to appear in IEEE Computer Graphics and Applications

Metamorphosis of Arbitrary Triangular Meshes

Takashi Kanai

Materials Fabrication Laboratory,
The Institute of Physical and Chemical Research (RIKEN),
2-1 Hirosawa, Wako-shi, Saitama, 351-0198, Japan.
Tel: +81-48-467-9319
Fax: +81-48-462-4657

Hiromasa Suzuki Fumihiko Kimura

Department of Precision Machinery Engineering,
Graduate School of Engineering, The University of Tokyo,
Room No.921, Engineering Building No.14,
3-1, Hongo 7-chome, Bunkyo-ku, Tokyo, 113-8656, Japan.
Tel: +81-3-3812-2111 ext. 6495
Fax: +81-3-3812-8849

Abstract

Recently, animations with deforming objects have been frequently used in various computer graphics applications. Metamorphosis (or morphing) of three-dimensional objects is one of the techniques which realizes shape transformation between two or more existing objects. In this paper, we present an efficient framework for metamorphosis between two topologically equivalent, arbitrary meshes with the control of surface correspondences by the user. The basic idea of our method is to partition meshes according to the reference shapes specified by the user, whereby vertex-to-vertex correspondences between the two meshes can be specified. Each of the partitioned meshes is embedded into a polygonal region on the plane with harmonic mapping. Those embedded meshes have the same graph structure as their original meshes. By overlapping those two embedded meshes, we can establish correspondence between them. Based on this correspondence, metamorphosis is achieved by interpolating the corresponding vertices from one mesh to the other. We demonstrate that the minimum control of surface correspondences by the user generates sophisticated results of the interpolation between two meshes.

Keywords: Geometric Modeling, Metamorphosis, Surface Correspondence, Harmonic Mapping

1 Introduction

Three-dimensional (3D) metamorphosis (or morphing) that establishes a smooth transition from a source object to a target object, is an active research area in computer graphics. We handle 3D geometric metamorphosis between two objects which are represented as triangular meshes. The primary issue in 3D metamorphosis is to establish surface correspondence between the source and target objects, by which each point on the surface of the source object is mapped to a point on the surface of the target object [16]. Once this correspondence is established, it is possible to generate a smooth transition

by interpolating corresponding points from the source to the target positions. To improve the quality of 3D metamorphosis between two triangular meshes, we particularly consider the following two issues:

Metamorphosis of arbitrary meshes In general, the two meshes have different topological structures, that is, vertex/edge/face graph structures. Moreover, meshes may have complicated geometric shapes such as non-convex regions. Surface correspondence between two such different and/or complicated meshes must be established.

Metamorphosis with user control An appropriate method for the user to control surface correspondence is needed. For example, the user may wish to map a bunny's nose onto a tiger's nose. We need a method to incorporate such user-specified surface correspondence with minimum user intervention.

The former issue can be addressed by a recently proposed method based on harmonic mapping by Kanai et al. [15]. In [15], each of the two meshes, which are topologically equivalent to a disk and have geometrically complicated shapes, is developed into a two-dimensional unit circle by harmonic mapping [10]. Surface correspondence between the two meshes is established by combining those two embeddings. However, this method does not consider the latter issue of how to allow the user to control surface correspondence. The purpose of our paper is to develop an effective method for 3D metamorphosis between two arbitrary meshes which are of the same topology. We extend our previously proposed method [15] to achieve user control of surface correspondence.

The rest of our paper is partitioned into the following sections: Section 2 reviews the background of 3D metamorphosis, definitions, problem statements, and related work. Section 3 describes an overview of the method for 3D metamorphosis [15] with minor modifications. Section 4 gives an extended method for 3D metamorphosis including user control of the surface correspondence. Section 5 shows our results, and we discuss from various viewpoints in Section 6. We conclude with suggestions for future work and applications in Section 7.

2 Background

2.1 Problem Statement

In our paper, we deal with 3D metamorphosis between two topologically equivalent, two-manifold, arbitrary meshes, from *the source mesh* $\mathcal{M}^1$ to *the target mesh* $\mathcal{M}^2$. A *mesh* $\mathcal{M}$ refers to a piecewise linear surface which consists of a set of triangular polygons. A mesh consists of two *realizations*: *Topological realization* K is a vertex(v)/edge(e)/face(f) graph structure, and *geometric realization* V is a set of vertex coordinates $V = \{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n\} \in \mathbf{R}^3$. The mesh can be arbitrary, that is, no special restrictions are imposed on those realizations. We can also handle a mesh with *boundaries*. A *boundary* of a mesh $\partial\mathcal{M}$ is a closed loop which consists of a set of edges, each of which has only one neighbor face.

Two meshes are said to be *topologically equivalent* if a continuous, invertible, one-to-one mapping between points on the two meshes exists. Such a mapping is referred to as *homeomorphism*. A mesh is said to be *Euler-valid* if its topology obeys *Euler's formula*:

$$n_v - n_e + n_f = 2 - 2g,$$

where n_v, n_e and n_f denote the number of vertices, edges, and faces, respectively. g denotes the genus of a mesh. If two meshes are topologically equivalent, the left-side members of Euler's formula (called *Euler's characteristic*) for both meshes are equal. In our paper, we assume that g is zero and discuss the extendability to the case for non-zero g in Section 5.

In this paper, *3D metamorphosis* is to interpolate two given meshes $\mathcal{M}^1$ and $\mathcal{M}^2$. Its more precise definition will be given in Section 3.4.

2.2 Related Work

Techniques which known as "metamorphosis" or "morphing" were originally referred to as image metamorphosis," or "image morphing." They are popular techniques used for creating a smooth transition between two images by approximately interpolating both color information and geometric shapes (for details, see a survey book[11]). To establish the metamorphosis between two 3D objects using above image-based techniques, at least two images, one of which displays one object and another displays the other object, are needed. But any time viewing or lighting parameter are changed, all processes of 2D morph must be recomputed.

3D metamorphosis (or morphing), direct geometric interpolation of two 3D objects, is independent of such viewing or lighting parameters. Therefore, it has become an area of active research in recent years (A detailed survey paper is [19], and also in [11]). Approaches for establishing 3D metamorphosis are classified into two categories: volumetric approach and mesh-based approach.

Volumetric Metamorphosis

One class of 3D metamorphosis algorithms deals with sampled or volumetric representation of the objects.

Hughes [14] has proposed the transformation of objects into the Fourier domain, and then the interpolation of low frequencies over time while the high frequencies are slowly added in, thus minimizing the object distortion caused by the high-frequency components. Intermediate objects are obtained from the inverse transform of the interpolated results. Cheung et al. [5] have proposed a similar method for use in the simulation of a metal forming process. Wavelet-based approach has been proposed for use instead of Fourier transform by He et al. [13] so as to further improve performance.

Lerios et al. [21] have extended Beier's feature-based image metamorphosis method [2] to volumetric representation. Two objects are translated into volumes, and each corresponding pair of volume values is interpolated according to the primitive shapes, which are defined by the user. Lucas et al. [23] have proposed a very similar method, using a 3D line set directly for the control of interpolation between two volumetric objects.

Cohen-Or et al. [6] have proposed an approach based on DFI (distance field interpolation) method. This approach is divided into two phases: the warp phase and the interpolation phase. In the warp phase, first each object is decomposed into contour level volume sets, then at each level a volume space is deformed so that two sets of feature points, extracted by the user, are coincident with each other. In the interpolation phase, volumes are transformed to a distance field, a linear interpolation between two distance fields deformed in previous phase is established.

These methods can deal with the topological changes in the surface mesh during transformation. It is also possible to apply these methods for meshes if these are transformed to volumes. However, the intermediate shape is represented as a volume, and extraction to isosurface by such means as the marching cube method [22] is required for obtaining meshes. Moreover, these objects may not be homogeneous to their original shapes.

Mesh-Based Metamorphosis

Another class of algorithms deals with polyhedron-based objects. In this class, an approach to the transformation from $\mathcal{M}^1$ to $\mathcal{M}^2$ usually involves two problem steps. The first step is to establish a correspondence from each point of $\mathcal{M}^1$ to a point of $\mathcal{M}^2$. Using this correspondence, the next step creates a series of intermediate objects by interpolating corresponding points from their original positions on $\mathcal{M}^1$ to the target positions on $\mathcal{M}^2$. These steps are called *correspondence problem* and *interpolation problem*, respectively [16]. Our approach falls in this class and follows these steps. Especially, we primarily discuss the former issue.

Kent et al. [17, 16] offer an algorithm for the metamorphosis of two polyhedral objects topologically equivalent to a sphere ($g = 0$). First, two objects are mapped onto a sphere and merged by clipping one sphere to another. Then, a new shape which has a combined graph information (vertices, edges, and faces) for the two objects is created. However, the technique described in [17] is applicable only to star-shaped polyhedral objects. It was further extended in [16] to the other types of objects such as tubular (swept or revolutionary) objects which involve objects reconstructed from cylindrical scan-type range images like Cyberware. To establish correspondences over these types of objects, the user must specify a lib along the center of the contour of an object, and must project it to a convex-hull as an intermediate object. In this approach, various types of objects can be treated, but it seems to be a rather troublesome work for the user to specify such a lib so as to establish correspondences especially for geometrically complicated objects.

Parent [26] presents a recursive algorithm which automatically finds a correspondence between the surfaces of two objects of equivalent topologies. This algorithm uses several *sheets* for covering the whole object. These two objects must have an equal number of sheets. Sheet boundaries must be composed of the edges of the object. Objects of genus g are automatically subdivided into $2(g+1)$ sheets. Each sheet is recursively subdivided down to the face level. Vertex-to-vertex interpolations and thus deformations between the two objects are completely established.

Delingette et al. [8] use a *simplex mesh*, and propose basic mesh operations to alter shape topologies. Thus

the method is restricted to this type of mesh. The interpolation is performed using a physically-based deformation approach in concert with a method derived from a data-fitting process. The process, however, seems to be time-consuming.

Decarlo et al. [7] describe another framework for the metamorphosis between two objects with different topologies, for example, the metamorphosis from a sphere to a torus. This method has a similar methodology to our method. To make the surface correspondence between two objects with different topologies, each of two meshes is divided into a set of sub-meshes, according to a set of the user-specified triangular or quadrilateral reference faces, called a *control mesh*. Metamorphosis between two objects with different topologies is established by associating that each triangular face in one control mesh changes to the corresponding quadrilateral face in the other. One difference between Decarlo's method and ours is that our approach establishes correspondences more easily for the user. For example, there is no restriction about the number of edges of the control mesh. The other difference is a relationship between the subdivision of meshes and the parameterization of each sub-mesh. Decarlo et al.'s method uses Kent et al.'s parametrization[16] which has the limitation for the type of meshes. In our method, the user does not need to care whether each sub-mesh can be parameterized successfully or not. Thus, our method presents more sophisticated approach for the user to establish correspondences.

