Multiresolution Mesh Morphing

Aaron W. F. Lee* Princeton University
David Dobkin† Princeton University
Wim Sweldens‡ Bell Laboratories
Peter Schröder§ Caltech

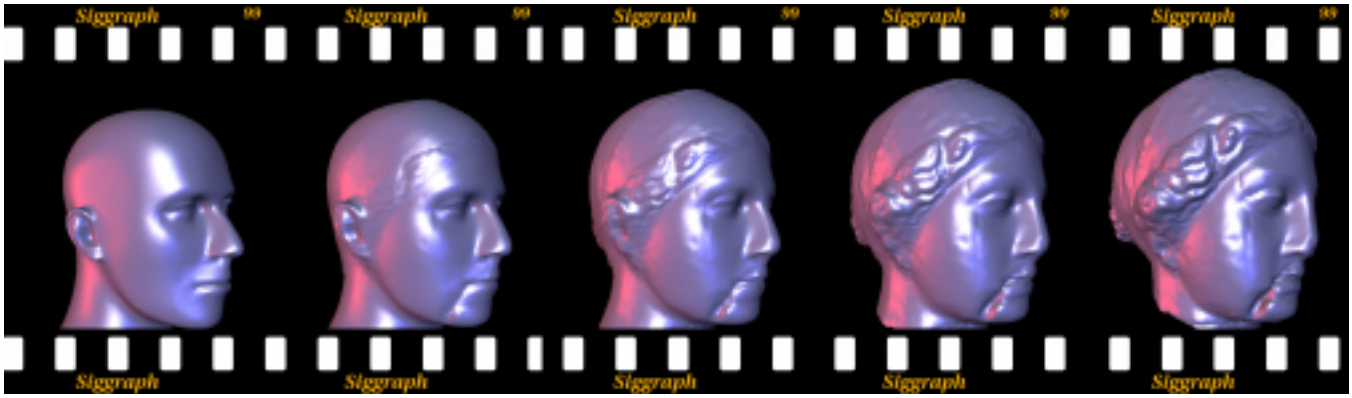

Figure 1: *Morph sequence between two topologically equivalent triangle meshes.*

Abstract

We present a new method for user controlled morphing of two homeomorphic triangle meshes of arbitrary topology. In particular we focus on the problem of establishing a correspondence map between source and target meshes. Our method employs the MAPS algorithm to parameterize both meshes over simple base domains and an additional harmonic map bringing the latter into correspondence. To control the mapping the user specifies any number of feature pairs, which control the parameterizations produced by the MAPS algorithm. Additional controls are provided through a direct manipulation interface allowing the user to tune the mapping between the base domains. We give several examples of æsthetically pleasing morphs which can be created in this manner with little user input. Additionally we demonstrate examples of temporal and spatial control over the morph.

CR Categories and Subject Descriptors: I.3.3 [**Computer Graphics**]: Picture/Image Generation - *Display Algorithms, Viewing Algorithms*; I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling - *Curve, Surface, Solid and Object Representations, Hierarchy and Geometric Transformations, Object Hierarchies*; I.3.6 [**Computer Graphics**]: Methodology and Techniques - *interaction techniques*; I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism - *animation*

Additional Keywords: Meshes, surface parameterization, mesh simplification, multiresolution, interpolation, morphing

*wailee@cs.princeton.edu
†dpd@cs.princeton.edu
‡wim@bell-labs.com
§ps@cs.caltech.edu

1 Introduction

Advances in 3D scanning and acquisition technology have made dense triangle meshes popular as representations of complex objects. These scanning devices typically create triangulations which form a surface of arbitrary topology. The large size of the meshes makes it difficult to manipulate them efficiently, an issue which can be addressed through the use of multiresolution representations.

Metamorphosis (or morphing) is the process of gradually changing a source object through intermediate objects into a target object. It has numerous applications from modeling to the generation of animation sequences for the movie and advertising industries. Much of the work done in this area has been on 2D metamorphosis, i.e., morphing of filmed or rendered sequences. 3D morphs on the other hand change the geometry of an object independent of subsequent rendering. Such morphs are significantly harder to compute and control.

Most techniques for morphing, both 2D and 3D, are based on a sparse set of user selected feature pairs. These are then used to establish a dense set of correspondences which in turn are used in subsequent interpolation between source and destination. The key to a successful method is its ability to achieve æsthetically pleasing morphs with few such feature pairs while providing means for very detailed control if so desired.

For example, patch based models, if they share the same control mesh, can be gracefully morphed into each other by associating corresponding control points. The underlying smooth surface representation provides the means to extend this sparse set of feature pairs in a predictable fashion to the entire surface. The situation is quite different when the source and destination surfaces are given as dense, irregular connectivity meshes with no obvious coarse level controls.

One possible approach is to transform the meshes into a sampled volumetric representation and apply 3D extensions of image morphing techniques. Instead we work with the meshes directly to avoid issues such as discretization artifacts, high computational cost, and difficulty of control, which volumetric methods generally exhibit.

Specifically we make the following contributions in this paper:

- **Dense correspondences for arbitrary meshes:** we address the problem of establishing dense correspondences between any two irregular connectivity meshes with the only requirement that they be topologically equivalent. This involves the construction of mappings from the fine meshes to their coarse base domains and of a mapping between the base domains. These mappings are realized through the *metamesh*, a topologically and geometrically merged version of source and destination meshes. The necessary computations are efficient enough to allow us to compute high quality morphs on meshes with thousands of triangles on a low end PC within several minutes.
- **Fine and coarse user control:** we provide easy and effective controls for the mapping from source to destination. In the case of fine features, such as vertices or connected sets of edges ("lines"), simply marking them on each mesh and pairing them up, is sufficient. Coarse control can be exercised by interactively modifying the mapping between the coarse source and destination domains. Providing a small set of feature pairs is generally sufficient to achieve æsthetically pleasing results.

Our algorithm proceeds by first creating parameterizations of the source and destination mesh using the MAPS (Multiresolution Adaptive Parameterization of Surfaces) algorithm of Lee et al. [24]. MAPS controls the parameterization using as few or as many features on the original meshes as the user desires. These two parameterizations are then put into correspondence through the construction of a map between the source and destination domains. This stage provides additional controls to the user to influence the morph in a broad fashion. The composition of these stages is used for subsequent shape interpolation.

2 Previous Work

Lazarus and Verroust [23] give an excellent survey of previous work on the 3D morphing problem. As they note, there are an unlimited number of ways to interpolate from one object to another. Such interpolations may be performed for geometry as well as attributes such as color. Algorithms for morphing are evaluated mainly by criteria related to the ease with which the results can be controlled and the æsthetic quality of the results themselves. Ease encompasses both the amount of work an artist has to invest, as well as the predictability of the result. Since æsthetic quality is subjective precise user control is important.

