

Shape Transformation for Polyhedral Objects

James R. Kent Wayne E. Carlson Richard E. Parent Department of Computer and Information Science Advanced Computing Center for the Arts and Design The Ohio State University Columbus, Ohio 43210

Abstract

Techniques that transform one two-dimensional image into another have gained widespread use in recent years. Extending these techniques to transform pairs of 3D objects, as opposed to 2D images of the objects, provides several advantages, including the ability to animate the objects independently of the transformation. This paper presents an algorithm for computing such transformations. The algorithm merges the topological structures of a pair of 3D polyhedral models into a common vertex/edge/face network. This allows transformations from one object to the other to be easily computed by interpolating between corresponding vertex positions.

Keywords: Computer Animation, Computer-Aided Geometric Design, Interpolation, Shape Transformation.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling; I.3.7 -[Computer Graphics]: Three-Dimensional Graphics and Realism

1.0 Introduction

In recent years, image processing techniques, popularly known as "morphing", have achieved widespread use in the entertainment industry. Morphing involves the transformation of one 2D image into another 2D image. These techniques involve first specifying some function that maps points from one image onto points of the other image, then simultaneously interpolating the color and the position of corresponding points to generate intermediate images. When viewed in sequence, these intermediate images produce an animation of the first image changing into the second. Variations of these techniques have been used to create astonishing special effects for commercials, music videos, and movies.

While morphing is useful for many applications, the fact that the intermediate stages of the transformation are images with no 3D geometry limits its use. In order to fully realize the benefits of transformations in animation and design, 3D models of the objects must be transformed, instead of just 2D images of these objects. Transforming 3D models as opposed to images allows for the objects to be animated independently of the transformation, using computer animation techniques such as keyframing. In addition, 3D transformations can be used in design to create objects that combine features of the original objects ([4], [8], [14]).

This paper presents an algorithm that, given two 3-D polyhedral models, generates two new models that have the same shape as the original ones, but that allow transformations from one to another to be easily computed. A previous paper [9] described an early version of the algorithm that was limited to star-shaped¹ polyhedral solids. Since then, the algorithm has been extended to allow for transformations between more complex polyhedral models. In addition, the computational complexity and robustness of the algorithm have been improved.

After some fundamental concepts are defined in Section 2, a description of the shape transformation problem for 3D objects is given in Section 3. This is followed by a brief review of previously published research in Section 4. Section 5 provides a detailed description of the algorithm. Section 6 addresses interpolation issues, including transforming non-geometric attributes, such as surface color. Sample transformations are presented in Section 7. The paper concludes with a discussion of open issues and future research in Section 8.

2.0 Fundamental Concepts

In order to discuss the shape transformation problem, it is useful to carefully define a few key terms. Throughout this discussion, the term *object* will be used to refer to an entity that has a 3D surface geometry. The *shape* of an object refers to the set of points in object space that comprise the object's surface. The term *model* will be used to refer to any complete description of the shape of an object. Thus, a single object may have many different models that describe its shape.

Following the terminology used by Weiler in [16], topology refers to the vertex/edge/face network of a model. Geometry refers to an instance of a topology for which the vertex coordinates have been specified. Vertices, edges, and faces are collectively referred to as topological elements.

Some concepts from mathematical topology also need to be defined. Two objects are said to be *homeomorphic*, or *topologically equivalent*, if a continuous, invertible, one-to-one mapping between

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

^{1.} Star-shaped refers to models for which at least one interior point, p, exists such that any semi-infinite ray originating at p intersects the surface of the object at exactly one point.

points on the surface of the two objects exists. Such a mapping is referred to as a *homeomorphism*. Finally, an object is said to be *Euler-valid* if its topology obeys the generalized Euler formula:

V - E + F = 2 - 2G

where V, E, and F are, respectively, the number of vertices, edges, and faces of the topological network, and G is the number of passages through the object (i.e. its *genus*).

3.0 The Shape Transformation Problem

A common approach to transforming one shape into another is to divide the problem into two steps. The first step is to establish a mapping from each point on one surface to some point on the second surface. Once these correspondences have been established, the second step is to create a sequence of intermediate models by interpolating corresponding points from their position on the surface of one object to their position on the surface of the other. The first step will be referred to as the *correspondence problem*, and the second step will be referred to as the *interpolation problem*. The two problems are interrelated since the method used to solve the interpolation problem is dependent upon the manner in which the correspondences are established.

