# Feature-based Surface Decomposition for Correspondence and Morphing between Polyhedra 

Arthur Gregory,<br>Andrei State, Ming C. Lin, Dinesh Manocha, Mark A. Livingston<br>Department of Computer Science,<br>University of North Carolina at Chapel Hill<br>\{gregory,andrei,lin,dm,livingst\}@cs.unc.edu<br>http://www.cs.unc.edu/~geom/3Dmorphing


#### Abstract

We present a new approach for establishing correspondence between two homeomorphic 3D polyhedral models. The user can specify corresponding feature pairs on the polyhedra with a simple and intuitive interface. Based on these features, our algorithm decomposes the boundary of each polyhedron into the same number of morphing patches. A $2 D$ mapping for each morphing patch is computed in order to merge the topologies of the polyhedra one patch at a time. We create a morph by defining morphing trajectories between the feature pairs and by interpolating them across the merged polyhedron. The user interface provides high-level control as well as local refinement to improve the morph. The implementation has been applied to several complex polyhedra composed of thousands of polygons. The system can also handle non-simple polyhedra that have holes.


## 1 Introduction

Image and object morphing techniques have gained increasing importance in the last few years. Given two objects, metamorphosis involves producing a sequence of intermediate objects that gradually evolve from one object to another. The techniques have been used in a number of applications, including scientific visualization, education, entertainment, and computer animation. Morphing involves establishing a correspondence between the graphical objects and interpolating between them, in conjunction with blending their colors or textures.

Main Contribution: We present a new approach for establishing correspondence between two homeomorphic polyhedra. Initially the user selects some corresponding
elements called feature pairs. Although we borrow this term from previous morphing algorithms for images or 3D volumetric models [2, 24], our concept of a feature is closer to the sparse control mesh used in [10]. Our algorithm includes a simple and intuitive user interface for feature specification and automatically generates a feature net. Based on the feature nets, the algorithm decomposes the boundary of the polyhedra into morphing patches, computes a mapping for each morphing patch to a 2 D polygon, merges them, and constructs a merged polyhedron whose topological connectivity is a combination of the topological connectivity of the input polyhedra. The merged polyhedron has a morphing trajectory for each vertex to move along from one input polyhedron to the other. The overall complexity of the algorithm is $\mathbf{O}(K(m+n))$, where $K$ is a user-defined constant and $m, n$ correspond to the number of vertices in the two input polyhedra.

## 2 Related Work

Similar problems in 2D have been extensively studied in computer graphics. The set of algorithms can be classified into those that operate on raster images such as [2, 9, 23, 35, 36] and those that operate on geometric representations [14, 31, 32, 34]. There is considerable literature on morphing between 3D volumetric models [7, 8, 15, 24, 29].

Several approaches related to establishing correspondence between 3D polygonal objects for shape transformation and metamorphosis have been proposed. Graphics techniques based on deformations [1,33] and particle systems [30] can be used for object metamorphosis. Hong et al. [17] present an approach for polyhedral models that matches the faces whose
centroids are closest. Chen and Parent [6] present an algorithm to transform piecewise linear 2D contours and extend it to 3D cylindrical objects. Bethel and Uselton [4] add degenerate vertices and faces to two polyhedra until they have a common vertex neighborhood graph. Kaul and Rossignac [19] transform a pair of polyhedra by using their Minkowski sums. Hodgins and Pollard [16] have presented an algorithm for interpolating between control systems of dynamic models. Wyvill [37] has described an approach for implicit surfaces. Parent [27] has presented an approach that splits the surface of the model into pairs of sheets of faces and recursively subdivides them until they have the same topology. Parent [27, 28] has also described a method for deformation of polyhedral objects based on implicit functions. Kent et al. [20, 21] have presented a shape transformation algorithm for genus-0 polyhedra that involves projecting the models onto a sphere. Chen et al. [5] have produced 3D morphs of cylindrical images. Galin and Akkouche [12] have presented an algorithm for blob metamorphosis based on Minkowski sums. Lazarus and Verroust [22] have proposed a method based on skeletal curves. Kanai et al. [18] have presented an algorithm for shape transformation of genus-0 polyhedra using harmonic maps. DeCarlo and Gallier [10] have proposed a morphing technique that establishes correspondence by allowing the user to divide the surface into triangular and quadrilateral patches which can be projected onto a plane. Our overall approach shares their theme. However, we improve upon several restrictions in their technique making it easier for the user to specify correspondence between complicated models. For example, we remove the requirement that the user must subdivide the surfaces of the input polyhedra into patches that can each be directly projected onto a plane.