Most methods for morphing 3D objects use either discrete or combinatoric representations for the objects themselves. Discrete representations typically voxelize objects or their distance functions and aim to extend 2D morphing [2, 25, 36] algorithms to 3D.

Lerios et al. [26] extended the work of Beier and Neely [2] and used fields of influence of 3D primitives to warp volumes. Hughes [16] proposed a method working in the Fourier domain. This provided novel controls over the morph by treating individual frequency bands with different functions of time. He et al. [15] extended these ideas to a wavelet setting. Whitaker and Breen [35] performed morphing through the application of evolution equations. Payne [28] described a distance-field volumetric cross-dissolving technique.

The main advantage of volumetric methods is the ease with which they support changing genus. This comes at the price of having to reduce a model to a sampled representation on a finite grid. Since the grid is three dimensional, memory and computation costs can be prohibitive, limiting the visual fidelity of the results.

The alternative is to work directly on boundary representations such as polygonal meshes or patch complexes. Methods for this approach [20, 27, 29, 22, 30, 33, 8, 9, 6, 17, 14] have to first solve the *vertex correspondence* problem, i.e., computing the association of vertices or triangles between the source mesh and the target mesh. Lazarus and Verroust identify this as the key problem and it forms the focus of our paper.

Many approaches to the correspondence problem have been described, but there appears to be no general solution. Kent et al. [20] merged the mesh connectivities under a projection. This works well for star-shaped, swept, or revolutionary objects. Kaul and Rossignac [19] computed the Minkowski sum of scaled versions of the models which works well when the polyhedra are convex. For details of other warping and morphing techniques the reader is referred to [13, 36].

The work closest in spirit to ours is that of Gregory et al. [14] and Kanai et al. [18]. Gregory et al. give a method that allows the user to specify pairs and then decompose the polyhedron into patches. Patches of the source and target meshes are paired and morphed. This approach allows them to morph a broad class of objects. However it requires the user to outline the *entire* network of top level patch boundaries. Especially for large meshes this can be slow and tedious.

Kanai et al. have also extended their previous work [17], which used harmonic maps for morphing, to arbitrary topology triangle meshes. The basic idea is to define reference shapes by using vertex-to-vertex correspondences between the two meshes. The reference shape defines a partition of the mesh and each of the partitioned meshes is embedded into a polygonal region in the plane through a harmonic map. By overlapping those two embedded meshes, they establish correspondence between them. The number of partitions has to be identical so that they can be paired up and mapped to the same plane.

Their harmonic map computations are performed at the finest level while we only invoke such a solver for the coarse base domains. Additionally, we only require a small set of feature pairs and the coarse domain of the two meshes can be quite different to better adapt to the geometries, providing more flexibility. The efficiency of our method allows us to do this for relatively large meshes ensuring that the final surface renderings are of high visual quality.

3 Computing the Correspondence Map

As discussed above the key problem in morphing from one mesh to another is the establishment of the correspondence map with suitable user controls. In this section we describe the different stages we employ to compute the correspondence map. To do so we first fix some notation.

Notation When describing meshes mathematically, it is useful to separate the topological and geometric information. To this end we introduce some notation inspired by [32]. We denote a triangle mesh as a pair $(\mathcal{P}, \mathcal{K})$, where $\mathcal{P}$ is a set of N point positions $p_i = (x_i, y_i, z_i) \in \mathbf{R}^3$ with $1 \leq i \leq N$, and $\mathcal{K}$ is an *abstract simplicial complex* which contains all the topological, i.e., adjacency information. The complex $\mathcal{K}$ is a set of subsets of $\{1, \ldots, N\}$. These subsets are called simplices and come in 3 types: vertices $v = \{i\} \in \mathcal{K}$, edges $e = \{i, j\} \in \mathcal{K}$, and faces $f = \{i, j, k\} \in \mathcal{K}$, so that any non-empty subset of a simplex of $\mathcal{K}$ is again a simplex of $\mathcal{K}$, e.g., if a face is present so are its edges and vertices.

The *geometric realization* $\varphi(a)$ for $a \in \mathcal{K}$ is the strictly convex hull of all points p_i with $i \in a$. Thus $\varphi(\{i\}) = p_i$, $\varphi(\{i, j\})$ is the open line segment between p_i and p_j, and $\varphi(\{i, j, k\})$ is the open triangle between p_i, p_j, and p_k. The geometric realization $\varphi(\mathcal{K})$ is given by $\cup_{a \in \mathcal{K}} \varphi(a)$ and forms a polyhedron embedded in $\mathbf{R}^3$.

Two vertices $\{i\}$ and $\{j\}$ are *neighbors* if $\{i, j\} \in \mathcal{K}$. A set of vertices is *independent* if no two vertices are neighbors. A set of vertices is *maximally independent* if no larger independent set contains it. The 1-ring neighborhood of a vertex $\{i\}$ is the set $\mathcal{V}(i) = \{j \mid \{i, j\} \in \mathcal{K}\}$. The *degree* of a vertex is its number of neighbors.

3.1 Overview of the Algorithm

In our setting we have two meshes: the *source* mesh $(\mathcal{S}, \mathcal{K}_s)$ with N_s vertices and the target mesh $(\mathcal{T}, \mathcal{K}_t)$ with N_t vertices. Our goal is the construction of a correspondence map $\mathbf{M}$ between $\varphi(\mathcal{S})$ and $\varphi(\mathcal{T})$. The correspondence map has to be a bijection to avoid cracks and folds in the morph. Note that in general the mapping $\mathbf{M}(s_i)$ of a vertex of the source is not a vertex of the target, but instead lies somewhere in a target triangle.

In the first stage we apply the MAPS algorithm [24] to both source and target mesh, constructing coarse base domains $(\mathcal{S}^{(0)}, \mathcal{K}_s^{(0)})$ and $(\mathcal{T}^{(0)}, \mathcal{K}_t^{(0)})$ through a simplification hierarchy, as well as two bijective mappings $\Pi_s : \varphi(\mathcal{S}) \rightarrow \varphi(\mathcal{S}^{(0)})$ and $\Pi_t : \varphi(\mathcal{T}) \rightarrow \varphi(\mathcal{T}^{(0)})$.