This paper presents a solution to the correspondence problem for Euler-valid, genus 0, polyhedral objects. By restricting ourselves to polyhedral objects, the correspondence step does not need to explicitly specify the mapping for every point on the surface. A sufficient solution is to specify correspondences for each vertex of the models. The interpolation problem is then solved by interpolating the positions of corresponding vertices. Since the main contribution of this paper is an algorithm for establishing correspondences, the majority of the paper is concerned with the solution to the correspondence problem. Issues that arise during the interpolation are briefly discussed in Section 6.0.

4.0 Previous Work

As mentioned in Section 1.0, "morphing" techniques for transforming images have demonstrated remarkable results and have achieved widespread use. Wolberg provides an excellent introduction to image morphing in [17]. These techniques rely on the user to specify pairs of points in the two images that correspond.

Several approaches to three-dimensional shape transformation have been published. Wyvill [18] describes a transformation algorithm for implicit surfaces (i.e. blobby objects). Brute force approaches for polyhedral models, such as that described by Terzides' [14], essentially require the user to specify, for every vertex, a corresponding vertex from the other model. Hong et al. [7] propose a solution for polyhedra based on matching the faces of the objects whose centroids are closest. Bethel & Uselton [1] describe an algorithm that adds degenerate vertices and faces to two polyhedra until a common topology is achieved. Chen & Parent [4] present a transformation algorithm for piecewise linear 2D contours, then briefly address an extension for 3D lofted objects. Parent [10] describes a solution for polyhedra that establishes correspondences by splitting the surface of the models into pairs of sheets of faces, then recursively subdividing them until the topology of each pair is identical. Kaul & Rossignac [8] transform pairs of polyhedra by computing the Minkowski sum of scaled versions of the models. By gradually scaling one model from 100% to 0% while simultaneously scaling the other from 0% to 100%, a transformation is obtained. Payne & Toga [11] first convert each polyhedra into a distance-field volumetric representation, interpolate the values at each point of the 3D volume, then find a new isosurface that represents some combination of the original objects.

Techniques that make use of the topology and geometry of the models tend to yield better results. For example, since Hong et al. and Payne & Toga do not make full use of the topological information, the surfaces of the models generated at intermediate steps are not guaranteed to remain connected. Similarly, Bethel & Uselton and Parent rely primarily on the topology to establish correspondences, ignoring most of the geometric information. This often results in severely distorted intermediate models.

Kaul & Rossignac's technique, as well as the one described in this paper, make full use of both the topology and the geometry of the models, resulting in intermediate models that have connected surfaces and that exhibit small amounts of distortion. One principal advantage of the method described herein is that our correspondence algorithm describes a homeomorphism. This provides a straightforward method for interpolating the surface attributes of the objects along with the geometry. In addition, it seems likely that our approach can be more easily extended to allow for greater user control over the transformation.

5.0 An Algorithm for Establishing Correspondences

Suppose that two genus 0 solid objects are specified. Now, imagine that it were possible to inflate these objects with air like balloons until they became spherical. Each point on the surface of each object maps onto a unique point on the surface of the sphere. Associating each point from one object with the point from the other object that maps to the same point on the sphere establishes a one-to-one correspondence between points on the surface of the two objects.

The above observations form the basis for the correspondence algorithm. First, the surface of each object is projected onto a unit sphere. This mapping is used to identify correspondences between points on the two original objects by associating pairs of points that map to the same location on the sphere. This approach can potentially be applied for non-polyhedral genus 0 objects (e.g. spline surface models) as long as a mapping from the surface of the object to the surface of the unit sphere can be found.

Bier and Sloan [2] describe a similar approach for solving the problem of wrapping a 2D texture onto a 3D object. The first step maps the texture onto an intermediate surface, such as a sphere or a cylinder. The second step maps the intermediate surface to the surface of the 3D object. Unfortunately, the techniques used to map the intermediate surface to the 3D object are not always one-to-one, and thus are not appropriate for our application.

This section describes an implementation of the correspondence algorithm for genus 0 polyhedral solids. The first step is to project the topology of both models onto the unit sphere. Next, the two topologies are merged by clipping the projected faces of one model to the projected faces of the other. The merged topology is then mapped onto the surface of both original models. This generates two new models that have the same shape as the original two models, but that share a common topology. This allows a transformation between the two shapes to be easily computed by interpolating the coordinates of each pair of corresponding vertices. Figure 1 shows a pair of models and the same pair with the merged topologies mapped onto their surfaces

Throughout the discussion, the original objects are referred to as A and B. The original polyhedral models of these objects are referred to as M_a and M_b . M_a has vertices, V_a , edges, E_a , and faces, F_a . Similarly, M_b has vertices, V_b , edges, E_b , and faces, F_b . The projection of M_a and M_b onto the unit sphere are referred to as $(M_a)_p$ and

 $(M_b)_p$, respectively, with vertices $(V_a)_p$ and $(V_b)_p$. When referring to a specific topological element of one of the models, lower case letters will be used. For example, e_a refers to a specific edge of object M_a , and $(v_b)_p$ refers to a specific projected vertex of M_b . The results of the correspondence algorithm (i.e. the two new models of A and B that share a common topology) are referred to as M_a^* and M_b^* .