Recently, Gregory et al. presented a similar approach [12]. Gregory et al.'s paper treats the metamorphosis between two objects with same topological types. First the user specifies a curve net, called a *feature net*, consists of vertices and edges of the mesh and the mesh is divided into several sub-meshes according to this net. Then surface correspondences between each sub-mesh of two objects are established using area-preserving mapping. Desirable metamorphosis for the user can be generated based on local refinements of a curve net. One difference between Gregory et al.'s method and our method is an approach for 'cutting' the mesh to divide into sub-meshes. Our cutting uses an approximate geodesic curve on the mesh, whereas Gregory et al's uses a shortest path of vertices/edges graph on the mesh. Detailed discussions appear in Section 4.2. The other difference is the parametrization of meshes. We use a harmonic mapping (Section 3.1) for the parameterization, while Gregory et al. propose an alternative parameterization approach called area-preserving mapping. Details are discussed in Section 6.1 and Section 6.2.

3 Surface Correspondence Problem

In this section, we overview a central part of the construction of the surface correspondence proposed by Kanai et al. [15]. Their basic idea is to construct the surface correspondence by combining two meshes in a common domain. First, we imbed each of the two meshes in a n-gonal region lying on an unit circle in $\mathbf{R}^2$ by harmonic mapping (Section 3.1) as the parametrization of the mesh. Second, those two n-gonal embeddings are combined with each other for the construction of surface correspondences (Section 3.2). Third, another mesh $\mathcal{F}^c$ is created for interpolation based on these surface correspondences (Section 3.3). $\mathcal{F}^c$ has the merged graph structure of $\mathcal{F}^1$ and $\mathcal{F}^2$ so that it can represent shapes of both $\mathcal{F}^1$ and $\mathcal{F}^2$ by alternating the vertex positions. Finally, by smoothly transitioning the vertex positions of $\mathcal{F}^c$ from positions of $\mathcal{F}^1$ to those of $\mathcal{F}^2$, we achieve 3D metamorphosis (Section 3.4).

3.1 Harmonic Mapping

To achieve the surface correspondence between two meshes, we prepare *embedding* $\mathcal{H} \subseteq \mathbf{R}^2$ in the two-dimensional unit circle. We map a three-dimensional topological disk $\mathcal{F} \subset \mathcal{M}$ to $\mathcal{H}$, where $\mathcal{F}$ is a subset of $\mathcal{M}$ and is homeomorphic to the topological disk. *Harmonic mapping*, $h : \mathcal{F} \mapsto \mathcal{H}$, is one of the mappings that realizes such an embedding. Note that $\mathcal{F}$ and $\mathcal{H}$ have the same graph structure of vertices, edges, and faces, and thus have a one-to-one correspondence between their vertices, edges, and faces. It is worth noting that a polyhedron $\mathcal{M}$ is usually homeomorphic to the topological sphere or torus of genus n. However, by inserting proper "cuts" to $\mathcal{M}$, we can divide $\mathcal{M}$ to a set of $\mathcal{F}$ each of which is homeomorphic to the disk. Our approach for cutting meshes is described later in Section 4.2.

To construct this mapping, we adopt a method proposed by Eck et al. [10]. Strictly speaking, Eck et al.'s mapping method is a piecewise linear approximation method for realizing embedding from $\mathcal{M}$ to $\mathcal{H}$. Our method for creating an embedding $\mathcal{H}$ from $\mathcal{F}$ is divided into two steps: (i) boundary mapping $g : \partial\mathcal{F} \rightarrow \partial\mathcal{H}$

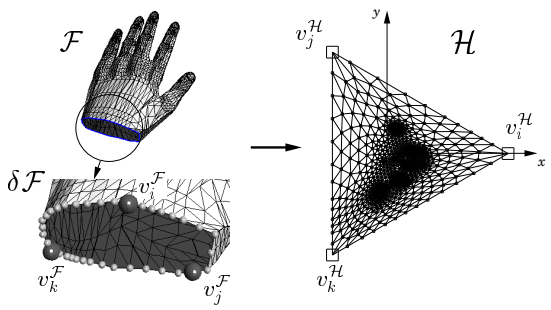


Figure 1: An embedding $\mathcal{H}$ from "hand" model $\mathcal{F}$. $\partial\mathcal{F}$, a closed boundary of $\mathcal{F}$, is composed of 36 vertices, and three of those vertices are corresponding vertices.

where $\partial\mathcal{F}$ is the boundary of $\mathcal{F}$, and $\partial\mathcal{H}$ is the boundary of $\mathcal{H}$. (ii) harmonic mapping h which maps internal vertices of $\mathcal{F}$ into $\mathcal{H}$.

At the boundary mapping step, n vertices are chosen from $\partial\mathcal{F}$ and positioned to a regular $n-$gon on the unit circle in $\mathbf{R}^2$ whose center is at the origin. We call these vertices *corresponding vertex* (CV), as discussed in Section 4.1. The rest of the vertices in $\partial\mathcal{F}$ are positioned on the edges of $n-$gon in the same order, and in such a way that the ratios of the mapped edge lengths in $\partial\mathcal{H}$ are equal to those of their original edge lengths in $\partial\mathcal{F}$.

At the harmonic mapping step, the positions of the rest of the vertices are calculated so that a total energy function E is minimized. E can be a sum of the elastic energy of springs placed along the edges of $\mathcal{H}$:

$$E = 1/2 \sum_{\{v_i,v_j\}\in\mathcal{H}} \kappa_{i,j}|v_i - v_j|^2, \quad (1)$$

$$\kappa_{i,j} = ({l_{i,k_1}}^2 + {l_{j,k_1}}^2 - {l_{i,j}}^2)/A_{i,j,k_1} + ({l_{i,k_2}}^2 + {l_{j,k_2}}^2 - {l_{i,j}}^2)/A_{i,j,k_2}, \quad (2)$$

where v_i, v_j denote 2D positions of the two end vertices of an edge $\{v_i^{\mathcal{H}}, v_j^{\mathcal{H}}\}$ in $\mathcal{H}$ and $l_{i,j}$ denotes the length of edge $\{v_i^{\mathcal{F}}, v_j^{\mathcal{F}}\}$ as measured in $\mathcal{F}$. A_{i,j,k_1}, A_{i,j,k_2} denote the areas of faces $\{v_i^{\mathcal{F}}, v_j^{\mathcal{F}}, v_{k_1}^{\mathcal{F}}\}$, $\{v_i^{\mathcal{F}}, v_j^{\mathcal{F}}, v_{k_2}^{\mathcal{F}}\}$ as measured in $\mathcal{F}$ respectively.

An unique solution that minimizes E can be found by solving a linear system $\nabla E = \mathbf{0}$, where ∇E is the gradient of E over v, because E is a positive quadratic function over v. We order x, y components of v into a $2N$ dimensional vector $\mathbf{V}$:

$$\mathbf{V} \equiv (v_1.x,\ v_1.y,\ v_2.x,\ v_2.y, \ldots,\ v_N.x,\ v_N.y), \quad (3)$$

where N denotes the number of vertices in $\mathcal{H}$. E is a quadratic function of every component in $\mathbf{V}$, so it can be represented as the quadratic form $E = \mathbf{V}^T\mathbf{H}\mathbf{V}$. Then the gradient of E can be expressed as $\nabla E = \partial E/\partial\mathbf{V}$.

As mentioned above, vertices on the boundary are fixed. To solve a linear system, we first divide a variable vector $\mathbf{V}$ into two parts: a free part $\mathbf{V}_h$ and a fixed part $\mathbf{V}_g$; the constant matrix $\mathbf{H}$ is also divided accordingly. Thus, the energy function E is rewritten as follows:

$$E = \left[\mathbf{V}_h^T \mathbf{V}_g^T\right] \begin{bmatrix} \mathbf{H}_{hh} & \mathbf{H}_{hg} \\ \mathbf{H}_{gh} & \mathbf{H}_{gg} \end{bmatrix} \begin{bmatrix} \mathbf{V}_h \\ \mathbf{V}_g \end{bmatrix}. \quad (4)$$

As this energy function is constant for a fixed part, we only have to solve a linear equation for the free parts $\mathbf{V}_h$ to minimize E:

$$\nabla E = \frac{\partial E}{\partial \mathbf{V}_h} = 2\mathbf{H}_{hh}\mathbf{V}_h + 2\mathbf{H}_{hg}\mathbf{V}_g = \mathbf{0} \quad (5)$$

Note that in $\mathbf{H}_{hh}$ a non-diagonal element at the i-th row and j-th column is non-zero only when there exists some edge connecting vertices related to the i-th and j-th column. Since there are fewer edges incident to a vertex than the total number of vertices of $\mathcal{H}$, $\mathbf{H}_{hh}$ is a sparse matrix. We use a bi-conjugate gradient method [27] to solve a sparse linear system in Equation (5). Its computation time is approximately $O(n)$.

Figure 1 illustrates an example of embedding. An original 3D "hand" model $\mathcal{F}$ ($n_v^{\mathcal{F}} = 2{,}414$, $n_f^{\mathcal{F}} = 4{,}760$) is topologically equivalent to a disk and has a closed boundary $\partial\mathcal{F}$ along its wrist. It has 36 vertices. By boundary mapping g, the three CVs $v_i^{\mathcal{F}}, v_j^{\mathcal{F}}, v_k^{\mathcal{F}}$ in $\partial\mathcal{F}$ (the big spheres in Figure 1) are located on the regular triangle as shown in the figure. The rest of the 33 vertices of $\partial\mathcal{F}$ (the small spheres in Figure 1) are positioned on the edges of the triangle. Internal vertices in $\mathcal{F}$ are mapped into the triangle by harmonic mapping h. The result shows that fingers which include concave regions develop into a disk without self-intersections. The computation time for this example is about five seconds on MIPS R4400, 250MHz.

3.2 Combining Two Embeddings

We combine two embeddings $\mathcal{H}^1$ and $\mathcal{H}^2$ to generate *a combined embedding* $\mathcal{H}^c$ as shown in Figure 2. $\mathcal{H}^c$ has a combined graph structure of $\mathcal{H}^1$ and $\mathcal{H}^2$. Surface correspondence between $\mathcal{H}^1$ and $\mathcal{H}^2$ and thus between $\mathcal{F}^1$ and $\mathcal{F}^2$ can be established by using $\mathcal{H}^c$.

An algorithm for generating $\mathcal{H}^c$ consists of the following four steps:

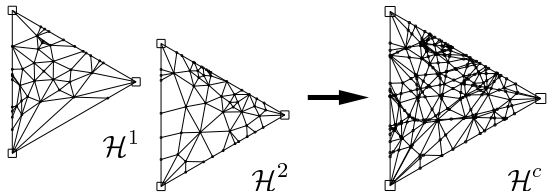


Figure 2: By combining two embeddings, $\mathcal{H}^c$ is generated. It has a combined graph structure of $\mathcal{H}^1$ and $\mathcal{H}^2$.