Next we compute a correspondence map $\mathbf{M}^{(0)}$ between the source base domain $\varphi(\mathcal{S}^{(0)})$ and target base domain $\varphi(\mathcal{T}^{(0)})$. Because the base domains are coarse this map can be computed quickly. The final correspondence map between the original meshes is then given as:

$$\mathbf{M} : \varphi(\mathcal{S}) \rightarrow \varphi(\mathcal{T}) \;\text{ with }\; \mathbf{M} = \Pi_t^{-1}\, \mathbf{M}^{(0)}\, \Pi_s. \qquad (1)$$

The user can control the computation of the correspondence map to the extent desired by specifying features in the original meshes. The MAPS algorithm ensures that these features are mapped to edges in the base domain. The part of the mapping $\mathbf{M}^{(0)}$ that is not determined by the user defined feature pairs is computed automatically but can still be adjusted by the user. Feature pairs can be given by vertices, such as the tip of the nose, and lines which are a sequence of connected edges such as the mouth (see Figure 2). The beginning and end points of a feature lines are also feature vertices.

The overall structure of the algorithm is illustrated by the commutative diagram in Figure 2. On the top row are the source and target meshes. The bottom row shows the corresponding source and target base domains. The user specified feature points and lines are highlighted in red (resp. yellow). All maps respect the corresponding feature pairs.

A Brief Review of MAPS The MAPS algorithm uses a mesh hierarchy built through successive removal of a maximally independent set of vertices [10], followed by retriangulation of the resulting holes. By never removing any of the feature points, we can assure that they are contained in the base domain. Say the user specified U feature points and assume the corresponding vertex indices are numbered from 1 to U. The parameterization is built so that

$$\Pi_s(s_i) = s_i \;\text{ and }\; \Pi_t(t_i) = t_i \;\text{ for }\; 1 \leq i \leq U. \qquad (2)$$

In case of a feature line, the parameterization will map all the points of the original feature line to a sequence of edges (possibly one) in the base domain.

Note that using linear interpolation the maps Π are defined for *every* point on the mesh, not only the vertices. The map Π^{-1} can also be computed for every point on the base domain using a point location algorithm [24].

3.2 The Base Domain Correspondence Map

Construction of the base domain correspondence map consists of the following steps:

- globally align the source and destination base domains and project the source base domain to the target base domain;
- apply an iterative relaxation procedure to improve the mapping;
- user adjustment of the coarse correspondence to produce the final mapping.

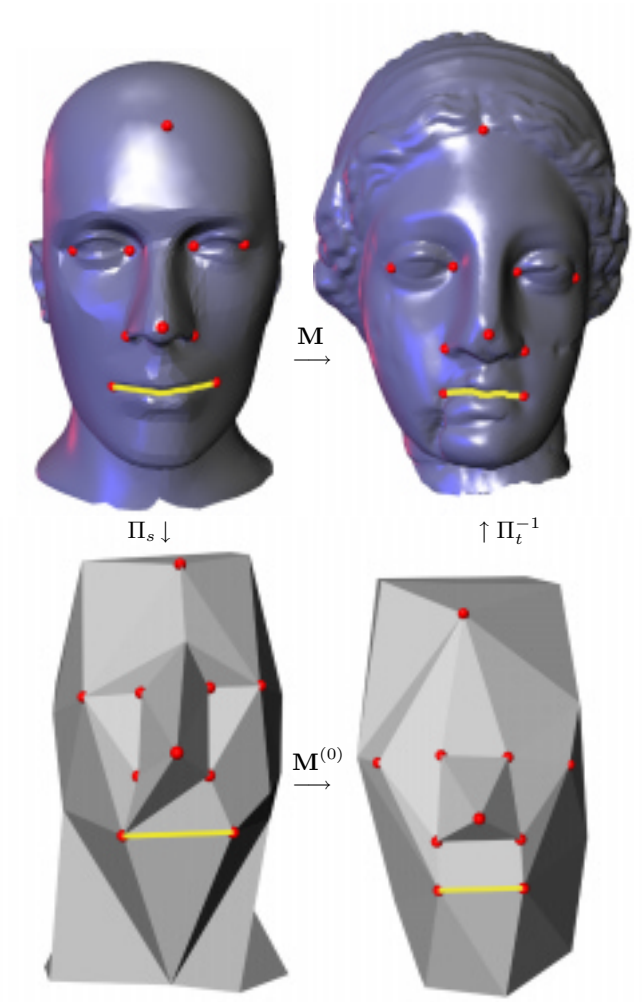

Figure 2: *Overview of the correspondence map computation. The user specifies pairs of feature points (red) and lines (yellow) in the original meshes (top). We then use MAPS to compute mappings Π_s and Π_t between the original meshes and the respective base domains (bottom). Next we compute the correspondence map $\mathbf{M}^{(0)}$ for the base domains. The final correspondence map $\mathbf{M}$ follows from composing these maps as $\Pi_t^{-1}\mathbf{M}^{(0)}\Pi_s$. As all the individual maps respect the feature pairs, so does the final map $\mathbf{M}$.*

Global alignment of base domains Given that feature points are guaranteed to be in the base domain, we can define their correspondence map as $\mathbf{M}^{(0)}(s_i) = t_i$ for $1 \leq i \leq U$. We still have to establish correspondences for the vertices of the source base domain which are not feature points. Assume that these points have indices $U+1 \ldots N_s^{(0)}$. We now need to find suitable positions for $\mathbf{M}^{(0)}(s_i)$ for $U+1 \leq i \leq N_s^{(0)}$ on the target base domain. This procedure begins by globally aligning the two base domains and then computing a starting guess for $\mathbf{M}^{(0)}(s_i)$ as the projection of s_i onto the closest triangle of $\varphi(\mathcal{K}_t^{(0)})$.

The global alignment can be done either manually or semi-automatically (see, e.g., [3, 7, 12]), and we have used Chen and Medioni's method with good success. Sometimes user intervention is required for the initial alignment if the source and the target objects are significantly different from each other. The initial projection is improved through an iterative relaxation procedure. Relaxation in planar settings is fairly straightforward and well understood. Computing relaxation *on* a mesh is non-trivial. Indeed,

a linear combination of neighboring points typically no longer lies on the mesh. To address this issue we base our relaxation algorithm on shortest path computations. Our relaxation method is similar to Turk's retiling technique [34]. He retiles a polygonal model by relaxing the new sample points so that they are evenly distributed over the model.

