step reconstructs the facets for the new morphing patch, i.e., a common interpolation mesh. Finally, we compute exact interpolation across the common interpolation meshes.

3.2 Specifying Corresponding Morphing Patches

Given two polyhedra A and B, animators interactively design correspondence by partitioning each polyhedron into the same number of regions called morphing patches. Each pair of morphing patches is denoted as (C_i^A, C_i^B), where i is the corresponding patch index. To define each pair of (C_i^A, C_i^B), animators must specify the same number of vertices (i.e., called ***extreme vertices*** [3]), too. These selected vertices also form corresponding point pairs in both models. The boundary of a morphing patch consists of several consecutive chains. Each chain is obtained by computing a shortest path between two consecutive selected vertices. Animators can partition two input polyhedra into arbitrary number of morphing patches, but each patch cannot cross or overlap the other patches. Once the models are partitioned into several corresponding morphing patches, the next is to compute the correspondence of interior vertices of (C_i^A, C_i^B).

3.3 Embedding 3D Morphing Patches on Regular 2D Polygons

In the following, we will first describe the basic idea of the proposed relaxation method to compute 3D-to-2D embeddings. This initial approach requires several iterations to be finished. It can be computationally expensive. Next, we propose to solve the linear system of our relaxation method. In this manner, the embedding can be computed very fast.

Given a pair of 3D morphing patches (C_i^A, C_i^B) defined by n extreme vertices, we embed each on an n-side regular 2D polygon called D_i by a relaxation method. Each n-regular polygon is inscribed in the unit circle and its center is at (0, 0). The relaxation algorithm consists of three steps. First, the extreme vertices of the morphing patches are mapped to the vertices of D_i. Next, each chain of the morphing patch is mapped to an edge of D_i. We need to find the 2D coordinates of non-extreme vertices along each chain. The 2D coordinates of these non-extreme vertices are interpolated based on the arc length of the chain. Third, we compute a 2D mapping for the interior vertices of C_i^A and C_i^B by initially mapping them to the center position (0,0). Then, these vertices are moved step by step by the following relaxation equation and this process will continue until all the interior points are stable, i.e., not moved.

$$p_i' = (1 - \alpha)\, p_i + \alpha \frac{\sum_{j=1}^{k_i} (\mu_j p_j)}{\sum_{j=1}^{k_i} \mu_j} \qquad (1)$$

In equation (1), there are several parameters defined as follows:

- p_i is an interior vertex and its initial position is at (0,0). It represents the 2D mapping of a 3D vertex P_i interior to a morphing patch.
- p_i' is a new position of p_i according to equation (1).
- p_j is a 2D mapping of P_j. P_j is one of P_i's neighbors and k_i is the number of neighbors of P_i in 3D.
- w_j is a pulling weight for p_j, and α controls the moving speed and its value is between 0 and 1.

We attempt to compute a good embedding which preserves the aspect ratio of the original triangle versus the mapped triangle and does not cause too much distortion To determine w_j, our idea is similar to Kanai et al.'s [10] weight formula used in their harmonic mapping. However, we use a different and a simpler formula. For example, in Figure 1, $p_i = 0$ and the weight of $p_j = 2$ is computed by the following equation:

$$w_2 = \cot \theta_1 + \cot \theta_3 \qquad (2)$$

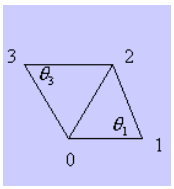


Fig. 1. The definition of a pulling weight

In equation (2), θ_1 is the angle between $edge_{21}$ and $edge_{01}$ and θ_3 is the angle between $edge_{23}$ and $edge_{03}$. These are 3D edges of a morphing patch. In this manner, all w_j can be computed. In equation (2), we can imagine that the whole system is a spring system. During iterations, p_i is pulled by several springs connecting to all its neighbors p_j. The idea behind the equation (2) is that long edges subtending to big angles are given relatively small spring constants compared with short edges that subtend to small angles. Based on the equation (1), we can use iteration methods to find all p_is and terminate iteration when all p_is are stable. However, in this manner, the computation time is not predictable and could be expensive. Therefore, we will not find p_is by iteration method and will solve it by the

following manner.

Using equation (1), as p_i is stable, ideally, $p_i' = p_i$. Thus, we will have the following.

$$p_i' = p_i = (1-\alpha)\, p_i + \alpha \frac{\sum_{j=1}^{k_i} (\mu_j p_j)}{\sum_{j=1}^{k_i} \mu_j}$$

$$\Rightarrow p_i = \frac{\sum_{j=1}^{k_i} (\mu_j p_j)}{\sum_{j=1}^{k_i} \mu_j} \qquad (3)$$

Therefore, assume the number of p_is is N, we can have the following linear system for the proposed relaxation method.

$$\begin{cases} p_1' = p_1 = \dfrac{\sum_{j=1}^{k_1} (\mu_j p_j)}{\sum_{j=1}^{k_1} \mu_j} \\ p_2' = p_2 = \dfrac{\sum_{j=1}^{k_2} (\mu_j p_j)}{\sum_{j=1}^{k_2} \mu_j} \\ \vdots \\ p_N' = p_N = \dfrac{\sum_{j=1}^{k_N} (\mu_j p_j)}{\sum_{j=1}^{k_N} \mu_j} \end{cases} \qquad (4)$$

