

Morphing Three Dimensional Polyhedral Objects

Carl Erikson
Honors Paper
5/16/94

Morphing Three Dimensional Polyhedral Objects

1.0 Definition of the Morphing Problem

Morphing is the act of smoothly transforming one shape into another. For example, Figure 1 shows a triangle morphing into a square.

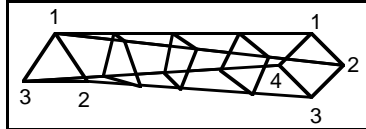


Figure 1

The morphing is smooth in the sense that any intermediate step of the transformation can be produced.

The morphing problem centers on determining how to produce such a smooth transformation. This problem can be split into two parts, the correspondence problem and the interpolation problem. The correspondence problem is to establish a mapping from each point of one surface to a point on the other surface. Once this mapping has been created, a series of intermediate shapes can be made by interpolating corresponding points from one object to the other. The way in which the points are interpolated is the interpolation problem. For example, in Figure 1, the correspondence problem has been solved by mapping vertices 1 to 1, 1 to 2, 2 to 3, and 3 to 4. The interpolation problem has been solved through linear interpolation, shown by the dotted lines in the figure above. This paper is primarily concerned with the correspondence problem and assumes that linear interpolation is used to solve the interpolation problem.

2.0 Background of Morphing

Morphing has been used extensively in the field of computer graphics for visual effects. It can be seen in television commercials such as an advertisement for Bic razors, music videos such as Michael Jackson's *Black or White*, and movies such as *Terminator 2*. Morphing is also being considered in the fields of pattern and character recognition, although not much has been accomplished in this area.

Most of the work done in the area of morphing has been in two dimensions. The success of these two dimensional morphing algorithms can be seen in the previously mentioned commercials, music videos, and movies. All of these algorithms work with digital images. In other words, the algorithms morph a flat picture to another flat picture.

The most successful of these algorithms rely on a user to pick out certain features of both pictures that he or she wishes to correspond during the morph. For example, the user might match an eyebrow in one picture to another eyebrow in the second to guide the algorithm's computation. *Feature-Based Image Metamorphosis* is a 1992 SIGGRAPH paper by Thaddeus Beier and Shawn Neely that describes this algorithm which relies on the association of features between pictures. Morphing between two dimensional polygonal (or wire frame) shapes that lack an internal image has also been worked on. These algorithms are generally less impressive than their digital image counterparts.

Three dimensional morphing has also been attempted. Some of these algorithms morph between three dimensional polyhedral objects and some morph between volumetric models.

Morphing three dimensional polyhedra is difficult because of the varying topologies that these figures may have. Every three dimensional polyhedra algorithm has solved the correspondence problem in some way, but not one has done so in a way that is generally pleasing for all objects. Volume morphing has been worked on somewhat, the most successful algorithm is described in another 1992 SIGGRAPH paper, *Scheduled Fourier Volume Morphing* by John Hughes. This algorithm interpolates smoothly between the Fourier transform of the objects, not the original figures themselves. So far, the results of these algorithms have not been as successful as their two dimensional counterparts because of the added complexity involved in dealing with three dimensional figures.

The advantages of two dimensional morphing over three dimensional morphing are that it is simpler to do, and the results are very visually pleasing. Unfortunately, since morphing is basically a visual effect, extra realism is always being sought. Three dimensional morphing gives the transformation a feeling of depth and perspective since it is in fact working with three dimensional data. Also, transforming objects in three dimensions allows for the use of animation techniques such as keyframing and panning since the morphing can be viewed from any angle. Again, the main disadvantage of three dimensional morphing is the added complexity of working with three dimensional data.

3.0 Basis of Honors Work

The basis of my honors work comes from two 1992 SIGGRAPH papers. The first is *A Physically Based Approach to 2-D Shape Blending* by Thomas Sederberg and Eugene Greenwood from Brigham Young University. The other is *Shape Transformation for Polyhedral Objects* by James Kent, Wayne Carlson, and Richard Parent from Ohio State University.

3.1 A Physically Based Approach to 2-D Shape Blending

This algorithm attempts to smoothly blend between two dimensional polygonal shapes. The algorithm treats the shapes as pieces of wire that can be stretched and bent. By minimizing the work done in bending the wire during the transformation, a visually pleasing shape blend is produced.