Relaxation *on* a Mesh Assume the guess for $\mathbf{M}^{(0)}(s_i)$ lies in a triangle $\varphi(t)$ ($t \in \mathcal{T}^{(0)}$) of the target base domain. The neighbors of s_i as defined by the source base domain connectivity, i.e., the $\mathbf{M}^{(0)}(s_j)$ with $j \in \mathcal{V}_s^{(0)}(i)$, need not lie in $\varphi(t)$. This is illustrated in Figure 3. The center vertex is $v = \mathbf{M}^{(0)}(s_i)$ and its neighbors $\mathbf{M}^{(0)}(s_j)$ are denoted by v_j. Compute the shortest paths between v and each of the v_j. Denote their lengths as measured on the mesh by l_j. The intersection between the boundary of $\varphi(t)$ and each shortest path is given by v_j' which define normalized directions:

$$\vec{d_j} = \frac{v_j' - v}{\|v_j' - v\|},$$

indicated by the bold arrows. The new, relaxed position is given by

$$v := (1 - \xi)v + \xi \sum_j \frac{\vec{d_j}}{l_j},$$

where the underrelaxation parameter $\xi < 1$ is chosen to assure that v moves no further than the boundary of $\varphi(t)$. This allows us to gracefully move into a neighboring triangle in the next iteration. Iterating this relaxation procedure will evenly distribute the source domain vertices on the target base domain. For feature

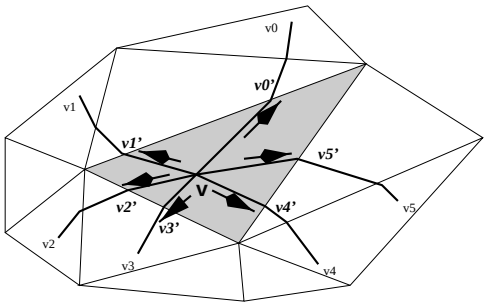

Figure 3: *Relaxation of source base domain vertices on the target base domain. The vertex is moved in a direction computed as a weighted average of the directions given by the shortest path (bold arrows).*

lines, i.e., a sequence of connected edges on the finest level, MAPS ensures their parameterization over a sequence of feature edges in the base domain (possibly one). Any vertex along such a source domain chain is mapped to the corresponding destination domain chain through linear scaling between the already fixed first and last vertices. These points are then also held fixed during relaxation.

Shortest Path Computation In general computing the exact shortest path between two points on a mesh is a difficult problem. Instead we use the method proposed by Lanthier et al. [21] to approximate the shortest path. Prior to the computation we introduce intermediate edge points (called *Steiner Points*) which subdivide each edge and construct a complete graph within each triangle. We do this for each triangle. Approximate shortest paths are calculated based on this graph using Dijkstra's algorithm [1].

Boundaries Notice that the mesh may contain boundaries. A boundary of a mesh ∂M is a closed loop which consists of a set of edges. Such boundaries must also be identified in the same manner as an open chain (feature line). In particular this implies that source and domain should have the same number of holes.

Caution We note that this relaxation algorithm depends on the user fixing some feature points in order to reduce the degrees of freedom, e.g., in the mannequin head to Spock head morphing, the user fixes one vertex at the top and four along the neck boundary. This works well in cases where the source and target base domains are similar. If the base domains are highly dissimilar, shortest paths may cross and flipped triangles may appear. The interface can flag them by computing their signed area. The problem can be addressed by fixing more points and repeating the relaxation.

4 Additional Controls

We can treat the result of the above relaxation procedure as an initial solution to the base domain correspondence. In general the base domain of the source mesh and the target mesh are quite different and this initial solution may not be what the user desires. The user can exercise further control in the base domain correspondence mapping as we now describe.

We allow the user to map a vertex on the source base domain onto any point on the target base domain to adjust the mapping. This is done by user interface controls that allow the user to map a vertex on one domain to a vertex, point on an edge, or a point in a triangle on the other base domain.

Since the number of base domain vertices is small, adjustment can be done quickly. Our experience is that for similar objects such as two heads, little (if any) further adjustment is needed. In the case of dissimilar objects the adjustment is more involved as illustrated in the horse to rabbit morph in Section 5.

4.1 Extending $\mathbf{M}^{(0)}$

At this point we have computed $\mathbf{M}^{(0)}$ only for the vertices of $\mathcal{S}^{(0)}$. We next describe how to compute the map for *any* point of the source base domain (see Figure 4). Consider a triangle $\{i, j, k\} \in \mathcal{K}_s^{(0)}$ of the source base domain. Put its vertices on the target base domain using $\mathbf{M}^{(0)}$ and call them $I = \mathbf{M}^{(0)}(s_i)$, $J = \mathbf{M}^{(0)}(s_j)$, and $K = \mathbf{M}^{(0)}(s_k)$. The points I, J, and K in general do not lie within a single triangle of the target base domain. We use the already computed shortest paths $\overline{IJ}$, $\overline{JK}$, and $\overline{KI}$ on the target base domain (thick line). This outlines a "triangular shaped" region IJK on the target base domain (shaded). The triangles of the target base domain cut this region IJK into polygons each of which we retriangulate. We next use the piecewise linear harmonic map technique of Eck et al. [11]. By taking the I, J, K as boundary points and mapping them to s_i, s_j, and s_k we compute a mapping between IJK and the corresponding source triangle. In this example only one interior vertex needs to be relaxed. In general the computation is fast since only a handful of vertices are involved.

By doing this for every triangle of the source base domain we effectively build the map $\mathbf{M}^{(0)}$ for every point of the source base domain.

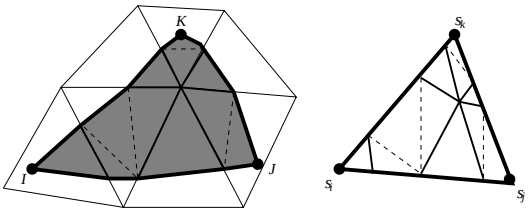

Figure 4: *The source base domain triangle maps to a triangular shaped region (shaded) on the target base domain. We compute the harmonic mapping of this region to a triangle so as to place the target base domain vertices on the source base domain.*

4.2 The Final Correspondence Map

Once we have the base domain correspondence map we can place any source mesh point onto the target using the composition $\Pi_t^{-1}\,\mathbf{M}^{(0)}\,\Pi_s$. The inverse map on the target mesh is computed using a point location algorithm [5] on the target base domain. This allows us to map *any* point of $\varphi(\mathcal{S})$ onto $\varphi(\mathcal{T})$.

In the morph we can now interpolate between s_i and $\mathbf{M}(s_i) \in \varphi(\mathcal{T})$. However, the source connectivity $\mathcal{K}_s$ and target connectivity $\mathcal{K}_t$ are quite different. Starting the morph from $(\mathcal{S}, \mathcal{K}_s)$ would get us $(\mathbf{M}(\mathcal{S}), \mathcal{K}_s)$, which are the source vertices placed on the target mesh with the *source* connectivity. This is a fairly arbitrarily remeshed version of the target. Even though it roughly captures the geometry of the target, it will still have numerous artifacts. Our goal is to reach exactly $(\mathcal{T}, \mathcal{K}_t)$, not some remeshed version. Therefore we introduce the notion of a metamesh.

4.3 The Metamesh

The purpose of the *metamesh* $(\mathcal{P}, \mathcal{K}_p)$ is to combine the source connectivity $\mathcal{K}_s$ and target connectivity $\mathcal{K}_t$. We want to do this in such a way that for certain metamesh point positions $\mathcal{P} = \mathcal{P}_s$, the geometric realization $\varphi(\mathcal{P}_s)$ coincides with $\varphi(\mathcal{S})$, while for other point positions $\mathcal{P} = \mathcal{P}_t$, the geometric realization coincides with $\varphi(\mathcal{T})$.