Let $\lambda_i = \sum_{j=1}^{k_i} \mu_j$ and $i = 1..N$, the above linear system can be represented as the following form:

$$\begin{aligned} &\sum_{j=1}^{k_1} (\mu_j p_j) = \lambda_1 p_1 \\ &\sum_{j=1}^{k_2} (\mu_j p_j) = \lambda_2 p_2 \\ &\vdots \\ &\sum_{j=1}^{k_N} (\mu_j p_j) = \lambda_N p_N \end{aligned} \qquad (5)$$

This linear system is not singular, so that it has a unique solution. Furthermore, for each p_i, the number of its neighbors is small compared to N. Therefore it is a sparse system and can be solved efficiently by using numerical method.

3.4 Aligning the Features and Foldover-Free Warping

Given a pair of morphing patches (C_i^A, C_i^B), (D_i^A, D_i^B) are their corresponding 2D embeddings. Their extreme vertices are automatically aligned by user specification. Using this initial correspondence, we could directly overlay two embeddings to get a merged embedding for morphing. For example in Figure 2 (a), we select a corresponding morphing patch on two given models and the number of extreme vertices is five. There are two extra vertex pairs (A, B) and (a, b) shown in both models, respectively. These extra vertices represent eye corners. In Figure 2(b), we show both (D_i^A, D_i^B) after embedding. It is obvious to see that vertex pairs (A, B) and (a, b) do not align if we directly overlay (D_i^A, D_i^B). Therefore, to compute better correspondence, we usually require animators to specify several extra corresponding features such as vertex pairs (A, B) and (a, b) on both (D_i^A, D_i^B). Then we employ a foldover-free warping function to align (A, B) and (a, b). Non-feature points will be automatically moved by the warping function, too. Like [5], to minimize distortion due to warping, we first move these extra corresponding feature points linearly to the point halfway between them and then perform warping.

Our warping is simply computed as a weighted sum of radial basis function (RBF). Suppose there are n extra feature pairs. Since (D_i^A, D_i^B) are both in 2D, the radial function R consists of two components (R_1, R_2), where each component has the following form.

$$R_j(p) = \sum_{i=1}^{n} a_i^j g(\|p - p_i\|) \quad , j = 1,2 \qquad (6)$$

In equation (6), a_i^j are coefficients to be computed, g is the radial function and p_i is a feature point. For each given p, we compute its new position by $(R_1(p), R_2(p))$ using equation (6). In total, there are $2n$ coefficients to compute. In current implementation, the radial basis function we use is a Gaussian function:

$$g(t) = e^{-\frac{t^2}{\sigma^2}} \qquad (7)$$

In equation (7), the variance σ controls the degree of locality of the transformation. In Figure 2 (c), we show (D_i^A, D_i^B) with warping by two extra feature points. This result is better than that of Figure 2(b). Therefore, we can overlay them now to get a merged embedding for morphing. Sometimes, the warping can lead to fold-over (self-intersections) on (D_i^A, D_i^B).

We need foldover-free embeddings. To solve foldover, we first check if self-intersections occur on (D_i^A, D_i^B) after warping. If self-intersections occur, we simply iterate equation (1) instead of solving equation (5). Usually, it requires a few iterations and self-intersections will not occur. In the following, we show how to check if self-intersections occur.

Our inputs are genus 0 3D polyhedral models with 1-ring structure. Therefore, if there is no self-intersection on both (D_i^A, D_i^B), each interior point of both embeddings must have a complete 1-ring structure in 2D. If any interior point of an embedding has an incomplete 1-ring structure, the self-intersection occurs. To check if a point has a complete 1-ring structure, we compute the following:

$$M = \overrightarrow{pa} \otimes \overrightarrow{pb} \quad (\otimes: \text{the right-hand vector cross product})$$

$$\begin{cases} complete & , M > 0 \\ incomplete & , M <= 0 \end{cases} \qquad (8)$$

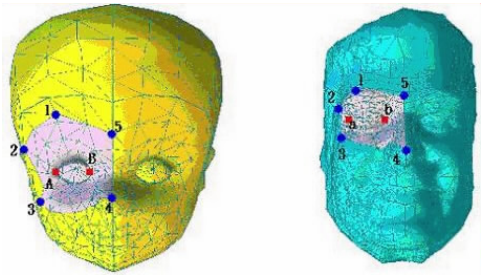
In equation (8), we need to check all nodes at p's 1-ring structure. If any violation (i.e., M <=0) occurs, it is an incomplete ring structure in 2D. Note that the vertices of a triangle are in counterclockwise order. Figure 3 is used to illustrate the equation (8). In Figure 3 (a), before embedding, P (i.e., p's corresponding vertex in 3D) has a complete 1-ring structure. After embedding and warping, a self-intersection occurs as shown in Figure 3 (b). In this case, we simply check all nodes at p's 1-ring structure and find (a, b) violates equation (8).