The interpolation problem is solved by using linear interpolation. Let A and B be two polygons with the same number of vertices where

$$A = [A_0, A_1, \dots, A_n] \text{ and } B = [B_0, B_1, \dots, B_n]$$

and similarly labeled vertices correspond. A linear interpolation between the two can be described by

$$\begin{aligned} P(t) &= [(1-t)A_0 + tB_0, (1-t)A_1 + tB_1, \dots, (1-t)A_n + tB_n] \\ &= [P_0(t), P_1(t), \dots, P_n(t)] \end{aligned}$$

where t ranges from 0 to 1. An example of linear interpolation can be seen in Figure 1.

The algorithm solves the correspondence problem by trying to find the correspondence that will produce the minimal work transformation. In order to accomplish this, the algorithm checks the angles between adjacent edges of the two dimensional polygons and also simulates the bending and stretching of the polygons as if they were wire.

By imposing a pair of constraints on the angles between adjacent edges of a polygonal figure during the transformation, the algorithm eliminates unwanted visual effects. The first situation that the algorithm tries to avoid is when an angle goes to zero; in other words, $\theta_i(t) = 0$ for $0 \leq t \leq 1$ for some angle i between two edges. If an angle goes to zero, then the

two edges that share that angle will pass over each other causing an ugly visual effect as in Figure 2.

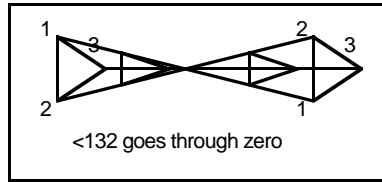


Figure 2

Secondly, the algorithm favors angles that change monotonically from $\angle_i(0)$ to $\angle_i(1)$ for every angle i . In other words, if $\angle_i(t_1) \leq \angle_i(t_2)$ where $t_1 \leq t_2$ and $0 \leq t_1, t_2 \leq 1$, or $\angle_i(t_1) \geq \angle_i(t_2)$ where $t_1 \leq t_2$ and $0 \leq t_1, t_2 \leq 1$, then the angle changes monotonically. Thus, the algorithm penalizes angles that first decrease, then increase and vice versa. An angle that does not change monotonically is shown in Figure 3.

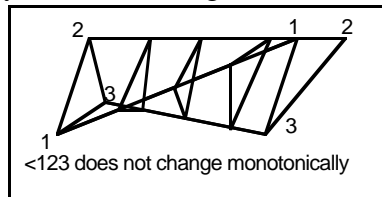


Figure 3

By interpreting the changing of the angle as a degree two Bézier curve, the algorithm can detect these two angle aberrations relatively easily.

Each polygon is treated as a pliable piece of wire. A wire has two possible movements. One is stretching and the other is bending. Both of these types of movements are simulated in this algorithm to determine how much work it takes for a particular transformation. The amount of work done in stretching or shrinking a wire is modeled by the equation

$$W = \frac{k |L_1 - L_0|^e}{(1 - c) \min(L_0 - L_1) + c \max(L_0 - L_1)}$$

where L_0 is the start length of the wire, L_1 is the end length, and c , e , and k are user defined constants which modify the wire's inherent physical properties. The amount of work done in bending a wire (i.e., the bending of an angle shared by two edges) is modeled by the equation

$$W = \begin{cases} \hat{E}k(+ m *)^e & \text{if } (t) \text{ never goes through zero} \\ \check{E}k(+ m *)^e + p & \text{if } (t) \text{ does go through zero} \end{cases}$$

where \hat{E} is the change of the angle, $*$ is how much the angle breaks monotonicity, and e , k , m , and p are user defined constants. The constant p penalizes angles which go through zero and the constant m penalizes angles that change non-monotonically.

Once a correspondence has been set up between vertices, the work done in the transformation can be calculated using these physically based equations. Thus, to find the minimal work transformation, the algorithm tries every possible correspondence between the vertices of the two polygonal objects. After every correspondence has been produced, the work calculation is done on the particular correspondence and compared with the minimal work transformation so far. If the new correspondence requires less work, then it becomes the minimal work transformation.

3.2 Shape Transformation for Polyhedral Objects

This algorithm by Kent, Carlson, and Parent also uses a physically based idea to solve the correspondence problem between two three dimensional polyhedra. The idea is to treat both objects as inflatable balloons and blow them up until they are both perfectly round. Once they share the common topology of a sphere, their topologies can be merged on the sphere. This merged topology can then be mapped back onto both of the original objects in an intuitive way. Thus, both objects will share a common topology making the correspondence problem trivial.

