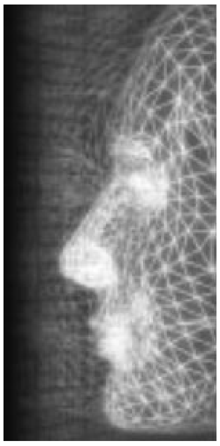
COMPUTER ANIMATION AND VIRTUAL WORLDS
Comp. Anim. Virtual Worlds 2006; **17**: 433–443
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cav.146

Generating genus-n-to-m mesh morphing using spherical parameterization



By Tong-Yee Lee, Chih-Yuan Yao, Hung-Kuo Chu, Ming-Jen Tai and Cheng-Chieh Chen*

Surface parameterization is a fundamental tool in computer graphics and benefits many applications such as texture mapping, morphing, and re-meshing. Many spherical parameterization schemes with very nice properties have been proposed and widely used in the past. However, it is well known that the spherical parameterization is limited to genus-0 models. In this paper, we first propose a novel framework to extend spherical parameterization for handling a genus-n surface. In this framework, we represent a surface S of arbitrary genus by a positive mesh O and several negative meshes N_i. Each negative surface is used to represent a hole. A positive surface O is obtained by removing all holes in the original surface S. Then, both positive and negative meshes are genus-0 and can be spherically parameterized, respectively. To compute S, we can use a Boolean difference operation to subtract negative N_i from a positive O. Next, we apply this novel framework to generate genus-n-to-m mesh morphing application without restriction of n = m. Finally, there are many interesting non-genus-0 mesh morphing sequences generated. Copyright © 2006 John Wiley & Sons, Ltd.

Received: 10 April 2006; Revised: 2 May 2006; Accepted: 10 May 2006

KEY WORDS: spherical parameterization; genus reduction; Poisson stitching; positive and negative objects; metamorphosis

Introduction

Related Work

Surface parameterization of 3D models is a fundamental tool in computer graphics. Many graphics applications benefit from surface parameterization, in particular: texture mapping, morphing and re-meshing. Surface parameterization embeds a given 3D surface onto simpler domains such as a planar region, a sphere or a simplicial complex. The planar parameterization schemes[1,2] often require the surface to be partitioned using cuts into disk-like charts and therefore discontinuity such as cutting seams is created. For texture-mapping application, planar parameterization may be the most natural way. However, for applications such as morphing and re-meshing, the surface should better be parameterized onto a domain topologically equivalent to it. On the other hand, for a genus-0 surface, the sphere is the most natural parameterization domain. Spherical parameterization does not require cutting the surface into several pieces. In the past, many spherical parameterization schemes[3,4,5,6,7,9] with very nice properties have been proposed and have been widely used in many applications. Haker *et al*.[3] have computed spherical parameterization by conformal approximation of meshes over the sphere. Alexa[4] proposed a relaxation approach; however this approach does not guarantee a valid spherical embedding. Sheffer *et al*.[5] proposed an angle-based method with a non-linear optimization procedure. Their approach guarantees a valid spherical embedding but it is, computationally, very intensive.

Gotsman *et al*.[6] generalized the method of barycentric coordinates for planar parameterization to compute the spherical embedding by solving a quadratic system. Praun *et al*.[7] introduced a robust method for spherical embedding. Furthermore, the minimization of their proposed stretch-based measure is used to reduce the

*Correspondence to: T.-Y. Lee, Computer Graphics Group, Visual System Laboratory, Department of Computer Science and Information Engineering, National Cheng-Kung University, No.1 Ta-Hsueh Road, Tainan 701, Taiwan.
E-mail: tonylee@mail.ncku.edu.tw

Contract/grant sponsor: National Science Council, Taiwan, Republic of China; contract/grant numbers: NSC-94-2213-E-006-063NSC-94-2213-E-006-005.

distortion. Later, Praun *et al.*[8] consistently parameterized several genus-0 models onto a user-specified simplicial complex. Asirvatham *et al.*[9] extend[7] and combine the idea in[8] to create a spherical parameterization under some constraints. Recently, Schreiner *et al.*[10] and Kraevoy *et al.*[11] consistently parameterize a set of models of arbitrary genus onto a simplicial complex. However, both techniques are very complicated for practical implementation and are usually time-consuming. Furthermore, these two schemes do not scale well in terms of the number of models to be consistently parameterized. In contrast, consistent spherical parameterization by Asirvatham *et al.*[9] can be better scaled for several objects. However, this approach is only valid for genus-0 meshes.

Contribution

Planar parameterization computes a 3D-to-2D embedding for a given 3D surface. Embedding a surface onto a sphere or a simplicial complex is a 3D-to-3D mapping. Intuitively, planar parameterization potentially incurs much distortion than the other two kinds of embeddings. Since a simplicial complex is always chosen to be very similar to a given 3D surface, therefore, this kind of embedding can potentially have the least distortion among these three parameterizations. However, this kind of embedding always requires the surface to be partitioned or clustered before embedding the surface onto a chosen simplicial complex. The choice of a good simplicial complex is also another issue. Furthermore, whenever a consistent parameterization is required for a set of models, the task of matching features makes this kind of embedding even more complicated as mentioned in References [10,11]. In contrast, spherical parameterization is very elegant with many nice properties such as continuity, smoothness, acceptable surface distribution over spherical areas, and no cuts or partitions required. In addition, embedding onto the spherical domain without or with consistency requirement is usually much simpler than that onto the simplical complex. However, in contrast to the other two embeddings, spherical embedding is inherently valid for genus-0 surface only.