## 3 Terminology

The term polyhedron refers to an arrangement of polygons such that two and only two polygons meet at an edge. It is possible to traverse the surface of the polyhedron by crossing its edges and moving from one polygonal face to another until all polygons have been traversed by this continuous path [26]. Topology refers to the vertex/edge/face connectivity of a polyhedron. Simple polyhedra are all polyhedra that can be continuously deformed into a sphere. It is possible for a polyhedron to have holes. Non-simple polyhedra are topologically equivalent to a solid object with holes in it. In this paper, we assume that each face of a polyhedron is homeomorphic to a closed disk. The genus $g$ of a polyhedron is the maximum number of non-intersecting
loops that do not divide its surface into two regions. Moreover, we assume that the given polyhedra satisfy the Euler-Poincaré formula: $v-e+f-2(1-g)=0$, where $v, e, f$ and $g$ are the number of vertices, edges, faces and genus of the polyhedron respectively. The genus of a simple polyhedron is zero.

## 4 Overview

Given two homeomorphic polyhedra, our goal is to generate a morph that results in a smooth and gradual transition from one polyhedron to the other. One key aspect of our system is to allow the user to identify the important features of each polyhedron and specify a correspondence between them. The rest of the algorithm consists of a combination of techniques that can produce the desired result from the given user input.

The main components of our approach are:

- User Input: The user selects vertex pairs on each polyhedron and defines a correspondence.
- Feature Net Computation: Given the vertex pairs, the algorithm constructs a feature net for each polyhedron. The feature net is a sub-graph of the vertex/edge connectivity graph of each polyhedron.
- Decomposition into Morphing Patches: Based on the feature nets, the algorithm decomposes the surface of each polyhedron into the same number of morphing patches, each being homeomorphic to a closed disk.
- Mapping: A pair of corresponding morphing patches are mapped to a 2 D polygon.
- Merging: The algorithm merges the topological connectivity of morphing patches in the 2D polygon.
- Reconstruction: Using the results from merging, the algorithm reconstructs the facets for the new morphing patch and generates a merged polyhedron with the combined topologies of the original two.
- Local Refinement: The user can make local changes to the feature net in some part of the model and then re-compute the merged polyhedron.


## 5 Correspondence between Polyhedra

In this section we present the algorithm to compute the correspondence between two homeomorphic polyhedra, $\mathbf{A}$ and $\mathbf{B}$. Their vertices are denoted as $\mathbf{V}^{\mathrm{A}}=$ $\mathrm{V}^{\mathrm{A}}, \mathrm{V}^{\mathrm{A}}, \ldots, \mathrm{V}^{\mathrm{A}}{ }_{\mathrm{m}}$ and $\mathbf{V}^{\mathrm{B}}=\mathrm{V}^{\mathrm{B}}, \mathrm{V}^{\mathrm{B}}{ }_{2}, \ldots, \mathrm{~V}^{\mathrm{B}}$. Superscripts represent the corresponding polyhedron. The edges and faces of the polyhedra are represented as $\mathbf{E}^{\mathrm{A}}, \mathbf{E}^{\mathrm{B}}, \mathbf{F}^{\mathrm{A}}$ and $\mathbf{F}^{\mathrm{B}}$. The output is a merged polyhedron with the topology of both input models, for which each vertex has a location on the two input models. The algorithm ensures
that each face of the input and output polyhedra is a triangle. Given the polyhedra, the algorithm computes a circularly ordered set of edges for each vertex. For each edge, the algorithm stores the following information: incident vertices, left and right adjacent faces, and preceding and succeeding edge in counter-clockwise order. We will use the symbols $\mathbf{G}^{\mathrm{A}}$ and $\mathbf{G}^{\mathrm{B}}$ to represent the adjacency graph of two polyhedra. Furthermore, we assign a weight to each edge of this graph. The weight corresponds to the Euclidean distance between the two vertices defining the edge. Based on the user's specification, the correspondence algorithm marks some of the vertices and edges in these graphs. To start with, each edge and vertex in these graphs is unmarked.

We illustrate the correspondence algorithm described in this section with Figs. 1 through 6. In our illustration, polyhedron A corresponds to a model of an igloo and polyhedron $\mathbf{B}$ corresponds to a model of a house (shown in Fig. 1). Upper case letters denote objects in 3D and lower case letters represent objects in 2D.