3.5 Efficient Local Merging

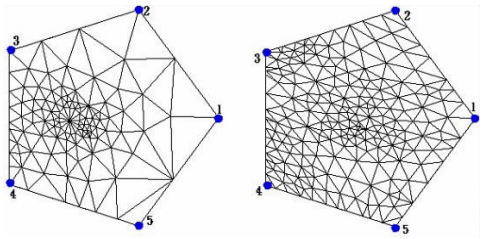
Given two embeddings (D_i^A, D_i^B), we merge them to produce a common embedding that contains the faces, edges and vertices. The complexity of a brute-force merging algorithm is $O(n^2+k)$, where n is the number of edges and k is the number of intersections. This naïve approach globally checks all edges to find the possible intersections. We present a novel method for checking edges locally and efficiently computing the intersections. The complexity of the proposed method is $O(n+k)$. Additionally, to efficiently implement our method, a lookup table is created.

3.5.1 The Classification of the Corresponding Positions

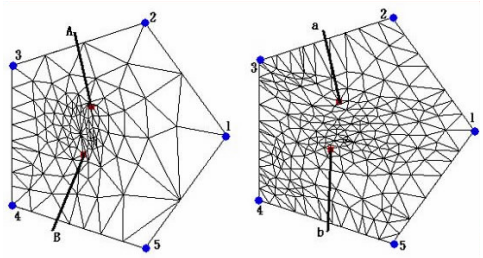
The merging algorithm wants to overlay each edge $\overline{P_S^B P_E^B} \in D_i^B$ on D_i^A, where S and E represent the starting and ending points of a given edge. Since the correspondence of each extreme vertex has been established before embedding by animators, we perform the overlay starting from a $\overline{P_S^B P_E^B} \in D_i^B$, where P_S^B is an extreme vertex. Since D_i^B is a connected planar graph, we can traverse all edges starting from $\overline{P_S^B P_E^B}$ and overlay them on D_i^A edge by edge. We can imagine that an edge $\overline{P_S^B P_E^B}$ consists of an infinite number of points. As we overlay this edge on D_i^A, the corresponding positions of these points have three kinds on D_i^A as illustrated in Figure 4. These three possibilities are:



(a) The user picks five extreme vertices (i.e., blue dots) and two extra feature vertices (i.e., red dots).

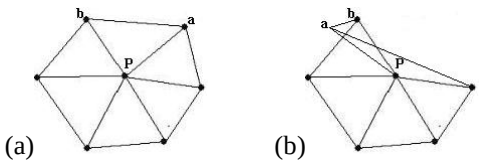


(b) Embeddings without warping.



(c) Embedding with warping by two extra features (i.e., red dots).

Figure 2. Embedding and warping.



(a) (b)

Figure 3 (a) prior to embedding and warping, P has a complete 1-ring structure in 3D and (b) P has an incomplete 1-ring structure in 2D after embedding and warping.

1. Some point $P^B \in D_i^B$ (i.e., $P^B \in \overline{P_S^B P_E^B}$) falls on a vertex P^A of a triangle $T^A \in D_i^A$.
2. Some point $P^B \in D_i^B$ falls on an edge E^A of a triangle $T^A \in D_i^A$.
3. Some point $P^B \in D_i^B$ falls on the interior of a triangle $T^A \in D_i^A$.

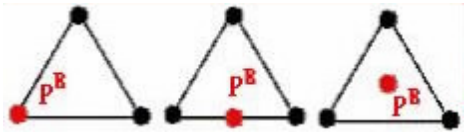


Figure 4. There are three kinds of corresponding positions (i.e., red dot) on D_i^A.

When an edge $\overline{SE}$ (for simplicity, $\overline{SE}$ is interchanged with $\overline{P_S^B P_E^B}$) is overlaid on D_i^A, this edge can be split into several line segments $\overline{se}$ by a triangle $T^A \in D_i^A$. The relationship between $\overline{se}$ and T^A can be classified into eighteen kinds of cases (as shown in Figure 5). For example in Figure 6, the edge $\overline{SE} \in D_i^B$ could be split into the following cases (i.e., according to Figure 5) 15-2-2-6-9-2-2-3, 14-15-2-6-9-2-2-3, or other sequences. But the former splitting sequence generates the minimum number of new points on D_i^A. We shall call such splitting as the optimal splitting.

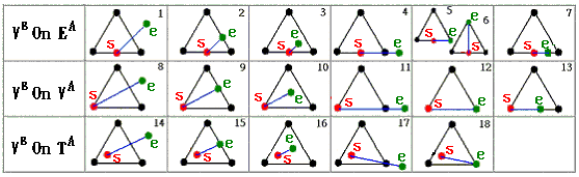


Figure 5. The relationship between a line segment $\overline{se}$ and T^A can be classified into eighteen kinds of cases. The left-most column is classified based on Figure 4.

