Fast and Intuitive Metamorphosis of 3D Polyhedral Models Using SMCC Mesh Merging Scheme

Tong-Yee Lee and Po-Hua Huang

Abstract—A very fast and intuitive approach to generate the metamorphosis of two genus 0 3D polyhedral models is presented. 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 finer correspondence control. Once these two levels of correspondence are established, the proposed schemes automatically and efficiently establish a complete one-to-one correspondence between two models. We propose a novel technique called SMCC (Structures of Minimal Contour Coverage) to efficiently and robustly merge corresponding embeddings. The SMCC scheme can compute merging in linear time. The performance of the proposed methods is comparable to or better than state-of-the-art 3D polyhedral metamorphosis. We demonstrate several examples of aesthetically pleasing morphs, which can be created very quickly and intuitively.

Index Terms—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 into a target object. This research focuses on morphs between two genus 0 3D polyhedra. In the literature, most polyhedral morphing techniques consist of two main steps: The first involves finding the one-to-one correspondence between two polyhedral meshes. The second involves defining the 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 three excellent 3D morphing surveys in [1], [2], [3].

In this study, our design goal was to provide animators with easy morph control and fast morph creation. There are two control levels for animators to intuitively establish correspondence. First, the animators specify the scatter features to decompose the models into the corresponding patches, called morphing patches in this paper. This higher-level control provides the approximate correspondence between two models. If 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, we only need to identify a few corresponding morphing patches. Afterward, there are three techniques that automatically and efficiently establish a complete one-to-one correspondence between the 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 but simple **SMCC** (structures of minimal contour coverage) merging method is presented to create a merged embedding that contains the faces, edges, and vertices of two given embeddings. The SMCC is designed to speed up the merging process and the merging is always computed in less than a second. In addition, there are several lookup tables designed for easy and efficient implementation. The **SMCC** merging scheme is the major contribution proposed in this paper. With one simple data structure, the SMCC can handle well all degenerated cases. Programmers can easily implement embedding merging that is always error-prone.
- An efficient relaxation method is proposed to embed 3D morphing patches onto 2D regular polygons. In contrast to other relaxation methods, such as in [5], our method is computed within a few seconds (usually less than 1 second in our examples).

The rest of our paper is organized as follows: Section 2 reviews the related work on 3D polyhedral metamorphosis. The proposed techniques are presented in Section 3. The proposed schemes are evaluated and experimentally compared with other work in Section 4. The conclusion and future work are presented in Section 5.

2 Related Work

Lazarus and Verroust [1] provide an excellent survey of the previous work on the 3D morphing problem. There are two

- *The authors are with the Computer Graphics Group/Visual System Laboratory, Department of Computer Science and Information Engineering, National Cheng Kung University, Tainan, Taiwan, ROC. E-mail: tonylee@mail.ncku.edu.tw, bohaw@vision.csie.ncku.edu.tw.*

Manuscript received 9 Oct. 2001; revised 12 Feb. 2002; accepted 10 May 2002. For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number 115162.

major classes of 3D morphing techniques: a volume-based approach and a surface-based approach. In this paper, we focus on the surface-based approach. For more work related to the volume-based approach, please refer to [1]. The surface-based approach works on boundary representations such as polyhedral meshes or patch complexes. Methods involving this approach usually create an interpolation mesh, which is a common embedding method for both the 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 the target mesh. This is a key issue in the surface-based approach.

A lot of work has been published on the correspondence issue. Kent et al. [7] introduce parameterizations for solving the correspondence problem. Their approaches project star-shaped objects onto spheres to accomplish parameterization. Similarly, Alexa [5] employs a relaxation method to embed polyhedral shapes onto spheres. Lazarus and Verroust [8] introduce skeletons for cylinder-like objects. This approach is an extension of [7] for objects that are star-shaped around an axis. Parent [9] presents a recursive algorithm that automatically finds a correspondence between the surfaces of two objects with equivalent topologies. Decarlo and Gallier [10] present a method to transform objects with different topologies. The user must identify a sparse control mesh on each object surface. This control mesh specifies how to transform one surface into another. It is therefore very complicated and difficult to transform complex shapes. Kanai et al. [11] and Zöckler et al. [6] utilize mesh parameterization techniques such as harmonic mapping to embed a mesh region into a 2D convex polygon. Gregory et al. [4] apply a user-specified control mesh to decompose the surface into a large number of disk-like patches and used a greedy area-preserve heuristic to compute 3D-to-2D embeddings. Lee et al. [12] employ the MAPS algorithm [13] to parameterize input meshes over simple base domains and an additional harmonic map bringing the latter into the correspondence. Their approach could have a fold-over problem and user interaction is required to manually fix this problem. Recently, Praun et al. [15] presented a versatile technique to achieve consistent mesh parameterizations among given models. This technique can be applied to polyhedral morphing and avoid model merging to create a common embedding. However, this technique requires remeshing to fit the original models, starting from a common based model. This remeshing is the bottleneck in the computational performance. In their experiments, this remeshing took about six minutes. If interactive control is an important issue for animators, this technique requires more advances to improve its fitting speed.

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 [4] and Alexa's work [5]. Our overall system structure is similar to their theme. Generally, the overall system structure of most surface-based morphing techniques is similar in spirit. For example, the works of Zöckler et al. [6], Bao and Peng [16] are similar to [4] and [5], too. But, new techniques are presented in [6] and [16]. In this paper, we present novel techniques in our design. The main procedures are listed below:

- **Selection of Vertex Pairs and Decomposition into Morphing Patches:** For a given two 3D polyhedral models, the animators select the corresponding vertices on each polyhedron to define the correspondence 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 corresponds to the high-level morph control in our design.
- **3D-to-2D Embedding:** Each 3D morphing patch is mapped onto a 2D regular polygon using the proposed relaxation method.
- **Aligning Feature Vertices:** The interior vertices in the regular 2D polygons are matched using a foldover-free warping technique. Users can specify extra feature vertices to have better correspondence control. This design corresponds to the lower-level morph control.
- **Merging, Remeshing, and Interpolation:** The algorithm merges the topological connectivity of the morphing patches into a regular 2D polygon. Additional retriangulated edges are inserted into the regions in the merged regular 2D polygon. This step reconstructs the facets for the new morphing patch, i.e., a common interpolation mesh. Exact interpolation meshes across the common interpolation meshes are then computed.

3.2 Specifying Corresponding Morphing Patches

Given two polyhedral models, 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 also specify the same number of vertices (i.e., called ***extreme vertices*** [4]). These selected vertices 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. Our shortest path was computed using Dijkstra's shortest path algorithm, but with some restrictions. Animators can partition two polyhedral models into an arbitrary number of morphing patches, but each patch cannot cross any other patches. Praun et al. [15] proposed a nice method for computing the shortest path method. This method can avoid path boundary crossing and produce smooth boundaries. For simplicity, we partition a given model into patches, patch by patch. When finding a shortest path, we will check if any edge will cross the boundaries or is contained in other existing patches. If this is true, this edge cannot be used to form the shortest path. Once the models are partitioned into several corresponding morphing

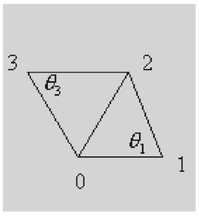


Fig. 1. The definition of a pulling weight.

patches, the next step is to compute the corresponding interior vertices of (C_i^A, C_i^B).