### 5.1 Specifying Corresponding Features

The user selects a pair of unmarked vertices on each of the input polyhedra, denoted by $\left\{\mathrm{V}^{\mathrm{A}}{ }_{\mathrm{i}}, \mathrm{V}^{\mathrm{A}}{ }_{\mathrm{i}}\right\}$ and $\left\{\mathrm{V}_{\mathrm{i}}^{\mathrm{B}}\right.$, $\left.\mathrm{V}^{\mathrm{B}}{ }_{\mathrm{i} 2}\right\}$ respectively (Fig. 1). The algorithm computes a shortest path between the vertices on each polyhedron using only unmarked vertices and edges in the adjacency graph. Let the shortest paths correspond to: $\left\{\mathrm{V}_{\mathrm{i}}^{\mathrm{A}}, \mathrm{V}_{\mathrm{j}, \ldots}^{\mathrm{A}}, \ldots\right.$
 All the intermediate vertices and edges on the shortest paths in each graph are marked. We shall call such a shortest path a chain, and the user-selected vertices are referred to as the extremal vertices of a chain. If no chain can be computed using unmarked edges and vertices in the adjacency graph, the feature pairs are not allowed.


Figure 1. Input polyhedra with user-specified correspondences.
The selected vertices and chains are used to formulate a feature net for each polyhedron. We will represent the feature nets as $\mathbf{N}^{A}$ and $\mathbf{N}^{B}$. They are sub-graphs of $\mathbf{G}^{A}$ and $\mathbf{G}^{\mathrm{B}}$, respectively. The user needs to specify a sufficient
number and arrangement of chains in order to partition the boundaries of the polyhedra. Each extremal vertex must be adjacent to at least two chains, and each chain must have a connected patch on each side. As a result, $\mathbf{N}^{\mathrm{A}}$ and $\mathbf{N}^{\mathrm{B}}$ have the same number of chains and extremal vertices and the user has specified a mapping between each extremal vertex and chain. In this way, the two feature nets define a bijection.

### 5.2 Decomposition into Morphing Patches

The feature nets are used to decompose the boundary of each polyhedron into the same number of morphing patches. A morphing patch (shown in Fig. 2) is simply a connected subset of a polyhedron which can be mapped to a 2D polygon. The vertices and edges are partitioned into interior and exterior vertices and edges. The exterior vertices of the morphing patches are those on the feature net (Fig. 3).


Figure 2. Two selected morphing patches.
The decomposition algorithm has two steps. First, the perimeter of a morphing patch is computed by traversing the feature nets. Second, the remaining edges and vertices and the faces, all of which are interior to a morphing patch, are partitioned.

The computation of the perimeter uses the circular ordering of edges at each vertex. Beginning from an arbitrary vertex and chain of the feature net, the algorithm walks through the tightest clockwise loop of feature net vertices it can find. From the first vertex and chain, it moves to the vertex at the other end of the chain. Then it uses the circular ordering of edges at that vertex to proceed to the closest clockwise chain. Next it follows that chain to the vertex at the other end. When it comes back to the original vertex, it has traversed the perimeter of a morphing patch. Note that this is possible because the chains may not cross. The bijective mapping between the feature nets implies that this process when performed on polyhedron A automatically selects the corresponding chains on polyhedron B.

The interior of a morphing patch is computed with a modified depth-first search algorithm. For a given perimeter, it starts with an arbitrary edge on the chain. It determines which adjacent face is interior to the patch
using the ordering of the exterior vertices. (If the face is on the interior, then the order must be consistently clockwise or counter-clockwise.) We cross the face that is on the interior of the morphing patch and recursively search from its other two edges. The recursion stops when the edge has already been traversed or the edge is part of the chain.

Once this patch has been identified, the algorithm selects another exterior vertex-chain pair and find another perimeter. This process is repeated until all vertices, edges, and faces of the original polyhedra have been traversed. These morphing patches are represented as $\mathbf{P}_{1}^{\mathrm{A}}, \ldots, \mathbf{P}_{\mathrm{K}}^{\mathrm{A}}$ and $\mathbf{P}_{1}^{\mathrm{B}}, \ldots, \mathbf{P}_{\mathrm{K}}^{\mathrm{B}}$. The algorithm ensures that each morphing patch has no holes in the interior and is homeomorphic to a closed disk. In other words, it is bounded by a single loop of adjacent vertices.

