Shape Distortion in Computer-Assisted Keyframe Animation

E. Wesley Bethel and Samuel P. Uselton

ABSTRACT

Distortion of one object shape into another is a tool that has been used effectively in two dimensions by animators in entertainment, education and communication for many years. This paper describes a method for computer support of a three-dimensional extension of this technique. If two object descriptions are topologically equivalent, then one object can be continuously deformed into the other by interpolation between the vertex positions of the two descriptions. Using a formal definition of polygon-based object descriptions, an algorithm for deciding the topological equivalence of two object descriptions (and for constructing a vertex correspondence) is developed. If the two descriptions are not topologically equivalent, some modifications to the vertices and edges of the descriptions are required before distortion of one into the other can be accomplished. This algorithm also computes an intermediate object description by the addition of duplicate vertices and degenerate edges and faces to one or both initial object descriptions. The intermediate object description can be distorted from one object to the other by vertex position interpolation.

KEYWORDS: keyframe, shape distortion, edges, faces

1. INTRODUCTION

Distortion of one object shape into another is a tool that has been used effectively in two dimensions by animators in entertainment, education and communication for many years. Stretching and squashing characters and objects to dramatize events and the characters' reactions to them is standard practice among the animators creating higher-quality cartoons ([LASS87], [THOM81]). In a more absurd vein, how many cartoon characters have exhibited heads distorted into the shape of some object that was just removed from their shoulders?

Shape distortion was one of the first difficulties addressed in providing computer support for two-dimensional keyframe animation [BURT71]. Most computer-assisted keyframe animation work, however, has concentrated on specifying the path followed by a single object that appears in different positions and orientations in consecutive keyframes ([REEV81], [STEK85] and [SHEL82]). Since only a single object description is used (with different transformations), the only shape changes are those that can be expressed by the usual matrix notation. Barr extended this form of shape distortion by permitting functions to replace constants in the matrix [BARR84]. Another type of shape distortion can be accomplished by allowing different transformations to be applied to disjoint subsets of the object's vertices or control points. In the limit, this method would use a separate transformation matrix for each point. While specifying a transformation for each point is awkward for an animator, the same thing can be accomplished by listing the starting and ending positions of each point. A set of transformations can be calculated, although interactive editing of the sequence will probably be desired. The distinctive feature that makes this strategy easy is that the organization of vertices into polygons, or control points into patches, is not changed.

Continuously deforming one object into another could be accomplished by this method, if the two objects have the same connectivity of vertices, edges and polygons. In general, two objects chosen as the starting and ending targets may have very different structures. The number of polygons or vertices may be different, the number of vertices defining some polygons may vary, or the number of polygonal faces meeting at a vertex may be different from one object to the other. Even if the numbers of vertices, edges and polygons all match, determining which particular points, edges and polygons correspond is a difficult problem.

This paper describes a method for computer support of three-dimensional shape distortion, by automating the process of determining this correspondence. The object description most commonly used in computer graphics is a polygon-based description in which each face of an object is represented by a sequence of vertices, each of which has a three-dimensional coordinate position. The edges between vertices may be implicit in the order of the vertices, or they may have an explicit representation. This structure of vertices and edges may be interpreted as a graph, in the mathematical sense.

The term "topologically equivalent" will be used to describe objects whose object descriptions are isomorphic graphs, that is, there is a one-to-one correspondence between faces, vertices and edges. If two objects are topologically equivalent, then one object can be continuously deformed into the other by interpolation between the vertex positions of the two descriptions using any of the previously developed techniques, once the correspondence between vertices is established. However, graph isomorphism is known to be a difficult problem [GARE77]. Specifying the vertex correspondence manually for even modestly complicated objects is quite tedious and error-prone. These difficulties have limited the development of computer support for three dimensional keyframe animation. This paper describes a major step in overcoming these difficulties.

A difficult problem arises if the number of geometric elements, or even their arrangement, varies. In this case the two descriptions are not topologically equivalent. Some modifications to the vertices, edges and faces of the descriptions are required before distortion of one into the other can be accomplished. The "Super-Constructor" Algorithm computes an intermediate object description by the addition of duplicate vertices and degenerate edges and faces to one or both initial object descriptions. This intermediate object description may be rendered to appear identical to either of the initial objects depending upon which vertices are geometrically duplicates and which faces and edges are degenerate. Since it is really only one description, it is topologically equivalent to itself and therefore its appearance can be distorted from that of one object to the other by the same vertex position interpolation methods referenced above.

Section Two describes the mathematical foundations required for discussing the solutions to these problems. Formal definitions of object descriptions and topological equivalence are presented and some consequences of these definitions are explored. Section Three develops the solution to the problem of constructing a new object from the two input objects. The vehicle for distortion from one input object to the other is vertex interpolation performed upon the new object. Section Four contains a discussion of the algorithm for constructing the new "super-object". Section Five presents example sequences resulting from the use of the Super-Constructor Algorithm, as implemented. The final section, Section Six, evaluates the results and describes possible extensions to this work.

2. MATHEMATICAL FOUNDATION

2.1. Theoretical Background

The most common type of object description used in computer graphics is a collection of polygons which define the surface boundary of the object. The polygons are usually specified by a sequence of edges or vertices. Both the geometric information and the connectivity between polygons of the same object are embodied in these lists of edges or vertices. The term "face" will be used in the following discussion to refer to polygons, spline patches or any similar surface definition. The mathematics needed for the formal analysis of matching objects has been developed in topology. Each face is a topological disk. Any topological disk has no holes nor isolated points within its interior and can be deformed to a circular region [MORT85].

