

Metamorphosis of 3D Polyhedral Models Using Progressive Connectivity Transformations

Chao-Hung Lin and Tong-Yee Lee, *Member, IEEE*

Abstract—Three-dimensional metamorphosis is a powerful technique to produce a 3D shape transformation between two or more existing models. In this paper, we propose a novel 3D morphing technique that avoids creating a merged embedding that contains the faces, edges, and vertices of two given embeddings. This novel 3D morphing technique dynamically adds or removes vertices to gradually transform the connectivity of 3D polyhedrons from a source model into a target model and simultaneously creates the intermediate shapes. In addition, a priority control function provides the animators with control of arising or dissolving of input models' features in a morphing sequence. This is a useful tool to control a morphing sequence more easily and flexibly. Several examples of aesthetically pleasing morphs are demonstrated using the proposed method.

Index Terms—Metamorphosis, connectivity transformation, embedding, scheduling.

1 INTRODUCTION

THREE-DIMENSIONAL metamorphosis is a powerful technique in entertainment and computer animation. This technique can generate a smooth shape transformation sequence from a source object into a target object. There are two major categories of 3D morphing techniques: a volume-based approach and a surface-based approach [1], [2]. The volume-based approach represents a 3D object as a set of voxels. Lerios et al. [3] propose a 3D morphing method using fields of influence of 3D primitives such as points and lines to warp volumes. Cohen-Or et al. [4] propose a distance field metamorphosis scheme. The distance field interpolation is guided using a warping function. In contrast to the surface-based approach, the volume-based approach can support genus changing well. However, the volume-based approach is usually very computationally expensive.

Generally, most surface-based approach techniques consist of two main steps [5], [6], [7], [8], [9], [10], [12]: 1) one-to-one correspondence establishment and 2) interpolation path finding for intermediate shapes. A common approach to establish a correspondence between two given models is to generate a common connectivity for both the source and target meshes. Common embedding is a well-known way to find a common connectivity. This is generally accomplished in three steps: decomposing the models into several corresponding patches, embedding the corresponding patches onto a 2D parametric domain, and merging the corresponding embeddings to form a common interpolation mesh. Embedding merging is an overlay problem and several optimal algorithms are known in computational geometry, such as [13]. Alexa [7] and Lee

and Hung [8] also propose optimal overlay algorithms to handle 3D morphing applications.

The main drawback of embedding merging is that it usually produces many times as many triangles and vertices as the input models [5], [6], [7], [8], [9], [10], [12]. In this paper, the major contribution is a novel technique to solve this drawback. This technique uses three primitive operations, called the vertex removal operation (VRO), vertex split operation (VSO), and edge swap operation (ESO), to dynamically add or remove vertices in a morphing sequence. This approach gradually changes the connectivity from the source object into the target object and simultaneously generates a smooth morphing sequence using linear interpolation. In addition, a priority control function is designed to provide animators with easy and flexible control over a morphing sequence.

The rest of this paper is organized as follows: Section 2 reviews the related work about 3D polyhedral metamorphosis. The proposed techniques are presented in Section 3. The proposed approaches are evaluated and the experimental results are presented in Section 4. The conclusion and future work are given in Section 5.

2 RELATED WORKS

Our work is a surface-based approach and it focuses on morphs between two 2-manifold 3D polyhedrons. In this section, the most related works are surveyed. For other interesting works, see two excellent survey papers [1], [2]. Most surface-based works use a merging strategy to obtain the input model correspondences [5], [6], [7], [8], [9], [10], [11], [12], [14]. Kanai et al. [9] use harmonic mapping and embedding-merging to establish vertex correspondence for corresponding meshes. Kanai et al. [10] further describe a user-specified method to partition meshes into submeshes and then use a similar scheme [9] to establish submesh correspondence. Gregory et al. [5] present a similar user-specified control mesh method to partition meshes. The surface correspondence between each submesh of two

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

Manuscript received 26 June 2003; revised 5 Jan. 2004; accepted 22 Jan. 2004.
For information on obtaining reprints of this article, please send e-mail to: tcg@computer.org, and reference IEEECS Log Number TVCG-0050-0603.

objects is then established using a greedy area-preserving mapping. Several works use feature alignments to build better correspondence before embedding merging [6], [7], [8]. In Zockler et al.'s work [6], correspondence patches can be disk-like or cylinder-like topology. Alexa [7] embeds polyhedral models onto unit spheres. Lee and Hung [8] describe a minimal contour coverage structure to assist the embedding overlay. Manually partitioning models [5], [6], [8], [9], [10] is not an easy task for animators. Shlafman et al. [12] propose a clustering method to automatically decompose models into meaningful patches. However, this approach cannot guarantee suitable corresponding patches generation. Lee et al. [11] specify vertex pairs on two original meshes and employ a multiresolution parameterization algorithm to generate coarse models. The coarse models are then merged to establish correspondence with the assistance of a harmonic map. This approach could have a fold-over problem and user interaction is required to manually fix this problem. Once correspondence is established, linear interpolation is frequently used in morphing because of its simplicity. However, this simple method can cause self-intersection and shape degeneration. To avoid these problems, Alexa et al. [14] present an "as-rigid-as-possible" method to transform both the boundaries and interiors of meshes. However, in 3D examples, the source and target meshes are tetrahedralized and merged. The overlay of two tetrahedralizations is more complicated than the mesh overlay approach. Without model merging, Michikawa et al. [15] and Praun et al. [16] propose multiresolution remeshing techniques to establish the common mesh connectivity for morphing. However, multiresolution remeshing techniques require a great number of refinement levels to achieve the desired accuracy (in particular for sharp features) for input meshes. Therefore, a tremendous number of triangles are always produced.

Both model merging and multiresolution remeshing schemes have the similar problem of producing an interpolation common connectivity of tremendous size. From the application point of view, we may apply adaptive refinements or LOD techniques to reduce the size. However, this may be not an elegant approach to solving this problem. Alternatively, without a common connectivity, the proposed technique gradually transforms the connectivity from the source mesh into the target mesh using VSO, VRO, and ESO operations. The intermediate mesh connectivity changes and is much less complicated than previous approaches. Similar to the proposed scheme, recently, Ahn and Lee [17] use vertex matching and edge swap, etc. to perform morphing. However, for two given models M_s and M_t , this approach completes $M_s \rightarrow M_t$ (i.e., \rightarrow denotes the connectivity transformations) using the following sequence: $M_s \rightarrow M'_s$, $M'_s \rightarrow M'_t$, and then $M'_t \rightarrow M_t$, where M'_s and M'_t are simplified meshes of M_s and M_t . This long transformation sequence is not very elegant and is unnecessary. Our approach directly transforms the source mesh into the target, bypassing the process of going through simpler meshes. In addition, the quality of the examples presented in [17] is not good. This might be due to poor edge operations scheduling or not considering the fold-over problem.

3 METHODOLOGY

3.1 Overview

The overall system structure of the proposed approach is similar to our previous work [8]. First, the animators select several corresponding vertex pairs on both input polyhedrons to define a corresponding patch pair. The input polyhedral models are automatically partitioned into several corresponding patches using a modified shortest path algorithm to avoid path boundary crossing [8]. Second, each corresponding patch is then embedded on a 2D disk. Third, the animators select extra features to align the features within the corresponding embeddings using a warping function. Most surface-based approaches consist of these three steps that are very similar and well-known in morphing literature. For more details about our implementations of these three steps, please see [8]. Without embedding merging, we use a novel technique that generates morphs using progressive connectivity transformations. The core of the proposed technique is described in detail in Section 3.3.