Non-simple Polyhedron: For a genus zero polyhedron, no morphing patch can contain a hole. If the input polyhedron has a hole, the user must specify the features in such a manner that each morphing patch is homeomorphic to a disk. Such a decomposition is always possible. An example is shown in Fig. 8 for a cup and a torus.

### 5.3 Mapping

Given a morphing patch, our goal is to compute a parameterization over a convex polygonal region in 2D. Construction of a parameterization for complex shapes over a simple domain is an important problem that occurs in different applications. A number of algorithms have been proposed by Kent et al [20], Malliot et al [25], and Eck et al [11]. The most obvious way to map to a 2D polygon is to use a harmonic map. They minimize the metric distortion and preserve the aspect ratios of the triangle, but can introduce area compression [11]. Given two triangular faces $F_{i}^{A}$ and $F_{j}^{A}$ of a morphing patch, let them map to the triangles $f_{i}^{A}$ and $f_{j}^{A}$, respectively, in the 2D polygon. Our goal, as detailed in [13] is to produce a mapping such that the ratio of the areas between $\mathrm{F}_{\mathrm{i}}^{\mathrm{A}}$ and $F_{j}^{A}$ is close to that of $f_{i}^{A}$ and $f_{j}^{A}$.


Figure 3. The morphing patches are mapped to regular 2D polygons.

Given a pair of morphing patches $\mathrm{P}_{\mathrm{i}}^{\mathrm{A}}$ and $\mathrm{P}_{\mathrm{i}}^{\mathrm{B}}$, we compute a mapping from the surface of the patch to a regular 2D polygon. Let these morphing patches consist of $m_{i}$ extremal vertices. We map the morphing patch onto a regular 2 D polygon, inscribed in the unit circle, with $m_{\mathrm{i}}$ edges. We represent the 2D polygon as $p_{\mathrm{i}}$.

To compute a mapping, our algorithm first establishes a bijection between the chains of the two feature nets by splitting edges on the respective chains on the two models. The splitting criterion is based on edge lengths. After splitting, the algorithm has ensured that the corresponding patches have the same number of exterior edges and vertices.

The extremal vertices of the morphing patches are mapped to the vertices of $\boldsymbol{p}_{\mathrm{i}}$. Each chain of the morphing patch is mapped to an edge of $\boldsymbol{p}_{\mathrm{i}}$. All the external vertices lying in the interior of a chain are mapped onto the edges of $\boldsymbol{p}_{\mathrm{i}}$. The 2D coordinates of the vertices along the chains are interpolated based on the arc length of the chain.

The next step is to compute a 2 D mapping for the interior vertices of $\mathbf{P}_{\mathrm{i}}^{\mathrm{A}}$ and $\mathbf{P}^{\mathrm{B}}$. We use a simple recursive technique that tries to preserve the ratio of areas of the triangles based on a greedy heuristic. The algorithm divides a morphing patch by selecting two exterior vertices $\mathrm{V}_{\mathrm{i}}^{\mathrm{A}}$ and $\mathrm{V}_{\mathrm{j}}^{\mathrm{A}}$ that do not lie on the same chain and the corresponding exterior vertices, $\mathrm{V}^{\mathrm{B}}$ ind $\mathrm{V}_{\mathrm{j}}^{\mathrm{B}}$, on the other morphing patch. Next it computes a shortest path across the interior of each morphing patch, and then optimizes it to preserve the area ratio on each side from the 3 D model to the 2 D polygon. The vertices and edges lying on these paths are mapped to the interior of $\boldsymbol{p}_{\mathrm{i}}$. Using this path, the algorithm recursively divides the morphing patch and maps the sub-patches to $\boldsymbol{p}_{\mathrm{i}}$.

At this point the algorithm has computed a parameterization for each morphing patch. Each triangular face $\mathrm{F}_{\mathrm{j}}^{\mathrm{A}}$ and $\mathrm{F}_{\mathrm{k}}^{\mathrm{B}}$ has been mapped to a corresponding triangle, $\mathrm{f}_{\mathrm{j}}^{\mathrm{A}}$ and $\mathrm{f}^{\mathrm{B}}$, respectively, in $\boldsymbol{p}_{\mathrm{i}}$.

Given an interior vertex of $\mathbf{P}_{i}^{A}$, the algorithm locates the triangle, say $f_{k}^{B}$ that contains the image of that interior vertex. Furthermore, the algorithm computes the corresponding point in $f_{k}^{B}$ and represents it using barycentric coordinates in terms of vertices of $f^{B}{ }_{k}$. Finally the barycentric coordinates are used to compute the corresponding point on $\mathrm{F}_{\mathrm{k}}^{\mathrm{B}}$.