3.3 Embedding 3D Morphing Patches on Regular 2D Polygons

In the following, the basic idea behind the proposed relaxation method for computing 3D-to-2D embeddings will be described. This initial approach requires several iterations. It can be computationally expensive. Next, a sparse linear system for our relaxation method is proposed. 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, each is embedded on an n-side regular 2D polygon called D_i (i.e., D_i^A, D_i^B) using 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 in the morphing patch is mapped to an edge of D_i. The 2D coordinates of the nonextreme vertices along each chain must then be found. The 2D coordinates of these nonextreme vertices are interpolated based on the arc length of the chain. Third, a 2D mapping for the interior vertices of C_i^A and C_i^B is computed by initially mapping them to the center position (0, 0). These vertices are then moved step by step using the following relaxation equation. This process will continue until all of the interior points are stable, i.e., not moved.

$$p_i' = (1-\lambda)p_i + \lambda \frac{\sum_{j=1}^{k_i}(\omega_j p_j)}{\sum_{j=1}^{k_i}\omega_j}. \tag{1}$$

In (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 on a morphing patch.
- p_i' is the new position of p_i according to (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 λ controls the movement speed and its value is between 0 and 1.

A good embedding that preserves the aspect ratio of the original triangle versus the mapped triangle and does not cause too much distortion must be computed. To determine w_j, our idea is similar to Kanai et al.'s [11] weight formula used in their harmonic mapping. However, we used a different and a simpler formula. For example, in Fig. 1, p_i is labeled 0 and the weight of a p_j labeled by 2 is computed using the following equation:

$$w_2 = \cot\theta_1 + \cot\theta_3. \tag{2}$$

In (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 angles correspond to the 3D edges of a morphing patch. In this manner, all w_j can be computed. In (2), the whole system is depicted as a spring system. During iterations, p_i is pulled by several springs that are connected to all of its neighbors p_j. The idea behind (2) is that the long edges subtended to big angles are given relatively small spring constants compared to short edges that are subtended to small angles. Based on (1), iteration methods can be used to find all p_i. The iteration is terminated when all p_i are stable. However, in this manner, the computation time is not predictable and could be expensive. Therefore, p_i will not be found using an iteration method. It will be solved in the following manner:

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

$$p_i' = p_i = (1-\lambda)p_i + \lambda \frac{\sum_{j=1}^{k_i}(\omega_j p_j)}{\sum_{j=1}^{k_i}\omega_j} => p_i = \frac{\sum_{j=1}^{k_i}(\omega_j p_j)}{\sum_{j=1}^{k_i}\omega_j}. \tag{3}$$

Therefore, assume that the number of p_i is N. The following linear system can then be used for the proposed relaxation method.

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

Let $\chi_i = \sum_{j=1}^{k_i}\omega_j$ and $i = 1..N$. The above linear system can be represented in the following form:

$$\begin{aligned} \sum_{j=1}^{k_1}(\omega_j p_j) &= \chi_1 p_1 \\ \sum_{j=1}^{k_2}(\omega_j p_j) &= \chi_2 p_2 \\ &\vdots \\ \sum_{j=1}^{k_N}(\omega_j p_j) &= \chi_N p_N. \end{aligned} \tag{5}$$

This linear system is not singular, so that it has a unique solution. Furthermore, for each p_i, the number of neighbors is small compared to N. Therefore, it is a sparse system that can be solved efficiently using a numerical method. For example, a biconjugate gradient method [17] can be used to solve this sparse system with a computational complexity of approximately $O(N)$. The computational complexity of our

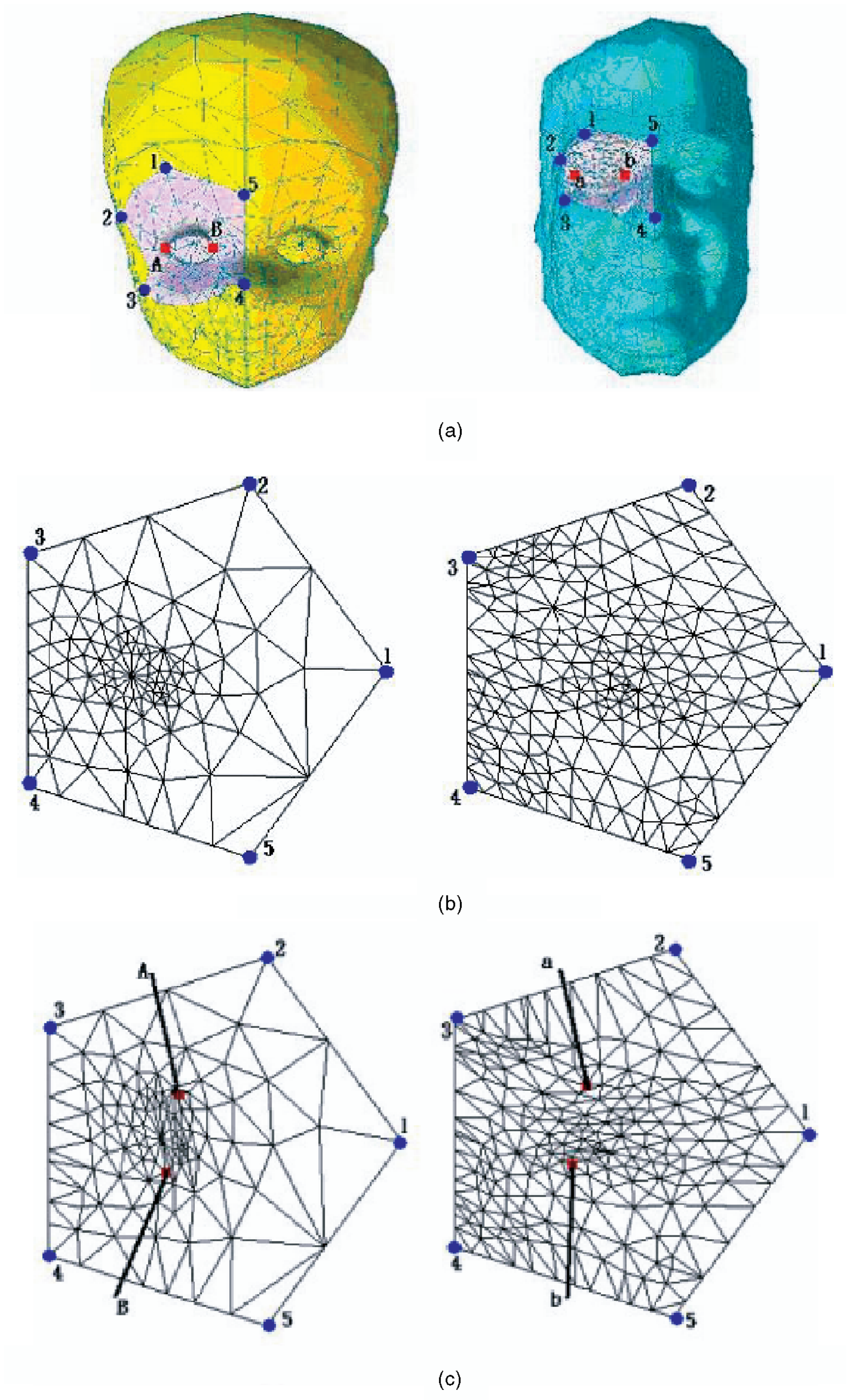


Fig. 2. Embedding and warping. (a) The user picks five extreme vertices (i.e., blue dots) and two extra feature vertices (i.e., red dots). (b) Embeddings without warping. (c) Embedding with warping using two extra features (i.e., red dots).

embedding scheme is comparable to [6], [11]. All three schemes require solving a linear sparse system.