3.2 Matching Feature Points within Corresponding Embeddings

A radial-based function is used to align features within corresponding embeddings [8]. However, sometimes a fold-over, i.e., edge self-intersection, could occur due to a deformation of embedding from this warping function. In [8], we suggest simply iterating a few relaxations to solve this problem. However, features are not guaranteed to be aligned using this simple approach. For better alignment, our improvement is described as follows: Once the fold-over is found, the two corresponding patches are partitioned into subpatches. The subpatches are embedded and warped until fold-over-free subembeddings are found. Intuitively, a partition path (i.e., line) on the embedding can be obtained by directly connecting the embedding center and one of the extra feature points. This path is selected to divide the extra feature points into two sets. Although this path is found on the 2D embedding, its corresponding 3D path can be easily found using the barycentric method. A similar idea of partitioning a patch into two patches with a line in the embedding has been proposed by Alliez et al. [18]. Note that the partition path may cross some fold-over triangles on the embedding. An incorrect path could be generated if this path is found by simply linking all triangle intersections with this partition line. To avoid an incorrect path, the selected partition path must be a continuous path in which adjacent vertices are on the same or neighboring triangles. Fig. 1 shows a repartition example. There are three extra feature points in both embeddings and the partition path passes through a feature point, as shown in Fig. 1b. This path can partition the patches into two pairs of corresponding subpatches, as shown in Fig. 1a and Fig. 1c. These subpatches are then reembedded into the 2D parametric plane again and finally warped to build a correspondence between them, as shown in Fig. 1d and Fig. 1e. Because no additional fold-over is found, the task is done.

In our practice, the above simple approach works well for scattered features. If very dense features are required, the fold-over-free warping becomes a difficult problem. To

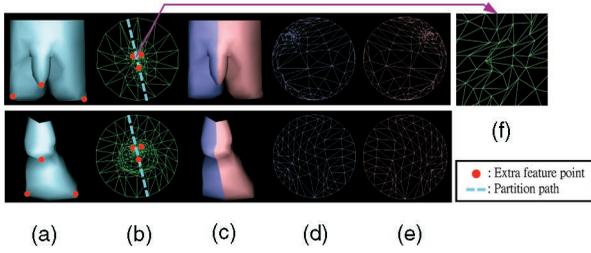


Fig. 1. (a) A pair of corresponding patch (a hoof of pig and a hoof of cow), (b) the embeddings of (a), (c) repartition (a) into two subpatches (shown in different colors) by a partition path defined in (b), (d) the embeddings of left subpatches (blue color), (e) the embeddings of right subpatches (pink color), (f) some fold-over in (b). Note that a partition path line on the embedding may correspond to a curve path in 3D as shown in (b) and (c).

alleviate this problem, the user is asked to manually partition more patches to keep a smaller number of features on each embedding. For the better approaches, two excellent works [19], [20] in texture mapping allowing dense feature constraints can be used to solve this problem.

3.3 Morphing with Connectivity Transformations

3.3.1 Progressive Connectivity Transformations

A mesh M is described by a pair (K, V) , where K is a simplicial complex representing the connectivity of vertices, edges, and faces; V describes the geometric positions of the vertices in R^3 . In the general setting of 3D morphing, two meshes, $M^S = (K_0, V_0)$ and $M^T = (K_1, V_1)$, are given. In the literature, a common method generates a sequence of intermediate meshes $M(t) = (K, V(t))$, $t \in [0, 1]$, where parameter t is the morphing time. The geometric connectivity K and the number of vertices $\|V(t)\|$ are the same among each intermediate mesh [5], [6], [7], [8], [9], [10], [12], [15], [16]. The proposed method generates intermediate meshes $M(t) = (K(t), V(t))$, $t \in [0, 1]$, where the geometric connectivity $K(t)$ and the number of vertices $\|V(t)\|$ are different among each intermediate mesh.

Given any two 2-manifold meshes M^S and M^T with the same genus, Hoppe et al. [21], [22] prove that M^S can be transformed into M^T using a finite sequence of edge splits, edge collapses, and edge swaps. In their proof, M^S can be transformed into $M^{S'}$, which is isomorphic to M^T , using a finite sequence of edge collapses and edge splits, where $M^{S'}$ is a subdivision of M^S and M^T is a subdivision of M^T . Similarly, $M^{S'}$ can be transformed into M^T using a finite sequence of edge splits and edge collapses. Although their original idea is used for mesh optimization, this proof gives us theoretical support for the correctness of our proposed algorithm. However, for 3D morphing, it is not practical to perform such a long sequence of edge splits and edge collapses to transform the connectivity of the source mesh into that of the target mesh. Alternatively, we gradually transform the mesh connectivity using a finite and suitable sequence of three primitive operations called VRO, VSO, and ESO, as shown in Fig. 2. Furthermore, maintaining a one-to-one and onto mapping function between the intermediate and input meshes is indispensable for 3D morphing. Any operation performed on an embedding must not produce incorrect fold-over triangles. The proposed method

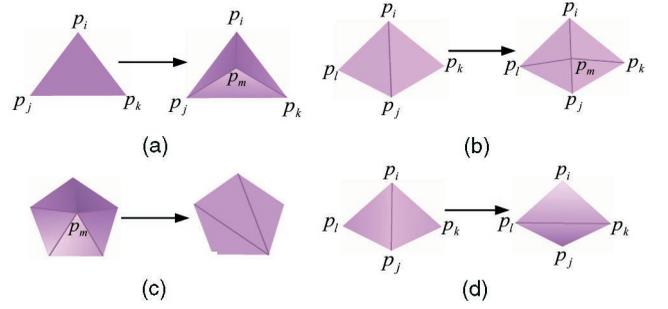


Fig. 2. The primitive operations; (a), (b): vertex split operation; (c): vertex removal operation; (d): edge swap operation.

employs VSO, VRO, and ESO to transform connectivity. Alternatively, if we employ edge collapse (ECO) instead of VRO, we will schedule ECO instead of VRO in the proposed method. In such an alternative, for example, if we schedule performing ECO on the green edge in Fig. 3a, a fold-over is created. In other words, some ECO cannot be executed at any morphing time. This is a restriction on the connectivity transform. Therefore, we employ VRO instead of ECO.

For the i th corresponding embedding pair H_i^S and H_i^T from M^S and M^T , the VRO operations are performed to remove all vertices on the source embedding H_i^S and the VSO and ESO operations are performed to insert all target embedding vertices and edges into the source embedding H_i^S , respectively. After executing all of the operations mentioned above, the connectivity of the source embedding H_i^S can be transformed into that of the target embedding H_i^T . The edge-insertion plays a critical role in the connectivity transformation. In the literature, inserting an edge into an embedding is a well-known problem in the computational geometry of constructing constrained Delaunay triangulations. Shewchuk [23] shows that an edge can be inserted into an embedding using a sequence of valid ESOs. A valid ESO will not cause a fold-over, as shown in Fig. 3b. This edge is called a valid edge. Any edge (v_m, v_n) from a target embedding can be successfully inserted into a source embedding if the vertices v_m and v_n both exist in the source and target embeddings. This insertion will not cause any fold-over. Before creating an edge (v_m, v_n) , we perform VSOs to insert vertices v_m and v_n if these two vertices are not on the source embedding. On the other hand, a source vertex or edge may be lying on an inserted target edge. To handle this case, edge insertion and vertex removal are required.

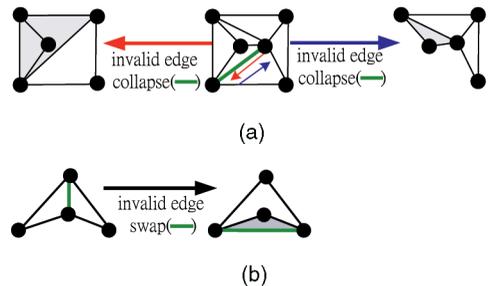


Fig. 3. (a) An edge collapse on green edge creates fold-over; (b) an invalid case of edge swap.