To define the abstract complex $\mathcal{K}_p$ we need to find vertices, edges, and faces. We start out by defining the vertices of $\mathcal{K}_p$ as $\mathcal{V}_p = \mathcal{V}_s \cup \mathcal{V}_t \cup \mathcal{V}_i$ where $\mathcal{V}_s$ (resp. $\mathcal{V}_t$) are the vertices of $\mathcal{K}_s$ (resp. $\mathcal{K}_t$) and $\mathcal{V}_i$ are new vertices introduced by intersection of source and target mesh edges.

Tracing Edge Segments To find the connectivity of the metamesh we start out by drawing edges *on the target mesh* between the points $\mathbf{M}(\mathcal{S})$. Take two source vertices s_i and s_j with $\{i, j\} \in \mathcal{K}_s$, i.e., an edge of the finest level source mesh, and consider their placement $\mathbf{M}(s_i)$ and $\mathbf{M}(s_j)$ on the target. If they belong to the same target triangle, we can directly connect them with a line. Otherwise we connect them with a segmented line given by $\mathbf{M}(\varphi(\{i, j\}))$. This segmented line can be computed as follows. Start with the segment $\varphi(\{i, j\})$ and trace it through all the different piecewise linear conformal maps that make up the map Π_s. Each of these conformal maps are used in the MAPS algorithm for flattening a local neighborhood. Thus it is easy to check when the segment $\varphi(\{i, j\})$ breaks into two segments. Put these segments in a list. Start tracing the two new segments and whenever they break add the subsegments to the list. If at any time during this procedure two consecutive segments lie within the same triangle again, we can merge them. Continue this procedure until one arrives at a list of segments connecting $\Pi(s_i)$ and $\Pi(s_j)$ on the source base domain. Because of the merging and the fact that the base domain is so much coarser than the original, almost all lists will contain a single segment.

Given this sequence of segments trace them through $\mathbf{M}^{(0)}$, i.e., from the source base domain to the target base domain. $\mathbf{M}^{(0)}$ is also made up of local "flattenings," this time coming from the base domain correspondence map. Once again it is easy to check when segments break into further subsegments. After this step we have a list of segments connecting $\mathbf{M}^{(0)}(\Pi(s_i))$ and $\mathbf{M}^{(0)}(\Pi(s_j))$ on the target base domain. Finally the same procedure is applied through the mapping Π_t^{-1}. If at any time two consecutive segments lie within the same triangle we merge them. Finally we arrive at a list of segments connecting $\mathbf{M}(s_i)$ and $\mathbf{M}(s_j)$ on the target mesh. All segments except the first and last connect points that lie on the edges of the target mesh, otherwise they would have been merged. Take those intersection points and add them as vertices to the metamesh. Their position in $\mathcal{P}_t$ is given by the intersection point while their position in $\mathcal{P}_s$ is given by $\mathbf{M}^{-1}$ applied to the intersection point.

In practice it is easiest to find the intersection points in the target base domain and then map them through $\Pi_t^{(0)}$. This is illustrated in Figure 5. The points $\mathbf{M}^{(0)}(\Pi_s(s_i))$ and $\mathbf{M}^{(0)}(\Pi_s(s_j))$ lie in different triangles of the triangle base domain and are connected with a segmented line. Consider neighboring points of the target mesh $\Pi_t(t_k)$. The intersection points (black dots) can easily be found and mapped through Π_t^{-1} to form $\mathcal{P}_t$.

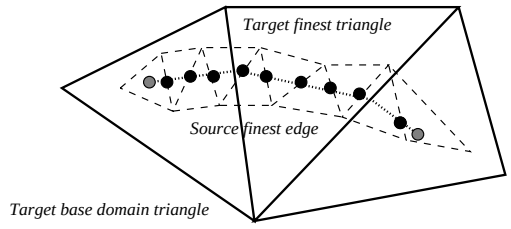

Figure 5: *Building the metamesh. The intersections between the source edge drawn on the target base domain and the target edges on the target base domain define the new vertices of the metamesh.*

Finally these segments have cut the target triangles into polygons. Retriangulate those polygons using a constrained Delaunay triangulation and add the new edges and triangles to $\mathcal{K}_p$. The metamesh is now done.

Numerical Stability In order to maintain numerical stability, we employ adaptive precision floating point exact arithematic for robust and fast evaluation of geometric predicates [31]. Also we take *coincidence issues* into account as mentioned in [20, 17]. The idea is fairly simple, instead of representing intersection points in $\mathbf{R}^3$, we represent them as barycentric coordinates in the target triangles. Hence, vertex to vertex correspondence is simply a permutation of $\{1, 0, 0\}$ and vertex to edge correspondence is permutation of $\{\alpha, \beta, 0\}$. This turns out to be very helpful in performing the constrained Delaunay triangulation since the "on-line" predicate does not suffer from numerical round off error.

Finding intersection points is easier if the source mesh contains triangles which are smaller than the target mesh triangles. Then many source edges will lie inside a single triangle of the target mesh and no path needs to be computed. Hence one may need to switch if there is a big difference in size. Once the map is computed one can switch back.

Properties of the Metamesh The metamesh has a number of interesting properties. For example, we can make the geometric realization of the metamesh look *exactly* like the geometric realization of both source and target mesh by letting its positions be either $\mathcal{P}_s$ or $\mathcal{P}_t$:

$$\varphi(\mathcal{S}, \mathcal{K}_s) = \varphi(\mathcal{P}_s, \mathcal{K}_p) \;\text{ and }\; \varphi(\mathcal{T}, \mathcal{K}_t) = \varphi(\mathcal{P}_t, \mathcal{K}_p).$$

In other words by taking the positions $\mathcal{P}_s$ the triangles of the metamesh line up to form the triangles of the source while by taking the positions $\mathcal{P}_t$ the triangles of the metamesh line up differently to form the triangles of the target.

In the worst case the size of the metamesh can grow as the product of the sizes of the source and target meshes as every edge of each mesh could intersect every edge of the other. In practice this is not the case. As the complexity of the mesh grows, the edge lengths decrease for a fixed geometric size. Since the meshes are about the same density, most of the intersections happen locally and thus are proportional to the degree of vertices which on average is a constant. As shown in Table 1, the metamesh size is no more than 10 times the size of the larger mesh in the examples we have considered.