Definition 1: A *polygon* is a (two-dimensional) topological disk whose circumference is divided into a certain number $R\,(R \geq 2)$ of arcs, called edges or sides (each of which is geometrically a straight line segment) by means of the same number, R, of points called vertices. The polygon is completely and uniquely specified by the topological disk and these R arcs. When $R>2$, a topological polygon is always homeomorphic (can be continuously deformed) to a convex (non-degenerate) polygon of the kind studied in elementary geometry [FREC67]. In this paper, the only topological disk or face allowed is a polygon with $R>2$ sides. Polygonal faces can be combined to form a polyhedron.

Definition 2: A *polyhedron* is a closed, oriented surface made of polygons such that two and only two polygons meet at each edge [MORT85]. Polygons meet only at their edges (interpenetrating surfaces are invalid), and polygons are oriented so that each edge is used exactly once in each orientation (Moebius strips, Klein bottles and the like are invalid). For any vertex, all the polygons of this system having this vertex in common can be arranged in a cyclic order $P_1, P_2, \ldots, P_n$ such that P_i and P_{i+1} $(1 \leq i \leq n$ where $P(n+1) \equiv P(1))$ have a common side (edge) passing through this vertex.

In all simple polyhedra with V vertices, E edges and F faces, Euler's formula for polyhedra ([MORT85],[FREC67]) states that

$$V-E+F=2 \tag{2.1}$$

For complex polyhedra (those with genus >1), Poincaré's extension to Euler's formula can be expressed as

$$N(0)-N(1)+N(2)- \cdots (-1)^{(n-1)}*N(n-1)=1-(-1)^n \tag{2.2}$$

where $N(i)$ is the number of items of dimension i. While it is possible to construct an object composed of multiple shells, we will not consider it to be a (single) polyhedron. The multiple-shell object is disallowed primarily because a traversal beginning at some face and proceeding from face to face across shared edges cannot possibly visit every face in the object. We will, however, permit polyhedra with holes. To compute the number of holes in an object:

$$V - E + F - H + 2P = 2B \qquad (2.3)$$

where H is the number of "holes in faces" or islands in polygons, P is the "number of passages through objects," and B denotes the number of shells or disjoint bodies in an object [MORT85]. V, E and F are as in (1.1). The B term will be 1, and the H term will be zero, as all faces must be topological disks. We are left with G holes, computed as

$$-(V - E + F - 2)/2 = G \qquad (2.4)$$

We may now examine the problem of determining the likeness of two objects. This comparison must use several classes of information. For example, if two objects to be compared have different numbers of faces, then the two objects are obviously different. However, two objects may have the same number of faces but still be different; for example, the number of edges bounding some faces may be different. A similar case applies for vertices; the number of vertices in the objects' descriptions may be the same but the number of edges incident can differ. These global object description comparisons are insufficient, because we must also be concerned with the arrangement and adjacencies of the object.

Consider a set V of vertices, a set E of edges and a set F of faces. A polyhedron may be thought of as a graph composed of vertices and edges, where each edge connects exactly two vertices. Two objects are said to be equal if the following conditions hold: 1) the vertices of each graph (object) may be placed in a one-to-one correspondence; 2) the edges of each graph may be placed in a one-to-one correspondence, and 3) for each edge E_i in one object connecting vertices V_j and V_k, the corresponding edge E_c in the other object connects the vertices V_d and V_e which correspond to V_j and V_k, respectively. The polygons forming the object can be consistently oriented. If a polygon is formed from a sequence $V_1, V_2, \ldots, V_n$ of vertices, then this polygon is different from one composed of the sequence of vertices $V_n, V_{n-1}, \ldots, V_2, V_1$. The task of comparing polyhedra may be thought of as a graph isomorphism problem. Unfortunately, the computational complexity of graph isomorphism is an open problem, conjectured to be NP-complete [GARE77]. However, the problem of graph isomorphism for graphs of bounded valence has been shown to be computable in polynomial time ([GALI87], [LUKS88]). The linear relationship between the numbers of vertices, edges and faces and the restricted structure of the polyhedron, however, make the complexity of comparing two polyhedra much easier to determine. The straightforward approach of trying all possible matchings for the necessary one-to-one mappings between vertices and between edges is clearly unacceptable; the complexity is $O(n!)$. However, trying to pair one vertex from the first polyhedron successively with each vertex from the other and using the adjacency constraints inherent in the graphs leads to an algorithm with $O(n^3)$ running time. While theoretically reasonable, for complex object descriptions this time is still unacceptable.

Determining whether two graphs are isomorphic can be accomplished quite quickly if the two graphs are ordered, rooted trees [GARE77]. In fact, the computation time is linear in the size of the tree. The graph of a polyhedron, however, contains many cycles. Is there a method that will replace the polyhedron graphs by trees that contain all the same information? The authors' implementation uses an arbitrary face of the polyhedron as the root of a tree. Next, using the orientation of the polygon to order the tree, a vertex (or edge) on that face is selected to break the circularity of the order. Then, a Breadth First Search (BFS) Tree [AHO83] is constructed of adjacent polygons, while keeping the "back-edges" (to previously found polygons). If the natural orientation of the polyhedra is used to control the order of expansion of nodes in the BFS, then the correspondence of positions in the trees can be used to establish the one-to-one mapping between (tree) vertices and between forward edges. The tree vertices correspond to polygons and the tree edges joining two vertices correspond to the polyhedron edges shared by the corresponding polygons. The back edges must also be checked, since they also correspond to polyhedron edges. The tree so constructed, plus the back edges, is the dual graph of the polyhedron. Isomorphism between the dual-trees implies isomorphism between the original graphs as well.