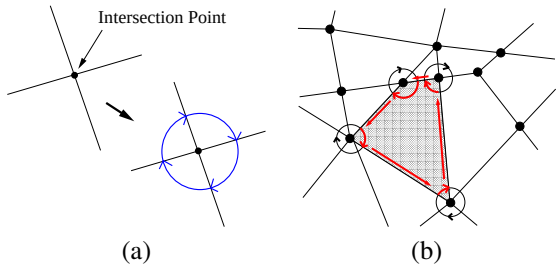


Figure 3: (a) An edge cycle at an intersection point is created by four intersecting edges. (b) Faces of $\mathcal{H}^c$ are created by vertices, edges, edge cycles along a vertex in $\mathcal{H}^c$.

STEP1 investigate intersection between edges of $\mathcal{H}^1$ and $\mathcal{H}^2$.

STEP2 sort intersection points to split edges of $\mathcal{H}^1$ and $\mathcal{H}^2$ to generate vertices and edges of $\mathcal{H}^c$.

STEP3 create a clockwise edge cycle at each vertex in $\mathcal{H}^c$.

STEP4 create faces of $\mathcal{H}^c$.

We use Kent et al.'s algorithms [16] for implementing **STEP1** and **STEP2**. **STEP2** utilizes topological information generated in **STEP1**. When an intersection point is found in **STEP1**, we obtain information about which adjacent faces the intersecting edge crosses. We record this information for sorting intersection points along edges to be split.

In **STEP2**, the vertices and edges of $\mathcal{H}^c$ are generated. The vertices of $\mathcal{H}^c$ consist of the vertices of $\mathcal{H}^1$ and $\mathcal{H}^2$ (*original type*) and intersection points between the edges of $\mathcal{H}^1$ and $\mathcal{H}^2$ (*split type*). The edges of $\mathcal{H}^c$ consist of the edges of $\mathcal{H}^1$ and $\mathcal{H}^2$, some of which are subdivided by intersection points.

At **STEP3**, a clockwise edge cycle at each vertex of $\mathcal{H}^c$ is generated for creating faces in **STEP4**. This is carried out according to the vertex type, *original* or *split*. If a vertex is original, its edge cycle is inherited from its original edge cycle of $\mathcal{H}^1$ or $\mathcal{H}^2$. If a vertex is split, its edge cycle is generated for intersection edges (Figure 3(a)) by using links to the four intersecting edges. These links are set to the vertices at **STEP1**.

In **STEP4**, faces of $\mathcal{H}^c$ are created with those vertices, edges, and clockwise edge cycles (Figure 3(b)). Faces of $\mathcal{H}^c$ are created by the following traversal method: We start from some vertex of $\mathcal{H}^c$ and traverse one of its incident edges. Then we arrive at the other vertex of that edge and choose the next edge connected counter-clockwise by using the edge cycle information. These steps are repeated until we arrive at the start vertex. Consequently, a counter-clockwise vertex loop is identified to generate a face. These faces created by this method may not be triangular faces. Therefore, we finally triangulate those faces. The subdivision is straightforward because the faces are all convex.

3.3 Interpolation Mesh

For an interpolation between $\mathcal{F}^1$ and $\mathcal{F}^2$, it is not enough to create $\mathcal{H}^c$. We need a mesh for interpolating between the two meshes. *Interpolation mesh* $\mathcal{F}^c$ has the same graph structure as $\mathcal{H}^c$. We first determine the geometric realization of $\mathcal{F}^c$ for $\mathcal{F}^2$. We denote it $\mathcal{F}^{c2}$. The idea here is to let $\mathcal{F}^c$ take on the same shape as $\mathcal{F}^2$. While the graph structure of $\mathcal{F}^{c2}$ are different from that of $\mathcal{F}^2$, the total shape of $\mathcal{F}^{c2}$ can be made the same as $\mathcal{F}^2$. We also define $\mathcal{F}^{c1}$, which is the geometric realization of $\mathcal{F}^c$ for $\mathcal{F}^1$.

To do this, $\mathbf{v}^{\mathcal{F}^{c2}}$, the 3D coordinates of $\mathcal{F}^c$ in $\mathcal{F}^2$ must be determined. Some of the vertices of $\mathcal{F}^c$ come from $\mathcal{F}^2$. For those vertices, their positions are defined as the original positions in $\mathcal{F}^2$. The other vertices either (i) come from $\mathcal{F}^1$ or (ii) are generated by edge intersection in **STEP1**. For those vertices we have to compute the 3D coordinates. As for the vertices of type (i), each of these vertices corresponds to a vertex in $\mathcal{F}^1$, which is mapped to a $v^{\mathcal{H}^1}$ in $\mathcal{H}^1$. Since $\mathcal{H}^1$ and $\mathcal{H}^2$ are just overlapping, there exists a triangle face of $\mathcal{H}^2$ involving $v^{\mathcal{H}^1}$. We compute the barycentric coordinates of $v^{\mathcal{H}^1}$ in this triangle.

In Figure 4, $f^{\mathcal{H}^2} = \{v_\alpha^{\mathcal{H}^2}, v_\beta^{\mathcal{H}^2}, v_\gamma^{\mathcal{H}^2}\}$ denotes a face involves a vertex $v^{\mathcal{H}^1}$. (α, β, γ) $(\alpha+\beta+\gamma=1)$ denotes its barycentric coordinate. $\mathbf{v}_\alpha^{\mathcal{F}^2}, \mathbf{v}_\beta^{\mathcal{F}^2}, \mathbf{v}_\gamma^{\mathcal{F}^2}$ denote corresponding 3D coordinates of vertices $v_\alpha^{\mathcal{H}^2}, v_\beta^{\mathcal{H}^2}, v_\gamma^{\mathcal{H}^2}$ in

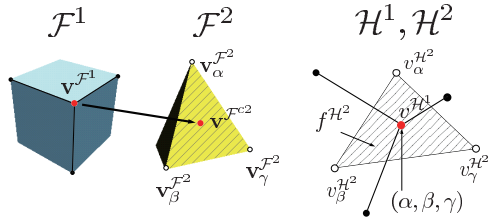


Figure 4: Calculate 3D coordinates of vertices in $\mathcal{F}^c$ from the barycentric coordinates in the face involving a vertex.

$\mathcal{F}^2$. 3D coordinates $\mathbf{v}^{\mathcal{F}^{c2}}$ for $\mathbf{v}^{\mathcal{F}^1}$ is calculated by the following equation:

$$\mathbf{v}^{\mathcal{F}^{c2}} = \alpha \mathbf{v}_\alpha^{\mathcal{F}^2} + \beta \mathbf{v}_\beta^{\mathcal{F}^2} + \gamma \mathbf{v}_\gamma^{\mathcal{F}^2}. \tag{6}$$

For each vertex of type (ii), its $\mathbf{v}^{\mathcal{F}^{c2}}$ is also obtained from the barycentric coordinates of its corresponding $\mathbf{v}^{\mathcal{H}^c}$ in a similar manner as type (i). $\mathbf{v}^{\mathcal{F}^{c1}}$ can be computed in the same way as $\mathbf{v}^{\mathcal{F}^{c2}}$.

3.4 Metamorphosis over $\mathcal{F}^c$

The major result obtained in the previous subsection is that we can now represent two different shapes of $\mathcal{F}^1$ and $\mathcal{F}^2$ in single, another mesh structure of $\mathcal{F}^c$. While $\mathcal{F}^{c1}$ and $\mathcal{F}^{c2}$ have the same shape as $\mathcal{F}^1$ and $\mathcal{F}^2$ respectively, they have the same graph structure as $\mathcal{F}^c$. So by interpolating vertex positions from $\mathbf{v}^{\mathcal{F}^{c1}}$ to $\mathbf{v}^{\mathcal{F}^{c2}}$, $\mathcal{F}^{c1}$ gradually changes its shape to $\mathcal{F}^{c2}$. 3D metamorphosis from $\mathcal{F}^1$ to $\mathcal{F}^2$ is realized by applying a simple linear interpolation method to the vertex positions. For *interpolation parameter* $t (0 \leq t \leq 1)$, $\mathbf{v}^{\mathcal{F}^c}(t)$ can be computed by using two 3D coordinates $\mathbf{v}^{\mathcal{F}^{c1}}(t), \mathbf{v}^{\mathcal{F}^{c2}}(t)$ as follows:

$$\mathbf{v}^{\mathcal{F}^c}(t) = (1-t)\, \mathbf{v}^{\mathcal{F}^{c1}} + t \mathbf{v}^{\mathcal{F}^{c2}}. \tag{7}$$

An intermediate shape at t is represented by $\mathcal{F}^c$ with the geometric realization $\mathbf{v}^{\mathcal{F}^c}(t)$.

4 Surface Correspondence with User Control

In this section, we extend the method for establishing surface correspondence between two meshes $\mathcal{M}^1$ and $\mathcal{M}^2$ discussed in the previous section, so that (minimum) user control of correspondence can be achieved.

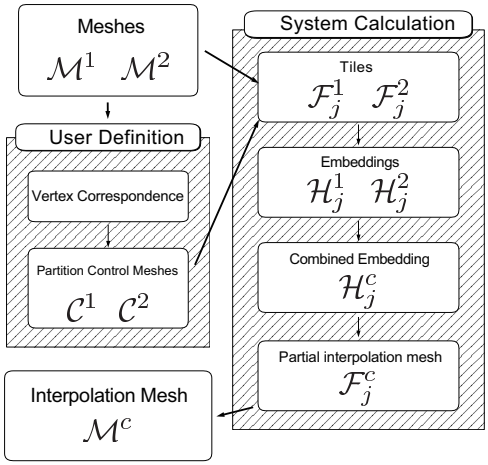


Figure 5: An overview of our method for 3D metamorphosis between two meshes $\mathcal{M}^1$ and $\mathcal{M}^2$ with user control of surface correspondence.

Though we can think of many kinds of user control, we focus on control on *vertex correspondence*. The vertex correspondence is a set of pairs of vertices, one from the source mesh and the other from the target. This pair is named *corresponding vertex pair* (CVP). CVP constrains so that the source vertex is transferred to the target vertex. It can be regarded as the most primitive control method.

One straightforward way to incorporate such constraints is to add constraint equations to Equation (5). For instance, in order for $\mathbf{v}^{\mathcal{F}^1}$ to correspond with $\mathbf{v}^{\mathcal{F}^2}$, their images of harmonic mapping, $v^{\mathcal{H}^1}$ and $v^{\mathcal{H}^2}$ must be coincident. By adding the contraint equation of $v^{\mathcal{H}^1} = v^{\mathcal{H}^2}$ to Equation (5), the vertex correspondence could be achieved. Unfortunately, from our experiments, we found that this method does not work well. If we solve Equation (5) with the condition that several internal vertices of $\mathcal{H}$ are on the fixed constraint, an undesirable self-intersection usually occurs along fixed vertices.