For example, in Fig. 2, the embedding results do not have much area compression, although some still occurs. Kanai et al. [11] use harmonic maps. They minimize the metric distortion, preserve the aspect ratios of the triangle, and introduce much area compression [14]. In [4], Gregory et al. propose an area preserving heuristic to embed 3D polyhedra into 2D n-gon disks. This greedy heuristic works well and improves over the harmonic map scheme. However, the greater the number of polygons in the morphing patch, the less successful this area preserving heuristic will be [18].

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 the user specifications. Using this initial correspondence, we could directly overlay two embeddings to get a merged embedding for morphing. For example, in Fig. 2a, a corresponding morphing patch is selected 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 Fig. 2b, we show both (D_i^A, D_i^B) after embedding. It is obvious that the vertex pairs (A, B) and (a, b) will not align if we directly overlay (D_i^A, D_i^B). Therefore, to compute better correspondence, the animators are required to specify several extra corresponding features such as vertex pairs (A, B) and (a, b) on both (D_i^A, D_i^B). Then, a foldover-free warping function is employed to align (A, B) and (a, b). The nonfeature points will automatically be moved by the warping function. As in [6], to minimize the distortion due to warping, these extra corresponding feature points are moved linearly to a point halfway between them and then warping is performed.

Our warping is simply computed as a weighted radial basis function (RBF) sum. Suppose there are n extra feature pairs. Because (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:

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

In (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, its new position is computed by $(R_1(p), R_2(p))$ using (6). In total, there are $2n$ coefficients to compute. In the current implementation, the radial basis function is a Gaussian function:

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

In (7), the variance σ controls the degree of locality of the transformation. In Fig. 2c, (D_i^A, D_i^B) are shown warped using two extra feature points. This result is better than that in Fig. 2b. They can be overlaid now to produce a merged embedding for morphing. Sometimes, the warping can lead to a fold-over (self-intersections) on (D_i^A, D_i^B). Foldover-free embeddings are necessary. To solve the foldover problem, a check if self-intersections have occurred on (D_i^A, D_i^B) is made after warping. If self-intersections have occurred, (1) is iterated instead of solving (5). Usually, a few iterations are required to prevent self-intersections from occurring. In the following, we show how to check if self-intersections have occurred.

The inputs were genus 0 3D polyhedral models with a 1-ring structure. 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, self-intersection has occurred. To check if a point has a complete 1-ring structure, the following is computed:

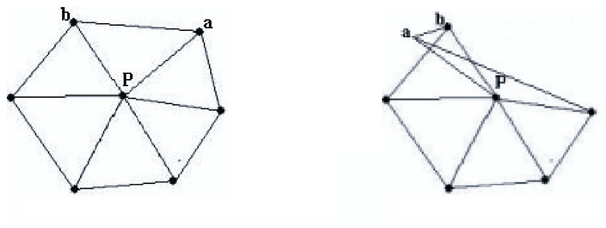


Fig. 3. (a) prior to embedding and warping, P has a complete 1-ring structure in 3D and (b) P has an incomplete 1-ring structure in 2D after embedding and warping.

$$Z = \vec{pa} \otimes \vec{pb} \qquad (\otimes: \text{ the righthand vector cross product})$$
$$\begin{cases} complete, & Z > 0 \\ incomplete, & Z <= 0. \end{cases} \tag{8}$$

In (8), all nodes at p's 1-ring structure are checked. If any violation (i.e., $Z <= 0$) has occurred, an incomplete ring structure in 2D will be found. Note that the vertices of a triangle are in counterclockwise order. Fig. 3 is used to illustrate (8). In Fig. 3a, 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 Fig. 3b. In this case, all nodes at p's 1-ring structure are checked, finding that (a, b) violates (8).

In the above procedure, a linear system for RBF is solved. However, since only a few features were specified compared to the number of interior vertices, the time complexity can almost be ignored. Assume that we have n interior vertices. $O(n)$ time is required to execute (6) (i.e., warping) and (8) (i.e., self-intersection check). The alignment problem using radial basis functions for the scattered data interpolation problem can also be found in [19], [20], [21]. Recently, the very same problem also appears in texture mapping with constraints and has received attention lately [22].

3.5 Efficient Local Merging Using SMCC

Two embeddings (D_i^A, D_i^B) are merged to produce a common embedding that contains the faces, edges, and vertices. The complexity of a brute-force merging algorithm is $O(n^2)$, where n is the number of edges. This naive approach globally checks all edges to find the possible intersections. Motivated by simplicity and robustness, Gregory et al. [4] adopt this approach to check all edge pairs for merging. In this paper, we present a novel SMCC method for checking the edges locally and efficiently computing the intersections. The complexity of the proposed method is $O(n + k)$, where k is the number of intersections. Additionally, a lookup table was created to efficiently implement this method. The merging problem is known as planar graph overlay in computational geometry and is solved in $O(n + k)$ [23], [24]. However, embedding merging is very erroror-pone in implementation. Our SMCC approach handles all degenerate cases with one simple data structure. It is easy to use SMCC to perform merging. We should also mention that Alexa [5] proposes an $O(n + k)$ merging algorithm to overlay two embeddings onto 3D spheres.

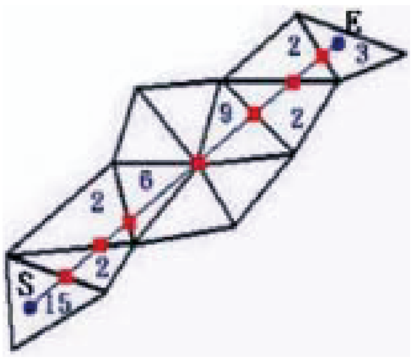


Fig. 4. Each intersection classification is labeled according to Fig. 6.

3.5.1 Classifying of the Corresponding Positions

The merging algorithm wants to overlay each edge $\overline{se} \in D_i^B$ on D_i^A, where s and e represent the starting and ending points of a given edge. There is a special type of edge $\overline{se}$ and its s point corresponds to an extreme vertex of D_i^A. This correspondence was established before embedding by the animators. Edge-merging is performed starting from one of these types of edges. Since D_i^B is a connected planar graph, we can traverse all of its remaining edges starting from any $\overline{se}$ belonging to this special type and overlay all remaining edges $\overline{se} \in D_i^B$ on D_i^A edge by edge.

When an edge $\overline{se} \in D_i^B$ is overlaid on D_i^A, this edge can be split into several line segments by triangles $T^A \in D_i^A$. An example is shown in Fig. 4. In this example, the red-dots represent the intersections created by merging. The position of s on a triangle $T^A \in D_i^A$ can be three possibilities: s is on the edge (i.e., E^A) of T^A (the first row in Fig. 5) or s is on the vertex (i.e., V^A) of T^A (the second row in Fig. 5) or s is inside T^A (the last row in Fig. 5). If e is outside T^A, the edge $\overline{se}$ is split and the new intersection point becomes a new s. This process will continue until we find a T^A, where e is on the edge of T^A or e is on the vertex of T^A or e is inside T^A. Fig. 5 shows our classifications based on the positions of s and e with respect to a given T^A. The last row cases occur only when s is the original starting point of an edge $\overline{se} \in D_i^B$ (i.e., not yet split) to be overlaid on D_i^A.