The first step of the algorithm is to inflate the objects onto a unit sphere. The vertices of the original object are projected from an interior central point onto the unit sphere. In order for this projection to be successful, each projected vertex must be visible from the central point, or projection point. In other words, the polyhedra being projected must be a star-shaped polyhedra. The definition of a star-shaped polyhedra is that there must exist at least one interior point such that all vertices of the polyhedra are visible from that point. The projection point can be any of the points that make the polyhedra a star-shaped polyhedra. Thus, the algorithm is limited in that only star-shaped polyhedra can be morphed. Note that all convex polyhedra meet the requirements of being a star-shaped polyhedra and, in fact, any interior point of a convex polyhedra could be a projection point. For example, the projection of a pyramid is shown in Figure 4.

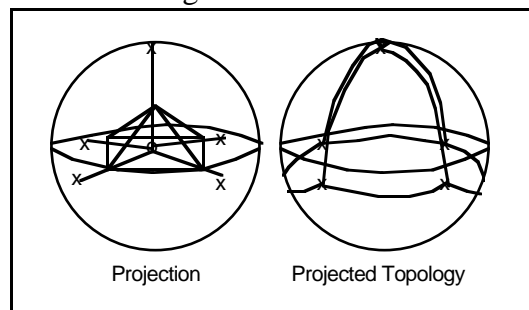


Figure 4

As the vertices are projected, so are the edges. Note that edges of the original object, when projected, become great arcs on the sphere.

Once both objects have been projected onto unit spheres, then the topologies of these two spheres can be merged. Calculating this new topology involves finding all the intersections of the arcs from the two spheres and from this information, building a new topology with the correct number of faces, edges, and vertices. Once it has been produced, it is then possible to map this combined topology back onto the surfaces of both of the original objects. The idea is that the barycentric coordinates of a merged vertex within a projected triangular face can be calculated on the sphere, and these coordinates determine the placement of the vertex on the original face. As soon as both objects have the same topology mapped onto themselves, then there exists a trivial correspondence between the vertices. Every vertex on the merged sphere gets projected back onto two places, one on the surface of the first object and the other on the surface of the second. If a correspondence between these two locations for each vertex is set up, then a correct correspondence is produced between these two three dimensional polyhedra. An example of this merging process is shown in Figure 5.

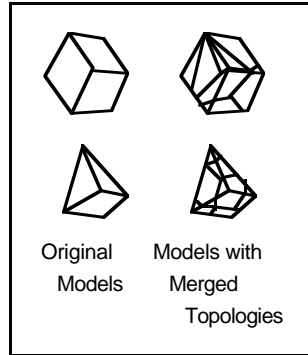


Figure 5

Once a correspondence has been established between the three dimensional polyhedra, the interpolation problem is solved by using linear interpolation.

4.0 Honors Work

The goal of my honors work was to devise a method to find the best correspondence solution between two three dimensional polyhedra resulting in the most visually pleasing transformation. Having studied the two papers described above, I decided to combine the merits of both algorithms into one. The result is an algorithm which produces a near minimal work morphing between two polyhedra.

4.1 Description of Algorithm

Some assumptions are made about the two polyhedra being transformed. First, they must both be star-shaped polyhedra in order for their topologies to be projected onto a unit sphere. Second, both polyhedra must be triangulated. In other words, every face of the polyhedra must be made up of only three edges. Triangulated polyhedra make the topology merging process much simpler. This triangulation is not a major restriction since a triangulated polyhedra can be constructed from any polyhedra.

The first step of the algorithm is to project each three dimensional polyhedra onto a unit sphere. An interior projection point must be found for each object. Since the polyhedra are star-shaped, at least one projection point is guaranteed to exist. An algorithm for finding an interior projection point involves intersecting the interior half spaces of all the planes of the faces of the model to create an interior volume. Any point within this interior volume may be chosen as an interior projection point. To find the projected topology, a ray is shot from the projection point through each vertex of the polyhedra. Wherever this ray hits the unit sphere is where the vertex lies on the sphere. Note that the edges of the polyhedra, once straight lines, become great arcs of a sphere when projected. Figure 6 shows a triangulated pyramid being projected onto a unit sphere.