Instead we would make efforts to avoid those ill-conditions, we take an another approach. We divide the mesh into several regions named *tile* and establish surface correspondence per tile. By allocating such corresponding vertices to the boundaries of those tiles, the vertex correspondence can be automatically satisfied.

An overview of our proposed method is shown in Figure 5. The process consists of *user definition process* and *system calculation process*. In the user definition process, the user specifies *corresponding vertex (*CV) for the control of a desired interpolation. Using CVs, the user also defines *partition control mesh* (PCM) $\mathcal{C}^1$ and $\mathcal{C}^2$ (Section 4.1). In the system calculation process, each of the two meshes $\mathcal{M}^1$ and $\mathcal{M}^2$ is partitioned by grouping its faces according to $\mathcal{C}^1$ and $\mathcal{C}^2$ to obtain $\hat{\mathcal{M}}^1$ and $\hat{\mathcal{M}}^2$ (Section 4.2):

$$\hat{\mathcal{M}}^1 = \bigcup_{j}^{n_f^{\mathcal{C}}} \mathcal{F}_j^1, \quad \hat{\mathcal{M}}^2 = \bigcup_{j}^{n_f^{\mathcal{C}}} \mathcal{F}_j^2. \tag{8}$$

The method discussed in Section 3 is applied to each pair of $\mathcal{F}_j^1 \subseteq \mathcal{M}^1$ and $\mathcal{F}_j^2 \subseteq \mathcal{M}^2$, an interpolation mesh $\mathcal{M}^c$ is created by merging *partial interpolation meshes* $\mathcal{F}_j^c$ (Section 4.3).

4.1 Partition Control Mesh

Figure 10 shows basic elements of this approach. As shown in Figure 10(a), the user first selects pairs of corresponding vertices (CV), one from each of the two meshes $\mathcal{M}^1$ and $\mathcal{M}^2$ to define the corresponding vertex pair (CVP). Next, as shown in Figure 10(b), the user creates partition control meshes (PCM) $\mathcal{C}^1$ and $\mathcal{C}^2$ by connecting CVs. By sequentially picking CVs to form a vertex loop, faces of the meshes are defined. The faces are not necessarily triangular. The directions of their vertex loops must be the same as those of faces on the original meshes (in a counter-clockwise direction in our paper). $\mathcal{C}^1$ ($\mathcal{C}^2$) is designed so as to be topologically equivalent to its original mesh $\mathcal{M}^1$ ($\mathcal{M}^2$). $\mathcal{C}^1$ and $\mathcal{C}^2$ must have the same graph structure, but only 3D coordinates of the vertices can be different. As shown in Figure 10(c),(f), according to $\mathcal{C}^1$ ($\mathcal{C}^2$), the mesh $\mathcal{M}^1$ ($\mathcal{M}^2$) is partitioned into tiles. For each face of $\mathcal{C}^1$ ($\mathcal{C}^2$), one tile is defined.

How much control of surface correspondence is necessary depends on the number of CVPs. The minimum number of such CVPs is determined by the parameterization we adopt. For the mesh topologically equivalent to a sphere, at least two faces are needed in a PCM to divide a topological sphere into two topological disks. As the minimum component of such a disk is a triangle, then at least three CVPs are needed per a face of a PCM. If the number of CVPs is only three, the user must define two faces which have same three vertices. In our current implementation, we put an additional restriction that more than four CVPs are needed to define a tetrahedral PCM because such a definition seems to be useful for the user to understand a PCM intuitively.

4.2 Partition of Meshes

According to the topology of partition control meshes $\mathcal{C}^1$ and $\mathcal{C}^2$, $\mathcal{M}^1$ and $\mathcal{M}^2$ are automatically divided into tiles $\mathcal{F}_j^1 \subseteq \mathcal{M}^1$ and $\mathcal{F}_j^2 \subseteq \mathcal{M}^2 (j = 1 \ldots n^{\mathcal{F}}$, where $n_f^{\mathcal{F}} = n_f^{\mathcal{C}})$ respectively. The method for partitioning a mesh consists of the following three steps; First, each edge of $\mathcal{C}$ is projected onto its original mesh $\mathcal{M}$ to generate a path on the mesh so that its two end points are CVs. Second, each face of $\mathcal{M}$ over which those paths cross is subdivided into triangular faces. Finally, a mesh grouping algorithm is applied to gather faces of $\mathcal{M}$ as a tile.

Projecting edges of $\mathcal{C}$ to $\mathcal{M}$

One approach to 'cut' the mesh, that is, to project edges of $\mathcal{C}$ onto $\mathcal{M}$, is to use a vertex/edge graph of $\mathcal{M}$ and to calculate a shortest path on a graph as described in [12]. However, we found from our experiment that a set of such paths composed by original vertices and edges of $\mathcal{M}$ causes some problems. First, it usually occurs that two close paths are overlapped each other. Moreover, it is quite a hard task for the user to specify vertices so as not to overlap. Especially, the finer we create a PCM, the more frequent it occurs.

We regard the projecting edges of $\mathcal{C}$ onto $\mathcal{M}$ as a calculation of the exact shortest path (it is also called *geodesic curve*) between two vertices of $\mathcal{M}$. However, in general, finding the shortest path between two vertices on a mesh is a difficult problem[25]. Chen et al. [4] have proposed a method for an exact solution of this problem in $O(n^2)$ time. Their method is based on transforming faces of meshes onto a plane. As 3D rotations are used, numerical errors for the paths or geometric structures can occur. Instead, we use a method for calculating an approximate shortest path between two vertices[18]. Before the calculation, *intermediate edges* are created on the faces of $\mathcal{M}$ as follows: first, several intermediate vertices (called *Steiner Point*) are created on the edges of $\mathcal{M}$. The number of added vertices per an edge is determined according to its length so that more vertices are added to a longer edge.

Now each face $f_i^{\mathcal{M}} (i = 1 \ldots n_f^{\mathcal{M}})$ of $\mathcal{M}$ has intermediate vertices on its edges in addition to the original vertices. From the total set of these vertices, we choose

every pair of vertices $\langle v_a, v_b \rangle$ and create an intermediate edge for the face if the pair satisfies either of two conditions:

- v_a and v_b are on the different edges.
- v_a and v_b are on the same edge and adjacent.

Approximate shortest paths are calculated based on a graph which is composed of vertices and edges of $\mathcal{M}$ and intermediate vertices and edges using Dijkstra's algorithm [1]. The computation time for this method is approximately $O(n \log n)$.

This method does not compute the shortest path exactly, but the difference is too small to recognize visually. Moreover, compare to Chen et al.'s method that calculates real paths, this method is not only fast but reduces the accumulation of numerical errors caused from frequent 3D rotation computations. Figure 10(c),(f) show the calculated shortest paths.

It is applicable for all approaches (ours is also included) that 'cutting' of the mesh based on calculating shortest paths do not always hold good for all situations. For example, extremely speaking, it is clear that the number of shortest paths between a north pole and a south pole of a sphere is infinite. It can occur that a path far from desirable one for the user is generated, if the geodesic distance between user-selected two vertices of a mesh is long. In our implementation, if an undesirable path is generated, such inconveniences are avoided by specifying more CVPs along failure paths so that shorter distance paths are recomputed.

Subdivision of Faces in $\mathcal{M}$

The faces which are determined to be crossed by these shortest paths are then subdivided into triangular faces (Figure 6(a)). Such faces can be classified into three subdivision patterns as shown in Figure 6(b). Subdivision is needed for the case (ii) and (iii). Subdivided faces are not always triangular faces, and those faces are triangulated. The triangulation is trivial because those faces are planar convex quadrilaterals. The above process is repeated for each of the edges of $\mathcal{C}$. At each repetition, the intermediate edges of those subdivided faces are updated. After the process is finished, the intermediate edges and vertices which do not contribute to the paths are removed.

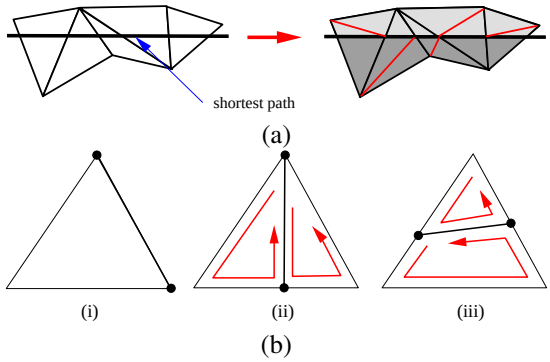


Figure 6: (a) Subdivision of faces of $\mathcal{M}$ on a shortest path. (b) Three patterns for a face subdivision.

Mesh grouping algorithm

After inserting all the shortest paths according to the partition control mesh and subdividing/triangulating faces, we define the tiles bounded by some number of shortest paths. A tile is a set of continuous faces and is thus obtained by grouping faces. Here we describe our mesh grouping algorithm.

Before applying the mesh grouping algorithm, we have to prepare some data structures. We represent a mesh $\mathcal{M}$ with a half-edge data structure [24] (Figure 7(a)). In addition to this structure, we also define a data structure for edges of the shortest path, we call them *path edges*, as shown in Figure 7(b). We define *edge direction* as the direction from a start vertex to an end vertex. It also has two links to *left_halfedge* and to *right_halfedge* based on this direction. In addition, we define a link from a path edge to each corresponding half-edge. As shown in Figure 7(c), a shortest path is generated for an edge of $\mathcal{C}$. And for a face of $\mathcal{C}$ we can think of a loop of edges which turns around a tile in a counter-clockwise direction. Referring to these edge directions, the directions of the shortest paths can be determined. And we set a direction for each path edge to the same direction as the direction of the path. Then, the left half-edges of those path edges are subject to grouping.

Figure 8 shows a pseudo-code of our mesh grouping algorithm GroupingMesh. This algorithm can partition faces into tiles and put a certain identifier `id` to the faces in the same tile.

In GroupingMesh, the search of grouping faces begins from the left face of a path edge in a tile. For this face, recursive function GroupingFace is executed.

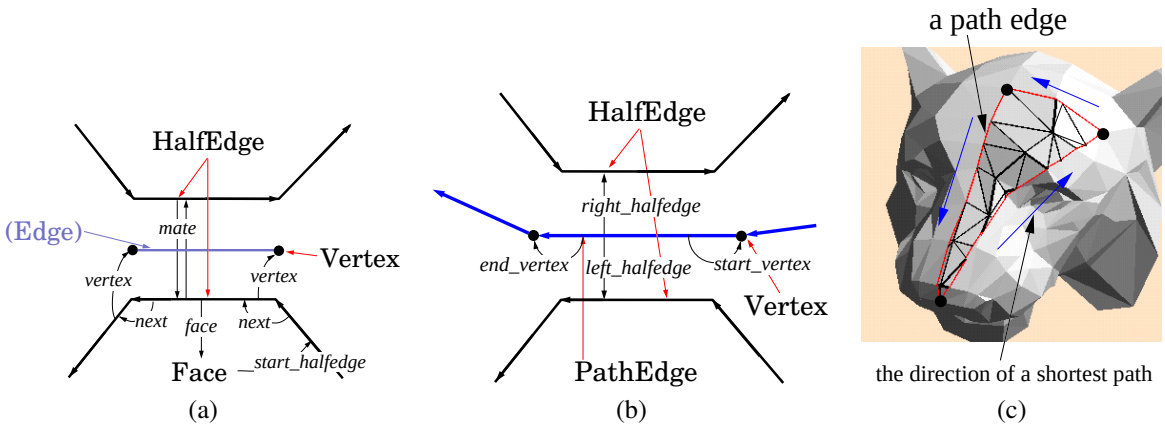


Figure 7: (a) Half-edge Data Structure. (b) Data structure of a path edge for grouping meshes. (b) Create directions of path edges along a tile.