3.5.2 Structures of Minimal Contour Coverage (SMCC)

In Fig. 5, the leftmost column shows three possible positions for s on a given T^A. Based on these three possibilities, three kinds of SMCC are defined for s on a given T^A in the following (as shown in Fig. 6):

1. If s falls on a vertex $P^A \in D_i^A$, its SMCC is P^A's 1-ring structure on D_i^A.
2. If s falls on an edge $E^A \in D_i^A$, its SMCC is a 4-sided polygon containing E^A on D_i^A.
3. If s falls inside a triangle $T^A \in D_i^A$, its SMCC is the triangle T^A.

During merging (i.e., edge splitting), if $\overline{se}$ intersects with some edge $E^A \in D_i^A$ outside s's SMCC, the edge $\overline{se}$ must intersect with s's SMCC (as shown in Fig. 7). The merging (intersection) can be locally computed with s's SMCC.

In Fig. 6, we show that each s is enclosed by its SMCC. In case (1), all contour vertices of SMCC have edges to s. In case (3), there is no edge between s and each SMCC contour vertex. To easily compute the intersections, we assumed that there is an imaginary line from s to each contour vertex of its SMCC. With this arrangement, the intersection computation can be classified under two conditions. The first is called the area condition at which $\overline{se}$ is not coincident with any imaginary line. If $\overline{se}$ is coincident with any imaginary line, this is called the line condition (as shown in Fig. 8). The intersection computation can then be evaluated using the following:

$$M = \vec{se} \otimes \vec{sa}, N = \vec{se} \otimes \vec{sb}, \text{ where } \vec{se}, \vec{sa}, \text{ and } \vec{sb} \in R^2$$
$$\begin{cases} \textit{Area Condition}, & M < 0 \textit{ and } N > 0 \\ \textit{Line Condition at } \overline{sa}, & M = 0 \\ \textit{Line Condition at } \overline{sb}, & N = 0 \\ \textit{Continue searching for other } (\overline{sa}, \overline{sb}) \textit{ pairs}, & \textit{other}. \end{cases} \tag{9}$$

Based on the above classifications, different formulas (i.e., (10) and (11)) can be used to determine whether $\overline{se}$ intersects with S's SMCC or not. The results from (10) and (11) lead to different conditions, as shown in Fig. 9.

1. Area Condition:

   Let $V = s + t^* \vec{se}$ and suppose V is on the $\overline{ab}$, therefore, $\vec{aV} \otimes \vec{ab} = 0$. Then,

   $$\vec{aV} = s + t^* \vec{se} - a = \vec{as} + t^* \vec{se}.$$

   Therefore, $\vec{aV} \otimes \vec{ab} = (\vec{as} + t^* \vec{se}) \otimes \vec{ab} = 0$ and we let

   $$Z_1 = \vec{as} \otimes \vec{ab} \quad Z_2 = \vec{se} \otimes \vec{ab} \quad t = Z_1/Z_2. \tag{10}$$

2. Line Condition (aasume coincident with $\overline{sa}$):

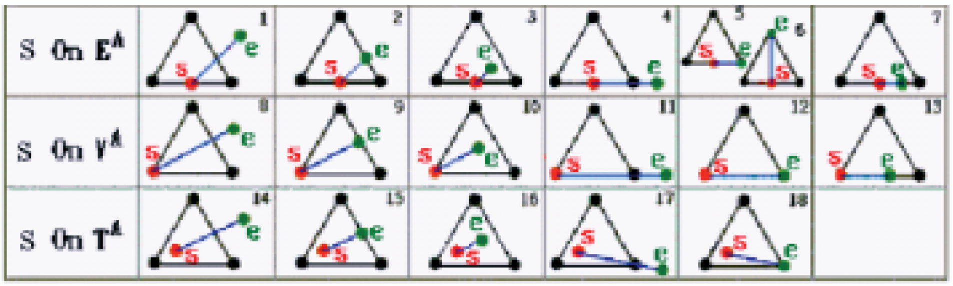


Fig. 5. The classifications based on the positions of s and e with respect to a given T^A.

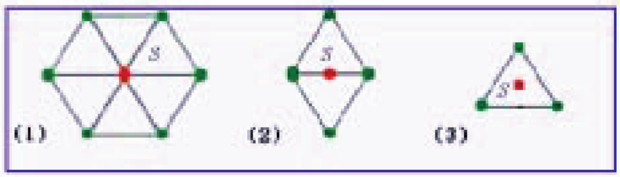


Fig. 6. There are three kinds of SMCC for s (red-dot) on D_i^A: (1) 1-ring, (2) 4-sided polygon, and (3) a triangle.

$$L1 = \left\|\vec{se}\right\| \qquad L2 = \left\|\vec{sa}\right\| \qquad t = L1/L2. \tag{11}$$

Based on the analysis of Fig. 9b, we can replace Fig. 5 with Fig. 9b. Fig. 9b can provide us a lookup table to efficiently implement our merging algorithm.

The merging algorithm must establish the SMCC structure prior to proceeding with the local merging. Fig. 6 shows how an SMCC structure is defined. In our design, once a new intersection occurs, the SMCC can be immediately established for further potential merging. For example, in Fig. 10, an edge $\overline{AB} \in D_i^B$ and A's SMCC was created using Fig. 6. Using Fig. 9b, we find that $\overline{AB}$ intersects with A's SMCC at C. C's SMCC is created next using Fig. 6 again. Again, with the help of Fig. 9b, the edge $\overline{CB}$ intersects with C's SMCC at D. In this manner, the above steps are repeated until B is reached. B's SMCC does not need to be created if B is the starting point of the other edge (i.e., not yet overlaid) from D_i^B. During merging, the s point is changed from A, C, D, E to F and the e point is always B. As discussed in Section 3.5.1, the local merging from an extreme vertex is the starting point. The SMCC of an extreme vertex is its 1-ring structure and this is known in advance. The subsequent SMCC for merging can therefore be found immediately using Fig. 6. Given two embeddings (D_i^A, D_i^B) in Fig. 11, a complete sequence of overlays D_i^B on D_i^A is shown using the proposed method.

3.5.3 *Analysis*

Assume that there are two embeddings (D_i^A, D_i^B) with m and n edges. n edges of D_i^B must be overlaid onto D_i^A and k intersections are created after the overlay. To travel all edges of D_i^B starting at edge $\overline{se}$ with s corresponding to an extreme vertex of D_i^A, $O(n)$ time is required. The merge (intersection) time is in proportion to the number of SMCCs used to locally compute the intersections. For all n edges of

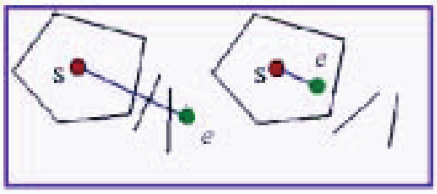


Fig. 7. The merging (intersection) can be locally computed with *s*'s SMCC.

D_i^B, n times are required to use SMCC for the s points. If any edge $\overline{se}$ requires splitting (i.e., the e point is outside s's SMCC), the intersection point will become a new s point and this s must use SMCC again for further intersection. If the total number of intersections is k, the number of times SMCC is used to locally compute the intersections is bounded by k. Our merging cost is therefore $O(n + k)$ time.