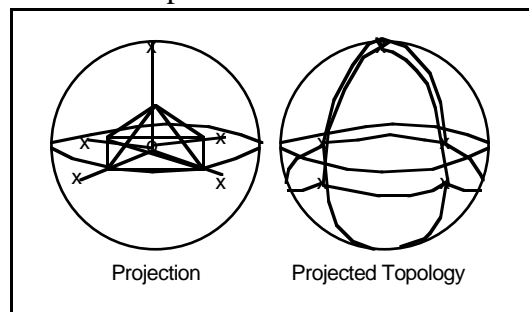


Figure 6

At this point, there are two unit spheres, both having a projected topology on its surface. Before merging the topologies of the two spheres, rotating the spheres will affect the resulting merged topology. In other words, the correspondence created by merging the spheres' topologies can be greatly affected by rotating one of the spheres. Figure 7 shows the effects of rotating one of the spheres before the merging process.

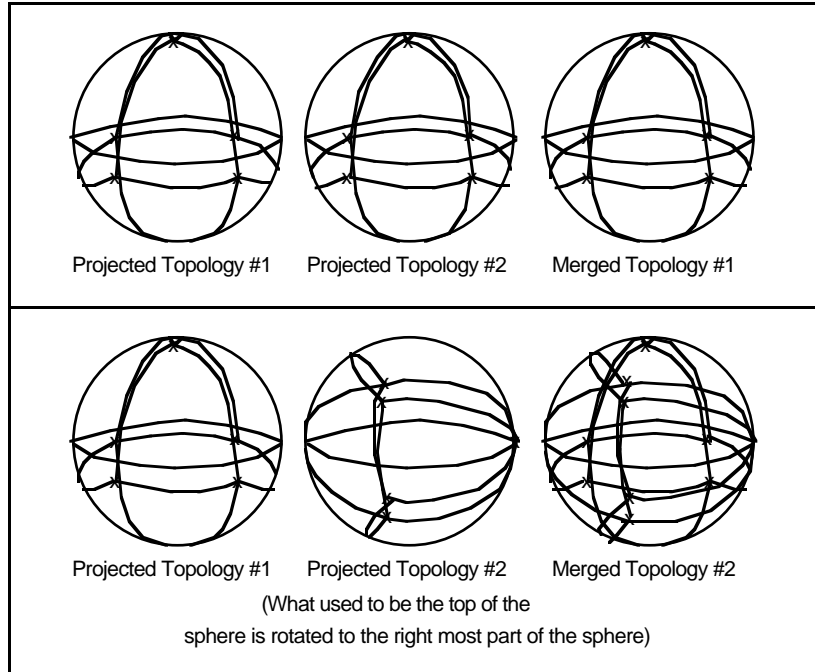


Figure 7

Thus, to determine the best correspondence between a pair of polyhedra, all possible rotations of the sphere must be tried. Unfortunately, there are an infinite number of possible rotations of a sphere so an approximate solution will have to suffice. This issue will be addressed later.

Now that the spheres' topologies have been lined up by rotation, the merging process can proceed. As in the previous algorithm, the arcs of combined topologies must be checked for intersection in order to figure out the vertices, edges, and faces of the merged topology. Once the list of arc intersections has been calculated, the arcs, noting the new intersections, can be traversed in a clockwise manner to carve out the new faces. This carving faces process is shown in Figure 8.

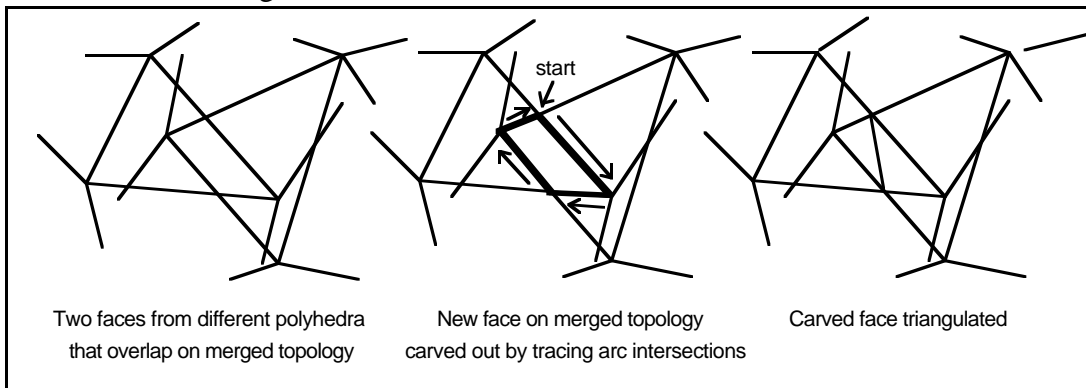


Figure 8

Note that if any of the resulting faces are not triangular, then they must be triangulated. This triangulation is necessary in order to insure that all faces will be co-planar during the morphing.