GroupingFace traces the half-edges of a face. GroupingFace is recursively executed to a face which the mate of a half-edge has. The traced face is marked `id`. This algorithm totally traces all faces in $\mathcal{H}$ once. At the worst case, all half-edges of each face are traced. However, the number of half-edges in a face is always three because a face is a triangle. It is quite smaller than the number of faces of a mesh in general. Thus, the execution time of this greedy-like algorithm is approximately $O(n)$.

In practice, the partitioning of $\mathcal{M}^1$ and $\mathcal{M}^2$ is performed separately. The faces of $\mathcal{M}^1$ and $\mathcal{M}^2$ which have the same `id` are stored as a tile $\mathcal{F}^1_{\mathtt{id}}$ and $\mathcal{F}^2_{\mathtt{id}}$ respectively. Vertices and edges which consist of faces of a tile are also stored together. Vertices which consist of CVP and path edges belong to more than two tiles.

4.3 Generating a Partial Interpolation Mesh

For each pair of tiles $\mathcal{F}^1_j$ and $\mathcal{F}^2_j$ of partitioned mesh $\hat{\mathcal{M}}^1$ and $\hat{\mathcal{M}}^2$, we apply our method described in Section 3 (Figure 9). First, embeddings $\mathcal{H}^1_j$ and $\mathcal{H}^2_j$ are created by developing $\mathcal{F}^1_j$ and $\mathcal{F}^2_j$ into the 2D unit circle, respectively. $\partial\mathcal{F}^1_j$ and $\partial\mathcal{F}^2_j$ are mapped onto the edges of the $n-$gonal region on the unit circle. CVs in $\partial\mathcal{F}^1_j$ and $\partial\mathcal{F}^2_j$ are mapped into the same vertices of the $n-$gon. In Figure 9, CVs on the meshes are mapped to the vertices on the embeddings. Next, those two embeddings are combined, and a combined embedding $\mathcal{H}^c_j$ is created. Third, a partial interpolation mesh $\mathcal{F}^c_j$ is created. 3D coordinates of vertices in $\mathcal{F}^c_j$ for $\mathcal{F}^1_j$ and $\mathcal{F}^2_j$ are computed from barycentric coordinates of the face which involves the vertex of $\mathcal{H}^1_j$ or $\mathcal{H}^2_j$. Finally, the total interpolation mesh $\mathcal{M}^c$ is created by merging all the $\mathcal{F}^c_j$. CVs and path edges belonging to several partitions are unified. 3D metamorphosis is made using $\mathcal{M}^c$ as described in Section 3.4.

4.4 Surface Correspondence for Meshes with Boundaries

The method proposed in our paper is applicable to the mesh $\mathcal{M}$ which has closed boundaries $\partial\mathcal{M}$ provided that we shall define proper CVPs, partition control meshes, and shortest paths.

The user must create more than three CVs on a closed boundary and also create an edge loop of $\mathcal{C}$ using those vertices along the boundary. Moreover, when the edges of $\mathcal{C}$, both of whose end vertices are composed of CVs on a closed boundary, are projected to $\mathcal{M}$, these paths are composed of part of the boundary edges. To create those paths, we don't need to use the method described in Section 4.2. We simply search the boundary edges of $\mathcal{M}$ to create such paths.

5 Results

We have implemented a prototype system on a graphics workstation SGI Indigo^2 High Impact (MIPS R4400

Examples		Fig.12(a)	Fig.12(b)	Fig.13	Fig.14	Fig.15
$\mathcal{M}^1$	$n_v^{\mathcal{M}^1}$	188	188	400	5,534	480
	$n_f^{\mathcal{M}^1}$	372	372	738	11,064	960
$\mathcal{M}^2$	$n_v^{\mathcal{M}^2}$	254	254	1,726	12,344	474
	$n_f^{\mathcal{M}^2}$	504	504	3,311	24,684	948
$\mathcal{C}$	$n_v^{\mathcal{C}}$	5	16	40	28	17
	$n_e^{\mathcal{C}}$	9	38	68	55	35
	$n_f^{\mathcal{C}}$	6	24	29	29	18
$\mathcal{M}^c$	$n_v^{\mathcal{M}^c}$	1,865	2,320	9,724	77,269	4,965
	$n_f^{\mathcal{M}^c}$	3,726	4,636	19,265	154,534	9,930
Time (Sec.)	T_s	24.6	104.4	99.7	169.23	109.1
	T_h	0.1	0.1	0.7	11.95	0.2
	T_c	0.4	0.5	2.1	18.77	1.3

Table 1: Statistics for 3D metamorphosis examples. T_s, T_h, and T_c denote the calculation of shortest paths, of embeddings, and of combining two embeddings, respectively. All of the time data were collected on MIPS R4400, 250MHz.

250MHz CPU, 128MB memory). We have examined several different examples using our proposed method for 3D metamorphosis. Table 1 summarizes the number of vertices, faces, and three computation times for those examples.

First, we used two different controls of 3D metamorphosis between the "bunny's head" model and "tiger's head" model, as shown in Figure 10(a). These two meshes are topologically equivalent to a sphere. Figure 10(a)-(c) shows an example of the control with only 5 CVPs and a partition control mesh which has 6 faces. Figure 12(a) illustrates the results where in the middle of an interpolation, the tiger's left ear is grown in the middle of the bunny's ear! An insufficient control causes such a poor result. Figure 10(d)-(f) show an example of finer control using 16 CVPs and a partition control mesh of 24 faces. To interpolate between the bunny's ear and the tiger's ear, 4 CVPs are specified around the base of both ears. Figure 12(b) illustrates the results where a more sophisticated control of the interpolation is achieved.

Next, we examined a 3D metamorphosis between two car bodies, from the "Delorean" model to the "Porsche 911 Carrera" model, as shown in Figure 11(a). These two meshes are topologically equivalent to a disk. Figure 11(b) shows an example of the control with 40 CVPs and a partition control mesh of 29 faces. Figure 13(a) illustrates the result of the metamorphosis. The tires are not included in the interpolation objects, being translated and scaled by offline editing. Figure 13(b)-(c) show a magnification of the results of the metamorphosis of the right headlight and the right door-mirror. For an interpolation of both headlights, 4 CVs are specified at the rims of the shapes. Figure 13(b) shows that the metamorphosis between different shapes is successfully interpolated. For the interpolation between the door-mirror and the flat shape, 4 CVs are specified at the root of the door-mirror and at the corners of a rectangular area of the other flat shape. Figure 13(c) shows that the door-mirror grows from a flat shape.

For testing the robustness of our approach, we examined a metamorphosis between two large meshes. Figure 14 (a) shows "star" model and "pai" (a karate fighter) model. Both models are topologically equivalent to a sphere, and have over 10,000 polygons. "star" model is created by 3D modeler, "pai" model is reconstructed from range images scanned by 3D digitizer. Figure 14 (b) denotes an example of the control by 28 CVPs and 29 faces of a PCM which involves non-triangular faces. Figure 14 (c) shows the result of the metamorphosis.

Above three examples are for the case where the genus of a mesh equals to zero ($g = 0$), that is, topologically equivalent to a sphere (which may include boundaries). But it is also possible to show the result of the metamorphosis between the "torus" model and the "bottle" model. The genus of these two models equals to 1 ($g = 1$). We defined a partition control

```
void GroupingMesh (Mesh M, Id id)
{
    e = one of path edges of group id;
    f = e.left_halfedge.face;
    GroupingFace ( f, id );
}

void GroupingFace (Face f, Id id)
{
    f.group_id = id;
    he = f.start_halfedge ;
    do {
        if ( (he.mate ≠ nil)
            && (he.pathedge == nil)) {
            nf = he.mate.face ;
            if ( nf.group_id ≠ id )
                GroupingFace ( nf, id );
        }
        he = he.next ;
    } while ( he ≠ f.start_halfedge ) ;
}
```

Figure 8: Pseudo-code of the mesh grouping algorithm.

mesh which has 17 CVPs and 18 faces. In addition to the homeomorphism condition, those two models must be "*tamely homeomorphic*"[3]. $\mathcal{M}^1$ is tamely homeomorphic to $\mathcal{M}^2$ if there is a homeomorphism of E^3 onto itself that carries $\mathcal{M}^1$ onto $\mathcal{M}^2$. This condition is related to "knots" of the mesh. Moreover we have to define proper partition control meshes. Although we can not at this moment demonstrate the complete condition of the proper partition control meshes for transforming $g \neq 0$ objects, we can say, at least from Figure 15, that the partition control mesh must cover "holes" of the object.

6 Discussions

6.1 Robustness

One part of our approach that the robustness should be considered is the generation of embeddings (Section 3.1). As described in [10], the parameterization based on Harmonic Mapping for generating embeddings does not occasionally work well if the number of faces are large. In our implementation, the number of faces to be computed for the parameterization is much smaller than

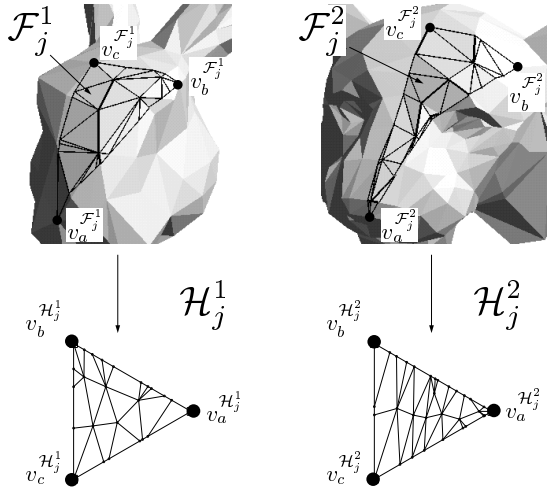


Figure 9: Creating embeddings of tiles $\mathcal{F}_j^1$ and $\mathcal{F}_j^2$.

that of the original mesh, because it is partitioned to a set of tiles by a PCM. In Figure 14, the average number of faces in each tile is approximately 1,000, the maximum is less than 3,000. In the actual calculation, that is, $29 \times 2 = 58$ times computation for generating embeddings, a self-intersection occurs at only a tile which has the largest number of faces. In our current approach, if such a self-intersection occurs, it is checked before proceeding the next step, and is avoided by recomputing an embedding with that $\kappa_{i,j}$ in Equation (2) is set to be uniform constant. It produces a good result as described in [10] [1].