3.6 Retriangulate the Merged Embeddings

Once the merging is completed, a nontriangulated planar graph is called D^M. In order to retriangulate D^M is produced, additional edges must be inserted to retriangulate D^M. For simplicity, our approach is very straightforward and described as follows: For each point P^M on D^M, the algorithm must connect the neighboring points of P^M to establish the 1-ring cyclic structure. This structure can easily be constructed by finding cycles with connectedness using the smallest interior angles, as in [4] (as shown in Fig. 12).

3.7 Reconstructing the Source Models and Interpolation

Once the preceding steps are completed, a complete correspondence between the two models is established. Our merging algorithm produces new points in 2D due to the intersections. The corresponding 3D points for these new points must be found in both models. The barycentric representation of a new point is computed on the basis of three old points in 2D. The barycentric representation is then used to interpolate the positions of these three old points in 3D. These old points are referred to as the original vertices on the input models. In this manner, the 3D position of a new point is determined. Similarly, for a new point, its other attributes, such as the color and texture coordinates, can be interpolated if required.

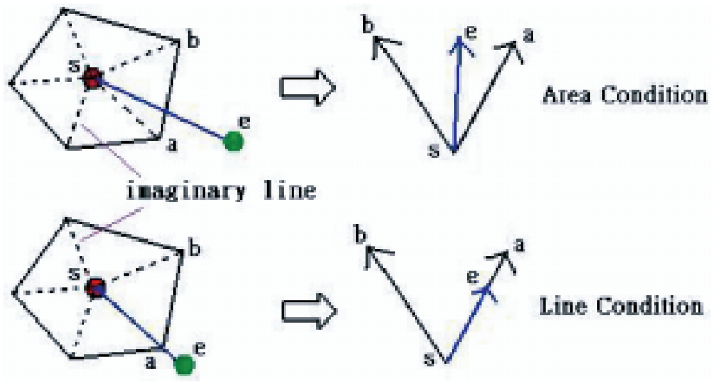


Fig. 8. There are two kinds of the local intersection computations.

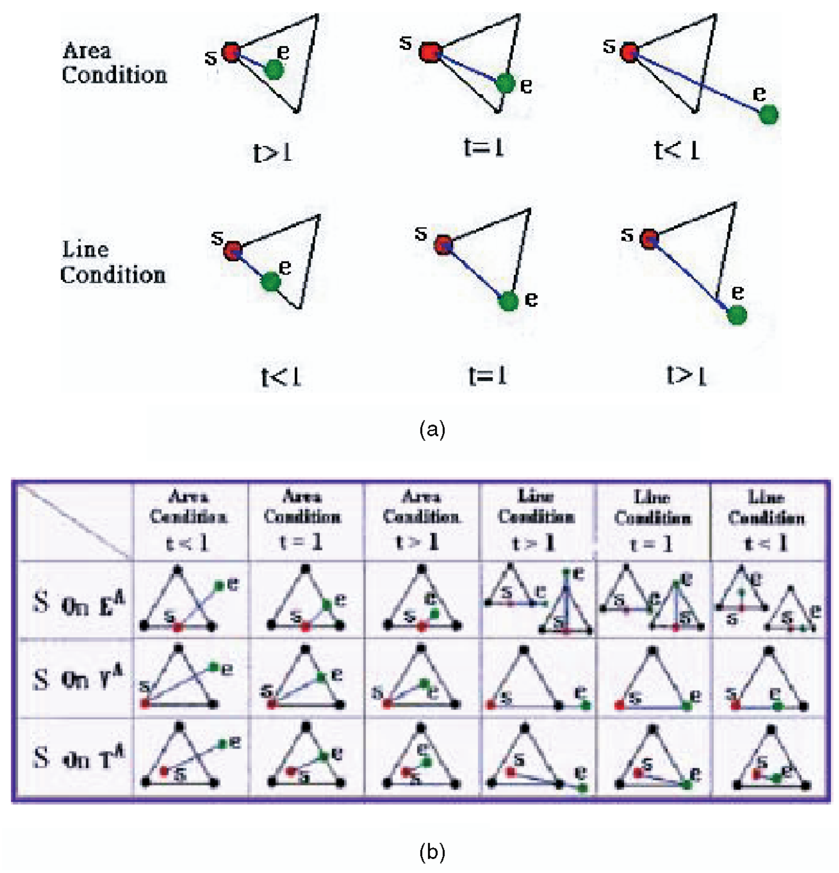


Fig. 9. Local merging condition classification conditions. (a) The results from (10) and (11). (b) Local merging condition table. This table corresponds to different kinds of SMCC defined in Fig. 6 (the first row: a 4-sided polygon, the second row: a 1-ring structure, and the third row: a triangle).

Once the above step is finished, the morphing sequence can easily be generated by linearly moving each vertex from its position in model A to the corresponding position in model B in terms of the time t. Other authors have mentioned that this kind of linear interpolation produces satisfying results in most cases. However, in some special cases, self-intersections can occur. Gregory et al. [4] proposed a user-specified morphing trajectory using cubic spline curves as an alternative to linear interpolation. This simple alternative can be included in the 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. The friendly GUIs in our system are also demonstrated on this Web site.

4.1 Performance Evaluation and Morphing Examples

In Fig. 13, an example of morphing a baby's head into a man's head is shown. In this example, the number of morphing patches is four and each patch is illustrated using a different color. There are seven extra feature points (i.e., four eye corners, one nose tip, two mouth corners) used for warping. Fig. 13c shows some timing information and the geometric information for two models in this experiment. The times are average execution times. Our algorithm was performed on a PC with a Pentium III 800 and 128MB. Note that our code was not fully optimized and more improvements can be done. All tasks were computed very fast compared with other recently reported timing information in [4], [5]. Some experimental comparisons are made at the

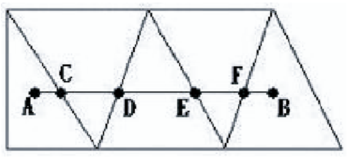


Fig. 10. The merging of $\overline{AB}$ can be completed step by step.

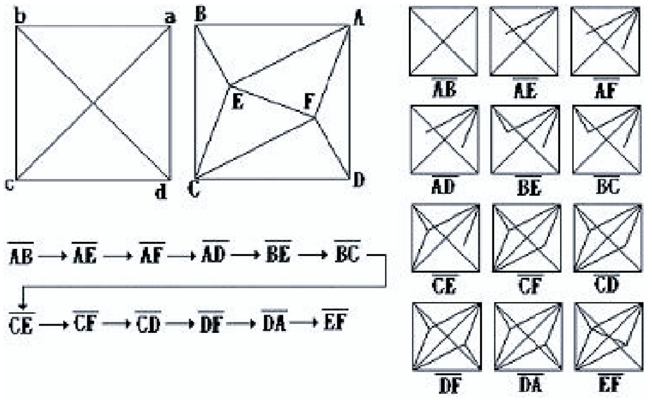


Fig. 11. An example of merging is completed using our algorithm step by step.

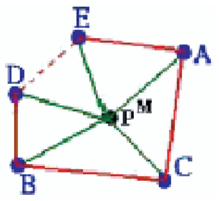


Fig. 12. P^M's 1-ring cyclic structure established by inserting several edges.