If one vertex from the first polyhedron is successively matched with each vertex of the second polyhedron, and for each such pair of vertices, one edge incident to the first vertex is matched with each edge incident to the second vertex, we again have an $O(n^3)$ algorithm for polyhedral graph isomorphism.

2.2. Application

At this point we may assume that the two objects in question are different. For the purposes of in-betweening, the two "key" objects must be similar so that the different positions of the corresponding vertices can be used to specify the incremental motion required for each frame. If the two polyhedra fail the test for polyhedral graph isomorphism (above), a more difficult problem must be solved. A third object description must be constructed so that by super-positioning (hence the name "super-constructor") some vertices, it matches the geometry of one polyhedron, and by another super-positioning it matches the second polyhedron.

An artifact of in-betweening may be capitalized upon when computing the positions of the vertices during the inbetween frames. The animator can make faces in an object disappear over the sequence of inbetween frames by translating all vertices on a face to a single point or edge. This disappearance is a simple geometric operation not affecting the well-formedness of an object. As no vertices, edges nor faces are added to or removed from the object description, the topological properties of the object do not change. Faces can be made to appear by calculating it as a disappearance in reverse.

The difficulty with this solution is that the new object description must have faces that correspond to all the faces in both of the original objects, arranged so that degenerate (zero area) faces are the only ones interfering with the original adjacency constraints. Methods for the addition of elements to well-formed object descriptions which preserve the well-formedness have been surveyed ([WEIL85], [BRAI78] and [MORT85]). These methods of addition (and subtraction) are known collectively as the "Euler Operators."

If we assume that the use of vertex and edge chamfering [BRAI78] can change any object into any other object (simple polyhedra), then the difficulty arises in determining exactly which vertices and edges in each object must be replaced to achieve the desired effect. To change the genus of an object description, the required modifications to an object description are even more difficult, as the position for a degenerate hole must be found.

3. CONSTRUCTION OF A "SUPER-OBJECT"

This section discusses the context of the problem and the approach used in the development of the Super-Constructor Algorithm presented in this work. As will be highlighted, the Super-Constructor Algorithm appears to be quite similar to (historical) NP-complete problems.

One method of finding differences in objects may be described in the following way. Given some starting face on each object, some set $S(f)$ of faces are "common" to each object. The common set may be thought of as the set of faces in each object which satisfies the one-to-one, onto requirement and the adjacency relation constraint previously outlined. Geometrically, S represents the set of faces which require no super-positioning in the representation of the two key objects. Intuitively, S may also be thought of as that set of faces which do not collapse to or expand from degenerate faces over the course of in-betweening. There must be some set $F-S(f)$ (F is the set of all faces in an object) which contains all faces not in the "common" set. Furthermore, there is a set $F-S(f)$ for each object. This generalization suggests that if the sets F and S are not equivalent in objects A and B that at least $(F(a)-S(f))+(F(b)-S(f))$ faces must be added to make the objects equal.

More precisely, given two objects A and B, $F(a)-S(f)$ faces must be added to object B, while $F(b)-S(f)$ faces must be added to object A. These are the minimum number of face additions required to satisfy Euler's formula. It appears this value is the absolute minimum number of face additions required to make the objects "equal". If object A is to represent object B via super-positioning, the addition of $F(b)-S(f)$ faces is necessary. Conversely for object B, $F(a)-S(f)$ face additions will be required. The net result is a minimum addition of $F(a)+F(b)-2S(f)$ faces.

Determining which faces from a pair of objects are in the common set $S(f)$ is not a trivial process. Since there is a cyclic ordering of faces about a vertex, there is also a cyclic ordering of edges about a face and faces that share these edges. If, for a pair of corresponding faces, the number or arrangement of these cycles of adjacent faces is not equivalent, then it is not straightforward to determine which of these faces is in S. In objects of modest complexity, the number of combinations of faces which could be in S becomes quite large, especially if the objects are very "different" or if $S(f)$ is small in proportion to F. Clearly, it is desirable to maximize the number of faces in $S(f)$, since this results in the fewest number of face additions to each object. An upper bound on the size of $S(f)$ is the number of faces in the object which has the fewest number of faces.

Since no obvious, efficient method of determining $S(f)$ exists, and since determining $S(f)$ is quite similar to the subgraph isomorphism problem (which is known to be NP-complete from [GARE77]), it is conjectured that determining the optimal $S(f)$ is also an NP-complete problem. Recall the intuitive basis for NP-completeness: there is no obvious way of isolating the optimal solution short of an exhaustive search of (a large fraction of) all possible combinations [SEDG83]. In the context of this paper, optimal has been defined as maximum $S(f)$, or alternately, as a minimal number of face insertions. This definition is important in the design of an algorithm, since there may be more than one $S(f)$ which is maximal in size. In that case, the first optimal $S(f)$ could be used. However, some other $S(f)$ might also be used which would yield a more visually pleasing result. With respect to algorithm design, a heuristic which results in "most visually pleasing" can be built into the algorithm. As this requirement is subjective, one implementation could require the animator to run the program many times, each time varying some parameters, to achieve the desired results. Another alternative could assume a sophisticated user interface allowing complete control over how $S(f)$ is constructed, guaranteeing the "most visually pleasing" result. The authors have chosen to use the former strategy in the implementation.