In this paper, we propose a novel framework to extend spherical parameterization for handling genus-n surface. Any previous spherical parameterization scheme[4–7,9] can be easily integrated into this framework. Therefore, our proposed framework still shares some good properties of spherical parameterizations. In the past, there have been many 3D morphing schemes proposed. Most of these schemes are concentrated on morphing genus-0 models[19,21–24] In contrast, we apply this framework to create three-dimensional morphing sequences between genus-n and genus-m models. With this novel framework, we can easily allow $n \neq m$ in the morphing application. In contrast to most of the previous work[10,11] both schemes always require $n = m$. Sometimes this requirement makes the morphing sequence weird. The major contributions of our paper are listed below:

- We generally extend spherical parameterization for handling surface of arbitrary genus.
- Practical methods to detect holes and eliminate its genus for a given model.
- New method to create a 3D morphing sequence between genus-n-to-m meshes without restriction of $n = m$.

Approach Overview

Our goal is to parameterize a surface S of arbitrary genus onto the spherical domain. Our approach can be described by the following steps:

- We detect possible holes on S.
- We attempt to remove each hole h_i by using Poisson stitching and also create a corresponding genus-0 mesh N_i.
- After removing holes, S becomes a genus-0 mesh called O.
- Since O and N_i are all genus-0 meshes, we can apply any spherical embedding to them.
- We term O as a positive mesh and each N_i is a negative mesh. We can compute S by subtracting all N_i from O using a Boolean difference operation.

In the next section, more details about the algorithm will be given. Based on this approach, we present a new 3D non-genus-0 surface morphing scheme in three-dimensional morphing of arbitrary genus.

Algorithm Details

For a given genus-n surface S, we need to compute a positive genus-0 O and several negative genus-0 meshes $N_1, N_2 \ldots, N_n$. A handle is defined as a region of S with genus-1. Our algorithm will locate n handles in S by constructing and analyzing a Reeb graph and removing each handle by Poisson stitching.

In spirit, our algorithm is closely related to Wood *et al*.[12] However, there are several distinctions from their approach described as follows. First, our algorithm is a polygonal-based approach and Wood *et al*. used a volumetric-based approach. Their algorithm makes an axis aligned sweep through the volume to locate handles; however, this approach may fail to detect holes with totally different orientations simultaneously. If the model is swept along the Z direction, those handles parallel to the X–Y plane potentially cannot be located. To avoid this problem, sometimes, human adjustment of sweeping orientation is required; therefore, this method is not robust and it can be tedious. In contrast, our approach does not require sweeping along an axis and it is also robust to detect all possible handles. Second, for better computing efficiency, Wood *et al*. attempt to use a triangle fan about the centroid of the handle to remove a handle. Depending on the shape of the handle, the triangles of the fan may intersect one another or intersect other regions of the surface. For a better solution, our approach removes a handle by a Poisson stitching. In addition, Wood *et al*. also suggest a robust volumetric approach to remove each handle but with less efficiency.

Locating a Handle

Given a surface, we construct and analyze a Reeb graph[13] to locate handles. Reeb graph is a useful data structure to encode the topology information of this surface. Constructing a Reeb graph usually involves in defining a scalar function f defined on a surface. Wood *et al*.[12] simply use a height function, i.e., a z value. The height function is not a good scalar function to build a correct Reeb graph for a surface. Therefore, the approach employed by Wood *et al*. potentially fails to detect some handles. For our approach, f is defined as a scalar harmonic function on a surface with respect to selected seed points.

With appropriate boundary conditions, a harmonic function f satisfies the Laplace equation $\nabla^2 f = 0$. If $\nabla^2 f \neq 0$, f is called the Poisson function. Considering piecewise linear functions over triangulated manifolds, we can find a scalar value based on Laplace equation for each vertex on a triangular surface S by solving a sparse linear system.[14] For a discrete scalar harmonic function, our boundary settings are: (1) we assign value 0 for a selected source seed, and (2) value 1 for a selected sink seed. The ideal candidates for source and sink seeds are any two vertices on the surface with the largest possible geodesic distance between them. Under these settings, the resulting harmonic functions smoothly vary between 0 and 1 over the surface S. A scalar function f defined by a normalized geodesic distance is another good choice to build a Reeb graph. Hilaga *et al*.[15] use this alternative to construct a Multiresolutional Reeb Graph (MRG). However, to compute a smooth normalized geodesic distance using *Dijkstra*'s algorithm on a triangulated manifold S, tasks such as resampling, subdivision of a mesh and adding extra short-cut edges on S are required in their implementation. In contrast, our approach does not require these extra steps but can approximate a smooth f on S.

By using *Euler's Theorem*, we can calculate the genus value of a discrete 2-manifold model easily. Let us consider a model that has $|V|$ number of vertices, $|E|$ number of edges and $|F|$ number of triangles. Let $X = |V| - |E| + |F|$, the genus value g of the surface is equal to $(2 - X)/2$. We use *Euler's theorem* to compute the genus of subsets of S while we are constructing a Reeb graph. Furthermore, if the graph is not a tree, it must contain a cycle. In graph algorithm, it is a very straightforward task to locate cycles for a given graph with cycles by a DFS of BFS graph traversal.

Once a scalar field f is determined over S, we can build a Reeb graph for it. In,[15] the MRG is constructed in a fine-to-coarse order by requiring the domain of function f to be divided into uniform intervals. Our purpose is to build a Reeb graph with n cycles for a given genus-n surface S. Our Reeb graph is constructed in a coarse-to-fine order. Initially, we have an initial Reeb graph G with a root node containing all triangles of S. Next, we build a Reeb graph by recursively calling the *make_Reeb*(G, n) function, where G is a Reeb graph with n cycles to build. The *make_Reeb* (G,n) consists of the following steps: *make_Reeb*(G, n)

Step 1: we subdivide the domain of function f of G into finer intervals and then we reconstruct a new Reeb graph G' as in Reference [15].

Step 2: If the number of cycles, denoted by m, in G' is equal to n, return G'.

Step 3: Now, we have m cycles, $C_1, C_2, \ldots, C_m$ and $m \neq n$. Each C_i corresponds to a subgraph of G', called G_i.

Step 4: For each G_i, we first compute the genus of sub-surface contained in G_i, called g_i and then recursively call *make_Reeb* (G_i, g_i).