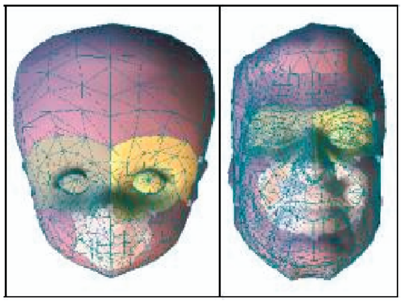
end of this section. Other interesting examples are shown in Figs. 14a-b, Figs. 14c-d, and 15.

4.2 Comparison with Other Related Work

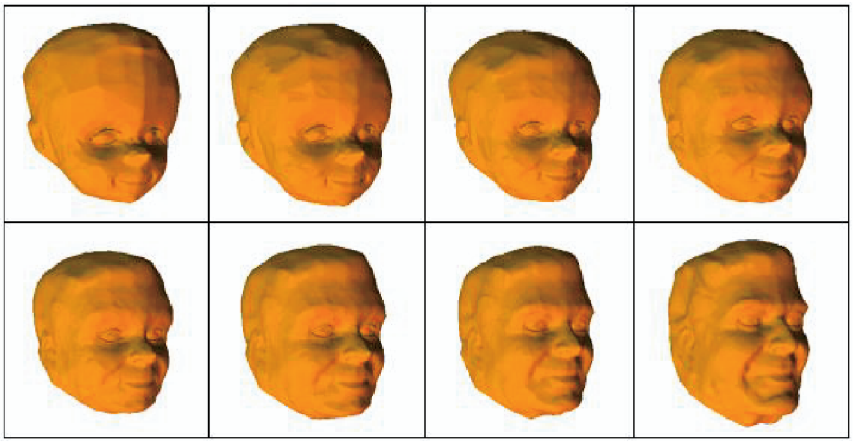
Recall that our work is closest in spirit to Gregory et al.'s [4] work and Alexa's work [5]. The former was evaluated on an SGI Onyx 2 and the latter on a SUN Ultra 10. We tested our method on a PC with a Pentium III 800 and 128MB. In [4], there is no detailed timing given for each task. Roughly, their algorithm takes about 1.5 minutes to compute merged embedding for models with a number of triangles ranging from 5,000 to 8,000. Alexa's work [5] embeds each model on a 3D sphere and computes the merged embedding on this sphere. To experimentally compare this work with ours, we decomposed the model into a large patch and a small patch using our algorithm and evaluated the performance for the large patch only. Note that the timing spent on a small patch was very small and can be ignored in this experiment. This evaluation is shown in Fig. 16.

Two performance tables reported in [5] for several models were used for the performance comparison. See Tables 1 and 2.

In Fig. 16, the sum of the vertices for the cow and the pig was 6,487. The number of triangles for the cow and the pig was 12,967. Our example roughly corresponds to the Pig-



(a)

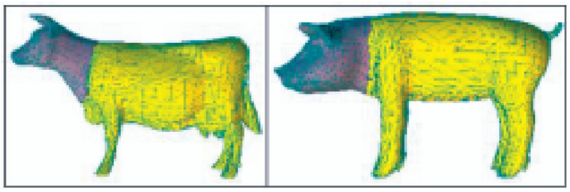


(b)

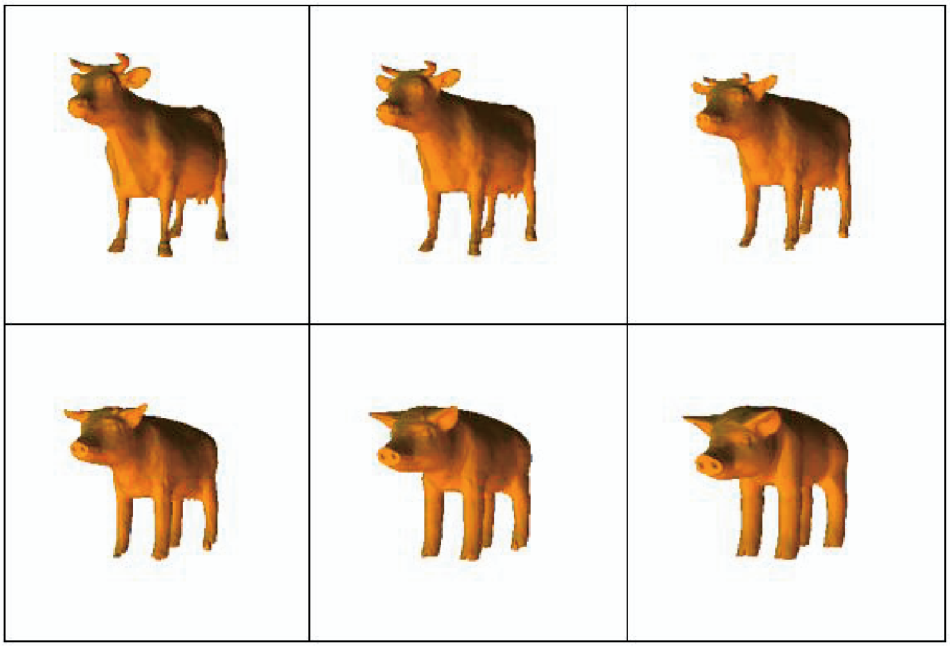
Model	Vertices	Triangles	Embedding Time	Warping Time	Merging Time
Baby head	570	1136	<1 sec.	<1 sec.	<1 sec.
Man head	1954	3904	<1 sec.	<1 sec.	

(c)

Fig. 13. Morphs between the models of a baby's head into a man's head. After merging, both models have 9,154 vertices and 20,127 triangles. (a) The input models are decomposed into four morphing patches. (b) A morphing sequence from a baby's head into a man's head. (c) Timing information for morphs between a baby's head into a man's head.



(a)



(b)

Fig. 14. Morphs between the models of a cow and a pig. After merging, both models have 25,816 vertices and 57,844 triangles. (a) Both a cow and pig were decomposed into two morphing patches. (b) A morphing sequence from a cow into a pig.