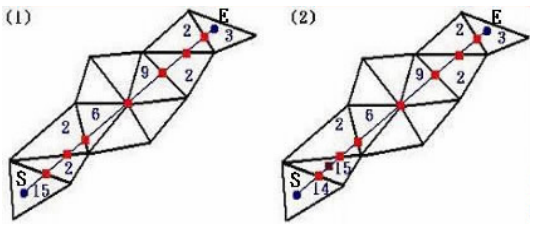


Figure 6. (a) The optimal splitting generates nine new points on D_i^A. (b) The non-optimal splitting generates ten new points on D_i^A. In (a) and (b), the label on each point is made according to Figure 5.

3.5.2 Structures of Minimal Contour Coverage (SMCC)

Whichever a new point generated by the optimal splitting can be found with the help of structures of minimal contour coverage (***SMCC***) on D_i^A. Based on the classification in Figure 4, there are three kinds of SMCC for the corresponding position (Figure 7) defined in the following.

1. For some point $P^B \in D_i^B$ falls on P^A, P^B's SMCC is P^A's 1-ring structure.
2. For some point $P^B \in D_i^B$ falls on E^A, its SMCC is a 4-sided polygon containing E^A.
3. For some point $P^B \in D_i^B$ falls on T^A, its SMCC is a triangle T^A.

In above, P^A, E^A and T^A are all defined in Figure 4.

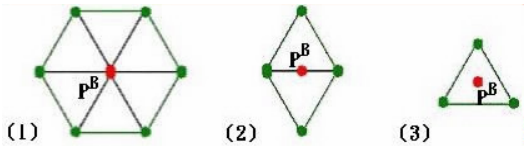


Figure 7. There are three kinds of SMCC for P^B on D_i^A: (1) 1-ring, (2) 4-sided polygon and (3) a triangle.

Assume the starting point S of an edge $\overline{SE} \in D_i^B$ has established its SMCC on D_i^A. If $\overline{SE}$ could generate new intersection points with some edges E^A outside S's SMCC, $\overline{SE}$ must generate a new intersection point with S's SMCC. To the contrary, if $\overline{SE}$ does not intersect with S's SMCC, it also will not intersect with other edges E^A outside S's SMCC (as shown in Figure 8). So the merging can be done locally with S's SMCC.

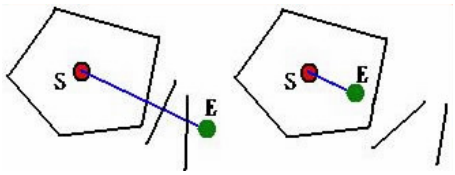


Figure 8. The merging can be locally done with S's SMCC.

The local merging can be further classified into two kinds of the merging conditions. One is called area condition at which the local case $\overline{se}$ is not co-incident with any imaginary line that links s with the contour vertex of SMCC. If $\overline{se}$ is co-incident with any imaginary line, this is called line condition (as shown in Figure 9). The determination of the local merging condition is evaluated by the following:

$M = \overrightarrow{se} \otimes \overrightarrow{sa}$, $N = \overrightarrow{se} \otimes \overrightarrow{sb}$, where $\overrightarrow{se}$, $\overrightarrow{sa}$ and $\overrightarrow{sb} \in R^2$.

$$\begin{cases} \text{Area Condition} & , \quad M < 0 \text{ and } N > 0 \\ \text{Line Condition at } \overline{sa} & , \quad M = 0 \\ \text{Line Condition at } \overline{sb} & , \quad N = 0 \\ \text{Continue searching for other } (\overline{sa}, \overline{sb}) \text{ pairs} & , \quad \text{other} \end{cases} \tag{9}$$

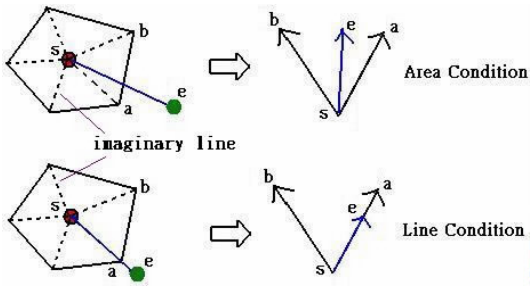


Figure 9. There are two kinds of the local merging conditions.

Based on the above classifications, we can use different formulas (i.e., equation (10) and (11)) to determine whether $\overline{se}$ intersects with S's SMCC or not. The results of equation (10) and (11) lead to different conditions as showed in Figure 10.

1. Area Condition:

Let $V = s + t * \overrightarrow{se}$, and suppose V on the $\overline{ab}$,

$\therefore \overrightarrow{aV} \otimes \overrightarrow{ab} = 0$

Then $\overrightarrow{aV} = s + t * \overrightarrow{se} - a = \overrightarrow{as} + t * \overrightarrow{se}$

$\therefore \overrightarrow{aV} \otimes \overrightarrow{ab} = (\overrightarrow{as} + t * \overrightarrow{se}) \otimes \overrightarrow{ab} = 0$

$\therefore$ Let $Z_1 = \overrightarrow{as} \otimes \overrightarrow{ab}$

$Z_2 = \overrightarrow{se} \otimes \overrightarrow{ab}$

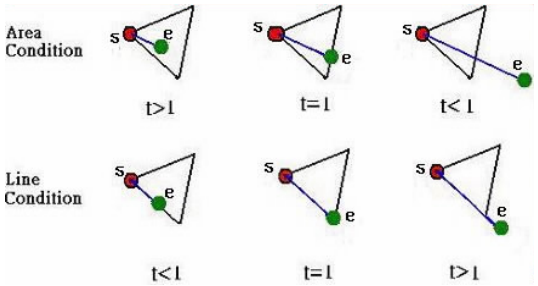
$t = -Z_1 / Z_2$ (10)