Figure 1 - An Example of The Correspondence Algorithm

5.1 Projection Methods

The first step of the correspondence algorithm is to project the surface of the two polyhedra onto the surface of the unit sphere. The projection must satisfy two conditions. First, it must be one-to-one, so that each point on the surface of the object projects to a unique point on the surface of the sphere. Second, the projection must be continuous in the sense that points within a small radius of a given point project to within a small radius of the projection of that point. Any method for projecting an object that satisfies these two conditions is acceptable.

The projected polyhedral models are completely specified by the topology of the original model together with the coordinates of the projected vertices. This enables the projected models to be saved, eliminating the need to recompute them each time the model is used.

Since the correspondences between the models are established by their mappings onto the sphere, different mappings result in different transformations. Thus, providing different projection methods gives the user some degree of control over the transformation. Sections 5.1.1 through 5.1.4 describe a collection of projection methods that allow a wide variety of polyhedral models to be transformed. While no completely general method has been found, the techniques presented work for a large number of commonly encountered types of models.

5.1.1 Convex and Star-Shaped Objects

The definition of a star-shaped polyhedral object is that at least one interior point of the polyhedron exists from which all the vertices of the object are visible. This definition suggests a method for projecting such an object. First, specify such an interior point, O, to be the center of the object, and translate the object so that O coincides with the origin. Then, move each vertex in or out along the ray from O through the vertex until it lies at unit distance from O.

A suitable center point can be algorithmically selected by first computing the intersection of the interior half spaces of all the planes of the faces of the model. The resulting volume is called the *kernel* of the polyhedral model. If the original polyhedron is star-shaped, its kernel is a non-empty convex polyhedron. Averaging the vertices of the kernel yields a suitable center point for the projection. The complexity of computing the kernel of a three-dimensional polyhedron is O(NlogN) [12].

Since the choice of center point affects the location of the projected vertices, providing the ability to select a center point gives the user some control over the transformation. Verifying that the selected point satisfies the vertex visibility condition can be performed in O(N) time by testing that the outward normal of each face is directed away from the point.

Note that convex polyhedra are a special case of star-shaped polyhedra for which all interior points satisfy the visibility condition. Thus, for convex polyhedra, any interior point may be specified as the center.

Some of the projection methods described in the following sections project the model onto a convex polyhedron (usually the convex hull of the object). To complete the projection to the sphere, the starshaped projection is applied to this convex polyhedron.

5.1.2 Methods Using Model Knowledge

Polyhedral models are often constructed using techniques such as revolving a contour about an axis, or extruding a planar polygon along a line [3]. By using information about how the model is constructed, efficient methods for projecting the object to the unit sphere can be found. This approach naturally lends itself to an object-oriented methodology, where "ProjectToSphere" could be one of the methods attached to an object.

The class of polyhedral models known as objects of revolution consist of a set of planar contours (ribs) arranged at angular increments around an axis. Such a model can be projected to a sphere in O(N)time by positioning the points of each rib along a longitudinal arc of the sphere whose "north/south" axis matches the axis of the model. Each arc should lie in the plane of the rib, have its endpoints on the axis, and be on the same side of the axis as the rib.

Two methods for spacing the rib points along a semicircle have been developed. The first positions them so that the arc lengths between points on the semicircle are proportional to the distance between the corresponding points on the rib. The second method first projects the rib onto its convex hull using a recursive method similar to that described by Ekoule et al. in [5]. Once the rib is mapped onto its convex hull, each point is moved in or out to the unit sphere along a ray from the midpoint of the axis through the point.

Any technique that maps a rib to a semicircle can be used for spacing the points with different transformations resulting. However, methods that preserve geometric information from the original model, such as the two described above, generally lead to more aesthetic transformations.

Another common class of polyhedral models, known as extruded objects, are generated by moving a planar polygon along a straight line, sweeping out a solid volume. Two copies of the polygon are used to cap the ends of the object. This class of models can be projected by mapping each of the two caps to its convex hull, using Ekoule's method as above. The resulting convex model can be projected to the unit sphere using the star-shaped projection from Section 5.1.1.