### 5.4 Merging

The algorithm has so far produced mappings into $\boldsymbol{p}_{\text {i }}$ such that the vertices $V^{A}{ }_{j}=>v^{A}, V^{B}{ }_{j}=>v^{B}{ }_{j}$, and edges $E^{A}{ }_{j}$ $\Rightarrow e^{A}, E_{j}^{B} \Rightarrow e^{B}$. The edges $e^{A}$ and $e^{B}$ will in general intersect. We compute the intersections, split the intersecting edges, and create new vertices (Fig. 4).


Figure 4. The 2D polygons are combined into one polygon with merged topology.
Motivated by simplicity and robustness, we used a algorithm of complexity $\mathrm{O}\left(n_{e}{ }^{2}\right)$ which checks all edge pairs for overlap. Since the intersection computations can fail on edges that are co-incident, we handle the case where edges lie on the same mapped path separately. Furthermore, to avoid creating an invalid topology, we calculate all the intersections, sort them and then create the edges into which they were split. After intersection computation and splitting, we denote the set of all vertices and edges in $\boldsymbol{p}_{\mathrm{i}}$ by $\mathbf{x}^{\mathrm{AB}}$ and $\mathbf{g}^{\mathrm{AB}}$, respectively.


Figure 5. The output after the merging and reconstruction steps.


Figure 6. The reconstructed merged polyhedron has the combined topology of the two input polyhedra.

### 5.5 Reconstruction

After computing the intersection of all the edges, the algorithm produces a planar straight-line graph (PSLG) [3] from those intersections. The PSLG is constructed
from the $\mathbf{x}^{\mathrm{AB}}$ and $\mathbf{g}^{\mathrm{AB}}$. We recursively subdivide the connected edges and vertices from the merging step into the smallest counter-clockwise cycles possible. Since the nature of the intersections in the merging step guarantees that these cycles are convex, from this point we can triangulate the regions (Fig. 5). After this step has been performed on all the morphing patches we get a merged polyhedron containing each of the input geometries as shown in Fig. 6.

## 6 Morphing

At the end of the merging and reconstruction process, the merged polyhedron has the combined topological connectivity of polyhedra $\mathbf{A}$ and $\mathbf{B}$. Each vertex on polyhedron $\mathbf{A}$ has a corresponding vertex on polyhedron B. In this section, we briefly discuss issues for interpolating between the two polyhedra in order to generate a morph.

During morphing, the vertices travel from their positions on $\mathbf{A}$ to their respective positions on $\mathbf{B}$ along morphing trajectories. Kent et al. [20] represented the paths using splines. In particular, they specified the tangent directions at each extremal vertex and used Hermite interpolation to compute the control points. Similarly we allow the user to represent the trajectory as a Bézier curve for each pair of extremal vertices. The system then interpolates the trajectories for all other vertices of the merged polyhedron using weighted contributions of the morphing trajectories at the extremal vertices. More details are given in [13].

In addition to the morphing trajectories required by the algorithm, other attributes of the input polyhedra need to be interpolated to generate a good morph. These include vertex colors, lighting coefficients, normal vectors, etc. Interpolation of these surface attributes occurs during the mapping and merging steps [13].

## 7 User Interface

The user interface is one of the most important aspects of a morphing system. Although it is easy for the user to conceptualize a morph between two objects, it can be rather difficult to design a system that allows the user to express this easily. Our system achieves this goal by allowing the user to draw the key correspondences on the surfaces of the input models, and to specify the paths that the corresponding features will follow during the morph.

The user specifies corresponding chains of the feature nets for input models $\mathbf{A}$ and $\mathbf{B}$ by selecting the chains' endpoints. In order to enforce a bijection between the two feature nets, the system requires the user to specify
the feature net vertices in corresponding pairs. The interior vertices of a chain are automatically computed as shortest Euclidean paths along the edges of the polyhedra. In some cases a path based on unmarked vertices and edges, may not be available. In that case, our system creates new vertices and edges by splitting a face.

As a simple extension to creating a single pair of corresponding chains, we also allow the user to create "multi-chains" and "loops". When creating a multichain, after specifying the first corresponding feature vertex pair, each additional vertex pair the user specifies creates a pair of chains connecting to the previous vertex pair. A loop is simply a multi-chain with the property that the last pair of corresponding vertices is connected to the first pair by an additional pair of chains.