2. Line Condition: (assume co-incident with $\overline{sa}$)

$L1 = \left\| \overrightarrow{se} \right\| \qquad L2 = \left\| \overrightarrow{sa} \right\| \qquad t = L1 / L2$ (11)

Based on the analysis of Figure 10 (b), we can replace Figure 5 with Figure 10 (b). Figure 10 (b) can provide us a lookup table to efficiently implement our merging algorithm.

Our merging algorithm needs to establish the SMCC structure prior to proceeding the local merging. In Figure 7, we have demonstrated how to find a SMCC structure. In our design, once a new intersection occurs, we will establish its SMCC immediately for further potential merging. For example, in Figure 11, an edge $\overline{AB} \in D_i^B$ and A's SMCC has been created using Figure 7. Using Figure 10 (b), we find $\overline{AB}$ intersects with A's SMCC at C. Next, we create C's SMCC using Figure 7 again. Again, with the help of Figure 10 (b), the edge $\overline{CB}$ intersects with C's SMCC at D. In this manner, we repeat the above steps until we reach at B. Of course, we need to create B's SMCC, since B could be the starting point of the other edge (i.e., not yet overlaid) from D_i^B. Finally, we recall that in Section 3.5.1 that we start the local merging from an extreme vertex. The SMCC of an extreme vertex is its 1-ring structure. Given two embeddings (D_i^A, D_i^B) in Figure 12, we show a complete sequence of overlaying D_i^B on D_i^A by the proposed method.



(a) The results of the equation (10) and (11).

	Area Condition t < 1	Area Condition t = 1	Area Condition t > 1	Line Condition t > 1	Line Condition t = 1	Line Condition t < 1
V^B On E^A						
V^B On V^A						
V^B On T^A						

(b) Local merging condition table.

Figure 10. Classifications of local merging conditions

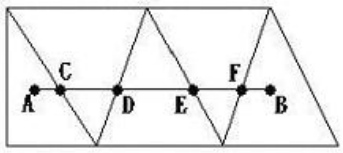


Figure 11. The merging of $\overline{AB}$ can be completed step by step.

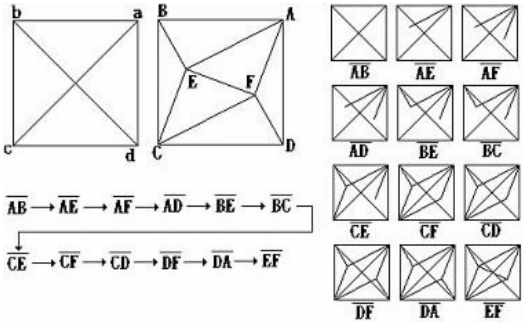


Figure 12. An example of merging is completed step by step by our algorithm.

3.6 Re-triangulate the Merged Embeddings

Once the merging is completed, we produce a non-triangulated planar graph called D^M. In order to re-triangulate the D^M, we need to insert additional edges to re-triangulate D^M. For simplicity, our approach is very straightforward and is described as follows. For each point P^M on D^M, the algorithm must connect the neighboring points of P^M to establish P^M's 1-ring cyclic structure. Our design principle (as shown in Figure 13) is that the inserted edge (i.e., red line in Figure 13) connecting the neighboring points is not allowed to generate any new intersection point with other existing edges (i.e., green lines). It is very easy to check if an inserted edge intersects with other edges or not. We can easily adapt the equation (10) for this purpose.

Figure 13. (a) The inserted $\overline{AB}$ is not legal and (b) P^M's 1-ring cyclic structure is established by inserting several legal edges.

3.7 Reconstructing the Source Models and Interpolation

Once we finish the preceding steps, we have established a complete correspondence between two models. Our merging algorithm produces new points in 2D due to intersections. For these new points, we need to find its corresponding 3D points in both models. We first compute the barycentric representation of a new point in the basis of three old points in 2D. Then, the barycentric representation is used to interpolate positions of these three old points in 3D. These old points are referred to the original vertices on the input models. In this manner we find 3D position of a new point. Similarly, for a new point, we can interpolate its other attributes such as color and texture coordinates if required.

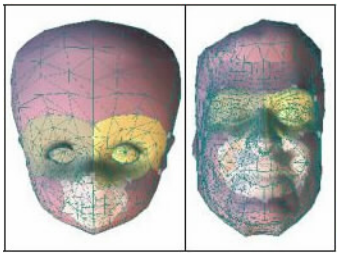
Once the above step is finished, the morphing sequence can be easily generated by linearly moving each vertex from its position in model A to the corresponding position in model B in term of time t. Other authors mentioned this kind of linear interpolation can produces satisfying results in most cases. However, in some special case, the self-intersections can occur. Gregory et al. [3] propose the user-specified morphing trajectory by the cubic spline curves for an alternative to linear interpolation. This simple alternative can be included in near future.