Step 5: After we separate m circles from G', assume we have m' non-cyclic subgraphs, $H_1, H_2, \ldots, H_{m'}$ and each H_j is disjointed to each other.

Step 6: For each H_j, we first compute the genus of subsurface contained in H_j, called g_j and then recursively call *make_Reeb*(H_j, g_j).
Step 7: In accordance with the connectivity of triangles contained in $G_1, G_2, \ldots, G_m$ and $H_1, H_2, \ldots, H_{m'}$ we unify these subgraphs to obtain a new G'.
Step 8: return G'.

Finally, we use a DFS graph traversal again to detect each cycle in our Reeb graph G reconstructed by *make_Reeb*( ). Each cycle represents a handle (i.e., hole), a region of S with genus-1. Given a genus-n surface S, our algorithm robustly finds n handles.

Removing a Handle and Spherical Parameterization

A Non-Separating Cut-Path Dong a Handle. Genus of surface S can also be defined as the largest number of non-intersecting simple closed curves that can be drawn on the surface at each specific time without separating S.[12] We can cut the model surface along such a curve without separating the model into disjointed parts. In this paper, we refer to such a curve as the *non-separating cut-path*. We then proceed to find the non-separating cut-path of each handle, i.e., hole as follows.

In Reeb graph, for each node of a cycle, we find its corresponding region of S. To obtain a non-separating cut-path of the hole, we first find a vertex of each node which is nearest to the centroid of the handle. Then, we connect vertices of each adjacent node using *Dijkastra's shortest path* algorithm. This forms an initial cut-path of the handle. Each path between adjacency nodes in Figure 1(a) is the shortest path between them. In this manner, we can obtain an initial non-separating cut-path, as indicated by the light grey path in Figure 1(b). This cut-path may not be very smooth. The smooth cut-path can be constructed by first growing outward a region along two sides of an initial non-separating cut-path. Grown patches are blue and green areas as shown in Figure 1(b). Second, we formulate the problem of finding a smooth path on grown patches as a network flow problem as in Reference [16]. So, we construct a dual graph of the grown patches. Each vertex of a dual graph is a face of the grown patches. An edge between two vertices in a dual graph means the corresponding faces of the triangle share an edge in the original patches. In our implementation, *Edmond–Karp's* algorithm[17] is used to solve a maximum flow/minimum cut problem. We compute Equation (1) to assign the capacity $c(E)$ to each edge $E = (v_1, v_2)$, where v_1 and v_2 are its two end vertices, and $\vec{n}_1$ and $\vec{n}_2$are the two normal directions of the triangles sharing this edge. The first term attempts to find a cut-path along the most convex or most concave path. The second term $\|v_1 - v_2\|$attempts to find a cut-path with the shortest possible geodesic distance. Figure 1(c) shows a smooth non-separating cut-path after this optimization.

$$c(E) = \frac{\vec{n}_1 \cdot \vec{n}_2}{\|\vec{n}_1\|\|\vec{n}_2\|} + \|v_1 - v_2\| \qquad (1)$$

Boundary Smoothing. In this section we propose the main concept of handle-removing. Blue and green triangles are boundary triangles while the darker paths are their corresponding boundary edges. Vertices on boundary edges are called boundary vertices. In this section, we use a Poisson stitching method to remove the hole with two appropriate 'cap meshes' while removing the interiors, i.e., blue and green triangles.

In order to fill the hole with a smooth boundary, we must grow faces from the non-separating cut-path. Since

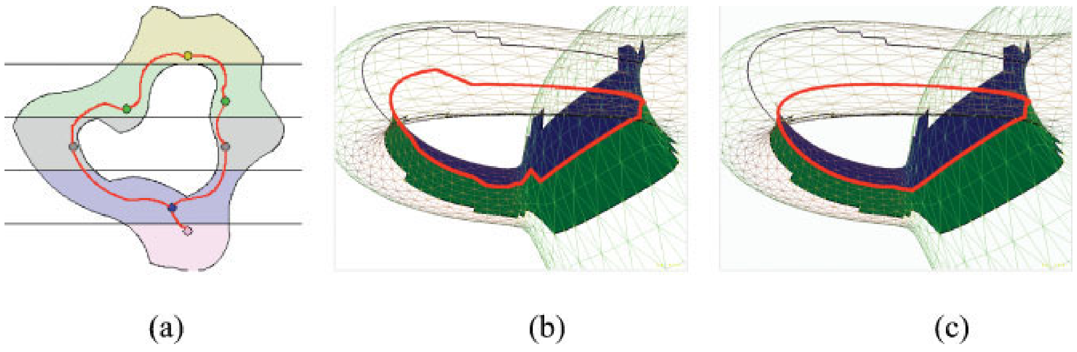


Figure 1. A smooth non-separating cut path along a handle.

the cut-path is a closed path, we can define two sides of the path. Our goal is to grow the face from both sides of the path to keep the outer area of the grown region most fit to the hole. However, how thick the region should grow is a subjective problem and depends on the model itself. Due to the difference in area and shape of the triangles, the grown region might be irregular if we simply grow one-ring face per step. Therefore, starting from the cut-path, we first find a vertex of the neighboring triangles that has the longest distance to the path, and use this distance as the growing distance per step. Figure 2 shows a hole in the handle of a teapot. The red path is the cut-path and the green and blue triangles are 1-ring neighboring faces of the path. The yellow arrows in Figure 2(a) indicate the shortest and longest distance between 1-ring neighboring vertices and vertices on the cut-path. To achieve a smooth boundary, we cannot simply grow boundary triangles without considering the shape of the 1-ring neighboring triangles. We want the filled hole to be as smooth as possible, therefore, we use the longest distance indicated by the right yellow arrow in Figure 2(a) as the growing distance and grow the faces from the cut-path. Figure 2(b) shows the result of the growth.