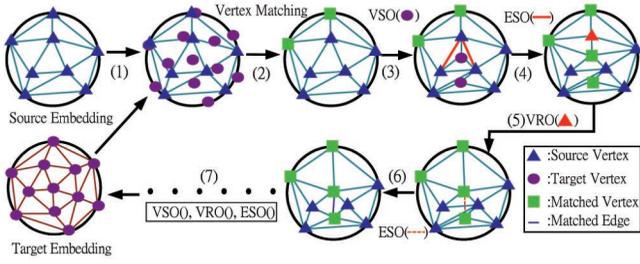


Fig. 4. A flowchart showing a progressive connectivity transformation from a source embedding into a target embedding using three primitive operations.

Therefore, after removing a source vertex lying on an inserted target edge, this inserted edge can still be successfully created using a finite sequence of valid edge swaps.

3.3.2 Scheduling Connectivity Transformations

Fig. 4 shows an overview of a progressive connectivity transformation from a source embedding into a target embedding using three primitive operations. In Step (1), the two corresponding embeddings are overlapped. In Step (2), rough vertex matching is used to decrease the number of times VRO and VSO are executed. The algorithm efficiency is therefore increased. Except for the matched vertices, the unmatched vertices of source embedding and target embedding are used to execute the VRO and VSO operations, respectively. These two operations are illustrated in Steps (3) and (5). Once VSO is performed, a vertex is inserted on the source embedding. This vertex then becomes a matched vertex. The VSO operation might produce the narrow triangle; thus, local refinement is performed for a narrow triangle in Step (4). The target embedding vertices must be inserted to raise the target embedding features. The edges of the target embedding must also be created, as shown in Step (6). Similarly, once a new edge is created, this edge becomes a matched edge. By iterating Steps (3)-(6), a sequence of primitive operations is performed, as shown in Step (7), to transform the source embedding into the target embedding. In our proposed technique, an important task is scheduling a suitable operation sequence for each frame. The goal of this scheduling is to produce a smooth morphing sequence. Therefore, a priority must be designed for each vertex and each edge of both input embeddings. This priority determines the execution orders for these operations.

The *Progressive_Connectivity_Transformations()* procedure shows the entire processes for the proposed technique. For each corresponding embedding pair, rough vertex matching is performed in the *VertexMatching()* procedure. If the position of vertex v_i^S in embedding $H_i^S(t=0)$ and the position of vertex v_j^T in embedding $H_j^T(t=0)$ are closest to one another and if embedding $H_i^S(t=0)$ is fold-over-free after assigning the position of v_i^S to that of v_j^T , we can then say that they are matched and let $v_i^S.match = v_j^T$ and $v_j^T.match = v_i^S$. Although this will cause a small error, it is meaningless for a position in $H_i^S(t=0)$ to exactly match the same position in $H_j^T(t=0)$ after a 3D-to-2D distorted embedding function. In addition, in the following pseudo-code, each H_i^S is changing as t varies but H_j^T does not. Our

implementations gradually modify the source embeddings to obtain morphs. The next procedure is *PriorityCalculation()*, which calculates a priority for each vertex and each edge in both embeddings. The features of both input objects must be preserved and gradually transformed in a morphing sequence; a simple metric for vertex priority is defined as the vertex's mean curvature multiplies using the face areas that surround the vertex. The metric for edge is defined as normal vector variations of the neighbor faces of this edge multiplied by the areas of these faces. The priority p_v of vertex and the priority p_e of edges can be written as the following equations:

$$p_v(v_i) = \text{curvature}(v_i) * \sum_{f_j \in 1\text{-ring}(v_i)} \text{area}(f_j), \quad (1)$$

$$p_e(e_i) = \|n_1 - n_2\| * (\text{area}(f_1) + \text{area}(f_2)), \quad (2)$$

where n_1, n_2 are the normal vectors of the neighbor faces f_1, f_2 of edge e_i .

```

Algorithm Progressive_Connectivity_Transformations ( $M^S, M^T$ )
{
  VertexMatching();
  PriorityCalculation();
  /*Calculate the priority of each vertex and each edge*/
  For  $t \leftarrow 0$  to 1 {
    For each correspondence embedding pair  $(H_i^S(t), H_j^T(t))$  {
      For each  $v_i \in H_i^S(t)$ 
        If  $v_i.priority \leq t$  and  $v_i.match = false$  (i.e., -1)
          VRO( $v_i$ ); /* Vertex removal operation*/
      For each  $v_i \in H_j^T(t)$ 
        If  $v_i.priority \leq t$  and  $v_i.match = false$ 
          VSO( $v_i$ );
          /* Vertex split operation: add vertex
          from  $H_j^T(t)$  onto  $H_i^S(t)$ */
      FirstPassEdgeSwap();
      /*Arise target's feature edges and dissolve source's feature
      edges by edge swap operations on embeddings*/
    }
    Interpolation( $M(t)$ );
    /*Generate the intermediate mesh  $M(t)$  using linear
    interpolation according to all  $(H_i^S(t), H_j^T(t))$  pairs*/
    SecondPassEdgeSwap( $M(t)$ );
    /*Refine the intermediate mesh  $M(t)$ */
    Output( $M(t)$ ); /*Output the intermediate mesh  $M(t)$ */
  }
}

```

In the *Progressive_Connectivity_Transformations()* procedure, the *VRO()* and *VSO()* procedures are performed first and then the *FirstPassEdgeSwap()* and *SecondPassEdgeSwap()* procedures are performed later. This means that the priorities for vertices are larger than the priorities for edges. Therefore, if a vertex is removed, the adjacent edges are also removed. In source and target embeddings, we can have four priorities $p_v^S, p_e^S, p_v^T,$ and p_e^T for each vertex and edge. Note that the priorities p_v^S, p_v^T determine the VRO execution order for vertices in the source embedding and the VSO execution order for vertices in the target embedding, respectively. The priority p_e^T determines the edge creation order in the target embedding. The priority p_e^S determines the edge dissolving order in the source embedding. Here, edge dissolving means that the edge is removed using VRO or ESO operations. To schedule these operations, a simple method we adopt involves relating the execution orders of these operations to the frame time. These priorities are sorted and

then normalized. Therefore, the range of the frame time and the priorities are the same, i.e., the value is between 0 and 1. We set $v_i.priority = p_v(v_i)$ and $e_i.priority = p_e(e_i)$ for each vertex and each edge in both embeddings. Here, the sorting order for p_v^S, p_e^S is from small to large and the sorting order for p_v^T, p_e^T is from large to small. To maintain the features in a morphing sequence, the features in the source embedding have higher preservation priorities and the features in the target embedding have higher creation priorities.

From the above descriptions, the priorities for vertices and edges are not synchronized, i.e., assigned and used independently. With this arrangement, different control functions can be designed to modify vertex and edge priorities. In this manner, more flexibility is provided for animators to control morphing sequences. These control functions are described in Section 3.5. If synchronization between vertices and edges is required, it can be achieved simply as follows: After calculating the vertex priorities, each edge priority is assigned according to the priorities of the edge's two end-vertices. Therefore, both vertices and edges can be handled at the same time, t . The VRO, VRSO, and ESO operations are described in the following.

Vertex split operation (VSO). This operation has two cases, as shown in Fig. 2a and Fig. 2b. The first case, an addition vertex p_m , lies in the interior of the triangle $\Delta p_i p_j p_k$. Edges are added from p_m to the vertices of this triangle. The second case is that p_m falls on an edge. Edges are added from p_m to vertices p_l and p_k . The 3D position of the inserted vertex p_m is computed using the barycentric method from the embedding. The priorities of these additional edges are set to zero, i.e., the lowest priority, because these edges do not belong to the input meshes. Therefore, these edges can be modified for local refinement purposes. This operation could produce a narrow triangle. Therefore, ESO is performed to locally refine this narrow triangle. In practice, an edge is swapped between two triangles if ESO improves the triangle aspect ratio of these two triangles. The criterion for this aspect ratio is defined as $area/perimeter^2$ and it is computed in 3D.