4. Experimental Results

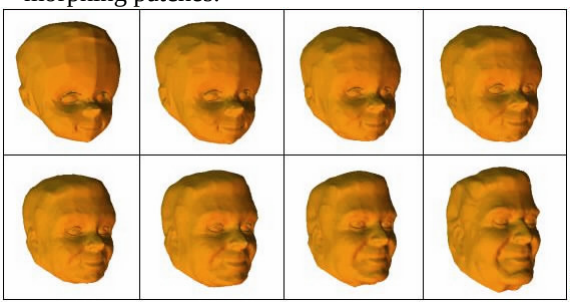
Digital video clips of all morphs used in this paper can be found at http://couger.csie.ncku.edu.tw/~vr/fast_morph.html. Additionally, the friendly GUIs of our system are also demonstrated at this Web site.

4.1 Performance Evaluation and Morphing Examples

In Figure 14, we show an example of morphing from a baby head to a man head. In this example, the number of morphing patches is four and each patch is illustrated using a different color. In addition, there are seven extra feature points (i.e., 4 eye corners, 1 nose tip, 2 mouth corners) used for warping. Figure 14 (c), we show some timing information and the geometric information of two models for this experiment. Times are average execution times. Our algorithm is performed on a PC with Pentium III 800 and 128MB. Note that our code is not fully optimized and more improvements can be done. All tasks are computed very fast. For this example, the user interaction time to specify the extreme vertices for patch decomposition is about 0.5-1 minute. Our results are very fast compared other recently reported timing information in [3,4]. Some experimental comparison is made in the end of this section.



(a) Input models are decomposed into four morphing patches.



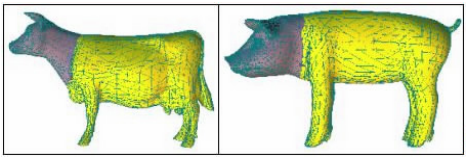
(b) A morphing sequence from a bay head to a man head.

Model	Vertices	Triangles	Embedding Time	Warping Time	Merging Time
Baby head	570	1136	<1 sec.	<1 sec.	<1 sec.
Man head	1954	3904	<1 sec.	<1 sec.	

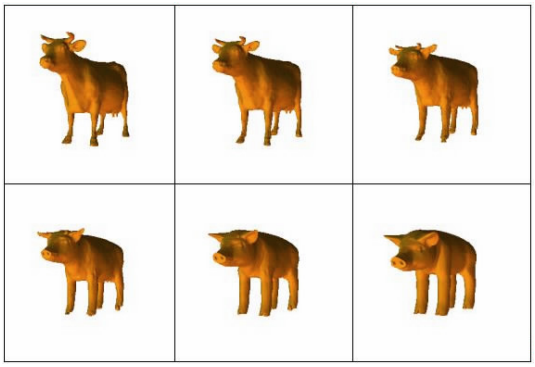
(c) Timing information for morphs between a baby head to a man head.

Figure 14. Morphs between the models of a baby head to a man head.

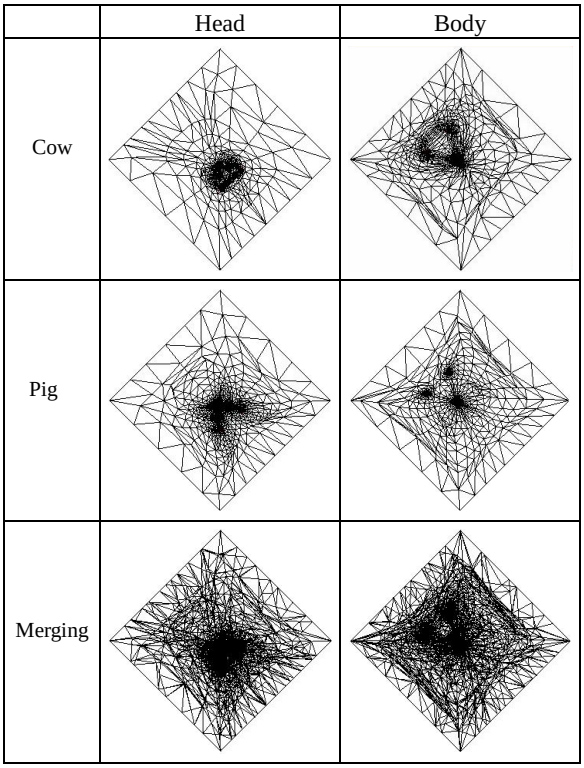
In the following, we also show other interesting examples in Figure 15 and 16.



(a) Both cow and pig are decomposed into two morphing patches. It takes about 15 seconds for users to specify extreme vertices for decomposing each model.



(b) A morphing sequence from a cow to a pig.

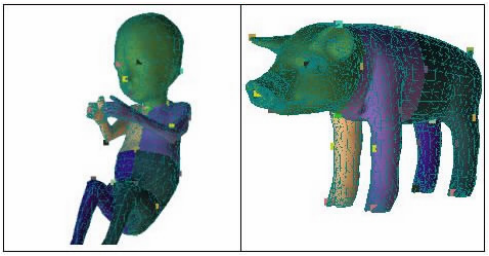


(c) Embedding results with warping. There are two embeddings (i.e., two morphing patches) per each model.