The positions of intermediate meshes needed in the morph are given by $\mathcal{P} = \theta\mathcal{P}_s + (1-\theta)\mathcal{P}_t, \quad (0 \le \theta \le 1)$ and the connectivity is always $\mathcal{K}_p$. Letting θ smoothly vary between 0 and 1, these

meshes transform from source into target. It is well known that linear interpolation methods can cause self intersections or excessive shape distortion. Some previous 2D work [29] is geared towards avoiding kinks or shrinkage during polygon morphing. Unfortunately these methods work well only for models with similar shape. While we have had success with the our simple interpolation, more sophisticated methods would be desirable.

To time schedule the morph the user can now specify how θ will vary with time (t). The simplest solution is to let $\theta(t)$ vary linearly with time: $\theta = t$. To have a gentle fade-in and fade-out one can let $\theta(t) = 1/2 - 1/2\cos(\pi t)$. The user also has spatial control by letting θ depend on location: $\theta(t, i)$ with $\{i\} \in \mathcal{K}_p$. This can be used to morph certain regions before others as shown in the mannequin to Spock head example.

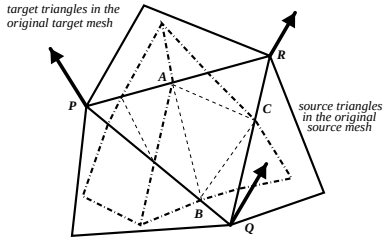

Figure 6: *Intersection between triangles at the source mesh and the target mesh, new edges (broken lines) are introduced for the constrained triangulation which preserves the source and target edges. The attribute vectors (bold arrows) for the target triangle PQR are shown.*

Attribute Interpolation and Rendering of the Metamesh We can also interpolate other attributes such as normal, texture and color information between the source and the target. Consider the most general case where the attributes are associated with each vertex per triangle. This allows us to morph between smooth and sharp objects. Consider the case when the metamesh is at the target ($\mathcal{P} = \mathcal{P}_t$). Remember that the metamesh has many more vertices than the original source/target mesh: In Figure 6 triangles (in the metamesh) that are created inside ΔPQR will have attributes derived from the attribute vector at P, Q and R using barycentric interpolation.

This hints at another application. Assume we have a scanned mesh like a human head with a scanned texture but no texture map. Say we want to put this texture on a model like the mannequin head. Once we have a correspondence map between the two heads, one can simply transfer the texture.

5 Results

We have implemented our system on a Pentium Pro 200MHz PC and used it to produce a number of different morphs, described below.

Mannequin to Venus Figure 1 shows a number of frames from this sequence. The user only needed 5 minutes to associate the features (Figure 2) and adjust the mapping. Note how ears morph to ears, lips to lips, nose to nose, and eyes to eyes. The source mesh (Mannequin head) is created by using the Loop subdivision scheme to enhance the smoothness. However, its special structure was not used.

Cup to Donut This morph illustrates that our system can handle higher genus manifolds as well as morph fairly dissimilar objects. See the color plate (Figure 8 top). The user needed 30 minutes to associate features and adjust the mapping. Note how the cup turns itself inside out to deform to the torus. In morphing two non-zero genus objects, not only do they have to satisfy the homeomorphism condition, they must also be *"tamely homeomorphic"* [4]. S is tamely homeomorphic to T if there is a homeomorphism of $\mathbf{R}^3$ onto itself that carries S onto T.

Mannequin to Spock Here we show an example of spatial control. See the color plate (Figure 8, middle). We first put the hair of Spock onto the mannequin head (middle frame) and then morph the rest of the face.

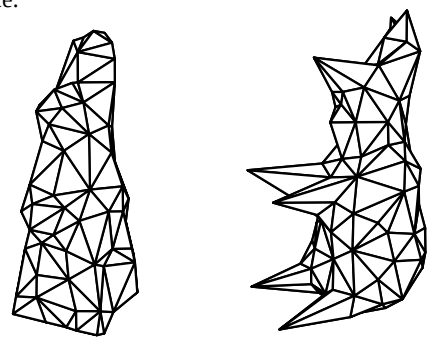

Figure 7: *Modification of the rabbit base domain to more closely match the horse base domain.*

Horse to Rabbit This is another example of morphing dissimilar objects. See the color plate (Figure 8 bottom). The user spends almost an hour to establish the base domain correspondences. There are three reasons: there are more feature pairs (60) which the user has to match due to the fact that the rabbit has long ears while the horse has small ones and the horse has a very long neck that the rabbit does not have. Effectively the user has to stretch the horse base domain using the additional control tools to match the rabbit's. Also, the legs of the horse are very noticeable, prominent features. The rabbit has no such features that can be identified. In order to create the morph, we allow the user further control by altering the shape of the rabbit's base domain (see Figure 7). By creating four "legs" on the rabbit base domain the base domain mapping can be quite smooth. These changes induced changes to the finest level correspondence through the MAPS parameterization.

6 Conclusions and Future Work

We have demonstrated an effective and easy to use system for user controlled morphing of dense, arbitrary connectivity triangle meshes of arbitrary topology. Future research can be pursued in several directions:

- The main restriction of our method is the requirement that source and target share the same genus. Thus fundamental work on extending MAPS to deal with genus changes is needed.
- Once a dense correspondence map is established the characteristics of the morph can be greatly influenced by the interpolation functions used. We have only explored spatially varying linear interpolation and more sophisticated controls would be desirable in production work.
- We can compute a wavelet transform on the metamesh and give the user the option to schedule different morphing speeds for different scales/frequencies as in [15].
- The user can have even more control over the actual morph by editing the metamesh in certain key frames. The morph will then smoothly adjust itself to those key frames.
- In case the source and target are quite dissimilar, the user needs to spend more time to guide the correspondence map. More tools which naturally combine user control with automated computation are needed.

Source-Target	Source size (triangles)	Target size (triangles)	Metamesh size (triangles)	Feature pairs	Corresp. map time	Metamesh time	User time
mann-venus	5422	90709	225502	24	3’	19’	5’
cup-donut	8452	2048	43188	30	1’20”	4’	30’
mann-spock	5422	14100	75427	24	1’	7’	5’
horse-rabbit	21130	21582	220201	60	22’	27’	60’

Table 1: *Selected statistics for the examples discussed in the text. All times were measured on a 200 MHz PentiumPro.*

Acknowledgment Work of David Dobkin and Aaron Lee was supported in part by NSF (CCR-9731535) and the US ARO (DAAH04-96-1-0181). Aaron Lee was also supported by a Wu Graduate Fellowship. Peter Schröder was supported in part by NSF (ACI-9624957, ACI-9721349, and DMS-9874082). Other support was provided by Alias|wavefront and a Packard Foundation Fellowship. Special thanks to Adam Finkelstein for many interesting and stimulating discussions; the University of Washington for providing the Spock and mannequin head; Cyberware for providing the horse and rabbit; Arthur Gregory for providing the cup and donut.