In designing an algorithm which finds a maximal $S(f)$, and in addressing the intuitive "search all possible combinations" requisite for NP-completeness, the question arises, "What exactly is a combination?" A combination consists of a dual graph of the faces (and face adjacency relationships) of each object in some $S(f)$. If the solution to this problem was designed as in traditional NP-complete problems, some initial $S(f)$ would be generated and successive "passes" of the algorithm would improve upon the initial solution and approach some optimal solution. The runtime of this type of algorithm is exponential with respect to the number of faces in an object.

Given that finding an optimal $S(f)$ is very expensive, an algorithm which generates one $S(f)$ will be developed. If well designed, the initial $S(f)$ should be a good approximation of an optimal $S(f)$. Further research topics would include generating better initial approximations, as well as the various improvements upon an initial $S(f)$ to yield an optimal

$S(f)$. The authors'
and then find the rer
For example, if a pai
of adjacent faces. If
a sequential face-by-
maximum contributio
don't make a contrib
added to (at least) or
Later, these degenera
will be compared to

4. THE SUPER-CC

The purpose of the
relationship between
an object description,
an object description.
Input to the Super-C
Given the dual-trees
objects will later be r

4.1. The Vertex Rel

In the following disc
represented by the du
comparison (of corres
tionship between the
is to create a conditic
and adjacent faces.
faces are similar the

As vertex pairs are p
one object to its cou
Algorithm in computi
edges about correspor

The heart of the Supe
upon that relation to
performed upon the
First, we must know
this is an "original" v
object description. Tl
relation which is mor
there are an equal nu
change from vertex t
comparing positionwis
the position we're cur
of vertices on each fa

The vertex relation, w
n-tuple, and impleme
segment-to-window re
digm, each bit in the

The fields in the bit s
zero indicates this ver
Bit one indicates the
"maps to" some verte
one. Bit four is set if
object two (this condi
and seven are used to
face from object one

$S(f)$. The authors' implementation approaches the problem in a slightly different way. Rather than find a "best" $S(f)$ and then find the remaining $F-S(f)$ faces which must be added to each object, a more "bottom-up" approach is used. For example, if a pair of corresponding faces each contain an equal number of sides, they also contain an equal number of adjacent faces. If the edges on each face correspond, then $S(f)$ is determined both locally and maximally. That is, in a sequential face-by-face evaluation of S, all unvisited adjacent faces must belong to S. On a local scale, this is the maximum contribution that can be made to S from each object. Faces which have been visited may also correspond, but don't make a contribution to S. If there are an unequal number of edges on each of corresponding faces, edges must be added to (at least) one of the faces, along with the same number of faces as edges in the new object to be constructed. Later, these degenerate faces (resulting from the addition of a degenerate vertex and edge) composed of only a vertex will be compared to original, unmodified faces (that is, faces with edges), and the edge/face modification process repeats.

4. THE SUPER-CONSTRUCTOR ALGORITHM

The purpose of the Super-Constructor Algorithm is to generate a pair of dual-trees in which there exists a one-to-one relationship between the vertices and edges of each dual-tree. In a dual-tree, a vertex represents a topological face from an object description, while an edge in the dual-tree represents an adjacency relation between two topological faces from an object description.

Input to the Super-Constructor Algorithm consists of a pair of dual-trees, one for each of the objects to be distorted. Given the dual-trees representing two objects, the goal is to compute not one but two super-objects. These two super-objects will later be rendered using vertex interpolation to give the illusion of shape distortion.

4.1. The Vertex Relation

In the following discussion, the terms face, vertex and edge refer to those constructs in the original object description represented by the dual-tree. Commencing at the root of each dual-tree, a parallel breadth-first traversal occurs. At each comparison (of corresponding faces), the relation among corresponding vertices is evaluated. Based upon the exact relationship between the two vertices, a variety of events may occur. Suffice it to say the goal at the face comparison level is to create a condition where a one-to-one, onto relationship exists between corresponding (positionwise) vertices, edges and adjacent faces. Note that if this condition is achieved, the corresponding faces are similar. If all corresponding faces are similar the objects are topologically equivalent.

As vertex pairs are processed, correspondence tables are constructed which portray the exact mapping of a vertex from one object to its counterpart vertex in the other object. The correspondence tables are used by the Super-Constructor Algorithm in computing the vertex relation and in processing to achieve the one-to-one relationship between vertices and edges about corresponding faces.

The heart of the Super-Constructor Algorithm lies in evaluating the relation between two vertices and subsequently acting upon that relation to achieve similarity at the face level. The relation represents the concatenation of a battery of tests performed upon the pair of the vertices. The elments of interest which comprise the relation describe several things. First, we must know if each vertex in the pair being examined is present in its respective object description. That is, if this is an "original" vertex. Second, we need to know if each vertex in the pair corresponds to any vertex in the other object description. The case where each vertex in the pair corresponds to the other vertex in the pair represents a unique relation which is more specific than simply mapping to some vertex in the other object. Fourth, we need to determine if there are an equal number of vertices on each of the corresponding faces (as we shall see, this relation may indeed change from vertex to vertex while processing a pair of corresponding faces). Finally, in the context of sequentially comparing positionwise vertices about corresponding faces, we are concerned with whether or not there exists a vertex in the position we're currently examining. This test becomes meaningful when combined with the test for an equal number of vertices on each face.

The vertex relation, which is the concatenation of the preceding tests, is conveniently represented and manipulated as an n-tuple, and implemented as an n-digit binary string. Such a representation is similar to the representation of the line segment-to-window relationship used in the Cohen-Sutherland Line Clipping Algorithm [NEWM79]. Using this paradigm, each bit in the string represents the state or result of one of the tests applied to the vertex pair.