Once the corresponding pairs of chains have been specified, we allow several techniques for local refinement. These include splitting a chain into two chains at a selected vertex on one of the input models, removing a chain, and moving extremal vertices of the feature net.

After a corresponding pair of extremal vertices has been specified on polyhedron $\mathbf{A}$ and polyhedron $\mathbf{B}$, the user can control the morphing trajectory. The user does this by positioning the tangents of a Bézier curve, whose endpoints are the locations of the vertices on the two input models.

## 8 Implementation and Performance

We have implemented the algorithm in C++ using the OpenGL and Tcl/Tk libraries. In our polyhedral data structures, each vertex has a list of edges stored in counter-clockwise order, each edge contains the incident vertices and two facets, and each facet contains three vertices and three edges also stored in counter-clockwise order. The system ensures that each polyhedron has valid topology and that it satisfies the Euler-Poincaré formula.

The merging algorithm described in Section 5.4 computes all intersections between the mapped edges of each morphing patch. Based on the decomposition algorithm described in Section 5.2, a morphing patch of a large polyhedron may consist of many thousands of edges. The number of intersections (and thereby the combinatorial complexity of the merged polyhedra) grows with the number of edges and in the worst case is quadratic. As a result, the merging and reconstruction steps can become a bottleneck in the overall computation. To overcome this problem, we recursively subdivide each morphing patch into smaller sub-patches,
using the same area preservation heuristic as in the mapping algorithm described in section 5.3.

## 9 Analysis and Results

The running time is dependent on the size of the morphing patches. For two polyhedra with $m$ and $n$ vertices, let the number of extremal vertex pairs specified by the user be $k$. The number of morphing patches can be at most $O(k)$. The computation of feature nets and morphing patches is bounded by $O(k(m+n))$. The user can set the subdivision algorithm to divide until each patch has $Q$ edges and the number of morphing subpatches can be $O((m+n) / Q)$. The worst case complexity of merging and reconstructing each pair of morphing sub-patches is $O\left(Q^{2}\right)$. $Q$ is typically small enough to be regarded as a constant when compared with $m$ and $n$. Worst case interpolation takes $O((m+n) k \log Q)$ time. As a result, the overall complexity of the algorithm is $O(K(m+n))$, where $K=\max \{k \log Q, Q\}$.

The visual "quality" of a morph created by our system is quite subjective. It is mainly determined by the number of extremal vertices. Beyond that, controlling the morphing trajectories strongly contributes to the appearance of the morph. Finally, morphs between objects that are "similar" in appearance result in smoother transitions than morphs between vastly different objects (cf. human heads morph in Fig. 7 vs. cup-donut morph in Fig. 8).

This system has been successfully created morphs between a number of complex polyhedral models. These include simple polyhedra (shown in Fig. 7 and 9) as well as non-simple polyhedra corresponding to a torus and a cup (shown in Fig. 8). We have presented the results in Table 1. The animated morphs generated by this algorithm are available at our WWW site.

## 10 Conclusion

We have presented a new approach for establishing a correspondence between two homeomorphic polyhedra, which includes a simple, intuitive user interface. It has been successfully applied to a number of simple and non-simple polyhedral models. We believe it is versatile enough to produce visually pleasing 3D morphs when coupled with an effective method for specifying the interpolation between the two models. More details about our system are available in [13].

The specification of the feature net suffers from two limitations: the feature net must be connected and the chains connecting the feature net vertices are currently restricted to lie on the edges of the source models.

It can be difficult to control the morphing trajectories for complicated models. In order to have a usable morphing system, a method which will allow the user to sculpt the intermediate models is need. We are currently working on removing these limitations.

## 11 Acknowledgements

This research has been supported in part by an Alfred P. Sloan Foundation Fellowship, ARO Contract CAAH04-96-1-0257, NSF Career Award CCR-9625217, ONR young Investigator Award (N0014-97-1-0631), Honda, Intel, and NSF/ARPA Center for Computer Graphics and Scientific Visualization.

Thanks to Robert and Shirley Gregory for additional funding, Rhinoceros for making the beta version of their modeling program freely available, Scott Nelson for the triceratops model we found on the web, Todd Gaul for video editing, and Henry Fuchs.