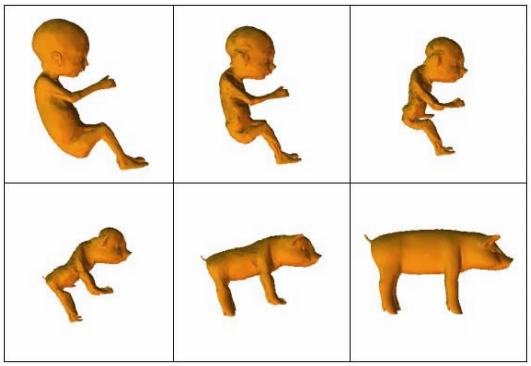
Model	Vertices	Triangles	Embedding Time	Warping Time	Merging Time
Cow	2903	5803	1.2 sec.	<1 sec.	<1 sec.
Pig	3584	7164	1.5 sec.	<1 sec.	

(d) Timing information about morphs between a cow and a pig.

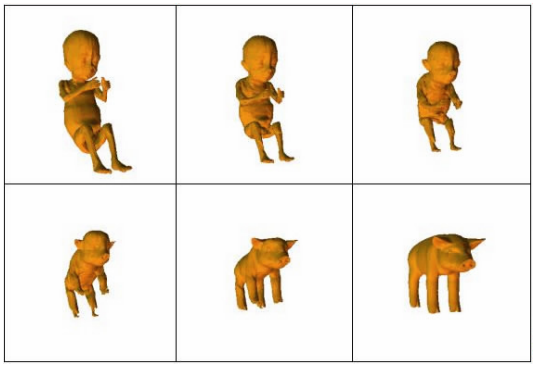
Figure 15. Morphs between the models of a cow and a pig.



(a) Both baby and pig are decomposed into five morphing patches. It takes about 80 seconds to specify extreme vertices for decomposing each model. Note that in this example, we explicitly show the extra features points used in warping.



(b) A morphing sequence (side-views) from a baby to a pig.



(c) The same morphing sequence rendered from another view.

Model	Vertices	Triangles	Embedding Time	Warping Time	Merging Time
Baby	2055	4106	<<1 sec.	<1 sec.	<<1 sec.
Pig	3584	7164	<<1 sec.	<1 sec.	

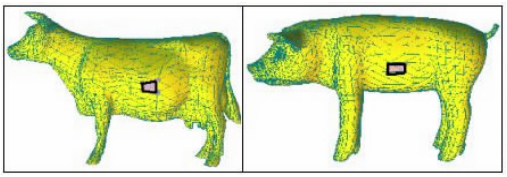
(d) Timing information for morphing from the baby to a pig.

Figure 16. Morphs between the models of the baby and a pig.

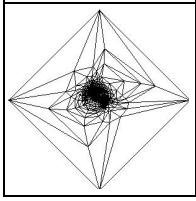
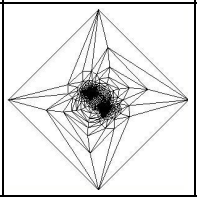
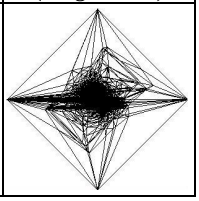
4.2 Performance Comparison with other Work

Recall that our work is closest in spirit to Gregory et al.'s [3] work and Alexa's work [4]. The former was evaluated on SGI Onyx 2 and the latter was on a SUN Ultra 10. We test our method on a PC with Pentium III 800 and 128MB. In [3], there is no detailed timing given for each task. Roughly, their algorithm takes about 1.5 minutes to compute merged embedding for models with number of triangles

ranging from 5000 to 8000. However, animators require several hours to decompose models into more than 50 morphing patches. It seems not to be very practical for daily use. Alexa's work [4] embeds each model on a 3D sphere and computes the merged embedding on this sphere. To experimentally compare this work with ours, we decompose the model into a large patch and a small patch by our algorithm, and evaluate the performance for the large patch only. We should note that the timing spent on a small patch is very small and can be ignored in this experiment. This evaluation is shown in Figure 17.



(a) Both models are decomposed into a large patch (i.e., yellow color) and a small patch (i.e., orange color).

Cow (Large Patch)	Pig (Large Patch)	Merged (Large Patch)
		

(b) Embedding and merged embedding results of the large patch.

Model	Vertices	Triangles	Embedding Time (Large Patch)	Merging Time (Large Patch)
Cow	2903	5803	1.8 sec.	1 sec.
Pig	3584	7164	1.9 sec.	

(c) Timing information for the embedding and merging

Figure 17 shows the experiment to decompose a model into a large patch and a small patch.

For performance comparison, we give two performance tables reported in [4] for several models.