The other geometric algorithm in our method that the robustness should be taken care of is the combination of two embeddings (Section 3.2). Kent et al.[16] point out that for large meshes small numerical inaccuracies in the geometric computations such as line-to-line intersections cause the failure of the construction of the combined embedding. Our algorithm described in Section 3.2 only need to judge an intersection between two edges in the 2D unit circle (**STEP1**) essential for the combined embedding construction, while Kent et al.'s construction algorithm needs to judge intersections between two arcs. We think that it is also possible to apply an exact-arithmetic approach such as [29]. But even in our implementation for combining embeddings using double floating point calculations, such numerical

[1] Recently, [9, 20] propose more robust and fast methods with regard to the parameterization of the mesh such as Harmonic Mapping.

inconsistencies do not occur for all of our examples.

6.2 Quality

Seen from the view of the correspondence, we can divide the main component of our method which have significant influences on the quality of the metamorphosis into two different levels; one is user-specified vertex correspondence level, and the other is the surface correspondence level which is largely dependent on the parameterization.

In the vertex correspondence level, the quality of correspondences can be improved by adding CVPs, or by rearranging CVs on the mesh suitably. It can be noted from our experiment that a rough correspondence between each feature part of two original meshes can be established by specifying only several (three to four) vertices along the boundary of such a feature part (the base of the ear, the shoulder joint, etc.). If the user wishes more sophisticated correspondences, additional CVPs should be added in the internal region of a feature part (the top of the ear, an elbow joint, the wrist, etc.), or along the boundary of a feature part. However, it is also noted that too many CVPs densely specified in a certain region do not improve the quality in the least, but only require an extra time-consuming work. Moreover, it seems that a larger number of CVPs are not always needed for a larger number of faces in its original mesh, if anything, that the number of features in one mesh mainly determines the number of CVPs. The more complicated shapes, the larger number of CVPs is needed to establish a high-quality correspondence.

Our current strategy of the PCM creation is as follows; first, the user creates a rough PCM as small number of faces as possible. Next, a metamorphosis result by a current PCM is displayed. CVPs are added along a part of a PCM which causes a visually poor interpolation, and reconstruct a part of a PCM along added CVPs. These operations are repeated until the user can achieve a satisfactory result.

In the surface correspondence level, the parameterization based on Harmonic Mapping has the property that minimizes the metric distortion of a triangle or preserves the aspect ratio of a triangle, but that does not preserve the area. Because of this, it can occur that a large number of triangles are crowded at a narrow region (for example fingers in Figure 1). We recognize that his property causes an bad effect that the surface correspondence of a tile is tends to be unbalanced within a paticular region(Gregory et al. propose an alternative parameterization approach called area-preserving mapping[12]). In our method, such an unbalance of the parameterization can be avoided by specifying additional CVPs into the corresponding region. For the shape in Figure 1, not only specifying CVPs onto the base of a finger, but to specify onto a second joint, or onto the top, are the better way to establish an uniformly sampling surface correspondence.

6.3 Performances

For the user, the performance of specifying CVPs or of creating a PCM largely depends on that of the graphics display and thus on the number of faces in the original mesh. the specification of CVPs itself is not quite hard task because all the user must work is to pick vertices on the mesh. Moreover, the user sequentially picks CVPs to create a PCM with looking at the original mesh, the creation of a PCM is not either a hard task. The user only need to create a PCM for one of two original meshes, because a vertex of a PCM (CVP) is corresponded to each vertex of two meshes. In our experiment with the prototype system, it takes about several minutes to create a PCM in Figure 10 (b) and (c). Even a PCM in Figure 14 (b) for large meshes can be created within 30 minutes. In this example, a bottleneck is time for displaying the original mesh.

From the user's point of view, there are two issues which should be taken into consideration whether our approach for creating a PCM is a really useful approach for the user or not.

First, it maybe seem to be a rather cumbersome work that the user must specify not only CVPs but a PCM at the same time. We think, however, that it is case by case whether this work needs. As described in Section 4.1, the creation of a PCM plays an important role for the grouping of meshes used in the surface correspondence. In the case that the user wants to establish the surface correspondence between a feature region of one mesh to a region of the other, the creation of a face in PCM is equivalent to a clear specification of the surface correspondence between two (rough) regions of the feature. We also think that it is a natural and an important process for the user to understand the surface correspondence intuitively. If such a clear specification as a region unit is not needed, a method for creating such a region automatically from a curve net proposed by Gregory et al.[12] seems to be useful for reducing tasks of the user.

Second, it maybe seem to be a difficult task to select

vertices on each mesh so that two sets of sub-meshes divided by a PCM are topologically equivalent. One can say that the more CVPs are, the clearer this difficulty appears. Even if it was so, such a difficulty could be improved to some extents by specifying a certain constraint in our strategy for creating a PCM. In our strategy described in Section 6.2, once a rough (a little CVPs) PCM is created, a constraint can be added for selecting a vertex of $\mathcal{M}$ in the next addition of CVPs as follows; when a vertex on a sub-mesh in one mesh is selected, a corresponding vertex selected by the user must be on a corresponding sub-mesh in the other mesh. For the user, this constraint can be a guideline to select a vertex easily for creating a CVP. On the other hand, the process to create a rough PCM is a burden for the user, while an invention, for example a rough PCM begins at a tetrahedral, could be a help. This issue is one of future works.

7 Conclusions and Future Work

We have introduced an efficient framework for making 3D metamorphosis with user control of correspondences, and have presented an algorithm for partitioning arbitrary meshes in order to utilize our method for establishing surface correspondences based on harmonic mapping. We have demonstrated a 3D metamorphosis between two topologically equivalent arbitrary meshes which include geometrically complicated shapes.

We believe that the following future works is essential for improving the quality of these metamorphoses.

In our method, the number of faces of $\mathcal{M}^c$ becomes very large. Therefore, it is slow in displaying animation during metamorphosis. To establish real-time animation of metamorphosis, a method for decreasing the number of faces without losing the surface features of the original objects is needed.

We have primarily discussed the correspondence problem, but we have not spent much time on the interpolation problem. We noticed that our linear interpolation method can cause some problems (self intersection, shape distortion, etc.). Some approaches address this problem ([28] for 2D polygon cases, [30] for 3D polyhedral polygon cases). Unfortunately, these are useful only for cases where two models have quite similar shapes. We hope to find an interpolation algorithm that can work well with our correspondence algorithm. We also think that it is quite difficult to avoid self-intersections during the animation in any case. However, it seems to be possible to reduce or make them unremarkable.

In this paper, we partition the mesh into tiles and apply the interpolation uniformly to all the tiles. We are now extending the mesh to make non-uniform interpolation by which each tile is interpolated individually. This gives us more flexibility for deforming object's shapes.

Acknowledgements

A part of this research was supported by The Ookawa Foundation for Information and Telecommunication. The "bunny" mesh is from the Stanford University Computer Graphics Laboratory. The "tiger", "Delorean", "Porsche 911 Carrera", and "bottle" models are courtesy of Viewpoint DataLabs. We wish to thank Tetsuya Yamamoto and Jun Mitani, graduate students, the University of Tokyo, for help in creating color results. We would like to thank Kinji Odaka and members of the development division, NABLA Inc. Japan, for their useful comments. We would also like to thank all the reviewers for making us a aware of references and giving us a lot of useful comments.

Note for readers

Some additional materials (images, movies and references) are available via the internet:
http://www.riken.go.jp/lab-www/mat-fab/personal/kanai/Gmorph.html

References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *Data structures and algorithms*. Addison-Wesley, Reading, Massachusetts, 1983.

[2] T. Beier and S. Neely. Feature-based image metamorphosis. In *Computer Graphics (Proc. SIGGRAPH 92)*, pages 35–42. ACM Press, New York, 1992.

[3] M. Boyer and N. F. Stewart. Modeling spaces for toleranced objects. *Int. J. Robotics Research*, 10(5):570–582, 1991.

[4] J. Chen and Y. Han. Shortest paths on a polyhedron. In *Proc. 6th ACM Sympo. on Computational Geometry*, pages 360–369. ACM Press, New York, 1990.

[5] K. Cheung, K. Yu, and K. Kui. Volume invarient metamorphosis for solid & hollow rolled shape. In *Proc. Shape Modeling International '97*, pages 226–232. IEEE CS Press, Los Alamitos, Calif., 1997.

[6] D. Cohen-Or, D. Levin, and A. Solomovici. Three dimensional distance field metamorphosis. *ACM Trans. on Graphics*, 17(2):116–141, Apr. 1998.

[7] D. DeCarlo and J. Gallier. Topological evolution of surfaces. In *Proc. Graphics Interface '96*, pages 194–203. Morgan Kaufmann, San Francisco, Calif., May 1996.

[8] H. Delingette, Y. Watanabe, and Y. Suenaga. Simplex based animation. In N. M. Thalmann and D. Thalmann, editors, *Proc. Computer Animation 93*, pages 13–28. Springer-Verlag, Berlin, 1993.

[9] T. Duchamp, A. Certain, T. DeRose, and W. Stuetzle. Hierarchical computation of PL harmonic embeddings. preprint, University of Washington, July 1997. http://www.math.washington.edu/~duchamp/preprints/preprints.html.

[10] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle. Multiresolution analysis of arbitrary meshes. In *Computer Graphics (Proc. SIGGRAPH 95)*, pages 173–182. ACM Press, New York, 1995.

[11] J. Gomes, L. Darsa, B. Costa, and L. Velho. *Warping and Morphing of Graphical Objects*. Morgan Kaufmann, San Francisco, Calif., 1998.

[12] A. Gregory, A. State, M. Lin, D. Manocha, and M. Livingston. Feature-based surface decomposition for correspondence and morphing between polyhedra. In *Proc. Computer Animation 98*, pages 64–71. IEEE CS Press, Los Alamitos, Calif., June 1998.

[13] T. He, S. Wang, and A. Kaufman. Wavelet-based volume morphing. In *Proc. IEEE Visualization '94*, pages 85–92. IEEE CS Press, Los Alamitos, Calif., 1994.

[14] J. F. Hughes. Scheduled Fourier volume morphing. In *Computer Graphics (Proc. SIGGRAPH 92)*, pages 43–46. ACM Press, New York, 1992.

[15] T. Kanai, H. Suzuki, and F. Kimura. Three-dimensional geometric metamorphosis based on harmonic maps. *The Visual Computer*, 14(4):166–176, 1998.

[16] J. R. Kent, W. E. Carlson, and R. E. Parent. Shape transformation for polyhedral objects. In *Computer Graphics (Proc. SIGGRAPH 92)*, pages 47–54. ACM Press, New York, 1992.