## 12 References

[1] A. Barr. Global and local deformations of solid primitives. ACM Computer Graphics, vol. 18, pp. 21--30, 1984.
[2]T. Beier and S. Neely. Feature-based image metamorphosis. In Computer Graphics (SIGGRAPH '92 Proceedings), vol. 26, 1992, pp. 35--42.
[3]M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. Computational Geometry: Algorithms and Applications: Springer-Verlag, 1997.
[4]E. Bethel and S. Uselton. Shape distortion in computerassisted keyframe animation. In State of the Art in Computer Animation, N. M.-T. a. D. Thalmann, Ed.: Springer-Verlag, 1989, pp. 215-224.
[5]D. Chen, A. State, and D. Banks. Interactive Shape Metamorphosis. Proc. of 1995 Symposium on Interactive 3D Graphics, pp. 43--44, 1995.
[6]E. Chen and R. Parent. Shape Averaging and its Applications to Industrial Design. IEEE Computer Graphics \& Applications, vol. 9, pp. 47-54, 1989.
[7] M. Chen, M. W. Jones, and P. Townsend. Methods for Volume Metamorphosis. In Image Processing for Broadcast and Video Production, Y. P. a. S. Wilbur, Ed., 1995.
[8]D. Cohen-Or, D. Levin, and A. Solomovici. Contour Blending Using Warp-Guided Distance Field Interpolation. In IEEE Visualization '96: IEEE, 1996.
[9]M. Covell and M. Withgott. Spanning the gap between motion estimation and morphing. Proc. of IEEE International Conference on Acoustics, Speech and Signat Processing, vol. 5, pp. 213-216, 1994.
[10] D. DeCarlo and J. Gallier. Topological Evolution of Surfaces. Prof. of Graphics Interface'96, pp. 229-235, 1996.
[11] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle. Multiresolution Analysis of Arbitrary Meshes. In SIGGRAPH'95, 1995, pp. 173--182.
[12] E. Galin and S. Akkouche. Blob Metamorphosis based on Minkowski Sums. Computer Graphics Forum (Eurographics'96), vol. 15, pp. 143-153, 1996.
[13] A. Gregory, A. State, M. Lin, D. Manocha, and M. Livingston. Feature-based Surface Decomposition for Correspondence and Morphing between Polyhedra. Department of Computer Science, University of North Carolina Technical Report TR98-14, 1998.
[14] L. Guibas and J. Hershberger. Morphing Simple Polygons. In Proc. 10th Annu. ACM Sympos. Comput. Geom., 1994, pp. 267--276.
[15] T. He, S. Wang, and A. Kaufmann. Wavelet-Based Volume Morphing. Proceedings of IEEE Visualization, pp. 85-91, 1994.
[16] J. Hodgins and N. Pollard. Adapting Simulated Behaviors for New Characters. SIGGRAPH'97 Conference Proceedings, pp. 153-162, 1997.
[17] T. Hong, N. Magnenat-Thalmann, and D. Thalmann. A general algorithm for 3D shape interpolation in a facetbased representation. In Proceedings of Graphics Interface '88, 1988, pp. 229--235.
[18] T. Kanai, H. Suzuki, and F. Kimura. 3D Geometric Metamorphosis based on Harmonic Maps. Pacific Graphics, pp. 97-104, 1997.
[19] A. Kaul and J. Rossignac. Solid-interpolating deformations: construction and animation of PIPs. In Proc. Eurographics, 1991, pp. 493--505.
[20] J. Kent, W. Carlson, and R. Parent. Shape transformation for polyhedral objects. SIGGRAPH '92 Conference Proceedings, pp. 47--54, 1992.
[21] J. Kent, R. Parent, and W. Carlson. Establishing Correspondences by Topological Merging: A New Approach to 3-D Shape Transformation. Proceedings of Graphics Interface '91, pp. 271--278, 1991.
[22] F. Lazarus and A. Verroust. Feature-based Shape Transformation for Polyhedral Objects. Fifth Eurographics Workshop on Animation and Simulation, 1994.
[23] S. Lee, K. Chwa, S. Shin, and G. Wolberg. Image Metamorphosis Using Snakes and Free-Form Deformations. SIGGRAPH 95 Conference Proceedings, pp. 439--448, 1995.
[24] A. Lerios, C. Garfinkle, and M. Levoy. Feature-Based Volume Metamorphosis. SIGGRAPH 95 Conference Proceedings, pp. 449--456, 1995.
[25] J. Maillot, H. Yahia, and A. Veroust. Interactive Texture Mapping. SIGGRAPH'93 Conference Proceedings, pp. 27-34, 1993.
[26] M. Mortensen. Geometric Modeling: John Wiley and Sons, 1997.
[27] R. Parent. Shape Transformation by Boundary Representation Interpolation: A Recursive Approach to Establishing Face Correspondences. The Journal of Visualization and Computer Animation, vol. 3, pp. 219-239, 1992.
[28] R. Parent. Implicit Function Based Deformations of Polyhedral Objects. In Implicit Surfaces '95, 1995.
[29] B. Payne and Ar. Toga. Distance field manipulation of surface models. IEEE Computer Graphics and Applications, vol. 12, pp. 65--71, 1992.
[30] W. T. Reeves. Particle Systems -- a Technique for Modeling a Class of Fuzzy Objects. ACM Trans. Graphics, vol. 2, pp. 91--108, April 1983.
[31] T. Sederberg, P. Gao, G. Wang, and H. Mu. 2D Shape Blending: An Intrinsic Solution to the Vertex Path Problem. SIGGRAPH '93 Conference Proceedings, pp. 15--18, 1993.
[32] T. Sederberg and E. Greenwood. A physically based approach to 2D shape blending. SIGGRAPH '92 Conference Proceedings, pp. 25--34, 1992.
[33] T. Sederberg and S.. Parry. Free-Form Deformation of Solid Geometric Models. Computer Graphics (SIGGRAPH '86 Proceedings), vol. 20, pp. 151--160, 1986.
[34] M. Shapira and A. Rappoport. Shape Blending Using The Star-Skeleton Representation. IEEE CG\&A, vol. 15, pp. 44--50, March 1995.
[35] G. Wolberg. Skeleton-Based Image Warping. The Visual Computer, vol. 5, pp. 95--108, 1989.
[36] G. Wolberg. Digital Image Warping: IEEE Computer Society Press, 1990.
[37] W. Wyvill. Metamorphosis of Implicit Surfaces. SIGGRAPH'90 Course 23- Modeling and Animation with Implicit Surfaces, 1990.