Vertex removal operation (VRO). We remove a vertex and then retriangulate the hole that is left by this vertex and its neighbor triangles. Fig. 2c shows an example. In the implementation, a triangulation method [24] is employed to fill the hole. However, there are many possible triangulations for a given hole. The triangulation that yields the least distance distortion in 3D is chosen. We use a greedy approach to check all interior edges of triangulation. Given an edge, if an edge swap can cause reduction in distortion in 3D, we execute ESO on this edge.

Edge swap operation (ESO). Consider an edge $\overline{p_i p_j}$ in an embedding, as illustrated in Fig. 2d. This edge is adjacent to two triangles $\Delta p_i p_j p_l$ and $\Delta p_i p_j p_k$. If these two triangles form a convex quadrangle, $\overline{p_l p_k}$ is a valid edge. We can obtain a new embedding by removing edge $\overline{p_i p_j}$ and then inserting edge $\overline{p_l p_k}$ instead.

In the *Progressive_Connectivity_Transformations()* procedure, the edge swap is executed at two other places. The first place is used to make the target edges emerge using a sequence of ESOs. This is described in the pseudocode *FirstPassEdgeSwap()*. If edge $e_i \in H^T$ is unmatched and its

priority is equal or less than the frame time, then this edge is inserted into the current embedding. The existence of a sequence of ESOs to insert a nonexistent edge was proven in [23], [25]. However, there is a contradiction in this procedure. If an operation ESO(e_j) (i.e., $e_j \in H^S$) belongs to this sequence, but the priority of edge e_j is larger than the frame time, the edge e_j of the source embedding cannot be moved at this time. However, ESO(e_j) needs to be executed to create the edge e_i . We solve this contradiction by stopping this procedure until the frame time is larger than the edge e_j priority. In other words, the priority of edge e_i is assigned equal to the edge e_j priority. In the *FirstPassEdgeSwap()* procedure, the intersected edges are handled in an order defined by the distance (i.e., defined in *FirstPassEdgeSwap()*) to create an unmatched edge. Parameter j is maintained to record the current edge to be handled. If edge e_j is an invalid edge, this invalid edge is bypassed and the next edge e_{j+1} is handled. If edge e_{j+1} is a valid edge, ESO(e_{j+1}) is executed and edge e_j is then handled again. In general, the number of intersections will decrease by 1 after ESO is executed. However, the number of intersections may not decrease after ESO(e_{j+1}) execution. For example in Fig. 5, edge e_1 is an invalid edge. After ESO(e_2) execution, edge e_1 becomes a valid edge and the number of intersections has not been decreased. Therefore, after ESO execution, we must check if the number of intersections has been decreased by this operation and parameter j is set to 1 to handle the first edge in edge set E again (defined in the *FirstPassEdgeSwap()* procedure). If the creation of edge e_i is completed, we will set $e_i.match \leftarrow true$.

```

Procedure FirstPassEdgeSwap()
{
  For each  $e_i \in H^T(t)$ 
  If  $e_i.priority \leq t$  and  $e_i.match = false$ 
  Find a set of edges  $E : \{e_1, e_2, \dots, e_n\} \in H^S(t)$  that intersect
  edge  $e_i$  and are listed in an order defined by the distance,
  where the distance is measured from an intersection point to
  one of the end points of edge  $e_i$  on the 2D embedding (e.g. in
  Fig. 5, the distance from each intersection to  $p_m$  is ordered as
   $e_1, e_2, e_3$ ).
  Let  $n_i$  be the number of intersections and let  $j \leftarrow 1$ ;
  /* The parameter  $j$  records the current edge to be handled in
   $E$  */
  Do
  If  $e_j.priority > e_i.priority$  then {
     $e_i.priority \leftarrow e_j.priority$ ;
    Stop this procedure; }
  Else If  $e_j$  is a valid edge then {
    ESO( $e_j$ );
    If the number of intersections has decreased by
    ESO( $e_j$ ) then {
      Remove the edge  $e_j$  from the sorted list  $S$ ;
       $n_i \leftarrow n_i - 1$ ;
       $j \leftarrow j + 1$ ;
    }
    Else If  $e_j$  is an invalid edge then  $j \leftarrow j + 1$ 
  }
  While  $n_i > 0$  /* for do while loop */
   $e_i.match \leftarrow true$ ;
}

```

After finishing the *FirstPassEdgeSwap()* procedure, the *Interpolation()* procedure is executed for all remaining vertices in the current embedding $H^S(t)$ after the VSO and VRO operations. These vertices include all matched vertices. For these vertices, their corresponding vertices in H^T are found and then linear interpolation is executed to

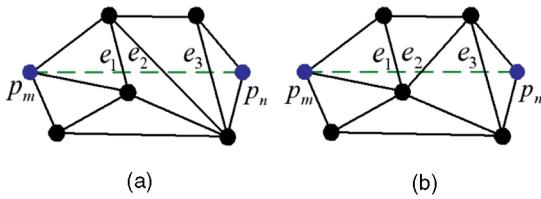


Fig. 5. (a) Edge $\overline{p_m p_n}$ interests edges e_1 - e_3 and edge e_1 is an invalid edge. After the execution of $ESO(e_2)$, (a) is transformed into (b).

compute the intermediate shapes $M(t)$. The connectivity of the intermediate mesh is determined using the connectivity of the current $H^S(t)$ at time t . The successive procedure *SecondPassEdgeSwap()* is performed for the refinement of the intermediate mesh. Because the vertex positions have been modified and may result in several narrow triangles in the intermediate mesh after several interpolation processes, a refinement process is performed in each frame.

3.4 Handling Patch Boundaries

The input meshes are decomposed into several patches and each patch is handled independently. To avoid cracks along the boundaries during the morphing, these three operations, VSO, VRO, and ESO, are carefully performed for the boundary condition as follows:

1. **VSO**: If we insert a new vertex on the boundary, we will create two triangles on each adjacent patch. For example, in Fig. 6a, if the edge (v_a, v_b) is a shared boundary, as we insert v_i on this boundary, we will create two triangles on each adjacent patch.
2. **VRO**: If a boundary vertex is removed, we will create a hole on each adjacent patch. Each hole is then independently retriangulated (as shown in Fig. 6b).
3. **ESO**: We do not allow edge flips on the boundary. The boundary edges of the source mesh can be naturally transformed into the boundary edges of the target mesh after a sequence of boundary vertex insertions and boundary vertex removals.

Using the above method, the proposed method does not create cracks along the boundaries.