[17] J. R. Kent, R. E. Parent, and W. E. Carlson. Establishing correspondences by topological merging: A new approach to 3-D shape transformation. In *Proc. Graphics Interface '91*, pages 271–278. Morgan Kaufmann, San Francisco, Calif., June 1991.

[18] M. Lanthier, A. Maheshwari, and J.-R. Sack. Approximating weighted shortest paths on polyhedral surfaces. In *Proc. 13th ACM Sympo. on Computational Geometry*, pages 274–283. ACM Press, New York, June 1997.

[19] F. Lazarus and A. Verroust. Three-dimensional metamorphosis: a survey. *The Visual Computer*, 14(8/9):373–389, 1998.

[20] A. W. F. Lee, W. Sweldens, P. Schröder, L. Cowsar, and D. Dobkin. MAPS: Multiresolution adaptive parameterization of surfaces. In *Computer Graphics (Proc. SIGGRAPH 98)*, pages 95–104. ACM Press, New York, 1998.

[21] A. Lerios, C. D. Garfinkle, and M. Levoy. Feature-Based volume metamorphosis. In *Computer Graphics (Proc. SIGGRAPH 95)*, pages 449–456. ACM Press, New York, 1995.

[22] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In *Computer Graphics (Proc. SIGGRAPH 87)*, pages 163–169. ACM Press, New York, 1987.

[23] L. Lucas, F. Trunde, and N. Bonnet. Time-dependent 3D data sets rendering: An extension of the morphing technique. *J. Visualization and Computer Animation*, 7(4):193–210, 1996.

[24] M. Mäntylä. *An Introduction to Solid Modeling*. Computer Science Press, Rockville, Maryland, 1988.

[25] J. S. B. Mitchell, D. M. Mount, and C. H. Papadimitriou. The discrete geodesic problem. *SIAM J. Computing*, 16(4):647–668, 1987.

[26] R. E. Parent. Shape transformation by boundary representation interpolation: a recursive approach to establishing face correspondences. *J. Visualization and Computer Animation*, 3(4):219–239, 1992.

[27] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical recipes in C*. Cambridge University Press, Cambridge, UK, 2nd edition, 1992.

[28] T. W. Sederberg, P. Gao, G. Wang, and H. Mu. 2D shape blending: An intrinsic solution to the vertex path problem. In *Computer Graphics (Proc. SIGGRAPH 93)*, pages 15–18. ACM Press, New York, 1993.

[29] K. Sugihara and M. Iri. A solid modelling system free from topological inconsistency. *J. Information Processing*, 12(4):380–393, 1989.

[30] Y. M. Sun, W. Wang, and F. Y. L. Chin. Interpolating polyhedral models using intrinsic shape parameters. In *Proc. Pacific Graphics '95*, pages 133–147. World Scientific, Singapore, 1995.

Biography

Takashi Kanai is a special postdoctoral researcher in the Institute of Physical and Chemical Research (RIKEN). His research interests include geometric modeling and its application to Computer-Aided Design, Computer Graphics. He received his doctor degree in precision machinery engineering from The University of Tokyo in 1998. He is a member of ACM, IEEE CS, JSPE (Japan Society for Precision Engineering) and IPSJ (Information Processing Society of Japan).

Hiromasa Suzuki is an associate professor in the Department of Precision Machinery Engineering at The University of Tokyo. His research interests include geometric modeling and physically based modeling, and their applications to mechanical CAD/CAM systems. He received his doctor degree in precision machinery engineering from The University of Tokyo in 1986. He is a member of JSPE, JSME (Japan Society for Mechanical Engineering), IEEE, ACM etc.

Fumihiko Kimura is a professor in the Department of Precision Machinery Engineering at the University of Tokyo. He has been active in the fields of solid modeling, free-form surface modeling, and product modeling. His research interests now include virtual manufacturing, product life cycle engineering, environmentally conscious manufacturing, and preventive maintenance. He is involved in ISO/TC184/SC4, and is a national representative of IFIP TC5, a member of IFIP WG5.2, WG5.3, and an active member of CIRP. He graduated from the Department of Aeronautics, the University of Tokyo, in 1968, and received a Dr.Eng. degree in 1974.

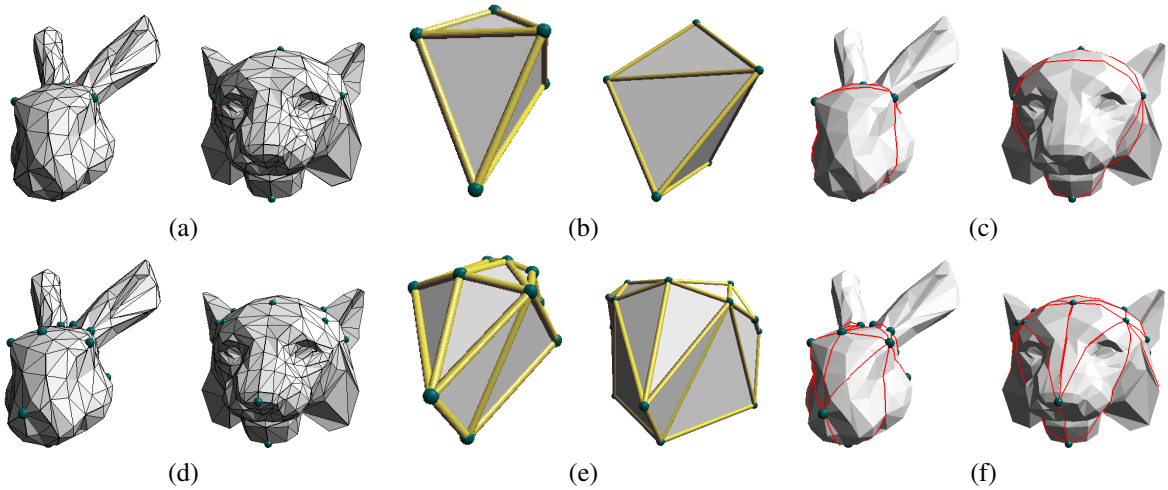


Figure 10: Surface correspondence controls. (a)-(c) and (d)-(f) are different controls specified by the user. (a) 5 CVPs (green spheres). (d) 16 CVPs. (b) Partition control meshes $\mathcal{C}^1, \mathcal{C}^2$ (5 vertices, 9 edges (yellow pipes), and 6 faces). (b) Partition control meshes $\mathcal{C}^1, \mathcal{C}^2$ (16 vertices, 38 edges, and 24 faces). (c),(f) Partitioned meshes $\hat{\mathcal{M}}^1, \hat{\mathcal{M}}^2$. Red lines indicate the shortest paths.

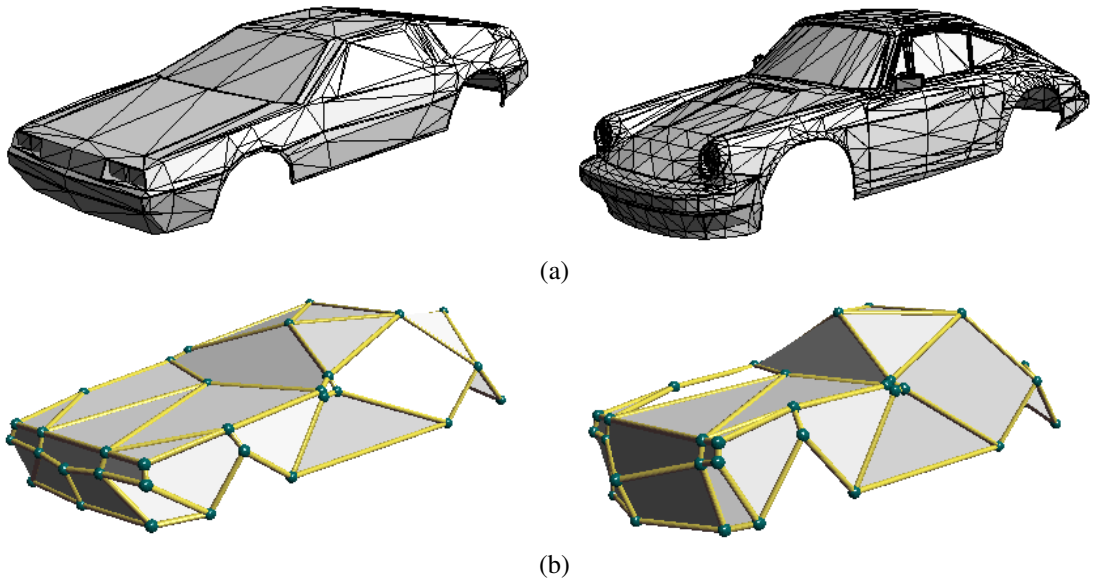
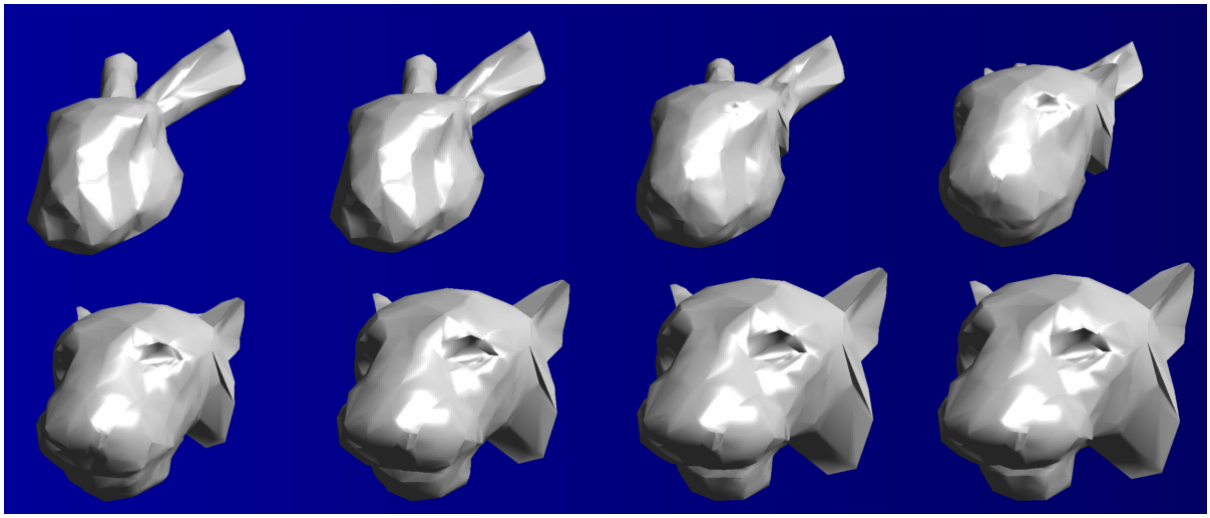
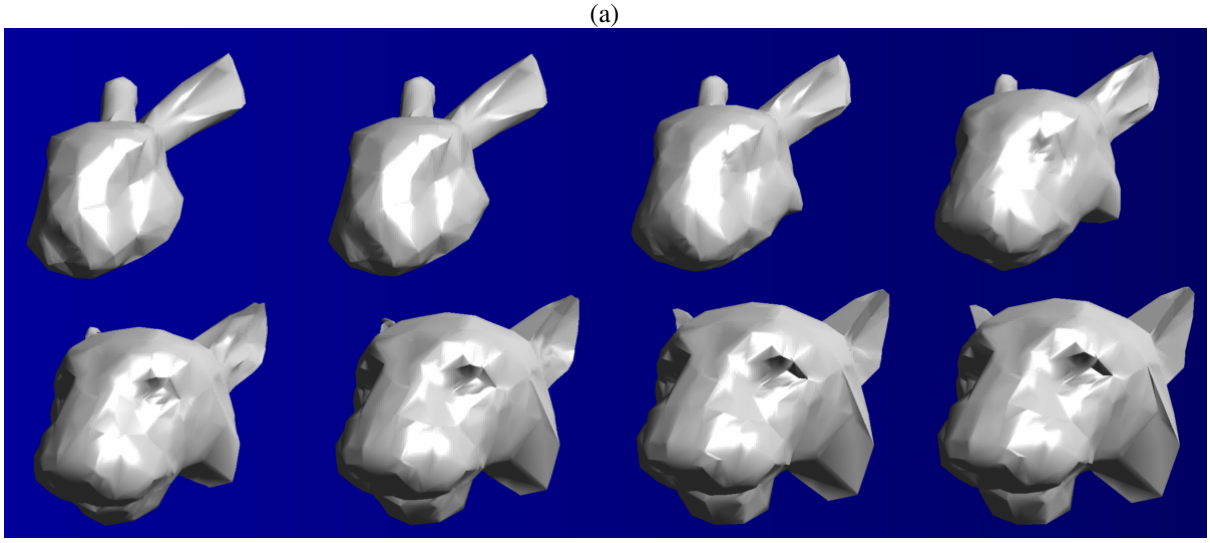