Due to the discrete property of 3D models with various sizes of triangles, we cannot always guarantee a smooth boundary simply by using the above step. A smooth boundary can make filling of holes more compact with nice visual appearance. To smoothen the boundary, we inflate the concave boundary edges or subdivide the triangle that is too sharp along the two boundary edges. For each boundary vertex v, we check the total angle of a fan of triangles from v bounded by its two adjacent boundary edges of v. If the total angle of these triangles is smaller than a user defined threshold, we inflate this concave boundary edges by the edges of fan triangles. On the other hand, we require subdivision of triangles if the boundary is too sharp. First we use a recursive method to determine a new smooth boundary edge. For a boundary vertex (the yellow vertex in Figure 2(c)), if the angle of the triangles covered by the fan from one of the adjacent boundary edge to the other is larger than a user-defined threshold (Figure 2(c)), we first pop the two neighboring boundary edges out of the boundary edge list and add a virtual edge (the red dot-dashed line in Figure 2(d)) by connecting the two end vertices (blue vertices in Figure 2(e)) of both boundary edges. Again we check the angle until it is smaller than a user-defined threshold. The final virtual edge is the new smooth boundary edge (the red dot-dashed line in Figure 2(f)).

Once we obtain the new boundary edge $E = (v_i, v_j)$ where v_i and v_j are its two end vertices, we subdivide the triangles intersected with the new boundary edge by a

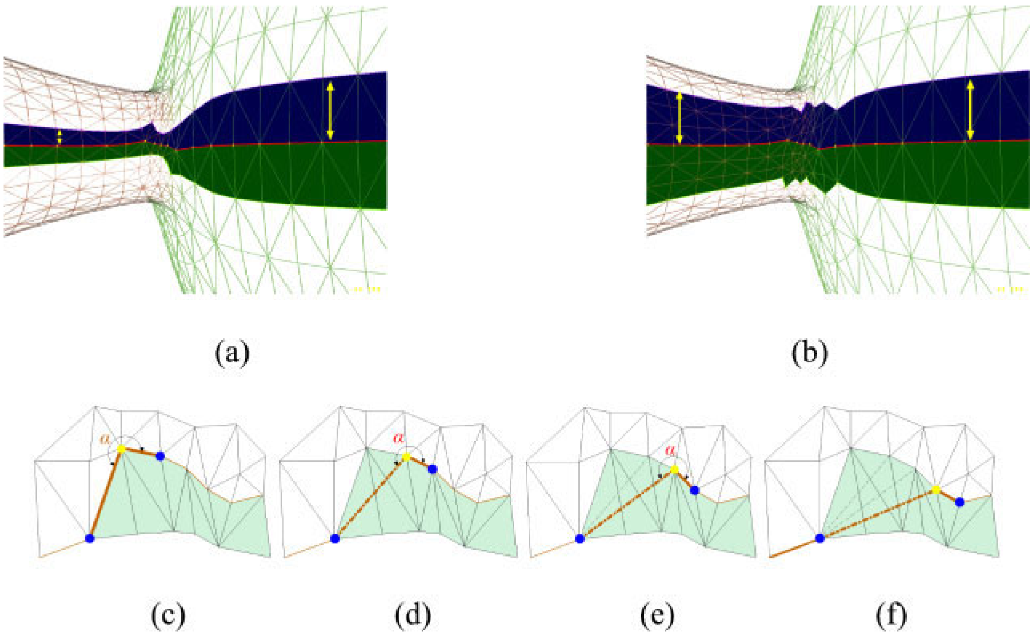


Figure 2. Growing boundary triangles and finding a new boundary edge.

cutting plane P of this edge. In order to perform the line–plane intersection test, we need to acquire the equation of the cutting plane. Assume the equation of the cutting plane is $P : ax + by + cz + d = 0$ where vector (a, b, c) is normal to the cutting plane P. First, we calculate the normal, $Normal_p$, of the plane by the average of $(n_i + n_j)$, where

$$n_i = (v_j - v_i) \times \frac{\sum_{k \in Ring(v_i)} Normal(f_k)}{\|Ring(v_i)\|}$$
$$n_j = (v_i - v_j) \times \frac{\sum_{k \in Ring(v_j)} Normal(f_k)}{\|Ring(v_j)\|} \qquad (2)$$

In the above equation, *Normal(f)* is the normal vector of the triangle *f* and *Ring(v)* is the set of one-ring triangles of vertex *v*. For each end vertex of the new boundary edge, we calculate the average normal vector of its one-ring triangles. Then we take the cross product of this average normal vector with the vector from current end vertex to the other to obtain the approximated normal vector from this end vertex. We perform the same steps to the other end vertex of the new boundary edge. Finally, we average these two normal vectors to obtain the normal vector of the cutting plane. Second, we need to calculate the parameter d of the cutting plane. Since we want to approximate the new boundary edge by the cutting plane, the two end vertices and their middle point are planned to lie on the plane. Thus we calculate d by following equations.

$$v_m = \frac{v_i + v_j}{2},$$
$$d = -(ax + by + cz) = -Normal_p \cdot v_m \qquad (3)$$

With the equation of the cutting plane, we then perform line–plane intersection test to detect the intersection of the edges on the boundary triangles. These intersections are the points where we will subdivide those triangles.

Poisson Stitching. Once we have two smooth boundaries B_i, $i = 1, 2$, on the two sides of the non-separating cut-path, we build two 'cap meshes', C_i, $i = 1, 2$, to stitch the hole. Those boundary triangles between the two cap meshes will be removed. In our implementation, the cap mesh is currently the shape defined by a uniform circle located on the origin. To ensure that we have one-to-one correspondence, the number of boundary vertices on a cap mesh C_i is required to be equal to the number of boundary vertices B_i. In addition, we also require the boundary vertices on a cap mesh C_i to be distributed according to the distance between adjacent vertices on B_i. In general, the shape of the initial cap meshes may not match the shape of the boundary of the hole. Since the vertices of the cap mesh and the boundary vertices have one-to-one correspondence, we transform and warp the cap mesh to make it fit to the hole. The basic concept is to first transform each triangle of the cap mesh to maintain boundary constraints and then to glue disjointed triangles by solving a Poisson equation. This idea was first proposed by Yu *et al.*[18] to perform the object merging. In this paper, we use this approach to modify the geometry of the cap mesh implicitly through gradient field manipulation to stitch the hole by solving a Poisson equation. In this manner, we merge cap meshes C_i, $i = 1,2$, with B_i, $i = 1,2$ in a smooth manner to remove this handle or hole. Figure 3(a) and 3(b) show the examples for removing the hole in the handle of a teapot and the hole near the tail of the dragon. Both examples show that the cap mesh now fits the shape of the boundary of the hole well. Figure 3(c) shows the results cited from Wood.[12] Our results in Figure 3(d) look smoother than their result.