3.5 Priority Control Function (PCF)

A priority control function is defined for easy and flexible control over the input model's arising and dissolving features. In the *PriorityCalculation()* procedure, a sorted and normalized priority p for each vertex and each edge is provided. These priorities can be modulated using different priority control functions $pcf(p)$ s. Each $pcf(p)$ function is a monotonically increasing function and $pcf(p)$ is constrained as $0 \leq pcf(p) \leq 1$, where $0 \leq p \leq 1$. The user defines four functions, $pcf_v^S(p)$, $pcf_e^S(p)$, $pcf_v^T(p)$, and $pcf_e^T(p)$ for the vertices and edges of both input models, respectively. The left side of Fig. 7 shows several examples using different priority control functions. Their corresponding effects are shown on the right side of Fig. 7. For example, in Fig. 7d, the geometric connectivity of the input models is gradually transformed and the number of vertices in the intermediate models is linearly increased. In contrast to Fig. 7d, Fig. 7e shows a nonlinear case of vertex priorities. The number of vertices increases slowly at the beginning and end of a

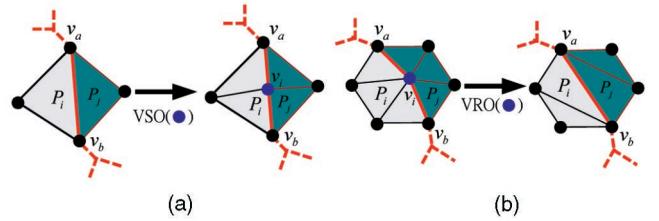


Fig. 6. Maintaining the patch boundaries as we are (a) inserting a boundary vertex or (b) removing a boundary vertex.

morphing sequence. The priority control function definition in Fig. 7c is proper when the target model has several times as many vertices as the source model, such as in the example in Fig. 10. Because executing VRO at the beginning of this morphing sequence might cause a popping problem due to the great variation, this priority control function performs VSO and ESO only before $t = 0.5$ to reduce the popping problem. In Fig. 7b, the priorities for the source and target vertices are set to 1 and 0, respectively. This implies that all VSOs will be performed at the beginning and there will be no VRO operation performed in this morphing sequence. Therefore, the number of vertices for intermediate meshes is $(m + n - k)$, where m and n are the number of vertices for the input meshes and k is the number of matched vertices in *VertexMatching()*. Fig. 7a shows a case resulting from a general approach that merges the embeddings. In this case, we do not need the control functions. The number of vertices for merged intermediate mesh is 10,300. This is about five times the number of vertices in the intermediate meshes in Fig. 7b. From these examples, we see that the priority control function can easily control the degree of detail represented in each frame of the morphing sequence. A comparison of the number of vertices for intermediate meshes using different control functions in Fig. 7 is shown in Fig. 8.

Priority control functions (PCF) provide flexibility to change the priorities of vertices and edges and the vertices and edges are handled independently. Using the proposed method with different PCF, the intermediate meshes can contain vertices and edges from both models. It is very natural since the morphing sequence is gradually changing from source to target meshes. In some extreme cases, for example, Fig. 7b, at $t = 0.0$, all target vertices are inserted into the source embedding using VSOs. Then, for each VSO, three or four new edges are created on the source embedding (see VSO in Fig. 2a and Fig. 2b.). This example can be treated like surface subdivision. It is impossible to create some extreme cases such as having all target vertices with all source edges without new edges for an intermediate mesh or vice versa. Such cases in terms of the shape seem meaningless for morphing application. In addition, it is also meaningless to schedule inserting a target edge ahead of its two vertices. But, the reverse can be fine, such as Fig. 7b. Finally, from the above discussion, using the proposed scheme with PCF, the number of vertices in an intermediate mesh is between $\min(m, n)$ and $m + n - k$, where m and n are the number of vertices in both models and k is the number of matched vertices in the *VertexMatching()* procedure.

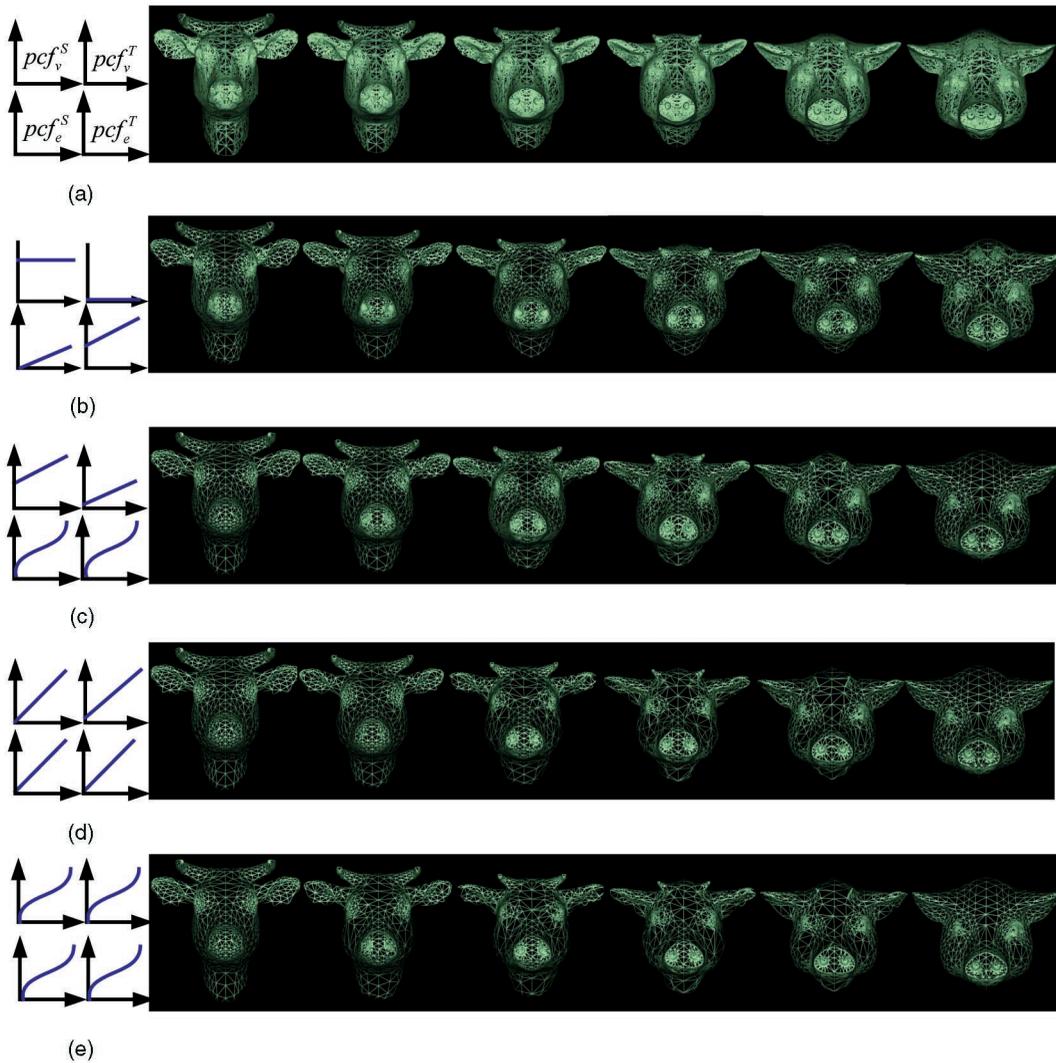


Fig. 7. Left: Different priority control functions; the x-axis represents the old priority p and the y-axis represents the new priority after modulation, i.e., $pcf(p)$. Right: The various effects of the morphing sequence using the priority control functions shown on the left.

4 EXPERIMENTAL RESULTS

In this section, we present experimental results that were executed on a Pentium 4 2.2G PC with 256M memory. Digital video clips of all morphs presented in this paper can be found on this Web site: <http://couger.csie.ncku.edu.tw/>

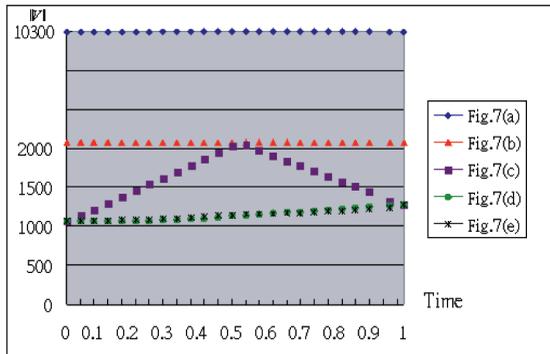


Fig. 8. A statistical plot shows the comparison of the number of vertices among all cases in Fig. 7.