The fields in the bit string are set if a given test is "true" or cleared if "false." The fields are assigned as follows. Bit zero indicates this vertex is an "original" vertex, that is, present in the original object description for object number one. Bit one indicates the same thing, but for the vertex in object number two. Bit two is set if the vertex from object one "maps to" some vertex in object two; similarly, bit three is set if the vertex in object two maps to some vertex in object one. Bit four is set if there exists a vertex in this vertex position for object one; bit five reflects the same condition for object two (this condition becomes relevant when, for example, two faces have differing numbers of vertices). Bits six and seven are used to indicate the "less than" relation of vertex counts on the faces being compared. If bit six is set, the face from object one has fewer vertices than the corresponding face on object two. Note that this relation can change over the course of processing corresponding faces. Bit eight is used to indicate that the vertex from object one "maps to" the vertex on object two. Note the difference between bit eight and bits two and three. Bits two and three indicate only that there is some mapping from each respective vertex into the set of vertices for the other object. Bit eight indicates specifically that these two vertices correspond. Bits eight, two and three are not mutually exclusive; in fact, if bit eight is set, so are bits two and three. It should be noted that computing the values of each of these bits is trivial, assuming the use of lookup tables to represent the mapping relations from all vertices of an object to all vertices of the other object.

Thus, at each vertex position in the face comparison process, the relation between the two vertices corresponding in position is computed and the "appropriate" action is taken to produce a pair of vertices which correspond. There are three broad classes of actions, only one of which occurs, based upon the specific value of the vertex relation. First, "no action" is required. This means that either the vertices indeed already correspond or that it is allowable to assign correspondence from each to the other. Of course, the correspondence tables must be updated to reflect this newly found mapping. Second, "limited action" is required. Limited action entails creating a new vertex and inserting it into the "appropriate" face at the "appropriate" location. Again, the correspondence tables must be updated to reflect both the creation of a new vertex and the new mapping associated with the new vertex. Third, "extensive action" is required. This usually involves changing an existing vertex mapping and the creation of a new vertex. Extensive action is required when, for example, a vertex from object A maps to some vertex in object B, but not the vertex in object B we're currently examining.

In addition to the various operations applied to the vertices and correspondence tables at each positionwise vertex comparison, degenerate faces are added as needed. The need for a degenerate face is easily detected when, for example, an adjacency relation is present in one object but not the other at some positionwise corresponding edge. The adding of degenerate faces occurs only in association with either limited or extensive operations, as described above.

The pseudocode for the Super-Constructor Algorithm follows, in a C-like syntax.

```
void
super_constructor(d1,d2)
dual_node *d1, *d2;
{
  dual_queue *dq1, *dq2;
  int i,r;

  dq1->dual_node_pointer = d1;
  dq2->dual_node_pointer = d2;
  dq1->next = dq2->next = NULL;

  while ((dq1 != NULL) && (dq2 != NULL)) {
    for (i=0;((i<=d1->number_of_vertices)&&(i<=d2->number_of_vertices));i++){
      r = compute_relation(d1->vertex[i],d2->vertex[i]);
      switch(r) {
        /* process relation performing "no action", "limited action"
           or "extensive action" as required */

        /* add faces to end of each queue (for breadth-first traversal) */
      }
    }
    dq1 = dq1->next,  dq2 = dq2->next;
  }
}
```

Note that this algorithm merely generates super-dual-trees, not super-objects. A final step is required to regenerate an object from a dual-tree.

The preceding algorithm has linear running time with respect to the number of faces in the object with more faces. However, hidden costs are introduced by the fact that additional faces could be created and must subsequently be processed. The extra processing can be estimated to be of order $F(a)+F(b)-2S(f)$ (from Section Three). Since the runtime of the algorithm is at least of order $\max(F(a),F(b))$, it is conjectured that the number of face additions will be bounded above by $F(a)+F(b)-1$, but is likely to be much closer to $\max(F(a),F(b)) + (\min(F(a),F(b))-S)$.

5. EXAMPLES

Three animation sequences were created to show the results of the Super-Constructor Algorithm. Wireframe rendering is used to show all vertices, edges and faces. The key objects for these sequences all have a different number of vertices, edges and faces. The part matching and reconstruction proceeded completely automatically, with only the initial part match specified by a user. In Animation Sequence One, a cube is changed into an obelisk. The face specified by the user as the initial part match is the face on the "bottom" of each of the cube and obelisk. Thus, the distortion of shape occurs in areas where these objects differ the "most," namely on the "top" of each of the objects. Animation Sequence Two illustrates the distortion of shape between objects of differing genus in which a cube is "changed" into a torus. Again, the initial part match specified by the user is the "bottom" of both the torus and the cube. Animation Sequence Three illustrates an octahedron changing into an icosahedron. The initial part match specified by the user is one of the faces on the upper left side of each of the objects.

Table 5.1
Super-Constructor Algorithm Results

	Animation Sequence 1			Animation Sequence 2			Animation Sequence 3		
	Source	Dest.	Super	Source	Dest.	Super	Source	Dest.	Super
Faces	6	9	9	6	16	16	8	20	20
Verts.	8	9	10	8	16	16	6	12	12

Table 5.1 contains information about the original object descriptions and the super-objects in the examples. Listed are the values for numbers of faces and vertices for each of the source, destination and super-objects. Perhaps the only unexpected result is that for the vertex count in the super-object for Animation Sequence One. In this case, the top of the obelisk has four faces, each of which has three sides. However, the top of the cube has only one face with four sides. This difference is rectified by adding a single vertex to the object description of the obelisk. In this way, one of the three-sided faces is changed to a four-sided face corresponding to the top of the cube.