It is important to note that the above techniques work for any model that can be described in an appropriate format, whether or not the object was originally modeled using the described techniques. For example, data from 3D digitizers can often be easily converted into the object of revolution format.

5.1.3 Physically-Based Methods

As mentioned in Section 5.0, the inspiration for the shape transformation algorithm involves an analogy with inflating the objects like a balloon. This idea led to experimentation with projection methods based upon physically-based simulation. The goal is to have the simulation convert the object into a convex object with the same topology while preserving as much of the geometric information contained in the model as possible. The simulations, based on the work of Haumann [6], treat the surface of the model as a flexible object. Each vertex of the topology is modeled as a mass and each edge of the topology as a spring.

Several types of simulations were tried. One approach was to model the forces involved in inflating a balloon. Weak spring forces were applied along the edges together with internal air pressure forces. The air pressure forces had a magnitude that was proportional to the area of each face and were applied to the centroid of each face in the direction of its outward normal. For some models, this approach worked well, but in general, the simulation did not always produce a convex model. This was usually due to the presence of cycles of short edges in the models. When stretched, these edges generated large forces that resisted further stretching. In addition, vertices would tend to drift around, which diminished the relationship between the geometry of the original objects and their projections.

Another approach that has been more successful is to first determine which vertices of the model lie on its convex hull. Fixing these points, and treating the non-hull vertices as free masses connected by springs along the edges to each other and to the hull vertices, a simulation is run to "snap" the model outward to its convex hull. Setting the strengths of the springs to be inversely proportional to their original lengths preserves the ratios of edge lengths as much as possible. In addition, fixing the hull vertices minimizes the drifting problem. Although this approach generally works better than the first, it does not work for arbitrary models.

In performing the above experiments, one scenario that consistently yields the desired results was discovered. This situation occurs whenever a concave region of the model is completely surrounded by a planar convex ring of edges that lie on the convex hull. Running a simulation by fixing the vertices that lie on this ring, and treating the network of edges and vertices that lie inside as a mass/ spring system quickly "snaps" out the interior into the plane of the surrounding ring. In the following, this scenario will be referred to as a "surrounded region".

To better understand this situation, consider the following analogy. Suppose you were to build a planar, convex wooden frame with a nail pounded into each corner. Next, attach a mesh of rubber bands to the nails. No matter how one pulls upon the rubber band mesh, as soon as it is let go, it snaps back into the plane of the wooden frame. In this analogy, the wooden frame corresponds to the surrounding ring of edges. The nails correspond to the vertices of the surrounding ring. The edges of the interior network are the rubber bands, and the vertices correspond to places where rubber bands are joined.

An approach that shows promise for solving the projection problem for arbitrary genus 0 polyhedra is to attempt to divide the projection into a set of subproblems, each of which involves a surrounded region. A heuristic approach to subdividing the problem in this manner is to use the faces of the convex hull of the model to define the set of surrounded regions as follows. Start by computing the convex hull of a model. Next, find a set of non-intersecting paths of edges that connect each pair of vertices connected by an edge of the hull. Finally, space the points of each path along the corresponding hull edge. If a set of non-intersecting paths has been found, each face of the convex hull will now define a surrounded region. Although this algorithm works for many models, it is not too difficult to create models for which no appropriate set of paths can be found. Further research is being conducted into algorithmically finding a suitable subdivision for any genus 0 polyhedra.

Figure 2 shows a polygonal model of a goblet in the upper left. The set of paths of edges found by the algorithm are shown in the upper right. Spacing the vertices of each path along the corresponding edge of the convex hull yields the object in the lower left. The results of the simulation are shown in the lower right. In this case, the simulation causes the network of vertices and edges that form the inside of the goblet's bowl to snap out onto the plane defined by the rim of the goblet.



Figure 2 - "Snapping" an Object to its Convex Hull

5.1.4 Hybrid Methods

In addition to the projection methods described in the proceeding three sections, two other techniques have been developed that combine model knowledge with physical simulation.

Lofted and tubular objects consist of a series of planar contours that are joined along a (possibly curved) path. Combining the methods of Section 5.1.2 and 5.1.3 generates an algorithm for projecting this class of models. Select two adjacent contours. If the contours are not convex, project them to their convex hull using Ekoule's method. The two contours define a pair of surrounded regions as described in Section 5.1.3. Running a "rubber-band mesh" simulation with the contour points fixed causes the interior of each region to snap onto the plane of the contour. Figure 3 shows a tubular object with the selected contours highlighted and the same object after the simulation is completed.