Using Spherical Parameterization for Genus-*n* Surface. Given a genus-*n* model S, once we remove genus.*n* by the proposed method, we can get a genus-0 surface S'. In this paper, we call S' as a positive surface O. Next, we can apply CSG techniques to subtract S from S'. As a result of subtraction, we get *n*surfaces, N_i, $i = 1, 2, \ldots, n$, respectively. In this paper, each N_i is called a negative surface and represents a surface to fill a hole. In reverse direction, we can also subtract all N_i, $i = 1, 2, \ldots, n$ from S' to get S back. In our framework, each N_i, $i = 1, 2, \ldots, n$ is the union of the following two parts: (1) two cap meshes, and (2) boundary triangles (e.g. blue and green regions in Figure 2(b)) on the two sides of the non-separating cut-path.

Now, we represent a surface S of genus-*n* by a positive surface O and a set of negative surfaces N_i, $i = 1, 2, \ldots, n$. Both positive and negative surfaces are genus-0. Therefore, we can use $n + 1$ spheres to parameterize these $n + 1$ surfaces independently, and thus we can accomplish the spherical parameterization of genus-*n* surface S. In our implementation, we use the approach in Reference [6] to parameterize each surface onto a sphere. Any other spherical parameterization can be used, too. In summary, given a genus-*n* surface, its spherical

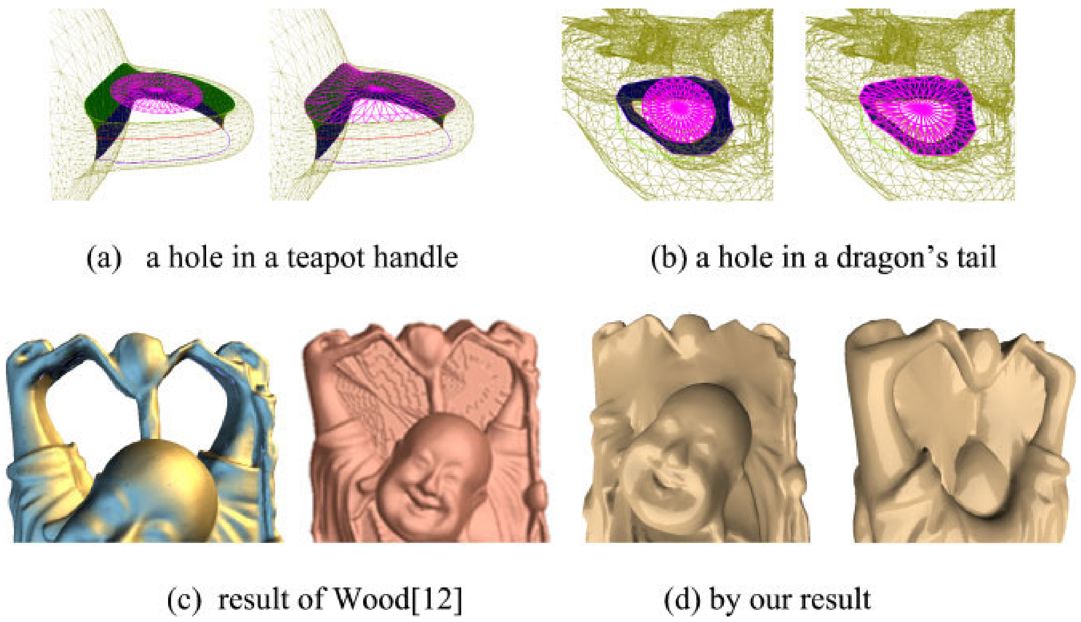


Figure 3. Removing a hole by the Poisson stitching, and comparing the result from,[12] (c), with ours, (d).

parameterization consists of $n+1$ spherical parameterizations in our framework.

Three-Dimensional Mesh Morphing of Arbitrary Genus

So far, we have introduced a novel framework to parameterize a surface of arbitrary genus using several spherical parameterizations. To demonstrate the usefulness of the proposed framework, we apply this framework to 3D morphing application as shown in Figure 4. In this figure, given two surfaces S and S', we compute a corresponding positive mesh pair O and O', and two negative mesh sets: $(N_1,N_2,\ldots,N_n)$ and $(N_1',N_2',\ldots,N_m')$ for holes. In the case of $n\neq m$, we can generate extra negative meshes called pseudo negative meshes to have an equal number of paired negative meshes. Then, we let (N_i, N_i') be a pair of negative meshes. Next, for a given pair of spherical embeddings, from,[6] we first specify feature points and then we independently compute a consistent spherical embedding for each corresponding pair (e.g., (N_i, N_i') for all i and (O, O') for computing morphing using Reference 9 or 19. Although $n=m$, we sometimes need to create extra pseudo-negative meshes for computing better morphing. For example, given a feline model with two holes in the tail and another model with two holes in the head, it is better not to simply pair holes without account of their positions in this situation. In this case, the better solution is to pair a hole (i.e., negative mesh) on a model with a pseudo hole (i.e., pseudo negative mesh) on another model. It is easy to handle this situation using the proposed scheme. Once each consistent embedding is computed, we independently compute a morphing sequence for each pair as shown in Figure 4. Finally, we apply a Boolean difference operation to subtract each intermediate negative object from an intermediate positive object for obtaining a desired morphing sequence. In our implementation, for a given pair of spherical parameterizations and feature points specified by users, we compute morphing using a progressive morphing scheme proposed by Lin *et al.*[19]. This technique[19] can align user-specified features between two input spherical embeddings. Other consistent spherical parameterizations such as References [4,9] can be used, too. But, Lin *et al.*'s[19] procedure can create less number of triangles for the intermediate morphing models. Figures 4 and 5 show experimental results using this proposed 3D morphing framework. In Figure 4, a genus-1 teapot is morphed into a genus-1 mug. Using the proposed framework, we can perform morphing between models with different genus such as genus-2 to genus-1 (Figure 5(a, c, d)). Matching of