[~vr/PM /Demo.htm](#). In Fig. 9, we show an example of partitioning model into patches for horse and cow models using the proposed method. Fig. 10a and Fig. 10b show the morphing sequences resulting from the proposed method and an embedding merging method, respectively, with the same surface decomposition. In terms of the complexity, i.e., the number of vertices, edges, and faces, the intermediate objects generated using the proposed method are more reasonable than those produced by an embedding merging method. In this example, the numbers of vertices



Fig. 9. Compatible patch decomposition of models (patches shown in different colors).

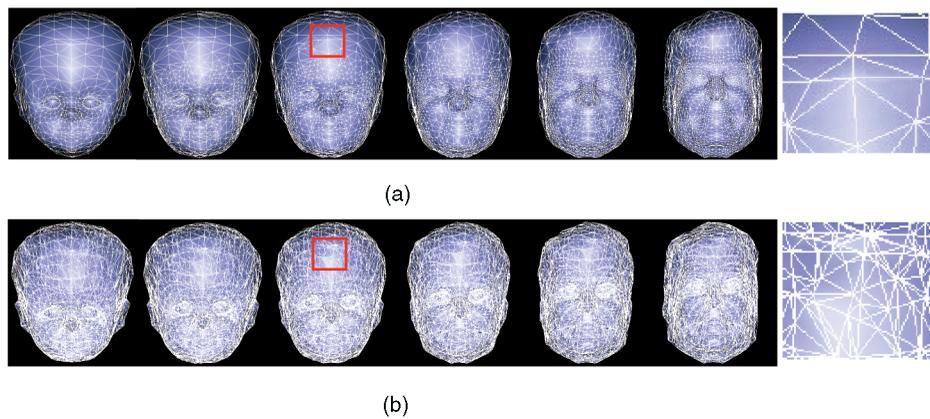


Fig. 10. (a) A morphing sequence generated by the proposed method. (b) A morphing sequence generated by an embedding merging method. Right: Two local regions enlarged from the red quadrangles.

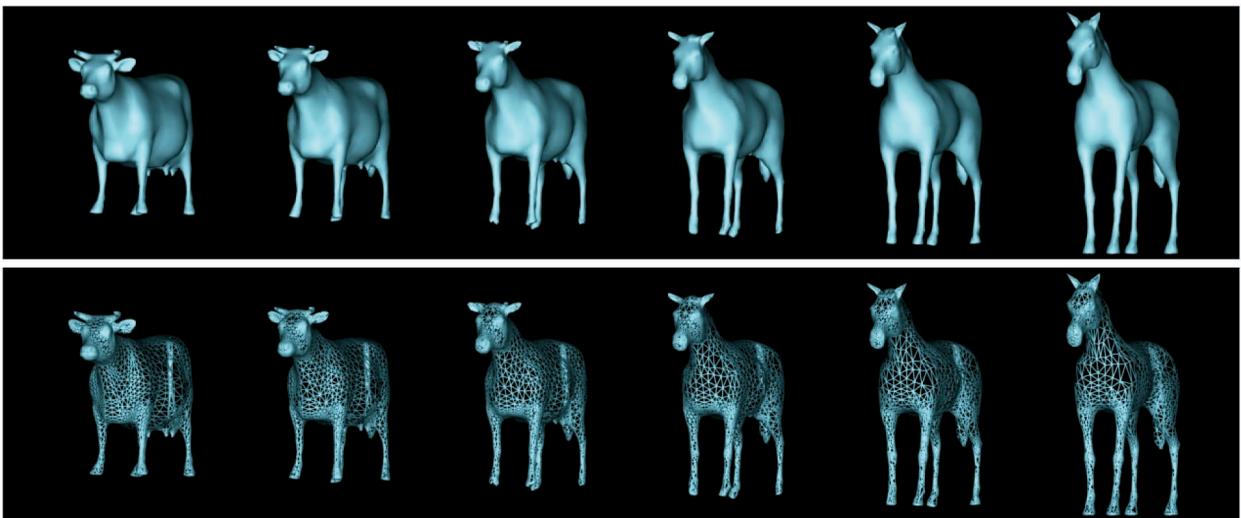


Fig. 11. A morphing between the models of a cow and a horse.

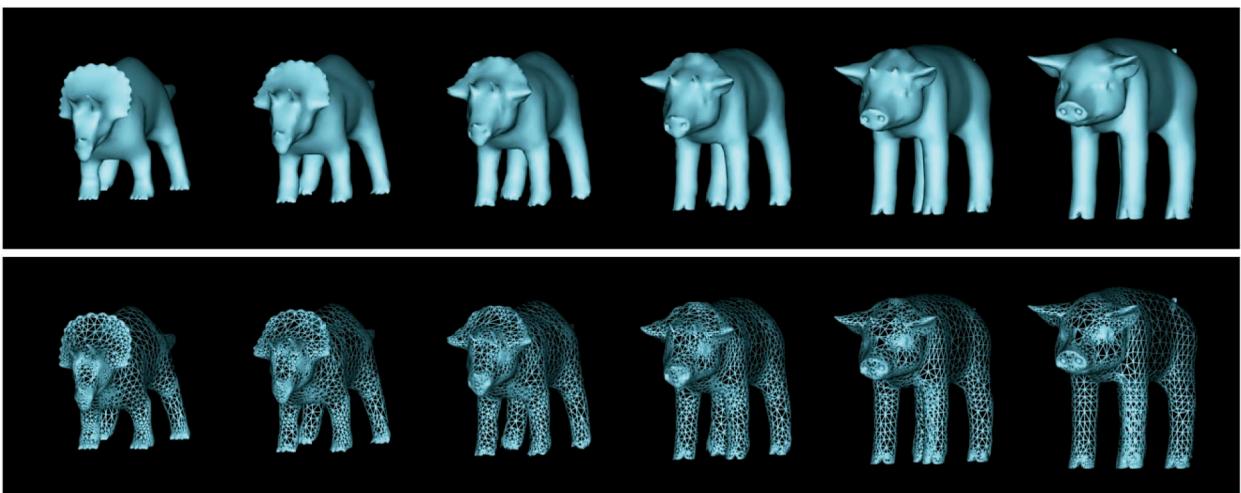


Fig. 12. A morphing between the models of a triceratops and a pig.

for the input models are 570 and 1,954, respectively. However, the numbers of vertices for the merged objects are up to 10,938 and the vertices in a flat region are very dense. With the proposed method, the number of vertices for the intermediate objects ranged between 570 and 1,954.

The proposed method gradually transforms the shapes between the input models and linearly increases the number of vertices. We demonstrated several pleasing shapes and the connectivity transformations in Figs. 11, 12, 13, and 14. These morphing examples between two meshes

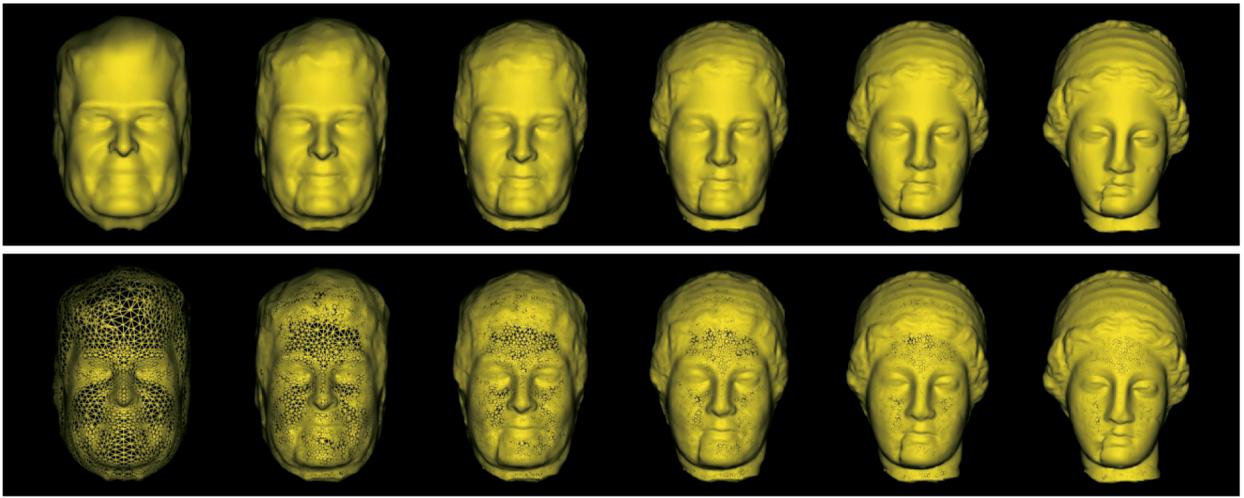


Fig. 13. A morphing between the models of a man's head and a Venus head.