Once the topologies have been merged on the unit sphere, the merged topology must be mapped back onto both of the original objects. To find the coordinate of a vertex on a polyhedra in which the vertex was not originally on the polyhedra, barycentric coordinates must be used. Figure 9 shows this process.

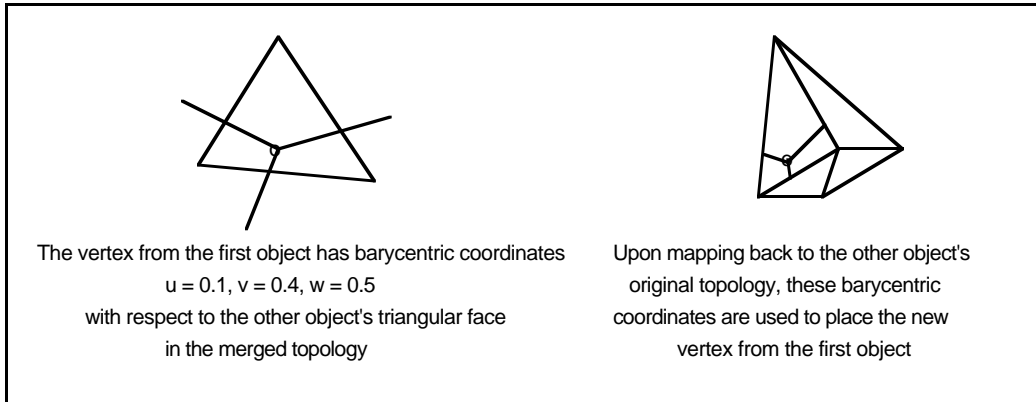


Figure 9

By applying this process to each vertex, the merged topology can be mapped back onto both objects.

Since each vertex in the newly mapped objects has a corresponding vertex in the other object, the correspondence problem has been solved and linear interpolation can be used to demonstrate the transformation.

Once the merged topology has been mapped back onto both objects, the work involved in this particular correspondence must be calculated. Treating the edges of the polyhedra as pliable wire allows us to calculate the work done in the transformation using a physically based system. As shown in Figure 10, each edge of a polyhedra is considered to be a wire that can be stretched or shrunk, and each angle between any two edges of a face allows us to bend the polyhedral wire frame.

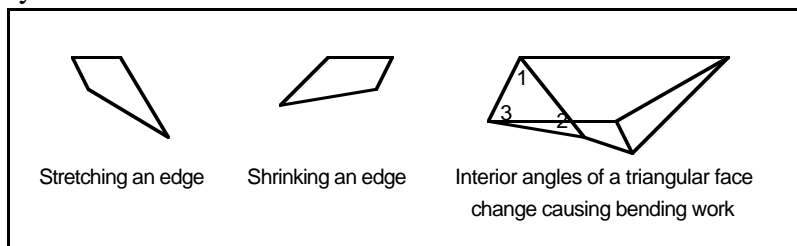


Figure 10

Since there is a one to one correspondence between vertices and edges in the two polyhedra, for each edge we can calculate the work done in stretching or shrinking it from the first to the second object. Let L_0 be the length of the edge from the first object and L_1 be the length of the edge from the second. Then, as in *A Physically Based Approach to 2-D Shape Blending*, the work done in stretching or shrinking this edge's wire is described by the equation

$$W = \frac{k |L_1 - L_0|^e}{(1 - c) \min(L_0 - L_1) + c \max(L_0 - L_1)}$$

he constants c, e, and k model collapsing edges, the plastic deformation of the wire, and the stiffness of the wire respectively.

The amount of bending work that is done in a transformation does not translate as easily from two dimensions to three dimensions as the stretching calculation does. Since the objects are in three dimensions, the approach in *A Physically Based Approach to 2-D Shape Blending* cannot be applied. Since the bending work is only being calculated at interior angles of the triangular faces along the transformation, the bending is happening on the plane of the face. Thus, the locations of the vertices of the start and end triangular faces can be mapped onto the XY plane by applying a projection of the respective face planes. This projection can be achieved by rotating the faces until their normals match up with the XY plane normal as shown in Figure 11.