Two features of this technique may not be evident. First, the two contours do not have to contain an equal number of points. The only requirement is both contours are a simple planar polygon. Second, the entire procedure can be performed with no user interaction, provided that knowledge of the manner in which the models are stored is available. However, it is desirable that the user be allowed to specify the pair of contours which are to remain fixed.



Figure 3 - "Snapping" a Tubular Model

The second hybrid method involves the user directly specifying the surrounded regions of the model. For example, to project a model of a man, the user might specify rings of edges around each arm at the shoulder, around each leg at the hip, and around the head at the base of the neck. A "rubber-band mesh" simulation is then run to snap in the extremities. It is up to the user to select surrounded regions that result in a convex model after the simulation is performed. Techniques for assisting the user in specifying the regions are currently being investigated.

Using this technique, the surrounded regions appear to "grow" out of the other model during the transformation. This interesting effect is due to the fact that points on the ends of the extremities have much larger distances to cover than do those at the base, and hence move at a greater velocity.

5.2 The Merging Algorithm

Once both models have been projected, the second step of the correspondence algorithm is to merge the topologies of the two models by clipping the projected faces of one object to the projected faces of the other. In an earlier paper [9], an $O(N^2)$ algorithm based on Weiler's polygon clipping algorithm ([15]) is described. The algorithm requires each projected edge of one model to be intersected with each projected edge of the other. Since the edges of the projected models map onto great circles of the unit sphere, these computations involve finding the intersection of pairs of circular arcs.

This algorithm works well for merging the projected topologies of objects that are not overly complex. However, for large models (> 1000 vertices) small numerical inaccuracies in the arc intersection calculations often result in an improper ordering of the intersection points along an edge. Since the algorithm is dependent upon maintaining a valid topological structure, improper ordering can cause the merging process to fail.

The original merging algorithm was also quite slow. If the number of edges of the models are N_a and N_b , respectively, in the worst case, there are $O(N_a N_b)$ intersections. However, for most models, since the faces are spread out across the entire surface of the sphere, an edge from one model only intersects a small number of edges of the other model. Thus, in the vast majority of cases, the number of intersections is much less than $N_a N_b$. This suggests that an algorithm whose execution time is dependent upon the number of intersections could significantly reduce the overall execution time.

This observation led to the development of a new merging algorithm that is faster and more robust than the original one described in [9]. The improvements are the result of exploiting the topological information contained in the models. The algorithm is similar in nature to the planar overlay algorithm described by Seidel in [13].

The following paragraphs describe the steps of this new algorithm and analyze its complexity. The description assumes that the faces of the model have been triangulated prior to execution. It also assumes that no projected vertices of the two models are coincident, and that no projected vertex of one model lies on a projected edge of the other. These degenerate cases can be handled by simple extensions of the basic algorithm.

Figure 4 contains a pseudocode description of the algorithm. The pseudocode assumes that arrays are used to store structures for each vertex, edge, and face of the models. For each vertex, this structure contains the original and projected locations of the vertex, as well

```
(Step 1)
Read in the Topology and Geometry of M<sub>a</sub> and M<sub>b</sub>, as
well as the Coordinates of the Projected Vertices,
\{V_{a}\}_{p} and \{V_{b}\}_{p}. Translate the models so their centers are at the origin.
(Step 2)

vl_a <-- first vertex of M_a

MapToB[vl_a] <-- face of (M_b)_p that contains (vl_a)_p

Add the edges originating at vl_ to Work List (WL)
Mark those edges Used
While (WL) is Not Empty
    e_a < -- next edge of WL
v1<sub>a</sub>, v2<sub>a</sub> <-- endpoints of el<sub>a</sub>
f<sub>b</sub> <-- MapToB[v1<sub>a</sub>]
    Add edges of f_b to Candidate List (CL) While CL is Not Empty
          eb <-- next edge of CL
         Intersect e_a and e_b
         If Successful
              Add Intersection Point, i, to Model
              Create links from \mathbf{e}_{a} and \mathbf{e}_{b} to i
              f_b <-- Face of M_b on other side of e_h
              Add two other edges of eb to CL
         End If
    End While
    MapToB[v2a] <-- fb
    Add the unused edges originating at v_2 to WL
    Mark those edges Used
End While
(Step 3)
For each edge, eb, of Mb
     v1b, v2b <-- endpoints of eb
     Sort the intersections of eb using topological
         Information from M.
     Set MapToA(vlb) and MapToA(v2b) to faces con-
         taining vl, and v2b, respectively
End For
(Step 4)
For each vertex, v<sub>g</sub>, of M<sub>g</sub>
Calculate the barycentric coordinates of (v<sub>g</sub>)<sub>p</sub>
with respect to the projected vertex coordi-
         nates of the face, MapToB[v_a], of M_b
    Use these barycentric coordinates and original
         vertex coordinates of the face, MapToB[v_]
         to determine where v_a maps to on the surface
         of Mb
End For
(Step 5)
Repeat Step 4 for each vertex of M<sub>h</sub>, using the faces
stored in array MapToA to identify the face of Ma
that contains each vertex of Ma
(Step 6)
Output the combined geometry and topology of both
models, Ma* and Mb*
```