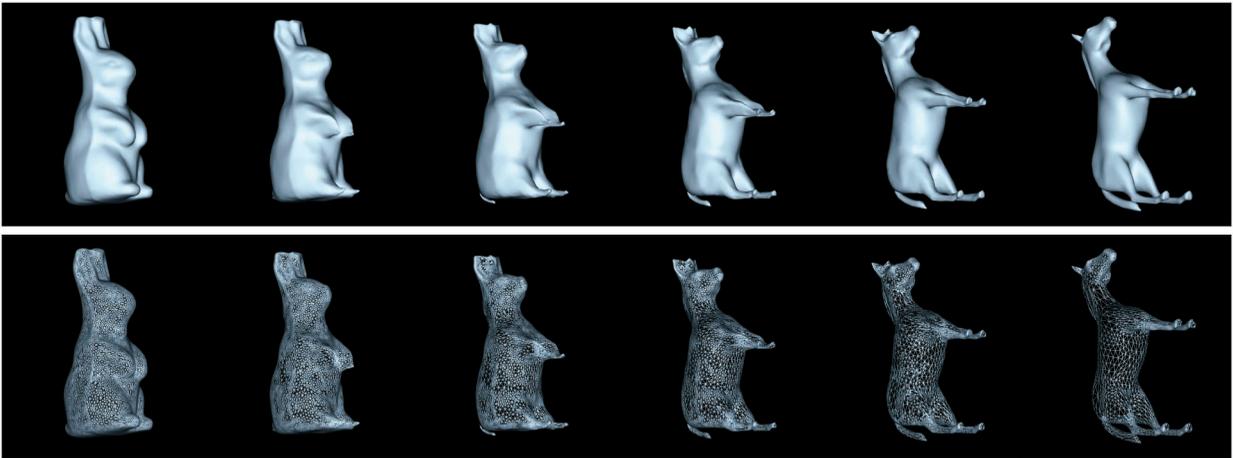


Fig. 14. A morphing between the models of a rabbit and a horse.

with different vertex and edge numbers varying from small to large differences are used to demonstrate the robustness of the proposed algorithm. For example, in Figs. 11 and 12, the vertex and edge numbers are close for both the source and target meshes. Fig. 13 shows an extreme case. The numbers of vertices for the input models are 1,975 and 50,002, respectively. The target model has about 25 times as many vertices as the source model. In Fig. 14, the legs of the horse are very noticeable, but rabbit's are not. Similarly, the ears of the rabbit are more prominent features than those of the horse. Therefore, the details of the input models are on different regions in this example [11]. In these examples, the proposed method generates a smooth shape and connectivity transformation.

Table 1 shows the experimental statistics for Figs. 11, 12, 13, and 14. The morphing time in Table 1 represents the computation time for executing three primitive operations and generating the intermediate objects in 50 frames. In each 50 frame morphing sequence, the time to generate an in-between mesh is approximately the same. In contrast to generating a sequence, to generate a single frame at $t = 0.75$ will take longer than that for a single frame at $t = 0.25$. This is because the connectivity transformations for $t = 0.25$ are

part of that for $t = 0.75$ (i.e., progressive connectivity transformations). In this table, we also show the number of matched vertices and edges. A higher matched number can reduce the computation time. Fig. 13 shows an extreme case in which all source vertices are matched. Therefore, we do not need to execute VROs and thus save some computation time. In this table, the number of ESOs in the first pass is not small. This is because we need to perform a sequence of ESOs to insert a target edge. To insert an edge, the number of ESOs is related to the number of intersections with the source edges. The upper bound of the number of ESOs to transform connectivity between two embeddings can be proven as the number of intersections between the edges of two embeddings [25]. In Table 1, the number of ESOs in the first pass is between the edge numbers of both models. Therefore, the number of ESOs is reasonable in the proposed method. In addition, consider two morphing sequences, one with 50 frames and the other with 100 frames. The proposed scheme relates the execution orders of primitive operations to the morphing time t , $0 \leq t \leq 1$. Using the proposed scheme, the 25th frame in the 50 frame sequence and the 50th frame in the 100 frame sequence will be the same, since both frames correspond to

TABLE 1
Statistics

	Source Model (#vertices / #edges)	Target Model (#vertices / #edges)	no. of matched vertices	no. of matched edges	no. of VROs	no. of VSOs	no. of ESOs			Morphing time (50 frames)
							VRO	1st Pass	2nd Pass	
Fig. 11	2703 / 7881	2982 / 8607	1557	2000	1146	1425	1489	6551	2031	22.62 sec
Fig. 12	2637 / 8229	2795 / 8979	1819	1558	818	976	1169	7038	1966	19.56 sec
Fig. 13	1954 / 5856	50002 / 150000	1954	254	0	48048	0	143380	43052	598.95 sec
Fig. 14	10002 / 30000	2982 / 8940	1836	977	8166	1146	10697	40220	10891	107.54 sec

Statistics: the first two columns show the number of vertices and edges for the input models. The 3th-4th columns show the number of matched vertices and edges after performing *VertexMatching()* procedure. The 5th-9th columns show the number of primitive operations to be performed. The last column shows the computation time for the proposed method.

TABLE 2
Statistics

	Source Model (#vertices/#faces)	Target Model (#vertices/#faces)	Priority control function (PCF)	The range of the number of vertices and faces in the in- between meshes
Fig. 11	2703 / 5402	2982 / 5960	Fig. 7(e)	2703~2982 / 5402~5960
Fig. 12	2637 / 5270	2795 / 5586	Fig. 7(c)	2637~3613 / 5270~7236
Fig. 13	1954 / 3849	50002 / 100000	Fig. 7(e)	1954~50002 / 3849~100000
Fig. 14	10002 / 20000	2982 / 5960	Fig. 7(e)	2982~10002 / 5960~20000

Statistics: the range of the number of vertices and faces in the in-between meshes

$t = 0.5$ in two sequences. The intermediate meshes at $t = 0.5$ for two sequences are generated after performing the same sequence of primitive operations in which their priorities are equal to or less than 0.5.

A popping effect could occur when intermediate objects are rendered. In the proposed paper, the following strategies are used to reduce the popping effects: 1) Because the popping effects are due to the execution of three primitive operations, vertex matching is performed to decrease the number of executions. Table 1 shows the number of matched vertices and edges after performing the *VertexMatching()* procedure. This procedure can decrease the number of primitive operations to be executed. 2) Appropriately scheduling three primitive operations can also reduce the popping effects. The priorities for both vertices and edges are assigned according to their 3D information and an appropriate priority control function is chosen simultaneously to maintain the input model feature and reduce the popping effects. Another possible solution for reducing the popping effects is to evaluate the suitable primitive operations in each frame. However, this method is time-consuming. Finally, we also show some statistics in Table 2 about examples Figs. 11, 12, 13, and 14. In contrast to previous work, the number of faces is not too many in these examples. Unlike other examples, the number of vertices and triangles in Fig. 12 is greater than those of input models because we choose Fig. 7c as its PCF. Using Fig. 7c, we begin to perform VROs after $t = 0.5$ and finish all VSOs at $t = 0.5$. Therefore, at $t = 0.5$, the number of vertices is maximum for $M(t = 0.5)$.