6. SUMMARY AND CONCLUSIONS

This work has presented the development of a tool which enables the in-betweening of objects of extremely different shape. The scope of objects has been defined as those which use modeling techniques for describing surfaces as piecewise, connected topological disks. A method of determining the likeness of object pairs has been identified. An algorithm outlining the construction of an output object which may geometrically represent the two source objects has been presented.

The automation of the part matching problem has been successfully solved and implemented for polygonal object types. Prior to this work, there has been no well-known solution enabling the in-betweening of three-dimensional objects which differ drastically in shape. The intent has been to provide a tool for animators allowing a new dimension in creativity, not unlike the effect created by the work of Burtnyk and Wein in the early 1970s for two-dimensional objects.

There are several possible extensions to the Super-Constructor Algorithm. All objects are assumed to be composed of a collection of abutting polygonal faces organized into a polyhedron. Fortunately, other surface representations methods are available to the surface modeler. The steps required to modify our previous work to perform such a distortion on a spline patch-based object are not very difficult. In fact it may be that the shape distortion methods presented are better suited for patch-based objects than for their polygonal counterparts.

A spline patch as a biparametric surface could be shown to be a topological disk (homeomorphic to a circle). A pathologic case, as far as this research is concerned, is the self-intersecting patch. Only locally two-dimensional (that is, not self-intersecting) spline patches will be considered in the following discussion. Objects such as the Utah teapot are composed of a system of spline patches. Such a system of patches can be shown to be as well-formed as a system of polygons forming a polyhedron.

Objects composed of spline patches lend themselves to the following generalization. "All faces have (or can be coerced to have) four edges." This observation will simplify the algorithm from the previous section somewhat: the case of a five- and three-sided polygon being compared will not occur. Thus, the edge-insertion algorithm need not approach the complexity required for polygonal objects, resulting in a relatively simplified implementation.

To process spline patch-based objects, the Super-Constructor's edge-following algorithm will require modification. In a polygon, a pair of adjacent vertices defines an edge of the face, whereas in the spline patch, a sequence of control points defines an edge. Another problem area is the situation in which control points must be added to a patch, or the degree of patch modified. These problems have been previously addressed ([COHE80] and [COHE85]).

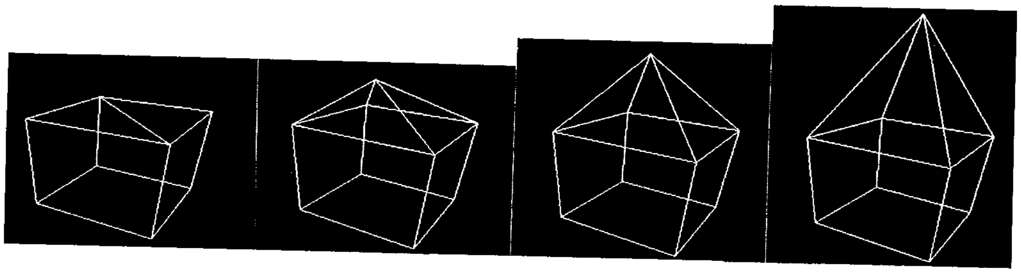
The use of spline patch-based objects in the extreme modification of shape appears to be a useful and viable extension to the work presented in the previous section. Modification of the Super-Constructor Algorithm would be useful due to the large number of surface modelers which use the spline patch as a basis surface descriptor. Viability has been demonstrated by the limited number and scope of changes required to the Super-Constructor Algorithm for the purposes of supporting the spline object type.

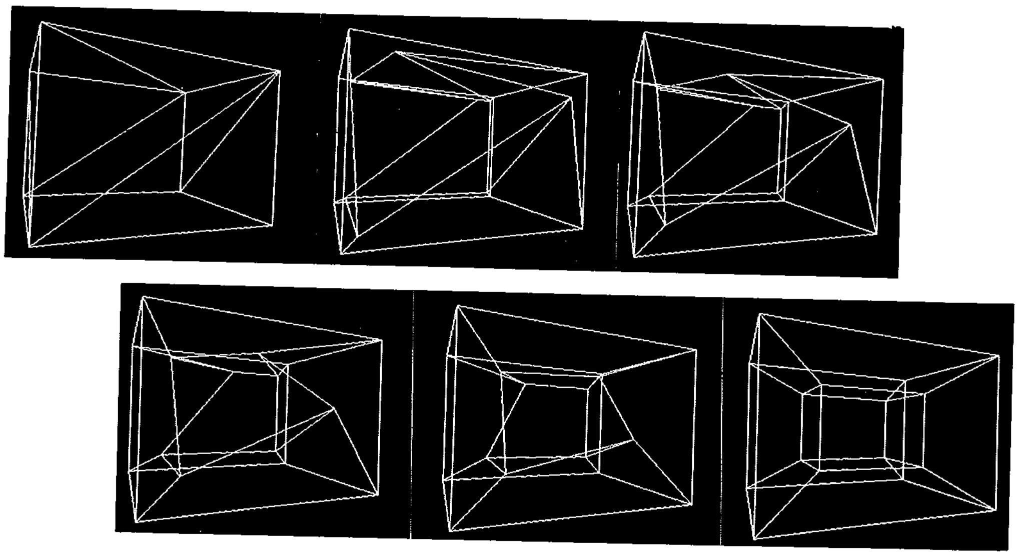
Additional extensions to this work could use information which is more global in nature in computing the relation between corresponding vertices. For example, the use of information about previous and subsequent vertex relationships (in the linear sequence of vertices about each face) could influence what happens at a given vertex.