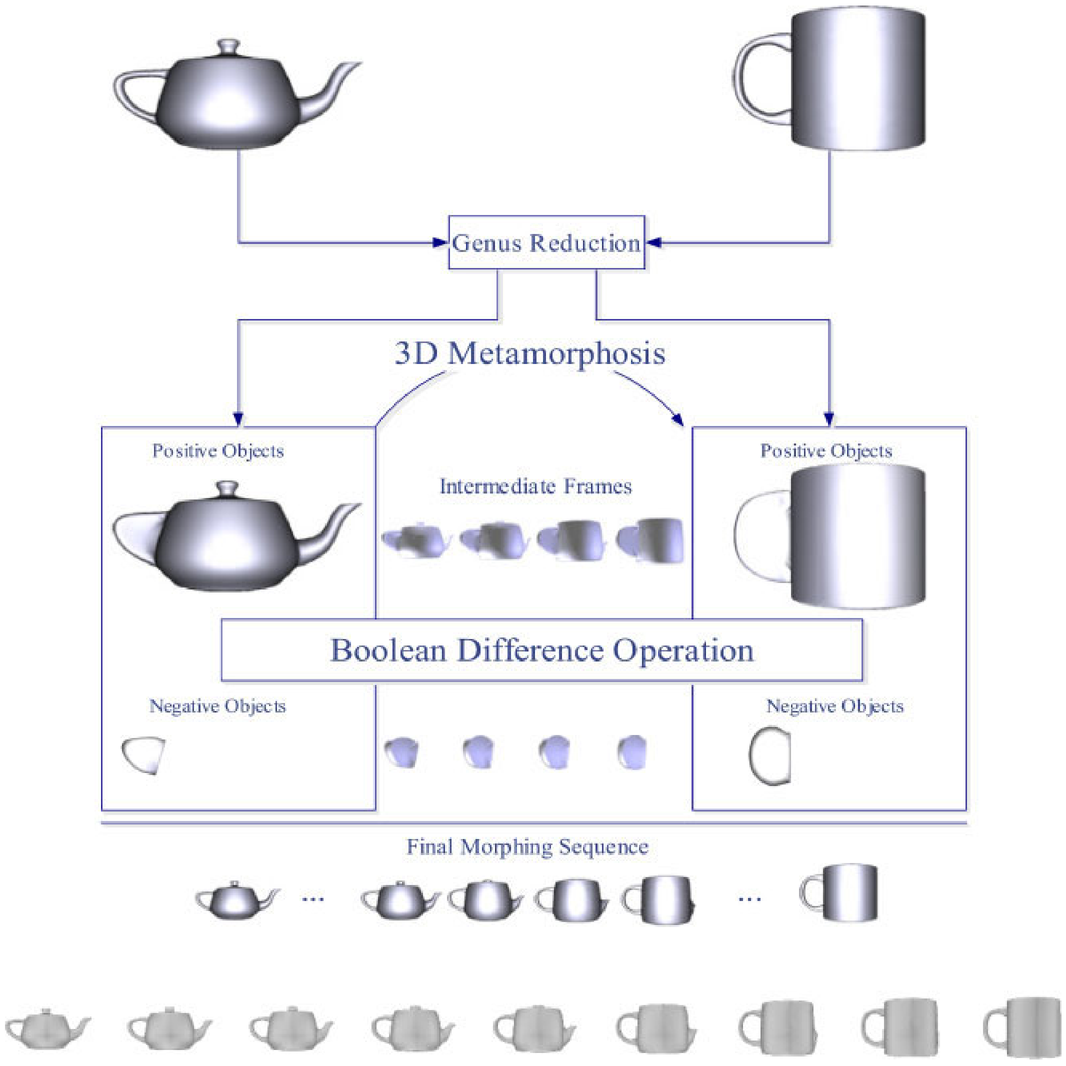


Figure 4. A novel framework for non-genus-0 surface morphing using spherical parameterization.

negative objects is necessary to deal with these cases, since the user specifies where each negative object from source model should be morphed to the negative object of the target model. In addition, the user needs to specify corresponding features, too. For example, in Figure 5(a), there are two holes in the source model and one hole in the target model. We can pair each hole in the source model with the same hole in the target model. We independently compute two intermediate negative objects and then subtract them from the intermediate positive object. On the other hand, if there exists a negative object of the source model but there is no corresponding negative object of the target model, we can also assume that there is a virtual hole (negative object) in the corresponding position of the target model with very small size. Thus the hole in the morphing sequence will become smaller gradually and finally disappear. In contrast, if there is no hole in the source model yet a hole exists in the target model, this method will also work (see Figure 5(e)).