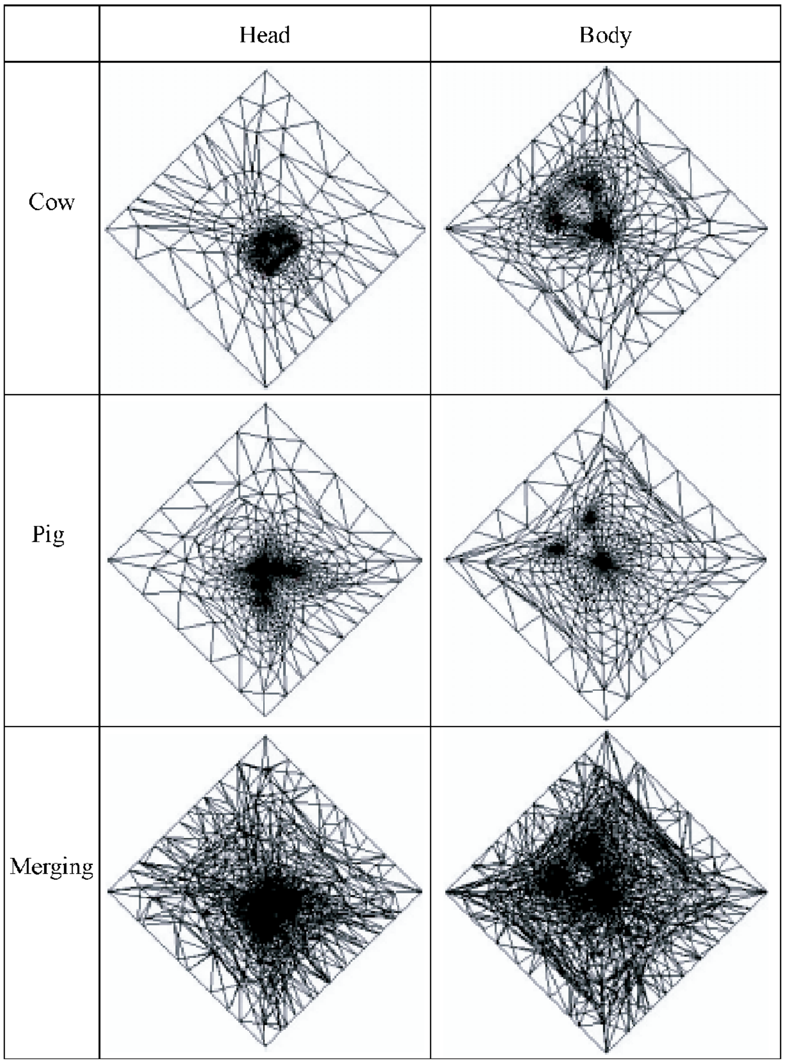
Horse case (the sum of vertices is 6,418) in Table 1. The merging algorithm proposed by [5] takes 4.2 seconds, while our merging algorithm only takes about 1 second. The embedding time is shown in Table 2. For the pig and piglet models, the embedding required about 277.9 and 113.1 seconds, respectively. However, our method required less than 2 seconds. In this respect, our method performed significantly better than [5]. Another work [6] was evaluated on an SGI Onyx 2 with R10000 processor. Unfortunately, [6] does not provide detailed timing information for each task. For the most complex example with 11,464 triangles, the embedding could be computed in less than 5 seconds. The merging took less than 3 seconds. As shown in Section 4.1, both embedding and merging were computed very quickly (in less than 1 or 2 seconds). The performance of our method was comparable to [6] or even much better than [4], [5], the state-of-the-art.

Finally, we comment on these related works in term of computational complexity. For mesh embedding, the proposed method and work in [6] require solving a linear sparse system. A biconjugate gradient method [17] can be used to solve this sparse system with a computational complexity of approximately $O(N)$. Alexa [5] embeds polyhedral on the unit sphere that implies nonlinear constraints, while the embedding in the plane such as our method and [6] is a linear problem. Therefore, Alexa [5] introduces a relaxation method for embedding. This method applies an iterative approach to find the solution. Generally, an iterative approach is time consuming in execution and sometimes it is not predictable if the termination condition is not reached quickly. For example, in Table 2, both Pig and Piglet models are similar in their geometrical complexity. However, their embedding times are significantly different.

For embedding merging, our work and Alexa's work [5] are computed in $O(n + k)$. Gregory et al. [4] use an $O(n^2)$ approach. Zöckler et al. [6] do not detail their merging scheme and mention that their merging method is suggested by [11], [25]. The merging in [11], [25] can be performed in $O(n \log n)$ with a spatial data structure such as quad-tree.

5 Conclusion and Future Work

We presented techniques for computing shape transitions between polygonal 3D objects. These techniques were proven to be fast and intuitive methods for 3D polygon morphing. The proposed embedding and merging methods performed well and are comparable to [6] or better than the-state-of-the-art [4], [5]. Both evaluated examples were computed very fast (less than 1 or 2 seconds). There are several opportunities for further work expanding on the method we propose. For example, linear interpolation can be replaced with other alternatives to avoid self-intersection. The merged embedding usually has about 3 to 8 times



(c)

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.	

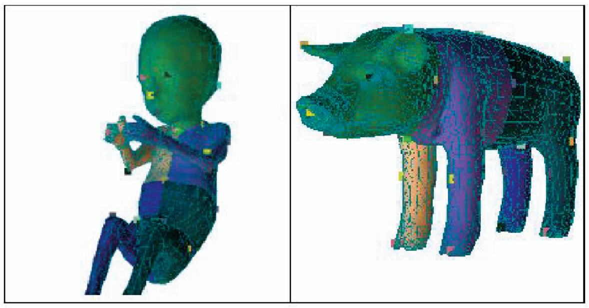
(d)

Fig. 14 (continued). (c) Embedding results with warping. There are two embeddings (i.e., two morphing patches) per each model. (d) Timing information about morphs between a cow and a pig.

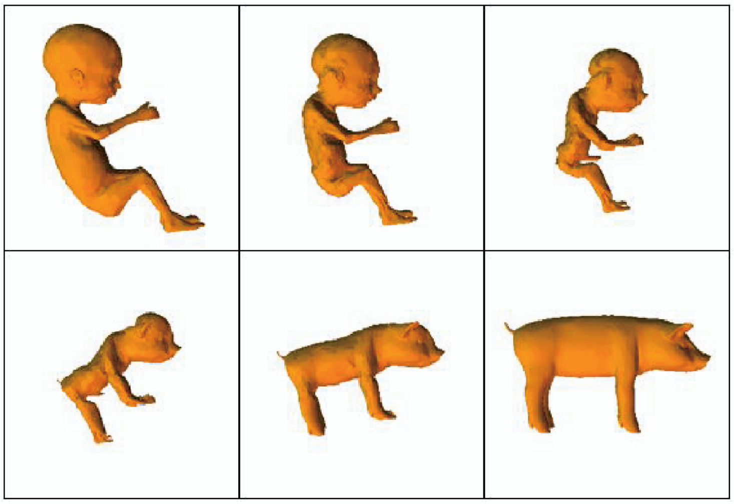
as many triangles and vertices as the input models. We plan to design a new 3D morphing method that does not require embedding merging. The morphed shapes in this extended method will be able to automatically adjust the required number of triangles and vertices. In addition, the retriangulation of the merged embeddings should better take the original geometries into account to avoid possibly unsatisfactory results.

Acknowledgments

The authors extend their thanks to A. Gregory for his explanation of the work in [4] and also to the anonymous reviewers for their helpful comments. This paper is supported by the National Science Council, Taiwan, Republic of China, under contract Nos. NSC-90-2213-E-006-085 and NSC-91-2213-E-006-078.



(a)



(b)

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.	

(c)

Fig. 15. Morphs between the models of the baby and the pig. After merging, both models have 20,458 vertices and 40,883 triangles. (a) Both the baby and pig were decomposed into five morphing patches. Note that, in this example, the extra features points used in warping are shown. (b) A morphing sequence from a baby into a pig. (c) Timing information for morphing from the baby into a pig.