References

[1] AHO, A. V., HOPCROFT, J. E., AND ULLMAN, J. D. *Data Structures and Algorithms*. Addison-Wesley, 1983.

[2] BEIER, T., AND NEELY, S. Feature-based image metamorphosis. In *Computer Graphics (SIGGRAPH ’92 Proceedings)*, 35–42, 1992.

[3] BESL, P. J., AND MCKAY, N. D. A Method for Registration of 3-D Shapes. *IEEE Trans. on Pattern Anal. and Machine Intelligence 14*, 2 (Feb. 1992), 239–258.

[4] BOYER, M., AND STEWART, N. F. Modeling spaces for toleranced objects. *Int. J. Robotics Research 10*, 5 (1991), 570–582.

[5] BROWN, P. J. C., AND FAIGLE, C. T. A Robust Efficient Algorithm for Point Location in Triangulations. Tech. rep., Cambridge University, February 1997.

[6] CARMEL, D. C. E. Warp-guided object-space morphing. *The Visual Computer 13*, 9–10 (1998), 46–478. ISSN 0178-2789.

[7] CHEN, Y., AND MEDIONI, G. Object Modeling by Registration of Multiple Range Images. *Int. J. of Image and Vision Computing 10*, 3 (Apr. 1992), 145–155.

[8] DECARLO, D., AND GALLIER, J. Topological Evolution of Surfaces. In *Graphics Interface ’96*, 194–203, May 1996.

[9] DECAUDIN, P. Geometric Deformation by Merging a 3D-Object with a Simple Shape. In *Graphics Interface ’96*, 55–60, May 1996.

[10] DOBKIN, D., AND KIRKPATRICK, D. A Linear Algorithm for Determining the Separation of Convex Polyhedra. *Journal of Algorithms 6* (1985), 381–392.

[11] ECK, M., DEROSE, T., DUCHAMP, T., HOPPE, H., LOUNSBERY, M., AND STUETZLE, W. Multiresolution Analysis of Arbitrary Meshes. In *Computer Graphics (SIGGRAPH ’95 Proceedings)*, 173–182, 1995.

[12] FAUGERAS, O. D., AND HERBERT, M. The Representation, Recognition, and Locating of 3D Objects. *The Int. J. Robotics Research 5*, 3 (1986), 27–49.

[13] GOMES, J., DARSA, L., COSTA, B., AND VELHO, L. *Warping and Morphing of Graphical Objects*. Morgan Kaufmann, San Francisco, Calif., 1998.

[14] GREGORY, A., STATE, A., LIN, M., MANOCHA, D., AND LIVINGSTON, M. Feature-based Surface Decomposition for Polyhedral Morphing. Tech. Rep. TR98-014, Department of Computer Science, University of North Carolina - Chapel Hill, Apr. 14 1998.

[15] HE, T., WANG, S., AND KAUFMAN, A. Wavelet-Based Volume Morphing. In *Proceedings of the Conference on Visualization*, 85–92, Oct. 1994.

[16] HUGHES, J. F. Scheduled Fourier volume morphing. In *Computer Graphics (SIGGRAPH ’92 Proceedings)*, 43–46, 1992.

[17] KANAI, T., SUZUKI, H., AND KIMURA, F. Three-dimensional geometric metamorphosis based on harmonic maps. *The Visual Computer 14*, 4 (1998), 166–176.

[18] KANAI, T., SUZUKI, H., AND KIMURA, F. Metamorphosis of Arbitrary Triangular Meshes with User-Specified Correspondence. *IEEE Computer Graphics and Applications* (to appear).

[19] KAUL, A., AND ROSSIGNAC, J. Solid-Interpolating Deformations: Construction and Animation of PIPs. In *Eurographics ’91*, 493–505, Sept. 1991.

[20] KENT, J. R., CARLSON, W. E., AND PARENT, R. E. Shape transformation for polyhedral objects. In *Computer Graphics (SIGGRAPH ’92 Proceedings)*, 47–54, 1992.

[21] LANTHIER, M., MAHESHWARI, A., AND SACK, J.-R. Approximating Weighted Shortest Paths on Polyhedral Surfaces. In *6th Annual Video Review of Computational Geometry, Proc. 13th ACM Symp. Computational Geometry*, 485–486, 4–6 June 1997.

[22] LAZARUS, F., AND VERROUST, A. Feature-based shape transformation for polyhedral objects. In *The 5th Eurographics Workshop on Animation and Simulation*, 1–14, 1994.

[23] LAZARUS, F., AND VERROUST, A. Three-dimensional metamorphosis: a survey. *The Visual Computer 14* (1998), 373–389.

[24] LEE, A. W. F., SWELDENS, W., SCHRÖDER, P., COWSAR, L., AND DOBKIN, D. MAPS: Multiresolution Adaptive Parameterization of Surfaces. *Computer Graphics (SIGGRAPH ’98 Proceedings)* (1998), 95–104.

[25] LEE, S., CHWA, K., SHIN, S. Y., AND WOLBERG, G. Image Metamorphosis Using Snakes and Free-Form Deformations. In *Computer Graphics (SIGGRAPH ’95 Proceedings)*, 439–448, 1995.

[26] LERIOS, A., GARFINKLE, C. D., AND LEVOY, M. Feature-Based Volume Metamorphosis. In *Computer Graphics (SIGGRAPH ’95 Proceedings)*, 449–456, 1995.

[27] PARENT, R. E. Shape transformation by boundary representation interpolation: a recursive approach to establishing face correspondences. *The Journal of Visualization and Computer Animation 3*, 4 (Oct.–Dec. 1992), 219–239.

[28] PAYNE, B. A., AND TOGA, A. W. Distance field manipulation of surface models. *IEEE Computer Graphics and Applications 12*, 1 (Jan. 1992), 65–71.

[29] SEDERBERG, T. W., GAO, P., WANG, G., AND MU, H. 2D Shape Blending: An Intrinsic Solution to the Vertex Path Problem. In *Computer Graphics (SIGGRAPH ’93 Proceedings)*, vol. 27, 15–18, Aug. 1993.

[30] SHAPIRA, M., AND RAPPOPORT, A. Shape Blending Using the Star-Skeleton Representation. *IEEE Computer Graphics and Applications 15*, 2 (1995), 44–50.

[31] SHEWCHUK, J. R. Adaptive Precision Floating-Point Arithmetic and Fast Robust Geometric Predicates. *Discrete & Computational Geometry 18*, 3 (Oct. 1997), 305–363.

[32] SPANIER, E. H. *Algebraic Topology*. McGraw-Hill, New York, 1966.

[33] SUN, Y. M., WANG, W., AND CHIN, F. Y. L. Interpolating Polyhedral Models using Intrinsic Shape Parameters. In *Pacific Graphics ’95*, Aug. 1995.