Conclusion and Future Work

In this paper, we first present a new framework of spherical parameterization for surface with arbitrary

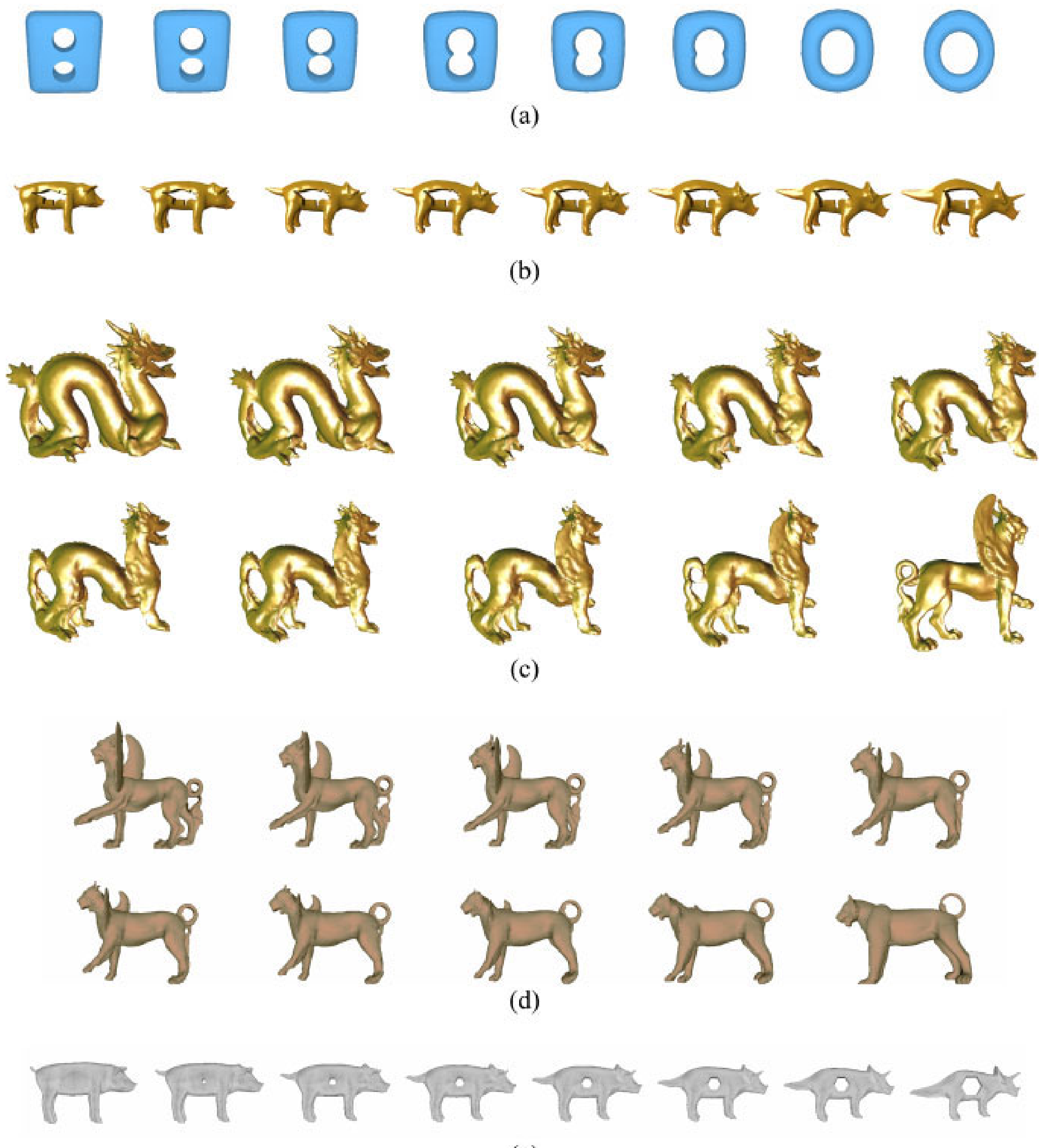


Figure 5. (a) Morphing from genus-2 object to genus-1object. (b) Morphing between a pig with a Triceratops's hole and a Triceratops with pig's hole. (c) Morphing between genus-1 Dragon model and genus-2 Felin model. (d) Morphing between genus-2 Feline model and genus-1 Lion model. (e) Morphing between a pig with no hole and a Triceratops with 1 hole. In this example, the proposed scheme creates a virtual hole in the source model (pig) for the corresponding hole in the target model (Triceratops).

genus. Then, we apply this framework to generate 3D genus-n-to-m mesh morphing without restriction of $n = m$. Experimental results verify the proposed framework. Although we do not present any new spherical parameterization scheme, the major contribution of this paper is to make any spherical parameterization feasible for non-genus surface. Any spherical parameterization can be benefited from our contribution to handle a non-genus surface. In addition, the proposed detection scheme for holes is very robust. Wood *et al.*[12] potentially do not detect holes. In contrast, the proposed method can do well. In comparison with other 3D morphing techniques, the proposed framework makes 3D surface morphing more general and much easier to implement under spherical parameterization. Using the proposed framework, genus-*n*-to-*m*morphing becomes straightforward in contrast to the previous work.

This paper presents a first effort to parameterize a genus-*n* surface using the spherical domain. There are many interesting steps are left, to be further explored in the future. First, we plan to apply the same framework to remeshing and surface texturing soon. Second, currently we use two caps to fill the hole. For most cases, this solution is general enough. However, for the case such as a hole within or crossing another hole, we cannot simply fill each hole with two caps. We need a better solution. In future, we may need a volumetric approach such as Reference [20] to handle more complicated cases. Finally, for the CSG difference operation, holes are reconstructed by subtracting the positive object by the negative objects. The boundary regions that are subtracted are then re-meshed. Hence the connectivity of triangles in these regions is different in each frame, although they look smooth in the morphing sequence. The problem of how to maintain a consistent connectivity for hole-regions in the morphing sequence is an interesting research for future exploration.

ACKNOWLEDGEMENTS

Our tested models are courtesy of Stanford Computer Graphics Laboratory.