| Models | Triangles |  | Feature Net <br> Vertices | Morphing <br> Patches | User Time | Time to Compute Merged <br> Polyhedron on R10000 CPU |
| :---: | ---: | ---: | :---: | :---: | :---: | :---: |
| House-Igloo (Fig. 3) | $\mathbf{8 2}$ | 40 | 16 | 10 | $\sim 5 \mathrm{~min}$ | $<1 \mathrm{~s}$ |
| Human-Triceratops (Fig. 12) | $\mathbf{5 , 6 6 0}$ | $\mathbf{1 7 , 5 2 8}$ | $\mathbf{1 7 6}$ | $\mathbf{8 6}$ | $\sim 6$ hours | 2.5 min |
| Human Heads (Fig. 10) | 3,426 | 4,020 | 134 | 67 | $\sim 3$ hours | $\mathbf{3 0 ~ s e c o n d s}$ |
| Donut-Cup (Fig. 11) | 4,096 | 8,452 | 63 | 50 | $\sim 4$ hours | 1 min |

Table 1: Performance of our algorithm on four different pairs of input polyhedra. The table shows model complexity, the number of feature net vertices and morphing patches, the times to specify the features and trajectories by the user and times to compute the merged polyhedron on a SGI Onyx 2


Figure 7. Human heads feature net (left) and morph (right).


Figure 8. Doughnut-cup feature net and morph.


Figure 9. Human-triceratops feature net and morph.

Figure 12. (a) Human heads morph. The male head consists of 3,426 triangles. The female head consists of 4,020 triangles. The feature nets (red) consist of 134 extremal vertices on each of the two polyhedra. All morphing trajectories are straight lines. (b) The merged polyhedron (computed in approx. 30 sec on an R10000 CPU) as it morphs between the heads.

Figure 13. (a) Doughnut-cup morph. The doughnut consists of 4,096 triangles. The cup consists of 8,452 triangles. The feature nets (red) consist of 63 extremal vertices on each of the two polyhedra. Most morphing trajectories are straight lines, except for a few around the rim of the cup, where material was "routed" along curved paths to avoid self-intersections. (b) The merged polyhedron (computed in approx. 1 min on an R10000 CPU) as it morphs between cup and doughnut.

Figure 14. (a) Human-triceratops morph. The human consists of 17,528 triangles. The triceratops consists of 5,660 triangles. The feature nets (red) consist of 185 extremal vertices on each of he two polyhedra. Most morphing trajectories are straight lines, except for a few around the rather flat tail, polyhedron (computed in approx 2.5 min on an R 10000 CPU ) as it morphs between human and triceratops.