Figure 4 - Pseudocode for the Merging Algorithm

as the edges beginning at that vertex, stored in clockwise order. Each edge structure includes the indices of the two endpoints and the indices of the two faces it separates. The edge structure also contains a pointer to the list of intersections of that edge. The face structure includes the indices of the three vertices and the three edges that comprise the face. In addition to these structures, as each intersection point if found, it is stored in an array of structures that contain the indices of the two edges that intersect, the parametric values of the intersection point relative to those edges, and pointers used to order the intersections along the edge.



Figure 5 - Calculating the Intersections of an Edge

The first step is straightforward and can be performed in O(N) time. Step 2 involves intersecting each edge of $(M_a)_p$ with a subset of the edges of $(M_b)_p$, as illustrated in Figure 5. First, vertex v_A of $(M_a)_p$ is determined to lie inside face f_{abc} of $(M_b)_p$. This can be done in O(N) time by casting a ray from the origin through v_A and finding the face of (M_b)_p it intersects. Once this is done, the edges originating at v_A are added to a list of edges to be processed, the work list. Assume e_{AB} is the first edge on this list. Since it is known that v_A lies on face f_{abc} of $(M_b)_p$, the first intersection of that edge must be with one of the edges of that face. Thus, e_{ab} , e_{ac} , and e_{bc} from $(M_b)_p$ are added to a list of candidate edges that eAB might intersect. In this case, eAB intersects ebc. The topology of Mb can be used to determine that eAB crosses over to face fbcd at the intersection point. Thus, edges e_{bd} and e_{cd} are added to the candidate list. Similarly, at the intersection of e_{cd} and e_{AB} , edge e_{AB} crosses onto face f_{cde} , and edges ece and ede are added to the candidate list. At the intersection of e_{de} and e_{AB} , edge e_{AB} crosses onto face f_{def} , and edges e_{df} and



Figure 6 - Sorting the Intersections

 e_{fb} are added to the candidate list. Since e_{AB} does not intersect either of these edges, vertex v_B must lie on face f_{def} . This fact is recorded and the edges originating at v_B are added to the work list. This continues until the work list is empty.

Step 3 of the algorithm sorts the intersections of each edge of $(M_b)_p$ using topological information from $(M_a)_p$ to ensure that the ordering is valid. As shown in Figure 6, basing the sort on this information avoids inconsistencies in the topology due to small numerical errors in the intersection calculations. This step is also used to determine which face of $(M_a)_p$ contains each vertex of $(M_b)_p$.

Steps 4 and 5 use the information that indicates which face of $(M_b)_p$ contains each vertex of $(M_a)_p$, and vice versa, to determine where the vertices of one model map onto the surface of the other. This is done using barycentric coordinates as shown in Figure 7. Step 6 involves tracing out of the faces of the combined models using the original topologies and the sorted intersections of each edge and can be performed in O(N) time.



Figure 7 - Determining the Vertex Locations

5.2.1 Analysis of the Merging Algorithm

Steps 1, 4, 5, and 6 can all be performed in O(N) time. The time required to complete Steps 2 and 3 is dependent upon the number of edges that intersect and is analyzed below. As in previous sections, N_a and N_b represent the number of edges of M_a and M_b .

In step 2, each edge of M_a is intersected with exactly $3 + 2 * I_e$ edges, where I_e is the number of intersections of the edge. Since this must be done for each edge, the total number of intersections is $3 * N_a + 2 * I_{tot}$, where I_{tot} is the total number of edge-edge intersections. Thus the running time of step 2 is $O(N_a + I_{tot})$. For complex models, the distribution of the faces on the sphere ensures that $I_{tot} << N_a N_b$.