References

1. Gu X, Gortler S, Hoppe H. Geometry images. *ACM SIGGRAPH* 2002; pp. 355–361.
2. Sorkine O, Cohen-OR D, Goldenthal R, Lischinski D. Bounded-distortion piecewise mesh parametrization. *Proceedings of the Conference on Visualization* 2002; 355–362.
3. Haker S, Angenent S, Tannenbaum S, Kikinis R, Sapiro G, Halle M. Conformal surface parametrization for texture mapping. *IEEE TVCG* 2000; **6**(2): 181–189.
4. Alexa M. Merging polyhedral shapes with scattered features. *The Visual Computer* 2000; **16**(1): 26–37.
5. Sheffer A, Gotsman C, Dyn N. Robust spherical parameterization of triangular meshes. 4th Israel-Korea Bi-National Conference on Geometric Modeling and Computer Graphics 2003; pp. 94–99.
6. Gotsman C, Gu X, Sheffer A. Fundamentals of spherical parameterization for 3D meshes. *Proceedings of ACM SIGGRAPH*. 2003; pp. 358–363.
7. Praun E, Hoppe H. Spherical parametrization and remeshing. *ACM SIGGRAPH* 2003; 340–349.
8. Praun E, Sweldens W, Schröder P. Consistent Mesh Parameterizations. *SIGGRAPH* 2001; 179–184.
9. Asirvatham A, Praun E, Hoppe H. Consistent spherical parameterization. International Conference on Computational Science 2005; (2): 265–272.
10. Schreiner J, Asirvatham A, Praun E, Hoppe H. Inter-surface mapping. *ACM SIGGRAPH. ACM Transactions* 2004; Graph. **23**(3): 870–877.
11. Kraevoy V, Sheffer A. Cross-parameterization and compatible remeshing of 3D models. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2004)* 2004; **23**(3): 861–869.
12. Wood Z, Hoppe H, Desbrun M, Schröder P. Removing excess topology from isosurfaces. *ACM Transactions on Graphics* 2004; **23**(2): 190–208.
13. Reeb G. Sur les points singuliers d'une forme de Pfaff compl'etement intÇegrable ou d'une fonction numérique. *Comptes Rendus de L'Acad emie ses S eances, Paris* 1946; **222**: 847–849.
14. Meyer M, Desbrun M, Schröder P, Barr A. Discrete differentialgeometry operators for triangulated 2-manifolds. In *Proceedings of Vis-Math*, 2002; pp. 35–57.
15. Hilaga M, Shinagawa Y, Kohmura T, Kunii TL. Topology matching for fully automatic similarity estimation of 3D shapes. *SIGGRAPH* 2001; 203–212.
16. Sagi Katz, Ayellet Tal. Hierarchical Mesh Decomposition using Fuzzy Clustering and Cuts, SIGGRAPH 2003. *ACM Transactions on Graphics* 2003; **22**(3): 954–961.
17. Edmonds J, Karp RM. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of ACM* 1972; **19**(2): 248–264.
18. Yu Y, Zhou K, Xu D, Shi X, Baoh H, Guo B, Shum H-Y. Mesh editing with Poisson-based gradient field manipulation. *SIGGRAPH 2004. ACM Transactions on Graphics* 2004; **23**(3): 644–651.
19. Chao-Hung Lin, Tong-Yee Lee, Hung-Kuo Chu, Zhi-Yuan Yao. Progressive Mesh Metamorphosis. *Journal of Computer Animation and Virtual Worlds* 2005; **16**(3–4): 487–498.
20. Tao Ju. Robust repair of polygonal models. *SIGGRAPH 2004, ACM Transactions on Graphics* 2004; **23**(3): 888–895.
21. Tong-Yee Lee, Huang PH. Fast and institutive polyhedra morphing using SMCC mesh merging scheme. *IEEE Transactions on Visualization and Computer Graphics* 2003; **9**(1): 85–98.
22. Tong-Yee LEE, Chien-Chi Huang. Dynamic and adaptive morphing of three-dimensional mesh using control maps. *IEICE Transactions on Information and Systems* 2005; **E88-D**(3): 646–651.
23. Aaron W, Lee F, David Dobkin, Wim Sweldens, Peter Schröder. Multiresolution Mesh Morphing. Computer Graphics Proceedings (SIGGRAPH 99), pp. 343–350.
24. Marc Alexa. Merging polyhedral shapes with scattered features. Proceedings of Shape Modeling International 1999. 1999; pp. 202–210.

Authors' biographies:

Tong-Yee Lee was born in Tainan county, Taiwan, Republic of China, in 1966. He received his BS degree in computer engineering from Tatung Institute of Technology in Taipei, Taiwan, in 1988, his MS degree in computer engineering from National Taiwan University in 1990, and his PhD in computer engineering from Washington State University, Pullman, in May 1995. Now, he is a professor at the Department of Computer Science and Information Engineering, National Cheng-Kung University in Tainan, Taiwan, Republic of China. He serves as a Guest Associate Editor for *IEEE Transactions on Information Technology in Biomedicine* from 2000 to 2006. His current research interests include computer graphics, non-photorealistic rendering, image-based rendering, visualization, virtual reality, surgical simulation, 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.

Chih-Yuan Yao was born in Tainan, Taiwan in 1980. He received his BS and MS degrees in computer engineering from National Cheng-Kung University, Taiwan, in 2002 and 2003, respectively. He is currently working towards his PhD at the Department of Computer Science and Information Engineering, National Cheng-Kung University. His research interests include computer graphics.

Hung-Kuo Chu received his BS degree in computer science/engineering from the National Cheng-Kung University, Tainan, Taiwan, in 2003. Now, he is pursuing his PhD form the Department of Computer Science and Information Engineering, National Cheng-Kung University. His research interests include computer graphics.

Ming-Jen Tai was born in Taoyuan, Taiwan, Republic of China, in 1981. He received BS degree from the Department of Computer Science and Information Engineering, National Cheng Kung University, Tainan, Taiwan, in 2003, and the MS degree from the Department of Computer Science and Information Engineering, National Cheng Kung University, Tainan, Taiwan, in 2005. His research interests include computer graphics, computer vision and image processing.

Cheng-Cheieh Chen received his BS degree from the Department of Computer Science and Information Engineering, from National Chiao-Tung University, Hsinchu, Taiwan, in 2005. Now, he is pursuing his MS degree from the Department of Computer Science and Information Engineering, National Cheng-Kung University. His research interest includes computer graphics.