5 CONCLUSIONS AND FUTURE WORK

A novel approach for computing morphs between polyhedral 3D objects is presented. This approach can add or remove vertices dynamically using three primitive operations. The connectivity of the source model can thus be transformed into that of the target model. In contrast to other previous works, the connectivity of an in-between mesh is gradually changing and less complicated than those of other embedding merging and multiresolution remesh approaches. Several future works can be done and are described as follows: Linear interpolation may cause shape self-intersection. We are planning to design new approaches to solve this problem. To perform 3D morphs with texture is other interesting and challenging future work. Simply interpolating texture attributes can only work for simple examples. A better method such as [19], [20] must be designed. Although the morphing sequences are smooth in the experiments, we cannot guarantee the popping problem will not occur in extreme examples. For example, if there are many primitive operations performed on a small region at the same time, the popping problem will occur in this region at that time. Evaluating a suitable selection of primitive operations for each frame could solve this problem. In addition, the priority functions introduced are based on a geometrical metric; however, the change in connectivity caused by introducing the element (vertex, edge) should be another possible metric to be considered in future work. However, the two approaches above may be time-consuming. Fast morphing can be possible if a set of transformations during

morphing are calculated as a preprocess and are recorded in advance [26]. For fast morphing, we can include this technique in the near future.

ACKNOWLEDGMENTS

The authors extend their thanks 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-92-2213-E-006-066, NSC-92-2213-E-006-067, and NSC-93-2213-E-006-026.

REFERENCES

- [1] F. Lazarus and A. Verroust, "Three-Dimensional Metamorphosis: A Survey," *The Visual Computer*, vol. 14, pp. 373-389, 1998.
- [2] M. Alexa, "Mesh Morphing," *STAR: State of The Art Report, EUROGRAPHICS*, 2001.
- [3] A. Lerioux, C.D. Garfinkle, M. Levoy, "Feature-Based Volume Metamorphosis," *ACM SIGGRAPH Conf. Proc.*, pp. 449-456, 1995.
- [4] D. Cohen-Or, D. Levin, and A. Solomovici, "Three-Dimensional Distance Field Morphing," *ACM Trans. Graphics*, vol. 17, no. 2, pp. 116-141, 1998.
- [5] 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.
- [6] M. Zockler, 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] M. Alexa, "Merging Polyhedral Shapes with Scattered Features," *The Visual Computer*, vol. 16, no. 1, pp. 26-37, 2000.
- [8] T.-Y. Lee and P.-H. Hung, "Fast and Intuitive Metamorphosis of 3D Polyhedral Models Using SMCC Mesh Merging Scheme," *IEEE Trans. Visualization and Computer Graphics*, vol. 9, no. 1, pp. 85-98, Jan.-Mar. 2003.
- [9] T. Kanai, H. Suzuki, and F. Kimura, "Three-Dimensional Geometric Metamorphosis Based on Harmonic Maps," *The Visual Computer*, vol. 14, pp. 166-176, 1998.
- [10] T. Kanai, H. Suzuki, and F. Kimura, "Metamorphosis of Arbitrary Triangular Meshes," *IEEE Computer Graphics and Applications*, pp. 62-75, 2000.
- [11] A. Lee, D. Dobkin, W. Sweldens, and P. Schroder, "Multi-resolution Mesh Morphing," *ACM SIGGRAPH Conf. Proc.*, pp. 343-350, 1999.
- [12] S. Shlafman, A. Tal, and S. Katz, "Metamorphosis of Polyhedral Surfaces Using Decomposition," *EUROGRAPHICS*, vol. 21, no. 3, pp. 219-228, 2002.
- [13] U. Finke and A. Hinrichs, "Overlaying Simply Connected Planar Subdivisions in Linear Time," *Proc. 11th Ann. Sym. Computational Geometry*, pp. 119-26, June 1995.
- [14] M. Alexa, D. Cohen-Or, and D. Levin, "As-Rigid -as-Possible Shape Interpolation," *ACM SIGGRAPH Conf. Proc.*, pp. 157-164, 2000.
- [15] T. Michikawa, T. Kanai, M. Fujita, and H. Chiyokura, "Multi-resolution Interpolation Meshes," *Proc. Ninth Pacific Graphics Int'l Conf. (Pacific Graphics 2001)*, pp. 60-69, Oct. 2001.
- [16] E. Praun, W. Sweldens, and P. Schroder, "Consistent Mesh Parameterization," *ACM SIGGRAPH Conf. Proc.*, pp. 179-184, 2001.
- [17] M. Ahn and S. Lee, "Mesh Metamorphosis with Topology Transformation," *Proc. Pacific Graphics 2002*, pp. 481-482, Oct. 2002.
- [18] P. Alliez, M. Meyer, and M. Desbrun, "Interactive Geometry Remeshing," *ACM SIGGRAPH Conf. Proc.*, pp. 347-354, 2002.
- [19] I. Eckstein, V. Surazhsky, and C. Gotsman, "Texture Mapping with Hard Constraints," *Computer Graphics Forum*, vol. 20, no. 3, pp. 95-104, 2001.
- [20] V. Kraevoy, A. Sheffer, and C. Gotsman, "Matchmaker: Constructing Constrained Texture Maps," *ACM SIGGRAPH Conf. Proc.*, pp. 326-333, 2003.
- [21] H. Hoppe, T. DeRose, T. Dunchamp, J. McDonald, and W. Stuetzle, "Mesh Optimization," *ACM SIGGRAPH Conf. Proc.*, pp. 19-26, 1993.
- [22] H. Hoppe, T. DeRose, T. Dunchamp, J. McDonald, and W. Stuetzle, "Mesh Optimization," Technical Report TR 93-01-01, Univ. of Washington, 1993.

- [23] J.R. Shewchuk, "Updating and Constructing Constrained Delaunay and Constrained Regular Triangulations by Flips," *Proc. Symp. Computational Geometry*, pp. 181-190, 2003.
- [24] R. Seidel, "A Simple and Fast Incremental Randomized Algorithm for Computing Trapezoidal Decompositions and Triangulating Polygons," *Computational Geometry: Theory and Applications*, pp. 51-64, 1991.
- [25] S. Hanke, T. Ottmann, and S. Schuierer, "The Edge-Flipping Distance of Triangulations," *J. Universal Computer Science*, vol. 2, no. 8, pp. 570-579, Aug. 1996.
- [26] H. Hoppe, "Efficient Implementation of Progressive Meshes," *Computers & Graphics*, vol. 22, no. 1, pp. 27-36, 1998.



Chao-Hung Lin received the BS degree in computer science/engineering from Fu-Jen University, Taipei, Taiwan, in 1997. He received the MS and PhD degrees in computer engineering from National Cheng-Kung University, Tainan, Taiwan, in 1998 and 2004, respectively. He is now serving in the Taiwan army. His research interests include computer graphics and image processing.



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 a 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* from 2000 to 2004. His current research interests include computer graphics, image-based rendering, visualization, virtual reality, surgical simulation, and distributed and collaborative virtual environment. He leads a Computer Graphics Group/Visual System Lab at National Cheng-Kung University (<http://couger.csie.ncku.edu.tw/~vr>). He is a member of the IEEE.

► For more information on this or any computing topic, please visit our Digital Library at www.computer.org/publications/dlib.