In step 3, the intersections of each edge of M_b must be sorted. If I_e is the number of intersections of an edge, the sorting of that edge requires time $O(I_e log I_e)$. Since in the worst case, each edge can be intersected $O(N_a)$ times, the worst case complexity is $O(N_bN_a log N_a)$. However, in terms of the total number of intersections, since the sum of $(I_e log I_e)$ for each edge is less than or equal to $(I_{tot} log I_{tot})$, the complexity is $O(I_{tot} log I_{tot})$.

Thus, the overall complexity of the algorithm in terms of output size equals that of step 3, $O(I_{tot} \log I_{tot})$. Although in the worst case, I_{tot} is $O(N^2)$, the distribution of the edges on the sphere causes I_{tot} to be much smaller than this in most cases.

6.0 Interpolation Issues

Up to now, this paper has concentrated on the correspondence step of the shape transformation problem. Once the combined models, M_a^* and M_b^* have been created, the transformation is computed by interpolating between each pair of corresponding vertex locations. In addition to linear interpolation, the use of a Hermite spline for the path of each vertex, with the tangent vectors of the spline set equal to the vertex normals, has proven effective.

Two potential problems may arise during the interpolation. First, for faces with more than three edges², interpolating vertices from one position to another will not guarantee that all faces remain planar. This problem can be solved by triangulating the faces of M_a^* and M_b^* , prior to the interpolation. Second, an object may penetrate itself during the interpolation. This may or may not be a problem, depending on the application. Possible solutions of this problem are being investigated.

Interpolating non-geometric surface attributes, such as color, texture, or transparency, along with the geometry of the models produces interesting effects. This can be easily done since the correspondence algorithm specifies a homeomorphic mapping between the two objects. Given a point on the surface of some intermediate model, barycentric coordinates can be used to locate that point relative to the vertices of the face that contain it. From these coordinates, the corresponding points on the original objects can be found. The value of the attribute for the point on the intermediate model is found by interpolating the values of the attribute for these two points.

7.0 Results

Figures 8 to 11 present some examples of the transformation algorithm. The examples were rendered using faceted shading and neutral colors to better illustrate the topological structure of the intermediate models.

Figure 8 shows a glass transforming into a spiral tube. The projections used for the two objects are those illustrated in Figure 2 and Figure 3, respectively. The spiral is used again in Figure 9, this time transforming into a 3D digitized sculpture. The sculpture data was obtained from a 3D digitizing device and is organized as a set of



Figure 8 - Transforming an Object Using the Convex-Hull Snapping Technique into a Tubular Object



Figure 9 - Transforming a Tubular Object into an Object of Revolution



Figure 10 - A Pair of Transformations Using Different Projection Methods for the "S"-shaped Object



Figure 11 - Each Column Illustrates the 0%, 25%, 50%, 75%, and 100% Points of a Transformation

planar ribs revolved around an axis. Thus, the object of revolution technique from Section 5.1.2 was used for the projection.

Figure 10 illustrates the results of using different projection methods upon the transformation. In the upper sequence, the extruded letter 'S' was projected using the convex hull snapping technique

^{2.} Although M_a and M_b must be triangulated, the faces of M_a* and M_b* will, in general, have up to six sides.

described in Section 5.1.3. In the lower sequence, the hybrid method for tubular objects described in Section 5.1.4 was used to project the 'S'. The object of revolution method was used to project the digitized head in both sequences. The two sequences illustrate that radically different results are possible by altering the projection method used.

Figure 11 shows three columns, each of which represents a transformation between a a pair of objects of revolution. The objects in the middle of each column are the models obtained at the 25%, 50%, and 75% points of the transformation. The base objects of the left and middle columns are objects of revolution. The base object of the rightmost column is an extruded 6-pointed star.

As a final note, the following statistics for the transformations in Figures 8 to 11 are provided to support the claim made in Section 5.2.1 that the total number of intersections, I_{tot} , is much less than N_aN_b for complex models.

	Na	N _b	NaNb	Itot
Figure 8	1.8K	2.7K	4.9M	6.5K
Figure 9	2.7K	18.7K	50.5M	19.9K
Figure 10				
top	66	18.4K	1.2M	4.OK
bottom	66	18.4K	1.2M	1.9K
Figure 11				
left	864	18.7K	16.2M	14.7K
middle	102	18.7K	1.9M	5.0K
right	72	18.7K	1.3M	3.9K

8.0 Future Research

Future research will focus on three areas. First, extensions of the algorithm to handle wider classes of polyhedra will be investigated. For genus 0 objects, this involves developing new ways to project the surface of a model onto a sphere. For non-genus 0 objects, cutting the objects to eliminate the passages through them, or replacing the sphere with a representative manifold (e.g. a torus for objects with one hole) are possibilities.