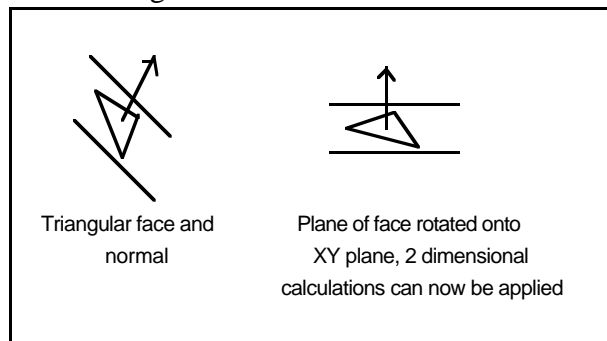


Figure 11

Once we have the vertices in a common two dimensional coordinate system, the two dimensional work algorithm can proceed as usual. As in *A Physically Based Approach to 2-D Shape Blending*, the bending calculation is

$$W = \begin{cases} \hat{E}k(+ m *)^e & \text{if (t) never goes through zero} \\ \ddot{E}k(+ m *)^e + p & \text{if (t) does go through zero} \end{cases}$$

where a degree two Bézier curve is used to measure * and detect if (t) goes through zero. The constants e, k, m, and p model the plastic deformation of the wire, the stiffness of the wire, the penalty for non-monotonicity, and the penalty for angles which go through zero respectively.

The total work involved in the transformation is the sum of the stretching and bending work.

As noted before, there are an infinite number of correspondences that can be set up through arbitrary rotation of an object's unit sphere before the merging process. To limit the search space, this algorithm sub-samples the possible rotations of the sphere and keeps track of a certain number of best (minimal work) solutions so far. Once a first pass of sub-sampling has been completed, the algorithm then goes back to the best solutions and sub-samples around those. This process of finer and finer sub-sampling continues until the algorithm is interrupted by the user or some time constraint. This sub-sampling process is shown in Figure 12.

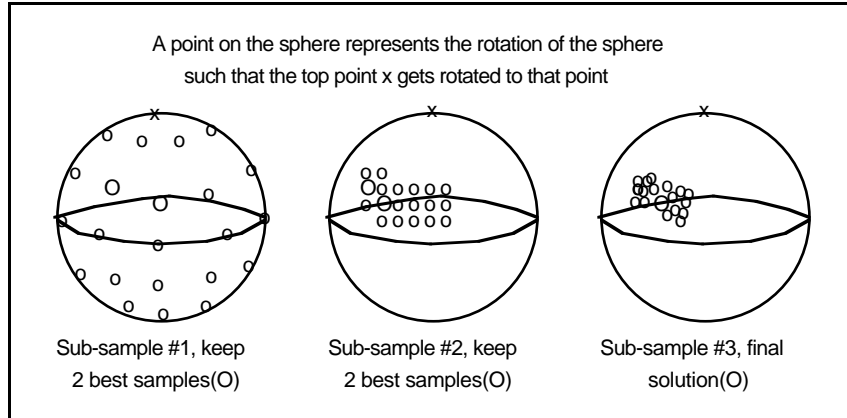


Figure 12

Since the effects of the rotation of the sphere before merging topologies is continuous, this sub-sampling method will be able to zero in on a good area of rotational values. The final solution, or rotation of the unit sphere, is the best (minimal work) solution found at the time the algorithm is interrupted.

Summarizing, the algorithm finds the projected topologies of both objects. Before merging these topologies, the algorithm sub-samples the rotational values to choose a rotation of one of the object's unit spheres. Then, the topologies are merged and mapped back onto the original objects. Last, the work calculation is computed. If the solution is one of the best so far, it is kept in a best solution list. The algorithm continues to sub-sample the rotational values and calculate the work involved for each sample until it is interrupted. At that point, the minimal work solution so far is returned as the final solution.

4.2 Analysis of Algorithm

To analyze the algorithm, each individual part will be analyzed separately. The first step of the algorithm is to find the projected topologies of both polyhedra. This process involves projecting each vertex of the polyhedra onto a unit sphere. If n is the number of vertices in the polyhedra, then projecting it onto a unit sphere is $O(n)$ since each projection is simply a constant time intersection calculation.

The second step involves sub-sampling the unit sphere. On each pass of sub-sampling, the algorithm takes a constant number of samples around each solution in the current best solution list. Since this best solution list is itself of constant size, then the number of rotational values that are sampled is constant per pass. In other words, each sub-sampling pass takes a constant amount of time. Therefore, if the algorithm is allowed to run a certain number of passes n , then this step takes $O(n)$.