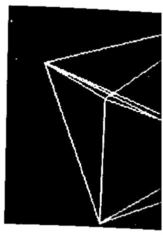
Further extensions could involve testing to ensure that all polygons are planar. It should be noted that this problem lies not in the design of the Super-Constructor Algorithm itself, but in the assigning of geometric values to the vertices created by the Super-Constructor. Object interpenetrations and self intersections could be prevented in future work, also at the final step when geometric values are assigned to newly created parts. This type of extension uses previous work on constraining motion paths [REEV81].

A characteristic of the Super-Constructor Algorithm, as demonstrated in the Animation Sequences, is that the "least" amount of shape distortion tends to occur "around" the initial part matches specified by the user. This characteristic could be used to the animator's advantage in an implementation where more than a single initial part match is specified. Multiple initial part matches would specify, in effect, that a given area is to remain "relatively" undistorted, hence provide extra degrees of freedom for the animator.

Another area for additional work is increased interaction with the animators. One of the goals of this research has been to automate the construction of the new, super-object. In terms of user intervention to control the construction process, two main items could be overridden by a user. The positions and attributes of newly inserted elements could be corrected or steered by an interactive user. These choices may be impacted by the interpolation paths chosen for the individual vertices. Therefore, a tightly coupled system including several vertex interpolation techniques would be very useful. These two ideas are well suited for further development.


Animation Sequence One


Animation Sequence Two



7. REFERENCES

[AHO83] Aho, Apc

[BARR84] Barr, Al
Three, August 19

[BETH86] Bethel, V
1986.

[BRAI78] Braid, I.,
Group Document

[BURT71] Burtnyk,
ture and Televisio

[BURT75] Burtnyk,
Number One, 197

[BURT76] Burtnyk,
of the ACM, Volu

[COHE80] Cohen, E
Design and Comp

[COHE85] Cohen, E
Graphics, July 198

[EAST79] Eastman,
ics in CAD/CAM

[GARE77] Garey, M
Freeman and Co.,

[GALI87] Galil, Z.,
Vegas Isomorphisn

[HERM83] Herman,
Graphics, and Imag

[MORT85] Mortenso

[NEWM79] Newman

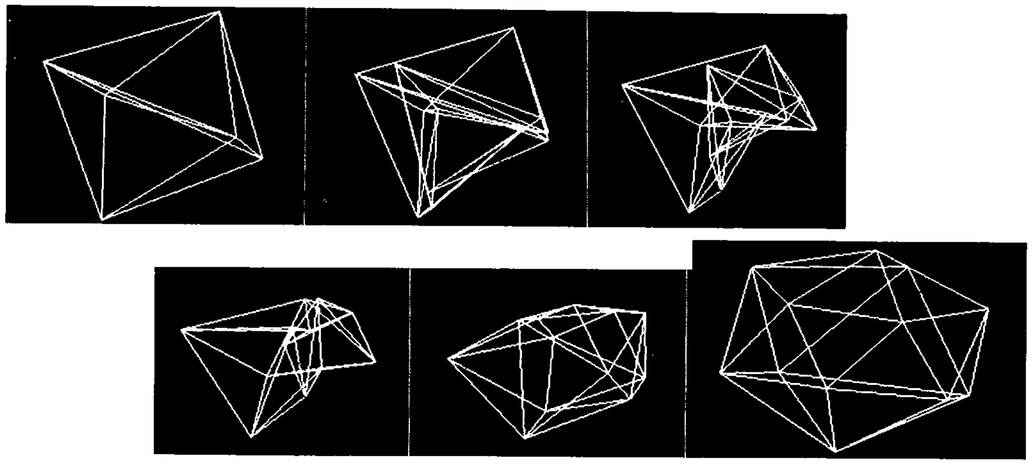
[PATT85] Patterson,
Graphics, October

[REEV81] Reeves, W
Volume 15, Numbe

[REYN82] Reynolds,
July 1982.

[SEDG83] Sedgewick

[SHEL82] Shelley, K
Number 3, August



Animation Sequence Three

7. REFERENCES

[AHO83] Aho, Apcroft and Ullman, Data Structures and Algorithms, Addison-Wesley, 1983.

[BARR84] Barr, Alan H., Global and Local Deformation of Solid Primitives, Computer Graphics, Volume 18, Number Three, August 1984.

[BETH86] Bethel, W., Computer Based Keyframe Animation of Shape Distortion, Master's Thesis, University of Tulsa, 1986.

[BRAI78] Braid, I., and R. Hillyard and I. Stroud, Stepwise Construction of Polyhedra in Geometric Modelling, CAD Group Document Number 100, University of Cambridge Computer Laborotory, October 1978.

[BURT71] Burtnyk, N. and M. Wein, Computer Generated Keyframe Animation, Journal of the Society of Motion Picture and Television Engineers, March 1971.

[BURT75] Burtnyk, N. and M. Wein, Computer Animation of Free Form Images, Computer Graphics, Volume Nine, Number One, 1975.

[BURT76] Burtnyk, N. and M. Wein, Interactive Skeleton Techniques for Enhancing Motion Dynamics, Communications of the ACM, Volume 19, Number Ten, October 1976.