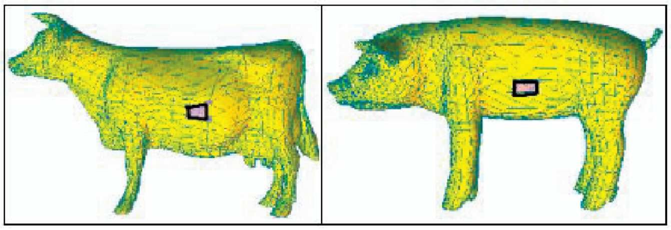
TABLE 1
The Merging Algorithm Performance Proposed by [5]

	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	

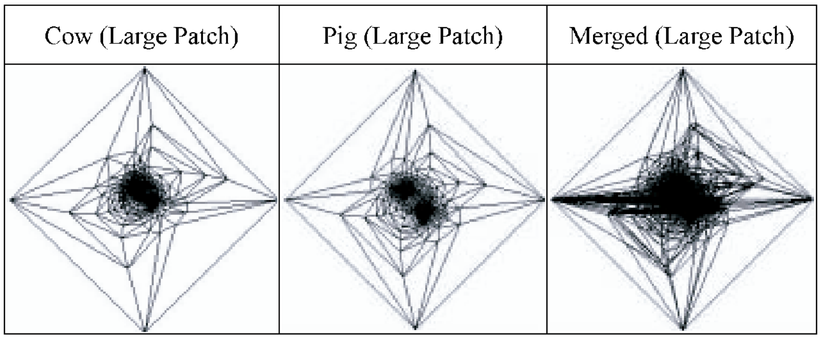
The timing information is given in the upper triangle matrix. The number of vertices in the merged model is given in the lower triangle matrix.

TABLE 2
The Performance of the EmbeddingAlgorithm Proposed by [5]

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



(a)



(b)

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)

Fig. 16. The experiment in which a model was decomposed into large and small patches. After merging, both models have 13,577 vertices and 27,897 triangles. (a) Both models were decomposed into a large patch (yellow color) and a small patch (orange color). (b) Embedding and merged embedding results of the large patch. (c) Timing information for the embedding and merging.

REFERENCES

[1] F. Lazarus and A. Verroust, "Three-Dimensional Metamorphosis: A Survey," *The Visual Computer,* vol. 14, nos. 8-9, pp. 373-389, 1998.

[2] J. Gomes, L. Darsa, B. Costa, and L. Velho, *Warping and Morphing of Graphical Objects.* Morgan Kaufmann, 1999.

[3] M. Alexa, "Mesh Morphing STAR," *Eurographics 2001 State-of-the-Art Reports,* 2001.

[4] A. Gregory, A. State, M. Lin, D. Manocha, and M. Livingston, "Interactive Surface Decomposition for Polyhedra Morphing," *The Visual Computer,* vol. 15, no. 9, pp. 453-470, 1999.

[5] M. Alexa, "Merging Polyhedral Shapes with Scattered Features," *The Visual Computer,* vol. 16, no. 1, pp. 26-37, 2000.

[6] M. Zöckler, D. Stalling, and H.-C. Hege, "Fast and Intuitive Generation of Geometric Shape Transitions," *The Visual Computer,* vol. 16, no. 5, pp. 241-253, 2000.

[7] J.R. Kent, W.E. Carlson, and R.E. Parent, "Shape Transformation for Polyhedral Objects," *Computer Graphics (Proc. SIGGRAPH 92),* vol. 26, no. 2, pp. 47-54, July 1992.

[8] F. Lazarus and A. Verroust, "Metamorphosis of Cylinder-Like Objects," *J. Visualization and Computer Animation,* vol. 8, no. 3, pp. 131-146, 1997.

[9] R.E. Parent, "Shape Transformation by Boundary Representation Interpolation: A Recursive Approach to Establish Face Correspondences," *J. Visualization and Computer Animation,* vol. 3, no. 4, pp. 219-239, 1992.

[10] D. DeCarlo and J. Gallier, "Topological Evolution of Surfaces," *Proc. Graphics Interface,* pp. 194-203, May 1996.

[11] T. Kanai, H. Suzuki, and F. Kimura, "Metamorphosis of Arbitrary Triangular Meshes," *IEEE Computer Graphics and Applications,* pp. 62-75, Mar./Apr. 2000.

[12] A. Lee, D. Dobkin, W. Sweldens, and P. Schroder, "Multiresolution Mesh Morphing," *Proc. SIGGRAPH '99,* pp. 343-350, Aug. 1999.

[13] A.W.F. Lee, W. Sweldens, P. Schroder, L. Cowsar, and D. Dobkin, "Maps: Multiresolution Adaptive Parameterization of Surfaces," *Proc. SIGGRAPH '98,* pp. 95-104, July 1998.

[14] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle, "Multiresolution Analysis of Arbitrary Meshes," *Proc. SIGGRAPH '95,* pp. 173-182, 1995.

[15] E. Praun, W. Sweldens, P. Schröder, "Consistent Mesh Parameterization," *Proc. SIGGRAPH 2001,* pp. 179-184, 2001.

[16] H. Bao and Q. Peng, "Interactive 3D Morphing," *Computer Graphics Forum,* vol. 17, no. 3, pp. 23-30, 1998.

[17] W.H. Press et al., *Numerical Recipes in C.* Cambridge, U.K.: Cambridge Univ. Press, 1992.

[18] A. Gregory, personal communication, 2002.

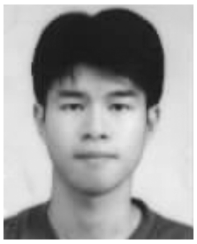
[19] N. Arad, N. Dyn, D. Reisfeld, and Y. Yeshurin, "Image Warping by Radial Basis Functions to Facial Expressions," *CVGIP: Graphical Models and Image Processing,* vol. 56, no. 2, pp. 161-172, 1994.

[20] D. Ruprecht and H. Muller, "Image Warping with Scattered Data Interpolation," *IEEE Computer Graphics and Applications,* vol. 15, no. 2, pp. 37-43, Mar. 1995.

[21] G. Wolberg, "Image Morphing: A Survey," *The Visual Computer,* vol. 14, nos. 8-9, pp. 360-372, 1998.

[22] I. Eckstein, V. Surazhskyk, and C. Gotsman, "Texture Mapping with Hard Constraints," *Proc. Eurographics 2001, Computer Graphics Forum,* vol. 20, no. 3, pp. 95-104, 2001.
[23] U. Finke and K. Hinrichs, "Overlaying Simply Connected Planar Cubdivisions in Lnear Time," *Proc. 11th Ann. ACM Symp. Computational Geometry,* pp. 119-126, 1995.
[24] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry: Algorithms and Applications.* Heidelberg: Springer-Verlag, 1997.
[25] T. Kanai, H. Suzuki, and F. Kimura, "Three-Dimensional Geometric Metamorphosis Based on Harmonic Maps," *The Visual Computer,* vol. 14, no. 4, pp. 166-176, 1998.

Tong-Yee Lee received the BS degree in computer engineering from Tatung Institute of Technology in Taipei, Taiwan, in 1988, the MS degree in computer engineering from National Taiwan University in 1990, and the PhD degree in computer engineering from Washington State University, Pullman, in May 1995. Now, he is an associate professor in the Department of Computer Science and Information Engineering at National Cheng-Kung University in Tainan, Taiwan, Republic of China. He served as a guest associate editor for the *IEEE Transactions on Information Technology in Biomedicine* in 2000, 2001 and 2002, respectively. His current research interests include computer graphics, visualization, virtual reality, surgical simulation, distributed and collaborative virtual environment. Dr. Lee is director of the Computer Graphics Group/Visual System Lab (CGVSL) at National Cheng-Kung University (http://couger.csie.ncku.edu.tw/uvr).



Po-Hua Huang received the BS degree from the Department of Computer and Information Science at National Chiao-Tung University in 1999 and the MS degree from the Department of Computer Science and Information Engineering, National Cheng-Kung University, Tainan, Taiwan, in 2001. His research interests include computer graphics, image processing, virtual reality, and game design.

▷ **For more information on this or any computing topic, please visit our Digital Library at** http://computer.org/publications/dlib.