The second area of interest is to examine the problem of self-intersections during the interpolation. A good solution to this problem has applicability for many other problems that involve interpolation, not just shape transformation.

The third area of investigation involves providing user control of the transformation. The remarkable results obtained by morphing are possible because the user maintains complete control over the transformation. Unlike the other published techniques for 3D shape transformation, the algorithms presented in this paper allow some control over the transformation through mechanisms such as selecting the center of the object and choosing the projection technique. However, to achieve results equivalent to those obtained by morphing images, techniques that provide a finer level of control over the transformation are needed. One possibility is to add a warping step after the models are mapped to the sphere, but before the topologies are merged.

Acknowledgments

We wish to thank the Department of Computer and Information Science, the Advanced Computing Center for the Arts and Design and the Ohio Supercomputer Center for the use of their facilities, and Hewlett-Packard and AT & T for equipment grants that make this research possible. We also wish to thank Dr. Rephael Wenger, Kevin Rodgers, Steve May, and Stephen Spencer for useful ideas and criticism, and Professor Charles Csuri for the use of his head.

Bibliography

- Bethel, E. and Uselton, S. Shape Distortion in Computer-Assisted Keyframe Animation. In State of the Art in Computer Animation. Magnenat-Thalmann, N. and Thalmann, D., eds., Springer-Verlag, New York, 1989, 215-224.
- Bier, E. and Sloan, K. Two-Part Texture Mappings. IEEE Computer Graphics and Applications 6, 9 (Sept. 1986), 40-53.
- Carlson, W. An Advanced Data Generation System for Use in Complex Object Synthesis For Computer Display. Proceedings of Graphics Interface '82 (1982) 197-204.
- Chen, E., and Parent, R. Shape Averaging and Its Applications to Industrial Design. *IEEE Computer Graphics and Applications* 9, 1 (Jan. 1989) 47-54.
- Ekoule, A., Peyrin, F. and Odet, C. A Triangulation Algorithm from Arbitrary Shaped Multiple Planar Contours. ACM Transactions on Graphics 10, 2 (April, 1991) 182-199.
- Haumann, D. and Parent, R. The Behavioral Test-Bed: Obtaining Complex Behavior from Simple Rules. Visual Computer 4, 6 (Dec. 1988) 332-347.
- Hong, T., Magnenat-Thalmann, N. and Thalmann, D. A General Algorithm for 3-D Shape Interpolation in a Facet-Based Representation. *Proceedings of Graphics Interface '88* (June 1988) 229-235.
- Kaul, A. and Rossignac, J. Solid-Interpolating Deformations: Construction and Animation of PIPs. Proceedings of Eurographics '91. In *Computers and Graphics* (1991).
- Kent, J, Parent, R. and Carlson, W. Establishing Correspondences by Topological Merging: A New Approach to 3-D Shape Transformation. *Proceedings of Graphics Interface '91* (Calgary, Alberta, June, 1991) 271-278.
- Parent, R. Shape Transformation by Boundary Representation Interpolation: A Recursive Approach to Establishing Face Correspondences. Technical Report OSU-CISRC-2/91-TR7. Computer and Information Science Research Center. The Ohio State University (1991).
- 11 Payne, B. and Toga, A. Distance Field Manipulation of Surface Models. *IEEE Computer Graphics and Applications 12*, 1 (Jan. 1992) 65-71.
- 12. Preparata, F. and Shamos, M. Computational Geometry An Introduction. Springer-Verlag, New York, 1985.
- Seidel, R. Output-Size Sensitive Algorithms for Constructive Problems in Computational Geometry. Ph.D. Thesis, Cornell University, 1986.
- Terzides, C. Transformational Design. Knowledge Aided Architectural Problem Solving and Design, NSF Project #DMC-8609893, Final Report, (June 1989).
- Weiler, K. Polygon Comparison Using a Graph Representation. Proceedings of SIGGRAPH '80 (Seattle, Washington, July 1980). In Computer Graphics 14, 3, (Aug. 1980), 10-18.
- 16. Weiler, K. Topology as a Framework for Solid Modeling. Proceedings of Graphics Interface '84, (May, 1984).
- Wolberg, G. Digital Image Warping. IEEE Computer Society Press, Los Alamitos, CA, 1990.
- Wyvill, B. Metamorphosis of Implicit Surfaces. Notes from SIGGRAPH '90 Course 23 - Modeling and Animating with Implicit Surfaces, (Dallas, Texas, Aug. 1990).