The algorithm's bottleneck is the process of merging topologies. In order to merge the topologies of the two polyhedra, the algorithm must find all intersections of the arcs of the projected topologies. The algorithm currently implemented checks for intersection between all pairs of arcs between the projected topologies. Each arc represents a projected edge from one of the three dimensional objects. Consequently, if there are m edges in the first object and n edges in the second, the arc intersection process takes $O(mn)$. In the paper, *Shape Transformation for Polyhedral Objects*, an algorithm is given which has worst case behavior $O(mn)$ but, in general, is much smaller than $O(mn)$. The tracing of the arc intersections to carve out the new faces from the merged topology involves following the path of an arc and its subsequent intersections in a clockwise manner. An arc in the resulting merged topology (i.e., an edge on one of the final objects) is never traversed more

than twice since it can only belong to two faces and no backtracking is needed. Thus, if the merged topology has n arcs or edges, then carving out the faces of this topology is $O(n)$. Triangulation of the faces on the merged topology is a trivial $O(1)$ process. Finally, mapping the merged topology back onto one of the original models involves using barycentric coordinates for each vertex not originally on that model. This barycentric coordinate calculation is $O(1)$. Thus, if there are m vertices in the first polyhedra and n vertices in the second, mapping the merged topology back is an $O(m + n)$ process. All the other actions in the merging process are of a smaller order than the $O(mn)$ arc intersection calculation. Therefore, the whole merging process is $O(mn)$ where m is the number of edges from the first polyhedra and n is the number of edges from the second polyhedra.

Since the process of merging topologies occurs inside the sub-sampling process, this procedure as a whole is $O(p) * O(mn) = O(mnp)$ where m is the number of edges from the first polyhedra, n is the number of edges from the second polyhedra, and p is the number of sub-sampling passes performed.

Finally, the $O(n)$ process of projecting the polyhedra on a unit sphere is overshadowed by the cost of the $O(mnp)$ merging within the sub-sampling procedure. Thus, the final analysis of the algorithm is that it is $O(mnp)$ where m is the number of edges from the first polyhedra, n is the number of edges from the second polyhedra, and p is the number of sub-sampling passes performed.

4.3 Implementation of Algorithm

To implement the algorithm described above, I used an Silicon Graphics Indigo Workstation. The application I created takes as input two three dimensional polyhedra and some input on how fine the sub-sampling process should be. After calculating the solution, a window is shown with both objects in three dimensional space. The scene can be viewed from any angle and the objects can be drawn as wire frame polyhedra, solid filled polyhedra or Gouraud shaded polyhedra. The morphing transformation can be seen from the first object to the second and vice versa. The speed of the morphing can be controlled by the user and the colors of the objects are interpolated during the transformation as well.

Visually, the algorithm is very effective. By finding a near minimal work morph, a visually pleasing transformation is produced. These solutions tend to have minimal bending and stretching between the start and destination models which is exactly what is desired.

A valid question arises; how does the transformation produced by this algorithm compare to the real minimal work morph? The algorithm, since it sub-samples an infinite set of rotational values, will almost never exactly find the minimal work transformation. On the other hand, since the whole sphere is sub-sampled on the first pass, a rough solution will be found even before finer sub-sampling occurs. In other words, out of all the rotation samples taken, one will be closest to the true minimal work rotation. Since rotational values very close to the minimal work rotation will produce nearly the same work value, the function $f(\text{rotation}) = \text{work}$ is continuous except in cases where changing the rotation value slightly creates a different number of faces in the merged topology. Thus, barring these discontinuities, the closest rotational value to the minimal work rotation will return a slightly larger work value than the minimal one. Most likely, this value will be smaller than any other sample taken because of its proximity to the real solution. Finer sub-sampling around this minimal sample will produce even closer approximations to the real solution. Consequently, this algorithm relies on the function f being continuous.

There could be two problems with this algorithm's performance. The first is that there are some discontinuities in the function f which would cause the sub-sampling process to fail in a particular area of the sphere. In my opinion, these discontinuities are not a problem because they are rare compared with continuous samples. In other words, if a rotational value is sampled, in all likelihood the function f will be continuous at that value. The second problem is that even though a sample could be very close to the real solution, it could return a work value that is significantly higher than the minimal work value causing the sample to be discarded. Picture the function f at the minimal work solution as a small valley surrounded by a large plateau. Although f is continuous, it will be nearly impossible for the sub-sampling process to find the valley. As shown in Figure 13, it will most likely sample values on the plateau and conclude that there is no near minimal solution around those samples.