[34] TURK, G. Re-Tiling Polygon Surfaces. *Computer Graphics (SIGGRAPH ’92 Proceedings)* (1992), 55–64.

[35] WHITAKER, R., AND BREEN, D. Level-Set Models for the Deformation of Solid Objects. In *Proceedings of the Third International Workshop on Implicit Surfaces*, 19–35, June 1998.

[36] WOLBERG, G. *Digital Image Warping*. IEEE Computer Society Press, 1990. IEEE Computer Society Press Monograph.

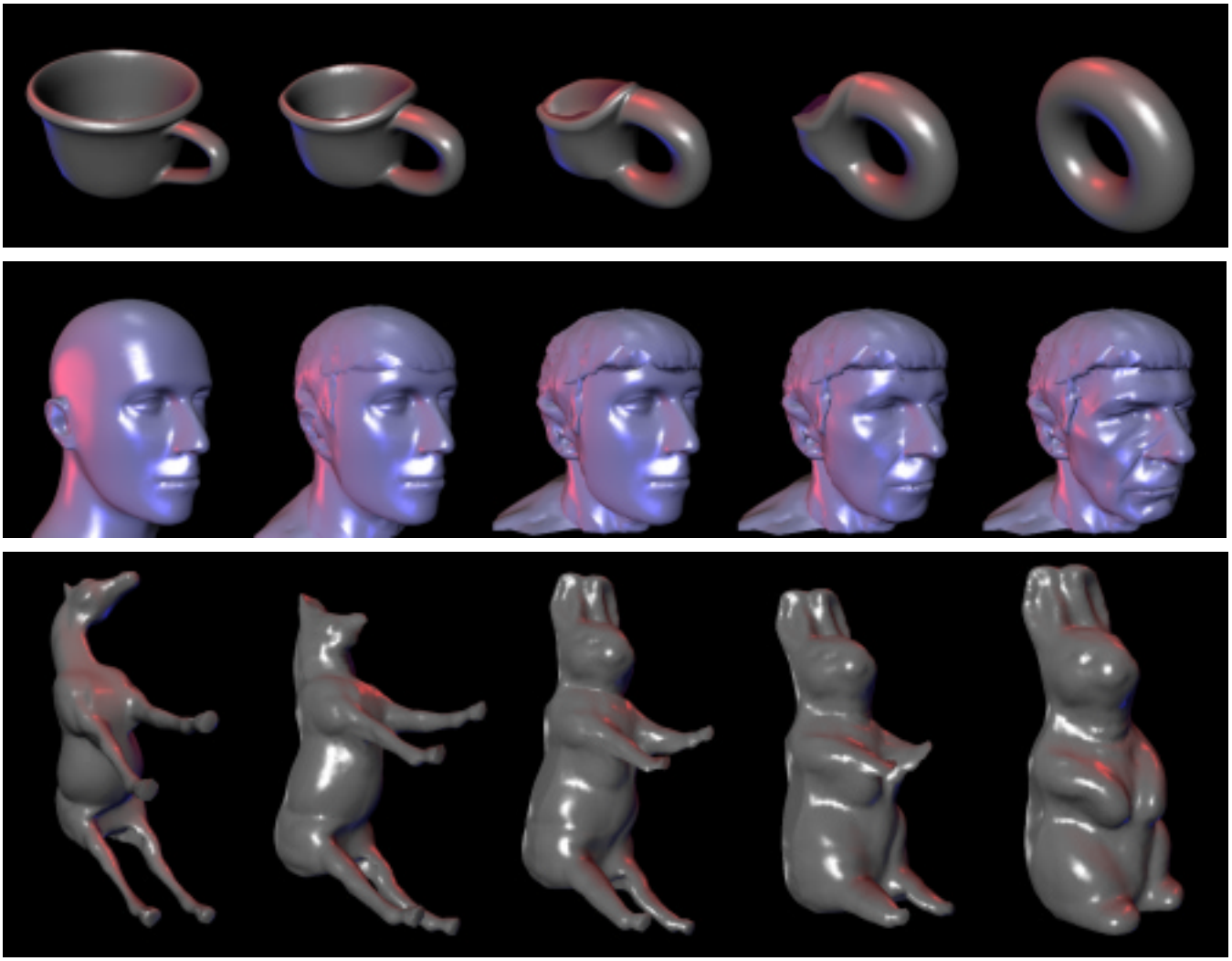

Figure 8: *Morphing gallery.*

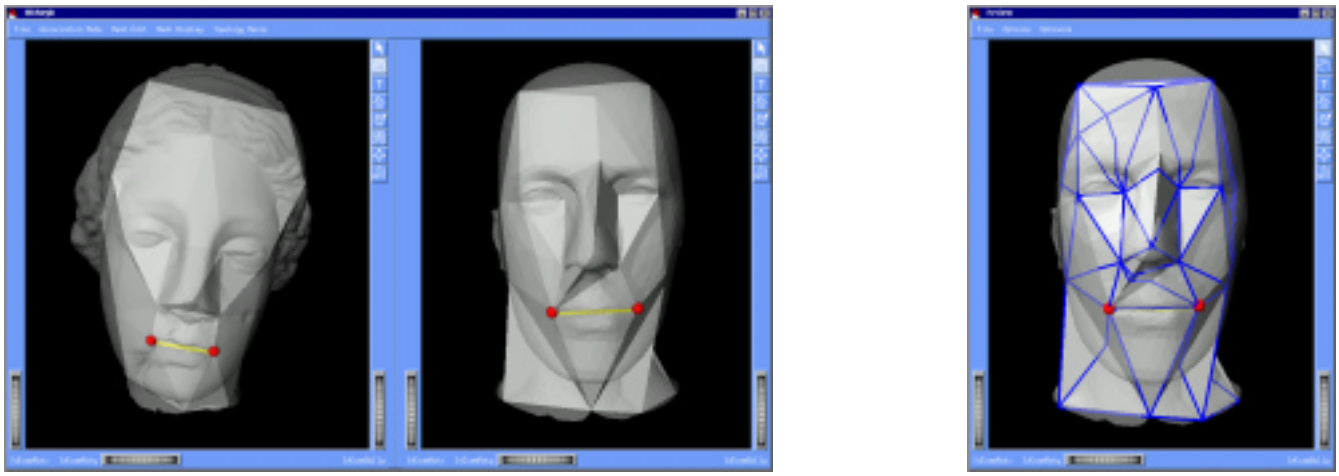

Figure 9: *User interface shows the finest and coarsest resolution of the source (Venus) and the target (Mannequin). It demonstrates the feature vertices and feature edges association (see Section 3.3). The right picture shows the result of mapping the source base domain edges (blue lines) onto the target base domain.*