[COHE80] Cohen, E., T. Lyche and R. Reisenfeld, Discrete B-splines and Subdivision Techniques in Computer-Aided Design and Computer Graphics, Computer Graphics and Image Processing, Volume 14, pp 87-111, 1980.

[COHE85] Cohen, E., T. Lyche and L. Schumaker, Algorithms for Degree Raising of Splines, ACM Transactions on Graphics, July 1985.

[EAST79] Eastman, C., and K. Weiler, Geometric Modeling Using the Euler Operators, Conference on Computer Graphics in CAD/CAM Systems, May 1979.

[GARE77] Garey, M., and D. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman and Co., 1979.

[GALI87] Galil, Z., C. Hoffmann, E. Luks, C. Schnorr and A. Weber, An $O(n^3 \log n)$ Deterministic and an $O(n^3)$ Las Vegas Isomorphism Test for Trivalent Graphs, JACM, Vol 34, Number Three, July 1987, pp. 513-531

[HERM83] Herman, G., and D. Webster, A Topological Proof of a Surface Tracking Algorithm, Computer Vision, Graphics, and Image Processing, Volume 23, pp 162-177, 1983.

[MORT85] Mortenson, M., Geometric Modeling, J. Wiley and Sons, New York, 1985.

[NEWM79] Newmann and Sproull, Prinicples of Interactive Computer Graphics, McGraw-Hill, 1979.

[PATT85] Patterson, R., Projective Transformations of the Parameter of a Bernstein-Bezier Curve, ACM Transactions on Graphics, October 1985.

[REEV81] Reeves, W., Inbetweening for Computer Animation Utilizing Moving Point Constraints, Computer Graphics, Volume 15, Number Three, August 1981.

[REYN82] Reynolds, C., Computer Animation Using Scripts and Actors, Computer Graphics, Volume 16, Number Three, July 1982.

[SEDG83] Sedgewick, R., Algorithms, Addison-Wesley, 1983.

[SHEL82] Shelley, K., and Greenberg, D., Path Specification and Path Coherence, Computer Graphics, Volume 16, Number 3, August 1982.

[STEK85] Stekete, S., and Badler, N., Parametric Keyframe Interpolation Incorporating Kinetic Adjustments and Phrasing Control, Computer Graphics, Volume 19, Number Three, August 1985.

[STER78] Stern, G., GAS: A System for Computer-Aided Keyframe Animation, PhD Dissertation, University of Utah, 1978.

[THOM81] Thomas, Frank and Johnston, Ollie, Disney Animation--The Illusion of Life, Abbeville Press, New York, 1981.

[WEIL85] Weiler, K., Edge-Based Data Structures for Solid Modeling in Curved Surface Environments, IEEE Computer Graphics and Applications, Volume Five, Number One, January 1985.

[WILS85] Wilson, P., Euler Formulas and Geometric Modeling, IEEE Computer Graphics and Applications, Volume Five, Number Eight, August 1985.

E. Wesly Bethel has been a member of the Applied Research Group at Island Graphics Corporation since 1987. Previously, he was Senior Graphics Engeneer at Geoscan, Incorporated in Tulsa, Oklahoma for two years. His research interests are in computer graphics, image processing and software architecture.
Mr. Bethel received a BS in Information Systems in 1983, and an MS in Computer Science in 1986 from the University of Tulsa. He is a member of ACM, SIGGRAPH and the IEEE Computer Society.

Address: Island Graphics Corpolation, 4000 Civic Center Drive, San Rafael, California, USA, 94903.
Usenet address: { uunet}!island!wes.

Samuel P. Uselton has been Assistant Professor of Computer Science in the Department of Mathematical and Computer Sciences at the University of Tulsa since 1982. Previously, he was an instructor at the University of Houston for three years. His research interests are mainly in computer graphics and image processing. His recent work has been in the areas of realistic image synthesis, scientific visualization and computer-assisted object description construction.
Dr. Uselton received the BA in Mathematics and Economics from the University of Texas(Austin)in 1973. He earned his MS in 1976 and PhD in 1981, both in Computer Science from the University of Texas at Dallas. He is a member of ACM, IEEE Computer Society, SIGGRAPH and an associate member of Sigma Xi.

Address: University of Tlusa, Department of Mathematical and Computer Sciences, 600 South College Avenue, Tulsa, Oklahoma, USA, 74104.

Auth

Arnaldi,
Badler,
Breen, I
Dumont,
Grosso,
Guenter,
Hégron,
Hild, H.
John, N.
Kunii, T
Lee, M.W
Magnenat
Max, N.
Ostby, E
Pins, M.
Quach, R
Selbie,
Spencer-
Susman,
Thalmann
Uselton,
Wesley B
Willis,
Wozny, M
Wyvill,

Nadia Magnenat-Thalmann
Daniel Thalmann (Eds.)

State-of-the-art in Computer Animation

Proceedings of Computer Animation '89

Springer-Verlag

State-of-the-art in Computer Animation

Selected topics and papers from the first international workshop on computer animation, held in Geneva in 1989, provide a comprehensive overview of the problems encountered in the rising field of computer animation. To foster interactive links between researchers, end-users, and artists, roundtables and discussions have been included as well as presentations of concepts and research themes such as keyframe to task-level animation, artificial intelligence, natural language and simulation for human animation, choreography, anthropometry for animated human figures, facial animation and expressions, the use of dynamic simulation, motion control and blur, and data-base oriented animation design.

ISBN 4-431-70046-3
ISBN 3-540-70046-3
ISBN 0-387-70046-3