Figure 11: Metamorphosis from the "Delorean" model to the "Porsche 911 Carrera" model. (a) Original meshes $\mathcal{M}^1$ and $\mathcal{M}^2$. (b) Partition control meshes $\mathcal{C}^1$ and $\mathcal{C}^2$ (40 vertices, 68 edges, and 29 faces).

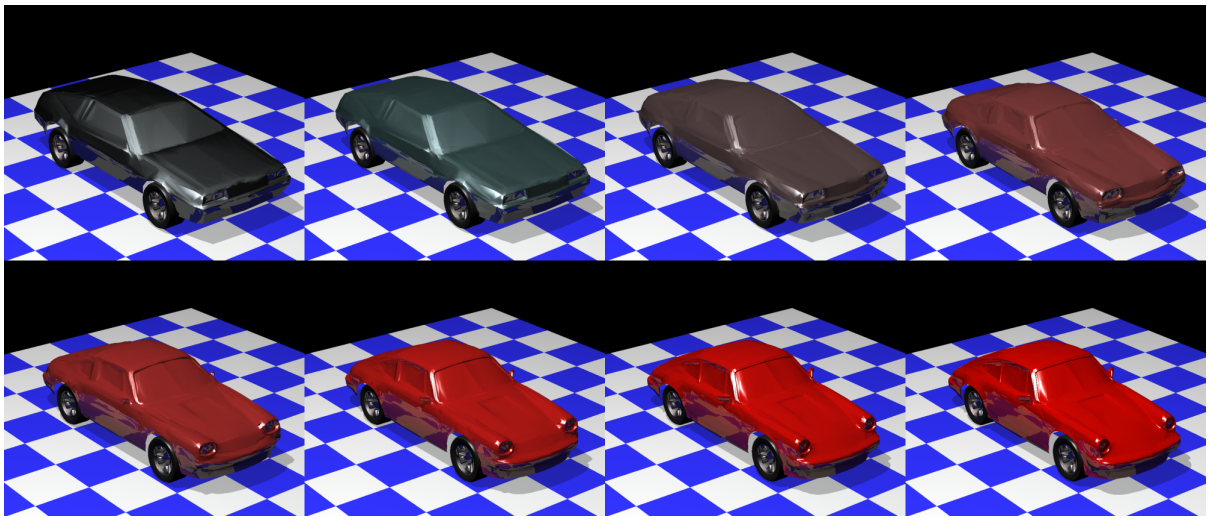


(a)

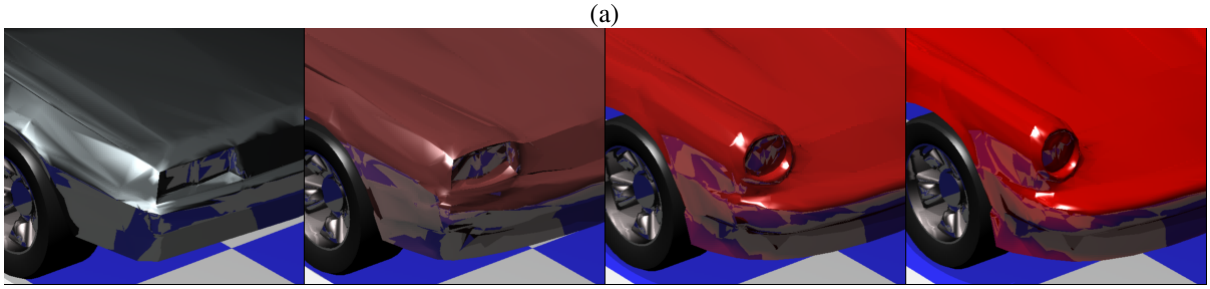


(b)

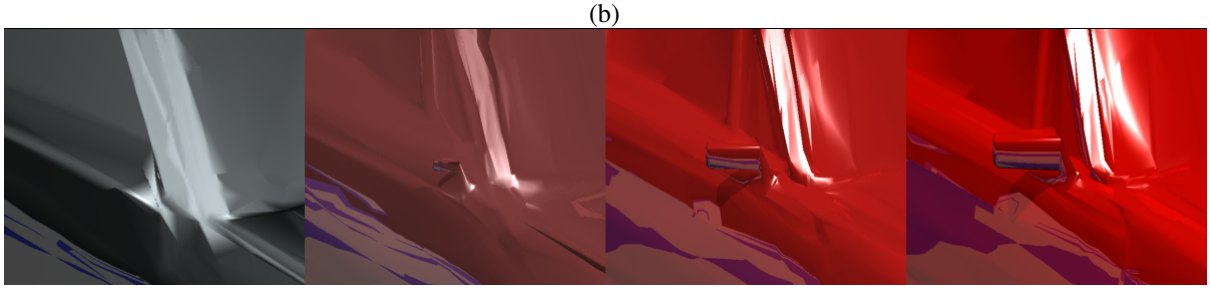
Figure 12: Result of metamorphosis from the “bunny’s head” model to the “tiger’s head” model. (a) and (b) indicate the results of the metamorphosis with the user control shown in Figure 10(a)-(c) and in Figure 10(d)-(f) respectively. In the upper right figure of (a), the tiger’s ear is sticking out from the middle of the bunny’s ear. By refining the partition control mesh as shown in Figure 10(e), this undesired deformation is removed as shown in (b).



(a)



(b)



(c)

Figure 13: Result of metamorphosis from the “Delorean” model to the “Porsche 911 Carrera” model. (a) Views of the whole shape. (b) Magnifications of a headlight region. (c) Magnifications of a door-mirror region.

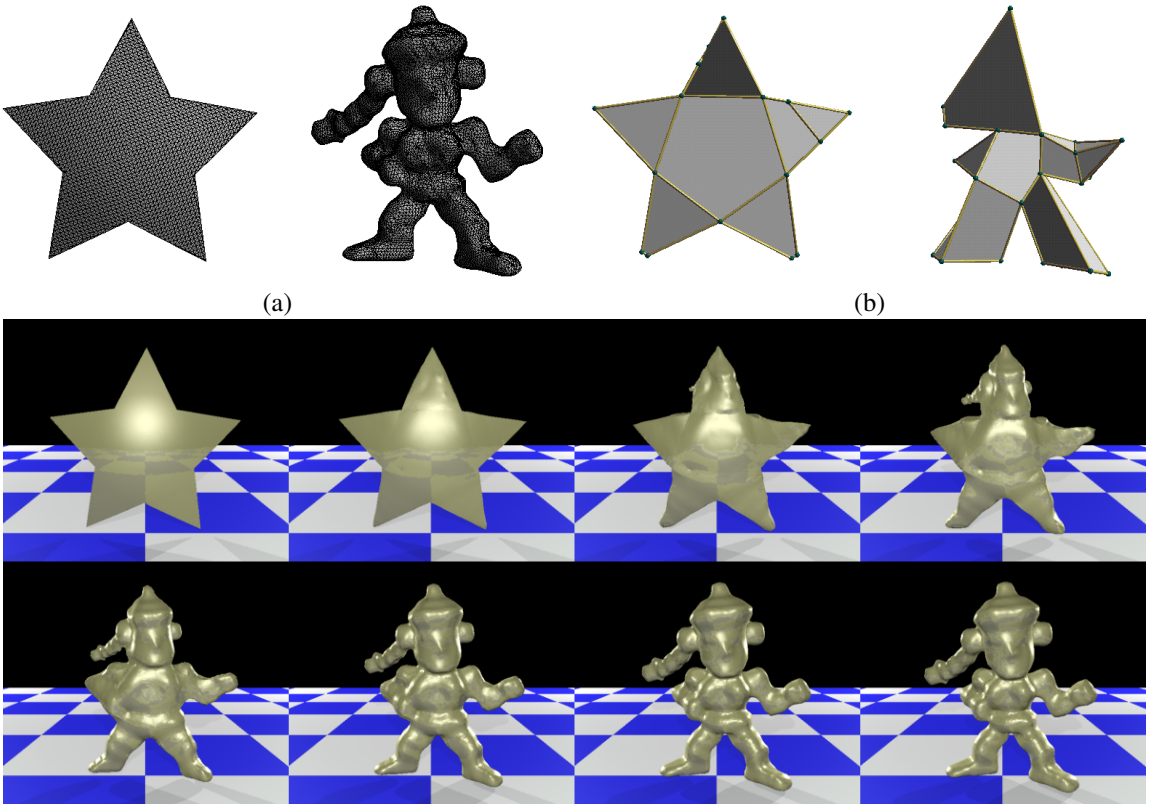


Figure 14: Metamorphosis from "star" model to "pai" model, both of which have large faces. (a) Original meshes $\mathcal{M}^1$ and $\mathcal{M}^2$. (b) Partition control meshes $\mathcal{C}^1$ and $\mathcal{C}^2$ (28 vertices, 55 edges, and 29 faces). (c) Result of metamorphosis.

Figure 15: Result of metamorphosis from the "torus" model to the body of the "bottle" model as a simple example of the non-zero genus ($g = 1$) objects. We have defined 17 CVPs. and a partition control mesh (17 vertices, 35 edges, and 18 faces) for the control of surface correspondence.