	Giraffe	Horse	Pig	Shark	Swordfish
Giraffe		4.7 s	8.1 s	7.5 s	6.8 s
Horse	7152		4.2 s	3.6 s	3.2 s
Pig	12 509	6418		6.7 s	8.3 s
Shark	11 825	6585	11 540		6.8 s
Swordfish	11 365	5448	14 645	11 243	

Table 1. The performance of the merging algorithm proposed by [4]. The timing information is given in the upper triangle matrix. The number of vertices of the merged model is given in the lower triangle matrix.

Model	Vertices	Edges	Faces	Time in seconds
Cow	2911	8714	5805	126.5
Dolphin	420	974	556	12.3
Duck	629	1597	970	9.4
Giraffe	4197	9537	5342	427.5
Horse	674	1535	863	23.3
Pig	3522	7689	4169	277.9
Piglet	3584	7869	4287	113.1
Shuttle	310	701	393	9.4

Table 2. The performance of the embedding algorithm proposed by [4].

For our example in Figure 17, the sum of the vertices of the cow and the pig is 6487, and the sum of the triangles of the cow and the pig is 12967. Our example can be roughly corresponding to the Pig-Horse case (the sum of vertices is 6418) in Table 1. The merging algorithm proposed by [4] takes 4.2 second, but our merging algorithm only takes about 1 second. Furthermore, let us see the times of embedding in Table 2. For pig and piglet models, the embeddings take about 277.9 and 113.1 seconds, respectively. However, ours takes less than 2 seconds. In this respect, ours performs significantly better than [4]. Finally, we should also mention another work [5] that was evaluated on SGI Onyx 2 with R10000 processor. Unfortunately, [5] does not provide detailed timing information for each task. However, in [5] the user interaction time ranges from 5 minutes up to 15 minutes to decompose models. For the most complex example with 11464 triangles, the embedding can be computed in less than 5 seconds. The merging takes less than 3 seconds. In contrast, we do not require too much time in the user interaction time. As shown in Section 4.1, both embedding and merging are computed very quickly (in less than 1 or 2 seconds). Finally, we conclude that our performance is comparable to or even much better than the above state-of-the-art.

5. Conclusion and Future Work

We have presented techniques for computing shape transitions between polygonal 3D objects. These techniques have proven to be fast and intuitive methods for 3D polygon morphing. Our technique does not require too much user interaction time in contrast to other previous wok [3,5]. Furthermore, the proposed embedding and merging methods perform well to be comparable to or even much better than the-state-of-art. In the evaluated examples, both are computed very fast (less than 1 or 2 seconds). A number of researches can be done in near future. For example, we can replace the linear interpolation with other alternative to avoid self-intersection. The merged embedding usually has about 2 to 5 times as many triangles and vertices as the input models. We plan to design a

new method of 3D morphing without the requirement of the embedding merging. The morphing shapes can automatically adjust the number of triangles and vertices in need.

Acknowledgements

This project is supported by NSC-90-2213-E-006-085

Reference

1. F. Lazarus and A. Verroust, “Three-dimensional metamorphosis: a survey”, ***The Visual Computer***, 14(8-9):373-389, 1998.
2. J. Gomes, L. Darsa, B. Costa, and L. Velho, “***Warping and Morphing of Graphical Objects***,” Morgan Kaufmann, 1999.
3. A. Gregory, A. State, M. Lin, D. Manocha, and M. Livingston,” Interactive surface decomposition for polyhedra morphing,” ***The Visual Computer***, 15(9): 453-470, 1999.
4. Marc Alexa, “Merging polyhedral shapes with scattered features,” ***The Visual Computer***, 16(1):26–37, 2000.
5. Malte Zockler, Detlev Stalling, and Hans-Christian Hege, ”Fast and intuitive generation of geometric shape transitions,” ***The Visual Computer***, 16(5):241–253, 2000.
6. James R. Kent, Wayne E. Carlson, and Richard E.Parent,. “Shape transformation for polyhedral objects,” ***Computer Graphics (Proceedings of SIGGRAPH 92***),26(2):47–54, July 1992.
7. F. Lazarus and A. Verroust, “Metamorphosis of cylinder-like objects,” ***The Journal of Visualization and Computer Animation,*** 8(3):131-146, 1997.
8. R.E. Parent, “Shape transformation by boundary representation interpolation: A recursive approach to establish face correspondences,” ***The Journal of Visualization and Computer Animation***, Vol. 3, No. 4, 1992, pp. 219-239.
9. D. DeCarlo and J. Gallier, “Topological evolution of surfaces,” ***Proc. Graphics Interface***, May 1996, pp. 194-203.
10. Takashi Kanai, Hiromasa Suzuki and Fumihiko Kimura, “Metamorphosis of arbitrary triangular meshes,” ***IEEE Computer Graphics and Applications***, March/April Issue, 2000, pp. 62-75.
11. Aaron Lee, David Dobkin, Wim Sweldens, and Peter Schroder, “Multiresolution mesh morphing,” **Proceedings of SIGGRAPH 99**, pages 343–350, August 1999.
12. Aaron W. F. Lee, Wim Sweldens, Peter Schroder, Lawrence Cowsar, and David Dobkin, ”Maps: Multiresolution adaptive parameterization of surfaces,” **Proceedings of SIGGRAPH 98**, pages 95–104, July 1998.