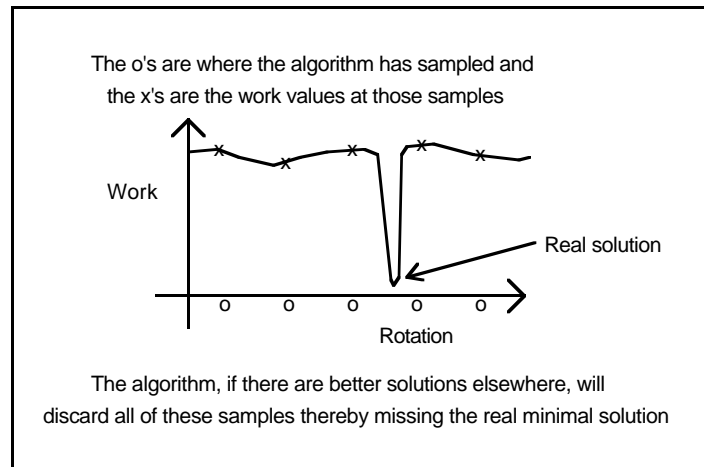


Figure 13

Thus, the minimal work solution will be lost. This problem is real, but it is inherent in a sub-sampling algorithm where certain best values can be skipped over. Therefore, the algorithm does not guarantee whatsoever to find a sample that comes to within a certain range of the real solution. In practice, the algorithm usually finds a reasonable solution, a visually pleasing transformation.

Theoretically, a near minimal work solution that is as close to the real solution as we desire can be obtained by letting the algorithm run as long as it needs. After each sub-sampling pass, the best solution so far will get closer and closer to the real solution. Eventually, the best solution will be within a certain tolerance range of the minimal work solution. Unfortunately, this method is not practical because the algorithm could be running for days. Also, the algorithm, as discussed in the previous paragraph, could have skipped over the real solution! In practice, however, very few passes of sub-sampling need to be completed before the best solution found is visually pleasing.

The complexity of the objects being transformed definitely affects the performance of the algorithm. Figure 14 is a table which shows some running times of the application on the SGI with different objects.

	Pyramid has 9 edges	Cube has 18 edges	Sphere has 36 edges
	Each run sub-samples the same number of times		
	Pyramid	Cube	Sphere
Pyramid	23	46	84
Cube	46	82	154
Sphere	84	154	268
	Running time in seconds		

Figure 14

4.4 Future Directions

For future paths dealing with this project, there are many possibilities. First, the faster algorithm described in *Shape Transformation for Polyhedral Objects* could be implemented. In fact, I attempted to implement this algorithm but did not have time to fully debug it.

Second, a different method of interpolation might produce more visually pleasing results. Linear interpolation by its nature produces some transformations that appear to shorten or lengthen objects in a somewhat unnatural way. This area could be investigated to see if another form of interpolation would be more beneficial.

Third, at the moment only star-shaped polyhedra can be transformed using the algorithm in this paper. This restriction is very severe if one wants to morph arbitrary three dimensional polyhedra. Work could be done to determine if complex polyhedra could be split up into several star-shaped polyhedra and then morphed in some way using the algorithm described in this paper.

Fourth, the way in which the algorithm produces a correspondence between the two three dimensional polyhedra might not be the best method. Other more general techniques of solving the correspondence problem between polyhedra could be studied.

5.0 Acknowledgments

I would like to thank my honors advisor Professor Robert Geitz for all the suggestions and guidance he has given me throughout my honors project. I would also like to thank the Oberlin College Computer Science department for providing me not only the chance to do an honors project, but the equipment to implement the project.

6.0 References

Thaddeus Beier and Shawn Neely. "Feature-Based Image Metamorphosis." *Computer Graphics*, 26(2):35-42, July 1992. Proceedings of SIGGRAPH '92 (Chicago, July 26-31, 1992).

Wayne Carlson, James Kent, and Richard Parent. "Shape Transformation for Polyhedral Objects." *Computer Graphics*, 26(2):47-54, July 1992. Proceedings of SIGGRAPH '92 (Chicago, July 26-31, 1992).

Eugene Greenwood and Thomas Sederberg. "A Physically Based Approach to 2-D Shape Blending." *Computer Graphics*, 26(2):25-34, July 1992. Proceedings of SIGGRAPH '92 (Chicago, July 26-31, 1992).

John Hughes. "Scheduled Fourier Volume Morphing." *Computer Graphics*, 26(2):43-46, July 1992. Proceedings of SIGGRAPH '92 (Chicago, July 26-31, 1992).